AUTORIZACIÓN PARA LA DIGITALIZACIÓN, DEPÓSITO Y DIVULGACIÓN EN RED DE PROYECTOS FIN DE GRADO, FIN DE MÁSTER, TESINAS O MEMORIAS DE BACHILLERATO

1º. Declaración de la autoría y acreditación de la misma.

El autor D. Eugenio Ramírez Moredero

DECLARA ser el titular de los derechos de propiedad intelectual de la obra: Predicción de la volatilidad con redes neuronales, que ésta es una obra original, y que ostenta la condición de autor en el sentido que otorga la Ley de Propiedad Intelectual.

2º. Objeto y fines de la cesión.

Con el fin de dar la máxima difusión a la obra citada a través del Repositorio institucional de la Universidad, el autor **CEDE** a la Universidad Pontificia Comillas, de forma gratuita y no exclusiva, por el máximo plazo legal y con ámbito universal, los derechos de digitalización, de archivo, de reproducción, de distribución y de comunicación pública, incluido el derecho de puesta a disposición electrónica, tal y como se describen en la Ley de Propiedad Intelectual. El derecho de transformación se cede a los únicos efectos de lo dispuesto en la letra a) del apartado siguiente.

3º. Condiciones de la cesión y acceso

Sin perjuicio de la titularidad de la obra, que sigue correspondiendo a su autor, la cesión de derechos contemplada en esta licencia habilita para:

a) Transformarla con el fin de adaptarla a cualquier tecnología que permita incorporarla a internet y hacerla accesible; incorporar metadatos para realizar el registro de la obra e incorporar "marcas de agua" o cualquier otro sistema de seguridad o de protección.
b) Reproducirla en un soporte digital para su incorporación a una base de datos electrónica, incluyendo el derecho de reproducir y almacenar la obra en servidores, a los efectos de garantizar su seguridad, conservación y preservar el formato.
c) Comunicarla, por defecto, a través de un archivo institucional abierto, accesible de modo libre y gratuito a través de internet.
d) Cualquier otra forma de acceso (restringido, embargado, cerrado) deberá solicitarse expresamente y obedecer a causas justificadas.
e) Asignar por defecto a estos trabajos una licencia Creative Commons.
f) Asignar por defecto a estos trabajos un HANDLE (URL *persistente)*.

4º. Derechos del autor.

El autor, en tanto que titular de una obra tiene derecho a:

a) Que la Universidad identifique claramente su nombre como autor de la misma
b) Comunicar y dar publicidad a la obra en la versión que ceda y en otras posteriores a través de cualquier medio.
c) Solicitar la retirada de la obra del repositorio por causa justificada.
d) Recibir notificación fehaciente de cualquier reclamación que puedan formular terceras personas en relación con la obra y, en particular, de reclamaciones relativas a los derechos de propiedad intelectual sobre ella.

5º. Deberes del autor.

El autor se compromete a:

a) Garantizar que el compromiso que adquiere mediante el presente escrito no infringe ningún derecho de terceros, ya sean de propiedad industrial, intelectual o cualquier otro.
b) Garantizar que el contenido de las obras no atenta contra los derechos al honor, a la intimidad y a la imagen de terceros.
c) Asumir toda reclamación o responsabilidad, incluyendo las indemnizaciones por daños, que pudieran ejercitarse contra la Universidad por terceros que vieran infringidos sus derechos e

intereses a causa de la cesión.

d) Asumir la responsabilidad en el caso de que las instituciones fueran condenadas por infracción de derechos derivada de las obras objeto de la cesión.

6ª. Fines y funcionamiento del Repositorio Institucional.

La obra se pondrá a disposición de los usuarios para que hagan de ella un uso justo y respetuoso con los derechos del autor, según lo permitido por la legislación aplicable, y con fines de estudio, investigación, o cualquier otro fin lícito. Con dicha finalidad, la Universidad asume los siguientes deberes y se reserva las siguientes facultades:

- La Universidad informará a los usuarios del archivo sobre los usos permitidos, y no garantiza ni asume responsabilidad alguna por otras formas en que los usuarios hagan un uso posterior de las obras no conforme con la legislación vigente. El uso posterior, más allá de la copia privada, requerirá que se cite la fuente y se reconozca la autoría, que no se obtenga beneficio comercial, y que no se realicen obras derivadas.
- La Universidad no revisará el contenido de las obras, que en todo caso permanecerá bajo la responsabilidad exclusive del autor y no estará obligada a ejercitar acciones legales en nombre del autor en el supuesto de infracciones a derechos de propiedad intelectual derivados del depósito y archivo de las obras. El autor renuncia a cualquier reclamación frente a la Universidad por las formas no ajustadas a la legislación vigente en que los usuarios hagan uso de las obras.
- La Universidad adoptará las medidas necesarias para la preservación de la obra en un futuro.
- La Universidad se reserva la facultad de retirar la obra, previa notificación al autor, en supuestos suficientemente justificados, o en caso de reclamaciones de terceros.

Madrid, a 3 de Julio de 2022

ACEPTA

Fdo.....

Motivos para solicitar el acceso restringido, cerrado o embargado del trabajo en el Repositorio Institucional:

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

Predicción de la volatilidad con redes neuronales

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.: (Nombre del alumno) Eugenio Francés Fecha: 03/07/2022

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Guillermo Gomez

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

COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI

GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

Predicción de la volatilidad con redes neuronales

Autor: Eugenio Francés Monedero
Director: Guillermo Gómez Larriba

Madrid

COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI

GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

Predicción de la volatilidad con redes neuronales

Autor: Eugenio Francés Monedero
Director: Guillermo Gómez Larriba

Madrid

Agradecimientos

A mis padres por todo el sacrificio que ha supuesto buscar en todo momento la mejor alternativa.

A mis profesores tanto en España, Suecia y otras universidades porqué han creído en la originalidad de este trabajo, especialmente a Guillermo que me ha tutorizado el TFG, y ha sabido darle una proyección como inicio de un gran proyecto.

Resumen del proyecto:

Palabras clave: Redes neuronales, predicción, Keras, LSTM, iron condor, volatilidad, retro propagación.

Introducción

La predicción con tecnología de IA cobra cada día más importancia en la evolución tecnológica, tanto socialmente (predicción de enfermedades en el campo de la medicina) como desde el punto de vista económico (Fintech, automatización de ventas, optimización de procesos…).

Nosotros nos vamos a centrar en el campo de la predicción de los mercados de valores con machine learning. El área en el que vamos a trabajar dentro del sector financiero son los productos derivados, ya que su valor depende tanto de la evolución del subyacente como de la especulación de los propios derivados en sí.

Nuestro proyecto combina la vanguardia de la inteligencia artificial, el deep learning (redes neuronales recurrentes) con la ingeniería financiera más sofisticada como son los spreads de opciones no estándar.

Definición del proyecto:

El proyecto que desarrollamos en este TFG, trata de diseñar un sistema de predicción, utilizando la tecnología de machine learning con una arquitectura RNR (red neuronal recurrente).

Inicialmente, el sistema se alimentaba con datos automáticos de los mercados de opciones como CBOE de Chicago y otras bases de datos de pago, aunque para comprobar el modelo de predicción, modificamos el planteamiento inicial para coger los datos directamente de archivos Excel leídos en formato CSV.

Como las opciones son derivados sobre un subyacente el sistema ha tenido que considerar el tratamiento de los datos como conceptos que a veces utilizan derivadas, a veces términos relativos y en otras ocasiones números individuales. A esto se une la complejidad de las estructuras de combinación de las opciones, donde además hemos intentado a partir de estrategias estándar realizar prototipos modificados según las variables encontradas, los diferentes escenarios de las RNR (redes neuronales recurrentes) y la retroalimentación del sistema con los parámetros de la predicción.

Descripción del modelo:

El modelo intenta predecir el valor de la volatilidad del índice SPX como variable fundamental de las opciones para la construcción de spreads no estándar, modificados en diferentes escenarios según los patrones resultantes de la RNR.

La red neuronal recurrente se puede entrenar usando el algoritmo “descenso de gradiente” y así poder optimizar de forma supervisada, porqué en la capa de salida de la red, cambian los pesos en función a la derivada del error en relación con el peso correspondiente. La “retro propagación” a través del tiempo es un concepto que se combina con la optimización del algoritmo “descenso de gradiente” en un conjunto de secuencias de entrenamiento.

Resultados:

Con nuestro modelo de predicción hemos logrado los objetivos que pretendíamos, y que podemos sintetizar en:

Hacer predicciones fiables de la volatilidad con un error acotable lo suficientemente manejable para lograr que esta sea operativa en nuestro modelo.

Con dicha variable poder estimar el valor del subyacente, en nuestro caso el SPX y poder controlar las tendencias de las opciones. Nuestra RNR (red neuronal recurrente) tiene la capacidad de identificar patrones o escenarios de mercado porque identifica series de números en vez de valores individuales frente a las otras RNA o redes neuronales tradicionales. Además, estas redes avanzadas manejan la variable tiempo de la serie identificada, de adelante hacia

atrás, pudiendo comparar esos escenarios actuales de los mercados financieros con escenarios pasados, lo que las dota de una metodología de predicción absolutamente óptima para el objeto que nos ocupa.

Finalmente, estas tendencias servirán para construir spread o estrategias no estándar que, aunque sean estrategias delta neutral recojan la asimetría estadística de la tendencia. Materializado en diferente número de contratos para cada vertical. Por último, la arquitectura de RNR tiene propiedades para la detección de anomalías y detección de atípicos que nos ayudará a predecir cambios de tendencias.

El resultado de todo ello ha sido el diseño de un spread no estándar como es un iron condor asimétrico que ha sido testado y obtiene rentabilidades de entre un 2,5 a un 10% mensual debido a que hemos podido acortar el DTE (tiempo de expiración) óptimo de 3 meses a 1 mes. Como utilidad adicional, este spread podría usarse como protección de delta (delta neutral o 0) para grandes carteras de inversión profesionales.

Conclusiones:

Nosotros buscábamos con este proyecto de predicción encontrar relaciones que no se pudieran detectar a simple vista para obtener una ventaja en la inversión profesional. Las RNR tienen unas características que se adecúan perfectamente a nuestro objetivo, porque clasifican y predicen secuencias mediante series de tiempo como criterio de clasificación. Estos patrones pueden estar antes o después en el tiempo, al contrario que las series temporales y mejorando por tanto a las redes neuronales artificiales tradicionales.

Utilizando redes según la arquitectura LSTM de RNR (redes neuronales recurrentes) no procesamos datos individuales sino secuencias de datos como patrones de precios en momentos de tiempo diferentes, esencial para el estudio que nos ocupa, donde los precios siguen patrones parecidos en distintos momentos de tiempo.

El tema de tener un conjunto de variables fundamentales que se relacionaban indirectamente con la principal también nos llevó adicionalmente a obtener resultados de las tendencias, donde

los sesgos estadísticos añadían una información rentable a la hora del diseño de los spreads de opciones.
Por último, la arquitectura de RNR tiene propiedades para la detección de anomalías y detección de atípicos que nos ayuda a predecir cambios de tendencias.

Como se ha dicho en el apartado anterior, hemos logrado predecir la volatilidad a través del VIX, y pudiendo predecir el SPX y las tendencias, hemos logrado diseñar con el programa thinkorswim un spread no estándar (iron condor asimétrico), testeado con ONE (option net explorer) con las rentabilidades mencionadas gracias al acortamiento de los DTE por la predicción.

Objetivo y Motivación:

La motivación por la cual escogí este trabajo fue hacer un proyecto en el que tuviese la posibilidad de conjugar los conocimientos de ingeniería con los financieros.

Para ello planteé este TFG junto con mi coordinador de prácticas con el objetivo de profundizar en el machine learning y ser capaz de emplear esas competencias para predecir valores y desarrollar una estrategia de opciones.

Project summary:

Key words: Keras, LSTM, prediction, asimetric iron condor, RNR, volatility and retropropagation.

Introduction:

Prediction with AI technology is becoming increasingly important in technological evolution, both socially (prediction of diseases in the medical field) and from an economic point of view (Fintech, sales automation, process optimisation...).

We are going to focus on the field of stock market prediction with machine learning. The area in which we are going to work within the financial sector is derivatives, as their value depends both on the evolution of the underlying and on the speculation of the derivatives themselves.

Our project combines the cutting edge of artificial intelligence, deep learning (recurrent neural networks) with the most sophisticated financial engineering such as non-standard option spreads.

Project definition:

The project that we develop in this TFG, tries to design a prediction system, using machine learning technology with a RNR (recurrent neural network) architecture.

Initially, the system was fed with automatic data from options markets such as the Chicago CBOE and other payment databases, although in order to test the prediction model, we modified the initial approach to take data directly from Excel files read in CSV format.

As options are derivatives on an underlying the system has had to consider the treatment of the data as concepts that sometimes use derivatives, sometimes relative terms and sometimes individual numbers. To this is added the complexity of the combination structures of the options where we have also tried to use standard strategies to create modified prototypes; according to the variables found, the different scenarios that the RNR (recurrent neural networks) defined for us and the feedback of the system with the prediction parameters

Description of the model:

The model tries to predict the volatility value of the SPX index as a fundamental variable of the options for the construction of non-standard spreads modified in different scenarios according to the resulting patterns of the RNR. The recurrent neural network can be trained using the "gradient descent" algorithm and thus be optimised in a supervised manner, because in the output layer of the network, it changes the weights as a function of the derivative of the error relative to the corresponding weight. Back propagation" over time is a concept that is combined with the optimisation of the gradient descent algorithm in a set of training sequences.

Results:

With our prediction model we have achieved our intended objectives, which can be summarised as follows:

To make reliable predictions of volatility with a sufficiently manageable bounding error to make it operational in our model.

With this variable, to be able to estimate the value of the underlying, in our case the SPX, and to be able to control the trends of the options. Our RNR (recurrent neural network) has the ability to identify patterns or market scenarios because it identifies series of numbers instead of individual values compared to other traditional ANNs or neural networks. Moreover, these advanced networks handle the time variable of the identified series, from front to back, and can compare these current financial market scenarios with past scenarios, which provides them with an absolutely optimal prediction methodology for the purpose at hand.

Finally, these trends will be used to construct spread or non-standard strategies which, although they are delta-neutral strategies, take into account the statistical asymmetry of the trend. Materialised in a different number of contracts for each vertical. Finally, the RNR architecture has properties for anomaly detection and outlier detection that will help us to predict trend changes.

The result of all this has been the design of a non-standard spread such as an asymmetric iron condor that has been tested and obtains returns of between 2.5-10% per month because we have been able to shorten the optimal DTE (time to expiry) from 3 months to 1 month.
As an additional utility, this speed could be used as delta protection (delta neutral or 0) for large professional investment portfolios.

Conclusions:

Our aim with this prediction project was to find relationships that could not be detected with the naked eye in order to gain an advantage in professional investment. RNRs have characteristics that are perfectly suited to our objective, because they classify and predict sequences using time series as a classification criterion. These patterns can be earlier or later in time, unlike time series and thus improving on traditional artificial neural networks.
Using networks according to RNR's LSTM architecture (recurrent neural networks) we do not process individual data but sequences of data as price patterns at different points in time, essential for the study at hand, where prices follow similar patterns at different points in time.

The issue of having a set of fundamental variables that were indirectly related to the main one also additionally led us to obtain trend results, where statistical biases added profitable information when designing option spreads.

Finally, the RNR architecture has properties for anomaly detection and outlier detection that help us to predict trend changes.

As mentioned in the previous section, we have been able to predict volatility through the VIX, and being able to predict the SPX and trends, we have been able to design with the thinkorswim program a non-standard spread (asymmetric iron condor), tested with ONE (option net explorer) with the mentioned returns thanks to the shortening of the DTE by the prediction.

Objective and Motivation:

The motivation for which I chose this work was to do a project in which I had the possibility to combine engineering knowledge with financial knowledge.

To this end, I proposed this TFG together with my internship coordinator with the aim of delving deeper into machine learning and being able to use these skills to predict values and develop an options strategy.

ÍNDICE DE CONTENIDO:

ÍNDICE DE FIGURAS:

Capítulo 1. INTRODUCCIÓN

Cuando empezamos con este trabajo, pretendíamos averiguar el valor del SPX estimando el valor de la volatilidad mediante modelos de redes neuronales recurrentes. Si preguntásemos a alguien que desconozca los componentes técnicos, para que le puede servir este estudio en el ámbito de los mercados profesionales financieros, te contestará que "no existe la bola de cristal" porque si no compraría los títulos cuando vayan a subir y vendería cuando vayan a bajar. Eso sería el primer estadio o nivel de utilidad de los modelos de predicción con machine learning, deep learning, redes neuronales e inteligencia artificial como concepto amplio.

Si ampliamos un poco más desde la óptica de la ingeniería financiera esta primera idea, y nos trasladamos al ámbito de las opciones, tendríamos que calcular en primer lugar el precio de un activo que depende del valor de un subyacente, (en nuestro caso dicho subyacente es un índice) es decir una media ponderada de acciones u otros valores representativos, donde además podríamos analizar su comportamiento estudiando su ETF, en este caso el SPY.

Para ello, este TFG busca utilizar los avances tecnológicos del deep learning para analizar y procesar datos históricos con los que predecir las opciones del SPX y construir con ello estrategias alcistas, bajistas o neutrales.

Gracias a esta información futura, a lo largo del TFG se elaborarán estrategias de opciones diferentes al estándar (Iron Condor) adaptando la información con la predicción.

Capítulo 2. DESCRIPCIÓN DE LAS TECNOLOGÍAS

2.1 Lenguaje de programación

La programación de las redes neuronales fue realizada con un entorno Python aunque también se sopesaron otras alternativas parcialmente para la inclusión de ciertos algoritmos avanzados. Está opción se desestimó finalmente por la excesiva extensión que tomaría este TFG aunque podría ser una tarea a considerar en una futura profundización del mismo.

Python fue la herramienta finalmente seleccionada, a pesar de que el autor del Trabajo de Fin de Grado carecía de conocimientos previos de este lenguaje de programación. La razón por la que se eligió es la herramienta de Google, Google Colab.

- Google Colab es una herramienta de Alphabet que permite al usuario codificar y ejecutar Python desde todos los navegadores con tan solo una cuenta de gmail. Es un servicio en red, apoyado en los Notebooks de Jupyter, que ofrece acceso gratuito tanto a las TPUs (un acelerador de IA para el aprendizaje de redes neuronales desarrollado por el propio Google) como a las GPUs (un coprocesador especializado en el procesamiento de gráficos y operaciones con datos). Google colab cuenta con una mejor oferta que resto de plataformas, ya que los servidores de Google son más eficientes y procesan los datos con mayor velocidad que un ordenador portátil. Además, la disponibilidad de GPU es útil para ahorrar tiempo de ejecución.

2.2 Hardware

Un ordenador con características técnicas medias es suficiente para el desarrollo de este proyecto debido a que la mayor parte se realiza en la nube por medio de Google Colab y el procesamiento de datos puede realizarse con Microsoft Excel.

2.3 Software

- Windows 10 de 64 bits.
- Office: Word para la realización de la memoria descriptiva y Excel para la interpretación de los datos.

- Thynkorswim: para el diseño y visualización de las estrategias de opciones.

