cct | College Dublin
Computing • IT • Business

PROBLEM SOLVING FOR INDUSTRY

HOUSE PRICE PREDICTION

Analysis and Prediction of Properties Price in Dublin

BSc (Hons) Computer Science and Information Technology

Year 4 | May 2021

Henrique Wegner (2017403)
Larissa Evaldt (2017270)
Leandro Silveira (2017369)
Maiara Figueredo (2017389)
Patricia Correia (2017352)

ABSTRACT

Provide international real estate investors with observations and predictions on Dublin property values. Losing money is not the intention when making an investment. There are companies and websites that offer house prices and other statistics to stakeholders, regardless of whether the city is wealthy or not. However, in order to satisfy the needs of overseas investors who want comprehensive knowledge about Dublin real estate prices in a single source, this machine learning model can forecast market prices based on the location and amenities nearby. “How much the properties are worth in Dublin”? Supervised machine learning, algorithm, regression, and data mining are applied in this study to answer this question.

Contents

Introduction

The Irish estate market has been growing, especially in Dublin in the last three years where there is a growing demand for commercial properties and houses, according to Padraic Whelan on the Deloitte website, when talking about the Irish market after Brexit (Whelan, 2021).

Either the investment is on commercial or residential properties we see a rise in the real estate market. Dublin is an extremely attractive area in the country. With so many tech companies in town, it is a region where investors would be interested in doing business (JLL, 2020).

However, traditionally investors would have to research the assets on the market to get information about whether the investment in Dublin a good idea would be or not, when to do it, or even which area to finance. In brief, we propose a predicting model, which uses IT technology, such as data mining, data analysis, machine learning algorithm, and regression to allow us to predict property prices in Dublin.

With this trend in the Irish market, we aim to develop a property prediction model in Dublin as a tool to give international investors insights into the Irish market. The idea of using IT and business strategy combined will benefit investors about decision making regarding the real estate market.

1. Roles and Responsibilities

Role	Responsibilities	Person(s)
Supervisor	· Provide support, project oversight and guidance. · Evaluate and mark the project.	Mark Morrisey Dr. Graham Glanville
Researcher	· Determine the appropriateness of the topic to be investigated. · Detect what it is needed to build the project. · Perceive and collect the pertinent data. · Detect which is the most suitable ML model to be used.	Larissa Evaldt Maiara Figueredo Patricia Correia Henrique Wegner Leandro Silveira
Developer	· Observe, analyse and understand the data. · Perform data cleaning and prepare the data. · Perform Data Feature engineering · Provide data visualization. · Develop the code tailoring the ML model to the project · Perform tests to assure successful results	Larissa Evaldt Maiara Figueredo Patricia Correia Henrique Wegner Leandro Silveira

Documentation content quality reviewer	· Assure the quality of documentation that will meet the project goals and objectives. · Assemble and systemize the documentation content. · Guarantee the conciseness of the paper, as well that it is clear and understandable	Larissa Evaldt Maiara Figueredo Patricia Correia Henrique Wegner Leandro Silveira
General	· Communicate project priorities, goals, status, and progress throughout the project. · Provide knowledge and recommendations by sharing their expertise and make suggestions. · Identify risks and issues and help in resolutions	Larissa Evaldt Maiara Figueredo Patricia Correia Henrique Wegner Leandro Silveira

2. Responsibility Assignment Matrix

Responsibility Assignment Matrix (RAM)

Detailed Task Divisions With Phases Breakdown

Responsible, Accountable, Consulted, Informed

Deliverable or Task	Status	Supervisor			Project Team						Other Resources		
Participants		Mark Morrisey	Dr. Graham		Patricia Correia	Maiara Figueredo	Henrique Wegner	Leandro Silveira	Larissa Evaldt		Other Lectuers	Outsourcing: Libraries, websites	Classmates
Phase 1													
Determine the appropriateness of the topic to be investigated	✔	A	A		R	R	R	R	R		C	C	
Provide support, project oversight and guidance	✔	R	R		I	I	I	I	I		S	C	
Phase 2													
Research and document	✔	C	C		D	R	S	A	R			C	
Data search and collection	✔	C	C		R	A	R	D	R		C	C	
ML model selection	✔	I	I		R	R	A	S	R		C	C	
Phase 3													
Data exploration, preparation and Feature engineering	✔	I	I		A	R	D	R	S			C	
Phase 4													
Development of ML tailored model and implemantation	✔	I	I		S	R	A	R	D			C	
Training and Testing	✔	C	I		S	R	S	D	A			C	S
Phase 5													
Project Evaluation	✔	C	C		R	R	R	R	R			C	S
Phase 6													
Deployment and Post-Deployment	✔	S	I		R	D	R	R	R		I	C	I
Marking Project		R	R		I	I	I	I	I		I	C	I

D	Driver	Assists those who are responsible for a task.
R	Responsible	Assigned to complete the task or deliverable.
A	Accountable	Has final decision-making authority and accountability for completion. Only 1 per task.
S	Support	Provides support during implementation.
C	Consulted	A subject matter expert who is consulted before a decision or action.
I	Informed	Must be informed after a decision or action. Could be informed of a final model

Table 1:Roles and Responsibilities

3. Business Understanding (CRISP-DM)

Our project process will assist us in comprehending the project's goals as well as all the criteria.

Our aim is to consider our clients' needs from a business standpoint and identify our business requirements to build resources to satisfy them.

3.1 Analysis of Irish Property Market

Real estate has been one of the most solid markets in our times. Andrew Carnegie and other billionaires have stated that 90% of millionaires became so wealthy due to investing in real estate (The Oracles, Contributor, 2019). Some of these millionaires are foreign investors putting money into properties overseas for various purposes.

This market plays a crucial role in the Irish economy especially after the financial crisis which erupted in 2008. In 2016, the Irish Times published that the real estate market in Ireland was dominated by non-Irish investors living in other countries (Fagan, 2016). Furthermore, the same article mentions that foreign investors continuously appear to make their business in areas like Blanchardstown – Dublin, Newbridge – Kildare, and many others (Fagan, 2016) and their interests are to purchase valuable commercial assets such as shopping centers and town centers, as well as properties to rent out, large buildings to turn into head offices and private houses to resell.

In addition to this, foreigners willing to invest in Ireland tend to have their attention caught by the attractiveness of Dublin city, the capital of Ireland. The website JLL (2020) points that capital coming from overseas was responsible for nearly two-thirds of overall investments in 2019, “and for office investment, they were attracted mainly by newly-developed office assets in central Dublin – as well as the city’s Silicon Docks reputation as a home for tech firms from Google and Facebook to LinkedIn and Airbnb”. Moreover, investors and big company names in the United Kingdom have rethought the idea of remaining based in London, reappointing their offices to Dublin and resulting in having their employees moving as well. Consequently, the real estate business in Dublin is highly affected, leading to the unpredictability of property prices which rise high even in the outskirts of the city.

Crucial pieces of reliable information regarding areas in Dublin are becoming harder to get. Some areas which had poor infrastructures have been rapidly developed in the past few years and are now a great opportunity to invest in. On the other hand, there are regions in Dublin taking the reverse path and foreign investors can only

retrieve the information about each area by researching through multiple sources and making sure they can trust what they read.

Among the foreign investors, there are well-off individuals and companies buying new properties to expand their business all over the world. There are also low-profile businesspeople or even house buyers looking into investing overseas. The way these different investors would put money into real estate in a foreign country would differ slightly, especially because many of them know very little about Ireland, however the mindset of not losing money will be alike. Andrew Henderson, an important consultant on international tax planning and investment immigration says that regardless of what the future holds, an investor must ensure that any property or land they purchase is in a location that will remain outstanding and attractive for ten, twenty, or even fifty years (Henderson, 2020).

This very same consultant also advises investors to be careful with some gorgeous properties that are advertised without the full information about the area where they are located, since homeowners will not reveal if the region is dangerous or not (Henderson, 2020).

On the other hand, property sellers have a contrasted goal. They aim at selling their properties in a short period of time and for the highest value possible, however they also lack a system that informs property buyers of how the real estate market is according to the areas in the city. The web page mtsproperty.com reveals that it could take up to seven months to sell a house in Ireland due to its location and how near it is to amenities and schools (Property Dublin, 2021). Six months can signify a long wait depending on the financial situation of sellers or any other reason why they would need to sell a property.

3.2. Market Target

It is based on these factors given above that this project attempts to assist potential foreign investors with their homework of researching everything about the area of the property they intend to purchase. Developing a prediction machine learning model that utilizes one dataset containing the development of Dublin city areas

through the years which, according to the Pinnacle list, can impact property prices (The Pinnacle List, 2021). These developments are new public transport lines such as Luas, dart, and bus lines, as well as schools, Garda station, and so on.

This list of attributes will be retrieved through data science methods to collect the data. The data collected will be treated and prepared when necessary. This will be done by utilizing data exploration and preparation methods to have a clean and precise dataset. Sequentially, this data will be analysed and processed in a regression machine learning algorithm, predicting the value properties in the Dublin area.

The final product will be a map of Dublin city that could be placed on a website and that users can interact with. This map will have the areas of the city so a user can insert an address, for example on Finglas. The map will then display the current average price of houses in that corresponding area. Responding to a click from the user in that exact same area. The user can then use a feature of the model which predicts how much that area cost.

This machine learning model shrinks the work of navigating on various websites, placing phone calls, and going place to place to gather information about the area where the property a person intends to buy is located. This information will be grouped in one place and available for these foreign investors.

3.3. The Current Market

According to Tara Struyk's article for Investopedia (2021), location is an extremely important factor to take into consideration when looking to invest in property and there are quite a few different things regarding the location that could influence how valuable the property is and how easy it would be to sell or rent in the future. Usually, when choosing a region, the most important things to look out for are accessibility, amenities, safety, and development in the area.

Regarding accessibility, you should check how easy it is to get to the area, an area with easy road access and good public transportation links is always better since most people need to commute to and from work almost every day. For amenities, check if there are supermarkets, shops, restaurants, and schools close by the properties. Even if you do not have kids, if you plan to sell the house this is something

that other people will be likely looking out for and good schools in the area could definitely make the property more attractive to future buyers.

Plans for new developments in the area that you want to invest in is also something important to be checked, for example, if there are new schools or hospitals planned to be built, new public transport links or any other civic infrastructure or commercial development could make the value of the property improve and could also make people more interested in that area.

Safety is a very important factor as well, so do not forget to look out for the crime rates in the area.

Another thing to watch out for is that usually houses located in very busy streets with a lot of transit could have a lower value or could be more difficult to sell because of the noise, as well as houses right beside schools, large churches because of the amount of transit of people and cars parked (Investopedia, 2021).

4. Current tools available in the market to research and analyse Dublin areas.

4.1.1 Google Maps

Google Maps: This is the simplest way of starting to search a region, most people already have either Google Maps or some other map app on their smartphones and there is a lot of information that you can find just by looking at the map, it would show most amenities in the area, parks, transport links, etc.

4.1.2 Maps Pobal

Maps.pobal.ie website: Pobal (2020) Maps is a free Geographical Information System that is built based on the 2006, 2011, and 2016 census and basically aims to show how affluent or deprived an area is. Factors taken into consideration are the proportion of skilled professionals, education levels, employment levels, and single-parent households found in an area. The map is easy to use, and areas are color-

coded from blue to green, yellow, and orange. Blue is the most affluent area and orange being the area in the biggest disadvantage.

Check out below two screenshots of the map of Dublin, one zoomed out so the whole county can be seen and one more zoomed-in showing that you can also click on a specific area and see more information, like the population numbers and the deprivation for each of the three censuses, so you can see whether the deprivation indices improved or decreased, giving good insights on whether the area developed over the years or is getting worse. For instance, the Saggart area in the picture below was considered very affluent in 2006, marginally above average in 2011, and affluent in 2016 (Pobal, 2020).

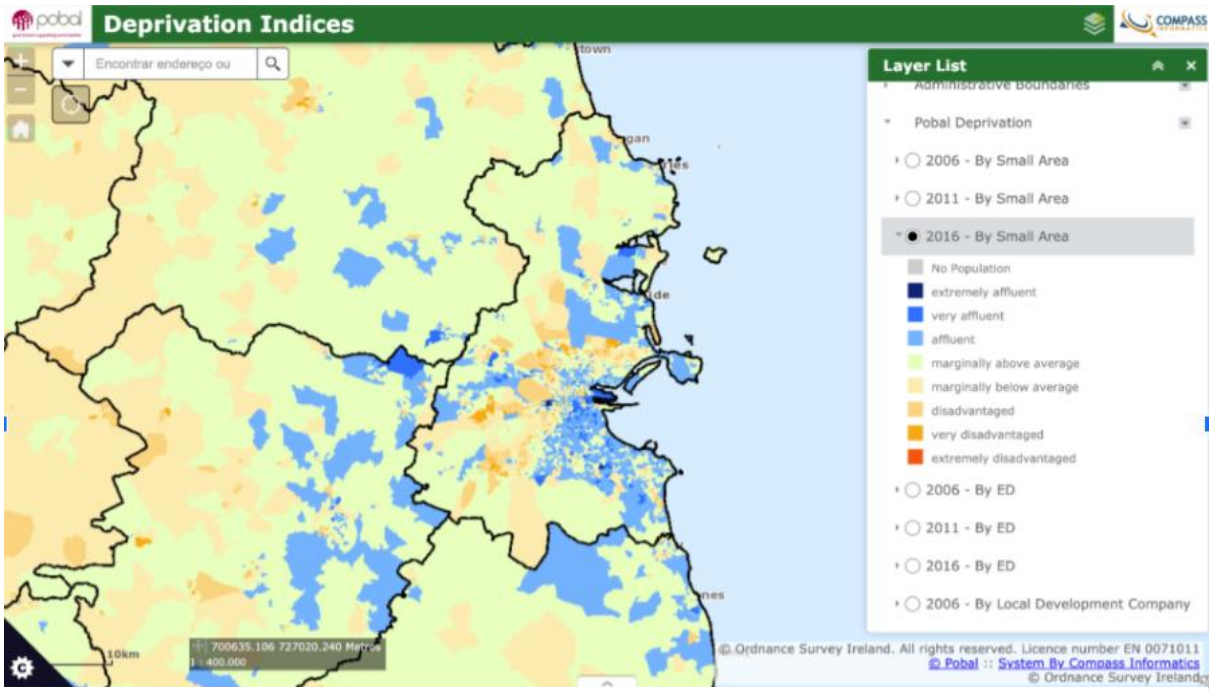

Figure 1: Pobal HP Deprivation Indices Map. Analysing areas with High Levels of affluence or disadvantages by small area. Retrieved March 5, 2021, from https://maps.pobal.ie/WebApps/DeprivationIndices/index.html. Screenshot by author.

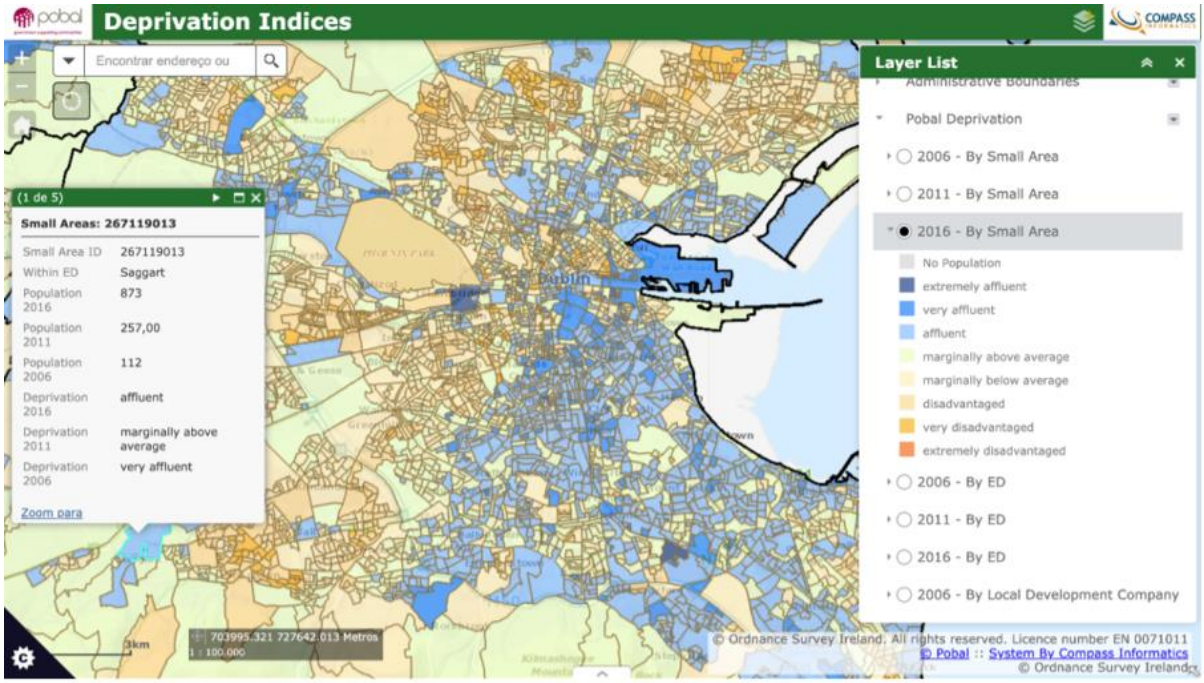

Figure 2. Pobal HP Deprivation Indices Map. Analysing areas with High Levels of affluence or disadvantages by small area. Retrieved March 5, 2021, from https://maps.pobal.ie/WebApps/DeprivationIndices/index.html. Screenshot by author.

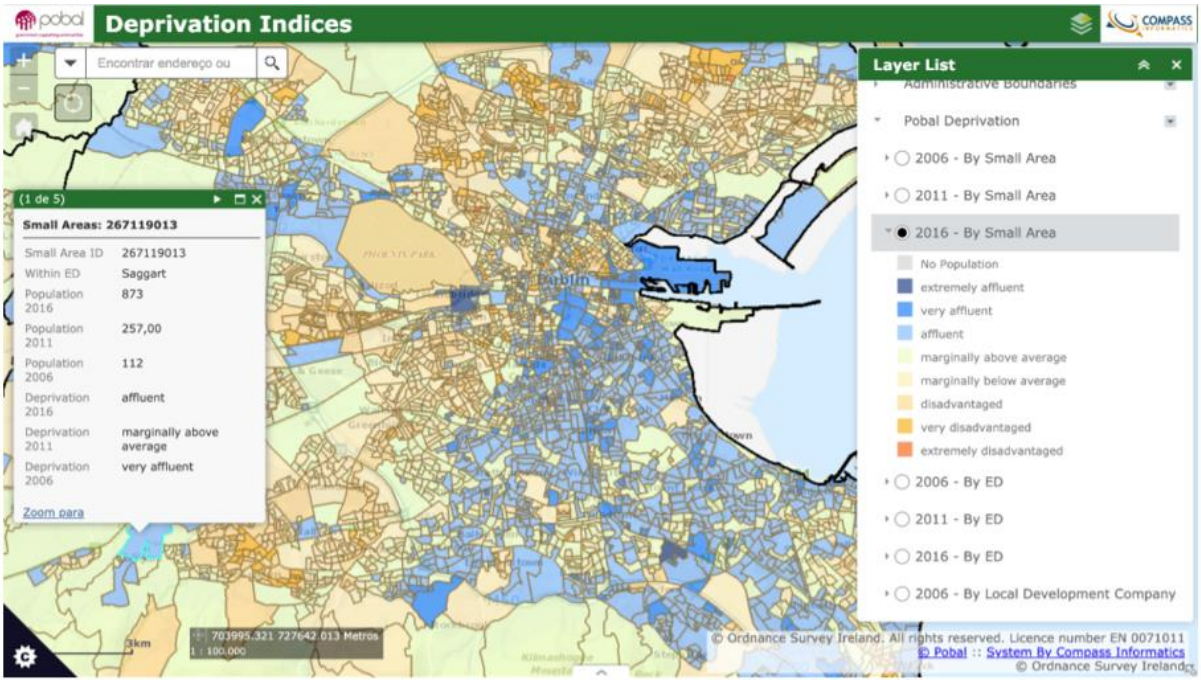

Figure 3.Pobal HP Deprivation Indices Map. Analysing areas with High Levels of affluence or disadvantages by small area. Retrieved March 5, 2021, from https://maps.pobal.ie/WebApps/DeprivationIndices/index.html. Screenshot by author.

4.1.3. House Observatory mapping viewer:

This mapping viewer is a collaborative public sector data project between Ordnance Survey Ireland (OSI), the All-Ireland Research Observatory (AIRO) at Maynooth University, and Dublin City Council. Other key contributors to the project include the Central Statistics Office (CSO) and the Residential Tenancies Board (RTB), with data found from the Residential Tenancies Board (RTB), the property price register (PPR), local authorities, the Revenue Commissioners, and the Central Statistics Office. This is a very good tool to gather more information about the different areas in Dublin, with many features that allow the user to interact with the application, in addition, there is a considerable amount of information available on this website, here are a few of the most interesting ones (Geohive, 2021):

- Local Services in the area: See all the hospitals, GPs, health centers, pharmacies, dentists, nursing homes, childcare, primary and post-primary schools, higher education institutes, garda stations, DART stops, Luas lines, proposed metro link line, train stations, Dublin bus stops, Bus Eireann stops and Dublin Bike stands.
- Current development plans by area.
- Rental Market, Sales, and Property Valuations information from the RTB like average rent price and local rent compared to the national average, tenancies by landlord type, tenancies by type of property, and by the number of bedrooms.
- CSO property price index information like mean and median sales price and volume of sales in 2019.
- Property Price Register (PPR) 2016-2020. Very interesting, we can see little dots pinning the location of every house that was sold, color-coded as red for the most expensive ones and blue and green for the cheapest, making it easy to see the most expensive areas. By clicking on the little dots, we get more information like the address, price, and date of sale. We can also filter for only new homes or only second-hand homes.
- Census key variables like household tenure, type, and composition, the year the houses were built, central heating and water supply information, population density per km2, etc.

- We can also see the mentioned above Pobal Deprivation index.
- Average Building energy ratings (BER) by area.

There is really a lot of information, all in one place. Here is a screenshot of the map viewing all the house sales in 2020:

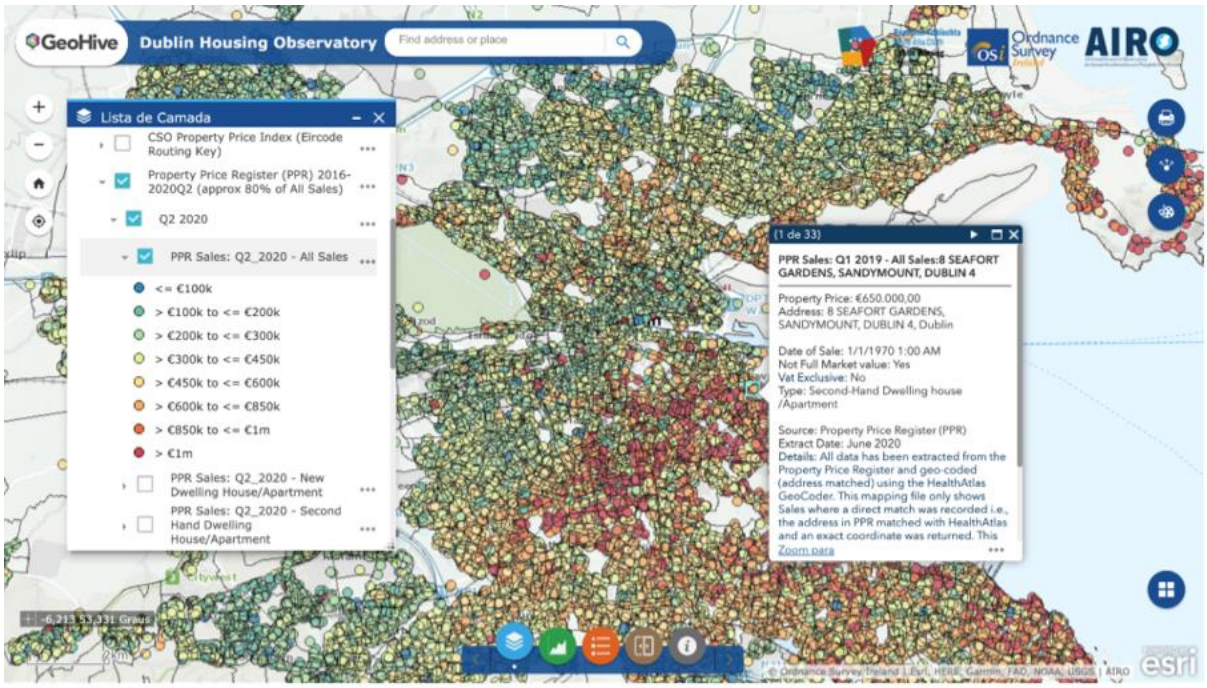

Figure 4. Dublin Housing Observatory. Map displaying all house sales in Dublin according to data from the Property Price Register. Retrieved March 5, 2021, from https://airomaps.geohive.ie/dho/. Screenshot by author.

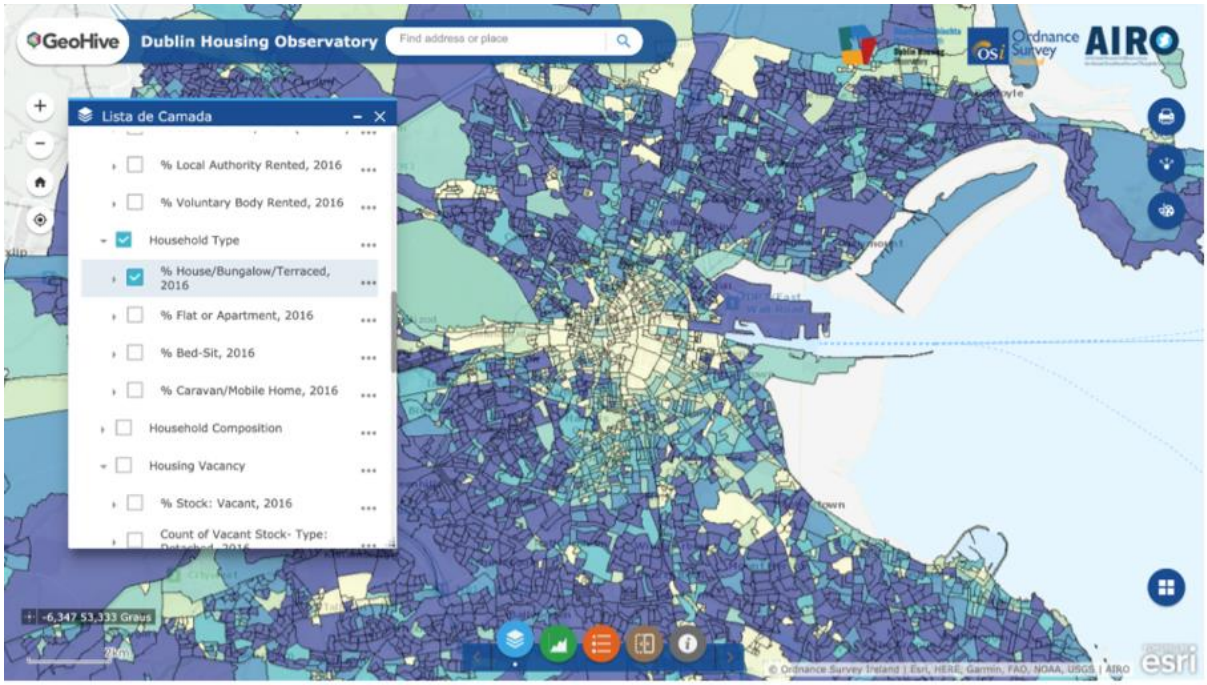

Figure 5. Dublin Housing Observatory. Map displaying percentage of type 'Houses/Bungalow/Terraced' in Dublin according to data from the 2016 Census. Retrieved March 5, 2021, from https://airomaps.geohive.ie/dho/. Screenshot by author.

4.1.4 Project Ireland 2040:

An interactive online map where you can view the details of a variety of projects being delivered as part of Project Ireland 2040 in your area. This is a good tool to check the development of the area and discover what is being built or planned to be built, you can filter by area and by sector. A screenshot showing the MetroLink project (Geohive, 2021):

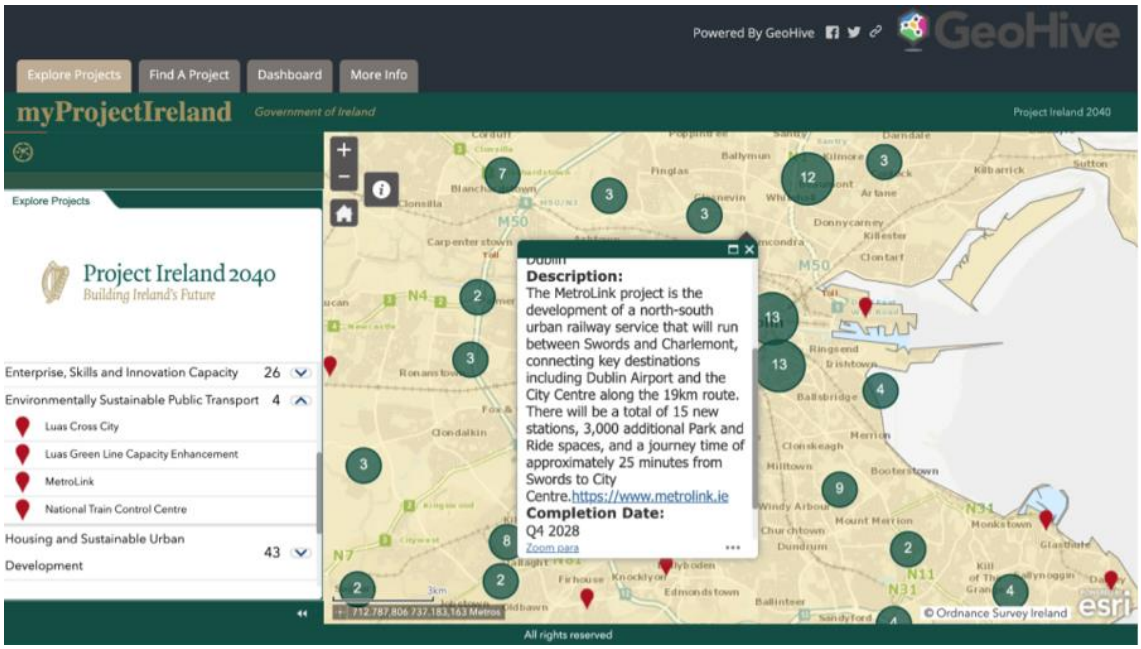

Figure 6. Project Ireland 2040. Map displaying information about the Metrolink project. Retrieved March 5, 2021. from https://geohive.maps.arcgis.com/apps/MapSeries/index.html?appid=f05a0324b1a887cd9d5d7103e22, screenshot by author.

4.1.5 Go Visit the area

If you live in Dublin or have the option to go visit the area, it is a good idea to go for a walk around to check the neighbourhood. You could also do your own research and ask people you know or local's information about the area, in regard to safety, local amenities, transport, etc., in order to make a decision whether it suits your needs. If they know anybody that lives there that you could talk to and ask how safe the area is, how good the schools and public transport links are. But for foreign investors, this is not a viable option.

4.1.6. Websites to search properties in Ireland.

After you have decided on the areas of your interest, you can start looking for properties. There are mainly two websites that are very popular to look for properties to either buy or rent in Ireland, they are called **Daft.ie** and **MyHome.ie**.
The nice thing about these websites is that you can add filters to your search, for instance, how many bedrooms you would like your house to have, in which region you would like your house to be, add in what is your budget, and the website will return a list of all currently available houses that meet your requirements.

4.1.7 Options available in the market aiming to help with property valuations & property market insights.

Property's iPPi (2021) (Independent Property Price Index):

Made for agents and investors to help save time, improve accuracy, and gain market insights. Save time searching for comparable with a database of over 1,000,000 for sale, sale agreed, and sold properties. Create professionally formatted property appraisal documents including comparable properties and market reports for vendors and buyers. Make better decisions for your business with enhanced market trend reports, iPPi can provide specific reports at individual street level, local area, and from a national perspective. Also contains a real-time property dashboard, so you can see property market performance in your area. Updated daily you can see current stock, house prices/ beds, days to sell, and demand index for residential sales and lettings all in one place.

5. Impact of Covid- 19 and Brexit on the Irish market

During our investigation into the state of the Dublin property market for investors, we encountered a variety of circumstances that altered the numbers we needed to analyse. Which is Brexit and COVID-19.

Brexit is the first case. Companies began to spend extensively in Dublin for the long run after the United Kingdom declared in 2016 that they would vote in a referendum to leave the European Union (Sandford, 2020).

As companies including JP Morgan, Bank of America and Barclays relocate their offices from the United Kingdom to Ireland, their employees relocate as well, which impacts the Irish economy.

Additionally, these businesses employed more local employees, resulting in higher wages for local workers. This ensures that the cost of renting and buying a home would increase. “The average cost of a 1-bed apartment in Dublin 1 (which includes IFSC) is €250,000 and generates an average rent of c. €1,605 resulting in a 7.7% yield.” (O'Regan, 2021).

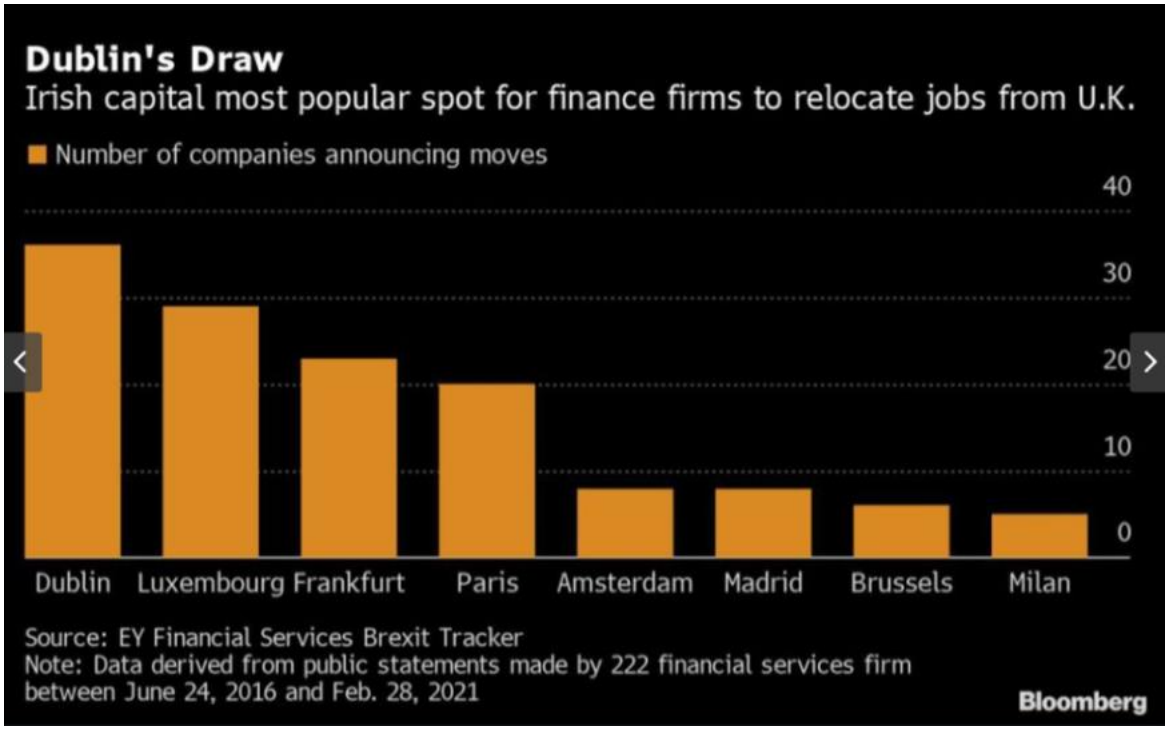

Figure 7. Dublin is Top Brexit Relocation Spot for Finance Firms, EY finds (Metcalf, 2021) Retrieved March 5, 2021, from https://au.finance.yeahoo.com/news/dublin-top-brexit-relocation-spot-000100322.html

According to data from the EY’s Financial Services Brexit Tracker, Dublin remains the most popular destination for staff relocations and new European hubs or offices, with 34 Financial Services Firms saying they are considering or have confirmed relocating operations and/or staff to the city (Graham, 2020).

When more residents move to Dublin, the demand for housing increases so does the price.

About the fact that Covid-19 shocked us and influenced many companies, the property market was not one of them.

