



GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

CURSO 2021-2022

A MACHINE LEARNING APPROACH TO THE REAL ESTATE MARKET

Autor: David Cocero Quintanilla

Director: Alan Pisano

Madrid

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

A Machine Learning approach to the Real Estate market

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico 2021/22 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.: David Cocero Quintanilla

Fecha: 29/06/2022



Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Alan Pisano

Fecha: 29/06/2022





GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO
CURSO 2021-2022

A MACHINE LEARNING APPROACH TO THE REAL ESTATE MARKET

Autor: David Cocero Quintanilla

Director: Alan Pisano

Madrid

Agradecimientos

Me gustaría agradecerle a mi familia el apoyo que me han dado en los últimos 4 años

También agradecer a los compañeros y profesores que he tenido durante la carrera y que me han facilitado llegar hasta aquí

Por último, gracias a la Universidad Pontificia de Comillas y a Boston University por darme la oportunidad de poder realizar este proyecto

MACHINE LEARNING ENFOCADO AL MERCADO MOBILIARIO

Autor: Cocero Quintanilla, David

Supervisor: Pisano, Alan

Entidad colaboradora: Boston University

RESUMEN DEL PROYECTO

Palabras clave: Mercado Inmobiliario, Probabilidad, Machine Learning, Aplicación

1. Introducción

Los bienes inmuebles incluyen terrenos, los recursos naturales sobre ellos y los edificios que están contruidos sobre él [1]. El mercado inmobiliario desempeña un papel muy importante en la economía global y más concretamente en la de EE. UU., llegando a unos beneficios de hasta \$202 mil millones el pasado año, 3.4% más que el año anterior [2]

Los agentes inmobiliarios son los que se encargan de negociar acuerdos entre vendedor y comprador, obteniendo una comisión (entorno el 5% sobre el precio de venta). Con el auge del mercado inmobiliario, más y más gente se está introduciendo en el sector, creando una mayor competencia. Por ello, cada agente intenta buscar el mínimo detalle que le dé una ventaja sobre el resto.

1.1 Planteamiento del Problema

Los agentes inmobiliarios están constantemente buscando ayudar a vender propiedades para poder así recibir una comisión sobre el precio final. Para ello, contactan a varios propietarios esperando que acepten poner su casa en venta. Sin embargo, en numerosas ocasiones el dueño no está interesado, así que los agentes pierden gran parte de su tiempo contactando propietarios sin éxito. Esto se debe a que muchos agentes eligen a los dueños de forma aleatoria, por lo que obtienen un porcentaje de éxito relativamente bajo.

1.2 Estado de la técnica

Estas son las plataformas del mercado inmobiliario más conocidas en EE.UU. Aunque tienen funciones similares al proyecto, no pueden resolver el problema planteado.

- **Zillow:** es la plataforma más popular del país [3]. Consiste en una web y una aplicación móvil donde los compradores, vendedores y agentes pueden buscar propiedades. Tiene información de 110 millones de casas en EEUU. Una de sus herramientas más útiles es *Zestimate*, que consigue aproximar el valor de una vivienda en base a sus características. La predicción suele ser bastante precisa, teniendo solo un 1.9 % error para propiedades que están en el mercado y 6.9 % para las que no.
- **Realtor.com:** se centra en los compradores, dándoles herramientas para que encuentren lo que están buscando [4].
- **Redfin:** cobran una comisión de venta menor que la de sus competidores [5]. Además, tienen una herramienta para estimar el precio de una propiedad, siendo menos precisa que el *Zestimate*.
- **Trulia:** ayuda a compradores y vendedores con recomendaciones locales y otras herramientas [6]. Fue adquirida por Zillow.

- **Opendoor:** web innovadora que permite la compraventa de casas online, sin necesidad de un agente [7]. El propietario solo tiene que introducir la dirección de su casa y algunos datos y la página le hará una oferta. Si la acepta, el dinero le llega de forma instantánea, y la comisión es menor que la que cobran los agentes.

1.3 Objetivo

El objetivo del proyecto es crear una aplicación multiplataforma que pueda ejecutarse tanto en Android como en IOS. Esta aplicación ahorrará tiempo a los agentes inmobiliarios ya que filtran los propietarios con los que establecer contacto. Esto se consigue intentando predecir como de probable es que cierta casa se ponga en venta. Los agentes simplemente contactarán aquellos propietarios cuyas casas tengan una alta probabilidad, descartando aquellas con baja probabilidad.

El trabajo se ha desarrollado en la asignatura de Senior Design de Boston University, y los clientes, que trabajan con una agencia inmobiliaria en California, son los que han propuesto este proyecto.

2. Descripción del Sistema

Este es un esquema de los distintos componentes del sistema y cómo interactúan entre ellos:

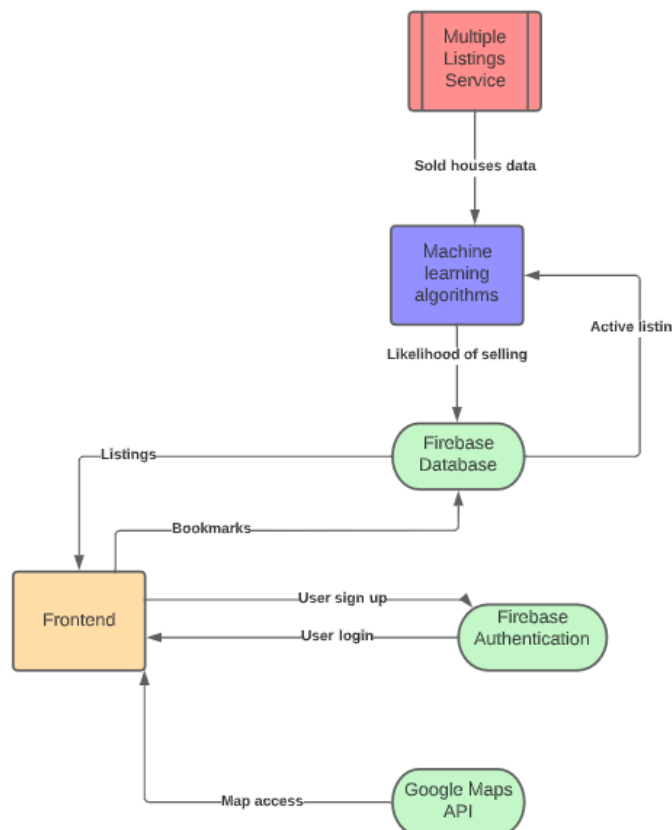


Figura 1- Interacciones entre los componentes

2.1 Front end

El componente principal es el front end, que es lo que el usuario puede ver y con lo que puede interactuar. Está compuesto por las diferentes pantallas de la aplicación, y programado en React.js, una librería de JavaScript que se usa para que tenga compatibilidad con distintas plataformas. A continuación, aquí se incluye una simple explicación de cada pantalla y sus interacciones.

- **Inicio de sesión:** este es el punto inicial de la aplicación, donde los usuarios pueden iniciar sesión usando su cuenta de Google o una cuenta propia de la aplicación. Si no tiene cuenta aún, el usuario se la puede crear introduciendo su información personal y contraseña. Esta pantalla usa Firebase Authentication, un servicio externo utilizado para manejar procesos de autenticación.
- **Filtrado:** las propiedades que se muestran en el mapa pueden filtrarse por 4 categorías: precio estimado, año de construcción, superficie y número de habitaciones.
- **Mapa:** las propiedades filtradas aparecerán sobre el mapa como iconos con forma de casa. El mapa se obtiene a partir de la API de Google Maps como se aprecia en el esquema. Cada una de las propiedades está coloreada dependiendo de cómo de probable es que se pongan en venta. Las casas con mayor probabilidad de ser vendidas aparecerán en verde oscuro, las siguientes en verde claro, amarillo, naranja y finalmente las de menor probabilidad serán rojas.
El usuario puede seleccionar una casa para ver más detalles e incluso añadirla a marcadores, guardándola para más adelante. Esta pantalla está conectada a la base de datos, obteniendo de allí los datos de las propiedades y actualizando los datos de los usuarios añadiendo los nuevos marcadores.
- **Marcadores:** aquí aparecerán las propiedades que el usuario tenga en marcadores. El usuario puede ver los detalles de estas casas, añadir comentarios o sacarlos de la lista de marcadores.
- **Perfil:** aquí se muestra información sobre la cuenta del usuario como el nombre, el número de marcadores y un registro de actividad que guarda los movimientos realizados con los marcadores.

2.2 Base de datos

La base de datos del sistema está implementada usando *Firebase Realtime Database*, una base de datos online donde la información se guarda en formato JSON y las actualizaciones son instantáneas. Tiene dos tablas principales:

- **Casas:** contiene toda la información de las propiedades que se muestran en el mapa. Esta información se obtiene del Multiple Listings Service (MLS)
- **Usuarios:** incluye toda la información personal de los usuarios y la lista de propiedades en marcadores de cada uno.

2.3 Machine Learning

La primera parte de Machine Learning es la creación de un modelo que estime el precio de una vivienda basándose en sus características. Para entrenar el modelo se usan los datos de todas las casas vendidas desde 2015 en el código postal 90046, localizado en Los Ángeles y en el que los clientes tenían interés. Estos datos deben ser procesados a conciencia, modificando el formato de ciertos campos, eliminando otros y descartando filas inválidas. Finalmente, este conjunto de datos se divide en dos: el 80% de las filas se usan para la fase de entrenamiento y el 20% restante para testear el modelo.

El modelo tendrá como variables de entrada las diferentes características de la propiedad, teniendo solo una variable de salida: el precio. El producto final tiene tres posibles implementaciones:

- **Regresión lineal:** es uno de los algoritmos más simples, donde se asume que el problema es lineal [8]
- **Random Forests:** está formado por varios árboles de decisión. El resultado final del modelo será el resultado más popular obtenido por los árboles [9]
- **Gradient Boosting Regressor:** es un proceso iterativo donde paso a paso se va mejorando la precisión de la predicción mediante el uso de los residuos [10]

Una vez el modelo esté entrenado, se usará para predecir el valor estimado de un conjunto de propiedades en que los agentes inmobiliarios estén interesados, normalmente es una lista de *cold calls* que suelen tener. A continuación, el algoritmo recorrerá los registros de ventas buscando cuándo fue la última vez que se vendió la propiedad y su precio. Este se comparará con la estimación actual para obtener el crecimiento medio anual del precio en porcentaje. Las propiedades se separarán en 5 partes en función de este porcentaje, ya que es un buen indicador de cómo de probable es que la casa se ponga en venta. Los clientes aseguran que, según su experiencia, los propietarios están más dispuestos a vender si pueden obtener un buen Retorno de la Inversión (ROI en inglés). Por último se suben los resultados a la base de datos para que el mapa pueda mostrar correctamente las casas coloreadas en función de la probabilidad de ser puestas en venta.

3. Resultados

En esta sección se mostrarán los resultados obtenidos en la implementación del algoritmo para estimar el precio. Lo primero para mostrar es esta matriz de correlación, obtenida a partir de los datos de las propiedades vendidas.

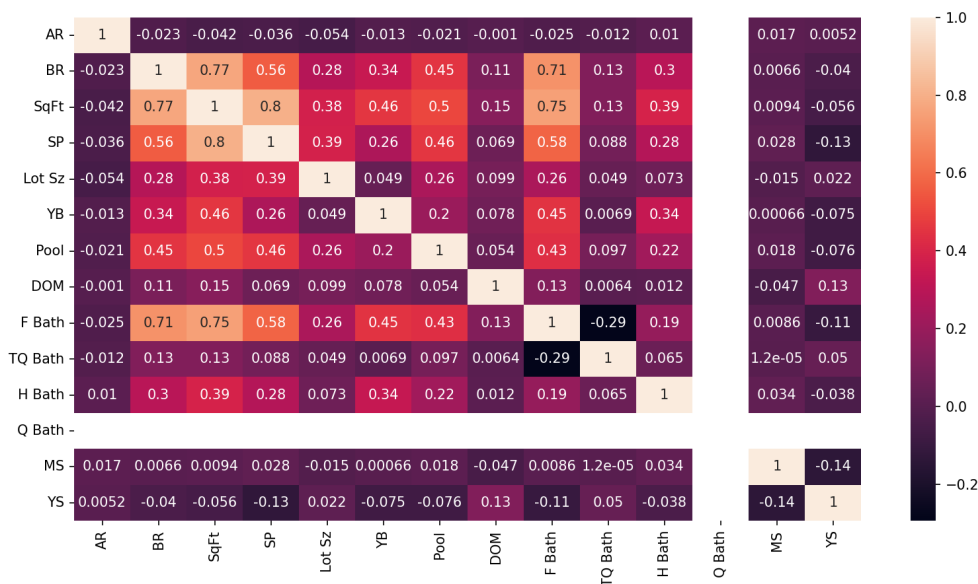


Figura 2: Matriz de correlación de las características

Esta matriz contiene coeficientes desde -1 a 1 que muestran cómo se relacionan las distintas características de las propiedades entre sí y con el precio final. Cuánto más cerca esté el

coeficiente de una característica a 1, más contribuye al precio final. Este es el caso del número de baños completos, el número de habitaciones y la superficie total. También hay otras características cercanas a cero, lo que significa que apenas tienen efecto en el precio, como el código de área (AR) o los días en el mercado (DOM). Estas características no se utilizarán en el modelo ya que pueden producir *overfitting*.

Teniendo en cuenta los coeficientes de esta matriz, se desarrolla el modelo con los tres algoritmos ya mencionados. En esta table se representa el rango de error medio de cada uno en la fase de entrenamiento y de testeo.

ALGORITMO	ERROR DE ENTRENAMIENTO	ERROR DE TESTEO
Regresión Lineal	20- 23 %	20- 23 %
Random Forest	6-7 %	18-19%
Gradient Boosting Regressor	8-16 %	16-20 %

El modelo de regresión lineal cuenta con el mayor porcentaje de error, lo que significa que el problema no es lineal. En los otros dos algoritmos, el error en el proceso de entrenamiento se ha reducido bastante, pero el error más importante, el de testeo, apenas ha bajado. Esta diferencia entre ambos errores suele ser un signo de que hay *overfitting*.

4. Conclusiones

Estas son algunas de las conclusiones que se pueden obtener del proyecto:

- Los clientes están satisfechos con la interfaz de usuario desarrollada
- El error del estimador de precio es ligeramente superior a lo deseado
- El problema con el estimador para estar más relacionado con la calidad de los datos de las propiedades utilizados que con la formulación del modelo.
- Todo apunta a que el algoritmo para calcular la probabilidad de venta de una propiedad será un éxito

5. Referencias

- [1] Turner, T. "How is Real Estate Defined", Annuity.org, Mayo 2022
<https://www.annuity.org/real-estate/>
- [2] "Real Estate Sales and Brokerage in the US", Ibisworld, Diciembre 2021
<https://www.ibisworld.com/industry-statistics/market-size/real-estate-sales-brokerage-united-states/>
- [3] "About Zillow", Zillow, Mayo 2021, <https://www.zillow.com/z/corp/about/>
- [4] "About Realtor.com", Realtor.com, Mayo 2022, <https://www.realtor.com/about/>
- [5] "Redfin", Wikipedia, Junio 2022, <https://en.wikipedia.org/wiki/Redfin>
- [6] "What is Trulia?", Trulia, Octubre 2021, <https://support.trulia.com/hc/en-us/articles/205995978-What-is-Trulia->

- [7] “How does selling to Opendoor work?”, Opendoor, Mayo 2022,
<https://www.opendoor.com/w/faq/how-does-selling-to-opendoor-work>
- [8] Brownlee, J. “Linear Regression for Machine Learning”, Machine Learning Mastery, Marzo 2016 <https://machinelearningmastery.com/linear-regression-for-machine-learning/>
- [9] You, T. “Understanding Random Forest”, Towards Data Science, Junio 2019,
<https://towardsdatascience.com/understanding-random-forest-58381e0602d2>
- [10] Masui, T. “All You Need to Know about Gradient Boosting Algorithm – Part 1. Regression”, Towards Data Science, Enero 2022, <https://towardsdatascience.com/all-you-need-to-know-about-gradient-boosting-algorithm-part-1-regression-2520a34a502>

A MACHINE LEARNING APPROACH TO THE REAL ESTATE MARKET

Author: Cocero Quintanilla, David

Supervisor: Pisano, Alan

Collaborating Entity: Boston University

PROYECT SUMMARY

Keywords: Real State, Likelihood, Machine Learning, Application

1. Introduction

Real Estate stands for the type of property that includes land, the natural resources on it and the buildings attached to it [1]. The Real Estate market is a huge part of the US economy, amounting \$202 billion in revenue in the last year, a 3.4% from the previous year [2]

Realtors are the people in charge of negotiating deals between the buyer and seller, obtaining a commission (around 5%) on the agreed sale. With the rise of the Real Estate industry, more people have gotten into the sector, and it has become quite competitive, so realtors want to search for every little thing that would give them an advantage over the rest.

1.1 Problem Statement

Real Estate agents are constantly searching for houses so that they can try to sell it to a new buyer and obtain their profits on the sale price. They will approach the owners of the houses, hoping they can agree to put the house on the market. However, most times the owners end up not being interested in the deal, so the realtors waste a tremendous amount of time contacting owners with no results. This is because many realtors approach owners randomly and without following any pattern which results in a low success rate.

1.2 State of the Art

Here are some of the most popular Real Estate platforms in the US. They provide a similar function to this project, but they can't solve the problem defined above.

- **Zillow:** the most popular platform in the country [3]. Has a website and a mobile application where buyers, sellers and realtors can search for listings. It contains data from over 110 million homes, including their address, features and photos. One of its most useful tools is the *Zestimate* which estimates the price of a house based on its features. This prediction is pretty accurate as it has 1.9 % average error rate for on-market houses and 6.9 % for off market houses.
- **Realtor.com:** mainly focus on people searching to buy a house, giving them tools to find the listings they want [4].
- **Redfin:** what makes this platform different is that they try to undercut competition by making sellers pay Redfin a discounted fee to list the seller's home [5]. They also have a tool to estimate the price of a listing but with less accuracy than the *Zestimate*
- **Trulia:** it uses recommendations, local insights, and map overlays to help buyers and sellers make their transactions [6]. It was acquired by Zillow, and it is now one of its subsidiaries

- **Opendoor:** this is an innovative website which allows houses to be sold and bought online, without the need of a realtor [7]. The owner will introduce the address of the home and its features, and the platform will make an offer. Should the offer be accepted, the owner will instantly receive the money, and the commission is less than the usual realtor fee.

1.3 Objective

The objective of this project is to create a multiplatform application that can be run both on Android and IOS. The application will help realtors save time by filtering the owners that are approached looking for a sale. This will be done by trying to predict how likely a house is to be sold by its owner. The realtors using the app will only contact the owners whose selling probability is high, discarding those with low probability.

The work was developed in the Senior Design class of Boston University and the clients, who work with a Real Estate Agency in California, proposed this project.

2. System description

Here is a schema of how the different components of the system interacts with each other:

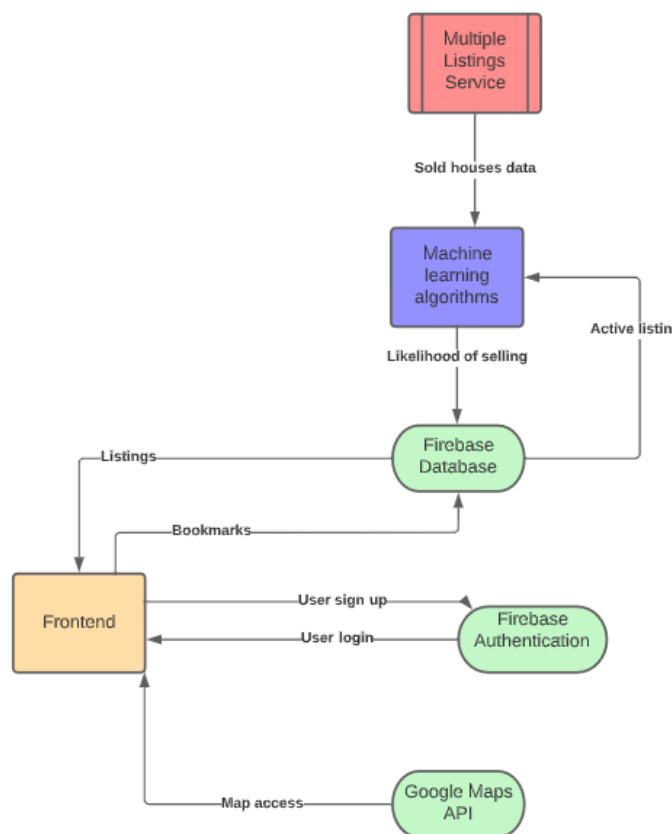


Figure 1 – Interactions between the components

2.1 Front end

The main component is the front end which is what the user can actually see and interact with. The front end is the collection of the different screens of the application. It was coded

in React.js, a JavaScript library which provides support for different platforms. Here is a basic explanation of each screen and its interactions with other components.

- **Login screen:** here the users can log in using their Google account or their application account. If the user does not have an account yet, it can be created by introducing the personal information and a new password. This screen will use Firebase Authentication, an external service useful for managing users account and authentication process.
- **Filter screen:** the listings to be displayed on the map can be filtered here based on 4 characteristics: estimated price, year of construction, surface in square feet and number of bedrooms.
- **Map screen:** the filtered listings appear as icons over a map. The map is obtained from the Google Maps API, as the schema above shows. Each of the listings is color coded based on the likelihood of sale. The listings with the highest selling probability appear in dark green, then will come light green, yellow, orange and the listings with the lowest probability are red.
The user may click on a listing to see more details and to bookmark it, saving it for later. This screen will be connected to the database, obtaining the listings data from there and storing the listings a user has bookmarked.
- **Bookmarks screen:** the listings that are bookmarked by a user will appear here. The user can see all the features of the bookmarked properties, add comments to them and delete them.
- **Profile screen:** this displays some information about the user's account such as the name, the number of bookmarked activities and an activity log which tracks the movements made by the user regarding the bookmarks.

2.2 Database

The database of the system will be implemented using the Firebase Realtime Database, an online database which stores the data as a JSON tree and where the updates are instantly available. The database consists of two main tables:

- **Houses table:** contains all the listings data that is displayed in the map screen. This data is manually obtained from the Multiple Listings Service (MLS)
- **Users table:** includes all the personal information of the user as well as the list of bookmarked properties the user currently has.

2.3 Machine Learning

The first part of the machine learning is the creation of a price estimator, that outputs the value of a house of a zip code based on its features. The data used for the training of the model consists of all the houses sold since 2015 in the 90046 zip code, located in Los Angeles and of interest of the clients. This data needs to be thoroughly processed, modifying some fields, eliminating others, and removing invalid rows. Then, this dataset is split into two parts: 80% of the rows are used for the actual training and the other 20% to test the model.

The price estimator will have multiple inputs (the different features of the listing) but just one output variable, the price. There are three different alternatives of implementing the model.

- **Linear Regression:** it is one of the simplest algorithms as it assumes that the problem is linear [8]
- **Random Forests:** it is formed by multiple independent decision trees. The final output of the model will be the most popular output of the trees [9]
- **Gradient Boosting Regressor:** it uses weak predictors to get a better prediction with each iteration. Each time it will use the residuals of the previous predictor to construct the next one [10]

Once the model is trained, it will be fed a list of properties the Realtor is interested in. Usually, these properties will come from a list of *cold calls* they have. Then, the algorithm will go through the sales record to find out the last time the property was sold and the selling price. This will be compared to the current estimated price to compute the average growth percentage per year. The listings will be split in 5 bins based on this percentage as it is good indicator of how likely the listing is to be put on the market. The clients believe this to be true as the owners of a house are, in their experience, more open to a sale if they can get a good ROI (Return on Investment) on the house. The results are uploaded to the online database so the map can display the color-coded houses based on these 5 bins.

3. Results

This section will be used to show the results obtained in the value estimator. First, here is the correlation matrix obtained from the listings data:

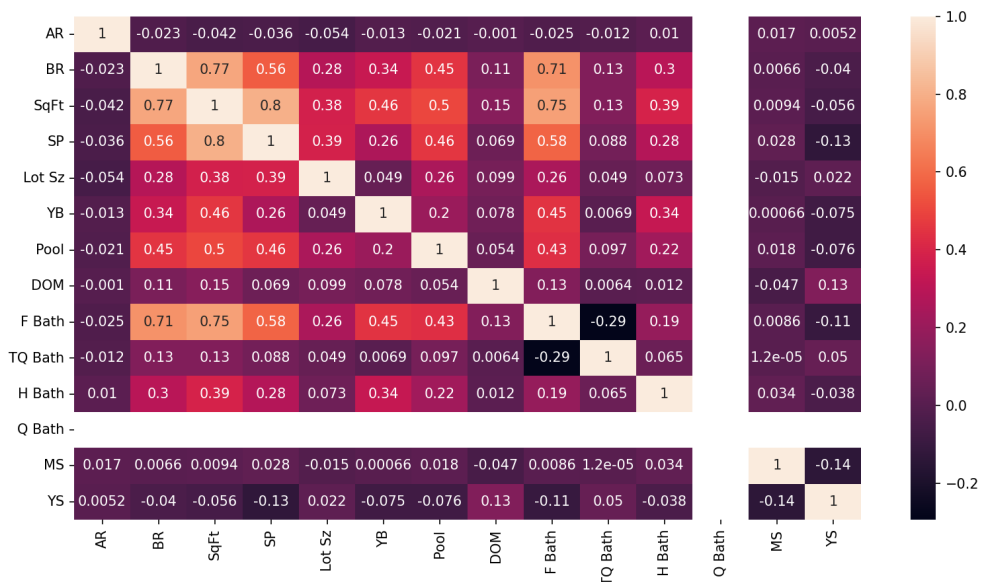


Figure 2: Correlation matrix of the listings features

This matrix will contain coefficients from -1 to 1 that show how the different features of the listings relate to each other and to the output, the sold price. The closer the value is to 1, the more the feature contributes to the price. This is the case for the number of full bathrooms, the number of bedrooms and the total surface. There are also other features that are close to 0, which means they have little to no effect in the final price, such as the Area code (AR) or

the days on market (DOM). These features had to be removed from the model as they produced overfitting.

