INFOSYS 722 Research Report

Factors Affecting Sydney House Prices

Iteration 3 OSAS (Steps 1 – 8)

Shiyu Lin

UPI: slin648

ID: 502522556

Department of Information Systems and Operations Management

(ISOM)

University of Auckland, New Zealand

Contents

1. Situation understanding

1.1 Identify the objectives of the situation

Sustainable cities and human settlements is one of the 17 Sustainable Development Goals of the UN. [1] Cities are centers of business, culture, science, productivity, social, human and economic development. Urban planning, transport systems, water, sanitation, waste management, disaster risk reduction, access to information, education and capacity building are all issues relevant to sustainable urban development.

In 2008, the global urban population exceeded the rural population for the first time in history. [1] This milestone marks the arrival of a new “urban millennium”, where by 2050 two-thirds of the world’s population is expected to live in urban areas.

In some cities, housing prices have experienced severe housing bubbles, such as Tokyo, where house prices were as high as $220,000 per square meter, and now are far below this value. Today, housing prices in some cities are seriously too high for local income levels, such as Shenzhen and Shanghai in China. All this poses a challenge to achieve the goal of sustainable cities and human settlements.

"Promoting sustainable human settlements development" is the subject of Chapter 7 of Agenda 21, which calls for providing adequate shelter for all, and providing adequate environment and support.

We know that it is very difficult to provide shelter for everyone while ensuring that the environment of the shelter is good. The price of a house is affected by many factors. This includes internal factors, such as the size of the house, the size of the house, etc.; it also includes external factors, such as the location of the house, the crime rate in the neighborhood where the house is located, etc. [2] A report by Etch Real Estate states that one of the most important short-term factors affecting house prices is interest rates. When the cost of borrowing money decreases, the number of qualified buyers increases. The price of a house is also related to the macroeconomic situation in the area.

1.2 Assess the situation

In the current situation, we need to collect data and analyze the data to report the factors affecting house prices, and our recommendations. This conclusion and recommendations are not based on experience - in fact I am not an expert in business or real estate. We need to analyze the data in the manner of a data scientist and report our findings.

The data we are looking for should contain one or more response variables to record house prices; it should also contain multiple explanatory variables that explain what factors affect house prices. Each explanatory variable can be a potentially significant factor, but it can also be a trivial factor. Importantly, the data needs to be relevant: we need a row of data to describe a particular property transaction, or the price of a particular property.

Since we are not allowed to gather our own data without having Ethics Approval, and cannot carry out interviews or perform surveys without applying for Ethics Approval from the University of Auckland, I need to look for publicly available datasets. These datasets are provided and made available to the public by some agencies, who have ensured that the datasets do not contain sensitive information, such as other people's privacy and personal information.

The data we collect may contain some quality issues, such as outliers, or incomplete data. We need to examine and process the data to make sure it is reliable. This allows us to build reliable models and draw meaningful conclusions.

There are many factors that affect house prices, some of which may not be described in the explanatory variables of the dataset. We assume that factors not described in the dataset are trivial, but in fact it may not be the case.

To complete this iteration, I decided to use R and Python for my work. Both R and Python are open-source software and belong to the Opensource Software Stack. I'll be doing some data collection/splitting in R and the all rest in Python. Some Python packages are also used, such as scikit-learn and matplotlib.

1.3 Determine data mining objectives

Our objectives are Explanation and Prediction.

Explanation: I'll make assertions such as “a 1% increase in neighborhood crime, a 2% decrease in home prices, ceteris paribus”. I will also build models to find out which factors have the greatest impact on house prices.

Prediction: I will split the dataset into training and test sets, model on the training set, and test the accuracy of the model on the test set.

Desired Outcome:
1. Get to know what factors affect house prices the most, and what's trivial.
2. Make the model obtain a high prediction accuracy in the test set.
3. Based on the actual situation, I will explain my conclusions and put forward some logical conjectures.