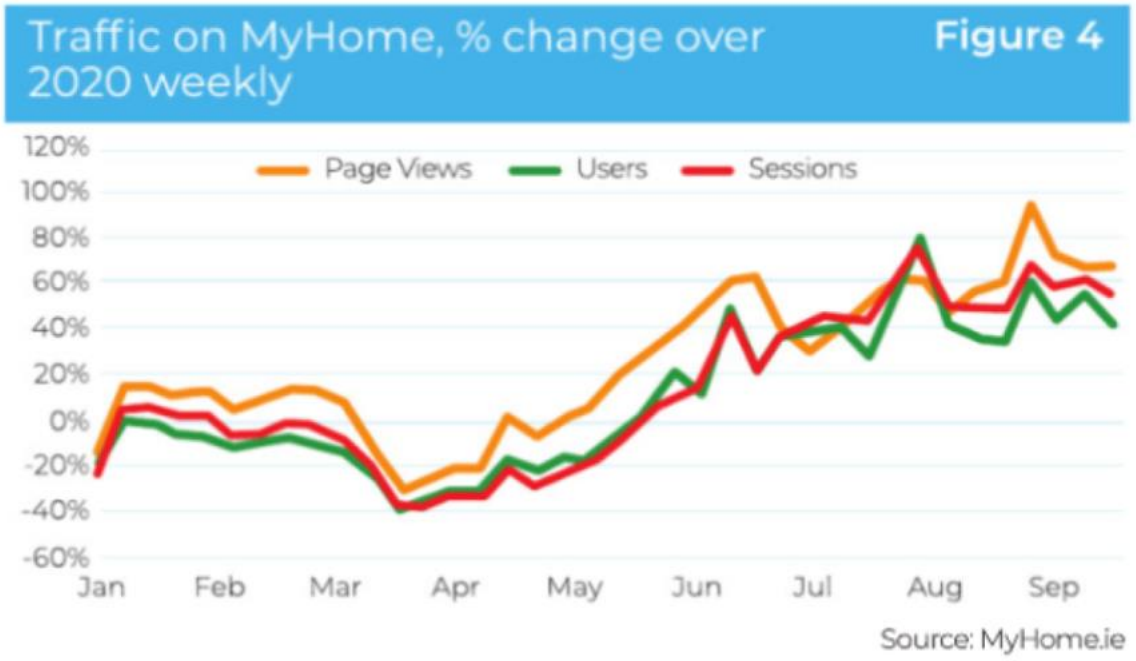

Figure 8. Traffic on MyHome, % change over 2020 weekly. (Epaper.irishtimes.com, 2020) Retrieved March 6, 2021, from http://epaper.irishtimes.com/irishtimesdemo/46/

People are still buying and investing in properties in Ireland. According to 2020 traffic on the MyHome website, up to 40%-60% in 2020 compared with 2019 on various metrics.

Although covid-19 has not had a strong impact on the real estate market, it has changed the way people live and where they live.

The website of the Central Statistics Office shows that in Dublin, residential property prices saw an increase of 1.2% in the year to December (2020), while property prices outside Dublin were 3.1% higher (Central Statistics Office, 2021).

During this Covid-19 pandemic, people experienced changes in the way they live in Ireland caused by the lockdown, because of this new trend there is a change related to housing search features, such as balconies, parks, and more manageable living quarters. As working from home becomes more common, some people are relocating from cities to the countryside.

Another interesting point to note is that, even though many businesses in Ireland shut down due to the Covid-19 pandemic, the construction industry remained strong. According to the Central Statistics Office (2020), the construction production value index growing by 15.4% year-on-year (y-o-y), and the construction production volume index growing by 14.9% y-o-y (WIRE, 2020).

The Central Statistics Office (CSO) has reported the highest level of Irish nationals returning home than at any time in the last 13 years, which is likely to be a consequence of Covid-19 and Brexit (Central Statistics Office, 2020).

These numbers reassure the team that our product would be a good use for the property market and a valuable tool for investors.

6. Business objectives with this Property Prediction Model

Homes prices are constantly changing through the years, depending on the economy. According to Eoin What is Moscow Prioritization?”, on the Irish times website, the property investment market has yielded 3 billion last year compared to 6 billion the year before. The decrease has to do with the current COVD pandemic, although the circumstances, the market is still attractive for real estate investors (Burke-Kennedy, 2021).

We aim to use Ireland properties data to predict prices of homes in Dublin areas, for investors who are not familiar with the locality. So, this model will help them to gather information with less research and less time-consuming. In this direction, a foreign investor would have to make use of different sources to gather information needed to decide whether the investment will be worthwhile. As when making an investment, the intention is to make a profit and not lose funds.

This machine learning l will provide such information with interactive graphs that will display the premise prices. With features that will include property prices based on the features of the area, such as schools, garda station, bus and luas stops, and Dart. As a result, this user interface platform would provide foreign investors with insights relating to sales advantages of where to invest in properties in Dublin.

7. Moscow Business Analysis Approach

For this report, we will use the Moscow prioritization method. The acronym MoSCoW stands for the categories: must-haves, should-haves, could-haves, and will not have at this time (Productplan, 2021).

After the group has listed all the candidates for scope such as features, functionalities, benefits, and capabilities, we have decided to prioritize:

MUST	SHOULD	COULD	WON'T
Machine Learning Model	Range of MAP functionalities	Website	Interactive Voice
MAP with predictions		No-SQL database	Properties offer
Data Analysis			

7.1. Must-haves

Our product will be designed for Investors that are interested in the property market in Dublin, Ireland. During our research, we could find valuable information on how effective it would be to have a tool that could predict the value of houses and apartments to help investors to know the market and know how much they will need to invest, or even for the population to plan their future life.

According to The Irish Times "Irish residential property prices… are set to rise by an average of 4 percent in 2021 as supply shortages continue to prop up valuations." (Brennan, 2021).

Based on the information we found during our research, the team has decided that the minimum requirement for this project would be to have the Machine Learning Model, to predict the prices per area.

Also, we must present a map to the user, so it can demonstrate visually the price predictions. The team will make use of data available in that manner, and we must analyse that data carefully, to present the right results.

7.2. Should-haves

We would like to add to the map some functionalities, such as a full address, and directions for the shops, schools, and hospitals, price tags, distance of public transport.

7.3. Could-haves

Because we will have less than 10 weeks to build our interactive map, we are not sure if the team will be able to do it, but our goal is to include our interactive map in a website, to facilitate user interaction. Also, it would include a real-time database, so once the user has the credentials, it would be easy to keep information on what this user is looking for in our platform.

7.4. Will not have

It would be interesting if the map had voice functionalities, so people that have hearing problems could have access to it. Although it could be very useful, as a team we know we will not have able time to do it.

For the property market industry, it would be interesting if the website has a property offer, it makes sense because the users might be looking for a property to buy.

That is not the goal for the team though, we are not selling properties, and because of that, we are not going to include property offers.

8. Technologies

8.1 Programming Language – Python

As our project will be dealing with data science and machine learning, we needed to choose a language that was most ideal when working with data and experimenting with algorithms. In a study made by Piatetsky (2021) the three most popular languages for data science and machine learning between the years 2017 and 2019 were Python, Rapid Miner, and R. As we all have worked with Python and R, we decided to use one of these two languages. We also checked the popularity

between these two to see it clearer which one is the ideal for us to use. According to PYPL (2021), Python is the most searched language on google with 30.5% of the total share while R has 3.82%.

Why is python so popular as a programming language and why is it the most chosen one for data science/machine learning? According to Coelho and Richert python is a high-level programming language that is also quite fast and built in a way that is much easier to experiment with data. It also has a great variety of libraries for data science and machine learning and a big user base which expands all those libraries, toolkits, and materials even more (Coelho and Richert, 2013, pp. 8-9).

All of us also used python during our last year and we feel quite comfortable using it for our final project. In the end, we chose Python as it is fast, easy to experiment with, has a great variety of libraries and a lot of material to work with.

8.2. IDE/Code Editor – VSC and Jupyter Notebook

Choosing which IDE or code editor will be used during our project was a bit more complicated as there are dozens of IDEs and code editors for python, and it proved to be quite difficult to choose the ideal one. The good thing about python is that most (and the best) of them are open source so there are plenty of tools that we can choose from.

We decided to use Jupyter Notebook as it is open-source, and it lets us share our code more easily than other IDE's. Jupyter notebook is not really an IDE but a web application that lets us share live code with each other and format our comments in a more organized and understandable way than with other IDE's. Jupyter Notebook is ideal for working with data and machine learning as it lets us experiment with your code in small pieces (Project Jupyter, 2021).

Jupyter Notebook is great to experiment with code and to share it, but when writing your full script sometimes it is better to use a proper IDE/Code Editor. As we said before there are many open-source options (PyDev, Atom, Spyder, VSC) for those and there is not really a proprietary software that matches them. They are all quite similar in functionality, some of them have great marketplaces, like PyDev with Eclipse, some of them are faster, others have better functionalities. Sublime is an

example of a proprietary code editor, but it does not offer anything new than other open-source alternatives already have. In the end, it all depends on who is using it and their personal preference (Fincher, 2021). Some of us already had a preferred code editor, for that reason we ended up choosing VSC as our code editor option together with Jupyter notebook.

8.3. Machine Learning Model

According to Kirk, machine learning is a way for the machines to figure out the "rules" in data given to them through algorithms, techniques, and tricks of the trade. With machine learning, you can discover hidden patterns, categorize data, and make predictions depending on the model that you use (Kirk, 2017, pp. 15-18).

Machine learning is all around us, it is used to categorize users of a website, create recommendations according to previous behaviours, predict stock market prices, calculate the chances of someone developing cancer, filtering spam emails, translating websites, and many other use cases. These machine learning models utilize data fed to them so they can learn from it and can create a proper output. Not all these problems that were exemplified have the same implementation, so the right tool needs to be chosen for a certain type of work. There are many types of machine learning algorithms that can be used for different tasks so the right one needs to be chosen for our project (Domingos, 2015, pp. xi-xxi).

Supervised learning is the best learning option for our model as we know what we want our output to be. For supervised models to learn they first need to be trained with a set of inputs and expected outputs so they can learn what prediction(output) to make when given an input. This data would come from datasets collected during our data mining phase and fed to our model to create predictions about the house market prices. There are other options, like unsupervised learning which is a knowledge discovery type of learning which is usually used for discovering patterns or new information in data. We have also reinforcement learning and semi-supervised learning which are used for more complex tasks e.g., teaching a computer how to play a video game or image classification (Bonaccorso, 2017, pp. 9-15).

We have various problems inside the machine learning area, therefore the right one needs to be identified so we can utilize the right algorithm for our model. Our

project's main objective is to **predict house market prices** using past data. That fits in the prediction learning area. According to (Gollapudi and Laxmikanth 2016, pp. 16-23) prediction models use past knowledge to generate a forecast of what the result of something would be in the future, in case there is not enough data, and that needs to be done with a technique called regression. Regression models output a continuous quantity, different from classification models which would classify your data into classes, or clustering models which create clusters of data according to their characteristics.

When dealing with accuracy, it is hard to set a starter goal on how accurate the predictions will be and how far our model will be from the real answer. Regression models do not deal with accuracy the same way as classification models. Classification models' answers are pretty much binary, or the classification is right, or it is wrong, so it is easier to set a goal. Now, working with regression can be a little bit more complicated as it deals with how far we are from the right prediction. Our plan is to be as accurate as possible, but we can only have an idea later after building our model (Mishra, 2018).

In conclusion, we will be using a supervised learning regression model for our project, this will be fleshed out later in our modelling phase: which algorithm will be used, why we chose it, and the way it will be implemented.

9. Timeline Plan

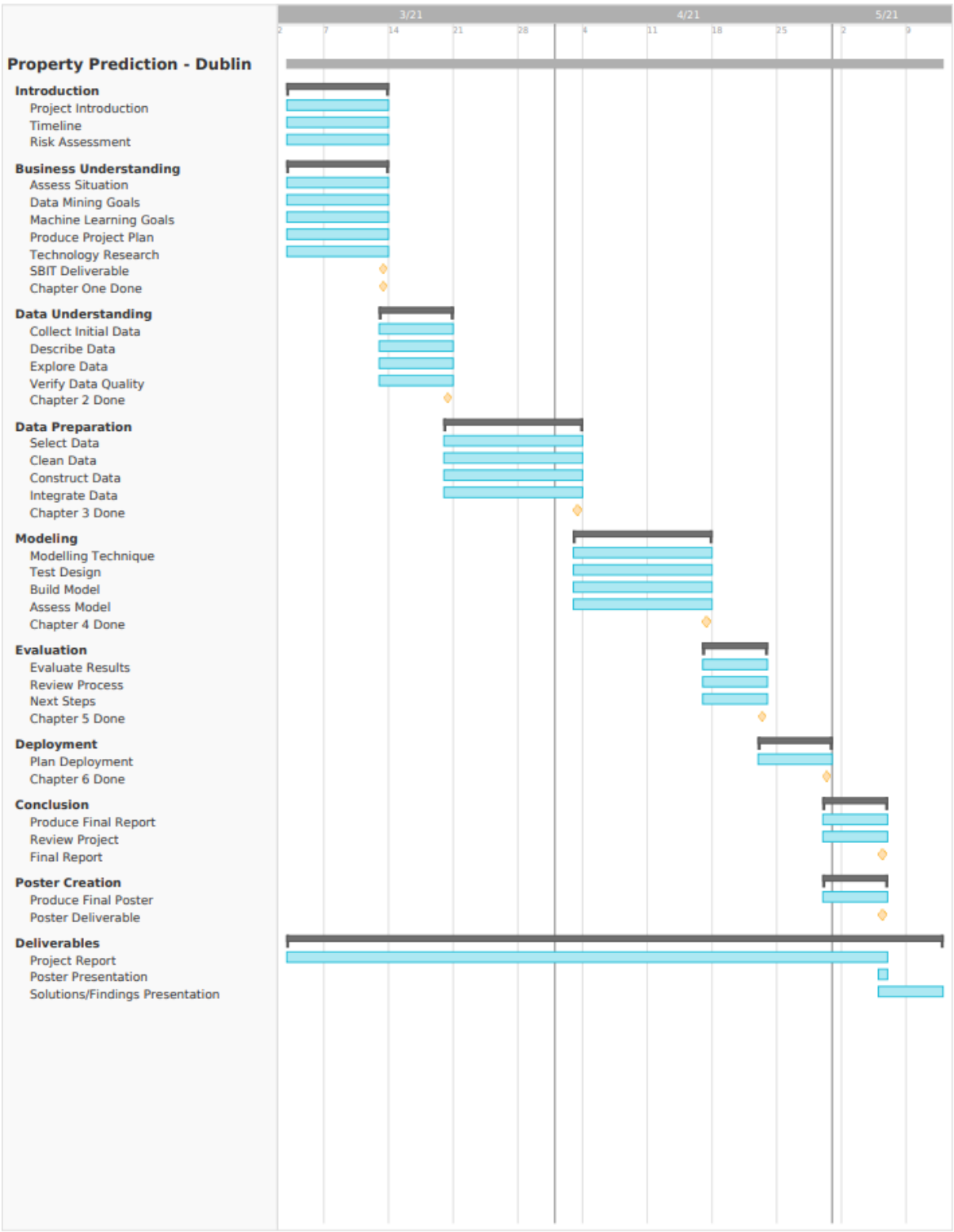

Table 2: TimeLine Plan

9.1. CRISP-DM Approach

This segment discusses how the team will spread the Model's Creations:

Phase I	Business Understanding
Phase 2	Data Understanding
Phase 3	Data Preparation
Phase 4	Modelling
Phase 5	Evaluation
Phase 6	Deployment

10. Risk and Management Plan

10.1 Risk Monitoring and Control

The team understands the limited period we have to finish this project, which is why analysing the chances of things going wrong is so important.
The table below will assist us in keeping track of potential threats as well as steps to minimize possible issues.

The Risk Appraisal can be used in accordance with the project schedule, and it can be reassessed and reclassified at any time (Lavanya and Malarvizhi, 2008).

10.2 Initial Project Risk Assessment

Risk	Risk Level L/M/H	Likelihood of Event	Response Strategy
Project Size			
Person Hours (Phase 1)	**H:** Over.	**Certainty**	Determine the objectives and goals, comprehensive project management approach, and communications plan.

Estimated Project Schedule (Phase 1)	**H:** 12 weeks.	**Certainty**	Created a comprehensive project timeline with frequent baseline reviews.
Team Size at Peak (All Phases)	**H:** Over 5 members.	**Certainty**	Comprehensive communications plan, frequent meetings, tight project management oversight.
Documentation (All Phases)	**H:**	**Certainty**	Develop documentation immediately, engage the whole team.
Project Definition			
Use of datasets (Phase 1, 2)	**H:** Research of usable datasets	**Certainty**	Assess global implications of a specific dataset, research.
Available documentation research establishment of baseline (Phase 1, 2)	**H:**	**Likely**	Balance of information to be gathered by the group.
Creating Modelling (Phase 3)	**M:** Modelling subject to revision.	**likely**	Scope initially defined in the project plan, reviewed weekly by the group to prevent undetected problems.

Consultant Project Deliverables unclear (Phase 4)	**L:** Well, defined.	**Unlikely**	Included in the project plan, subject to amendment.
Change Management Procedures undefined (Phase 4)	**M:** Distributed team makes availability questionable.	**Somewhat likely**	Continuous review of project drive by all levels. Consultant to identify any impacts caused by unavailability. If necessary, increase commitment by participants to full-time status
Quality Management Procedures (Phase 5)	**M:** Team to test, and improve any functionalities needed.	**Likely**	The use of the tool created a comprehensive Communications Plan.
Deployment	**H:** Use of tools.	**Certainty**	End of documentation and use of the Model created.

11. Data Understanding and Preparation

11.1 Select and Construct Data

The team was very careful to gather only high-quality data during this process, and the key source of our databases was governmental websites such as data.smartdublin.ie.

The following datasets are the ones we are going to use during our data process:

11.1.1 primary_schools.csv and secondary_schools.csv

These datasets were collected from the website data.smartdublin.ie, and it has the name of Primary and secondary institutions and location in Dublin. The source for this data is the National School Annual Census for 2019/2020, Dept. Education (Council, 2021).

Let us explore our datasets and see what information it will give us.
The first dataset is the one with the primary schools in the Dublin area.
In the image below we can see that initially this dataset had 452, rows and 20 columns.

```
df.head(5)
```

	Roll Number	Official Name	Address (Line 1)	Address (Line 2)	Address (Line 3)	Address (Line 4)	County Description	Local Authority Description	Phone No.	Principal Name	Email	Eircode	Ga In
0	00697S	ST BRIGIDS MXD N S	Beechpark Lawn	Castleknock	Dublin 15	NaN	Dublin	Fingal County Council	18214040	Denis Courtney	principal@saintbrigids.ie	D15P820	
1	00714P	LUCAN B N S	Chapel Hill	Lucan	Dublin	NaN	Dublin	South Dublin County Council	16281857	Dara Burke	info@stmarysbnslucan.com	K78YD27	
2	00729F	CLOCHAR LORETO N S	Grange Road	Rathfarnham	Dublin 14	NaN	Dublin	South Dublin County Council	14931640	Sr. Maria Hyland	info@loretograngeroad.ie	D14YY28	
3	00752A	CENTRAL SENIOR MXD N S	Marlborough Street	City Centre	Dublin1	NaN	Dublin	Dublin City Council	18788103	Deirdre Gartland	modesenmix@eircom.net	D01P027	
4	01170G	S N NA H-AILLE	Westown	Naul	Dublin	NaN	Dublin	Fingal County Council	18413821	Edel McMahon	naulns@eircom.net	K32PX40	

```
df.shape
```

```
(452, 20)
```

Figure 9. 5 first columns of the initial school dataset

First, we needed to rename some columns so it will be easier to find the latitude and longitude of the addresses.

Rename columns

```
df_new = df.rename({'Address (Line 1)':'address1', 'Address (Line 2)':'address2', 'Address (Line 3)':'address3', 'Address (Li
```

```
df_new.head()
```

7]:

	Roll Number	schoolName	address1	address2	address3	address4	city	Local Authority Description	Phone No.	Principal Name	Email	Eircode	Gae Ind
0	00697S	ST BRIGIDS MXD N S	Beechpark Lawn	Castleknock	Dublin 15	NaN	Dublin	Fingal County Council	18214040	Denis Courtney	principal@saintbrigids.ie	D15P820	
1	00714P	LUCAN B N S	Chapel Hill	Lucan	Dublin	NaN	Dublin	South Dublin County Council	16281857	Dara Burke	info@stmarysbnslucan.com	K78YD27	
2	00729F	CLOCHAR LORETO N S	Grange Road	Rathfarnham	Dublin 14	NaN	Dublin	South Dublin County Council	14931640	Sr. Maria Hyland	info@loretograngeroad.ie	D14YY28	
3	00752A	CENTRAL SENIOR MXD N S	Marlborough Street	City Centre	Dublin1	NaN	Dublin	Dublin City Council	18788103	Deirdre Gartland	modesenmix@eircom.net	D01P027	
4	01170G	S N NA H-AILLE	Westown	Naul	Dublin	NaN	Dublin	Fingal County Council	18413821	Edel McMahon	naulns@eircom.net	K32PX40	

```
df_new.shape
```

8]: (452, 20)

Figure 10. 5 first columns after renaming some columns in the school dataset.

We then install and call the necessary libraries to get the geolocation:

```
#pip install geopy
```

```
from geopy.exc import GeocoderTimedOut
from geopy.geocoders import Photon
from geopy.geocoders import Nominatim
import time
from geopy.extra.rate_limiter import RateLimiter
```

Figure 11. Importing necessary packages to use in Jupyter notebook.

We then Create a locator to hold the Geocoding service, in that case we used Photon:

```
locator = Photon(user_agent="measurements")
```

Figure 12. Geolocator

After that we concatenate address columns into one that is appropriate for geocoding:

```
df_new['full_address'] = df_new.address1 + "," + df_new.address2 + "," + df_new.city
```

```
#geocode
df_new['gcode'] = df_new.full_address.apply(locator.geocode)
```

Figure 13. Concatenating some columns

- We create a 'location' column by applying the geocode we created.
- We create latitude, longitude, and altitude as a single tuple column.
- Finally, we split latitude, longitude, and altitude columns into three separate columns.

```
 #1 - conveneint function to delay between geocoding calls
geocode = RateLimiter(locator.geocode, min_delay_seconds=1)
# 2- - create location column
df_new['location'] = df_new['full_address'].apply(geocode)
# 3 - create longitude, laatitude and altitude from location column (returns tuple)
df_new['point'] = df_new['location'].apply(lambda loc: tuple(loc.point) if loc else None)
# 4 - split point column into latitude, longitude and altitude columns
df_new[['latitude', 'longitude', 'altitude']] = pd.DataFrame(df_new['point'].tolist(), index=df_new.index)
```

Figure 14. Creating latitude, longitude, and altitude columns

Now we have the latitude and longitude necessary to use in our map.

Phone No.	Principal Name	...	Female	Male	Total	full_address	gcode	location	point	latitude	longitude	altitude
214040	Denis Courtney	...	447	502.0	949	Beechpark Lawn,Castleknock,Dublin	(Beechpark Lawn, D15V9KN, Castleknock, Éire / ...	(Beechpark Lawn, D15V9KN, Castleknock, Éire / ...	(53.375855, -6.361625, 0.0)	53.375855	-6.361625	0.0
281857	Dara Burke	...	0	513.0	513	Chapel Hill,Lucan,Dublin	(Chapel Hill, K78 N8X6, Lucan, Éire / Ireland,...	(Chapel Hill, K78 N8X6, Lucan, Éire / Ireland,...	(53.360019, -6.4388757, 0.0)	53.360019	-6.438876	0.0
931640	Sr. Maria Hyland	...	501	0.0	501	Grange Road,Rathfarnham,Dublin	(Grange Road, DUBLIN 14, Rathfarnham, Éire / I...	(Grange Road, DUBLIN 14, Rathfarnham, Éire / I...	(53.2953171, -6.2825391, 0.0)	53.295317	-6.282539	0.0
788103	Deirdre Gartland	...	133	143.0	276	Marlborough Street,City Centre,Dublin	(LRG, Marlborough Street, D01P7K8, Marlborough...	(LRG, Marlborough Street, D01P7K8, Marlborough...	(53.3493141, -6.2576704, 0.0)	53.349314	-6.257670	0.0
413821	Edel McMahon	...	67	72.0	139	Westown,Naul,Dublin	(Westown, Naul, Éire / Ireland, (53.5792861000...	(Westown, Naul, Éire / Ireland, (53.5792861000...	(53.579286100000004, -6.299280855181822, 0.0)	53.579286	-6.299281	0.0

Figure 15. dataset with new columns

We delete the columns that are irrelevant, considering that we do not need the addresses anymore because we now have the latitude and longitude.

```
# delete columns that are not relevant
df_new = df_new.drop(['address1', 'address2', 'address3', 'address4', 'full_address', 'altitude', 'city', 'Local Authority De
```

```
df_new['Type'] = 'Primary School'
```

```
df_new.shape
```

```
(452, 12)
```

Figure 16. Deleting columns that are not relevant, and creating new column Type

We will prepare the secondary schools data set in a similar way, and after getting the geolocation for this second dataset we will prepare the datasets to be merged.

```
df2.head()
```

al y	Eircode	Principal Name	...	FEMALE	MALE	Total	address	gcode	location	point	latitude	longitude	altitude
al y :il	K32R248	MS. ANN M MCDONOUGH	...	1,282	0	1,282	Brick Lane,Balbriggan,Dublin	(Brick Lane, K32 WY80, Balbriggan, Éire / Irel...	(Brick Lane, K32 WY80, Balbriggan, Éire / Irel...	(53.6122056, -6.1865397, 0.0)	53.612206	-6.186540	0.0
al y :il	D13W208	MS. EDEL GREENE	...	274	0	274	Baldoyle,Baldoyle,Dublin	(Baldoyle ED, Fingal, Éire / Ireland, (53.3997...	(Baldoyle ED, Fingal, Éire / Ireland, (53.3997...	(53.399792149999996, -6.138721275936534, 0.0)	53.399792	-6.138721	0.0
n e n	A94FK84	MR. ALAN T MAC GINTY	...	0	1,024	1,024	Blackrock College,Blackrock,Dublin	(Blackrock College, Rock Road, D04 R2C5, Rock ...	(Blackrock College, Rock Road, D04 R2C5, Rock ...	(53.3047146, -6.192405228920508, 0.0)	53.304715	-6.192405	0.0
n e n	A94TW98	MR. ALAN JAMES THOMAS ROGAN	...	0	207	207	Rock Road,Blackrock,Dublin	(Blackrock College, Rock Road, D04 R2C5, Rock ...	(Blackrock College, Rock Road, D04 R2C5, Rock ...	(53.3047146, -6.192405228920508, 0.0)	53.304715	-6.192405	0.0

Figure 17. New Dataset with information about secondary schools

We have created a new column that will describe the type of school, if it is primary or secondary.

```
df_new['Type'] = 'Primary School'
```

```
df2['Type'] = 'Secondary School'
```

We then merge both of data frames together.

```
df_new_df2 = df_new.append(df2)
```

```
df_new_df2
```

	Roll Number	schoolName	Phone No.	Principal Name	Email	Ethos Description	Female	Male	Total	latitude	...	Fee Paying School (Y/N)	Pupil Attendance Type
0	00697S	ST BRIGIDS MXD N S	18214040.0	Denis Courtney	principal@saintbrigids.ie	Catholic	447	502	949	53.375855	...	NaN	NaN
1	00714P	LUCAN B N S	16281857.0	Dara Burke	info@stmarysbnslucan.com	Catholic	0	513	513	53.360019	...	NaN	NaN
2	00729F	CLOCHAR LORETO N S	14931640.0	Sr. Maria Hyland	info@loretograngeroad.ie	Catholic	501	0	501	53.295317	...	NaN	NaN
3	00752A	CENTRAL SENIOR MXD N S	18788103.0	Deirdre Gartland	modesenmix@eircom.net	Catholic	133	143	276	53.349314	...	NaN	NaN
4	01170G	S N NA H-AILLE	18413821.0	Edel McMahon	naulns@eircom.net	Catholic	67	72	139	53.579286	...	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...
178	91339F	Hartstown Community School	NaN	MS. LUCIA GRIFFIN	info@hartstowncs.com	INTER DENOMINATIONAL	535	595	1,130	53.392859	...	N	Day
179	91342R	Pobalscoil Neasáin	NaN	MR. PATRICK MCKENNA	office@psn.ie	INTER DENOMINATIONAL	306	468	774	53.395830	...	N	Day

Figure 18: both datasets merged.

We have now a data frame with 635 rows and 33 columns, we deleted columns that are not relevant for our project and the result is a data frame with 635 rows and 10 columns.

635 rows × 33 columns

Deliting columns that are not relevant for the project

```
# delete columns that are not relevant
df_new_df2 = df_new_df2.drop(['address1', 'address2', 'address3', 'address4', 'address', 'altitude', 'County', 'Eircode', 'DE
```

```
df_new_df2.head()
```

5]:

	Roll Number	schoolName	Principal Name	Ethos Description	Female	Male	Total	latitude	longitude	Type
0	00697S	ST BRIGIDS MXD N S	Denis Courtney	Catholic	447	502	949	53.375855	-6.361625	Primary School
1	00714P	LUCAN B N S	Dara Burke	Catholic	0	513	513	53.360019	-6.438876	Primary School
2	00729F	CLOCHAR LORETO N S	Sr. Maria Hyland	Catholic	501	0	501	53.295317	-6.282539	Primary School
3	00752A	CENTRAL SENIOR MXD N S	Deirdre Gartland	Catholic	133	143	276	53.349314	-6.257670	Primary School
4	01170G	S N NA H-AILLE	Edel McMahon	Catholic	67	72	139	53.579286	-6.299281	Primary School

```
df_new_df2 = df_new_df2.fillna(1)
#df_new_df2 = df_new_df2.dropna()
```

```
df_new_df2.shape
```

7]: (635, 10)

Figure 19: dataset after deleting unnecessary columns.

We then create a map that show both of our data frames in a map and save it in a html format.

```
import folium
```

```
# create the map.
map_main = folium.Map( location=[53.5000, -6.20000])

# adding the latitude and longitude points to the map.
df_new_df2.apply(lambda row:folium.CircleMarker(location=[row["latitude"], row["longitude"]] ).add_to(map_main), axis=1)

# display the map: just ask for the object representation in juypter notebook.
map_main

# optional: save the map.
map_main.save('map_main.html')
```

Figure 20: Folium Code to produce a map.

And that is the result:

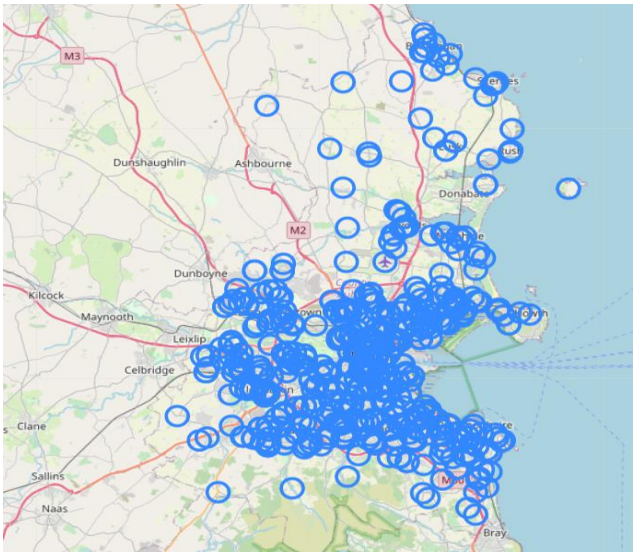

Figure 21: Map of the location of all the schools in Dublin.

11.1.2 blt_df.csv (Transportation).

The next dataset we will prepare is the one that gives us the necessary information for transportation in Dublin. All the data collected came from the Transport for Ireland (2021) website. There are many ways of public transportation in Dublin: buses, trains, luas, coaches and bikes for example. We will focus only on the long-distance ones that are used by almost everyone which are the buses, trains, and Luas.

As all these services run separately, there is no general dataset for all the stops. We found 3 different datasets which contained all the stops needed and that were last updated at the end of 2019. These are:

- Bus Stops: NaPTAN_2019_12_23.csv

- Luas Stops: Luas.csv

- Irish Rail: irish_rail_stops.csv

The Luas and Irish Rail datasets were txt files that we had to first convert to csv utilizing Excel. After that was done, we could then work with all the files, first through Data Exploration. In the image below we can see that both the Irish Rail and the Luas dataset are very similar and have similar sizes. They both have 4 features with the same names and types of data. This is great as they are similar, it is easier to merge them together and work with them.

```
luas_df = pd.read_csv('Data/Luas.csv')
```

```
luas_df
```

	stop_id	stop_name	stop_lat	stop_lon
0	822GA00058	St.Stephen's Green	53.339129	-6.261117
1	822GA00062	Harcourt	53.333651	-6.262692
2	822GA00070	Charlemont	53.330615	-6.258536
3	822GA00071	Charlemont	53.330834	-6.258737
4	822GA00074	Ranelagh	53.326140	-6.256121
...	...	...	...	...
115	gen:57102:3571:1	O'Connell Upper	53.351612	-6.261030
116	gen:57102:3573:1	Parnell	53.353105	-6.260503
117	gen:57102:3577:1	Dominick	53.351345	-6.265547
118	gen:57102:3584:1	Broadstone - DIT	53.354071	-6.273759
119	gen:57102:3587:1	Cabra	53.364337	-6.281969

120 rows × 4 columns

Figure 22: Luas stop dataset.

```
train_df = pd.read_csv('Data/irish_rail_stops.csv')
```

```
train_df
```

	stop_id	stop_name	stop_lat	stop_lon
0	700G002158	Lisburn Train Station	54.513918	-6.045427
1	700G002159	Lurgan Train Station	54.467303	-6.337867
2	700G002160	Portadown Train Station	54.424267	-6.446564
3	700G002161	Newry Train Station	54.188723	-6.362550
4	821GIR0001	Bagenalstown	52.699135	-6.952594
...	...	...	...	...
144	850GIR0091	Castlerea	53.761530	-8.485715
145	851GIR0092	Sligo	54.272022	-8.481918
146	851GIR0093	Ballymote	54.088191	-8.520779
147	851GIR0094	Collooney	54.186516	-8.494881
148	g:31400:11	Belfast City Centre, Lanyon Place	54.595530	-5.917331

149 rows × 4 columns

Figure 23: Train station dataset.

Working with the bus stops proved to be a little bit more complicated, as the dataset had a lot more features and they are not as consistent as the first two datasets. There are a lot of bus services in Dublin, and this Dataset encompasses ALL bus stops in Dublin. As you can see below, there is a lot of missing data and the triple of columns. It is also a quite large dataset as it has around 20000 rows.

```
bus_df = pd.read_csv('Data/NaPTAN_2019_12_23.csv')
```

```
bus_df
```

	AtcoCode	PlateCode	SCN_English	SCN_Gaeilge	NptgLocalityRef	BusStopType	CompassPoint	Latitude	Longitude	Easting	Northing	Modific
0	7000B6310001	631001.0	Belcoo	Béal Cú	NaN	MKD	NE	54.296217	-7.871793	608346.0	838613.0	2016-
1	700000004183	108061.0	Banbridge	Droichead na Banna	NaN	MKD	SW	54.346455	-6.271543	712376.0	845574.0	2019-
2	700000004149	108061.0	Banbridge	Droichead na Banna	NaN	MKD	NE	54.346297	-6.271180	712400.0	845557.0	2018-
3	7000B141561	141561.0	Derrylin	Doire Loinn	NaN	CUS	NaN	54.193380	-7.567849	628202.0	827247.0	2019-
4	7000B140871	140871.0	Derrylin	Doire Loinn	NaN	NaN	NaN	54.193317	-7.567941	628196.0	827240.0	2019-
...	...	...	...	...	...	...	...	...	...	...	...	
19970	854000001	NaN	NaN	NaN	E0854007	NaN	NaN	54.157089	-7.138513	656269.0	823465.0	2019-
19971	854000002	NaN	NaN	NaN	E0854029	NaN	NaN	54.216834	-7.042790	662431.0	830194.0	2019-
19972	854000011	NaN	NaN	NaN	E0854045	NaN	NaN	54.125713	-6.900559	671865.0	820189.0	2019-
19973	8540LL10198	NaN	Scotshouse	Teach an Scotaigh	E0854040	NaN	NaN	54.122226	-7.248889	649101.0	819503.0	2019-
19974	8540LL10199	NaN	Scotshouse	Teach an Scotaigh	E0854040	NaN	NaN	54.122220	-7.249318	649073.0	819502.0	2019-

19975 rows × 12 columns

Figure 24: Bus stop dataset.

