GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

TRABAJO FIN DE GRADO

OPTIMIZACIÓN DE VARIABLES DE ENTRADA EN MODELOS DE MACHINE LEARNING

Autor: María Teresa Piergili de la Escalada

Director: Carlos Morras Ruiz-Falcó

Madrid

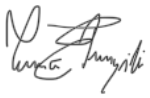
Julio de 2022

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

Optimización de variables de entrada en modelos de machine learning

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el curso académico **2021/2022** es de mi autoría, original e inédito y no ha sido presentado con anterioridad a otros efectos. El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido tomada de otros documentos está debidamente referenciada.

Fdo.: **María Teresa Piergili de la Escalada** Fecha:15/ 07/ 2022



Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: (Nombre del Director) Fecha: ……/ ……/ ……

GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

TRABAJO FIN DE GRADO

OPTIMIZACIÓN DE VARIABLES DE ENTRADA EN MODELOS DE MACHINE LEARNING

Autor: María Teresa Piergili de la Escalada

Director: Carlos Morras Ruiz-Falcó

Madrid

Julio de 2022

OPTIMIZACIÓN DE VARIABLES DE ENTRADA EN MODELOS DE MACHINE LEARNING

Autor: Piergili de la Escalada, María Teresa.
Director: Apellidos, Nombre.
Entidad Colaboradora: ICAI – Universidad Pontificia Comillas

RESUMEN DEL PROYECTO

El proyecto se dedica a realizar un sistema que sea de capaz de limpiar y preparar cualquier set de datos con técnicas de preprocesamiento. Partiendo de una librería capaz de realizar las acciones anteriores de forma simple y sencilla, se procede a añadir siete nuevas funciones y automatizar algunas ya existentes, volviendo a esta librería completamente automática y usando funciones de más alto nivel con el fin de poder ser analizado con técnicas de Machine Learning. Este proyecto tiene como finalidad evitar el tedioso trabajo que supone realizar cada cambio de forma manual y de forma repetitiva. Para dar transparencia y legitimidad al proyecto, se muestra por pantalla las acciones realizadas y detalla las transformaciones y eliminaciones realizadas.La validez del proceso ha sido comprobada gracias a dos test muy comunes (Titanic dataset y Credit approval dataset) obteniendo buenos resultado en las métricas de validación del modelo. Cabe destacar que la librería ha transformado 839 variables creadas por el programa anterior programa, a 7 variables con las modificaciones implementadas, que posteriormente analizada, genera un modelo logit con una curva ROC de calidad. El programa usa librerías Open source y siguiendo esa misma actitud de colaboración y compartición, se devolverá el código a GitHub.

Palabras clave: Aprendizaje automático, Python, Librería, Automatización

1. Introduccion

La inteligencia artificial con el análisis de datos está en pleno esplendor apogeo hoy en día, siendo de las carreras más demandas por las empresas. El acúmulo de datos es cada vez mayor y los datos brutos deben de ser limpiados y procesados para poder hacer modelos de clasificación y predicción sobre ellos. El problema recae en que el 80% del tiempo empleado en el análisis y modelaje de los datos se destina a en convertir los datos brutos en datos útiles [1]. Se ha querido solventar este problema

creando una librería de código abierto que automatiza la limpieza y procesado de los datos partiendo de una librería ya creada. En este caso, AutoDataCleaner en Python.

2. Definición del proyecto

El proyecto consta de la creación de 7 funciones y se han automatizado dos existentes. Las funciones nuevas son la conversión, desde la lectura del archivo, de las variables de tiempo y hora al formato datetime64, la detección de filas duplicadas y su eliminación, la detección de variables únicas y su eliminación, la detección de variables vacías y dependiendo del porcentaje de vacíos, se elimina la columna, la fila o se rellena dependiendo el tipo de variable por la mediana o la moda, la codificación por árbol de decisión y la optimización de la variable que debe ser codificada por arboles de decisión según el parámetro de exactitud, la detección del rango de valores atípicos y la eliminación de ellos, la detección de variable altamente correlacionadas, duplicadas o con poca varianza entre ellas y su correspondiente eliminación y por último la automatización de elección de variables que necesitan la normalización según el parámetro de exactitud. Todas estas funciones tienen sus respectivos comentarios para tener una visión interna de lo que hace exactamente la función.

3. Descripción del modelo/sistema/herramienta

Se ha implementado en spyder a través del anaconda3. El lenguaje de programación es Python y la siguiente llamada a la función `clean_me ()` de AutoDataCleaner muestra los parámetros.

```
data=adc.clean_me (df,
            detect_binary=True,
            numeric_dtype=True,
            decision_tree=True,
            one_hot=True,
            normalize=True,
            datetime_columns= [],
            remove_columns=['Name'],
            high_corr_elimination=True,
            low_var_elimination=True,
            measuring_variable='Survived',
            variable_to_encode= [],
            outlier_removal=True,
            duplicated_var=True,
            duplicated_rows_remove=True,
            variables_uni=True,
            clean=True,
            verbose=True)
```

4. Resultados

Los resultados obtenidos muestran que las métricas del modelo como la exactitud, precisión, especifidad y sensibilidad no tienen mucha diferencia con respecto al antes y después de las modificaciones. El objetivo principal de este proyecto era limpiar y procesar las variables completamente de forma automática y se ha conseguido reducir las variables de 839 a 8 con el nuevo programa. Cabe destacar que la nueva función tiene incorporada 7 funciones más que la anterior. A continuación, se muestran las variables restantes para el antes (izquierda) y el después (derecha).

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Columns: 839 entries, PassengerId to Embarked_S
dtypes: float64(2), int64(6), uint8(831)
memory usage: 778.9 KB
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 820 entries, 0 to 890
Data columns (total 8 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  820 non-null    float64
 1   Survived     820 non-null    int64
 2   Pclass       820 non-null    float64
 3   Sex          820 non-null    int64
 4   Age          820 non-null    float64
 5   SibSp        820 non-null    float64
 6   Parch        820 non-null    float64
 7   Fare         820 non-null    float64
dtypes: float64(6), int64(2)
memory usage: 57.7 KB
```

Ilustración 1 – Resultado final del antes (antes) y después (derecha)

5. Conclusiones

Las conclusiones obtenidas después de la ejecución del programa y comparando las métricas obtenidas entre el antes y el después y observando la curva ROC con su AUC (*Ilustración 2 – Curva ROC de AutoDataCleaner antes de las modificaciones y después de las modificacione*) se ha concluido que, aunque los dos programas tienen el mismo AUC y que el programa mejorado supera al original en sensibilidad, el objetivo principal era automatizar las funciones y mejorar dicho programa ya que hemos pasado de 839 columnas a 7 columnas.

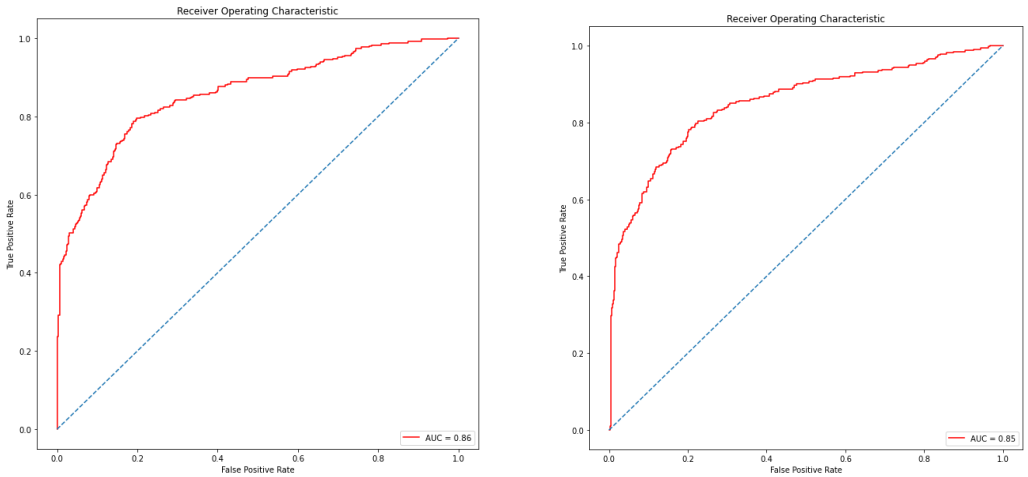


Ilustración 2 – Curva ROC de AutoDataCleaner antes de las modificaciones y después de las modificaciones

6. Referencias

[1] X. Wang y C. Wang, «Time Series Data Cleaning: A Survey,» *IEEE Access,* vol. 8, pp. 1866-1881, 6 Enero 2020.

[2] S. Galli, N. Galli, E. Sohayb y P. Suryawanshi, «Feature_engine,» 22 Enero 2021. [En línea]. Available: https://feature-engine.readthedocs.io/en/1.0.x/index.html.

[3] A. Géron, Hand-On Machine Learning with Scikit-Learn, Keras & TensorFlow, Croydon: O'Reilly, 2019.

[4] A. C. Álvarez, Programas con Python3, Wroclaw: Independently published (8 marzo 2021), 2021.

[5] J. VanderPlas, Python DataScience Handbook, Sebastopol: O'reilly, 2016.

OPTIMIZATION OF INPUT VARIABLES IN MACHINE LEARNING ALGORITHMS

Author: Piergili de la Escalada, María Teresa.
Supervisor: Apellidos, Nombre.
Collaborating Entity: ICAI – Universidad Pontificia Comillas

ABSTRACT

The project aims to create a system capable of cleaning and preparing any dataset with pre-processing techniques. Starting from a library capable of performing the above actions in a simple and easy way, we proceed to add seven new functions and automate some existing ones, making this library completely automatic and using higher level functions to be able to be analyzed with Machine Learning techniques. This project aims to avoid the tedious work involved in making each change manually and repetitively. To give transparency and legitimacy to the project, the actions carried out are shown on the screen and the transformations and eliminations are detailed. The validity of the process has been checked thanks to two very common tests (Titanic dataset and Credit approval dataset) obtaining good results in the validation metrics of the model. It should be noted that the library has transformed 839 variables created by the previous program into 7 variables with the implemented modifications, which subsequently analyzed, generates a logit model with a quality ROC curve. The program uses open-source libraries and following the same attitude of collaboration and sharing, the code will be returned to GitHub.

Keywords: Machine Learning, Python, Libraries, Automatization.

1. Introduction

Artificial intelligence with data analysis is currently in full bloom, being one of the most demanded careers by companies. The accumulation of data is increasing, and raw data needs to be cleaned and processed in order to perform classification and predictive modelling on it. The problem is that 80% of the time spent on data analysis and modelling is into converting raw data into useful data [1]. This problem has been solved with the creation of an open source library that automates data cleaning and

processing based on an already created library. In this case, AutoDataCleaner in Python.

2. Definition of the project

The project consists of the creation of 7 functions and two existing ones have been automated. The new functions are the conversion, from the reading of the file, of the time and hours variables to datetime64 format, the detection of duplicate rows and their elimination, the detection of unique variables and their elimination, the detection of empty variables and depending on the percentage of empty variables, the column, the row is eliminated or filled depending on the type of variable by the median or the mode, the coding by decision tree and the optimization of the variable to be coded by decision trees according to the accuracy parameter, the detection of the range of outliers and their elimination, the detection of highly correlated variables, duplicates or with little variance between them and their corresponding elimination and finally the automation of the choice of variables that need to be normalized according to the accuracy parameter. All these functions have their respective comments to get an inside view of what exactly the function does.

3. Description of the model/system/tool

It has been implemented in spyder via anaconda3. The programming language is Python and the following call to the AutoDataCleaner `clean_me()` function displays the parameters.

```
data=adc.clean_me(df,
            detect_binary=True,
            numeric_dtype=True,
            decision_tree=True,
            one_hot=True,
            normalize=True,
            datetime_columns=[],
            remove_columns=['Name'],
            high_corr_elimination=True,
            low_var_elimination=True,
            measuring_variable='Survived',
            variable_to_encode=[],
            outlier_removal=True,
            duplicated_var=True,
            duplicated_rows_remove=True,
            variables_uniques=True,
            clean=True,
            verbose=True)
```

4. Results

The results obtained show that the model metrics such as accuracy, precision, specificity and sensitivity do not differ much from before and after the modifications. The main objective of this project was to clean and process the variables completely automatically and it has been achieved to reduce the variables from 839 to 7 with the new program. It should be noted that the new function has 7 more built-in functions than the previous one. The remaining variables for before (left) and after (right) are shown below.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Columns: 839 entries, PassengerId to Embarked_S
dtypes: float64(2), int64(6), uint8(831)
memory usage: 778.9 KB
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 820 entries, 0 to 890
Data columns (total 8 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  820 non-null    float64
 1   Survived     820 non-null    int64
 2   Pclass       820 non-null    float64
 3   Sex          820 non-null    int64
 4   Age          820 non-null    float64
 5   SibSp        820 non-null    float64
 6   Parch        820 non-null    float64
 7   Fare         820 non-null    float64
dtypes: float64(6), int64(2)
memory usage: 57.7 KB
```

Ilustración 1 – Resultado final del antes (antes) y después (derecha)

5. Conclusions

The conclusions obtained after the execution of the program and comparing the metrics obtained between before and after and observing the ROC curve with its AUC (*Illustration 2 – ROC curve of AutoDataCleaner before modifications and after the modifications.*) it has been concluded that, although the two program have the same AUC and that the improved program surpasses the original in sensitivity, the main objective was to automate the functions and to improve the program as we have gone from 839 columns to 7 columns.

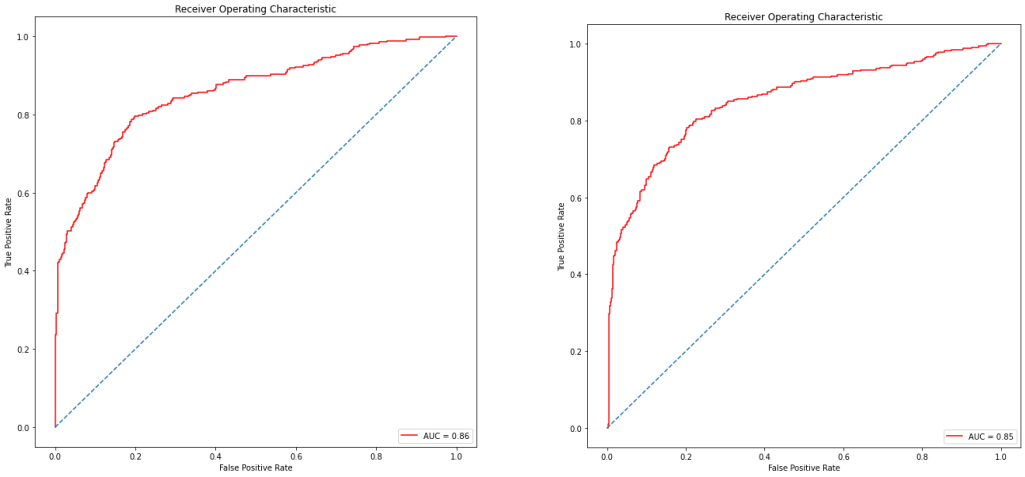


Illustration 2 – ROC curve of AutoDataCleaner before modifications and after the modifications.

6. References

[1] X. Wang y C. Wang, «Time Series Data Cleaning: A Survey,» *IEEE Access,* vol. 8, pp. 1866-1881, 6 Enero 2020.

[2] S. Galli, N. Galli, E. Sohayb y P. Suryawanshi, «Feature_engine,» 22 Enero 2021. [En línea]. Available: https://feature-engine.readthedocs.io/en/1.0.x/index.html.

[3] A. Géron, Hand-On Machine Learning with Scikit-Learn, Keras & TensorFlow, Croydon: O'Reilly, 2019.

[4] A. C. Álvarez, Programas con Python3, Wroclaw: Independently published (8 marzo 2021), 2021.

[5] J. VanderPlas, Python DataScience Handbook, Sebastopol: O'reilly, 2016.

Índice de la memoria

Capítulo 1. Introducción 7

1.1 Motivación del proyecto 8

Capítulo 2. Estado de la Cuestión y Justificación 10

Capítulo 3. Objetivos y Metodología 14

3.1 Objetivos del proyecto 14

3.2 Objetivos del desarrollo sostenible 15

3.3 Metodología 16

Capítulo 4. Descripción de las Tecnologías 17

4.1 Ordenador 17

4.2 Anaconda 18

Conda 20

Kernel 22

4.3 Python 23

4.4 Spyder 24

Características de Spyder 24

Librerías 26

4.5 Kaggle 39

4.6 OpenML 40

Capítulo 5. Creación del entorno virtual 41

Capítulo 6. AutoDataCleaner 54

Capítulo 7. Tratamiento de las variables 58

7.1 Read_csv 60

7.2 Valores Atípicos 62

7.3 Valores Faltantes 65
Eliminando variables vacías *65*
Imputación variables vacías *69*
7.4 Codificación 73
DecisionTreeDiscretization *74*
Automatización de la eliminación de variables para hacer decisión tree encoder *77*
7.5 Valores altamente correlacionados 79
7.6 Valores con poca varianza 82
7.7 Columnas duplicadas 84
7.8 Automatización de la normalización 86
7.9 Automatización de la eliminación de variables únicas 89
7.10 Eliminación de filas duplicadas 90
Capítulo 8. AutoDataCleaner con otros conjuntos de datos 93
8.1 Titanic 93
8.2 Credit Approval 97
Capítulo 9. Análisis de Resultados 101
Capítulo 10. Conclusiones y Trabajos Futuros 106
10.1 Conclusiones 106
10.2 Trabajos Futuros 108
Capítulo 11. Bibliografía 109
ANEXO I. Códigos 115
Titanic Proyecto 115
Script Titanic *115*
AutoDataCleaner Titanic *117*
Consola Titanic *138*

Titanic Comprobación 143

Script Titanic comprobación 143

AutoDataCleaner Titanic Comprobación 145

Consola Titanic comprobación 155

ANEXO II. Licencias 157

Licencia para DataCleaner de Randal S. Olson 157

Titanic Dataset 158

ANEXO III. Conjuntos de datos 159

Titanic Dataset 159

Antes AutoDataCleaner 159

Despues AutoDataCleaner 159

Credit Approval DataSet 160

Antes AutoDataCleaner 160

Despues AutoDataCleaner 160

Índice de figuras

Ilustración 1 – Propiedades de anaconda3 18
Ilustración 2 – Anaconda Navigator GUI 20
Ilustración 3 – Creación de entornos siendo copias del entorno base [18] 22
Ilustración 4 – Arquitectura del sistema 23
Ilustración 5 – Editor de Textos de Spyder 25
Ilustración 6 – Consola de IPython 25
Ilustración 7 – Explorador de variables 26
Ilustración 8 – AutoDataCleaner [10] 28
Ilustración 9 – Función de ayuda (help()) de la librería AutoDataCleaner 29
Ilustración 10 – DataFrame 30
Ilustración 11 – librerías dentro de sklearn. preprocessing 32
Ilustración 12 – Consola tras el entorno creado 43
Ilustración 13 – Consola después de la instalación de spyder-kernel 45
Ilustración 14 – Uso sencillo de AutoDataCleaner 54
Ilustración 15 – Representación del DataFrame (df) antes de la ejecución 55
Ilustración 16 – Información de df antes de aplicar AutoDataFrame 56
Ilustración 17 – Información sobre los cambios realizados en los datos 56
Ilustración 18 – Representación del DataFrame (df) después de la ejecución 57
Ilustración 19 – Arboles de decisiones 74
Ilustración 20 – Curva ROC de AutoDataCleaner antes de las modificaciones y después de las modificaciones 105

Índice de tablas

Tabla 1 – Formas de calcular los máximos y mínimos en winsorizer. 34
Tabla 2 – Parámetros y funcionalidades de la función Winsorizer 35
Tabla 3 – Parámetros y funcionalidades de la función DropCorrelatedFeatures 36
Tabla 4 – Parámetros y funcionalidades de la función DropConstantFeatures 37
Tabla 5 – Parámetros y Funcionalidad de la función DropDuplicatedFeatures 38
Tabla 6 – Parámetros y funcionalidad de la función DecisionTreeEncoder 39
Tabla 7 – Parámetros y funcionalidades de clean_me 55
Tabla 8 – Construcción de la función OutlierTrimmer 63
Tabla 9 – Outlier Removal y respuesta por consola 64
Tabla 10 – Eliminación de las variables para tratamiento de variables vacías (columnas). 67
Tabla 11 – Eliminación de las variables para tratamiento de variables vacías (filas) 69
Tabla 12 – Rellenar las variables para tratamiento de variables vacías 72
Tabla 13 – Decisión tree y respuesta por consola 77
Tabla 14 – High correlación variables y respuesta por consola 82
Tabla 15 – Duplicated features y respuesta por consola 84
Tabla 16 – Low variación elimination y respuesta por consola 86
Tabla 17 – Decisión tree y respuesta por consola 89
Tabla 18 – Variables uniques y su impresión por consola 90
Tabla 19 – Variables uniques y su impresión por consola 92
Tabla 20 – Matriz de confusión para titanic dataset con Proyecto 95
Tabla 21 – Matriz de confusión para titanic con Manac Sehgal proyecto 95
Tabla 22 – Exactitudes de Titanic dataset 95
Tabla 23 – Precisión de titanic 96
Tabla 24 – Especifidad de titanic 96
Tabla 25 – Sensibilidad de titanic 97
Tabla 26 – Matriz de confusión para credit approval con Proyecto 98
Tabla 27 – Matriz de confusión para credit approval con ZJ Low proyecto 98
Tabla 28 – Exactitudes de Credit approval 99

Tabla 29 – Precisión de Credit approval 99
Tabla 30 – Especifidad de Credit approval 99
Tabla 31 – Sensibilidad de Credit approval..... 100
Tabla 32 – Matriz de confusión para titanic dataset antes de las modificaciones 103
Tabla 33 – Matriz de confusión para titanic dataset después de las modificaciones..... 103
Tabla 34 – Métricas de clasificación del algoritmo antes y después de modificaciones... 103

Capítulo 1. INTRODUCCIÓN

El Machine Learning está presente en nuestro día a día y cuanto antes la sociedad se suba a lomos de este caballo ganador llamado tecnología revolucionaria, mejor se adaptará al cambio. Y sí, el cambio es innegable e inevitable.

La tecnología aplicada de forma correcta ofrece comodidad al ser humano y resuelve problemas que nunca podrían haberse solucionado. La tecnología es el futuro y aunque hay una filosofía en contra de los avances tecnológicos llamada Neoludismo [2] respaldada por el impacto negativo que la tecnología tiene en la sociedad en general, los pertenecientes a este grupo son una minoría.

Hay que destacar que hay un gran escepticismo a la hora de hablar de las tecnologías y su impacto positivo en la sociedad. Debido a los sucesos ocurridos en los últimos años, como la recopilación y tráfico ilegal de datos, entre muchos otros. No obstante, la tecnología tiene ya su sitio establecido en nuestra vida cotidiana y avanza a pasos de gigantes.

En especial, la tecnología dedicada a los datos y su análisis. Desde que empezó la era de la computación con Turing y Good cuando crearon la maquina capaz de analizar documentos cifrados por los alemanes en 1940 [3], el manejo de dato se ha vuelto esencial para las empresas. Tan esencial, tanto económica como físicamente que las carreras relacionadas con este campo están mas solicitadas.

Sería erróneo considerar el análisis de datos como una tecnología de lo más reciente, ya que está presente y consolidada en la sociedad. En los últimos 20 años, los analistas se han centrado en cómo crear modelos más precisos, como reducir el tiempo de computación, como crear librerías para facilitar la implementación de modelos, pero ha habido pocos avances con el tedioso trabajo que todos ellos han debido de pasar para llegar a observar el modelo. Ese manejo de datos, donde los analistas pasan más del 80% del tiempo

depurándolos es donde se quiere hacer énfasis en este proyecto [1]. La pérdida de tiempo en el análisis de datos ha sido la motivación intrínseca para crear este proyecto. La sociedad necesita, en concentro, reducir este tiempo de depuración de datos y es lo que se pretende ofrecer a la sociedad. Ya que el tiempo, como bien es conocido, es lo más importante que se tiene.

Un gran inconveniente es que muchas veces, la recolección de los datos brutos no llega a la calidad esperada y según el "Harvard Business Review" señala que solo 3% de los datos son de calidad [4]. Si los datos brutos no son de calidad y hay que convertirlos en información útil. La información útil, no será de calidad lo que afectará gravemente a los resultados del modelo y esto incrementará los costes, las pérdidas de tiempo y habrá aumento significativo de la toma de decisiones erróneas.

1.1 MOTIVACIÓN DEL PROYECTO

La motivación principal de este proyecto es ofrecer una herramienta para agilizar el proceso de preparación de los datos y la automatización de todo el proceso. Esta herramienta que se pretende crear es una librería de Python, partiendo de librerías ya existentes. El preproceso de datos se encarga de su limpieza, su integración, transformación y reducción para ser más tarde cotejados. Esta librería se encargará de procesar los datos y de optimizarlos y se devolverán los datos preparados con sus respectivos cambios y con las mejoras oportunas. El termino *Smart.* Solo se estudiará la optimización del preprocesado de las variables de entrada, ya que consume poco espacio de máquina y es donde más tiempo se le dedica.

Un buen procesamiento de datos nos lleva a una buena exactitud del modelo. Una pobre depuración de datos implica una predicción o clasificación errónea y muchas veces con imposibilidad de llevar el modelo a cabo.

Sin embargo, una buena limpieza de datos no es tarea fácil y por ello, el número de personal cualificado necesario es alto. Con esta nueva herramienta, se podría llegar a colocar a los analistas en distintas tareas y así conseguir que el proceso sea el más eficiente posible.

La cantidad de datos acumulados por persona es uno de los grandes problemas que tiene el procesamiento de datos. La agilidad de los algoritmos se ve radicalmente reducida en gran medida por este crecimiento y acumulo de datos y este problema lleva a la imposibilidad de procesado. “Sin embargo, en esta era de Big Data, los algoritmos de pre procesamiento tienen dificultades para trabajar con tal cantidad de datos, siendo necesario nuevos modelos que mejoren su capacidad de escalado.” [5]

La reducción del tiempo empleado tanto en la transformación de datos brutos en datos útiles como en la trasformación de datos útiles en datos utilizables por las maquinas será significativamente grande. Con una llamada la función que ocupa una línea de código, la limpieza de datos se ejecutará. Cabe destacar que la reducción del tiempo de maquina también es significativamente grande. Esta significativa reducción del tiempo llevara a la empresa a gestionar sus acciones cotidianas de forma eficaz y eficiente y el tiempo podría reinvertirse en otras áreas de la empresa como en I+D.

La implementación de esta librería puede llevar a pequeñas empresas a mejorar su rendimiento sin la más mínima inversión dando paso a que puedan llegar a hacer frente a gigantes tecnológicos en sectores altamente competitivos. A partir de este momento, las empresas pueden introducirse en nuevos mercados que antes, eran imposibles.

Capítulo 2. ESTADO DE LA CUESTIÓN Y JUSTIFICACIÓN

Con la creación en 1962 de *"The Future of Data Analysis"* [6], John Tukey paso a ser uno de los grandes fundadores del "*Data Science"*. Seguido por Peter Naur con la aclaración de este mismo termino en 1974 en *"Concise Survey of Computer Methods"* [7] y finalmente en 1977, con Jeff Wu, el cual reclamó que la estadística se llama *"Data Science"* y que los estadísticos se llamarán, a partir de ese momento, *"Data scientist"* [8].

Hoy en día, el Aprendizaje Automático o Machine Learning, ML por sus siglas en ingles está presente en nuestro día a día. Una simple búsqueda en Google scholar y en menos de 0,05 segundos aparecen más de 5.030.000 trabajos que contienen esa palabra y en 2022 se han añadido hasta día de hoy 39.900 resultados. El ML está en auge y es innegable.

Por la búsqueda realizada, nos lleva a indicar que ha habido muchos trabajos dedicados al ML pero a decir verdad, los artículos que aparecen en *google schoolar* son usando esa herramienta en vez de mejorarla.

Según un estudio realizado por Kaggle en 2017, donde obtuvieron 16.000 respuestas indica que donde la mayoría de analistas encuentran mayor dificultad en su profesión es en el manejo de datos sucios obteniendo un 36% [9].

Cabe destacar que la comunidad de inteligencia artificial y machine learning son propensos al uso y divulgación de código abierto. La liberación de la tecnología comenzó hace 30 años, cuando se llevó a cabo el proyecto GNU[1] lo que implica una colaboración tecnológica global capaz de unir fuerzas para encontrar soluciones aparentemente imposibles de resolver. Por esta razón, han aparecido programas como GitHub, OpenML o Kaggle donde analistas y

[1] Uno de los primeros sistemas operativos de software libre que permite usar un ordenador sin software tradicional. [62]

comunidad de machine learning, comparten programas, progresos y errores. Ofrecen ayuda y a cambio son ayudados por toda la comunidad.
Gracias a la creación de estos programas y debido a que son de software libre, la comunidad machine learning avanza a pasos de gigante.

Después de una búsqueda intensa en estos programas, se ha encontrado que para el pre procesado de datos hay unas librerías muy interesantes pero cuando tratamos de buscar su automatización, la búsqueda se vuelve más complicada.
A continuación, se muestran unas librerías bastante avanzadas sobre la limpieza inicial de los datos.

La primera librería que se especificara más adelante en el apartado 4.4.1.7 es AutoDataCleaner [10]. Esta librería cuenta que, con una simple línea de código, las principales funciones para limpieza de datos sean aplicadas al conjunto de datos proporcionado.

La segunda librería que se va a detallar es Feature-Engine [11] en el apartado 4.4.1.12. Es la librería más completa de todas las observadas, el único inconveniente que tiene es su nula automatización. Es comprensible que no haya ninguna automatización debido a que cada conjunto de datos es único y la tarea de buscar una solución global para todo conjunto de datos puede llevar a una limpieza errónea y a unos resultados imprecisos sobre la muestra.

Una librería muy interesante es Dora [12] y fue diseñada pata simplificar el análisis de datos. Esta librería está completamente automatizada pero no devuelve un pandas dataframe. Sus funciones son las siguientes, imputa los valores que faltan (utilizando la media de cada columna), escala los valores de las variables de entrada (centrar a la media y escalar a la varianza unitaria), selecciona las características / elimina una característica, extrae un rasgo ordinal a través de la codificación, extrae una transformación de otra característica entre otras características.

Otra librería interesante es datacleaner [13]. Sus funciones son las de eliminar cualquier fila con un valor ausente, sustituir los valores perdidos por la moda (para las variables categóricas) o la mediana (para las variables continuas) columna por columna y de codificar las variables no numéricas (por ejemplo, las variables categóricas con cadenas) con equivalentes numéricos. El inconveniente de esta librería es que se queda un poco pequeña para este proyecto. Cabe destacar que de esta librería se ha obtenido la función para rellenar los valores ausentes por la moda o mediana como se explica en el apartado Imputación variables vacías.

Hay algunas librerías también creadas para automatizar el proceso de construcción del modelo como Auptimizer [14].

En este proyecto no se han considerado variables relacionales o temporales y por tanto no se ha usado Deep Feature Synthesis [15].

Al final de este proyecto en el Capítulo 9. , se quiere comprobar como de preciso es el programa creado. Para ello se usará la librería Learn2Clean [16]. Esta librería se encarga del pre procesamiento y la limpieza de datos basados en una técnica de aprendizaje por refuerzo sin modelos llamada Q-Learning. Se encarga de seleccionar un modelo y un rendimiento de calidad con el objetivo de maximizar la calidad del modelo.

Como se puede observar, ha habido bastantes proyectos dedicados a la limpieza de datos, pero la mayoría faltan de procesos claves o de optimización.

Este proyecto se encarga de reunir las funciones de los programas mencionados para hacer una librería lo más completa posibles. Una de las características añadidas, basadas en datacleaner, es la posibilidad de añadir comentarios y una vez ejecutado el programa, se imprimirá por pantalla todas las funciones ejecutadas. Con esta función se quiere dar al analista una visión interna del programa.

Es importante tener en cuenta que como mencionado en [17] aun teniendo un buen pre procesado de datos, es de vital importancia la calidad de los datos obtenidos y que el óptimo será individual a cada conjunto de datos.

Capítulo 3. OBJETIVOS Y METODOLOGÍA

3.1 OBJETIVOS DEL PROYECTO

Los principales objetivos que se desea logar al final de este proyecto se presentan a continuación.