1.4 Produce a project plan

Step	Duration	Resources	Risks
Situation understanding	3 days	references	insufficient progress
Technical preparations	3 days	Technical Support	Software cannot be installed
Data preprocessing	5 days	EXCEL R Python etc.	Data and technical issues
Modeling	6 days	R Python etc.	Failed to write correct codes
Evaluation	3 days	R Python etc.	Failed to write correct codes

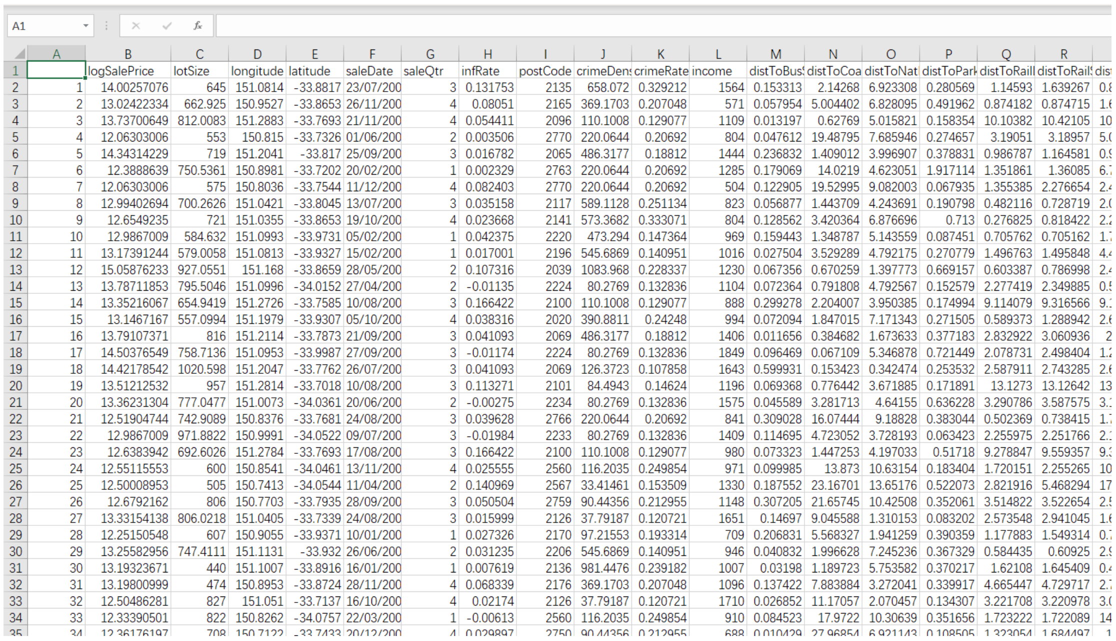
Table 1.4.1: Project Plan

2. Data understanding

2.1 Collect initial data

For this research, the data came from the R package [3]HRW: Datasets, Functions and Scripts for Semiparametric Regression Supporting Harezlak, Ruppert & Wand (2018). This is an open-source R package that contains a large number of datasets.

First, I load the HRW package in R and load the SydneyRealEstate dataset. Then I export the dataset file in CSV format.