To be able to merge our datasets, we need to first make sure that they have similar features. As we only need to know where the stops are located, we do not need names, codes or addresses. We only need an ID, Latitude and Longitude. We also need to know what they are, if that stop is a bus stop, a Luas stop or a train stop, so a new column with the type of point needs to be added to each dataset before merging.

Below you can see the columns that are not needed being dropped, renaming the ‘AtcoCode’ columns to Stop_Id and creating a new column named Type and setting all of them to ‘Bus Stop’.

```
bus_df = bus_df.drop(columns=['PlateCode','SCN_English','SCN_Gaeilge','NptgLocalityRef','BusStopType','CompassPoint','Easting','N
```

```
bus_df = bus_df.rename({'AtcoCode':'Stop_Id'},axis = 'columns')
bus_df['Type'] = 'Bus Stop'
```

```
bus_df
```

	Stop_Id	Latitude	Longitude	Type
0	7000B6310001	54.296217	-7.871793	Bus Stop
1	700000004183	54.346455	-6.271543	Bus Stop
2	700000004149	54.346297	-6.271180	Bus Stop
3	7000B141561	54.193380	-7.567849	Bus Stop
4	7000B140871	54.193317	-7.567941	Bus Stop
...	...	...	...	...
19970	854000001	54.157089	-7.138513	Bus Stop
19971	854000002	54.216834	-7.042790	Bus Stop
19972	854000011	54.125713	-6.900559	Bus Stop
19973	8540LL10198	54.122226	-7.248889	Bus Stop
19974	8540LL10199	54.122220	-7.249318	Bus Stop

19975 rows × 4 columns

Figure 25: Dataset after renaming columns and create Type column.

The same was done with both Luas as Irish Rail data frames, even though only the column of the stop name needed to be dropped and the column names were changed. After doing that we just needed to append them as shown below. As we append them, the indexes get all messed up so we need to reset them and drop the old column.

```
blt_df = bus_df.append(luas_df)
```

```
blt_df = blt_df.append(train_df)
```

```
blt_df = blt_df.reset_index()
```

```
blt_df = blt_df.drop(columns='index')
```

```
blt_df
```

	Stop_Id	Latitude	Longitude	Type
0	7000B6310001	54.296217	-7.871793	Bus Stop
1	700000004183	54.346455	-6.271543	Bus Stop
2	700000004149	54.346297	-6.271180	Bus Stop
3	7000B141561	54.193380	-7.567849	Bus Stop
4	7000B140871	54.193317	-7.567941	Bus Stop
...	...	...	...	...
20239	850GIR0091	53.761530	-8.485715	Train Stop
20240	851GIR0092	54.272022	-8.481918	Train Stop
20241	851GIR0093	54.088191	-8.520779	Train Stop
20242	851GIR0094	54.186516	-8.494881	Train Stop
20243	g:31400:11	54.595530	-5.917331	Train Stop

20244 rows × 4 columns

Figure 26: Merging datasets together.

We then checked if there were any empty values in the Data frame, as there were very little (only 4) we chose to drop the rows. Then it could be saved in a Dataset, so it could be used with the other datasets.

```
blt_df.isnull().sum()
```

```
Stop_Id      0
Latitude     2
Longitude    2
Type         0
dtype: int64
```

```
blt_df = blt_df.dropna()
```

```
blt_df.to_csv ('Data\BLT_stops.csv', index = False, header=True)
```

Figure 27: Saving dataset after data cleaning.

The first 1000 values were put on a map, just like the Schools datasets, to see if they are supposed to be representing Dublin locations, only a 1000 were added as it takes a long time and it is quite a heavy process to do more than that. You can see below that they are correct, so we checked the whole dataset in a scatterplot to see if there are any outliers. We found that there was at least 1, but we decided to leave them for now and sort it out when we merged all of them together. The bus stops and Irish rail encompass all of Ireland, but there was no way for us to separate what was only in Dublin, as there are no areas in the stops, and not all of them were filled up, so we had to use all of it for this part.

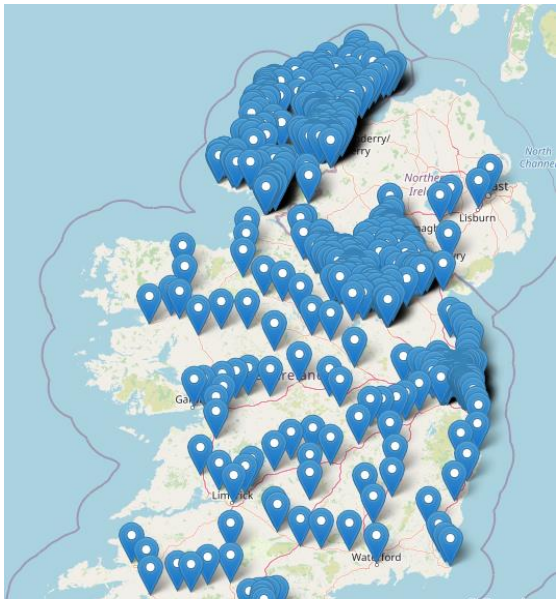

Figure 28: Map of the location of all the bus stops transportation in Dublin/ The first 1000 values were put on a map.

```
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize = (15,8))
sns.scatterplot(scatter_test["Latitude"], scatter_test["Longitude"])
```

```
: <matplotlib.axes._subplots.AxesSubplot at 0x1a81630fb20>
```

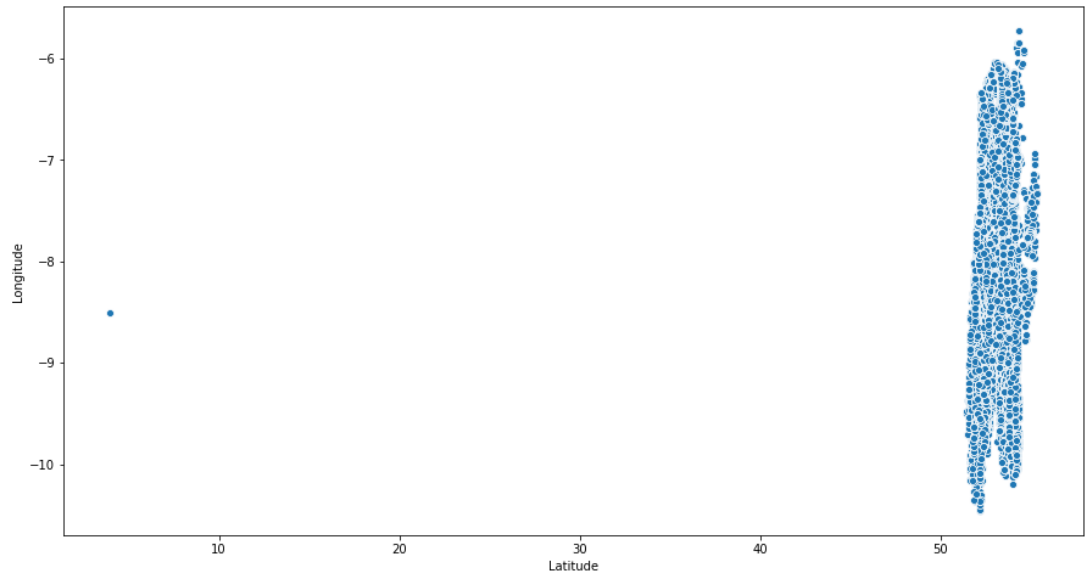

Figure29: Visualization of all the transportation stops in Dublin/ The first 1000 values were put on a map.

11.1.3 garda.csv

The first idea of the group regarding adding crime rates to the dataset was to find a data frame with strong information such as, the number of crimes that happened in a specific location and the respective Garda stations. When looking for data, in websites like kaggle.com, CSO.com, data.gov.ie and more, our team could not find a dataset with the exact information the project demanded. Because of that we decided to go forward with a dataset called IRELAND_CRIME_GARDA_DIVISION_wise_2003-2019 (Crime in Ireland | Kaggle. 2021). This dataset was gathered from the website www.kaggle.com and provided comprehensive data on all forms of crimes committed in Ireland from 2003 to 2019.

The data is provided by the Garda Division (Ireland's administrative division) and is updated quarterly. The StatBank, Central Statistics Office (CSO), government of Ireland's website https://statbank.cso.ie is the original source of this dataset. There is some necessary information about this data which should be mentioned as told by Statbank, the crime data is under 'Reservation' and according to Statbank does not meet the prerequisites of the CSO benchmarks. The number of Garda Divisions has been reduced from 28 to 19. The most recent decrease in the number of Garda Divisions is not reflected in this dataset (Kaggle, 2021).

The original dataset had 5 rows and 72 columns as described in the picture below:

```
In [7]: 1 garda_stations.head(5)
```

Out[7]:

	REGION	GARDA DIVISION	OFFENCE CODE	OFFENCE	TYPE OF OFFENCE	2003Q1	2003Q2	2003Q3	2003Q4	2004Q1	...	2017Q2	2017Q3	2017Q4	2018Q1	
0	NORTHERN REGION	CAVAN/MONAGHAN	111	Murder	HOMICIDE OFFENCES	0	0	0	0	0	...	0	0	1	0	
1	NORTHERN REGION	CAVAN/MONAGHAN	112	Manslaughter	HOMICIDE OFFENCES	0	0	0	0	0	...	0	0	0	1	
2	NORTHERN REGION	CAVAN/MONAGHAN	113	Infanticide	HOMICIDE OFFENCES	0	0	0	0	0	...	0	0	0	0	
3	NORTHERN REGION	CAVAN/MONAGHAN	12	Dangerous driving leading to death	HOMICIDE OFFENCES	1	0	1	0	0	...	1	1	1	0	
4	NORTHERN REGION	CAVAN/MONAGHAN	21	Rape and sexual assault	SEXUAL OFFENCES	24	15	5	5	14	...	16	8	14	23	

5 rows × 72 columns

Figure30: Garda Station dataset.

However, it contained much more information than we thought we would need so we performed some feature engineer on it, changing columns names, filtering the garda stations located only in Dublin, calculating the mean of all crime columns to create a unique crime_mean column to point out the area with the highest crime value. The results of this data preparation can be seen in the coming picture.

```
In [82]: 1 dublin_crime_mean.set_index("REGION")
```

Out[82]:

REGION	GARDA DIVISION	crimes_mean
DUBLIN METROPOLITAN REGION	D.M.R. SOUTH CENTRAL	0.181818
DUBLIN METROPOLITAN REGION	D.M.R. SOUTH CENTRAL	0.000000
DUBLIN METROPOLITAN REGION	D.M.R. SOUTH CENTRAL	0.000000
DUBLIN METROPOLITAN REGION	D.M.R. SOUTH CENTRAL	0.181818
DUBLIN METROPOLITAN REGION	D.M.R. SOUTH CENTRAL	35.090909
...	...	...
DUBLIN METROPOLITAN REGION	D.M.R. WESTERN	6.909091
DUBLIN METROPOLITAN REGION	D.M.R. WESTERN	2.272727
DUBLIN METROPOLITAN REGION	D.M.R. WESTERN	0.272727
DUBLIN METROPOLITAN REGION	D.M.R. WESTERN	0.545455
DUBLIN METROPOLITAN REGION	D.M.R. WESTERN	363.909091

348 rows × 2 columns

Figure 310: Result of the garda station dataset preparation.

A dataset renamed as dublin_crime_mean with 348 rows and 2 features. However, after analysing this result the group evaluated that we would need features such as latitude and longitude to be able to link this dataset with the others. The information given in the picture above is about the region where the Garda station is located and its division, which is not enough for the process of extracting latitude and longitude values through the API used in this project (Google Developers. 2021). For these reasons, this dataset was disqualified to be used in this project and a new searching process started.

The new dataset was found in the census website and its name is Recorded Crime Offences Under Reservation (Central Statistics Office, 2021). This new dataset contains slightly more relevant information than the previous one, since it holds an identifier number and the name of the region of each Garda station in Dublin. By the time this dataset was found, the group had already revised the data that our model would need to make the predictions and the consensus was that the only relevant feature was the Garda Station feature, which can be seen in the picture below. The

idea now has slightly converse to that by having the Garda Stations location, the prediction model will relate them to the property and perform more precise predictions without the need of the number of crimes that happened. In fact, there would not be a formula to relate the information about the number of crimes to the other columns of the tailor dataset we prepared. There are different types of crimes and it would not be precise to claim that one area is more dangerous than another only based on the number of crimes there occurred and not based on the classification of each crime.

```
In [64]: 1 import pandas as pd
```

reading from the file

```
In [88]: 1 crime = pd.read_csv('crimes.csv', encoding='latin-1')
```

describing data

```
In [92]: 1 crime
```

Out[92]:

	st	Year	Garda Station	Type of Offence	UNIT	VALUE
0	Recorded Crime Offences Under Reservation	2017	63101 Balbriggan, D.M.R. Northern Division	Attempts/threats to murder, assaults, harassme...	Number	118
1	Recorded Crime Offences Under Reservation	2017	63101 Balbriggan, D.M.R. Northern Division	Dangerous or negligent acts	Number	51
2	Recorded Crime Offences Under Reservation	2017	63101 Balbriggan, D.M.R. Northern Division	Kidnapping and related offences	Number	3
3	Recorded Crime Offences Under Reservation	2017	63101 Balbriggan, D.M.R. Northern Division	Robbery, extortion and hijacking offences	Number	18
4	Recorded Crime Offences Under Reservation	2017	63101 Balbriggan, D.M.R. Northern Division	Burglary and related offences	Number	141
...	...	...	...	...	...	...
1471	Recorded Crime Offences Under Reservation	2019	64302 Terenure, D.M.R. Southern Division	Controlled drug offences	Number	103
1472	Recorded Crime Offences Under Reservation	2019	64302 Terenure, D.M.R. Southern Division	Weapons and Explosives Offences	Number	6
1473	Recorded Crime Offences Under Reservation	2019	64302 Terenure, D.M.R. Southern Division	Damage to property and to the environment	Number	96
1474	Recorded Crime Offences Under Reservation	2019	64302 Terenure, D.M.R. Southern Division	Public order and other social code offences	Number	77
1475	Recorded Crime Offences Under Reservation	2019	64302 Terenure, D.M.R. Southern Division	Offences against government, justice procedure...	Number	11

1476 rows × 6 columns

Figure 32: Garda station dataset.

The only feature engineering done in this data frame was to drop all the columns, but the Garda Stations Column, since we processed that it was the only relevant feature to be used. The picture below shows the code used to get to this result and the data described.

Dropping the non relevant columns which happens to be all of them, since we decided to use only the address of the garda stations.

```
In [90]: 1 garda_st = crime.drop(['st', 'Year', 'Type of Offence', 'UNIT', 'Year','VALUE'], axis=1)
```

```
In [91]: 1 garda_st
```

Out[91]:

	Garda Station
0	63101 Balbriggan, D.M.R. Northern Division
1	63101 Balbriggan, D.M.R. Northern Division
2	63101 Balbriggan, D.M.R. Northern Division
3	63101 Balbriggan, D.M.R. Northern Division
4	63101 Balbriggan, D.M.R. Northern Division
...	...
1471	64302 Terenure, D.M.R. Southern Division
1472	64302 Terenure, D.M.R. Southern Division
1473	64302 Terenure, D.M.R. Southern Division
1474	64302 Terenure, D.M.R. Southern Division
1475	64302 Terenure, D.M.R. Southern Division

1476 rows × 1 columns

Figure33: Garda station dataset after dropping unnecessary columns.

Now, the point of having this garda dataset is to check whether there is a garda station around 1km from the house or not, so we need to have the latitude and longitude for each garda station in Dublin. This dataset contains the names of the garda stations. Let us have a look at the shape of the dataset:

```
In [3]: df.shape
Out[3]: (1476, 1)
```

We can see that there are 1476 rows, but there will be a lot of duplicates since these datasets contained a number of different types of crimes for each garda station and the quantity each crime was registered. So, we need to keep only the unique rows:

```
In [5]: df['Garda Station'].unique()
Out[5]: array(['63101 Balbriggan, D.M.R. Northern Division',
               '66201 Ballyfermot, D.M.R. Western Division',
               '63201 Ballymun, D.M.R. Northern Division',
               '65101 Blackrock, Co Dublin, D.M.R. Eastern Division',
               '66101 Blanchardstown, D.M.R. Western Division',
               '62101 Bridewell Dublin, D.M.R. North Central Division',
               '65201 Cabinteely, D.M.R. Eastern Division',
               '66102 Cabra, D.M.R. Western Division',
               '66202 Clondalkin, D.M.R. Western Division',
               '63401 Clontarf, D.M.R. Northern Division',
               '63301 Coolock, D.M.R. Northern Division',
               '64101 Crumlin, D.M.R. Southern Division',
               '61101 Donnybrook, D.M.R. South Central Division',
               '63202 Dublin Airport, D.M.R. Northern Division',
               '65203 Dun Laoghaire, D.M.R. Eastern Division',
               '65102 Dundrum, D.M.R. Eastern Division',
               '66103 Finglas, D.M.R. Western Division',
               '62202 Fitzgibbon Street, D.M.R. North Central Division',
               '63102 Garristown, D.M.R. Northern Division',
               '63402 Howth, D.M.R. Northern Division',
               '61102 Irishtown, D.M.R. South Central Division',
               '61301 Kevin Street, D.M.R. South Central Division',
               '61302 Kilmainham, D.M.R. South Central Division',
               '66301 Lucan, D.M.R. Western Division',
               '63103 Lusk, D.M.R. Northern Division',
               '63302 Malahide, D.M.R. Northern Division',
               '62203 Mountjoy, D.M.R. North Central Division',
               '61202 Pearse Street, D.M.R. South Central Division',
               '63403 Raheny, D.M.R. Northern Division',
               '66203 Rathcoole, D.M.R. Western Division',
               '64201 Rathfarnham, D.M.R. Southern Division',
               '64301 Rathmines, D.M.R. Southern Division',
               '66302 Ronanstown, D.M.R. Western Division',
               '63203 Santry, D.M.R. Northern Division',
               '65205 Shankill, D.M.R. Eastern Division',
               '63105 Skerries, D.M.R. Northern Division',
               '62301 Store Street, D.M.R. North Central Division',
               '64102 Sundrive Road, D.M.R. Southern Division',
               '63303 Swords, D.M.R. Northern Division',
               '64202 Tallaght, D.M.R. Southern Division',
               '64302 Terenure, D.M.R. Southern Division'], dtype=object)

In [6]: len(df['Garda Station'].unique())
Out[6]: 41
```

Figure 34: Garda Station unique values dataset.

Here we can see that we have the name of 41 Garda Stations in D.M.R which stands for Dublin Metropolitan region. These 41 names are everything we need to keep, so we will drop all duplicates and reset the index:

```
In [7]: df = df.drop_duplicates()

In [9]: df = df.reset_index(drop=True)
```

Now we need to get the latitude and longitude of these Garda Stations. But the Google Maps API is not able to find the addresses in this way that the Irish Census formats the name of the station. For example, if we search Google Maps for:

- 63101 Balbriggan, D.M.R. Northern Division

It will not be able to find it. But if instead we search for:

- Balbriggan Garda Station

Then It can find it! So, let's format our entries to these new formats.

```
In [15]: for x in range(len(df)):
             original = df['Garda Station'][x] #select original name
             without_numbers = original.lstrip('0123456789 ') #delete numbers and space from the front
             without_end = without_numbers.split(',')[0] #delete everything after the comma
             final = without_end + ' Garda Station' #at this point string is smt like for instance: 'Balbriggan Garda Station'
             df['Garda Station'][x] = final #update entry 'x' with new name
```

This code is looping through every row of the dataset. For each iteration, select the garda station name. Ex: On the first iteration, original will be = "63101 Balbriggan, D.M.R. Northern Division". Then delete the numbers and space from the front. It will become: "Balbriggan, D.M.R. Northern Division".

Delete everything after the comma. String will be just: "Balbriggan". Add " Garda Station" to the end to have the final name, it will be: "Balbriggan Garda Station". Update data frame to new garda name that will be easier for the API to find latitude and longitude.

Result data frame:

```
In [16]: df
```

Out[16]:

	Garda Station
0	Balbriggan Garda Station
1	Ballyfermot Garda Station
2	Ballymun Garda Station
3	Blackrock Garda Station
4	Blanchardstown Garda Station
5	Bridewell Dublin Garda Station
6	Cabinteely Garda Station
7	Cabra Garda Station
8	Clondalkin Garda Station
9	Clontarf Garda Station
10	Coolock Garda Station
11	Crumlin Garda Station
12	Donnybrook Garda Station
13	Dublin Airport Garda Station
14	Dun Laoghaire Garda Station
15	Dundrum Garda Station
16	Finglas Garda Station
17	Fitzgibbon Street Garda Station
18	Garristown Garda Station
19	Howth Garda Station
20	Irishtown Garda Station
21	Kevin Street Garda Station
22	Kilmainham Garda Station
23	Lucan Garda Station
24	Lusk Garda Station
25	Malahide Garda Station
26	Mountjoy Garda Station
27	Pearse Street Garda Station
28	Raheny Garda Station
29	Rathcoole Garda Station
30	Rathfarnham Garda Station
31	Rathmines Garda Station
32	Ronanstown Garda Station
33	Santry Garda Station
34	Shankill Garda Station
35	Skerries Garda Station
36	Store Street Garda Station
37	Sundrive Road Garda Station
38	Swords Garda Station
39	Tallaght Garda Station
40	Terenure Garda Station

Figure 35: Data frame with the unique name of the garda stations.

Now we need to pass each of these strings, like “Balbriggan Garda Station” for the example of the first row, to the Google Maps API and retrieve the Latitude and Longitude for those addresses.

Importing the libraries.

```
In [17]: from googlemaps import Client as GoogleMaps
         from time import sleep
```

Creating two new columns to hold latitude and longitude.

```
In [18]: df['Longitude'] = ""
         df['Latitude'] = ""

In [19]: df.head(1)
```

Out[19]:

	Garda Station	Longitude	Latitude
0	Balbriggan Garda Station		

Figure 36: Creating new columns to hold latitude and longitude.

Adding the API Key, and then making the request for every row in the dataset, and storing the latitude and longitude in their respective column in the data frame.

```
In [20]: gmaps = GoogleMaps('API_KEY_WAS_HERE')

In [21]: for x in range(len(df)):
             try:
                 sleep(1) #to add delay in case of large DFs
                 geocode_result = gmaps.geocode(df['Garda Station'][x])
                 df['Latitude'][x] = geocode_result[0]['geometry']['location'] ['lat']
                 df['Longitude'][x] = geocode_result[0]['geometry']['location']['lng']
             except IndexError:
                 print("Address was wrong...")
             except Exception as e:
                 print("Unexpected error occurred.", e )
```

Figure 3711: Code with the API Key to find the latitude and longitude.

Final dataset is the name of each garda station with latitude and longitude:

```
In [22]: df
```

Out[22]:

	Garda Station	Longitude	Latitude
0	Balbriggan Garda Station	-6.19098	53.614378
1	Ballyfermot Garda Station	-6.358053	53.344754
2	Ballymun Garda Station	-6.263922	53.394374
3	Blackrock Garda Station	-6.177503	53.299793
4	Blanchardstown Garda Station	-6.380952	53.389924
5	Bridewell Dublin Garda Station	-6.274125	53.347081
6	Cabinteely Garda Station	-6.150646	53.26079
7	Cabra Garda Station	-6.307702	53.365012
8	Clondalkin Garda Station	-6.394952	53.323146
9	Clontarf Garda Station	-6.22021	53.363411
10	Coolock Garda Station	-6.201027	53.390315
11	Crumlin Garda Station	-6.315002	53.31958
12	Donnybrook Garda Station	-6.235774	53.32172
13	Dublin Airport Garda Station	-6.244229	53.429669
14	Dun Laoghaire Garda Station	-6.133399	53.289923
15	Dundrum Garda Station	-6.242477	53.289491
16	Finglas Garda Station	-6.306434	53.389656
17	Fitzgibbon Street Garda Station	-6.255665	53.357832
18	Garristown Garda Station	-6.383217	53.567335
19	Howth Garda Station	-6.06862	53.387424
20	Irishtown Garda Station	-6.223035	53.338111
21	Kevin Street Garda Station	-6.269445	53.338328
22	Kilmainham Garda Station	-6.304575	53.341948
23	Lucan Garda Station	-6.450605	53.356019
24	Lusk Garda Station	-6.167394	53.523377
25	Malahide Garda Station	-6.152084	53.451088
26	Mountjoy Garda Station	-6.266834	53.36053
27	Pearse Street Garda Station	-6.256193	53.345689
28	Raheny Garda Station	-6.178056	53.378896
29	Rathcoole Garda Station	-6.464697	53.281814
30	Rathfarnham Garda Station	-6.288549	53.29741
31	Rathmines Garda Station	-6.266966	53.321368
32	Ronanstown Garda Station	-6.406166	53.337878
33	Santry Garda Station	-6.250638	53.389967
34	Shankill Garda Station	-6.120737	53.233488
35	Skerries Garda Station	-6.106979	53.579586
36	Store Street Garda Station	-6.251706	53.350485
37	Sundrive Road Garda Station	-6.298697	53.3302
38	Swords Garda Station	-6.221112	53.456024
39	Tallaght Garda Station	-6.367589	53.286807
40	Terenure Garda Station	-6.28781	53.30999

Figure38: data frame with the latitude and Longitude of the garda Stations.

11.1.4 Properties Dataset

These datasets were collected from the website data.smartdublin.ie, and it contains date of sale, price and address of all residential properties purchased in Dublin since the 1st of January 2010, as declared to the Revenue Commissioners for stamp duty purposes. The source for this data is the Property Price Register from the Property Services Regulatory Authority. We used the dataset from smart Dublin instead of downloading directly from the PRSA website just because it was already separated by year and it was Dublin only, and not the whole country.

We decided to use only the years 2017, 2018 and 2019, because since the data we got for transport and schools were from around that time, one of the Luas lines was finished in 2017, so it would not be coherent to use older data for houses. Let us look at the year 2019:

```
In [1]: import pandas as pd
In [2]: df = pd.read_csv('ppr-2019-dublin.csv')
In [3]: df.head()
```

Out[3]:

	Date of Sale (dd/mm/yyyy)	Address	Postal Code	County	Price (€)	Not Full Market Price	VAT Exclusive	Description of Property	Property Size Description
0	01/01/2019	2 LEIX RD, CABRA, DUBLIN 7	Dublin 7	Dublin	€30,000.00	Yes	No	Second-Hand Dwelling house /Apartment	NaN
1	01/01/2019	2 WOODALE VIEW, BALLYCULLEN, FIRHOUSE DUBLIN 24	Dublin 24	Dublin	€30,000.00	Yes	No	Second-Hand Dwelling house /Apartment	NaN
2	01/01/2019	APT 50, EARLSFIELD COURT, FRANCIS ST DUBLIN 8	Dublin 8	Dublin	€15,000.00	Yes	No	Second-Hand Dwelling house /Apartment	NaN
3	02/01/2019	3 Somerton Copse, Lucan	NaN	Dublin	€290,750.00	No	Yes	New Dwelling house /Apartment	NaN
4	02/01/2019	31 ALL SAINTS PARK, RAHENY, DUBLIN 5	Dublin 5	Dublin	€460,000.00	No	No	Second-Hand Dwelling house /Apartment	NaN

```
In [4]: df.shape
Out[4]: (9880, 9)
In [5]: df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9880 entries, 0 to 9879
Data columns (total 9 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   Date of Sale (dd/mm/yyyy)   9880 non-null   object
 1   Address                     9880 non-null   object
 2   Postal Code                 6538 non-null   object
 3   County                      9880 non-null   object
 4   Price (€)                   9880 non-null   object
 5   Not Full Market Price       9880 non-null   object
 6   VAT Exclusive               9880 non-null   object
 7   Description of Property     9880 non-null   object
 8   Property Size Description   31 non-null     object
dtypes: object(9)
memory usage: 694.8+ KB
```

Figure 39: Property 2019 dataset.

The dataset has 9880 rows and 9 columns. Postal Code and Property Size Description have a lot of missing values unfortunately, and it contains addresses but not longitude and latitude. We will need to get that to plot them on a map, so we used the Google Maps API to obtain this information, by passing the Address column, have a look at the code for this process:

First we added two empty columns to the df to hold the longitude and latitude data.

```
In [15]: df['long'] = ""
         df['lat'] = ""
```

Imported GoogleMaps, created an object passing the API Key and then made a request passing the the address of the first row to test if it works

```
In [16]: from googlemaps import Client as GoogleMaps
In [21]: gmaps = GoogleMaps('API_KEY_WAS_HERE')
In [22]: geocode_result = gmaps.geocode(df['Address'][0])
```

Figure 40: Using Google API to find the latitude and Longitude.

An example of the object that the api returns. Here we can see the first row in the dataset returned 2 objects:

```
In [23]: geocode_result

Out[23]: [{'address_components': [{'long_name': '2',
                                   'short_name': '2',
                                   'types': ['street_number']},
                                  {'long_name': 'Leix Road', 'short_name': 'Leix Rd', 'types': ['route']},
                                  {'long_name': 'Cabra East',
                                   'short_name': 'Cabra East',
                                   'types': ['neighborhood', 'political']},
                                  {'long_name': 'Dublin 7',
                                   'short_name': 'Dublin 7',
                                   'types': ['postal_town']},
                                  {'long_name': 'County Dublin',
                                   'short_name': 'D',
                                   'types': ['administrative_area_level_1', 'political']},
                                  {'long_name': 'Ireland',
                                   'short_name': 'IE',
                                   'types': ['country', 'political']},
                                  {'long_name': 'D07 TC9C',
                                   'short_name': 'D07 TC9C',
                                   'types': ['postal_code']}],
                                 'formatted_address': '2 Leix Rd, Cabra East, Dublin 7, D07 TC9C, Ireland',
                                 'geometry': {'location': {'lat': 53.3616891, 'lng': -6.2836897},
                                  'location_type': 'ROOFTOP',
                                  'viewport': {'northeast': {'lat': 53.3630380802915,
                                    'lng': -6.282340719708498},
                                   'southwest': {'lat': 53.3603401197085, 'lng': -6.285038680291502}}},
                                 'partial_match': True,
                                 'place_id': 'ChIJf44Al9ENZ0gRggieTEAYLzg',
                                 'plus_code': {'compound_code': '9P68+MG Dublin 7, Ireland',
                                  'global_code': '9C5M9P68+MG'},
                                 'types': ['street_address']},
                                {'address_components': [{'long_name': '7',
                                   'short_name': '7',
                                   'types': ['street_number']},
                                  {'long_name': 'Leix Road', 'short_name': 'Leix Rd', 'types': ['route']},
                                  {'long_name': 'Cabra East',
                                   'short_name': 'Cabra East',
                                   'types': ['neighborhood', 'political']},
                                  {'long_name': 'Dublin 7',
                                   'short_name': 'Dublin 7',
                                   'types': ['postal_town']},
                                  {'long_name': 'County Dublin',
                                   'short_name': 'D',
                                   'types': ['administrative_area_level_1', 'political']},
                                  {'long_name': 'Ireland',
                                   'short_name': 'IE',
                                   'types': ['country', 'political']},
                                  {'long_name': 'D07 NX79',
                                   'short_name': 'D07 NX79',
                                   'types': ['postal_code']}],
                                 'formatted_address': '7 Leix Rd, Cabra East, Dublin 7, D07 NX79, Ireland',
                                 'geometry': {'location': {'lat': 53.3619821, 'lng': -6.284398299999999},
                                  'location_type': 'ROOFTOP',
                                  'viewport': {'northeast': {'lat': 53.36333108029149,
                                    'lng': -6.283049319708497},
                                   'southwest': {'lat': 53.3606331197085, 'lng': -6.285747280291502}}},
                                 'partial_match': True,
                                 'place_id': 'ChIJFbaDv9ENZ0gRcQzcq6ua84k',
                                 'plus_code': {'compound_code': '9P68+Q6 Dublin 7, Ireland',
                                  'global_code': '9C5M9P68+Q6'},
                                 'types': ['street_address']}]
```

Figure 41: An example of the object that the google API returns.

By looking at the object returned by the API, we noticed it returns a lot of information besides latitude and longitude, so we decided to try to store that information as well, in case we needed it later, so we created columns to hold all this other info:

```
In [77]: df['street_number'] = ''
         df['route'] = ''
         df['neighborhood'] = ''
         df['postal_town'] = ''
         df['county'] = ''
         df['postal_code'] = ''
```

Figure 42: Creating new Columns.

Code used to make the requests for the whole dataset:

```
In [63]: from time import sleep

In [80]: for x in range(len(df)):
             try:
                 sleep(1) #to add delay in case of large DFs
                 geocode_result = gmaps.geocode(df['Address'][x])
                 df['lat'][x] = geocode_result[0]['geometry']['location'] ['lat']
                 df['long'][x] = geocode_result[0]['geometry']['location']['lng']
                 for i in range(len(geocode_result[0]['address_components'])):
                     for j in range(len(geocode_result[0]['address_components'][i]['types'])):
                         if geocode_result[0]['address_components'][i]['types'][j] == "street_number":
                             df['street_number'][x] = geocode_result[0]['address_components'][i]['long_name']
                         elif geocode_result[0]['address_components'][i]['types'][j] == "route":
                             df['route'][x] = geocode_result[0]['address_components'][i]['long_name']
                         elif geocode_result[0]['address_components'][i]['types'][j] == "neighborhood":
                             df['neighborhood'][x] = geocode_result[0]['address_components'][i]['long_name']
                         elif geocode_result[0]['address_components'][i]['types'][j] == "postal_town":
                             df['postal_town'][x] = geocode_result[0]['address_components'][i]['long_name']
                         elif geocode_result[0]['address_components'][i]['types'][j] == "administrative_area_level_1":
                             df['county'][x] = geocode_result[0]['address_components'][i]['long_name']
                         elif geocode_result[0]['address_components'][i]['types'][j] == "postal_code":
                             df['postal_code'][x] = geocode_result[0]['address_components'][i]['long_name']
             except IndexError:
                 print("Address was wrong...")
             except Exception as e:
                 print("Unexpected error occurred.", e )
```

Figure 43: Code used to make the requests for the whole dataset.

This code was looping through the whole dataset, waiting one second not to overload Google's servers, and then making the request passing the value in the Address column, then selecting longitude e latitude and saving to the data frame. And this code is selecting just the first object returned, as we noticed above, sometimes it could return more than one object, but the first always is the one with the highest chances of being the right one. The code to get the other information about the address was a little annoying, we had to make nested loops, because the object returns lots of separate address components with different types, it was needed to loop through each address component and check what type it was to save it to the right column. In the end we could not even use these columns because this did not select it right for every row, there was a lot of missing information too, so It was a bit of an unnecessary effort. Have a look at the first rows after the code was run:

```
In [85]: df.head(30)
```

ostal Code	County	Price (€)	Not Full Market Price	VAT Exclusive	Description of Property	Property Size Description	long	lat	street_number	route	neighborhood	postal_town	county	postal_code
ublin 7	Dublin	30000.00	Yes	No	Second-Hand Dwelling house /Apartment	NaN	-6.28369	53.361689	2	Leix Road	Cabra East	Dublin 7	County Dublin	D07 TC9C
ublin 24	Dublin	30000.00	Yes	No	Second-Hand Dwelling house /Apartment	NaN	-6.336379	53.275809	2	Wood Dale View		Baile Átha Cliath 24	County Dublin	D24 ACF9
ublin 8	Dublin	15000.00	Yes	No	Second-Hand Dwelling house /Apartment	NaN	-6.271268	53.342518		Nicholas Street	Wood Quay	Dublin 8	County Dublin	

```
In [84]: df.shape
Out[84]: (9880, 17)

In [83]: df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9880 entries, 0 to 9879
Data columns (total 17 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   Date of Sale (dd/mm/yyyy)   9880 non-null   datetime64[ns]
 1   Address                     9880 non-null   object
 2   Postal Code                 6538 non-null   object
 3   County                      9880 non-null   object
 4   Price (€)                   9880 non-null   float64
 5   Not Full Market Price       9880 non-null   object
 6   VAT Exclusive               9880 non-null   object
 7   Description of Property     9880 non-null   object
 8   Property Size Description   31 non-null     object
 9   long                        9880 non-null   object
 10  lat                         9880 non-null   object
 11  street_number               9880 non-null   object
 12  route                       9880 non-null   object
 13  neighborhood                9880 non-null   object
 14  postal_town                 9880 non-null   object
 15  county                      9880 non-null   object
 16  postal_code                 9880 non-null   object
dtypes: datetime64[ns](1), float64(1), object(15)
memory usage: 1.3+ MB
```

Figure 44: Dataset EDA.