El principal objetivo es la creación de una librería de código abierto que, dado unos datos, los procese y los limpie, devolviendo al usuario de esa librería, un conjunto de datos listo para usarse. La librería esta casi completamente automatizada, esto quiere decir, que el usuario solo tiene que indicar los mínimos parámetros.

La librería se encarga de las siguientes funciones:

- Limpiar variables vacías (eliminando columnas o filas)
- Rellenar variables vacías por mediana o moda
- Eliminar los datos atípicos
- Detectar columnas binarias
- Convertir variables con One Hot encoding
- Normalizar las variables
- Cambiar variables de caracteres en numéricas
- Codificación por arboles de decisión
- Eliminar variables altamente correlacionadas
- Eliminar variables duplicadas
- Eliminar variables con poca varianza entre ellas
- Comentarios por cada línea de código procesada

El objetivo de la librería es reducir el tiempo de depuración en modelos de Machine Learning, optimizando las variables de entrada para reducir tiempo de ejecución, error humano y reducir costes en forma de maquinaria y personal. Se comprobará su optimización

a través de un modelo de regresión logística donde, también, encontrar la matriz de confusión a la vez que la curva ROC. Esto se detallará en el Capítulo 8.

3.2 OBJETIVOS DEL DESARROLLO SOSTENIBLE

Los objetivos del desarrollo sostenible se han tenido en cuenta y cumplimentado la mayoría de ellos. La ventaja de crear una librería en Python es que es de gran utilidad en todos los sectores.

La creación de los 17 objetivos del desarrollo sostenible[2] proviene de la voluntad intrínseca de erradicar la pobreza, proteger el planeta y asegurar la prosperidad de cada ser humano. Los principales objetivos que este proyecto abarca son Hambre Cero con la implementación del ML en la optimización de recursos, Salud y Bienestar con la creación de nuevos proyectos de ML como la identificación de imágenes, Educación y Calidad con la posibilidad de educación personalizada y automatizar los procesos más repetitivos, Agua Limpia y Saneamiento con la mejora del sistema de la redistribución del agua, Energía Asequible y No Contaminante con la posibilidad de predicción de posibles fallas en el sistema de abastecimiento de energía o ser más resistentes a las fluctuaciones en la demanda, Trabajo Docente y Crecimiento Económico ofreciendo con el ML la posibilidad de disminuir la brecha digital, Industria, Innovación e Infraestructura con el incremento de las infraestructuras basándose en el análisis de datos, Producción y el Consumo Responsable a través de optimización de producción con posibilidad de proyecciones futuras, Acción por El Clima como la prevención de catástrofes por la recogida de datos u optimización de sistemas de tráfico, Vida Submarina para prevenir la contaminación de los océanos a través de un nuevo parámetro de medición de contaminación (plancton), Vida de Ecosistemas Terrestres facilitando el estudio de especies en extinción por la recogida de los datos y por ultimo Paz Justicia e Instituciones sólidas gracias a la anticipación del crimen y eliminar el sesgo basados en los datos recogidos de forma racista.

[2] Objetivos del desarrollo sostenible (ODS) se establecieron en 2015 para que en el 2030 estuvieran todos implementados.

3.3 METODOLOGÍA

La metodología que se usara a lo largo de este proyecto para lograr los objetivos establecidos serán cambios de forma interactiva, para que el proyecto se vaya construyendo progresivamente a lo largo del año.

Para comenzar, después de una búsqueda exhaustiva de librerías en Python ya creadas que se encarguen específicamente en el pre proceso de datos de Machine Learning, se decidirá cuál es la más completa y fácil de usar. La búsqueda se realizará a través de GitHub que se explicará más adelante en el apartado GitHub del Capítulo 4.

A la vez que se elige la librería, se debe descargar Anaconda3 que se explicará más adelante en el Capítulo 4. en el apartado 4.2. Para no modificar Anaconda con las descargas de las librerías encontradas en GitHub, ya que cada una tiene versiones distintas de cada librería, se creará un entorno virtual (especificado en el Capítulo 5.) para cada librería y así probar con seguridad. Una vez encontrada la librería con la que se desea trabajar, se procederá a realizar los cambios correspondientes.

Posteriormente, se procederá a realizar los cambios en la función principal y mejorando la nueva función en cada vuelta de revisión.

Una vez obtenido el nivel que se desea de limpieza con esa función, se procederá a la creación/ mejora de las siguientes funciones.

Finalizado el código, se añadirán los comentarios que explicarán al usuario el cambio que se ha realizado al conjunto de datos. Este último paso será como tener una visión de la parte interna de la librería sin tener que abrirla.

Capítulo 4. DESCRIPCIÓN DE LAS TECNOLOGÍAS

Las tecnologías implementadas en este Trabajo de Fin de Grado son varias ya que el trabajo es de crear una librería *open source* para la optimización del preprocesado de los datos. Se parte de una librería open source llamada "AutoDataCleaner" de GitHub.

Primero, se explicará el ordenador con el sistema operativo que ha sido usado, se proseguirá con la explicación de *Anaconda software*, con el uso de *conda* y su respectivo entorno creado, el uso del *kernel* y se justificará la decisión de su uso. Seguidamente, se procederá a explicar, primero, el lenguaje *Python*, seguidamente, se explicará *Spyder y* sus distintas partes incluyendo las librerías a usar.

4.1 ORDENADOR

En este trabajo de fin de grado es necesario un ordenador con sistema operativo Windows, macOS o Linux. Un sistema operativo Windows 7 o posterior es válido. En este caso en particular se usa la siguiente configuración de Windows.

Procesador	Intel(R) Core(TM) M-5Y51 CPU @ 1.10GHz 1.20 GHz
RAM instalada	8,00 GB
Id. del producto	00326-10012-33254-AA741
Tipo de sistema	Sistema operativo de 64 bits, procesador basado en x64

Se instalará Anaconda por instalador grafico de 64 bits para Windows. Es preferible instalar Anaconda con las opciones por defecto para evitar conflictos futuros.
El disco duro debe contener al menos 1 GB libre. Puede ser que se necesite un espacio mayor dependiendo de las bases de datos usadas.

El programa anaconda contiene las siguientes propiedades (Ilustración 1 – Propiedades de anaconda3).

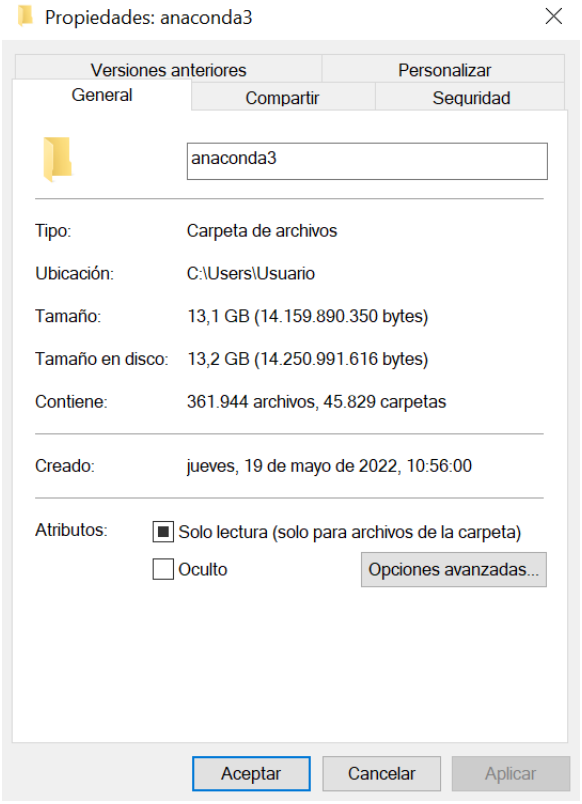


Ilustración 1 – Propiedades de anaconda3

4.2 ANACONDA

Anaconda es una distribución[3] de los lenguajes de programación Python y R para la computación científica de uso libre. La misión de Anaconda es de simplificar el tratamiento de los paquetes y su implantación. Se puede obtener tanto para Windows como para Linux o macOS. El manejo de los paquetes de Anaconda está dirigido por el sistema libre conda.

Anaconda fue creada en 2012 por Travis Oliphant y Peter Wang, el cual, hoy en día, es el presidente de la empresa Anaconda Inc.

[3] Una distribución (distro) es un proceso de entrega de software desde el desarrollador hasta el usuario final. (*¿What Is Distribution Software? - Definition From Techopedia*, n.d.)

La distribución Anaconda viene con 250 paquetes automáticamente instalados y se pueden instalar paquetes de libre acceso a través de PyPi[4], el administrador de paquetes conda o un entorno virtual diferente. La gran diferencia entre PyPi y el administrador de paquetes conda es el manejo de las dependencias entre paquetes. La descarga de librerías a través de PyPi es automática sin importar las dependencias que ese paquete conlleva con los paquetes instalados previamente y por tanto podría descargar paquetes de versiones anteriores. Usando el administrador conda, esto ya no ocurre, ya que la propia función indica las dependencias y despliega un aviso de imposibilidad de instalación.

Los entornos virtuales sirven para usar las distintas versiones de los paquetes virtuales, han sido creados con el fin de aislar librerías y otros entornos. La creación del nuevo entorno se puede hacer de varias maneras. En este proyecto, se implementará a través de conda. La razón del uso de un entorno virtual es para evitar cualquier tipo de dependencias con el entorno base.

Dentro de Anaconda distribución (Ilustración 2 – Anaconda Navigator GUI), se puede acceder al Anaconda Navigator, una interfaz gráfica del usuario (GUI)[5] donde permite a los usuarios lanzar aplicaciones y gestionar paquetes, entornos y canales de conda sin utilizar comandos.

[4] El PiPy llamado Python Package Index es la tercera parte del repositorio de software, donde se almacenan los paquetes de software que pueden después ser instalados en un ordenador, de Python.
[5] Una interfaz gráfica del usuario (GIU) permite al usuario interactuar con componentes electrónicos a través de iconos gráficos o indicadores de audio.

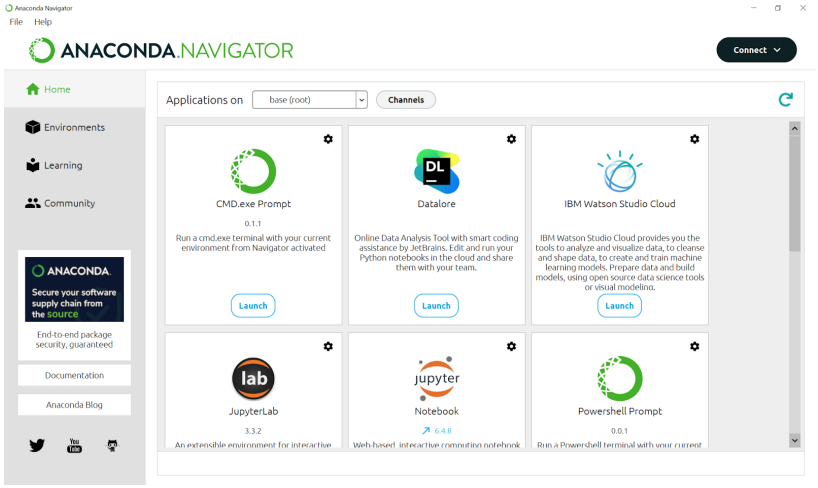


Ilustración 2 – Anaconda Navigator GUI

Se puede observar en la, las siguientes aplicaciones disponibles por defecto en Anaconda Navigator:

1. JupyterLab
2. QtConsole
3. Spyder
4. Jupyter Notebook
5. Glue
6. Naranja
7. RStudio
8. Código de Visual Studio

En este trabajo, se implementará todo a través de la herramienta Spyder.

CONDA

Conda es un gestor de paquetes y de entornos que utiliza los comandos en el *Anconda Prompt*. Se creo con la intención de manejar los problemas que causaban la inclusión de paquetes en Python a los analistas de datos, pero debido a su gran utilidad, ha terminado siendo un paquete en sí mismo, permitiendo tener distintos entornos virtuales con distintos paquetes. Conda también se encarga de la gestoría de paquetes en R. Este gestor está publicado bajo la licencia de BSD por Continuum Analytics.

Las siguientes líneas de código se ejecutan en el símbolo del sistema, también llamado, *Comand Prompt* o cmd.exe[6]. La primera es para saber que versión de Python es y la segunda es para saber a qué versión de conda pertenece.

```
Microsoft Windows [Version 10.0.19042.1415]
(c) Microsoft Corporation. Todos los derechos reservados.

(base) C:\Users\Usuario>python --versión
Python 3.10.4

(base) C:\Users\Usuario>conda --versión
conda 4.13.0
```

4.2.1.1 Conda Environment

El entorno virtual de conda es un directorio donde se instalan una colección de paquetes conda. Los paquetes implementados en este entorno no influyen al entorno base ya que han sido creados a través de una copia del fichero Lib. La gran utilidad sobre los entornos es que, por un lado, aíslan la ejecución de los proyectos y por tanto cada proyecto cuenta con su propio entorno virtual, por otro lado, previene errores como la sobre escritura no deseada de variables, métodos y clases. Los entornos virtuales deben ser activados para poder usarse.
A continuación, se presenta el código para su creación, activación, desactivación y su eliminación a través del cmd.exe de Anaconda.

```
Conda create--name MiEntorno

Conda activate MiEntorno

Conda deactivate

Conda remove --name MiEntorno -all
```

[6]Cmd.exe ejecuta acciones en el sistema operativo. [30]

Una vez creado el nuevo entorno, Anaconda lo guardara en la carpeta *envs* con todos los demás entornos.

La Ilustración 3 – Creación de entornos siendo copias del entorno base muestra la creación de entornos virtuales a través de copias del entorno base. Se puede observar cómo se copia la carpeta *Lib* en todos los otros directorios y previene que, habiendo un cambio de ese fichero en un entorno, no afecta a los demás. Los scripts se ejecutarán en cada entono y no con el entorno base.

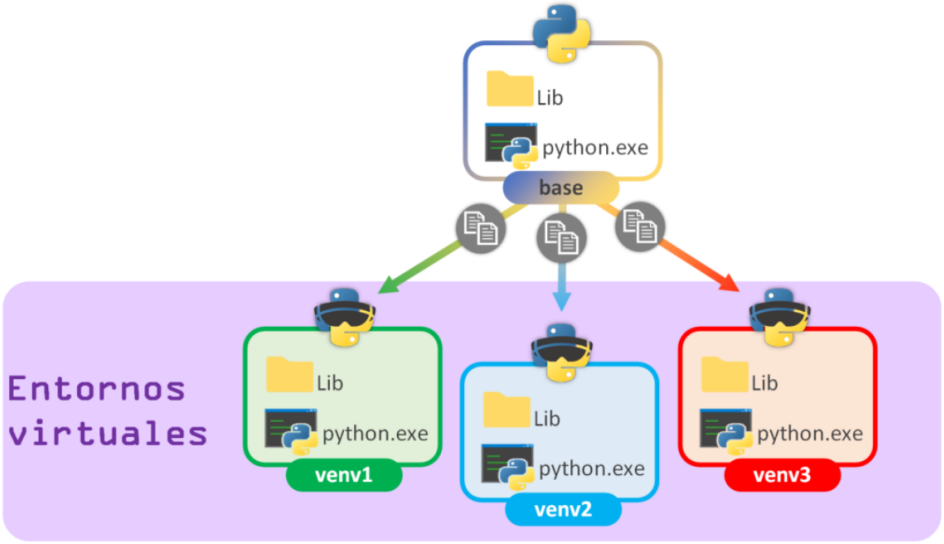


Ilustración 3 – Creación de entornos siendo copias del entorno base [18]

El entorno conda que usaremos será llamado "OptimizacionVarEntrada" que se explicará en el Capítulo 5. más adelante.

La implementación del entorno virtual será cambiando el interpretador de Python del sistema preconfigurado en Spyder. Para ello, habrá que pulsar debajo del kernel en la barra de estado el botón donde aparece el entorno. Una vez pulsado, aparecerá la opción de *Change default environment in preferences* y se seleccionará el entorno nuevo.

KERNEL

El kernel (o núcleo) es el núcleo del sistema operativo y se encarga de conceder acceso al hardware de forma segura para todo software que lo necesite. Un comando tecleado por el usuario es interpretado por el Shell para que el kernel lo ejecute. El usuario se

comunica a través del Shell con el kernel. En la Ilustración 4 – Arquitectura del sistema se muestra un esquema del proceso mencionado en este apartado, en otras palabras, la arquitectura del sistema. El kernel se ejecuta encima del SO básico del ordenador, en nuestro caso Windows 10.

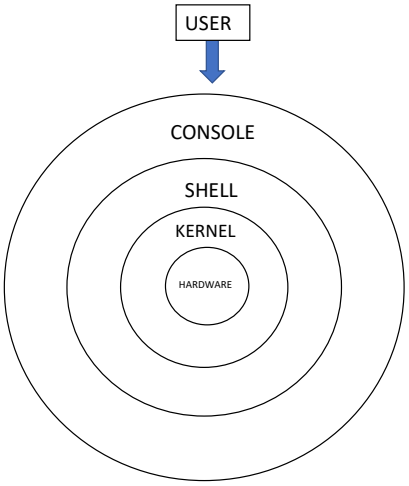


Ilustración 4 – Arquitectura del sistema

El kernel de Jupyter es un proceso especifico del lenguaje de la programación que ejecuta el contenido en un cuaderno de Jupyter en un terminal distinto. Hay que tener en cuenta que al crear un nuevo entorno en conda habrá que cargar la librería "spyder-kernels" mostrada a continuación. Esta librería proporciona el kernel de Jupyter en Spyder.

```
conda install spyder-kernels
```

4.3 PYTHON

Python fue creado en 1991 por Guido van Rossum y desarrollado posteriormente por *Python Software Foundation.* El nombre que se instauro para este lenguaje de programación proviene de la serie producida por la BBC "Monty Python's Flying Circus". La intención detrás de este lenguaje era su legibilidad del código y productividad avanzada al desarrollador. La última versión estable de Python es 3.14.4. Se recomienda usar versiones de Python 3.X ya que ha habido muchos cambios comparados con Python versión 2.X.

Python es el lenguaje de programación más usado y conocido en el mundo entero. Gracias a varios factores, se ha convertido en estos 30 años en el lenguaje preferido por los analistas de datos en el mundo entero.

Por un lado, Python es un lenguaje de programación open source con una sintaxis fácil. En segundo lugar, Python es un lenguaje orientado a objetos[7], lo que provoca una disminución de los errores. Por otro lado, posee una comunidad activa y una amplia gama de bibliotecas y recursos y, por tanto, está estrechamente vinculado a las nuevas tecnologías. También hay que tener en cuanto que la ratio entre el proceso de desarrollo y la complejidad del lenguaje en Python es mucho más alta que en otros lenguajes de programación, favoreciendo su uso.

En este Trabajo, se usará la versión de Python 3.7 ya que el spyder-kernel da problemas de compatibilidad y, por tanto, es la versión estable con spyder.

4.4 SPYDER

Scientific Python Development Environment llamado también como Spyder es un IDE[8] libre incluido en Anaconda completamente escrito en Python. Está pensado para analistas de datos, ingenieros y científicos.

A continuación, se explicarán las características de Spyder. Para comenzar, como en todo IDE, se explicará el editor de texto, el *debugger*, la consola, el explorador de variables y el *profile*r.

CARACTERÍSTICAS DE SPYDER

4.4.1.1 Editor de texto

La escritura del código es muy sencilla debido a que utiliza un editor de código multilenguaje, también tiene resaltado de sintaxis (`pygments`), análisis de código en tiempo real (`pyflakes`), análisis de estilo (`pycodestyle`), finalización bajo demanda en otras muchas características. En Ilustración 5 – Editor de Textos de Spyder se puede ver este editor.

[7] La programación orientada a objetos se basa en el concepto de crear un modelo del problema de destino en sus programas. [62]
[8] IDE (Integrated Development Environment) son aplicaciones para programar, compilar, descompilar y hacer debug.

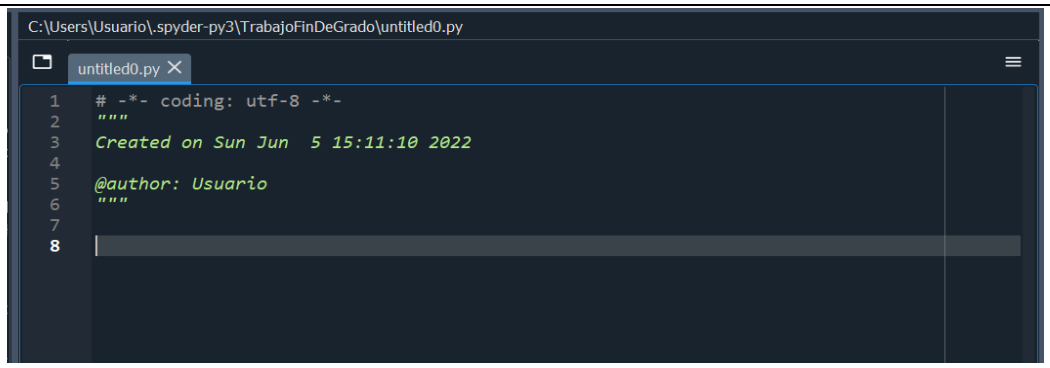


Ilustración 5 – Editor de Textos de Spyder

4.4.1.2 Debugger

Se cometen errores programando y, por tanto, necesitamos un debugger. El debugger que utiliza Spyder es `ipdb` se encarga de detectar errores y se va deteniendo en cada error para entender lo que está pasando. Estos errores aparecen en la consola.

4.4.1.3 IPython Console

La Ilustración 6 – Consola de IPython muestra la consola de Spyder llamada IPython console. Esta consola permite al usuario dar entradas en forma de comandos y devuelve mensajes del nucleo, advertencias y salidas. Permite al usuario interpretar el codigo del editor o de forma iterativa.

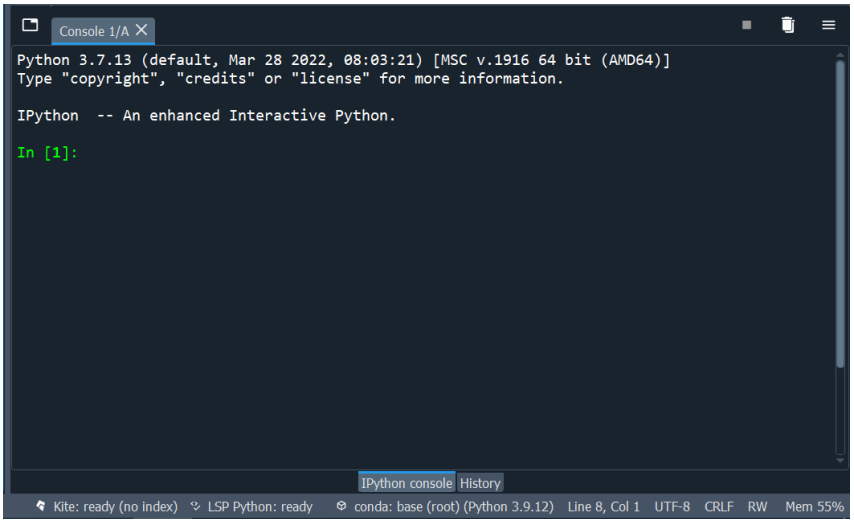


Ilustración 6 – Consola de IPython

4.4.1.4 Explorador de Variables

La Ilustración 7 – Explorador de variables muestra el explorador de variables. El explorador de variables te permite cargar interactivamente y manejar los objetos creados por el código interpretado. El explorador de variables te muestra todo lo que está en el *namespace* de la consola IPython actual. También, te permite añadir, eliminar y editar valores a través de una variedad de editores basados en GUI. Permite obtener saber la información como nombres, tamaños y valores de cada objeto.

Ilustración 7 – Explorador de variables

4.4.1.5 Profiler

El panel profiler determina recursivamente el tiempo de ejecución y el número de llamadas de cada función. Indica las sentencias exactas más críticas para la optimización, permite identificar los cuellos de botella del código, y mide el delta de rendimiento después de los cambios de seguimiento.

LIBRERÍAS

Las librerías son un conjunto de herramientas que eliminan al usuario de tener que escribir el código desde cero. Hoy en día hay más de 137 000 librerías creadas. Las librerías son de vital importancia para desarrollar el aprendizaje automático, visualizar los datos y su manipulación

4.4.1.6 Spyder-kernel

La primera librería que se necesita es `spyder-kernels` ya que sin ella no se podrá conectar el kernel con la consola del spyder.

4.4.1.7 AutoDataCleaner

La siguiente librería que se instalara en el nuevo entorno es la AutoDataCleaner en su versión 1.1.3 (la más reciente). Esta librería de código abierto es para que con una línea de código devuelva, de forma automática, los datos limpios con comentarios sobre los distintos cambios sobre los datos.

El programa AutoDataCleaner ha sido hallado en GitHub gracias a una búsqueda muy delicada. Se tuvo que ponderar cada código por sus características y por las funciones faltantes en cada uno de ellos. La elección de esta librería se explica en el Capítulo 2. En este capítulo se puede observar las características de las demás librerías estudiadas y la elección de AutoDataCleaner que recae en la robustez de la librería y su adecuada estabilidad y por ello se ha elegido librería principal. La librería AutoDataCleaner es la librería más completa, con más automatizaciones y con el punto de partida más cercano al punto final.

Se encarga del siguiente preprocesado:

- one-hot encoding
- conversión de fecha y hora a dtype
- detección automática de columnas binarias
- conversión segura de columnas no numéricas a tipos de datos numéricos
- la limpieza de valores sucios/vacíos
- la normalización de valores
- eliminación de columnas no deseadas

La Ilustración 8 – AutoDataCleaner muestra de forma gráfica el preprocesado que se llevara a cabo.

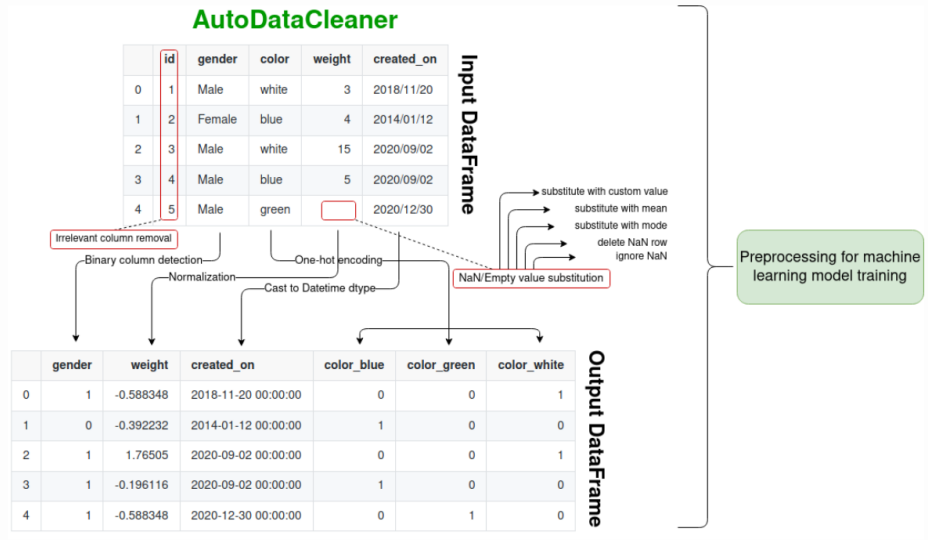


Ilustración 8 – AutoDataCleaner [10]

Su instalación se lleva a cabo tras ejecutar la siguiente línea de código.

```
Pip install AutoDataCleaner
```

La constitución de la función es la siguiente.

```
import AutoDataCleaner.AutoDataCleaner as adc

data=adc.clean_me(df,
            detect_binary=True,
            numeric_dtype=True,
            decision_tree=True,
            one_hot=True,
            normalize=True,
            datetime_columns=[],
            remove_columns=['name','home.dest'],
            high_corr_elimination=True,
            low_var_elimination=True,
            measuring_variable='survived',
            variable_to_encode=['cabin','pclass','embarked'],
            outlier_removal=True,
            duplicated_var=True,
            verbose=True)
```

Cabe destacar que también tiene una función propia de help donde vienen explicados los comandos `Adc.help()`.

Pasando el comando help por la consola llamando con anterioridad a adc (AutoDataCleaner), se obtiene la siguiente información (Ilustración 9 – Función de ayuda (help()) de la librería AutoDataCleaner).

```
In [3]: adc.help()

    ++++++++++++++++++ AUTO DATA CLEANER HELP ++++++++++++++++++++++
    FUNCTION CALL:
    AutoDataCleaner.clean_me(df, one_hot=True, na_cleaner_mode="mean", normalize=True,
remove_columns=[], verbose=True)

    FUNCTION PARAMETERS:
    df: input Pandas DataFrame on which the cleaning will be performed
    one_hot: if True, all non-numeric columns will be encoded to one-hot columns
    na_cleaner_mode: what technique to use when dealing with None/NA/Empty values. Modes:
        False: do not consider cleaning na values
        'remove row': removes rows with a cell that has NA value
        'mean': substitues empty NA cells with the mean of that column
        'mode': substitues empty NA cells with the mode of that column
        '*': any other value will substitute empty NA cells with that particular value
passed here
    normalize: if True, all non-binray (columns with values 0 or 1 are excluded) columns
will be normalized.
    remove_columns: list of columns to remove, this is usually non-related featues such as
the ID column
    verbose: print progress in terminal/cmd
    returns: processed and clean Pandas DataFrame
    ++++++++++++++++++ AUTO DATA CLEANER HELP ++++++++++++++++++++++
```

Ilustración 9 – Función de ayuda (help()) de la librería AutoDataCleaner

Se puede observar a que se refiere cada parámetro dentro de la función y como se debe implementar para su correcto funcionamiento.

4.4.1.7.1 GitHub

GitHub es una plataforma de código libre para crear proyectos. Su principal característica es la posibilidad de que todos pueden aportar nuevas funciones, características, optimizaciones o formas más limpias de conseguir un código mejor y más elaborado. Resumiendo, varias personas pueden estar modificando el mismo programa, estas modificaciones pueden ser actualizaciones de mantenimiento, errores señalados en el apartado de *"issues"* etc.

El final de este proyecto se implementará en GitHub para aportar un granito de arena a la comunidad científica.

4.4.1.8 Pandas

Pandas es una librería de Python para el manejo y análisis de estructura de datos.
Las características son las siguientes:

- Nuevas estructuras de datos basados en los arrays de NumPy
- Lee fácilmente y escribe ficheros en formato CSV, Excel y bases de datos en SQL.
- Permite acceder a los datos a través de indicies o nombres para filas y columnas

Pandas contiene tres tipos de datos diferentes, series (una dimensión), DataFrames (dos dimensiones (tablas)) y Panel (tres dimensiones(cubos)).

Los DataFrames (Ilustración 10 – DataFrame) son un conjunto de datos estructurados en forma de tabla donde cada columna es un objeto de tipo Serie, todos los datos por columna son del mismo tipo [19].

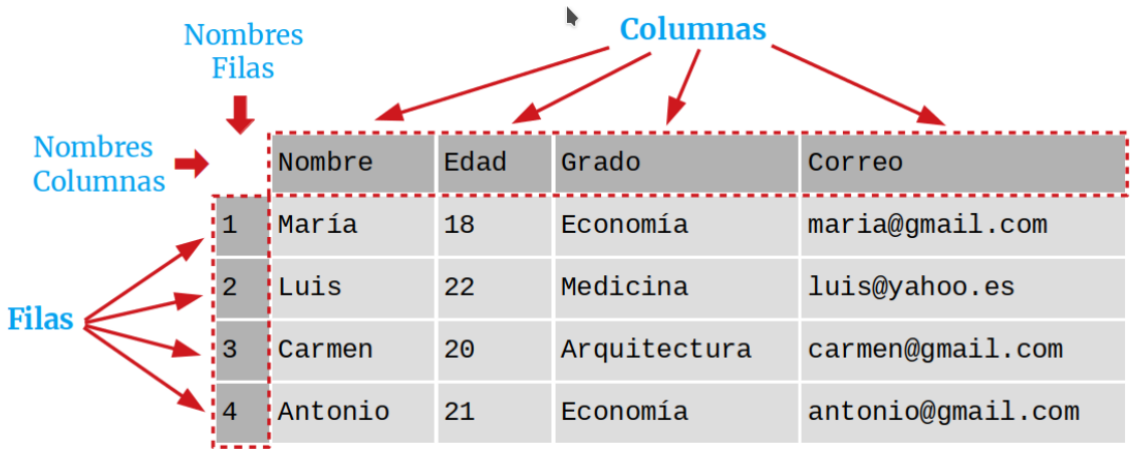


Ilustración 10 – DataFrame

Esta librería viene instalada con su correspondiente versión (1.3.5) gracias a la descarga de AutoDataCleaner.