Taking into account the coefficients of the matrix, a model was developed for the three algorithms previously mentioned. This table represents the range of the average error percentage, for the training and test sets.

ALGORITHM	TRAINING ERROR	TESTING ERROR
Linear Regression	20- 23 %	20- 23 %
Random Forest	6-7 %	18-19%
Gradient Boosting Regressor	8-16 %	16-20 %

The linear regression model has the highest error which probably means that the problem is not linear. The other two algorithms seem to do much better in the training set, but the improvement in the testing error is not that notable. This is probably a sign of overfitting.

4. Conclusions

These are some of the conclusions obtained from the project:

- The clients were satisfied with the UI as they considered it user-friendly and visually pleasing.
- The error obtained in the value estimator is slightly higher than desired.
- The issue with the estimator seems to be related to the quality of the data rather than the model formulation.
- The algorithm to obtain the likelihood of sale is expected to be a success

5. References

- [1] Turner, T. "How is Real Estate Defined", Annuity.org, May 2022
<https://www.annuity.org/real-estate/>
- [2] "Real Estate Sales and Brokerage in the US", Ibisworld, December 2021
<https://www.ibisworld.com/industry-statistics/market-size/real-estate-sales-brokerage-united-states/>
- [3] "About Zillow", Zillow, May 2021, <https://www.zillow.com/z/corp/about/>
- [4] "About Realtor.com", Realtor.com, May 2022, <https://www.realtor.com/about/>
- [5] "Redfin", Wikipedia, June 2022, <https://en.wikipedia.org/wiki/Redfin>
- [6] "What is Trulia?", Trulia, October 2021, <https://support.trulia.com/hc/en-us/articles/205995978-What-is-Trulia->
- [7] "How does selling to Opendoor work?", Opendoor, May 2022,
<https://www.opendoor.com/w/faq/how-does-selling-to-opendoor-work>

[8] Brownlee, J. “Linear Regression for Machine Learning”, Machine Learning Mastery, March 2016 <https://machinelearningmastery.com/linear-regression-for-machine-learning/>

[9] You, T. “Understanding Random Forest”, Towards Data Science, June 2019, <https://towardsdatascience.com/understanding-random-forest-58381e0602d2>

[10] Masui, T. “All You Need to Know about Gradient Boosting Algorithm – Part 1. Regression”, Towards Data Science, Jan 2022, <https://towardsdatascience.com/all-you-need-to-know-about-gradient-boosting-algorithm-part-1-regression-2520a34a502>

Table of Contents

<i>Table of Contents</i>	<i>I</i>
<i>Figure Index</i>	<i>III</i>
<i>Tables Index</i>	<i>V</i>
Chapter 1. Introduction	6
1.1 PROYECT MOTIVATION	7
Chapter 2. Technologies' description	8
2.1 Front End.....	8
2.2 Database and Authentication.....	9
2.3 Python Libraries	10
2.4 Machine Learning Algorithms	11
2.4.1 Linear Regression.....	11
2.4.2 Random Forests.....	13
2.4.3 Gradient Boosting Regression.....	14
Chapter 3. State Of The Art	15
3.1 Zillow	15
3.2 Realtor.com	17
3.3 Redfin.....	17
3.4 Trulia.....	18
3.5 Opendoor.....	18
Chapter 4. Project Definition	22
4.1 Justification	22
4.2 Objectives.....	24
4.3 Methodology	25
4.3.1 Front End	25
4.3.2 Backend.....	25

4.3.3 <i>Machine Learning</i>	26
4.4 Workplan and Estimated Budget.....	27
4.4.1 <i>Workplan</i>	27
4.4.2 <i>Estimated Budget</i>	29
Chapter 5. <i>Developed System</i>	31
5.1 Design Overview	31
5.2 Front End.....	35
5.2.1 <i>Login Screen</i>	35
5.2.2 <i>Filter Screen</i>	43
5.2.3 <i>Map Screen</i>	46
5.2.4 <i>Bookmarks Screen</i>	51
5.2.5 <i>Profile Screen</i>	58
5.3 Backend.....	62
5.3.1 <i>Firebase Authentication</i>	62
5.3.2 <i>Firebase Realtime Database</i>	63
5.4 Machine Learning.....	71
5.4.1 <i>Data Collecting</i>	71
5.4.2 <i>Data Cleaning</i>	73
5.4.3 <i>Correlation matrix</i>	76
5.4.4 <i>Linear Regression</i>	80
5.4.5 <i>Random Forests</i>	84
5.4.6 <i>Gradient Boosting Regressor</i>	87
5.4.7 <i>Likelihood of Selling</i>	91
Chapter 6. <i>Results Analysis</i>.....	93
6.1 Price Estimator	93
Chapter 7. <i>Conclusions and Future Work</i>.....	96
7.1 Work Completed	96
7.2 Objectives Fulfillment.....	98
7.3 Future Improvements	100
Chapter 8. <i>Bibliography</i>	102
ANNEX I: <i>PROYECT ALIGNMENT WITH THE SDGs</i>.....	106

Figures Index

Figure 1: Simple linear regression example [16].....	12
Figure 2: Example of a Decision Tree [17]	13
Figure 3: Tree obtained from residuals [20].....	14
Figure 4: Zestimate median error for active listings in different metros of the US [23].....	16
Figure 5: Zestimate median error for off-market houses.....	
in different metros of the US [24]	16
Figure 6: Price estimator by Opendoor [30].....	19
Figure 7: Net Profits calculator by Opendoor [32].....	20
Figure 8: Workplan of the project	28
Figure 9: Basic overview of the project.....	31
Figure 10: Detailed overview of the components of the system	33
Figure 11: Login Screen with Google Sign in.....	36
Figure 12: Email screen with Google Sign in.....	36
Figure 13: Login Screen with email sign in	38
Figure 14: Create an account	40
Figure 15: Reset Password	42
Figure 16: Filter Screen	44
Figure 17: Map Screen I.....	48
Figure 18: Map Screen II.....	47
Figure 19: Map Screen with details I.....	49
Figure 20: Map Screen with details II	48
Figure 21: Bookmarks screen	52
Figure 22: Bookmarks details screen.....	53
Figure 23: Bookmarks details with comments.....	54
Figure 24: User Profile Screen.....	59

Figure 25: Users registered in the app	62
Figure 26: Sign in providers	63
Figure 27: Listing stored in the database	64
Figure 28: User stored in the database.....	68
Figure 29: MLS listings search [34]	72
Figure 30: Correlation matrix of the listings features.....	77
Figure 31: Correlation matrix after removing several features	79
Figure 32: Predicted Sold Price vs Actual Sold Price	83
Figure 33: Sold Price vs Predicted Sold Price in Random Forest	87
Figure 34: Sold Price vs Predicted Sold Price in Gradient Boosting Regressor	90
Figure 35: Average income of realtors by hours worked per week [40]	107

Tables Index

Table 1: Cost Breakdown for the first year.....	30
Table 2: Coefficients obtained in Linear Regression.....	82
Table 3: Average error of the algorithms for the training and testing	94

Chapter 1. INTRODUCTION

In order to get a better understanding of the project, it is key to understand what exactly Real Estate is and what does a realtor do.

Real Estate is defined as the land, and everything attached to it. This could be something man-built like a building or something natural like a tree. Real Estate market is where all the transactions regarding Real Estate are made. This sector is one of the most important of the economy as usually the prices are a good indicator of the economic stability of a country [1]. Also, the affordability of a house directly affects the purchasing power of both the one who rents or buys and the landlord or seller.

A realtor or Real Estate agent is a person who assists other people buying, selling, or renting buildings or land. They need to obtain a license and they obtain their profit via a commission on the house price they helped buy or sell. The commission is usually around 5% and is split between the selling and the buying agent [2]. Realtors will contact both potential buyers and sellers offering them to work through the transaction in order to receive the commission.

In the United States the Real Estate market comprises around 15 to 18% of the GDP [3], which is divided into residential investment and consumption spending on housing services. Since 2017, the real estate industry has grown by around 4% on average every year [4] and is expected to continue at this rate. Following this tendency, in 2020 and 2021 combined there was a 60% increase in the amount of Real Estate agents within the field compared to the two years prior [5]. A large part of this rise is attributed to the pandemic and the “working from home” possibility that the Real Estate industry provides. This increased number of realtors results in a tough competition to survive within the industry.

1.1 PROYECT MOTIVATION

I am doing this project in the Senior Design class in Boston University. This is a mandatory class for all engineering students where they have to develop a project chosen by outside clients. My clients were Nicholas Musella, Varshith Anilkumar and Andrew Vargas. Andrew Vargas is a realtor working in California and he posed the following problem.

Realtors must determine when a property will be put on the market so that they can sell it to the next homeowner. However, a tremendous amount of time is wasted on trying to find potential homeowners that are willing to put their houses on the market. It is challenging to assess which homeowners are more likely to agree to put their properties on the market. That is why realtors sometimes have no choice but to contact a large number of homeowners which may or may not agree to a sale.

Chapter 2. TECHNOLOGIES' DESCRIPTION

Here I am going to describe the technologies used for the project. Because of the nature of the project, all of them are software tools or languages.

2.1 FRONT END

The first one I am going to talk about is React.js. The whole front end of the application was developed using React.js as the clients made it clear they wanted a cross-platform application that worked both in Android and IOS. To accomplish this, one possible solution would be to code first the application in Android using Java o Kotlin and then re write the code in Swift so that it can be run on an IOS device. The other, more logical, solution is to use a cross-platform tool like React.js or Flutter and just write the code once for both kind of devices.

React was released in 2013 and auto defines as a “JavaScript library for building user interfaces” [6]. JavaScript is usually associated with webpages, allowing to display dynamic content on the browser and manage connections with the server. In this case is not actually used for websites but for front end development. JavaScript is a just in time (JIT) compiled language [7] and therefore it can be run on multiple devices (Android and IOS) so that is why it can be used for cross-platform development.

React.js also makes the development simpler because it is declarative, component-based and reusable. Each component can be rendered to the DOM with the `render ()` method and have a state that can be modified. When the state of a component is updated, its `render ()` method is invoked so that everything displayed on the screen is accurate.

The React.js code was written using Visual Studio Code [8] which is a popular source-code editor. This platform can be really helpful as there are many extensions which can be used to correct the syntax of a language and autocomplete lines. In this case the extension used was *ES7+ React/Redux/React-Native snippets*, making the coding faster. Visual Studio Code

also contains several tools for debugging and for Git, so pushing and pulling code from a Github repository to the editor can be easily done.

Another tool used was Android Studio which is an IDE designed for Android development [9]. While the application is compatible with IOS, the main development and testing was done on Android devices. Android Studio included an emulator where an Android Virtual Device (AVD) can be created to simulate an actual physical device. The AVDs available are different Google Pixel models with different versions of Android. The functioning of the application was tested on different AVD models before integration as it was essential to make sure the code was compatible with most Android versions currently in the market.

2.2 DATABASE AND AUTHENTICATION

In the backend, it is important to be familiar with the technology used for the database. It was decided to work on a NoSQL (non-relational) database instead of a relational database because of the data used on the project, as the NoSQL approach allows more flexibility. Also, this type of databases are usually lighter in size which is why they are really popular nowadays.

The platform used for the database is *Firebase Realtime Database* [9]. It stores data in the cloud in JSON format. Its main feature is that it works in real-time, meaning that if something is updated, the data is synchronized with the connected clients. Keeping the data synchronized at all times is a really important feature of this project. The developers can also configure the rules of the database to deny or allow certain users to read or modify certain records. Another advantage is that it can work connectionless, that is, if a client loses internet connection it can still see and work with the data and then when the internet connection is back the database will be synchronized. Its biggest problem is that it does not allow the execution of complex queries, but that should not be an issue in this case.

In order to cope with the task of authenticating and managing the accounts of the users, another Firebase tool will be used called *Firebase Authentication* [10]. It offers services to authenticate users through several methods: email, Google, Facebook, Twitter, phone

number... It can be integrated in applications, and it uses standards like OAuth 2.0 and OpenID Connect.

2.3 PYTHON LIBRARIES

The most important part of the project is probably the machine learning one where Python was used. While most people are familiar with Python, it is still be interesting to explain the specific libraries and algorithms used.

First, Pandas was used to transform the raw data to a more suitable form to be inputted in the algorithms. Pandas is a Python software library widely used for data analysis and manipulation [11]. It is characterized by the use of a certain data structure called the Dataframe which is a 2D structure used to store different data types (strings, integers, floats,) similarly to a SQL table [12]. Pandas was a really helpful solution for data cleaning as the data can be imported from csv or JSON files into Dataframes which can easily be manipulated using several operations the library contains, such as reshaping or sorting. The Dataframes can then be exported to different types of output files which can then later be fed to the algorithms.

Matplotlib is a Python library used to draw plots from code [13]. In the project it was used for the visualization of the results obtained, as this was advantageous in many cases to get a better understanding. This relatively basic library allows the user to draw scatter plots, histograms, pie charts and others, while also including functions to modify the shape of markers, the grid, the labels, or the line of the plot.

Scikit-learn is a popular machine learning library that includes several algorithms like linear regression, k-means, SVM and others [14]. Scikit-learn is a great tool for machine learning beginners as it has a great amount of documentation and several examples on the Kaggle website. It also includes functions that can be used to obtain the optimal parameters to minimize the error. It might be confused with Tensor flow, but while Tensor flow is a lower-level library commonly used in deep learning (like neural networks), Scikit-learn is high level and used for machine learning.

2.4 MACHINE LEARNING ALGORITHMS

Three different machine learning algorithms were used to estimate the value of a property. Before this is explained with more detail it is essential to know the basics about these algorithms to better understand the process and the results obtained.

2.4.1 LINEAR REGRESSION

The assumption here is that it is a linear model which means that the input and the output have a linear relationship [15]. The goal is to determine the coefficients that define this relationship. The most straightforward type is the Simple Linear Regression where there is only one input variable, so just two coefficients must be determined according to this formula:

$$Y = a + b * X$$

- X is the input variable and Y the output variable.
- a is called the intercept and b the slope

Basically, here we will try to determine the output value based on an input by finding out the constant and the value that modifies the input. The result will be a straight line with the input variable in the x axis and the output in the y axis.

This is an example of a simple linear regression, where the input variable is the years of experience of an employee, and the output is the salary. The red line represents the relationship obtained with the chosen coefficients and the blue points are the actual values. Ideally, all the blue points should be on the red line as that would mean the model works perfectly. Unfortunately, this rarely happens in the real world as we will see later.



Figure 1: Simple linear regression example [16]

When n input variables are involved, the objective will be to find the value of $n+1$ coefficients, one for each input variable plus the constant. The line would now be a $n-1$ hyperplane

To obtain the coefficients there are different approaches but the most used is Ordinary Least Squares which tries to minimize the sum of the squared residuals (differences between the points and the line of coefficients).

This is one of the simplest machine learning algorithms and although it works for some cases, it is generally not good enough. The problem is that Linear Regression assumes the relationship between input and output is linear which is not often the case. Linear Regression also performs poorly when the variables used are somewhat noisy or when the input variables are highly correlated.

2.4.2 RANDOM FORESTS

Before getting into Random Forests, we first have to talk about Decision Trees. A Decision tree is a relatively basic structure in which the output is obtained by answering several questions about the input variables. For example, here is a simple decision tree to determine what a soldier should do next. Based on the input variables (vision of enemy, level of ammunition and supplies) the soldier will decide its next action.

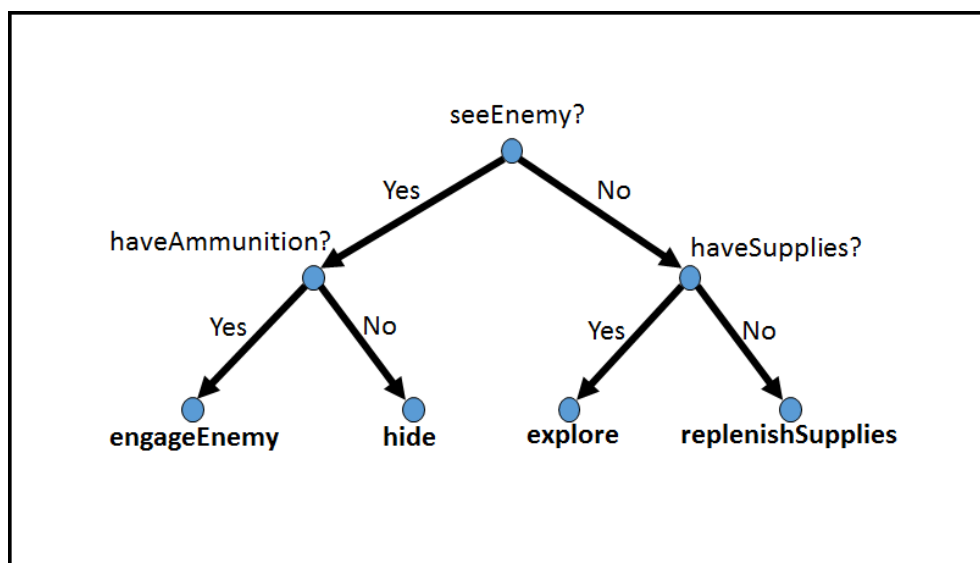


Figure 2: Example of a Decision Tree [17]

Random Forest uses several different Decision Trees for the same problem so they will have different questions and branches [18]. As a result, the same input may produce a different output for two different Decision Trees. After constructing the different Decision Trees, they will be fed the same input variables and therefore each one will output the resulting outlook. The output of the Random Forest will be the most repeated output of the trees.

The advantage of this method is that uses uncorrelated models to determine the result, so if a tree has some faults, they will be corrected by the other trees. This is why it often overperforms many other methods where the models are somehow correlated.

2.4.3 GRADIENT BOOSTING REGRESSION

The idea is to create weak models and then combine them to obtain a strong model. It is gaining popularity as it usually produces a good result. These are the main steps of the procedure [19]:

1. We first use the mean as a prediction and calculate the residuals or prediction error, which is the result of subtracting the predicted value (in this case, the mean) to the actual value
2. We then build a simple regression tree using x as the input variable and the residuals obtained as the output. This is a possible example:

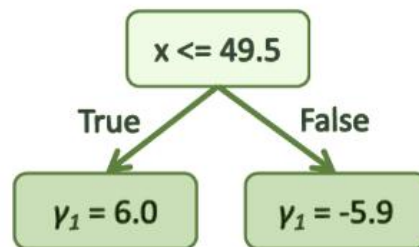


Figure 3: Tree obtained from residuals [20]

3. We add this new prediction scaled by the learning rate (v) to the first prediction.

$$P = P_0 + P_1 * v$$

The learning tree is a value from 0 to 1 chosen by the engineer. The closer to 1 it is, the more effect the tree will have on the prediction. Usually, a low value like 0.1 or 0.15 is used.

4. We use the new residuals to create an updated tree like in step 2 and step 3.
5. Repeat the procedure as many times as desired. With every iteration the prediction gets closer to the actual value.

Chapter 3. STATE OF THE ART

3.1 ZILLOW

One could argue that this is the most popular Real Estate website [21]. It can be used both by realtors and the public and gives the user a seamless experience with pictures and videos (if provided by the lister) for each of the properties. Zillow also has an app that continues the experience from the website with a well put-together user interface. The app was first launched in IOS, and it is now also available for Android. Zillow separates properties by zip code, making it easy for the user to view listings if they have certain areas or neighborhoods in mind. Another key characteristic of Zillow is its huge database, as it contains data from 110 million homes across the USA. It also has a tool called Zillow Advice where people can contact experts about questions they may have about listings.

Zillow's premiere feature is Zestimate; this home valuation model calculates estimates for home prices based on numerous factors. Zestimate is calculated using a state-of-the art neural network-based model that incorporates data from these sources [22]:

- Several home features, including square footage, the number of bedrooms and bathrooms and location
- On-market data such as listing price, similar homes in the same zip code and days on the market
- Off-market data like tax assessments, prior sales and other publicly available records
- Market trends, including seasonal changes in demand

Zestimate is quite accurate in its prediction as the mean error for on-market homes is 1.9 % This error varies depending on the area studied due to several reasons: higher prices, different taxes, inaccurate data... Here is a table with the error rate for some of the biggest metropolitan areas of the United States.

Metropolitan Areas	Median Error ⓘ	Homes With Zestimates ⓘ	Within 5% of Sales Price ⓘ	Within 10% of Sales Price ⓘ	Within 20% of Sales Price ⓘ
Las Vegas	1.4%	10.0K	91.6%	98.1%	99.5%
Los Angeles-Long Beach-Anaheim	2.0%	27.3K	83.6%	96.4%	99.5%
Miami-Fort Lauderdale	2.3%	50.2K	81.5%	95.2%	98.9%
Minneapolis-St Paul	2.0%	17.3K	83.1%	96.8%	99.2%
Nashville	1.5%	9.8K	87.3%	96.2%	98.9%
New York	2.4%	107.7K	78.6%	94.3%	98.8%
Orlando	1.9%	14.2K	87.5%	97.4%	99.4%
Philadelphia	2.2%	28.3K	79.7%	93.7%	97.5%
Phoenix	1.4%	19.9K	89.9%	98.2%	99.7%
Pittsburgh	2.6%	9.1K	73.7%	91.4%	98.0%

Figure 4: Zestimate median error for active listings in different metros of the US [23]

We can see that the algorithm is more accurate in cities like Phoenix, Las Vegas, or Nashville. On the other hand, it struggles in New York, Pittsburgh, and Philadelphia but it is still a pretty good indicator in those areas.

For houses that are off market the median error is 6.9 % much higher than before. The problem is that now the algorithm does not have access to many features like the listing price or the days on market, so it obviously won't be that accurate.

Metropolitan Areas	Median Error ⓘ	Homes With Zestimates ⓘ	Within 5% of Sales Price ⓘ	Within 10% of Sales Price ⓘ	Within 20% of Sales Price ⓘ
Las Vegas	4.4%	719.3K	55.1%	80.5%	93.9%
Los Angeles-Long Beach-Anaheim	5.4%	2.9M	47.1%	74.3%	91.9%
Miami-Fort Lauderdale	6.7%	2.2M	39.5%	65.5%	87.5%
Minneapolis-St Paul	5.7%	1.2M	45.1%	72.0%	90.7%
Nashville	6.0%	682.7K	43.3%	67.7%	85.9%
New York	8.5%	5.6M	31.9%	56.7%	81.7%
Orlando	5.3%	842.4K	47.4%	73.6%	90.8%
Philadelphia	8.1%	1.9M	33.4%	58.3%	82.6%
Phoenix	5.3%	1.5M	47.9%	74.5%	91.9%
Pittsburgh	10.2%	820.3K	27.5%	49.5%	74.5%

Figure 5: Zestimate median error for off-market houses in different metros of the US [24]

Notice that while all of them are higher than before, the ones with the lower error are still the same cities (Las Vegas, Phoenix, Orlando) while cities like New York or Pittsburgh are once again getting the highest error.

Even considering the error rate increase, the prediction is still decent, and more than half of the houses are predicted within a 10 % of the house price in all metros except Pittsburgh.

3.2 REALTOR.COM

This website offers a list of for-sale properties and the information and tools to make real estate decisions [25]. Realtor.com is mainly used by home shoppers, as they define themselves as The Home of Home Search. It is known for its accuracy and dependability. Realtor.com also has a feature called My Home dashboard which allows homeowners to manage their homes through tools like tracking the home value over time or visualizing other similar properties in the same zip code. Being affiliated with the National Association of Realtors certainly has its perks for Realtor.com. Realtors.com is also available as a mobile app which has an interesting feature which allows users to draw a shape on the app's map so that the user can focus on the properties within the shape's area. This ability to draw a region on the app's map is a feature that may be integrated in the project. Realtor.com's app gives a nice outlook on what to expect.

3.3 REDFIN

This website started in Seattle in 2004 and has not stopped growing since [26]. It is used by homeowners to post their own homes and by potential buyers to search for listings. What makes them different is their idea to try to undercut competition. They make sellers pay Redfin a discounted fee, either 1 or 1.5 % to list the seller's home. Like Zillow, they also have a tool that can estimate the price of a house so home sellers can get an idea of how much their properties are worth. However, this estimator has a median error rate of 8.7% for off-market houses and 2.99 % for on-market houses. Redfin's estimator is less accurate than Zillow's estimator which has a median error rate of 6.9% for off-market houses and 1.9 %

for on-market houses. A neat feature on Redfin's app is the ability to take 3D tours of home listings. A feature as complex as this is not always available, but it is certainly intuitive and worth looking into to get ideas.