We noticed here that it was showing as if there were no null values for anything in those added columns, which is not true. It is because it was just as an empty string, so we converted all the empty strings to nan, to have an idea of how many information we could not get:

```
In [90]: import numpy as np

In [96]: df = df.apply(lambda x: x.str.strip() if isinstance(x, str) else x).replace('', np.nan)

In [97]: df.head(25)
```

ostal Code	County	Price (€)	Not Full Market Price	VAT Exclusive	Description of Property	Property Size Description	long	lat	street_number	route	neighborhood	postal_town	county	postal_code
ublin 7	Dublin	30000.00	Yes	No	Second-Hand Dwelling house /Apartment	NaN	-6.283690	53.361689	2	Leix Road	Cabra East	Dublin 7	County Dublin	D07 TC9C
ublin 24	Dublin	30000.00	Yes	No	Second-Hand Dwelling house /Apartment	NaN	-6.336379	53.275809	2	Wood Dale View	NaN	Baile Átha Cliath 24	County Dublin	D24 ACF9
ublin 8	Dublin	15000.00	Yes	No	Second-Hand Dwelling house	NaN	-6.271268	53.342518	NaN	Nicholas Street	Wood Quay	Dublin 8	County Dublin	NaN

```
In [98]: df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9880 entries, 0 to 9879
Data columns (total 17 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   Date of Sale (dd/mm/yyyy)   9880 non-null   datetime64[ns]
 1   Address                     9880 non-null   object
 2   Postal Code                 6538 non-null   object
 3   County                      9880 non-null   object
 4   Price (€)                   9880 non-null   float64
 5   Not Full Market Price       9880 non-null   object
 6   VAT Exclusive               9880 non-null   object
 7   Description of Property     9880 non-null   object
 8   Property Size Description   31 non-null     object
 9   long                        9748 non-null   float64
 10  lat                         9748 non-null   float64
 11  street_number               7288 non-null   object
 12  route                       9202 non-null   object
 13  neighborhood                6731 non-null   object
 14  postal_town                 4318 non-null   object
 15  county                      9745 non-null   object
 16  postal_code                 5834 non-null   object
dtypes: datetime64[ns](1), float64(3), object(13)
memory usage: 1.3+ MB

In [88]: df.to_csv('df_2019_coords.csv')
```

Figure 12: Dataset EDA.

Now we can see that unfortunately we have a lot of missing information, the most important thing was latitude and longitude though, and that has 132 missing rows. The information was saved to a csv file, and we then did the exact same thing with all the other years. Because of the second we had to wait between each request, it took a long time. About 4 days of the computer running the code day and night.

One quite annoying problem was that some addresses were typed wrong, and the API didn't get them, we could see that 2019 had 132 missing rows, and it was similar with 2018 and 2019. For those addresses we had to manually type in google maps, figure out what was wrong and tweak the address to be able to find and manually fill those ones.

After doing that we still had the data in separate datasets, so we concatenated them all into one dataset:

Importing the libraries

In [1]:
```
import glob, os
import pandas as pd
```

Reading the 3 csv files of years 2017, 2018 and 2019 into one dataframe

In [2]:
```
df = pd.concat(map(pd.read_csv, glob.glob(os.path.join('', "*.csv"))))
```

Figure 46: Concatenating Property datasets from 2017,2018, and 2019, to one data frame.

Have a look at the dataset now:

Checking if all it worked. 2017 file had 17,953 rows, 2018 had 18,625 and 2019 had 9,881

In [3]:
```
df.shape
```

Out[3]: (46458, 18)

We can see that it worked because the number of rows in the dataframe equals the sum of all 3 files. Let's have a look in the top 5 rows:

In [4]:
```
df.head()
```

Out[4]:

	Unnamed: 0	Date of Sale (dd/mm/yyyy)	Address	Postal Code	County	Price (€)	Not Full Market Price	VAT Exclusive	Description of Property	Property Size Description	long	lat	street_number	
0	0	02/01/2018	1 ABBEY ST, HOWTH, DUBLIN	Dublin 7	Dublin	710000.0	No	No	Second-Hand Dwelling house /Apartment	NaN	-6.0646082	53.388117	1	Abbey
1	1	02/01/2018	1 ROSEVILLE, LOWER ROAD, SHANKILL	Dublin 18	Dublin	627230.7	No	No	Second-Hand Dwelling house /Apartment	NaN	-6.1253428	53.232940	1	R
2	2	02/01/2018	19 ULVERTON RD, DALKEY, DUBLIN	NaN	Dublin	1130000.0	No	No	Second-Hand Dwelling house /Apartment	NaN	-6.107343	53.279602	19	L
3	3	02/01/2018	236 PHIBSBORO RD, DUBLIN 7, DUBLIN	Dublin 7	Dublin	277000.0	No	No	Second-Hand Dwelling house /Apartment	NaN	-6.2733223	53.355613	236	Phibsb
4	4	02/01/2018	3 COLLEGE COURT, PORTRANE ROAD, DONABATE	NaN	Dublin	348017.62	No	No	Second-Hand Dwelling house /Apartment	NaN	-6.1412172	53.489876	NaN	P

There is an extra index column that is not need so I am going to delete it

In [5]:
```
df = df.drop(columns=['Unnamed: 0'])
```

Figure 47: Data frame after merging the Properties datasets together.

Check the data types:

```
In [11]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 46458 entries, 0 to 17951
Data columns (total 17 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   Date of Sale (dd/mm/yyyy)   46458 non-null  object
 1   Address                     46458 non-null  object
 2   Postal Code                 29767 non-null  object
 3   County                      46458 non-null  object
 4   Price (€)                   46458 non-null  object
 5   Not Full Market Price       46458 non-null  object
 6   VAT Exclusive               46458 non-null  object
 7   Description of Property     46458 non-null  object
 8   Property Size Description   8294 non-null   object
 9   long                        46404 non-null  object
 10  lat                         46404 non-null  float64
 11  street_number               33257 non-null  object
 12  route                       43053 non-null  object
 13  neighborhood                32648 non-null  object
 14  postal_town                 20525 non-null  object
 15  county                      45874 non-null  object
 16  postal_code                 27835 non-null  object
dtypes: float64(1), object(16)
memory usage: 6.4+ MB
```

We can see that date is of type String so let's convert it to DateTime format

```
In [12]: df['Date of Sale (dd/mm/yyyy)'] = pd.to_datetime(df['Date of Sale (dd/mm/yyyy)'], format='%d/%m/%Y')
```

Figure 48: Checking the data types in the dataset.

By checking the first 5 rows and the last 5, we noticed that it was not sorted by year. It was starting in 2018, then 2019 and 2017 was at the end:

```
In [14]: df.head()
```

Out[14]:

	Date of Sale (dd/mm/yyyy)	Address	Postal Code	County	Price (€)	Not Full Market Price	VAT Exclusive	Description of Property	Property Size Description	long	lat	street_number	route	ne
0	2018-01-02	1 ABBEY ST, HOWTH, DUBLIN	Dublin 7	Dublin	710000.0	No	No	Second-Hand Dwelling house /Apartment	NaN	-6.0646082	53.388117	1	Abbey Street	
1	2018-01-02	1 ROSEVILLE, LOWER ROAD, SHANKILL	Dublin 18	Dublin	627230.7	No	No	Second-Hand Dwelling house /Apartment	NaN	-6.1253428	53.232940	1	Roseville	
2	2018-01-02	19 ULVERTON RD, DALKEY, DUBLIN	NaN	Dublin	1130000.0	No	No	Second-Hand Dwelling house /Apartment	NaN	-6.107343	53.279602	19	Ulverton Road	
3	2018-01-02	236 PHIBSBORO RD, DUBLIN 7, DUBLIN	Dublin 7	Dublin	277000.0	No	No	Second-Hand Dwelling house /Apartment	NaN	-6.2733223	53.355613	236	Phibsborough Road	
4	2018-01-02	3 COLLEGE COURT, PORTRANE ROAD, DONABATE	NaN	Dublin	348017.62	No	No	Second-Hand Dwelling house /Apartment	NaN	-6.1412172	53.489876	NaN	Portrane Road	

```
In [15]: df.tail()
```

Out[15]:

	Date of Sale (dd/mm/yyyy)	Address	Postal Code	County	Price (€)	Not Full Market Price	VAT Exclusive	Description of Property	Property Size Description	long	lat	street_number	route
17947	2017-12-29	13 Boyd Avenue, Honeypark, Dun Laoghaire	NaN	Dublin	€640,968.00	No	Yes	New Dwelling house /Apartment	greater than or equal to 38 sq metres and less...	-6.142628	53.279831	13	Boyd Avenue
17948	2017-12-29	14 ST ANTHONY'S ROAD, RIALTO, DUBLIN 8	NaN	Dublin	€215,000.00	No	No	Second-Hand Dwelling house /Apartment	NaN	-6.294655	53.336367	14	Saint Anthony's Road
17949	2017-12-29	8 ST ANTHONYS RD, RIALTO, DUBLIN 8	Dublin 8	Dublin	€215,000.00	No	No	Second-Hand Dwelling house /Apartment	NaN	-6.294849	53.336090	8	Saint Anthony's Road
17950	2017-12-30	39 GLENCARRIG DR, FIRHOUSE, DUBLIN 24	Dublin 24	Dublin	€350,000.00	No	No	Second-Hand Dwelling house /Apartment	NaN	-6.349446	53.275307	39	Glencarrig Drive
17951	2017-12-30	98 KILDONAN AVE, FINGLAS WEST, DUBLIN 11	Dublin 11	Dublin	€220,000.00	No	No	Second-Hand Dwelling house /Apartment	NaN	-6.311186	53.392124	98	Kildonan Avenue

Figure 49: Property dataset before sorting.

Sorting the data frame by date:

```
In [16]: df = df.sort_values(by='Date of Sale (dd/mm/yyyy)')
```

The price was read as type String, so I removed the Euro symbol that it had in the front and converted to Float

```
In [18]: df[df.columns[4]] = df[df.columns[4]].replace('[\€,]', '', regex=True).astype(float)
```

Renaming some of the columns

```
In [20]: df.rename({'Date of Sale (dd/mm/yyyy)':'Date of Sale', 'Price (€)':'Price', 'long': 'Longitude', 'lat': 'Latitude', 'n
```

```
In [21]: df
```

Out[21]:

	Date of Sale	Address	Postal Code	County	Price	Not Full Market Price	VAT Exclusive	Description of Property	Property Size Description	Longitude	Latitude	street_number	route	Neigl
1	2017-01-01	11 PINTAIL HOUSE, REDCOURT OAKS, SEAFIELD RD EAST	Dublin 3	Dublin	242424.0	Yes	No	Second-Hand Dwelling house /Apartment	NaN	-6.1814048	53.362710	NaN	Seafield Road East	

Figure 50: Property dataset after sorting.

Full code to rename the columns as it did not show all in the print screen:
df = df.rename({'Date of Sale (dd/mm/yyyy)':'Date of Sale', 'Price (€)':'Price', 'long': 'Longitude', 'lat': 'Latitude', 'neighborhood':'Neighborhood', 'postal_town': 'Postal Town'}, axis='columns')

Next thing was to transform empty spaces to NaN.

Transforming all the empty spaces to NaN

```
In [22]: import numpy as np
```

```
In [23]: df = df.apply(lambda x: x.str.strip() if isinstance(x, str) else x).replace('', np.NaN)
```

Figure 51: transforming null rows with NaN value.

Checking data types and null values again:

```
In [25]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 46458 entries, 1 to 9873
Data columns (total 17 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Date of Sale              46458 non-null  datetime64[ns]
 1   Address                   46458 non-null  object
 2   Postal Code               29767 non-null  object
 3   County                    46458 non-null  object
 4   Price                     46458 non-null  float64
 5   Not Full Market Price     46458 non-null  object
 6   VAT Exclusive             46458 non-null  object
 7   Description of Property   46458 non-null  object
 8   Property Size Description  8294 non-null   object
 9   Longitude                 46404 non-null  object
 10  Latitude                  46404 non-null  float64
 11  street_number             33257 non-null  object
 12  route                     43053 non-null  object
 13  Neighborhood              32648 non-null  object
 14  Postal Town               20525 non-null  object
 15  county                    45874 non-null  object
 16  postal_code               27835 non-null  object
dtypes: datetime64[ns](1), float64(2), object(14)
memory usage: 6.4+ MB
```

Figure 52: Checking for Null values.

We can see that we have a few missing values. There are 16,691 missing values for Postal Code, 38,164 missing values for property size Description.

Columns 9 to 16 were generated with the Google Maps API, because we needed to get the latitude and longitude of the addresses and because I knew we possibly would want to separate the properties by area I generated columns 11 to 16 to get the area and neighbourhood since the Postal Code column in the original dataset has a lot of missing values. It did not work so well, the only two columns that would be useful would be 'neighbourhood' and 'postal-town' but unfortunately the API did not return a result for every row, leaving us with 13,810 empty rows for neighbourhood and 25,933 for postal town.

From the columns returned by Google Maps, the only 2 that could still be useful are 'neighbourhood' and 'postal_town' so we dropped street_number, route, county and postal_code. And since we selected only Dublin data, there is no point in having a county column, because all the entries are County Dublin:

```
In [26]: df['County'].unique()
Out[26]: array(['Dublin'], dtype=object)

In [27]: df = df.drop(['County', 'street_number', 'route', 'county', 'postal_code'], axis =1)

In [28]: df.shape
Out[28]: (46458, 12)

In [29]: df.head()
Out[29]:
```

	Date of Sale	Address	Postal Code	Price	Not Full Market Price	VAT Exclusive	Description of Property	Property Size Description	Longitude	Latitude	Neighborhood	Postal Town
1	2017-01-01	11 PINTAIL HOUSE, REDCOURT OAKS, SEAFIELD RD EAST	Dublin 3	242424.0	Yes	No	Second-Hand Dwelling house /Apartment	NaN	-6.1814048	53.362710	Clontarf	Dublin 3

Figure 53: Drop unnecessary columns.

Investigating what are the values for some columns:

```
In [44]: df['Not Full Market Price'].unique()
Out[44]: array(['Yes', 'No'], dtype=object)

In [45]: df['VAT Exclusive'].unique()
Out[45]: array(['No', 'Yes'], dtype=object)

In [46]: df['Description of Property'].unique()
Out[46]: array(['Second-Hand Dwelling house /Apartment',
                'New Dwelling house /Apartment', 'Teach/�ras�n C�naithe Ath�imhe'],
               dtype=object)

In [47]: df['Property Size Description'].unique()
Out[47]: array([nan,
                'greater than or equal to 38 sq metres and less than 125 sq metres',
                'greater than or equal to 125 sq metres', 'less than 38 sq metres'],
               dtype=object)
```

We can see that Property Size Description was considered as:

- less than 38 sq metres
- greater than or equal to 38 sq metres and less than 125 sq metres
- greater than or equal to 125 sq metres

It would be a very useful feature but we have only 8294 values out of 46458 so aprox. 82% of the values are missing, because of that I am going to drop that column as well

```
In [48]: df = df.drop(['Property Size Description'], axis =1)
```

Figure 54: Investigating dataset.

Investigating the unique values for some columns:

```
In [44]: df['Not Full Market Price'].unique()
Out[44]: array(['Yes', 'No'], dtype=object)

In [45]: df['VAT Exclusive'].unique()
Out[45]: array(['No', 'Yes'], dtype=object)

In [46]: df['Description of Property'].unique()
Out[46]: array(['Second-Hand Dwelling house /Apartment',
                'New Dwelling house /Apartment', 'Teach/Óras Dún Cúnaithe Athláimhe'],
               dtype=object)

In [47]: df['Property Size Description'].unique()
Out[47]: array([nan,
                'greater than or equal to 38 sq metres and less than 125 sq metres',
                'greater than or equal to 125 sq metres', 'less than 38 sq metres'],
               dtype=object)
```

We can see that Property Size Description was considered as:

- less than 38 sq metres
- greater than or equal to 38 sq metres and less than 125 sq metres
- greater than or equal to 125 sq metres

It would be a very useful feature but we have only 8294 values out of 46458 so aprox. 82% of the values are missing, because of that I am going to drop that column as well

```
In [48]: df = df.drop(['Property Size Description'], axis =1)
```

Figure 55: Investigating unique values.

Checking data types and null values for the data set again for the features left:

```
In [50]: df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 46458 entries, 0 to 46457
Data columns (total 10 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Date of Sale            46458 non-null  datetime64[ns]
 1   Address                 46458 non-null  object
 2   Postal Code             32607 non-null  object
 3   Price                   46458 non-null  float64
 4   Not Full Market Price   46458 non-null  object
 5   VAT Exclusive           46458 non-null  object
 6   Description of Property 46458 non-null  object
 7   Longitude               46404 non-null  object
 8   Latitude                46404 non-null  float64
 9   Neighborhood            32648 non-null  object
dtypes: datetime64[ns](1), float64(2), object(7)
memory usage: 3.5+ MB
```

Figure 56: Checking data types.

Apparently, Longitude is as String and Latitude as Float. This was because one or more entries had a comma in front of the number, for instance: ',-6.23454', and that was my fault when manually copying and pasting from Google Maps for the rows that the API didn't get. It should be numeric so we will convert it to Float.

```
In [51]: df[df.columns[7]] = df[df.columns[7]].replace('[\Ó,]', '', regex=True).astype(float)
```

There are a few missing values for Latitude and Longitude, these are values that we couldn't understand what the correct address was, and we preferred not to input it since we could be putting it in the wrong place. So we just dropped those:

```
In [53]: df = df.loc[df['Longitude'].notnull()]
```

Fixing indexes and writing final data frame to csv file:

```
In [56]: df = df.reset_index(drop=True)

In [58]: df.to_csv('df_allyears_coords.csv')
```

11.2 Integrate data.

Now that we have all our data pre-prepared and we know what to do, the next step is integrating all our data so we can format and use it in our model. Our group planned to do it in 3 steps:

- Merge the schools/transportation/garda datasets into one complete dataset with the location of all of them.
- Compare the properties dataframe with the schools/transportation/garda dataset to check how many of these facilities are close to each property.
- Create a dataframe with the quantity of facilities and merge to our first properties dataset so we can then work with our final dataset.

11.2.1 Merging the schools/transportation/garda datasets.

We first imported all our datasets so we can start working with them, as the group separated working with datasets in the first part, some of them have extra columns and extra data that will not be used. You can notice below that the garda stations dataset has an extra index column and they do not really have IDs or a Type column, same happens for the Schools dataset, which has a lot of information we did not use. They all need to be formatted according to our transport dataset so they can be merged later.

```
blt_df = pd.read_csv('Data/blt_stops.csv')
```

```
blt_df.head()
```

	Stop_Id	Latitude	Longitude	Type
0	7000B6310001	54.296217	-7.871793	Bus Stop
1	700000004183	54.346455	-6.271543	Bus Stop
2	700000004149	54.346297	-6.271180	Bus Stop
3	7000B141561	54.193380	-7.567849	Bus Stop
4	7000B140871	54.193317	-7.567941	Bus Stop

Figure 57: Bus stop dataset.

```
garda_df = pd.read_csv('Data/garda_coords.csv')
```

```
garda_df.head()
```

	Unnamed: 0	Garda Station	Longitude	Latitude
0	0	Balbriggan Garda Station	-6.190980	53.614378
1	1	Ballyfermot Garda Station	-6.358053	53.344754
2	2	Ballymun Garda Station	-6.263922	53.394374
3	3	Blackrock Garda Station	-6.177503	53.299793
4	4	Blanchardstown Garda Station	-6.380952	53.389924

Figure 58: garda station dataset.

```
schools_df = pd.read_csv('Data/schools_location.csv')
```

```
schools_df.head()
```

	Roll Number	schoolName	Principal Name	Ethos Description	Female	Male	Total	latitude	longitude	Type
0	00697S	ST BRIGIDS MXD N S	Denis Courtney	Catholic	447	502.0	949	53.375855	-6.361625	Primary School
1	00714P	LUCAN B N S	Dara Burke	Catholic	0	513.0	513	53.360019	-6.438876	Primary School
2	00729F	CLOCHAR LORETO N S	Sr. Maria Hyland	Catholic	501	0.0	501	53.295317	-6.282539	Primary School
3	00752A	CENTRAL SENIOR MXD N S	Deirdre Gartland	Catholic	133	143.0	276	53.349314	-6.257670	Primary School
4	01170G	S N NA H-AILLE	Edel McMahon	Catholic	67	72.0	139	53.579286	-6.299281	Primary School

Figure 58: School datasets.

After cleaning and formatting all those three datasets we were able to merge them as you can see in the figure below.

Merging Datasets

Append all datasets to a new one, reset index and drop index.

```
blts_df = blt_df.append(schools_df)
```

```
blts_df = blts_df.append(garda_df)
```

```
blts_df = blts_df.reset_index()
```

```
blts_df = blts_df.drop(columns='index')
```

```
blts_df
```

	ID	Latitude	Longitude	Type
0	7000B6310001	54.296217	-7.871793	Bus Stop
1	700000004183	54.346455	-6.271543	Bus Stop
2	700000004149	54.346297	-6.271180	Bus Stop
3	7000B141561	54.193380	-7.567849	Bus Stop
4	7000B140871	54.193317	-7.567941	Bus Stop
...	...	...	...	...
20913	-5114991191610204258	53.350485	-6.251706	Garda Station
20914	5282813725658455909	53.330200	-6.298697	Garda Station
20915	-4165365203422798977	53.456024	-6.221112	Garda Station
20916	7037954590483101354	53.286807	-6.367589	Garda Station
20917	5215355667214597849	53.309990	-6.287810	Garda Station

20918 rows × 4 columns

Figure 59: merging all datasets.

After merging all our datasets, we got a huge final dataset, which would be a problem as we already have our properties dataset with a size of around 47.000

rows, and every one of these rows would need to be checked against this dataset. Fortunately, a lot of these entries are not in the Dublin area, so we just needed to remove what was not inside the Dublin county. We first did a scatterplot to check that, and we got a few outliers in it as you can see below.

```
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize = (15,8))
sns.scatterplot(scatter_test["Latitude"], scatter_test["Longitude"])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x256c27747c0>
```

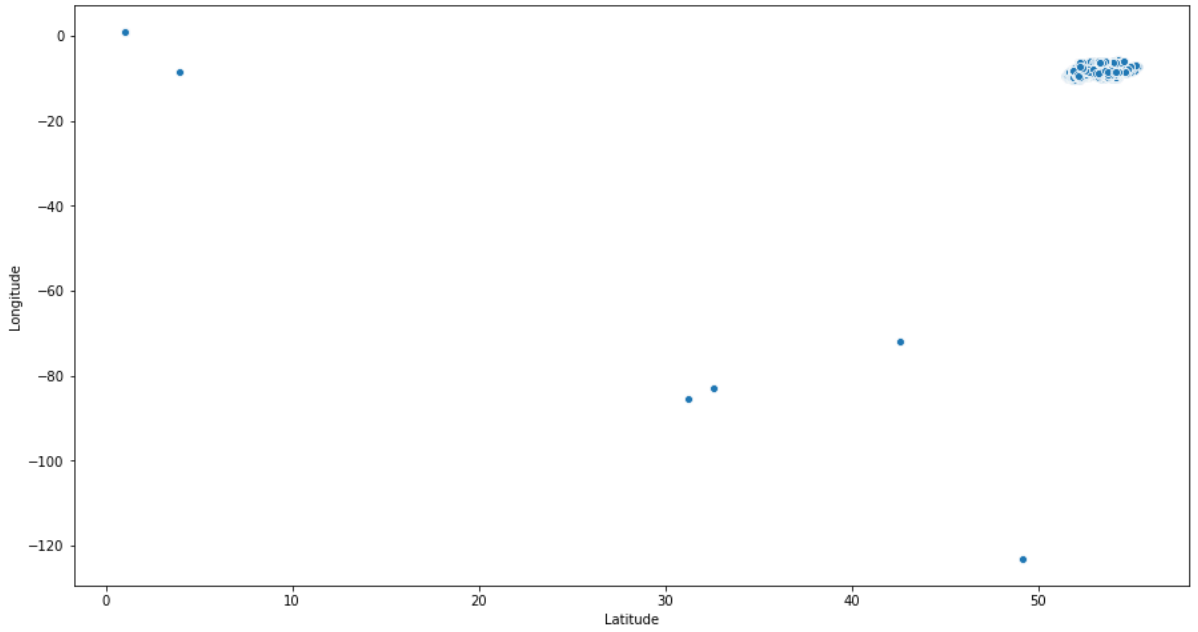

Figure 60: outliers in the dataset.

From there we knew that if we managed to get an area around Dublin, that would be enough to get rid of a lot of this data and the outliers at the same time. As there was no way to get the exact size of Dublin County, we set a radius around Dublin that we wanted to get all the inside points. We utilized the Haversine formula to get this, according to Setyorini and Ramayanti (2019) “Haversine Formula is a formula used in navigation that can provide a large circle between two points on the ball / earth from longitude and latitude”. Because we are using geographic points, we needed to utilize this formula to set our radius.

Utilizing Google Maps (2021) we set a central point in Dublin (53.385574, -6.280166) and then we put another point to the North to the furthest border. We then got this distance to set our Radius, which was around 29.5km but we rounded it to 30km. Of course, this area would still get points outside of Dublin, but as the properties are Dublin based, the houses will only get around the Dublin area.

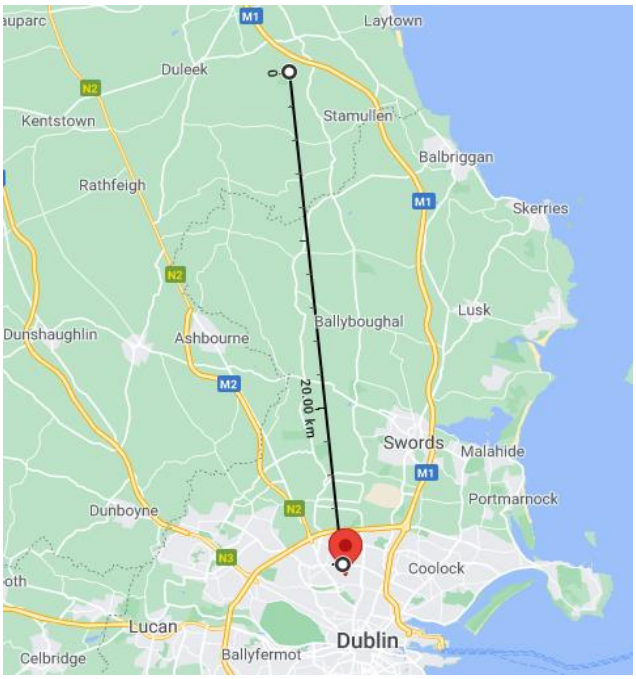

Figure 61: 30km radius from Dublin.

We referenced our Haversine formula code from a Stack Overflow (2017) user called ytomo, in which we were able to select this radius of 30km and delete anything outside of it. As you can see below, we managed to get rid of around 12.000 entries of the dataset.

```
from math import radians, cos, sin, asin, sqrt

def haversine(lon1, lat1, lon2, lat2):

    lon1, lat1, lon2, lat2 = map(radians, [lon1, lat1, lon2, lat2])

    dlon = lon2 - lon1
    dlat = lat2 - lat1
    a = sin(dlat/2)**2 + cos(lat1) * cos(lat2) * sin(dlon/2)**2
    c = 2 * asin(sqrt(a))
    r = 6371
    return c * r

center_point = [{'lat': 53.385574, 'lng': -6.280166}]


lat1 = center_point[0]['lat']
lon1 = center_point[0]['lng']

radius = 30.00

for label1, row1 in blts_df.iterrows():

    test_point = [{'lat': row1['Latitude'], 'lng': row1['Longitude']}]

    lat2 = test_point[0]['lat']
    lon2 = test_point[0]['lng']

    a = haversine(lon1, lat1, lon2, lat2)

    if a <= radius:

    else:
        blts_df = blts_df.drop(label1)
```

Figure 62: We managed to get rid of around 12.000 entries of the dataset.

```
blts_df
```

	ID	Latitude	Longitude	Type
145	8220DB001738	53.363451	-6.223914	Bus Stop
146	8220DB007516	53.402076	-6.171727	Bus Stop
147	8220DB000592	53.382733	-6.148383	Bus Stop
148	8220DB001228	53.392429	-6.218513	Bus Stop
149	8220DB001237	53.392679	-6.218983	Bus Stop
...	...	...	...	...
20913	-5114991191610204258	53.350485	-6.251706	Garda Station
20914	5282813725658455909	53.330200	-6.298697	Garda Station
20915	-4165365203422798977	53.456024	-6.221112	Garda Station
20916	7037954590483101354	53.286807	-6.367589	Garda Station
20917	5215355667214597849	53.309990	-6.287810	Garda Station

7391 rows × 4 columns

Figure 63: all datasets together.

To check if everything was correct and inside the Dublin area, we plotted a new scatterplot and put the coordinates against a map.

```
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize = (15,8))
sns.scatterplot(scatter_test["Latitude"], scatter_test["Longitude"])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x256c913b4f0>
```

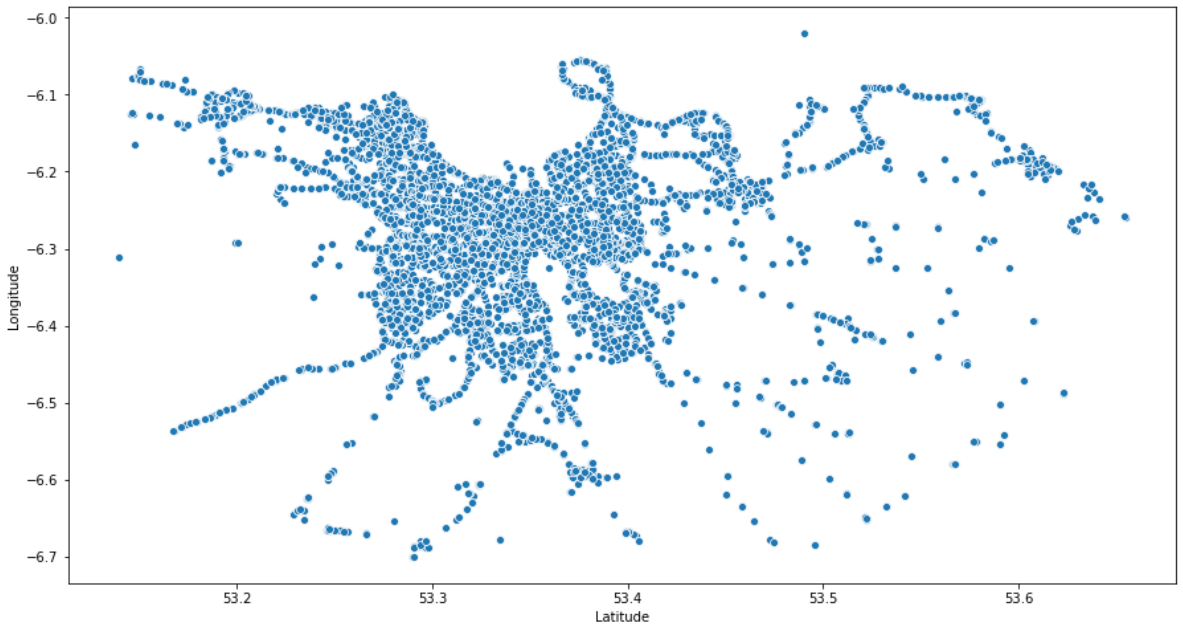

Figure 64: Plot to check if the dataset covers the hole Dublin area.

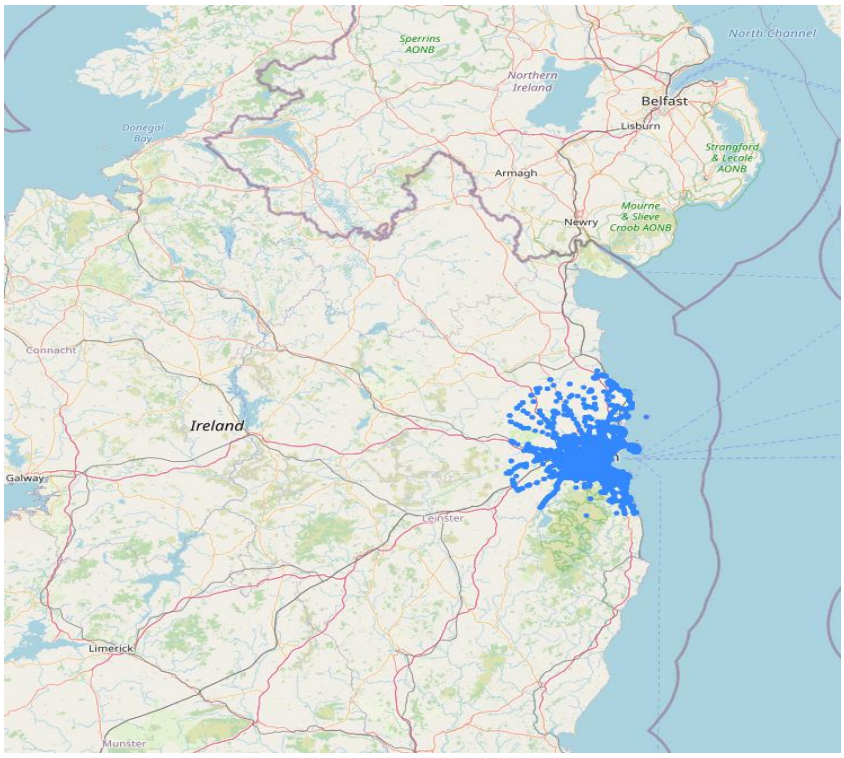

Figure 65: Plot to check if the dataset covers the hole Dublin area.

11.2.2 Compare the property's locations against the schools/transportation/garda locations.

After reducing and checking our blts_df.csv dataset we can now use it with our properties dataset to check how many of these are around each property. We first need to check again our properties dataset and prepare it to be worked with. As we already have the geographical points, we do not need any addresses or areas, we can just drop those. We will only need Date of Sale, Price, Not Full Market Price, Vat Exclusive, Description of Property, Latitude and Longitude. Some of these will not be used in the future, we will just keep them for now. We will also add an ID column according to their index as we will need to identify them later.

```
houses_df
```

	Date of Sale	Price	Not Full Market Price	VAT Exclusive	Description of Property	Longitude	Latitude	ID
0	2017-01-01	242424.0	Yes	No	Second-Hand Dwelling house /Apartment	-6.181405	53.362710	0
1	2017-01-01	242424.0	Yes	No	Second-Hand Dwelling house /Apartment	-6.181405	53.362710	1
2	2017-01-01	66000.0	Yes	No	Second-Hand Dwelling house /Apartment	-6.182833	53.362984	2
3	2017-01-01	271800.0	Yes	No	Second-Hand Dwelling house /Apartment	-6.181405	53.362710	3
4	2017-01-01	830000.0	No	No	Second-Hand Dwelling house /Apartment	-6.181892	53.365139	4

Figure 66: adding ID index.

Our objective was to create a new dataframe where it will have the property IDs with the quantity of each point, to do that we need to calculate which points are 1km from that property utilizing geodesic which calculates the distance in km from one point to another.

The objective of this script was to first loop through our property's dataset, for every row it would set our "counter" variables (luas, bus, train, PS, SS, garda) to 0.

```
from geopy.distance import geodesic

#Create new dataframe.
df = pd.DataFrame(columns = ['ID', 'Luas_quantity', 'Bus_quantity','Train_quantity','PS_quantity','SS_quantity','Garda_quantity']

#Loop through the properties dataset.
for label, row in houses_df_test.iterrows():
    #Set the counter variables.
    luas = 0
    bus = 0
    train = 0
    PS = 0
    SS = 0
    garda = 0

    #Loop through the blts dataset.
    for label1, row1 in blts_df.iterrows():
        #Calculate the distance from the property to the current blts row.
        distance_in_km = geodesic((row['Latitude'],row['Longitude']),(row1['Latitude'],row1['Longitude'])).km
        #Set if statements to add to counter if certain type and smaller or equal to 1km.
        if distance_in_km <= 1 and row1['Type'] == "Bus Stop":
            bus += 1

        elif distance_in_km <= 1 and row1['Type'] == "Luas Stop":
            luas += 1


        elif distance_in_km <= 1 and row1['Type'] == "Train Stop":
            train += 1

        elif distance_in_km <= 1 and row1['Type'] == "Primary School":
            PS += 1

        elif distance_in_km <= 1 and row1['Type'] == "Secondary School":
            SS += 1

        elif distance_in_km <= 1 and row1['Type'] == "Garda Station":
            SS += 1
    #Append in the end to our dataframe.
    df = df.append({'ID': row['ID'],'Luas_quantity': luas,'Bus_quantity':bus, 'Train_quantity':train, 'PS_quantity':PS,'SS_quanti
```

Figure 67: script to first loop through our property's dataset and calculated the distance in kilometres between the current property and the current row in this dataset.