- Python y sus distintas librerías:

- NumPy: un paquete clave para la programación científica en Python que cuenta con un objeto de matriz multidimensional, una gran variedad de rutinas para operaciones en matrices a gran velocidad y varios objetos derivados. NumPy ha sido utilizado en este proyecto para estructurar y gestionar la base de datos de la red neuronal.
- Pandas: un kit de herramientas de análisis de datos específico de Python que permite analizar sintácticamente múltiples formatos de archivo. Esta librería ha sido necesaria para la correcta importación de los datos, hacer una primera limpieza de ellos y estructurarlos como lasalida y entrada de la red neuronal.
- Matplotlib: es una librería de software de Python útil en la generación de gráficos en base a datos contenidos en vectores o listas. Todos los gráficos resultantes han sido generados mediante esta librería.
- TensorFlow: esta librería con código abierto permite la implementación de la computación numérica del machine learning. Esta herramienta agrupa algoritmos y una gran cantidad de modelos tanto de aprendizaje automático como profundo. Esta librería utiliza Python para el desarrollo de aplicaciones con el framework, mientras usa esas aplicaciones para aumentar al máximo la eficiencia mediante C++. Tensorflow ha sido necesario para la construcción y entrenamiento del aprendizaje profundo.
- Keras: es una API de redes neuronales de gran nivel que permite ir de la idea al resultado reduciendo lo máximo posible el retraso. Permite la ejecución del código tanto en la CPU como en la GPU, facilita el desarrollo de prototipos de modelos con aprendizaje profundo y es capaz de soportar arquitecturas de red arbitrarias. Siendo idóneo para la construcción de cualquier modelo de aprendizaje profundo. Esta herramienta ha sido esencial en el proyecto para entrenar y modelizar las redes neuronales.
- Keras Tuner: es una librería que trabaja con Keras para definir redes neuronales y optimizar el modelo final determinando en un espacio de búsqueda para los parámetros (arquitectura) e hiperparámetros (opciones de configuración) del modelo. En este TFG, Keras Tuner ha sido implementada para la optimización del número de filtros, neuronas y la ratio de aprendizaje de los modelos.
- Scikit-learn: es una libreria de aprendizaje de Python que incorpora algoritmos de clasificación, regresión y agrupación con máquinas de vectores de bosques aleatorios, apoyo, aumento de gradientes, DBSCAN y k-means que interopera con bibliotecas

numéricas y científicas tanto de Python como de NumPy y SciPy. Sckikit-learn ha sido necesario en el trabajo para calcular funciones de coste de MSE, MAE y RMSE y estandarizar todos los datos en una escala entre uno negativo y uno positivo.

Esta estandarización de datos es crucial para el desarrollo del Deep Learning permitiendo optimizar y acelerar el entrenamiento, obteniendo con mayor rapidez la convergencia, así como habilitando la comparación de los datos variados. En el caso de este TFG, la capitalización y el volumen tienen unas unidades de millones, mientras que otros valores cuentan con valores de centenas o miles. Llevar todos los valores a un mismo rango permite que todos se ejecuten a una velocidad similar.

2.4 Datos

Los datos necesarios para la realización de este TFG han sido sacados en su mayoría de Yahoo Finance. Desde la propia página puede elegirse la frecuencia y el rango de fechas para descargar los datos. Dado que este trabajo tiene como objetivo el predecir la cotización a cierre, se ha escogido la opción de frecuencia diaria.

Los datos del VIX (índice sobre el que se va a trabajar) han sido descargados en un archivo csv desde Yahoo finance y los datos de la volatilidad implícita mediante programas de desarrolladores independientes de la página de proyectos GITHub.

Capítulo 3. ESTADO DE LA CUESTIÓN

El crecimiento tecnológico es perceptible en el aumento de compañías tecnológicas a nivel global, especialmente en Asia y EEUU, gracias al volumen de datos del que disponen y a su equipamiento (Fernández, 2019). Al observar en detalle el nivel de cambio ocupacional en USA en estos últimos años (visible en el Anexo 1), identificamos un crecimiento elevado en STEM (Ciencia, tecnología, ingenería y matemáticas) (Elvery, 2019), entre las que encontramos la robotización, el término cloud computing o el Big Data (tratamiento masivo de datos). Tecnologías que "están siendo utilizadas como palancas para la transformación digital de muchas organizaciones" (Banco de España, 2020).

La inteligencia artificial (IA) tiene papeletas para ser el cambio tecnológico más veloz en la historia. Los líderes mundiales en la incorporación de inteligencia artificial son Asia y el Pacífico, lugares donde el veinte por ciento de las empresas ha implementado ya la IA en sus estructuras de negocio; seguido por Norte América con el diez por ciento (MMC Ventures, 2019). Europa, algo retrasada, cuenta con un crecimiento exponencial en el emprendimiento de empresas relacionadas con IA. Así, en 2020, casi el diez por ciento de las nuevas empresas contaba en su propuesta de valor con IA; mientras que, en 2013, solo un dos por ciento. En 2019 había más de 1600 empresas relacionadas con IA a nivel europeo. Destacando Reino Unido por encima del resto con una tercera parte del total, seguido por Francia, Alemania y España (en cuarta posición) (MMC Ventures, 2019). "Los niveles de adopción generalizada entre los empresarios actuales de la IA, es un indicador importante de un futuro en el que la IA es omnipresente tanto a corto como a largo plazo" (MMC Ventures, 2019: 100).

Al ser la IA un sector importante y creciente en la actualidad, estudiar su impacto y posibilidades es de gran interés. Es por ello por lo que, en este TFG, nos centraremos en el sector financiero y más en concreto en los mercados de opciones.

¿Porque centrarse en el mercado financiero?

La cantidad del volumen de datos manejados en este sector incentiva la incorporación de esta tecnología. "Los bancos aun no son conscientes de las oportunidades que tienen de más del 80%

de los datos acumulados" (United Consulting Group, 2018). La automatización y la digitalización, las instituciones financieras han encontrado una nueva oportunidad de especializarse en el tratamiento de los datos y ofrecer un valor añadido a sus clientes, anticipándose a sus necesidades mediante las distintas herramientas analíticas disponibles.

El Índice de Digitalización de la Industria de MGI (Manyika & Bughin, 2018), muestra como el sector financiero es uno de los principales inversores y usuarios de IA.

Por ello, el concepto Fintech (finance and technology) está revolucionando el sector financiero (Philippon, 2016). Una revolución impulsada por el aprendizaje automático, la inteligencia artificial y el Smart Data (Cuya, 2016).

J.P. Morgan define como una nueva era en el trading gracias al aprendizaje. Mientras que los algoritmos anteriores estaban codificados con reglas, J.P. Morgan está explorando la siguiente generación de programación, que permite al aprendizaje automático descubrir de forma independiente estrategias de negociación de alto rendimiento a partir de datos brutos.

Es por ello que, en este TFG, haremos especial hincapié en la utilización del machine learning en los mercados financieros, trabajando en un modelo que sea capaz de analizar y clasificar datos para predecir valores futuros.

Capítulo 4. DEFINICIÓN DEL TRABAJO

La inteligencia artificial (IA) es el término más amplio utilizado para clasificar los programas que imitan la inteligencia humana. Se utiliza para predecir, automatizar y optimizar tareas que los humanos han realizado históricamente, como pueden ser, el reconocimiento de voz, el reconocimiento facial, la traducción o la automatización de decisiones.

4.1 Tipos de IA y su relación con el DL, ML y RN

La inteligencia artificial se compone de tres categorías principales: La Inteligencia General Artificial (AGI), la Inteligencia Artificial Estrecha (ANI), y la Súper Inteligencia Artificial (ASI).

La Inteligencia Artificial Estrecha, también conocido como IA débil, se define por su capacidad para completar una tarea muy específica, como ganar una partida de ajedrez o identificar a un individuo entre una serie de fotos. A medida que avanzamos hacia formas más fuertes de IA, como AGI y ASI, se va viendo paulatinamente una incorporación de comportamientos más humanos, como la capacidad de interpretar el tono y la emoción.

La Inteligencia General Artificial (AGI) y la Súper Inteligencia Artificial (ASI) se consideran como IA fuerte y se caracterizan por ser comparable sus reacciones y respuestas con los humanos. (AGI) funciona de forma parecida a un ser humano, mientras que (ASI), supera la inteligencia y las capacidades de los humanos, siendo conocida como superinteligencia. Estas IA fuertes no existen todavía pero su investigación continúa.

La inteligencia artificial, el aprendizaje profundo, el aprendizaje automático y las redes neuronales son términos similares con matices particulares. Todas ellas están relacionadas como si de muñecas rusas se tratara, donde cada concepto es esencialmente un componente del término anterior. El machine learning es un subcampo de la inteligencia artificial, el deep learning del machine learning y las redes neuronales constituyen la columna vertebral de los algoritmos del deep learning.

4.2 Diferencia entre el aprendizaje profundo y las redes neuronales

La principal diferencia es el número de capas ocultas, mientras que las redes neuronales se componen de hasta 3 capas, el aprendizaje profundo está compuesto por varias capas ocultas. Toda red neuronal que conste de más de tres capas, (incluyendo la salida y las entradas), puede ser considerado como un algoritmo de aprendizaje profundo. Refiriéndose por "profundo" a la profundidad de las capas en una red neuronal.

4.3 Diferencia entre deep learning y machine learning

La principal diferencia se encuentra en la cantidad de datos que utiliza cada tipo de algoritmo y en cómo aprende cada uno. El deep learning automatiza un patrón con las características propias del proceso, eliminando una gran parte de la participación humana y permitiendo el uso de grandes conjuntos de datos, ganándose el título de "machine learning escalable". El deep learning es simplemente un subconjunto del machine learning.

El machine learning clásico, o "no profundo", tiene una mayor dependencia de las acciones humanas para aprender, siendo necesarios para determinar la jerarquía de las variables y poder diferenciar entre las entradas de datos, requiriendo datos más estructurados como inputs para aprender.

El deep learning tiene la capacidad de aprovechar los conjuntos de datos etiquetados para informar a su algoritmo, tratar datos no estructurados, y determinar automáticamente el conjunto de variables necesarias.

Al observar los patrones en los datos, un modelo de deep learning puede agrupar las entradas de manera adecuada, podríamos agrupar características o variables en función de las similitudes o diferencias identificadas. Sin embargo, Un modelo de deep learning requeriría más datos para mejorar su precisión, mientras que un modelo de machine learning se basa en menos datos por manejar una estructura de datos similar. El deep learning se aprovecha principalmente para casos de estructuras menos evidentes o más complejas.

4.4 Redes neuronales

Las redes neuronales artificiales (RNA), imitan el cerebro humano a través de un conjunto de algoritmos. Las RNA se componen de cuatro componentes principales: pesos, entradas, un sesgo y una salida (que serán explicados a continuación).

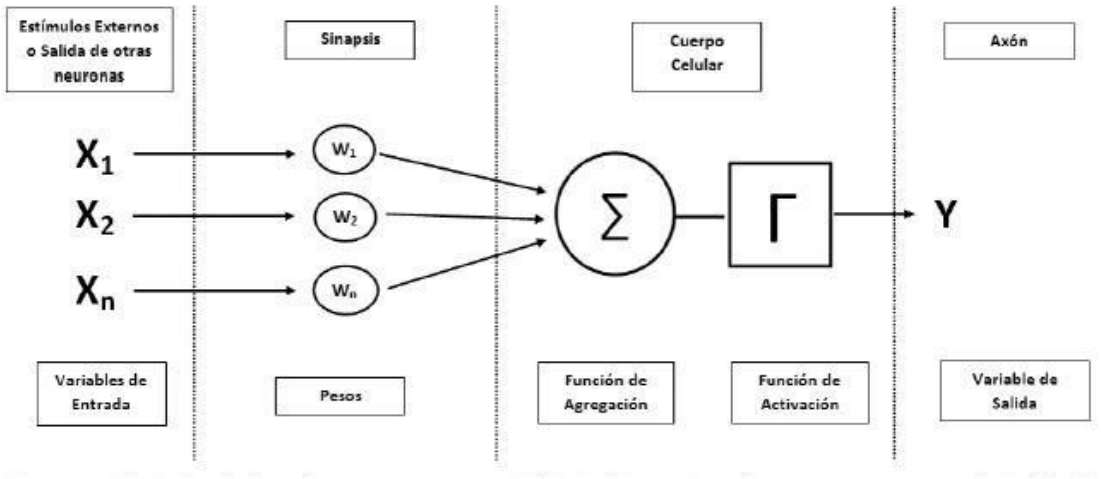


Figura 1. Funcionamiento de Redes Neuronales

Variables de entrada:

- Tenemos las X1 hasta Xn de entrada, eso vendrían a ser los datos de entrada que queremos que el algoritmo aprenda.

Pesos:

- A estos datos se le aplican una operación, que es la multiplicación de los datos por unos pesos, el valor de estos pesos y su optimización serán los que permitirá al algoritmo hacer buenas predicciones. El valor de los pesos en principio son aleatorios. Más adelante entenderemos el porqué de los pesos.

Función de agregación:

- Esta sería la parte de la neurona, donde se realiza un sumatorio de la multiplicación de cada dato por su peso. Una suma ponderada.

Función de activación:

- Al salir los datos del sumatorio le aplicamos una función de activación, lo que hace es romper la linealidad. ¿Qué significa esto? El algoritmo tiene que encontrar relaciones y patrones entre los datos. En el mundo real esta relaciones y patrones no son lineales, necesitamos hacer esta transformación porque el resultado de salida del sumatorio es lineal porque tenemos una suma ponderada, es como si la neurona realizará una

regresión lineal. Pero para resolver problemas complejos necesitamos algo más que una regresión lineal.

Variable de salida:

- Será el resultado final de las operaciones realizadas en los datos. Nuestra predicción

En la siguiente imagen (Figura 2) tenemos un problema de clasificación, en el gráfico A podemos ver lo que decíamos de la linealidad, es un problema que se puede resolver linealmente. En cambio, el gráfico B es totalmente distinto, necesitamos algo más que una función lineal para resolver el problema.

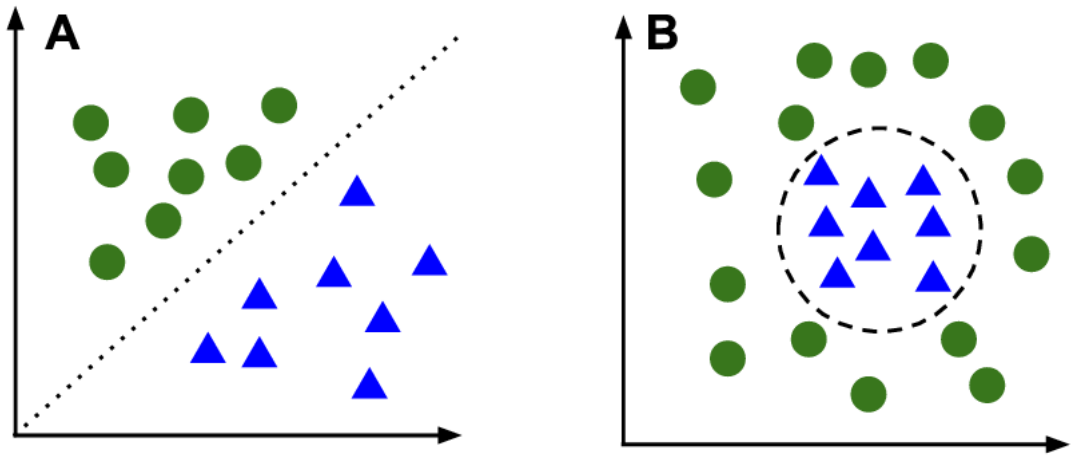


Figura 2. Linealidad

Por eso la importancia de las funciones de activación en las neuronas, aplicamos una operación a los datos de salida de la neurona que rompen la linealidad, como vimos en el gráfico B.

Ahora que hemos entendido como función una neurona, entendamos que sucede dentro de una red neuronal artificial. Esta es la pinta que tiene una red neuronal artificial. Hay tres componentes esenciales, la capa de entrada, la capa oculta y la capa de salida.

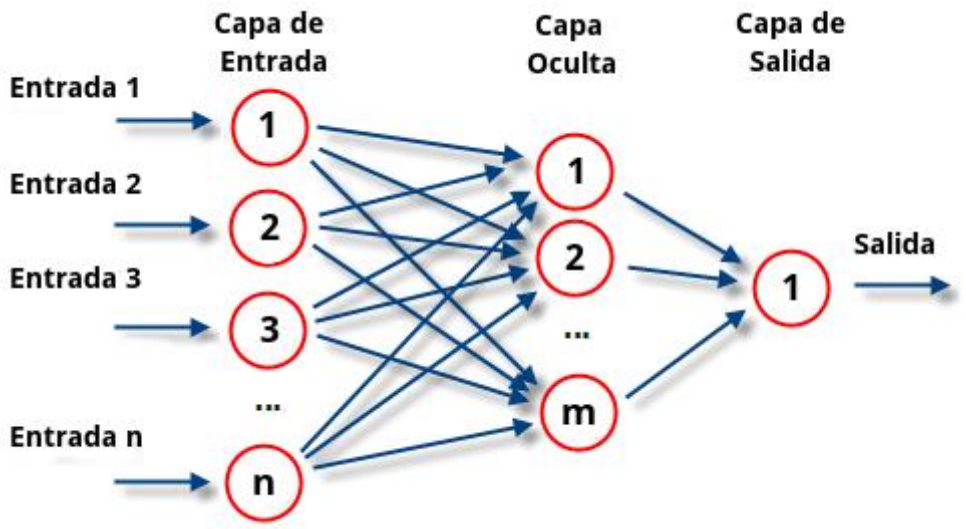


Figura 3. Red Neuronal Artificial

Capa de entrada:

- Corresponde a los datos de entrada, habrá tantas neuronas como datos le suministramos al algoritmo. Como datos, tenemos que entender que tendremos una neurona de entrada para cada variable que utilicemos para que la red neuronal aprenda.

Capa oculta:

- En la imagen anterior se ve solo una, pero dentro de la capa oculta puede haber tantas capas como quieras. Cuantas más capas, más profunda será, de ahí el nombre de Deep Learning "aprendizaje profundo". En las capas ocultas no hay restricción de neuronas, puedes poner todas las que quieras en cada capa.

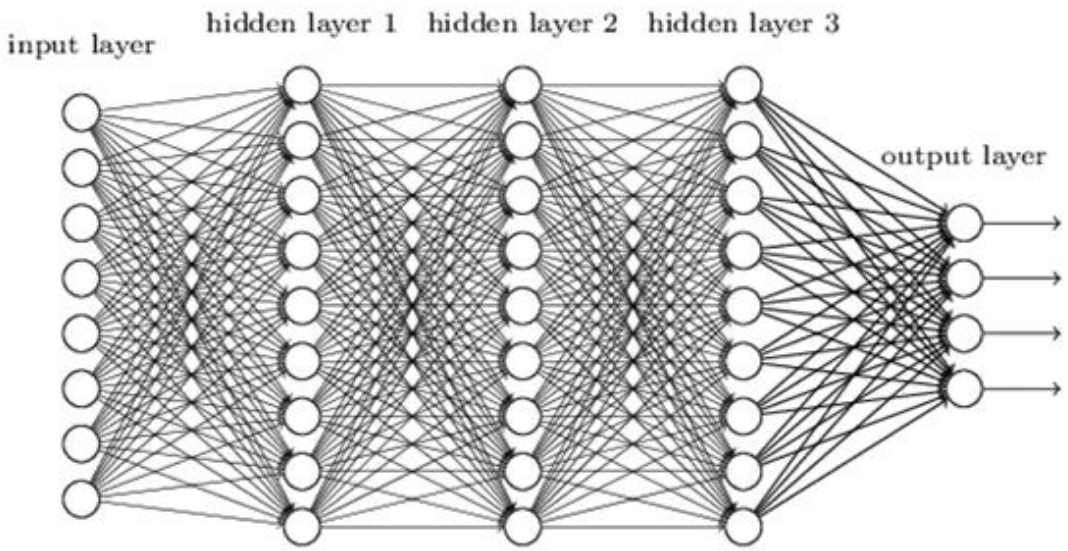


Figura 4. Red Neuronal Artificial a mayor escala

Capa de salida:

- Será el resultado final, en este caso la predicción del algoritmo. El número de neuronas en la capa de salida vendrá restringido por el tipo de problema que queramos resolver. Si es un problema de regresión, que predice un valor numérico, como el valor de las acciones de Opciones del SPX, solo tendremos una neurona de salida. Si es un problema de clasificación, tendremos tantas neuronas de salida, como objetos estamos clasificando.

Ahora que ya entendemos la estructura de la red neuronal. Veamos cómo es que una red neuronal artificial, aprende y se hace inteligente como lo hace un humano.

