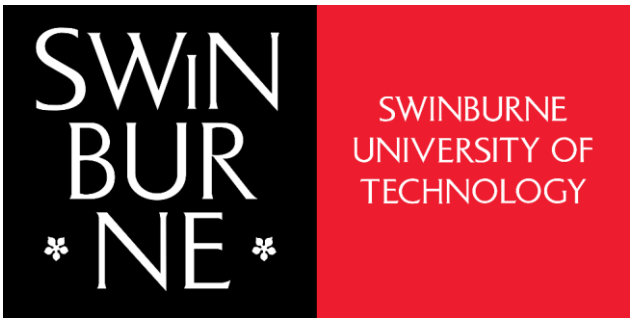


Housing Market Analysis and Price Prediction Using Machine Learning

Conducted by

Cao Minh – 104179289
Nguyen Xuan Huy – 103848944
Jacob Peter Alex – 104673202

Tutor: Ningran Li

September 29, 2024

1. Introduction

The real estate market is dynamic and influenced by various factors, making accurate price predictions challenging yet crucial for homebuyers, investors, and real estate professionals. This project aims to utilize machine learning techniques to predict housing prices and provide insights into the housing market's trends. The primary motivation behind this project is to equip users with data-driven tools to make informed decisions regarding property investments, sales, and purchases. By employing advanced machine learning algorithms, this project helps improve price accuracy and enhances understanding of how different features affect property values. The intended users for this machine learning project include real estate agents, potential homebuyers, investors, and policymakers who need accurate pricing data to navigate the complex housing market.

2. Problem Framing

Accurately predicting housing prices is a challenge in real estate markets due to the vast number of factors influencing property values. These factors include the number of bedrooms, and bathrooms, property size, and proximity to the Central Business District (CBD), as well as broader economic conditions such as interest rates and inflation. The relationships between these factors are often complex and non-linear, making traditional statistical models less effective for price prediction.

The Challenge

The primary challenge this project addresses is the need to predict housing prices more accurately in highly dynamic urban markets like Sydney and Melbourne. Real estate markets are influenced by numerous, interdependent factors, and each suburb or property can have unique characteristics that significantly impact pricing. Traditional models often fail to generalize well to new data, while machine learning models offer better adaptability. Thus, the housing market also experiences

variability due to external factors like economic changes or localized trends, making prediction even more difficult. Traditional models assume a linear relationship between features and price, which is often not the case in real estate markets. From that, traditional methods such as linear regression often assume a direct, linear relationship between variables, which fails to capture the true complexity of the real estate market. Machine learning models can handle larger datasets and more complex variables than traditional methods. So, these models typically do not scale well with large datasets or handle multiple variables effectively. This creates a gap where the accuracy of housing price predictions remains limited, leading to suboptimal decision-making for buyers, investors, and real estate professionals.

Limitations of Existing Solutions

- **Linear Assumptions:** Traditional models assume linear relationships. According to Truong et al. (2020), machine learning can capture non-linear patterns, providing more accurate predictions (Truong et al., 2020).
- **Limited Feature Handling:** Traditional methods struggle with large datasets, whereas machine learning incorporates multiple variables. Ho et al. (2020) highlight that machine learning is more effective at handling complex data (Ho et al., 2020).
- **Generalization Issues:** Traditional models often don't generalize well to new data. Machine learning offers better adaptability and accuracy.

3. Data Collection

For the purpose of building a reliable machine learning model to predict housing prices in Sydney and Melbourne, several datasets were collected from well-established and trustworthy sources. Each dataset provides critical information relevant to the project, ranging from property details to macroeconomic indicators. The credibility of these sources is essential to ensure the accuracy and relevance of

the data used.

a. Target Users

Home buyers are our main client targets, and we have gathered all essential data for target users Homebuyers: Seeking to comprehend real estate values and market trends to make well-informed judgments. Property features help buyers prioritize specific property features based on their needs, such as the number of bedrooms, bathrooms, yard space, etc. Also, many Customers see their purchase as an investment for the future. They consider neighborhood trends, market projections, and prospective property appreciation that could have an impact on future resale value.

b. Data Source Criteria

Relevance: Datasets focus on residential properties in both Melbourne and Sydney to ensure that analyses capture the relevant factors affecting these markets.

Timeliness: We favored data collected from 2016 to 2022 to be representative of today's market conditions and trends

Quality of data: Databases of government and established real estate platforms were focused on in order to ensure the high quality of the data. All data through web scraping were always supported by official sources to check the accuracy and coherence

c. Data Source

Melbourne Housing Market Data

Trustworthiness: This dataset is sourced from Domain.com.au, one of Australia’s most trusted property platforms. Domain.com.au compiles data from real, monthly reports of house sales and purchases, making it a highly

reliable source of actual housing market activity. The Melbourne dataset provides essential details such as property type, price, location, and number of rooms, all based on real transactions in the market.

Purpose: This dataset serves as the foundation for predicting housing prices in Melbourne, leveraging real property sales data to provide insights into the city's real estate market.

Sydney Housing Market Data

Trustworthiness: Similar to the Melbourne dataset, this data is also sourced from Domain.com.au. Given that Domain is a leading real estate platform in Australia, the data reflects actual housing transactions and is updated monthly based on reports from the real estate market. The accuracy of this data ensures it is suitable for building predictive models for Sydney.

Purpose: This dataset is essential for modeling house prices in Sydney, offering key details like house size, number of rooms, and property characteristics.

Australian Postcode Data

Trustworthiness: This dataset is based on official Australian postcode information, making it a highly reliable source for geographic data. The accuracy of the data is crucial for linking properties to specific locations and identifying regional influences on property prices.

Purpose: Used to connect properties to postal codes and provide information on geographic proximity, such as distance to key amenities and city centers. This data helps in analyzing how location impacts housing prices.

Residential Property Index Data

Trustworthiness: The macroeconomic data is scraped from rba.gov.au, the

official website of the Reserve Bank of Australia (RBA). As RBA is a government institution, any data retrieved from its site is considered highly accurate and authoritative. This dataset includes economic indicators such as interest rates and inflation, which are critical factors influencing housing markets.

Purpose: This dataset provides the economic context that affects housing prices, such as monetary policies and economic trends. Integrating this data allows for a more comprehensive analysis of how broader economic factors influence the real estate market.

Financial Data

Trustworthiness: As a government body, ABS is the official provider of statistical data for Australia, making it one of the most trusted sources of information. The Residential Property Price Indexes data offers reliable historical trends on housing prices across the eight major capital cities in Australia, including Sydney and Melbourne.

Purpose: This data tracks the changes in housing prices over time and provides historical context, enhancing the prediction model by identifying price trends and volatility.

d. Challenges and Solutions

During the phases of data collecting and processing, we faced many obstacles such as missing datasets and problems with data accuracy. These were referred to by carrying out pandas library to find missing values and deal with them properly. Cross-referencing data from several sources to confirm data accuracy and make sure our analysis is supported by trustworthy sources

4. Data Processing

Cleaning data plays a crucial role in analyzing data and predicting data, especially in

predicting the housing market which requires a high accuracy. Preprocessing data can affect the machine learning model's prediction because poor preprocessed data can lead to a decrease in the ML model's performance (Bojan Karlaˇs et al., 2020). Cleaning data mainly handles 2 typical issues which are outliers and noise. Noise constitutes data that is either irrelevant or devoid of meaning (Dictionary). Noise data comes from labeling data. Labeling mistakes may arise from automated label extraction (Majkowska, A. et al, 2020), uncertainties in input and output domains (Beyer, L., Hénaff, O. J., Kolesnikov, A., Zhai, X. & van den Oord, A., 2020), or human mistakes (Peterson, J. C., Battleday, R. M., Griffiths, T. L. & Russakovsky, O., 2019). There are the main reasons why noise occurs in datasets and we have to handle them. Noise data can negatively influence the ML model's performance and it may lead to one of the big problems is overfitting. On the other hand, according to the definition of Grubbs in 1969, an outlier, also known as an outlying observation, is a data point that significantly differs from the other elements within the sample it belongs. Although outliers can impact the distribution of data, the outliers are unlikely to be noise data. The outliers can be correct data and can be useful for distinguishing other parts of data. Thus, it is not appropriate to remove the outliers and we can handle them by other techniques. There are 4 main parts to data processing

a. *Data Analysis (Overview of Data)*

Importing essential libraries is very important to visualize the data and explore the data's insight.

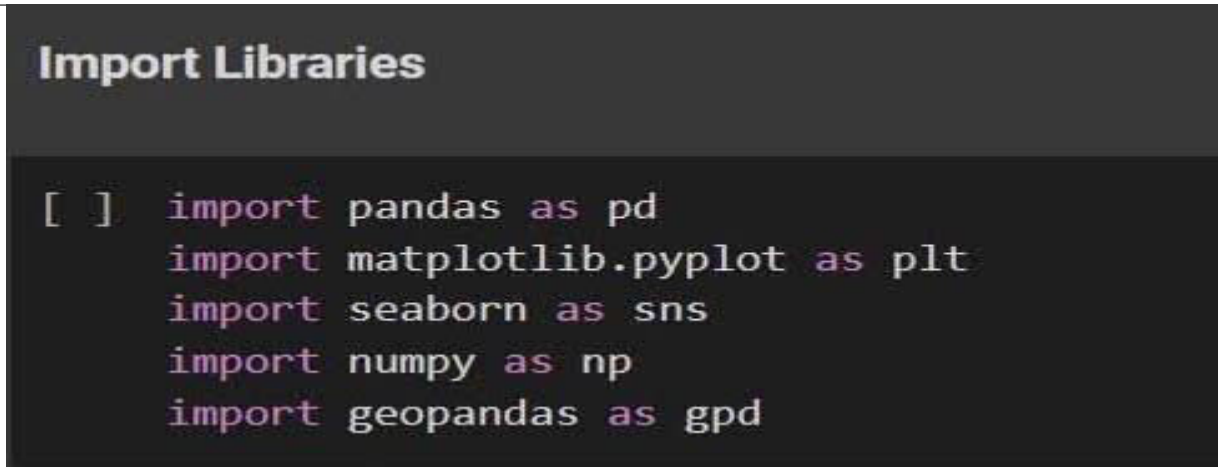


Figure 1: Import libraries for visualizing and cleaning data

Melbourne Data Overview

Load the Melbourne data from the CSV file and show the first 5 rows in the dataset to view the housing market data and attributes.

1. Melbourne Data Overview

```
melbourne_data = pd.read_csv('Melbourne_housing_FULL.csv')
melbourne_data.head()
```

	Suburb	Address	Rooms	Type	Price	Method	SellerG	Date	Distance	Postcode	...	Bathroom	Car	Landsize	BuildingArea	YearBuilt	CouncilArea	Lattitude	Longtitude	Regionname	Propertycount
0	Abbotsford	68 Studley St	2	h	NaN	SS	Jellis	3/09/2016	2.5	3067.0	...	1.0	1.0	126.0	NaN	NaN	Yarra City Council	-37.8014	144.9958	Northern Metropolitan	4019.0
1	Abbotsford	85 Turner St	2	h	1480000.0	S	Biggin	3/12/2016	2.5	3067.0	...	1.0	1.0	202.0	NaN	NaN	Yarra City Council	-37.7996	144.9984	Northern Metropolitan	4019.0
2	Abbotsford	25 Bloomburg St	2	h	1035000.0	S	Biggin	4/02/2016	2.5	3067.0	...	1.0	0.0	156.0	79.0	1900.0	Yarra City Council	-37.8079	144.9934	Northern Metropolitan	4019.0
3	Abbotsford	18/659 Victoria St	3	u	NaN	VB	Rounds	4/02/2016	2.5	3067.0	...	2.0	1.0	0.0	NaN	NaN	Yarra City Council	-37.8114	145.0116	Northern Metropolitan	4019.0
4	Abbotsford	5 Charles St	3	h	1465000.0	SP	Biggin	4/03/2017	2.5	3067.0	...	2.0	0.0	134.0	150.0	1900.0	Yarra City Council	-37.8093	144.9944	Northern Metropolitan	4019.0

5 rows × 21 columns

Figure 2: Loading and showing Melbourne data

shape() function shows that the data has 34857 samples with 21 columns (features). The describe() function shows summary statistics, such as count, mean, standard deviation, minimum, 25th percentile (Q1), median (50th percentile), 75th percentile (Q3), and maximum, for each numerical column. This is necessary because I can evaluate the overall data based on these metrics.

```
melbourne_data.shape
```

```
(34857, 21)
```

```
melbourne_data.describe().T
```

	count	mean	std	min	25%	50%	75%	max
Rooms	34857.0	3.031012e+00	0.969933	1.00000	2.00000	3.0000	4.000000e+00	1.600000e+01
Price	27247.0	1.050173e+06	641467.130105	85000.00000	635000.00000	870000.0000	1.295000e+06	1.120000e+07
Distance	34856.0	1.118493e+01	6.788892	0.00000	6.40000	10.3000	1.400000e+01	4.810000e+01
Postcode	34856.0	3.116063e+03	109.023903	3000.00000	3051.00000	3103.0000	3.156000e+03	3.978000e+03
Bedroom2	26640.0	3.084647e+00	0.980690	0.00000	2.00000	3.0000	4.000000e+00	3.000000e+01
Bathroom	26631.0	1.624798e+00	0.724212	0.00000	1.00000	2.0000	2.000000e+00	1.200000e+01
Car	26129.0	1.728845e+00	1.010771	0.00000	1.00000	2.0000	2.000000e+00	2.600000e+01
Landsize	23047.0	5.935990e+02	3398.841946	0.00000	224.00000	521.0000	6.700000e+02	4.330140e+05
BuildingArea	13742.0	1.602564e+02	401.267060	0.00000	102.00000	136.0000	1.880000e+02	4.451500e+04
YearBuilt	15551.0	1.965290e+03	37.328178	1196.00000	1940.00000	1970.0000	2.000000e+03	2.106000e+03
Lattitude	26881.0	-3.781063e+01	0.090279	-38.19043	-37.86295	-37.8076	-3.775410e+01	-3.739020e+01
Longtitude	26881.0	1.450019e+02	0.120169	144.42379	144.93350	145.0078	1.450719e+02	1.455264e+02
Propertycount	34854.0	7.572888e+03	4428.090313	83.00000	4385.00000	6763.0000	1.041200e+04	2.165000e+04

Figure 3: Data samples and summary statistics

After that, I use a for loop to visualize all the columns containing numbers through histplot chart. Using a histplot chart can be useful because we can know the distribution of data in each column. From that, we can know the frequency count of data and we can handle missing values by using this. For instance, in

Figure 4, the chart shows that most of the house has several rooms from 2 to 4. We can use this to fill the data that missing rooms data and this can make data not become too chaotic.

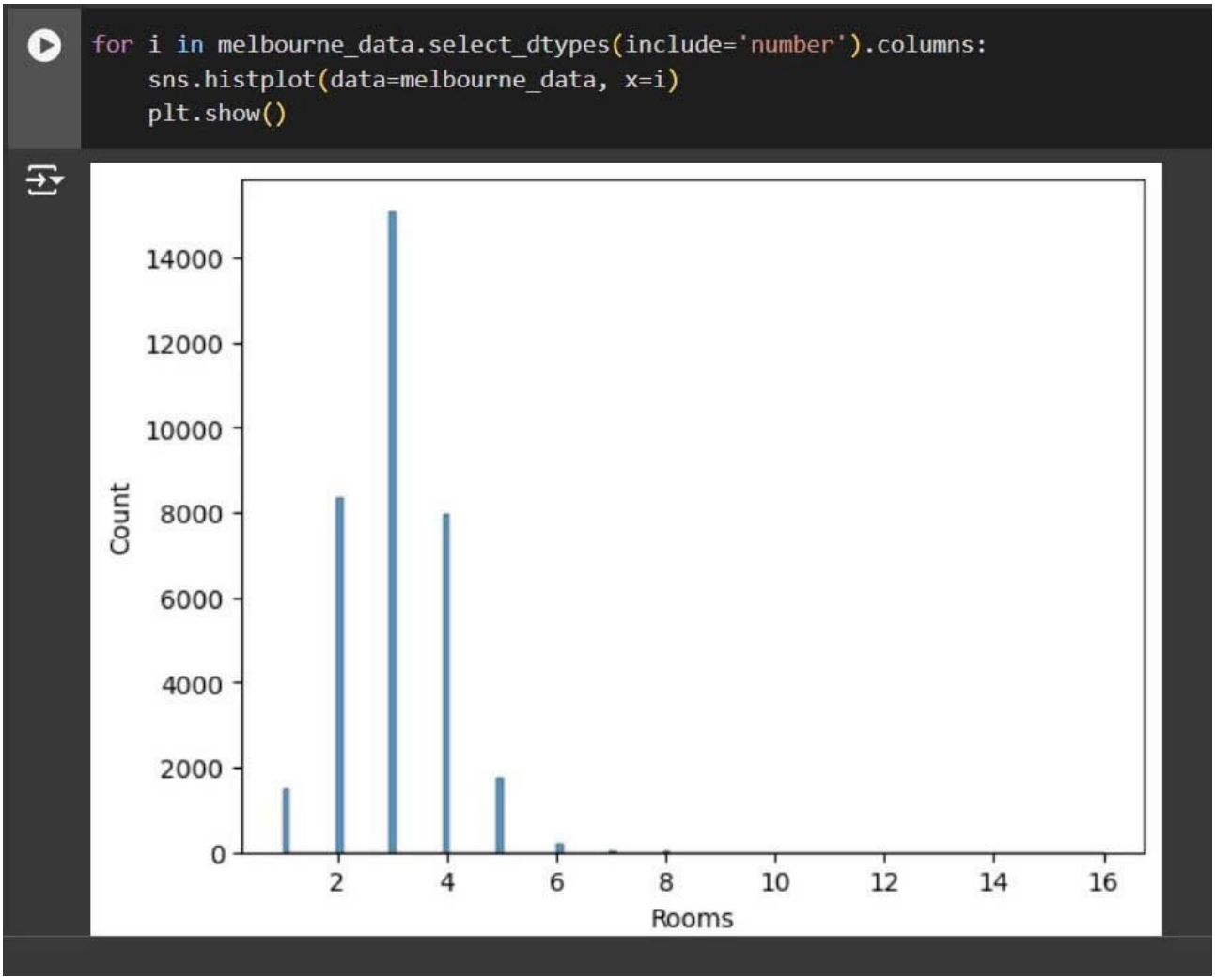


Figure 4: Visualize columns using histplot chart

Then, I also use a for loop to visualize each column with a boxplot chart. This chart can help to handle the outliers by visualizing them. For example, in Figure 5, the boxplot shows that the minimum number of rooms is 1, the maximum is 7, and the number of rooms frequently in a house is 2 to 4. The white dots on the chart show that they are outliers which indicates they might be big houses or mansions. We can delete them or not based on our target client.

```
[ ]  for i in melbourne_data.select_dtypes(include='number').columns:
         sns.boxplot(data=melbourne_data, x=i)
         plt.show()
```

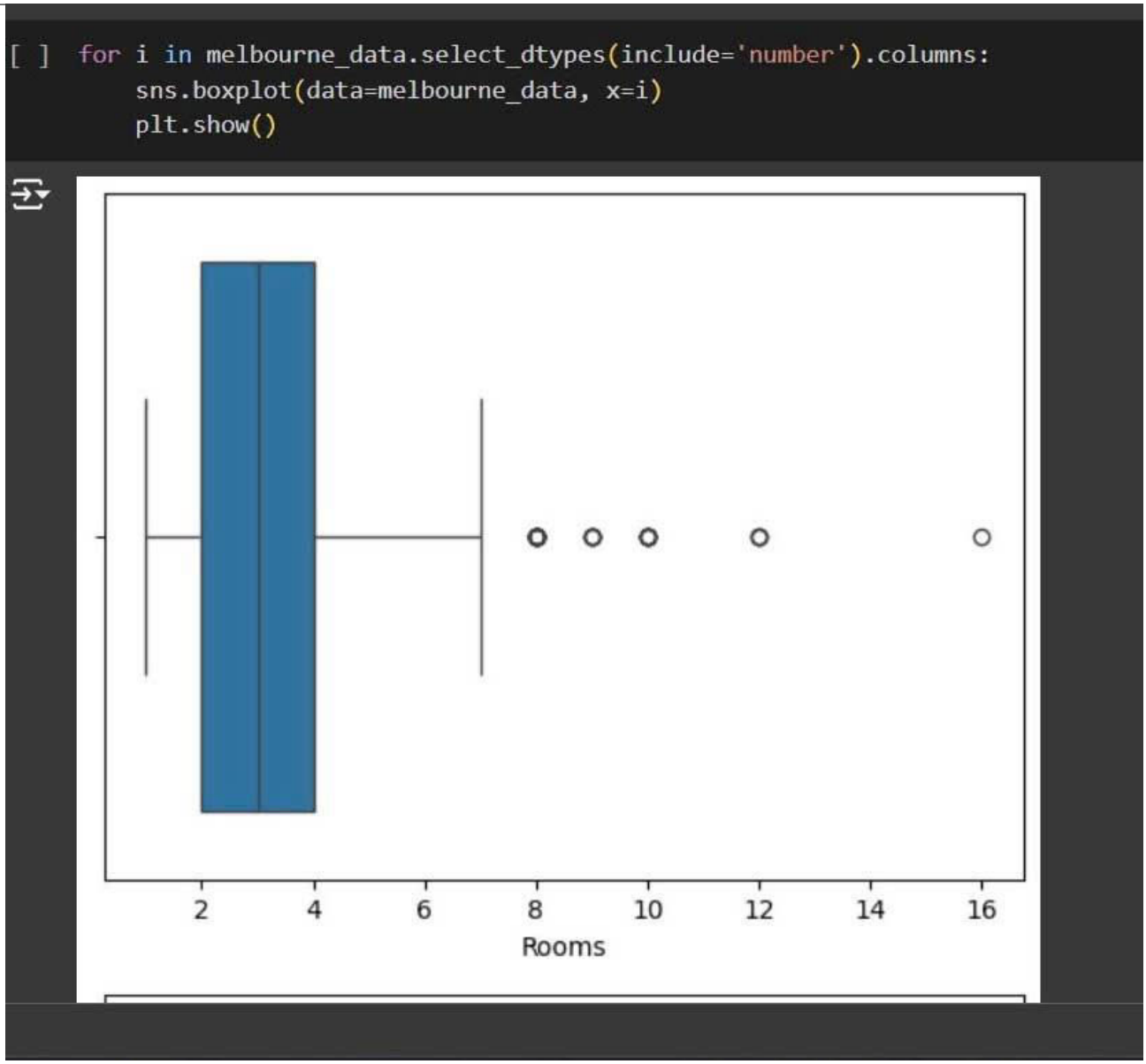


Figure 5: Visualize columns using boxplot chart

Next, I also use the same method above to visualize each column with scatterplot charts. This chart shows the correlation between price and target feature. For instance, the chart shows the number of dots which are the number of rooms in a house. There are upward trends in the chart, thus there is a correlation between rooms and price. In this case, if the number of rooms increases, the price will increase as well.

```
[ ]  for i in melbourne_data.select_dtypes(include='number').columns:
         sns.scatterplot(data=melbourne_data, x=i, y='Price')
         plt.show()
```