4.4.1.9 NumPy

La librería NumPy es una librería de Python centrada en el cálculo numérico y en el análisis de datos.

En NumPy hay una nueva clase de objetos llamados arrays. La ventaja de los arrays es su rápida velocidad de procesamiento.

Esta librería viene instalada con su correspondiente versión (1.21.6) gracias a la descarga de AutoDataCleaner.

4.4.1.10 Scikit-learn

Scikit-learn es una librería open source de machine learning que maneja tanto aprendizaje supervisado como el no supervisado.

Contiene más de 12 paquetes con distintos fines, uno de ellos es el paquete *preprocessing*.

Su instalación se hace a través de conda con la siguiente línea de código.

```
conda install -c anaconda scikit-learn
```

4.4.1.10.1 Preprocessing

El paquete preprocessing de scikit learn será de gran utilidad ya que transforma los datos rudos en datos con posibilidad de tratamiento.

Las librerías contenidas en `sklearn.preprocessing` se muestran a continuación (Ilustración 11 – librerías dentro de sklearn. preprocessing).

```
from ._function_transformer import FunctionTransformer

from ._data import Binarizer
from ._data import KernelCenterer
from ._data import MinMaxScaler
from ._data import MaxAbsScaler
from ._data import Normalizer
from ._data import RobustScaler
from ._data import StandardScaler
from ._data import QuantileTransformer
from ._data import add_dummy_feature
from ._data import binarize
from ._data import normalize
from ._data import scale
from ._data import robust_scale
from ._data import maxabs_scale
from ._data import minmax_scale
from ._data import quantile_transform
from ._data import power_transform
from ._data import PowerTransformer

from ._encoders import OneHotEncoder
from ._encoders import OrdinalEncoder

from ._label import label_binarize
from ._label import LabelBinarizer
from ._label import LabelEncoder
from ._label import MultiLabelBinarizer

from ._discretization import KBinsDiscretizer

from ._polynomial import PolynomialFeatures
from ._polynomial import SplineTransformer
```

Ilustración 11 – librerías dentro de sklearn. preprocessing

4.4.1.11 Matplotlib

Matplotlib es una librería para crear visualizaciones y trazados gráficos de datos en Python. Se instala desde la consola como:

```
conda install matplotlib
```

4.4.1.12 Feature-Engine

Esta librería "open source" creada por Soledad Galli, Nicolas Galli, Chris Samiullah en 2017. Fue creada con la intención de acortar el tiempo de preprocesado de datos y por ellos, crearon una librería con los métodos más empleados en esta área. Se ha decidido cargar esta librería para hacer uso de las funciones Outlier Handling, DropDuplicatedFeatures, DropCorrelatedFeatures y DecisionTreeEncoder. Estas funciones descritas en los siguientes libreria, proporcionan unos comandos claves para el buen procesamiento de los datos. La librería incluye las funciones `fit ()` y `transform()` proporcionadas por sklearn. Con estas

dos funciones ya incorporadas, se puede encontrar las variables a través del comando `fit` y eliminar esas variables a través del comando `transform`.

La siguiente línea de código se implementará en el cmd.exe, expresamente en el nuevo entorno.

```
conda install -c conda-forge feature_engine
```

Se usarán las siguientes funciones y se automatizarán.

4.4.1.12.1 Outlier Handling

Los dos métodos que se analizaran para tratar los valores atípicos en una base de datos con librería Feature-engine se presentaran a continuación. El método de ArbitraryOutlierCapping no se ha considerado ya que se cambiarían los valores atípicos por un valor en concentro especificado por el usuario y, por tanto, la imposibilidad de su automatización.

El uso del comando `time. time()` ha sido empleado en cada método con el fin de escoger el programa más rápido. A continuación, se muestra la línea de código para cargar la librería `time. time().`

```
Import time
Start=time. Time()
…
End=time. Time()
Time_of_execution=End-Start
```

Winsorizer

Winsorizer se encarga de la detección de los valores máximos y mínimos de una variable, su limitación de los límites máximos y mínimos [20] y de su correspondiente cambio. Esta función es solo aplicable a variables numéricas y puede ser aplicada a una variable predeterminada o por defecto, la función seleccionará todas las variables del conjunto.

Su funcionamiento interno es el siguiente, primero se calcula los valores topes de la distribución siguiendo una forma de las tres mencionadas en Tabla 1 – Formas de calcular los máximos y mínimos en winsorizer.

	Gaussian	**IQR**	**Percentiles**
Cola derecha	$mean + 3 \cdot \sigma$[9]	$75th\ quantile + 3 \cdot IQR$	$95th\ percerntile$
Cola izquierda	$mean - 3 \cdot \sigma$	$25th\ quantile - 3 \cdot IQR$	$5th\ percerntile$

Tabla 1 – Formas de calcular los máximos y mínimos en winsorizer.

Cada forma es válida pero la que se ve menos afectada por outliers debido a su cálculo, es el rango Inter cuartil (IQR).

El API [10] se muestra a continuación.

```
feature_engine.outliers.Winsorizer(capping_method='gaussian', tail='right', fold=
3, variables=None, missing_values='raise')
```

Los parámetros incluidos en esta función se presentan a continuación (Tabla 2 – Parámetros y funcionalidades de la función Winsorizer).

Parámetros	**Funcionalidad**
capping_method	Elegir el método para hacer el proceso de winsorizer (guassian, IQR o quantiles)
tail	Eliminar los valores atípicos por la cola izquierda, derecha o ambas

[9] $\sigma = desviacion\ estandart$ es una medida que ofrece información sobre la dispersión media de una variable. [47, 62, 62, 62, 62]

[10] Application Programming Interface, por sus siglas en ingles API, permite que dos sistemas informáticos interactúen entre sí. [62]

fold	Se refiere al número el cual multiplicara a la desviación estándar o al rango Inter cuartil (Por defecto es 3) Se recomienda entre 2 y 3 para el método gaussiano y entre 1,5 a 3 para el rango Inter cuartil.
variables	Ofrece la posibilidad de eliminar los valores atípicos solo a unas cuantas variables (Por defecto se aplican a todas).
missing_values	Ofrece la oportunidad de que se muestre un error cuando una variable contenga un valor faltante (missing_values = 'raise')

Tabla 2 – Parámetros y funcionalidades de la función Winsorizer

OutlierTrimmer

OutlierTrimmer detecta las observaciones con valores atípicos del conjunto de datos y procede a cambiarlos por valores más céntricos. Esta función es solo aplicable a variables numéricas y puede ser aplicada a una variable predeterminada o por defecto, la función seleccionara todas las variables del conjunto.

Su funcionamiento interno es el siguiente, primero se calcula los valores topes de la distribución siguiendo una forma de las tres mencionadas en la Tabla 1 – Formas de calcular los máximos y mínimos en winsorizer..

Cada forma es válida pero la que se ve menos afectada por outliers debido a su cálculo, es el rango Inter cuantil (IQR).
El API se muestra a continuación.

```
feature_engine.outliers.OutlierTrimmer(capping_method='gaussian', tail='right', f
old=3, variables=None, missing_values='raise')
```

Los parámetros incluidos en esta función se han presentado previamente en la Tabla 2 – Parámetros y funcionalidades de la función Winsorizer.

4.4.1.12.2 DropedCorrelatedFeature

La función DropCorrelatedFeature proviene de la librería selección de feature_engine. Se encarga de encontrar y eliminar las variables altamente correlacionadas con el método primer encontrado, primer eliminado y existe la posibilidad de indicar el nivel de correlación a considerar. Esta función acepta valores faltantes. Los valores de tipo categórico no pueden ser procesados por esta función y por tanto se excluirán del análisis.

El API se muestra a continuación.

```
Feature_engine.selection.DropCorrelatedFeatures(variables=None,method='pearson';
threshold=0.8, missing_values='ignore')
```

Los parámetros incluidos en esta función se presentan a continuación (Tabla 3 – Parámetros y funcionalidades de la función DropCorrelatedFeatures)

Parámetros	**Funcionalidad**
variables	Ofrece la posibilidad de eliminar los valores atípicos solo a unas cuantas variables (Por defecto se aplican a todas).
método	Ofrece la posibilidad de elección del método (Pearson, kendall o spearman) propuesto para identificar las variables correlacionadas.
threshold	Valor por el cual se consideran correlacionadas. Por defecto se ha instaurado 0.8
missing_values	Ofrece la oportunidad de que se muestre un error cuando una variable contenga un valor faltante (missing_values = 'raise'

Tabla 3 – Parámetros y funcionalidades de la función DropCorrelatedFeatures

4.4.1.12.3 DropConstantFeatures

La función DropConstantFeature proviene de la librería selección de feature_engine. Se encarga de encontrar y eliminar las variables con poca o nula varianza y existe la posibilidad

de indicar el nivel de semejanza a considerar. Esta función acepta valores faltantes. Los valores de tipo categórico pueden ser procesados por esta función.

El API se muestra a continuación:

```
Feature_engine.selection.DropConstantFeatures(tol=1,variables=None,missing_values
='ignore')
```

Los parámetros incluidos en esta función se presentan a continuación (Tabla 4 – Parámetros y funcionalidades de la función DropConstantFeatures)

Parámetros	**Funcionalidad**
tol	Umbral para detectar si las variables son constantes o casi constantes dentro de una columna
variables	Ofrece la posibilidad de eliminar los valores atípicos solo a unas cuantas variables (Por defecto se aplican a todas).
missing_values	Ofrece la oportunidad de que se muestre un error cuando una variable contenga un valor faltante (missing_values = 'raise'

Tabla 4 – Parámetros y funcionalidades de la función DropConstantFeatures

4.4.1.12.4 DropDuplicatedFeatures

La función DropDuplicatedFeatures proviene de la librería *selección* de feature_engine. Se encarga de encontrar y seguidamente, de eliminar las variables duplicadas y funciona con valores categóricos como valores faltantes. La condición es que todas las observaciones deben de tener el mismo valor para que se consideren duplicadas.

El API se muestra a continuación.

```
Feature_engine.selection.DropDuplicatedFeatures(variables=None,
missing_values='ignore')
```

Los parámetros que se muestra en la Tabla 5 – Parámetros y Funcionalidad de la función DropDuplicatedFeatures corresponden a esta DropDuplicatedFeatures.

Parámetros	**Funcionalidad**
variable	Ofrece la posibilidad de eliminar los valores atípicos solo a unas cuantas variables (Por defecto se aplican a todas).
missing_values	Ofrece la oportunidad de que se muestre un error cuando una variable contenga un valor faltante (missing_values = 'raise'

Tabla 5 – Parámetros y Funcionalidad de la función DropDuplicatedFeatures

4.4.1.12.5 DecisionTreeaEncoder

La función DecisionTreeDiscretiser proviene de la librería *discretisation* de Feature_engine. Se encarga de remplazar las variables numéricas continuas por variables discretas estimadas por un árbol de decisión[11]. Solo funciona con valores numéricos.

Su implementación es debida a que es de gran utilidad para los analistas de datos ya que muestran cómo afectan unas decisiones a todo el proceso. Permite comparar diferentes decisiones y acciones. Por ejemplo, una variable continua donde haya que categorizar, se puede remplazar por sus rangos y establecer las categorías, como ocurre en las variables de riesgo con sus categorías de riesgo alto y bajo.

La API se muestra a continuación.

```
Feature_engine.encoding.DecisionTreeEncoder(encoding_method='arbitrary',cv=3,scor
ing='neg_mean_squared_error',param_grid=None,regression=True,random_state=None,
variables=None)
```

Los parámetros incluidos en esta función se presentan a continuación (Tabla 6 – Parámetros y funcionalidad de la función DecisionTreeEncoder).

Parámetros	**Funcionalidad**
encoding_method	El método que se usara para codificar las categorías en valores numéricos.

[11] "Un árbol de decisión es un mapa de los posibles resultados de una serie de decisiones relacionadas." [62]

cv	Numero de validaciones cruzadas que se utilizaran para ajustar el árbol de decisión.
scoring	Métrica deseada para optimizar el rendimiento del árbol.
variables	Ofrece la posibilidad de eliminar los valores atípicos solo a unas cuantas variables (Por defecto se aplican a todas).
param_grid	Lista de parámetros sobre los que el árbol de decisión debe de ser optimizado durante la búsqueda de la cuadricula
regresión	Si el discretizador debe entrenar una regresión o una clasificación
random_state	Inicializar el entrenamiento del árbol de decisión

Tabla 6 – Parámetros y funcionalidad de la función DecisionTreeEncoder

4.5 KAGGLE

Kaggle es una plataforma web que permite a los usuarios encontrar y publicar conjunto de datos, crear modelos, trabajar con científicos de datos e ingenieros de IA con posibilidad de participación en concursos sobre desafíos de ciencias de datos. [21].

Se usará Kaggle para encontrar el conjunto de datos y gracias a sus desafíos, se podrá comprobar la exactitud de la función desarrollada sobre las librerías AutoDataCleaner, feature_engine y DataCleaner. Se usará su método de puntuación para comprobar los data sets siguientes.

En el Capítulo 8. se explicará sus distintas características.

4.6 OPENML

Open Media Library por sus siglas OpenML es una plataforma web de código abierto para compartir conjuntos de datos, algoritmos y experimentos. [22]. OpenML se ha usado para poder usar el *Titanic DataSet.*

UCI machine learning DataSet

UCI machine learning DataSet es una Plataforma de credit screening[12] donde se puede encontrar conjuntos de datos abiertos para el machine learning. [23]

De esta plataforma web, se descargará el conjunto de datos llamado *Credit Approval dataset.*

En el Capítulo 8. se explicará sus distintas características.

[12] Credit screening es el proceso de recopilación de información de agencias de crédito y de registros públicos para verificar la información presentada y para poder evaluar la estabilidad financiera y la fiabilidad de los arrendatarios. [62]

Capítulo 5. CREACIÓN DEL ENTORNO VIRTUAL

La optimización de variables de entrada en los modelos de machine learning es de vital importancia para la comunidad científica ya que, una buena optimización reduce costes. Dichos costes son tanto de personal como de máquina. Una buena optimización reduce el riesgo de crear un modelo inexacto y en muchos casos, completamente erróneo.

Por ello partiremos de un entorno virtual nuevo llamado OptimizacionVarEntrada a través del cmd.exe. El código siguiente muestra su creación y los paquetes importados del entorno base.

```
(base) C:\Users\Usuario>conda create -n OptimizacionVarEntrada python=3.7
Continue creating environment (y/[n])? y

Collecting package metadata (current_repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: C:\Users\Usuario\anaconda3\envs\OptimizacionVarEntrada

  added / updated specs:
    - python=3.7


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    certifi-2022.5.18.1        |   py37haa95532_0         157 KB
    ------------------------------------------------------------
                                           Total:         157 KB

The following NEW packages will be INSTALLED:

  ca-certificates    pkgs/main/win-64::ca-certificates-2022.4.26-haa95532_0
  certifi            pkgs/main/win-64::certifi-2022.5.18.1-py37haa95532_0
```

```
  openssl            pkgs/main/win-64::openssl-1.1.1o-h2bbff1b_0
  pip                pkgs/main/win-64::pip-21.2.4-py37haa95532_0
  python             pkgs/main/win-64::python-3.7.13-h6244533_0
  setuptools         pkgs/main/win-64::setuptools-61.2.0-py37haa95532_0
  sqlite             pkgs/main/win-64::sqlite-3.38.3-h2bbff1b_0
  vc                 pkgs/main/win-64::vc-14.2-h21ff451_1
  vs2015_runtime     pkgs/main/win-64::vs2015_runtime-14.27.29016-h5e58377_2
  wheel              pkgs/main/noarch::wheel-0.37.1-pyhd3eb1b0_0
  wincertstore       pkgs/main/win-64::wincertstore-0.2-py37haa95532_2


Proceed ([y]/n)? y


Downloading and Extracting Packages
certifi-2022.5.18.1              |       157       KB                                |
################################################################################
| 100%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
#
# To activate this environment, use
#
#     $ conda activate OptimizacionVarEntrada
#
# To deactivate an active environment, use
#
#     $ conda deactivate


(base) C:\Users\Usuario>conda activate OptimizacionVarEntrada
```

Para tener una visión de las librerías importadas, se ha procedido con el siguiente código después de la creación del entorno. Como se puede observar, las librerías típicas como pandas, NumPy, Scikit-Learn y matplot, entre otras, no están presentes. Las librerías que se pueden observar son las cargadas de la librería base (Lib).

```
(OptimizacionVarEntrada) C:\Users\Usuario>conda list
# packages in environment at
C:\Users\Usuario\anaconda3\envs\OptimizacionVarEntrada:
#
# Name                    Version                   Build  Channel
ca-certificates           2022.4.26            haa95532_0
certifi                   2022.5.18.1       py37haa95532_0
openssl                   1.1.1o               h2bbff1b_0
pip                       21.2.4            py37haa95532_0
python                    3.7.13               h6244533_0
setuptools                61.2.0            py37haa95532_0
sqlite                    3.38.3               h2bbff1b_0
vc                        14.2                 h21ff451_1
vs2015_runtime            14.27.29016          h5e58377_2
wheel                     0.37.1             pyhd3eb1b0_0
wincertstore              0.2               py37haa95532_2
```

El mensaje siguiente (Ilustración 12 – Consola tras el entorno creado) aparece tras cambiar del entorno base al entorno deseado y nos permite cambiar del entorno base al entorno deseado.

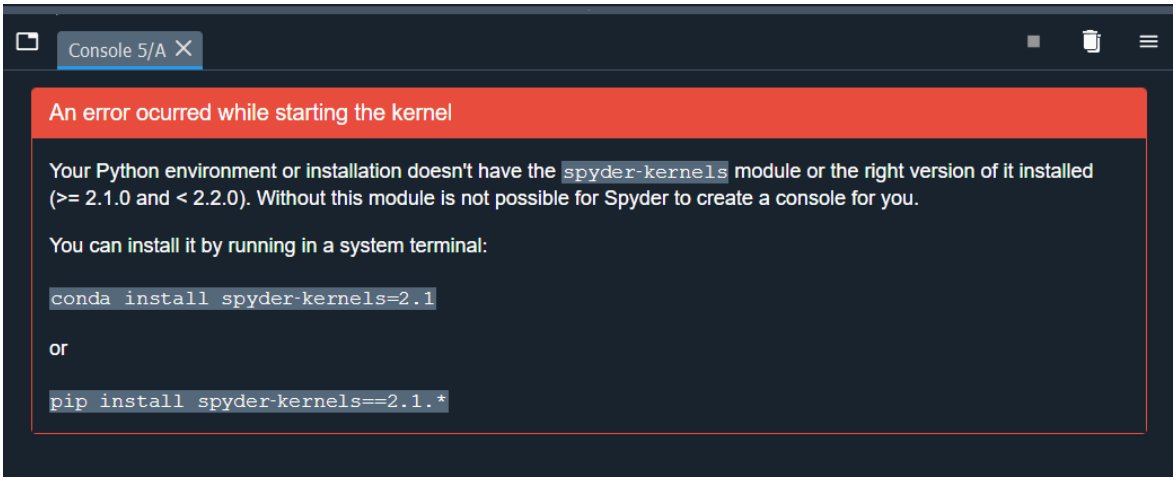


Ilustración 12 – Consola tras el entorno creado

Como se ha mencionado en Librerías, la primera librería necesaria para el nuevo entorno es la librería que conecta el kernel con la consola en spyder. El código mostrado a continuación presenta su instalación a través del comando conda y las librerías importadas vinculadas a `spyder-kernel`.

```
(OptimizacionVarEntrada) C:\Users\Usuario>conda install spyder-kernels=2.1
Collecting package metadata (current_repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: C:\Users\Usuario\anaconda3\envs\OptimizacionVarEntrada

  added / updated specs:
    - spyder-kernels=2.1


The following NEW packages will be INSTALLED:

  backcall           pkgs/main/noarch::backcall-0.2.0-pyhd3eb1b0_0
  cloudpickle        pkgs/main/noarch::cloudpickle-2.0.0-pyhd3eb1b0_0
  colorama           pkgs/main/noarch::colorama-0.4.4-pyhd3eb1b0_0
  debugpy            pkgs/main/win-64::debugpy-1.5.1-py37hd77b12b_0
  decorator          pkgs/main/noarch::decorator-5.1.1-pyhd3eb1b0_0
  ipykernel          pkgs/main/win-64::ipykernel-6.9.1-py37haa95532_0
  ipython            pkgs/main/win-64::ipython-7.31.1-py37haa95532_0
  jedi               pkgs/main/win-64::jedi-0.18.1-py37haa95532_1
  jupyter_client     pkgs/main/noarch::jupyter_client-6.1.12-pyhd3eb1b0_0
  jupyter_core       pkgs/main/win-64::jupyter_core-4.10.0-py37haa95532_0
  matplotlib-inline  pkgs/main/noarch::matplotlib-inline-0.1.2-pyhd3eb1b0_2
  nest-asyncio       pkgs/main/win-64::nest-asyncio-1.5.5-py37haa95532_0
  parso              pkgs/main/noarch::parso-0.8.3-pyhd3eb1b0_0
  pickleshare        pkgs/main/noarch::pickleshare-0.7.5-pyhd3eb1b0_1003
  prompt-toolkit     pkgs/main/noarch::prompt-toolkit-3.0.20-pyhd3eb1b0_0
  pygments           pkgs/main/noarch::pygments-2.11.2-pyhd3eb1b0_0
  python-dateutil    pkgs/main/noarch::python-dateutil-2.8.2-pyhd3eb1b0_0
  pywin32            pkgs/main/win-64::pywin32-302-py37h2bbff1b_2
  pyzmq              pkgs/main/win-64::pyzmq-22.3.0-py37hd77b12b_2
  six                pkgs/main/noarch::six-1.16.0-pyhd3eb1b0_1
  spyder-kernels     pkgs/main/win-64::spyder-kernels-2.1.3-py37haa95532_0
  tornado            pkgs/main/win-64::tornado-6.1-py37h2bbff1b_0
  traitlets          pkgs/main/noarch::traitlets-5.1.1-pyhd3eb1b0_0
  wcwidth            pkgs/main/noarch::wcwidth-0.2.5-pyhd3eb1b0_0


Proceed ([y]/n)? y
```

```
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
```

Una vez instalada dicha librería, el siguiente mensaje aparece por consola (Ilustración 13 – Consola después de la instalación de spyder-kernel).

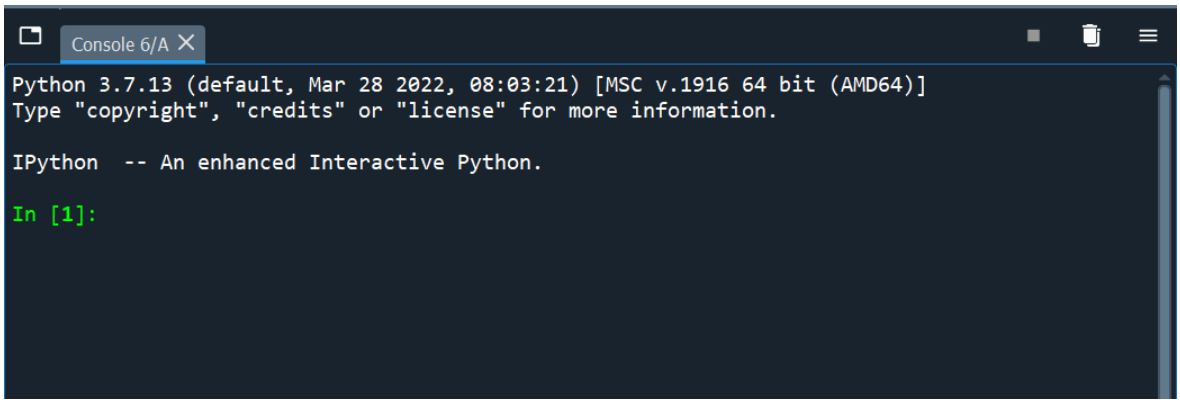


Ilustración 13 – Consola después de la instalación de spyder-kernel

La siguiente librería que se instala por el cmd.exe será AutoDataCleaner en el entorno creado. A Continuación, se muestra la consola.

```
(OptimizacionVarEntrada) C:\Users\Usuario>pip install AutoDataCleaner
Collecting AutoDataCleaner
  Using cached AutoDataCleaner-1.1.3-py3-none-any.whl
Collecting pandas
  Downloading pandas-1.3.5-cp37-cp37m-win_amd64.whl (10.0 MB)
     |████████████████████████████████| 10.0 MB 2.2 MB/s
Collecting numpy>=1.17.3
  Downloading numpy-1.21.6-cp37-cp37m-win_amd64.whl (14.0 MB)
     |████████████████████████████████| 14.0 MB 29 kB/s
Requirement already satisfied: python-dateutil>=2.7.3 in
c:\users\usuario\anaconda3\envs\optimizacionvarentrada\lib\site-packages (from
pandas->AutoDataCleaner) (2.8.2)
Collecting pytz>=2017.3
  Using cached pytz-2022.1-py2.py3-none-any.whl (503 kB)
Requirement already satisfied: six>=1.5 in
c:\users\usuario\anaconda3\envs\optimizacionvarentrada\lib\site-packages (from
python-dateutil>=2.7.3->pandas->AutoDataCleaner) (1.16.0)
Installing collected packages: pytz, numpy, pandas, AutoDataCleaner
Successfully installed AutoDataCleaner-1.1.3 numpy-1.21.6 pandas-1.3.5 pytz-2022.1
```

Es interesante observar las librerías que conlleva la descarga de AutoDataCleaner. Las siguientes líneas de código presentan la información obtenida a través de `conda lista` .

```
(OptimizacionVarEntrada) C:\Users\Usuario>conda list
#               packages               in               environment               at
C:\Users\Usuario\anaconda3\envs\OptimizacionVarEntrada:
#
# Name                    Version                   Build  Channel
autodatacleaner           1.1.3                     pypi_0    pypi
backcall                  0.2.0              pyhd3eb1b0_0
ca-certificates           2022.4.26            haa95532_0
certifi                   2022.5.18.1       py37haa95532_0
cloudpickle               2.0.0              pyhd3eb1b0_0
colorama                  0.4.4              pyhd3eb1b0_0
debugpy                   1.5.1             py37hd77b12b_0
decorator                 5.1.1              pyhd3eb1b0_0
ipykernel                 6.9.1             py37haa95532_0
ipython                   7.31.1            py37haa95532_0
jedi                      0.18.1            py37haa95532_1
jupyter_client            6.1.12             pyhd3eb1b0_0
jupyter_core              4.10.0            py37haa95532_0
matplotlib-inline         0.1.2              pyhd3eb1b0_2
nest-asyncio              1.5.5             py37haa95532_0
numpy                     1.21.6                    pypi_0    pypi
openssl                   1.1.1o               h2bbff1b_0
pandas                    1.3.5                     pypi_0    pypi
parso                     0.8.3              pyhd3eb1b0_0
pickleshare               0.7.5            pyhd3eb1b0_1003
pip                       21.2.4            py37haa95532_0
prompt-toolkit            3.0.20             pyhd3eb1b0_0
pygments                  2.11.2             pyhd3eb1b0_0
python                    3.7.13               h6244533_0
python-dateutil           2.8.2              pyhd3eb1b0_0
pytz                      2022.1                    pypi_0    pypi
pywin32                   302               py37h2bbff1b_2
pyzmq                     22.3.0            py37hd77b12b_2
setuptools                61.2.0            py37haa95532_0
six                       1.16.0             pyhd3eb1b0_1
spyder-kernels            2.1.3             py37haa95532_0
```

```
sqlite                    3.38.3               h2bbff1b_0
tornado                   6.1               py37h2bbff1b_0
traitlets                 5.1.1               pyhd3eb1b0_0
vc                        14.2                 h21ff451_1
vs2015_runtime            14.27.29016          h5e58377_2
wcwidth                   0.2.5               pyhd3eb1b0_0
wheel                     0.37.1              pyhd3eb1b0_0
wincertstore              0.2               py37haa95532_2
```

Se puede observar que la librería NumPy junto con la librería pandas han sido instaladas con su versión. Las librerías faltantes son scikit-learn y matplotlib.