Aprendizaje:

- Primero entendamos como un humano aprende ante algo nuevo. Imagínese que una persona nunca ha visto un coche y no tiene ni idea de su funcionamiento. Lo que hará será tomar decisiones aleatorias, que le devolverán un resultado, recibiendo un feedback, que lo utilizará para mejorar las decisiones posteriores. Esta persona irá tocando los pedales obteniendo resultados, en un primer momento erróneo con lo que

irá aprendiendo lo que tiene o no que hacer. Esto mismo ocurre dentro de una red neuronal artificial gracias a los pesos en las neuronas y las técnicas de forward, back propagation y funciones de pérdida. Esta capacidad de las redes neuronales recurrentes para analizar hacia delante y hacia atrás los datos, siendo capaces de identificar patrones es la razón principal por la cual este TFG emplea este tipo de redes.

En la técnica de forward propagation, propagación hacia delante, los datos avanzan de neurona tras neurona capa a capa. Produciéndose cálculos en cada neurona, donde multiplicamos los datos por los pesos específicos de esa neurona que empiezan siendo aleatorios, igual que en el ejemplo de la conducción que tomábamos decisiones aleatorias cuando no sabíamos qué hacer. Por ese mismo motivo los pesos empiezan siendo aleatorios. Hasta llegar a la salida del resultado final, que sería el resultado de haber apretado algún pedal.

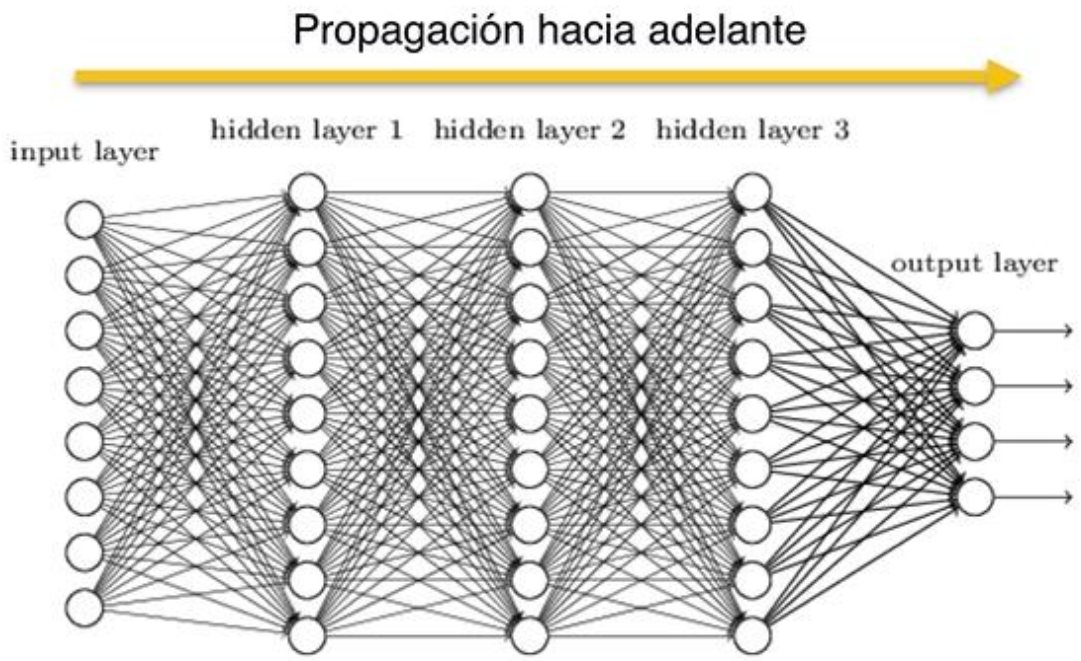


Figura 5. Propagación hacia delante

Entonces, la red lo que hace es comparar este resultado predicho con el real obteniendo el error de predicción. Con la técnica de backpropagation, este error lo devolvemos a la red para que se optimice.

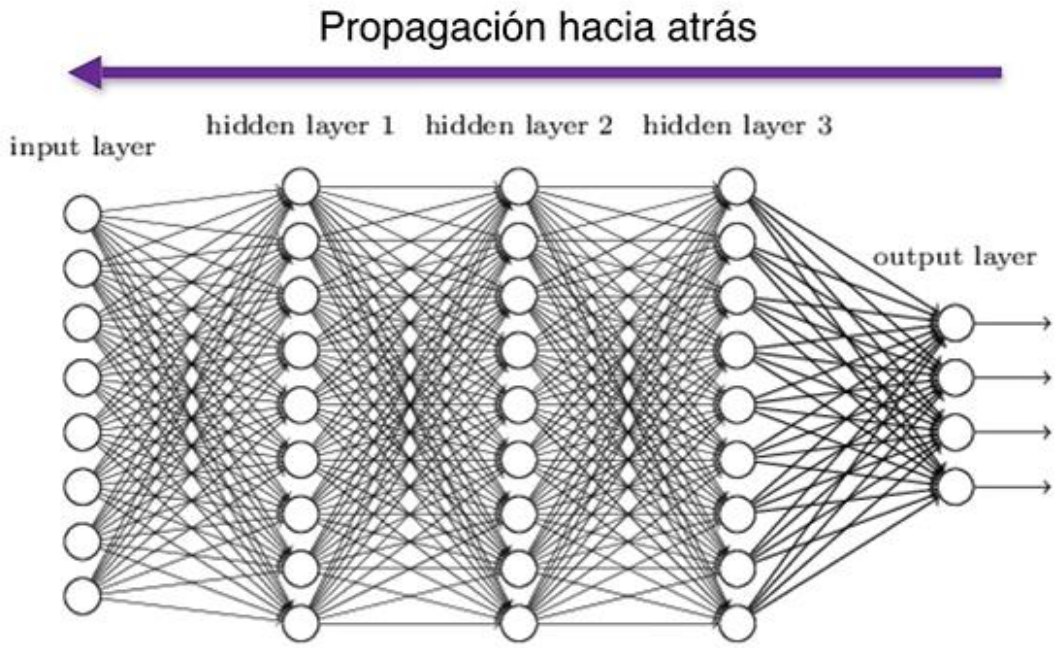


Figura 6. Propagación hacia atrás

La red neuronal repite este proceso de ir pasando los datos hacia adelante, obteniendo predicciones con un error y comunicándoselo a sus neuronas para que modifiquen sus pesos y se reduzca dicho error. Hasta llegar al resultado deseado, una predicción lo más cercana al valor real.

Esto se produce en la parte del entrenamiento, donde se pasan a la red tanto los valores que utilizaremos para enseñar al programa como los valores reales que queremos predecir. Es la misma manera en la que aprendemos nosotros, cuando realizamos una acción obtenemos un resultado, y si no es el deseado, utilizamos esa información para mejorar nuestra decisión.

Una vez entendido el funcionamiento de las redes neuronales avanzadas, es necesario conocer la otra cara de este TFG, las estrategias con opciones.

4.5 Opciones

El término opción se refiere a un instrumento financiero que otorga al tenedor el derecho de comprar o vender valores o bienes basados en el subyacente, como las acciones. En el caso de las opciones, el titular tiene la posibilidad de ejercer o no su derecho de compra o venta antes de la expiración de la fecha de vencimiento específica para dicho contrato. El precio establecido al cual el comprador tiene la posibilidad de comprar-vender el subyacente al emisor de la opción, se conoce como precio de ejercicio (strike). Los brokers minoristas son los medios usuales para adquirir estos contratos.

Para este TFG, haremos uso de la estrategia Iron Condor, que es la que más se ajusta a nuestras necesidades.

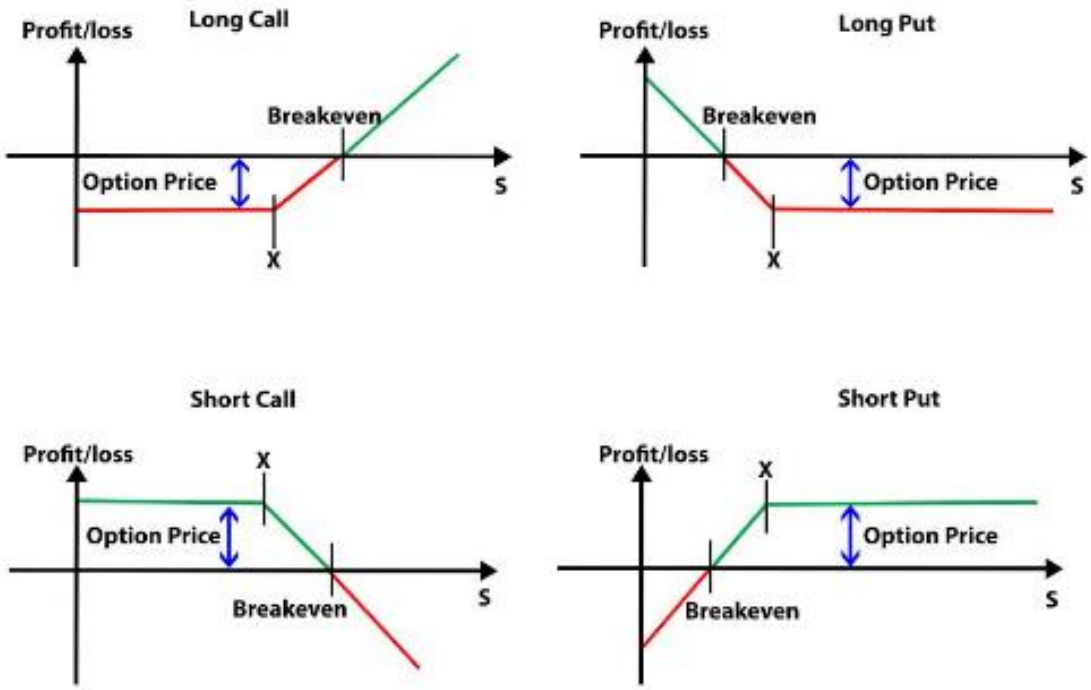


Figura 7. Opciones

4.6 Iron Condor

Iron Condor es una estrategia formada con cuatro opciones**,** dos opciones put y dos opciones call. Esta estrategia se caracteriza por no verse afectada por el movimiento del precio dentro de un determinado rango (lo cual será beneficioso para nosotros para compensar el error) y por verse beneficiada por el paso del tiempo. Está formada por una delta neutral y theta positiva. Esta estrategia será utilizada y explicada con más detalle tras la predicción con ML.

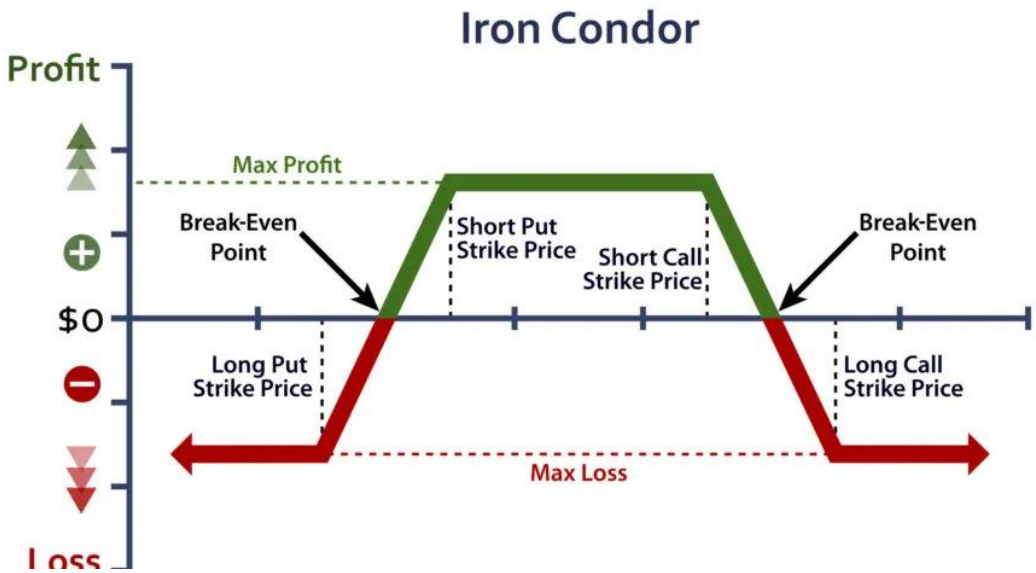


Figura 8. Iron Condor

4.7 Volatilidad

La volatilidad suele referirse al grado de incertidumbre o riesgo en el valor de un título por sus cambios de valor. Una mayor volatilidad en un valor supone que este se reparta en un mayor rango de valores, pudiendo cambiar su valor de manera drástica en cualquier dirección en un corto periodo de tiempo. Esto significa que el precio del valor puede cambiar drásticamente en un corto periodo de tiempo en cualquier dirección. Por el contrario, si la volatilidad es inferior, el valor de un título es más estable sin grandes variaciones de precio en un corto periodo de tiempo.

Una forma de medir la variación de un activo es cuantificar la cantidad de variación de un título de manera diaria (medido en porcentaje). El grado de variabilidad de los rendimientos de un activo se representa mediante la volatilidad histórica o realizada, la cual carece de unidades y se mide en porcentaje.

La volatilidad se define matemáticamente como la desviación estándar de los rendimientos de un activo durante un periodo de tiempo determinado. Para el cálculo de la volatilidad, se toma la raíz cuadrada de la varianza multiplicada por la raíz cuadrada del tiempo (en días).

4.8 VIX

El VIX o "el índice del miedo", es un índice que representa la volatilidad del precio de las opciones sobre el SP500, para aquellas opciones con expiración inferior a 30 días. Cuando los niveles del VIX son relativamente altos, es decir hay altas volatilidades en los precios de las opciones, suele haber descensos en la cotización del valor, aumentando el precio del VIX cuando hay caídas en los mercados.

La volatilidad, o la rapidez con la que cambian los precios, suele considerarse una forma de medir el sentimiento del mercado y, en particular, el grado de temor de los participantes en él.

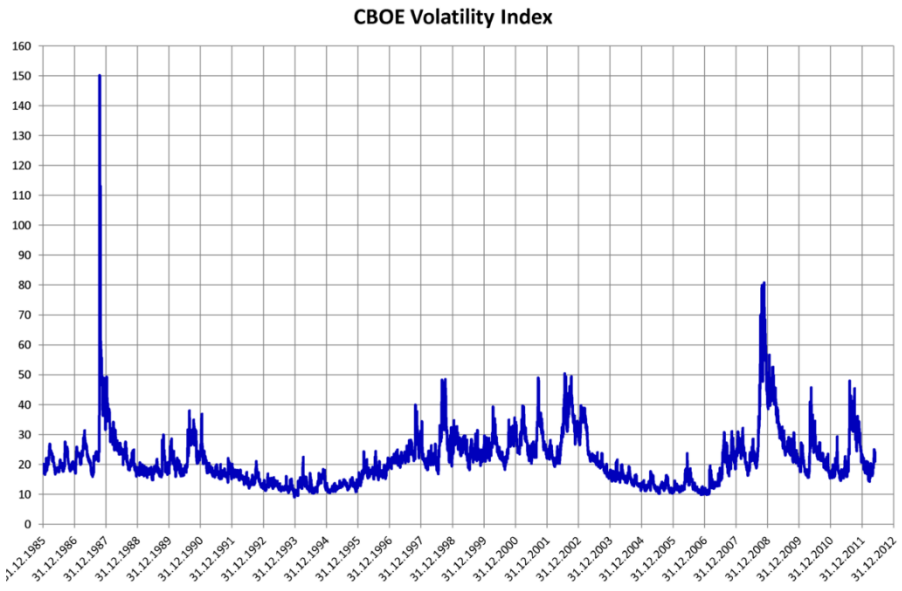


Figura 9. VIX

4.9 Métricas de riesgo de las opciones: Las griegas

Las griegas son unas relaciones de control de las opciones, matemáticamente expresadas como derivadas. Las griegas que nosotros vamos a considerar son:

Delta:

- Delta (Δ) es una variable fundamental en una opción que sirve para representar la variación entre el subyacente y el precio de una opción. La delta tiene un rango contenido entre uno negativo y uno positivo, siendo entre cero y uno si es una opción de compra y entre uno negativo y cero si es una opción de venta. Delta indica también la ratio de cobertura necesario para formar una posición delta-neutral para los compradores. Otro uso menos común de la delta de una opción es la probabilidad existente de que expire ITM (figura).

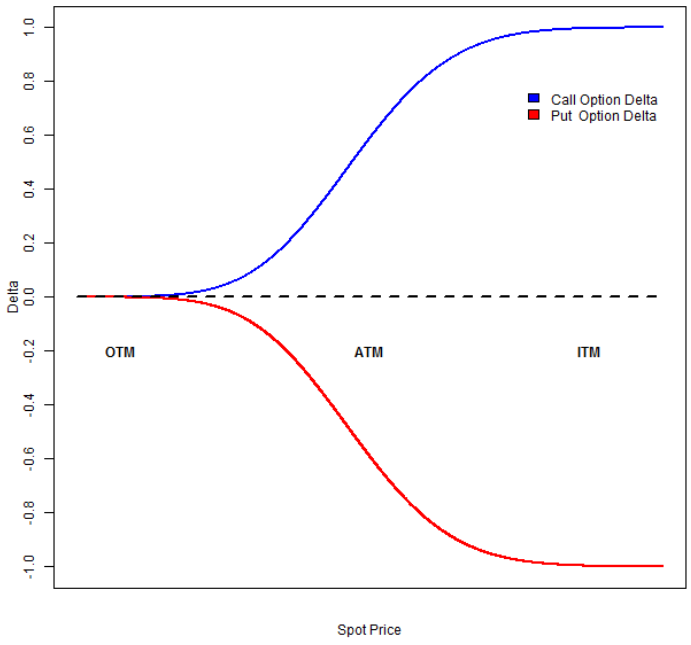


Figura 10. Delta

Theta:

- Theta (Θ) constituye la tasa de cambio existente entre el precio de una opción y el tiempo de expiración de esta, representando de esta manera la sensibilidad al tiempo, "time decay" de una opción (explicado en página 41). Theta es útil para identificar cuanto baja el precio de una opción con relación al precio, disminuyendo si nos acercamos al tiempo de expiración (figura 11). Aquellas opciones que se encuentren más cercanas a su vencimiento tendrán un decaimiento temporal acelerado. Las opciones short call y short put tienen una Theta positiva, mientras que, las opciones long call y las opciones long put tienden a tener una Theta negativa. Es importante comentar que la Theta incrementa cuando las opciones se encuentran ATM, y

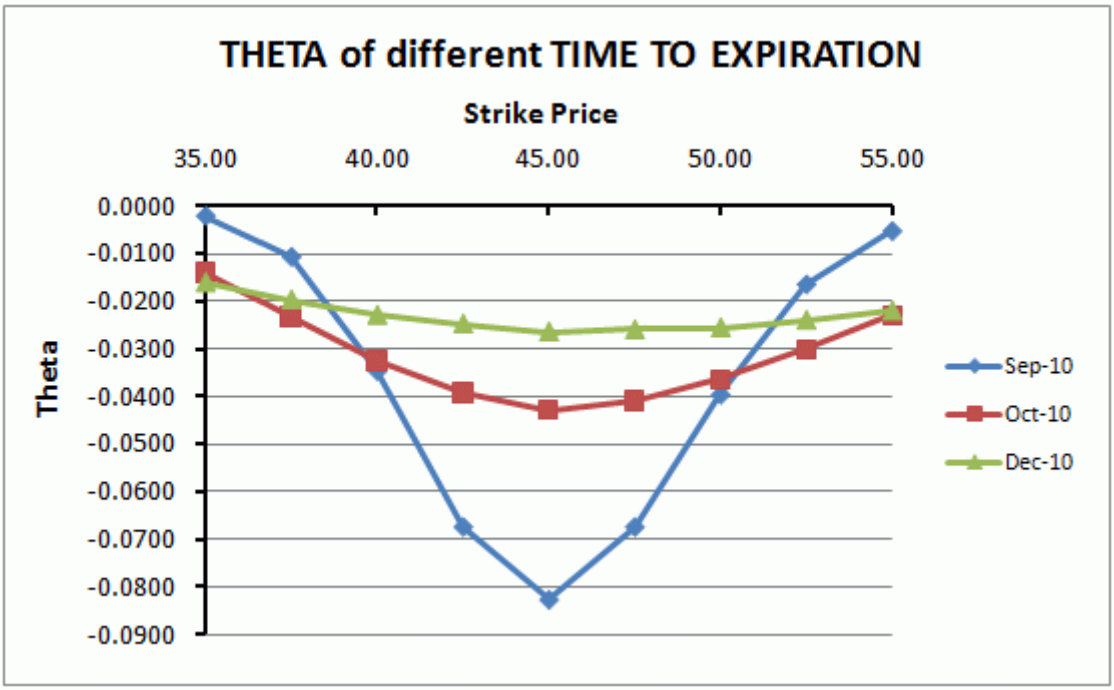


Figura 11. Theta

disminuye cuando las opciones están ITM y OTM

Gamma:

- Gamma (Γ) indica la relación que hay entre el Delta (Δ) de una opción y el valor del activo subyacente, siendo útil para identificar la cantidad que cambiaría la delta al producirse un incremento o disminución de un dólar en el valor subyacente. En otras palabras, la gamma determina la estabilidad de la delta de una opción. Cuanto más alto sea el valor de gamma, mayor volatilidad tendrá la delta en respuesta a los movimientos del subyacente y menor volatilidad cuanto menor sea gamma. El valor de gamma es menor para las opciones ITM y OTM y mayor para las opciones ATM (figura 12), acelerando su precio según se acerca el vencimiento por lo que cuanto mayor sea el vencimiento, menos sensible será a los cambios de delta.