Then, it used a nested loop for the transport/schools/garda dataset. It then calculated the distance in kilometres between the current property and the current row in this dataset. There it used if statements to add to the counters according to the type of each row and if they were located one kilometre or less from the property. Then after the loop finished, it would add the property ID and the quantity of each point/amenity to the new dataframe that was created in the beginning.

Because there were 46,404 rows in the property's dataset and 7,391 in the transport/schools/garda dataset, there was a lot of information to be checked. In total there were 342,971,964 rows that the code ran through. Each properties dataset row took around 2.22 seconds, we calculated around 29 hours for the code to run and the result was approximately that.

```
time: 1d 5h 20min 31s (started: 2021-04-21 19:43:25 +01:00)
```

Below you can see that the right number of rows were returned and how the dataframe looked like.

	ID	Luas_quantity	Bus_quantity	Train_quantity	PS_quantity	SS_quantity	Garda_quantity
0	0.0	0.0	32.0	0.0	1.0	0.0	0.0
1	1.0	0.0	32.0	0.0	1.0	0.0	0.0
2	2.0	0.0	34.0	0.0	1.0	0.0	0.0
3	3.0	0.0	32.0	0.0	1.0	0.0	0.0
4	4.0	0.0	33.0	0.0	1.0	0.0	0.0
...	...	...	...	...	...	...	...
46399	46399.0	0.0	49.0	1.0	5.0	0.0	0.0
46400	46400.0	4.0	64.0	0.0	5.0	2.0	0.0
46401	46401.0	0.0	39.0	1.0	3.0	1.0	0.0
46402	46402.0	6.0	49.0	0.0	3.0	1.0	0.0
46403	46403.0	0.0	91.0	0.0	6.0	6.0	0.0

46404 rows × 7 columns

Figure 68: Final dataframe, getting shape.

From here we had to transform each value to an int datatype as they were returned as floats. And then we could merge them with our old properties dataset.

```
df = df.astype(int)
```

```
time: 0 ns (started: 2021-04-23 13:16:59 +01:00)
```

Now the dataframe are all integers.

```
df
```

	ID	Luas_quantity	Bus_quantity	Train_quantity	PS_quantity	SS_quantity	Garda_quantity
0	0	0	32	0	1	0	0
1	1	0	32	0	1	0	0
2	2	0	34	0	1	0	0
3	3	0	32	0	1	0	0
4	4	0	33	0	1	0	0
...	...	...	...	...	...	...	...
46399	46399	0	49	1	5	0	0
46400	46400	4	64	0	5	2	0
46401	46401	0	39	1	3	1	0
46402	46402	6	49	0	3	1	0
46403	46403	0	91	0	6	6	0

46404 rows × 7 columns

```
time: 16 ms (started: 2021-04-23 13:17:03 +01:00)
```

Figure 69: transform each value to an int datatype.

	ID	Luas_quantity	Bus_quantity	Train_quantity	PS_quantity	SS_quantity	Garda_quantity	Date of Sale	Price	Not Full Market Price	VAT Exclusive	Description of Property	Longitude	La
0	0	0	32	0	1	0	0	2017-01-01	242424.0	Yes	No	Second-Hand Dwelling house /Apartment	-6.181405	53.3
1	1	0	32	0	1	0	0	2017-01-01	242424.0	Yes	No	Second-Hand Dwelling house /Apartment	-6.181405	53.3
2	2	0	34	0	1	0	0	2017-01-01	66000.0	Yes	No	Second-Hand Dwelling house /Apartment	-6.182833	53.3
3	3	0	32	0	1	0	0	2017-01-01	271800.0	Yes	No	Second-Hand Dwelling house /Apartment	-6.181405	53.3
4	4	0	33	0	1	0	0	2017-01-01	830000.0	No	No	Second-Hand Dwelling house /Apartment	-6.181892	53.3

Figure 70: Final dataframe.

11.3 Feature Engineering.

One of the most important stages in the implementation of machine learning is feature engineering. To train our model, we need the data to be in a numeric and tidy format. “Feature engineering is the act of extracting features from raw data and transforming them into formats that are suitable for the machine learning model” (Casari & Zheng, 2018, p.vii).

Feature engineering converts raw data into numbers that can be used to construct your feature matrix, which means taking whatever information you have about your problem and translating it into numbers.

To start our feature engineering, we will check the data with info () method.

```
1  df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 46404 entries, 0 to 46403
Data columns (total 14 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   ID                      46404 non-null  int64
 1   Luas_quantity           46404 non-null  int64
 2   Bus_quantity            46404 non-null  int64
 3   Train_quantity          46404 non-null  int64
 4   PS_quantity             46404 non-null  int64
 5   SS_quantity             46404 non-null  int64
 6   Garda_quantity          46404 non-null  int64
 7   Date of Sale            46404 non-null  object
 8   Price                   46404 non-null  float64
 9   Not Full Market Price   46404 non-null  object
 10  VAT Exclusive           46404 non-null  object
 11  Description of Property 46404 non-null  object
 12  Longitude               46404 non-null  float64
 13  Latitude                46404 non-null  float64
dtypes: float64(3), int64(7), object(4)
memory usage: 5.0+ MB
```

Figure 71: Checking dataframe for possible null values.

We will rename the 'Date of Sale' to 'Date_of_Sale, 'Not Full Market Price' to 'Not_full_market_price', 'VAT Exclusive' to 'VAT_exclusive' and 'Description of Property' to 'Description_of_property' columns to eliminate white spaces.

```
1  df.rename(columns={'Date of Sale':'Date_of_Sale',
2                     'Not Full Market Price':'Not_full_market_price',
3                     'VAT Exclusive' : 'VAT_exclusive',
4                     'Description of Property': 'Description_of_property'},
5            inplace=True)
```

Figure 72: renaming columns to eliminate white spaces.

Following, will check NaN values and unique values within the dataset.

```
print('Luas_quantity:',df.Luas_quantity.unique())
print('Bus_quantity :',df.Bus_quantity .unique())
print('Train_quantity :',df.Train_quantity.unique())
print('PS_quantity:',df.PS_quantity.unique())
print('SS_quantity:',df.SS_quantity.unique())
print('Garda_quantity:',df.Garda_quantity.unique())
print('Date_of_Sale:',df.Date_of_Sale.unique())
print('Price:',df.Price.unique())
print('Not_full_market_price:',df.Not_full_market_price.unique())
print('VAT_exclusive:',df.VAT_exclusive.unique())
print('Description_of_property :',df.Description_of_property.unique())
```

Figure 73: Checking unique values in the dataset.

Then will be encoding the 'Not_full_market_price' and 'VAT_exclusive' column.

```
df['Not_full_market_price'] = df['Not_full_market_price'].map({'Yes':1, 'No':0})

df['VAT_exclusive'] = df['VAT_exclusive'].map({'Yes':1, 'No':0})
```

Figure 74: Encoding Not_full_market price and VAT_ exclusive columns.

After evaluating the column ‘Description_of_property’ we can see that it has some odd characters and three rows.

```
df['Description_of_property'].value_counts()
```

```
Second-Hand Dwelling house /Apartment    35459
New Dwelling house /Apartment            10942
Teach/�ras�n C�naithe Ath�imhe               3
Name: Description_of_property, dtype: int64
```

Figure 75: cleaning data process.

So,we will drop it.

```
df.drop(df[df.Description_of_property == 'Teach/-ras-n C-naithe Ath-imhe'].index, inplace = True)
```

Figure 76: cleaning data process.

Correcting the values inside ‘Description_of_property’ to eliminate white spaces before encoding.

```
'] = df['Description_of_property'].replace(['Second-Hand Dwelling house /Apartment'],'Second_Hand_Dwelling_house_Apartment')

tion_of_property'] = df['Description_of_property'].replace(['New Dwelling house /Apartment'],'New_Dwelling_house_Apartment')
```

Figure 77: Cleaning data process

For this binary encoding we will use OneHotEncoder from sklearn.

```
from sklearn.preprocessing import OneHotEncoder

oe_style = OneHotEncoder()
oe_results = oe_style.fit_transform(df[["Description_of_property"]])
pd.DataFrame(oe_results.toarray(), columns=oe_style.categories_).head()
```

	New_Dwelling_house_Apartment	Second_Hand_Dwelling_house_Apartment
0	0.0	1.0
1	0.0	1.0
2	0.0	1.0
3	0.0	1.0
4	0.0	1.0

Figure 78: preparing dataset.

After the encoding, we will drop the Description_of_property as we do not need it anymore.

```
df.drop('Description_of_property', inplace=True, axis=1)
```

After encoding the values of 'Description_of_property', the column names were added with a (,). To fix that we will rename the columns and add them to a new dataframe.

```
 13  (New_Dwelling_house_Apartment,)          46398 non-null  float64
 14  (Second_Hand_Dwelling_house_Apartment,)  46398 non-null  float64
dtypes: datetime64[ns](1), float64(5), int64(9)
memory usage: 6.9 MB
```

```
df.rename(columns={'(New_Dwelling_house_Apartment,)':'New_Dwelling_house_Apartment',
 '(Second_Hand_Dwelling_house_Apartment,)':'Second_Hand_Dwelling_house_Apartment'},
       inplace=True)
```

```
df1 = df.rename(columns={ df.columns[14]: "Second_Hand_Dwelling_house_Apartment" })
df2 = df1.rename(columns={ df1.columns[13]: "New_Dwelling_house_Apartment" })
```

```
df2.head()
```

ull_market_price	VAT_exclusive	Longitude	Latitude	New_Dwelling_house_Apartment	Second_Hand_Dwelling_house_Apartment
1	0	-6.181405	53.362710	0.0	1.0
1	0	-6.181405	53.362710	0.0	1.0
1	0	-6.182833	53.362984	0.0	1.0
1	0	-6.181405	53.362710	0.0	1.0
0	0	-6.181892	53.365139	0.0	1.0

Figure 79: Preparing dataset.

Subsequently, after having checked the info in the new dataframe, we could see some NaN values. So, we will drop it and add into a new dataframe.

```
print('New_Dwelling_house_Apartment :',df2.New_Dwelling_house_Apartment .unique())
print('Second_Hand_Dwelling_house_Apartment :',df2.Second_Hand_Dwelling_house_Apartment.unique())
```

```
New_Dwelling_house_Apartment : [ 0.  1. nan]
Second_Hand_Dwelling_house_Apartment : [ 1.  0. nan]
```

```
df3 = df2.dropna()
```

Figure 80: Cleaning data process

The NaN values are gone. As seen above the data type is float. Then we will convert that to int.

```
print('New_Dwelling_house_Apartment :',df3.New_Dwelling_house_Apartment .unique())
print('Second_Hand_Dwelling_house_Apartment :',df3.Second_Hand_Dwelling_house_Apartment.unique())
```

```
New_Dwelling_house_Apartment : [0. 1.]
Second_Hand_Dwelling_house_Apartment : [1. 0.]
```

```
ient']] = df3[['New_Dwelling_house_Apartment', 'Second_Hand_Dwelling_house_Apartment']].astype(int)
```

Figure 81: Cleaning data process

For the last, we will drop two columns 'ID' and ‘Date of Sale’ because it is not relevant for training the model.

```
In [38]: df = df.drop(columns=['ID', 'Date_of_Sale'])
```

Let us have a look at price distribution and outliers.

```
In [40]: df.Price.describe()
Out[40]: count    4.639800e+04
         mean     4.527555e+05
         std      1.451157e+06
         min      5.250000e+03
         25%      2.511013e+05
         50%      3.303965e+05
         75%      4.675000e+05
         max      1.391650e+08
         Name: Price, dtype: float64

In [41]: df.Price.min()
Out[41]: 5250.0

In [42]: df.Price.max()
Out[42]: 139165000.0
```

Figure82: Cleaning data process

It looks like we have some very odd numbers. Minimum being just over 5 thousand, it does not make any sense since no property in Dublin would cost so little. And 139 million euro is a large, unrealistic value, which by investigating those entries, it seems to be lots of apartments sold together. Let us check price in a box plot:

```
In [43]: import matplotlib.pyplot as plt

In [44]: fig = plt.figure(figsize =(10, 7))
         plt.boxplot(df.Price)
         plt.show()
```

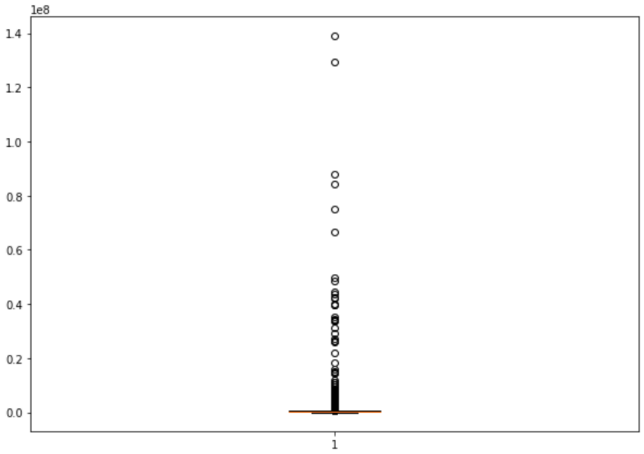

Figure 83: Cleaning data process

We can see that we have some large values and that those are outliers. Let us experiment a bit with the data:

```
In [45]: df.shape
Out[45]: (46398, 13)

In [46]: lessthan5000000 = df[df['Price'] < 5000000.0]

In [47]: lessthan5000000.shape
Out[47]: (46318, 13)
```

Figure 84: data preparation process.

Apparently only 80 rows of the 46,398 are priced over 5 million euro.

```
In [48]: fig = plt.figure(figsize =(10, 7))
         plt.boxplot(lessthan5000000.Price)
         plt.show()
```

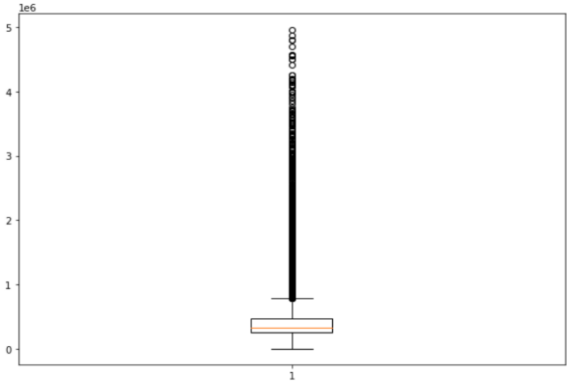

Figure 85: plot to visualise data preparation process.

The box plot looks better when we delete those, but we seem to still have a lot of outliers. Let us see if we use data for prices less than 2 million:

```
In [49]: lessthan2000000 = df[df['Price'] < 2000000.0]

In [50]: lessthan2000000.shape
Out[50]: (45943, 13)
```

Figure 86: data preparation process.

455 rows out of the 46,398 were priced above 2 million euro. Let us see the box plot:

```
In [51]: fig = plt.figure(figsize =(10, 7))
         plt.boxplot(lessthan2000000.Price)
         plt.show()
```

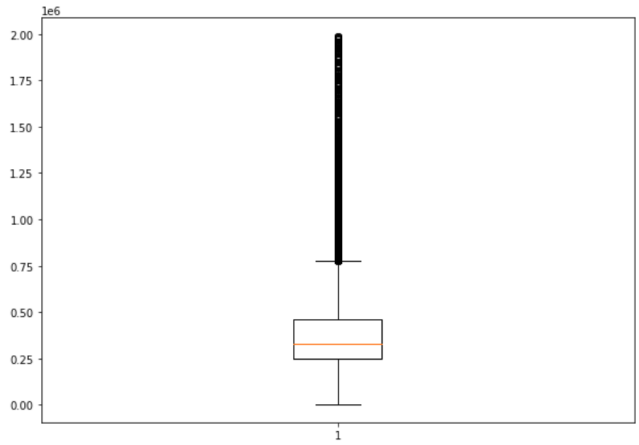

Figure 87: data preparation process.

Now we can see that there is no point separated on its own at the top, and 2 million is a realistic number of the price of a property in Dublin, they are probably real outliers, so we are going to keep this data of prices up to 2 million euro. Let us check the small values.

```
In [55]: reallysmall = df[df['Price'] < 70000.0]

In [56]: reallysmall
```

Out[56]:

	Luas_quantity	Bus_quantity	Train_quantity	PS_quantity	SS_quantity	Garda_quantity	Price	Not_full_market_price	VAT_exclusive	Longitude	Latitu
2	0	34	0	1	0	0	66000.00	1	0	-6.182833	53.362
189	0	8	0	1	0	0	25000.00	1	0	-6.094022	53.533
249	0	0	0	0	0	0	30000.00	1	0	-6.243575	53.562
360	0	19	0	2	2	0	15000.00	1	0	-6.174446	53.406
384	4	102	0	8	2	1	40000.00	0	0	-6.271994	53.334
...	...	...	...	...	...	...	...	...	...	...	...
44749	0	2	0	0	0	0	54000.00	0	0	-6.097783	53.489
44858	0	42	0	2	1	0	30000.00	1	0	-6.132789	53.253
45082	4	76	0	4	2	1	57500.00	1	0	-6.267361	53.322
45278	0	32	1	1	1	1	53000.00	0	0	-6.184652	53.610
45363	0	34	0	4	2	0	19890.92	1	0	-6.430740	53.396

414 rows × 13 columns

Figure 88: data preparation process.

There are 414 rows that have a price less than 70 thousand euro. That is a little bit unrealistic for anyone to buy a property for less than 70 thousand in Dublin. Because of that, we will keep our dataset with values from over 70 thousand to less than 2 million.

```
In [57]: df = df[df['Price'] > 70000.0]

In [58]: df = df[df['Price'] < 2000000.0]

In [59]: df.shape
Out[59]: (45515, 13)
```

Figure 89: data preparation process.

The data now has 45,515 rows. It had 46,398 before, so by selecting the range of prices we dropped 883 rows. Check the box plot of the final range:

```
In [60]: fig = plt.figure(figsize =(10, 7))
         plt.boxplot(df.Price)
         plt.show()
```

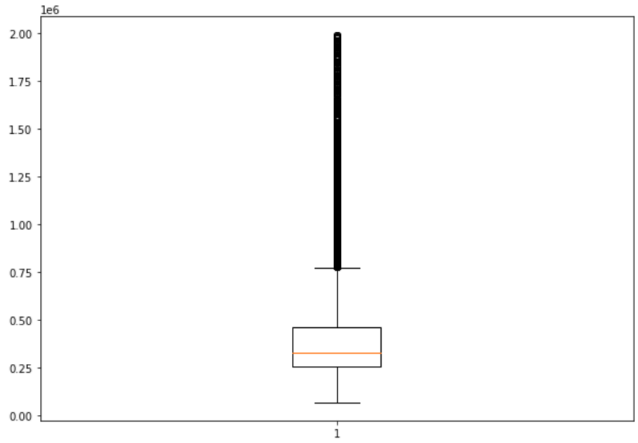

Figure 90: data preparation process.

Now let us have a look at Not_full_market_price feature:

```
In [61]: df['Not_full_market_price'].value_counts()

Out[61]: 0    43528
         1     1987
         Name: Not_full_market_price, dtype: int64

In [62]: # Pie chart, where the slices will be ordered and plotted counter-clockwise:
         labels = 'Not full price', 'Full price'
         colors = ['lightskyblue', 'lightcoral']
         sizes = [1987, 43528]
         explode = (0.1, 0)  # only "explode" the 1st slice

         fig1, ax1 = plt.subplots()
         ax1.pie(sizes, explode=explode, colors=colors, labels=labels, autopct='%1.1f%%',
                 shadow=True, startangle=30)
         ax1.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
         plt.title('Number of Properties with Full Market Price and Not Full Market Price')
         plt.show()
```

Figure 91: Visualising Not_full_market_price features, data preparation process.

We can see 4.4% of the entries are marked to be Not Full Market Price. All the explanation there was in the Property Price Register website about column 'Not Full Market Price':

In a small number of transactions included in the Register the price shown does not represent the full market price of the property concerned for a variety of reasons. All such properties are marked.

Since these not full market price does not represent the real full price, we will be keeping only full market price properties and deleting this Not_Full_Market_Price column.

```
In [63]: df2 = df[df['Not_full_market_price'] == 0]

In [64]: df2['Not_full_market_price'].value_counts()

Out[64]: 0    43528
         Name: Not_full_market_price, dtype: int64

In [65]: df2 = df2.drop(columns=['Not_full_market_price'])

In [66]: df2.shape

Out[66]: (43528, 12)
```

Figure 92: data preparation process.

Now let us have a look at new_Dwelling vs Second_Hand properties:

```
In [67]: df2['New_Dwelling_House_Apartment'].value_counts()

Out[67]: 0    33156
         1    10372
         Name: New_Dwelling_House_Apartment, dtype: int64
```

Figure 93: data preparation process.

Because these columns were encoded using one hot encoding, we know that everything that is 0 for new dwelling is second hand. So, let's see it in a Pie chart:

```
In [68]: # Pie chart, where the slices will be ordered and plotted counter-clockwise:
         labels = 'New Dwelling', 'Second Hand'
         colors = ['lightskyblue', 'lightcoral']
         sizes = [10372, 33156]
         explode = (0.1, 0)  # only "explode" the 1st slice

         fig1, ax1 = plt.subplots()
         ax1.pie(sizes, explode=explode, colors=colors, labels=labels, autopct='%1.1f%%',
                 shadow=True, startangle=30)
         ax1.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
         plt.title('Percentage of Second Hand and New Dwelling Properties')
         plt.show()
```

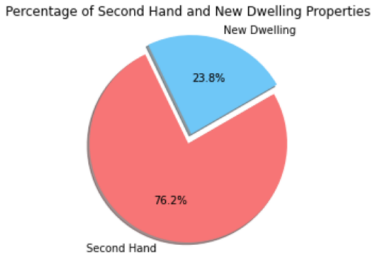

Figure 94: Visualising New_Dwelling and Second Hand features, data preparation process.

About 3/4 of our dataset was second-hand houses/apartments and 1/4 was new houses/apartments. Let us check the 'VAT Exclusive' column, the explanation in the Property Price Register:

If the property is a new property, the price shown should be exclusive of VAT at 13.5%.

By this explanation, the assumption we get is that it means all new properties are VAT Exclusive = yes and second hand are VAT exclusive = no. But let us check if this assumption is right.

```
In [69]: df2['VAT_exclusive'].value_counts()
Out[69]: 0    33082
         1    10446
         Name: VAT_exclusive, dtype: int64

In [70]: # Pie chart, where the slices will be ordered and plotted counter-clockwise:
         labels = 'VAT Exclusive', 'Not VAT Exclusive'
         colors = ['lightskyblue', 'lightcoral']
         sizes = [10446, 33082]
         explode = (0.1, 0)  # only "explode" the 1st slice

         fig1, ax1 = plt.subplots()
         ax1.pie(sizes, explode=explode, colors=colors, labels=labels, autopct='%1.1f%%',
                 shadow=True, startangle=30)
         ax1.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
         plt.title('Percentage of VAT Exclusive and Not VAT Exclusive Properties')
         plt.show()
```

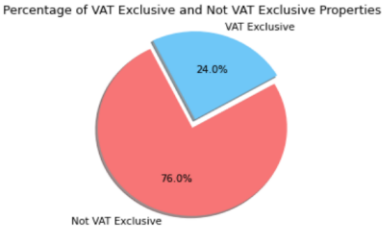

Figure 95: Visualizing Not_VAT_Exculsive and VAT Exclusive, data preparation process.

It seems there are a little bit more values for VAT Exclusive than new dwellings. So, our assumption is probably not correct, but let us look further:

```
In [71]: vat_exclusive_new_dwelling = df2[(df2['VAT_exclusive'] == 1) & (df2['New_Dwelling_House_Apartment'] == 1)]

In [72]: vat_exclusive_new_dwelling.shape
Out[72]: (5293, 12)

In [73]: vat_exclusive_second_hand = df2[(df2['VAT_exclusive'] == 1) & (df2['Second_Hand_Dwelling_House_Apartment'] == 1)]

In [74]: vat_exclusive_second_hand.shape
Out[74]: (5153, 12)
```

Figure 9613: More investigation in our data, data preparation process.

Approximately half of the VAT exclusive rows were second hand and half were new dwellings. So, it's not just all new dwellings that are vat exclusive. We cannot really understand what this column means, and not to add bias in the data we will just be leaving this column as it is.

Let us have a look at Latitude and Longitude:

```
In [75]: df2.Latitude.describe()
Out[75]: count    43528.000000
         mean        53.321151
         std          0.969011
         min        -37.857255
         25%         53.293951
         50%         53.342788
         75%         53.387985
         max         56.415157
         Name: Latitude, dtype: float64

In [76]: df2.Longitude.describe()
Out[76]: count    43528.000000
         mean        -6.388826
         std          3.555182
         min       -122.675026
         25%         -6.329320
         50%         -6.258684
         75%         -6.200129
         max        152.972980
         Name: Longitude, dtype: float64
```

Figure 97: data preparation process.

We seem to have a few wrong values. Latitude for Dublin should start with 53 and Longitude with -6. That was probably when we were passing the address to Google Maps API, it picked up the wrong coordinates. So, let us check how many are wrong.

```
In [77]: latitude_over54 = df2[(df2['Latitude'] > 54)]

In [78]: latitude_over54.shape
Out[78]: (20, 12)
```

Figure 98: data preparation process.

```
In [80]: latitude_less53 = df2[(df2['Latitude'] < 53)]

In [81]: latitude_less53.shape
Out[81]: (104, 12)

In [83]: longitude_lessminus7 = df2[(df2['Longitude'] < -7)]

In [84]: longitude_lessminus7.shape
Out[84]: (101, 12)

In [86]: longitude_greaterminus6 = df2[(df2['Longitude'] > -6)]

In [87]: longitude_greaterminus6.shape
Out[87]: (28, 12)
```

Figure 99: data preparation process.

We will drop these values since Google API did not recognize the address and returned a wrong value for Latitude and Longitude. As we can see there is no Luas, Bus, Train, Secondary school, or primary school near those entries, so they are not in Dublin.

```
In [89]: latitude_over54.describe()
```

Out[89]:

	Luas_quantity	Bus_quantity	Train_quantity	PS_quantity	SS_quantity	Garda_quantity	Price	VAT_exclusive	Longitude	Latitude	New_Dwelling
count	20.0	20.0	20.0	20.0	20.0	20.0	2.000000e+01	20.000000	20.000000	20.000000	
mean	0.0	0.0	0.0	0.0	0.0	0.0	3.936130e+05	0.100000	-5.232388	55.193121	
std	0.0	0.0	0.0	0.0	0.0	0.0	3.569133e+05	0.307794	2.176978	0.775423	
min	0.0	0.0	0.0	0.0	0.0	0.0	1.018182e+05	0.000000	-8.529977	54.051399	
25%	0.0	0.0	0.0	0.0	0.0	0.0	2.175000e+05	0.000000	-7.348944	54.701841	
50%	0.0	0.0	0.0	0.0	0.0	0.0	3.165000e+05	0.000000	-4.942970	55.009643	
75%	0.0	0.0	0.0	0.0	0.0	0.0	4.025000e+05	0.000000	-4.320116	55.949972	
max	0.0	0.0	0.0	0.0	0.0	0.0	1.690000e+06	1.000000	-1.615853	56.415157	

```
In [90]: latitude_less53.describe()
```

Out[90]:

	Luas_quantity	Bus_quantity	Train_quantity	PS_quantity	SS_quantity	Garda_quantity	Price	VAT_exclusive	Longitude	Latitude	New_Dwelli
count	104.0	104.0	104.0	104.0	104.0	104.0	1.040000e+02	104.000000	104.000000	104.000000	
mean	0.0	0.0	0.0	0.0	0.0	0.0	4.107579e+05	0.221154	-56.328222	40.038852	
std	0.0	0.0	0.0	0.0	0.0	0.0	3.239999e+05	0.417034	53.007772	14.670190	
min	0.0	0.0	0.0	0.0	0.0	0.0	1.150000e+05	0.000000	-122.675026	-37.857255	
25%	0.0	0.0	0.0	0.0	0.0	0.0	2.395000e+05	0.000000	-87.346565	38.401070	
50%	0.0	0.0	0.0	0.0	0.0	0.0	3.020950e+05	0.000000	-74.701368	40.776030	
75%	0.0	0.0	0.0	0.0	0.0	0.0	4.100000e+05	0.000000	-8.560870	49.894067	
max	0.0	0.0	0.0	0.0	0.0	0.0	1.766388e+06	1.000000	152.972980	52.994295	

Figure 100: Checking the dataset, data preparation process.

```
In [91]: longitude_lessminus7.describe()
```

Out[91]:

	Luas_quantity	Bus_quantity	Train_quantity	PS_quantity	SS_quantity	Garda_quantity	Price	VAT_exclusive	Longitude	Latitude	New_Dwelli
count	101.0	101.0	101.0	101.0	101.0	101.0	1.010000e+02	101.000000	101.000000	101.000000	
mean	0.0	0.0	0.0	0.0	0.0	0.0	4.060971e+05	0.207921	-63.547284	42.651330	
std	0.0	0.0	0.0	0.0	0.0	0.0	3.355176e+05	0.407844	38.105272	8.030063	
min	0.0	0.0	0.0	0.0	0.0	0.0	9.500000e+04	0.000000	-122.675026	25.863214	
25%	0.0	0.0	0.0	0.0	0.0	0.0	2.150000e+05	0.000000	-87.788719	39.083997	
50%	0.0	0.0	0.0	0.0	0.0	0.0	2.970000e+05	0.000000	-75.118187	40.844985	
75%	0.0	0.0	0.0	0.0	0.0	0.0	4.317180e+05	0.000000	-9.381351	52.682823	
max	0.0	0.0	0.0	0.0	0.0	0.0	1.766388e+06	1.000000	-7.312264	55.135019	

```
In [92]: longitude_greaterminus6.describe()
```

Out[92]:

	Luas_quantity	Bus_quantity	Train_quantity	PS_quantity	SS_quantity	Garda_quantity	Price	VAT_exclusive	Longitude	Latitude	New_Dwellir
count	28.0	28.0	28.0	28.0	28.0	28.0	2.800000e+01	28.000000	28.000000	28.000000	
mean	0.0	0.0	0.0	0.0	0.0	0.0	4.305264e+05	0.178571	14.595326	44.037333	
std	0.0	0.0	0.0	0.0	0.0	0.0	3.247776e+05	0.390021	47.775835	27.371986	
min	0.0	0.0	0.0	0.0	0.0	0.0	1.800000e+05	0.000000	-5.945565	-37.857255	
25%	0.0	0.0	0.0	0.0	0.0	0.0	2.592500e+05	0.000000	-4.320162	49.894067	
50%	0.0	0.0	0.0	0.0	0.0	0.0	3.200000e+05	0.000000	-1.616687	53.041960	
75%	0.0	0.0	0.0	0.0	0.0	0.0	4.125000e+05	0.000000	2.295753	55.239455	
max	0.0	0.0	0.0	0.0	0.0	0.0	1.690000e+06	1.000000	152.972980	56.415157	

Figure 101: Checking the dataset, data preparation process.

All these Latitudes and Longitudes are wrong, so we will be dropping these entries.

```
In [93]: df3 = df2[(df2['Latitude'] > 53)]

In [94]: df3.shape
Out[94]: (43424, 12)

In [95]: df3.Latitude.describe()
Out[95]: count    43424.000000
         mean        53.352962
         std          0.084828
         min         53.016029
         25%         53.294227
         50%         53.342892
         75%         53.388046
         max         56.415157
         Name: Latitude, dtype: float64

In [96]: df4 = df3[(df3['Latitude'] < 54)]

In [97]: df4.shape
Out[97]: (43404, 12)

In [98]: df4.Latitude.describe()
Out[98]: count    43404.000000
         mean        53.352114
         std          0.073313
         min         53.016029
         25%         53.294198
         50%         53.342823
         75%         53.388007
         max         53.968158
         Name: Latitude, dtype: float64

In [99]: df5 = df4[(df4['Longitude'] > -7)]

In [100]: df5.shape
Out[100]: (43393, 12)

In [101]: df5.Longitude.describe()
Out[101]: count    43393.000000
          mean        -6.269044
          std          0.113424
          min         -6.908180
          25%         -6.328327
          50%         -6.258499
          75%         -6.200048
          max          6.440954
          Name: Longitude, dtype: float64

In [102]: df6 = df5[(df5['Longitude'] < -6)]

In [103]: df6.shape
Out[103]: (43392, 12)
```

Figure 102: Checking the dataset, data preparation process.

Comparing number of rows before and after deleting the wrong number for latitude and longitude.

```
In [105]: df2.shape
Out[105]: (43528, 12)

In [106]: df6.shape
Out[106]: (43392, 12)
```

Figure 103: Checking the dataset, data preparation process.

We will visualize to see how the latitude and longitude look after the changes done in the dataframe.

```
1 plt.figure(figsize=(10, 5))
2 df6.plot(kind="scatter", x="Longitude", y="Latitude", alpha=0.9)
3 plt.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mappi
ng will have precedence in case its length matches with *x* & *y*.  Please use the *color* keyword-ar
gument or provide a 2-D array with a single row if you intend to specify the same RGB or RGBA value f
or all points.
```

```
<Figure size 720x360 with 0 Axes>
```

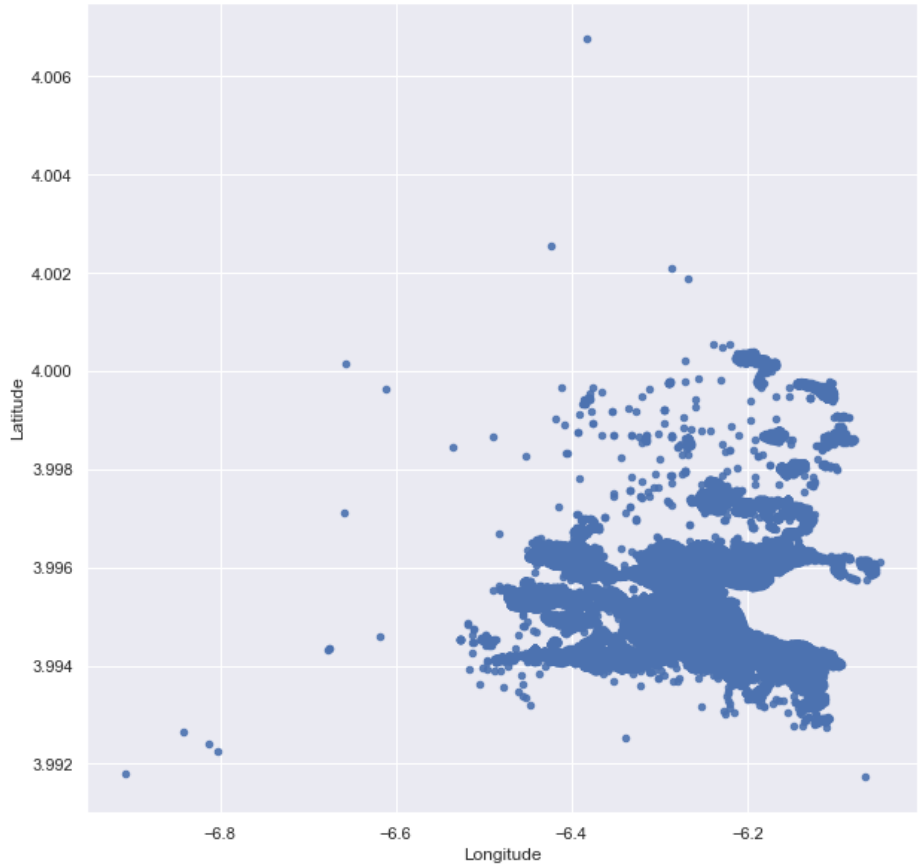

Figure 104: Checking the dataset, visualising changes after cleaning, data preparation process.