3.4 TRULIA

Trulia is an online real estate marketplace which aims to facilitate buyers and sellers function using recommendations, local insights, and map overlays [27]. It was acquired by Zillow in 2014 for \$3.5 billion and is now a Zillow's subsidiary. It contains tools like Trulia Neighborhoods that enables home buyers, sellers, and renters to find key information about a certain neighborhood they may be interested in. Another interesting tool is What Locals Say, where home buyers and sellers can inquire on what local residents think about the neighborhood they live in through polls and reviews. Like Redfin, Trulia is worth looking into because of its app; specifically, how the app which makes it as easy as possible for users to search for homes in certain neighborhoods. It is certainly worth looking into Trulia's ease of access and user interface because it can help get an idea for the future development of the app

3.5 OPENDOOR

This is a company which buys and sell houses online. It utilizes an innovative way to make these transactions which is quickly gaining popularity in the Real Estate market.

The iBuying, which is the denomination for this, does not require the homeowner to put their house on the market. Opendoor will make an offer to them based on the features and market data [28]. Should the owner accept the offer, the movement and the payment will be completed online. After buying the property, Opendoor will make the necessary repairs and improvements and then it will post the house on their website so that it can be sold.

This process eliminates the need for Realtors as Opendoor will act as an intermediary between the seller and the buyer. The way they make profit is by charging a fee to the people

selling their home. This fee is around 4.5 % which is slightly lower than the one usually charged by the Real Estate agents (5-6%) and the process is easier for the owner as in most cases they do not need to physically show the house. There are still some concerns for them like receiving an offer that is not good enough or they may be charged for some of the repairs needed in their house.

Opendoor has a feature where the user can see how much is his or her house worth [29]. They will need to input the address and some features, and a price tag will be computed. Opendoor claims to build the algorithm based on millions of data point, the current situation of the house market and most importantly on details inputted by the owner like the condition, reforms (ex if the construction of basement is finished) and also on how safe the neighborhood is or the noise levels perceived in the house. Adding these characteristics to the algorithm is what differentiates it from the Zestimate by Zillow. Opendoor will then show the final value to Real Estate agents for an expert opinion before making an offer to the owner.

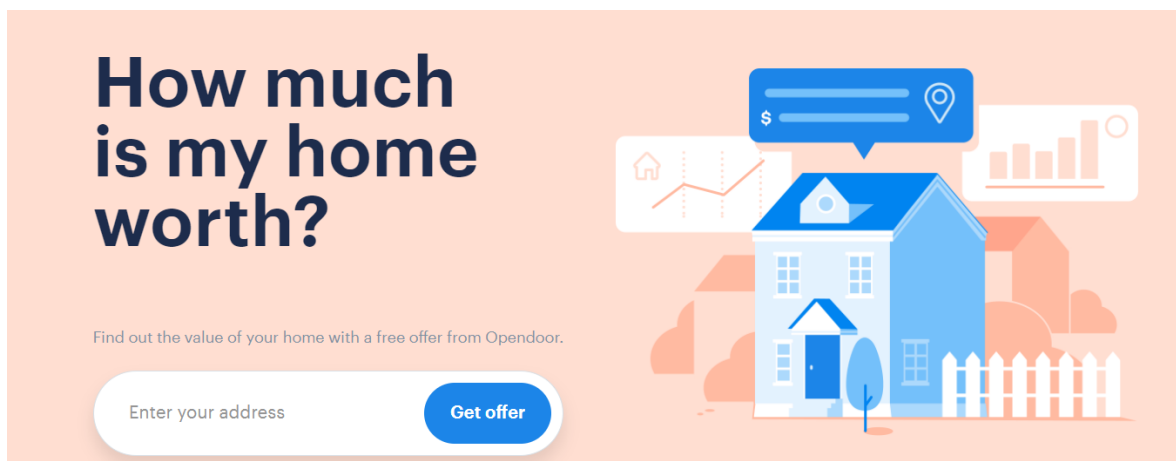


Figure 6: Price estimator by Opendoor [30]

Unlike in Zestimate, the error rate cannot be computed with this algorithm as the selling price is the same as the prediction, so it is hard to compare both of them. Opendoor claims that 92 % of their clients are satisfied with the offer they received, but this may be biased as

some of them might underestimate the value of their house. It could be argued that Opendoor gets a better understanding of the house value because of the added features in the valuation but the huge database used by Zillow and their strong data model cannot be disregarded.

Opendoor also incorporates a service to calculate the potential net profit of selling the house by two alternative paths [31].

The first is the traditional way in which the user will need to input the home price and the website will estimate the necessary fees:

- **Real Estate Agent Fees:** around 6%
- **Staging and Prep Work:** around 1%
- **Seller concessions:** between 1.5% and 2%
- **Home ownership and overlap costs:** around 1 %
- **Title, escrow, notary, and transfer tax:** usually from 1 to 3%

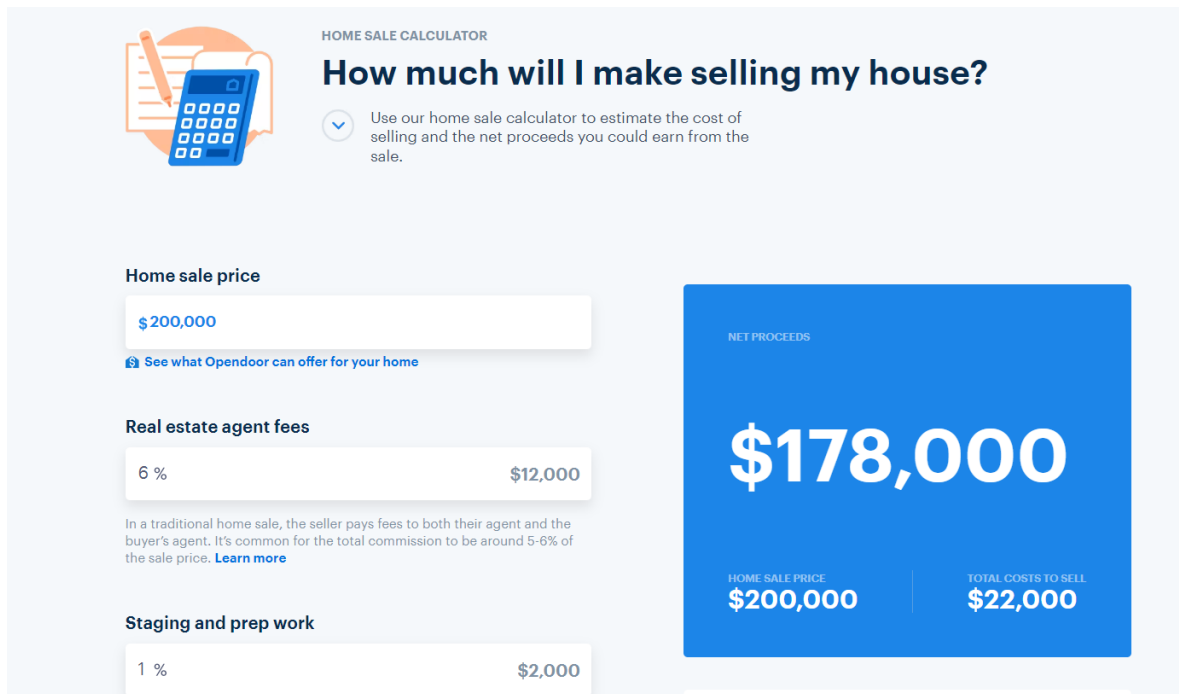


Figure 7: Net Profits calculator by Opendoor [32]

The second is with iBuying, so the website will use the previously mentioned algorithm to make an offer to the owner. The offer will not include many of these fees so the net profits are usually larger.

Chapter 4. PROJECT DEFINITION

4.1 JUSTIFICATION

In the State of the Art section, several websites and applications about the Real Estate market were exposed. While they all have their own characteristics, they have one thing in common: their target users. These platforms are all oriented for the general public: the people looking to buy or sell a house and the ones which are just curious and want to take a look at the listings. Zillow does also have a space for realtors, but that is not the main use of the page.

The application proposed, however, is exclusively made for Real Estate Agents as they will be able to access to the application through a subscription plan not available to the general public. This is what makes this project different from the current solutions. As mentioned in the introduction, the number of realtors is increasing, and the market is getting more and more competitive, so an application created to help them stand out is long due by now.

The goal of the application is to save thousands of hours of the realtors' time as they spend too much effort contacting owners who end up rejecting the offer of selling their home. The plan to achieve this is to determine how likely is an off-market house to be put on the market. Using this information, the agents will only contact those owners who are predisposed to selling, as a good amount of them will probably accept the offer to sell. Realtors might also stay away from those houses which are deemed as unlikely to be sold in the near future as contacting the owners will probably be useless.

The prediction is obviously not perfect. There will be owners who won't still sell their home even with the application determining the sale is likely and vice versa. Realtors have to understand that and not always take the prediction religiously. Even considering this, the application intends to be a major help for realtors, filtering down several houses of each neighborhood and boosting the success percentage of the realtors' contacts with

homeowners. None of the apps mentioned in the previous section had anything like this feature.

The clients of the project mentioned that there were some big Real Estate Agencies who had internal software that helped the agents obtain leads. There is not much information on how they achieve this as these programs are usually kept a secret from the public for obvious reasons. We do, however, know that the agents who want to use it have to be members of those agencies. The application of the project is available to every realtor of the country whether they are part of an organization, or they work on their own.

Now that almost everybody has a smartphone, they will be able to access these leads anytime and anywhere. Also, the small subscription fee is something worth paying as the realtors using this app are expected to make a better use of their time and therefore sell more houses with the commission they bring. Realtors' commissions are usually worth thousands of dollars, making the app fee almost insignificant.

The prediction of the likelihood of a house being sold requires first to estimate the current price of the house based on the current market and the features of the house. We have already seen both Zillow and Opendoor value estimators and explained how they differ from each other. The prediction for the application is not expected to be as good as these two websites. The reason is that this project does not remotely have the same resources as the amount of listings data, computational power, data models and the Real Estate insight provided by experts. While the value estimator might not be as accurate, the plan is to obtain a decent price prediction that will not negatively affect the final likelihood prediction.

4.2 OBJECTIVES

This are the original objectives the project should satisfy:

- Create a cross-platform application able to run on Android and IOS. The user interface must be friendly and intuitive
- Manage the user accounts and the authentication functionalities while maintaining privacy and security. User must be able to create an account or sign into their account using their email/password or using Google Authentication
- In the filter screen the user must be able to move filter bars to select the desired range on the features provided. Once user clicks the “Filter” button, the map with filtered properties should appear
- The map screen must show the filtered listings with information about each property. The properties must be color coded based on the likelihood to be sold. The user has the ability to traverse through the map and bookmark the listings they are interested in
- The user must be able to draw farms in the Map screen. A farm is an area within a zip code which might be interesting for a realtor. The user can save a farm in the profile and see the data of the listings within.
- In the Bookmarks screen the users must be able to view all their saved listings in the order in which they saved them. The user can click on one of them to view more details and add comments on it
- There must be a profile screen where the user information is displayed in one location. This screen must also show the activity log of all the actions the user makes like bookmarking, deleting a bookmark or commenting a bookmark. The user’s bookmarked listings must be saved within their profile so they can be accessed whenever they log back into the application. From the profile screen the user must also be able to log out of the account
- Develop a value estimation algorithm to predict the value of the property based on its features. These features include square footage, lot size, number of bedrooms,

number of bathrooms and their type (quarter bath, half bath, three quarter bath and full bath), area code, zip code and whether it has a pool or not. The average error rate must be around 15 %

- Implement an algorithm to predict the probability that the owner of a house in a zip code will sell the property

4.3 METHODOLOGY

4.3.1 FRONT END

To develop a cross-platform application that works in Android and IOS, instead of making two different applications (one for each operative system), the ideal solution is to develop just one application using a language that will be able to run in both systems. In this case the chosen technology was React.js which is a JavaScript library already discussed. React.js has many elements that help in the task of giving the user interface a nice appearance and making it easy to use.

The front end will be divided into several screens, having one file per screen. Each screen will use different elements of React.js: for example, the filter screen will use sliders to set the filters and the map screen will incorporate the Google Maps API. There will be a navigation bar to move between the screens.

4.3.2 BACKEND

The backend will also be split into files, each with different functions which will be called from the front end.

There will be functions used for the sign up and sign in of users. There will be a function for each of the two possible sign in ways: using the email address and using the Google account. All of this will be processed using the Firebase Authentication tool as it really facilitates the management of user accounts.

The information about the listings is stored as a JSON tree in the Firebase Realtime Database. This database contains all the features including the address, square footage, number of bedrooms and bathrooms... Apart from the table with the properties, there is another table containing the users' information like the first and last name, email address and bookmarked listings.

The backend will read and update the database using a connection with a token. The database must have security rules to prevent people accessing and modifying the data from the outside as the application must always ensure the privacy and security of the data.

4.3.3 MACHINE LEARNING

The data needs to be collected and cleaned before being input into the price estimator. A Python script is created to handle this task, removing unnecessary columns, and changing the format of others.

The price estimator will be obtained with a machine learning approach, using the Python's library called Scikit-learn. Multiple algorithms are implemented: Linear Regression, Random Forest and Gradient Boosting Regressor. For each of these algorithms several hyperparameters are tried, choosing those which minimize the average error of the training set.

The algorithm that computes the likelihood of being sold uses the estimated value of the house to predict the potential return on investment (ROI) the owner would get by selling the house at that moment. Then, if the ROI is high, it probably means that the owner is more open to sell as it would be a good time for it. Therefore, that house would be designated as likely to sell.

4.4 WORKPLAN AND ESTIMATED BUDGET

4.4.1 WORKPLAN

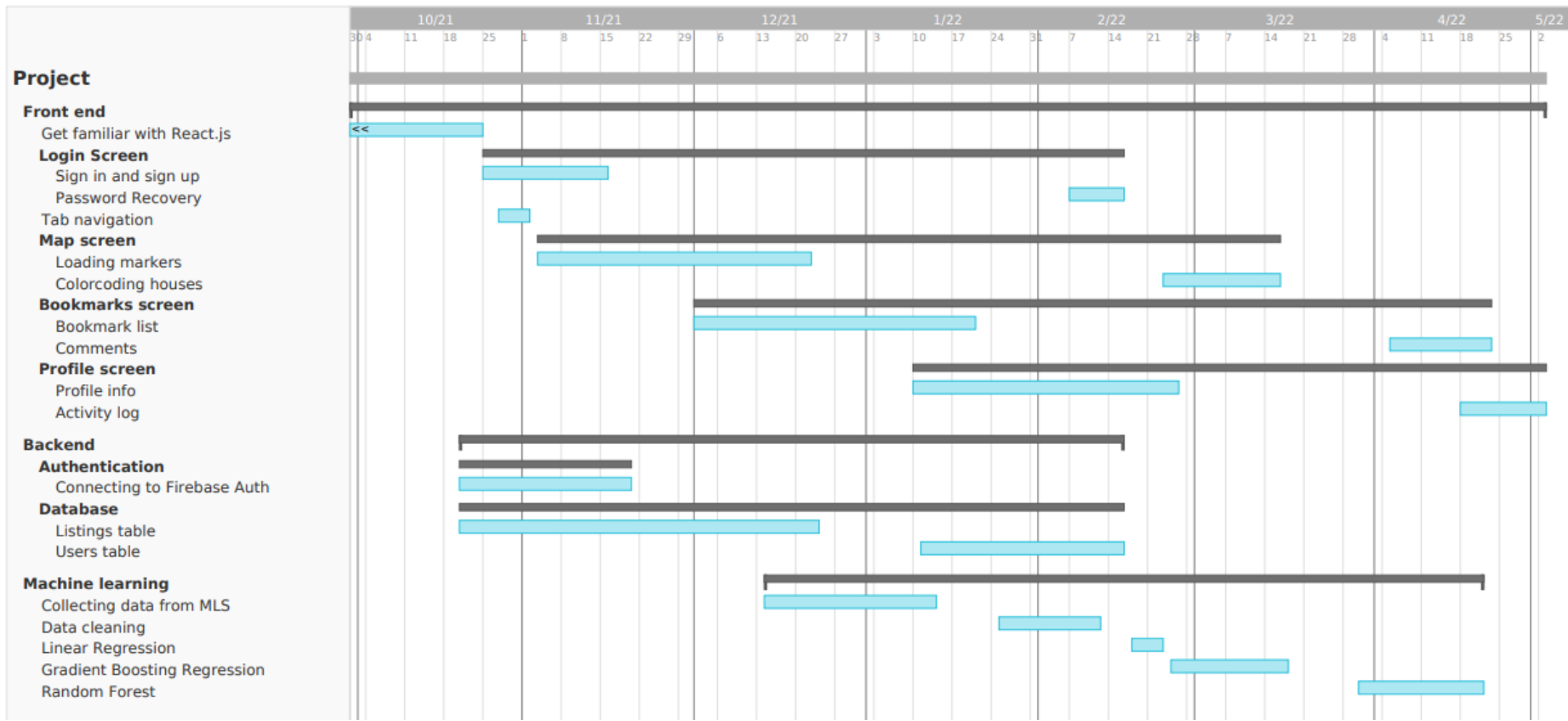


Figure 8: Workplan of the project

The workplan is divided into 3 different sections: front-end, backend and machine learning.

The first task was to familiarize with React.js. I coded a small app to understand better React.js and learn new concepts as this library was completely unknown to me. The intention was to make the future app development much smoother and faster.

The first part of the year the focus was the front end. The front end tasks are divided into the different screens of the application: login, map, bookmarks and profile. Each of them contained different features that were added during the year. Most of them were completed in the first semester but in the second semester discussion with the clients lead to modifications and the adding of other frontend features.

The backend was mostly consisting of the connections to Firebase and the management of the database. The database was divided into the Houses table and the Users table. Both needed to be created, filled with data, and most importantly security rules had to be implemented to prevent unauthorized access.

The backend was done parallel to the frontend because many front end features required access to the database or to the authentication service.

The machine learning part started much later than the others, spending most of the second semester on it. The reason is that the clients preferred to have a skeleton of the application first for approval before getting into the algorithms. The tasks included in this part are the collection and cleaning of the data and the implementation of the three algorithms already mentioned: Linear Regression, Gradient Boosting Regressor and Random Forest.

4.4.2 ESTIMATED BUDGET

The project only contains software components which means that there are not hardware costs, which are usually the bigger than the software costs. Different Google Firebase functionalities like Realtime Database and Authentication are implemented in the project and while all of them have a free trial, there is an extra charge after a certain number of uses. The application has not been released yet, as it is more meant as a proof of content. That means that the free trial is more than enough to cover the current requests which are merely done for testing and debugging purposes.

For the sake of this section let's assume the application is released in both the App Store and the Play Store. We can estimate that within the first months the app could have up to 1000 users so that is what will be used to calculate the costs.

The extra storage in the Realtime Database will be used to store more user data and listings from different zip codes around the country. The number of Gigabytes required will mostly depend on how many zip codes end up being included. There are currently no fixed plans on the zip codes expansion but let's assume we add thousands of zip codes that translate to 4 GB of space. User data will also increase the storage, but its influence is almost insignificant as not many fields are recorded within a user.

However, the number of users does directly affect the number of user verifications required in Firebase Authentication. The first time a user logs in with a new device, this service is used to verify their identity, so let's assume the number of uses is around 1200 per year.

Moreover, every time the map screen is loaded, the application calls the Google Maps API. Again, if the monthly number of calls is below a threshold, the service is free. But with 1000 users, the number of calls will increase considerably and there will be charges added to the Google account. Based on the number of users, we could make an estimation that around 500000 calls are made per year.

Finally, we must not forget that both the Google Play Store and the IOS App Store charge a fee for posting the application on their platforms. For Google Play, the cost is just a one-time

\$25 fee. The App Store is much more expensive, demanding a \$99/year subscription for this service.

Taking all of this into account this is a breakdown of the cost of the application in the first year and with approximately 1000 users.

Project Costs for Application in the first year				
Item	Quantity	Description	Unit Cost	Extended Cost
1	500000	Google Maps API calls	\$0.002	\$1000
2	1200	User verifications in Firebase Authentication	\$0.01	\$12
3	4	Extra GB of storage in Firebase Realtime Database	\$5	\$20
4	1	Google Play fee	\$25	\$25
5	1	App Store fee	\$99	\$99
Total Cost				\$1156

Table 1: Cost breakdown for the first year

Chapter 5. DEVELOPED SYSTEM

5.1 DESIGN OVERVIEW

The image below gives a basic overview of how the components of the project interact with each other. It is high level but it still helps the reader understand the flow of data.

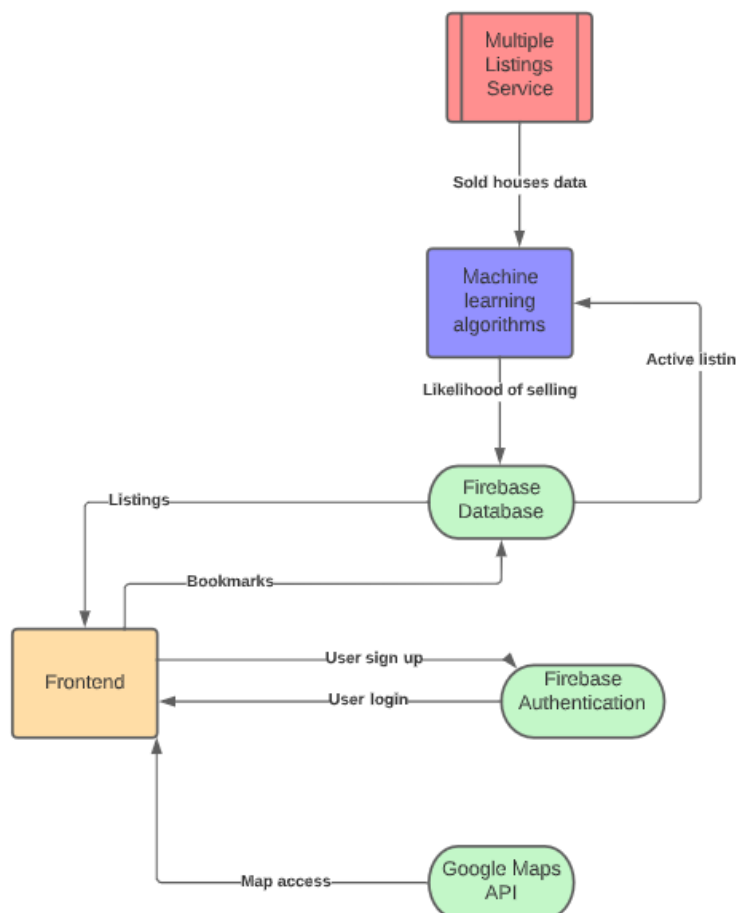


Figure 9: Basic overview of the project

The main component we will find is the front end. This is what the user can see, the different screens with buttons, maps, labels, etc. Every mobile application must have a front end in order to present the content in a visually pleasant way to the user. The front end cannot work on its own as it would be just a shell, something which looks good, but which does not have an actual functionality.

In this project the front end must have access to the database, which in this case is the Firebase Realtime Database. The connection is done using a React.js library and with the token of the Google account. As we can observe in the schema, the flow of the data will go both ways: the frontend will retrieve from the database all the data about the listings and then it will send back to the database information about which listings the user has bookmarked. The connection needs to be established when the map is opened to display the listings and when a user bookmarks or removes a bookmark.

The front end also needs the support from the Firebase Authentication service so that it can manage the auth process. Again, the communication goes both ways: the users that sign up will be saved to the service using the information they input in the register screen and when they are already registered the frontend will send the email and password to Firebase Authentication so that it can check if both are correct.

The Google Maps API is also a vital component for the front end as it is needed for the map screen. The call to the API is executed using a token from a Google Cloud Services account. In this case the connection is only in one way as the frontend will not send anything back to the API

Finally, we have the machine learning component. It will first collect the data from the MLS (Multiple Listings Service) and after the cleaning and processing, it will be used to train the model. After the training, it will need to get the data about the current listings and obtain their estimated price and how likely they are to be sold. Then these results are sent back to the database so that they can be displayed on the map Screen.

The following image also shows the interactions between the modules but this time at a lower level as we can observe how is the flow between the different screens of the frontend. Plus, it is interesting to see which screens are connected to the outside modules.