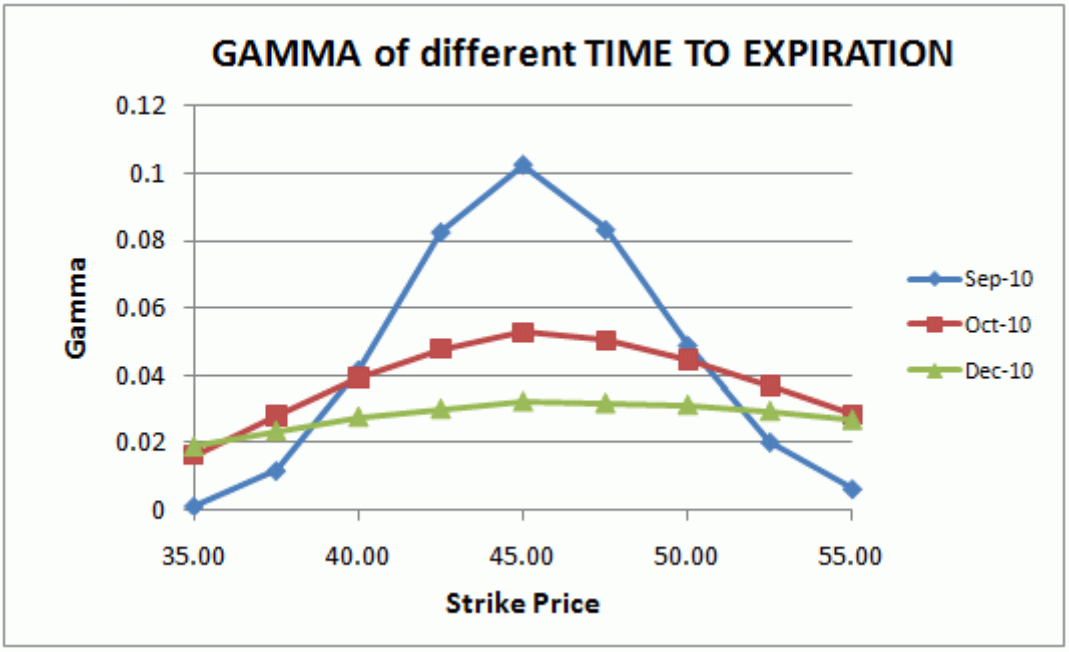


Figura 12. Gamma

Vega:

- Vega (V) es una variable fundamental en una opción que mide la relación entre la volatilidad implícita del activo subyacente y el precio de la opción. Representa el cambio en el precio de una opción ante una variación del uno por ciento en la volatilidad implícita.

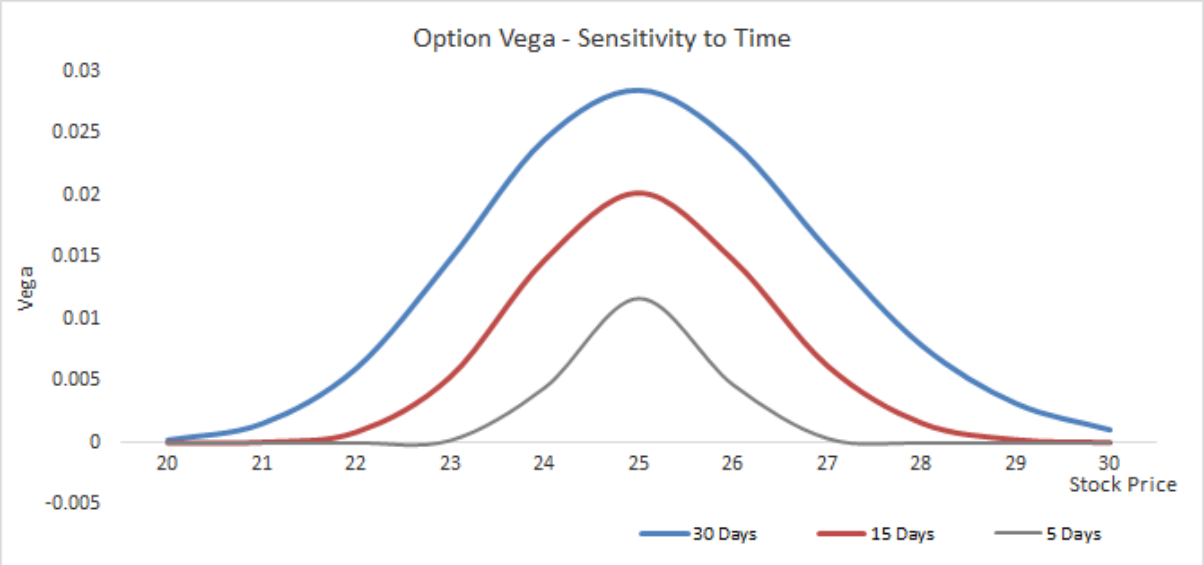


Figura 13. Vega

Como vemos en la figura 13, vega aumenta a medida que nos acercamos a la fecha de expiración debido al aumento de la volatilidad.

Ampliación del concepto theta decay (pagina 38)

Theta decay:

- Al hablar de este término, nos referimos a la tasa de depreciación del valor de las opciones con respecto al tiempo. Si ni la cotización ni los factores de volatilidad implícita IV variaran a lo largo del tiempo, el precio de nuestras opciones descendería a una tasa igual al theta decay. Con el paso del tiempo, la incertidumbre disminuye y por tanto la prima pierde valor (figura 14).

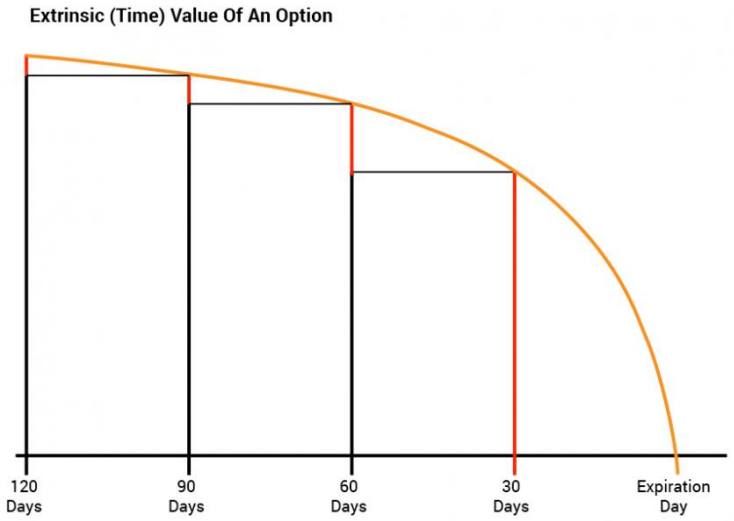


Figura 14. Theta decay

Capítulo 5. DESARROLLO DEL PROYECTO

A continuación, introduciremos los datos de las bases de datos en el programa sobre RNR (redes neuronales recurrentes) para predecir con un error operativo los valores de la volatilidad y deducir los valores del índice SPX. Posteriormente con los precios de las opciones put y call y las verticales bull put, bear call y bear put, diseñaremos un spread aprovechando las tendencias que indican los patrones de la RNR para aprovechar el sesgo estadístico.

5.1. Carga, preprocesado y visualización de los datos

Antes de poder crear cualquier algoritmo y predicción, un trabajo muy importante es la parte del preprocesado de los datos, dejar el dato en el formato correcto para poderlo introducir en el algoritmo. Veamos los diferentes pasos del preprocesado de datos.

Primero importamos las librerías básicas para la importación y manipulación de los datos. Para ello hacemos uso de las librerías de Pandas (para manipular los datos) de Numpy (para aplicar operaciones matemáticas a los datos y manipular arrays que son vectores o matrices) y Matplotlib (para poder graficar los datos obtenidos) y por supuesto keras.

```
import keras
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense, LSTM
import matplotlib.pyplot as plt

import math
import pandas_datareader as web
import numpy as np
import pandas as pd
```

Una vez tenemos las librerías necesarias, importamos el dataset que contiene los datos de entrenamiento e imprimimos por pantalla las 5 primeras filas del dataset con la función head, para poder ver de los datos de los que disponemos

```
from google.colab import files
files.upload()
```

```
Upload widget is only available when the cell has been executed in the current browser session.
Please rerun this cell to enable.
Saving ^VIX (1).csv to ^VIX (1).csv
```

```
{'^ b'Date,Open,High,Low,Close,Adj Close,Volume\n1994-06-
14,11.620000,11.930000,11.380000,11.600000,11.600000
```

De todo el dataset, nosotros nos centraremos en la columna Close, que representa los valores de apertura del índice VIX. Ahora mismo los datos están en formato data frame, pero para poder introducir los datos en una red neuronal, tenemos que transformarlos ya que solo funcionan con datos en formato de matriz. Necesitamos convertir la columna Open a un formato numpy array. Este proceso es básico y esencial dominarlo, y entender que es un vector fila, un vector columnas y una matriz.

Para seleccionar la columna que queramos, podemos usar la función loc. Pero al extraer solo una columna obtendremos la siguiente dimensión y tipo de dato.

```
#Store The data
df=pd.read_csv('^VIX (1).csv')
#show the data
Df
```

	Date	Open	High	Low	Close	Adj Close	Volume	
0	1994-06-14	11.620000	11.930000	11.380000	11.600000	11.600000		
1	1994-06-15	11.360000	12.390000	11.210000	11.520000	11.520000		
2	1994-06-16	11.360000	11.890000	11.200000	11.220000	11.220000		
3	1994-06-17	10.520000	12.330000	10.480000	12.310000	12.310000		
4	1994-06-20	13.770000	14.210000	13.380000	13.960000	13.960000	0	
...	...	...	...	...	...	...	...	
7046	2022-06-07	25.540001	26.240000	23.879999	24.020000	24.020000	0	
7047	2022-06-08	24.370001	24.780001	23.740000	23.959999	23.959999	0	
7048	2022-06-09	24.290001	26.240000	23.820000	26.090000	26.090000	0	
7049	2022-06-10	26.260000	29.629999	26.049999	27.750000	27.750000	0	
7050	2022-06-13	31.370001	35.049999	31.290001	34.020000	34.020000	0	

```
7051 rows × 7 columns
```

```
#Get the number of rows and columns in the data set
df.shape
```

Visualizamos los datos de la columna close

```
#Visualize the closing price history
plt.figure(figsize=(16,8))
plt.title('Close Price History')
plt.plot(df['Close'])
plt.xlabel('Date', fontsize=18)
plt.ylabel('Close Price USD ($)', fontsize=18)
plt.show()
```

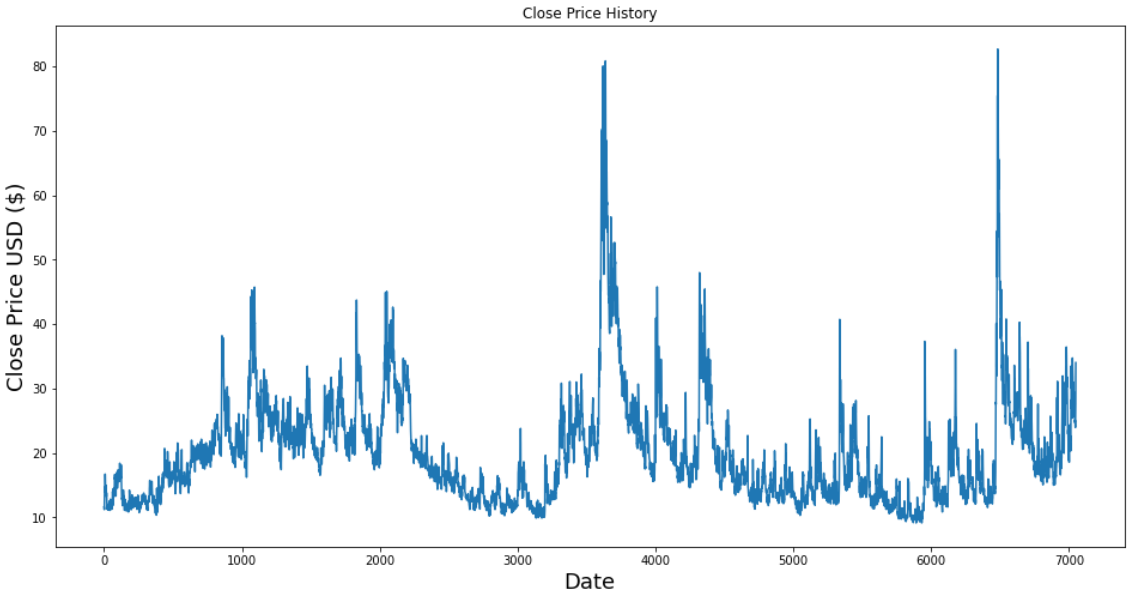


Figura 15. Visualización de datos

Como podemos observar el tamaño es 7051 lo que significa que es un vector fila. Pero para poder introducir datos en una red neuronal tiene que estar en formato matricial o vector columna, que es una matriz con solo una variable.

Para convertir nuestro vector fila a vector columna, solo necesitaremos poner entre corchetes el nombre de la variable y añadir el atributo values, que convertirá los datos de tipo pandas series a formato numpy array, que es el formato matricial.

Ahora nuestras variables están en el formato correcto, vector columna o matricial para poder introducirlo a la red neuronal.

```
#Create a new dataframe with only the 'Close column
data = df.filter(['Close'])
#Convert the dataframe to a numpy array
dataset = data.values
#Get the number of rows to train the model on
training_data_len = math.ceil( len(dataset) * .8 )
training_data_len
```

5.2 Escalado

Ahora que ya tenemos los datos seleccionados para realizar la predicción, necesitamos pasar la siguiente etapa de transformación de los datos, que será escalar los datos. Consiste en transformar los datos en una escala diferente. Las dos mejores formas de escalar variables son, la estandarización y la normalización.

```
#Scale the data
scaler = MinMaxScaler(feature_range=(0,1))
scaled_data = scaler.fit_transform(dataset)

scaled_data
```

```
array([[0.03344663],
       [0.03235894],
       [0.02828008],
       ...,
       [0.23045547],
       [0.25302515],
       [0.33827327]])
```

En la estandarización pasamos los datos a una distribución normal, restando la media y dividiendo por su desviación estándar mientras que la normalización es transformar los datos al rango [0,1]. A cada dato se le resta el mínimo y se divide por la resta del máximo menos el mínimo.

5.3 Normalización

Podemos usar el método que queramos, pero es más recomendable la normalización, por la forma en la que funciona internamente una red neuronal artificial.

Para esta tarea, utilizaremos otra librería muy importante en Machine Learning, scikit-learn donde tenemos métodos de preprocesado de datos y utilizaremos la función MinMaxScaler para realizar el escalado.

Ahora los valores han pasado a estar entre el rango [0, 1], siendo 0 el valor mínimo de la acción y 1 el máximo. El resto de los valores estarán comprendidos entre ese rango. Esta operación de normalizar los datos es muy común en Machine Learning, sobre todo en problemas de clasificación.

5.4 Estructuras dimensionales

Ahora ya tenemos los datos con las transformaciones adecuadas para la red neuronal artificial. La arquitectura interna de la Red Neuronal Recurrente es diferente a una red neuronal artificial, ya que tiene la capacidad de tener memoria, gracias a la capa LSTM. Por lo que tenemos que crear una estructura de datos, que especifique lo que la red neuronal recurrente tendrá que recordar, para realizar una predicción en base a los valores anteriores, lo que se denomina como el número de timesteps.

```
#Create the training data set
#Create the scaled training data set
train_data = scaled_data[0:training_data_len , :]
#Split the data into x_train and y_train data sets
x_train = []
y_train = []

for i in range(60, len(train_data)):
 x_train.append(train_data[i-60:i, 0])
 y_train.append(train_data[i, 0])
 if i<= 61:
   print(x_train)
   print(y_train)
   print()
```

Nuestra estructura de datos tendrá 60 timesteps de entrada y un valor de salida. Los 60 timesteps, significan que, en cada momento del tiempo, la red neuronal será capaz de mirar 60 días atrás. Serán 60 precios de las acciones antes del día actual y en base a eso, intentar predecir el precio de la acción, del día siguiente. Son sesenta días financieros, teniendo en cuenta que la bolsa no abre los fines de semana, equivalen a 3 meses de valores bursátiles, un trimestre.

Con el siguiente bucle ‘for’ crearemos la estructura de datos de los 60 timesteps:

```
[array([0.03344663, 0.03235894, 0.02828008, 0.04309993, 0.06553365,
        0.07559483, 0.06634942, 0.06770904, 0.10305913, 0.08687967,
        0.08021754, 0.0717879, 0.0792658, 0.07097213, 0.07858599,
        0.07559483, 0.06621346, 0.05601631, 0.06376614, 0.05791978,
        0.04459551, 0.03399048, 0.02909585, 0.03127124, 0.03412644,
        0.03480625, 0.03412644, 0.02828008, 0.03154317, 0.03426241,
        0.03793338, 0.039293, 0.02705642, 0.02760027, 0.02895989,
        0.03004759, 0.04133243, 0.04350782, 0.04445955, 0.04622706,
        0.03766145, 0.04065262, 0.02651258, 0.03970088, 0.03059143,
        0.03263086, 0.03983684, 0.05016995, 0.04731475, 0.03548606,
        0.03303875, 0.03915703, 0.04146839, 0.0364378, 0.02841604,
        0.03847723, 0.03698164, 0.0307274, 0.03467029, 0.03630183])]
[0.0369816441337418]

[array([0.03344663, 0.03235894, 0.02828008, 0.04309993, 0.06553365,
        0.07559483, 0.06634942, 0.06770904, 0.10305913, 0.08687967,
        0.08021754, 0.0717879, 0.0792658, 0.07097213, 0.07858599,
        0.07559483, 0.06621346, 0.05601631, 0.06376614, 0.05791978,
        0.04459551, 0.03399048, 0.02909585, 0.03127124, 0.03412644,
        0.03480625, 0.03412644, 0.02828008, 0.03154317, 0.03426241,
        0.03793338, 0.039293, 0.02705642, 0.02760027, 0.02895989,
        0.03004759, 0.04133243, 0.04350782, 0.04445955, 0.04622706,
        0.03766145, 0.04065262, 0.02651258, 0.03970088, 0.03059143,
        0.03263086, 0.03983684, 0.05016995, 0.04731475, 0.03548606,
        0.03303875, 0.03915703, 0.04146839, 0.0364378, 0.02841604,
        0.03847723, 0.03698164, 0.0307274, 0.03467029, 0.03630183]), array([0.03235894,
0.02828008, 0.04309993, 0.06553365, 0.07559483,
```

0.06634942, 0.06770904, 0.10305913, 0.08687967, 0.08021754,
0.0717879, 0.0792658, 0.07097213, 0.07858599, 0.07559483,
0.06621346, 0.05601631, 0.06376614, 0.05791978, 0.04459551,
0.03399048, 0.02909585, 0.03127124, 0.03412644, 0.03480625,
0.03412644, 0.02828008, 0.03154317, 0.03426241, 0.03793338,
0.039293 , 0.02705642, 0.02760027, 0.02895989, 0.03004759,
0.04133243, 0.04350782, 0.04445955, 0.04622706, 0.03766145,
0.04065262, 0.02651258, 0.03970088, 0.03059143, 0.03263086,
0.03983684, 0.05016995, 0.04731475, 0.03548606, 0.03303875,
0.03915703, 0.04146839, 0.0364378 , 0.02841604, 0.03847723,
0.03698164, 0.0307274 , 0.03467029, 0.03630183, 0.03698164])]
[0.0369816441337418, 0.05411284693098986]

Antes de seguir, hay que tener un concepto muy claro, que es la dimensionalidad. Como hemos visto acabamos de crear una matriz que tiene filas y columnas. Donde cada fila contiene 60 días y cada columna es cada uno de los 60 días, desde el 1 hasta el 60.