In overall, we can see that there were 130 rows deleted. Let us see the final dataset:

```
In [107]: df6.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 43392 entries, 4 to 46400
Data columns (total 12 columns):
 #   Column                                Non-Null Count  Dtype
---  ------                                --------------  -----
 0   Luas_quantity                         43392 non-null  int64
 1   Bus_quantity                          43392 non-null  int64
 2   Train_quantity                        43392 non-null  int64
 3   PS_quantity                           43392 non-null  int64
 4   SS_quantity                           43392 non-null  int64
 5   Garda_quantity                        43392 non-null  int64
 6   Price                                 43392 non-null  float64
 7   VAT_exclusive                         43392 non-null  int64
 8   Longitude                             43392 non-null  float64
 9   Latitude                              43392 non-null  float64
 10  New_Dwelling_House_Apartment          43392 non-null  int64
 11  Second_Hand_Dwelling_House_Apartment  43392 non-null  int64
dtypes: float64(3), int64(9)
memory usage: 4.3 MB
```

```
In [108]: df6.describe()
```

Out[108]:

	Luas_quantity	Bus_quantity	Train_quantity	PS_quantity	SS_quantity	Garda_quantity	Price	VAT_exclusive	Longitude	Latitude	Nev
count	43392.000000	43392.000000	43392.000000	43392.000000	43392.000000	43392.000000	4.339200e+04	43392.000000	43392.000000	43392.000000	
mean	1.653231	52.232393	0.413555	3.232117	1.424364	0.397078	4.061754e+05	0.240136	-6.269336	53.352115	
std	3.521235	46.834973	0.744732	2.330548	1.245450	0.650880	2.432843e+05	0.427171	0.095615	0.073166	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	7.070000e+04	0.000000	-6.908180	53.149884	
25%	0.000000	25.000000	0.000000	2.000000	0.000000	0.000000	2.599119e+05	0.000000	-6.328328	53.294220	
50%	0.000000	43.000000	0.000000	3.000000	1.000000	0.000000	3.360000e+05	0.000000	-6.258501	53.342823	
75%	2.000000	62.000000	1.000000	4.000000	2.000000	1.000000	4.713656e+05	0.000000	-6.200054	53.388007	
max	22.000000	386.000000	5.000000	13.000000	7.000000	5.000000	1.989282e+06	1.000000	-6.048917	53.968158	

Figure 105: Checking the dataset, data preparation process.

Now, let's plot the relationship of our label Price in relation with the features Luas_quantity, Bus_quantity and Train_quantitty, Garda_quantity, PS_quantitity and SS_quantity.

```
1 plt.figure(figsize=(20, 5))
2 sns.pairplot(df6, x_vars=['Luas_quantity', 'Bus_quantity', 'Train_quantity'], y_vars='Price', size
```

```
<seaborn.axisgrid.PairGrid at 0x2046ec2f580>

<Figure size 1440x360 with 0 Axes>
```

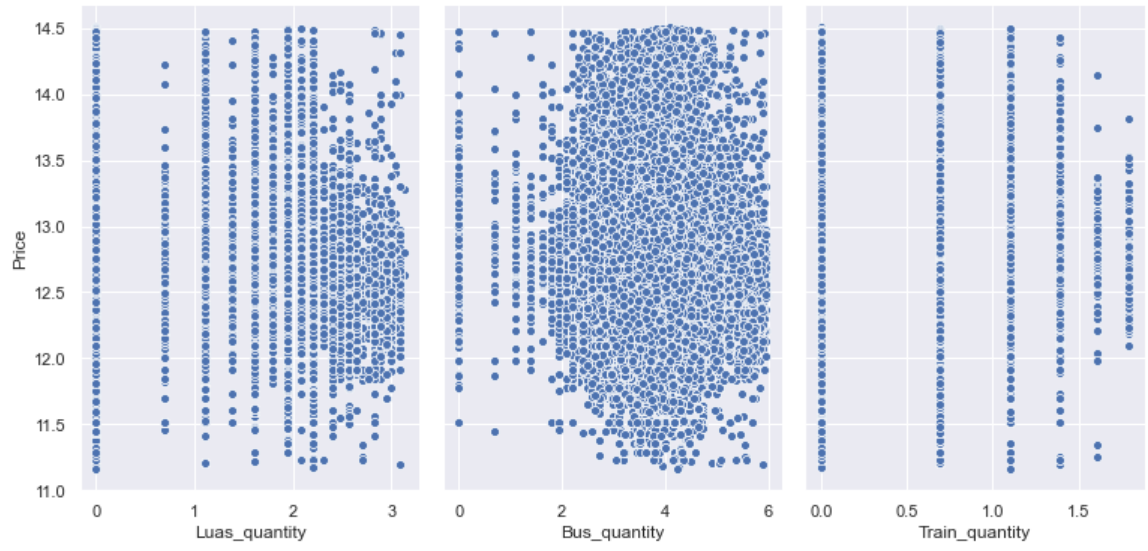

Figure 106: Checking the dataset, visualising changes after cleaning, data preparation process.

```
plt.figure(figsize=(20, 5))
sns.pairplot(df6, x_vars=['Garda_quantity','PS_quantity','SS_quantity'], y_vars='Price', size=5, a
```

```
<seaborn.axisgrid.PairGrid at 0x20470f06220>

<Figure size 1440x360 with 0 Axes>
```

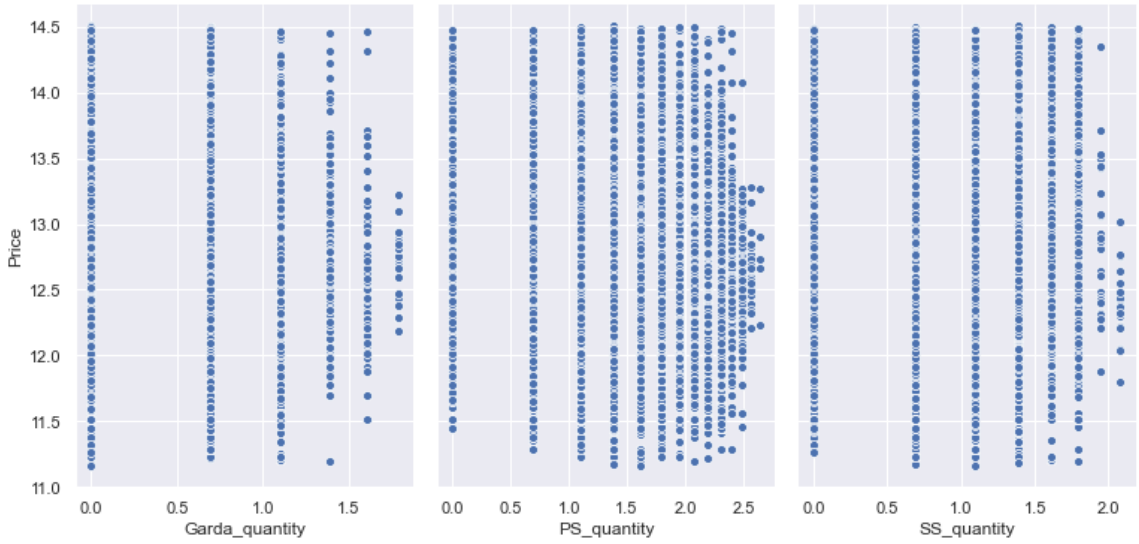

Figure 107: Checking the dataset, visualising changes after cleaning, data preparation process.

We can see that the outliers are gone. The final dataset has 12 columns and 43,392 rows. It contains the sale of houses in Dublin from 2017 to 2019 as registered in the Property Price Register and it contains the price of the houses if they were new or second hand and VAT exclusive or not. We got the latitude and longitude for all these houses and checked how many Luas, Bus stops, Train stops, Primary and Secondary schools and garda stations there were around 1km from each house.

12. Modelling

12.1 Select Machine Learning modelling techniques.

Machine learning is a subfield of artificial intelligence, in which an algorithm extracts patterns from data, in essence the concept of Machine learning is about computers making sense of data in a similar way as humans do (Kirk, 2017).

There are three types of Machine learning, supervised learning, unsupervised learning and reinforcement learning.

To predict the property prices in Dublin we will use supervised learning. It is a type of function approximation process, where we can fit the data previously collected into different types of functions to get the prediction (Kirk, 2017).

Machine learning is a very efficient way to solve problems with its different types of algorithms, it is divided in classification and regression. For our current problem we

will use regression because we are trying to predict continuous values. In this direction we will test Decision Trees, SVR Support Vector Regression, Random Forest and Linear Regression algorithm, in order to evaluate which process works best for training our model, given the highest accuracy found.

We will have a quick understanding about the Models the team will test before deciding which one would be more efficient.

12.1.1 Decision Trees

Decision Trees are used for grouping and regression; it divides the dataset into smaller subsets and shrinks the trees by converting certain branch nodes to leaf nodes, each branch representing a possible reaction. Decision Trees resemble a flowchart, beginning with a dataset and dividing it until all the data belongs to the same class. (Sehra, 2018)

Simple Example of a Decision Tree:

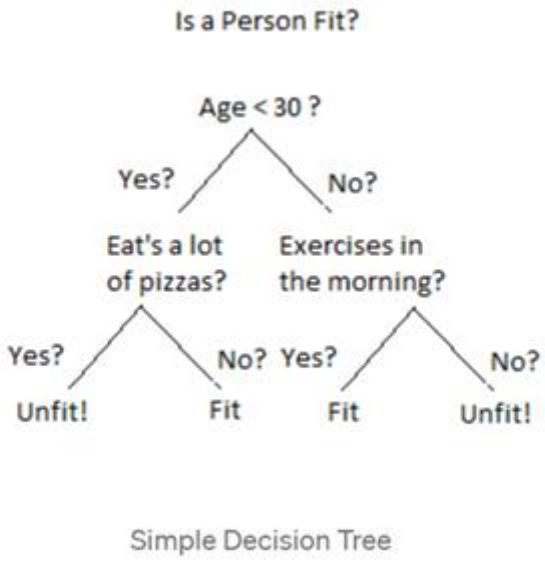

Figure 108: Example of a Decision Tree.

A benefit of using Decision Tree is that it is very straightforward and simple to explain, and it does not necessitate a lot of data planning during the process.

One downside is that it is possible to trigger instability; even minor changes in the data will result in significant changes in the nature of the decisions. (Dhiraj K, 2019)

12.1.2 Random Forest

Random Forest will be putting it to test since it is a Supervised learning algorithm that can be used to solve Classification and Regression problems.

The name Random Forest is derived from the concepts of randomization (Random) and multiple decision trees (Build multiple Decision Trees) (Forest). Random Forest is based on the Bagging method, which aids in the performance of the algorithm. Random Forest is a fast, reliable, and powerful feature importance indicator that outperforms linear algorithms.

Explaining it in a simple way: "random forest builds multiple decision trees and merges them together to get a more accurate and stable prediction" (Donges, 2020).

Because of the size of our dataset, Random Forest is also a good option. Overfitting is a common problem in machine learning, and Random Forest increases error and bias by reducing variance and adding more trees to train different data samples.

However, it is a powerful model that is important to keep in mind, since we need to store information from hundreds of individual trees, Random Forest needs more memory for storage (Lyashenko, 2021).

12.1.3 Linear Regression

A collection of techniques for estimating relationships is known as regression. For example, a relationship between house prices and the location.

Linear Regression is used to predict real values, for instance cost of a property, using continuous variables. By fitting the best line, it can be established a relationship between the independent and dependent variables. A regression line is the best fit line (Abdi, 2016).

It is expressed by a linear equation **Y= m*X + c**.

Where **Y** means the dependent variable, **m** the slope, **X** the independent variable and **c** the intercept. In this case Y and X are the data features and m and b are the values we can adjust for our prediction.

In case there are more than two variables it is called multiple linear regression. For instance, our problem to predict the property prices contains multiple features such as price, which is the dependent variable and schools nearby, buses and luas stop around and etc., which are the independent variables.

The linear regression with multiple variables is expressed with the equation **Y = c0 + m1c1 + m2c2 + m3c3 + … … mncn.**

The linear regression algorithm works by fitting multiple lines on the data points and returning the line with the least amount of error.

12.1.4 Support Vector Machine (SVM)

Support vector machines are a powerful and flexible type of supervised algorithm. It produces significant accuracy with less computation power. Support Vector Machine, abbreviated as SVM can be used for both regression and classification tasks, however, mainly for classification (Gandhi, 2018). SVM supports linear and nonlinear regression that can be referred to as Support Vector Regression.

The support vector machine algorithm's goal is to identify a line or curve (in two dimensions) or a hyperplane in an N-dimensional space (N — the number of features) that distinguishes between data points, classifying the data points.

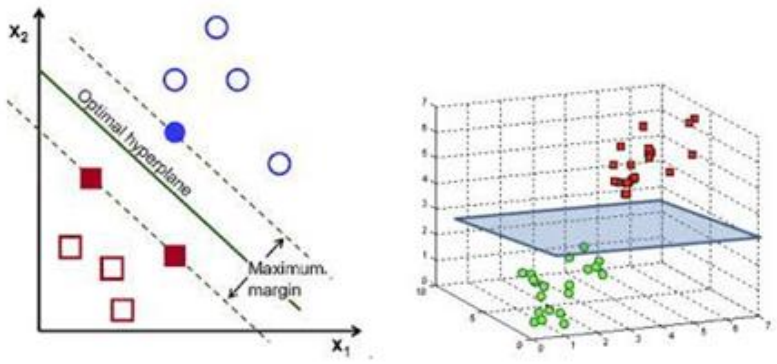

Figure 109: Hyperplanes in 2D and 3D feature space

In Support Vector Regression, the hyperplane is the line between the decision boundaries that is used to predict the continuous output (Cross Validated, 2017).

A decision boundary is a demarcation line that separates positive and negative examples on one side. The examples can be categorized as positive or negative along this axis. Support Vector Regression would use the same SVM definition (Cross Validated, 2017).

A kernel aids in the search for a hyperplane in higher-dimensional space while reducing the computational cost (Cross Validated, 2017). As the size of the data becomes larger, the computing cost usually rises. Based on a training set, the problem of regression is to find a function that approximates mapping from an input domain to real numbers.

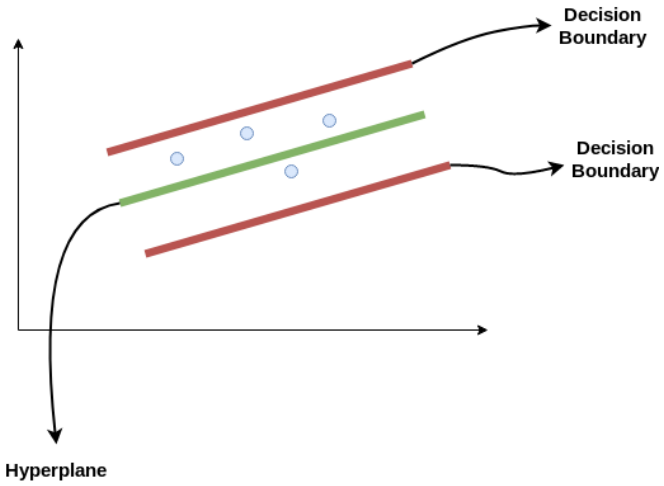

Figure 110: Hyperplanes with decision boundaries

Looking at the picture above, SVR considers the points which are within the decision boundary line. Having as the best fit line the hyperplane with the greatest number of points. In other words, instead of fitting the largest path between two classes while limiting margin violations, SVR works the other way around, trying to fit as many instances as possible in between the decision boundaries while limiting margin violations.

In SVR each data point in the training represents its own dimension. After the kernel is evaluated between a test point and a point in the training set, the resulting value gives the coordinate of the test point in that specific dimension. The vector found when evaluating the test point for all points in the training set, k is the representation of the point in the higher dimensional space. This is the vector to be used to perform the regression.

12.1.5 XG Boost (Gradient Boosting)

XGBoost is a highly effective, scalable, and portable distributed gradient boosting library. It is a gradient boosting-based ensemble supervised Learning algorithm based on decision trees. XGBoost is a parallel tree boosting algorithm also known as GBDT or GMB that solves a wide range of data science problems quickly and accurately (XGBoost, 2021), such as prediction problems involving unstructured data, outperforming all other algorithms or systems in this regard in many machine learning algorithms competitions (Morde, 2019).

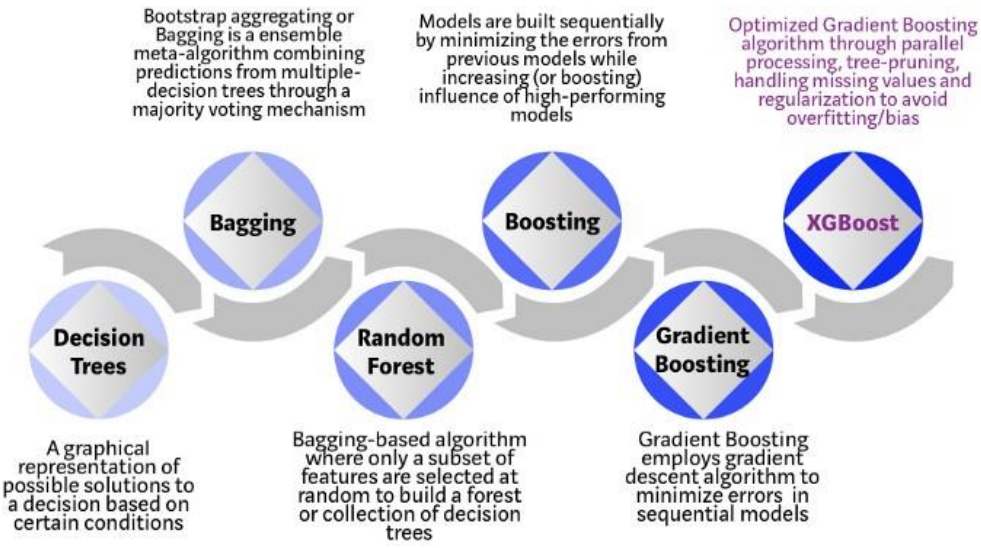

Figure 111: Evolution of XGBoost Algorithm from Decision Trees

This popular ML library was first built as a C++ command line application and later on adapted within the ML comunity (Brownlee, 2016). Sequentially to that, some of the C++ functions were implemented in other languages, such as Python which is the language used to build this prediction model. For having enhancements like out-of-core computing, block structure, continued training, regularization, sparsity awareness, weighted quantile sketch, built-in cross-validation, and parallelizable core, XGBoost speed and performance became outstanding which enabled this core to harness all the processing power of modern multi-core computers. It can be used to train models on small, medium, and large structured or tabular data, achieving great performances in many ML tasks (Brownlee, 2016).

This model combines the predictions of a series of simpler, weaker models to attempt to accurately predict a target variable. In regression each regression tree maps its leaves that have a continuous score to an input data point. XGBoost incorporates a convex loss function with a penalty term for model complexity to minimize a regularized (L1 and L2) objective function. After that, the training is repeated in a loop, with new added trees to predict the remaining errors of previous trees, which are then integrated with previous trees to produce a desired prediction. Gradient boosting gets its name from the fact that it uses a gradient descent algorithm to mitigate loss while inserting new models (AWS, 2021).

12.2 Generate test design

The first step to generate our test design is to do our feature selection. Before starting building and training our model we need to first separate our data, what labels we want to use for the prediction, and which label we want to predict (our X and y). In this case we want to predict the price of the property, so that will be our y and all the other labels will be our X. The objective of this is to only choose the most important features for better accuracy (Bonnacorso, 2017, pp. 44). Of course, we must go back to this step during the process of testing as we need to try different features to create a better prediction.

We selected all features for our first test design as wanted to test how it would perform with our complete data, we only dropped the “Unamed 0:” column as it was created during our feature engineering process. Our prediction variables were: 'Luas_quantity', 'Bus_quantity', 'Train_quantity', 'Second_Hand_Dwelling_House_Apartment', 'New_Dwelling_House_Apartment', 'PS_quantity', 'SS_quantity', 'Garda_quantity', 'VAT_exclusive', 'Longitude', 'Latitude'. and the label that we wanted to predict was ‘Price’.

```
X = df_final[prediction_variables]
y = df_final["Price"]
```

Figure 112: Split the features.

After selecting all the labels, we want to work with, we need to separate our dataset into test and training. This is important as we need to check how accurate the model is so we split the dataset into two, we can then train the model into one part of it (train) and then test into a smaller part of the data (test). Doing that it will shuffle our dataset, so it does not have an order that affects the learning (Bonnacorso, 2017, pp. 45-46). We separated our data into 80% train and 20% test.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=20)
```

Figure 113: Split data for testing and train.

Depending on the model we do need to change a few things in our data, scaling for example, but this will be talked about and done for each model. For now, our data is ready to be tested.

12.3 Build model.

We decided on testing many algorithms when building our model to check which ones would perform the best as we were not 100% sure and during the testing phase is where surprises happen. We decided testing on a variety of algorithms:

- Linear Regression.

- Ridge.
- Lasso
- Bayesian Ridge.
- SVR.
- Random Forests.
- Decision Trees.
- Gradient Boosting and Histogram Gradient Boosting.
- Stacking Regression.
- Voting Regression.
- XGBoost.
- LightGBM.

12.3.1 Linear Models

We first built our linear models, we started with linear regression but ended up adding extra models (Ridge, Lasso and Bayesian Ridge) just for the purpose of curiosity and testing. But as our data does not look very linear, we thought it would not perform well and because they are all linear models they would perform similarly.

Linear Regression

```
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

```
LinearRegression()
```

```
y_pred = regressor.predict(X_test)
```

Figure 114: Linear Regression Model.

Ridge

```
ridge = Ridge(alpha = 0.5)
ridge.fit(X_train, y_train)
pred_ridge = ridge.predict(X_test)
```

Figure 115: Ridge Model.

Bayesian Ridge

```
bayesian = BayesianRidge()
bayesian.fit(X_train, y_train)
pred_bayesian = bayesian.predict(X_test)
```

Figure 116: Bayesian Ridge Model.

Lasso

```
lasso = Lasso(alpha = 0.01)
lasso.fit(X_train, y_train)
pred_lasso = lasso.predict(X_test)
```

Figure 117: Lasso Model.

12.3.2 Support Vector Machines

For our SVR model, we had to create a pipeline for scaling our data. We were not 100% sure if this was done correctly, but this was our final try.

SVR

```
from sklearn.svm import SVR
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler

n_samples, n_features = 10, 5
rng = np.random.RandomState(0)
regr = make_pipeline(StandardScaler(), SVR(C=1.0, epsilon=0.2))
regr.fit(X_test, y_test)
```

```
Pipeline(steps=[('standardscaler', StandardScaler()),
                ('svr', SVR(epsilon=0.2))])
```

Figure 118: SVR Model.

12.3.3 Decision Trees and Random Forests

Random forests and decision trees are straightforward to work with and easy to implement, we only had to mix with their parameters but nothing much.

Random Forests

```
from sklearn.ensemble import RandomForestRegressor
regressor = RandomForestRegressor(n_estimators=100, random_state=2)
regressor.fit(X_train, y_train)
```

```
random_predict = regressor.predict(X_test)
```

Figure 119: Random Forest.

Decision Trees

```
from sklearn.tree import DecisionTreeRegressor
regressor = DecisionTreeRegressor(random_state = 0)
regressor.fit(X_train, y_train)

pred_decision_trees = regressor.predict(X_test)
```

Figure 120: Decision Tree Model.

12.3.4 Gradient Boosting:

We decided to train 5 types of gradient boosting models, starting with the simplest one (Gradient Boosting/Hist Gradient Boosting) to using libraries like XGBoost, LightGBM and Catboost that give better accuracy.

Gradient Boosting

```
from sklearn.datasets import make_hastie_10_2
from sklearn.ensemble import GradientBoostingRegressor
```

```
clf = GradientBoostingRegressor(n_estimators=25, learning_rate=0.6, max_depth=5, random_state=0).fit(X_train, y_train)
```

```
pred_GB = clf.predict(X_test)
```

Figure 121: Gradient Boosting Model.

Hist Gradient Boosting

```
>>> # explicitly require this experimental feature
>>> from sklearn.experimental import enable_hist_gradient_boosting  # noqa
>>> # now you can import normally from ensemble
>>> from sklearn.ensemble import HistGradientBoostingRegressor
```

```
hgb = HistGradientBoostingRegressor(max_depth = 10, learning_rate = 0.4,random_state=2, l2_regularization = 0.5).fit(X_train, y_t
pred_HGB = hgb.predict(X_test)
```

Figure 122: Hist Gradient Boosting.

XGBoost

```
import xgboost as xgb

xg_reg = xgb.XGBRegressor(learning_rate = 0.1, n_estimators = 40,
                          max_depth=14, min_child_weight=2,
                          gamma = 1, subsample = 0.7,
                          objective="reg:squarederror",
                          seed=1)

xg_reg.fit(X_train,y_train)

pred_XGB = xg_reg.predict(X_test)
```

Figure 123: XGBoost Model.

Catboost

```
from catboost import CatBoostRegressor

model = CatBoostRegressor(iterations = 100,
                          learning_rate=0.8,
                          depth=7)

model.fit(X_train, y_train)

pred_catboost = model.predict(X_test)
```

Figure 124: CatBoost Model.

LightGBM ¶

```
import lightgbm as lgb

model_lgb = lgb.LGBMRegressor(objective='regression', num_leaves=7,
                              learning_rate=0.6, n_estimators=2000,
                              random_state = 2)

model_lgb.fit(X_train, y_train)
lgb_train_pred = model_lgb.predict(X_test)
```

Figure 115: LightGBM Model.

12.3.5 Extra Models

We also tested some extra models like Stacking and Voting which are both ensemble algorithms (just like Random Trees and Gradient Boosting).

Voting Regression

```
from sklearn.ensemble import VotingRegressor

lr = LinearRegression()
forest = RandomForestRegressor(n_estimators=100, random_state=1)
hgb = HistGradientBoostingRegressor(max_depth = 10, learning_rate = 0.4,random_state=2, l2_regularization = 0.5)
clf = GradientBoostingRegressor(n_estimators=25, learning_rate=0.6, max_depth=5, random_state=0)

er = VotingRegressor([('lr', lr), ('rf', forest), ('hgb',hgb), ('clf',clf)])

er.fit(X_train,y_train)

pred_voting = er.predict(X_test)
```

Figure 126: Voting Regression Model.

Stacking ¶

```
from sklearn.ensemble import StackingRegressor

estimators = [
('lr', LinearRegression()),
('clf',  GradientBoostingRegressor(n_estimators=25, learning_rate=0.6, max_depth=5))
]
reg = StackingRegressor(
     estimators=estimators,
    final_estimator=RandomForestRegressor(n_estimators=100,
                                          random_state=42)
)

reg.fit(X_train, y_train)
```

```
pred_stack = reg.predict(X_test)
```

Figure 116: Stacking Model.

12.4 Assess Model

After training all our models and testing our predictions, we need to assess which one is the model that performs best from all of them. There are many ways to assess your models and depending on which type is choses different metrics and scoring methods are needed to evaluate them. As we are working with regression, we will need metrics functions to assess our models (Scikit-learn, 2021).

We decided to use explained variance score and r2 score as we have big values to work with, so the difference between prediction and real value might be quite big, which means that using a Mean Absolute Error approach for example, might be hard to read and have an idea of how the model is doing.

12.4.1 Explained Variance Score

The Scikit-learn (2021) website says that "*If y^ is the estimated target output, y the corresponding (correct) target output, and Var is Variance, the square of the standard deviation, then the explained variance is estimated as follow:* "

$$explained\_variance(y, \hat{y}) = 1 - \frac{Var\{y - \hat{y}\}}{Var\{y\}}$$

This means that 1 minus the Variance of the difference correct value and our predicted value, divided by the variance of our correct value will give a result between 0.00 and 1.00, being that 1.00 is a perfect score and 0.00 is our worst possible score.

12.4.2 R2 Score

The r2 score has a similar approach in visualizing our results with 1.00 being our best possible value, but it can go negative differently from the Explained Variance Score. It tries to score how much the variables affect the variance of our predicted variable (Scikit-learn, 2021).

The Scikit-learn (2021) website says that "*If y^i is the predicted value of the i-th sample and yi is the corresponding true value for total n samples, the estimated R² is defined as:*

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

where y¯=1n∑i=1nyi and ∑i=1n(yi−y^i)2=∑i=1nϵi2."

13. Evaluation

Now that we prepared all of our models and decided how we would evaluate all of them, it is time to start checking how well each of them performed. We separated them again into types of models and checked how well they performed.

13.1 Evaluate Results

13.1.1 Linear Models Results

As we were expecting from our linear models, the accuracy was quite low as there is not really a linear relationship between the features with a EVS and R2 score of 0.15. All models had a similar score as they were all linear.

Linear Regression Results

```
print('Explained Variance Score of Linear Regression model is {}'.format(evs(y_test, y_pred)))
print('R-Squared of Linear Regression model is {}'.format(r2(y_test, y_pred)))
```

```
Explained Variance Score of Linear Regression model is 0.1572285332768133
R-Squared of Linear Regression model is 0.15721062966250354
```

Figure 128: Linear Regression Results.

Ridge Results

```
print('Explained Variance Score of Ridge model is {}'.format(evs(y_test, pred_ridge)))
print('R-Squared of Ridge model is {}'.format(r2(y_test, pred_ridge)))
```

```
Explained Variance Score of Ridge model is 0.157214107766802
R-Squared of Ridge model is 0.15719620016330604
```

Figure 117: Ridge Results.

Lasso Results

```
print('Explained Variance Score of Lasso model is {}'.format(evs(y_test, pred_lasso)))
print('R-Squared of Lasso model is {}'.format(r2(y_test, pred_lasso)))
```

```
Explained Variance Score of Lasso model is 0.1572285268568372
R-Squared of Lasso model is 0.15721062327214175
```

Figure 118: Lasso Results

Bayesian Ridge Results

```
print('Explained Variance Score of Bayesian model is {}'.format(evs(y_test, pred_bayesian)))
print('R-Squared of Bayesian model is {}'.format(r2(y_test, pred_bayesian)))
```

```
Explained Variance Score of Bayesian model is 0.15721796057168347
R-Squared of Bayesian model is 0.15720005396540182
```

Figure 131: Bayesian Ridge Results.

13.1.2 Support Vector Machine

For our SVR model, we think that something went wrong when doing the pipeline and scaling, so our results were not great at all with an EVS of 0.0005 and an R2 score of -0.08.

SVR Results

```
print('Explained Variance Score of SVR model is {}'.format(evs(y_test, pred_svr)))
print('R-Squared of SVR model is {}'.format(r2(y_test, pred_svr)))
```

```
Explained Variance Score of SVR model is 0.0005366174713297145
R-Squared of SVR model is -0.08295277285076086
```

Figure 132: SVR Results.

13.1.3 Decision Trees and Random Forests Results

These two types of models started giving us better results, we had to tune their parameters to get better results, but they were much more efficient than our other models. The decision trees model had an EVS and an R2 score of 0.33 while the random forests had a much better result scoring 0.58 for both EVS and R2.

Decision Trees Results

```
print('Explained Variance Score of Decision Trees model is {}'.format(evs(y_test,pred_decision_trees)))
print('R-Squared of Decision Trees model is {}'.format(r2(y_test, pred_decision_trees)))
```

```
Explained Variance Score of Decision Trees model is 0.3332395965295819
R-Squared of Decision Trees model is 0.3324537598967301
```

Figure 133: Decision Trees Results.

Random Forests Results

```
print('Explained Variance Score of Random Forest Tree model is {}'.format(evs(y_test, random_predict)))
print('R-Squared of Random Forest Tree model is {}'.format(r2(y_test, random_predict)))
```

```
Explained Variance Score of Random Forest Tree model is 0.5827743074912873
R-Squared of Random Forest Tree model is 0.5825285472633464
```

Figure 134: Random Forest Results

13.1.4 Gradient Boosting Results

As for our gradient boosting models, we had good results, ranging from 0.40 to almost 0.57. We had to tweak a lot of the parameters to get the best results we could, and we managed to increase most of them, XGBoost was especially great with a 0.56 score.

Hist Gradient Boosting Results

```
print('Explained Variance Score of Gradient Boosting model is {}'.format(evs(y_test,pred_HGB)))
print('R-Squared of Gradient Boosting model is {}'.format(r2(y_test, pred_HGB)))
```

```
Explained Variance Score of Gradient Boosting model is 0.49959483463300247
R-Squared of Gradient Boosting model is 0.499496912029446
```

Figure 135: Hist Gradient Boosting Results.

Gradient Boosting Results

```
print('Explained Variance Score of Gradient Boosting model is {}'.format(evs(y_test,pred_GB)))
print('R-Squared of Gradient Boosting model is {}'.format(r2(y_test, pred_GB)))
```

```
Explained Variance Score of Gradient Boosting model is 0.43979002211754326
R-Squared of Gradient Boosting model is 0.43976830296062275
```

Figure 136: Gradient Boosting Results.

XGBoost Results

```
print('Explained Variance Score of XGBoost model is {}'.format(evs(y_test, pred_XGB)))
print('R-Squared of XGBoost model is {}'.format(r2(y_test, pred_XGB)))
```

```
Explained Variance Score of XGBoost model is 0.5681369302837654
R-Squared of XGBoost model is 0.5668303539542203
```

Figure 137: XGBoost Results.

CatBoost Results

```
print('Explained Variance Score of Catboost model is {}'.format(evs(y_test,pred_catboost)))
print('R-Squared of Catboost model is {}'.format(r2(y_test, pred_catboost)))
```

```
Explained Variance Score of Catboost model is 0.47316430793632125
R-Squared of Catboost model is 0.4731559226270149
```

Figure 138: CatBoost Results.

LightGBM Results

```
print('Explained Variance Score of Gradient Boosting model is {}'.format(evs(y_test,lgb_train_pred)))
print('R-Squared of Gradient Boosting model is {}'.format(r2(y_test, lgb_train_pred)))
```

```
Explained Variance Score of Gradient Boosting model is 0.5012842442125294
R-Squared of Gradient Boosting model is 0.5012830462900238
```

Figure 19: LightBM Results

15.1.5 Extra Models

The extra ensemble models that we tested performed well, but we did not go deep on them with tweaking parameters and different models used in them. We did not get many different results, so we ended up not testing them anymore, the Voting Regressor performed quite well with an EVS and R2 of 0.51 while the Stacking Regressor did not perform as well with a EVS and R2 score of 0.34.

Stacking Results

```
print('Explained Variance Score of Stacking model is {}'.format(evs(y_test,pred_stack)))
print('R-Squared of Stacking model is {}'.format(r2(y_test, pred_stack)))
```

```
Explained Variance Score of Stacking model is 0.3432575073399349
R-Squared of Stacking model is 0.3414730045867128
```

Figure 140: Stacking Results.

Voting Results

```
print('Explained Variance Score of Voting model is {}'.format(evs(y_test,pred_voting)))
print('R-Squared of Voting model is {}'.format(r2(y_test, pred_voting)))
```

```
Explained Variance Score of Voting model is 0.5146589420321566
R-Squared of Voting model is 0.5146584517646812
```

Figure 141: Voting Results.

13.2 Review process.

Now that all the models were assessed we can select which would be our best option. As we saw above our top 3 models were:

- Random Forests - EVS and R2: 0.58.
- XGBoost - EVS and R2: 0.56
- Voting Regressor - EVS and R2: 0.51

You can see below with a cross validation plot, how well the random forests model performed against the linear regression one.