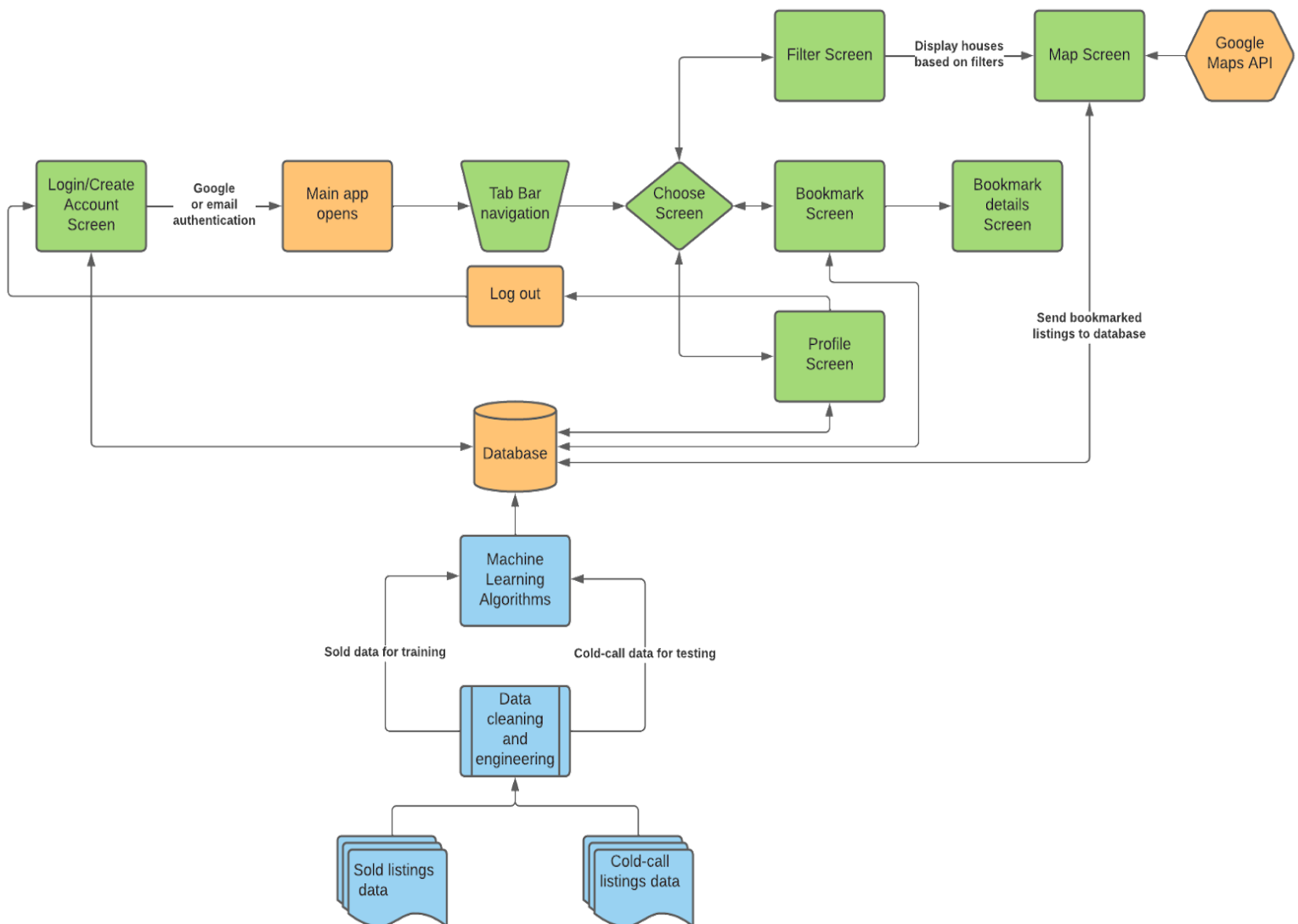


Figure 10: Detailed overview of the components of the system

The first screen we arrive in is the login screen. Here the user can choose between signing in using Google or with their email. They may also create a new account using their email if they don't have one yet. These functionalities are all provided using the connection to Firebase Authentication.

Once the user is logged in, the main app will be opened, and a tab bar navigation will appear at the bottom of the screen. This bar will allow the user to move across the three main screens: filter, bookmarks, and profile. After being logged in, the user will be directed to the filter screen by default.

The filter screen will show the different possible parameters that can be adjusted to limit the search of listings. Once the user has selected its decided values, the listings satisfying all of them will appear on the map screen. Here the user can scroll around and click on a listing for details and bookmarking.

From the map screen is where the Google Maps API will be called so that the map can appear. This screen also requires the two connections to the database previously mentioned: one to load the listings data and the other to send to the database the listings that are bookmarked by a user.

The bookmarks screen is where the user can see all of its bookmarked listings in a list. Also, by selecting one he will be able to access the bookmarks details page where extra features appear. In this page it is also possible to delete a listing or add comments to it.

The bookmarks screen also needs a connection to the database to retrieve the listings bookmarked by the user that is currently signed in. Moreover, the bookmarks details page will have to send information to the database to update it if the user decides to remove a listing from its bookmark or to comment something on it.

Finally, we have the profile screen. Here the user can look at its personal info and at an activity log which captures some of the movements made by the user in the app. This also requires access the database, and more specifically, to the users table.

5.2 FRONT END

In this section each of the front end modules will be explained with details, adding screenshots of the app and code snippets to help the reader get a better idea of how the application works.

5.2.1 LOGIN SCREEN

The user will be redirected to this screen the first time they open the application. The first operation will be the login where the user can choose between Google sign in or email sign in. There is also the possibility to create a new account if the user does not already have one in the app.

5.2.1.1 Google Sign-In

This is the process followed when the user decides to sign in using Google

- 1- The user must introduce the email address associated with the Google account he or she wants to use
- 2- The application will check that the introduced address is valid
- 3- The user will be asked to introduce the password of the account. If the password is correct, the user will be asked to agree the terms and conditions of Google to complete the verification of the account.
- 4- The user can log in and will be redirected to the filter screen.

Below there are two snapshots of the application that show the process :

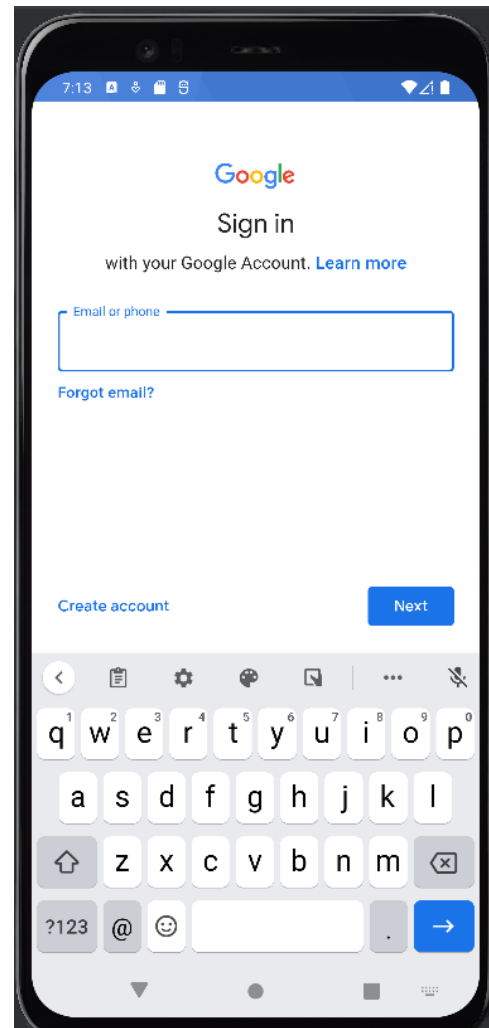
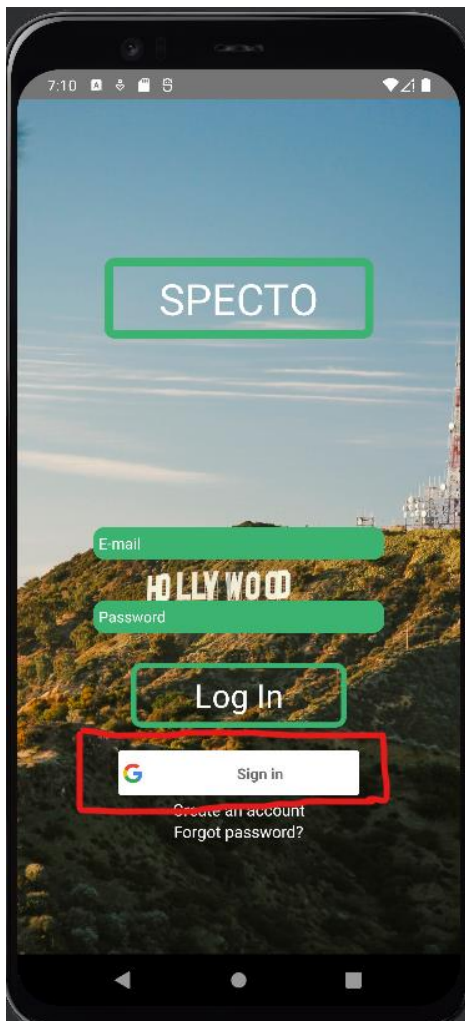


Figure 11: Login Screen with Google Sign in Figure 12: Email screen with Google Sign in

These are some of the most important lines of codes used for this functionality:

First, we need to import react-native google sign in library and add the client id.

```
import {GoogleSignin} from '@react-native-google-signin/google-signin';
GoogleSignin.configure({
  webClientId:
    '709665518531-
rd4e6n9redmmq0jg7uc8lc43bn6nfcph.apps.googleusercontent.com',
});
```

We will use the *signIn()* function which will initiate the Google Sign in process. If the verification is correct, it will return a token id, which then can be used to get the Google Credentials.

```
const {idToken} = await GoogleSignIn.signIn();

// Create a Google credential with the token
const googleCredential = auth.GoogleAuthProvider.credential(idToken);
```

The Google credentials are used to sign into the application

```
const userCredential = await auth()
  .signInWithCredential(googleCredential)
```

Finally, the user must be added to the app database if it is his or her first time signing in with Google. The name and email address of the user is retrieved from the Google Credentials.

```
const user = auth().currentUser;
var ref = database().ref('users');
ref.once('value').then(function (snapshot) {
  var b = snapshot.child(user.uid).exists(); // false
  if (!b) {
    var newUserRef = ref.child(user.uid);
    const displayName = user.displayName.split(' ');
    newUserRef.set({
      uid: user.uid,
      first: displayName[0],
      last: displayName[1],
      email: user.email,
      numListings: 0,
    });
```

5.2.1.2 Email Sign-In

If the user decides to sign in using their own account, they will just need to input their email and password. If both are correct the user will be redirected to the filter page. If the password and/or the email introduced are not correct, the application will return an abnormal result. It will display an error message informing the user of the incorrect introduced data.

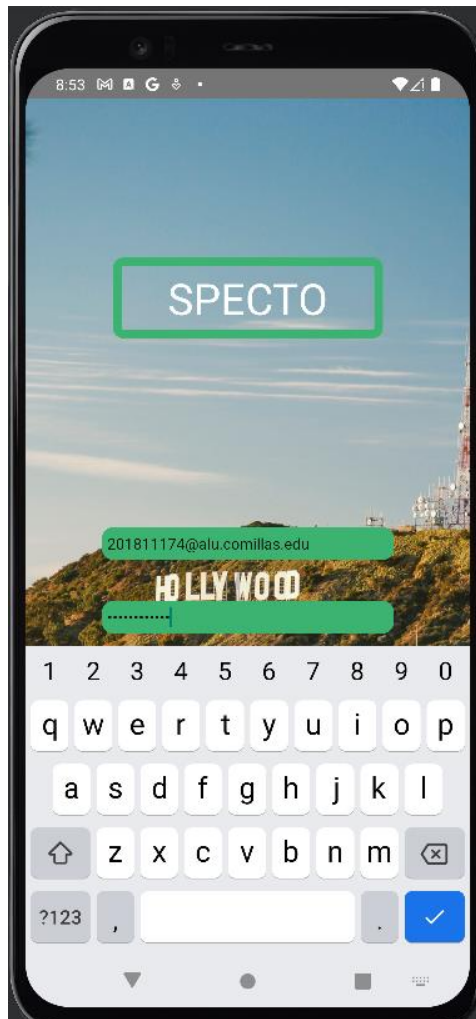


Figure 13: Login Screen with email sign in

In the code we will be using the Firebase Authentication function called *signInWithEmailAndPassword()* which will receive the email and password introduced by the user and try to log in.

Here we must include a catch clause to handle the possible errors. Some of the possible errors include introducing an invalid or already in use email address and writing an incorrect password. Plus, the email introduced might not be associated with any user or the user might have been disabled for several possible reasons.


```
auth()
  .signInWithEmailAndPassword(email, password)
  .then(() => {
    console.log(email + ' logged in!');
  })
  .catch(error => {
    if (error.code === 'auth/email-already-in-use') {
      console.log('This email address is already in use!');
    }

    if (error.code === 'auth/invalid-email') {
      console.log('This email address is invalid!');
    }

    if (error.code === 'auth/wrong-password') {
      console.log('Incorrect Password. Please try again!');
    }

    if (error.code === 'auth/user-disabled') {
      console.log('This user has been disabled');
    }

    if (error.code === 'auth/user-not-found') {
      console.log('No user corresponding to the given email');
    }
  })
```

5.2.1.3 Create an account

If the user does not have an account, he or she may create one with the following process:

- 1-The user must introduce personal data including first and last name and email address. The user must also create a new password.
- 2-Once the account is created the user is logged in and redirected to the filter screen.
- 3-The password and email can be used in the future to log in.

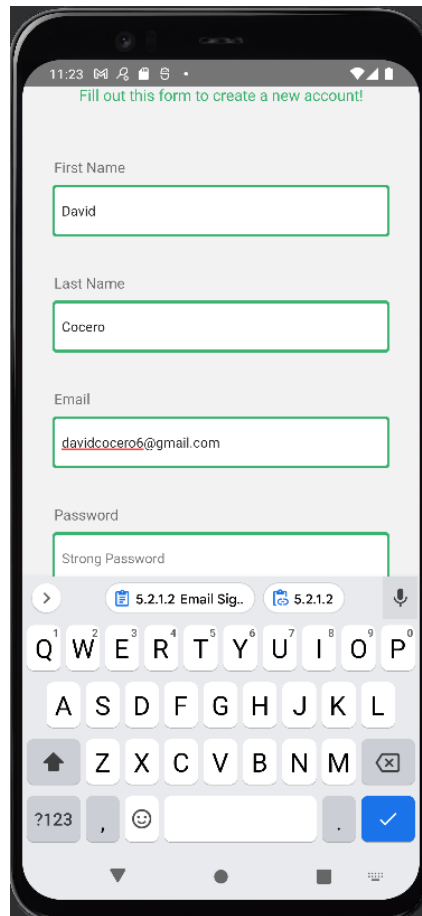


Figure 14: Create an account

We will be using the `createUserWithEmailAndPassword()` function, passing the email and password as argument. We also need to save the new user to the users table of the database.

In the database we will be storing the first and last name, the email address and the number of listings, which is originally set to 0. We will not include the password as the database will not be used for authentication and it would be a security risk to have it there. The key of the user (remember that the database is in JSON format) we will be using a `userid`, generated by the Authentication service.

```
auth()
  .createUserWithEmailAndPassword(email, password)
  .then(() => {
```

```
console.log('User account created & signed in!');
const userListRef = database().ref('/users');
const user = auth().currentUser;
// var newUserRef = userListRef.push();
var newUserRef = userListRef.child(user.uid);
newUserRef.set({
  uid: user.uid,
  first: first,
  last: last,
  email: email,
  numListings: 0,
});
console.log('User added to database');
})
```

There may be some possible input errors that the app needs to address, like the email being already in use or the password being too weak.

```
.catch(error => {
  if (error.code === 'auth/email-already-in-use') {
    console.log('This email address is already in use!');
  }

  if (error.code === 'auth/invalid-email') {
    console.log('This email address is invalid!');
  }

  if (error.code === 'auth/operation-not-allowed') {
    console.log('This email address is disabled!');
  }

  if (error.code === 'auth/weak-password') {
    console.log('This password is not strong enough!');
  }
})
```

5.2.1.4 Change Password

If a user does not remember the password of its account and wants to reset it, he or she may click on the *Forgot Password?* link.

- 1- The user will need to enter the email address of the account

- 2- An email will be sent to that address with a link containing the instructions for the change. The user can introduce the new password and the Authentication service will be updated

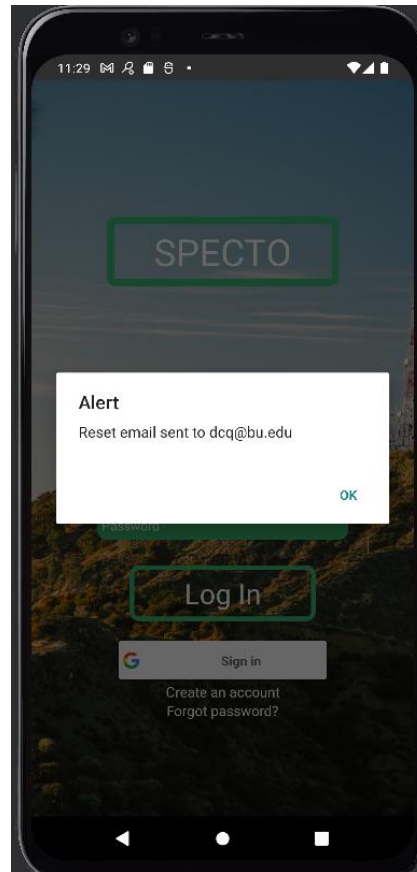


Figure 15: Reset Password

Here, we will be using the `sendPasswordResetEmail ()` function which will send the email to the address and reset the password.

```
auth()  
  .sendPasswordResetEmail(email)  
  .then(() => {  
    alert('Reset email sent to ' + email);  
  })  
  .catch(function (e) {  
    console.log(e);  
  });
```

5.2.2 FILTER SCREEN

This could be considered like the ‘Home Screen’ of the app as the user will be taken here by default after the login. The purpose of this screen is to help the realtors limit the listings that appear on the map screen. This can be extremely helpful if they are looking for a specific type of property.

The search can be filtered by the following 4 parameters:

- **Estimated Price:** this is the price prediction the machine learning algorithm produces for a house based on its features. Some realtors might be interested in expensive houses to get a larger fee and others might prefer easier sales of cheaper houses. The range goes from \$0 to \$10 million.
- **Year Built:** this is the year the house was constructed, so this time the choice is between old and new houses. The range goes from 1910 to 2022.
- **Living Space SqFt:** this is the size of the house (living space). Because we are working with houses in the US, the units will be Square Feet instead of Square Meters. The range is from 300 to 8100 SqFt
- **Number of bedrooms:** this just indicates how many bedrooms the property has. The range is from 0 to 8

The user can also select the zip code so that the result are the houses within the zip code that satisfy all the filters parameters. In this case the clients only asked to include one zip code: 90046 but the application is prepared for when more zip codes are added.

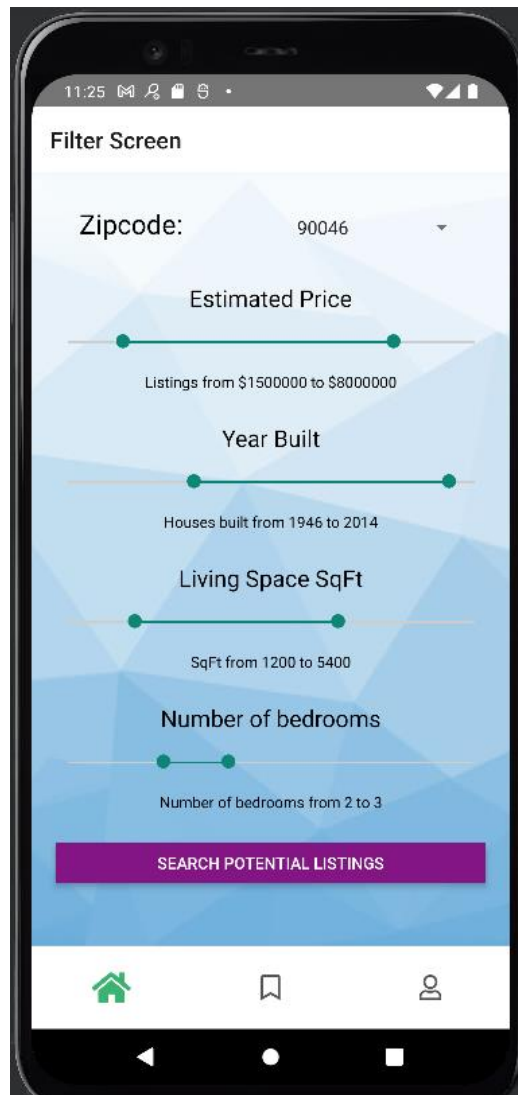


Figure 16: Filter Screen

The filters are coded using the *MultiSlider* element like in this example:

```
<Text style={styles.sliderText}>Living Space SqFt</Text>
  <MultiSlider
    sliderLength={350}
    values={SqFtValues}
    max={8100}
    step={300}
    onValuesChange={values => setSqFtValues(values)}
  />
```

```
<Text style={styles.labelText}>
  SqFt from {SqFtValues[0]} to {SqFtValues[1]}
</Text>
```

The filtering first requires pulling the listings from the database and cleaning them to extract the numerical values of the parameter.

```
filterHouses = () => {
  reference.once('value').then(snapshot => {
    var data = snapshot.toJSON();
    var houses = [];
    for (var house in data) {
      var id = house.toString();
      var price = Number(data[id].LP.replace(/^[^0-9.-]+/g, ''));
      var YB = Number(data[id].YB);

      if (data[id].SqFt === undefined) var SqFt = 0;
      else var SqFt = Number(data[id].SqFt.replace(',', ''));
      var Br = Number(data[id].BR);
```

Then the listings satisfying the parameters are put into an array and passed to the Map Screen as a parameter.

```
if (
  priceValues[0] <= price &&
  priceValues[1] >= price &&
  ybValues[0] <= YB &&
  ybValues[1] >= YB &&
  SqFtValues[0] <= SqFt &&
  SqFtValues[1] >= SqFt &&
  numBedValues[0] <= Br &&
  numBedValues[1] >= Br
) {
  houses.push(data[id]);
}
}
navigation.navigate('MapScreen', {listingsFiltered: houses});
```

5.2.3 MAP SCREEN

All the listing satisfying the filters will be displayed over a map. The house icons will be color coded to show the results of the machine learning algorithm. This color will show a prediction on how likely the house is to be put on the market. For this purpose, the houses will be divided into 5 bins, assigning one color to each bin:

- **Dark Green:** Really likely to be sold
- **Light Green:** Probable to be sold
- **Yellow:** Could be sold
- **Orange:** Not expected to be sold
- **Red:** Really unlikely to be sold

As the user scrolls around the map, the listings will appear and disappear to avoid loading them all at once, overloading the app.

Below there are two examples of how the houses appear color coded in the map.

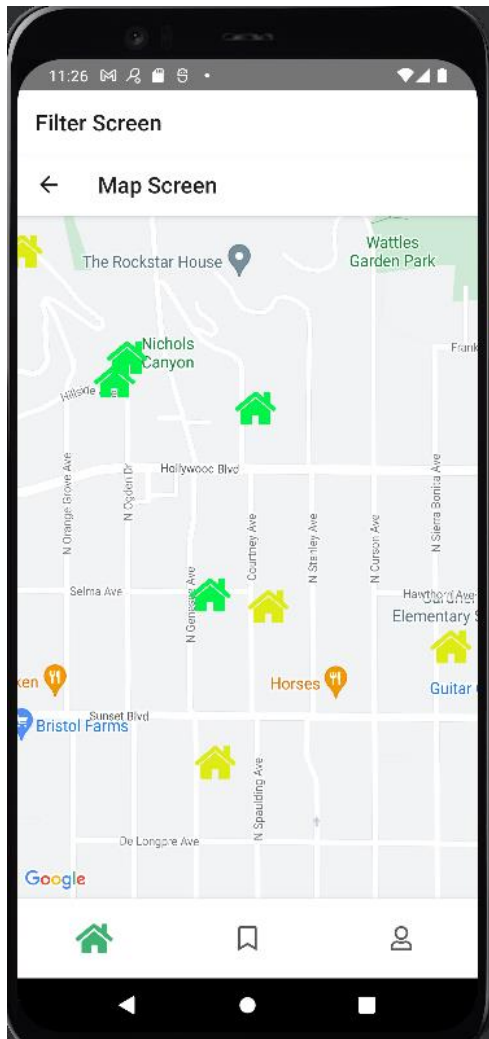


Figure 17: Map Screen I

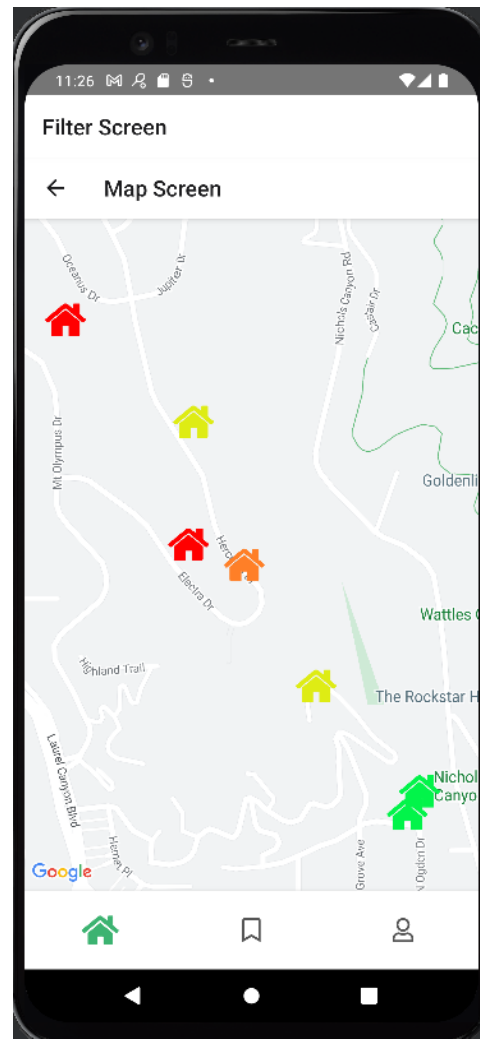


Figure 18: Map Screen II

If the user wants to see more details about a listing, he/she just needs to click on the icon. Then a bubble will pop up with some basic features like the address, the number of bedrooms and the surface in square feet. It will also display the estimated price obtained from the machine learning value estimator and the price per square feet which is also a valuable field for Real Estate Agents.