Este tipo de estructura, se le conoce como matriz bidimensional, donde tenemos dos dimensiones, las filas y las columnas. Ahora mismo, estamos introduciendo una matriz bidimensional a la red neuronal, porque simplemente estamos utilizando una variable, que es el precio de apertura. Pero si quisiéramos añadir otras variables como, por ejemplo, el precio de otras empresas para que ayuden a la hora de la predicción, tendríamos que añadir otra dimensión a nuestra matriz. Sería una matriz multidimensional, donde tendría varias dimensiones, tantas como variables utilicemos. Esto se conoce como tensor, un tensor no es nada más que una matriz de matrices.

```
#Convert the x_train and y_train to numpy arrays
x_train, y_train = np.array(x_train), np.array(y_train)
10
#Reshape the data
x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))
x_train.shape
```

Con shape podemos comprobar el tamaño del array X_train que hemos creado y crear una dimensión más con el método reshape de numpy. Podemos ver cómo al hacer shape de nuevo, nos aparece la dimensión.

Este proceso es importante entenderlo, por si queremos añadir más variables. Inicialmente vamos a utilizar únicamente, los datos de apertura para la predicción, pero también podríamos utilizar los datos de cierre, los valores de empresas o de otros índices bursátiles. Todo esto es información que puede ayudar a la red neuronal a entender los datos y extraer patrones, pero para este ejemplo solo utilizaremos una variable, el precio de cierre.

Ahora que ya tenemos los datos, en el formato correcto para poder introducirlos en la Red Neuronal Recurrente, veamos cómo se construye una red desde cero.

5.5 Construcción de la Red Neuronal Recurrente

Para programar nosotros mismos la red neuronal desde cero en unos sencillos pasos, utilizaremos la librería keras. Necesitaremos dos métodos, models y layers.

```
#Build the LSTM model
model = Sequential()
model.add(LSTM(50, return_sequences=True, input_shape= (x_train.shape[1], 1)))
model.add(LSTM(50, return_sequences= False))
model.add(Dense(25))
model.add(Dense(1))
```

Con models, importamos la función Sequential, que se encarga de iniciar la red neuronal y nos permitirá añadir de manera secuencial las capas de la red. Una vez iniciada, necesitamos añadirle capas.

El método layers nos permitirá agregar las siguientes capas:

Dense: Para añadir la capa de neuronas artificiales
LSTM: Para añadir la capa de neuronas que tendrán la capacidad de memoria. Esta capa, es lo que convierte a nuestra red neuronal artificial, en una red neuronal recurrente.
Dropout: Esta técnica de regularización, nos permite eliminar conexiones neuronales, para evitar tanto el sobreajuste, como que el algoritmo memorice los datos, en vez de aprender de ellos.
Una vez entendemos las partes de la que se encarga cada capa de la red, solo tenemos que completar el código, para crear una red neuronal recurrente.
Repetimos este proceso cuatro veces, añadiendo 4 capas LSTM con lo que tendremos una red neuronal recurrente, con una profundidad de 4 capas y 50 neuronas por capa. Esto son muchas conexiones neuronales, porque necesitamos que la red tenga muchas neuronas, para que sea

capaz de captar correlaciones entre los datos. Cuanto más profunda sea la red, mayor capacidad tendrá de abstraer patrones más complejos ocultos en el precio.

Compilamos:

```
#Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')
print(model.compile)
```

Una vez tenemos creada nuestra red neuronal con todas sus capas, necesitamos compilar toda la red. La fase de compilación es donde se conectan las capas y se activan los pesos de la red, es como si pusiéramos en marcha la red. Para realizar todo este proceso con Keras, solo necesitaremos la función compile.

En el compilador, especificamos de qué manera se va a optimizar la red y que función de pérdida utilizaremos. Esta función de pérdida nos permite calcular el error de la predicción, en la fase de entrenamiento.

Optimizer: Como ya vimos en la parte del funcionamiento de una red neuronal el optimizador es el que se encarga de optimizar y actualizar los pesos de la red. En el ejemplo anterior del coche, sería el que nos permite coger el feedback de nuestro error y mejorar nuestra decisión para obtener mejores resultados. Pero en este caso para la predicción del índice VIX, utilizaremos el algoritmo adam que es el que mejores resultados nos aporta.

Loss: Cuando la red hace una predicción, cogerá el valor real y lo comparará con la predicción. Para calcular el error de predicción, se utilizará el método error cuadrático medio, ese error, será devuelto a la red, para que haga las actualizaciones pertinentes de los pesos.

5.5.1 Entrenamiento

Ahora solo tenemos que entrenar la red con todos los datos, para que empiece a aprender las relaciones que hay entre ellos, para poder dar predicciones precisas. Lo haremos con la función fit.

```
#Train the model
model.fit(x_train, y_train, batch_size=1, epochs=1)
```

Epochs: Especificamos las veces que la red propagará el error hacia atrás para aprender y hacer mejores predicciones.

Batch_size: Para este proceso de predicción, corrección y propagación hacia atrás, no emplearemos un solo bloque de datos, ya que si no, el entrenamiento sería demasiado pesado para la red, al tener que actualizar los pesos a cada dato.
Con este tamaño de la red, el entrenamiento tan solo tardará algunos minutos, alrededor de veinte en mi caso. Cabe destacar que este proceso se ha hecho en un ordenador con una tarjeta gráfica de alta potencia, ya que los algoritmos de inteligencia artificial corren en la GPU del ordenador.

```
#Create the testing data set
#Create a new array containing scaled values from index 1543 to 2002
test_data = scaled_data[training_data_len - 60: , :]
#Create the data sets x_test and y_test
x_test = []
y_test = dataset[training_data_len:, :]
for i in range(60, len(test_data)):
  x_test.append(test_data[i-60:i, 0])
```

Convertimos esos datos en su formato adecuado

```
#Convert the data to a numpy array
x_test = np.array(x_test)
```

```
#Reshape the data
x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1 ))
```

5.5.2 Predicción

Ahora ya tenemos la red neuronal recurrente entrenada, donde ha aprendido sobre los datos, para poder predecir qué valor tendrá el VIX mañana. Pero antes de realizar predicciones, necesitamos los datos reales que vamos a predecir, para saber la eficiencia real de la red neuronal prediciendo.

```
#Get the models predicted price values
predictions = model.predict(x_test)
predictions = scaler.inverse_transform(predictions)
```

Para la predicción, tendremos que hacer transformaciones al igual que hicimos con los datos de entrenamiento. Cómo utilizamos los 60 días anteriores para predecir el siguiente, eso quiere decir que, si queremos predecir el primer día del mes, utilizaremos los datos de entrenamiento, que son previos a dicho mes. Por lo que necesitamos concatenar los datos de entrenamiento con los de testing.

Gracias a la librería pandas, podemos hacerlo con la función concat. También debemos tener en cuenta, que los datos de entrenamiento están escalados, por lo que tendremos que realizar el mismo procedimiento a los de testing.

Para cada uno de los treinta días del mes, es necesario retroceder para atrás sesenta, por eso es necesaria realizar esta transformación. Necesitamos pasarle a la red en formato vector, los 60 días anteriores de cada día del mes. Para ello utilizaremos un bucle for, con un rango de los días que queremos predecir y cada iteración añadirá a la variable X_test con los 60 días anteriores necesarios que necesita la red, como dato para poder realizar la predicción.

Ahora ya sí que pasamos a realizar la predicción con la función predict, pasando el vector de datos X_test a la red neuronal recurrente.

5.5.3 Visualización y análisis del resultado

Ya hemos finalizado todo el proceso de transformación de los datos, entrenamiento y predicción, ahora toca evaluar los resultados del algoritmo. Con la ayuda de matplotlib, vamos a graficar los resultados y comparar los valores reales con lo que el algoritmo ha predicho.

```
#Plot the data
train = data[:training_data_len]
valid = data[training_data_len:]
valid['Predictions'] = predictions
#Visualize the data
plt.figure(figsize=(16,8))
plt.title('Model')
plt.xlabel('Date', fontsize=18)
plt.ylabel('Close Price USD ($)', fontsize=18)
plt.plot(train['Close'])
plt.plot(valid[['Close', 'Predictions']])
```

```
plt.legend(['Train', 'Val', 'Predictions'], loc='lower right')
plt.show()
```

```
#Get the root mean squared error (RMSE)
rmse=np.sqrt(np.mean(((predictions- y_test)**2)))
rmse
```

```
#Visualize the closing price history
plt.figure(figsize=(16,8))
plt.title('Close Price History')
plt.plot(df['Close'])
plt.xlabel('Date', fontsize=18)
plt.ylabel('Close Price USD ($)', fontsize=18)
plt.show()
```

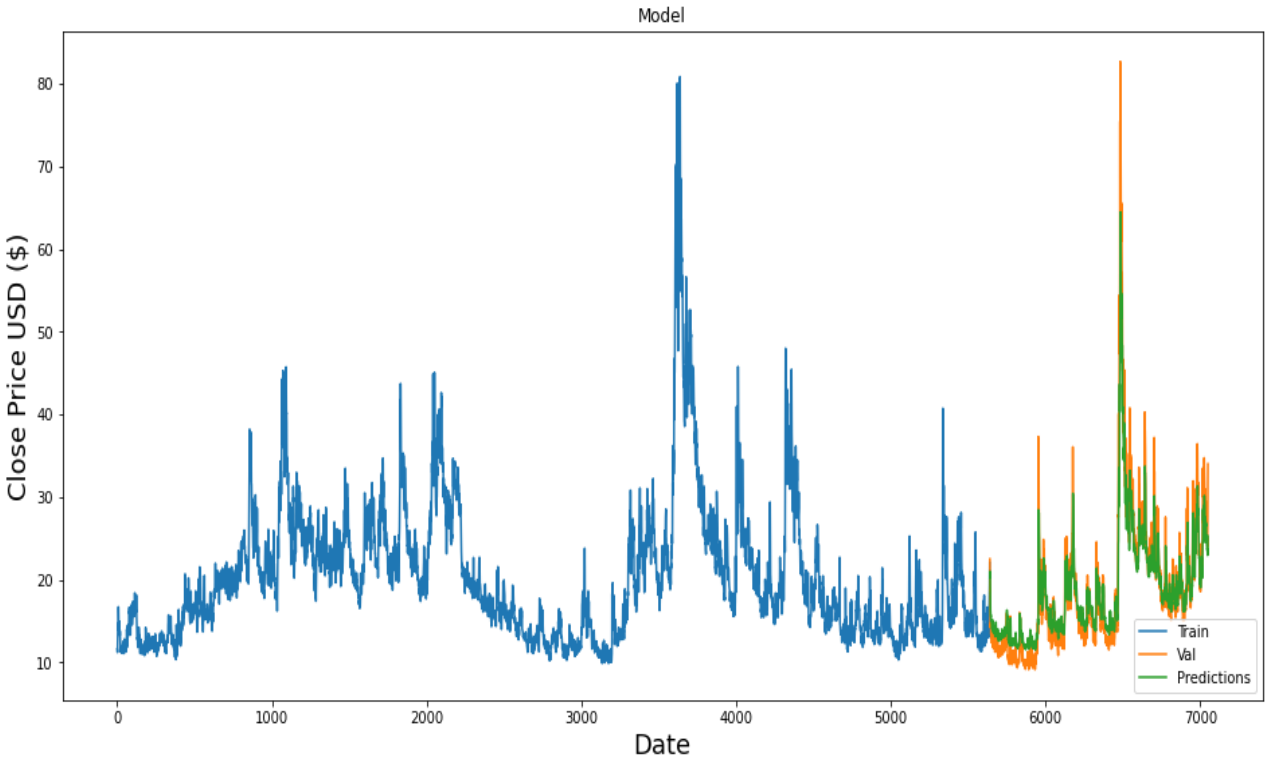


Figura 16. Resultado

La línea naranja corresponde a los valores reales del VIX y la línea verde a la predicción hecha por el algoritmo. Como podemos observar, el algoritmo aproxima bastante bien su predicción al valor real, teniendo un error promedio de 2,8.

A continuación, pediremos al programa los datos de manera numérica.

```
#Show the valid and predicted prices
valid
```

	Close	Predictions
5641	22.510000	20.374233
5642	18.709999	21.029654
5643	18.740000	19.089134
5644	14.380000	18.651354
5645	14.740000	16.283443
...	...	...
7046	24.020000	23.736668
7047	23.959999	23.154179
7048	26.090000	22.960350
7049	27.750000	24.101616
7050	34.020000	25.287573

1410 rows × 2 columns

5.6 Medidas del error

Ahora, comparamos el valor predicho con el valor real para obtener el error en el

```
#Get the root mean squared error (RMSE)
rmse=np.sqrt(np.mean(((predictions- y_test)**2)))
rmse
```

Obteniendo un error de:

2.8661890584950984

Los datos obtenidos en esta predicción del VIX, se encuentran expuestos en el anexo 3. Este programa lo hemos vuelto a realizar con los datos del SPX para obtener la predicción con redes neuronales mostrada en el anexo 4. Estos dos índices nos han permitido reducir el error, gracias a su perfecta correlación (anexo 5) y mejorar la predicción.

5.7 Elección de una estrategia neutral Iron Condor:

Dados los datos obtenidos diseñaremos una estrategia neutral como es el Iron Condor asimétrico. Cabe resaltar que el Iron Condor es una estrategia con un riesgo definido que consiste en dos verticales de crédito, una bull put y una bear call "fuera del dinero" (OTM).

Figura 17. Iron Condor

Recibiremos una prima por vender una bull put OTM y bear call OTM. Creando de esta manera un intervalo de beneficio entre los dos breakeven, donde el objetivo es que el precio de cotización oscile entre nuestros 'strikes' en la fecha de expiración.
De mantenerse, las opciones habrán perdido todo su valor en la fecha de vencimiento, y la prima recibida por la venta en la transacción original se convertirá en nuestro beneficio al extinguirse la posición corta de ambos spreads.

Esta estrategia mencionada, se considera como una estrategia delta-neutral de riesgo definido. Las claves que definen la estrategia son las siguientes:

1. La delta-neutralidad se logra gracias al spread de puts cortos (una posición alcista), con un Delta positivo, junto con un spread de calls cortos (una posición bajista) con un Delta negativo. De esta manera, al vender ambos spreads de manera simultánea, a una cotización determinada, ambos spreads van a oponerse, contrarrestando el Delta positivo de los puts con el Delta negativo de las calls, obteniendo de esta manera un Delta-neutral.

2. El Iron Condor tiene un riesgo definido, esto se debe a que la toma de las posiciones es tomada OTM para seleccionar el riesgo y controlar las máximas pérdidas (max-profit). Sin embargo, el beneficio potencial también se ve reducido ya que la prima recibida por la transacción es menor si se compran las opciones OTM.

Una vez entendido su funcionamiento, es necesario ver las distintas posibilidades con nuestra estrategia si la cotización sube, cae o se mantiene. En esta imagen encontramos en verde la situación ideal y en naranja otras situaciones más perjudiciales. Estos casos son:

A. La cotización del índice sube (nuestro caso), los posibles escenarios que se pueden plantear son:

Figura 18. Las distintas estrategias

- El precio de las call subirá al acercarse a estar ATM (en el dinero), subiendo por lo tanto su valor. Las opciones de menor valor (muy alejadas del dinero "OTM") a medida que se acercan o entran dentro de ATM. Obtenemos por tanto un beneficio por el hecho de vender estas opciones que compramos en un valor menor. Sin embargo, las opciones con posición short serán las que afecten de manera negativa a nuestro balance de beneficio / pérdidas (P/L). Cuando vendemos un spread de calls short a un precio de ejercicio, y ese precio o valor incrementa, estamos obligados a comprarlas a un precio superior, afectando a nuestro balance de forma negativa, siendo un resultado negativo para la estrategia inicialmente planteada.

- Por otro lado, se verá disminuido el precio de las put, a consecuencia de una caída en la cotización, nuestro spread de short puts se aleja de manera exponencial del rango ATM, al estar establecido el precio en las puts, alejándose la cotización alcista. Este escenario significará un resultado positivo en nuestro balance al perder valor.

- El peor de los escenarios posibles, se da cuando la cotización supera el precio fijado tanto de nuestro call corto como de nuestro call largo. En este escenario, nuestro spread dentro del intervalo de ATM, sufrirá por lo tantos pérdidas. Aun así, cabe una pequeña posibilidad de que la cotización aumente, aunque en este caso, la

subida sería ligera, lo cual sería beneficioso para nuestra posición, sobre todo si sucede en un periodo de tiempo extendido. Esto ocurre debido a que las opciones pierden valor a lo largo del tiempo, mediante lo que se conoce como theta decay. En este escenario, el theta decay nos favorece, ya que para nuestra posición es favorable cuanto menor valor tengan las opciones.

- En el posible escenario en el que la cotización se mueva lateralmente, un poco al alza o a la baja, pero siempre dentro de un rango, obtendremos un resultado positivo con esta estrategia. Es por ello que en la figura 17 este escenario se encuentra en color naranja en vez de en rojo, ya que un valor superior de la cotización no implica necesariamente un efecto negativo para nuestro balance, aunque no sería la situación ideal, ofreciendo un menor porcentaje de ganancias que si se mantuviera constante.

B. Cuando la cotización se mantiene constante nos encontramos ante el escenario ideal:

- Esta es la situación idónea, en la que la theta decay juega a nuestro favor, ya que, al vender nuestro spread a un precio superior al actual, y con la bajada de las opciones a lo largo del tiempo, existirá la oportunidad de comprar las mismas opciones a un precio inferior al anterior, consiguiendo beneficio. Es en esta situación en la que el Iron Condor es más eficiente.

C. Si la cotización cae, nos encontramos con la situación opuesta a si sube:

- Una caída de la cotización implica un aumento en el valor de nuestras puts y una pérdida en el valor de las call. Una brusca caída de estas las pondría ATM, pero mientras que esta caída sea relativamente moderada no suponen grandes pérdidas y entra dentro de una volatilidad esperada. Si se mantiene dentro del rango establecido en la fecha de vencimiento, las opciones perderán todo su valor y la prima nos dejaría con beneficio.

5.8 Desarrollo del iron condor

Una vez entendido el funcionamiento del Iron Condor, es hora de utilizar ThinkorSwim.