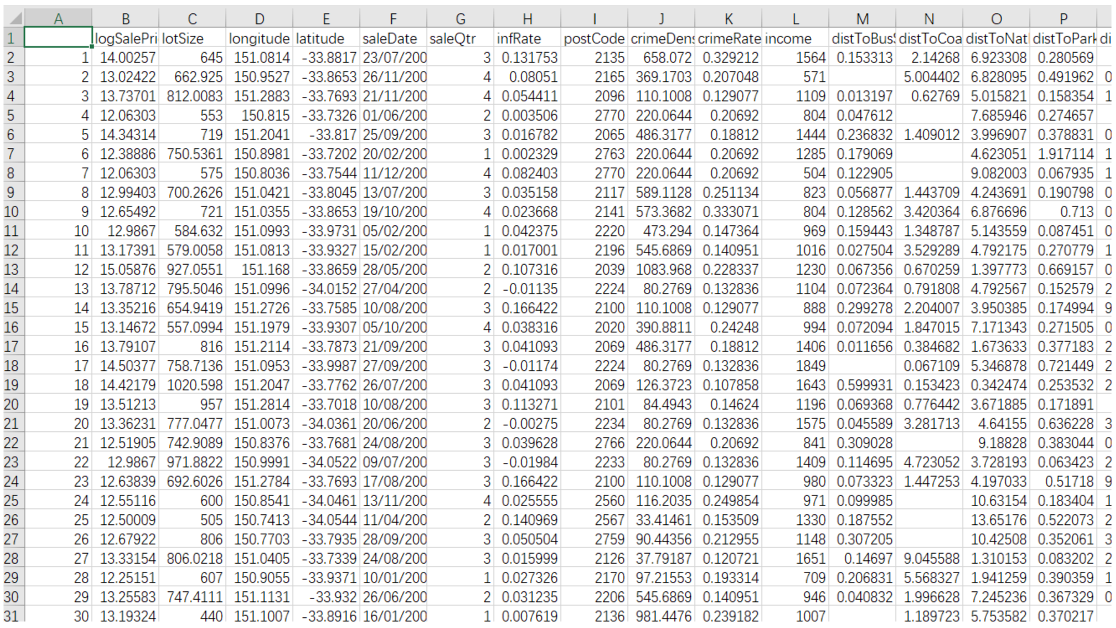
A1

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	
1		logSalePrice	lotSize	longitude	latitude	saleDate	saleQtr	infRate	postCode	crimeDens	crimeRate	income	distToBus	distToCoa	distToNat	distToPark	distToRail	distToRail	dis
2	1	14.00257076	645	151.0814	-33.8817	23/07/200	3	0.131753	2135	658.072	0.329212	1564	0.153313	2.14268	6.923308	0.280569	1.14593	1.639267	0.8
3	2	13.02422334	662.925	150.9527	-33.8653	26/11/200	4	0.08051	2165	369.1703	0.207048	571	0.057954	5.004402	6.828095	0.491962	0.874182	0.874715	1.6
4	3	13.73700649	812.0083	151.2883	-33.7693	21/11/200	4	0.054411	2096	110.1008	0.129077	1109	0.013197	0.62769	5.015821	0.158354	10.10382	10.42105	10
5	4	12.06303006	553	150.815	-33.7326	01/06/200	2	0.003506	2770	220.0644	0.20692	804	0.047612	19.48795	7.685946	0.274657	3.19051	3.18957	5.0
6	5	14.34314229	719	151.2041	-33.817	25/09/200	3	0.016782	2065	486.3177	0.18812	1444	0.236832	1.409012	3.996907	0.378831	0.986787	1.164581	0.9
7	6	12.3888639	750.5361	150.8981	-33.7202	20/02/200	1	0.002329	2763	220.0644	0.20692	1285	0.179069	14.0219	4.623051	1.917114	1.351861	1.36085	6.7
8	7	12.06303006	575	150.8036	-33.7544	11/12/200	4	0.082403	2770	220.0644	0.20692	504	0.122905	19.52995	9.082003	0.067935	1.355385	2.276654	2.4
9	8	12.99402694	700.2626	151.0421	-33.8045	13/07/200	3	0.035158	2117	589.1128	0.251134	823	0.056877	1.443709	4.243691	0.190798	0.482116	0.728719	2.0
10	9	12.6549235	721	151.0355	-33.8653	19/10/200	4	0.023668	2141	573.3682	0.333071	804	0.128562	3.420364	6.876696	0.713	0.276825	0.818422	2.2
11	10	12.9867009	584.632	151.0993	-33.9731	05/02/200	1	0.042375	2220	473.294	0.147364	969	0.159443	1.348787	5.143559	0.087451	0.705762	0.705162	1.7
12	11	13.17391244	579.0058	151.0813	-33.9327	15/02/200	1	0.017001	2196	545.6869	0.140951	1016	0.027504	3.529289	4.792175	0.270779	1.496763	1.495848	4.4
13	12	15.05876233	927.0551	151.168	-33.8659	28/05/200	2	0.107316	2039	1083.968	0.228337	1230	0.067356	0.670259	1.397773	0.669157	0.603387	0.786998	2.4
14	13	13.78711853	795.5046	151.0996	-34.0152	27/04/200	2	-0.01135	2224	80.2769	0.132836	1104	0.072364	0.791808	4.792567	0.152579	2.277419	2.349885	0.5
15	14	13.35216067	654.9419	151.2726	-33.7585	10/08/200	3	0.166422	2100	110.1008	0.129077	888	0.299278	2.204007	3.950385	0.174994	9.114079	9.316566	9.1
16	15	13.1467167	557.0994	151.1979	-33.9307	05/10/200	4	0.038316	2020	390.8811	0.24248	994	0.072094	1.847015	7.171343	0.271505	0.589373	1.288942	2.6
17	16	13.79107371	816	151.2114	-33.7873	21/09/200	3	0.041093	2069	486.3177	0.18812	1406	0.011656	0.384682	1.673633	0.377183	2.832922	3.060936	2
18	17	14.50376549	758.7136	151.0953	-33.9987	27/09/200	3	-0.01174	2224	80.2769	0.132836	1849	0.096469	0.067109	5.346878	0.721449	2.078731	2.498404	1.2