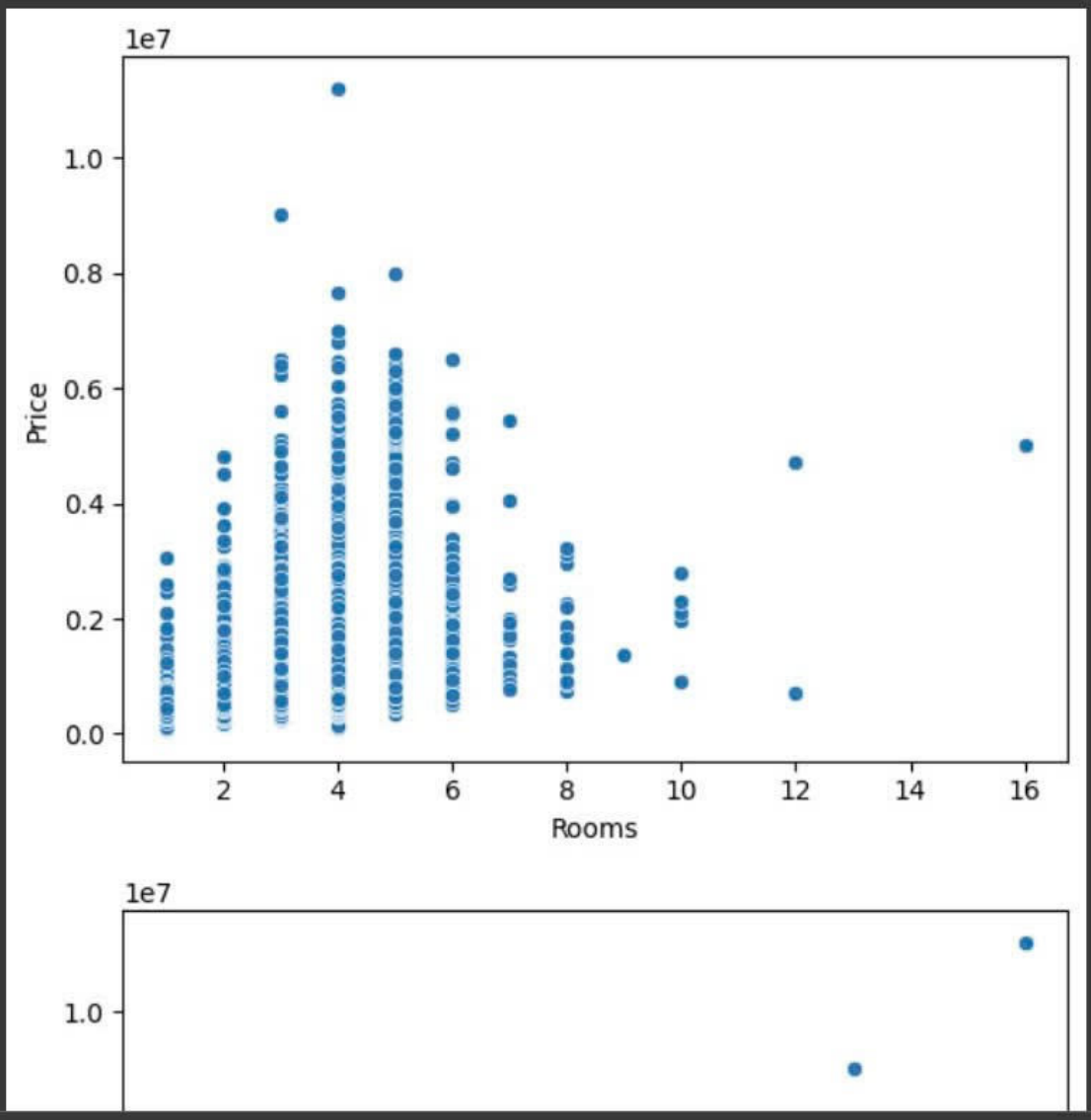


Figure 6: Visualize columns using scatterplot chart

Finally, we not only know the correlation between price and other features but also there is a method to show the correlation between all features with numeric. Using Seaborn lib to visualize the heatmap. The positive numbers here indicate that they are proportional with the target feature and the negative numbers indicate that they are inverse ratio with the target value. Moreover, if the number is near 0, this shows that the correlation between the feature and the target feature

is very low. The feature can disappear on the heatmap due to too low correlation or equal 0. In Figure 8, the correlation between rooms and price is very high (0.47) the year built has an inverse ratio with price (-0.33), and the land size has a very low correlation with price. The price increases along with the price is common sense and can be understandable. The year built has an inverse ratio with the price because the houses in this data might be villas with land houses with historical value or new apartments built with affordable prices for residents. This leads to the old houses having high prices and new houses having low prices. The land size has a very low correlation and this is very in conflict with the practical world where lands are high-value property. Therefore, this is not reasonable, so this data might be missed many data in this feature and we have to clean this raw data to evaluate correctly.

```
[ ]  plt.figure(figsize=(15,10))
     sns.heatmap(melbourne_data.corr(numeric_only=True), annot=True)
     plt.show()
```

Figure 7: Using heatmap to show the correlation

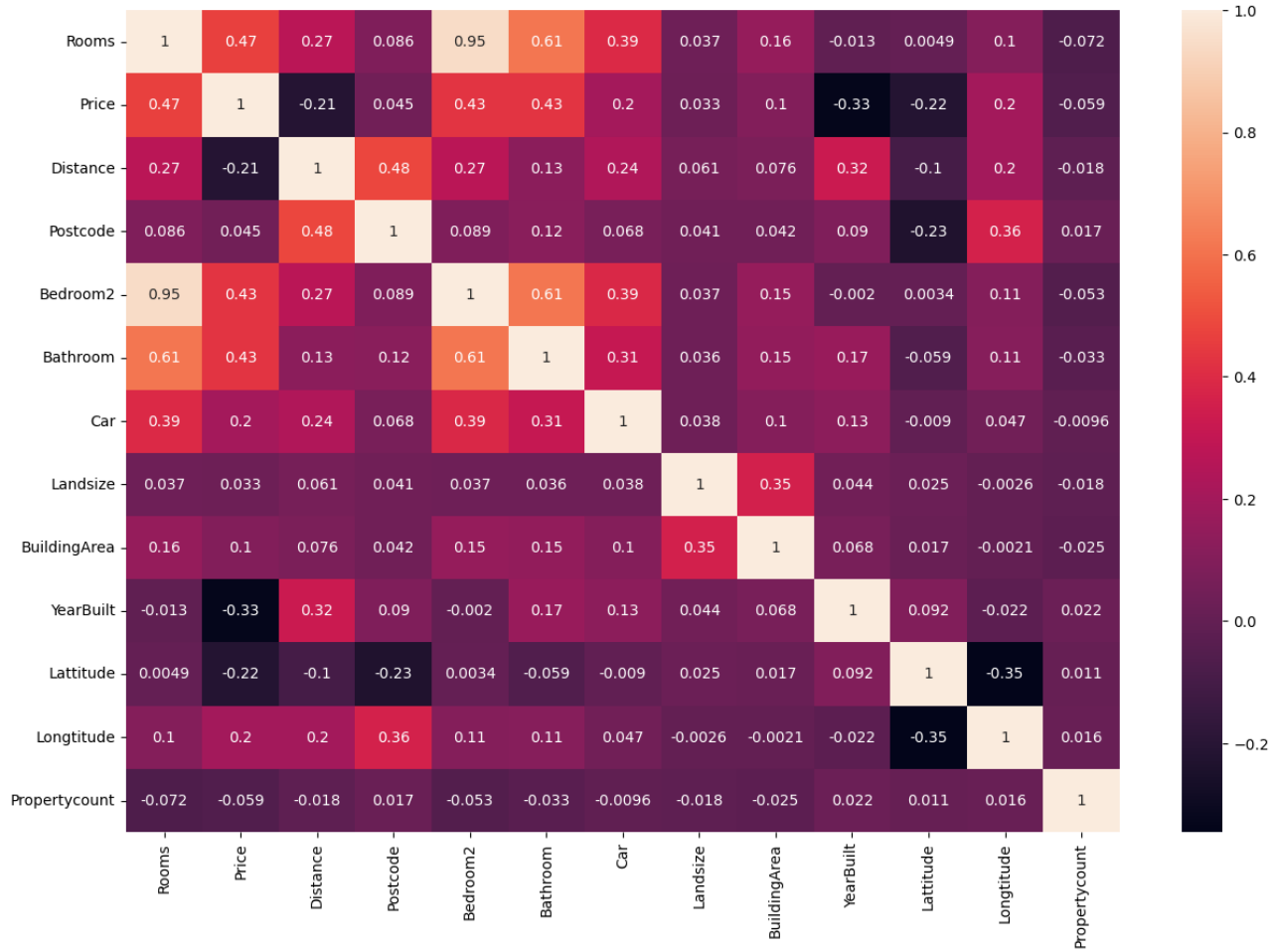


Figure 8: The heatmap of Melbourne data

Sydney Data Overview

In this section, I use the same method with Melbourne data, so I will discuss the example in each image.

In Figure 9, I can view the housing market data from Sydney and its attributes.

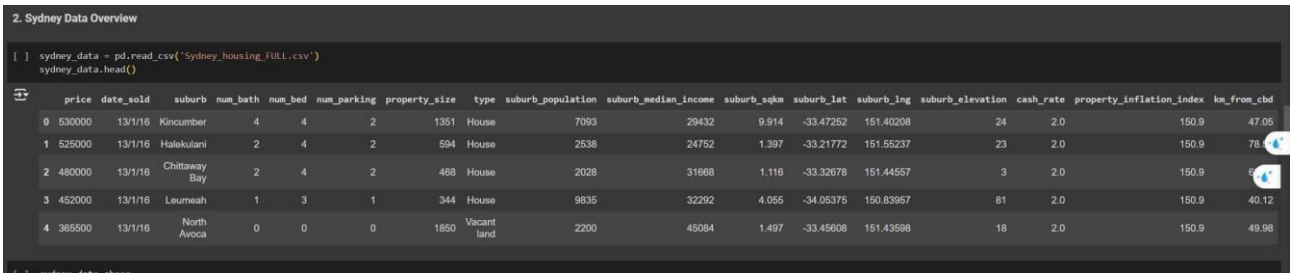


Figure 9: Loading and showing the data of Sydney

In Figure 10, the data has 11160 samples and 17 columns. Furthermore, I can

view the overall of each column in the data.

```
[ ] sydney_data.shape

(11160, 17)

[ ] sydney_data.describe().T
```

	count	mean	std	min	25%	50%	75%	max
price	11160.0	1.675395e+06	1.290371e+06	225000.00000	1.002000e+06	1.388000e+06	2.020000e+06	6.000000e+07
num_bath	11160.0	2.073566e+00	1.184881e+00	0.00000	1.000000e+00	2.000000e+00	3.000000e+00	4.600000e+01
num_bed	11160.0	3.758961e+00	1.559743e+00	0.00000	3.000000e+00	4.000000e+00	4.000000e+00	4.700000e+01
num_parking	11160.0	2.017473e+00	1.454560e+00	0.00000	1.000000e+00	2.000000e+00	2.000000e+00	5.000000e+01
property_size	11160.0	7.230124e+02	1.048984e+03	7.00000	4.300000e+02	6.000000e+02	7.650000e+02	5.910000e+04
suburb_population	11160.0	9.311560e+03	7.541636e+03	22.00000	3.977000e+03	7.457000e+03	1.215825e+04	4.717600e+04
suburb_median_income	11160.0	4.016824e+04	1.108996e+04	14248.00000	3.244800e+04	3.910400e+04	4.555200e+04	9.750000e+04
suburb_sqkm	11160.0	5.054877e+00	5.824663e+00	0.08900	1.776000e+00	3.566000e+00	6.568000e+00	8.715400e+01
suburb_lat	11160.0	-3.378141e+01	2.024778e-01	-34.10624	-3.392148e+01	-3.380918e+01	-3.371551e+01	-3.316376e+01
suburb_lng	11160.0	1.510967e+02	2.134562e-01	150.55384	1.509510e+02	1.511095e+02	1.512278e+02	1.515733e+02
suburb_elevation	11160.0	5.560672e+01	5.280232e+01	0.00000	2.100000e+01	4.000000e+01	7.500000e+01	4.050000e+02
cash_rate	11160.0	6.313611e-01	6.586239e-01	0.10000	1.000000e-01	1.100000e-01	1.500000e+00	2.000000e+00
property_inflation_index	11160.0	1.884897e+02	2.444155e+01	150.90000	1.676000e+02	1.766000e+02	2.201000e+02	2.201000e+02
km_from_cbd	11160.0	2.738183e+01	1.847011e+01	0.31000	1.296000e+01	2.231000e+01	4.099000e+01	8.479000e+01

Figure 10: The number of samples in data and overall metrics

In Figure 11, the charts show the distribution of price, and most prices of houses are in the range of 0.0..1 x 1e7 to 0.5 x 1e7.

```
[ ]  for i in sydney_data.select_dtypes(include='number').columns:
         sns.histplot(data=sydney_data, x=i)
         plt.show()
```

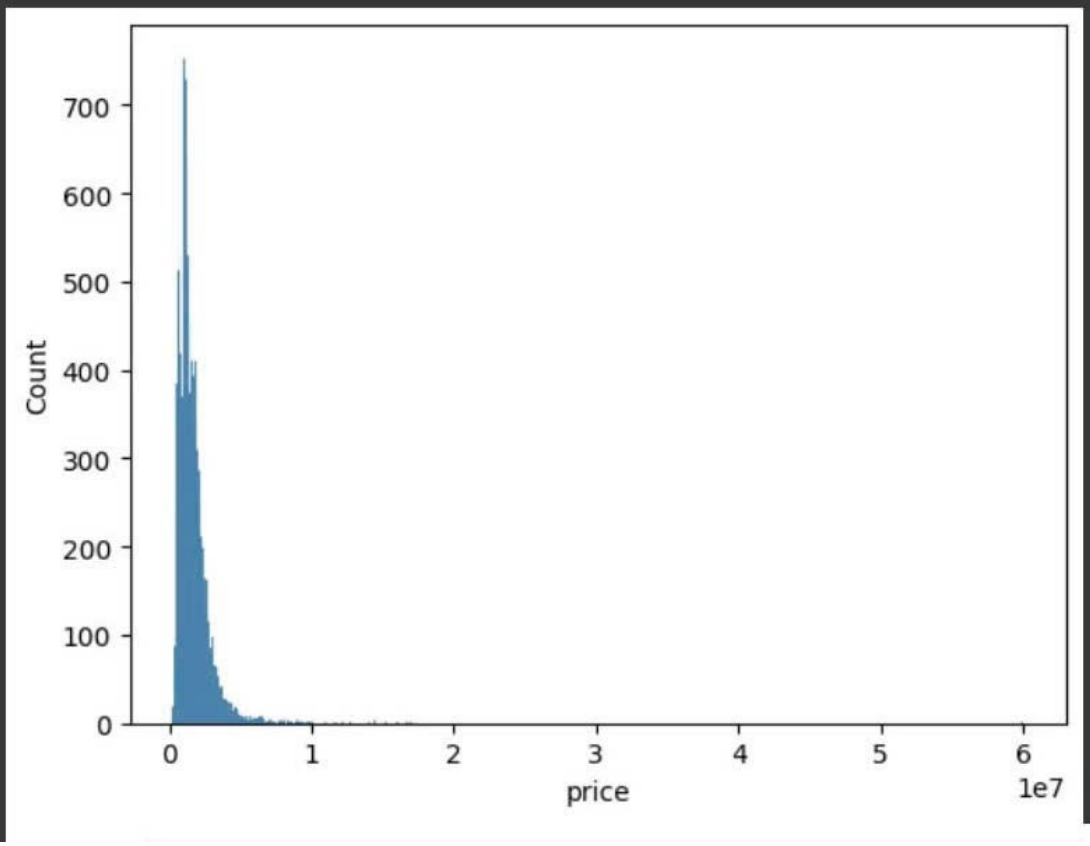


Figure 11: Visualize the histplot chart

In Figure 12, the boxplot shows the average value from 0.1 to 0.2 the white dots are outliers and the dot with x=6 is the extreme outlier. This indicates that this is a very expensive house or building.

```
[ ]  for i in sydney_data.select_dtypes(include='number').columns:
         sns.boxplot(data=sydney_data, x=i)
         plt.show()
```

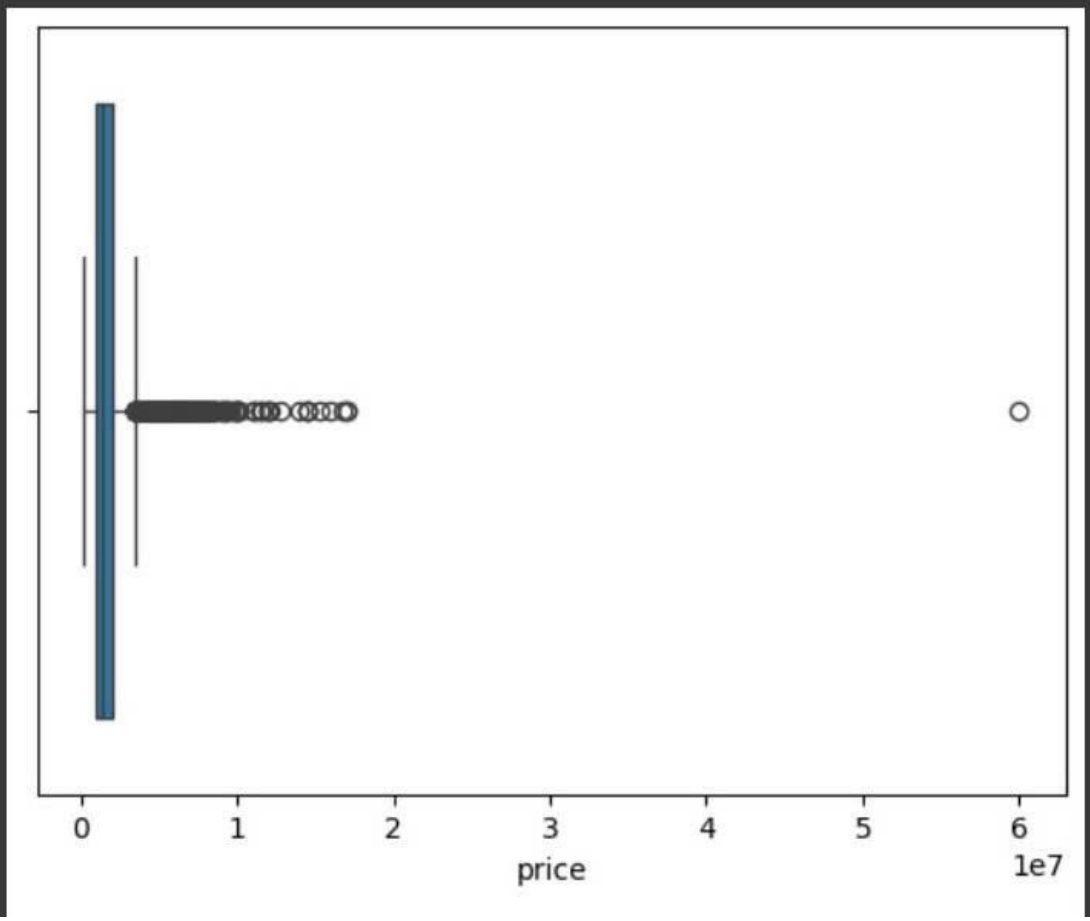


Figure 12: Visualize the boxplot chart

In Figure 13, there are upward trends in the scatterplot chart and this indicates that the number of baths is proportional to the price.

```
for i in sydney_data.select_dtypes(include='number').columns:
    sns.scatterplot(data=sydney_data, x=i, y='price')
    plt.show()
```

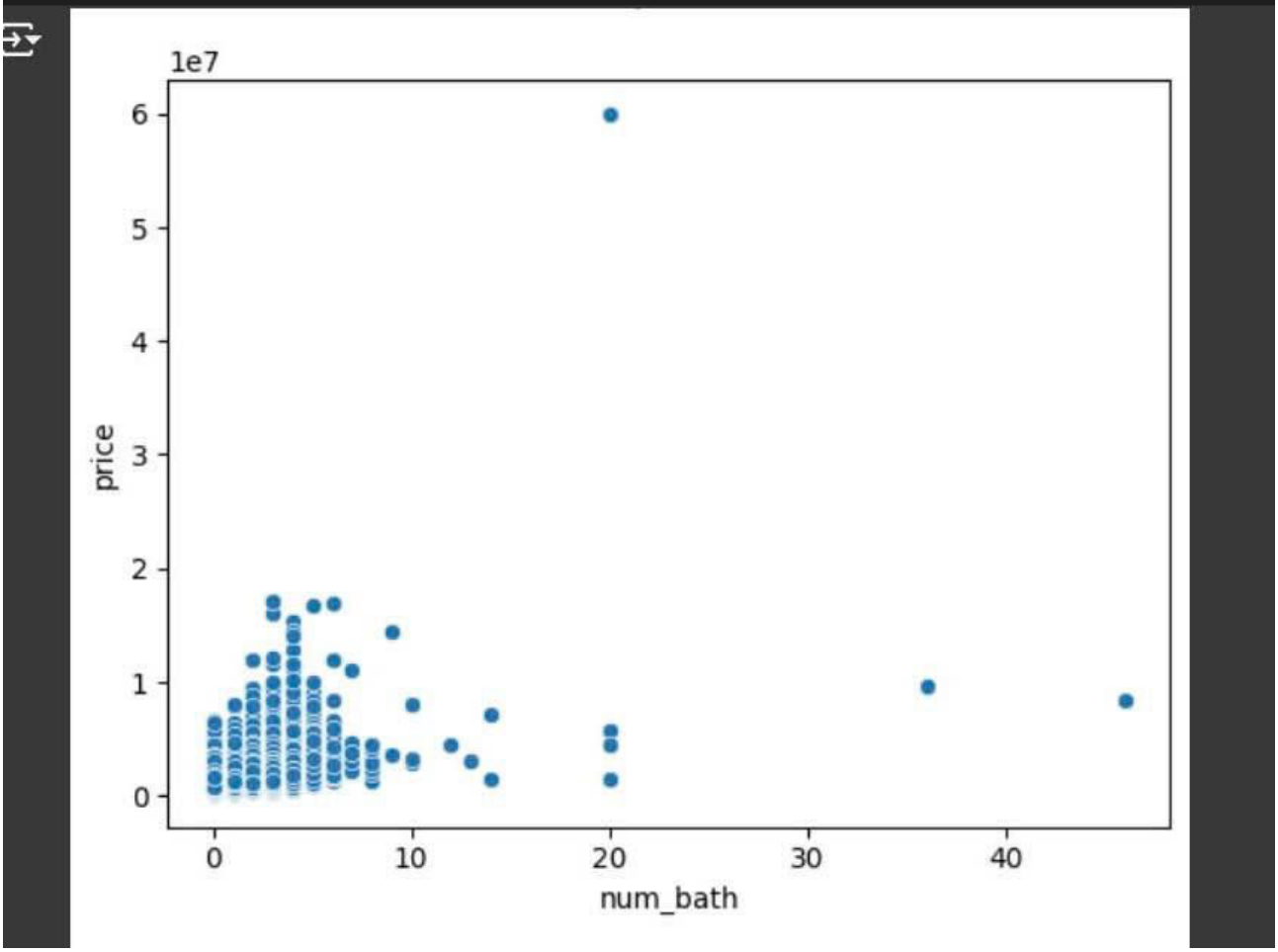


Figure 13: Visualize the scatterplot chart

In Figure 15, the heatmap shows that the number of baths has the highest correlation with price (0.4). This points out that the number of baths is proportional to the price of houses. This is reasonable because only big houses can have more bathrooms and big houses are often more expensive than others. This leads to an increase in price along with the number of bathrooms. In contrast, the feature ‘km_from_cbd’ has the lowest value, but this doesn’t mean it is not vital. This indicates that it is an inverse ratio to the price of the house. This makes sense because ‘km_from_cbd’ calculates the distance between the position of the house to the center of the city. From that, if the houses are located near the city which has a low value in this feature, the price will increase along with the

distance. Moreover, the feature ‘suburb_elevation’ has the weakest correlation to price (-0.0084). This feature indicates the height of the building. This feature may be deleted because it doesn’t contribute to the influence of price and it is not necessary.

```
[ ]  plt.figure(figsize=(15,10))
     sns.heatmap(sydney_data.corr(numeric_only=True), annot=True)
     plt.show()
```