Para ello debemos tener en cuenta que los valores están ordenados en orden ascendente, quedando en la columna de las calls los valores OTM por encima en azul y los valores ITM por debajo en negro. Mientras que en la columna de las puts los valores ITM se encuentran por encima en azul y los valores OTM por debajo en negro, como podemos observar en la siguiente figura (19).

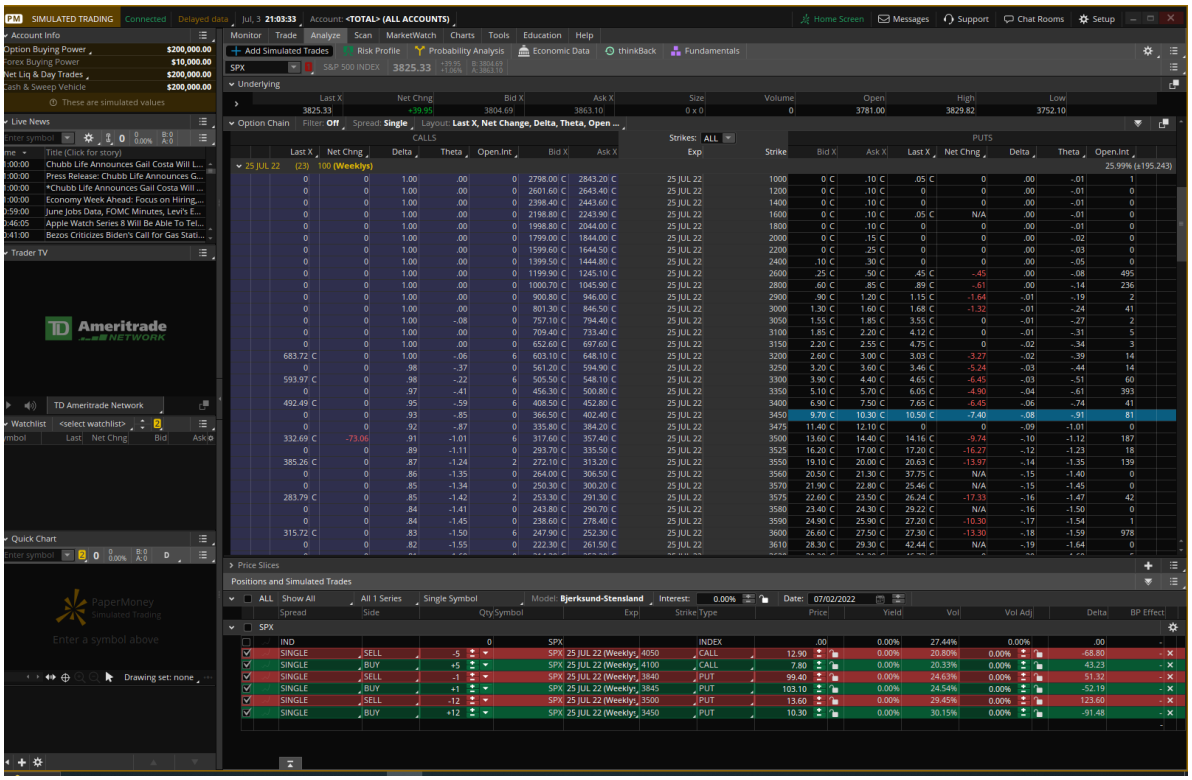


Figura 19. Pantalla de inicio

A continuación, es hora de seleccionar y da valor a las bear call, bear put, long call, long put, short call y short put para formar nuestro Iron Condor deseado.

Para ello, vamos primero a seleccionar las dos posiciones de la bear call (la vertical superior) situada sobre el valor de la predicción de 4100, la long call y 4050 la short call. Como podemos observar en la parte inferior de la figura 19 donde están las posiciones básicas de la vertical, el

short call tiene un color rojo puesto que es nuestra posición de crédito o vendida mientras que la long call siempre aparecerá en verde como posición comprada.

La línea azul de la vertical representa los puntos del gráfico de riesgo en el momento de la expiración del spread, mientras que la línea morada nos indica los puntos de la función en el momento presente o actual de nuestra posición. También en amarillo podemos observar la probabilidad estimada de acierto en nuestro spread. Cabe recordar que en el eje de abscisas tenemos los diferentes valores del subyacente considerado en nuestro caso el SPX mientras que en el eje de ordenadas podemos analizar el beneficio, la pérdida y el punto de equilibrio de nuestra estrategia.

Figura 20. Formación del Iron Condor 1

Una vez formada la bear call, es hora de crear la bear put cuya función es prestar una cobertura de volatilidad a lo que sería un Iron Condor tradicional. Este put, debe situarse ligeramente OTM, seleccionando la short put en el strike 3840 y la long put en el 3845. La bear call colocada en el interior del Iron Condor nos permite tener un spread con menor volatilidad a un coste relativamente alto que tenemos que financiar con los contratos de la Bull Put.

Figura 21. Formación del Iron Condor 2

El penúltimo paso es seleccionar la bull put que nos va a generar suficiente crédito para pagar la bear put. Para ello seleccionamos el short put de la vertical sobre nueve o diez deltas, incluyendo la variable delta dentro del menú de las posibles opciones como se puede ver en la figura 22.

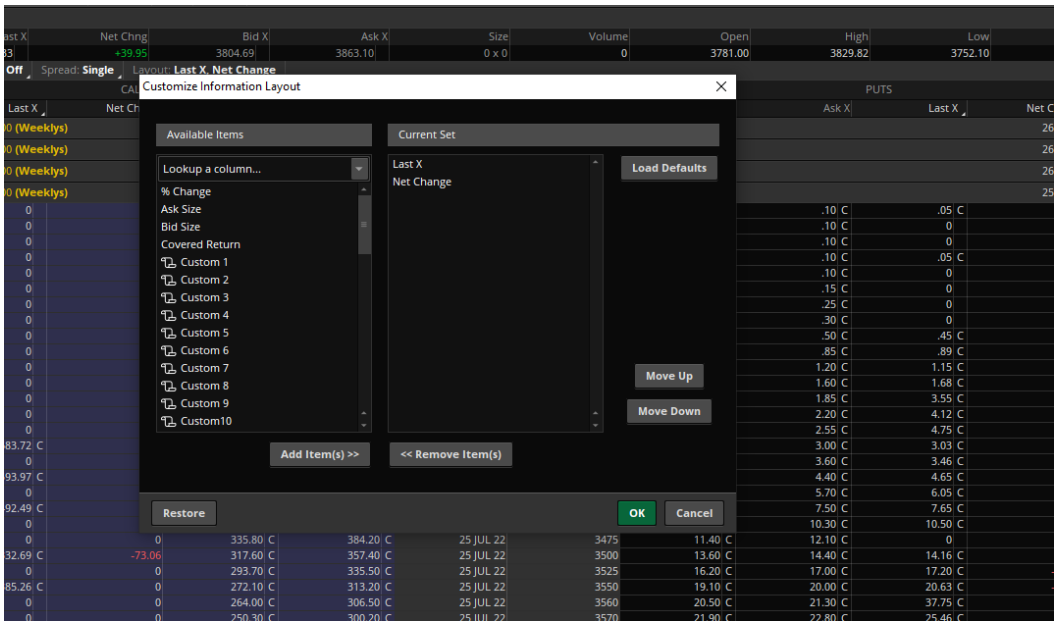


Figura 22. Menú de especificaciones

Por último, deberíamos neutralizar delta situando el short call de la bear call en unos rangos de valores entre nueve y doce. Obteniendo el Iron Condor final de la figura 24.

El resultado de este Iron Condor sería vender entre doce y quince bull puts para cubrir el crédito objetivo. Como la bull put tiene un crédito de 3,30 a nuestro favor necesitaríamos doce contratos para cubrir algo más de los 3700 de la bear put. 3,3*12*100=3960.

Como se puede ver en el pie de la foto, la diferencia entre el short put y la long put es de un débito de 370 por contrato.

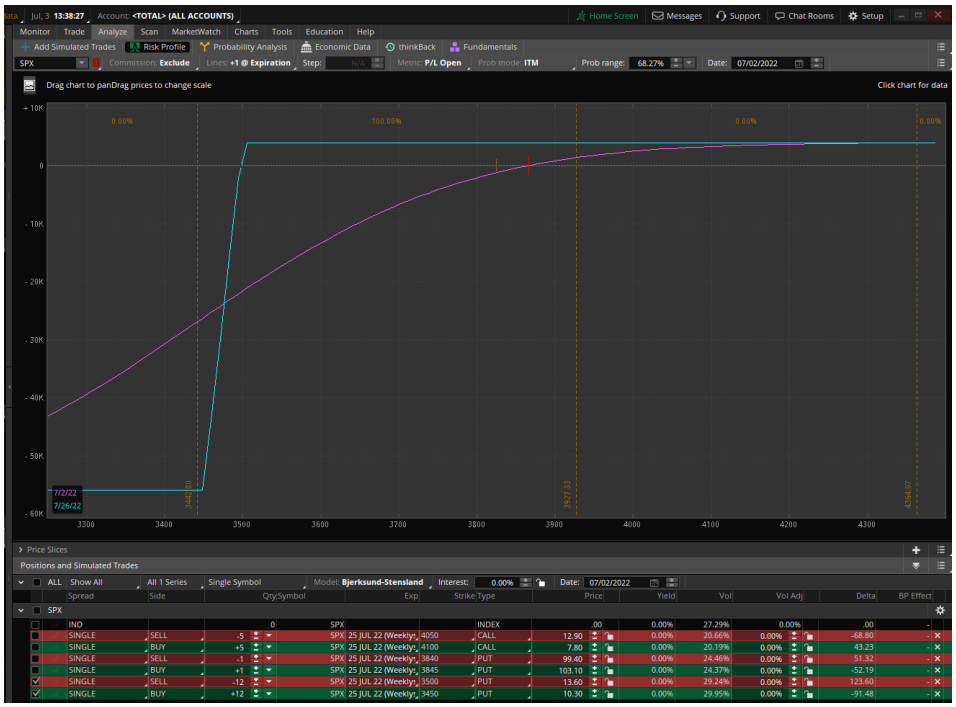


Figura 23. Formación del Iron Condor 3

El resultado del conjunto de estos tres verticales mostradas en las figuras 22, 20 y 19 se ve reflejada en la siguiente foto adjunta.

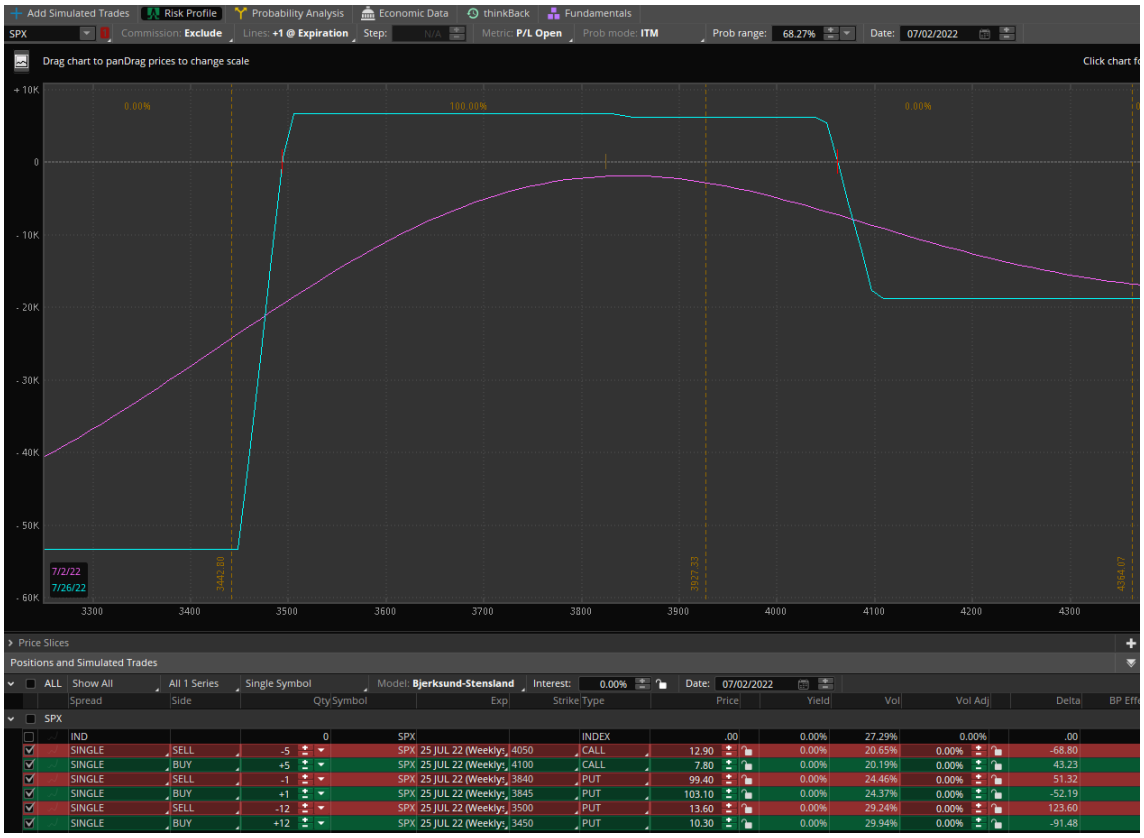


Figura 24. Formación Final del Iron Condor

Sería un spread con una probabilidad de ganancia cercana al 100 % como nos indica el gráfico de riesgo. Además, como sería un spread de crédito porque así lo hemos calculado para el coste de las bear call, bullput y bearput, en principio no tendríamos que desembolsar recursos. Nos daría tiempo además a cerrarlo antes de llegar al valor de su predicción porque dada la naturaleza del spread nunca podríamos llegar a la fecha de expiración, 15 días antes prácticamente theta habría llegado a su máximo time decay, por la propia naturaleza de la estrategia.

Capítulo 6. CONCLUSIÓN, TRABAJO FUTURO y DESARROLLO SOSTENIBLE

6.1 Trabajo Futuro

Tras la elaboración de este TFG, podemos establecer una guía de trabajo futuro que permita complementar y mejorar los modelos y resultados propuestos reduciendo el error lo máximo posible.

Para lograr el objetivo inicial de este TFG, será necesario profundizar en los nuevos conocimientos y desarrollos para rediseñar el número de units en las capas de arquitectura y mejorar los parámetros y la estructura.

Otra de las mejoras necesarias en las que trabajar sería el ajustar el batch size, aumentando el número de variables descartadas debido a la falta de potencia de mi ordenador y a como afectaban a la eficacia.

Por último, creo que sería interesante aplicar este programa en otros ámbitos para probar su eficacia. Así como implementarlo en el análisis de otro tipo de datos de la vida cotidiana como señales de tráfico o para desarrollar un sistema de conducción autónoma.

6.2 Conclusión

La conclusión que sacamos de este TFG es que es posible analizar y procesar una gran variedad de datos para detectar patrones y tendencias mediante redes neuronales recurrentes para predecir valores futuros con un error operativo y diseñar estrategias de ingeniería financiera en base a ello, spread de opciones.

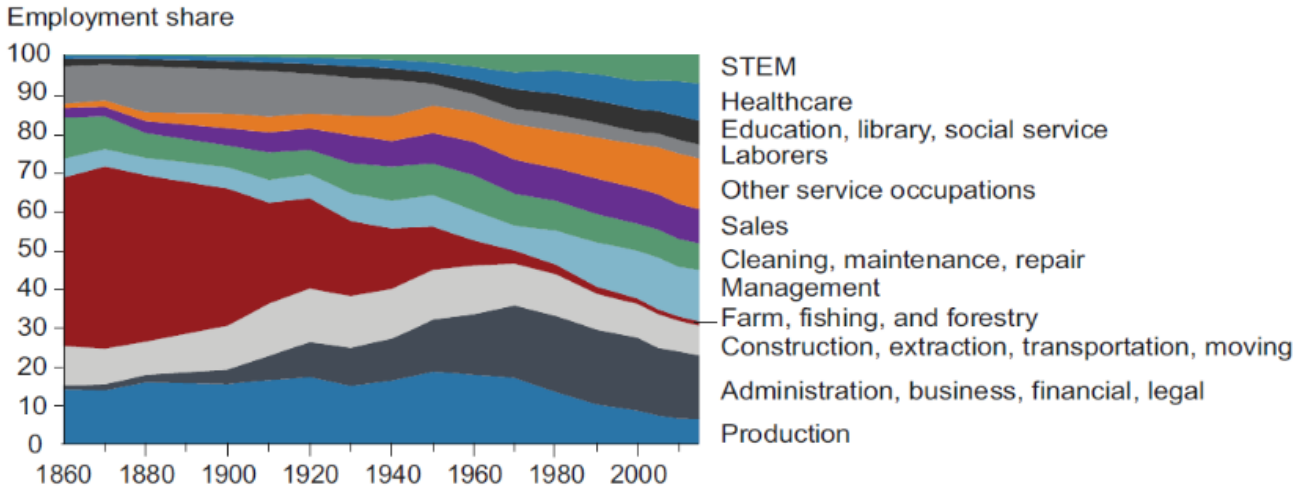
6.2 Desarrollo sostenible

En relación con los Objetivos de Desarrollo Sostenible, este trabajo permite avanzar en el progreso de las cooperativas africanas de cultivos, susceptibles de cotizar en los mercados de opciones y futuros, como el cacao.

Posibilitando a estas cooperativas acceder a un precio y a una logística detallada en las especificaciones de los contratos de futuro, donde con la adecuada formación en las normas del mismo se consigue la eliminación de intermediarios, mejores precios y la creación de un ecosistema de industrias alrededor del producto.

Dentro de este ecosistema de industrias, podemos incluir: la logística de transporte, instalaciones de almacenamiento, regulaciones fitosanitarias, técnicos en los mercados internacionales y un largo etc.