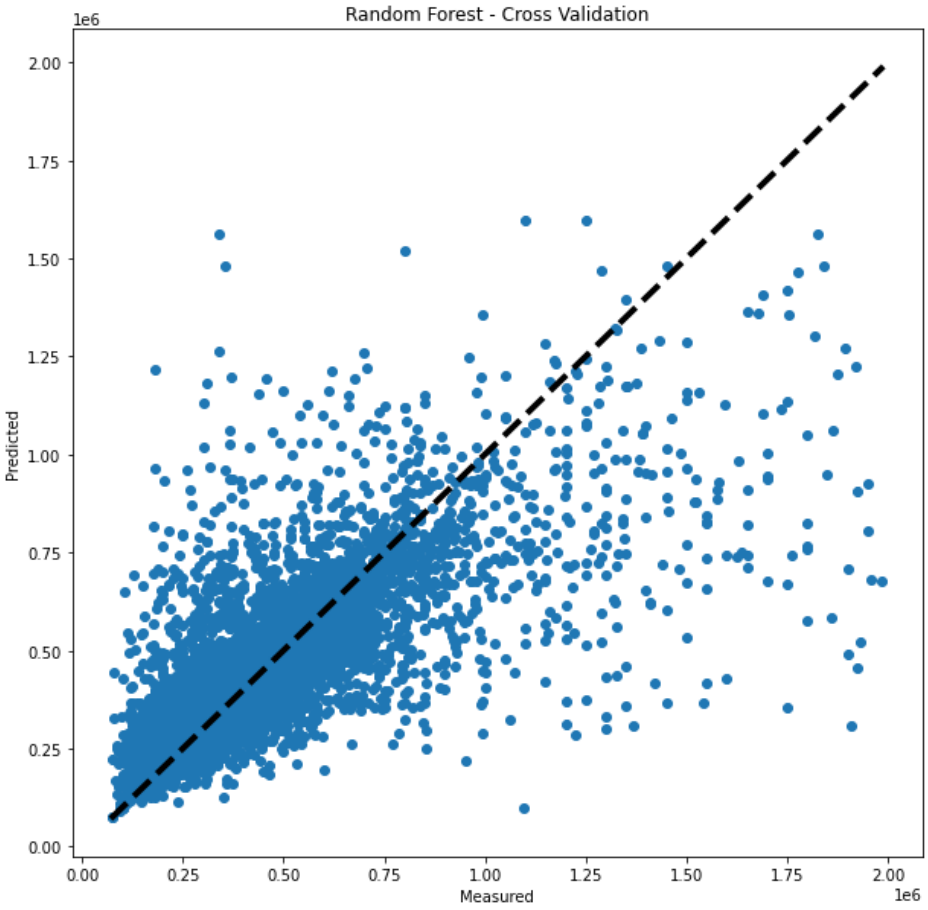

Figure 142: How well the random forests model performed against the linear regression.

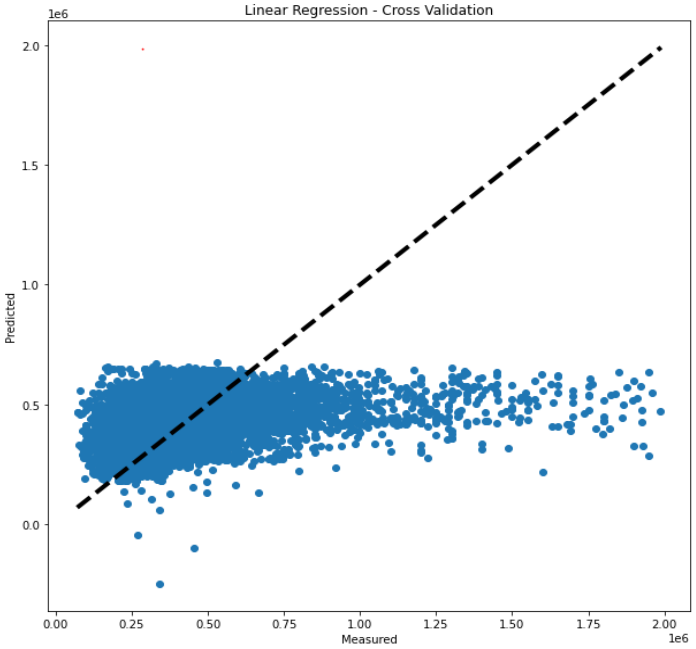

Figure 143: How well the random forests model performed against the linear regression.

We decided to experiment with those 3 models as they were our 3 best options by a bit, especially the random forests and the XGBoost models. We decided to apply feature selection again on the first two especially as they had a better performance.

As we already had tuned the parameters for our first test, we already had an idea of what to tune again, so little was changed in this par.

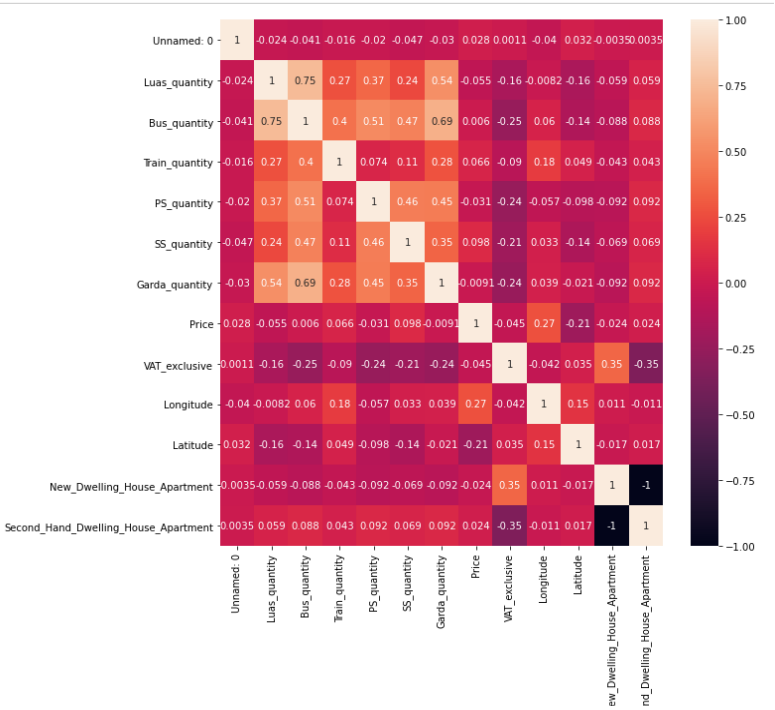

Figure 144: Features Correlation.

You can see that most of the features do not have a big relation with the price feature besides the longitude and latitude, so we tried to take a few of these for testing, we started with latitude and longitude to test their correlation. You can see below how badly they performed when those features were taken out, it reduced to almost 50% of their EVS and R2 scores.

Random Forests Results

```
: print('Explained Variance Score of Random Forest Tree model is {}'.format(evs(y_test, random_predict)))
  print('R-Squared of Random Forest Tree model is {}'.format(r2(y_test, random_predict)))

Explained Variance Score of Random Forest Tree model is 0.2615126948614882
R-Squared of Random Forest Tree model is 0.2614985451784406
```

Figure 145: Random Forest Results after removing some important Features.

XGBoost Results

```
: print('Explained Variance Score of XGBoost model is {}'.format(evs(y_test, pred_XGB)))
  print('R-Squared of XGBoost model is {}'.format(r2(y_test, pred_XGB)))

Explained Variance Score of XGBoost model is 0.3057131756055703
R-Squared of XGBoost model is 0.30473994666946636
```

Figure 146: XGBoost Results after removing some important Features.

We then decided to take anything below 0.00, which were only the bus stops and garda stations. Without them it decreased the score by around 0.01%, so it still needs them to perform better. We did some other tests, but nothing changed significantly so we decided to keep the first configuration of features we already had. In the end it was decided to save our Random Forests model as our final model to use into our deployment phase.

13.3 Determine next steps.

With the model finished and ready, we can now test it to determine if it is ready or not for deployment. We need to get all the variables that will be used on the model to create a prediction and then insert them into the function created to predict the model. The function basically gets all the info that the user will insert (latitude and longitude if the house is second hand or not and if it has VAT or not). The function will then run the haversine formula with our amenities dataset to check what is around that location it will then append everything to a dataframe. The function calls our saved model, inserts our data into the model and returns the prediction value. You can see the code below.

Import amenities dataset

```
amenities = pd.read_csv("data/amenities_df.csv")
```

Create variables.

```
latitude = 53.375275
longitude= -6.227172
new_house = 1
second_hand_house = 0
VAT = 0
```

Figure 147: Testing Model Process.

Getting the points according to what the user inserted and creating a df:

```
def predict_price(latitude,longitude,new_house,second_hand_house,VAT):
    #Create new dataframe.
    df = pd.DataFrame(columns = ['Luas_quantity', 'Bus_quantity','Train_quantity','PS_quantity','SS_quantity','Garda_quantity','L
    #Set the counter variables.
    luas = 0
    bus = 0
    train = 0
    PS = 0
    SS = 0
    garda = 0
    #Loop through the blts dataset.
    for label1, row1 in amenities.iterrows():
        #Calculate the distance from the property to the current blts row.
        distance_in_km = geodesic((latitude,longitude),(row1['Latitude'],row1['Longitude'])).km
        #Set if statements to add to counter if certain type and smaller or equal to 1km.
        if distance_in_km <= 1 and row1['Type'] == "Bus Stop":
            bus += 1

        elif distance_in_km <= 1 and row1['Type'] == "Luas Stop":
            luas += 1

        elif distance_in_km <= 1 and row1['Type'] == "Train Stop":
            train += 1

        elif distance_in_km <= 1 and row1['Type'] == "Primary School":
            PS += 1

        elif distance_in_km <= 1 and row1['Type'] == "Secondary School":
            SS += 1

        elif distance_in_km <= 1 and row1['Type'] == "Garda Station":
            garda += 1
    #Append in the end to our dataframe.
    df = df.append({'Luas_quantity': luas,'Bus_quantity':bus, 'Train_quantity':train, 'PS_quantity':PS,'SS_quantity':SS, 'Garda_c
                    'VAT_Exclusive':VAT,
                    'Latitude':latitude, 'Longitude': longitude,
                   'New_Dwelling_House_Apartment': new_house, 'Second_Hand_Dwelling_House_Apartment':second_hand_house}, ignore_i
    #Loading the model.
    filename = 'random_forest_model.sav'
    random_forest_loaded_model = pickle.load(open(filename, 'rb'))
    #Predicting the value.
    prediction = random_forest_loaded_model.predict(df)
    #Return value of the prediction.
    return int(prediction[0])
```

Figure148: Getting the point according to what the user inserts and creating a df:

This will be inserted in a map in which it will show the location of the address(coordinates) that were inserted by the user. You can see below how the

prediction looks after the function was called and inserted into the map. Of course, that in the deployment phase this all will be changed to fit into the website.

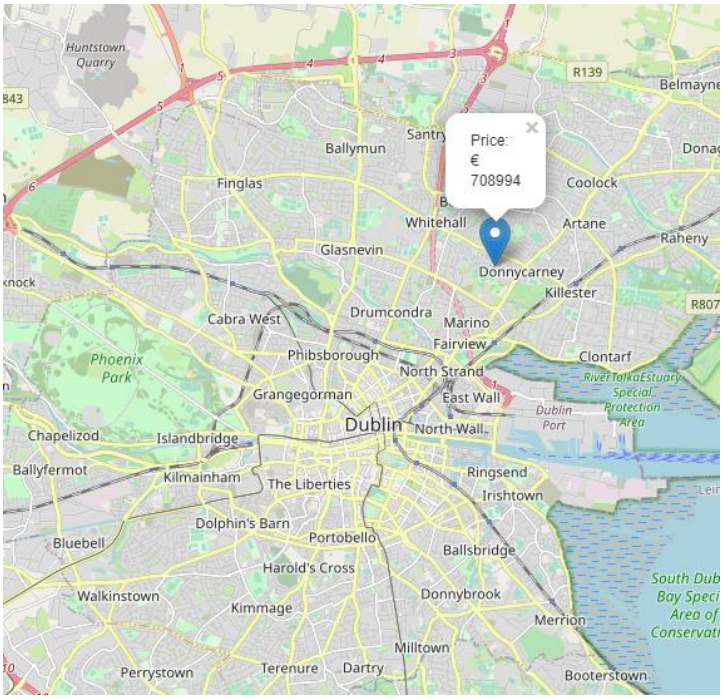

Figure 149: Result of our prediction Model, it works!

Our model can now create predictions and we can implement it into our website, everything is ready now.

14. Deployment

The team has built a prototype of a simple website that could facilitate the visualization of the interactive Map:

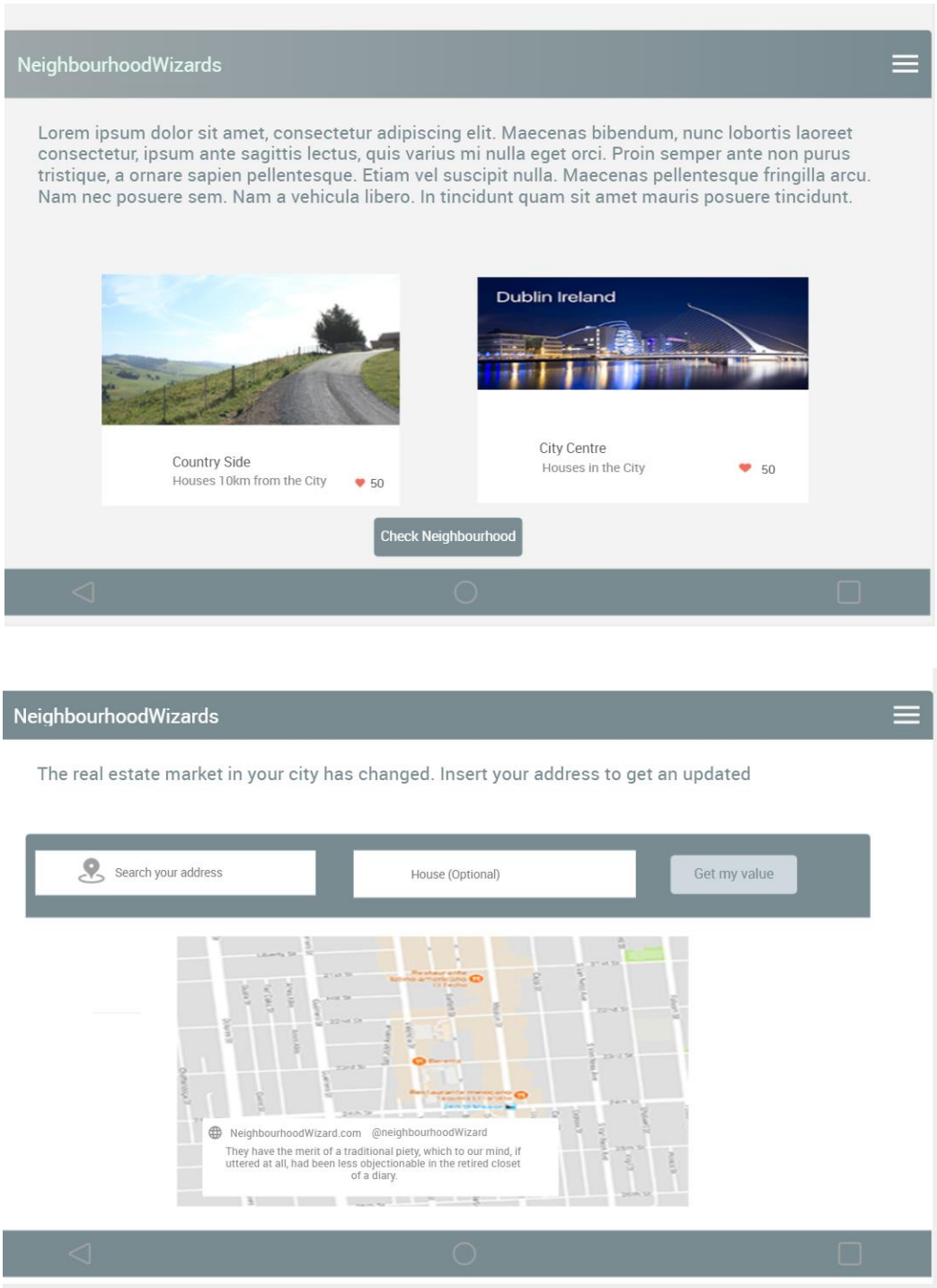

Figure 20: Prototype of our future website.

14.1 The Application

We have finished developing our Machine Learning model, and we are ready to put it into production. The Machine Learning Model was created in Python using Scikit-Learn and Jupiter Notebook (EDA), and we want it to be visualized in a website that will derive and show the values from our model to the user. The team, however, had to deal with the difficulty of transforming this model into a model that consumers could access in a device.

We looked at different approaches and realized we could use Django or Flask to support us. Since we are new to web development and are more familiar with Python, we determined that Flask would be a better choice for developing our website. Flask is more explicit than Django, and it gives us more freedom about how we choose to execute stuff on our website.

In our case, we are not using a database and just want to obtain information from the model to be included in a user request on the website. So, the team selected Flask because it is lighter, more scalable, and easy to use to create a web application. (Devel.tech, 2021)

14.1.1 The front-end of our Web application

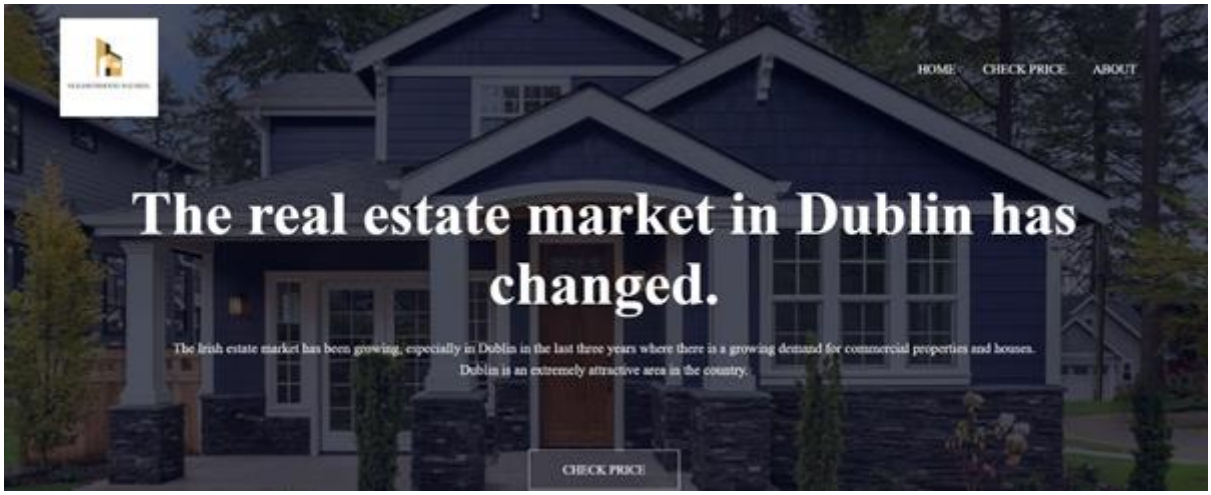

Figure 21: Front-end of our website.

In the picture above, we have the front page of our application where the customer can read about what this website is about and input information about where they are searching for a property and whether they like a new or used property. Finally, when they press search, they will be presented with a Map that shows the position of the property as well as the price.

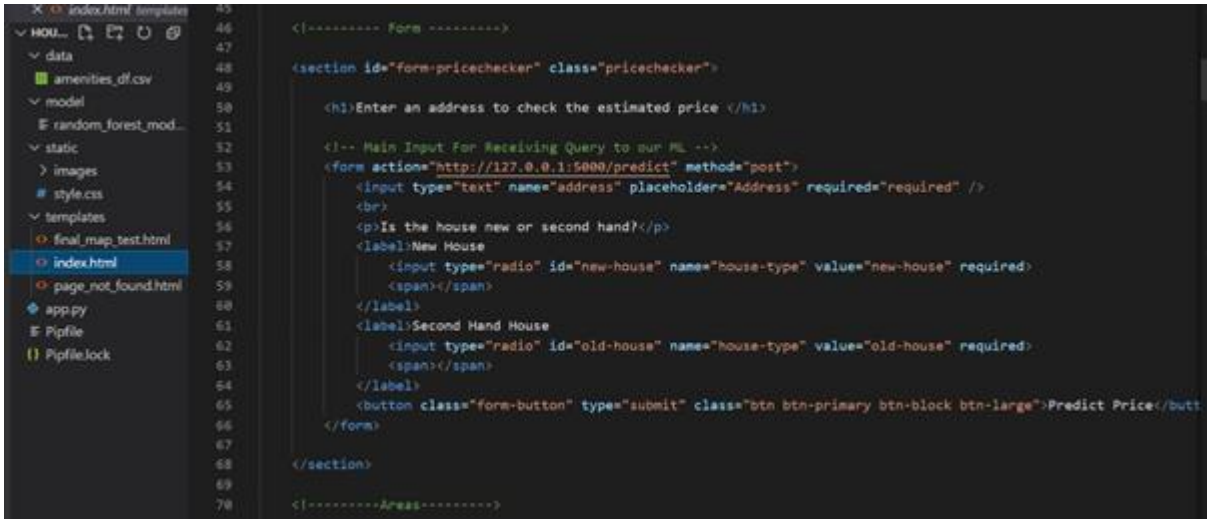

Figure152: HTML of our front-end.

Figure 22: Front-end for user input.

When clicking on the front page “Check Price” button it will send the user straight away to the form. The pictures above show where the user will enter the

address values and whether he needs to look for a new house/apartment or a send-hand one in the front-end. Finally, the user gets the result of the search with the price:

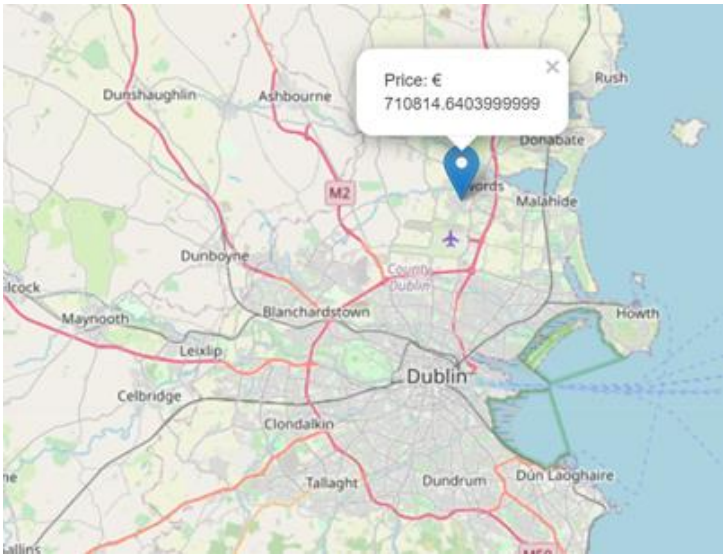

Figure 154: Page that show results of user input.

14.1.2 The back-end of our application

The back-end of our app is where everything happens, from transforming the address to latitude and longitude to displaying our map with the prediction. We first imported all the necessary libraries and documents (our amenities dataset) needed to use in our predict function. This is pretty much the same code as we did in the jupyter notebook but with some extra steps and different implementation.

We have our home function, which calls our home page whenever the app starts (which is when we deploy it), a predict function and an error handler in case something goes wrong. Our predict function is the most important one, where everything happens.

```python
from flask import Flask, render_template, request, redirect, url_for, flash
import pandas as pd
import pickle
import folium
from geopy.distance import geodesic
from geopy.geocoders import Nominatim

geolocator = Nominatim(user_agent="housepriceapp")
amenities = pd.read_csv("data/amenities_df.csv")

app = Flask(__name__)
app.config['DEBUG'] = True
app.secret_key = 'ahsuyafstfatrfshahjhd'


@app.route("/")
def home():
    return render_template("index.html")


```

Figure 15523: app.py file.

We get the form info that the user entered and make a request for the Geolocator API to transform that address into latitude and longitude so we can work with it. It then validates if the coordinates are correct so we can go to the next part which is creating a dataframe and comparing the distance to our amenities dataset to check the amenities around a 1km radius. It then appends it all to a dataset which is used to make a prediction with our pickled model. It then adds a marker to the map using Folium with the price of the prediction. It then renders the map and shows to the user.

```python
@app.route('/predict', methods=['POST'])
def predict():

    #Gets the address from the form.
    address = request.form["address"]
    #Gets info from the geolocator API and returns info about the address (latitude and longitude)
    location = geolocator.geocode(address)
    #If the location is not valid, shows a message and returns the user to the main page.
    if location is None:
        flash('Unfortunately we could not find that address. Please enter another one: ')
        return(redirect(url_for('home')))
    #Else get the location latitude and longitude.
    else:
        latitude = float(location.latitude)
        longitude = float(location.longitude)
        #Latitude and longitude needs to be in the right range, if it not redirect the user to the menu again and asks for a Dublin address.
        if latitude > 54 or latitude < 53 or longitude > -6 or longitude < -7:
            flash(
                'We found an address outside of Dublin. Please enter a Dublin address: ')
            return(redirect(url_for('home')))
        #Else it gets the form value for the house type and set the variables.
        else:
            house_type = request.form['house-type']

            if house_type == "new-house":
                new_house = 1
                second_hand_house = 0
                VAT = 1

            else:
                new_house = 0
                second_hand_house = 1
                VAT = 0


        # Create new dataframe.
        df = pd.DataFrame(columns=['Luas_quantity', 'Bus_quantity', 'Train_quantity', 'PS_quantity', 'SS_quantity', 'Garda_quantity',
                                   'Latitude', 'Longitude', 'New_Dwelling_House_Apartment', 'Second_Hand_Dwelling_House_Apartment', 'VAT_Exclusive'])

        # Set the counter variables.
        luas = 0
        bus = 0
        train = 0
        PS = 0
        SS = 0
        garda = 0

        # Loop through the blts dataset.
        for label1, row1 in amenities.iterrows():
            # Calculate the distance from the property to the current blts row.
            distance_in_km = geodesic(
                (latitude, longitude), (row1['Latitude'], row1['Longitude'])).km
            # Set if statements to add to counter if certain type and smaller or equal to 1km.
            if distance_in_km <= 1 and row1['Type'] == "Bus Stop":
                bus += 1

            elif distance_in_km <= 1 and row1['Type'] == "Luas Stop":
                luas += 1

            elif distance_in_km <= 1 and row1['Type'] == "Train Stop":
                train += 1

            elif distance_in_km <= 1 and row1['Type'] == "Primary School":
                PS += 1

            elif distance_in_km <= 1 and row1['Type'] == "Secondary School":
                SS += 1

            elif distance_in_km <= 1 and row1['Type'] == "Garda Station":
                garda += 1

        # Append in the end to our dataframe.
        df = df.append({'Luas_quantity': luas, 'Bus_quantity': bus, 'Train_quantity': train, 'PS_quantity': PS, 'SS_quantity': SS, 'Garda_quantity': garda,
                        'VAT_Exclusive': VAT,
                        'Latitude': latitude, 'Longitude': longitude,
                        'New_Dwelling_House_Apartment': new_house, 'Second_Hand_Dwelling_House_Apartment': second_hand_house}, ignore_index=True)

```

```
            # Loading the model.
            filename = 'model/random_forest_model.sav'
            random_forest_loaded_model = pickle.load(open(filename, 'rb'))

            # Predicting the value.
            prediction = random_forest_loaded_model.predict(df)
            #Creates the map with the center on the entered coordinates (center of Dublin).
            map_pickup = folium.Map(location=[53.347409, -6.272184])
            #Adds a marker with the location coordinates and adds the prediction value to the market.
            folium.Marker(location=[latitude, longitude],
                          popup="Price: € " + str(prediction[0])).add_to(map_pickup)
            #Save the map to the templates folder.
            map_pickup.save('templates/final_map_test.html')
            #Render the map.
            return render_template("final_map_test.html")


@app.errorhandler(404)
def page_not_found(error):
    return render_template('page_not_found.html'), 404


if __name__ == "__main__":
    # app.run(debug=True)
    app.run()
```

Figure 156: app.py file

Graphic Image of how our application works:

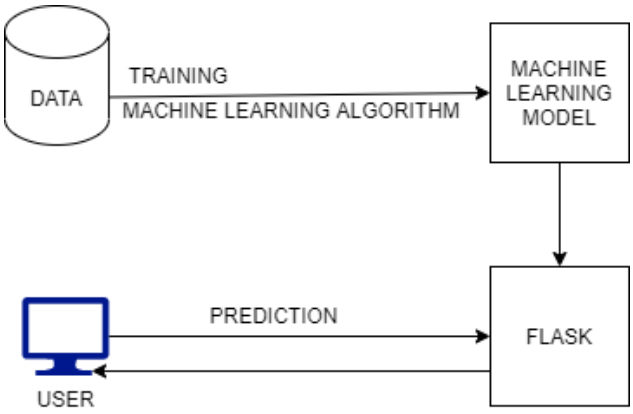

Figure 157: Graphic oh how our application works in the back-end.

14.1.3 Deploying to Heroku

The last step is deploying our web app to a cloud platform. We chose Heroku (2021) as our final platform as it is easy to deploy apps and there is no need to worry about the infrastructure. As our app is pretty simple and their platforms provide free services, we decided to deploy our final application to Heroku.

Deploying to Heroku is simple, you just need to create an account and connect your Github account and repository that your app is located on. It automatically sets

your environment and launches your app with a URL domain, in this case you can check our web app at this address: https://housepricingapp.herokuapp.com/

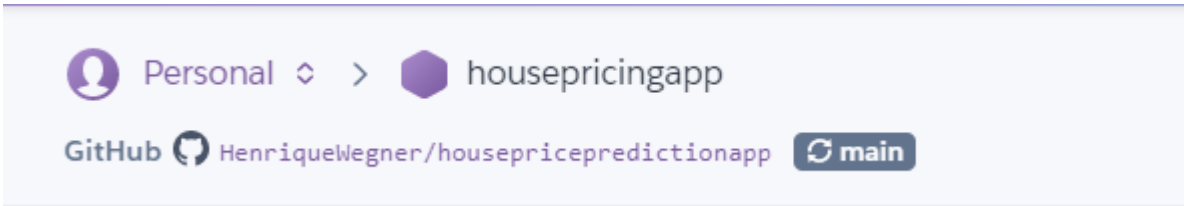

Figure 158: Application Deployed in Heroku.

15. Troubleshooting

15.1 Data collection from Google API call

At the first stage of gathering the latitude and longitude from google API for our property dataset. I ran a piece of code to select the houses from 2010 to 2014 and saved into a new dataframe.

```
df_houses =  df[(df['Date of Sale (dd/mm/yyyy)'] > '2010-01-01') & (df['Date of Sale (dd/mm/yyyy)'] <= '2015-01-01 ')]
```

```
df_houses.head()
```

	Date of Sale (dd/mm/yyyy)	Address	Postal Code	County	Price (€)	Not Full Market Price	VAT Exclusive	Description of Property	long	lat
1	2010-01-03	134 Ashewood Walk, Summerhill Lane, Portlaoise	NaN	Laois	€185,000.00	No	Yes	New Dwelling house /Apartment		
2	2010-01-04	1 Meadow Avenue, Dundrum, Dublin 14	NaN	Dublin	€438,500.00	No	No	Second-Hand Dwelling house /Apartment		
3	2010-01-04	1 The Haven, Mornington	NaN	Meath	€400,000.00	No	No	Second-Hand Dwelling house /Apartment		
4	2010-01-04	11 Melville Heights, Kilkenny	NaN	Kilkenny	€160,000.00	No	No	Second-Hand Dwelling house /Apartment		
5	2010-01-04	12 Sallymount Avenue, Ranelagh	NaN	Dublin	€425,000.00	No	No	Second-Hand Dwelling house /Apartment		

Figure 159: Properties dataset from 2010-2014

So that I could collect the latitude and longitude for those houses addresses from the API, however the code took 2 days running and it did not finish the query. Afterwards the group decided that this data would not be needed as the Luas feature was implemented only after 2017.

```python
for x in range(len(df_houses)):
    try:
        sleep(1) #to add delay in case of large DFs
        geocode_result = gmaps.geocode(df['Address'][x])
        df_houses['lat'][x] = geocode_result[0]['geometry']['location'] ['lat']
        df_houses['long'][x] = geocode_result[0]['geometry']['location']['lng']
        for i in range(len(geocode_result[0]['address_components'])):
            for j in range(len(geocode_result[0]['address_components'][i]['types'])):
                if geocode_result[0]['address_components'][i]['types'][j] == "street_number":
                    df_houses['street_number'][x] = geocode_result[0]['address_components'][i]['long_name']
                elif geocode_result[0]['address_components'][i]['types'][j] == "route":
                    df_houses['route'][x] = geocode_result[0]['address_components'][i]['long_name']
                elif geocode_result[0]['address_components'][i]['types'][j] == "neighborhood":
                    df_houses['neighborhood'][x] = geocode_result[0]['address_components'][i]['long_name']
                elif geocode_result[0]['address_components'][i]['types'][j] == "postal_town":
                    df_houses['postal_town'][x] = geocode_result[0]['address_components'][i]['long_name']
                elif geocode_result[0]['address_components'][i]['types'][j] == "administrative_area_level_1":
                    df_houses['county'][x] = geocode_result[0]['address_components'][i]['long_name']
                elif geocode_result[0]['address_components'][i]['types'][j] == "postal_code":
                    df_houses['postal_code'][x] = geocode_result[0]['address_components'][i]['long_name']
    except IndexError:
        print("Address was wrong...")
    except Exception as e:
        print("Unexpected error occurred.", e )
```

```
<ipython-input-19-98482346cfee>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame
```

Figure 160: Getting latitude and longitude using Google API.

There were 73.556 requests all together.

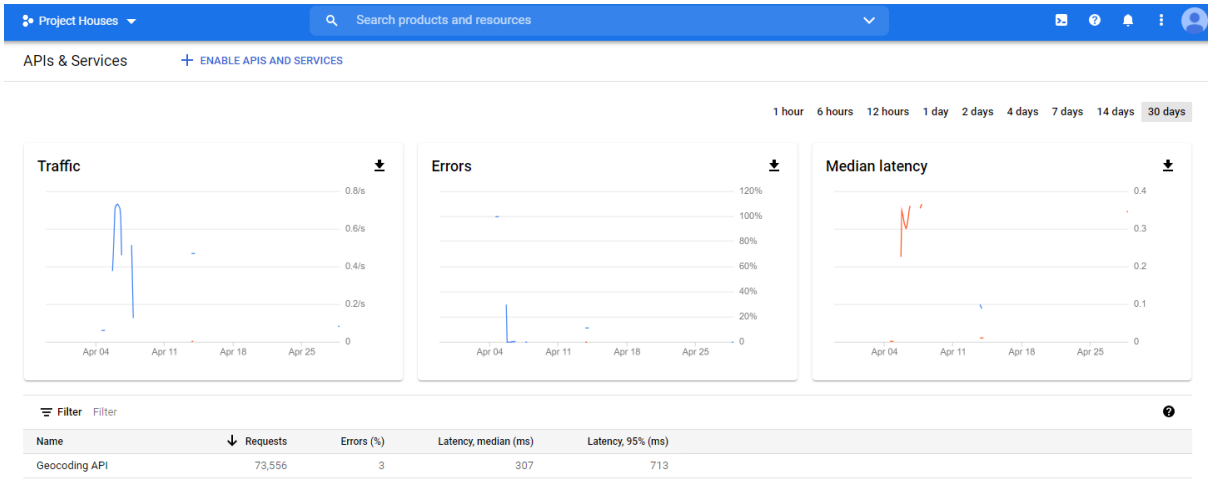

Figure161: Errors from the Google API.

15.1.1 Data Integration and Formatting

When we started formatting our final dataset, we realized that all the garda_quantity values were 0 when we checked for unique values. This was not supposed to happen, there would be at least 1 garda station around a few properties.

```
print('Luas_quantity:',df.Luas_quantity.unique())
print('Bus_quantity :',df.Bus_quantity .unique())
print('Train_quantity :',df.Train_quantity.unique())
print('PS_quantity:',df.PS_quantity.unique())
print('SS_quantity:',df.SS_quantity.unique())
print('Garda_quantity:',df.Garda_quantity.unique())
print('Date_of_Sale:',df.Date_of_Sale.unique())
print('Price:',df.Price.unique())
print('Not_full_market_price:',df.Not_full_market_price.unique())
print('VAT_exclusive:',df.VAT_exclusive.unique())
print('Description_of_property :',df.Description_of_property.unique())
 231 280 113 356 204 124 122 249 186 271 339 175 154 190 201 299 157 256
 251 182 171 373 337 170 229 294 259 286 329 152 145 136 224 239 209 164
 304 192 219 146 281 193 358 162 288 158 247 123 160 168 220 236 179 311
 316 274 374 222 293 283 172 371 309 133 295 246 191 163 169 240 118 324
 285 161 277 345 128 269 126 307 254 180   4 199 255 181 266 377 364 165
 368 313 260 177 198 289 173 166 187 376 278 333 202 340 214 369 225 200
 343 237 174 276 242 140 206 207 156 381 290 197 188 296 372 303 228 361
 141 233 248 213 318 300 263 267 338 350 250 366 279 282 226 297 352 305
 210 351 205 382 328 330 323 230 258 298 312 301 176 379 306 262 195 386
 363 208 375 354 362 265 284 383 273 321 325 347 331 244 216 349 270 234
 217 360 315 314 211 310]
Train_quantity : [0 1 2 4 3 5]
PS_quantity: [ 1  7  0  4  5  3  2  6  8  9 10 11 12 13]
SS_quantity: [0 2 4 1 3 6 5 8 7 9]
Garda_quantity: [0]
Date_of_Sale: ['2017-01-01' '2017-01-02' '2017-01-03' '2017-01-04' '2017-01-05'
 '2017-01-06' '2017-01-09' '2017-01-10' '2017-01-11' '2017-01-12'
 '2017-01-13' '2017-01-14' '2017-01-16' '2017-01-17' '2017-01-18'
 '2017-01-19' '2017-01-20' '2017-01-21' '2017-01-23' '2017-01-24'
 '2017-01-25' '2017-01-26' '2017-01-27' '2017-01-28' '2017-01-30'
```

Figure 162: garda station dataset preparation.