Figure 14: Using heatmap to show the correlation

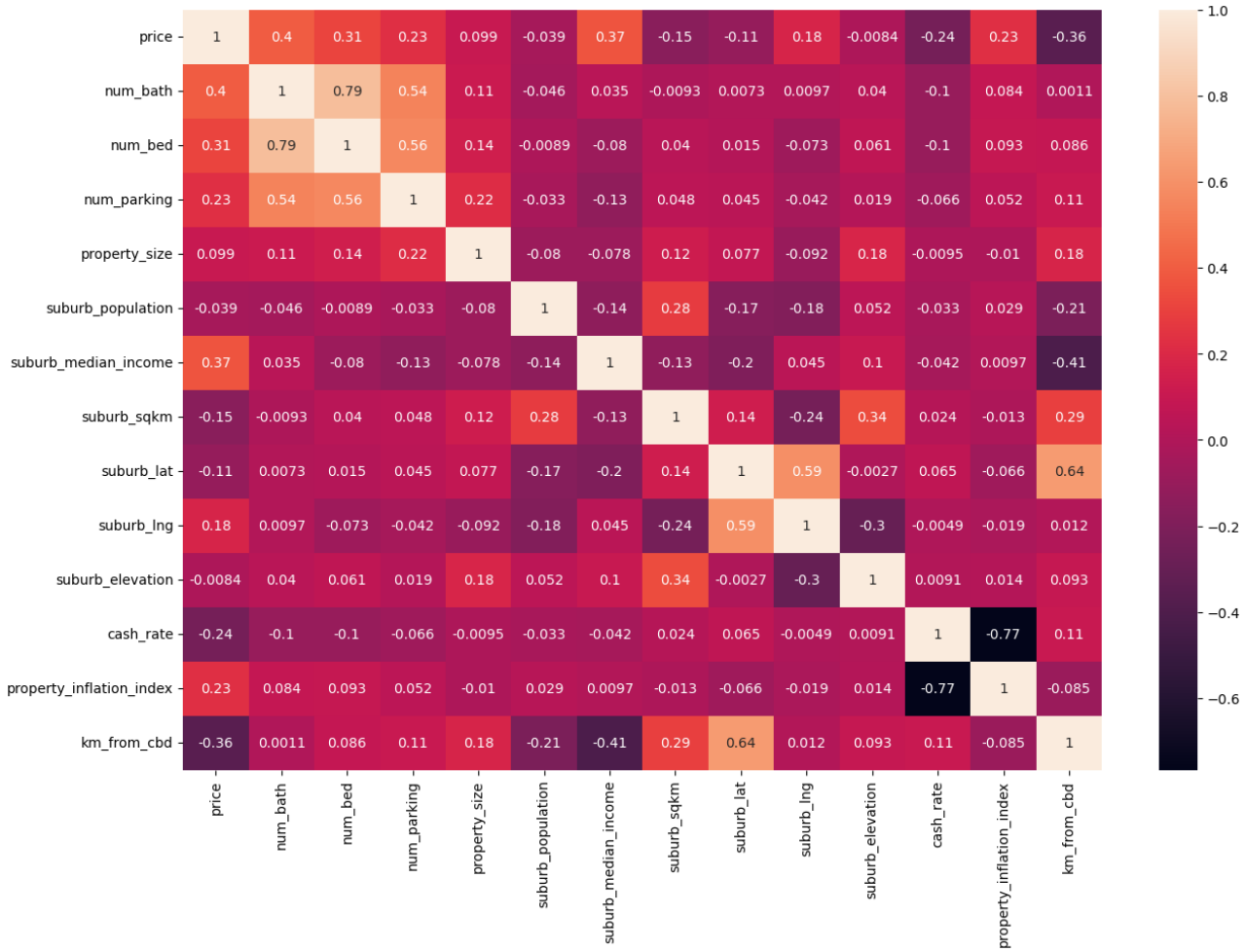


Figure 15: The heatmap of Sydney data

Additional datasets play a role in supporting some metrics to distinguish between 2 main datasets (Melbourne and Sydney) or adding missing features. In these

datasets, I will visualize an overview of data to take key features that can support our main dataset.

Postcode Data Overview

The postcode data plays a role in adding more columns for Sydney data because Sydney data doesn't have a postcode and council area column. This addition is crucial because these features have a decent correlation to the price house and these features can distinguish between Melbourne and Sydney data when merging 2 datasets.

3. Postcode Data Overview (Additional Dataset)

```
[ ] postcode_data = pd.read_csv('australian_postcodes.csv')
    postcode_data.head()
```

	id	postcode	locality	state	long	lat	dc	type	status	sa3	...	altitude	chargezone	phn_code	phn_name	lgaregion	lgacode	electorate	electoraterating	sed_code	sed_name
0	230	200	ANU	ACT	149.11900	-35.277700	NaN	NaN	Updated 3-Dec-2022	NaN	...	NaN	N2	NaN	NaN	Unincorporated ACT	89399.0	Durack	NaN	NaN	NaN
1	21820	200	Australian National University	ACT	149.11890	-35.277700	NaN	NaN	Updated 3-Dec-2022	NaN	...	NaN	N2	NaN	NaN	Unincorporated ACT	89399.0	Durack	NaN	NaN	NaN
2	232	800	DARWIN	NT	130.83668	-12.458684	NaN	NaN	Updated 3-Dec-2022	70101.0	...	NaN	NT1	PHN701	Northern Territory	Darwin Waterfront Precinct	71150.0	Solomon	Inner Metropolitan	70022.0	Port Da
3	24049	800	DARWIN CITY	NT	130.83668	-12.458684	NaN	NaN	Updated 3-Dec-2022	70101.0	...	NaN	NT1	PHN701	Northern Territory	Darwin Waterfront Precinct	71150.0	Solomon	Inner Metropolitan	70022.0	Port Darwin
4	233	801	DARWIN	NT	130.83668	-12.458684	NaN	NaN	Updated 3-Dec-2022	70101.0	...	NaN	NT1	PHN701	NaN	Darwin	71000.0	Lingiari	Rural	NaN	NaN

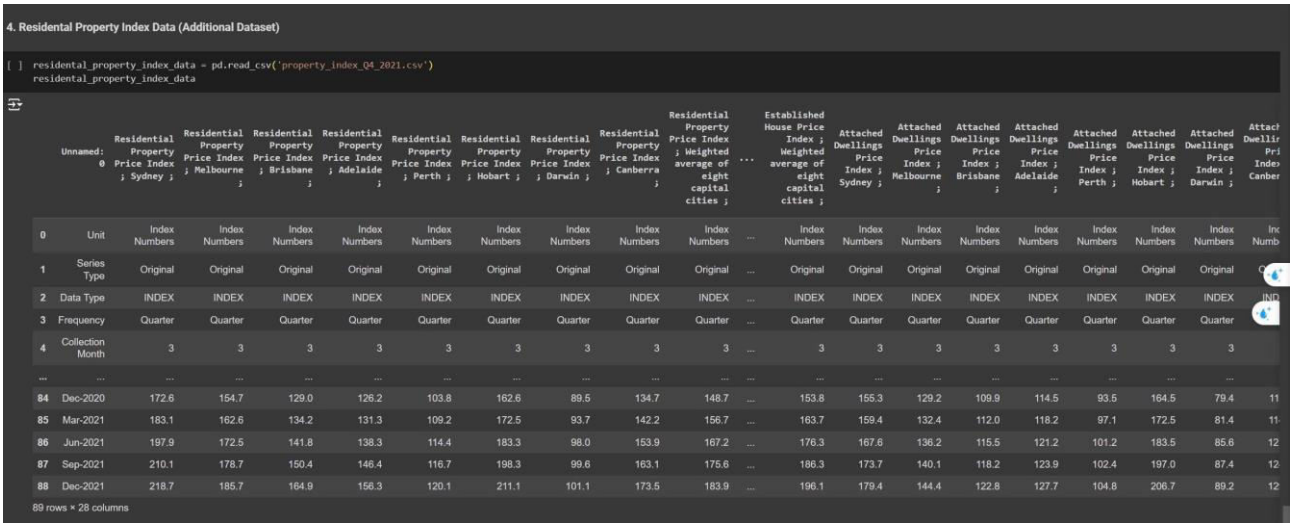
5 rows × 41 columns

Figure 16: Overview of postcode data

Residential Property Index Data

The residential property index data shows the residential property price index, established house index, and attached dwelling index in 8 big cities in Australia. These metrics are vital because the Melbourne and Sydney data contain 'date' attributes. These metrics can show the influence of these metrics on house prices in different periods. Residential Property Price Index affects the overall market, providing insight into whether house prices are generally rising or falling. The established House Price Index is more focused on older homes and indicates how demand for these properties is changing, often influenced by location and historical value. The attached Dwelling Price Index reflects the demand for apartments and townhouses, often impacted by urbanization, affordability, and lifestyle trends. Based on the description of each metric, I will take the residential

property price index and the attached dwelling price index because they have an essential relationship with house prices over time. For the established hour price index, I don't use this metric because I don't have the building age feature in Sydney data. Therefore, I will delete the year built in Melbourne data as well.



4. Residental Property Index Data (Additional Dataset)

```
residental_property_index_data = pd.read_csv('property_index_Q4_2021.csv')
residental_property_index_data
```

	Unnamed: 0	Residential Property Price Index ; Sydney ;	Residential Property Price Index ; Melbourne ;	Residential Property Price Index ; Brisbane ;	Residential Property Price Index ; Adelaide ;	Residential Property Price Index ; Perth ;	Residential Property Price Index ; Hobart ;	Residential Property Price Index ; Darwin ;	Residential Property Price Index ; Canberra ;	Residential Property Price Index ; Weighted average of eight capital cities ;	...	Established House Price Index ; Weighted average of eight capital cities ;	Attached Dwellings Price Index ; Sydney ;	Attached Dwellings Price Index ; Melbourne ;	Attached Dwellings Price Index ; Brisbane ;	Attached Dwellings Price Index ; Adelaide ;	Attached Dwellings Price Index ; Perth ;	Attached Dwellings Price Index ; Hobart ;	Attached Dwellings Price Index ; Darwin ;	Attac Dwelli Pri Inde Canbe
0	Unit	Index Numbers	Index Numbers	Index Numbers	Index Numbers	Index Numbers	Index Numbers	Index Numbers	Index Numbers	Index Numbers	...	Index Numbers	Index Numbers	Index Numbers	Index Numbers	Index Numbers	Index Numbers	Index Numbers	Index Numbers	Inc Numb
1	Series Type	Original	Original	Original	Original	Original	Original	Original	Original	Original	...	Original	Original	Original	Original	Original	Original	Original	Original	
2	Data Type	INDEX	INDEX	INDEX	INDEX	INDEX	INDEX	INDEX	INDEX	INDEX	...	INDEX	INDEX	INDEX	INDEX	INDEX	INDEX	INDEX	INDEX	IND
3	Frequency	Quarter	Quarter	Quarter	Quarter	Quarter	Quarter	Quarter	Quarter	Quarter	...	Quarter	Quarter	Quarter	Quarter	Quarter	Quarter	Quarter	Quarter	
4	Collection Month	3	3	3	3	3	3	3	3	3	...	3	3	3	3	3	3	3	3	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
84	Dec-2020	172.6	154.7	129.0	126.2	103.8	162.6	89.5	134.7	148.7	...	153.8	155.3	129.2	109.9	114.5	93.5	164.5	79.4	11
85	Mar-2021	183.1	162.6	134.2	131.3	109.2	172.5	93.7	142.2	156.7	...	163.7	159.4	132.4	112.0	118.2	97.1	172.5	81.4	11
86	Jun-2021	197.9	172.5	141.8	138.3	114.4	183.3	98.0	153.9	167.2	...	176.3	167.6	136.2	115.5	121.2	101.2	183.5	85.6	12
87	Sep-2021	210.1	178.7	150.4	146.4	116.7	198.3	99.6	163.1	175.6	...	186.3	173.7	140.1	118.2	123.9	102.4	197.0	87.4	12
88	Dec-2021	218.7	185.7	164.9	156.3	120.1	211.1	101.1	173.5	183.9	...	196.1	179.4	144.4	122.8	127.7	104.8	206.7	89.2	12

89 rows × 28 columns

Figure 17: Overview of the residential property index data

Financial Data Overview

The financial data shows the cash rate value from 2011 to 2023. This is useful data when I can know the value of money in each period. From that, I can know the value of the house in each period which affects to house price a lot.

5. Financial Data Overview (Additional Dataset)

```
melbourne_financial_data = pd.read_csv('cash_rate_melbourne.csv')
melbourne_financial_data
```

	date	value	vard	vardd
0	04/01/2011	4.75	title	Cash Rate Target
1	05/01/2011	4.75	narrow cat.	Interest Rates
2	06/01/2011	4.75	broad cat.	Interest Rates and Yields - Money Market - Daily
3	07/01/2011	4.75	unit	Per cent
4	10/01/2011	4.75	prices	-
...	...	...	...	...
3281	19/12/2023	4.35	NaN	NaN
3282	20/12/2023	4.35	NaN	NaN
3283	21/12/2023	4.35	NaN	NaN
3284	22/12/2023	4.35	NaN	NaN
3285	27/12/2023	4.35	NaN	NaN

3286 rows × 4 columns

Figure 18: Overview of financial data

b. Feature Engineering

Postcode Data Feature

Delete unnecessary columns and keep columns that will be used later. I want to add the postcode and council area column for Sydney data based on latitude and longitude. From that, I keep the ‘postcode’, ‘lat’, ‘long’, and ‘lgaregion’ for later use.

1. Postcode Data

```
[ ] postcode_data = postcode_data[['id', 'postcode', 'state', 'long', 'lat', 'lgaregion']]
    postcode_data.head()
```

	id	postcode	state	long	lat	lgaregion
0	230	200	ACT	149.11900	-35.277700	Unincorporated ACT
1	21820	200	ACT	149.11890	-35.277700	Unincorporated ACT
2	232	800	NT	130.83668	-12.458684	Darwin Waterfront Precinct
3	24049	800	NT	130.83668	-12.458684	Darwin Waterfront Precinct
4	233	801	NT	130.83668	-12.458684	Darwin

Figure 19: Drop unnecessary columns in postcode data

I use the unique() function to know the states of Australia in the ‘state’ feature, from that, I can filter the rows I need. Because Melbourne and Sydney are sequentially located in Victoria and New South Wales state, thus I filter 2 states and can get rows that contain Sydney and Melbourne postcode data.

```
[ ] print(postcode_data['state'].unique())

['ACT' 'NT' 'SA' 'WA' 'NSW' 'QLD' 'VIC' 'TAS']

[ ] postcode_data = postcode_data[postcode_data['state'].isin(['NSW', 'VIC'])]
    postcode_data.shape

(9142, 6)

    postcode_data
```

	id	postcode	state	long	lat	lgaregion
447	458	1001	NSW	151.268071	-33.794883	Sydney
448	459	1002	NSW	151.268071	-33.794883	Sydney
449	460	1003	NSW	151.268071	-33.794883	Sydney
450	461	1004	NSW	151.268071	-33.794883	Sydney
451	462	1005	NSW	151.268071	-33.794883	Sydney
...	...	...	...	...	...	...
18529	24130	8438	VIC	144.811079	-37.798099	South Gippsland
18530	24131	8511	VIC	144.811079	-37.798099	South Gippsland
18531	11331	8785	VIC	145.208504	-38.016114	Greater Dandenong
18532	24132	8785	VIC	145.208504	-38.016114	Greater Dandenong
18544	23878	9999	VIC	144.956776	-37.817403	Melbourne

9142 rows × 6 columns

Figure 20: Filtering necessary rows

Residential Property Index Feature

I have to show the first 10 rows because when I observe the data above, it shows that many columns have text inside. These texts are not necessary because they show irrelevant data that I don't need. Therefore, I delete the first 10 rows and now the table shows the data I need now (Figure 22).

2. Residental Property Index Data

```
[ ] first_10_rows = residental_property_index_data.iloc[:10]
    first_10_rows
```

	Unnamed: 0	Residential Property Price Index ; Sydney ;	Residential Property Price Index ; Melbourne ;	Residential Property Price Index ; Brisbane ;	Residential Property Price Index ; Adelaide ;	Residential Property Price Index ; Perth ;	Residential Property Price Index ; Hobart ;	Residential Property Price Index ; Darwin ;	Residential Property Price Index ; Canberra ;	Residential Property Price Index ; Weighted average of eight capital cities ;	...	Established House Price Index ; Weighted average of eight capital cities ;	Attached Dwellings Price Index ; Sydney ;	Attached Dwellings Price Index ; Melbourne ;	Attached Dwellings Price Index ; Brisbane ;	Attached Dwellings Price Index ; Adelaide ;	Attached Dwellings Price Index ; Perth ;	Attached Dwellings Price Index ; Hobart ;	Attached Dwellings Price Index ; Darwin
0	Unit	Index Numbers	Index Numbers	Index Numbers	Index Numbers	Index Numbers	Index Numbers	Index Numbers	Index Numbers	Index Numbers	...	Index Numbers	Index Numbers	Index Numbers	Index Numbers	Index Numbers	Index Numbers	Index Numbers	Index Numbe
1	Series Type	Original	Original	Original	Original	Original	Original	Original	Original	Original	...	Original	Original	Original	Original	Original	Original	Original	Origina
2	Data Type	INDEX	INDEX	INDEX	INDEX	INDEX	INDEX	INDEX	INDEX	INDEX	...	INDEX	INDEX	INDEX	INDEX	INDEX	INDEX	INDEX	IN
3	Frequency	Quarter	Quarter	Quarter	Quarter	Quarter	Quarter	Quarter	Quarter	Quarter	...	Quarter	Quarter	Quarter	Quarter	Quarter	Quarter	Quarter	Q
4	Collection Month	3	3	3	3	3	3	3	3	3	...	3	3	3	3	3	3	3	3
5	Series Start	Sep-2003	Sep-2003	Sep-2003	Sep-2003	Sep-2003	Sep-2003	Sep-2003	Sep-2003	Sep-2003	...	Mar-2002	Sep-2003	Sep-2003	Sep-2003	Sep-2003	Sep-2003	Sep-2003	Sep-200
6	Series End	Dec-2021	Dec-2021	Dec-2021	Dec-2021	Dec-2021	Dec-2021	Dec-2021	Dec-2021	Dec-2021	...	Dec-2021	Dec-2021	Dec-2021	Dec-2021	Dec-2021	Dec-2021	Dec-2021	Dec-202
7	No. Obs	74	74	74	74	74	74	74	74	74	...	80	74	74	74	74	74	74	7
8	Series ID	A83728383L	A83728392R	A83728401F	A83728410J	A83728419C	A83728428F	A83728437J	A83728446K	A83728455L	...	A83728456R	A83728385T	A83728394V	A83728403K	A83728412L	A83728421R	A83728430T	A8372843
9	Mar-2002	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	52.1	NaN	NaN	NaN	NaN	NaN	NaN	Na

10 rows × 28 columns

Figure 21: Show the first 10 rows of data

```
residental_property_index_data = residental_property_index_data.drop(residental_property_index_data.index[:9])
residental_property_index_data.head()
```

	Unnamed: 0	Residential Property Price Index ; Sydney ;	Residential Property Price Index ; Melbourne ;	Residential Property Price Index ; Brisbane ;	Residential Property Price Index ; Adelaide ;	Residential Property Price Index ; Perth ;	Residential Property Price Index ; Hobart ;	Residential Property Price Index ; Darwin ;	Residential Property Price Index ; Canberra ;	Residential Property Price Index ; Weighted average of eight capital cities ;	...	Established House Price Index ; Weighted average of eight capital cities ;	Attached Dwellings Price Index ; Sydney ;	Attached Dwellings Price Index ; Melbourne ;	Attached Dwellings Price Index ; Brisbane ;	Attached Dwellings Price Index ; Adelaide ;	Attached Dwellings Price Index ; Perth ;	Attached Dwellings Price Index ; Hobart ;	Attached Dwellings Price Index ; Darwin ;	Attach Dwellin Pri Index Can
9	Mar-2002	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	52.1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
10	Jun-2002	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	55.2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
11	Sep-2002	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	57.5	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
12	Dec-2002	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	59.6	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
13	Mar-2003	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	61.2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na

5 rows × 28 columns

Figure 22: Delete the irrelevant data

After that, I continue to filter the data. I filter the column that has ‘Melbourne’ and ‘Sydney’ words inside to get the data relating to 2 cities. Then, I change the name of the column containing the date and modify to pandas format for further use.

```
[ ] residental_property_index_data = residental_property_index_data[['Unnamed: 0'] + [col for col in residental_property_index_data.columns if 'Sydney' in col or 'Melbourne' in col]]

    residental_property_index_data.head()
```

	Unnamed: 0	Residential Property Price Index ; Sydney ;	Residential Property Price Index ; Melbourne ;	Established House Price Index ; Sydney ;	Established House Price Index ; Melbourne ;	Attached Dwellings Price Index ; Sydney ;	Attached Dwellings Price Index ; Melbourne ;
9	Mar-2002	NaN	NaN	66.4	47.7	NaN	NaN
10	Jun-2002	NaN	NaN	71.4	50.7	NaN	NaN
11	Sep-2002	NaN	NaN	74.8	51.9	NaN	Na
12	Dec-2002	NaN	NaN	77.5	53.1	NaN	NaN
13	Mar-2003	NaN	NaN	78.5	54.2	NaN	NaN

```
residental_property_index_data['Unnamed: 0'] = pd.to_datetime(residental_property_index_data['Unnamed: 0'], format='%b-%Y')
residental_property_index_data.rename(columns={'Unnamed: 0': 'Date'}, inplace=True)
residental_property_index_data.head()
```

	Date	Residential Property Price Index ; Sydney ;	Residential Property Price Index ; Melbourne ;	Established House Price Index ; Sydney ;	Established House Price Index ; Melbourne ;	Attached Dwellings Price Index ; Sydney ;	Attached Dwellings Price Index ; Melbourne ;
9	2002-03-01	NaN	NaN	66.4	47.7	NaN	NaN
10	2002-06-01	NaN	NaN	71.4	50.7	NaN	NaN
11	2002-09-01	NaN	NaN	74.8	51.9	NaN	Na
12	2002-12-01	NaN	NaN	77.5	53.1	NaN	NaN
13	2003-03-01	NaN	NaN	78.5	54.2	NaN	NaN

Figure 23: Filter and change the data format

I visualize the missing data through a bar chart to see the overview of missing data. The bar chart shows that there are only a few missing data, thus I decide to delete the rows containing missing values (Figure 25).