Capítulo 7. Anexo



Anexo 1: Cambios en la estructura ocupacional en EEUU: 1860 a 2015. Fuente: Elvery (2019: 2).

```
import keras
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense, LSTM
import matplotlib.pyplot as plt
```

```
[ ] !pip install yfinance

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting yfinance
  Downloading yfinance-0.1.72-py2.py3-none-any.whl (27 kB)
Collecting requests>=2.26
  Downloading requests-2.28.1-py3-none-any.whl (62 kB)
     |████████████████████████████████| 62 kB 1.1 MB/s
Requirement already satisfied: pandas>=0.24.0 in /usr/local/lib/python3.7/dist-packages (from yfinance) (1.3.5)
Requirement already satisfied: numpy>=1.15 in /usr/local/lib/python3.7/dist-packages (from yfinance) (1.21.6)
Requirement already satisfied: multitasking>=0.0.7 in /usr/local/lib/python3.7/dist-packages (from yfinance) (0.0.10)
Collecting lxml>=4.5.1
  Downloading lxml-4.9.0-cp37-cp37m-manylinux_2_17_x86_64.manylinux2014_x86_64.manylinux_2_24_x86_64.whl (6.4 MB)
     |████████████████████████████████| 6.4 MB 11.0 MB/s
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dist-packages (from pandas>=0.24.0->yfinance) (2.8.2)
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-packages (from pandas>=0.24.0->yfinance) (2022.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (from python-dateutil>=2.7.3->pandas>=0.24.0->yfinance) (1.15.0)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.7/dist-packages (from requests>=2.26->yfinance) (1.24.3)
Requirement already satisfied: charset-normalizer<3,>=2 in /usr/local/lib/python3.7/dist-packages (from requests>=2.26->yfinance) (2.0.12)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests>=2.26->yfinance) (2.10)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages (from requests>=2.26->yfinance) (2022.6.15)
```

```
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages (from requests>=2.26->yfinance) (2022.6.15)
[ ] Installing collected packages: requests, lxml, yfinance
      Attempting uninstall: requests
        Found existing installation: requests 2.23.0
        Uninstalling requests-2.23.0:
          Successfully uninstalled requests-2.23.0
      Attempting uninstall: lxml
        Found existing installation: lxml 4.2.6
        Uninstalling lxml-4.2.6:
          Successfully uninstalled lxml-4.2.6
    ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour is the source of the followir
    google-colab 1.0.0 requires requests~=2.23.0, but you have requests 2.28.1 which is incompatible.
    datascience 0.10.6 requires folium==0.2.1, but you have folium 0.8.3 which is incompatible.
    Successfully installed lxml-4.9.0 requests-2.28.1 yfinance-0.1.72
```

```
[ ] pip install nasdaq-data-link

    Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
    Collecting nasdaq-data-link
      Downloading Nasdaq_Data_Link-1.0.2-py2.py3-none-any.whl (28 kB)
    Collecting inflection>=0.3.1
      Downloading inflection-0.5.1-py2.py3-none-any.whl (9.5 kB)
    Requirement already satisfied: numpy>=1.8 in /usr/local/lib/python3.7/dist-packages (from nasdaq-data-link) (1.21.6)
    Requirement already satisfied: requests>=2.7.0 in /usr/local/lib/python3.7/dist-packages (from nasdaq-data-link) (2.23.0)
    Requirement already satisfied: python-dateutil in /usr/local/lib/python3.7/dist-packages (from nasdaq-data-link) (2.8.2)
    Requirement already satisfied: more-itertools in /usr/local/lib/python3.7/dist-packages (from nasdaq-data-link) (8.13.0)
    Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from nasdaq-data-link) (1.15.0)
    Requirement already satisfied: pandas>=0.14 in /usr/local/lib/python3.7/dist-packages (from nasdaq-data-link) (1.3.5)
    Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-packages (from pandas>=0.14->nasdaq-data-link) (2022.1)
```

```
    Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests>=2.7.0->nasdaq-data-link) (2.10)
[ ] Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.7/dist-packages (from requests>=2.7.0->nasdaq-data-lir
    Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from requests>=2.7.0->nasdaq-data-link) (3.0.4)
    Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages (from requests>=2.7.0->nasdaq-data-link) (2022.6.15)
    Installing collected packages: inflection, nasdaq-data-link
    Successfully installed inflection-0.5.1 nasdaq-data-link-1.0.2
```

```
[ ] import math
    import pandas_datareader as web
    import numpy as np
    import pandas as pd
```

```
[ ] Defining some constants for data mining
```

```
[ ] from google.colab import files
    files.upload()
```

```
    Choose Files  No file chosen    Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
    Saving ^VIX (1).csv to ^VIX (1).csv
    {'^VIX (1).csv': b'Date,Open,High,Low,Close,Adj Close,Volume\n1994-06-14,11.620000,11.930000,11.380000,11.600000,11.600000,0\n1994-06-
    15,11.360000,12.390000,11.210000,11.520000,11.520000,0\n1994-06-16,11.360000,11.890000,11.200000,11.220000,11.220000,0\n1994-06-
    17,10.520000,12.330000,10.480000,12.310000,12.310000,0\n1994-06-20,13.770000,14.210000,13.380000,13.960000,13.960000,0\n1994-06-
    21,13.700000,14.990000,13.600000,14.700000,14.700000,0\n1994-06-22,14.750000,14.760000,14.020000,14.020000,14.020000,0\n1994-06-
    23,13.310000,14.280000,13.290000,14.120000,14.120000,0\n1994-06-24,15.350000,17.270000,15.310000,16.719999,16.719999,0\n1994-06-
    27,17.360001,17.709999,15.510000,15.530000,15.530000,0\n1994-06-28,13.390000,15.750000,13.390000,15.040000,15.040000,0\n1994-06-
```

```
29,14.630000,14.750000,14.260000,14.420000,14.420000,0\n1994-06-30,13.920000,15.430000,13.860000,14.970000,14.970000,0\n1994-07-
01,14.890000,15.260000,14.360000,14.360000,14.360000,0\n1994-07-05,14.840000,15.480000,14.600000,14.920000,14.920000,0\n1994-07-
06,15.120000,15.630000,14.440000,14.700000,14.700000,0\n1994-07-07,13.240000,14.640000,13.240000,14.010000,14.010000,0\n1994-07-
08,14.880000,14.880000,12.850000,13.260000,13.260000,0\n1994-07-11,13.300000,14.610000,13.030000,13.830000,13.830000,0\n1994-07-
12,13.900000,14.160000,13.040000,13.400000,13.400000,0\n1994-07-13,12.940000,12.980000,12.170000,12.420000,12.420000,0\n1994-07-
14,12.690000,12.880000,11.390000,11.640000,11.640000,0\n1994-07-15,11.130000,11.540000,11.000000,11.280000,11.280000,0\n1994-07-
18,11.490000,11.680000,11.160000,11.440000,11.440000,0\n1994-07-19,11.490000,11.910000,11.000000,11.650000,11.650000,0\n1994-07-
20,11.510000,12.130000,11.500000,11.700000,11.700000,0\n1994-07-21,12.490000,12.560000,11.620000,11.650000,11.650000,0\n1994-07-
22,10.960000,11.890000,10.960000,11.220000,11.220000,0\n1994-07-25,11.870000,12.380000,11.440000,11.460000,11.460000,0\n1994-07-
26,11.780000,12.020000,11.650000,11.660000,11.660000,0\n1994-07-27,11.930000,12.260000,11.860000,11.930000,11.930000,0\n1994-07-
28,11.990000,12.150000,11.690000,12.030000,12.030000,0\n1994-07-29,11.230000,11.740000,11.050000,11.130000,11.130000,0\n1994-08-
01,11.710000,11.760000,11.130000,11.170000,11.170000,0\n1994-08-02,10.910000,11.430000,10.750000,11.270000,11.270000,0\n1994-08-
03,11.420000,11.610000,11.150000,11.350000,11.350000,0\n1994-08-04,11.500000,12.300000,11.460000,12.180000,12.180000,0\n1994-08-
05,12.850000,12.980000,12.330000,12.340000,12.340000,0\n1994-08-08,11.710000,12.700000,11.690000,12.410000,12.410000,0\n1994-08-
09,11.120000,13.320000,11.120000,12.540000,12.540000,0\n1994-08-10,12.610000,12.630000,11.660000,11.910000,11.910000,0\n1994-08-
11,11.550000,12.890000,11.270000,12.130000,12.130000,0\n1994-08-12,11.980000,12.130000,11.030000,11.090000,11.090000,0\n1994-08-
15,11.920000,12.730000,11.650000,12.060000,12.060000,0\n1994-08-16,11.720000,12.790000,11.170000,11.390000,11.390000,0\n1994-08-
17,11.170000,11.620000,11.160000,11.540000,11.540000,0\n1994-08-18,11.760000,12.600000,11.650000,12.070000,12.070000,0\n1994-08-
19,12.710000,13.340000,12.440000,12.830000,12.830000,0\n1994-08-22,12.870000,13.230000,12.550000,12.620000,12.620000,0\n1994-08-
23,11.780000,11.850000,11.140000,11.750000,11.750000,0\n1994-08-24,12.030000,12.050000,11.560000,11.570000,11.570000,0\n1994-08-
25,11.620000,12.270000,11.520000,12.020000,12.020000,0\n1994-08-26,11.160000,12.840000,11.020000,12.190000,12.190000,0\n1994-08-
29,12.810000,12.810000,11.660000,11.820000,11.820000,0\n1994-08-30,11.730000,11.810000,11.220000,11.230000,11.230000,0\n1994-08-
31,11.180000,12.430000,11.180000,11.970000,11.970000,0\n1994-09-01,12.170000,12.660000,11.860000,11.860000,11.860000,0\n1994-09-
02,11.210000,11.730000,11.170000,11.400000,11.400000,0\n1994-09-06,11.500000,12.240000,11.480000,11.690000,11.690000,0\n1994-09-
07,11.830000,12.490000,11.600000,11.810000,11.810000,0\n1994-09-08,12.010000,12.030000,11.400000,11.860000,11.860000,0\n1994-09-
09,11.620000,13.560000,11.620000,13.120000,13.120000,0\n1994-09-12,13.470000,14.910000,13.440000,14.010000,14.010000,0\n1994-09-
13,12.960000,14.590000,12.960000,13.770000,13.770000,0\n1994-09-14,13.950000,14.380000,12.920000,13.090000,13.090000,0\n1994-09-
15,11.870000,13.000000,11.430000,11.880000,11.880000,0\n1994-09-16,12.060000,13.120000,11.750000,12.800000,12.800000,0\n1994-09-
19,12.890000,13.260000,12.710000,13.060000,13.060000,0\n1994-09-20,13.350000,14.450000,13.350000,14.450000,14.450000,0\n1994-09-
21,13.650000,16.080000,13.630000,14.490000,14.490000,0\n1994-09-22,13.560000,13.940000,13.430000,13.890000,13.890000,0\n1994-09-
23,13.790000,14.870000,13.790000,14.530000,14.530000,0\n1994-09-26,14.940000,15.010000,14.520000,14.560000,14.560000,0\n1994-09-
27,14.900000,15.130000,14.130000,14.130000,14.130000,0\n1994-09-28,13.620000,13.820000,12.970000,13.250000,13.250000,0\n1994-09-
29,13.550000,14.400000,13.230000,13.970000,13.970000,0\n1994-09-30,13.330000,14.560000,12.850000,14.280000,14.280000,0\n1994-10-
03,15.690000,16.400000,15.420000,15.440000,15.440000,0\n1994-10-04,14.080000,16.969999,14.080000,16.660000,16.660000,0\n1994-10-
05,16.260000,16.719999,15.000000,15.920000,15.920000,0\n1994-10-06,15.650000,16.370001,15.300000,16.240000,16.240000,0\n1994-10-
07,15.310000,15.730000,14.850000,14.920000,14.920000,0\n1994-10-10,14.260000,15.620000,14.260000,14.720000,14.720000,0\n1994-10-
11,13.930000,14.160000,13.400000,13.550000,13.550000,0\n1994-10-12,13.660000,15.210000,13.590000,14.510000,14.510000,0\n1994-10-
13,12.750000,14.970000,12.750000,14.900000,14.900000,0\n1994-10-14,14.330000,14.330000,13.180000,13.320000,13.320000,0\n1994-10-
17,14.250000,15.310000,14.130000,14.760000,14.760000,0\n1994-10-18,14.400000,15.600000,14.380000,14.590000,14.590000,0\n1994-10-
19,13.910000,15.330000,13.870000,15.130000,15.130000,0\n1994-10-20,15.500000,16.450001,15.500000,15.990000,15.990000,0\n1994-10-
21,16.139999,16.459999,15.810000,15.910000,15.910000,0\n1994-10-24,15.970000,17.219999,15.870000,16.969999,16.969999,0\n1994-10-
25,17.180000,17.180000,16.110001,16.129999,16.129999,0\n1994-10-26,15.700000,16.030001,15.530000,15.700000,15.700000,0\n1994-10-
27,15.200000,15.900000,15.200000,15.670000,15.670000,0\n1994-10-28,15.110000,15.220000,14.560000,14.560000,14.560000,0\n1994-10-
31,14.730000,15.170000,14.200000,14.560000,14.560000,0\n1994-11-01,14.610000,15.180000,14.610000,14.840000,14.840000,0\n1994-11-
02,15.280000,15.470000,14.410000,15.340000,15.340000,0\n1994-11-03,14.320000,16.430000,14.320000,15.580000,15.580000,0\n1994-11-
04,14.990000,16.870001,14.870000,16.750000,16.750000,0\n1994-11-07,16.879999,17.870001,16.820000,17.370001,17.370001,0\n1994-11-
08,15.570000,16.920000,15.540000,16.299999,16.299999,0\n1994-11-09,15.950000,17.330000,15.830000,16.420000,16.420000,0\n1994-11-
10,16.110001,16.740000,15.670000,16.000000,16.000000,0\n1994-11-11,15.070000,16.910000,14.990000,16.500000,16.500000,0\n1994-11-
14,16.959999,16.990000,16.680000,16.680000,16.680000,0\n1994-11-15,14.750000,17.010000,14.750000,16.660000,16.660000,0\n1994-11-
16,15.730000,16.030001,15.560000,15.560000,15.560000,0\n1994-11-17,15.020000,16.139999,14.990000,15.910000,15.910000,0\n1994-11-
18,15.770000,16.270000,15.610000,16.000000,16.000000,0\n1994-11-21,15.470000,16.379999,15.280000,16.129999,16.129999,0\n1994-11-
22,16.570000,17.389999,16.020000,17.240000,17.240000,0\n1994-11-23,18.270000,19.570000,18.270000,18.410000,18.410000,0\n1994-11-
25,17.639999,18.510000,17.230000,17.440001,17.440001,0\n1994-11-28,17.090000,17.620001,16.830000,16.830000,16.830000,0\n1994-11-
29,16.010000,16.760000,15.650000,16.150000,16.150000,0\n1994-11-30,15.990000,16.870001,15.530000,15.950000,15.950000,0\n1994-12-
01,15.490000,17.040001,15.420000,16.690001,16.690001,0\n1994-12-02,16.110001,17.139999,16.100000,16.549999,16.549999,0\n1994-12-
05,15.880000,17.030001,15.690000,16.100000,16.100000,0\n1994-12-06,15.800000,16.860001,15.660000,16.049999,16.049999,0\n1994-12-
07,16.610001,17.639999,16.170000,16.280001,16.280001,0\n1994-12-08,16.129999,18.160000,16.120001,18.150000,18.150000,0\n1994-12-
09,17.000000,17.219999,16.010000,16.040001,16.040001,0\n1994-12-12,17.440001,17.440001,15.460000,15.460000,15.460000,0\n1994-12-
13,15.040000,15.170000,14.330000,14.480000,14.480000,0\n1994-12-14,15.090000,15.110000,12.660000,13.030000,13.030000,0\n1994-12-
15,13.790000,13.790000,12.670000,12.870000,12.870000,0\n1994-12-16,12.870000,13.130000,11.870000,12.790000,12.790000,0\n1994-12-
19,13.020000,13.670000,13.020000,13.020000,13.020000,0\n1994-12-20,12.940000,13.480000,12.780000,12.980000,12.980000,0\n1994-12-
```

30,11.070000,11.510000,10.750000,11.380000,11.380000,0\n1995-07-03,11.360000,12.550000,11.230000,11.570000,11.570000,0\n1995-07-
05,11.860000,12.390000,11.190000,12.100000,12.100000,0\n1995-07-06,12.350000,12.390000,11.430000,11.520000,11.520000,0\n1995-07-
07,11.760000,12.110000,11.720000,11.770000,11.770000,0\n1995-07-10,11.190000,12.870000,10.930000,12.190000,12.190000,0\n1995-07-
11,11.970000,13.060000,11.870000,12.250000,12.250000,0\n1995-07-12,12.250000,12.750000,12.070000,12.310000,12.310000,0\n1995-07-
13,11.130000,13.300000,10.920000,12.560000,12.560000,0\n1995-07-14,12.730000,12.960000,11.990000,12.080000,12.080000,0\n1995-07-
17,12.350000,12.690000,12.220000,12.260000,12.260000,0\n1995-07-18,12.550000,13.050000,12.450000,12.680000,12.680000,0\n1995-07-
19,13.480000,15.260000,12.990000,13.490000,13.490000,0\n1995-07-20,13.460000,13.800000,12.770000,12.780000,12.780000,0\n1995-07-
21,11.940000,13.270000,11.730000,12.370000,12.370000,0\n1995-07-24,12.640000,12.770000,12.390000,12.550000,12.550000,0\n1995-07-
25,12.730000,13.050000,12.540000,12.770000,12.770000,0\n1995-07-26,12.850000,13.670000,12.850000,13.180000,13.180000,0\n1995-07-
27,13.340000,13.350000,13.020000,13.180000,13.180000,0\n1995-07-28,13.050000,13.820000,13.030000,13.180000,13.180000,0\n1995-07-
31,13.220000,13.870000,13.220000,13.490000,13.490000,0\n1995-08-01,13.440000,14.150000,13.390000,13.560000,13.560000,0\n1995-08-
02,13.280000,13.980000,13.090000,13.660000,13.660000,0\n1995-08-03,14.460000,14.490000,13.760000,13.780000,13.780000,0\n1995-08-
04,12.900000,13.810000,12.830000,13.210000,13.210000,0\n1995-08-07,13.620000,13.990000,13.060000,13.100000,13.100000,0\n1995-08-
08,13.250000,13.310000,13.060000,13.060000,13.060000,0\n1995-08-09,12.780000,13.020000,12.600000,12.750000,12.750000,0\n1995-08-
10,12.870000,13.350000,12.570000,13.000000,13.000000,0\n1995-08-11,12.450000,13.340000,12.450000,12.900000,12.900000,0\n1995-08-
14,12.690000,14.350000,12.670000,13.240000,13.240000,0\n1995-08-15,12.310000,12.760000,12.270000,12.350000,12.350000,0\n1995-08-
16,11.600000,12.590000,11.470000,12.450000,12.450000,0\n1995-08-17,12.170000,12.830000,12.090000,12.130000,12.130000,0\n1995-08-
18,11.470000,12.270000,11.360000,12.030000,12.030000,0\n1995-08-21,12.080000,13.780000,11.890000,13.420000,13.420000,0\n1995-08-
22,12.540000,12.980000,12.430000,12.600000,12.600000,0\n1995-08-23,11.820000,13.370000,11.800000,13.170000,13.170000,0\n1995-08-
24,13.310000,13.520000,12.860000,12.940000,12.940000,0\n1995-08-25,12.230000,12.690000,12.110000,12.330000,12.330000,0\n1995-08-
28,12.150000,13.200000,12.050000,12.480000,12.480000,0\n1995-08-29,11.680000,13.800000,11.630000,12.650000,12.650000,0\n1995-08-
30,11.770000,12.420000,11.740000,12.030000,12.030000,0\n1995-08-31,11.610000,11.950000,11.360000,11.520000,11.520000,0\n1995-09-
01,11.150000,11.680000,10.830000,11.290000,11.290000,0\n1995-09-05,11.440000,12.030000,11.280000,11.650000,11.650000,0\n1995-09-
06,10.310000,12.080000,10.190000,11.650000,11.650000,0\n1995-09-07,10.650000,12.530000,10.650000,11.850000,11.850000,0\n1995-09-
08,11.010000,11.940000,10.490000,11.160000,11.160000,0\n1995-09-11,11.790000,12.090000,11.310000,11.510000,11.510000,0\n1995-09-
12,11.670000,12.150000,11.320000,11.460000,11.460000,0\n1995-09-13,11.660000,12.000000,11.180000,11.350000,11.350000,0\n1995-09-
14,11.290000,11.930000,10.870000,11.100000,11.100000,0\n1995-09-15,11.130000,11.840000,11.000000,11.580000,11.580000,0\n1995-09-
18,12.380000,12.720000,12.300000,12.340000,12.340000,0\n1995-09-19,12.500000,13.310000,12.400000,12.670000,12.670000,0\n1995-09-
20,12.750000,12.810000,12.350000,12.350000,12.350000,0\n1995-09-21,11.950000,13.180000,11.890000,12.500000,12.500000,0\n1995-09-
22,12.790000,12.940000,12.370000,12.460000,12.460000,0\n1995-09-25,12.330000,13.430000,12.310000,13.220000,13.220000,0\n1995-09-
26,12.210000,13.230000,12.080000,12.900000,12.900000,0\n1995-09-27,12.730000,13.950000,12.480000,12.900000,12.900000,0\n1995-09-

```
[ ]  #Store The data
     df=pd.read_csv('^VIX (1).csv')
     #show the data
     df
```

	Date	Open	High	Low	Close	Adj Close	Volume
0	1994-06-14	11.620000	11.930000	11.380000	11.600000	11.600000	0
1	1994-06-15	11.360000	12.390000	11.210000	11.520000	11.520000	0
2	1994-06-16	11.360000	11.890000	11.200000	11.220000	11.220000	0
3	1994-06-17	10.520000	12.330000	10.480000	12.310000	12.310000	0
4	1994-06-20	13.770000	14.210000	13.380000	13.960000	13.960000	0
...	...	...	...	...	...	...	...
7046	2022-06-07	25.540001	26.240000	23.879999	24.020000	24.020000	0
7047	2022-06-08	24.370001	24.780001	23.740000	23.959999	23.959999	0
7048	2022-06-09	24.290001	26.240000	23.820000	26.090000	26.090000	0
7049	2022-06-10	26.260000	29.629999	26.049999	27.750000	27.750000	0
7050	2022-06-13	31.370001	35.049999	31.290001	34.020000	34.020000	0