19	18	14.42178542	1020.598	151.2047	-33.7762	26/07/200	3	0.041093	2069	126.3723	0.107858	1643	0.599931	0.153423	0.342474	0.253532	2.587911	2.743285	2.6
20	19	13.51212532	957	151.2814	-33.7018	10/08/200	3	0.113271	2101	84.4943	0.14624	1196	0.069368	0.776442	3.671885	0.171891	13.1273	13.12642	13
21	20	13.36231304	777.0477	151.0073	-34.0361	20/06/200	2	-0.00275	2234	80.2769	0.132836	1575	0.045589	3.281713	4.64155	0.636228	3.290786	3.587575	3.1
22	21	12.51904744	742.9089	150.8376	-33.7681	24/08/200	3	0.039628	2766	220.0644	0.20692	841	0.309028	16.07444	9.18828	0.383044	0.502369	0.738415	1.7
23	22	12.9867009	971.8822	150.9991	-34.0522	09/07/200	3	-0.01984	2233	80.2769	0.132836	1409	0.114695	4.723052	3.728193	0.063423	2.255975	2.251766	2.1
24	23	12.6383942	692.6026	151.2784	-33.7693	17/08/200	3	0.166422	2100	110.1008	0.129077	980	0.073323	1.447253	4.197033	0.51718	9.278847	9.559357	9.3
25	24	12.55115553	600	150.8541	-34.0461	13/11/200	4	0.025555	2560	116.2035	0.249854	971	0.099985	13.873	10.63154	0.183404	1.720151	2.255265	10
26	25	12.50008953	505	150.7413	-34.0544	11/04/200	2	0.140969	2567	33.41461	0.153509	1330	0.187552	23.16701	13.65176	0.522073	2.821916	5.468294	17
27	26	12.6792162	806	150.7703	-33.7935	28/09/200	3	0.050504	2759	90.44356	0.212955	1148	0.307205	21.65745	10.42508	0.352061	3.514822	3.522654	2.5
28	27	13.33154138	806.0218	151.0405	-33.7339	24/08/200	3	0.015999	2126	37.79187	0.120721	1651	0.14697	9.045588	1.310153	0.083202	2.573548	2.941045	1.6
29	28	12.25150548	607	150.9055	-33.9371	10/01/200	1	0.027326	2170	97.21553	0.193314	709	0.206831	5.568327	1.941259	0.390359	1.177883	1.549314	0.7
30	29	13.25582956	747.4111	151.1131	-33.932	26/06/200	2	0.031235	2206	545.6869	0.140951	946	0.040832	1.996628	7.245236	0.367329	0.584435	0.60925	2.9
31	30	13.19323671	440	151.1007	-33.8916	16/01/200	1	0.007619	2136	981.4476	0.239182	1007	0.03198	1.189723	5.753582	0.370217	1.62108	1.645409	0.4
32	31	13.19800999	474	150.8953	-33.8724	28/11/200	4	0.068339	2176	369.1703	0.207048	1096	0.137422	7.883884	3.272041	0.339917	4.665447	4.729717	2.7
33	32	12.50486281	827	151.051	-33.7137	16/10/200	4	0.02174	2126	37.79187	0.120721	1710	0.026852	11.17057	2.070457	0.134307	3.221708	3.220978	3.0
34	33	12.33390501	822	150.8262	-34.0757	22/03/200	1	-0.00613	2560	116.2035	0.249854	910	0.084523	17.9722	10.30639	0.351656	1.723222	1.722089	14
35	34	12.36176197	708	150.7122	-33.7433	20/12/200	4	0.029897	2750	90.44356	0.212955	688	0.010429	27.96854	6.921143	0.108505	1.323054	1.684497	1

Figure2.1.1: Clean Dataset

Since this dataset is "clean", I added some missing values so that the data processing steps can follow.