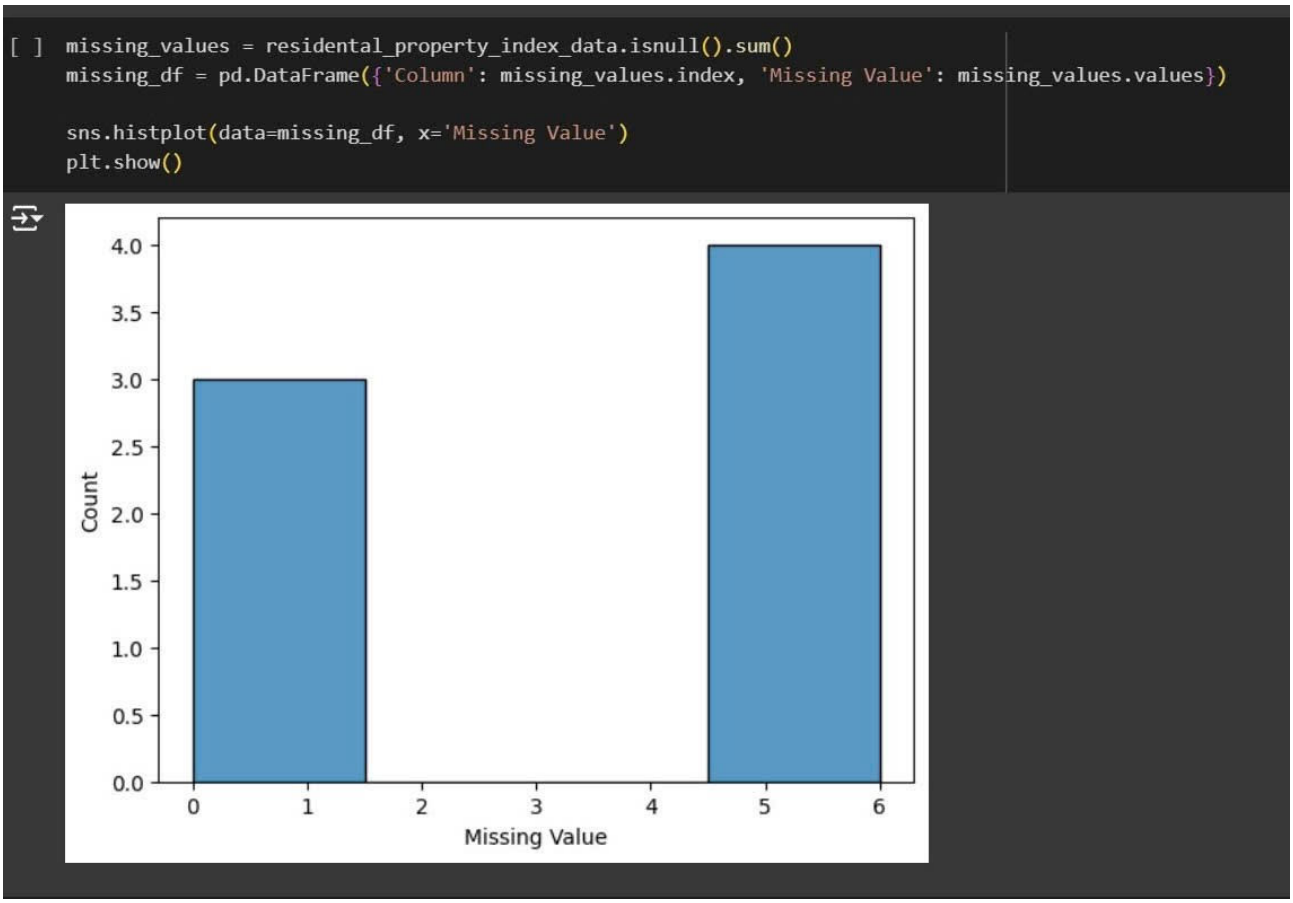


Figure 24: Bar chart shows the missing values

```
residental_property_index_data.dropna(inplace=True)
residental_property_index_data.isnull().sum()
```

	0
Date	0
Residential Property Price Index ; Sydney ;	0
Residential Property Price Index ; Melbourne ;	0
Established House Price Index ; Sydney ;	0
Established House Price Index ; Melbourne ;	0
Attached Dwellings Price Index ; Sydney ;	0
Attached Dwellings Price Index ; Melbourne ;	0

dtype: int64

Figure 25: Drop rows containing missing values

I filter the data and only take the rows starting from 2016 because the Melbourne and Sydney data both start from 2016. Then, I split the table into 2 small tables that carry the residential property index data of Melbourne and Sydney

```
residental_property_index_data = residental_property_index_data[residental_property_index_data['Date'].dt.year >= 2016]
residental_property_index_data.head()
```

	Date	Residential Property Price Index ; Sydney ;	Residential Property Price Index ; Melbourne ;	Established House Price Index ; Sydney ;	Established House Price Index ; Melbourne ;	Attached Dwellings Price Index ; Sydney ;	Attached Dwellings Price Index ; Melbourne ;
65	2016-03-01	150.9	127.3	154.8	131.6	143.2	114.0
66	2016-06-01	155.2	130.7	159.7	135.7	146.1	115.5
67	2016-09-01	159.3	132.9	164.4	138.5	149.2	11
68	2016-12-01	167.6	140.0	174.4	146.8	153.8	119.2
69	2017-03-01	172.7	144.4	179.5	152.8	159.1	118.4

```
residental_property_index_melbourne = residental_property_index_data[['Date'] + [col for col in residental_property_index_data.columns if 'Melbourne' in col]].copy()
residental_property_index_sydney = residental_property_index_data[['Date'] + [col for col in residental_property_index_data.columns if 'Sydney' in col]].copy()
```

Figure 26: Filter data by year and split table

I change the name of each column and show the data after split into 2 different

tables (Figure 27 and Figure 28)

```
residental_property_index_melbourne.rename(
    columns={
        'Residential Property Price Index ;  Melbourne ;': 'Residential Property Price Index',
        'Established House Price Index ;  Melbourne ;': 'Established House Price Index',
        'Attached Dwellings Price Index ;  Melbourne ;': 'Attached Dwellings Price Index'
    },
    inplace=True
)

residental_property_index_melbourne.head()
```

	Date	Residential Property Price Index	Established House Price Index	Attached Dwellings Price Index
65	2016-03-01	127.3	131.6	114.0
66	2016-06-01	130.7	135.7	115.5
67	2016-09-01	132.9	138.5	115.9
68	2016-12-01	140.0	146.8	119.2
69	2017-03-01	144.4	152.8	118.4

Figure 27: Residential property index Melbourne data starting from 2016

```
residental_property_index_sydney.rename(
    columns={
        'Residential Property Price Index ;  Sydney ;': 'Residential Property Price Index',
        'Established House Price Index ;  Sydney ;': 'Established House Price Index',
        'Attached Dwellings Price Index ;  Sydney ;': 'Attached Dwellings Price Index'
    },
    inplace=True
)

residental_property_index_sydney.head()
```

	Date	Residential Property Price Index	Established House Price Index	Attached Dwellings Price Index
65	2016-03-01	150.9	154.8	143.2
66	2016-06-01	155.2	159.7	146.1
67	2016-09-01	159.3	164.4	149.2
68	2016-12-01	167.6	174.4	153.8
69	2017-03-01	172.7	179.5	159.1

Figure 28: Residential property index Sydney data starting from 2016

Financial Data

In Figure 29, I check the duplicated rows, and null values to ensure the data doesn't contain noise. There are only null values in the 'vard' and 'vardd' columns and these are not necessary columns because they don't express any meaning to the data (cash rate). Thus, I drop 2 columns to avoid noise.

3. Financial Data

```
[ ] melbourne_financial_data.columns
```

```
Index(['date', 'value', 'vard', 'vardd'], dtype='object')
```

```
[ ] melbourne_financial_data.duplicated().sum()
```

```
0
```

```
[ ] melbourne_financial_data.isnull().sum()
```

	0
date	0
value	0
vard	3278
vardd	3278

dtype: int64

```
[ ] melbourne_financial_data.drop(columns=['vard', 'vardd'], inplace=True)
    melbourne_financial_data.rename(columns={'value': 'CashRate'}, inplace=True)

    melbourne_financial_data.head()
```

	date	CashRate
0	04/01/2011	4.75
1	05/01/2011	4.75
2	06/01/2011	4.75
3	07/01/2011	4.75
4	10/01/2011	4.75

Figure 29: Handle noise data

Then, I change the date value into pandas format for further use and filter the data to start from 2016.

```
[ ] import pandas as pd

    melbourne_financial_data['date'] = pd.to_datetime(melbourne_financial_data['date'], format='%d/%m/%Y')

    melbourne_financial_data = melbourne_financial_data[(melbourne_financial_data['date'].dt.year >= 2016) & (melbourne_financial_data['date'].dt.year <= 2021)]

    melbourne_financial_data.head()
```

	date	CashRate
1265	2016-01-04	2.0
1266	2016-01-05	2.0
1267	2016-01-06	2.0
1268	2016-01-07	2.0
1269	2016-01-08	2.0

Figure 30: Change date format and filter data

I visualize the scatterplot of the cash rate to check the number of values that appear in each year. From the scatterplot, this shows that the cash rate in each year doesn't change much, but the value will change by year. This can be useful when I want to specify the difference in price each year when combining 2 datasets with 2 different periods (Melbourne data from 2016-2018, Sydney data from 2016-2022)

```
[ ] melbourne_financial_data['CashRate'].unique()

array([2.  , 1.75, 1.5 , 1.25, 1.  , 0.75, 0.5 , 0.25, 0.1 ])

sns.scatterplot(data=melbourne_financial_data, x=melbourne_financial_data['date'].dt.year, y='CashRate')
plt.show()
```

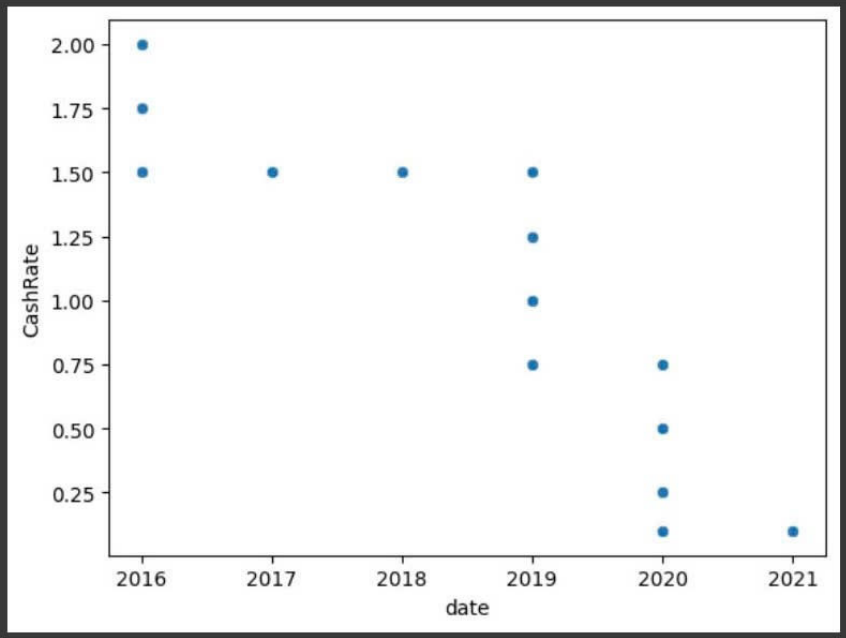
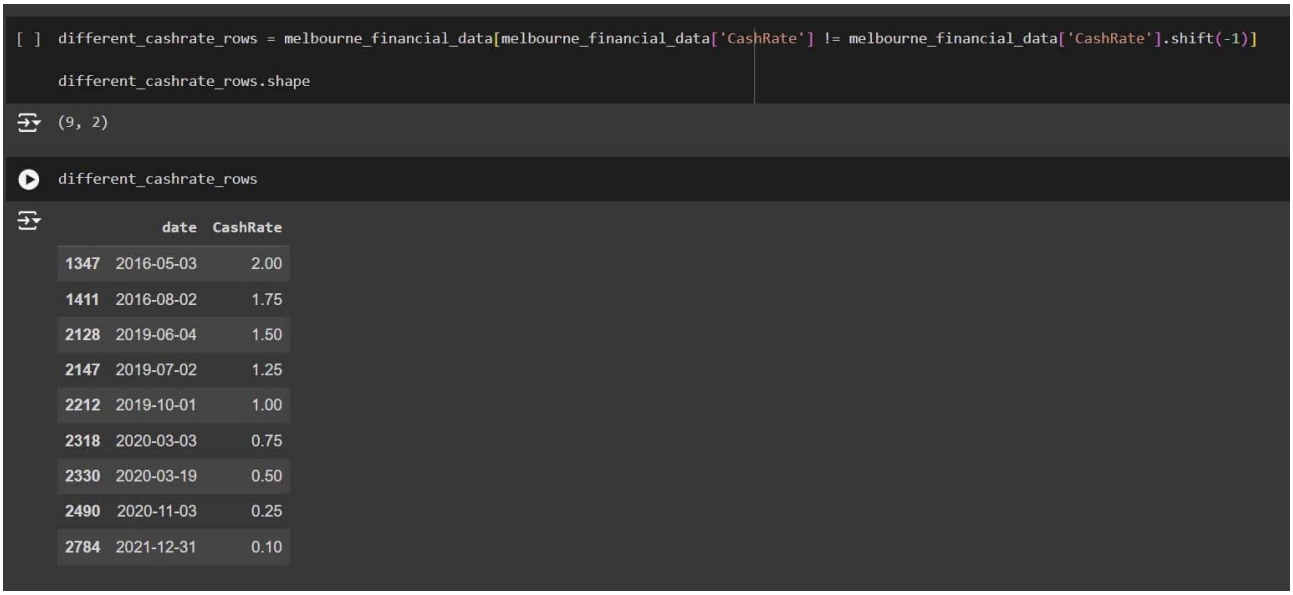


Figure 31: Visualize the cash rate with the scatterplot chart

I create a table for identifying and filtering rows in the financial data where the 'CashRate' changes compared to the next row. This makes every row play a role as a flag when the cash rate value changes. This will be used later.



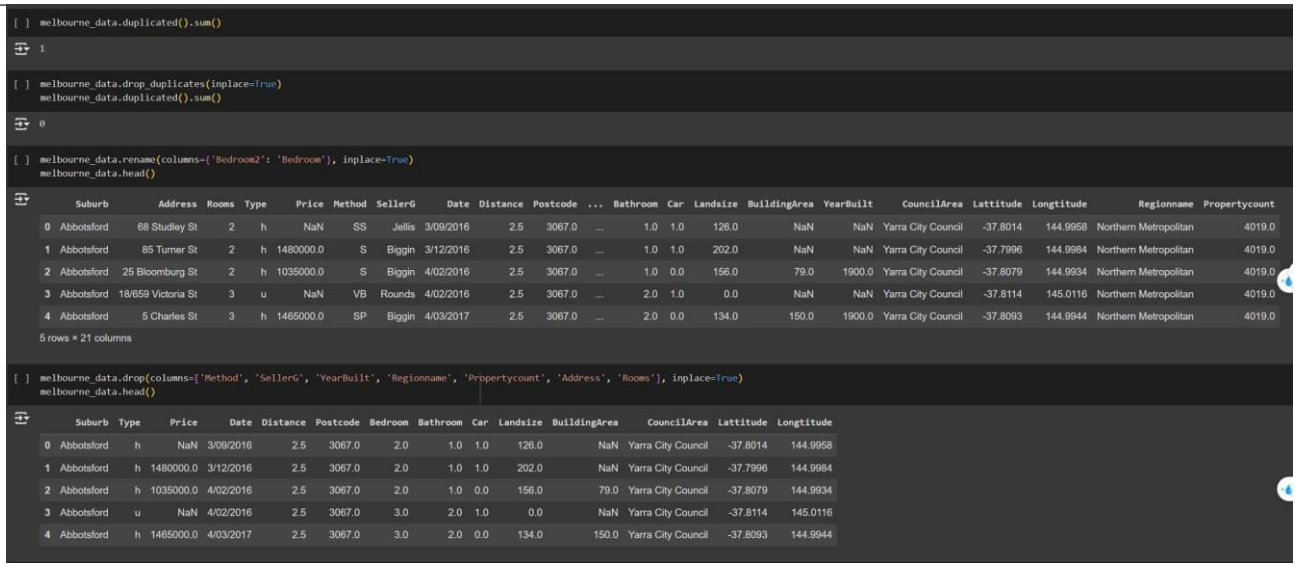
```
[ ] different_cashrate_rows = melbourne_financial_data[melbourne_financial_data['CashRate'] != melbourne_financial_data['CashRate'].shift(-1)]
    different_cashrate_rows.shape
(9, 2)
different_cashrate_rows
```

	date	CashRate
1347	2016-05-03	2.00
1411	2016-08-02	1.75
2128	2019-06-04	1.50
2147	2019-07-02	1.25
2212	2019-10-01	1.00
2318	2020-03-03	0.75
2330	2020-03-19	0.50
2490	2020-11-03	0.25
2784	2021-12-31	0.10

Figure 32: Flag value table

Melbourne Data Feature

First, I check for the duplicated rows and remove duplicated rows to avoid noise. Then, I rename the column ‘bedroom’ and drop unnecessary columns. Most of the columns have a low correlation to house prices based on the heatmap and removing them is for merging the dataset later (the Sydney dataset doesn’t have these columns). After that, I add a new column named city and filled it with ‘Melbourne’ to identify when merging 2 main datasets



```
[ ] melbourne_data.duplicated().sum()
1
[ ] melbourne_data.drop_duplicates(inplace=True)
    melbourne_data.duplicated().sum()
0
[ ] melbourne_data.rename(columns={'Bedroom2': 'Bedroom'}, inplace=True)
    melbourne_data.head()
```

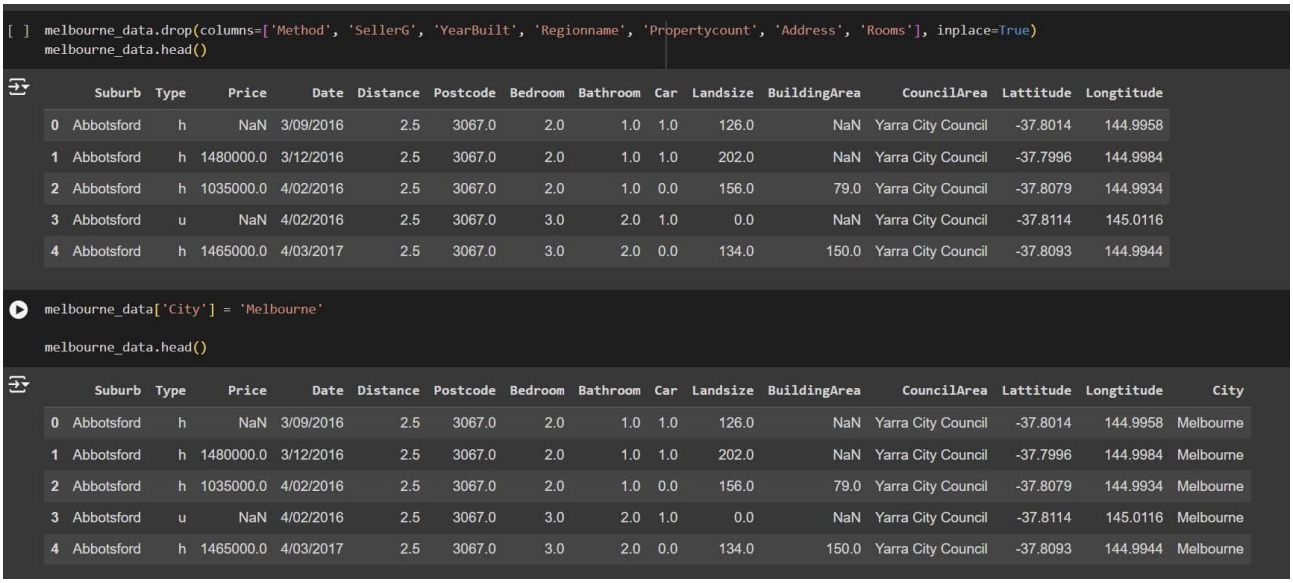
	Suburb	Address	Rooms	Type	Price	Method	SellerG	Date	Distance	Postcode	...	Bathroom	Car	Landsize	BuildingArea	YearBuilt	CouncilArea	Lattitude	Longtitude	Regionname	Propertycount
0	Abbotsford	68 Studley St	2	h	NaN	SS	Jellis	3/09/2016	2.5	3067.0	...	1.0	1.0	126.0	NaN	NaN	Yarra City Council	-37.8014	144.9958	Northern Metropolitan	4019.0
1	Abbotsford	85 Turner St	2	h	1480000.0	S	Biggin	3/12/2016	2.5	3067.0	...	1.0	1.0	202.0	NaN	NaN	Yarra City Council	-37.7996	144.9984	Northern Metropolitan	4019.0
2	Abbotsford	25 Bloomburg St	2	h	1035000.0	S	Biggin	4/02/2016	2.5	3067.0	...	1.0	0.0	156.0	79.0	1900.0	Yarra City Council	-37.8079	144.9934	Northern Metropolitan	4019.0
3	Abbotsford	18/659 Victoria St	3	u	NaN	VB	Rounds	4/02/2016	2.5	3067.0	...	2.0	1.0	0.0	NaN	NaN	Yarra City Council	-37.8114	145.0116	Northern Metropolitan	4019.0
4	Abbotsford	5 Charles St	3	h	1465000.0	SP	Biggin	4/03/2017	2.5	3067.0	...	2.0	0.0	134.0	150.0	1900.0	Yarra City Council	-37.8093	144.9944	Northern Metropolitan	4019.0

5 rows × 21 columns

```
[ ] melbourne_data.drop(columns=['Method', 'SellerG', 'YearBuilt', 'Regionname', 'Propertycount', 'Address', 'Rooms'], inplace=True)
    melbourne_data.head()
```

	Suburb	Type	Price	Date	Distance	Postcode	Bedroom	Bathroom	Car	Landsize	BuildingArea	CouncilArea	Lattitude	Longtitude
0	Abbotsford	h	NaN	3/09/2016	2.5	3067.0	2.0	1.0	1.0	126.0	NaN	Yarra City Council	-37.8014	144.9958
1	Abbotsford	h	1480000.0	3/12/2016	2.5	3067.0	2.0	1.0	1.0	202.0	NaN	Yarra City Council	-37.7996	144.9984
2	Abbotsford	h	1035000.0	4/02/2016	2.5	3067.0	2.0	1.0	0.0	156.0	79.0	Yarra City Council	-37.8079	144.9934
3	Abbotsford	u	NaN	4/02/2016	2.5	3067.0	3.0	2.0	1.0	0.0	NaN	Yarra City Council	-37.8114	145.0116
4	Abbotsford	h	1465000.0	4/03/2017	2.5	3067.0	3.0	2.0	0.0	134.0	150.0	Yarra City Council	-37.8093	144.9944

Figure 33: Handle duplicated rows and drop unnecessary columns



```
[ ] melbourne_data.drop(columns=['Method', 'SellerG', 'YearBuilt', 'Regionname', 'Propertycount', 'Address', 'Rooms'], inplace=True)
    melbourne_data.head()
```

	Suburb	Type	Price	Date	Distance	Postcode	Bedroom	Bathroom	Car	Landsize	BuildingArea	CouncilArea	Lattitude	Longtitude
0	Abbotsford	h	NaN	3/09/2016	2.5	3067.0	2.0	1.0	1.0	126.0	NaN	Yarra City Council	-37.8014	144.9958
1	Abbotsford	h	1480000.0	3/12/2016	2.5	3067.0	2.0	1.0	1.0	202.0	NaN	Yarra City Council	-37.7996	144.9984
2	Abbotsford	h	1035000.0	4/02/2016	2.5	3067.0	2.0	1.0	0.0	156.0	79.0	Yarra City Council	-37.8079	144.9934
3	Abbotsford	u	NaN	4/02/2016	2.5	3067.0	3.0	2.0	1.0	0.0	NaN	Yarra City Council	-37.8114	145.0116
4	Abbotsford	h	1465000.0	4/03/2017	2.5	3067.0	3.0	2.0	0.0	134.0	150.0	Yarra City Council	-37.8093	144.9944

```
melbourne_data['City'] = 'Melbourne'

melbourne_data.head()
```

	Suburb	Type	Price	Date	Distance	Postcode	Bedroom	Bathroom	Car	Landsize	BuildingArea	CouncilArea	Lattitude	Longtitude	City
0	Abbotsford	h	NaN	3/09/2016	2.5	3067.0	2.0	1.0	1.0	126.0	NaN	Yarra City Council	-37.8014	144.9958	Melbourne
1	Abbotsford	h	1480000.0	3/12/2016	2.5	3067.0	2.0	1.0	1.0	202.0	NaN	Yarra City Council	-37.7996	144.9984	Melbourne
2	Abbotsford	h	1035000.0	4/02/2016	2.5	3067.0	2.0	1.0	0.0	156.0	79.0	Yarra City Council	-37.8079	144.9934	Melbourne
3	Abbotsford	u	NaN	4/02/2016	2.5	3067.0	3.0	2.0	1.0	0.0	NaN	Yarra City Council	-37.8114	145.0116	Melbourne
4	Abbotsford	h	1465000.0	4/03/2017	2.5	3067.0	3.0	2.0	0.0	134.0	150.0	Yarra City Council	-37.8093	144.9944	Melbourne

Figure 34: Add city column and fill with ‘Melbourne’