The bubble will also have a button to bookmark the property. This means that if a user is interested in something he/she may bookmark it to save it for later. This is a feature the clients insisted on including because in their career they felt many times that they did not have an easy way to remember some listings they were keen on.

In the image below there are two examples of how the bubble with the details of the listing looks like. In the first one, the listing is not bookmarked so the button will read “Bookmark Property”. In the second one, however, the user had previously saved that bookmark so this time it will show “Property already bookmarked” and if the button is pressed, the property will be removed from the user’s bookmarks.

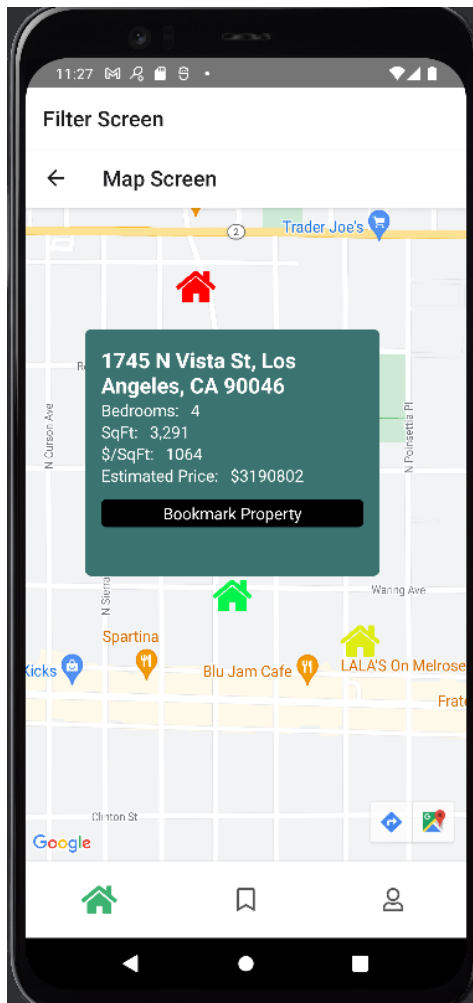


Figure 19: Map Screen with details I

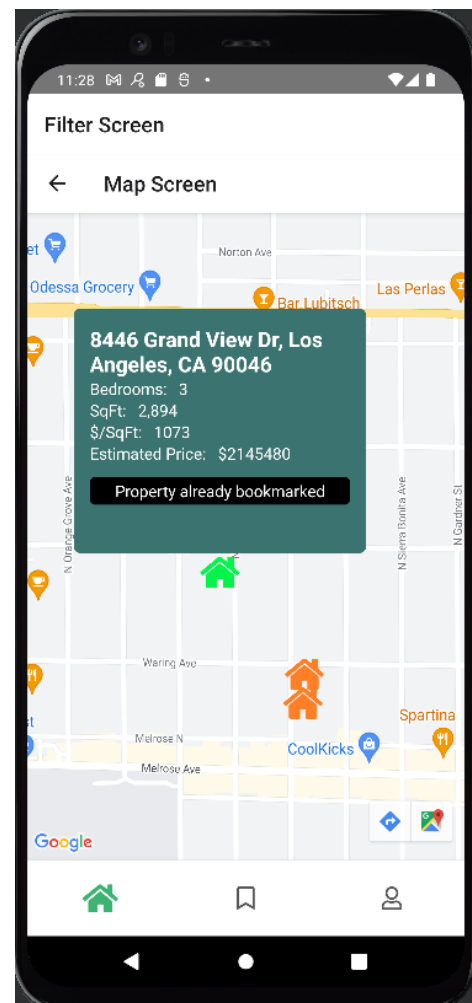


Figure 20: Map Screen with details II

It is interesting to highlight a few lines of the code like the map element *MapView* used:

```
<MapView
    provider={PROVIDER_GOOGLE}
```

```
style={styles.map}
initialRegion={region}
onRegionChangeComplete={region => setRegion(region)}>
{mapMarkers(region, listings, bookmarks)}
</MapView>
```

Thanks to using a state variable for the region of the map, the region can be moved dynamically as the user scrolls around.

Also, *mapMarkers()* is used to display all the house icons in the map. This function will receive the current region of the map (only the listings located there will be loaded), the array of listings received from the filter screen and an array with the bookmarks of the user.

In the *mapMarkers()* function, we will first iterate through the listing array, filtering them by their position in the map and assigning to each icon a color based on the bin they belong to.

```
if (
  Math.abs(house.lat - region.latitude) <= 0.007 &&
  Math.abs(house.long - region.longitude) < 0.007
) {
  if (house.bin == 1) icon_type = require('../images/red.png');
  else if (house.bin == 2) icon_type =
require('../images/orange.png');
  else if (house.bin == 3) icon_type =
require('../images/yellow.png');
  else if (house.bin == 4) icon_type =
require('../images/light_green.png');
  else icon_type = require('../images/dark_green.png');
```

Then it is time to create the *Marker* objects (the icons), setting their coordinates, reference, and id.

```
<Marker
  key={house.id}
  ref={_marker => {
    marker_ref[house.id] = _marker;
```

```

    }}
    coordinate={{latitude: house.lat, longitude: house.long}}
    image={icon_type}>

```

The *Callout* object was used to construct the bubble, including all the features and the button for bookmarking the property.

```

<TouchableHighlight style={styles.bubble}>
  <View>
    <Text style={styles.calloutTitleText}>
      {house['Full Address']}
    </Text>
    <Text style={styles.calloutSubtitleText}>
      {'Bedrooms:  '}
      {house.BR}
    </Text>
    <Text style={styles.calloutSubtitleText}>
      {'SqFt:  '}
      {house.SqFt}
    </Text>
    <Text style={styles.calloutSubtitleText}>
      {'$/SqFt:  '}
      {Math.round(
        parseFloat(house['Estimated Price']) /
        (parseInt(house.SqFt) * 1000),
      )}
    </Text>
    <Text style={styles.calloutSubtitleText}>
      {'Estimated Price:  $'}
      {Math.round(house['Estimated Price'])}
    </Text>
    <Pressable style={styles.calloutButton}>
      <Text style={styles.calloutSubtitleText}>
        {!listingsBookmarked.some(item =>
          shallowEqual(item, {key: house.id}),
        )
          ? 'Bookmark Property'
          : 'Property already bookmarked'}
      </Text>
    </Pressable>
  </View>
</TouchableHighlight>

```

Also, to update the listings of a user we will first modify the bookmarks array by adding the new listing id.

```
var listingsAux = [...listingsBookmarked, {key: house.id}];
```

If the listing is already bookmarked, we will remove this id.

```
listingsAux = [];  
for (var i = 0; i < listingsBookmarked.length; i++) {  
  if (i < index) {  
    listingsAux.push(listingsBookmarked[i]);  
  } else {  
    if (i < listingsBookmarked.length - 1) {  
      listingsAux.push(listingsBookmarked[i + 1]);  
    }  
  }  
}
```

We will finally save the array to the database to complete the update process. The text that appears on the button will also be changed at that moment.

```
setListingsBookmarked(listingsAux);  
database().ref(userRef).update({  
  listings: listingsAux,  
  numListings: listingsAux.length,  
});
```

5.2.4 BOOKMARKS SCREEN

The bookmarks screen will be the place where the user can take a look at the listings, they have previously bookmarked. They will appear as a list including their complete address (Street number, city, state and zip code) and their estimated price.

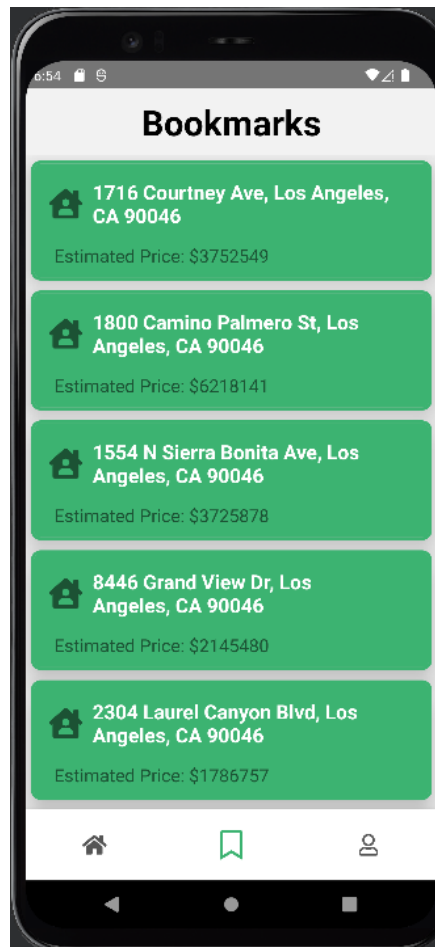


Figure 21: Bookmarks screen

The user can also click on a certain bookmark to see more details about it: the number of bedrooms, the surface is Square feet, the lot size in Square Feet, the estimated price per Square Foot and the total estimated price.

This screen also allows the user to delete the property from bookmarks and to add comments to it. For example a realtor may write "I have contacted the homeowner" on the bookmark to remember about it later.

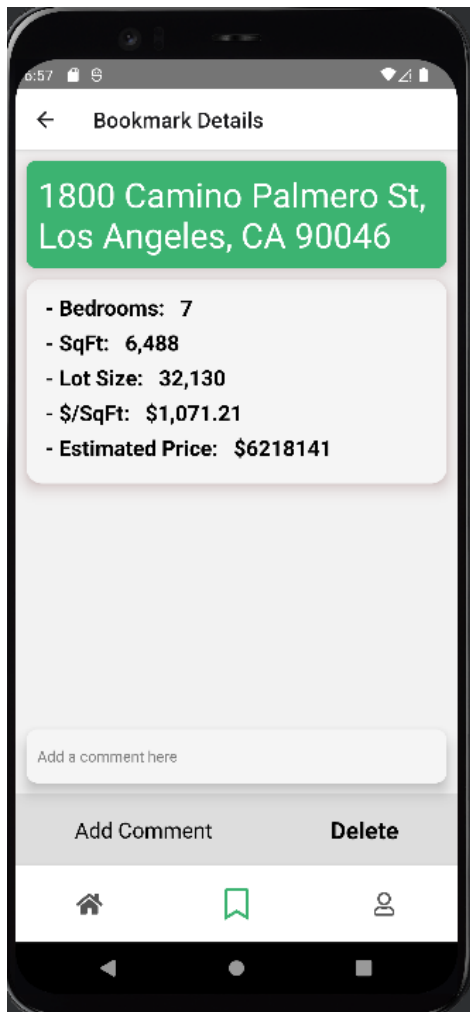


Figure 22: Bookmarks details screen

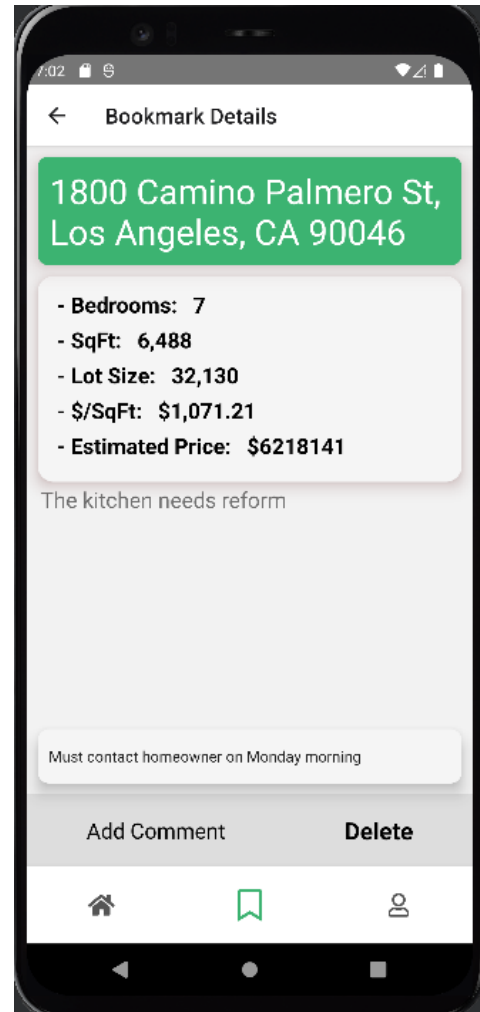


Figure 23: Bookmarks details with comments

To retrieve the bookmarks data from the database, we first need to obtain the ids of the house bookmarked from the Users table. Then we will use those ids to fetch the data of the listings from the Houses table.

```
database()
.ref(userRef + '/listings')
.on(
  'value',
  snapshot => {
    if (snapshot.val() === null) {
      setBookmarks([]);
    } else {
      var bookmarked = snapshot.val();
      listings = [];
    }
  }
)
```

```
for (let i = 0; i < bookmarked.length; i++) {
  if (bookmarked[i].key !== null) {
    var listingRef = '/houses/' + bookmarked[i].key;
    database()
      .ref(listingRef)
      .on('value', snapshot => {
        data = snapshot.toJSON();
        if (data !== null) {
          listings.push(data);
        }
        if (i == bookmarked.length - 1) {
          setBookmarks(listings);
        }
      });
  }
}
```

The listings will then be displayed using the *SafeAreaView* component. Within it, we will create a *FlatList*, feeding it the array of bookmarked listings we just fetched from the database.

We will include the address, the estimated price and a Touchable element that will take the user to the details page when pressed.

```
<SafeAreaView>
  <View style={styles.mainView}>
    <Text style={styles.title}>Bookmarks</Text>
    <FlatList
      data={bookmarks}
      renderItem={({item}) => (
        <TouchableWithoutFeedback
          onPress={() =>
            navigation.navigate('BookmarkedDetails', {listingData:
item})
          }>
          <View style={styles.itemBox}>
            <View style={styles.nameView}>
              <FontAwesome5 name='house-user' size={30} />
              <Text style={styles.itemName}>{item['Full
Address']}</Text>
            </View>
            <Text style={styles.itemPrice}>
              Estimated Price: ${Math.round(item['Estimated Price'])}
            </Text>
          </View>
        </TouchableWithoutFeedback>
      )}
    />
  </View>
</SafeAreaView>
```



```

        </View>
    </TouchableWithoutFeedback>
    )}
    extraData={bookmarks}
  />
</View>
</SafeAreaView>

```

The screen showing the details of the bookmark will also use the *SafeAreaView*. But this time we will include a *ScrollView* in it with the main features of the listing: number of bedrooms, surface in square feet, lot size in square feet, price per square feet and estimated price.

```

<SafeAreaView style={{flex: 1}}>
  <View style={styles.main}>
    <ScrollView contentContainerStyle={styles.infoView}>
      <View style={styles.titleView}>
        <Text style={styles.titleText}>{house['Full Address']}</Text>
      </View>
      <View style={{padding: 5}}></View>
      <View style={styles.subtitleView}>
        <Text style={styles.subtitleText}>
          {' - Bedrooms:  '}
          {house.BR}
        </Text>
        <Text style={styles.subtitleText}>
          {' - SqFt:  '}
          {house.SqFt}
        </Text>
        <Text style={styles.subtitleText}>
          {' - Lot Size:  '}
          {house['Lot Sz']}
        </Text>
        <Text style={styles.subtitleText}>
          {' - $/SqFt:  '}
          {house['LP dollars per SqFt']}
        </Text>
        <Text style={styles.subtitleText}>
          {' - Estimated Price:  $'}
          {Math.round(house['Estimated Price'])}
        </Text>
        /* <Text style={styles.subtitleText}>{" - ID:
        "{house.id}</Text> */
      </View>

```

```
{/* <Text>{comment}</Text> */}
```

Below the features, we will include a *View* with the comments that have been written in the listing.

```
<View>
  {comments
    .slice()
    .reverse()
    .map(c => (
      <Text key={c.key} style={styles.comment}>
        {c.comment}
      </Text>
    ))}
</View>
</ScrollView>
<View
  style={{
    width: '100%',
    height: '25%',
    justifyContent: 'flex-end',
    alignItems: 'center',
  }}>
```

Moreover, there has to be a place to write the comments. A *TextInput* will do the trick, as well as an “Add a comment here” button to call the *addComment()* function.

```
<TextInput
  style={styles.input}
  onChangeText={onChangeComment}
  value={comment}
  placeholder="Add a comment here"
  placeholderTextColor="gray"
/>
<View style={{padding: 5}}></View>
<View style={styles.button}>
  <Text
    style={styles.addComment}
    onPress={() => {
      addComment(comment, house.id), onChangeComment('');
    }}>
```

```
        Add Comment
    </Text>
    <Text style={styles.delete} onPress={() =>
deleteAlert(house.id)}>
        Delete
    </Text>
</View>
</View>
</View>
</SafeAreaView>
```

The *addComment()* function will first check that the comment is not empty and then it will retrieve from the Users database all the comments associated with that particular listing, placing them in an array. If there are no comments yet, an array will be created.

The comment will be added at the end of the array and the Users database will be updated by storing the new array.

```
if (comment === null || comment === '' || comment === undefined) {
    return;
}
database()
.ref(userRef)
.child('listings')
.once('value', snapshot => {
    if (snapshot.val() === null) {
        setBookmarks([]);
        setComments([]);
    } else {
        var bookmarked = snapshot.val();
        for (let i = 0; i < bookmarked.length; i++) {
            if (bookmarked[i].key === houseId) {
                // console.log("Here")
                var c = [];
                if (
                    bookmarked[i].comments === undefined ||
                    bookmarked[i].comments === null
                ) {
                    c = [];
                } else {
                    c = bookmarked[i].comments;
                }
                c.push({key: c.length.toString(), comment: comment});
            }
        }
    }
});
```

```
        // c.push(comment)
    database()
        .ref(userRef)
        .child('listings/' + i.toString())
        .update({
            comments: c,
        });
    console.log(c);
    setComments(c);
    console.log('Comment added');
    return;
    }
    }
    });
};
```

5.2.5 PROFILE SCREEN

This is the screen where the user can see its personal information: first name, last name and the number of bookmarks currently bookmarked. This number will be updated every time the user bookmarks a listing or removes a listing from bookmarks.

The profile screen also has an activity log which keeps track of every time the user bookmarks or remove from bookmarks a house. This is a great way for realtors to know where they first became interested on a certain listing or when they discarded it.

The entry in the log for a house being bookmarked has this format:

“House *Address* was bookmarked on *Date*”

And for removing a bookmark:

“House *Address* was removed on *Date*”

The activity log might become too crowded after a while, and it might be hard to search for a specific record. That is why the profile screen has a button which allows the user to clear the activity log, deleting all the previous activities.

The profile screen also contains a “Log Out” button, which will log the current user out of his account and redirect the user to the login screen so that he/she may try to log in with a different account.

The image below shows a snapshot of the profile screen. In this case the full name of the user is David Cocero, and he currently has 6 bookmarked properties. However, he only has 3 activities in the activity log which means that he at one point cleared the log.

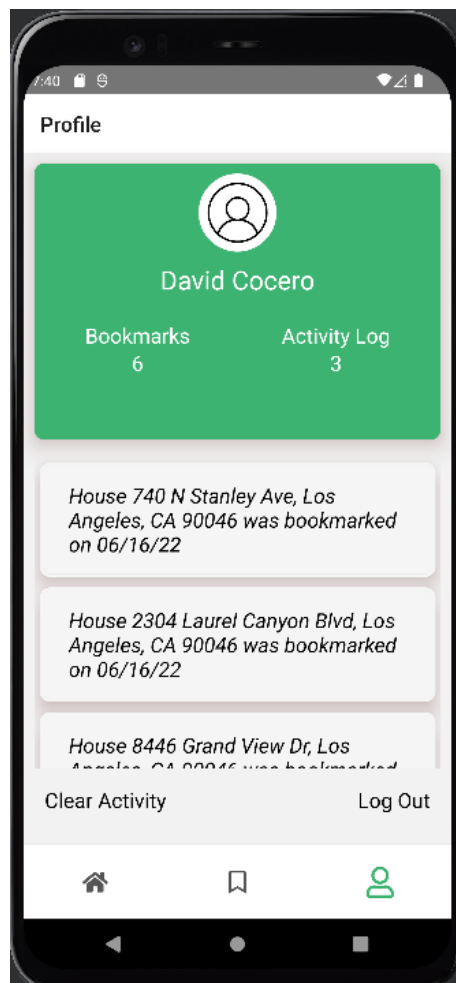


Figure 24: User Profile Screen

The *SafeAreaView* element was used once again in the code of the profile screen. The image at the top is for now a default image for all the users but the future plans include that the user can upload a profile picture to be displayed there.

Below the picture, we will display the number of listings and the length of the array of activities. The activities will be listed from the activity array using a *Flatlist*.

The “Log Out” and the “Clear Activity” buttons are also included in the layout.

```
<SafeAreaView style={styles.main}>
  <View style={styles.infoView}>
    <Image
      source={require('../images/profile-icon.jpg')}
      style={styles.image}></Image>
    <Text style={styles.name}>{name}</Text>
    <View style={styles.infoBox}>
      <View style={{justifyContent: 'center', alignItems: 'center'}}>
        <Text style={styles.info}>Bookmarks</Text>
        <Text style={styles.info}>{numListings}</Text>
      </View>
      <View style={{justifyContent: 'center', alignItems: 'center'}}>
        <Text style={styles.info}>Activity Log</Text>
        <Text style={styles.info}>{activity.length}</Text>
      </View>
    </View>
  </View>
</View>
<View style={styles.activityView}>
  <FlatList
    data={activity}
    renderItem={({item}) => (
      <View style={styles.activityBox}>
        <Text style={styles.activityText}>{item}</Text>
      </View>
    )}
    extraData={activity}
  />
</View>
<View style={styles.buttonView}>
  <Text style={styles.button} onPress={clearActivity}>
    Clear Activity
  </Text>
  <Text style={styles.button} onPress={logOff}>
```

```
Log Out
</Text>
</View>
</SafeAreaView>
```

In order to display the information on the screen, we must first make a read to the database and in particular to the Users table.

The name is obtained adding the ‘first’ and ‘last’ fields of the user. The number of listings bookmarked will also appear as a field in the database.

Finally, we will fetch the list of activities from the database. Before displaying it, we must first reverse it as we want to show the most recent movements first.

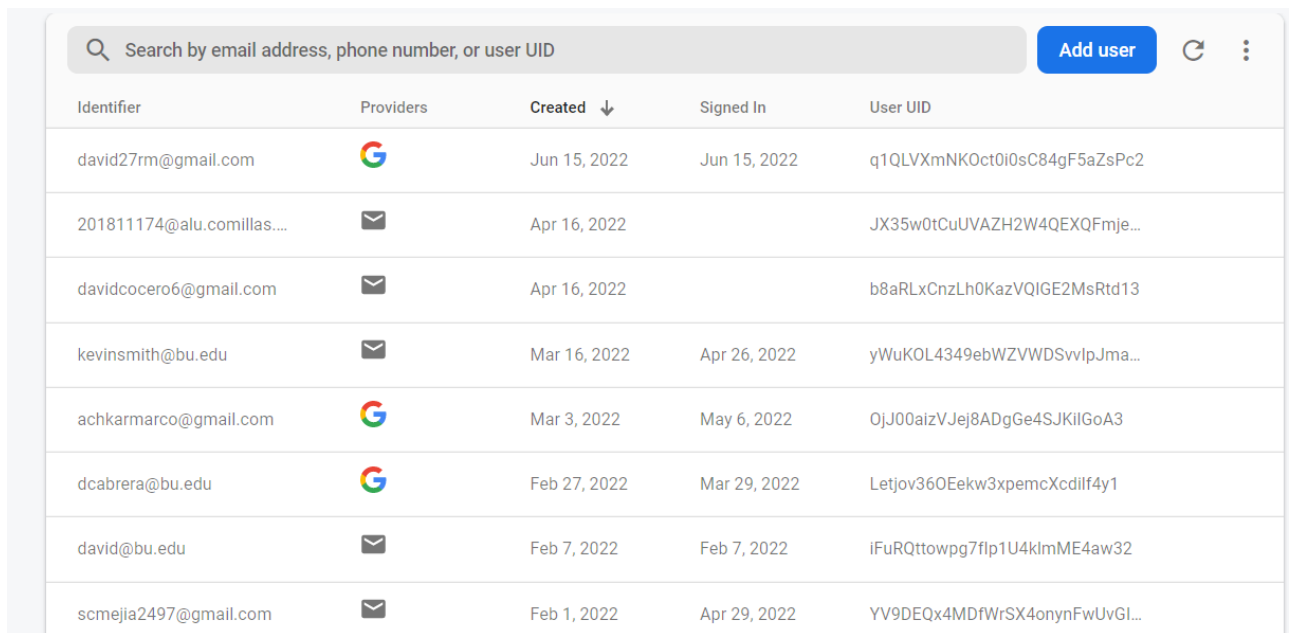
```
useEffect(() => {
  database()
    .ref(userRef)
    .on('value', snapshot => {
      if (snapshot.val() === null) {
        setName('');
        setNumListings(0);
        setActivity([]);
      } else {
        setName(snapshot.val().first + ' ' + snapshot.val().last);
        setNumListings(snapshot.val().numListings);
        var temp = [];
        if (
          snapshot.val().activity !== null &&
          snapshot.val().activity !== undefined
        ) {
          var temp = snapshot.val().activity;
          temp.reverse();
        }
        setActivity(temp);
      }
    });
}, []);
```

5.3 BACKEND

5.3.1 FIREBASE AUTHENTICATION

This service will provide the functionalities needed to maintain a log in service. It can be accessed using a library that is available in many languages, including React.js. There is also a website where the developers can monitor the movements made and change some settings.