La instalación de scikit-learn se procede como se ha mencionado en el apartado 4.4.1.10. A continuación, se muestra su instalación realizada a través de la consola.

```
(OptimizacionVarEntrada) C:\Users\Usuario>conda install -c anaconda scikit-learn
Collecting package metadata (current_repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: C:\Users\Usuario\anaconda3\envs\OptimizacionVarEntrada

  added / updated specs:
    - scikit-learn


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    blas-1.0                   |              mkl           6 KB  anaconda
    ca-certificates-2022.4.26  |       haa95532_0         163 KB  anaconda
    certifi-2021.10.8          |   py37haa95532_2         156 KB  anaconda
    icc_rt-2019.0.0            |       h0cc432a_1         9.4 MB  anaconda
    intel-openmp-2021.4.0      |    haa95532_3556         3.2 MB  anaconda
    joblib-1.1.0               |     pyhd3eb1b0_0         211 KB  anaconda
    mkl-2021.4.0               |     haa95532_640       181.6 MB  anaconda
    mkl-service-2.4.0          |   py37h2bbff1b_0          55 KB  anaconda
```

```
    mkl_fft-1.3.1              |   py37h277e83a_0         154 KB  anaconda
    mkl_random-1.2.2           |   py37hf11a4ad_0         249 KB  anaconda
    numpy-1.21.5               |   py37h7a0a035_2          24 KB  anaconda
    numpy-base-1.21.5          |   py37hca35cd5_2         5.6 MB  anaconda
    openssl-1.1.1n             |       h2bbff1b_0         5.8 MB  anaconda
    scikit-learn-1.0.2         |   py37hf11a4ad_1         6.8 MB  anaconda
    scipy-1.7.3                |   py37h0a974cb_0        17.7 MB  anaconda
    threadpoolctl-2.2.0        |     pyh0d69192_0          16 KB  anaconda
    ------------------------------------------------------------
                                           Total:       231.1 MB

The following NEW packages will be INSTALLED:

  blas               anaconda/win-64::blas-1.0-mkl
  icc_rt             anaconda/win-64::icc_rt-2019.0.0-h0cc432a_1
  intel-openmp       anaconda/win-64::intel-openmp-2021.4.0-haa95532_3556
  joblib             anaconda/noarch::joblib-1.1.0-pyhd3eb1b0_0
  mkl                anaconda/win-64::mkl-2021.4.0-haa95532_640
  mkl-service        anaconda/win-64::mkl-service-2.4.0-py37h2bbff1b_0
  mkl_fft            anaconda/win-64::mkl_fft-1.3.1-py37h277e83a_0
  mkl_random         anaconda/win-64::mkl_random-1.2.2-py37hf11a4ad_0
  numpy              anaconda/win-64::numpy-1.21.5-py37h7a0a035_2
  numpy-base         anaconda/win-64::numpy-base-1.21.5-py37hca35cd5_2
  scikit-learn       anaconda/win-64::scikit-learn-1.0.2-py37hf11a4ad_1
  scipy              anaconda/win-64::scipy-1.7.3-py37h0a974cb_0
  threadpoolctl      anaconda/noarch::threadpoolctl-2.2.0-pyh0d69192_0

The following packages will be SUPERSEDED by a higher-priority channel:

  ca-certificates                                  pkgs/main --> anaconda
  certifi            pkgs/main::certifi-2022.5.18.1-py37ha~ --> anaconda::certifi-
2021.10.8-py37haa95532_2
  openssl              pkgs/main::openssl-1.1.1o-h2bbff1b_0 --> anaconda::openssl-
1.1.1n-h2bbff1b_0


Proceed ([y]/n)? y


Downloading and Extracting Packages
[…]
```

```
Preparing transaction: done
Verifying transaction: done
Executing transaction: \

    Windows 64-bit packages of scikit-learn can be accelerated using scikit-learn-
intelex.
    More details are available here: https://intel.github.io/scikit-learn-intelex

    For example:

        $ conda install scikit-learn-intelex
        $ python -m sklearnex my_application.py


Done
```

La configuración siguiente para tener el entorno virtual listo para su uso es tras la instalación de matplotlib como se mencionó en 4.4.1.11.

```
(OptimizacionVarEntrada) C:\Users\Usuario>conda install matplotlib
Collecting package metadata (current_repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: C:\Users\Usuario\anaconda3\envs\OptimizacionVarEntrada

  added / updated specs:
    - matplotlib


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    ca-certificates-2022.4.26  |       haa95532_0         124 KB
    kiwisolver-1.4.2           |   py37hd77b12b_0          58 KB
    libtiff-4.2.0              |       he0120a3_1         754 KB
    matplotlib-3.5.1           |   py37haa95532_1          29 KB
    matplotlib-base-3.5.1      |   py37hd77b12b_1         5.6 MB
```

```
    pillow-9.0.1               |   py37hdc2b20a_0         916 KB
    pyqt-5.9.2                 |   py37hd77b12b_6         3.3 MB
    sip-4.19.13                |   py37hd77b12b_0         260 KB
    zstd-1.5.2                 |       h19a0ad4_0         509 KB
    ------------------------------------------------------------
                                           Total:        11.5 MB

The following NEW packages will be INSTALLED:

  brotli             pkgs/main/win-64::brotli-1.0.9-ha925a31_2
  cycler             pkgs/main/noarch::cycler-0.11.0-pyhd3eb1b0_0
  fonttools          pkgs/main/noarch::fonttools-4.25.0-pyhd3eb1b0_0
  freetype           pkgs/main/win-64::freetype-2.10.4-hd328e21_0
  icu                pkgs/main/win-64::icu-58.2-ha925a31_3
  jpeg               pkgs/main/win-64::jpeg-9e-h2bbff1b_0
  kiwisolver         pkgs/main/win-64::kiwisolver-1.4.2-py37hd77b12b_0
  libpng             pkgs/main/win-64::libpng-1.6.37-h2a8f88b_0
  libtiff            pkgs/main/win-64::libtiff-4.2.0-he0120a3_1
  libwebp            pkgs/main/win-64::libwebp-1.2.2-h2bbff1b_0
  lz4-c              pkgs/main/win-64::lz4-c-1.9.3-h2bbff1b_1
  matplotlib         pkgs/main/win-64::matplotlib-3.5.1-py37haa95532_1
  matplotlib-base    pkgs/main/win-64::matplotlib-base-3.5.1-py37hd77b12b_1
  munkres            pkgs/main/noarch::munkres-1.1.4-py_0
  packaging          pkgs/main/noarch::packaging-21.3-pyhd3eb1b0_0
  pillow             pkgs/main/win-64::pillow-9.0.1-py37hdc2b20a_0
  pyparsing          pkgs/main/noarch::pyparsing-3.0.4-pyhd3eb1b0_0
  pyqt               pkgs/main/win-64::pyqt-5.9.2-py37hd77b12b_6
  qt                 pkgs/main/win-64::qt-5.9.7-vc14h73c81de_0
  sip                pkgs/main/win-64::sip-4.19.13-py37hd77b12b_0
  tk                 pkgs/main/win-64::tk-8.6.12-h2bbff1b_0
  typing_extensions  pkgs/main/noarch::typing_extensions-4.1.1-pyh06a4308_0
  xz                 pkgs/main/win-64::xz-5.2.5-h8cc25b3_1
  zlib               pkgs/main/win-64::zlib-1.2.12-h8cc25b3_2
  zstd               pkgs/main/win-64::zstd-1.5.2-h19a0ad4_0

The following packages will be UPDATED:

  certifi           anaconda::certifi-2021.10.8-py37haa95~ --> pkgs/main::certifi-
2022.5.18.1-py37haa95532_0
  openssl             anaconda::openssl-1.1.1n-h2bbff1b_0 --> pkgs/main::openssl-
1.1.1o-h2bbff1b_0
```

```
The following packages will be SUPERSEDED by a higher-priority channel:

  ca-certificates                                     anaconda --> pkgs/main


Proceed ([y]/n)? y


Downloading and Extracting Packages
pyqt-5.9.2                                          |    3.3    MB               |
################################################################################
| 100%
libtiff-4.2.0                                       |     754     KB               |
################################################################################
| 100%
matplotlib-base-3.5.        |        5.6        MB                                |
################################################################################
| 100%
zstd-1.5.2                                          |    509    KB               |
################################################################################
| 100%
matplotlib-3.5.1                            |      29     KB                        |
################################################################################
| 100%
pillow-9.0.1                                        |    916    KB               |
################################################################################
| 100%
ca-certificates-2022        |        124        KB                                |
################################################################################
| 100%
kiwisolver-1.4.2                            |      58     KB                        |
################################################################################
| 100%
sip-4.19.13                                         |    260    KB               |
################################################################################
| 100%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
```

El último paquete incorporado al entorno es feature-engine, su función ha sido mencionada en el apartado Feature-Engine

```
(OptimizacionVarEntrada)    C:\Users\Usuario>conda    install    -c    conda-forge
feature_engine
Collecting package metadata (current_repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: C:\Users\Usuario\anaconda3\envs\OptimizacionVarEntrada

  added / updated specs:
    - feature_engine


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    statsmodels-0.13.2         |   py37h3a130e4_0        10.5 MB  conda-forge
    ------------------------------------------------------------
                                           Total:        10.5 MB

The following NEW packages will be INSTALLED:

  bottleneck         conda-forge/win-64::bottleneck-1.3.4-py37h3a130e4_1
  feature_engine     conda-forge/noarch::feature_engine-1.3.0-pyhd8ed1ab_0
  numexpr            pkgs/main/win-64::numexpr-2.8.1-py37hb80d3ca_0
  pandas             pkgs/main/win-64::pandas-1.3.5-py37h6214cd6_0
  patsy              conda-forge/noarch::patsy-0.5.2-pyhd8ed1ab_0
  python_abi         conda-forge/win-64::python_abi-3.7-2_cp37m
  pytz               conda-forge/noarch::pytz-2022.1-pyhd8ed1ab_0
  statsmodels        conda-forge/win-64::statsmodels-0.13.2-py37h3a130e4_0

The following packages will be UPDATED:

  ca-certificates     pkgs/main::ca-certificates-2022.4.26-~ --> conda-forge::ca-
certificates-2022.5.18.1-h5b45459_0
```

```
The following packages will be SUPERSEDED by a higher-priority channel:

  certifi                     pkgs/main::certifi-2022.5.18.1-py37ha~  -->  conda-
forge::certifi-2022.5.18.1-py37h03978a9_0
  openssl           pkgs/main::openssl-1.1.1o-h2bbff1b_0 --> conda-forge::openssl-
1.1.1o-h8ffe710_0


Proceed ([y]/n)? y


Downloading and Extracting Packages
statsmodels-0.13.2                  |       10.5       MB                    |
####################################################################################     |
100%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
```

Una vez finalizado la creación del entorno, se procede a importar las librerías anteriormente mencionadas en un nuevo script.

La ejecución será primero sobre el código ejemplo de AutoDataCleaner para observar si funciona o si hay alguna librería faltante, después, se procederá a modificar las librerías para incorporar otras funciones de pre procesado.

Capítulo 6. AUTODATACLEANER

Una vez creado el entorno virtual OptimizacionVarEntrada, se procederá con el código llamado "Quick One-line usage" [10]. Se procederá con el siguiente código, para ver el funcionamiento de AutoDataCleaner, observar sus puntos débiles, ver cómo funciona la librería internamente, también para familiarizarse con esta librería y así explotar sus funcionalidades al máximo.

```
import pandas as pd
import AutoDataCleaner.AutoDataCleaner as adc

df = pd.DataFrame([
    [1, "Male", "white", 3, "2018/11/20"],
    [2, "Female", "blue", "4", "2014/01/12"],
    [3, "Male", "white", 15, "2020/09/02"],
    [4, "Male", "blue", "5", "2020/09/02"],
    [5, "Male", "green", None, "2020/12/30"]],
    columns=['id', 'gender', 'color', 'weight', 'created_on'])

adc.clean_me(df,
             detect_binary=True,
             numeric_dtype=True,
             one_hot=True,
             na_cleaner_mode="mode",
             normalize=True,
             datetime_columns=["created_on"],
             remove_columns=["id"],
             verbose=True)

```

Ilustración 14 – Uso sencillo de AutoDataCleaner

La Ilustración 14 – Uso sencillo de AutoDataCleaner muestra una llamada a la función `clean_me ()` que se encuentra dentro de AutoDataCleaner.AutoDataCleaner.

Esta función tiene varios parámetros que se muestran en la Tabla 7 – Parámetros y funcionalidades de clean_me.

Parámetros	**Funcionalidad**
df	DataFrame con los datos a limpiar
detect_binary	Detección automática de variables binarias y remplazar sus valores por 0 y 1
numeric_dtype	Las variables se convertirán en numeric dtypes

one_hot	Las variables no numéricas se convertirán a one hot enconding
na_cleaner_mode	Que técnica usar cuando se trabaja con valores faltantes. Hay tres opciones (remove_row, mean, mode)
normalize	Todas las columnas no binarias se normalizarán
datetime_columns	Lista de columnas con valores del tipo fecha y hora
remove_columns	Lista de columnas para eliminar
verbose	Imprime los cambios a medida que se van realizando en los datos

Tabla 7 – Parámetros y funcionalidades de clean_me

El DataFrame creado se representa de la siguiente manera (Ilustración 15 – Representación del DataFrame (df) antes de la ejecución).

	id	gender	color	weight	created_on
0	1	Male	white	3	2018/11/20
1	2	Female	blue	4	2014/01/12
2	3	Male	white	15	2020/09/02
3	4	Male	blue	5	2020/09/02
4	5	Male	green	None	2020/12/30

Ilustración 15 – Representación del DataFrame (df) antes de la ejecución

La Ilustración 15 – Representación del DataFrame (df) antes de la ejecución muestra que la columna *gender* es una columna con características binarias, con dos cadenas de texto solución ("Male" o "female"). También se puede observar que la columna *color* es una columna con cadenas de texto que pueden convertirse en valores categóricos y siendo esas categorías "White", "blue", "green". En la Ilustración 16 – Información de df antes de aplicar AutoDataFrame se puede observar que la variable *weight* contiene un valor vacío ("None") lo que provoca que la columna, debido a que pertenece a la categoría de DataFrame tienen que ser todos los valores en esa columna del mismo tipo, tenga categoría de objeto y la Ilustración 15 – Representación del DataFrame (df) antes de la ejecución muestra que su rango de valores es muy amplio. Por último, la columna *created_on* es una columna con fechas.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   id          5 non-null      int64
 1   gender      5 non-null      object
 2   color       5 non-null      object
 3   weight      4 non-null      object
 4   created_on  5 non-null      object
dtypes: int64(1), object(4)
memory usage: 328.0+ bytes
None
```

Ilustración 16 – Información de df antes de aplicar AutoDataFrame

La ejecución del código mostrado en Ilustración 14 – Uso sencillo de AutoDataCleaner ha sido correctamente interpretado y no ha devuelto ningún error. La información sobre los cambios que AutoDataCleaner ha realizado aparecen por consola (Ilustración 17 – Información sobre los cambios realizados en los datos).

```
In [4]: runfile('C:/Users/Usuario/.spyder-py3/TrabajoFinDeGrado/TFG.py', wdir='C:/Users/
Usuario/.spyder-py3/TrabajoFinDeGrado')
 +++++++++++++++ AUTO DATA CLEANING STARTED +++++++++++++++
 =  AutoDataCleaner: Casting datetime columns to datetime dtype...
  + converted column created_on to datetime dtype
 =  AutoDataCleaner: Performing removal of unwanted columns...
  + removed 1 columns successfully.
 =  AutoDataCleaner: Performing One-Hot encoding...
  + detected 1 binary columns [['gender']], cells cleaned: 5 cells
 = AutoDataCleaner: Converting columns to numeric dtypes when possible...
  + 1 minority (minority means < %25 of 'weight' entries) values that cannot be converted to
numeric dtype in column 'weight' have been set to NaN, nan cleaner function will deal with them
  + converted 5 cells to numeric dtypes
 =  AutoDataCleaner: Performing One-Hot encoding...
  + one-hot encoding done, added 2 new columns
 =  AutoDataCleaner: Performing None/NA/Empty values cleaning...
  + cleaned the following NaN values: {'weight NaN Values': 1}
 =  AutoDataCleaner: Performing dataset normalization...
  + normalized 5 cells
 +++++++++++++++ AUTO DATA CLEANING FINISHED +++++++++++++++
```

Ilustración 17 – Información sobre los cambios realizados en los datos

La Ilustración 18 – Representación del DataFrame (df) después de la ejecución muestra los cambios mencionados por la consola en la Ilustración 17 – Información sobre los cambios realizados en los datos. Se puede observar claramente que la columna *id* ha sido eliminada debido a su inutilidad. La columna *gender* ha sido tramsformada a valores binarios siendo la categoria 0 “Female”. La columna *weight* ha sido normalizada y el valor nulo ha sido sustituido por la moda. La columna *created_on* ha sido modificada a al tipo datetime64. Por ultimo, la columna *color*

ha sido transformada a sus valores categoricos obteniendo 3 categorias ("Blue", "Green", "White").

	gender	weight	created_on	color_blue	color_green	color_white
0	1	-0.588348	2018-11-20	0	0	1
1	0	-0.392232	2014-01-12	1	0	0
2	1	1.765045	2020-09-02	0	0	1
3	1	-0.196116	2020-09-02	1	0	0
4	1	-0.588348	2020-12-30	0	1	0

Ilustración 18 – Representación del DataFrame (df) después de la ejecución

Tras la primera implementación de AutoDataCleaner, se ha comprobado su funcionamiento y su utilidad.

Los inconvenientes de AutoDataCleaner son la indicación de que columna es la que se desea eliminar y su imposibilidad de pensar por sí mismo que tipo de valor debería de remplazar un valor nulo. En los siguientes capítulos se intentará implementar estas funciones.

En el Capítulo 8. se implementará AutoDataCleaner, pero para otra base de datos más compleja.

Capítulo 7. TRATAMIENTO DE LAS VARIABLES

Una vez experimentado con el código ejemplo ofrecido por AutoDataCleaner, se ha realizado un análisis de funciones faltantes a la vez que un análisis de funciones necesarias para este proyecto. Estas nuevas funcionalidades, destinadas a la mejora de la calidad de los datos, es esencial para el uso y la explotación de los datos, se agrupan en dos tipos. Lo primero es la limpieza de los datos y se encarga de identificar y sustituir los datos incompletos e inexactos. Por otro lado, tenemos el data *wrangling* que se encarga de transformar los datos brutos en formato adecuado para el análisis. Los modelos realizados con datos erróneos están sesgados u ofrecen bajos rendimientos del modelo.

Gracias al tratamiento de variables, los datos obtenidos serán precisos, completos, coherentes, uniformes, trazables y ordenados.

En los siguientes apartados se explicará cómo se ha procedido a las siguientes funciones para obtener un programa óptimo. Contará con una eliminación de datos duplicados (`drop_columns_duplicated_features()`), una eliminación de valores vacíos (`clean_nan()`) y la eliminación de valores atípicos (`remove_outlier()`). También, contiene el `read_csv()` que detecta desde el inicio los valores del tipo tiempo y los introduce en el data set como datetime64, la eliminación de valores altamente correlacionados con la función `drop_columns_HighCorr()` y la eliminación de los valores con poca varianza (`drop_columns_constant_or_cuasi_constant()`). La normalización también se realiza, pero se ha automatizado la elección de variables que deben normalizarse (`deciding_normalization_variables()`), la detección de variables únicas y así eliminar parámetros como, por ejemplo, el nombre o las calles (`variables_uniques()`), también se ha creado la función para eliminar las filas duplicadas que se ejecuta con la siguiente línea de código `duplicated_rows()` y por último, la codificación de variables categóricas en numéricas (`decision_tree_encoder ()`) a través de árboles de probabilidad con la automatización de las variables a transformar (`deciding_tree_variables(()`).

Cabe destacar que el orden de implementación de cada función dentro de la librería del proyecto es esencial y forma parte del valor añadido de este proyecto. Primero se realiza la eliminación de variables dichas por el usuario. La eliminación de estas variables, que no le interesan al usuario, usará de forma innecesaria memoria y tiempo de máquina en su procesado, se eliminaran las filas duplicadas para reducir el tamaño de la muestra y poder seguir reduciendo el tiempo de máquina y memoria y, una vez eliminadas, se descartarán las variables completamente únicas ya que por lo dicho anteriormente, las variables con cada valor único, no aportarán nada al modelo y tendrán un uso de máquina y tiempo de ejecución mayor. Una vez eliminadas las variables y reducido el conjunto de datos, se procede a limpiar los valores vacíos (eliminando o rellenándolos) como se explicará en los siguientes apartados. Este paso es necesario ya que los modelos no pueden tratar los valores vacíos. Una vez limpios, se procede a detectar las columnas binarias y a convertirlas en variables dicotómicas y haciendo que el modelo pueda interpretar estas variables. Una vez convertidas las variables binarias en dicotómicas, se procede a la codificación por arboles de decisión, seguido de la conversión de caracteres a *string* a través de la función `to_numeric()`, este paso es muy importante ya que, sin él, no se pueden interpretar esas variables. Debido a esta última conversión, si las variables no se pueden cambiar a numéricas y son menos del 25% de las entradas, se rellenarán con NaN y por tanto habrá que volver a limpiarlas. Una vez limpias, se procede a encontrar los límites de los valores atípicos y eliminar valores que sobrepasen esos límites ya que pueden llevar al modelo a dar errores de predicción o clasificación. Una vez reducido el set de datos, se encarga de convertir las variables, dichas por el usuario, de tipo fecha y hora escritas como *object* en *datetime64* para su correcta interpretación. Una vez realizada la conversión, se procede al uso del *One_Hot_Encoding* para convertir variables no numéricas en variables numéricas, esto se realiza por seguridad hacia las demás funciones. Se verá que la aparición del mensaje por pantalla muestra que no se realiza ningún cambio, lo cual es correcto ya que, en ese punto, todas las variables han sido transformadas. Se procede a la eliminación de variables con correlaciones altas, variables duplicadas y variables constantes o cuasi constantes , donde cada función se explicará más detalladamente en este capítulo, seguido de la normalización como último paso, ya que la mayoría de los modelos asumen distribuciones gaussianas de los datos proporcionados y la normalización centra las valores en torno al 0.

7.1 READ_CSV

En este apartado se explicará la idea de cómo cambiar los datos de tipo fecha y hora en formato datetime64. Esto es necesario ya que debido a que los datos son leídos y provienen de un archivo .csv, por norma general, las variables de tipo fecha u hora, al contener puntos, doble puntos, guiones, barras etc. , convierten esas variables en objetos y no en datetime64 y el programa malinterpretará esa variable, lo que supondría una pérdida de información valiosa. Este cambio es necesario para cualquier modelo.

Para resolver el problema presentado anteriormente, se le ofrece al analista dos opciones a través de un comentario que se imprimirá por pantalla una vez ejecutado la función `adc.read_csv_transformado()` de AutoDataCleaner. La aparición del comentario por pantalla esta descrita en las siguientes líneas de código.

```
def read_csv_transformado():
    text = """
    ++++++++++++++++++    AUTO    DATA    CLEANER    READ    CSV    TRANSFORMADO
++++++++++++++++++++++
    OPTION 1:
        IMPORT LIBRARY:
            import pandas as pd
        FUNCTION CALL:
            df=pd.read_csv('mydata.csv')

        DO NOT FORGET TO INCLUDE IN datetime_columns=[] , the variables date/time
that
        need to be changed.
    OPTION 2:
    IMPORT LIBRARY:
        from dt_auto import read_csv
    FUNCTION CALL:
        df=read_csv('mydata.csv')

    FUNCTION
    import pandas as pd
```

```
    def dt_inplace(df):
        #Automatically detect and convert (in place!) each
        #dataframe column of datatype 'object' to a datetime just
        #when ALL of its non-NaN values can be successfully parsed
        #by pd.to_datetime().  Also returns a ref. to df for
        #convenient use in an expression.

        from pandas.errors import ParserError
        for c in df.columns[df.dtypes=='object']: #don't cnvt num
            try:
                df[c]=pd.to_datetime(df[c])
            except (ParserError,ValueError): #Can't cnvrt some
                pass # ...so leave whole column as-is unconverted
        return df
    def read_csv(*args, **kwargs):
        #Drop-in replacement for Pandas pd.read_csv. It invokes
        #pd.read_csv() (passing its arguments) and then auto-
        #matically detects and converts each column whose datatype
        #is 'object' to a datetime just when ALL of the column's
        #non-NaN values can be successfully parsed by
        #pd.to_datetime(), and returns the resulting dataframe.

        return dt_inplace(pd.read_csv(*args, **kwargs))
    DO NOT FORGET TO REMOVE THE # FROM EVERY LINE
    ++++++++++++++++++ AUTO DATA CLEANER HELP ++++++++++++++++++++++
    """
    print(text)
```

Podrá elegir si leer los datos directamente del archivo csv y de ahí, el analista decidir cuáles son de tipo fecha u hora e indicarlas en el parámetro `datetime_columns=[]` o directamente leer el archivo .csv con una función `read_csv()`. Hay que destacar que, al inicio del script, habrá que añadir la siguiente línea de código para poder acceder a esta última funcionalidad `from AutoDataCleaner.AutoDataCleaner import read_csv`.

Hay que remarcar que la función `dt_auto()` contiene dos funciones, `dt_inplace()` y `read_csv()` y, debido a que no estaba incluida en AutoDataCleaner, han sido creadas cada función de forma individual en vez de forma conjunta con `dt_auto()`. Si se desea crearla en

un nuevo archivo, copiando las líneas de código anteriores se llegaría a crear esta función separada de AutoDataCleaner.

Las líneas de código han sido creadas por David B Rosen el 16 de agosto 2021. [24].

7.2 VALORES ATÍPICOS

La identificación de los valores atípicos[13] y su cambio por valores dentro del rango de no atípicos es uno de los procesos más importantes para el preprocesado de los datos. Sin esta implementación, como los datos atípicos pesan más que los datos relativamente cercanos a la media y si no son representativos de la población, pueden distorsionar los resultados [25]. Algunos de los modelos, incluyen la eliminación de variables atípicas pero

El código que ha sido usado en esta parte proviene de Feature-Engine. Feature_engine, en su apartado de outlier, ofrece la posibilidad de encontrar y cambiar los outliers de tres formas distintas (Winsorizer, ArbitraryOulierCapper y OutlierTrimmer). Para escoger uno de los métodos, se ha utilizado la función `time.time()` especificada en el apartado de Outlier Handling y usando, debido a su menor tiempo de ejecución, OutlierTrimmer.

El código siguiente muestra su llamada a la función `remove_outlier()` y la función.

```
# Eliminate outliers (Teresa) ------------------------------------------------
-----------------------------------------
    if outlier_removal:
        if verbose:
            print("=AutoDataCleaner: Performing Outliers removal... ")
    df=remove_outlier(df,verbose)
```

```
# Function remove_outlier()
def remove_outlier(df,verbose):
    capper = outliers.OutlierTrimmer(capping_method = "gaussian" ,
                                     tail='both', fold=3 ,missing_values="ignore")
    df=capper.fit_transform(df)
```

[13] observaciones con características diferentes de las demás

```
    maximum=capper.right_tail_caps_
    minimum=capper.left_tail_caps_
    if verbose:
        print("+ The following [{}] shows dictionary with the maximum values above
which values will be removed ".format(maximum))
        print("+ The following [{}] shows dictionary with the minimum values below
which values will be removed ".format(minimum))
    return df
```

Las características mencionadas en la Tabla 2 – Parámetros y funcionalidades de la función Winsorizer se han rellenado de la siguiente forma (Tabla 8 – Construcción de la función OutlierTrimmer).

Parámetro	**Valor**	**Razón**
capping_method	`capping_method = "gaussian"`	Ventaja:Mejor aproximación a la vida real Teorema central del Limite Simplicidad Desventaja: Sensible a los outliers [26][14]
tail	`tail='both'`	Eliminar todo tipo de outliers
fold	`Fold=3`	Valor por defecto
variables		Se aplica a todas las variables
missing_values	`missing_variables= "ignore"`	Los errores son ignorados ya que ya se han realizado la eliminación de valores faltantes. Por comodidad.

Tabla 8 – Construcción de la función OutlierTrimmer

[14] Leyenda: verde (ventajas) y rojo (desventajas)

Una vez ejecutado el código completo, se puede observar por consola los valores considerados outlliers tanto de la cola derecha como cola izquierda. Esto se ha conseguido a través del parámetro `verbose = True`. Este parámetro pertenece a la función clean_me (Ilustración 14 – Uso sencillo de AutoDataCleaner) y las siguientes líneas de código muestran los comandos para su impresión por consola.

```
    # Clean None (na) values (Teresa) ----------------------------------------
    if verbose:
        print(" = AutoDataCleaner: Performing None/NA/Empty values cleaning... ")
        print("No cleaner mode for treating the nan")
        df=clean_nan(df,verbose)
```

La impresión se muestra a continuación.

Comando	Consola
`Outlier_removal=True`	`AutoDataCleaner: Performing Outliers removal...` `+ The following [{'PassengerId': 1218.0615260456902, 'Survived': 1.8436157466329595, 'Pclass': 4.816855698239796, 'Sex': 2.0815571936086097, 'Age': 68.42067214450208, 'SibSp': 3.8312381532214728, 'Parch': 2.799765378316916, 'Ticket': 1.6761273636457816, 'Fare': 181.2844937601173}] shows dictionary with the maximum values above which values will be removed` `+ The following [{'PassengerId': -326.06152604569024, 'Survived': -1.0759389789561917, 'Pclass': -0.19957174762251206, 'Sex': -0.7863832317679811, 'Age': -9.697507161337093, 'SibSp': -2.7852224405390933, 'Parch': -2.0365779484628193, 'Ticket': -0.9084505959690259, 'Fare': -116.87607782296811}] shows dictionary with the minimum values below which values will be removed`

Tabla 9 – Outlier Removal y respuesta por consola

La Tabla 9 – Outlier Removal y respuesta por consola muestra el rango de valores por variable que no son considerados outlier. Los valores por debajo o por encima de ese límite, son considerados atípicos y por lo tanto eliminados.

7.3 VALORES FALTANTES

Todos los archivos de datos contienen datos faltantes. Esto puede deberse a la transcripción de los datos o la falta de respuesta para una variable. El problema de los datos faltantes es que no se pueden tratar. La forma de tratar estos valores puede ser eliminándolos o remplazándolos. La eliminación de los datos reduciría el tamaño muestral dando lugar a perdida de información valiosa, dando lugar a que los resultados sean más inciertos con mayor probabilidad de realizar un modelo erróneo en cambio remplazando las variables vacías, no perdemos la información.

Todo ello se ejecutará con las siguientes líneas de código que están incluidas en la función `clean_me()` de AutoDataCleaner.

```
# Clean None (na) values (Teresa) ------------------------------------------------
-------------------------------------------
    if clean:
        if verbose:
            print(" =  AutoDataCleaner: Performing None/NA/Empty values cleaning...
")
        df=clean_nan(df,verbose)
```

En este proyecto se ha asumido que los datos faltantes son todos aleatorios (MCAR) es decir, que las variables faltantes no dependen de otras variables, sino que su falta es debida a la aleatoriedad de la muestra. Esto se explicará más adelante en el Capítulo 10. como apartado de trabajos futuros.

ELIMINANDO VARIABLES VACÍAS

Las variables vacías aparecen como NaN y su eliminación se puede realizar de varias maneras. Se puede realizar por eliminación de columna o de fila.

7.3.1.1 Columnas

La automatización de la eliminación de una columna entera lleva a una reducción significativa de la muestra perdiendo seguridad en los parámetros obtenidos y puede llevar a la eliminación de una columna significativa para el modelo que se desea implementar.
Se eliminarán las columnas siguiendo varios parámetros. En la función `clean_nan()` se analizará solo las columnas que contienen demasiados valores faltantes.

En los siguientes apartados se analizará las columnas primeramente los valores altamente correlacionados, seguido de los valores con poca varianza entre ellos y por ultimo los valores duplicados.

7.3.1.1.1 Gran número de datos faltantes

Una variable que no aporta nada de información es aquella que tiene más de un 50% de sus valores faltantes.

La eliminación de una columna entera ha sido programada para que elimine la columna si el porcentaje de valores faltantes en esa columna es igual o mayor de 60% de los datos totales de la columna.

El código siguiente muestra la parte del código de `clean_nan()` que se encarga de eliminar las columnas con su condición del porcentaje de valores faltantes.

```
    #1. Drop the col if there are 60% of values missing
    if verbose:
        print('Inside the funtion for cleaning variable, we proceed to eliminate
the data with more than 60% of values beeing missing values')
    perc = 60.0
    min_count =  int(((100-perc)/100)*df.shape[0] + 1)
    null_percentage = df.isnull().sum()/df.shape[0]*100
    col_to_drop=null_percentage[null_percentage>perc].keys()
    df = df.dropna( axis=1,thresh=min_count)
    if verbose:
        print(" + the following columns '{}' had {}% or more of the values being
missing ".format(col_to_drop,perc))
```

Cabe remarcar que, si se desea bajar o subir el umbral para la eliminación de la columna, se puede hacer directamente cambiando el parámetro `perc =60.0`

Una vez ejecutado el código completo, se puede observar por consola, las columnas que contenían más de un 60% de valores faltantes. Esto se ha conseguido a través del parámetro `verbose = True` . Este parámetro pertenece a la función clean_me (Ilustración 14 – Uso sencillo de AutoDataCleaner) y las siguientes líneas de código muestran los comandos para su impresión por consola.

```
[…]
print('Inside the funtion for cleaning variable, we proceed to eliminate the data
with more than 60% of values beeing missing values')
[…]
if verbose:
        print(" + the following columns '{}' had {}% or more of the values being
missing ".format(col_to_drop,perc))
```

La impresión se muestra a continuación.

Comando	Consola
Automático	= AutoDataCleaner: Performing None/NA/Empty values cleaning... Inside the funtion for cleaning variable, we proceed to eliminate the data with more than 60% of values beeing missing values + the following columns 'Index(['Cabin'], dtype='object')' had 60.0% or more of the values being missing

Tabla 10 – Eliminación de las variables para tratamiento de variables vacías (columnas)

En la Tabla 10 – Eliminación de las variables para tratamiento de variables se muestra que variables con más de un 60% de valores faltantes, son eliminadas. Esta variable es *cabin*. Esta variable está demasiado vacía para poder hacer un relleno de los datos, y por esa razón, se ha decidido eliminar y poner el límite al 60%.

7.3.1.2 Filas

Una fila que contiene muchos valores faltantes supone un gran problema a la hora de manejar esas variables. Los inconvenientes se han detallado el apartado 7.3.1.1.

7.3.1.2.1 Filas con valores faltantes

En este proyecto se ha optado por la eliminación de las filas cuyos valores faltantes sean iguales o superiores al 5% de los valores totales de esa fila.

El código siguiente muestra la parte del código de `clean_nan()` que se encarga de eliminar las filas con su condición del porcentaje de valores faltantes.

```
#2. Drop the line if there is more than 5% of values missing (95% is preserved)
    if verbose:
        print('Inside the funtion for cleaning variable, we proceed to eliminate
the row with more than 5% of values beeing missing values')
    perc=5.0
    df =df[df.isnull().sum(axis=1) < perc/100*len(df)]
    null_percentage = df.isnull().sum(axis=1)/df.shape[1]*100
    line_to_drop= null_percentage[null_percentage>perc].keys()
    if verbose:
        print(" + the following rows '{}' had 5% or more of the values being missing
".format(line_to_drop))
```

Cabe remarcar que, si se desea bajar o subir el umbral para la eliminación de la fila, se puede hacer directamente cambiando el parámetro `perc =5.0.`

Una vez ejecutado el código completo, se puede observar por consola, las filas que contenían más de un 5% de valores faltantes. Esto se ha conseguido a través del parámetro `verbose = True` . Este parámetro pertenece a la función clean_me (Ilustración 14 – Uso sencillo de AutoDataCleaner) y las siguientes líneas de código muestran los comandos para su impresión por consola.