Next, I merge 2 datasets (Melbourne data and Financial data) by date to get the cash rate by day. However, after merging there are many Nan values (96.55%). This happens because the financial data doesn’t contain all the dates in a year.

```
[ ] melbourne_data['Date'] = pd.to_datetime(melbourne_data['Date'], format='%d/%m/%Y')
    melbourne_data.head()

    melbourne_data = pd.merge(
        melbourne_data,
        melbourne_financial_data[['date', 'CashRate']],
        left_on='Date',
        right_on='date',
        how='left'
    )

    melbourne_data = melbourne_data.drop(columns=['date'])
    melbourne_data
```

	Suburb	Type	Price	Date	Distance	Postcode	Bedroom	Bathroom	Car	Landsize	BuildingArea	CouncilArea	Lattitude	Longtitude	City	CashRate
0	Abbotsford	h	NaN	2016-09-03	2.5	3067.0	2.0	1.0	1.0	126.0	NaN	Yarra City Council	-37.80140	144.99580	Melbourne	NaN
1	Abbotsford	h	1480000.0	2016-12-03	2.5	3067.0	2.0	1.0	1.0	202.0	NaN	Yarra City Council	-37.79960	144.99840	Melbourne	NaN
2	Abbotsford	h	1035000.0	2016-02-04	2.5	3067.0	2.0	1.0	0.0	156.0	79.0	Yarra City Council	-37.80790	144.99340	Melbourne	2.0
3	Abbotsford	u	NaN	2016-02-04	2.5	3067.0	3.0	2.0	1.0	0.0	NaN	Yarra City Council	-37.81140	145.01160	Melbourne	2.0
4	Abbotsford	h	1465000.0	2017-03-04	2.5	3067.0	3.0	2.0	0.0	134.0	150.0	Yarra City Council	-37.80930	144.99440	Melbourne	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
34851	Yarraville	h	1480000.0	2018-02-24	6.3	3013.0	4.0	1.0	3.0	593.0	NaN	Maribyrnong City Council	-37.81053	144.88467	Melbourne	NaN
34852	Yarraville	h	888000.0	2018-02-24	6.3	3013.0	2.0	2.0	1.0	98.0	104.0	Maribyrnong City Council	-37.81551	144.88826	Melbourne	NaN
34853	Yarraville	t	705000.0	2018-02-24	6.3	3013.0	2.0	1.0	2.0	220.0	120.0	Maribyrnong City Council	-37.82286	144.87856	Melbourne	NaN
34854	Yarraville	h	1140000.0	2018-02-24	6.3	3013.0	NaN	NaN	NaN	NaN	NaN	Maribyrnong City Council	NaN	NaN	Melbourne	NaN
34855	Yarraville	h	1020000.0	2018-02-24	6.3	3013.0	2.0	1.0	0.0	250.0	103.0	Maribyrnong City Council	-37.81810	144.89351	Melbourne	NaN

34856 rows × 16 columns

```
[ ] melbourne_data['CashRate'].isnull().sum() / melbourne_data.shape[0] * 100
```

```
96.5515627955015
```

Figure 35: Merge financial data and Melbourne data by date

This issue can be addressed by using a flag table (Figure 32 and Figure 36). I add the first date of the year 2016 to compare. After that, all the rows containing Nan value in Melbourne data will be transmitted into a for loop to compare the date. If a row is in the range of the date that is specified, this row will be filled with the cash rate value in the end value of the range. For instance, if there is a row with the day sold as 2016-05-09, this row satisfies the condition in the range of 2016-05-03 to 2016-08-02 and it will be filled with the cash rate value of the row containing 2016-05-03 (1.75).

```
[ ]  melbourne_data_cop = melbourne_data.copy()
     different_cashrate_rows_cop = different_cashrate_rows.copy()

     new_row = pd.DataFrame({
         'date': [pd.to_datetime('2016-01-01')],
         'CashRate': [2]
     })

     different_cashrate_rows_cop = pd.concat([new_row, different_cashrate_rows_cop]).reset_index(drop=True)


     different_cashrate_rows_cop
```

	date	CashRate
0	2016-01-01	2.00
1	2016-05-03	2.00
2	2016-08-02	1.75
3	2019-06-04	1.50
4	2019-07-02	1.25
5	2019-10-01	1.00
6	2020-03-03	0.75
7	2020-03-19	0.50
8	2020-11-03	0.25
9	2021-12-31	0.10

```
[ ]  for i in range(different_cashrate_rows_cop.shape[0] - 1):
         mask = (melbourne_data_cop['Date'] >= different_cashrate_rows_cop['date'][i]) & \
                (melbourne_data_cop['Date'] <= different_cashrate_rows_cop['date'][i + 1]) & \
                (melbourne_data_cop['CashRate'].isna())

         melbourne_data_cop.loc[mask, 'CashRate'] = different_cashrate_rows_cop['CashRate'].iloc[i+1]

     melbourne_data_cop['CashRate'].isnull().sum()
```

```
0
```

Figure 36: Fill the row containing the Nan value in the cash rate column

After that, I use the same method with the Melbourne residential property index data because the data is recorded by the quarter.

```
[ ] melbourne_data_cop2 = melbourne_data_cop.copy()
    residental_property_index_melbourne_cop = residental_property_index_melbourne.copy()

    new_row = pd.DataFrame({
        'Date': [pd.to_datetime('2021-12-31')],
        'Residential Property Price Index': [185.7],
        'Established House Price Index': [200.1],
        'Attached Dwellings Price Index': [144.4]
    })

    first_row = pd.DataFrame({
        'Date': [pd.to_datetime('2016-01-01')],
        'Residential Property Price Index': [127.3],
        'Established House Price Index': [131.6],
        'Attached Dwellings Price Index': [114.0]
    })

    residental_property_index_melbourne_cop = pd.concat([first_row, residental_property_index_melbourne_cop]).reset_index(drop=True)
    residental_property_index_melbourne_cop = pd.concat([residental_property_index_melbourne_cop, new_row]).reset_index(drop=True)


    residental_property_index_melbourne_cop
```

	Date	Residential Property Price Index	Established House Price Index	Attached Dwellings Price Index
0	2016-01-01	127.3	131.6	114.0
1	2016-03-01	127.3	131.6	114.0
2	2016-06-01	130.7	135.7	115.5
3	2016-09-01	132.9	138.5	115.9
4	2016-12-01	140.0	146.8	119.2
5	2017-03-01	144.4	152.8	118.4
6	2017-06-01	148.7	157.4	121.6
7	2017-09-01	150.4	159.9	121.1
8	2017-12-01	154.3	164.0	124.2

Figure 37: Melbourne residential property index data

```
[ ] for i in range(residental_property_index_melbourne_cop.shape[0] - 1):
        mask = (melbourne_data_cop2['Date'] >= residental_property_index_melbourne_cop['Date'][i]) & \
               (melbourne_data_cop2['Date'] <= residental_property_index_melbourne_cop['Date'][i + 1]) & \
               (melbourne_data_cop2['Residential Property Price Index'].isna()) & \
               (melbourne_data_cop2['Attached Dwellings Price Index'].isna())

        melbourne_data_cop2.loc[mask, 'Residential Property Price Index'] = float(residental_property_index_melbourne_cop['Residential Property Price Index'].iloc[i+1])
        melbourne_data_cop2.loc[mask, 'Attached Dwellings Price Index'] = float(residental_property_index_melbourne_cop['Attached Dwellings Price Index'].iloc[i+1])

    print(melbourne_data_cop2['Residential Property Price Index'].isnull().sum())
    print(melbourne_data_cop2['Attached Dwellings Price Index'].isnull().sum())
```

```
0
0
```

Figure 38: Fill the data with Melbourne residential property index data

Then, I split the date column into 3 columns day sold, month sold, and year sold and these become numerical columns for giving these as input to the machine learning model. After that, I handle the text column in lowercase and remove the blank for label encoding later. If not doing this, the numerical values after encoding can be up to 100k and this is not a good thing.

```
[ ] melbourne_data_cop2['Date'] = pd.to_datetime(melbourne_data_cop2['Date'], format='%d/%m/%Y')

    melbourne_data_cop2['DaySold'] = melbourne_data_cop2['Date'].dt.day
    melbourne_data_cop2['MonthSold'] = melbourne_data_cop2['Date'].dt.month
    melbourne_data_cop2['YearSold'] = melbourne_data_cop2['Date'].dt.year

    melbourne_data_cop2 = melbourne_data_cop2.drop(columns=['Date'])
    melbourne_data_cop2.head()
```

	Suburb	Type	Price	Distance	Postcode	Bedroom	Bathroom	Car	Landsize	BuildingArea	CouncilArea	Lattitude	Longtitude	City	CashRate	Residential Property Price Index	Attached Dwellings Price Index	DaySold	MonthSold	YearSold
0	Abbotsford	h	NaN	2.5	3067.0	2.0	1.0	1.0	126.0	NaN	Yarra City Council	-37.8014	144.9958	Melbourne	1.5	140.0	119.2	3	9	2016
1	Abbotsford	h	1480000.0	2.5	3067.0	2.0	1.0	1.0	202.0	NaN	Yarra City Council	-37.7996	144.9984	Melbourne	1.5	144.4	118.4	3	12	2016
2	Abbotsford	h	1035000.0	2.5	3067.0	2.0	1.0	0.0	156.0	79.0	Yarra City Council	-37.8079	144.9934	Melbourne	2.0	127.3	114.0	4	2	2016
3	Abbotsford	u	NaN	2.5	3067.0	3.0	2.0	1.0	0.0	NaN	Yarra City Council	-37.8114	145.0116	Melbourne	2.0	127.3	114.0	4	2	2016
4	Abbotsford	h	1465000.0	2.5	3067.0	3.0	2.0	0.0	134.0	150.0	Yarra City Council	-37.8093	144.9944	Melbourne	1.5	148.7	121.6	4	3	2017

```
[ ] for i in melbourne_data_cop2.select_dtypes(include='object').columns:
        melbourne_data_cop2['CouncilArea'] = melbourne_data_cop2['CouncilArea'].str.replace('Council', '', regex=False)
        melbourne_data_cop2[i] = melbourne_data_cop2[i].str.lower()
        melbourne_data_cop2[i] = melbourne_data_cop2[i].str.strip()
        melbourne_data_cop2[i] = melbourne_data_cop2[i].str.replace(' ', '')

    melbourne_data_cop2.head()
```

	Suburb	Type	Price	Distance	Postcode	Bedroom	Bathroom	Car	Landsize	BuildingArea	CouncilArea	Lattitude	Longtitude	City	CashRate	Residential Property Price Index	Attached Dwellings Price Index	DaySold	MonthSold	YearSold
0	abbotsford	h	NaN	2.5	3067.0	2.0	1.0	1.0	126.0	NaN	yarracity	-37.8014	144.9958	melbourne	1.5	140.0	119.2	3	9	2016
1	abbotsford	h	1480000.0	2.5	3067.0	2.0	1.0	1.0	202.0	NaN	yarracity	-37.7996	144.9984	melbourne	1.5	144.4	118.4	3	12	2016
2	abbotsford	h	1035000.0	2.5	3067.0	2.0	1.0	0.0	156.0	79.0	yarracity	-37.8079	144.9934	melbourne	2.0	127.3	114.0	4	2	2016
3	abbotsford	u	NaN	2.5	3067.0	3.0	2.0	1.0	0.0	NaN	yarracity	-37.8114	145.0116	melbourne	2.0	127.3	114.0	4	2	2016

Figure 39: Split date and handle text columns

Sydney Data Feature

First, I check the duplicated rows but there are no duplicated rows so I move to the next section. After that, I drop unnecessary columns to merge the dataset. Although there are some important columns such as suburb median income and property inflation index, I still have to delete them because I can't find the same data in Melbourne. Instead, I add some additional metrics like cash rate, residential property index, and attached dwelling index to distinguish 2 cities. After that, I filter the data and take the rows from 2016-2021 because there are only 2 rows in 2022.

```
[ ] sydney_data.duplicated().sum()
```

```
0
```

```
[ ] sydney_data = sydney_data.drop(columns=['suburb_population', 'suburb_median_income', 'suburb_sqkm','suburb_elevation', 'property_inflation_index'])
    sydney_data.head()
```

	price	date_sold	suburb	num_bath	num_bed	num_parking	property_size	type	suburb_lat	suburb_lng	cash_rate	km_from_cbd
0	530000	13/1/16	Kincumber	4	4	2	1351	House	-33.47252	151.40208	2.0	47.05
1	525000	13/1/16	Halekulani	2	4	2	594	House	-33.21772	151.55237	2.0	78.54
2	480000	13/1/16	Chittaway Bay	2	4	2	468	House	-33.32678	151.44557	2.0	63.59
3	452000	13/1/16	Leumeah	1	3	1	344	House	-34.05375	150.83957	2.0	40.12
4	365500	13/1/16	North Avoca	0	0	0	1850	Vacant land	-33.45608	151.43598	2.0	49.98

```
sydney_data['date_sold'] = pd.to_datetime(sydney_data['date_sold'], dayfirst=True, errors='coerce')

sydney_data.head()
```

```
Hiện kết quả đã ẩn
```

```
[ ] sydney_data = sydney_data[sydney_data['date_sold'].dt.year <= 2021]

    sydney_data.head()
```

	price	date_sold	suburb	num_bath	num_bed	num_parking	property_size	type	suburb_lat	suburb_lng	cash_rate	km_from_cbd
0	530000	2016-01-13	Kincumber	4	4	2	1351	House	-33.47252	151.40208	2.0	47.05
1	525000	2016-01-13	Halekulani	2	4	2	594	House	-33.21772	151.55237	2.0	78.54
2	480000	2016-01-13	Chittaway Bay	2	4	2	468	House	-33.32678	151.44557	2.0	63.59
3	452000	2016-01-13	Leumeah	1	3	1	344	House	-34.05375	150.83957	2.0	40.12
4	365500	2016-01-13	North Avoca	0	0	0	1850	Vacant land	-33.45608	151.43598	2.0	49.98

Figure 40: Drop columns and filter data by date

After that, I use geopandas to perform a spatial join between two Sydney data and postcode data, in order to associate Sydney suburbs with their corresponding postcodes and local government areas (LGA). This helps match suburb data with postcode and LGA data based on geographical proximity (latitude and longitude).

```
[ ] import geopandas as gpd
    from shapely.geometry import Point


    sydney_data['Coordinates'] = sydney_data.apply(lambda row: Point(row['suburb_lng'], row['suburb_lat']), axis=1)
    sydney_gdf = gpd.GeoDataFrame(sydney_data, geometry='Coordinates')
    sydney_gdf = sydney_gdf.set_crs("EPSG:4326")

    postcode_data['Coordinates'] = postcode_data.apply(lambda row: Point(row['long'], row['lat']), axis=1)
    postcode_gdf = gpd.GeoDataFrame(postcode_data, geometry='Coordinates')
    postcode_gdf = postcode_gdf.set_crs("EPSG:4326")

    sydney_gdf = sydney_gdf.to_crs(epsg=32756)
    postcode_gdf = postcode_gdf.to_crs(epsg=32756)

    sydney_postcode_joined = gpd.sjoin_nearest(sydney_gdf, postcode_gdf, how='left', distance_col='distance')

    if 'index_right' in sydney_postcode_joined.columns:
        sydney_postcode_joined = sydney_postcode_joined.drop(columns=['index_right'])

    sydney_postcode_joined = sydney_postcode_joined.reset_index(drop=True)

    sydney_data['Postcode'] = sydney_postcode_joined['postcode']
    sydney_data['Local Government Area (LGA)'] = sydney_postcode_joined['lgaregion']

    sydney_data.drop(columns=['Coordinates'], inplace=True)

    sydney_data.head()
```

	price	date_sold	suburb	num_bath	num_bed	num_parking	property_size	type	suburb_lat	suburb_lng	cash_rate	km_from_cbd	Postcode	Local Government Area (LGA)
0	530000	2016-01-13	Kincumber	4	4	2	1351	House	-33.47252	151.40208	2.0	47.05	2251	Central Coast (NSW)
1	525000	2016-01-13	Halekulani	2	4	2	594	House	-33.21772	151.55237	2.0	78.54	2251	Central Coast (NSW)
2	480000	2016-01-13	Chittaway Bay	2	4	2	468	House	-33.32678	151.44557	2.0	63.59	2251	Central Coast (NSW)
3	452000	2016-01-13	Leumeah	1	3	1	344	House	-34.05375	150.83957	2.0	40.12	2251	Central Coast (NSW)
4	365500	2016-01-13	North Avoca	0	0	0	1850	Vacant land	-33.45608	151.43598	2.0	49.98	2251	Central Coast (NSW)

Figure 41: Get the postcode and Local Government Area based on latitude and longitude

After that, I use the same method above to merge Sydney data and Sydney residential property index data to get the column as Melbourne data.

```
sydney_data_cop = sydney_data.copy()
residental_property_index_sydney_cop = residental_property_index_sydney.copy()

new_row = pd.DataFrame({
    'Date': [pd.to_datetime('2021-12-31')],
    'Residential Property Price Index': [218.7],
    'Established House Price Index': [242.0],
    'Attached Dwellings Price Index': [179.4]
})

first_row = pd.DataFrame({
    'Date': [pd.to_datetime('2016-01-01')],
    'Residential Property Price Index': [150.9],
    'Established House Price Index': [154.8],
    'Attached Dwellings Price Index': [143.2]
})

residental_property_index_sydney_cop = pd.concat([first_row, residental_property_index_sydney_cop]).reset_index(drop=True)
residental_property_index_sydney_cop = pd.concat([residental_property_index_sydney_cop, new_row]).reset_index(drop=True)


residental_property_index_sydney_cop
```

	Date	Residential Property Price Index	Established House Price Index	Attached Dwellings Price Index
0	2016-01-01	150.9	154.8	143.2
1	2016-03-01	150.9	154.8	143.2
2	2016-06-01	155.2	159.7	146.1
3	2016-09-01	159.3	164.4	149.2
4	2016-12-01	167.6	174.4	153.8
5	2017-03-01	172.7	179.5	159.1
6	2017-06-01	176.6	183.3	163.2
7	2017-09-01	174.2	180.9	160.9
8	2017-12-01	174.0	180.5	160.9
9	2018-03-01	171.9	178.0	159.7
10	2018-06-01	169.8	175.8	157.7
11	2018-09-01	166.6	172.1	155.6

Figure 42: Sydney residential property index data

```
for i in range(residental_property_index_sydney_cop.shape[0] - 1):
    mask = (sydney_data_cop['date_sold'] >= residental_property_index_sydney_cop['Date'][i]) & \
           (sydney_data_cop['date_sold'] <= residental_property_index_sydney_cop['Date'][i + 1]) & \
           (sydney_data_cop['Residential Property Price Index'].isna()) & \
           (sydney_data_cop['Attached Dwellings Price Index'].isna())

    sydney_data_cop.loc[mask, 'Residential Property Price Index'] = float(residental_property_index_sydney_cop['Residential Property Price Index'].iloc[i+1])
    sydney_data_cop.loc[mask, 'Attached Dwellings Price Index'] = float(residental_property_index_sydney_cop['Attached Dwellings Price Index'].iloc[i+1])

print(sydney_data_cop['Residential Property Price Index'].isnull().sum())
print(sydney_data_cop['Attached Dwellings Price Index'].isnull().sum())
```

```
0
0
```

```
sydney_data_cop.head()
```

	price	date_sold	suburb	num_bath	num_bed	num_parking	property_size	type	suburb_lat	suburb_lng	cash_rate	km_from_cbd	Postcode	Local Government Area (LGA)	Residential Property Price Index	Attached Dwellings Price Index
0	530000	2016-01-13	Kincumber	4	4	2	1351	House	-33.47252	151.40208	2.0	47.05	2251	Central Coast (NSW)	150.9	143.2
1	525000	2016-01-13	Halekulani	2	4	2	594	House	-33.21772	151.55237	2.0	78.54	2251	Central Coast (NSW)	150.9	143.2
2	480000	2016-01-13	Chittaway Bay	2	4	2	468	House	-33.32678	151.44557	2.0	63.59	2251	Central Coast (NSW)	150.9	143.2
3	452000	2016-01-13	Leumeah	1	3	1	344	House	-34.05375	150.83957	2.0	40.12	2251	Central Coast (NSW)	150.9	143.2
4	365500	2016-01-13	North Avoca	0	0	0	1850	Vacant land	-33.45608	151.43598	2.0	49.98	2251	Central Coast (NSW)	150.9	143.2

Figure 43: Merging Sydney data and Sydney residential property index data

Next, I handle the house type in Sydney because the data records many different kinds of houses. For the property such as land, I will delete those samples because my target client is one who wants to buy a house or an apartment. Then, I modify all these kinds of houses into 3 main kinds of houses which are house, townhouse, and unit (or apartment)

```
[ ] sydney_data_cop['type'].unique()

array(['House', 'Vacant land', 'Townhouse', 'Apartment / Unit / Flat',
       'Semi-Detached', 'New House & Land', 'Duplex', 'Villa', 'New land',
       'Terrace', 'Studio', 'Block of Units', 'Development Site',
       'Acreage / Semi-Rural', 'New Apartments / Off the Plan', 'Rural'],
      dtype=object)

[ ] sydney_data_cop2 = sydney_data_cop.copy()

    land_related_types = [
        'Vacant land', 'New land', 'Development Site', 'Acreage / Semi-Rural', 'Rural'
    ]

    sydney_data_cop2 = sydney_data_cop2[~sydney_data_cop2['type'].isin(land_related_types)]

    sydney_data_cop2.shape

(10960, 16)

[ ] property_type_mapping = {
        'House': 'h',
        'Semi-Detached': 'h',
        'Duplex': 'h',
        'Villa': 'h',
        'Terrace': 'h',
        'New House & Land': 'h',
        'Townhouse': 't',
        'Apartment / Unit / Flat': 'u',
        'Studio': 'u',
        'Block of Units': 'u',
        'New Apartments / Off the Plan': 'u'
    }

    sydney_data_cop2['type'] = sydney_data_cop2['type'].replace(property_type_mapping)

    sydney_data_cop2['type'].unique()

array(['h', 't', 'u'], dtype=object)
```

Figure 44: Drop rows and change the house types

Finally, I handle the text columns and split the date column as well by using the same method as the Melbourne data

c. Preprocessing Data

Melbourne Data Preprocess

Handle Nan Values

First, I have to check the Nan value in each column to decide whether to remove or impute them (Figure 45).

```
[ ]  melbourne_data_cop2.isnull().sum() / melbourne_data_cop2.shape[0] * 100
```

	0
Suburb	0.000000
Type	0.000000
Price	21.829814
Distance	0.002869
Postcode	0.002869
Bedroom	23.574134
Bathroom	23.599954
Car	25.040165
Landsize	33.882258
BuildingArea	60.577806
CouncilArea	0.008607
Lattitude	22.882717
Longtitude	22.882717
City	0.000000
CashRate	0.000000
Residential Property Price Index	0.000000
Attached Dwellings Price Index	0.000000
DaySold	0.000000
MonthSold	0.000000
YearSold	0.000000

Figure 45: Percentage of missing value in each column

I fill those features by using random numbers in the fill list. The fill list contains the number of values in one column. From that, these features are not biased and don’t concentrate on 1 point on the chart.

```
[ ] print(melbourne_data_cop2['Bedroom'].unique())
    print(melbourne_data_cop2['Bathroom'].unique())
    print(melbourne_data_cop2['Car'].unique())

    [ 2.  3.  4.  6. nan  1.  5.  7.  0.  8.  9. 30. 20. 10. 16. 12.]
    [ 1.  2. nan  3.  7.  4.  0.  5.  6. 12.  8.  9.]
    [ 1.  0.  2. nan  6.  5.  4.  3.  8.  7. 11. 26.  9. 10. 18. 12.]

[ ] fill_list = melbourne_data_cop2['Bathroom'].dropna().unique().tolist()
    melbourne_data_cop2['Bathroom'] = melbourne_data_cop2['Bathroom'].fillna(pd.Series(np.random.choice(fill_list, len(melbourne_data_cop2.index))))

    fill_list = melbourne_data_cop2['Bedroom'].dropna().unique().tolist()
    melbourne_data_cop2['Bedroom'] = melbourne_data_cop2['Bedroom'].fillna(pd.Series(np.random.choice(fill_list , len(melbourne_data_cop2.index))))

    fill_list = melbourne_data_cop2['Landsize'].dropna().unique().tolist()
    melbourne_data_cop2['Landsize'] = melbourne_data_cop2['Landsize'].fillna(pd.Series(np.random.choice(fill_list , len(melbourne_data_cop2.index))))

    fill_list = melbourne_data_cop2['Car'].dropna().unique().tolist()
    melbourne_data_cop2['Car'] = melbourne_data_cop2['Car'].fillna(pd.Series(np.random.choice(fill_list , len(melbourne_data_cop2.index))))
```