All the users registered will appear in this list, which will show the email address, the way of signing in, the day it was created and the last time the user logged in. Users can also be added in this page by including an email and a password.











Identifier	Providers	Created ↓	Signed In	User UID
david27rm@gmail.com		Jun 15, 2022	Jun 15, 2022	q1QLVXmNKOct0i0sC84gF5aZsPc2
201811174@alu.comillas...		Apr 16, 2022		JX35w0tCuUVAZH2W4QEXQFmje...
davidcocero6@gmail.com		Apr 16, 2022		b8aRLxCnzLh0KazVQIGE2MsRtd13
kevinsmith@bu.edu		Mar 16, 2022	Apr 26, 2022	yWuKOL4349ebWZVWDSvvlpJma...
achkarmarco@gmail.com		Mar 3, 2022	May 6, 2022	OjJ00aizVJej8ADgGe4SJKIIGoA3
dcabrera@bu.edu		Feb 27, 2022	Mar 29, 2022	Letjov360Eekw3xpemcXcdllf4y1
david@bu.edu		Feb 7, 2022	Feb 7, 2022	iFuRQtowpg7flp1U4klmME4aw32
scmejia2497@gmail.com		Feb 1, 2022	Apr 29, 2022	YV9DEQx4MDfWrSX4onynFwUvGl...

Figure 25: Users registered in the app

The console also allows making changes to the system like adding providers to have more ways to sign in. These providers include Facebook, Twitter, Apple, GitHub and more. There is also the possibility to sign in using the phone number.

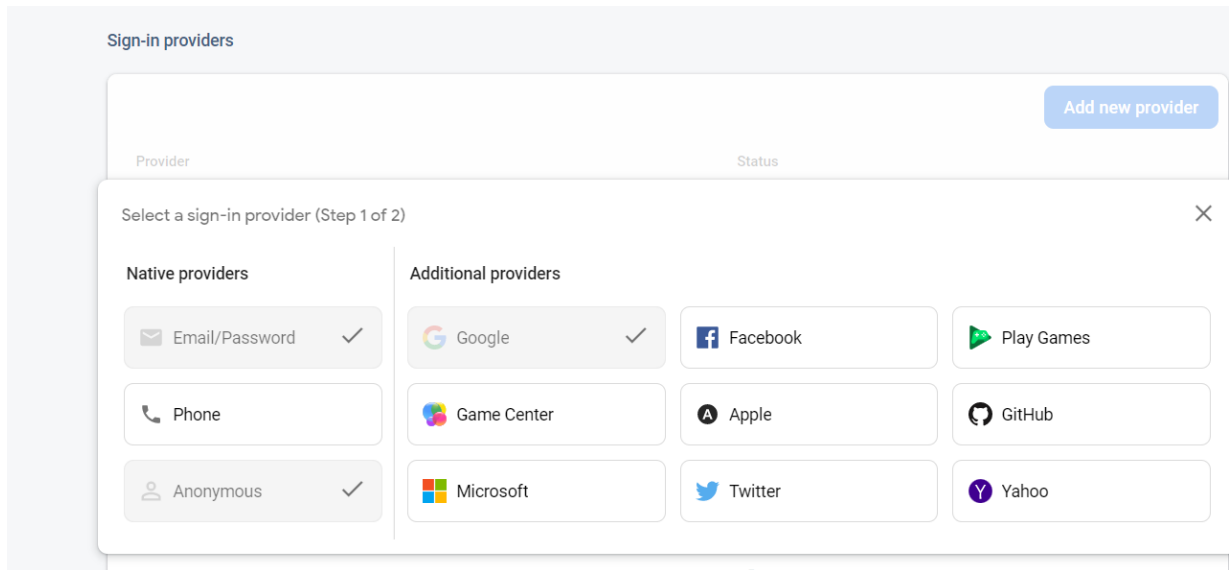


Figure 26: Sign in providers

5.3.2 FIREBASE REALTIME DATABASE

This is where all the information about the properties and listings is stored. Again, the data is updated, inserted, and deleted using the React.js library but there is also an online console when one can access the data stored.

5.3.2.1 Houses table

The biggest table of the two will be the Houses table. The database uses the format of a JSON tree and the key to an item will be its identifier (id). The id of the property will be generated using the SHA-256 algorithm to encrypt its full address. The SHA-256 was chosen because it has not been broken so far, so it is almost impossible to decrypt a cyphertext. Because of this, the id looks like a random string from which people cannot obtain data, so it is safer. Also, if a property currently in the database is taken out of the market but then is later put back in the market, the application will now it's the same listing because of its id.

Below, there is an example of how a property is stored in the houses table.

```
▼ — 032105ef7a17b063ae5565fd2c05be46d46d8581cda3255b53be6c3ab35cf74a
  — AR: 3
  — Address: "8356 Yucca Trl"
  — BR: 2
  — City: "Los Angeles"
  — DOM: 3
  — Estimated Price: 1029532.9865771811
  — F Bath: 1
  — Full Address: "8356 Yucca Trl, Los Angeles, CA 90046"
  — Growth per year: 13.9929362026
  — H Bath: 0
  — LP: "$895,000"
  — LP dollars per SqFt: "$1,060.43"
  — Last sold price: 365201.5421417572
  — Last time sold: "10/06/2008"
  — Lot Sz: "3,161"
  — Pool: 0
  — Q Bath: 0
  — SqFt: "844"
  — TQ Bath: 0
  — YB: 1962
  — bin: 4
  — id: "032105ef7a17b063ae5565fd2c05be46d46d8581cda3255b53be6c3ab35cf74a"
  — lat: 34.0995291
  — long: -118.357951
```

Figure 27: Listing stored in the database

These are the names of the fields of a listing. Some of these terms may be unknown to people that are not familiar with Real Estate, so a brief description is included.

- **'AR'**: area code. This is a number that denominates a territory within a zip code

- **'Address'**: the street name and number
- **'BR'**: number of bedrooms
- **'City'**
- **'DOM'**: days on market. This is a term that is used by realtors to show the number of days that have gone by since a property was put on the market. Houses with a bigger number of days on market are usually harder to sell.
- **'Estimated Price'**: this is the value that the machine learning predicts a house is worth based on its features.
- **'F Bath'**: number of full bathrooms. A full bathroom is one that has at least 4 main features, which are usually a sink, a toilet and a bathtub and a shower [33].
- **'TQ Bath'**: number of three-quarter bathrooms. A three-quarter bathroom is one that has three features: a sink, a toilet and a bathtub or a shower.
- **'H Bath'**: number of half bathrooms. A half bathroom just needs to have 2 features. Most of the times these are a sink and a toilet.
- **'Q Bath'**: number of quarter bathrooms. A quarter bathroom only has one feature, predictably a toilet. This type of bathrooms is extremely rare.
- **'Full Address'**: contains the street name and number, the city, the state, and the zip code
- **'Growth per year'**: average annual price increase percentage since the last sale. It is a value generated by the machine learning algorithms
- **'LP'**: listing price
- **'LP dollars per SqFt'**: listing price averaged per SqFt
- **'Last sold price'**: price of its last sale

- **'Last time sold'**: date of its last sale
- **'Lot Sz'**: lot size in SqFt
- **'YB'**: year when it was built
- **'Pool'**: 1 if the house has a pool, 0 otherwise
- **'Bin'**: bin of the house based on the 'Growth per year'. Used for the color-coding
- **'id'**: unique id of the house, calculated by computing the SHA256 of the 'Full Address'
- **'lat'**: latitude coordinates
- **'long'**: longitude coordinates

The latitude and longitude coordinates were not included in the initial data, they had to be calculated so that the houses could be properly placed on the map screen. These coordinates were obtained from each house's address using the Geocoding API in a Python script.

The “requests” library was used to call the API, including an API key that was obtained from a Google Maps Platform account. Once the latitude and longitude are fetched, they are stored as a new field of the property.

```
def extract_lat_long_via_address(address_or_zipcode):
    lat, lng = None, None
    api_key = "AIzaSyAVFAAL8igljE6l9UPUsk0SoX-nXFtjgfc"
    base_url = "https://maps.googleapis.com/maps/api/geocode/json"
    endpoint = f"{base_url}?address={address_or_zipcode}&key={api_key}"
    r = requests.get(endpoint)
    if r.status_code not in range(200, 299):
        return None, None
    try:
        '''
        This try block incase any of our addresses are not valid
        '''
```

```
results = r.json()['results'][0]
lat = results['geometry']['location']['lat']
lng = results['geometry']['location']['lng']
except:
    pass
return lat, lng
```

5.3.2.2 Users table

The other table of the database is the Users table which has many less fields. The key to a user element is the *userid* which is generated by Firebase Authentication when an account is created or the first time a user signs in with Google. There is function in React.js called *auth().currentUser.uid* which returns the *userid* of the user that is currently logged in. This is extremely useful to retrieve the bookmarks, activities and similar data.

In the next page there is an example of how a user item is stored in the Users table of the database.

```

SRLbcbhMzMeh3qHSE1wUatMg30v1
├── activity
│   ├── 0: "House 8446 Grand View Dr, Los Angeles, CA 90046 was bookmarked on 06/15/22"
│   ├── 1: "House 2304 Laurel Canyon Blvd, Los Angeles, CA 90046 was bookmarked on 06/16/22"
│   └── 2: "House 740 N Stanley Ave, Los Angeles, CA 90046 was bookmarked on 06/16/22"
├── email: "dcq@bu.edu"
├── first: "David"
├── last: "Cocero"
├── listings
│   ├── 0
│   │   └── key: "47509e50ea8673ce22116e42f1d2f31efd680ee8f0d22e61a135a3f7c4dd9eca"
│   ├── 1
│   │   ├── comments
│   │   │   ├── 0
│   │   │   │   ├── comment: "The kitchen needs reform"
│   │   │   │   └── key: "0"
│   │   │   └── 1
│   │   │       ├── comment: "Must contact homeowner on Monday morning"
│   │   │       └── key: "1"
│   │   └── key: "92ed59eece83be52b26f4acfcdf2ee09f63bea2e322b73551dd0b8c32a76e999"
│   ├── 2
│   │   └── key: "a58f943db2e241aee71227c7836f0941312b3629dfb11f8da93f2b3996464472"
│   ├── 3
│   │   └── key: "5ccd786c932f57b40a99274917c7a4a4a6c5cb0027a7a449c20fa1b9a471fa2d"
│   ├── 4
│   │   └── key: "5a7683d190385cf241583bb11f4d02cc16eca44b148f950640d7f713e57794f6"
│   └── 5
│       └── key: "3507e843bbc4ae4fedd117d6b743284924f7406ce23654a96479ba0b15c29569"
├── numListings: 6
└── uid: "SRLbcbhMzMeh3qHSE1wUatMg30v1"

```

Figure 28: User stored in the database

These are the fields contained in a User item:

- **'activity'**: an array of the activities currently registered in the activity log. It will be updated every time a user bookmarks and removes a bookmark. It can be emptied if the user selects the option of 'Clear Activity log' in the user profile screen.
- **'email'**: email address of the user
- **'first'**: first name of the user
- **'last'**: last name of the user
- **'listings'**: an array of the listings bookmarked. Each element of the array can have one or two fields. The first one is called 'key' and is the id of the listing bookmarked. Although this database is not relational, this field could be considered as the foreign key of the Users table, with the ids of the listings as the primary key of the Houses table. Again, this is not a proper constraint and is not strictly enforced, but it is still a way to establish a relationship between both tables. If the bookmark has comments, the element will contain a second field, 'comments' which is an array of the comments made by a user about a listing.
- **'numListings'**: the number of listings the user has currently bookmarked
- **'uid'**: the user id

5.3.2.3 Database rules

Once we have all the data in the database, some rules must be established to restrict the access to it. The default configuration is that the access to the database for reading and writing is public, so Firebase sets a maximum of 30 days to modify the rules before taking the database out of use for security risks.

The creation of rules is pretty flexible, as different rules can be established for reading and writing and a rule can also depend on more specific parameters like a field of an element.

In the case of the Houses table, both read and write privileges were set to false so that they only can be accessed using a token. There were considerations to set the read to true, as the listings data can be found in other websites, but it was later decided to set it to false as there are some fields that are the results of the machine learning process, and it is better to keep them secret for potential competitors.

The users table rules are slightly different. This time the data of a user can only be read and updated by the own user. That means that the user id of the user logged in must be the same as the id of the user that he/she is trying to retrieve or modify in any way. This is probably one of the most strict rules, but it is absolutely necessary to maintain the privacy of the users of the app.

```
{
  "rules": {
    "houses":{
      ".read": false,
      ".write": false
    },
    "users":{
      "$uid": {
        ".write": "$uid === auth.uid",
        ".read": "$uid === auth.uid"
      }
    }
  }
}
```


5.4 MACHINE LEARNING

5.4.1 DATA COLLECTING

At the start, the clients provided some sample data to start working on the model. However, at the project advanced, more data was required to advance. To get that extra data, the clients granted access to their MLS (Multiple Listings Service) account which allows searching listings across the USA.

Following the clients' directives, the application is just focusing for now on a certain zip code, 90046 located in West Hollywood, California. In the MLS website we are just going to search for listings in this zip code. I arrived at the conclusion that only data since 2015 should be used to train the model, because previous data might be less significant.

The web allows the user to filter the listings by kind: active, hold, pending, sold, expired, withdrawn, or cancelled. They can also be selected by the period in which they were sold, expired...

In this case we are interested in listings that were either sold, expired, cancelled, or withdrawn since 2015, and the selected zip code is 90046. The main problem of this webpage is that it only allows up to 500 listings to be displayed at a time, so instead of choosing the period of time from 2015 to now, the time range had to be smaller (usually around 10 months) to fit a maximum of 500.

The following image is an example of what to expect. The platform displays all the listings in the map, with the listed price and using color coding to distinguish between the different status (red is sold, purple is cancelled, gold is expired and black is withdrawn).

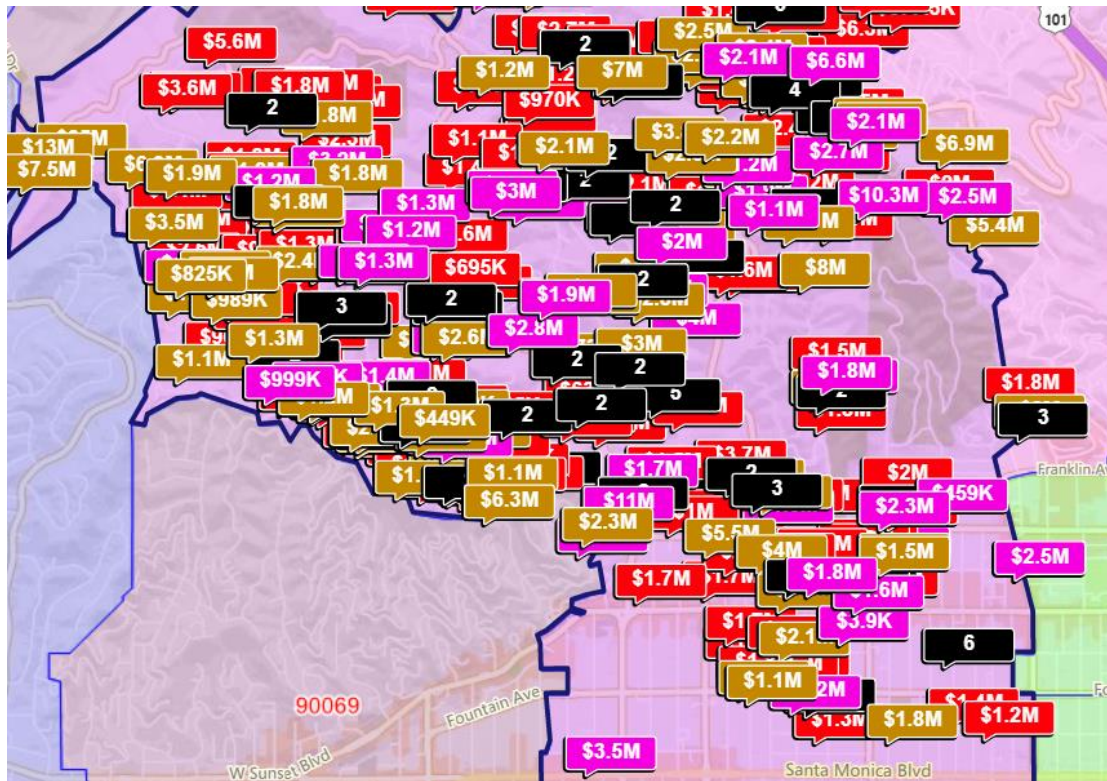


Figure 29: MLS listings search [34]

The listings in the map can also be displayed as a table by selecting the option “one-line hot sheet”. Here we can see the different characteristics of each listing that will be useful for the model: year built, address, number of bedrooms and bathrooms, days on market (DOM), the status (sold, cancelled, expired, or withdrawn), the square footage...

This list can then be exported to a csv file so that it can be used for the models. As discussed, the main issue is that the website only allows a maximum of 500 listings for each csv file, so all the csv files must later be joined.

5.4.2 DATA CLEANING

The data in the MLS is usually entered by Real Estate Agents and thus it has flaws like formatting inconsistencies or missing data. The task now is to try to minimize the impact of those flaws in the data so that the model is not affected. This involves removing rows and columns, standardizing fields, changing some formats, and making some fields more suitable to the future models.

The first thing we are going to deal with is the bathroom column as it has different possible formats. The most common one involves writing the total number of bathrooms first and then how many bathrooms there are of each type. So if for example in a row the value is “3 (2 1 0 0)” that means it has 3 total bathrooms, two of them full bathrooms and one three-quarter bathroom. But having this information in the same field is not a great idea, it would be better to separate the field into four fields, one for every type of bathrooms, as these features could have relevance in the model.

In the Python script used to clean the data, this function was created to split the bathrooms.

```
def get_bathrooms(data: pd.core.series.Series) -> int:
    bath_string = data['Baths(FTHQ)']
    temp = bath_string.split('(')[1]
    bath_vals = temp[:-1].split()
    for i in range(4):
        bath_vals[i] = int(bath_vals[i])
    return bath_vals[0], bath_vals[1], bath_vals[2], bath_vals[3]
```

Then, the function is called and based on the counts returned, the bath count is deemed as valid or not. If all the counts of bathrooms are 0, it could mean two things: the data for that entire field is missing or the format is different. For example, the bathrooms column could appear as a unique value that represents the weighted average of the different types of bathrooms (1 for full bathrooms, 0.75 for three-quarter bathrooms, 0.5 for half-bathrooms and 0.25 for quarter-bathrooms). The problem with this format is that the number of

bathrooms of each type cannot be extracted from that weighted average. Therefore, to avoid confusions, the few rows with that format and with missing bathroom data are removed.

The count for each type of bathroom is stored in a new field.

```
for index, data in listings.iterrows():
    fcount, tcount, hcount, qcount = get_bathrooms(data)
    print
    if fcount == 0 and tcount == 0 and hcount == 0 and qcount == 0:
        bad_baths.append(index)
    else:
        f_bath.append(fcount)
        t_bath.append(tcount)
        h_bath.append(hcount)
        q_bath.append(qcount)
```

Another field that had to be processed is the date in which the listing was sold. It could be interesting to know if the price depends on the date it was sold. The market is said to have been on the rise for the last years and if we consider inflation, listings sold years ago probably had a lower price than nowadays. That is why we will include two new fields: 'Years since' and 'Months since' which represent the years since the sale and the months since the sale.

The processing is simple, getting the current date and making the necessary calculations.

```
def get_MSandYS(chng_date: str) -> int:
    now = date.today()
    mon, day, yr = chng_date.split('/')
    temp = yr + '-' + mon + '-' + day
    then = date.fromisoformat(temp)
    out = now - then
    YS = float(out.days / 365.0)
    MS = int(mon)
    return MS, YS
```

Also, the price of the listing was converted to a numeric value from a string, removing the dollar sign (\$) and the comma (,)

```
def string_to_num(string_num: str) -> int:
    string_num = string_num.replace(", ", "")
    string_num = string_num.replace("$", "")
    num = int(string_num)
    return num

# fix string type numbers to numeric type
listings = listings.applymap(lambda x: string_to_num(x) if
isininstance(x,str) else x, na_action='ignore')
```

We will be deleting the columns that are useless to the algorithm. These will be the Baths column (we split it into 4 different columns), the address, city, the map, Homeowner Associated Dues (HOD, a Real Estate term), the change type and date (whether it was sold, cancelled, withdrawn or expired and when it happened), the S column (true if it was sold), its Open Houses, the link of the Photo and the MLS number.

Some columns also needed to be removed due to target leakage as they were too correlated with the target variable (Sold Price). These columns include the Listing Price (LP) and the Listing Price per Sq Ft. The reason to eliminate them is that they will provide a model with a really good test error as the sold price will be relatively close to the listing price, but when applied to houses that are not on the market, the error will be much larger as these types of listings won't have the listing price available.

```
listings =
listings.drop(['Baths (FTHQ)', 'Address', 'City', 'Map', 'HOD', 'Change
Type', 'S', 'Change Date'], axis = 1)

listings = listings.drop(['Open Houses', 'Photo', 'MLS', 'LP
$/SqFt', 'LP'], axis = 1 )

\
```

Finally, we will remove the invalid rows. These are listings where the number of bedrooms or the amount of surface is not displayed or they contain ‘Not a Number (NaN)’ values, All of these have to disappear from the file because they will affect the performance of the model negatively.

```
# remove invalid rows
listings = listings.dropna()
listings = listings[ (listings['BR'] != 0) & (listings['SqFt'] != 0)
& (listings['Lot Sz'] != 0) ].copy()
```

5.4.3 CORRELATION MATRIX

When we have a lot of variables and we are not sure where to start, it is usually a good idea to obtain the correlation matrix of the variables so that one can see how they relate and if they are important for the target variable. It could be argued that a correlation matrix can be used to summarize data.

The correlation matrix is also important in this case as we will be using Multiple Linear Regression where it is important that the independent variables are not highly correlated with each other. If not, this could derive in multicollinearity, causing several problems to the model [35]. The correlation matrix will tell us if the variables are highly correlated so that we can avoid this situation.

We will be using Python to create the correlation matrix, and specifically the heatmap function of the Seaborn library.

```
corrMatrix = solds_df.corr()
sn.heatmap(corrMatrix,annot = True)
plt.show()
```

This is the result obtained:

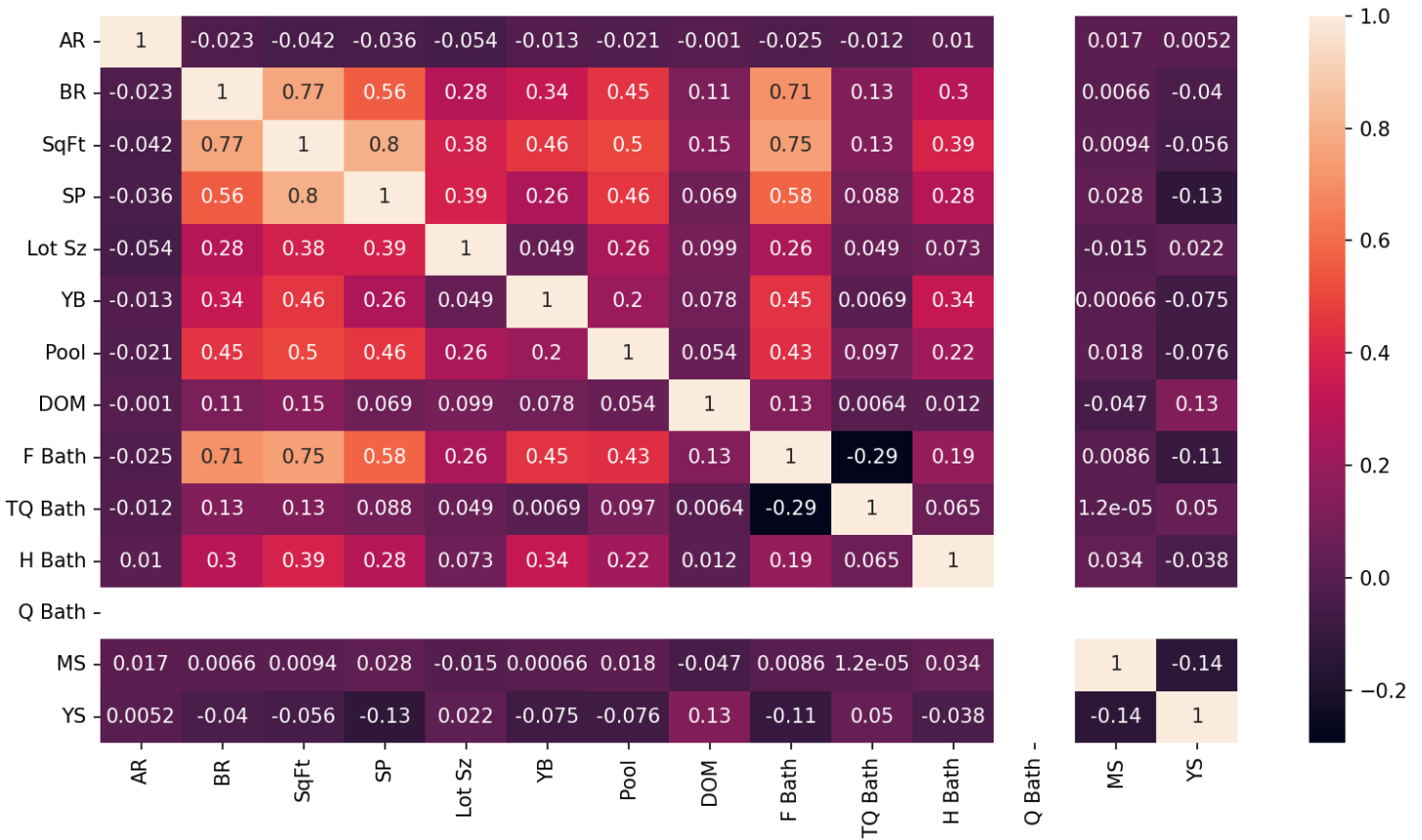


Figure 30: Correlation matrix of the listings features

There are some remarks we can say about this correlation matrix. First of all, the variable with the highest correlation with the target variable (Sold Price or SP) is the surface in SqFt, followed by the number of bedrooms and the number of full bathrooms. This makes a lot of sense if you think about it because most times the bigger the house, the higher the price is within a zip code. Also, houses with more bedrooms and bathrooms are usually more expensive.