```
[…]
if verbose:
```

```
        print('Inside the funtion for cleaning variable, we proceed to eliminate
the row with more than 5% of values beeing missing values')
[…]
if verbose:
        print(" + the following rows '{}' had {}% or more of the values being
missing ".format(line_to_drop,perc))
```

La impresión se muestra a continuación.

Comando	Consola

Automático	`Inside the funtion for cleaning variable, we proceed to eliminate the row with more than 5% of values beeing missing values`
	` + the following rows 'Int64Index([ 5, 17, 19, 26, 28, 29, 31, 32, 36, 42,`
	`...`
	`832, 837, 839, 846, 849, 859, 863, 868, 878, 888],`
	`dtype='int64', length=179)' had 5% or more of the values being missing`

Tabla 11 – Eliminación de las variables para tratamiento de variables vacías (filas)

En la Tabla 11 – Eliminación de las variables para tratamiento de variables vacías (filas) se puede observar también que, si una línea le faltan más del 5% de variables, se eliminará la fila. Se puede observar el número de las filas que se han eliminado. Una fila donde le falten más del 5% de los datos, no nos puede aportar suficiente valor al conjunto de datos y por ello, se ha decidido eliminarla.

IMPUTACIÓN VARIABLES VACÍAS

Otro método posible para tratar las variables vacías es la imputación. Este método trata de hallar las variables vacías y rellenarlas. El relleno puede ser por mediana o por la moda. La media también se puede usar, aunque es menos precisa y por esa razón se ha descartado en este análisis.

La imputación está incluida en el código `clean_me()` pero pertenece a Randal S. Olson con su programa *DataCleaner* disponible en GitHub. En el apartado Licencia para DataCleaner de Randal S. Olson , puede encontrarse una copia del mensaje copyright. [13].

El código se encarga de valorar si es mejor rellenar los valores faltantes por la mediana o por la moda dependiendo del tipo de variable. Si la variable es numérica, se remplazará por la mediana. En cambio, si la variable es de tipo string, el resultado se rellenará por la moda de esa variable.

El código siguiente muestra la decisión y el relleno de las variables faltantes por la mediana o la moda.

```
#3. Replace NaNs with the median or mode of the column depending on the column type
    if verbose:
        print('Inside the funtion for cleaning variable, we proceed to replace the
missing values for the median/mode depending on type of varibles')
    for column in df.columns.values:
    # Replace NaNs with the median or mode of the column depending on the column
type
        try:
            df[column].fillna(df[column].median(), inplace=True)
            median_col=df[column].name
            if verbose:
                print(" + The following variables '{}' have been fill by the
median".format(median_col))
        except TypeError:
            most_frequent = df[column].mode()
        # If the mode can't be computed, use the nearest valid value
        # See https://github.com/rhiever/datacleaner/issues/8
            if len(most_frequent) > 0:
                df[column].fillna(df[column].mode()[0], inplace=True)
                mode_col=df[column].name
                if verbose:
                    print(" + The following variables '{}' have been fill by the
mode".format(mode_col))

            else:
```

```
                df[column].fillna(method='bfill', inplace=True)
                df[column].fillna(method='ffill', inplace=True)
    return df
```

7.3.1.3 Median

Rellenar las variables faltantes por la mediana se muestra a continuación.

```
#3. Replace NaNs with the median or mode of the column depending on the column type
    if verbose:
        print('Inside the funtion for cleaning variable, we proceed to replace the
missing values for the median/mode depending on type of varibles')
    for column in df.columns.values:
    # Replace NaNs with the median or mode of the column depending on the column
type
        try:
            df[column].fillna(df[column].median(), inplace=True)
            median_col=df[column].name
            if verbose:
                print(" + The following variables '{}' have been fill by the
median".format(median_col))
```

Como se puede observar, las únicas variables que se rellenaran por la mediana son de tipo numérico son o un int o un float.

Una vez ejecutado el código completo, las variables que serán transformadas y rellenadas por la mediana y aparecerán por consola. Esto se ha conseguido a través del parámetro `verbose = True` . Este parámetro pertenece a la función clean_me (Ilustración 14 – Uso sencillo de AutoDataCleaner) y las siguientes líneas de código muestran los comandos para su impresión por consola.

```
[…]
if verbose:
                print(" + The following variables '{}' have been fill by the
median".format(median_col))
```

La impresión se muestra a continuación.

Comando	Consola

Automático	`Inside the funtion for cleaning variable, we proceed to replace the missing values for the median/mode depending on type of varibles`
	` + The following variables 'PassengerId' have been fill by the median`
	` + The following variables 'Survived' have been fill by the median`
	` + The following variables 'Pclass' have been fill by the median`
	` + The following variables 'Sex' have been fill by the mode`
	` + The following variables 'Age' have been fill by the median`
	` + The following variables 'SibSp' have been fill by the median`
	` + The following variables 'Parch' have been fill by the median`
	` + The following variables 'Ticket' have been fill by the mode`
	` + The following variables 'Fare' have been fill by the median`
	` + The following variables 'Embarked' have been fill by the mode`

Tabla 12 – Rellenar las variables para tratamiento de variables vacías

En la Tabla 12 – Rellenar las variables para tratamiento de variables vacías se puede ver las variables que quedan con valores nulos y como se han rellenado según su categoría. La información de cada variable, como el tipo de dato que son, con el comando `df.info()`, se muestra en el ANEXO III. Conjuntos de datos en el apartado Titanic Dataset.

7.3.1.4 Moda

Rellenar las variables faltantes por la moda se muestra a continuación.

```
except TypeError:
            most_frequent = df[column].mode()
        # If the mode can't be computed, use the nearest valid value
        # See https://github.com/rhiever/datacleaner/issues/8
            if len(most_frequent) > 0:
```

```
                df[column].fillna(df[column].mode()[0], inplace=True)
                mode_col=df[column].name
                if verbose:
                    print(" + The following variables '{}' have been fill by the
mode".format(mode_col))
            else:
                df[column].fillna(method='bfill', inplace=True)
                df[column].fillna(method='ffill', inplace=True)
    return df
```

Una vez ejecutado el código completo, las variables que serán transformadas y rellenadas por la moda y aparecerán por consola. Esto se ha conseguido a través del parámetro `verbose = True` . Este parámetro pertenece a la función clean_me (Ilustración 14 – Uso sencillo de AutoDataCleaner) y las siguientes líneas de código muestran los comandos para su impresión por consola.

```
if verbose:
                    print(" + The following variables '{}' have been fill by the
mode".format(mode_col))
```

En la Tabla 12 – Rellenar las variables para tratamiento de variables vacías se puede observar la impresión por pantalla. Las variables rellenas por la moda son las variables que no se han podido rellenar por mediana, es decir, las variables que contienen caracteres.

7.4 CODIFICACIÓN

El proceso de codificación es transformar las variables categóricas[15] en variable binarias o numéricas enteras para que estos datos, una vez convertidos, se proporcionen a los modelos. Es un proceso esencial debido a que la mayoría de los modelos trabajan con valores enteros solamente y a veces con valores diferentes entendibles por algunos modelos solamente. Hay dos tipos de variables categóricas, las variables nominales[16] y las variables ordinales[17] . Con

[15] Las variables categóricas son datos que constan de valores posibles finitos. [62].
[16] Las variables nominales son el nombre de una variable sin ningún valor numérico.
[17] Las variables ordinales son variables que contienen un conjunto de ordenes o escalas.

este proceso se quiere obtener una estructura más simplificada de los datos y conlleva a una mejor interpretación.

En este proyecto, solo se ha creado nuevo la codificación por árboles. Cabe destacar que la función AutoDataCleaner ya contaba con la codificación de variables binarias y su automática detección y la codificación de variables de tipo categóricas en variables numéricas.

Aunque no se hayan cubierto todas las técnicas de codificación existentes en el mundo de machine learning, se ha cubierto la mayoría de ellas. La implementación de otros codificadores se explicará en el apartado Capítulo 10. en la sección de trabajos futuros.

DECISIONTREEDISCRETIZATION

La codificación por árboles, como indica, utiliza árboles para representar las consecuencias de cada decisión de forma gráfica. Los árboles están compuestos de tres partes. La primera parte, el nodo raíz seguido del nodo hoja y por último las ramas. La representación gráfica del árbol de decisiones está en la Ilustración 19 – Arboles de decisiones

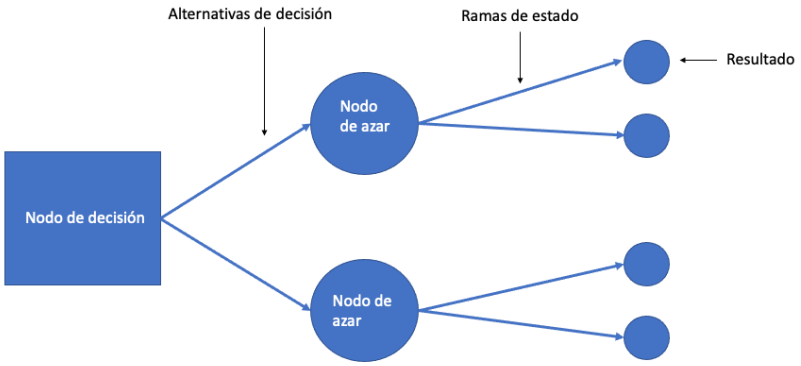


Ilustración 19 – Arboles de decisiones

Gracias a este proceso, los valores continuos serán remplazados por variables discretas predichas por el árbol de decisión. Este método está justificado cuando los datos no están

normalizados o la existencia de valores extremos. Este método de codificación ayuda a las variables relevantes a tener una mayor interpretación en el modelo.

En la llamada a la función `clean_me()` hay que poner el parámetro `decisión_tree = True` y si se quieren dar las variables que hay que codificar, se pueden añadir como `variable_to_encode=['…', '…'].`

A continuación, se muestra el código de llamada a la función `decision_tree_encoder ().`

```
# Decision tree encoder + Automatization (Teresa (Delete the parameter dtype))---
-----------------------------------------------------
    if decision_tree:
        if len(variable_to_encode)>0:
            if verbose:
                print(" =   AutoDataCleaner: Performing dataset decision tree
encoder to the variable [{}] ".format(variable_to_encode))

df=decision_tree_encoder(df,variable_to_encode,measuring_variable,verbose,
datetime_columns)
        else:
            if verbose:
                print(" =   AutoDataCleaner: Performing dataset decision tree
encoder to the variables  ...")

df=decision_tree_encoder(df,variable_to_encode,measuring_variable,verbose,datetim
e_columns)
```

El código siguiente muestra la creación de la función `decision_tree_encoder ().`

```
from feature_engine.encoding import DecisionTreeEncoder
[…]
# Desiton tree encoder


def decision_tree_encoder(df,var,y,verbose):
    # set up the encoder
    if verbose:
        if len(var)>0:
```

```
            print("  + The following variables are beeing transforme by decision
tree encoder [{}]".format(var))
        else:
            print("     +   All   the   variables   are   beeing   transformed[{}]
".format(df.columns.values))
    encoder = DecisionTreeEncoder(regression=False, random_state=0)
    # fit the encoder
    df=encoder.fit_transform(df,df[y])
    return df
```

Hace falta remarcar que en el script AutoDataCleaner, hay que cargar la librería `from feature_engine.selection import DecisionTreeEncoder.`

Se puede observar que se ha usado la función `DecisionTreeEncoder()` de feature_engine como se mencionó en el apartado 4.4.1.12.5.

Se puede observar en el código anterior un comentario donde indica que se debería ejecutar primero el script donde está la función DropCorrelatedFeatures que está situada en el siguiente directorio.

```
C:\Users\Usuario\anaconda3\envs\OptimizacionVarEntrada\Lib\site-
packages\feature_engine\encoding\decision_tree
```

Una vez ejecutado el código completo, se puede observar por consola, las variables que han sido transformadas. Esto se ha conseguido a través del parámetro `verbose = True` . Este parámetro pertenece a la función clean_me (Ilustración 14 – Uso sencillo de AutoDataCleaner) y las siguientes líneas de código muestran los comandos para su impresión por consola.

```
[…]
print("  + The following variables are beeing transforme by decision tree encoder
[{}]".format(var))
[…]
print("        +    All    the    variables    are    beeing    transformed[{}]
".format(df.columns.values))
```

La impresión se muestra a continuación. La tabla mostrada a continuación muestra los comandos usados para activar la función y su respuesta por consola.

Comando	Consola

`decision_tree=True measuring_variable='Survived'`	`= AutoDataCleaner: Performing dataset decision tree encoder to the variables ...`
	`+ converted 891 cells to numeric dtypes`
	`The column [Ticket] is transformed by decision tree encoder encoder which produces an accuracy in the model of 100.0 which is over 80%`
	`The variable [Embarked] will not be converted to decision tree encoder because its accuracy is too low`
	`+ the variables that are beeing transformed are [['Ticket']]`

Tabla 13 – Decisión tree y respuesta por consola

En la Tabla 13 – Decisión tree y respuesta por consola se puede observar que se ha automatizado las variables a las que se le desea que se codifiquen con un árbol de decisión a través del parámetro de exactitud.

AUTOMATIZACIÓN DE LA ELIMINACIÓN DE VARIABLES PARA HACER DECISIÓN TREE ENCODER

La automatización de las variables a las que se desea implementar la codificación según decisión tree encoder es una función vital para este proyecto ya que según el modelo de árboles de decisión y su nivel de exactitud calculado para cada variable independientemente cambiada, si la exactitud de ese modelo con esa variable codificada por decisión tree encoder es mayor al 80%, se realizara dicho cambio en el conjunto de variables.

El analista si lo desea, puede indicar las variables donde se desea aplicar una decisión tree encoder, pero gracias a esta función, el analista no va a tener que valorar que variables desea cambiar. El propio programa entrega al analista las variables cambiadas.

A continuación, se muestra el código de llamada a la función `deciding_tree_variables()`.

```
def decision_tree_encoder(data,var,y,verbose,datetime_columns):
    # set up the encoder
    name=[]
    if len(var)>0:
        print("  + The following variables are beeing transforme by decision tree encoder [{}]".format(var))
        encoder       =       DecisionTreeEncoder(variables=var,regression=False, random_state=0)
        data=encoder.fit_transform(data,data[y])
    else:
        var=[]
        for col in data.columns:
            if (data[col].dtype==np.object):
                name.append(col)

    for col in name:
        columna=deciding_tree_variables(data,col,datetime_columns,y,verbose)
        if columna:
            if col in columna:
                var.append(col)
        else:
            print('The variable [{}] will not be converted to decision tree encoder because its accuracy is too low'.format(col))
    if len(var)==len(name):
        encoder       =       DecisionTreeEncoder(variables=var,regression=False, random_state=0)
        data=encoder.fit_transform(data,data[y])
        print("   + the variables that are beeing transformed by the automatic decision tree encoder are [{}] ".format(var))
    else:
        data=convert_numeric_df(data,var, verbose=False)
        data= clean_nan(data,verbose=False)
        encoder       =       DecisionTreeEncoder(variables=var,regression=False, random_state=0)
        data=encoder.fit_transform(data,data[y])
        print("  + the variables that are beeing transformed are [{}] ".format(var))
    return data
```

El código siguiente muestra la creación de la función `deciding_tree_variables()`.

```
def deciding_tree_variables(X,cl,datetime_columns,y,verbose):
    encoder = DecisionTreeEncoder(variables=cl,regression=False, random_state=42)
    encoder.fit(X,X[y])
    X=encoder.transform(X)
    X=convert_numeric_df(X, datetime_columns,verbose)
    X = clean_nan(X,verbose=False)
    X_train, X_test, y_train, y_test = train_test_split(X,X[y], random_state=0,
train_size = .8)

    clf = LogisticRegression()
    clf.fit(X_train, y_train)
    preds=clf.predict(X_test)
    score_preds=(metrics.accuracy_score(y_test,preds))*100

    if score_preds>=80:
        colum=cl
        print('The column [{}] is transformed by decision tree encoder which
produces an accuracy in the model of {} which is over
80%'.format(colum,score_preds))
        return colum
```

Se puede observar por consola las variables que han sido normalizadas por esta función. Esto se ha conseguido a través del parámetro `verbose = True` . Este parámetro pertenece a la función clean_me (Ilustración 14 – Uso sencillo de AutoDataCleaner) y las siguientes líneas de código muestran los comandos para su impresión por consola.

Las variables halladas para realizar una decisión tree encoder se muestran en la Tabla 17 – Decisión tree y respuesta por consola. En este caso, la variable “Ticket”. Esta variable, será eliminada por alta correlación con la variable a medir *y* más adelante.

7.5 VALORES ALTAMENTE CORRELACIONADOS

El coeficiente de correlación de Pearson cuantifica la relación lineal entre dos variables en un análisis de correlación. Se considera un coeficiente de Pearson alto si es mayor a 0,7. En

este caso, se ha utilizado un umbral de 0,8 para realizar el descarte de las variables altamente correlacionadas.

Las altas relaciones entre dos variables explicativas en el modelo pueden llevar a problemas de multicolinealidad. Hay dos tipos de multicolinealidad, la perfecta, es cuando la relación lineal es perfecta y este problema no es grave debido a que es fácil de identificar y la multicolinealidad imperfecta grave, es cuando hay una relación muy alta con una o varias variables explicativas, pero no perfecta y este problema es grave y difícil de detectar.

La multicolinealidad imperfecta grave produce los siguientes problemas: imprecisión, inferencia, inestabilidad y problemas de interpretación.

La matriz de correlaciones es una buena forma de detección de este problema y la corrección he dicho problema se puede hacer de varias maneras. En este proyecto se ha optado por la eliminación de la variable causante del problema.

El inconveniente de eliminar la variable que causa el problema es que podemos caer en problemas de endogeneidad.

En el Capítulo 10. , se estudiará la posibilidad de implementación en el futuro de la técnica de análisis de componentes principales para resolver el problema de multicolinealidad.

A continuación, se muestra el código de llamada a la función `drop_columns_HighCorr()`.

```
# Eliminating columns with high corr (Teresa) ----------------------------------
------------------------
    if high_corr_elimination:
        if verbose:
            print(" =    AutoDataCleaner:  Performing  dataset  high  correlation
variables elimination...")
        df = drop_columns_HighCorr(df,verbose)
```

El código siguiente muestra la creación de la función `drop_columns_HighCorr()`.

```
from feature_engine.selection import DropCorrelatedFeatures
[…]

# Elimination of columns with high correlation over 0.8

def drop_columns_HighCorr(df,verbose):
    tr = DropCorrelatedFeatures(variables=None, method='pearson', threshold=0.8)
    df=tr.fit_transform(df)
    #Before running this code, you should run the code for DropCorrelatedFeatures
for it to understand self.correlated_features_sets
    correlete=tr.correlated_feature_sets_
    if verbose:
        print("  + droping [{}] cells".format(correlete))
    return df
```

Hace falta remarcar que en el script AutoDataCleaner, hay que cargar la librería `from feature_engine.selection import DropCorrelatedFeatures`.

Se puede observar que se ha usado la función `drop_columns_HighCorr()` de feature_engine como se mencionó en el apartado DropedCorrelatedFeature.

Se puede observar en el código anterior un comentario donde indica que se debería ejecutar primero el script donde está la función DropCorrelatedFeatures que está situada en el siguiente directorio.

```
C:\Users\Usuario\anaconda3\envs\OptimizacionVarEntrada\Lib\site-
packages\feature_engine\selection\drop_correlated_features
```

Cabe remarcar que, si se desea bajar o subir el umbral para la eliminación de la fila, se puede hacer directamente cambiando el parámetro `threshold = 0.8.`

Una vez ejecutado el código completo, se puede observar por consola, las variables que contenían valores altamente correlacionados. Esto se ha conseguido a través del parámetro `verbose = True` . Este parámetro pertenece a la función clean_me (Ilustración 14 – Uso sencillo

de AutoDataCleaner) y las siguientes líneas de código muestran los comandos para su impresión por consola.

```
if verbose:
        print("  + dropping [{}] cells".format(correlete))
```

La impresión se muestra a continuación.

Comando	Consola
`high_corr_elimination=True`	`=  AutoDataCleaner: Performing dataset high correlation variables elimination...` `  + dropan [[{'Ticket', 'Survived'}]] cells`

Tabla 14 – High correlación variables y respuesta por consola

La Tabla 14 – High correlación variables y respuesta por consola muestra que las variables “ticket” y “survived" están altamente correlacionadas. Esto quiere decir que tienen un coeficiente de Pearson mayor a 0,8 y, por tanto, la variable ticket ha sido eliminada.

7.6 VALORES CON POCA VARIANZA

Si una variable es constante a lo largo de todas las observaciones, se eliminará dicha variable ya que no aporta ninguna información útil. El umbral se ha establecido a 1 de semejanza. La eliminación de variables con poca varianza recae en que las filas son iguales o casi no varían entre ellas para una variable determinada y, por lo tanto, no aportan información relevante al conjunto de datos.

A continuación, se muestra el código de llamada a la función `drop_columns_constant_or_cuasi-constant()`.

```
    # Eliminating constant or nearly constant values (Teresa)--------------------
    if low_var_elimination:
        if verbose:
            print(" =  AutoDataCleaner: Performing dataset dropping constant or
nearly constant  features...")
        df=drop_columns_constant_or_cuasi_constant(df,verbose)
```

El código siguiente muestra la creación de la función `drop_columns_constant_or_cuasi_constant()`.

```
from feature_engine.selection import DropConstantFeatures
[…]

# Elimination of columns with little variance (constant or quasi constant 90%)


def drop_columns_constant_or_cuasi_constant(df,verbose):
    trans= DropConstantFeatures(tol=0.9)
    df=trans.fit_transform(df)
    const=trans.features_to_drop_
    if verbose:
        print("  +  droping [{}] cells".format(const))
    return df
```

Hace falta remarcar que en el script AutoDataCleaner, hay que cargar la librería `from feature_engine.selection import DropConstantFeatures.`

Se puede observar que se ha usado la función `drop_columns_constant_or_cuasi_constant()` de feature_engine como se mencionó en el apartado 4.4.1.12.3.

Se puede observar en el código anterior un comentario donde indica que se debería ejecutar primero el script donde está la función DropConstantFeatures que está situada en el siguiente directorio.

```
C:\Users\Usuario\anaconda3\envs\OptimizacionVarEntrada\Lib\site-
packages\feature_engine\selection\drop_constant_features
```

Cabe remarcar que, si se desea bajar o subir el umbral para la eliminación de una variable con poca o nula varianza entre sus observaciones, se puede hacer directamente cambiando el parámetro `threshold = 1.`

Una vez ejecutado el código completo, se puede observar por consola, las variables cuyas observaciones contenían valores con poca o nula varianza han sido eliminadas. Esto se ha conseguido a través del parámetro `verbose = True` . Este parámetro pertenece a la función clean_me (Ilustración 14 – Uso sencillo de AutoDataCleaner) y las siguientes líneas de código muestran los comandos para su impresión por consola.

```
if verbose:
        print("  +  droping [{}] cells".format(const))
```

La impresión se muestra a continuación.

Comando	Consola
`duplicated_var=True`	`=  AutoDataCleaner: Performing dataset droping duplicated features...` `  + droping [[]] cells`

Tabla 15 – Duplicated features y respuesta por consola

La Tabla 15 muestra que no se han hallado variables duplicadas en todo el conjunto.

7.7 COLUMNAS DUPLICADAS

Las columnas duplicadas se detectarán y se eliminarán. No se tiene en cuenta que las columnas se llamen igual, solo se eliminan si las observaciones dentro de la columna son iguales. Es de vital importancia ya que, debido a una columna doble, ya que no aportan información nueva dando lugar a graves casos de colinealidad.

A continuación, se muestra el código de llamada a la función `drop_columns_duplicated_features()`.

```
# Eliminating Duplicated features  (Teresa) ------------------------------------
----------------------
    if duplicated_var:
        if verbose:
            print(" =   AutoDataCleaner: Performing dataset droping duplicated
features...")
        df=drop_duplicated_features(df,verbose)
```

El código siguiente muestra la creación de la función `drop_columns_duplicated_features()`.

```
from feature_engine.selection import DropDuplicateFeatures
[…]
# Elimination of duplicated features
def drop_duplicated_features(df,verbose):

    transformer = DropDuplicateFeatures()
    # fit and transform the data
    df=transformer.fit_transform(df)
    dup=transformer.duplicated_feature_sets_
    if verbose:
        print("  + droping [{}] cells".format(dup))
    return df
```

Hace falta remarcar que en el script AutoDataCleaner, hay que cargar la librería `from feature_engine.selection import DropDuplicatedFeatures`. Se puede observar que se ha usado la función `drop_columns_duplicated_features()` de feature_engine como se mencionó en el apartado 4.4.1.12.4.

Se puede observar en el código anterior un comentario donde indica que se debería ejecutar primero el script donde está la función DropDuplicatedFeatures que está situada en el siguiente directorio.

```
C:\Users\Usuario\anaconda3\envs\OptimizacionVarEntrada\Lib\site-
packages\feature_engine\selection\drop_duplicated_features
```

Una vez ejecutado el código completo, se puede observar por consola, las variables cuyas observaciones contenían valores con poca o nula varianza han sido eliminadas. Esto se ha conseguido a través del parámetro `verbose = True` . Este parámetro pertenece a la función clean_me (Ilustración 14 – Uso sencillo de AutoDataCleaner) y las siguientes líneas de código muestran los comandos para su impresión por consola.

```
if verbose:
```

```
        print("  + droping [{}] cells".format(dup))
```

La impresión se muestra a continuación.

Comando	Consola
`low_var_elimination=True,`	`= AutoDataCleaner: Performing dataset droping constant or nearly constant features...` `+ droping [[]] cells`

Tabla 16 – Low variación elimination y respuesta por consola

La Tabla 16 muestra que no se han encontrado variables con poca o nula variación.

7.8 AUTOMATIZACIÓN DE LA NORMALIZACIÓN

La automatización de la normalización trata de crear una función dentro de la función de normalización, explicada en el apartado Capítulo 6. , para que el usuario no tenga que expresar manualmente las variables que quiere normalizar, sino que a través de la nueva función llamada `deciding_normalization_variables()`, el programa realice un modelo de Regresión lineal con todas las variables, pero con una variable normalizada del conjunto de datos y obteniendo para cada variable la exactitud del modelo a través del comando `score()`. Una vez obtenido la exactitud de predicción de cada modelo, las variables que hayan hecho que el modelo tenga una porción por encima de un 80%, serán normalizadas. Esta función automatizará la selección de variables normalizadas y el propio algoritmo sabrá a cuál variable habrá que implementar la normalización.

La normalización es de vital importancia ya que simplifica el proceso y reducir la redundancia, aunque los modelos de machine learning que se basen en que los datos poseen una distribución gaussiana es necesaria una normalización, muchas veces viene incluido en el modelo como, por ejemplo, el modelo de regresión lineal.

A continuación, se muestra el código de llamada a la función `deciding_normalization_variables()`.

```
def normalize_df(data,mea_var, exclude=[], verbose=True):
```

```
    """
    normalize_df function performs normalization to all columns of dataframe
excluding binary (1/0) columns

    :param df: input Pandas DataFrame
    :param exclude list of columns to be excluded when performing normalization
(usually datetime columns)
    :param verbose: print progress in terminal/cmd
    :returns: normalized Pandas DataFrame
    """
    stats = 0
    var=[]
    for col in data.columns.to_list():
        if col in exclude:
            continue
        # check if column is binray
        col_unique = data[col].unique().tolist()
        if len(col_unique) == 2 and 0 in col_unique and 1 in col_unique:
            continue
        else:
            columna=deciding_normalization_variables(data,col,mea_var)
            if columna:
                var.append(col)
            else:
                print('The variable [{}] will not be normalized because its
accuracy is too low'.format(col))
        for col in data.columns.to_list():
            if col in var:
                data[col] = (data[col]-data[col].mean())/data[col].std()
                stats += data.shape[0]
                columns=[data[col].name]
                if verbose:
                    print("  + normalized [{}]".format(columns))
                if verbose:
                    print("  + normalized {} cells".format(stats))
    return data
```

El código siguiente muestra la creación de la función `deciding_normalization_variables()`.

```
def deciding_normalization_variables(X,cl,y):
```

```
    X[cl] = (X[cl]-X[cl].mean())/X[cl].std()
    X_train, X_test, y_train, y_test = train_test_split(X,X[y], random_state=42,
train_size = .8)

    clf = LogisticRegression()
    clf.fit(X_train, y_train)
    preds=clf.predict(X_test)
    score_preds=(metrics.accuracy_score(y_test,preds))*100
    if score_preds>=80:
        colum=cl
        print('The column [{}] is transformed which produces an accuracy in the
model of {} which is over 80%'.format(colum,score_preds))
        return colum
```

Se puede observar por consola las variables que han sido normalizadas por esta función. Esto se ha conseguido a través del parámetro `verbose = True` . Este parámetro pertenece a la función clean_me (Ilustración 14 – Uso sencillo de AutoDataCleaner) y las siguientes líneas de código muestran los comandos para su impresión por consola.

```
if verbose:
      print("  + normalized [{}]".format(columns))
if verbose:
      print("  + normalized {} cells".format(stats))
```

La impresión se muestra a continuación. La tabla mostrada a continuación muestra los comandos usados para activar la función y su respuesta por consola.

Comando	Consola

`normalize=True`	`= AutoDataCleaner: Performing dataset normalization...`
	`[…]`
	`+ normalized [['PassengerId']]`
	`+ normalized 13120 cells`
	`+ normalized [['Pclass']]`
	`+ normalized 13940 cells`
	`+ normalized [['Age']]`
	`+ normalized 14760 cells`
	`+ normalized [['SibSp']]`
	`+ normalized 15580 cells`
	`+ normalized [['Parch']]`

	+ normalized 16400 cells + normalized [['Fare']] + normalized 17220 cells

Tabla 17 – Decisión tree y respuesta por consola

En la Tabla 17 – Decisión tree y respuesta por consola se puede observar que todas las variables han sido normalizadas debido a que el parámetro exactitud ha mostrado las variables que deberían de normalizarse.

7.9 AUTOMATIZACIÓN DE LA ELIMINACIÓN DE VARIABLES ÚNICAS

La automatización de la eliminación de las variables únicas se ha realizado de la siguiente forma. Con la función `unique()`, se ha creado una funcion llamada `variables_uniques()` que detecta si las variables son únicas en cada valor y si tiene el mismo numero de variables únicas que de variables en total, esa variable será eliminada del conjunto de datos.

A continuación, se muestra el código de llamada a la función `variables_uniques()`.

```
# Droping variables uniques  (Teresa) ------------------------------------------
    if variables_uni:
        if verbose:
            print(" = AutoDataCleaner: Performing recognition of unique variables
and droping them...")
        df=variables_uniques(df,verbose)
```

El código siguiente muestra la creación de la función `variables_uniques()`.

```
def variables_uniques(X,verbose):
    for col in X.columns:
        count=len(X[col].unique())
        if count==len(X[col].index):
            X.drop(columns=col,inplace=True)
            if verbose:
                print('Droping [{}] columsn because they contained {} unique values
'. format(col,count))
        else:
            print('Feature [{}] is not unique '.format(col))
            data=X
```

```
    return data
```

Se puede observar por consola las variables que han sido normalizadas por esta función. Esto se ha conseguido a través del parámetro `verbose = True` . Este parámetro pertenece a la función clean_me (Ilustración 14 – Uso sencillo de AutoDataCleaner) y las siguientes líneas de código muestran los comandos para su impresión por consola.

```
if verbose:
                print('Droping [{}] columsn because they contained {} unique values
'. format(col,count))
```

La impresión se muestra a continuación. La tabla mostrada a continuación muestra los comandos usados para activar la función y su respuesta por consola.

Comando	Consola
	= AutoDataCleaner: Performing recognition of unique variables and droping them... Droping [PassengerId] columsn because they contained 891 unique values Feature [Survived] is not unique Feature [Pclass] is not unique Feature [Sex] is not unique Feature [Age] is not unique Feature [SibSp] is not unique Feature [Parch] is not unique Feature [Ticket] is not unique Feature [Fare] is not unique Feature [Cabin] is not unique Feature [Embarked] is not unique

Tabla 18 – Variables uniques y su impresión por consola

La Tabla 18 – Variables uniques y su impresión por consola muestra las variables únicas y las elimina de la data set ya que una variable única no aporta nada al conjunto de datos.

7.10 ELIMINACIÓN DE FILAS DUPLICADAS

La eliminación de las filas duplicadas es muy importante para el aseguramiento del debido funcionamiento del apartado 7.9 donde explica el uso de la función `variables_uniques()` ya

que este detectara que las variables únicas no coinciden con el número de variables totales en el conjunto de datos lo que puede llevar a que nunca elimine variables completamente únicas. Una ventaja de esta función es que compara la fila entera, por lo tanto, si hay un conjunto de datos y hay una variable llamada nombre donde contiene el valor llamado María González, y en ese conjunto de datos, hay varias María González, pero cada una con otros rasgos en el data set (coincidiendo solo el nombre), no eliminara esa fila.

A continuación, se muestra el código de llamada a la función `duplicated_rows()`.