7051 rows × 7 columns

```
[ ]  #Get the number of rows and columns in the data set
     df.shape
```

```
     (7051, 7)
```

```
[ ]  #Visualize the closing price history
     plt.figure(figsize=(16,8))
     plt.title('Close Price History')
     plt.plot(df['Close'])
     plt.xlabel('Date', fontsize=18)
     plt.ylabel('Close Price USD ($)', fontsize=18)
     plt.show()
```

```
[ ]  #Create a new dataframe with only the 'Close column
     data = df.filter(['Close'])
     #Convert the dataframe to a numpy array
     dataset = data.values
     #Get the number of rows to train the model on
     training_data_len = math.ceil( len(dataset) * .8 )
     training_data_len
```

```
     5641
```

```
[ ]  #Scale the data
     scaler = MinMaxScaler(feature_range=(0,1))
     scaled_data = scaler.fit_transform(dataset)

     scaled_data
```

```
     array([[0.03344663],
            [0.03235894],
            [0.02828008],
            ...,
            [0.23045547],
            [0.25302515],
            [0.33827327]])
```

```
[ ]  #Create the training data set
```

```
[ ]  #Create the training data set
     #Create the scaled training data set
     train_data = scaled_data[0:training_data_len , :]
     #Split the data into x_train and y_train data sets
     x_train = []
     y_train = []

     for i in range(60, len(train_data)):
       x_train.append(train_data[i-60:i, 0])
       y_train.append(train_data[i, 0])
       if i<= 61:
         print(x_train)
         print(y_train)
         print()
```

```
[array([0.03344663, 0.03235894, 0.02828008, 0.04309993, 0.06553365,
       0.07559483, 0.06634942, 0.06770904, 0.10305913, 0.08687967,
       0.08021754, 0.0717879 , 0.0792658 , 0.07097213, 0.07858599,
       0.07559483, 0.06621346, 0.05601631, 0.06376614, 0.05791978,
       0.04459551, 0.03399048, 0.02909585, 0.03127124, 0.03412644,
       0.03480625, 0.03412644, 0.02828008, 0.03154317, 0.03426241,
       0.03793338, 0.039293  , 0.02705642, 0.02760027, 0.02895989,
       0.03004759, 0.04133243, 0.04350782, 0.04445955, 0.04622706,
       0.03766145, 0.04065262, 0.02651258, 0.03970088, 0.03059143,
       0.03263086, 0.03983684, 0.05016995, 0.04731475, 0.03548606,
       0.03303875, 0.03915703, 0.04146839, 0.0364378 , 0.02841604,
       0.03847723, 0.03698164, 0.0307274 , 0.03467029, 0.03630183])]
[0.0369816441337418]
```

```
]  [0.0369816441337418]

   [array([0.03344663, 0.03235894, 0.02828008, 0.04309993, 0.06553365,
          0.07559483, 0.06634942, 0.06770904, 0.10305913, 0.08687967,
          0.08021754, 0.0717879 , 0.0792658 , 0.07097213, 0.07858599,
          0.07559483, 0.06621346, 0.05601631, 0.06376614, 0.05791978,
          0.04459551, 0.03399048, 0.02909585, 0.03127124, 0.03412644,
          0.03480625, 0.03412644, 0.02828008, 0.03154317, 0.03426241,
          0.03793338, 0.039293  , 0.02705642, 0.02760027, 0.02895989,
          0.03004759, 0.04133243, 0.04350782, 0.04445955, 0.04622706,
          0.03766145, 0.04065262, 0.02651258, 0.03970088, 0.03059143,
          0.03263086, 0.03983684, 0.05016995, 0.04731475, 0.03548606,
          0.03303875, 0.03915703, 0.04146839, 0.0364378 , 0.02841604,
          0.03847723, 0.03698164, 0.0307274 , 0.03467029, 0.03630183]), array([0.03235894, 0.02828008, 0.04309993, 0.06553365, 0.07559483,
          0.06634942, 0.06770904, 0.10305913, 0.08687967, 0.08021754,
          0.0717879 , 0.0792658 , 0.07097213, 0.07858599, 0.07559483,
          0.06621346, 0.05601631, 0.06376614, 0.05791978, 0.04459551,
          0.03399048, 0.02909585, 0.03127124, 0.03412644, 0.03480625,
          0.03412644, 0.02828008, 0.03154317, 0.03426241, 0.03793338,
          0.039293  , 0.02705642, 0.02760027, 0.02895989, 0.03004759,
          0.04133243, 0.04350782, 0.04445955, 0.04622706, 0.03766145,
          0.04065262, 0.02651258, 0.03970088, 0.03059143, 0.03263086,
          0.03983684, 0.05016995, 0.04731475, 0.03548606, 0.03303875,
          0.03915703, 0.04146839, 0.0364378 , 0.02841604, 0.03847723,
          0.03698164, 0.0307274 , 0.03467029, 0.03630183, 0.03698164])]
   [0.0369816441337418, 0.05411284693098986]
```

```
[ ]  #Convert the x_train and y_train to numpy arrays
     x_train, y_train = np.array(x_train), np.array(y_train)
     10
     #Reshape the data
     x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))
     x_train.shape
```

```
(5581, 60, 1)
```

```
[ ]  #Build the LSTM model
     model = Sequential()
     model.add(LSTM(50, return_sequences=True, input_shape= (x_train.shape[1], 1)))
     model.add(LSTM(50, return_sequences= False))
     model.add(Dense(25))
     model.add(Dense(1))
```

```
[ ]  #Compile the model
     model.compile(optimizer='adam', loss='mean_squared_error')
```

```
[ ]  print(model.compile)
```

```
<bound method Model.compile of <keras.engine.sequential.Sequential object at 0x7f10e7cfcfd0>>
```

```
[ ]  #Train the model
     model.fit(x_train, y_train, batch_size=1, epochs=1)
```

```
5581/5581 [==============================] - 180s 32ms/step - loss: 0.0013
<keras.callbacks.History at 0x7f10e3db97d0>
```

```
[ ]  #Create the testing data set
     #Create a new array containing scaled values from index 1543 to 2002
     test_data = scaled_data[training_data_len - 60: , :]
     #Create the data sets x_test and y_test
     x_test = []
     y_test = dataset[training_data_len:, :]
     for i in range(60, len(test_data)):
       x_test.append(test_data[i-60:i, 0])
```

```
[ ]  print(test_data)
```

```
[[0.03453433]
 [0.03276682]
 [0.03630183]
 ...
 [0.23045547]
 [0.25302515]
 [0.33827327]]
```

```
[ ] #Show the valid and predicted prices
    valid
```

	Close	Predictions
5641	22.510000	20.374233
5642	18.709999	21.029654
5643	18.740000	19.089134
5644	14.380000	18.651354
5645	14.740000	16.283443
...	...	...
7046	24.020000	23.736668
7047	23.959999	23.154179
7048	26.090000	22.960350
7049	27.750000	24.101616
7050	34.020000	25.287573

1410 rows × 2 columns

```
[ ] # Based on the Univariate Vanilla LSTM example from this page:
[ ] #Convert the data to a numpy array
    x_test = np.array(x_test)
```

```
[ ] #Reshape the data
    x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1 ))
```

```
[ ] #Get the models predicted price values
    predictions = model.predict(x_test)
    predictions = scaler.inverse_transform(predictions)
```

```
[ ] #Get the root mean squared error (RMSE)
    rmse=np.sqrt(np.mean(((predictions- y_test)**2)))
    rmse
```

```
    2.8661890584950984
```

```
[ ] #Plot the data
    train = data[:training_data_len]
    valid = data[training_data_len:]
    valid['Predictions'] = predictions
    #Visualize the data
```

```
[ ]  #Convert the data to a numpy array
     x_test = np.array(x_test)
```

```
[ ]  #Reshape the data
     x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1 ))
```

```
[ ]  #Get the models predicted price values
     predictions = model.predict(x_test)
     predictions = scaler.inverse_transform(predictions)
```

```
[ ]  #Get the root mean squared error (RMSE)
     rmse=np.sqrt(np.mean(((predictions- y_test)**2)))
     rmse
```

```
     2.8661890584950984
```

```
[ ]  #Plot the data
     train = data[:training_data_len]
     valid = data[training_data_len:]
     valid['Predictions'] = predictions
     #Visualize the data
```

```
[ ]  # Based on the Univariate Vanilla LSTM example from this page:
     # How to Develop LSTM Models for Time Series Forecasting
     # https://machinelearningmastery.com/how-to-develop-lstm-models-for-time-series-forecasting
     #
     # input sequence is a slope:
     # - y = 4 + 3*x
     # - we are using only y
     #
     import numpy as np
     from keras.models import Sequential
     from keras.layers import Dense, LSTM

     # split a univariate sequence into samples
     def split_sequence(sequence, n_steps):
         X, y = list(), list()
         for i in range(len(sequence)):
             # find the end of this pattern
             end_ix = i + n_steps
             # check if we are beyond the sequence
             if end_ix > len(sequence)-1:
                 break
             # gather input and output parts of the pattern
             seq_x, seq_y = sequence[i:end_ix], sequence[end_ix]
             X.append(seq_x)
             y.append(seq_y)
         return np.array(X), np.array(y)

     # choose a number of time steps
```

```
[ ]  # reshape from [samples, timesteps] into [samples, timesteps, features]
     n_features = 1
     print('X.shape[0] = {}'.format(X.shape[0]))
     print('X.shape[1] = {}'.format(X.shape[1]))
     X = X.reshape((X.shape[0], X.shape[1], n_features))
     print('reshaped X = {}'.format(X))

     # define model
     def get_model(m):
         if m == 'Vanilla_LSTM':
             model = Sequential(name=m)
             model.add(LSTM(50, activation='relu', input_shape=(n_steps, n_features)))
             model.add(Dense(1))
             model.compile(optimizer='adam', loss='mse')
         elif m == 'Stacked_LSTM':
             model = Sequential(name=m)
             model.add(LSTM(50, activation='relu', return_sequences=True, input_shape=(n_steps, n_features)))
             model.add(LSTM(50, activation='relu'))
             model.add(Dense(1))
             model.compile(optimizer='adam', loss='mse')
         model.summary()
         return model
     #model = get_model('Vanilla_LSTM')
     model = get_model('Stacked_LSTM')

     # fit model
     model.fit(X, y, epochs=200, verbose=0)
```

```
[ ]  # choose a number of time steps
     n_steps = 3

     # define input sequence
     def fx(x):
         y = 4 + 3*x
         return y

     x_end = 6
     raw_seq = []
     for x in range(0, x_end):
         y = fx(x)
         raw_seq.append(y)
     print('raw_seq = {}'.format(raw_seq))

     # predict data
     x_input = raw_seq[-1*n_steps:]
     print('x_input = {}'.format(x_input))
     y_expected = fx(x_end)
     print('y_expected = {}'.format(y_expected))

     # split into samples
     X, y = split_sequence(raw_seq, n_steps)
     print('X = {}'.format(X))
     print('y = {}'.format(y))

     # reshape from [samples, timesteps] into [samples, timesteps, features]
     n_features = 1
```

```
[ ]  # show prediction
     x_input = np.array(x_input)
     x_input = x_input.reshape((1, n_steps, n_features))
     yhat = model.predict(x_input, verbose=0)
     print('prediction: for x_input = {}, yhat = {}, y_expected = {}'.format(x_input, yhat, y_expected))
```

```
raw_seq = [4, 7, 10, 13, 16, 19]
x_input = [13, 16, 19]
y_expected = 22
X = [[ 4  7 10]
 [ 7 10 13]
 [10 13 16]]
y = [13 16 19]
X.shape[0] = 3
X.shape[1] = 3
reshaped X = [[[ 4]
  [ 7]
  [10]]

 [[ 7]
  [10]
  [13]]

 [[10]
  [13]
  [16]]]
Model: "Stacked_LSTM"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 lstm_2 (LSTM)                (None, 3, 50)             10400
```

```
[ ]
# split a multivariate sequence into samples
def split_sequences(sequences, n_steps_in, n_steps_out):
  X, y = list(), list()
  for i in range(len(sequences)):
    # find the end of this pattern
    end_ix = i + n_steps_in
    out_end_ix = end_ix + n_steps_out
    # check if we are beyond the dataset
    if out_end_ix > len(sequences):
      break
    # gather input and output parts of the pattern
    seq_x, seq_y = sequences[i:end_ix, :], sequences[end_ix:out_end_ix, :]
    X.append(seq_x)
    y.append(seq_y)
  return array(X), array(y)

# define input sequence
in_seq1 = array([10, 20, 30, 40, 50, 60, 70, 80, 90])
in_seq2 = array([15, 25, 35, 45, 55, 65, 75, 85, 95])
out_seq = array([in_seq1[i]+in_seq2[i] for i in range(len(in_seq1))])
# convert to [rows, columns] structure
in_seq1 = in_seq1.reshape((len(in_seq1), 1))
in_seq2 = in_seq2.reshape((len(in_seq2), 1))
out_seq = out_seq.reshape((len(out_seq), 1))
# horizontally stack columns
dataset = hstack((in_seq1, in_seq2, out_seq))
# choose a number of time steps
n_steps_in, n_steps_out = 3, 2
# convert into input/output
```

```
[ ]  n_steps_in, n_steps_out = 3, 2
     # covert into input/output
     X, y = split_sequences(dataset, n_steps_in, n_steps_out)
     # the dataset knows the number of features, e.g. 2
     n_features = X.shape[2]
     # define model
     model = Sequential()
     model.add(LSTM(200, activation='relu', input_shape=(n_steps_in, n_features)))
     model.add(RepeatVector(n_steps_out))
     model.add(LSTM(200, activation='relu', return_sequences=True))
     model.add(TimeDistributed(Dense(n_features)))
     model.compile(optimizer='adam', loss='mse')
     # fit model
     model.fit(X, y, epochs=300, verbose=0)
     # demonstrate prediction
     x_input = array([[60, 65, 125], [70, 75, 145], [80, 85, 165]])
     x_input = x_input.reshape((1, n_steps_in, n_features))
     yhat = model.predict(x_input, verbose=0)
     print(yhat)
```

```
[[[ 90.62796  95.95483 186.96849]
  [101.11542 106.13463 207.67973]]]
```

```
[ ]  # univariate cnn lstm example
     from numpy import array
     from keras.models import Sequential
     from keras.layers import LSTM
```

```
[ ]
# univariate cnn lstm example
from numpy import array
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers import TimeDistributed
from keras.layers.convolutional import Conv1D
from keras.layers.convolutional import MaxPooling1D

# split a univariate sequence into samples
def split_sequence(sequence, n_steps):
  X, y = list(), list()
  for i in range(len(sequence)):
    # find the end of this pattern
    end_ix = i + n_steps
    # check if we are beyond the sequence
    if end_ix > len(sequence)-1:
      break
    # gather input and output parts of the pattern
    seq_x, seq_y = sequence[i:end_ix], sequence[end_ix]
    X.append(seq_x)
    y.append(seq_y)
  return array(X), array(y)

# define input sequence
raw_seq = [10, 20, 30, 40, 50, 60, 70, 80, 90]
# choose a number of time steps
n_steps = 4
```

```
[ ]
raw_seq = [10, 20, 30, 40, 50, 60, 70, 80, 90]
# choose a number of time steps
n_steps = 4
# split into samples
X, y = split_sequence(raw_seq, n_steps)
# reshape from [samples, timesteps] into [samples, subsequences, timesteps, features]
n_features = 1
n_seq = 2
n_steps = 2
X = X.reshape((X.shape[0], n_seq, n_steps, n_features))
# define model
model = Sequential()
model.add(TimeDistributed(Conv1D(filters=64, kernel_size=1, activation='relu'), input_shape=(None, n_steps, n_features)))
model.add(TimeDistributed(MaxPooling1D(pool_size=2)))
model.add(TimeDistributed(Flatten()))
model.add(LSTM(50, activation='relu'))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
# fit model
model.fit(X, y, epochs=500, verbose=0)
# demonstrate prediction
x_input = array([60, 70, 80, 90])
x_input = x_input.reshape((1, n_seq, n_steps, n_features))
yhat = model.predict(x_input, verbose=0)
print(yhat)
```

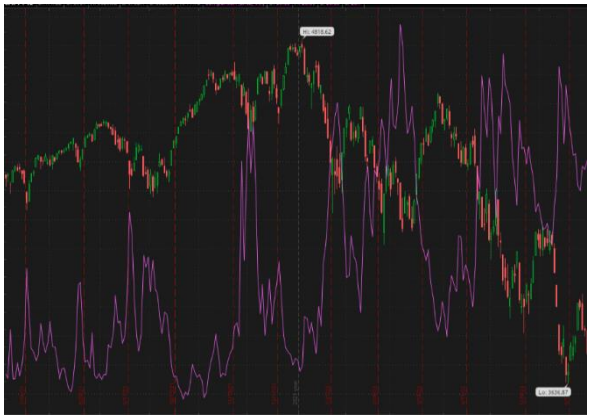
Anexo 3

Fecha prediccion volatilidad	vix error medio prediccion	VIX RMSE
02/07/2022	1,96%	2,84%
11/07/2022	1,99%	3,02%
18/07/2022	2,46%	3,72%
25/07/2022	2,73%	4,14%
02/08/2022	2,85%	4,49%
16/08/2022	3,18%	5,00%
31/08/2022	3,33%	5,30%
30/09/2022	3,41%	5,75%
31/10/2022	3,57%	6,07%
30/11/2022	3,62%	5,78%
30/12/2022	4,23%	6,56%
29/03/2023	4,50%	7,27%
26/06/2023	4,48%	6,92%
12/06/2024	4,57%	7,31%

Anexo 4

Fecha predicción SPX	SPX error medio predicción	Predicción SPX
	1,96%	
7/11/2022	1,99%	3890
7/18/2022	2,46%	3824

Anexo 5



Capítulo 8. BILIOGRAFÍA

Banco de España. (2020). Plan Estratégico 2024. Eurosistema

Benklifa, M., & Benklifa, M. (2011). *Soaring with condor options*. Upper Saddle River, N.J.: FTPress Delivers.

Cuya, M. L. (2016). La disrupción de las startups FinTech en el mundo financiero.

Derivatives Analytics with Python: Data Analysis, Models, Simulation, Calibration and Hedging por Yves Hilpisch | Agosto 3, 2015

Económico, (JUN)

Elvery, J. (2019). Changes in the Occupational Structure of the United States: 1860 to

2015. Economic Commentary, (2019-09).

Fernández, A. (2019). Inteligencia artificial en los servicios financieros. Boletín

Hilpisch, Y. *Python for Finance, 2nd Edition*.

Hilpisch, Y. *Python for algorithmic trading*.

Jansen, S. *Machine learning for algorithmic trading*.

Sinclair, E., n.d. *Volatility trading*.

Manyika, J., & Bughin, J. (2018). The promise and challenge of the age of artificial

intelligence. McKinsey Global Institute Executive Briefing.

MMC Ventures (2019). The State of AI 2019: Divergence. Recuperado el 25 de marzo

de 2020 de https://www.stateofai2019.com/

Nelli, F. *Python data analytics*.

Pedrycz, W., & Chen, S. (2013). *Time series analysis, modeling and applications*. Heidelberg: Springer.

Philippon, T. (2016). The fintech opportunity (No. w22476). National Bureau of

Economic Research.

Services Sector. Recuperado de http://www.ucg-sa.com/wp-

content/uploads/2018/12/Artificial-Intelligence-Effects-on-the-Financial-Sector.pdf

United Consulting Group. (2018). Artificial Intelligence Effects on the Financial