We then checked our code that created the final dataframe and found out that one of our if statements had a wrong variable, so it would never be able to count the garda stations. Instead of adding 1 to every ‘Garda Station’ type found to the ‘garda’ counter variable it was adding to the ‘SS’ counter variable, which is for the secondary schools.

```
from geopy.distance import geodesic

#Create new dataframe.
df = pd.DataFrame(columns = ['ID', 'Luas_quantity', 'Bus_quantity','Train_quantity','PS_quantity','SS_quantity','Garda_quantity']

#Loop through the properties dataset.
for label, row in houses_df_test.iterrows():
    #Set the counter variables.
    luas = 0
    bus = 0
    train = 0
    PS = 0
    SS = 0
    garda = 0

    #Loop through the blts dataset.
    for label1, row1 in blts_df.iterrows():
        #Calculate the distance from the property to the current blts row.
        distance_in_km = geodesic((row['Latitude'],row['Longitude']),(row1['Latitude'],row1['Longitude'])).km
        #Set if statements to add to counter if certain type and smaller or equal to 1km.
        if distance_in_km <= 1 and row1['Type'] == "Bus Stop":
            bus += 1

        elif distance_in_km <= 1 and row1['Type'] == "Luas Stop":
            luas += 1


        elif distance_in_km <= 1 and row1['Type'] == "Train Stop":
            train += 1

        elif distance_in_km <= 1 and row1['Type'] == "Primary School":
            PS += 1

        elif distance_in_km <= 1 and row1['Type'] == "Secondary School":
            SS += 1

        elif distance_in_km <= 1 and row1['Type'] == "Garda Station":
            SS += 1
    #Append in the end to our dataframe.
    df = df.append({'ID': row['ID'],'Luas_quantity': luas,'Bus_quantity':bus, 'Train_quantity':train, 'PS_quantity':PS,'SS_quanti
```

Figure 163: Dataset Preparation.

This code took around 29 hours to run, so we needed a quicker way, we separated the properties dataset into 4 datasets (11,601 lines for each) and ran into 4 separate Jupyter Notebooks at the same time, this way we reduced the time to around 6 to 7 hours for the whole dataframe.

```
df1 = houses_df_test.iloc[0:11601]
df2 = houses_df_test.iloc[11601:23202]
df3 = houses_df_test.iloc[23202:34803]
df4 = houses_df_test.iloc[34803:46404]
```

```
time: 0 ns (started: 2021-04-25 19:36:42 +01:00)
```

```
df1
df1.to_csv('data/final/df1.csv',index = False, header=True)
```

```
time: 47 ms (started: 2021-04-25 19:38:40 +01:00)
```

```
df2
df2.to_csv('data/final/df2.csv',index = False, header=True)
```

```
time: 47 ms (started: 2021-04-25 19:38:59 +01:00)
```

```
df3
df3.to_csv('data/final/df3.csv',index = False, header=True)
```

```
time: 47 ms (started: 2021-04-25 19:39:06 +01:00)
```

```
df4
df4.to_csv('data/final/df4.csv',index = False, header=True)
```

```
time: 47 ms (started: 2021-04-25 19:39:23 +01:00)
```

Figure 164: Dataset Preparation.

15.1.2 School's dataset

When starting to work with the school's dataset, we came across a problem, that thanks to visualization we could see that something was not right.

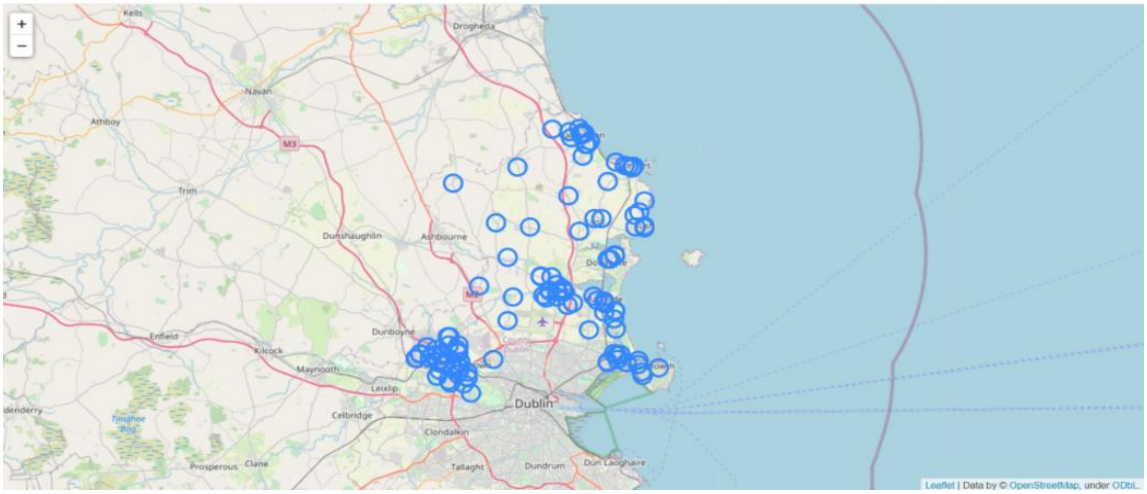

Figure 165: Visualization of the school's dataset.

Although the website where we got the dataset, said that it was a full list of all the primary schools in Dublin, we can see in the visualization, that areas such as Talaght, Drumdrum, Dun Laughaire, Cloundakin, did not have schools, and this does not seem right. When we looked closer, we saw that the dataset had information only about the northside of Dublin.

That is why this process is important, so we can eliminate potential errors that could compromise our model. We got another dataset for our project.

15.1.3 Deployment

In our deployment phase when deploying to Heroku, whenever we tried to predict our house price, something would go wrong with our application and it broke. When checking the logs, we realized there was something going wrong with our pickle file, which is the file that holds our machine learning model.

```
2021-05-02T00:29:05.358879+00:00 app[web.1]:     reraise(exc_type, exc_value, tb)
2021-05-02T00:29:05.358879+00:00 app[web.1]:   File "/app/.heroku/python/lib/python3.8/site-packages/flask/_compat.py", line 39, in reraise
2021-05-02T00:29:05.358880+00:00 app[web.1]:     raise value
2021-05-02T00:29:05.358880+00:00 app[web.1]:   File "/app/.heroku/python/lib/python3.8/site-packages/flask/app.py", line 1950, in full_dispatch_request
2021-05-02T00:29:05.358881+00:00 app[web.1]:     rv = self.dispatch_request()
2021-05-02T00:29:05.358881+00:00 app[web.1]:   File "/app/.heroku/python/lib/python3.8/site-packages/flask/app.py", line 1936, in dispatch_request
2021-05-02T00:29:05.358882+00:00 app[web.1]:     return self.view_functions[rule.endpoint](**req.view_args)
2021-05-02T00:29:05.358882+00:00 app[web.1]:   File "/app/app.py", line 95, in predict
2021-05-02T00:29:05.358883+00:00 app[web.1]:     random_forest_loaded_model = pickle.load(open(filename, 'rb'))
2021-05-02T00:29:05.358883+00:00 app[web.1]: _pickle.UnpicklingError: invalid load key, 'v'.
2021-05-02T00:29:05.359763+00:00 app[web.1]: 10.38.60.5 - - [02/May/2021:00:29:05 +0000] "POST /predict HTTP/1.1" 500 290
"https://housepricedublin.herokuapp.com/" "Mozilla/5.0 (Macintosh; Intel Mac OS X 11_2_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/89.0.4389.128
Safari/537.36"
Autoscroll with output                                                                                                                    Save
```

Figure 166: Picked file.

We started searching for a solution and found out on a post at StackOverflow (2020) by the user Humber, that git-lts needs to be installed in the Heroku environment to run that. On the Heroku (2021) website, the user raxod502 posted a build pack that does the process for us, we only had to add an environment variable to our Heroku app to be able to get the assets. After doing that and deploying our app again, everything started to work fine.

15.1.4 Website Map

When testing our website, we realized that whenever we predicted the house price and it went to the map part of our site, the map would never “refresh” with the new location that we added. So, it would always show the same prediction. We thought it was a variable problem, where they were not being reset and they would always show the same latitude and longitude.

We ran it locally, and realized that the map file was changing, but the flask app was not actually getting the updated template it was getting a cached template. We found a post by a user on Reddit (2017) where they added a debug=true line to the flask configurations to stop the caching. This solved our problem in the end and our app started working just as we wanted.

16. Conclusion

The team began this Machine Learning project with the aim of training a model to predict Dublin property prices.

In the last few weeks, we discovered a slew of tasks that needed to be completed to reach our goal.

We combined the datasets into the data frame we were going to use to train the model after collecting data from official websites and spending a considerable amount of time extracting the latitude and longitude from the datasets we had. After we finished our dataset, we ran some tests to see how much precision we were achieving, and the results were not good enough; we were only getting a maximum of 20% accuracy using XGBoost. More time was spent cleaning the data and removing the features needed for modelling, but we now have the final dataset to train the algorithm with.

We evaluated several regression models, including Linear Regression, XGBoost, CatBoost, LightGBM, Gradient Boosting, Hist Gradient Boosting, and our champion, Random Forest, who achieved up to 58 percent accuracy. Overall, we are very pleased with the success of our final model, particularly given the limited time we had to complete this project. The use of feature selection and data cleaning significantly enhanced the efficiency of our model.

Having said that, the team concluded that the choice of our house’s dataset was not the right one. If we had more time or expertise, we may have studied more about our datasets or attempted to scrape data from websites such as Daft.ie and

MyHome.ie and see if it could be combined with additional features and methods to enhance the model's efficiency. We were pleased with the outcome, given the nature of our project and the time constraints. We were able to train and deploy our model on a website.

17. Appendix

17.1. Group Reflection

Since day one, the group has been dedicated to achieving our goal. We met once a week, often twice a week, to check up with each other, what needed to be done, and how we were doing. At each point of the project, we gained a better understanding of the issues we were encountering and the variables that might affect the consistency of our model.

When all did not work as intended, the team worked together to develop a new solution to solving the problem. We all started learning about Machine Learning this semester only, so it was a big challenge to do that as our final project and be able to manage to deploy our own application using a model. All of us had difficulties in specific areas during the project but working together we managed to solve all of them.

We would have done some parts of the project differently if we had more time, such as collecting better data for our project. We spent a lot of time in the first few weeks deciding what our project would be, so we were especially confused when we started searching for our data as we did not know 100% what we wanted to do. The project was 12 weeks long, but it was much shorter as we did not have to get together, find an idea and develop it into something usable.

Searching for data took a long part of this project and we could not get into a consensus of what was usable and what was not, our time just kept getting shorter and that is why we ended up with the model we have, if we already had the knowledge, we have today things would be done differently.

Doing all this work while remote is also quite hard, as not all of our timetables matched. we had to find time between our work and commitments to be able to do this project besides all our other college work that still needed to be done during the same time. The Covid-19 situation also affected everyone mentally as it has been a hard year on everyone, but we still had to push through it.

No major problems occurred during our project, it was mostly small things that we got stuck on but all of them were resolved and we managed to evaluate and see our mistakes during these 12 weeks.

18. Individual Report

18.1. Henrique Atanásio Wegner -2017403

This was a hard year for most of us, and with this project it was not different. None of us knew what our final project would be, and I felt that we were not prepared for what was asked of us, not in a bad way, but in a surprising way. I was expecting an individual project for our final year and we got another group project (which I am not complaining about, it just was just unexpected). When we formed our group, we struggled a lot to understand what our project was supposed to be and what our main idea would be.

I think we spent a lot of time just forming our idea, we went through facial recognition, to sentiment analysis, and ended up with a house price prediction web app. I tried to contribute with ideas to our project, but I was a bit lost myself and tried mostly to trim the idea to a better size, where we could manage the workload. We were in this stage for about 3 to 4 weeks, where we really started to do our work.

We started looking for data, and I was responsible for gathering the transport data (Luas, train, and bus stops). I managed to find the data in the Transport for Ireland website, and proceeded to merge them so we could use in our model. As I discovered how to merge them, I became also responsible for dealing with merging all the datasets. After the datasets were finished, we had to check what was around the properties points and I came with the idea to set a radius of 1km around each property and get what was around, but we had too many points that were outside of Dublin. I found a way to reduce these points utilizing the haversine formula. I then ran the script I wrote to separate which points were close to the properties.

After our feature engineering was done, we had to start modelling and we got a bit stuck in that area, I build the models with the help of everyone and decided how we were going to score. Together we assessed all the models and decided which one would be the best to be used. I did a few extra steps when modelling like, feature importance and some plots to visualize our models a bit better, also tunning the models.

Before our deployment phase I wrote some code of how it would be implemented in a web app as a function, which was later implemented by me Patricia and Larissa in the deployment phase. I helped with some troubleshooting in our deployment phase and parts of the implementation (mostly the back end).

For our documentation, I mostly did the technology research in the beginning (which was also part of the Strategic Business for IT module), our timeline plan, documentation on the parts I was responsible for and a bit of our reflection. The documentation was quite hard to build, especially with five people in our group where everyone is writing at the same time and things can get mixed up.

If I could change something in our project, it would be not wasting so much time deciding what we were going to do (which was inevitable, but I would try to spend less

time now that I know how to do a project like this), and in the collection of our data. Our model had a bad accuracy because our data was not good at all, I think web scrapping from websites like daft.ie and rent.ie would be a much better idea (which we thought about in the beginning but ended up giving up on the idea). We did not have enough information on the property itself, and that is a big part of how the price of a house is decided and it would probably affect the model a lot.

In the end I think everyone gave a lot of themselves for the project and we would not have managed to finish it if we had not. I am quite proud of this project and what we managed to accomplish in such a short time. I was never great with coding and in this project and I managed to do a great deal which was surprising and felt great. Even though we went through some hard times in the beginning I think this was a cool project, I learned a lot with everyone in the group and by myself.

18.1.2 Maiara Figueredo -2017389

For this group project we all worked harmoniously since the beginning to develop our plan and strategy on how to develop the project.

At the first stage, I researched about the Irish market and I wrote about Business objectives with our prediction model, introduction and abstract. In the phase of data understanding, we all brainstormed together to come up with ideas of the features for the model. Then I researched for datasets from the census and the property price registered.

After we had all the data needed, we started cleaned the data before join in one dataset. I had the responsibility of getting the latitude and longitude for the houses from 2010 to 2014 records. I got an API key from google Maps, and it took me more than 48 hours with the querying running in order to get the data from the API, however the group decided that we wouldn't use this data because the Luas was on from 2017, so our data set would be with data from 2017 only.

On the data preparing phase, after the date being joint in one dataset, I started performing, EDA, data engineering and visualization. Along with documenting for our project, research about machine learning modelling techniques and Linear Regression model. After the model was done, I did the first draft of the poster and the slides presentation for the feature engineering part.

Is fair to say that we were all involved in all the phases of the project, even if it was working on a different task or working together in one task. We had our documentation on google docs, which allowed us all to work simultaneously on the report along with basecamp.

The project itself was hard in regarding to the time frame we had to develop it and for the fact we were new to machine learning, but the group was very engaged and I learned a lot from this project and from my peers.

If I could do something differently, I would try to gather more information about the description of the houses, such as number of bedrooms, square foot etc., as I believe that those features would give us a better linear regression and a higher accuracy for the prediction model.

18.1.3 Patricia Correia – 2017352

I am very proud to have completed this project, and I am very pleased with the outcome that this group was able to produce. We were anticipating an individual project for the last semester, but I couldn't tell if it would be as effective as this one.

Since day one, the team has been really excited about this project; we just wanted to do something that we all love working on, and I can tell that was it. The team was involved in the whole process, which, to be honest, was not that long. We were concerned at first because of the time constraints, but it turned out well in the end.

I believe that my commitment to this project was critical in achieving the outcomes that we did. Every week, I would contact the group and arrange an online meeting and see how we were doing with the project and to see who was available to continue assisting or to move on to another task.

I assisted with the decision on what we were going to do with the project, as well as research such as Risk Analysis, how the Dublin housing market is, how the Moscow tool could help us concentrate on a strategy, how Brexit could affect the Dublin housing market, how the Corona Virus could affect the economy, and so on.

I engaged in the collection of primary and secondary school's data in Dublin during the data collection process. During the data processing, I assisted in cleaning some of the data that the team had gathered, as well as cleaning, visualizing, and preparing for merging, so that we could build our own dataframe.

During the Model testing, I assisted Henrique in doing research on each model that we could test and train in order to get the best outcomes from each model and choose the best one for our model.

During the implementation process, I designed the front-end of our page with HTML/CSS and collaborated with Larissa and Henrique to make the back-end work with Flask, which was a new experience for us, and we helped each other learn and deploy our program.

Finally, at the end of our project, I completed the final version of our poster, based on Maiara's draft. I have assisted in writing in the Google Docs document, collecting References, labelling all 166 images in the document, and reviewing spelling and format of the final document.

I learned a lot during these 12 weeks; it was a joy to serve in this squad, and it provided me with a very good idea of how it would be to work in a team in an organization.

Throughout this project, I learned how important it is to choose the correct data and how important it is to clean and organize the data before beginning any work. For a technical subject like Machine Learning, I have developed my writing and reading

skills. I may tell that this project helped me to develop my leadership skills as well, and even though I wasn't a project manager in this case, I found myself delegating assignments, debating with my group what the best solution would be to a situation we had, and, of course, keeping myself and the others motivated.

18.1.4 Leandro Silveira – 2017369

This project was challenging from the beginning to the end. To work as a team facilitates a lot because we can share the load of responsibilities, but this lockdown removes the possibilities to get physically together. Even having amazing tools such as GitHub, zoon, basecamp, WhatsApp which allow us to communicate and work through distance, I would still prefer to be close to my team members when doing a group project.

Despite all this inconvenience that COVID brought, I am proud to be part of this team that did everything they could to deliver the end goal. Each member had their hands on the various tasks, making themselves available to contribute on the project.

When the project was given to us, our group struggled to decide what to do. We all had good ideas, but some of them would have been impossible to deliver in time and some others would have been unfeasible due to our lack of knowledge or for not having powerful enough machines to work them out.

My main contribution was with research and documentation. Together with my group, I performed and delivered multiple researches to solid build the business understanding concept of our project, analysing and describing the Irish real estate market and defining its target.

In the data understanding step, I ran an investigation and search for relevant datasets. I was in charge to find Dublin criminal rates data from each area, but it changed during the searching process. I also tried to find relevant data about the development of the city throughout the years, such as amenities, shopping centres and such. I ended up not finding much, but 2 or 3 garda stations datasets.

In data preparation I worked with the 3 garda station datasets, performing some feature engineering, data cleaning and preparing the data. I also had many discussions with my group mates, until we decided which one to use and what exactly to do with it.

In the modelling step I contributed with researching about 2 of the models that were tested SVM and CG Boost. These documents help the reader to better understand the reasons why we ran tests with these models. When we were training the model, I contributed with troubleshooting the code and providing code alternatives.

The evaluation and deployment process were discussed within the group and there I had the chance to give ideas and critics to improve our results. During this time, I also took the responsibility to document what we encountered. I created a roles and responsibilities document and a responsibility assessment matrix to critically describe how each member contributed from start to end of the project.

Not only me but the whole group had to review the documentation after it was done, watching out for errors, but also for improvements that could be done. I was also in charge of creating and designing a visual presentation of our project and to edit the final presentation video.

Ultimately, I would like to say that each member participated in the development of the project. We made ourselves available to help each other and to share the load. I really think that the demands were fairly distributed among the group and no one had to overwork.

18.1.5 Larissa Justo Evaldt – 2017270

In the beginning of the semester our first idea was to do something about sentiment analysis or natural language processing but that idea was discarded by the coordinators as it was too complex for the time we had, then our group was struggling a lot in deciding the subject, what our project would be about, and since investing in the house market is something that I am very interested in, I came up with the idea of doing something related to house price prediction. But of course, even though I suggested it, we talked a lot about it in group and shaped the idea together.

In the first part of the project, I was responsible for searching what was already on the market, what tools people who want to buy houses already have at their disposal, what is important to check before buying a house and what aspects usually influence on the price of a house.

For the data collection I searched for the houses dataset and found the Property Price Register website, then this was the data that I was responsible for exploring and cleaning, this dataset was huge and it didn't have latitude and longitude for the houses, so I had to research on what API's could I use to pass the address and retrieve the coordinates. I divided the data by year to do it in parts and used the Google Maps API, I was waiting one second between requests not to overload the APIs servers, so this process took more than 3 days with my computer on day and night, since there were more than 40 thousand requests to make. After this, since I was already used to retrieving latitude and longitude, I did the same process with the Garda dataset. I then wrote the documentation for this part of the process.

After I helped in the feature engineering part together with Patricia, because even though Maiara had done some feature engineering, we were unfortunately getting low accuracy when training the models, so Patricia and I tried to have a second look to see what else could be done with the data and we manage to change a few other things, like outliers in the price and we decided to keep only full market price entries and delete a column.

Then I helped in the development of the web application. Patricia did the home page and the footer of the frontend, and Henrique did most of the backend. I added the form and connected it to the backend, also did some changes to the backend for error handling, then I added the project to Github and tried to do the deployment to Heroku. I was getting an error because of the use of git-lfs to store the large model file, and Henrique was able to fix that and deploy the app.

For the presentation I talked about the data collection and cleaning and put together everybody's videos into one final video, since Leandro had problems with the audio of his file. Finally, I uploaded it to Youtube.

Overall, it was a great experience to work in this group, I felt that we divided tasks equally, so no one was overloaded, and everybody was really giving their best and trying to help each other whenever someone was stuck with something.

19. References

AWS. (2021). *How XGBoost Works - Amazon SageMaker*. [online] Available at: https://docs.aws.amazon.com/sagemaker/latest/dg/xgboost-HowItWorks.html. (Accessed: 29 April 2021).

Bonaccorso, G. (2017). *Machine Learning Algorithms*. Birmingham, UK: Packt Publishing. Available at:http://search.ebscohost.com/login.aspx?direct=true&db=e250xww&AN=1562685&site=eds-live (Accessed: 07 March 2021).

Brennan, J. (2021). *Irish house prices forecast to rise by 4% as supply struggles to meet demand. The Irish Times.* [online] Available at: https://www.irishtimes.com/business/economy/irish-house-prices-forecast-to-rise-by-4-as-supply-struggles-to-meet-demand-1.4460558 (Accessed: 27 February 2021).

Brownlee, J. (2016). *A Gentle Introduction to XGBoost for Applied Machine Learning.* [online] Available at: https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/. (Accessed: 29 April 2021).

Burke-Kennedy, E. (2021). *Home sellers warned to price their properties 'realistically'. The Irish Times.* [online] Available at: https://www.irishtimes.com/business/economy/home-sellers-warned-to-price-their-properties-realistically-1.4452724. (Accessed: 05 March 2021).

Casari, A. and Zeng, A. (2018) *Feature Engineering for Machine Learning*. Sebastopol, CA: O'Reilly Media.

Council, F. (2021). *Schools Data.smartdublin.ie.* [online] Available at: https://data.smartdublin.ie/dataset/analyze/b57bfdfe-5a67-4ad3-9164-5837acab3a74 (Accessed: 01 April 2021).

Central Statistics Office (2021). *Residential Property Price Index December 2020 - CSO - Central Statistics Office*. [online] Available at:

https://www.cso.ie/en/releasesandpublications/ep/p-rppi/residentialpropertypriceindexdecember2020/ (Accessed: 06 March 2021).
Central Statistics Office (2021). *Data.cso.ie*. [online] Available at: https://data.cso.ie/. (Accessed: 21 April 2021).

Cross Validated. (2017*). Support vector machines and regression - Cross Validated.* [online] Available at: https://stats.stackexchange.com/questions/13194/support-vector-machines-and-regression. (Accessed: 28 April 2021).

Deloitte (2021). *Brexit, what does it mean for the real estate sector in Ireland?* [online] Available at: https://www2.deloitte.com/ie/en/pages/globalmarkets/articles/brexit-real-estate-sector-ireland.html: (Accessed: 08 March 2021).

Devel.tech (2021). *Django vs Flask* [online] Available at: https://devel.tech/features/django-vs-flask. (Accessed: 02 May 2021).

Dhiraj K. (2019). *Top 5 advantages and disadvantages of Decision Tree Algorithm*. – Medium [online] Available at: https://dhirajkumarblog.medium.com/top-5-advantages-and-disadvantages-of-decision-tree-algorithm-428ebd199d9a. (Accessed 26 April 2021).

Domingos, P. (2015). *The Master Algorithm*. Basic Books. New York, USA: Basic Books.

Donges, N., (2020). *A complete guide to the random forest algorithm*. [online] Built In. Available at: https://builtin.com/data-science/random-forest-algorithm (Accessed: 26 April 2021).

Department of Enterprise Trade and Employment (2021). *Department of Enterprise, Trade and Employment Website.* [online] Available at: https://enterprise.gov.ie/en/What-We-Do/Trade-Investment/Foreign-Direct-Investment-FDI-/#:~:text=Foreign%20Direct%20Investment%20(FDI)%20has,or%20indirectly%20attributable%20to%20FDI. (Accessed: 09 March 2021).

Fagan, J. (2016). *Foreign investors still dominating Irish real estate market.* [online] Available at: https://www.irishtimes.com/business/commercial-property/foreign-investors-still-dominating-irish-real-estate-market-1.2893822. (Accessed: 11 March 2021).

Fincher, J. (2021). *Python IDEs and Code Editors (Guide) – Real Python*. Available at: https://realpython.com/python-ides-code-editors-guide/. (Accessed: 01 March 2021).

Gallagher, A. (2021). *11 tips to follow when buying a new home*. [online] Available at: https://www.irishtimes.com/business/economy/home-sellers-warned-to-price-their-properties-realistically-1.4452724. (Accessed: 05 March 2021).

Gandhi, R. (2018). *Support Vector Machine — Introduction to Machine Learning Algorithms | Towards Data Science*. [online] Available at: https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47. (Accessed: 28 April 2021).

Geohive (2021). *ArcGIS Web Application*. [online] Available at: https://airomaps.geohive.ie/dho/. (Accessed: 06 March 2021).

GeoHive (2021). *Story Map Series*. [online] Available at: https://geohive.maps.arcgis.com/apps/MapSeries/index.html?appid=f05a07c5a0324b1a887cd9d5d7103e22. (Accessed: 06 March 2021).

Gollapudi, S. and Laxmikanth, V. (2016). *Practical Machine Learning*. Birmingham, UK: Packt Publishing (Community Experience Distilled). Available at: http://search.ebscohost.com/login.aspx?direct=true&db=e020mww&AN=1163839&site=eds-live. (Accessed: 07 March 2020).

Google Developers. (2021). *Geolocation API | Google Developers.* [online] Available at: https://developers.google.com/maps/documentation/geolocation/overview. (Accessed: 21 April 2021).

Google Maps (2021). *Google Maps*. [online] Google Maps. Available at: https://www.google.com/maps. (Accessed: 22 April 2021).

Graham, S. (2020.) *Press release 1 Oct 2020 London, GB EY Financial Services Brexit Tracker: Financial Services Firms continue moving staff ahead of Brexit deadline, with total jobs relocating from London to Europe now over 7,500*. [online]. Available at: https://www.ey.com/en_uk/news/2020/09/ey-financial-services-brexit-tracker-fs-firms-continue-moving-staff-ahead-of-brexit-deadline. (Accessed: 06 March 2021).

Heroku (2021). *raxod502/heroku-buildpack-git-lfs - Buildpacks - Heroku Elements*. [online] Available at: https://elements.heroku.com/buildpacks/raxod502/heroku-buildpack-git-lfs. (Accessed: 02 May 2021).

Heroku (2021). *What is Heroku | Heroku*. [online] Available at: https://www.heroku.com/what. (Accessed 2 May 2021).

Henderson, A. (2020). *How to Invest in Real Estate Overseas: The Ultimate Guide Nomad Capitalist.* [online] Available at: https://nomadcapitalist.com/2020/04/25/how-to-invest-in-real-estate-overseas/. (Accessed: 05 March 2021).

Investopedia (2021). *The 5 Factors of a "Good" Location*. [online] Available at: https://www.investopedia.com/financial-edge/0410/the-5-factors-of-a-good-location.aspx. (Accessed: 06 March. 2021).

iPPi (2021). *Better property valuations in minutes*. [online] Available at: https://ippi.io/ (Accessed: 06 March 2021).

Irish Times. (2020). *The Irish Times Demo Sep 30. 2020*. [online] Available at: http://epaper.irishtimes.com/irishtimesdemo/46/. (Accessed: 06 March 2021).

JLL (2020). *Foreign capital takes Ireland real estate investment to record highs.* [online] Available at: https://www.jll.ie/en/trends-and-insights/investor/foreign-capital-takes-ireland-real-estate-investment-to-record-highs. (Accessed: 09 March 2021).

Kaggle. (2021). *Crime in Ireland | Kaggle*. [online] Available at: https://www.kaggle.com/sameerkulkarni91/crime-in-ireland. (Accessed 19 April 2021).

Kirk, M. (2017). *Thoughtful Machine Learning with Python*. Sebastopol, USA: O'Reilly.

Lavanya, N. and Malarvizhi, T. (2008). *Risk analysis and management a vital key to effective project management*. [online] Available at: https://www.pmi.org/learning/library/risk-analysis-project-management-7070 (Accessed: 03 March 2021).

Lyashenko, V. (2021). *Random Forest Regression - The Definitive Guide*. [online]. Available at: https://cnvrg.io/random-forest-regression/?gclid=CjwKCAjwmv-DBhAMEiwA7xYrdwpiW5knYU6PsQMc7Z52e_u-Z4lw4tz3hWbSpniSmROvII6KxgffhBoC8AcQAvD_BwE (Accessed 26 April 2021).

MacCoille, C. (2020). *Covid-19 has led to a tighter housing market*. [online] Available at: https://www.davy.ie/market-and-insights/insights/myhome/impact-of-covid-on-housing-market.html (Accessed: 6 March 2021).

Metcalf, T. (2021). *Dublin is Top Brexit Relocation Spot for Finance Firms, EY Finds*. [online] Available at: https://au.finance.yahoo.com/news/dublin-top-brexit-relocation-spot-000100322.html?guccounter=1 (Accessed: 05 March 2021).

Mishra, A. (2018). *Metrics to Evaluate your Machine Learning Algorithm*. *Medium*. [online] Available at: https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234 (Accessed: 11 March 2021).

Morde, V. (2019). *XGBoost Algorithm: Long May She Reign! | Towards Data Science.* [online] Available at: https://towardsdatascience.com/https-medium-com-

vishalmorde-xgboost-algorithm-long-she-may-rein-edd9f99be63d. (Accessed: 29 April 2021).

MTS Property Dublin (2021). *How Long Does It Take to Sell a House in Ireland?* [online] Available at: https://www.mtsproperty.com/how-long-does-it-take-to-sell-a-house-in-ireland/ (Accessed: 01 March 2021).

O'Regan, C. (2021). *Dublin Property Impact: The Benefit of Brexit.* [online] Available at: https://spirecapital.ie/dublin-property-impact-the-benefit-of-brexit/#:~:text=Brexit%20Impact%20on%20Dublin%20Property,demand%20for%20local%20rental%20housing.&text=The%20average%20cost%20of%20a,an%20average%20rent%20of%20c. (Accessed: 06 March 2021).

Piatetsky, G. (2021). *Python leads the 11 top Data Science, Machine Learning platforms: Trends and Analysis.* [online] Available at: https://www.kdnuggets.com/2019/05/poll-top-data-science-machine-learning-platforms.htm.l (Accessed: 26 February 2021).

Pobal (2020). *ArcGIS Web Application*. [online] Available at: https://maps.pobal.ie/WebApps/DeprivationIndices/index.html. (Accessed: 06 March 2021).

Productplan (2021). *What is MoSCoW Prioritization?* [online] Available at: https://www.productplan.com/glossary/moscow-prioritization/. (Accessed: 01 March 2021).

Project Jupyter (2021). *Project Jupyter | Home*. [online] Available at: https://jupyter.org/. (Accessed: 01 March 2021).

PYPL (2021). *PYPL PopularitY of Programming Language index.* [online] Available at: https://pypl.github.io/PYPL.html. (Accessed: 26 February 2021).

Reddit (2017). *[AF] Having trouble with flask not updating HTML when I make changes in the templates*. [online] Available at: https://www.reddit.com/r/flask/comments/75zfy7/af_having_trouble_with_flask_not_updating_html/. (Accessed: 02 May 2021).

Richert, W. and Pedro, L. (2013). *Building Machine Learning Systems with Python*. Birmingham: Packt Publishing limited.

Samuel, M. (2016). *Five CIO tips for building an IT strategy in the digital age.* [online] Available at: https://www.computerweekly.com/feature/Five-CIO-tips-for-building-an-IT-strategy-in-the-digital-age. (Accessed: 11 March 2021).

Sandford, A. (2020). *Brexit Timeline 2016–2020: key events in the UK's path from referendum to EU exit*. *euronews.com.* [online] Available at: https://www.euronews.com/2020/01/30/brexit-timeline-2016-2020-key-events-in-the-uk-s-path-from-referendum-to-eu-exit. (Accessed: 06 March 2021).

Scikit-learn (2021). *3.3. Metrics and scoring: quantifying the quality of predictions — scikit-learn 0.22.1 documentation*. [online] Available at: https://scikit-learn.org/stable/modules/model_evaluation.html. (Accessed: 30 April 2021)

Setyorini, I. and Ramayanti, D. (2019) '*Finding Nearest Mosque Using Haversine Formula on Android Platform'*, JOIN: Jurnal Online Informatika, 4(1), pp. 57–62. doi: 10.15575/join.v4i1.267.

Stack Overflow. (2020). *python - Deploying on heroku bert pytorch model using flask: ERROR: _pickle.UnpicklingError: invalid load key, "v."* [online] Available at: https://stackoverflow.com/questions/61884256/deploying-on-heroku-bert-pytorch-model-using-flask-error-pickle-unpicklingerr. (Accessed 2 May 2021).

Stack Overflow. (2017). *location - How to check if coordinate inside certain area Python*. [online] Available at: https://stackoverflow.com/questions/42686300/how-to-check-if-coordinate-inside-certain-area-python. (Accessed: 22 April 2021).

The Pinnacle List (2021). *From Broadband to Amenities – How Location Factors Can Affect Property Values*. [online] Available at: https://www.thepinnaclelist.com/articles/broadband-amenities-how-location-factors-can-affect-property-values/. (Accessed: 01 March 2021).

The Oracles, Contributor. (2019). *Real estate is still the best investment today, millionaires say*. [online] Available at: https://www.cnbc.com/2019/10/01/real-estate-is-still-the-best-investment-you-can-make-today-millionaires-say.html. (Accessed: 11 March 2021).

Transport for Ireland. (2021). *GTFS*. [online] Available at: https://www.transportforireland.ie/transitData/PT_Data.html. (Accessed: 08 March 2021).

WIRE, B. (2020). *Construction in Ireland - Key Trends and Opportunities to 2024 Post Covid-19 - ResearchAndMarkets.com.* [online] Available at: https://www.businesswire.com/news/home/20200915005899/en/Construction-in-Ireland---Key-Trends-and-Opportunities-to-2024-Post-Covid-19---ResearchAndMarkets.com#:~:text=The%20Irish%20construction%20industry%20expanded,building%20and%20civil%20engineering%20works.&text=Furthermore%2C%20the%20slowing%20building%20permits,the%20short%20and%20medium%20terms. (Accessed: 06 March 2021).

XGBoost. (2021). *XGBoost Documentation | xgboost 1.5.0-SNAPSHOT documentation.* [online] Available at: https://xgboost.readthedocs.io/en/latest/. (Accessed: 29 April 2021).