```
# Droping duplicates (Teresa) ----------------------------------------------------
--
    if duplicated_rows_remove:
        if verbose:
            print(" = AutoDataCleaner: Performing removal of duplicated rows...
")
        df=duplicated_rows(df)
```

El código siguiente muestra la creación de la función `duplicated_rows()`.

```
def duplicated_rows(df):
    data=df.drop_duplicates()
    count=len(df)-len(data)
    print('Duplicated columns {}'.format(count))
    print(data)
    return data
```

Se puede observar por consola las variables que han sido normalizadas por esta función. Esto se ha conseguido a través del parámetro `verbose = True` . Este parámetro pertenece a la función clean_me (Ilustración 14 – Uso sencillo de AutoDataCleaner) y las siguientes líneas de código muestran los comandos para su impresión por consola.

```
if verbose:
        print(" = AutoDataCleaner: Performing removal of duplicated rows... ")
[…]
print('Duplicated columns {}'.format(count))
```

La impresión se muestra a continuación. La tabla mostrada a continuación muestra los comandos usados para activar la función y su respuesta por consola.

Comando	Consola
	`= AutoDataCleaner: Performing removal of duplicated rows...` `Duplicated columns 0`

Tabla 19 – Variables uniques y su impresión por consola

La consola muestra que no hay ninguna fila duplicada.

Capítulo 8. AUTODATACLEANER CON OTROS CONJUNTOS DE DATOS

En este capítulo, se verificará la función `clean_me()` para dos tipos distintos de conjuntos de datos. En este capítulo se comparará este proyecto con los resultados obtenidos por usuarios de kaggle y expuestos en dicha página de los dos conjuntos de datos. En el Capítulo 9. se compararán los resultados obtenidos al inicio del proyecto (ya que partimos de una librería ya creada) hasta donde se ha conseguido llevar este. Los datos se han adjuntado en el ANEXO III. Conjuntos de datos.

Se ha considerado el uso de dos data set para poder observar como `clean_me()` afecta a varios conjuntos de datos. Si los limpia y los procesa como es debido, llevará a cabo todos los cambios que se desean realizar. Este paso es de vital importancia ya que es una forma única de saber cómo el programa funciona, no solo con el conjunto de datos con el que se ha ido construyendo el programa entero, sino con otros conjuntos de datos. Estos conjuntos de datos se han escogido debido a que son conjuntos de datos típicos en el área de machine learning con una extensa flota de respuestas disponibles para comparar.

8.1 TITANIC

Para crear el programa, se ha utilizado una base de datos para ir viendo los cambios necesarios. Dicho conjunto de datos es Titanic dataset. Fue creado por Frank E. Harrell Jr. y Thomas Carson. En el ANEXO II. Licencias, se puede observar la información citada sobre Titanic Dataset y en el ANEXO III. Conjuntos de datos en el apartado Titanic Dataset en la zona de Antes AutoDataCleaner se puede observar un archivo .csv convertido a .pdf con los datos de partida del Titanic Dataset. Este conjunto de datos contiene 1310 observaciones con 14 variables. Se eligió este data set por un clásico muy estudiado que nos proporciona la posibilidad de una comparación de nuestro programa.

Para este set de datos, se ha tenido que preparar de la siguiente forma.

```
from AutoDataCleaner.AutoDataCleaner import read_csv
def load_titanic():
        data = read_csv('https://www.openml.org/data/get_csv/16826755/phpMYEkMl')
        data = data.replace('?', np.nan)
        data['cabin'] = data['cabin'].astype(str).str[0]
        data['pclass'] = data['pclass'].astype('O')
        data['embarked'].fillna('C', inplace=True)
        return data

# load data as pandas dataframe
df = load_titanic()
```

El uso de `clean_me()` puede observarse en el ANEXO III. Conjuntos de datos en el apartado Titanic Dataset en la zona Despues AutoDataCleaner con un total de 1248 variables y 9 columnas. Las variables finales coinciden con el resultado expuesto en Kaggle creado por Manac Sehgal y que se explicara detalladamente en el Capítulo 9. La única diferencia recae en la variable *cabin* donde Manav Sehgal ha observado que cabin tiene muchos valores vacíos, pero al realizar la apertura en este proyecto del https, no hay valores vacíos dentro de la variable cabin.

Las matrices de confusión se muestran a continuación. Las matrices de confusión contienen en sus diagonales principales, los estimados de forma correcta por el modelo, tanto los verdaderos positivos y verdaderos negativos. La otra diagonal es lo que se llama error de tipo 1 y error de tipo 2. Normalmente es peor tener un error de tipo 1, que es rechazar la hipótesis nula siendo esta verdadera.

Proyecto	**Valores reales**		
Valores predichos		**Negativos**	**Positivos**
	Negativos	429	71
	Positivos	98	222

Tabla 20 – Matriz de confusión para titanic dataset con Proyecto

Manac Sehgal	**Valores reales**		
Valores predichos		**Negativos**	**Positivos**
	Negativos	473	76
	Positivos	112	230

Tabla 21 – Matriz de confusión para titanic con Manac Sehgal proyecto

En las Tabla 20 y Tabla 21se pueden ver las matrices de confusión para cada uno de los dos proyectos, el creado en este TFG y el creado por con Manac Sehgal y colgado de forma publica en kaggle. De estas matrices obtendremos la exactitud, sensibilidad, precisión y especificidad.

La comparación del conjunto de datos preparado por kaggle y el del proyecto se muestra a continuación. Se ha comparado la exactitud de nuestro programa realizando un análisis de regresión logística con el realizado por Manac Sehgal y colgado en Kaggle. Este parámetro sirve para observar el porcentaje de predicciones correctas frente al total.

Proyecto	**Manac Sehgal**
79,40%	80.36%

Tabla 22 – Exactitudes de Titanic dataset

En la Tabla 22 – Exactitudes, se puede ver ambas exactitudes para un modelo de regresión logística siendo el valor de Manac Sehgal casi un 1% mayor que el proyecto creado. Esto se puede deber al hecho de haber limpiado más nuestro conjunto de datos, hayamos perdido un poco de información, pero se ha ganado tiempo y todo está automatizado. Para la simplicidad de usar este proyecto, en vez de hacerlo a mano, un 1% es insignificante con todo lo que se ha ganado. En el Capítulo 9. se analizará en concreto, la matriz de confusión y la curva

ROC[18] realizadas con un modelo de regresión logística y con el área AUC[19] de donde se partió hasta donde se ha conseguido llevar este proyecto.

La precisión que mide lo cerca que se ha quedado el resultado de la predicción del valor positivo se muestra para ambos casos en la Tabla 23 – Precisión de titanic .

Proyecto	**Manac Sehgal**
75,51%	75,16%

Tabla 23 – Precisión de titanic

La precisión del modelo es en ambos casos muy similares y se puede decir que son igual de precisos.

La siguiente tabla muestra la especifidad que mide la tasa de verdaderos negativos y se muestra a continuación en la Tabla 24 – Especifidad de

Proyecto	**Manac Sehgal**
85,60%	86,15%

Tabla 24 – Especifidad de titanic

La especificad es un poco más mayor en el proyecto creado por Manac Sehgal que en el proyecto TFG. Esto quiere decir que el programa creado por Manac Sehgal tiene más capacidad de discriminar casos negativos.

La sensibilidad que se refiere a la tasa de verdaderos positivos y se muestra a continuación en la Tabla 25 – Sensibilidad de .

[18] ROC (Receiver Operating Characteristic) son curvas en las que se presenta la sensibilidad en función de los falsos positivos (complementario de la especificidad) para distintos puntos de corte. [62]
[19] El área AUC, es decir el área bajo la curva ROC (Receiver Operating Characteristic o Característica Operativa del Receptor), es una herramienta estadística que se utiliza para medir el acierto en la predicción de eventos binarios, es decir, eventos que bien ocurren o no ocurren. [62].

Proyecto	Manac Sehgal
69,375%	67,25%

Tabla 25 – Sensibilidad de titanic

De la Tabla 25 – Sensibilidad de se puede concluir que al proyecto creado en este TFG se le escapan menos positivos, es decir detecta bien los casos positivos.

8.2 CREDIT APPROVAL

```
El siguiente conjunto de datos es Credit Approval dataset fue creado por D. Dua y
C. Graff datado de 2019 y creado en Irvine, California. Este conjunto de datos
contiene 690 observaciones con 16 variables. En el ANEXO III. Conjuntos de datos
apartado
```

Credit Approval DataSet en la zona de Antes AutoDataCleaner puede observar un archivo *".data"* convertido en pdf con los datos de partida.

Para este set de datos, se ha tenido que preparar de la siguiente forma.

```
import random
import pandas as pd
import numpy as np
from AutoDataCleaner.AutoDataCleaner import read_csv

# load data
data = read_csv('crx.data', header=None)

# create variable names according to UCI Machine Learning information
varnames = ['A'+str(s) for s in range(1,17)]
data.columns = varnames

# replace ? by np.nan
data = data.replace('?', np.nan)

# re-cast some variables to the correct types
data['A2'] = data['A2'].astype('float')
data['A14'] = data['A14'].astype('float')

# encode target to binary
data['A16'] = data['A16'].map({'+':1, '-':0})

# save the data
data.to_csv('creditApprovalUCI.csv', index=False)
```

```
El uso de clean_me() puede observarse en el ANEXO III. Conjuntos de datos en el
apartado
```

Credit Approval DataSet en la zona del Despues AutoDataCleaner con un total de 586 valores y 14 columnas. El resultado expuesto en Kaggle creado por ZJ Low se explicará detalladamente en el Capítulo 9. La diferencia recae en que, en nuestro proyecto, las variables A5 y A13 han sido eliminadas debido a que la columna A5 tiene un alto grado de correlación con A4 y que la columna A13 tiene un alto número de valores constantes.

Las matrices de confusiones se muestran a continuación.

Proyecto	**Valores reales**		
Valores predichos		**Negativos**	**Positivos**
	Negativos	153	19
	Positivos	27	192

Tabla 26 – Matriz de confusión para credit approval con Proyecto

ZJ Low	**Valores reales**		
Valores predichos		**Negativos**	**Positivos**
	Negativos	93	10
	Positivos	27	98

Tabla 27 – Matriz de confusión para credit approval con ZJ Low proyecto

En las Tabla 26 – Matriz de confusión para credit approval con Proyecto y Tabla 27 – Matriz de confusión para credit approval con ZJ Low proyecto se pueden ver las matrices de confusión para cada uno de los dos proyectos, el creado en este TFG y el creado por ZJ Low y colgado de forma publica en kaggle. De estas matrices obtendremos la exactitud, sensibilidad, precisión y especificidad.

La exactitud de nuestro programa y la exactitud obtenida a través del programa creado ZJ Low y que se encuentra en Kaggle se encuentra en la Tabla 28 – Exactitudes de Credit approval Este parámetro sirve para observar el porcentaje de predicciones correctas frente al total.

Proyecto	ZJ Low
88,23%	87,01%

Tabla 28 – Exactitudes de Credit approval

La exactitud del modelo se ha obtenido con el comando `score(X_train,y_train)` y se ha obtenido utilizando kaggle notebooks. Se puede observar que el proyecto creado tiene más exactitud que el colgado en kaggle.

La precisión que mide lo cerca que se ha quedado el resultado de la predicción del valor positivo se muestra para ambos casos en la Tabla 29 – Precisión de Credit approval.

Proyecto	ZJ Low
90,99%	90,74%

Tabla 29 – Precisión de Credit approval

La precisión del modelo es en ambos casos muy similares y se puede decir que son igual de precisos.

La siguiente tabla muestra la especifidad que mide la tasa de verdaderos negativos y se muestra a continuación en la Tabla 30 – Especifidad de Credit approval.

Proyecto	ZJ Low
88,95%	90,29%

Tabla 30 – Especifidad de Credit approval

La especificad es un poco más mayor en el proyecto creado por ZJ Low que en el proyecto. Esto quiere decir que el programa creado por ZJ Low tiene más capacidad de discriminar casos negativos.

La sensibilidad que se refiere a la tasa de verdaderos positivos y se muestra a continuación en la Tabla 31 – Sensibilidad de Credit approval.

Proyecto	ZJ Low
87,67%	78,46%

Tabla 31 – Sensibilidad de Credit approval

De la Tabla 31 – Sensibilidad de Credit approval, se puede concluir que al proyecto creado en este TFG se le escapan menos positivos, es decir detecta bien los casos positivos.

Se ha podido comprobar que, para distintos data set, el proyecto funciona y, por tanto, se puede afirmar que no debería haber ningún tipo de incidente a la hora de probar con otros conjuntos de datos. También se quiere remarcar que el proyecto ha quedado parecido al obtenido por Kaggle por distintos usuarios en cada conjunto de datos y que, en algunas métricas, los valores obtenidos por este proyecto, son más altos. No obstante, los resultados son bastante parecidos, lo cual era de esperar, ya que el proyecto tiene como fin ahorrar tiempo de maquina y de ejecución al usuario y obteniendo un resultado final, donde se encuentre unas métricas de medición del modelo buenas y razonables.

Capítulo 9. ANÁLISIS DE RESULTADOS

En el capítulo que viene a continuación, se resumirán los resultados obtenidos solo con las funciones modificadas o añadidas. Se usará la librería llamada Titanic dataset . La información detallada de este conjunto de datos se encuentra en el Capítulo 8. .

La viabilidad de la función se ha comprobado observando en Kaggle la sección de *"code",* donde los usuarios han participado en la competición abierta sobre la predicción de sobrevividos con el conjunto de datos especificado en el ANEXO III. Conjuntos de datos subapartado Titanic Dataset, y escogiendo el más votado por los usuarios. El más votado fue creado por Manav Sehgal con más de un millón de visitas y se creó hace 3 años [27, 28] . Esta comprobación y verificación se ha realizado en el Capítulo 8. En este capítulo, se pretende analizar cada función por separado, comparando lo que nos daría si no se hubiera modificado y añadido las librerías a AutoDataCleaner. La misión de este Trabajo de Fin de Grado es añadir un peldaño más a la limpieza y depuración de datos, automatizando las principales funciones necesarias para ejecutar modelos de machine learning y añadiendo funciones necesarias que la librería AutoDataCleaner no tenía.

En el ANEXO I. Códigos se ha copiado y pegado en la sección Titanic Comprobación, el código base de donde se comenzó a realizar este proyecto. Se puede encontrar su script principal con la llamada a la función clean_me, perteneciente en este caso a ADC_2 y el cual se ha pegado, en la siguiente sección, para que se pueda comprobar con la versión expuesta en GitHub de AutoDataCleaner, que no ha habido ningún cambio en esta parte del programa con el fin de analizar cuanto se ha cambiado. La impresión por consola se detalla también en dicho anexo, en la sección

Consola Titanic comprobación.

En los siguientes apartados se comprobará las medidas de precisión, exactitud, especifidad y sensibilidad del modelo a la vez que las respectivas curvas ROC realizadas con un modelo de regresión logística y el valor AUC para el antes y el después de este proyecto.
Cabe destacar que ambos programas que se pondrán bajo observación son a través de Titanic dataset.

El AutoDataCleaner de donde se partió, contaba con 6 funciones mencionadas en el apartado 4.4.1.7. Esas functions son `cols_to_datetime_dtype()`, `remove_columns()`, `detect_binary()`, `convert_numeric_df()`, `one_hot_df()` y `clean_na_df()`.

El AutoDataCleaner, después del proyecto, consta con 7 funciones añadidas y la modificación de algunas ya existentes como el `clean_nan()` o `normalize_df()`. En este proyecto se han automatizado la detección e variable tiempo y su conversión, las variables que necesitan normalización y las que necesitan decisión tree encoder. Algunas funciones ya están automatizadas de por si como la búsqueda inteligente de variables altamente correlacionadas, las variables duplicadas, filas duplicadas, las variables con poca varianza, las variables binarias, las variables únicas, detección de valores atípicos por rangos y la detección de valores nulos, y automáticamente clasificarlos si van a eliminar la variable, la fila o a rellenar los valores faltantes, sin contar que dichas funciones tiene la eliminación dentro de cada función. La normalización esta también automatizada y la conversión de variable a numéricas que no hayan sido convertidas por el árbol de decisión también.

Una copia del script donde aparece la función clean_me sin modificaciones, aparece en el ANEXO I. Códigos en el apartado Titanic Comprobación en el subapartado AutoDataCleaner Titanic Comprobación.

La matriz de confusión antes y después de las funciones añadidas se muestra a continuación.

Antes modificaciones	**Valores reales**		
Valores predichos		**Negativos**	**Positivos**
	Negativos	478	71
	Positivos	108	234

Tabla 32 – Matriz de confusión para titanic dataset antes de las modificaciones

Proyecto	**Valores reales**		
Valores predichos		**Negativos**	**Positivos**
	Negativos	429	71
	Positivos	98	222

Tabla 33 – Matriz de confusión para titanic dataset después de las modificaciones

En la Tabla 32 – Matriz de confusión para titanic dataset muestra el número de valores predichos correctamente y los errores de tipo 1 y tipo 2. El error de tipo 1 se ha cometido con 71 valores de la muestra, también llamado un falso positivo, el error de tipo 2 se ha cometido en 108 valores de la muestra, también llamado falso negativo. Por norma general, el error de tipo 1 es más peligroso que el error de tipo 2. De esta matriz, se pueden obtener la sensibilidad, exactitud, especifidad y precisión.

En la Tabla 33 – Matriz de confusión para titanic dataset después de las modificaciones, se pueden observar la matriz de confusión con el programa AutoDataCleaner transformado, es decir, con las modificaciones realizadas por este proyecto.

	Antes	**Después**
Exactitud	79,91%	79,40%
Sensibilidad	68,42%	69,38%
Precisión	76,72%	75,77%
Especifidad	87,07%	85,80%

Tabla 34 – Métricas de clasificación del algoritmo antes y después de modificaciones

La Tabla 34 – Métricas de clasificación del algoritmo antes y después de modificaciones muestra las métricas de clasificación del modelo. En este caso, el proyecto antes y después tiene unas métricas muy parecidas. El proyecto creado tiene una diferencia muy pequeña comparado con el antes del proyecto, siendo la del proyecto entre menos de un 1% de diferencia hasta casi un 2%. Estas discrepancias son mínimas para el efecto global que se ha conseguido obtener con este proyecto. Cabe destacar que una vez alcanzados valores altos en cada una de las métricas del modelo, es complicado aumentarlas.

A continuación, se muestra las longitudes del conjunto de datos obtenido antes y después de las modificaciones creadas en la librería AutoDataCleaner mencionados en Capítulo 7.

	Columnas	**Filas**
Antes	839	891
Después	8	820

[19]Cabe destacar, una vez más, que el conjunto de datos obtenido con el proyecto modificado, como se puede ver **¡Error! No se encuentra el origen de la referencia.**, tiene una reducción significativa de columnas, de 839 que creo el AutoDataCleaner antes de las modificaciones a 7. Esta creación innecesaria ha ocurrido debido a que la librería creo categorías a valores distintos encontrados en el conjunto de datos. Gracias al proyecto aquí presente, dando las mismas mínimas indicaciones a cada una de las librerías, se ha conseguido reducir los valores de 839 columnas a 7 columnas. Lo que ha sido un avance muy grande.

A continuación, se muestran las curvas ROC realizadas con un modelo de regresión logística con el área AUC y su importancia en este proyecto.

La medición del rendimiento se hace a través de las curvas ROC y el área bajo esa curva, el AUC. La curva ROC indica que tan bueno es nuestro modelo para distinguir entre dos categorías. Las ventajas de este método de medición del rendimiento del modelo es que es invariable a la escala y al umbral de clasificación. [28].

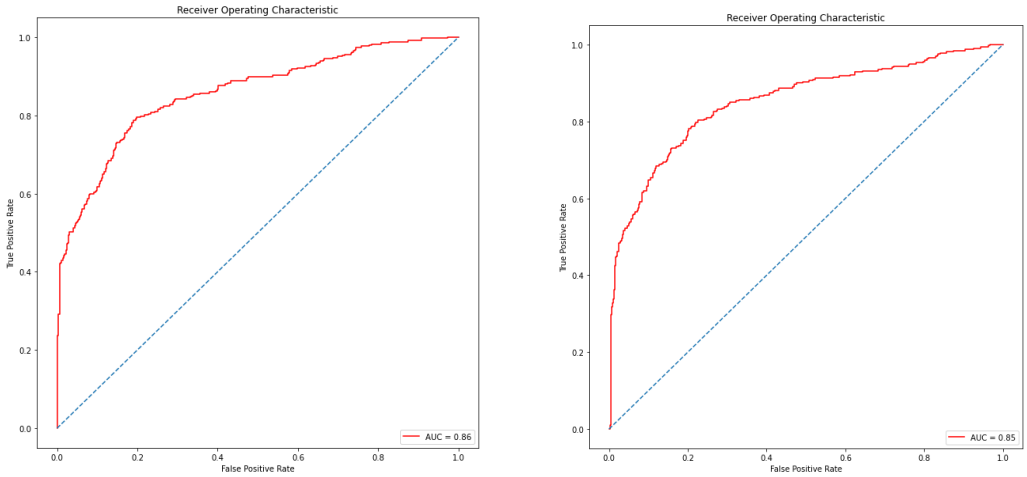


Ilustración 20 – Curva ROC de AutoDataCleaner antes de las modificaciones y después de las modificaciones

Las Ilustración *20 – Curva ROC de AutoDataCleaner antes de las modificaciones* muestran las curvas ROC del antes y el después de las modificaciones realizadas. La Ilustración 24 muestra la curva ROC antes de las modificaciones y tiene un AUC del 0,86 (se puede observar en la esquina derecha inferior de la Ilustración 24) y en cambio, la Ilustración 25 muestra la curva ROC despues de todas las modificaciones y tiene un AUC de 0,85 (se puede, tambien ver, en la esuqina inferior derecha de la Ilustración 25). Un reusltado perfecto e imposible de obtener es un AUC de 1 y el peor de los casos seria un AUC de 0,5. Ambos AUC poresentados en estas ilustraciones muestran que los valores han sido disgnosticados.

Aunque estas dos ilustraciones muestren un AUC con un punto de diferencia entre ellos, se puede decir que tienen la misma validez de prediccion. Se vuelve a destacar que, aunque tenga el modelo creado un punto menos de AUC que el antes del proyecto, la reduccion de variables junto a la automatizacion de mas categorias era el objetivo primordial de este proyecto.

Capítulo 10. CONCLUSIONES Y TRABAJOS FUTUROS

10.1 CONCLUSIONES

Para finalizar este proyecto de fin de carrera, se retrocederá al Capítulo 1. donde se introdujo la problemática. Este proyecto ha cumplido con todos los objetivos destacados en el apartado 3.1 incluidos los objetivos del desarrollo sostenible. La librería creada con el principal objetivo de realizar la limpieza y depuración de los datos se ha realizado de forma razonable y las transformaciones realizadas son las principales que todo analista necesita. En el apartado siguiente, se puede ver los siguientes trabajos futuros propuestos.

Cabe añadir a la conclusión, que hay muchas transformaciones que no han sido abordadas, este proyecto solo abarca las principales funciones y principalmente sus automatizaciones con los respectivos comentarios para dar al analista un punto de vista interno sobre las funciones de la librería y una explicación del porqué de cada acción.

Una vez creada la librería y ejecutada, el conjunto de datos final podrá ser utilizado para los siguientes pasos en análisis de datos. Como se ha comprobado en el Capítulo 8. la creación del proyecto ha sido un éxito. La exactitud de este programa con la librería básica AutoDataCleaner es muy similar, lo que comprueba el funcionamiento de este proyecto. Esta comprobación se ha realizado solo para el conjunto titanic en el Capítulo 9. El proyecto de forma general ha sido probado en el Capítulo 8. no solo para el conjunto de datos con el que se ha trabajado (Titanic) sino que también, para el conjunto de datos Credit Approval UCI y los resultados obtenidos han sido comparados con Kaggle.

Cabe destacar que, aunque no se haya hecho una demostración con todos los parámetros cambiados del programa, está listo para su uso.

Este proyecto será expuesto en GitHub más adelante para devolver a la comunidad científica esta librería para su uso y sus posibles mejoras y dar la oportunidad a otros analistas de perfeccionar y ampliar este proyecto. Es de vital importancia el respeto hacia el código abierto, como se mencionó en el Capítulo 2. ya que el progreso viene de la divulgación libre de la tecnología, su uso y explotación. El concepto de propiedad intelectual ha sido criticado por el Instituto Von Mises, siendo las opiniones a favor de la propiedad intelectual donde resaltan que al tener una protección sobre la propiedad intelectual, aumentan la cuota de mercado de las empresas, mejorando su rendimiento, y abriendo nuevos mercados y oportunidad de internalización y según el punto de vista liberal sobre la propiedad intelectual, cabe destacar que Boldrine y Levine en 2008, entre otros, estipularon que "Por otra parte, los economistas de la corriente principal han demostrado que las patentes y los derechos de autor, lejos de promover la innovación, en realidad obstaculizan el desarrollo económico y la destrucción creativa schumpeteriana. Esto se debe a que los titulares de patentes y derechos de autor son efectivamente monopolistas intelectuales, capaces de cortar de raíz el desarrollo comercial de cualquier idea dada".

10.2 TRABAJOS FUTUROS

A medida que este proyecto ha ido cogiendo forma, han surgido nuevas funciones que podrían ser interesantes a la hora de presentar un trabajo próximo.

Este programa abarca las principales funciones que deberían implementarse en un proceso de limpieza y de data wrangling del set de datos y existe la posibilidad de futuras mejoras tanto en el programa como su redacción.

Par comenzar, la implementación y creación de variables nuevas, también puede ser de gran aportación. Feature-engine ofrece 3 posibles funciones para la creación de variables y podrían implementarse y automatizarse para que ML tome las decisiones, basándose en la exactitud del modelo.

En segundo lugar, la implementación de más tipos de codificadores de datos disponibles en feature_engine, como CountFrequencyEncoder, OrdinalEncoder o algunos mas sofisticados como WoEEncoder que se encarga de reemplazar las categorías por el peso de las evidencias y con una validación del modelo para volver estas implementaciones automáticas.

En tercer lugar, la elección automática del modelo puede ser un trabajo futuro muy interesante y comparando la validez con cada modelo (Linear Regression, Logistic Regression, KNN or k-Nearest Neighbors, Support Vector Machines, Naive Bayes classifier, Decision Tree, Random Forrest, Perceptron, Artificial neural network, RVM o Relevance Vector Machine) para que el analista no tenga que elegirlo.

En cuarto lugar, que ML elija las variables que son relevantes observando y aprendiendo con otros conjuntos de datos y comparándolos. Estos e realiza con el fin de que el algoritmo aprenda a ver que variables podrían descartarse debido a su significado.

Capítulo 11. BIBLIOGRAFÍA

[1] S. Digital, «Neoludismo: ¿vuelven a resurgir los anti-tecnología en 2020?,» *La Razon* , 24 Febrero 2020.

[2] A. E. Vidal, «ALAN TURING Y EL NACIMIENTO DE LA INTELIGENCIA ARTIFICIAL,» *Antena de Telecomunicación,* p. 45, Marzo 2007.

[3] X. Wang y C. Wang, «Time Series Data Cleaning: A Survey,» *IEEE Access,* vol. 8, pp. 1866-1881, 6 Enero 2020.

[4] T. Nagle, T. C. Redman y D. Sammon, «Harvard Business Review,» 11 Septiembre 2017. [En línea]. Available: https://hbr.org/2017/09/only-3-of-companies-data-meets-basic-quality-standards. [Último acceso: 7 Abril 2022].

[5] S. García, S. Ramírez-Gallego, J. Luengo y F. Herrera, «Big Data: Preprocesamiento y calidad de datos,» *Big Data Monografía,* vol. 237, pp. 17-23, Julio-Octubre 2016.

[6] J. W. Tukey, «The Future of Data Analysis,» *The Annals of Mathematical Statistics,* vol. 33, nº 1, pp. 1-67, 1962.

[7] P. Naur, Concise Survey of Computer Methods, Studentlitteratur AB, 1981.

[8] R. España, «Agencia b12,» 10 Septiembre 2019. [En línea]. Available: https://agenciab12.com/noticia/que-es-ciencia-de-datos. [Último acceso: 8 Junio 2022].

[9] K. Quintana, «Data Science Cental. A community for big data practitioners,» 30 Julio 2018. [En línea]. Available: https://www.datasciencecentral.com/top-10-challenges-to-practicing-data-science-at-work/. [Último acceso: 19 Junio 2022].

[10] S. Titanic, «pypi,» 22 Mayo 2021. [En línea]. Available: https://pypi.org/project/AutoDataCleaner/. [Último acceso: 9 Marzo 2022].

[11] S. Galli, N. Galli, E. Sohayb y P. Suryawanshi, «Feature_engine,» 22 Enero 2021. [En línea]. Available: https://feature-engine.readthedocs.io/en/1.0.x/index.html. [Último acceso: 27 Febrero 2022].

[12] N. Epstein, «GitHub,» 30 Junio 2020. [En línea]. Available: https://github.com/NathanEpstein/Dora. [Último acceso: 17 Marzo 2022].

[13] R. S.Olson, «GitHub,» 06 Marzo 2016. [En línea]. Available: https://github.com/rhiever/datacleaner. [Último acceso: 18 Abril 2022].

[14] L. E. Inc., «Pypi,» 2 Mayo 2021. [En línea]. Available: https://pypi.org/project/Auptimizer/. [Último acceso: 30 Marzo 2022].

[15] A. Inc, «FeatureTools,» 2020. [En línea]. Available: https://featuretools.alteryx.com/en/stable/getting_started/afe.html. [Último acceso: 23 Marzo 2022].

[16] L. Berti, «GitHub,» 16 Marzo 2021. [En línea]. Available: https://github.com/LaureBerti/Learn2Clean/blob/master/readme.rst. [Último acceso: 10 Marzo 2022].

[17] S. B. Kotsiantis, D. Kanellopoulos y P. Pintelas , «Data Preprocessing for Supervised Leaning,» *INTERNATIONAL JOURNAL OF COMPUTER SCIENCE* , pp. 111-117, 2006.

[18] R. Invarato, «jarroba,» 17 Septiembre 2020. [En línea]. Available: https://jarroba.com/entornos-virtuales-de-python-comun-y-anaconda/. [Último acceso: 17 Abril 2022].

[19] A. S. Aberca, «Aprende con Alf,» 3 Mayo 2022. [En línea]. Available: https://aprendeconalf.es/docencia/python/manual/pandas. [Último acceso: 5 Abril 2022].

[20] «Quick & Data Science,» 07 Octubre 2021. [En línea]. Available: https://quickdatascienceds.blogspot.com/2021/10/outliers-capping.html. [Último acceso: 20 Marzo 2022].

[21] «Wikipedia,» 1 Abril 2022. [En línea]. Available: https://es.wikipedia.org/wiki/Kaggle. [Último acceso: 7 Junio 2022].

[22] G. Khronos, «OpenML,» [En línea]. Available: https://www.openml.org/. [Último acceso: 20 Mayo 2022].

[23] D. Dua y C. Graff, «Indexof/ml/machine-learning-databases/credit-screening,» 2019. [En línea]. Available: http://archive.ics.uci.edu/ml/machine-learning-databases/credit-screening/. [Último acceso: 20 Junio 2022].

[24] D. R. Rosen, «Towards Data Science,» 16 Agosto 2021. [En línea]. Available: https://towardsdatascience.com/auto-detect-and-set-the-date-datetime-datatypes-when-reading-csv-into-pandas-261746095361. [Último acceso: 28 Mayo 2022].

[25] F. M. O. Peinado, «Universidad de Granada,» [En línea]. Available: https://www.ugr.es/~fmocan/MATERIALES%20DOCTORADO/Tratamiento%20de%20outliers%20y%20missing.pdf. [Último acceso: 1 Julio 2022].

[26] A. Parbhakar, «Towards Data Science,» 27 Mayo 2018. [En línea]. Available: https://towardsdatascience.com/why-data-scientists-love-gaussian-6e7a7b726859. [Último acceso: 17 Junio 2022].

[27] M. Sehgal, «Kaggle,» 2019. [En línea]. Available: https://www.kaggle.com/code/startupsci/titanic-data-science-solutions. [Último acceso: 10 Julio 2022].

[28] R. Shubham y S. Majumdar, «GeeksforGeeks,» 12 Febrero 2020. [En línea]. [Último acceso: Marzo 2022].

[29] L. Gonzalez, *Curvas Roc y Area bajo la curva (AUC),* Curso Machine Learning con Python, 2019.

[30] F. E. H. Jr y T. Cason, «OpenML,» 16 Octubre 2017. [En línea]. Available: https://www.openml.org/search?type=data&sort=runs&id=40945&status=active. [Último acceso: 26 Abril 2022].

[31] «¿Que Significa CMD?,» [En línea]. Available: http://www.quesignifica.org/cmd/. [Último acceso: 5 Junio 2022].

[32] J. Wang, Encyclopedia of Data Warehousing and Mining, Second Edition, Hershey, New York : Information science Reference.

[33] P. R. d. l. Santos, «Think Big,» 13 Diciembre 2021. [En línea]. Available: https://empresas.blogthinkbig.com/como-interpretar-la-matriz-de-confusion-ejemplo-practico/. [Último acceso: 13 Julio 2022].

[34] A. Géron, Hand-On Machine Learning with Scikit-Learn, Keras & TensorFlow, Croydon: O'Reilly, 2019.

[35] A. C. Álvarez, Programas con Python3, Wroclaw: Independently published (8 marzo 2021), 2021.

[36] J. VanderPlas, Python DataScience Handbook, Sebastopol: O'reilly, 2016.

[37] K. Ates, «Rentspree,» 17 Enero 2020. [En línea]. Available: https://www.rentspree.com/blog/what-is-credit-screening. [Último acceso: 20 Abril 2022].

[38] «CESUR,» [En línea]. Available: https://www.cesurformacion.com/blog/profesiones-del-futuro. [Último acceso: 2 Junio 2022].

[39] H. R. y. Cajal. [En línea]. Available: http://www.hrc.es/bioest/roc_1.html. [Último acceso: 12 Julio 2022].

[40] O. P. d. L. Creator, «DelftStack,» 17 Diciembre 2020. [En línea]. Available: https://www.delftstack.com/es/api/numpy/python-numpy-unique/. [Último acceso: 12 Julio 2022].

[41] M. Drehmann y K. Tsatsaronis, «La brecha crédito/PIB y los colchones de capital anticíclicos: preguntas y respuestas1,» Informe Trimestral del BPI, 9 Marzo 2014. [En línea]. Available: https://www.bis.org/publ/qtrpdf/r_qt1403z_es.htm#:~:text=El%20%C3%A1rea%20AUC%2C%20es%20decir,bien%20ocurren%20o%20no%20ocurren.. [Último acceso: 15 Julio 2022].

[42] A. Fernandez, «Ander Fernández Jauregui. Data Scientist & Business Intelligence,» [En línea]. Available: https://anderfernandez.com/blog/como-crear-api-en-python/. [Último acceso: 22 Junio 2022].

[43] E. Heuson, «GitHub,» 5 Abril 2022. [En línea]. Available: https://github.com/spyder-ide/spyder/issues/17616. [Último acceso: 30 Abril 2022].

[44] «IBM,» 17 08 2021. [En línea]. Available: https://www.ibm.com/docs/es/spss-modeler/SaaS?topic=language-object-oriented-programming. [Último acceso: 24 Junio 2022].

[45] E. F. Lastra, «Artyco,» [En línea]. Available: https://artyco.com/que-es-un-arbol-de-decision-y-su-importancia-en-el-data-driven/. [Último acceso: 10 Mayo 2022].

[46] «Lucidchart,» [En línea]. Available: https://www.lucidchart.com/pages/es/que-es-un-diagrama-de-arbol-de-decision. [Último acceso: 1 Mayo 2022].

[47] J. F. López, «Desviación estándar o típica.,» 2 Octubre 2017. [En línea]. Available: https://economipedia.com/definiciones/desviacion-tipica.html. [Último acceso: 19 Abril 2022].

[48] F. Machuca, «Crehana,» 21 Marzo 2021. [En línea]. Available: https://www.crehana.com/blog/data-analitica/data-cleansing/. [Último acceso: 20 Enero 2022].

[49] «Oracle,» [En línea]. Available: https://www.oracle.com/mx/database/what-is-database/. [Último acceso: 3 Mayo 2022].

[50] J. S. d. Pablos, «Camara de comercio de España,» 10 October 2016. [En línea]. Available: https://www.camara.es/blog/innovacion-y-competitividad/la-importancia-de-la-propiedad-industrial-en-el-exito-empresarial. [Último acceso: 4 Junio 2022].

[51] «Programador clic,» [En línea]. Available: https://programmerclick.com/article/74081369288/. [Último acceso: 26 Mayo 2022].

[52] «Python,» 12 Junio 2022. [En línea]. Available: https://docs.python.org/es/3/tutorial/venv.html. [Último acceso: 15 Marzo 2022].

[53] S. Ranjan y A. Singh, «GeeksforGeeks,» 01 Julio 2021. [En línea]. Available: https://www.geeksforgeeks.org/python-pandas-dataframe-ffill/. [Último acceso: 28 Junio 2022].

[54] Rebeca, «DataScientest,» 7 Abril 2022. [En línea]. Available: https://datascientest.com/es/datacleaning-limpieza-de-datos-definicion-tecnicas-importancia-en-data-

science#:~:text=Sin%20la%20limpieza%2C%20es%20probable,de%20la%20transformaci%C3%B3n%20de%20datos.. [Último acceso: 4 Febrero 2022].

[55] A. Smith, «Building the future with software,» [En línea]. Available: https://www.adamsmith.haus/python/answers/how-to-create-a-timed-loop-in-python. [Último acceso: 25 Mayo 2022].

[56] M. Sotaquirá, «codificandobits {cb},» 15 Junio 2021. [En línea]. Available: https://www.codificandobits.com/blog/manejo-datos-faltantes/. [Último acceso: 24 Marzo 2022].

[57] P. Tejeiro, «Thinking Big,» [En línea]. Available: https://blogthinkbig.com/la-irrupcion-del-open-machine-learning/. [Último acceso: 22 Junio 2022].

[58] «Universidad de Valencia,» [En línea]. Available: https://www.uv.es/webgid/Descriptiva/23_valores_faltantes.html. [Último acceso: 20 Marzo 2022].

[59] Y. Verma, «Analytics in diamag,» 6 September 2021. [En línea]. Available: https://analyticsindiamag.com/a-complete-guide-to-categorical-data-encoding/. [Último acceso: 5 Mayo 2022].

[60] J. B. Wiśniewski, «Mises.org,» 08 Mayo 2020. [En línea]. Available: https://mises.org/es/library/sobre-la-imposibilidad-de-la-propiedad-intelectual. [Último acceso: 8 Julio 2022].

[61] Zach, «Statology,» 12 Marzo 2021. [En línea]. Available: https://www.statology.org/interquartile-range-vs-standard-deviation/. [Último acceso: 23 Abril 2022].

ANEXO I. CÓDIGOS

TITANIC PROYECTO

SCRIPT TITANIC