Figure 46: Fill missing values by random numbers of values in the column

The building area in the real-world is a very important feature. Although this feature has many missing values (60.57%), I still try to impute with random forest. Random Forest can handle outliers, so I decide to use features like bedroom, bathroom, and land size to predict the building area.

```
    from sklearn.model_selection import train_test_split

    melbourne_data_cop2_temp = melbourne_data_cop2.copy()

    melbourne_data_cop2_temp.dropna(subset=['BuildingArea'], inplace=True)
    X = melbourne_data_cop2_temp[['Bedroom', 'Bathroom', 'Landsize']]
    y = melbourne_data_cop2_temp['BuildingArea']

    X_train_temp, X_test_temp, y_train_temp, y_test_temp = train_test_split(X, y, test_size=0.2, random_state=42)

    model = RandomForestRegressor()
    model.fit(X_train_temp, y_train_temp)


    missing_value_BuildingArea = melbourne_data_cop2[melbourne_data_cop2['BuildingArea'].isnull()].index
    melbourne_data_cop2.loc[missing_value_BuildingArea, 'BuildingArea'] = model.predict(melbourne_data_cop2.loc[missing_value_BuildingArea, ['Bedroom', 'Bathroom', 'Landsize']])

    melbourne_data_cop2['BuildingArea'] = np.clip(melbourne_data_cop2['BuildingArea'], a_min=0, a_max=None)

[ ] sns.boxplot(data=melbourne_data_cop2, x=melbourne_data_cop2['BuildingArea'])

    <Axes: xlabel='BuildingArea'>
```

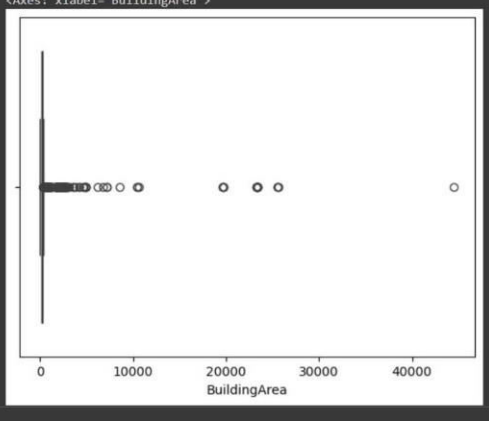


Figure 47: Impute missing values using random forest

For the other missing value columns, I decide to drop them because filling them

might get noise and I want to decrease the sample of Melbourne data (34800 samples) to near with Sydney data (11000 samples). This makes our model work well with both data and not biased to 1 city.

```
[ ]  melbourne_data_cop2.dropna(subset=['CouncilArea'], inplace=True)
     melbourne_data_cop2.dropna(subset=['Price'], inplace=True)
     melbourne_data_cop2.dropna(subset=['Longtitude'], inplace=True)
     melbourne_data_cop2.dropna(subset=['Lattitude'], inplace=True)
     melbourne_data_cop2.isnull().sum()
```

	0
Suburb	0
Type	0
Price	0
Distance	0
Postcode	0
Bedroom	0
Bathroom	0
Car	0
Landsize	0
BuildingArea	0
CouncilArea	0
Lattitude	0
Longtitude	0
City	0
CashRate	0
Residential Property Price Index	0
Attached Dwellings Price Index	0
DaySold	0
MonthSold	0
YearSold	0

dtype: int64

Figure 48: Drop rows have missing values

Handle Outliers

At first, I visualize the data with a boxplot using a for loop to determine which features should be handled with the specific technique. Based on the Australian postcode, I filter the data with the postcode over 3800 is too far for Melbourne, so I remove that data.

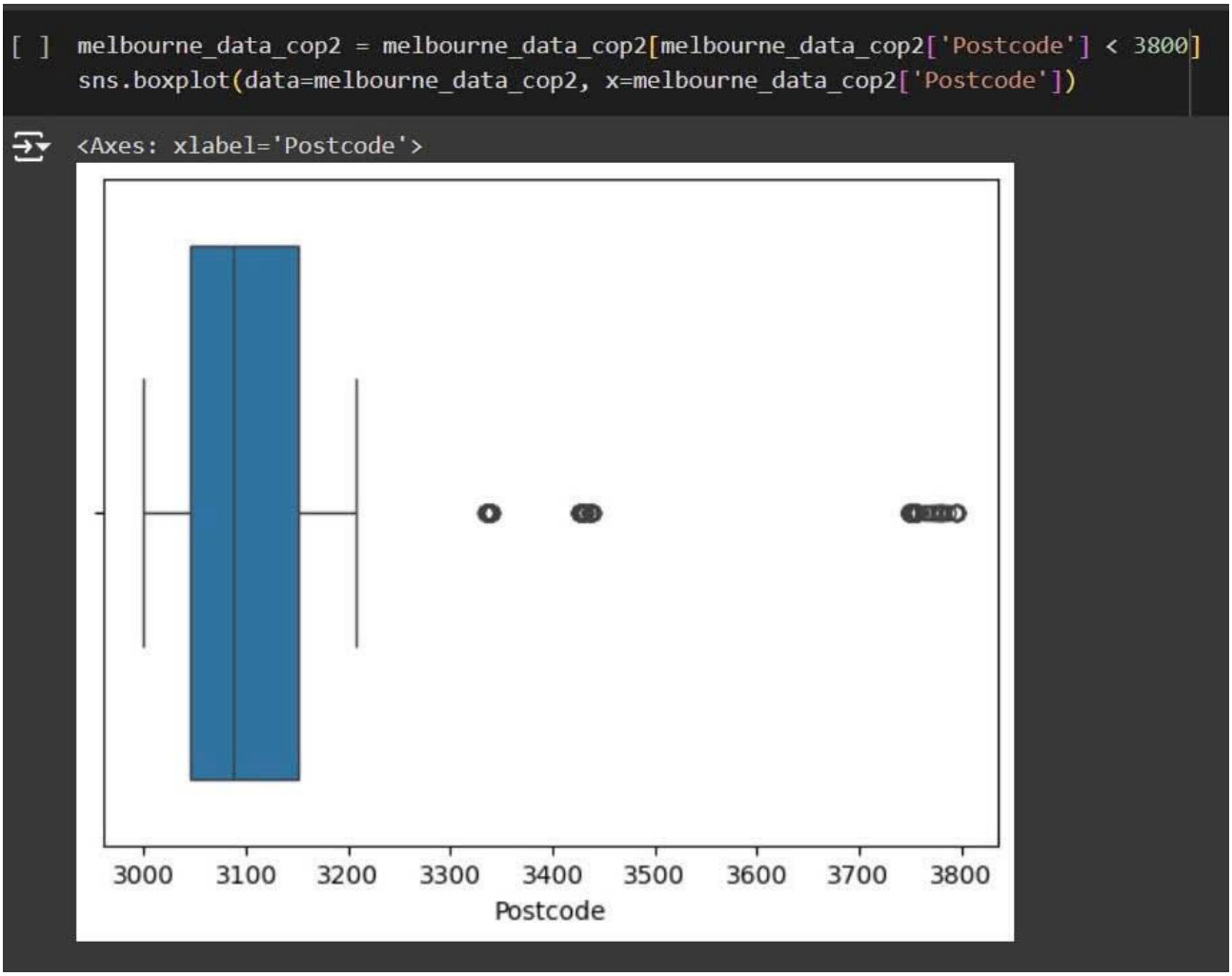


Figure 49: Handle outliers by removing

After that, other features are handled by the Interquartile Range (IQR) method. The code aims to pinpoint and substitute outliers in designated numerical columns by employing the IQR method, thereby ensuring that extreme values are limited to the determined lower and upper thresholds. This safeguards against outliers distorting the analysis.

```
def boundaries(col):
  q1, q3 = np.quantile(col, [0.25, 0.75])
  iqr = q3 - q1
  lb = q1 - (1.5 * iqr)
  ub = q3 + (1.5 * iqr)
  return lb, ub

for col in ['Price', 'Distance', 'Bedroom', 'Bathroom', 'Car', 'Landsize', 'BuildingArea', 'Longtitude', 'Lattitude']:
  lb, ub = boundaries(melbourne_data_cop2[col])
  for index, value in melbourne_data_cop2[col].items():
    if melbourne_data_cop2[col].dtype != 'object':
      if value < lb:
        melbourne_data_cop2.at[index, col] = lb
      elif value > ub:
        melbourne_data_cop2.at[index, col] = ub
```

Figure 50: Handle outliers by using the IQR method

The features added to show that they have a low correlation to price (cash rate: -0.042, residential property index: 0.039, attached dwellings price index: 0.047). They have low values due to the diversity of data. This data contains only house prices in Melbourne and this is the factor that makes the cash rate, residential property index, and attached dwelling index can't significantly influence the price. Also, these features show the change in real estate and price over time, thus these features may have more impact on the price if the data has a longer period. Furthermore, this can be demonstrated in the Merge Data part, where the data is merged with increasing diversity and different periods.

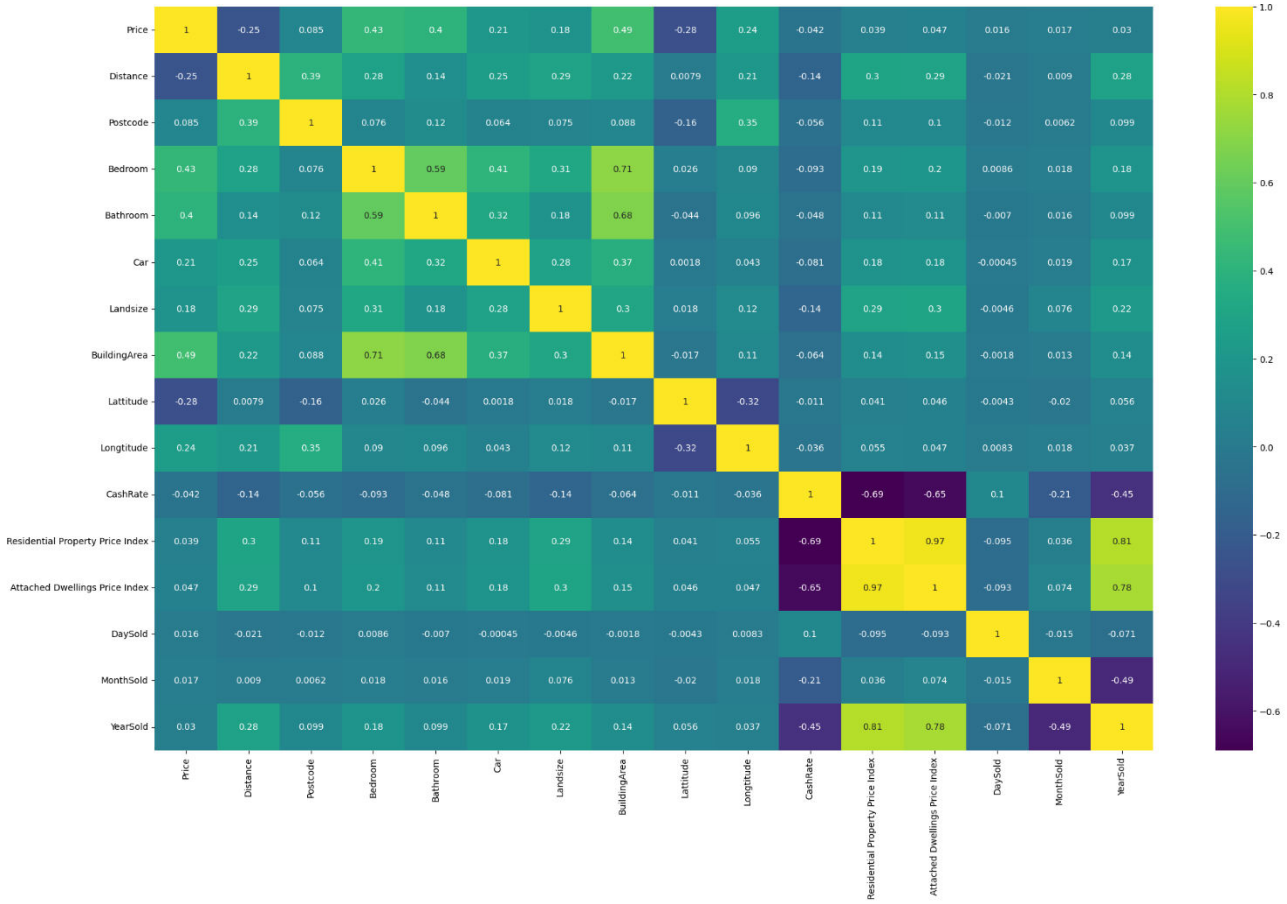


Figure 51: Heatmap of Melbourne after preprocessing data

Sydney Data

Handle Missing Value

Since this data doesn't have missing values, I will skip this part.

```
[ ]  sydney_data_cop2.isnull().sum()
```

	0
price	0
suburb	0
num_bath	0
num_bed	0
num_parking	0
property_size	0
type	0
suburb_lat	0
suburb_lng	0
cash_rate	0
km_from_cbd	0
Postcode	0
Local Government Area (LGA)	0
Residential Property Price Index	0
Attached Dwellings Price Index	0
City	0
DaySold	0
MonthSold	0
YearSold	0

dtype: int64

Figure 52: Numbers of missing values in Sydney data

Handle Outliers

At first, I still visualize data in each column with a boxplot to view the outliers and most of the data I don't need to handle specifically. Therefore, I handle the outliers with the same method in Melbourne data.

```
[ ]  def boundaries(col):
       q1, q3 = np.quantile(col, [0.25, 0.75])
       iqr = q3 - q1
       lb = q1 - (1.5 * iqr)
       ub = q3 + (1.5 * iqr)
       return lb, ub

     for col in ['Price', 'Distance', 'Bedroom', 'Bathroom', 'Car', 'BuildingArea', 'Longtitude', 'Lattitude']:
       lb, ub = boundaries(sydney_data_cop2[col])
       for index, value in sydney_data_cop2[col].items():
         if sydney_data_cop2[col].dtype != 'object':
           if value < lb:
             sydney_data_cop2.at[index, col] = lb if sydney_data_cop2[col].dtype != 'int64' else int(lb)
           elif value > ub:
             sydney_data_cop2.at[index, col] = ub if sydney_data_cop2[col].dtype != 'int64' else int(ub)
```

```
[ ]  for i in sydney_data_cop2.select_dtypes(include='number').columns:
         sns.boxplot(data=sydney_data_cop2, x=i)
         plt.show()
```

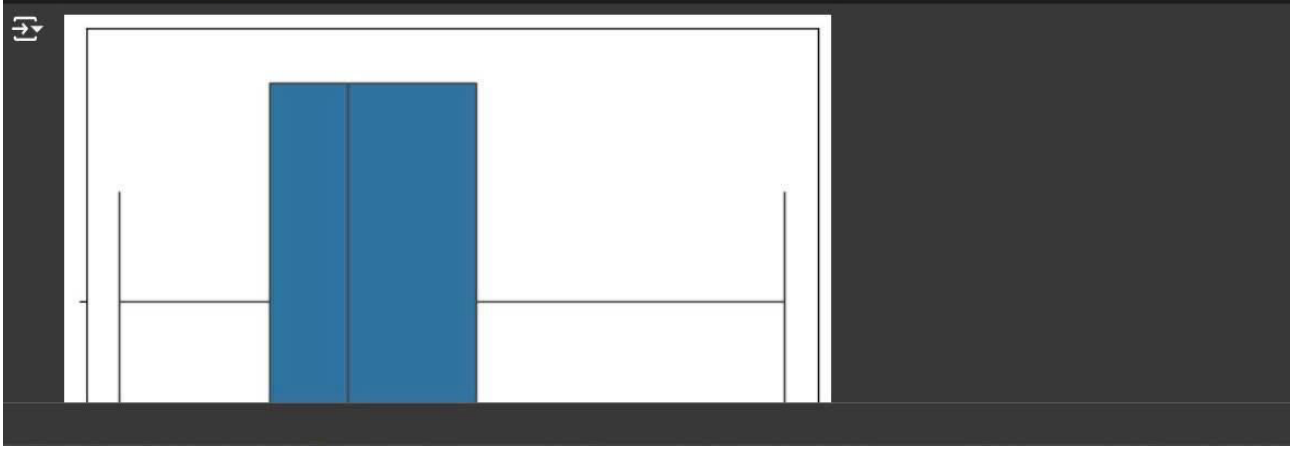


Figure 53: Handle outliers with IQR

In contrast with Melbourne data, the correlation between price and the 3 financial features added is moderately high (cash rate: -0.32, residential property index: 0.32, attached dwelling index: 0.3). The data has a range of period longer than the Melbourne data (2016-2021) and this is a factor may affecting the correlation between price with 3 financial data.

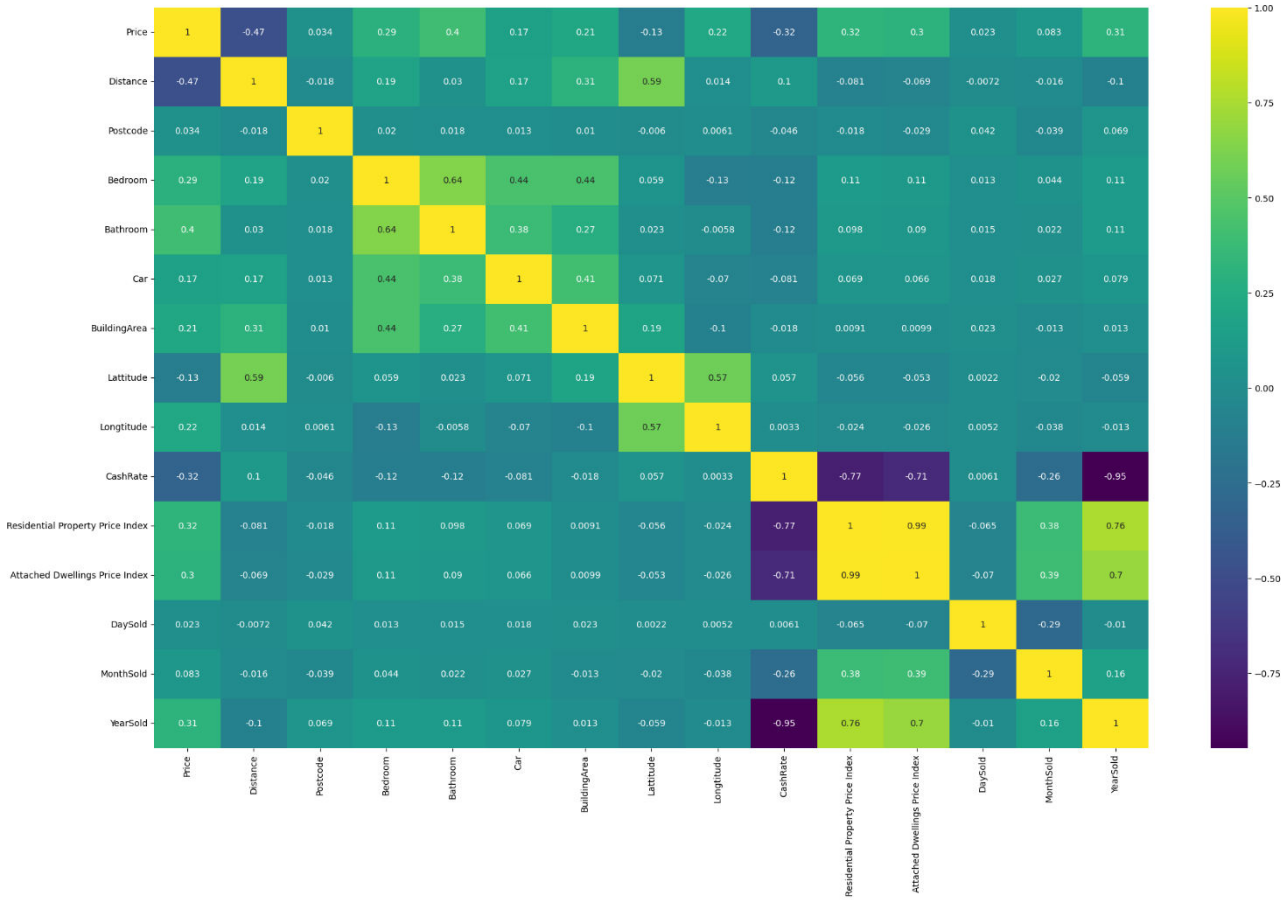


Figure 54: Heatmap of Sydney data

d. Merge Data

Merge the Melbourne and Sydney data vertically because most of the columns I have filled with values and handled properly except the land size column in Sydney and this will be addressed below.