The number of three-quarter baths has the lowest correlation with the SP. This is probably due to the fact that most houses have full baths instead of three-quarter baths, so the count of three-quarter baths is not indicative of the price. Notice also that the quarter baths column is completely empty which means that no house has a quarter bath in this zip code and thus it will play no part in the model.

Surprisingly, the number of years and months since the last sale are almost uncorrelated to the sold price. It was expected to have some effect, because as the market price has vastly increased, the same house several years ago was probably sold for less than its current value. Therefore, the Years Since column should have a high negative correlation with the sold price. However, while the correlation is negative, the value is still kind of low.

We must also discuss the correlation between the different input variables. Oftentimes when a group of features exhibits a very high correlation among themselves, all but one of them (usually that with highest correlation to the target feature) are removed. This deletion is done because each extra feature adds little marginal information and lowering the complexity of the model is usually preferred due to computational efficiency and allegedly lower risks of overfitting. This is related to the multicollinearity problem previously mentioned. In this case, however, none exhibit a correlation greater than 0.9, the popular benchmark of “too high”.

We chose to remove those features with an absolute value correlation of less than 0.1 as they are expected to have a low impact on the Sold Price. The Q Bath (Quarter Bath) column is also removed because the value is zero in every row.

Once we have removed those columns, we will display a correlation again but this time only with the remaining features.

```
threshold = 0.1
low_correlations = corrMatrix[corrMatrix['SP'] < threshold]
low_correlations = low_correlations[low_correlations['SP'] > -
1*threshold]
solds_df = solds_df.drop(columns=low_correlations.index.values)
solds_df = solds_df.drop(['Q Bath'], axis = 1) # all zeros
# recompute and plot correlation matrix
corrMatrix = solds_df.corr()
sn.heatmap(corrMatrix,annot = True)
plt.show()
```


Now we obtain this correlation matrix:

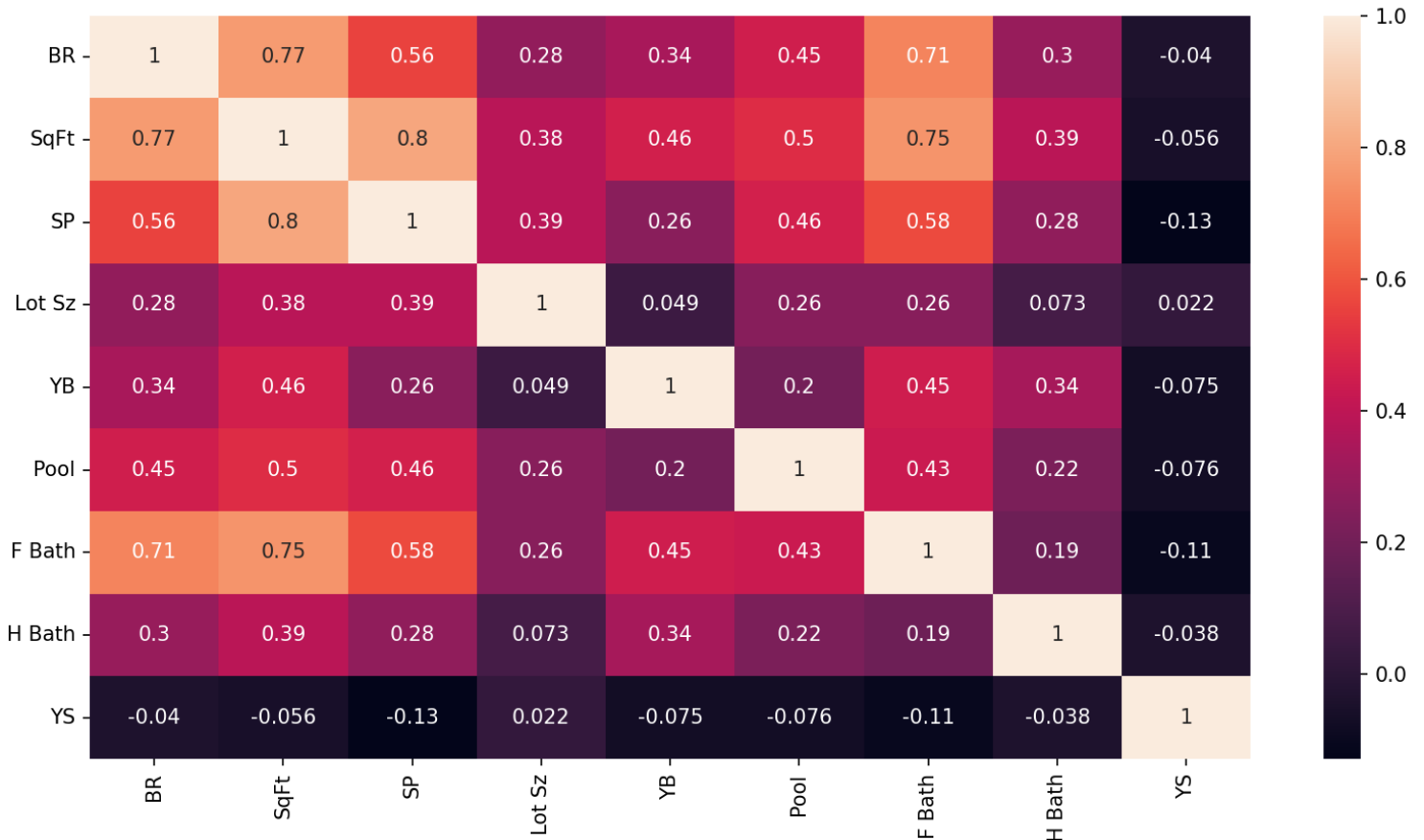


Figure 31: Correlation matrix after removing several features

Comparing both correlation matrixes we can see that the features that are deleted are Months Since (MS), number of quarter baths (Q Bath), number of three quarter baths (TQ Baths), Days on Market (DOM) and Area code (AR)

Here all the correlations with the Sold Price are positive except for the Years Since column. The positive values all have a correlation with the Sold Price of at least 0.26 which is not great, but it shows they can contribute to the price.

This matrix allows a better interpretation of the results, as we only have the features, we figure are more relevant so the correlations are clearer. However, this does not mean that the

deleted features should be completely forgotten for the price estimator. We will later have to try including and removing them to see if it makes any difference.

5.4.4 LINEAR REGRESSION

A good way to start tackling this problem is to try Linear Regression. This is probably the simplest machine learning algorithm, but it can also be very useful. In this case we will use it as a starting point, to get an idea of where we stand.

As we already discussed there are four basic assumptions in linear regression:

1. **Linear relationship between the input variables and the output**
2. **Residuals are independent**
3. **Residuals have constant variance for different values of the input variable**
4. **Residuals follow a normal distribution**

All the assumptions look reasonable for the price estimator, so we are going to implement the algorithm and then discuss the results.

We will be implementing Multilinear Regression, in which several input variables are used to obtain the target variable. We will need to figure out the value of the coefficients for every variable. In Multilinear Regression it is also really important to remove those variables with high correlation with each other.

The first thing to do is to define which are the input columns and which is the output variable (Sold Price)

```
y = solds_df['SP'].to_numpy(copy = True)
X = solds_df.drop(['SP'],axis = 1).to_numpy(copy = True)
```

Then the dataset is split into two parts: the training set and the test set. The first one will be used to train the model to obtain the coefficients and the second one will be fed into the created model to obtain information about the accuracy of the model. This will be extremely useful to detect possible underfitting and overfitting.

We will use 20% of the data as test set and 80 % as the training set. The split is made randomly so is important to set an integer value as the random state so that the split is always the same.

```
xtrain, xtest, ytrain, ytest = train_test_split(X,y,test_size = 0.20,  
random_state = 42)
```

Then we will use the training set to fit the linear regression model

```
reg = LinearRegression().fit(xtrain,ytrain)
```

Both the weights of each feature and the intercept can be obtained from the reg variable.

```
w = reg.coef_  
print('weights:')  
print(w)  
b = reg.intercept_  
print('bias:')  
print(b)
```

These are the coefficients obtained in one of the executions:

FEATURE	COEFFICIENT
Number of bedrooms (BR)	-3.044
Surface in Square Feet (SqFt)	1.111
Lot size in Square Feet (Lot sz)	6.718
Year built (YB)	-6.747

Pool	2.522
Number of full bathrooms (F Bath)	-1.419
Number of half bathrooms (H Bath)	-9.205
Years since last sale (YS)	-6.8557

Table 2: Coefficients obtained in Linear Regression

The intercept is 13644445.67 which is higher than most of the houses. This explains the fact that most of the weights are negative, as the price prediction is usually going to be lower than the intercept. It is hard to address while some of them are higher than others, but I believe the models may have some flaws. For example, the number of bedrooms should be a positive weight as we expect a house with more bedrooms to be more expensive.

Now that we know the weights and the intercept, the model can be used to predict the value of some listings. We are now going to use the test set for the prediction.

```
pred = reg.predict(xtest)
```

We are going to create a plot with the Predicted Price in the Y axis and the actual Sold Price in the X axis.

```
plt.figure(figsize=(20, 5))  
plt.scatter(ytest, pred)  
plt.xlabel('SP')  
plt.ylabel('Predicted SP')  
plt.show()
```

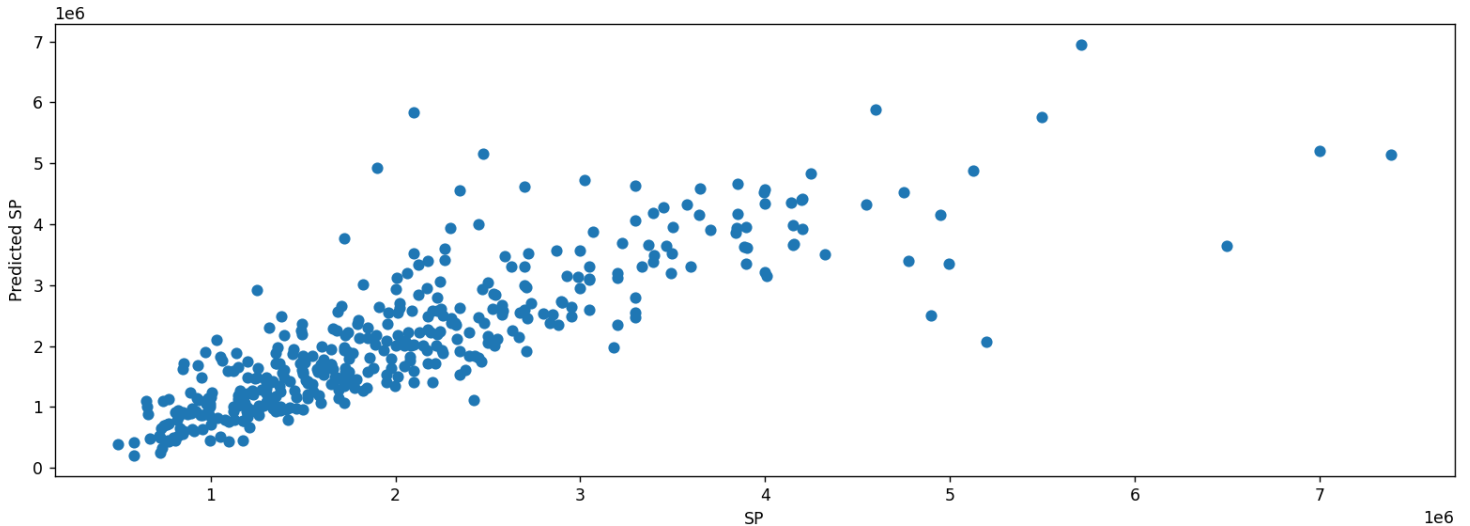


Figure 32: Predicted Sold Price vs Actual Sold Price

Ideally, we would want to have all the points aligned, where the predicted price is the same as the sold price. This would mean that the problem is linear, and the model is perfect (is always correct). Obviously, this is almost impossible to achieve with real life data, but still in a really accurate model all the points would be near to that hypothetical line.

In this model, one could draw a line trying to join as many points as possible, but there are still many points that are far from their prediction. This means that while it is not a bad model, it has some inaccuracies.

After going through an eye test of the model, it is usually a good idea to compute its average error to have an actual number to measure the accuracy. The error is computed as the average difference between the prediction and actual value.

```
cum_error = 0
for i in range(len(ytest)):
    abs_error = abs(pred[i] - ytest[i])
    percent_error = abs_error/ytest[i]
    cum_error += percent_error
avg_percent_error = cum_error/len(ytest)*100
print('average percent error:', avg_percent_error)
```

The error varies for different executions, as the model created is different every time. In general, the error goes from 20% to 23 %

The error is higher than desired, as 20% is a significant difference which can affect the performance of the app. To try to minimize the error we will next try with a different approach called Random Forests.

5.4.5 RANDOM FORESTS

A Random Forest builds several independent decision trees and then uses the most popular output of the trees as the final output. This usually helps improve the performance as it removes certain bias of using correlated input.

In the Python script the first steps will be common with the linear regression: defining the input and output columns and splitting the dataset into training set (80%) and test set (20%). Then, we will be implementing the Random Forest.

When using the Random Forest algorithm of the Scikit learn library there are several parameters that need to be defined [36]:

- **n_estimators:** number of decision trees. The default is 100
- **criterion:** this is the measure used to create the trees. It can be squared error, absolute error or poisson. The default is squared error.
- **max_depth:** this is the maximum depth of the trees of the forest. The default value is None which means it continues until the leaves are pure
- **min_samples_split:** the minimum number of samples required to split a node in a tree. The default is 2
- **min_samples_leaf:** the minimum number of samples required to be at a leaf node. The default is 1

- **max_features:** this is the number of features that are considered when making a split at a decision tree. It can be an int, a float of one of these:
 - **Auto or None:** the same as the number of features (n_features)
 - **Sqrt:** the square root of the number of features
 - **Log2:** the log2 of the number of featuresThe default is 1
- **random_state:** deals with the reproductivity of the result by controlling the randomization of the sampling. The default is None
- **ccp_alpha:** this is used to control the complexity of the algorithm. It determines the minimal Cost-Complexity Pruning cost. The default is 0.0

There are other input parameters, but they will not be used as they would have little to no effect in the problem.

It can be extremely hard to determine the values for every parameter that optimize the algorithm to obtain the best result. This would probably need to be done by trial and error. Fortunately, the *RandomizedSearchCV()* function of the Scikit learn library will do that for us.

This will receive the type of algorithm, in this case *RandomForestRegressor*, a list of possible parameters and the total number of iterations that must be performed.

These will be the possible parameter values that will be passed as arguments to the optimizer:

```
param_distributions = {
    'criterion': ['squared_error', 'absolute_error', 'poisson'],
    'max_depth': [None],
    'min_samples_split': [1, 2, 3, 4],
    'min_samples_leaf': [1, 2, 3, 4],
    'n_estimators': stat.randint(low=50, high=300),
    'max_features': ['auto', 'sqrt', 'log2'],
    'random_state': [None],
    'ccp_alpha': [0.0]
}
```

In some of the fields we won't discard any possible option like in the criterion or `max_features`. However, others like the minimum samples split and minimum samples leaf will have 1 to 4 as possible values. Also, other fields will just have the default value like the `ccp_alpha` field.

The function will receive only the training set, so the optimal parameters are obtained from that data. 50 iterations will be performed.

```
n_iter = 50
cv = RandomizedSearchCV(estimator = RFR, param_distributions =
param_distributions, n_iter = n_iter)
```

The process takes about 3-4 minute to complete, resulting in the following optimal parameters:

```
{'ccp_alpha': 0.0, 'criterion': 'absolute_error', 'max_depth': None,
'max_features': 'log2', 'min_samples_leaf': 1, 'min_samples_split': 3,
'n_estimators': 248, 'random_state': None}
```

The training error is now on average 6-7 % which is considerably low. The testing error is slightly lower than for the linear regression but much higher than the training error, at 18-19 %

The testing error being significantly higher than the training error is usually a sign of overfitting. This happens when the model is overly specified so the model is accurate during training, but it struggles in the testing. This is because the *RandomizedSearchCV()* function uses the training set to find the optimal parameters, but does not take into account the test set.

Here is a graphic that compares the predicted sold price vs the actual sold price of the different listings for one of the executions. Several lines have been added indicating different margins of error: 10,20 and 40%.

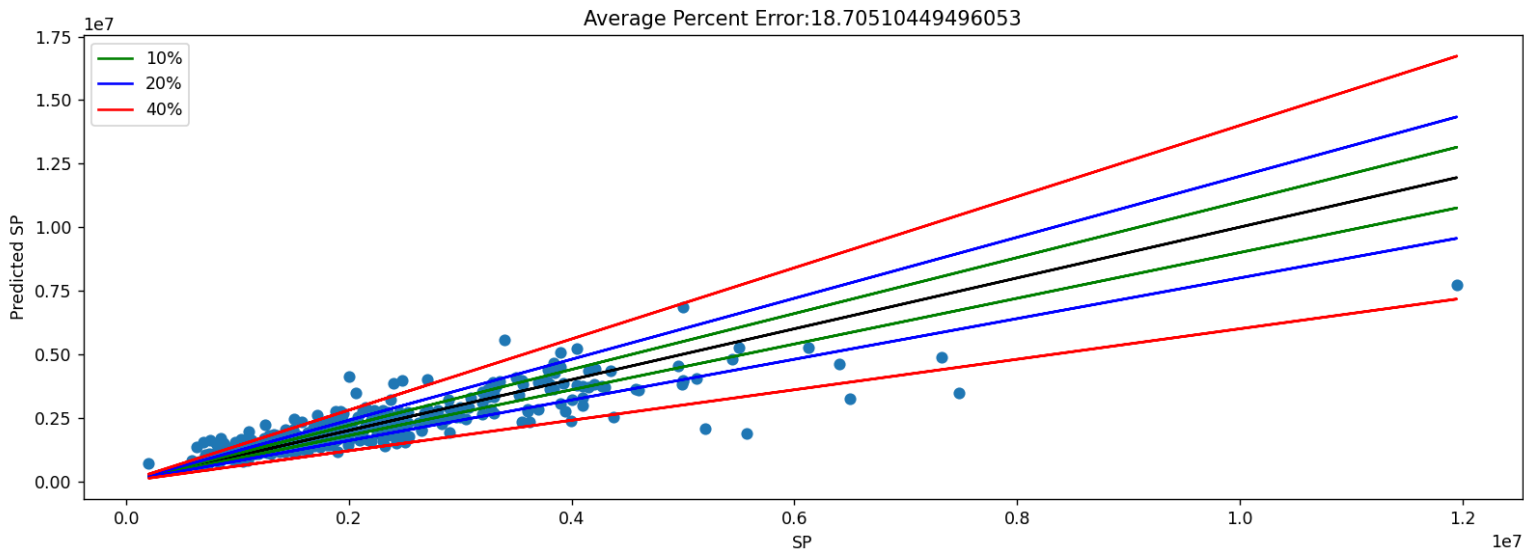


Figure 33: Sold Price vs Predicted Sold Price in Random Forest

Notice that most of the dots are within the 40% error margin, with the majority of the inaccuracies in the lower price homes, which makes sense as the margin error is smaller in this zone. There are however many listings that scape both the 10 and 20% lines which can be kind of worrying.

To try to improve these results, we will next try the Gradient Boosting Regressor

5.4.6 GRADIENT BOOSTING REGRESSOR

The gradient boosting regressor is an algorithm that starts from a weak estimator like the mean and from there with each iteration it obtains a more accurate model. It tries to minimize the loss function by fitting a regression tree on the negative gradient of the function.

The Python script must once again define the input and output columns and select the training and testing sets before implementing the Gradient Boosting Regressor.

The Gradient Boosting Regressor algorithm from Sci-kit learn will have the following parameters [37]:

- **loss:** the loss function that needs to be optimized. It can be the squared error, the absolute error, huber (combination of squared and absolute error) and quantile. The default is squared error.
- **learning_rate:** it is the factor that controls the contribution of the new tree in the algorithm. The bigger it is the faster the algorithm converges, but it may affect the accuracy of the model. The default is 0.1
- **n_estimators:** the number of iterations (stages) that the algorithm performs. The bigger it is, the more robust the algorithm is supposed to be.
- **subsample:** it is the fraction of samples used for the base learners. It goes from 0 to 1 with 1 being the default
- **max_features:** it is the same parameter that is used for Random Forests
- **max_leaf_nodes:** it sets the maximum number of leaf nodes when growing a tree. The default is None (no limit)

The other parameters of the function are not considered.

We will be using *RandomizedSearchCV()* again to look for the optimal parameters to feed the algorithm. The number of iterations will be 50 with these possible parameter values:

```
param_distributions = {  
    loss = ['squared_error'],  
    learning_rate : [0.02, 0.05, 0.07, 0.1, 0.15, 0.2, 0.25, 0.3],  
    n_estimators : [40, 60, 80, 100, 120, 150, 200, 250, 300],  
    subsample : [0.3, 0.4, 0.5, 0.6, 0.7, 0.8],  
    max_features: [3, 6, 9],  
    max_leaf_nodes : [8]}
```

The optimal parameters and the testing and training errors were significantly distant for the different executions. The training error had a higher mean and variance compared to the random forest, with values between 8 and 16%. However, the testing error was usually lower than the previous algorithm as it was around 16-20%.

Having the training and testing error relatively close mean that the overfitting is not that significant. This algorithm is usually robust against overfitting which means that the optimal number of estimators in this case is always going to be the maximum possible (300).

Let's study with detail the results obtained for an execution that was along the average:

- Optimal Parameters:

```
{'subsample': 0.7, 'n_estimators': 300, 'max_leaf_nodes': 8,  
'max_features': 6, 'loss': 'squared_error', 'learning_rate': 0.05}
```

- Training error: 14.54%
- Testing error: 18.77%

- Predicted sold price vs actual sold price graphic:

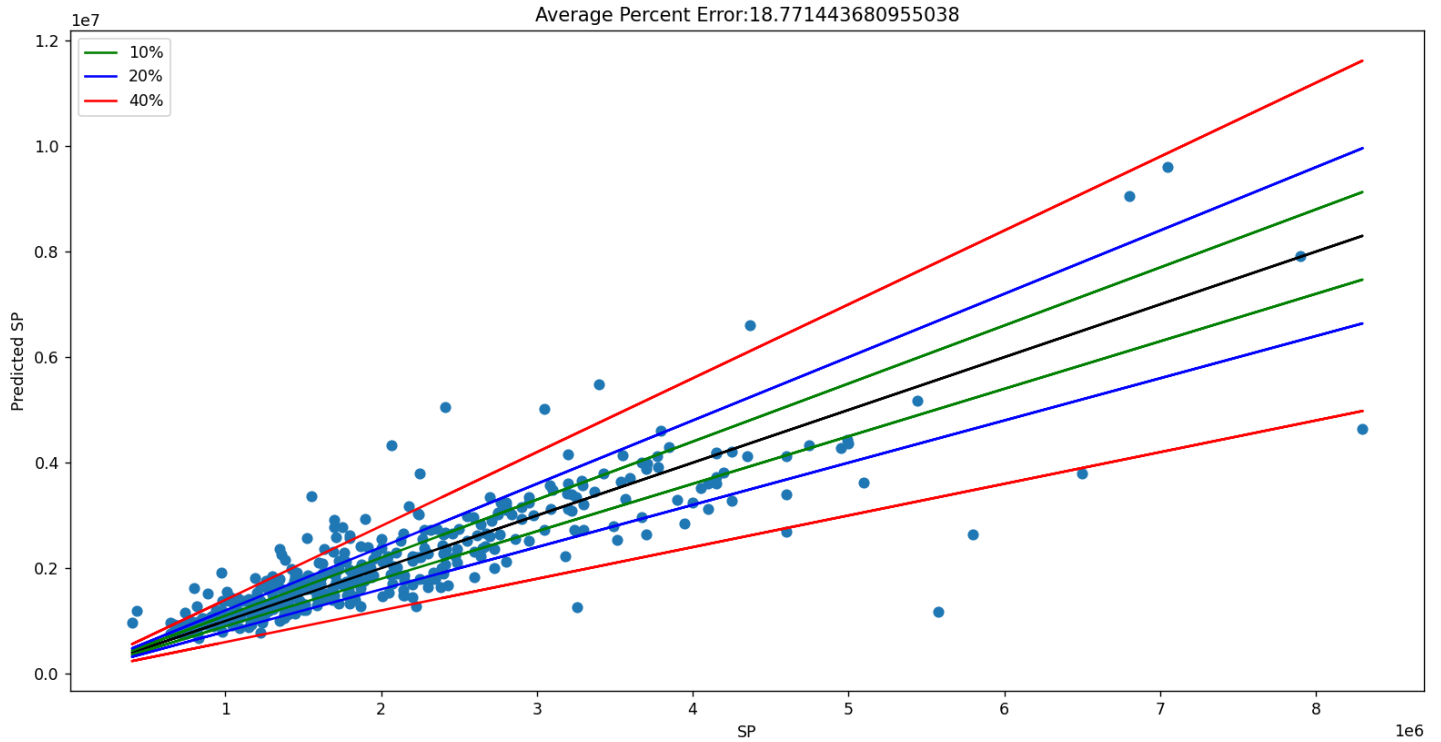


Figure 34: Sold Price vs Predicted Sold Price in Gradient Boosting Regressor

A more detailed comparison about the three studied algorithms is presented in the *Results Analysis* section. The final product offered to the client is a script where the user can select which of the three algorithms to use and certain parameters of the *RandomizedSearchCV()* like the number of iterations.