```
import time
import numpy as np
import AutoDataCleaner.AutoDataCleaner as adc
from AutoDataCleaner.AutoDataCleaner import read_csv
from sklearn import metrics
from sklearn.linear_model import LogisticRegression

ini=time.time()

print('----------------SHOWINGTHE DATA SET -----------------')

df =read_csv('train_t.csv')
print(df.info())
data=adc.clean_me(df,
            detect_binary=True,
            numeric_dtype=True,
            decision_tree=True,
            one_hot=True,
            normalize=True,
            datetime_columns=[],
            remove_columns=['Name'],
            high_corr_elimination=True,
            low_var_elimination=True,
            measuring_variable='Survived',
            variable_to_encode=[],
            outlier_removal=True,
            duplicated_var=True,
            duplicated_rows_remove=True,
            variables_uni=True,
            clean=True,
            verbose=True)

print(data)
fin=time.time()
```

```
tiempo_ejecucion=fin-ini
print(tiempo_ejecucion)



y_train=data['Survived']
X_train=data.drop(columns=['Survived'])
df_test=read_csv('test_t.csv')
y_test=data['Survived']
X_test=data.drop(columns=['Survived'])

X=data
y=data['Survived']
model_LR= LogisticRegression()
model_LR.fit(X_train,y_train)
y_prob = model_LR.predict_proba(X_test)[:,1] # This will give you positive class
prediction probabilities
y_pred = np.where(y_prob > 0.5, 1, 0) # This will threshold the probabilities to
give class predictions.
acc_log=model_LR.score(X_train,y_train)
model_LR.score(X_test, y_pred)
confusion_matrix=metrics.confusion_matrix(y_test,y_pred)
print(confusion_matrix)
auc_roc=metrics.roc_auc_score(y_test,y_pred)
print(auc_roc)
from sklearn.metrics import roc_curve, auc
false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, y_prob)
roc_auc = auc(false_positive_rate, true_positive_rate)
print(roc_auc)
import matplotlib.pyplot as plt
plt.figure(figsize=(10,10))
plt.title('Receiver Operating Characteristic')
plt.plot(false_positive_rate,true_positive_rate, color='red',label = 'AUC = %0.2f'
% roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],linestyle='--')
plt.axis('tight')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
```

```
import pandas as pd
import numpy as np

#Charged libraries from Teresa
from feature_engine import outliers
from feature_engine.selection import DropDuplicateFeatures
from feature_engine.selection import DropCorrelatedFeatures
from feature_engine.selection import DropConstantFeatures
from feature_engine.encoding import DecisionTreeEncoder
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.linear_model import LogisticRegression

def clean_me(dataframe,
             detect_binary=True,
             numeric_dtype=True,
             decision_tree=True,
             one_hot=True,
             normalize=True,
             datetime_columns=[],
             remove_columns=[],
             high_corr_elimination=True,
             low_var_elimination=True,
             measuring_variable=[],
             variable_to_encode=[],
             outlier_removal=True,
             duplicated_var=True,
             duplicated_rows_remove=True,
             variables_uni=True,
             clean=True,
             verbose=True):


    """
    clean_me function performs automatic dataset cleaning to Pandas DataFrame as
per the settings parameters passed to this function

    :param dataframe: input Pandas DataFrame on which the cleaning will be performed
```

```
    :param detect_binray: if True, any column that has two unique values, these
values will be replaced with 0 and 1 (e.g.: ['looser', 'winner'] => [0,1])
    :param numeric_dtype: if True, columns will be converted to numeric dtypes when
possible **see [1] in README.md**
    :param decision_tree: if True, automatic detection of which variables will be
transformed by decision tree and coded
    :param one_hot: if True, all non-numeric columns will be encoded to one-hot
columns
    :param normalize: if True, all non-binray (columns with values 0 or 1 are
excluded) columns will be normalized and if True, automatic detection of which
variables should be normalized
    :param datetime_columns: a list of columns which contains date or time or
datetime entries (important to be announced in this list, otherwise hot-encoding
will mess up these columns)
    :param remove_columns: list of columns to remove, this is usually non-related
featues such as the ID column
    :param high_corr_elimination: eliminated high corr variables (Teresa)
    :param low_var_elimination: eliminates low variance variables (under 0.1)
(Teresa)
    :param measuring_variable: The y of the model (Teresa)
    :param variable_to_encode: if decision_tree is true, variables that want to be
changed by tree_decision
    :param outlier_removal: if outlier_removal is True, variables will be cleaned
of the outliers
    :param duplicated_var: if duplicated_var is True, the variables will be
eliminated
    :param duplicated_rows_remove: if duplicated_rows_remove is True,the rows that
are duplicated will be eliminated
    :param variables_uniques:if variables_uniques is True, the variables that are
unique in all rows will be deleted
    :param clean: if clean is True, the data will be cleaned of NAN Values
    :param verbose: print progress in terminal/cmd
    :return: processed and clean Pandas DataFrame.
    """

    df = dataframe.copy()



    if verbose:
        print(" +++++++++++++++ AUTO DATA CLEANING STARTED +++++++++++++++ ")
```

```
        print('You have to choose if you want a tree encoder or just a numerical
dtype (decision_tree=True)')
        print('You have to select the variable to eliminate such as name...
(var_elimination=[' ',' '])')
        print('You have to select the variable to measure (Y) in
(measuring_variable=[' ']) ')
        print('You have to choose which variables do you want to encode by decsion
tree (variable_to_encode=[])')
        print('Check if the columns of date and time have been changed, if not,
please introduce in datetime_columns=[''] the columns that need to be changed')


    # Validating user input to __init__ function
    assert type(one_hot) == type(True), "Please ensure that one_hot param is bool
(True/False)"
    # assert na_cleaner_mode in na_cleaner_modes, "Please choose proper value for
naCleaner parameter. (Correct values are only: {})".format(self.__naCleanerOps)
    assert type(df) == type(pd.DataFrame()), "Parameter 'df' should be Pandas
DataFrame"


    # Removing unwanted columns (usually its the ID columns...) ----------------
----------------------------------------
    if len(remove_columns) > 0:
        if verbose:
            print(" =  AutoDataCleaner: Performing removal of unwanted columns...
")
        df = remove_columns_df(df,remove_columns,verbose)


    # Droping duplites (Teresa) ------------------------------------------------
-------------------------------------------
    if duplicated_rows_remove:
        if verbose:
            print(" = AutoDataCleaner: Performing removal of duplicated rows... ")
        df=duplicated_rows(df)


    # Droping variables uniques  (Teresa) --------------------------------------
-----------------------------------------------------
    if variables_uni:
```

```
        if verbose:
            print(" = AutoDataCleaner: Performing recognition of unique variables
and droping them...")
        df=variables_uniques(df,verbose)


    # Clean None (na) values (Teresa) ------------------------------------------
-----------------------------------------------
    if clean:
        if verbose:
            print(" =  AutoDataCleaner: Performing None/NA/Empty values cleaning...
")
        df=clean_nan(df,verbose)


    # Detecting binary columns -------------------------------------------------
---------------------------------------
    if detect_binary:
        if verbose:
            print(" =  AutoDataCleaner: Performing One-Hot encoding... ")
        df = detect_binary_df(df, verbose)


    # Decision tree encoder + Automatization (Teresa (Delete the parameter dtype))-
-----------------------------------------------------
    if decision_tree:
        if len(variable_to_encode)>0:
            if verbose:
                print(" =   AutoDataCleaner:  Performing  dataset  decision  tree
encoder to the variable [{}] ".format(variable_to_encode))

df=decision_tree_encoder(df,variable_to_encode,measuring_variable,verbose,
datetime_columns)
        else:
            if verbose:
                print(" =   AutoDataCleaner:  Performing  dataset  decision  tree
encoder to the variables  ...")

df=decision_tree_encoder(df,variable_to_encode,measuring_variable,verbose,datetim
e_columns)
```

```
    if numeric_dtype:
        if verbose:
            print(" = AutoDataCleaner: Converting columns to numeric dtypes when
possible ...")
        df=convert_numeric_df(df, datetime_columns, verbose)
        df=clean_nan(df,verbose)


    # Eliminate outliers (Teresa) ------------------------------------------------
------------------------------------------------
    if outlier_removal:
        if verbose:
            print("=AutoDataCleaner: Performing Outliers removal... ")
    df=remove_outlier(df,verbose)


    # Casting datetime columns to datetime dtypes  ------------------------------
----------------------------------------
    if len(datetime_columns)>0:
        if verbose:
            print(" =   AutoDataCleaner: Casting datetime columns to datetime
dtype... ")
        df = cols_to_datetime_dtype(df,datetime_columns,verbose)


    # Converting non-numeric to one hot encoding --------------------------------
----------------------------------------
    if one_hot:
        if verbose:
            print(" =  AutoDataCleaner: Performing One-Hot encoding... ")
        cols_num_before =df.shape[1]
        df = one_hot_df(df)
        if verbose:
            print("        +    one-hot    encoding    done,    added    {}    new
columns".format(df.shape[1] - cols_num_before))


    # Eliminating columns with high corr (Teresa) -------------------------------
---------------------------
    if high_corr_elimination:
```

```
        if verbose:
            print(" =    AutoDataCleaner:  Performing  dataset  high  correlation
variables elimination...")
        df = drop_columns_HighCorr(df,verbose)


    # Eliminating Duplicated features  (Teresa) ------------------------------
-------------------------
    if duplicated_var:
        if verbose:
            print(" =   AutoDataCleaner:  Performing  dataset  droping  duplicated
features...")
        df=drop_duplicated_features(df,verbose)


    # Eliminating constant or nearly constant values (Teresa)------------------
---------------------------------------
    if low_var_elimination:
        if verbose:
            print(" =   AutoDataCleaner: Performing dataset droping constant or
nearly constant  features...")
        df=drop_columns_constant_or_cuasi_constant(df,verbose)


    # Normalize all columns (binary 0,1 columns are excluded) AUTOMATIC (Teresa) -
-----------------------------------------------------------
    if normalize:
        if verbose:
            print(" =  AutoDataCleaner: Performing dataset normalization... ")
        df                                                                    =
normalize_df(df,measuring_variable,exclude=[datetime_columns,measuring_variable],
verbose=True)


    if verbose:
        print(" +++++++++++++++ AUTO DATA CLEANING FINISHED +++++++++++++++ ")


    return df
```

```
""" ------------------------------------------------------------------------------
------------------------------------------ """

def datetime_dtype_series(series, verbose=True):
    """
    datetime_dtype_series function casts date columns to datetime dtype

    :param df: input Pandas Series
    :returns: processed Pandas Series
    """
    try:
        series = pd.to_datetime(series)
        if verbose:
            print("  + converted column {} to datetime dtype".format(series.name))
        return series
    except Exception as e:
        print(" ERROR {}".format(e))


def cols_to_datetime_dtype(df, cols, verbose=True):
    """
    cols_to_datetime_dtype function casts given columns in dataframe to datetime
dtype

    :param df: input Pandas DataFrame
    :returns: processed Pandas DataFrame
    """
    for c in cols:
        df[c] = datetime_dtype_series(df[c], verbose)
        if verbose:
            print(" +  The following columns have been change [{}]".format(cols))
    return df


def remove_columns_df(df, remove_columns, verbose=True):
    """
    remove_columns_df function removes columns in 'remove_columns' param list and
returns df

    :param df: input Pandas DataFrame
```

```
    :param remove_columns: list of columns to be removed from the dataframe
    :param verbose: print progress in terminal/cmd
    :returns: processed Pandas DataFrame
    """
    stat = 0
    for col in remove_columns:
        assert col in df.columns.to_list(), "{} is marked to be removed, but it
does not exist in the dataset/dataframe".format(col)

        df.drop(columns=col, inplace=True)
        stat += 1
    if verbose:
        print("  + removed {} columns successfully.".format(stat))
        print("  + removed the following columns: [{}] ".format(remove_columns))
    return df

def detect_binary_df(df, verbose=True):
    """
    detect_binray function detects columns that has two unique values (e.g.: yes/no
OR true/false etc...)
    and converts it to a boolean column containing 0 or 1 values only
    :param df: input Pandas DataFrame
    :param verbose: print progress in terminal/cmd
    :returns: processed Pandas DataFrame
    """
    stat_cols = 0
    stat_cols_names = []
    stat_rows = 0
    for col in df.columns.to_list():
        # check if column has two unique values
        if len(df[col].unique().tolist()) == 2:
            unique_values = df[col].unique().tolist()
            unique_values.sort()  # to ensure consistency during training and
predicting
            df[col] = df[col].replace(unique_values[0], 0)
            df[col] = df[col].replace(unique_values[1], 1)
            stat_cols += 1
            stat_cols_names.append(col)
            stat_rows += df.shape[0]
    if verbose:
```

```
        print("    + detected {} binary columns [{}], cells cleaned: {} cells".format(stat_cols, stat_cols_names, stat_rows))
    return df

def convert_numeric_series(series, force=False, verbose=True):
    """
    convert_numeric_series function converts columns of dataframe to numeric dtypes when possible safely
    if the values that cannot be converted to numeric dtype are minority in the series (< %25), then
    these minority values will be converted to NaN and the series will be forced to numeric dtype
    :param series: input Pandas Series
    :param force: if True, values which cannot be casted to numeric dtype will be replaced with NaN 'see pandas.to_numeric() docs' (be careful with force=True)
    :param verbose: print progress in terminal/cmd
    :returns: Pandas series
    """
    stats = 0
    if force:
        stats += series.shape[0]
        return pd.to_numeric(series, errors='coerce'), stats
    else:
        # checking if values that cannot be converted to numeric are < 25% of entries in this series
        non_numeric_count = pd.to_numeric(series, errors='coerce').isna().sum()
        if non_numeric_count/series.shape[0] < 0.25:
            # values that cannot be numeric are minority; hence, we set this as NaN and force that column to be
            # casted to numeric dtype, the 'clean_na_series' function will handle these NaN values laters
            stats += series.shape[0]
            if verbose and non_numeric_count != 0:
                print("   + {} minority (minority means < %25 of '{}' entries) values that cannot be converted to numeric dtype in column '{}' have been set to NaN, nan cleaner function will deal with them".format(non_numeric_count, series.name, series.name))
            return pd.to_numeric(series, errors='coerce'), stats
        else:
            # this series probably cannot be converted to numeric dtype, we will just leave it as is
```

```
            return series, stats


def convert_numeric_df(df, exclude=[], force=False, verbose=True):
    """
    convert_numeric_df function converts dataframe columns to numeric dtypes when
possible safely
    if the values in a particular columns that cannot be converted to numeric dtype
are minority in that column (< %25), then
    these minority values will be converted to NaN and the column will be forced
to numeric dtype
    :param df: input Pandas DataFrame
    :param exclude: list of columns to be excluded whice converting dataframe
columns to numeric dtype (usually datetime columns)
    :param force: if True, values which cannot be casted to numeric dtype will be
replaced with NaN 'see pandas.to_numeric() docs' (be careful with force=True)
    :param verbose: print progress in terminal/cmd
    :returns: Pandas DataFrame
    """
    stats = 0
    for col in df.columns.to_list():
        if col in exclude:
            continue
        df[col], stats_temp = convert_numeric_series(df[col], force, verbose)
    stats += stats_temp
    if verbose:
        print("  + converted {} cells to numeric dtypes".format(stats))
    return df


def one_hot_df(df):
    """
    one_hot_df returns one-hot encoding of non-numeric columns in all the columns
of the passed Pandas DataFrame

    :param df: input Pandas DataFrame
    :returns: Pandas DataFrame
    """
    return pd.get_dummies(df)

def normalize_df(data,mea_var, exclude=[], verbose=True):
```

```
    """
    normalize_df function performs normalization to all columns of dataframe
excluding binary (1/0) columns

    :param df: input Pandas DataFrame
    :param exclude: list of columns to be excluded when performing normalization
(usually datetime columns)
    :param verbose: print progress in terminal/cmd
    :returns: normalized Pandas DataFrame
    """
    stats = 0
    var=[]
    for col in data.columns.to_list():
        if col in exclude:
            continue
        # check if column is binray
        col_unique = data[col].unique().tolist()
        if len(col_unique) == 2 and 0 in col_unique and 1 in col_unique:
            continue
        else:
            columna=deciding_normalization_variables(data,col,mea_var)
            if columna:
                var.append(col)
            else:
                print('The variable [{}] will not be normalized because its
accuracy is too low'.format(col))
        for col in data.columns.to_list():
            if col in var:
                data[col] = (data[col]-data[col].mean())/data[col].std()
                stats += data.shape[0]
                columns=[data[col].name]
                if verbose:
                    print("  + normalized [{}]".format(columns))
                if verbose:
                    print("  + normalized {} cells".format(stats))
    return data



def help():
    help_text = """
```

```
    ++++++++++++++++++ AUTO DATA CLEANER HELP ++++++++++++++++++++++
    FUNCTION CALL:

AutoDataCleaner.clean_me(df,detect_binary=True,numeric_dtype=True,decision_tree=T
rue,
one_hot=True,normalize=True,datetime_columns=[],remove_columns=['Name'],high_corr
_elimination=True,low_var_elimination=True,measuring_variable='Survived',variable
_to_encode=[],outlier_removal=True,duplicated_var=True,duplicated_rows_remove=Tru
e,variables_uni=True,clean=True,verbose=True)

    FUNCTION PARAMETERS:
clean_me function performs automatic dataset cleaning to Pandas DataFrame as per
the settings parameters passed to this function
    :param dataframe: input Pandas DataFrame on which the cleaning will be performed
    :param detect_binray: if True, any column that has two unique values, these
values will be replaced with 0 and 1 (e.g.: ['looser', 'winner'] => [0,1])
    :param numeric_dtype: if True, columns will be converted to numeric dtypes when
possible **see [1] in README.md**
    :param decision_tree: if True, automatic detection of which variables will be
transformed by decision tree and coded
    :param one_hot: if True, all non-numeric columns will be encoded to one-hot
columns
    :param normalize: if True, all non-binray (columns with values 0 or 1 are
excluded) columns will be normalized and if True, automatic detection of which
variables should be normalized
    :param datetime_columns: a list of columns which contains date or time or
datetime entries (important to be announced in this list, otherwise hot-encoding
will mess up these columns)
    :param remove_columns: list of columns to remove, this is usually non-related
featues such as the ID column
    :param high_corr_elimination: eliminated high corr variables (Teresa)
    :param low_var_elimination: eliminates low variance variables (under 0.1)
(Teresa)
    :param measuring_variable: The y of the model (Teresa)
    :param variable_to_encode: if decision_tree is true, variables that want to be
changed by tree_decision
    :param outlier_removal: if outlier_removal is True, variables will be cleaned
of the outliers
    :param duplicated_var: if duplicated_var is True, the variables will be
eliminated
```

```
    :param duplicated_rows_remove: if duplicated_rows_remove is True,the rows that
are duplicated will be eliminated
    :param variables_uniques:if variables_uniques is True, the variables that are
unique in all rows will be deleted
    :param clean: if clean is True, the data will be cleaned of NAN Values
    :param verbose: print progress in terminal/cmd
    :return: processed and clean Pandas DataFrame.
    +++++++++++++++++ AUTO DATA CLEANER HELP ++++++++++++++++++++++
    """
    print(help_text)


# Creation of new funtions (Teresa)  ------------------------------------------
-------------------------------------------------------


def remove_outlier(df,verbose):
    capper = outliers.OutlierTrimmer(capping_method = "gaussian" ,
                                tail='both', fold=3 ,missing_values="ignore")
    df=capper.fit_transform(df)
    maximum=capper.right_tail_caps_
    minimum=capper.left_tail_caps_
    if verbose:
        print("+ The following [{}] shows dictionary with the maximum values above
which values will be removed ".format(maximum))
        print("+ The following [{}] shows dictionary with the minimum values below
which values will be removed ".format(minimum))
    return df


"""
#Elimination of nan values

Copyright (c) 2016 Randal S. Olson

Permission is hereby granted, free of charge, to any person obtaining a copy of
this software
and associated documentation files (the "Software"), to deal in the Software without
restriction,
including without limitation the rights to use, copy, modify, merge, publish,
distribute, sublicense,
```

```
and/or sell copies of the Software, and to permit persons to whom the Software is
furnished to do so,
subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial
portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
INCLUDING BUT NOT
LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
NONINFRINGEMENT.
IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES
OR OTHER LIABILITY,
WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
CONNECTION WITH THE
SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
"""


# Clean nan (removing the column if 60% is nan, removing 5% if the line is nan and
mode or mean depending on type of variables )


def clean_nan(df,verbose):
    #1. Drop the col if there are 60% of values missing
    if verbose:
        print('Inside the funtion for cleaning variable, we proceed to eliminate
the data with more than 60% of values beeing missing values')
    perc = 60.0
    min_count =  int(((100-perc)/100)*df.shape[0] + 1)
    null_percentage = df.isnull().sum()/df.shape[0]*100
    col_to_drop=null_percentage[null_percentage>perc].keys()
    df = df.dropna( axis=1,thresh=min_count)
    if verbose:
        print(" + the following columns '{}' had {}% or more of the values being
missing ".format(col_to_drop,perc))
    #2. Drop the line if there is more than 5% of values missing (95% is preserved)
    if verbose:
        print('Inside the funtion for cleaning variable, we proceed to eliminate
the row with more than 5% of values beeing missing values')
```

```
    perc=5.0
    df =df[df.isnull().sum(axis=1) < perc/100*len(df)]
    null_percentage = df.isnull().sum(axis=1)/df.shape[1]*100
    line_to_drop= null_percentage[null_percentage>perc].keys()
    if verbose:
        print(" + the following rows '{}' had 5% or more of the values being missing
".format(line_to_drop))
    #3. Replace NaNs with the median or mode of the column depending on the column
type
    if verbose:
        print('Inside the funtion for cleaning variable, we proceed to replace the
missing values for the median/mode depending on type of varibles')
    for column in df.columns.values:
    # Replace NaNs with the median or mode of the column depending on the column
type
        try:
            df[column].fillna(df[column].median(), inplace=True)
            median_col=df[column].name
            if verbose:
                print(" + The following variables '{}' have been fill by the
median".format(median_col))
        except TypeError:
            most_frequent = df[column].mode()
        # If the mode can't be computed, use the nearest valid value
        # See https://github.com/rhiever/datacleaner/issues/8
            if len(most_frequent) > 0:
                df[column].fillna(df[column].mode()[0], inplace=True)
                mode_col=df[column].name
                if verbose:
                    print(" + The following variables '{}' have been fill by the
mode".format(mode_col))

            else:
                df[column].fillna(method='bfill', inplace=True)
                df[column].fillna(method='ffill', inplace=True)
    return df



# Elimination of columns with high correlation over 0.8
```

```
def drop_columns_HighCorr(df,verbose):
    tr = DropCorrelatedFeatures(variables=None, method='pearson', threshold=0.8)
    df=tr.fit_transform(df)
    #Before running this code, you should run the code for DropCorrelatedFeatures
for it to understand self.correlated_features_sets
    correlete=tr.correlated_feature_sets_
    if verbose:
        print("  + droping [{}] cells".format(correlete))
    return df


# Elimination of columns with little variance (constant or cuasi constant 90%)


def drop_columns_constant_or_cuasi_constant(df,verbose):
    trans= DropConstantFeatures(tol=0.9)
    df=trans.fit_transform(df)
    const=trans.features_to_drop_
    if verbose:
        print("  +  droping [{}] cells".format(const))
    return df


# Elimination of duplicated features


def drop_duplicated_features(df,verbose):

    transformer = DropDuplicateFeatures()
    # fit and transform the data
    df=transformer.fit_transform(df)
    dup=transformer.duplicated_feature_sets_
    if verbose:
        print("  + droping [{}] cells".format(dup))
    return df


# Desiton tree encoder
```

```
def decision_tree_encoder(data,var,y,verbose,datetime_columns):
    # set up the encoder
    name=[]
    if len(var)>0:
        print("  + The following variables are beeing transforme by decision tree
encoder [{}]".format(var))
        encoder       =       DecisionTreeEncoder(variables=var,regression=False,
random_state=0)
        data=encoder.fit_transform(data,data[y])
    else:
        var=[]
        for col in data.columns:
            if (data[col].dtype==np.object):
                name.append(col)

    for col in name:
        columna=deciding_tree_variables(data,col,datetime_columns,y,verbose)
        if columna:
            if col in columna:
                var.append(col)
        else:
            print('The variable [{}] will not be converted to decision tree encoder
because its accuracy is too low'.format(col))
    if len(var)==len(name):
        encoder       =       DecisionTreeEncoder(variables=var,regression=False,
random_state=0)
        data=encoder.fit_transform(data,data[y])
        print("   + the variables that are beeing transformed by the automatic
decision tree encoder are [{}] ".format(var))
    else:
        data=convert_numeric_df(data,var, verbose=False)
        data= clean_nan(data,verbose=False)
        encoder       =       DecisionTreeEncoder(variables=var,regression=False,
random_state=0)
        data=encoder.fit_transform(data,data[y])
        print("  + the variables that are beeing transformed are [{}] ".format(var))
    return data