Part III: Merge Dataset

```
[ ] combined_data = pd.concat([melbourne_data_cop2, sydney_data_cop2], axis=0, ignore_index=True)
    combined_data
```

	Suburb	Type	Price	Distance	Postcode	Bedroom	Bathroom	Car	Landsize	BuildingArea	CouncilArea	Lattitude	Longtitude	City	CashRate	Residential Property Price Index	Attached Dwellings Price Index	DaySold	MonthSold	YearSold
0	abbotsford	h	1480000.0	2.50	8.028455	2.0	1.0	1.0	202.0	55.154619	yarracity	-37.79960	144.99840	melbourne	1.50	144.4	118.4	3	12	2016
1	abbotsford	h	1035000.0	2.50	8.028455	2.0	1.0	0.0	156.0	79.000000	yarracity	-37.80790	144.99340	melbourne	2.00	127.3	114.0	4	2	2016
2	abbotsford	h	1465000.0	2.50	8.028455	3.0	2.0	0.0	134.0	150.000000	yarracity	-37.80930	144.99440	melbourne	1.50	148.7	121.6	4	3	2017
3	abbotsford	h	850000.0	2.50	8.028455	3.0	2.0	1.0	94.0	144.539405	yarracity	-37.79690	144.99890	melbourne	1.50	148.7	121.6	4	3	2017
4	abbotsford	h	1600000.0	2.50	8.028455	3.0	1.0	2.0	120.0	142.000000	yarracity	-37.80720	144.99410	melbourne	1.75	132.9	115.9	4	6	2016
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
31807	carlingford	h	2055000.0	18.20	7.618742	5.0	2.0	2.0	NaN	689.000000	bayside(nsw)	-33.77523	151.04540	sydney	0.10	218.7	179.4	30	12	2021
31808	mascot	h	1900000.0	9.35	7.618742	4.0	3.0	2.0	NaN	285.000000	bayside(nsw)	-33.94660	151.18371	sydney	0.10	218.7	179.4	30	12	2021
31809	kellyville	h	1900000.0	30.08	7.618742	4.0	3.0	2.0	NaN	540.000000	bayside(nsw)	-33.69583	150.95622	sydney	0.10	218.7	179.4	31	12	2021
31810	sevenhills	h	1300000.0	26.58	7.618742	5.0	3.0	2.0	NaN	1208.000000	bayside(nsw)	-33.77743	150.94272	sydney	0.10	218.7	179.4	31	12	2021
31811	sydney	u	1025000.0	0.31	7.602900	2.0	2.0	1.0	NaN	129.000000	sydney	-33.86794	151.20998	sydney	0.10	218.7	179.4	31	12	2021

31812 rows × 20 columns

Figure 55: Merge 2 main datasets vertically

After that, I use a label encoder to encode all the values in the text column into numerical values for machine learning.

```
[ ] from sklearn.preprocessing import LabelEncoder
    encoder = LabelEncoder()

    for i in combined_data.select_dtypes(include='object').columns:
        combined_data[i] = encoder.fit_transform(combined_data[i])
```

```
[ ] combined_data
```

	Suburb	Type	Price	Distance	Postcode	Bedroom	Bathroom	Car	Landsize	BuildingArea	CouncilArea	Lattitude	Longtitude	City	CashRate	Residential Property Price Index	Attached Dwellings Price Index	DaySold	MonthSold	YearSold
0	1	0	1480000.0	2.50	8.028455	2.0	1.0	1.0	202.0	55.154619	68	-37.79960	144.99840	0	1.50	144.4	118.4	3	12	2016
1	1	0	1035000.0	2.50	8.028455	2.0	1.0	0.0	156.0	79.000000	68	-37.80790	144.99340	0	2.00	127.3	114.0	4	2	2016
2	1	0	1465000.0	2.50	8.028455	3.0	2.0	0.0	134.0	150.000000	68	-37.80930	144.99440	0	1.50	148.7	121.6	4	3	2017
3	1	0	850000.0	2.50	8.028455	3.0	2.0	1.0	94.0	144.539405	68	-37.79690	144.99690	0	1.50	148.7	121.6	4	3	2017
4	1	0	1600000.0	2.50	8.028455	3.0	1.0	2.0	120.0	142.000000	68	-37.80720	144.99410	0	1.75	132.9	115.9	4	6	2016
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
31807	159	0	2055000.0	18.20	7.618742	5.0	2.0	2.0	NaN	689.000000	1	-33.77523	151.04540	1	0.10	218.7	179.4	30	12	2021
31808	537	0	1900000.0	9.35	7.618742	4.0	3.0	2.0	NaN	285.000000	1	-33.94660	151.18371	1	0.10	218.7	179.4	30	12	2021
31809	450	0	1900000.0	30.08	7.618742	4.0	3.0	2.0	NaN	540.000000	1	-33.69583	150.95622	1	0.10	218.7	179.4	31	12	2021
31810	754	0	1300000.0	26.58	7.618742	5.0	3.0	2.0	NaN	1208.000000	1	-33.77743	150.94272	1	0.10	218.7	179.4	31	12	2021
31811	804	2	1025000.0	0.31	7.602900	2.0	2.0	1.0	NaN	129.000000	58	-33.86794	151.20998	1	0.10	218.7	179.4	31	12	2021

31812 rows × 20 columns

Figure 56: Encode text data into numerical values

The Nan values occur in the land size column because the Sydney data doesn't have this column. Therefore, when concatenating 2 columns, the code automatically fills the column in Sydney data with Nan value.

```
[ ]  combined_data.isnull().sum()
```

	0
Suburb	0
Type	0
Price	0
Distance	0
Postcode	0
Bedroom	0
Bathroom	0
Car	0
Landsize	10960
BuildingArea	0
CouncilArea	0
Lattitude	0
Longtitude	0
City	0
CashRate	0
Residential Property Price Index	0
Attached Dwellings Price Index	0
DaySold	0
MonthSold	0
YearSold	0

dtype: int64

Figure 57: Missing values in the land size of combined data

To address this issue, I decide to use the KNN Imputer as an imputer to predict based on car, bedroom, bathrooms, and building area to fill the missing value in the land size column. After merging 2 data, the dataset has the diversity, outliers, and missing values are handled properly. Therefore, using KNN imputer is reasonable in this case.

```
from sklearn.preprocessing import MinMaxScaler
import pandas as pd

# Initialize the imputer and scaler
imputer = KNNImputer()
scalerTemp = MinMaxScaler()

combined_data_temp = combined_data.copy()

# Apply scaling
combined_data_temp[['Landsize']] = scalerTemp.fit_transform(combined_data_temp[['Landsize']])

# Apply KNN Imputation to fill missing values
combined_data_temp[['Landsize']] = imputer.fit_transform(combined_data_temp[['Landsize']])

# Reverse the scaling after imputation
combined_data_temp[['Landsize']] = scalerTemp.inverse_transform(combined_data_temp[['Landsize']])

# Update the original DataFrame
combined_data = combined_data_temp.copy()
# Display the number of rows to check if the count is correct
combined_data.shape
```

```
(31812, 20)
```

```
[120] sns.boxplot(data=combined_data, x=combined_data['Landsize'])
```

```
<Axes: xlabel='Landsize'>
```

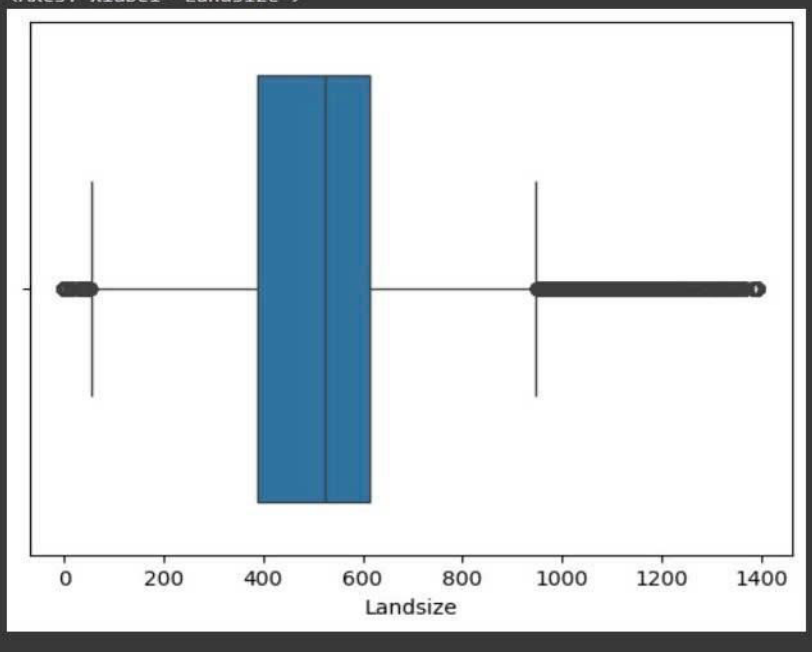


Figure 58: Impute missing values in the land size column with random forest

e. Data Analysis (After Preprocessing)

In Figure 59, the heatmap below shows the correlation between all features, but we will view the correlation of price to other features. First, the feature that has the lowest correlation is suburb. The suburb has a low value like that because it is affected by latitude and longitude which list the position of a house or building. Thus, the suburb has a very low contribution in influencing the price. Second, 3 features (cash rate, residential property index, and attached dwelling price index) have a pretty high correlation (-0.42, 0.42, 0.42). The reason why these features have high correlation comes from 2 main factors: diversity and period. After merging 2 housing market datasets in 2 different cities, the diversity in datasets is increased. The diversity here not only indicates having many different house features (suburb, city, postcode, council area, latitude, longitude) but also the residential property index and attached dwelling price index have different metrics for each city. This makes the difference in the price of each city, thus the correlation between price and these 2 features is high. Moreover, the cash rate is the same in 2 cities because the cash rate has to apply all over Australia. Nevertheless, the Melbourne data (2016-2018) and Sydney data (2016-2021) have 2 different periods of the housing market. In addition, the cash rate has significantly changed since 2018 (Figure 31). Thus, these changes impact the value of money and become a reason for changing the price of the house. In conclusion, adding 3 new features has a positive impact on distinguishing 2 different housing markets and this is good for the machine learning model to identify 2 markets and give a better prediction.

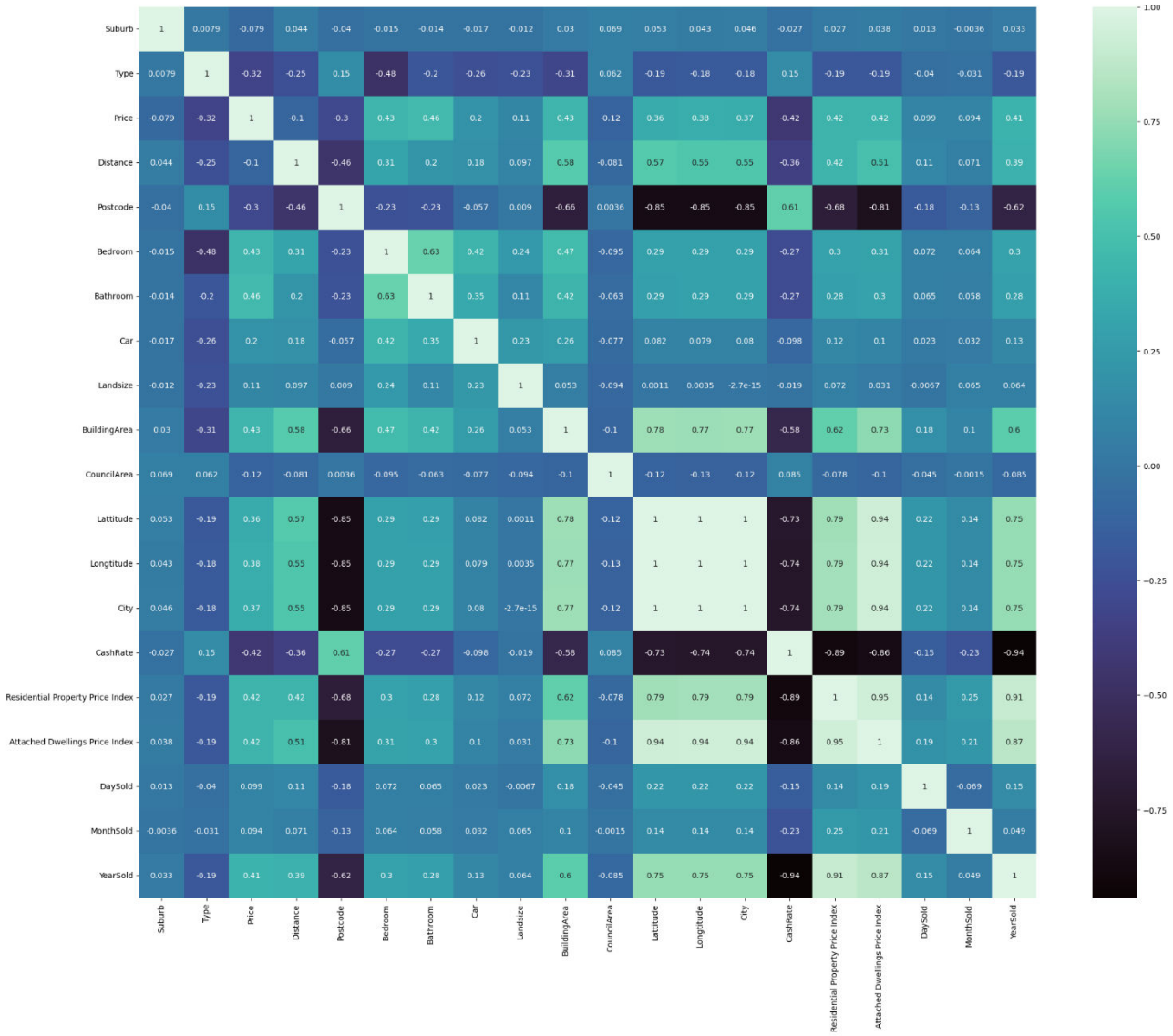


Figure 59: The heatmap after merging 2 data

In Figure 60, use the Plotly library to visualize the scatter map box and this can be interacted with using Google Colab. The map expresses the correlation between latitude and longitude to price.

Figure 60: Interacted scatter box map

5. Machine Learning Model Selection

The problem I set initially (Problem Framing) is predicting price, therefore the nature of this task is regression. Thus, most models used in this task are regression models. But, I also use another algorithm (clustering) but it serves as a contributor to support for the regression models. For instance, the KNN imputer can impute missing values based on the features in the data. Furthermore, Eichholtz, Veld, & Schweitzer state that the location is a crucial element in the purchasing and selling of real estate due to the significant regional variations present in real estate markets (2000). Therefore, I use DBScan to check the outliers in the dataset and explore the price house based on longitude and latitude

a. Criteria

The criteria are designed based on the objective of the problem and the dataset to fit with the model.

- The model has to get high accuracy in evaluation metrics (R^2 score and MSE score). The model must meet the threshold (Over 80% in R^2 score

and Under 0.01 score in MSE)

- The model doesn't require training fast and gives fast prediction because this task aims for the highest accuracy possible
- The model can handle outliers, different distributions, and complexity because combining 2 main datasets generates many problems in data.
- The computational cost is not considered in this task

b. Model Selection Discuss

This is the comparison between the 3 models based on their advantages, disadvantages, and the model's accuracy. Although the final conclusion is shown in this section, the detailed demonstrates are in the **Accuracy Table** part. The demonstration includes comparing when train, test on combined data, and train, and test on single dataset (Melbourne data).

Based on the criteria above, the models considered are Random Forest, Multilayer Perceptron (MLP), and XGBoost.

Random Forest Model

- **Advantage:** The model can handle outliers, low chance of overfitting, and works well with high dimension data, especially in combined data like this (20 features).
- **Disadvantage:** The computational and resource costs are high when dealing the data with too many samples. Also, the time of prediction is slow. However, these disadvantages are not big problems in this task which focuses on the performance and accuracy of prediction
- **Conclude:** This model can be a good choice to alternate the XGBoost model because the score of evaluation metrics is slightly weaker than the XGBoost model

Multilayer Perceptron Model (MLP)

- **Advantage:** The model can capture complex patterns and non-linear relationships between features very well. This is suited to the data that has high dimensions like housing market price which is impacted by many features
- **Disadvantage:** The model is prone to overfitting. It needs more samples to get better accuracy which is not suit for data conditions (only over 30,000 samples).
- **Conclude**: This model is not a good choice for this task. It not only satisfies the condition but also the result of this model doesn't meet the requirement minimum.

XGBoost Model

- **Advantage:** The model has the highest accuracy among the 3 models and this model can work well with data that has different distributions. Also, it can adapt to high-dimension data and learn non-linear relationships between features. Moreover, the model can handle differences in the distributions of combined data. In addition, the training and prediction time of the model are very impressive and this can be a bonus point
- **Disadvantage:** This model is sensitive to noise, so the cleaning data part must be done properly.
- **Conclude**: This model is the best choice among the 3 models because of the highest accuracy in both metrics on the test set and fast training and prediction time.

6. Implementation

a. Technical Implementation

Split Data To Train and Test set

Using the train_test_split() function in sklearn to shuffle and split the data into train set (80%) and test set (20%)

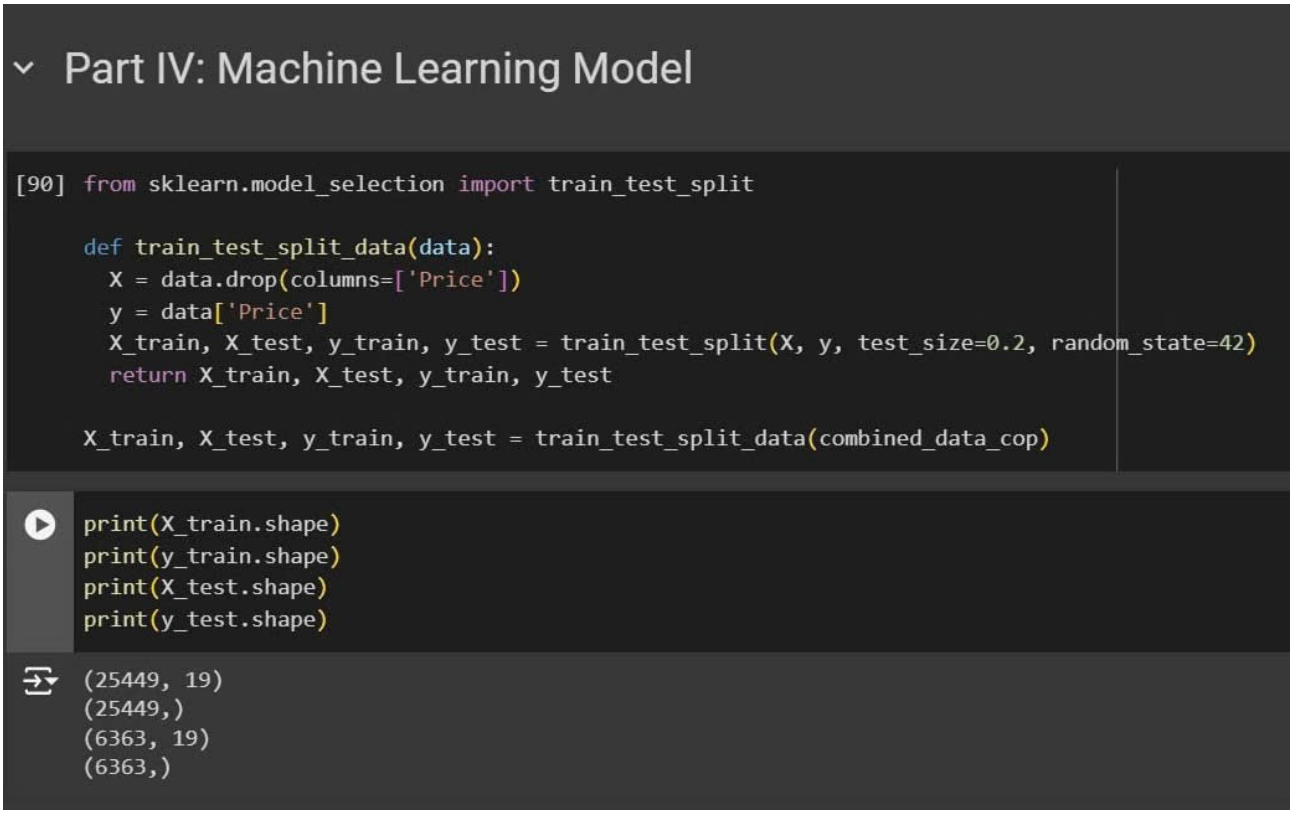


Figure 61: Split Data into train and test set

Cross Validation (5 folds)

Cross-validation is a technique used to assess how well the model generalizes to unseen data. It splits the train data set into k parts (5 parts in this case) and the model trains on k-1 part (4 parts) and the remaining (1 part) is for validating.

In this part, I use cross-validation for MLP, random forest, and xgboost to evaluate the metrics. From that, I can fine-tune the parameters to get the best accuracy possible, and this is vital in modifying the parameters of the MLP model.

```
[74] from sklearn.ensemble import RandomForestRegressor
     from sklearn.metrics import mean_squared_error, r2_score
     from sklearn.model_selection import KFold
     import numpy as np

     mse_scores = []
     r2_scores = []
     i = 0

     for train_index, val_index in KFold(n_splits=5, shuffle=True, random_state=42).split(X_train):
         i += 1

         x_train_fold, x_val_fold = X_train.iloc[train_index], X_train.iloc[val_index]
         y_train_fold, y_val_fold = y_train.iloc[train_index], y_train.iloc[val_index]

         model_temp = RandomForestRegressor()
         model_temp.fit(x_train_fold, y_train_fold)

         y_pred_val = model_temp.predict(x_val_fold)

         mse = mean_squared_error(y_val_fold, y_pred_val)
         r2 = r2_score(y_val_fold, y_pred_val)

         print("-----------------------------------------")
         print("Phase: ", i)
         print("Mean Squared Error:", mse)
         print("R-squared:", r2)

         mse_scores.append(mse)
         r2_scores.append(r2)

     print("\n")
     print("-----------------------------------------------")
     print("| Mean Squared Error:", np.mean(mse_scores),"    |")
     print("| R-squared:", np.mean(r2_scores), "              |")
     print("-----------------------------------------------")
```

Figure 62: Cross-validation on random forest model

```python
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import KFold
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Input
import numpy as np

mse_scores = []
r2_scores = []
i = 0

for train_index, val_index in KFold(n_splits=5, shuffle=True, random_state=42).split(X_train):
    i += 1

    x_train_fold, x_val_fold = X_train.iloc[train_index], X_train.iloc[val_index]
    y_train_fold, y_val_fold = y_train.iloc[train_index], y_train.iloc[val_index]

    neural_model_temp = Sequential()
    neural_model_temp.add(Input(shape=(X_train.shape[1],)))
    neural_model_temp.add(Dense(128, activation='relu'))
    neural_model_temp.add(Dropout(0.2))
    neural_model_temp.add(Dense(64, activation='relu'))
    neural_model_temp.add(Dropout(0.2))
    neural_model_temp.add(Dense(1))
    neural_model_temp.compile(optimizer='adam', loss='mse', metrics=['mae'])

    neural_model_temp.fit(X_train, y_train, epochs=35, batch_size=32, verbose=0)

    y_pred_val = neural_model_temp.predict(x_val_fold)

    mse = mean_squared_error(y_val_fold, y_pred_val)
    r2 = r2_score(y_val_fold, y_pred_val)

    print("-------------------------------------------")
    print("Phase: ", i)
    print("Mean Squared Error:", mse)
    print("R-squared:", r2)

    mse_scores.append(mse)
    r2_scores.append(r2)


print("\n")
print("-----------------------------------------------")
print("| Mean Squared Error:", np.mean(mse_scores), "   |")
print("| R-squared:", np.mean(r2_scores), "              |")
print("-----------------------------------------------")
```

Figure 63: Cross-validation on the MLP model

```python
from xgboost import XGBRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import KFold
import numpy as np

mse_scores_2 = []
r2_scores_2 = []
i = 0

for train_index, val_index in KFold(n_splits=5, shuffle=True, random_state=42).split(X_train):
    i += 1

    x_train_fold, x_val_fold = X_train.iloc[train_index], X_train.iloc[val_index]
    y_train_fold, y_val_fold = y_train.iloc[train_index], y_train.iloc[val_index]

    model_temp = XGBRegressor()
    model_temp.fit(x_train_fold, y_train_fold)

    y_pred_val = model_temp.predict(x_val_fold)

    mse = mean_squared_error(y_val_fold, y_pred_val)
    r2 = r2_score(y_val_fold, y_pred_val)

    print("-----------------------------------------")
    print("Phase: ", i)
    print("Mean Squared Error:", mse)
    print("R-squared:", r2)

    mse_scores_2.append(mse)
    r2_scores_2.append(r2)

print("\n")
print("-------------------------------------------------")
print("| Mean Squared Error:", np.mean(mse_scores_2), "   |")
print("| R-squared:", np.mean(r2_scores_2), "              |")
print("-------------------------------------------------")
```

Figure 64: Cross-validation on the Xgboost model

Model Training

In Figure 65, I use DBScan which is a clustering algorithm to group properties based on latitude, longitude, and price. This can help real estate companies or investors identify regions with similar pricing structures.

```
[146] import pandas as pd
      from sklearn.cluster import DBSCAN
      from sklearn.preprocessing import MinMaxScaler
      from sklearn.metrics import silhouette_score
      import matplotlib.pyplot as plt
      import numpy as np


      df = pd.DataFrame(combined_data_cop)

      clustering_features = df[['Longtitude', 'Lattitude', 'Price']]

      scaler = MinMaxScaler()
      clustering_features_scaled = scaler.fit_transform(clustering_features)

      dbscan = DBSCAN(eps=0.3, min_samples=2)
      labels = dbscan.fit_predict(clustering_features_scaled)

      df['Cluster'] = labels

      core_samples_mask = (labels != -1)
      if np.sum(core_samples_mask) > 1:
          silhouette_avg = silhouette_score(clustering_features_scaled[core_samples_mask], labels[core_samples_mask])
          print(f'Silhouette Score: {silhouette_avg:.2f}')
      else:
          print("Not enough points in clusters to calculate Silhouette Score")

      n_noise = np.sum(labels == -1)
      print(f'Number of noise points: {n_noise}')

      plt.figure(figsize=(8, 6))
      plt.scatter(df['Longtitude'], df['Lattitude'], c=df['Cluster'], cmap='viridis', s=100)
      plt.title('DBSCAN Clustering by Longtitude, Lattitude, and Price')
      plt.xlabel('Longtitude')
      plt.ylabel('Lattitude')
      plt.colorbar(label='Cluster')
      plt.show()

      Silhouette Score: 0.84
      Number of noise points: 0
```