5.4.7 LIKELIHOOD OF SELLING

Once we have obtained the estimation of the price using any of the three possible algorithms developed, this value must be used to try to find how likely is that a house will be put in the market. This task proves to be extremely challenging, as it is hard to find a basis to do this calculation. It can be argued that there are different possible values that can influence on the decision of an owner to possibly sell his or her house.

One interesting way would be to use the information of the owners to get a good idea of their current situation. For example, if the owner was living there with his or her family, some facts like the age of the children could play a role in this calculation. Data about their professional career could also be useful, like the number of years the owner has been in his or her current job and the working area. These could be possible input variables in the likelihood of selling algorithm. However, this data is protected by law and there is no way to access it.

Trying to search for possible alternatives the clients suggested that a possibility could be to do this calculation based on the potential return on investment (ROI) of the owners. They said that if the property value had skyrocketed since the owners bought it, the owners might be more willing to sell it. Therefore, the Real Estate agent could approach the owner of those types of listings showing them the estimated current value and the potential profit they would be getting. The positive response from the owner is still not a sure thing but it is still much more likely than when realtors approach random owners.

These are the steps of the algorithm:

1. Get a list of listings the realtors could be interested in. In this case the clients provided a list of *cold calls*.
2. Find out the last time each of the listings was sold and the selling price
3. Input the listings into the price estimator, obtaining an estimated price based on the features.

4. Compute the difference between the estimated price and the price of the last sale.
Use that value to obtain the percentage of how much the price has gone up
5. Use that percentage and the date of the last sale to compute the average annual growth percentage of the price.
6. Split the houses into 5 bins based on their average annual growth percentage
7. Save all these values into the csv and upload it to the database so the results can be displayed on the application

This is an example of the code used when the average annual growth percentage is obtained for each listing, and they are assigned accordingly to a bin. The limits of the bins are not fixed values, they can vary depending on the list of listings being used. In this case, where the *cold calls* of the clients are used, this limits seem reasonable to distribute the listings.

```
price_growth_per_year=[]
bins=[]
for index, data in listings.iterrows():
    date=data['Last time sold'].split('/')
    time_difference =
relativedelta(datetime.datetime.now(),datetime.datetime(int(date[2]),int(
date[0]),int(date[1])))
    difference_in_years = time_difference.years
    percentage_growth=(data['Estimated Price']/data['Last sold
price']-1)*100
    growth_per_year=percentage_growth/difference_in_years
    price_growth_per_year.append(growth_per_year)
    bin=1
    if(growth_per_year>5.0 and growth_per_year<=8.0):
        bin=2
    if(growth_per_year>8.0 and growth_per_year<=12.0):
        bin=3
    if(growth_per_year>12.0 and growth_per_year<=16.0):
        bin=4
    if(growth_per_year>16.0):
        bin=5
    bins.append(bin)
listings['Growth per year']=price_growth_per_year
listings['bin']=bins
listings.to_csv('active_listings_price_growth.csv',index = False)
```

Chapter 6. RESULTS ANALYSIS

In this section, one is supposed to go through the results obtained and do a critical analysis. However, as this project consists in the development of mobile application this section might feel out of place. It is hard to figure out what would be the results of a project like this. The application could be considered the result itself, but in that case this section would make no sense. Instead, we are going to take a look at the results obtained from the different price estimators and further explore their consequences.

6.1 PRICE ESTIMATOR

Three different price estimators were used: Linear Regression, Random Forests and Gradient Boosting Regressor. Here is a table comparing the training and testing error obtained for each algorithm. The error varies for every simulation run, so the table will display a typical range of the error.

ALGORITHM	TRAINING ERROR	TESTING ERROR
Linear Regression	20- 23 %	20- 23 %
Random Forests	6-7 %	18-19%
Gradient Boosting Regressor	8-16 %	16-20 %

Table 3: Average error of the algorithms for the training and testing

The first thing we can see in this table is that the Linear Regression has the highest testing error. This high error rate is most certainly due to bias, as the relationship between the training features and the target feature is likely not linear. Hence, a linear regression cannot

accurately capture the relationship. The testing and training error being close means that overfitting is not a factor.

The decision trees used in the other two algorithms, by contrast, exhibit nonlinear decision boundaries. Hence, they have relatively low bias. On the other hand, they are susceptible to overfitting and have higher variance than a simple linear regression.

Random Forests will have the lowest training error. Remember that this algorithm builds a large set of decision trees based on randomly subsampling the data points and features (bagging). Then the Random Forest takes a majority vote among the collection of trees to select the output. Having different uncorrelated decision trees improves the overall performance as the trees usually cover the distinct flaws of each other. The low training error is due to the use of *RandomizedSearchCV()* to get the parameters that minimize that error. However, because this function obtains the optimal parameter for the training dataset, this will result in overfitting, with the testing error being considerably higher.

Gradient Boosting Regression builds a series of shallow decision trees (weak learners with high bias and low variance) that improve on their predecessors' errors (boosting), iteratively lowering the error rate. This algorithm will have a bigger error range for different simulations. The training error is higher than the one of the Random Forests. Even though the *RandomizedSearchCV()* function tries to minimize the error, this is not as effective as before. Also notice that the difference between the testing and training error here is smaller. This means that overfitting is not a big factor in this algorithm and therefore the more iterations are used the more accurate the model gets.

Even though the Gradient Boosting Regression is an improvement from the others, the average error rate is still higher than desired. Recall that the average median error of Zillow's *Zestimate* for off-houses market is 6.9 % Of course aiming for an error as low as that seems far-fetched. Zillow has a great number of resources including access to a database containing more than 100 million houses, great computational power, and a group of expert realtors. This project does not even come close to this as the listings data only consist of one zip code. Considering this, the target error for this estimator was set to around 15 %

Many actions have been attempted to try to bring the error rate down. There was a belief that the number of data samples used for the estimator (around 2500) could not be enough, so listings from earlier years were included. This was an attempt to improve the accuracy of the model by feeding it more training data. However, the listings from those years seemed to cause a bias to the model and ended up increasing the error rate. The issue was that those listings were sold for prices that would be considered too low in today's market, bringing down the estimated price of most properties.

There was also an attempt of modifying the features used for the model like including those that were dropped for their low correlation with the Sold Price field. This was, nevertheless, proven unsuccessful as the overfitting increased because of adding columns irrelevant to the target value. Moreover, the removal of any of the features from the model seemed to increase the error rate instead of decreasing it which means that they are all important pieces of the algorithm.

The conclusion obtained is that the model was correctly defined, and the problem resided in the data. There was just data from one zip code which limited the number of samples, compromising the training phase. Plus, the data was badly formatted in many rows causing them to be removed, decreasing even more the number of listings used. We also have to take into account that the zip code we are working with is in Hollywood, which means that most of the listings are luxury houses. The prices of these houses can throw off the algorithm as the features of the listing may not align with the selling price. The model would probably work better in a regular neighborhood of a city where the prices are closer to the norm.

Chapter 7. CONCLUSIONS AND FUTURE WORK

This section will serve as a summary of the project, including an overview of the work that has been completed, an evaluation on the fulfilment of the initial objectives and possible future improvements that can be added.

7.1 WORK COMPLETED

The following enumeration may serve to the reader as a superficial rundown of the tasks that were completed during the duration of the project.

- The User Interface of the app was completed which included the following screens:
 - The login screen where the user can login with their own account or using a Google's account
 - The register screen where the user can create a new account by entering an email, personal data, and a new password
 - The filter screen where the listings search can be filtered by different features like the year of construction, number of bedrooms, estimated price, and the surface in SqFt
 - The map screen where the listings that satisfy the previous filters are displayed on a map provided by the Google Maps API. The listings can be clicked for further details like the address, surface, estimated price, number of bedrooms or price per SqFt. Here the listings can also be bookmarked in the user's profile for later use.
 - The bookmarks screen where the listings that are bookmarked by the user will appear in a list
 - The bookmarks details screen, which is accessed by clicking on a listing in the bookmarks screen. It allows the user to see all the features of the listing, remove it from bookmarks and add comments to it

- The profile screen, containing personal information of the user like the name and the number of current bookmarks. It also has an activity log, where the actions of bookmarking or removing bookmarks are recorded with a timestamp. This log can be cleared by the user.
- The connection and set up of the Firebase Authentication service to manage tasks such as the creation of user accounts, login, and storage of user's data. This information is controlled with utterly privacy and security.
- The creation of a database with two tables. The information is stored using Firebase Realtime Database in JSON tree format. Rules are added to the database to properly control who can access the records. The listings table will contain the complete information of a listing, including the main features and the results from the machine learning algorithms. The users table contains the basic user's information (name, email, etc.) plus the array of bookmarked listings and the entries of the activity log.
- The cleaning and processing of listings raw data so that the properties can be used in the price estimator algorithms. This phase required modifying the format of some fields, dropping columns and rows and other similar tasks.
- The development of a price estimator of a property in a certain zip code based on its features. This estimator is available with three different algorithms: Linear Regression, Gradient Boosting Regressor and Random Forests. The script will receive the data of houses that are currently off-market and return a suitable price tag.
- The creation of an algorithm that splits the listings into 5 different bins based on how likely they are to be sold by their owners. After the current price of a house is computed, it will be compared to the last sold price to obtain the annual percentage price growth. This percentage will serve as a way to divide and color code the listings accordingly.

7.2 OBJECTIVES FULFILLMENT

The **development of the application** has been defined as a success by the clients as they are satisfied with the result. They claim that the User Interface is visually appealing, intuitive, and easy to use. Moreover, the app functioning is smooth and flawless as all the major bugs have already been fixed.

There have been some modifications to the initial objectives of the app development. Firstly, the farm delineation has not been included. This is a feature that would have allowed the user to draw areas in the map screen so that the agent can focus on the listings within that territory. The coding of this feature proved to be harder than expected and the proposed solution was different from what the clients had in mind. They finally decided to drop the feature due to the added complexity and the fact that the filter screen was more useful than this delineation.

On the other side, there has been two features which did not appear in the initial plan: the comments on bookmarks and the activity log. They were proposed to the clients during the development of the app, and they agreed that they would be great additions. Adding comments on a bookmark allow realtors to keep notes so they can easily remember things about a listing. Also, the activity log allows the user to keep track of the different movements made on the bookmarks, helping them remember the date and address of the property involved.

The **price estimator** could be considered a partial success. The product offered to the clients is a functional package where the user can select between three effective algorithms (Linear Regression, Gradient Boosting Regressor and Random Forests) and they can also choose certain parameters of the algorithms, with the parameters that are not specified obtained via the *RandomizedSearchCV()* for optimality. The issue with the estimator is that the average testing error is higher than desired in most of the simulations. This error is usually between 17-22 % with the Linear Regression having the highest error rate and Gradient Boosting Regressor the lowest. This range is slightly higher than the 15% originally set as a target,

and no modifications to the models seemed to lower the error rate. I arrived at the conclusion that the problem is not actually related to the model definition but rather with the data of the listings. As mentioned in the *Results Analysis* section, the Hollywood zip code studied is filled with luxury houses where the price does not always correspond with the main features. Also, the number of properties used to train the model may not be enough to achieve a great level of accuracy.

In conclusion, the average error percentage is slightly worse than anticipated, and higher than the one from other platforms like Zillow. However, the rate is still decent and a useful value for the ultimately purpose which is to compute the sale likelihood. Also, the issue is not due to an incorrect model but to inconclusive data, and a better accuracy could be achieved with better data and resources.

It looks like the algorithm that computes **how likely is a house to be sold** could really help realtors filter the owners they approach about a sale. The clients' suggestion that this could be based on the potential return on investment (ROI) for the owner is reasonable, so the obtained algorithm that divides the houses into 5 bins can be considered a success.

7.3 FUTURE IMPROVEMENTS

This project is expected to serve as a proof of concept to the clients so the application will not actually be published in the Play Store or App Store. Instead, the clients are expected to use the project to fully develop and deploy an app for realtors in the future as they have more time and resources to do so.

Should someone want to continue with this project, here are some possible improvements that could be made:

- **Include the contact info of the homeowner of a listing.** Adding a phone number and an email would save a lot of time to the realtors when it is time to approach an owner. The problem is that this information is hard to get, and not all the Real Estate Agencies have it.
- **Allow customization of the app.** Adding a profile picture of the user in the profile screen or giving the user some customization tools like allowing them to change the color are some simple ways to improve the User Experience of the application.
- **Retrieve the listings data from an API.** The data is now obtained from the MLS website manually which is not ideal. If one had access to an API, the data collection could be done automatically and periodically. For example, every night a script could be run to get the new listings sold on the day and update the data. The clients did not provide an API access because it is quite expensive, but someone with intentions to deploy the application should make that investment.
- **Minimize the error average of the price estimator.** This could be done by adding listings from more zip codes to the training data. These zip codes should be mostly located in zones where the prices are reasonable. This would increase the number of samples used and improve the quality of the data.
- **Explore new ways to obtain the likelihood of sale.** The algorithm used in this project is expected to work. However, realtors must still explore possible paths as they might find a way to predict this with better accuracy in the future.

- **Host the machine learning block in the cloud.** Right now, the machine learning algorithms are run locally, so only the computational power of laptop is used. With the current volume of data, this is acceptable. But as the data grows, so does the complexity, and a computer would not be able to handle the processing. My recommendation is to use AWS or Google Azure to execute the algorithms in the cloud.

Chapter 8. BIBLIOGRAPHY

- [1] “The importance of real Estate”, European Real Estate Forum, June 2016
<http://www.europeanrealestateforum.eu/economy/#:~:text=Real%20estate%20is%20one%20of,tenants%20and%20their%20consumer%20spending>
- [2] Bortz, D. “Real Estate Agent vs. Broker vs. Realtor: What’s the Difference?”, Realtor.com, May 2022, <https://www.realtor.com/advice/buy/whats-difference-real-estate-salesperson-broker/>
- [3] “Housing’s Contribution to Gross Domestic Product”, National Association of Home Builders (NAHB), December 2021, <https://www.nahb.org/news-and-economics/housing-economics/housings-economic-impact/housings-contribution-to-gross-domestic-product#:~:text=Housing's%20combined%20contribution%20to%20GDP,homes%2C%20and%20brokers'%20fees>
- [4] “Real Estate Sales and Brokerage in the US”, Ibisworld, December 2022
<https://www.ibisworld.com/industry-statistics/market-size/real-estate-sales-brokerage-united-states/>
- [5] Taylor, A “Great Resignation quitters are rushing to get real estate licenses. Coldwell Banker’s CEO explains why that is, and how he keeps his team happy”, Fortune, May 2022, <https://fortune.com/2022/05/02/real-estate-agent-career-popularity-coldwell-banker-ceo/>
- [6] “Getting Started with React”, Mdn Web Docs, October 2018,
[https://developer.mozilla.org/es/docs/Learn/Tools_and_testing/Client-side JavaScript frameworks/React getting started](https://developer.mozilla.org/es/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/React_getting_started)
- [7] “JavaScript”, Mdn Web Docs, June 2022,
<https://developer.mozilla.org/es/docs/Web/JavaScript>
- [8] “Visual Studio Code”, Wikipedia, November 2021,
https://en.wikipedia.org/wiki/Visual_Studio_Code
- [9] “Firebase Realtime Database”, Firebase Documentation, June 2022,
<https://firebase.google.com/docs/database>

- [10] “Firebase Authentication”, Firebase Documentation, June 2022,
<https://firebase.google.com/docs/auth>
- [11] “Pandas (software)”, Wikipedia, Jan 2022
[https://es.wikipedia.org/wiki/Pandas_\(software\)](https://es.wikipedia.org/wiki/Pandas_(software))
- [12] “Trabajando con Pandas DataFrames en Python”, Data Carpentry, July 2018
<https://datacarpentry.org/python-ecology-lesson-es/02-starting-with-data/#:~:text=Un%20DataFrame%20es%20una%20estructura,de%20SQL%20o%20el%20data>
- [13] “What is Matplotlib?”, W3Schools, February 2021,
https://www.w3schools.com/python/matplotlib_intro.asp
- [14] “Scikit-learn, ¿qué es?”, Itop Tecnología y negocio, March 2021,
<https://www.itop.es/soluciones-tecnologicas/business-analytics-business-intelligence/scikit-learn.html>
- [15] Brownlee, J. “Linear Regression for Machine Learning”, Machine Learning Mastery, March 2016,
<https://machinelearningmastery.com/linear-regression-for-machine-learning/>
- [16] Srinidhi, S. “Linear Regression in Python using SciKit Learn”, Medium, July 2018,
<https://contactsunny.medium.com/linear-regression-in-python-using-scikit-learn-f0f7b125a204>
- [17] Glavin, F. “An example of a simple high-level decision tree for a first-person shooter NPC”, ResearchGate, September 2015,
https://www.researchgate.net/figure/An-example-of-a-simple-high-level-decision-tree-for-a-first-person-shooter-NPC_fig3_292141020
- [18] You, T. “Understanding Random Forest”, Towards Data Science, June 2019,
<https://towardsdatascience.com/understanding-random-forest-58381e0602d2>
- [19] [20] Masui, T. “All You Need to Know about Gradient Boosting Algorithm – Part 1. Regression”, Towards Data Science, Jan 2022,
<https://towardsdatascience.com/all-you-need-to-know-about-gradient-boosting-algorithm-part-1-regression-2520a34a502>
- [21] “About Zillow”, Zillow, May 2021,
<https://www.zillow.com/z/corp/about/>
- [22] [23] [24] “What is a Zestimate?”, Zillow, May 2022,
<https://www.zillow.com/z/zestimate/>
-

- [25] “About Realtor.com”, Realtor.com, May 2022, <https://www.realtor.com/about/>
- [26] “Redfin”, Wikipedia, June 2022, <https://en.wikipedia.org/wiki/Redfin>
- [27] “What is Trulia?”, Trulia, October 2021, <https://support.trulia.com/hc/en-us/articles/205995978-What-is-Trulia->
- [28] “How does selling to Opendoor work?”, Opendoor, May 2022, <https://www.opendoor.com/w/faq/how-does-selling-to-opendoor-work>
- [29][30] “How much is my house worth?”, Opendoor, May 2022, <https://www.opendoor.com/w/home-value>
- [31][32] “How much will I make selling my house?”, Opendoor, May 2022 <https://www.opendoor.com/w/home-sale-calculator>
- [33] Wilson, D. “What Is a Half-Bath, Quarter Bath and Three-Quarter Bath?”, The Realty Firms, August 2020, <https://therealtyfirms.com/half-bath-quarter-bath-three-quarter-bath/>
- [34] “The MLS.com”, MLS, May 2022, <https://www.themls.com/>
- [35] Zach. “How to read a correlation matrix”, Statology, January 2020, <https://www.statology.org/how-to-read-a-correlation-matrix/>
- [36] “sklearn.ensemble.RandomForestRegressor”, Scikit learn, March 2022, <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html>
- [37] “sklearn.ensemble.GradientBoostingRegressor”, Scikit learn, March 2022, <https://scikitlearn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html>
- [38] “Do you know all 17 SDGs?”, United Nations, June 2022, <https://sdgs.un.org/goals>
- [39] “Goal 8 | Department of Economic and Social Affairs”, United Nations, June 2022, <https://sdgs.un.org/goals/goal8>
- [40] “How much does the average Real Estate Agent make?”, Real Estate Express, February 2022, <https://www.realestateexpress.com/career-hub/blog/how-much-does-the-average-real-estate-agent-make/>

[41] “Goal 13 | Department of Economic and Social Affairs”, United Nations, June 2022,
<https://sdgs.un.org/goals/goal13>

ANNEX I: PROYECT ALIGNMENT WITH THE SDGs

The Sustainment Development Goals (SDG) can be defined as a set of 17 goals that are designed to make the world a better place [38]. This annex will serve as an explanation of how the project can contribute to these goals. Out of those goals I have chosen one to be the main SDG and another to be the secondary SDG.

The main SDG of the project is the number 8: *Promote sustained, inclusive, and sustainable economic growth, full and productive employment, and decent work for all* [39] This goal is in the economic dimension, and it tries to ensure that the economy of every country can provide a good life for all their citizens independently of their background. One good indicator of this could be the Gross Domestic Product (GDP) per capita of a country, and also facts about the current situation of the job market: unemployment rate, average salary... More specifically, the target is 8.5: *By 2030, achieve full and productive employment and decent work for all women and men, including for young people and persons with disabilities, and equal pay for work of equal value*. This basically means that every person should have access to a job with good conditions and that at least allows them to buy the basic necessities humans have. This should be satisfied independently of the race, gender, or physical abilities of the person, and the salary received should be the same for the same job. Now let's see how this SDG and goal comes into play in the project.

Real Estate agents obtain profits with a commission on the sale price, which is usually around 5%. Therefore, the more houses they sell the more they usually make, while also getting bigger earnings when the house is expensive. Selling a house requires a lot of time: approaching the owner, negotiating the deal, organizing open houses... Consequently, there is a correlation with the number of hours that a realtor works and salary they obtain, as we can see in this graphic.

Average Income by Hours Worked Per Week

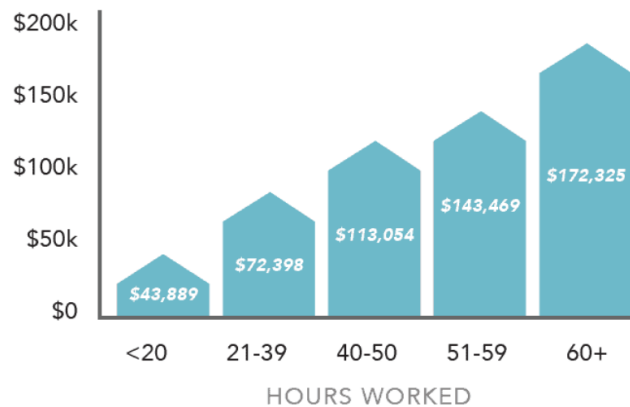


Figure 35: Average income of realtors by hours worked per week [40]

The aim of the project is to allow the realtors have an idea of how willing a homeowner would be to selling. Therefore, this would help realtors save time by filtering the number of homeowners approached. Imagine an agent which on average has to contact 6 homeowners in order to get one to agree to a sale (16.67%). With the help of this application, we could make an estimation that the same agent would only need to contact 2 homeowners on average. This is because the agent would only approach those owners whose house has a high likelihood of sale according to the app. This prediction is obviously not perfect, but it still results in a higher success rate. This higher rate means that the realtor will save time in the search process and in estimating the price tag of the house, so let's assume that the agent will now spend on average around 33% less time for each house. If for example the realtor works 20 hours per week with the help of the app, this would be equivalent to 30 hours per week without the app.

If we go back to the previous graphic, the salary of the realtor got higher as the working time per week increased. The app is intended to aid a realtor sell more houses per hour worked, so it basically increases their average hourly earnings, which refers to one of the indicators of the chosen goal: *8.5.1 Average hourly earnings of female and male employees, by occupation, age and persons with disabilities* If we try to quantize this increase, based on

the previous estimations each house will take around 33% less time, which means that the hourly salary for the realtors using this app could increase as much as 50 %

The secondary SDG I have chosen is 13: *Take urgent action to combat climate change and its impacts* [41] that belongs to the Biosphere dimension. Climate change is one of the biggest issues the humanity faces due to its negative consequences: extreme heat, wildfires, food and water scarcity and floodings. Climate change is being accelerated by human activities like fossil fuels burning, so this must be controlled to avoid critical consequences. In particular, the target selected is 13.2: *Integrate climate change measures into national policies, strategies, and planning*. The objective here is to reduce climate change by trying to regulate the human activities causing it. These regulations must be put in place by countries so that they can be effective against climate change.

The project does not directly contribute to the SDG. It is actually a consequence of the app what will be positive towards the achievement of the goal. As we already mentioned for the main SDG, the app tries to make a prediction on which houses are more likely to sell. This helps realtors filter down the owners they will approach. Contacting owners does not just involve a phone call or email, as the agent will usually go to the house in person to talk to the owner and to take a more detailed look at the house. These trips often end up being useless as the owner might decide not to sell. Also, most times the agents will drive to the houses as they are usually located in the suburbs. Cars emit several greenhouse effect gases like carbon dioxide, methane and nitrous oxygen which are harmful to the environment as they accelerate climate change. The application intends to notably reduce the realtors' trips that do not end up in a deal, actively reducing the greenhouse effect gases emission. This aligns with one of the indicators of this goal: *13.2.2 Total greenhouse gas emissions per year*