# dt_auto
```

```
def dt_inplace(df):
    """Automatically detect and convert (in place!) each
    dataframe column of datatype 'object' to a datetime just
    when ALL of its non-NaN values can be successfully parsed
    by pd.to_datetime().  Also returns a ref. to df for
    convenient use in an expression.
    """
    from pandas.errors import ParserError
    for c in df.columns[df.dtypes=='object']: #don't cnvt num
        try:
            df[c]=pd.to_datetime(df[c])
        except (ParserError,ValueError): #Can't cnvrt some
            pass # ...so leave whole column as-is unconverted
    return df
def read_csv(*args, **kwargs):
    """Drop-in replacement for Pandas pd.read_csv. It invokes
    pd.read_csv() (passing its arguments) and then auto-
    matically detects and converts each column whose datatype
    is 'object' to a datetime just when ALL of the column's
    non-NaN values can be successfully parsed by
    pd.to_datetime(), and returns the resulting dataframe.
    """
    return dt_inplace(pd.read_csv(*args, **kwargs))


def read_csv_transformado():
    text = """
    ++++++++++++++++++    AUTO    DATA    CLEANER    READ    CSV    TRANSFORMADO
++++++++++++++++++++++
    OPTION 1:
        IMPORT LIBRARY:
            import pandas as pd
        FUNCTION CALL:
            df=pd.read_csv('mydata.csv')

        DO NOT FORGET TO INCLUDE IN datetime_columns=[] , the variables date/time
that
        need to be changed.
    OPTION 2:
    IMPORT LIBRARY:
```

```
        from dt_auto import read_csv
    FUNCTION CALL:
        df=read_csv('mydata.csv')

    FUNCTION
    import pandas as pd
    def dt_inplace(df):
        #Automatically detect and convert (in place!) each
        #dataframe column of datatype 'object' to a datetime just
        #when ALL of its non-NaN values can be successfully parsed
        #by pd.to_datetime().  Also returns a ref. to df for
        #convenient use in an expression.

        from pandas.errors import ParserError
        for c in df.columns[df.dtypes=='object']: #don't cnvt num
            try:
                df[c]=pd.to_datetime(df[c])
            except (ParserError,ValueError): #Can't cnvrt some
                pass # ...so leave whole column as-is unconverted
        return df
    def read_csv(*args, **kwargs):
        #Drop-in replacement for Pandas pd.read_csv. It invokes
        #pd.read_csv() (passing its arguments) and then auto-
        #matically detects and converts each column whose datatype
        #is 'object' to a datetime just when ALL of the column's
        #non-NaN values can be successfully parsed by
        #pd.to_datetime(), and returns the resulting dataframe.

        return dt_inplace(pd.read_csv(*args, **kwargs))
    DO NOT FORGET TO REMOVE THE # FROM EVERY LINE
    ++++++++++++++++++ AUTO DATA CLEANER HELP ++++++++++++++++++++++
    """
    print(text)



def deciding_normalization_variables(X,cl,y):
    X[cl] = (X[cl]-X[cl].mean())/X[cl].std()
    X_train, X_test, y_train, y_test = train_test_split(X,X[y], random_state=42,
train_size = .8)
```

```
    clf = LogisticRegression()
    clf.fit(X_train, y_train)
    preds=clf.predict(X_test)
    score_preds=(metrics.accuracy_score(y_test,preds))*100
    if score_preds>=80:
        colum=cl
        print('The column [{}] is transformed which produces an accuracy in the
model of {} which is over 80%'.format(colum,score_preds))
        return colum

def variables_uniques(X,verbose):
    for col in X.columns:
        count=len(X[col].unique())
        if count==len(X[col].index):
            X.drop(columns=col,inplace=True)
            if verbose:
                print('Droping [{}] columsn because they contained {} unique values
'. format(col,count))
        else:
            print('Feature [{}] is not unique '.format(col))
            data=X
    return data



def deciding_tree_variables(X,cl,datetime_columns,y,verbose):
    encoder = DecisionTreeEncoder(variables=cl,regression=False, random_state=42)
    encoder.fit(X,X[y])
    X=encoder.transform(X)
    X=convert_numeric_df(X, datetime_columns,verbose)
    X = clean_nan(X,verbose=False)
    X_train, X_test, y_train, y_test = train_test_split(X,X[y], random_state=0,
train_size = .8)

    clf = LogisticRegression()
    clf.fit(X_train, y_train)
    preds=clf.predict(X_test)
    score_preds=(metrics.accuracy_score(y_test,preds))*100

    if score_preds>=80:
        colum=cl
```

```
        print('The column [{}] is transformed by decision tree encoder which
produces an accuracy in the model of {} which is over
80%'.format(colum,score_preds))
        return colum


def duplicated_rows(df):
    data=df.drop_duplicates()
    count=len(df)-len(data)
    print('Duplicated columns {}'.format(count))
    print(data)
    return data
```

CONSOLA TITANIC

```
----------------SHOWINGTHE DATA SET -----------------
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
None
 +++++++++++++++ AUTO DATA CLEANING STARTED +++++++++++++++++
You have to choose if you want a tree encoder or just a numerical dtype
(decision_tree=True)
You have to select the variable to eliminate such as name... (var_elimination=[,])
You have to select the variable to measure (Y) in (measuring_variable=[])
You have to choose which variables do you want to encode by decsion tree
(variable_to_encode=[])
Check if the columns of date and time have been changed, if not, please introduce
in datetime_columns=[] the columns that need to be changed
 =  AutoDataCleaner: Performing removal of unwanted columns...
  + removed 1 columns successfully.
  + removed the following columns: [['Name']]
 = AutoDataCleaner: Performing removal of duplicated rows...
Duplicated columns 0
     PassengerId  Survived  Pclass  ...     Fare  Cabin  Embarked
0              1         0       3  ...   7.2500    NaN         S
1              2         1       1  ...  71.2833    C85         C
2              3         1       3  ...   7.9250    NaN         S
```

```
3              4         1       1  ...  53.1000   C123         S
4              5         0       3  ...   8.0500    NaN         S
..           ...       ...     ...  ...      ...    ...       ...
886          887         0       2  ...  13.0000    NaN         S
887          888         1       1  ...  30.0000    B42         S
888          889         0       3  ...  23.4500    NaN         S
889          890         1       1  ...  30.0000   C148         C
890          891         0       3  ...   7.7500    NaN         Q

[891 rows x 11 columns]
 = AutoDataCleaner: Performing recognition of unique variables and droping them...
Droping [PassengerId] columsn because they contained 891 unique values
Feature [Survived] is not unique
Feature [Pclass] is not unique
Feature [Sex] is not unique
Feature [Age] is not unique
Feature [SibSp] is not unique
Feature [Parch] is not unique
Feature [Ticket] is not unique
Feature [Fare] is not unique
Feature [Cabin] is not unique
Feature [Embarked] is not unique
 =  AutoDataCleaner: Performing None/NA/Empty values cleaning...
Inside the funtion for cleaning variable, we proceed to eliminate the data with
more than 60% of values beeing missing values
 + the following columns 'Index(['Cabin'], dtype='object')' had 60.0% or more of
the values being missing
Inside the funtion for cleaning variable, we proceed to eliminate the row with more
than 5% of values beeing missing values
 + the following rows 'Int64Index([  5,  17,  19,  26,  28,  29,  31,  32,  36,
42,
            ...
            832, 837, 839, 846, 849, 859, 863, 868, 878, 888],
           dtype='int64', length=179)' had 5% or more of the values being missing
Inside the funtion for cleaning variable, we proceed to replace the missing values
for the median/mode depending on type of varibles
 + The following variables 'Survived' have been fill by the median
 + The following variables 'Pclass' have been fill by the median
 + The following variables 'Sex' have been fill by the mode
 + The following variables 'Age' have been fill by the median
 + The following variables 'SibSp' have been fill by the median
```

```
 + The following variables 'Parch' have been fill by the median
 + The following variables 'Ticket' have been fill by the mode
 + The following variables 'Fare' have been fill by the median
 + The following variables 'Embarked' have been fill by the mode
 =  AutoDataCleaner: Performing One-Hot encoding...
  + detected 2 binary columns [['Survived', 'Sex']], cells cleaned: 1782 cells
 =  AutoDataCleaner: Performing dataset decision tree encoder to the variables  ...
  + converted 891 cells to numeric dtypes
The column [Ticket] is transformed by decision tree encoder which produces an
accuracy in the model of 100.0 which is over 80%
  + converted 891 cells to numeric dtypes
The variable [Embarked] will not be converted to decision tree encoder because its
accuracy is too low
  + the variables that are beeing transformed are [['Ticket']]
 = AutoDataCleaner: Converting columns to numeric dtypes when possible ...
  + converted 891 cells to numeric dtypes
Inside the funtion for cleaning variable, we proceed to eliminate the data with
more than 60% of values beeing missing values
 + the following columns 'Index(['Embarked'], dtype='object')' had 60.0% or more
of the values being missing
Inside the funtion for cleaning variable, we proceed to eliminate the row with more
than 5% of values beeing missing values
 + the following rows 'Int64Index([], dtype='int64')' had 5% or more of the values
being missing
Inside the funtion for cleaning variable, we proceed to replace the missing values
for the median/mode depending on type of varibles
 + The following variables 'Survived' have been fill by the median
 + The following variables 'Pclass' have been fill by the median
 + The following variables 'Sex' have been fill by the median
 + The following variables 'Age' have been fill by the median
 + The following variables 'SibSp' have been fill by the median
 + The following variables 'Parch' have been fill by the median
 + The following variables 'Ticket' have been fill by the median
 + The following variables 'Fare' have been fill by the median
=AutoDataCleaner: Performing Outliers removal...
+  The  following  [{'Survived':  1.8436157466329595,  'Pclass':  4.816855698239796,
'Sex': 2.0815571936086097, 'Age': 68.42067214450208, 'SibSp': 3.8312381532214728,
'Parch':     2.799765378316916,     'Ticket':     1.6761273636457816,     'Fare':
181.2844937601173}] shows dictionary with the maximum values above which values
will be removed
```

```
+ The following [{'Survived': -1.0759389789561917, 'Pclass': -0.19957174762251206,
'Sex':    -0.7863832317679811,    'Age':    -9.697507161337093,    'SibSp':    -
2.7852224405390933, 'Parch': -2.0365779484628193, 'Ticket': -0.9084505959690259,
'Fare': -116.87607782296811}] shows dictionary with the minimum values below which
values will be removed
 =  AutoDataCleaner: Performing One-Hot encoding...
  + one-hot encoding done, added 0 new columns
 =  AutoDataCleaner: Performing dataset high correlation variables elimination...
  + droping [[{'Ticket', 'Survived'}]] cells
 =  AutoDataCleaner: Performing dataset droping duplicated features...
  + droping [[]] cells
 =   AutoDataCleaner: Performing dataset droping constant or nearly constant
features...
  +  droping [[]] cells
 =  AutoDataCleaner: Performing dataset normalization...
The column [Pclass] is transformed which produces an accuracy in the model of 100.0
which is over 80%
  + normalized [['Pclass']]
  + normalized 820 cells
The column [Age] is transformed which produces an accuracy in the model of 100.0
which is over 80%
  + normalized [['Pclass']]
  + normalized 1640 cells
  + normalized [['Age']]
  + normalized 2460 cells
The column [SibSp] is transformed which produces an accuracy in the model of 100.0
which is over 80%
  + normalized [['Pclass']]
  + normalized 3280 cells
  + normalized [['Age']]
  + normalized 4100 cells
  + normalized [['SibSp']]
  + normalized 4920 cells
The column [Parch] is transformed which produces an accuracy in the model of 100.0
which is over 80%
  + normalized [['Pclass']]
  + normalized 5740 cells
  + normalized [['Age']]
  + normalized 6560 cells
  + normalized [['SibSp']]
  + normalized 7380 cells
```

```
  + normalized [['Parch']]
  + normalized 8200 cells
The column [Fare] is transformed which produces an accuracy in the model of 100.0
which is over 80%
  + normalized [['Pclass']]
  + normalized 9020 cells
  + normalized [['Age']]
  + normalized 9840 cells
  + normalized [['SibSp']]
  + normalized 10660 cells
  + normalized [['Parch']]
  + normalized 11480 cells
  + normalized [['Fare']]
  + normalized 12300 cells
 +++++++++++++++ AUTO DATA CLEANING FINISHED +++++++++++++++
     Survived    Pclass  Sex       Age     SibSp     Parch      Fare
0           0  0.833486    1 -0.611553  1.051314 -0.450866 -0.623326
1           1 -1.585837    0  0.702781  1.051314 -0.450866  1.524179
2           1  0.833486    0 -0.282970 -0.563063 -0.450866 -0.600689
3           1 -1.585837    0  0.456344  1.051314 -0.450866  0.914360
4           0  0.833486    1  0.456344 -0.563063 -0.450866 -0.596496
..        ...       ...  ...       ...       ...       ...       ...
886         0 -0.376175    1 -0.200824 -0.563063 -0.450866 -0.430487
887         1 -1.585837    0 -0.857991 -0.563063 -0.450866  0.139648
888         0  0.833486    0 -0.118678  1.051314  3.087028 -0.080022
889         1 -1.585837    1 -0.282970 -0.563063 -0.450866  0.139648
890         0  0.833486    1  0.209906 -0.563063 -0.450866 -0.606558

[820 rows x 7 columns]
3.9180638790130615
[[429  71]
 [ 98 222]]
0.775875
0.852196875
```

TITANIC COMPROBACIÓN

SCRIPT TITANIC COMPROBACIÓN

```
import pandas as pd
import ADC_2 as adc
import numpy as np
from AutoDataCleaner.AutoDataCleaner import read_csv
from sklearn.model_selection import train_test_split
from datacleaner.datacleaner import autoclean
from sklearn import metrics
from sklearn.linear_model import LogisticRegression



# load data as pandas dataframe
df =read_csv('train_t.csv')
print(df.info())
data=adc.clean_me(df,
             detect_binary=True,
             numeric_dtype=True,
             one_hot=True,
             na_cleaner_mode="mode",
             normalize=True,
             datetime_columns=[],
             remove_columns=['Name','Ticket'],
             verbose=True)

y_train=data['Survived']
X_train=data.drop(columns=['Survived'])
df_test=read_csv('test_t.csv')
y_test=data['Survived']
X_test=data.drop(columns=['Survived'])

X=data
y=data['Survived']
model_LR= LogisticRegression()
model_LR.fit(X_train,y_train)
y_prob = model_LR.predict_proba(X_test)[:,1] # This will give you positive class
prediction probabilities
y_pred = np.where(y_prob > 0.5, 1, 0) # This will threshold the probabilities to
give class predictions.
```

```
model_LR.score(X_test, y_pred)
confusion_matrix_1=metrics.confusion_matrix(y_test,y_pred)
print(confusion_matrix_1)
auc_roc_1=metrics.roc_auc_score(y_test,y_pred)
print(auc_roc_1)
from sklearn.metrics import roc_curve, auc
false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, y_prob)
roc_auc_1 = auc(false_positive_rate, true_positive_rate)
print(roc_auc_1)
import matplotlib.pyplot as plt
plt.figure(figsize=(10,10))
plt.title('Receiver Operating Characteristic')
plt.plot(false_positive_rate,true_positive_rate, color='red',label = 'AUC = %0.2f'
% roc_auc_1)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],linestyle='--')
plt.axis('tight')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
```

```
import pandas as pd

na_cleaner_modes = ["remove row", "mean", "mode"]
def clean_me(dataframe,
             detect_binary=True,
             numeric_dtype=True,
             one_hot=True,
             na_cleaner_mode="mean",
             normalize=True,
             datetime_columns=[],
             remove_columns=[],
             verbose=True):
    """
    clean_me function performs automatic dataset cleaning to Pandas DataFrame as per the settings parameters passed to this function

    :param dataframe: input Pandas DataFrame on which the cleaning will be performed
    :param detect_binray: if True, any column that has two unique values, these values will be replaced with 0 and 1 (e.g.: ['looser', 'winner'] => [0,1])
    :param numeric_dtype: if True, columns will be converted to numeric dtypes when possible **see [1] in README.md**
    :param one_hot: if True, all non-numeric columns will be encoded to one-hot columns
    :param na_cleaner_mode: what technique to use when dealing with None/NA/Empty values. Modes:
        False       : do not consider cleaning na values
        'remove row': removes rows with a cell that has NA value
        'mean'      : substitues empty NA cells with the mean of that column
        'mode'      : substitues empty NA cells with the mode of that column
        '*'             : substiture empty NA cells with the value passed in 'na_cleaner_mode' param
    :param normalize: if True, all non-binray (columns with values 0 or 1 are excluded) columns will be normalized.
    :param datetime_columns: a list of columns which contains date or time or datetime entries (important to be announced in this list, otherwise hot-encoding will mess up these columns)
    :param remove_columns: list of columns to remove, this is usually non-related featues such as the ID column
    :param verbose: print progress in terminal/cmd
```

```
    :return: processed and clean Pandas DataFrame.
    """
    df = dataframe.copy()

    if verbose:
        print(" +++++++++++++++ AUTO DATA CLEANING STARTED +++++++++++++++ ")

    # Validating user input to __init__ function
    assert type(one_hot) == type(True), "Please ensure that one_hot param is bool
(True/False)"
    # assert na_cleaner_mode in na_cleaner_modes, "Please choose proper value for
naCleaner parameter. (Correct values are only: {})".format(self.__naCleanerOps)
    assert type(df) == type(pd.DataFrame()), "Parameter 'df' should be Pandas
DataFrame"



    # casting datetime columns to datetime dtypes ------------------------------
----------------------------------------
    if len(datetime_columns) > 0:
        if verbose:
            print(" =   AutoDataCleaner: Casting datetime columns to datetime
dtype... ")
        df = cols_to_datetime_dtype(df, datetime_columns, verbose)

    # removing unwanted columns (usually it's the ID columns...) ---------------
----------------------------------------
    if len(remove_columns) > 0:
        if verbose:
            print(" =  AutoDataCleaner: Performing removal of unwanted columns...
")
        df = remove_columns_df(df, remove_columns, verbose)



    # detecting binary columns -------------------------------------------------
----------------------------------------
    if detect_binary:
        if verbose:
            print(" =  AutoDataCleaner: Performing One-Hot encoding... ")
        df = detect_binary_df(df, verbose)
```

```
    # checking if any columns can be converted to numeric dtypes  ---------------
-----------------------------------------
    if numeric_dtype:
        if verbose:
            print(" = AutoDataCleaner: Converting columns to numeric dtypes when
possible...")
        df   =   convert_numeric_df(df,   exclude=datetime_columns,   force=False,
verbose=verbose)



    # converting non-numeric to one hot encoding -------------------------------
-----------------------------------------
    if one_hot:
        if verbose:
            print(" =  AutoDataCleaner: Performing One-Hot encoding... ")
        cols_num_before = df.shape[1]
        df = one_hot_df(df)
        if verbose:
            print("        +    one-hot    encoding    done,    added    {}    new
columns".format(df.shape[1] - cols_num_before))



    # clean None (na) values ---------------------------------------------------
-----------------------------------------
    if na_cleaner_mode != False:
        if verbose:
            print(" =  AutoDataCleaner: Performing None/NA/Empty values cleaning...
")
        df = clean_na_df(df, na_cleaner_mode, verbose)




    if verbose:
        print(" +++++++++++++++ AUTO DATA CLEANING FINISHED +++++++++++++++ ")
    return df
```

```
""" ----------------------------------------------------------------------------
------------------------------------------- """

def datetime_dtype_series(series, verbose=True):
    """
    datetime_dtype_series function casts date columns to datetime dtype

    :param df: input Pandas Series
    :returns: processed Pandas Series
    """
    try:
        series = pd.to_datetime(series)
        if verbose:
            print("  + converted column {} to datetime dtype".format(series.name))
        return series
    except Exception as e:
        print(" ERROR {}".format(e))


def cols_to_datetime_dtype(df, cols, verbose=True):
    """
    cols_to_datetime_dtype function casts given columns in dataframe to datetime
dtype

    :param df: input Pandas DataFrame
    :returns: processed Pandas DataFrame
    """
    for c in cols:
        df[c] = datetime_dtype_series(df[c], verbose)
    return df


def remove_columns_df(df, remove_columns, verbose=True):
    """
    remove_columns_df function removes columns in 'remove_columns' param list and
returns df

    :param df: input Pandas DataFrame
```

```
    :param remove_columns: list of columns to be removed from the dataframe
    :param verbose: print progress in terminal/cmd
    :returns: processed Pandas DataFrame
    """
    stat = 0
    for col in remove_columns:
        assert col in df.columns.to_list(), "{} is marked to be removed, but it
does not exist in the dataset/dataframe".format(col)

        df.drop(columns=col, inplace=True)
        stat += 1
    if verbose:
        print("  + removed {} columns successfully.".format(stat))
    return df

def detect_binary_df(df, verbose=True):
    """
    detect_binray function detects columns that has two unique values (e.g.: yes/no
OR true/false etc...)
    and converts it to a boolean column containing 0 or 1 values only

    :param df: input Pandas DataFrame
    :param verbose: print progress in terminal/cmd
    :returns: processed Pandas DataFrame
    """
    stat_cols = 0
    stat_cols_names = []
    stat_rows = 0
    for col in df.columns.to_list():
        # check if column has two unique values
        if len(df[col].unique().tolist()) == 2:
            unique_values = df[col].unique().tolist()
            unique_values.sort()  # to ensure consistency during training and
predicting
            df[col] = df[col].replace(unique_values[0], 0)
            df[col] = df[col].replace(unique_values[1], 1)
            stat_cols += 1
            stat_cols_names.append(col)
            stat_rows += df.shape[0]
    if verbose:
```

```
        print("    + detected {} binary columns [{}], cells cleaned: {} cells".format(stat_cols, stat_cols_names, stat_rows))
    return df

def convert_numeric_series(series, force=False, verbose=True):
    """
    convert_numeric_series function converts columns of dataframe to numeric dtypes when possible safely
    if the values that cannot be converted to numeric dtype are minority in the series (< %25), then
    these minority values will be converted to NaN and the series will be forced to numeric dtype

    :param series: input Pandas Series
    :param force: if True, values which cannot be casted to numeric dtype will be replaced with NaN 'see pandas.to_numeric() docs' (be careful with force=True)
    :param verbose: print progress in terminal/cmd
    :returns: Pandas series
    """
    stats = 0
    if force:
        stats += series.shape[0]
        return pd.to_numeric(series, errors='coerce'), stats
    else:
        # checking if values that cannot be converted to numeric are < 25% of entries in this series
        non_numeric_count = pd.to_numeric(series, errors='coerce').isna().sum()
        if non_numeric_count/series.shape[0] < 0.25:
            # values that cannot be numeric are minority; hence, we set this as NaN and force that column to be
            # casted to numeric dtype, the 'clean_na_series' function will handle these NaN values laters
            stats += series.shape[0]
            if verbose and non_numeric_count != 0:
                print("   + {} minority (minority means < %25 of '{}' entries) values that cannot be converted to numeric dtype in column '{}' have been set to NaN, nan cleaner function will deal with them".format(non_numeric_count, series.name, series.name))
            return pd.to_numeric(series, errors='coerce'), stats
        else:
```

```
            # this series probably cannot be converted to numeric dtype, we will
just leave it as is
            return series, stats


def convert_numeric_df(df, exclude=[], force=False, verbose=True):
    """
    convert_numeric_df function converts dataframe columns to numeric dtypes when
possible safely
    if the values in a particular columns that cannot be converted to numeric dtype
are minority in that column (< %25), then
    these minority values will be converted to NaN and the column will be forced
to numeric dtype

    :param df: input Pandas DataFrame
    :param exclude: list of columns to be excluded whice converting dataframe
columns to numeric dtype (usually datetime columns)
    :param force: if True, values which cannot be casted to numeric dtype will be
replaced with NaN 'see pandas.to_numeric() docs' (be careful with force=True)
    :param verbose: print progress in terminal/cmd
    :returns: Pandas DataFrame
    """
    stats = 0
    for col in df.columns.to_list():
        if col in exclude:
            continue
        df[col], stats_temp = convert_numeric_series(df[col], force, verbose)
    stats += stats_temp
    if verbose:
        print("  + converted {} cells to numeric dtypes".format(stats))
    return df


def one_hot_df(df):
    """
    one_hot_df returns one-hot encoding of non-numeric columns in all the columns
of the passed Pandas DataFrame

    :param df: input Pandas DataFrame
    :returns: Pandas DataFrame
    """
```

```
    return pd.get_dummies(df)

def clean_na_series(series, na_cleaner_mode):
    """
    clean_nones function manipulates None/NA values in a given panda series
according to cleaner_mode parameter

    :param series: the Panda Series in which the cleaning will be performed
    :param na_cleaner_mode: what cleaning technique to apply, 'na_cleaner_modes'
for a list of all possibilities
    :returns: cleaned version of the passed Series
    """
    if na_cleaner_mode == 'remove row':
        return series.dropna()
    elif na_cleaner_mode == 'mean':
        mean = series.mean()
        return series.fillna(mean)
    elif na_cleaner_mode == 'mode':
        mode = series.mode()[0]
        return series.fillna(mode)
    elif na_cleaner_mode == False:
        return series
    else:
        return series.fillna(na_cleaner_mode)


def clean_na_df(df, na_cleaner_mode, verbose=True):
    """
    clean_na_df function cleans all columns in DataFrame as per given
na_cleaner_mode

    :param df: input DataFrame
    :param na_cleaner_mode: what technique to apply to clean na values
    :param verbose: print progress in terminal/cmd
    :returns: cleaned Pandas DataFrame
    """
    stats = {}
    for col in df.columns.to_list():
        if df[col].isna().sum() > 0:
            stats[col] = df[col].isna().sum()
            try:
```

```
                df[col] = clean_na_series(df[col], na_cleaner_mode)
            except:
                pass
            print("  + could not find mean for column {}, will use mode instead to
fill NaN values".format(col))
            df[col] = clean_na_series(df[col], 'mode')
    if verbose:
        print("  + cleaned the following NaN values: {}".format(stats))
    return df


def normalize_df(df, exclude=[], verbose=True):
    """
    normalize_df function performs normalization to all columns of dataframe
excluding binary (1/0) columns

    :param df: input Pandas DataFrame
    :param exclude: list of columns to be excluded when performing normalization
(usually datetime columns)
    :param verbose: print progress in terminal/cmd
    :returns: normalized Pandas DataFrame
    """
    stats = 0
    for col in df.columns.to_list():
        if col in exclude:
            continue
        # check if column is binray
        col_unique = df[col].unique().tolist()
        if len(col_unique) == 2 and 0 in col_unique and 1 in col_unique:
            continue
        else:
            df[col] = (df[col]-df[col].mean())/df[col].std()
            stats += df.shape[0]
    if verbose:
        print("  + normalized {} cells".format(stats))
    return df



def help():
    help_text = """
```

```
    ++++++++++++++++++ AUTO DATA CLEANER HELP ++++++++++++++++++++++
    FUNCTION CALL:
    AutoDataCleaner.clean_me(df,       one_hot=True,       na_cleaner_mode="mean",
normalize=True, remove_columns=[], verbose=True)

    FUNCTION PARAMETERS:
    df: input Pandas DataFrame on which the cleaning will be performed
    one_hot: if True, all non-numeric columns will be encoded to one-hot columns
    na_cleaner_mode: what technique to use when dealing with None/NA/Empty values.
Modes:
        False: do not consider cleaning na values
        'remove row': removes rows with a cell that has NA value
        'mean': substitues empty NA cells with the mean of that column
        'mode': substitues empty NA cells with the mode of that column
        '*': any other value will substitute empty NA cells with that particular
value passed here
    normalize: if True, all non-binray (columns with values 0 or 1 are excluded)
columns will be normalized.
    remove_columns: list of columns to remove, this is usually non-related featues
such as the ID column
    verbose: print progress in terminal/cmd
    returns: processed and clean Pandas DataFrame
    ++++++++++++++++++ AUTO DATA CLEANER HELP ++++++++++++++++++++++
    """
    print(help_text)
```

CONSOLA TITANIC COMPROBACIÓN

```
runfile('C:/Users/Usuario/.spyder-py3/TrabajoFinDeGrado/prueba_1.py',
wdir='C:/Users/Usuario/.spyder-py3/TrabajoFinDeGrado')
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
None
 +++++++++++++++ AUTO DATA CLEANING STARTED ++++++++++++++++
 =  AutoDataCleaner: Performing removal of unwanted columns...
  + removed 1 columns successfully.
 =  AutoDataCleaner: Performing One-Hot encoding...
  + detected 2 binary columns [['Survived', 'Sex']], cells cleaned: 1782 cells
 = AutoDataCleaner: Converting columns to numeric dtypes when possible...
  + 177 minority (minority means < %25 of 'Age' entries) values that cannot be
converted to numeric dtype in column 'Age' have been set to NaN, nan cleaner
function will deal with them
  + converted 0 cells to numeric dtypes
 =  AutoDataCleaner: Performing One-Hot encoding...
  + one-hot encoding done, added 828 new columns
 =  AutoDataCleaner: Performing None/NA/Empty values cleaning...
  + could not find mean for column Age, will use mode instead to fill NaN values
  + cleaned the following NaN values: {'Age': 177}
 +++++++++++++++ AUTO DATA CLEANING FINISHED ++++++++++++++++
[[478  71]
```

```
 [108 234]]
0.7774422394784776
0.8571672045931464
```

ANEXO II. LICENCIAS

LICENCIA PARA DATACLEANER DE RANDAL S. OLSON

"""

Copyright (c) 2016 Randal S. Olson

Permission is hereby granted, free of charge, to any person obtaining a copy of this software

and associated documentation files (the "Software"), to deal in the Software without restriction,

including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense,

and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so,

subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial

portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT

LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.

IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,

WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE

SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

"""

TITANIC DATASET

The original Titanic dataset, describing the survival status of individual passengers on the Titanic. The titanic data does not contain information from the crew, but it does contain actual ages of half of the passengers. The principal source for data about Titanic passengers is the Encyclopedia Titanica. The datasets used here were begun by a variety of researchers. One of the original sources is Eaton & Haas (1994) Titanic: Triumph and Tragedy, Patrick Stephens Ltd, which includes a passenger list created by many researchers and edited by Michael A. Findlay.

Thomas Cason of UVa has greatly updated and improved the titanic data frame using the Encyclopedia Titanica and created the dataset here. Some duplicate passengers have been dropped, many errors corrected, many missing ages filled in, and new variables created.

For more information about how this dataset was constructed: http://biostat.mc.vanderbilt.edu/wiki/pub/Main/DataSets/titanic3info.txt [29]

ANEXO III. CONJUNTOS DE DATOS

TITANIC DATASET

ANTES AUTODATACLEANER

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

DESPUES AUTODATACLEANER

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 820 entries, 0 to 890
Data columns (total 8 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Survived     820 non-null    int64
 2   Pclass       820 non-null    float64
 3   Sex          820 non-null    int64
 4   Age          820 non-null    float64
 5   SibSp        820 non-null    float64
 6   Parch        820 non-null    float64
 7   Fare         820 non-null    float64
dtypes: float64(5), int64(2)
memory usage: 51.2 KB
```

CREDIT APPROVAL DATASET

ANTES AUTODATACLEANER

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 462 entries, 0 to 461
Data columns (total 18 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Unnamed: 0  462 non-null    int64
 1   A1          454 non-null    object
 2   A2          457 non-null    float64
 3   A3          462 non-null    float64
 4   A4          456 non-null    object
 5   A5          456 non-null    object
 6   A6          455 non-null    object
 7   A7          455 non-null    object
 8   A8          462 non-null    float64
 9   A9          462 non-null    object
 10  A10         462 non-null    object
 11  A11         462 non-null    int64
 12  A12         462 non-null    object
 13  A13         462 non-null    object
 14  A14         450 non-null    float64
 15  A15         462 non-null    int64
 16  A16         462 non-null    object
 17  A16.1       462 non-null    object
dtypes: float64(4), int64(3), object(11)
memory usage: 65.1+ KB
```

DESPUES AUTODATACLEANER

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 391 entries, 0 to 461
Data columns (total 15 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Unnamed: 0  391 non-null    float64
 1   A1          391 non-null    int64
 2   A2          391 non-null    float64
 3   A3          391 non-null    float64
 4   A4          391 non-null    float64
 5   A6          391 non-null    float64
 6   A7          391 non-null    float64
 7   A8          391 non-null    float64
 8   A9          391 non-null    int64
 9   A10         391 non-null    int64
 10  A11         391 non-null    float64
 11  A12         391 non-null    int64
 12  A14         391 non-null    float64
 13  A15         391 non-null    float64
 14  A16         391 non-null    int64
dtypes: float64(10), int64(5)
memory usage: 48.9 KB
```