Figure 65: DBScan to cluster data based on latitude, longitude, and price

In Figure 66, I use sklearn to build 2 ensemble models: random forest (regressor) and xgboost (regressor) to predict the price. Leveraging the outliers handle and ensemble learning can give good prediction accuracy.

```
[128] from sklearn.ensemble import RandomForestRegressor
      from sklearn.metrics import mean_squared_error, r2_score

      model1 = RandomForestRegressor()
      model1.fit(X_train, y_train)

      y_pred = model1.predict(X_test)

      mse = mean_squared_error(y_test, y_pred)
      r2 = r2_score(y_test, y_pred)

      print("Score: ", model1.score)
      print("Mean Squared Error:", mse)
      print("R-squared:", r2)

      Score:  <bound method RegressorMixin.score of RandomForestRegressor()>
      Mean Squared Error: 0.006765148140227325
      R-squared: 0.8269231419534173
```

Figure 66: Training process of random forest model

```
from xgboost import XGBRegressor
from sklearn.metrics import mean_squared_error, r2_score

model3 = XGBRegressor()
model3.fit(X_train, y_train)

y_pred3 = model3.predict(X_test)

mse = mean_squared_error(y_test, y_pred3)
r2 = r2_score(y_test, y_pred3)

print("Mean Squared Error:", mse)
print("R-squared:", r2)

Mean Squared Error: 0.0063792747759382415
R-squared: 0.8367951725595018
```

Figure 67: Training process of the xgboost model

In Figure 68, I use TensorFlow to build a simple neural network model named

multilayer perceptron (MLP). This model can learn non-linear data through backpropagation, thus it may give a good prediction on this dataset. I have to fine-tune many times and this setting gives the best prediction accuracy. Although I try to use the L2 regularizer when the prediction is bad, applying the regularizer gives a worse result. This model may not get the overfitting, thus applying L2 can't address the issue. Moreover, I try to change the activation function, learning rate, batch size, optimizer, and epoch and this parameter setting gives a good result.

```
[124] from tensorflow.keras.models import Sequential
      from tensorflow.keras.layers import Dense, Dropout, Input
      from tensorflow.keras.optimizers import Adam

      learning_rate = 0.0009

      adam_optimizer = Adam(learning_rate=learning_rate)

      model2 = Sequential()
      model2.add(Input(shape=(X_train.shape[1],)))
      model2.add(Dense(128, activation='relu'))
      model2.add(Dropout(0.2))
      model2.add(Dense(64, activation='relu'))
      model2.add(Dropout(0.2))
      model2.add(Dense(1))

      model2.compile(optimizer=adam_optimizer, loss='mse', metrics=['mae'])

      model2.fit(X_train, y_train, epochs=35, batch_size=32)

      796/796 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 0.0124 - mae: 0.0779
      Epoch 8/35
      796/796 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 0.0121 - mae: 0.0769
      Epoch 9/35
```

Figure 68: Training process of the MLP model

```
from sklearn.metrics import mean_squared_error, r2_score

y_pred2 = model2.predict(X_test)
model2.evaluate(X_test, y_test)
print('Mean Squared Error Score: ', mean_squared_error(y_test, y_pred2))
print('R2 Score: ', r2_score(y_test, y_pred2))
model2.summary()
```

```
199/199 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
199/199 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0100 - mae: 0.0690
Mean Squared Error Score:  0.009662767966440378
R2 Score:  0.7527915893341441
Model: "sequential_2"
```

Layer (type)	Output Shape	Param #
dense_6 (Dense)	(None, 128)	2,560
dropout_4 (Dropout)	(None, 128)	0
dense_7 (Dense)	(None, 64)	8,256
dropout_5 (Dropout)	(None, 64)	0
dense_8 (Dense)	(None, 1)	65

```
Total params: 32,645 (127.52 KB)
Trainable params: 10,881 (42.50 KB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 21,764 (85.02 KB)
```

Figure 69: MLP evaluation metrics

b. Implementation Evaluation

Cross-validation Metrics

The cross-validation table below for 3 algorithms shows the MSE (Mean Absolute Error) and R^2 score in each fold.

- The R^2 score of random forest and xgboost are consistent across all the folds. Also, the R^2 score of both models is high and this indicates that the model has good performance and its capability to capture the patterns in data very well. For the MLP, the R^2 score varies more than the 2 models above. Also, the R^2 has a relatively low value (0.76) which indicates that the model can only explain 76% variance of data.
- The MSE score of the XGBoost model has the lowest value (0.0066)

which indicates that the model has very few errors between the predicted and actual values. The random forest with 0.0070 which is slightly higher than the XGBoost. However, it still indicates that this value is good with very few errors between the predicted and actual price. The MLP has the highest value in MSE (0.0092) which is moderately higher than the 2 models above. However, this value is acceptable.

Fold Number	Random Forest R^2	XGBoost R^2	MLP R^2	Random Forest MSE	XGBoost MSE	MLP MSE
1	0.822643	0.831631	0.776060	0.007203	0.006838	0.009095
2	0.822733	0.832578	0.761020	0.006955	0.006569	0.009377
3	0.822008	0.837873	0.779264	0.006927	0.006394	0.008590
4	0.831834	0.834298	0.770987	0.006926	0.006824	0.009431
5	0.816634	0.829044	0.761099	0.007297	0.006839	0.009509
Average Score	0.823172	0.833085	0.769686	0.007061	0.006669	0.009202

Accuracy Table

Each algorithm has different evaluation metrics, the DBScan is not exceptional. The silhouette can measure how similar each point is to its own cluster compared to other clusters. The number of noise points in DBSCAN refers to the data points that do not belong to any cluster. These points are considered outliers or anomalies because they are too far away from other points or regions of high

density. Therefore, this metric can be a tool to identify the outliers in data. The silhouette score of DBScan is very high which points out that The clusters are well-defined, and points are well-associated with their respective clusters. Moreover, there is no noise in the data and there are no points considered as outliers. Therefore, the metrics show that the data is cleaned well and this is a suitable choice for exploring the housing market price based on latitude and longitude.

Evaluation Metrics Table on clustering algorithm

	DBScan
Silhouette Score	0.84
Number of noise points	0

These metrics show that all models can deal with unseen data and learn the non-linear relationship of all features. The model gives the best performance is XGBoost with very good scores in both evaluation metrics. Following that, the random forest also gives very good scores and the performance is slightly weaker than the XGBoost model. The MLP model has the worst score in both evaluation metrics. I have tried many ways from fine-tuning the model, changing the scaler, and adding L2 but this parameter setting gives the best accuracy compared to other configurations. Therefore, this result of MLP may come from the distribution of data and the complexity is increased. The reason why I conclude this is because when the model is trained on a single dataset (Melbourne data). The score of the R^2 evaluation metric gives a better result (Figure 70). In contrast, combining 2 data gives better metrics for xgboost and random forest. The metrics of the random forest, when trained on single data (MSE: 0.0083 - R^2:

0.84), were worse than training on combined data (MSE: 0.0067 - R^2: 0.82) (Figure 71). However, the xgboost model gives a significant improvement in the MSE metric (MSE: 0.08 - R^2: 0.85). The MSE score of xgboost when trained on Melbourne data gives only 0.08 but when trained on combined data, it reduces significantly to 0.0067 (Figure 72). This indicates that the model can handle well on complex data and this might be the result of increasing the data samples. Overall, the XGBoost is the well-suit model for combined data with impressed metrics. Moreover, the random forest gives a consistently good on both data (single Melbourne data and combined data). The MLP gives the worst result in the 3 models in both single data trained and combined data trained. Thus, it is not suitable for this task. In conclusion, XGBoost is the model chosen for this task

Evaluation Metrics Table on combined data

	Random Forest	**XGBoost**	**MLP**
R^2 Score	0.826923	0.836795	0.752791
MSE Score	0.006765	0.006379	0.009662

```
132/132 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
132/132 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.0095 - mae: 0.0688
Mean Squared Error Score:  0.009666073697711932
R2 Score:  0.8222586978010296
Model: "sequential"
```

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 128)	2,944
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 64)	8,256
dropout_1 (Dropout)	(None, 64)	0
dense_2 (Dense)	(None, 1)	65

```
Total params: 33,797 (132.02 KB)
Trainable params: 11,265 (44.00 KB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 22,532 (88.02 KB)
```

Figure 70: Evaluation metrics of the MLP model train and test on Melbourne data (single dataset)

```
[ ] from sklearn.ensemble import RandomForestRegressor
    from sklearn.metrics import mean_squared_error, r2_score

    model1 = RandomForestRegressor()
    model1.fit(X_train2, y_train2)

    y_pred = model1.predict(X_test2)

    mse = mean_squared_error(y_test2, y_pred)
    r2 = r2_score(y_test2, y_pred)

    print("Score: ", model1.score)
    print("Mean Squared Error:", mse)
    print("R-squared:", r2)
```

```
Score:  <bound method RegressorMixin.score of RandomForestRegressor()>
Mean Squared Error: 0.0083538586515692
R-squared: 0.8463879170021723
```

Figure 71: Evaluation metrics of the Random Forest model train and test on Melbourne data (single dataset)

```python
from xgboost import XGBRegressor
from sklearn.metrics import mean_squared_error, r2_score

model3 = XGBRegressor()
model3.fit(X_train, y_train)

y_pred3 = model3.predict(X_test)

mse = mean_squared_error(y_test, y_pred3)
r2 = r2_score(y_test, y_pred3)

print("Mean Squared Error:", mse)
print("R-squared:", r2)
```

```
Mean Squared Error: 0.08704046267419392
R-squared: 0.8536665824583486
```

Figure 72: Evaluation metrics of the Xgboost model train and test on Melbourne data (single dataset)

Result Visualization

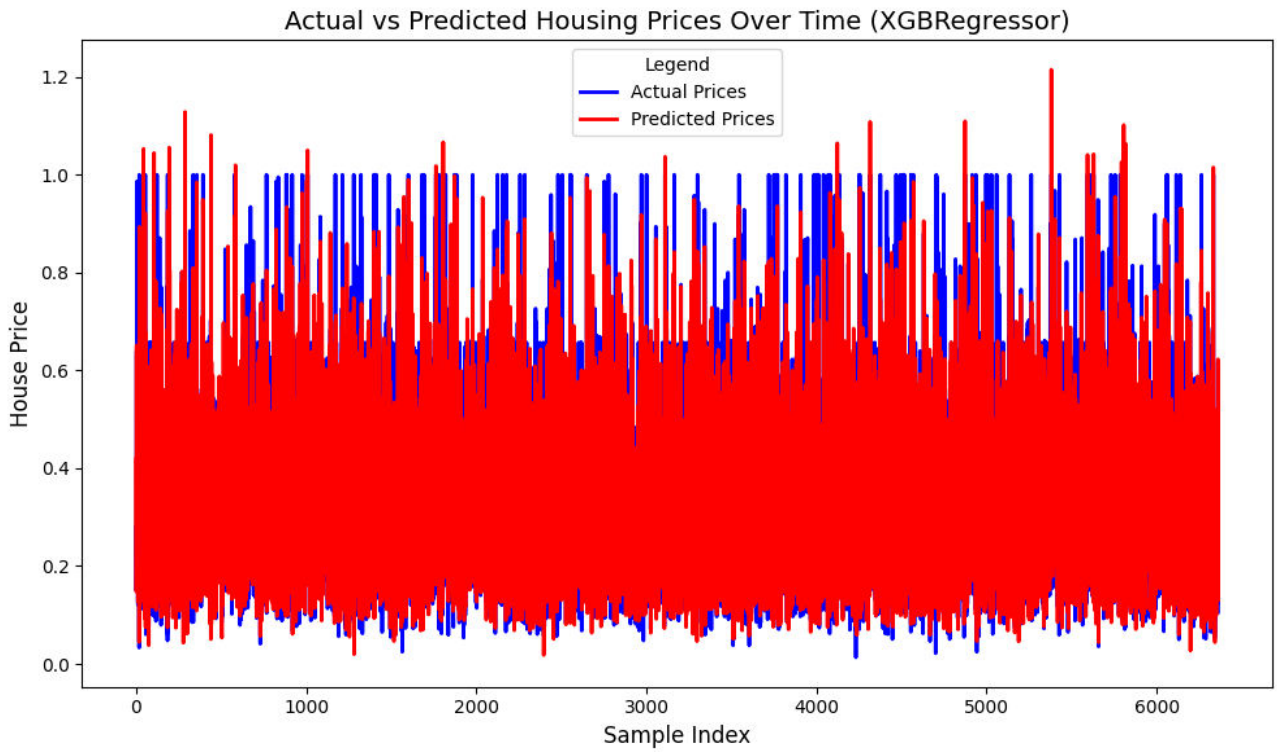


Figure 73: Comparing actual and predicted price using the XGBoost model

Figure 74: Comparing the actual and predicted price with a scatterplot using the Random Forest model

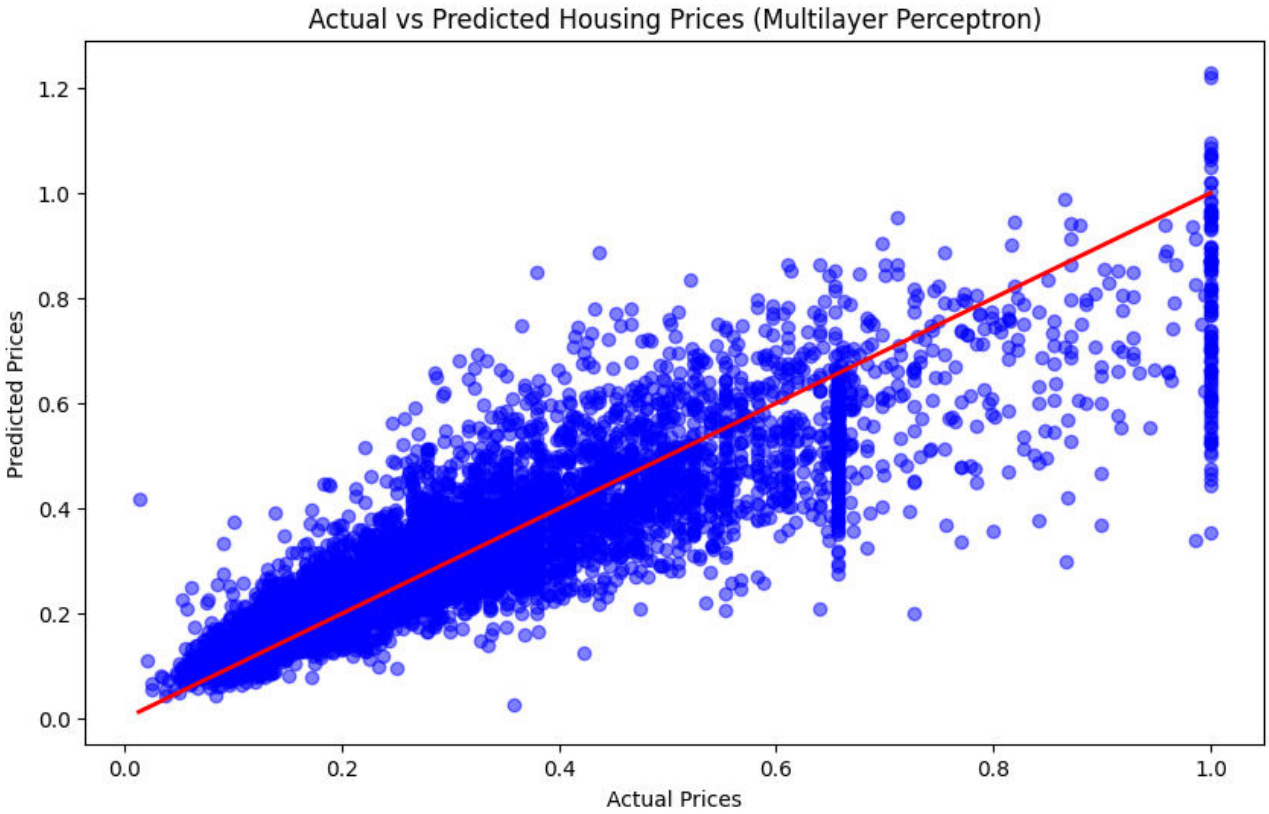


Figure 75: Comparing the actual and predicted price with a scatterplot using the MLP model

Figure 76: Comparing the actual and predicted price with a scatterplot using the XGBoost model

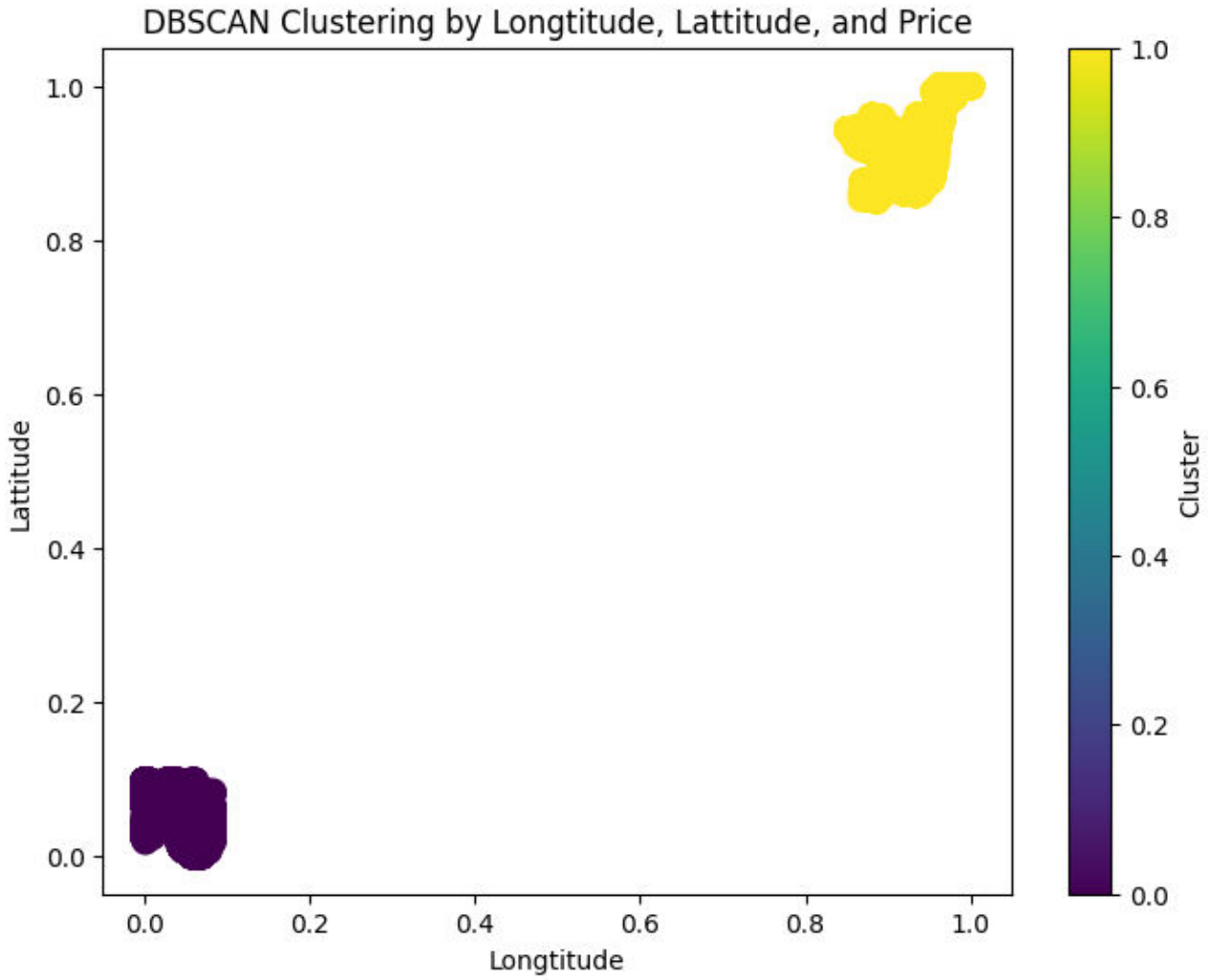


Figure 77: The visualization of the DBScan model after clustering

7. Conclusion

In conclusion, the model's result gives good prediction accuracy, thus it can help homebuyers select the house suited to their demand at affordable prices. Furthermore, combining 2 main datasets (Melbourne and Sydney) aims to clients in 2 different cities which have a wide range of clients, and clients can compare the price of real estate in 2 different cities to decide where to live. Also, combining 3 additional financial and real estate data makes the client understand the differences in real estate features of each city and the house price in each period. However, the model still has some constraints, especially in aiming for a wider range of clients (all of Australia) and the model's accuracy can be enhanced more to get the best score.

References

Beyer, L., Hénaff, O.J., Kolesnikov, A., Zhai, X. and Oord, A. van den (2020). Are we done with ImageNet? *arXiv:2006.07159 [cs]*. [online] Available at: https://arxiv.org/abs/2006.07159.

Eichholtz, Veld and Schweitzer (2024). *Performance of Financial Institutions*. [online] Google Books. Available at: https://books.google.com.au/books?hl=en&lr=&id=uyhKV9Mok1YC&oi=fnd&pg=PA199&ots=yTWlmurnW_&sig=NC3ISSDMhNoi3ypEcjg6O0KKZXo&redir_esc=y#v=onepage&q&f=false [Accessed 1 Oct. 2024].

Grubbs, F.E. (1969). Procedures for Detecting Outlying Observations in Samples. *Technometrics*, [online] 11(1), p.1. doi:https://doi.org/10.2307/1266761.

Ho, W.K.O., Tang, B.-S. and Wong, S.W. (2020). Predicting property prices with machine learning algorithms. *Journal of Property Research*, 38(1), pp.1–23. doi:https://doi.org/10.1080/09599916.2020.1832558.

Karlaš, B., Li, P., Wu, R., Gürel, Nezihe Merve, Chu, X., Wu, W. and Zhang, C. (2024). *Nearest Neighbor Classifiers over Incomplete Information: From Certain Answers to Certain Predictions*. [online] arXiv.org. Available at: https://arxiv.org/abs/2005.05117 [Accessed 1 Oct. 2024].

Majkowska, A., Mittal, S., Steiner, D.F., Reicher, J.J., McKinney, S.M., Duggan, G.E., Eswaran, K., Cameron Chen, P.-H., Liu, Y., Kalidindi, S.R., Ding, A., Corrado, G.S., Tse, D. and Shetty, S. (2020). Chest Radiograph Interpretation with Deep Learning Models: Assessment with Radiologist-adjudicated Reference Standards and Population-adjusted Evaluation. *Radiology*, 294(2), pp.421–431. doi:https://doi.org/10.1148/radiol.2019191293.

Truong, Q., Nguyen, M., Dang, H. and Mei, B. (2020). Housing Price Prediction via Improved Machine Learning Techniques. *Procedia Computer Science*, 174, pp.433–442. doi:https://doi.org/10.1016/j.procs.2020.06.111.

Wang, X., Peng, Y., Lu, L., Lu, Z., Bagheri, M. and Summers, R. (2017). *ChestX-ray8: Hospital-scale Chest X-ray Database and Benchmarks on Weakly-Supervised Classification and Localization of Common Thorax Diseases*. [online] Available at: http://openaccess.thecvf.com/content_cvpr_2017/papers/Wang_ChestX-ray8_Hospital-Scale_Chest_CVPR_2017_paper.pdf.

Appendix

Melbourne data analysis and prediction

There is 1 file that cleans, trains, and tests all the models in 1 single dataset (Melbourne Dataset)

Data Source

https://www.kaggle.com/datasets/anthonypino/melbourne-housing-market?select=Melbourne_housing_FULL.csv

https://www.kaggle.com/datasets/alexlau203/sydney-house-prices/data

https://www.matthewproctor.com/australian_postcodes#downloadlinks

http://ausmacrodata.org/series.php?id=crtdoirymmdir

https://www.abs.gov.au/statistics/economy/price-indexes-and-inflation/residential-property-price-indexes-eight-capital-cities/dec-2021