For the distToBusStop variable, I randomly lost 20% of the data; for distToCoastline, I dropped more data with larger values. See section 2.4 for specific discarding rules.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	
1		logSalePri	lotSize	longitude	latitude	saleDate	saleQtr	infRate	postCode	crimeDens	crimeRate	income	distToBus	distToCoa	distToNat	distToPark	di
2	1	14.00257	645	151.0814	-33.8817	23/07/200	3	0.131753	2135	658.072	0.329212	1564	0.153313	2.14268	6.923308	0.280569	
3	2	13.02422	662.925	150.9527	-33.8653	26/11/200	4	0.08051	2165	369.1703	0.207048	571		5.004402	6.828095	0.491962	0
4	3	13.73701	812.0083	151.2883	-33.7693	21/11/200	4	0.054411	2096	110.1008	0.129077	1109	0.013197	0.62769	5.015821	0.158354	1
5	4	12.06303	553	150.815	-33.7326	01/06/200	2	0.003506	2770	220.0644	0.20692	804	0.047612		7.685946	0.274657	
6	5	14.34314	719	151.2041	-33.817	25/09/200	3	0.016782	2065	486.3177	0.18812	1444	0.236832	1.409012	3.996907	0.378831	0
7	6	12.38886	750.5361	150.8981	-33.7202	20/02/200	1	0.002329	2763	220.0644	0.20692	1285	0.179069		4.623051	1.917114	1
8	7	12.06303	575	150.8036	-33.7544	11/12/200	4	0.082403	2770	220.0644	0.20692	504	0.122905		9.082003	0.067935	1
9	8	12.99403	700.2626	151.0421	-33.8045	13/07/200	3	0.035158	2117	589.1128	0.251134	823	0.056877	1.443709	4.243691	0.190798	0
10	9	12.65492	721	151.0355	-33.8653	19/10/200	4	0.023668	2141	573.3682	0.333071	804	0.128562	3.420364	6.876696	0.713	0
11	10	12.9867	584.632	151.0993	-33.9731	05/02/200	1	0.042375	2220	473.294	0.147364	969	0.159443	1.348787	5.143559	0.087451	0
12	11	13.17391	579.0058	151.0813	-33.9327	15/02/200	1	0.017001	2196	545.6869	0.140951	1016	0.027504	3.529289	4.792175	0.270779	1
13	12	15.05876	927.0551	151.168	-33.8659	28/05/200	2	0.107316	2039	1083.968	0.228337	1230	0.067356	0.670259	1.397773	0.669157	0
14	13	13.78712	795.5046	151.0996	-34.0152	27/04/200	2	-0.01135	2224	80.2769	0.132836	1104	0.072364	0.791808	4.792567	0.152579	2
15	14	13.35216	654.9419	151.2726	-33.7585	10/08/200	3	0.166422	2100	110.1008	0.129077	888	0.299278	2.204007	3.950385	0.174994	9
16	15	13.14672	557.0994	151.1979	-33.9307	05/10/200	4	0.038316	2020	390.8811	0.24248	994	0.072094	1.847015	7.171343	0.271505	0
17	16	13.79107	816	151.2114	-33.7873	21/09/200	3	0.041093	2069	486.3177	0.18812	1406	0.011656	0.384682	1.673633	0.377183	2
18	17	14.50377	758.7136	151.0953	-33.9987	27/09/200	3	-0.01174	2224	80.2769	0.132836	1849		0.067109	5.346878	0.721449	2
19	18	14.42179	1020.598	151.2047	-33.7762	26/07/200	3	0.041093	2069	126.3723	0.107858	1643	0.599931	0.153423	0.342474	0.253532	2
20	19	13.51213	957	151.2814	-33.7018	10/08/200	3	0.113271	2101	84.4943	0.14624	1196	0.069368	0.776442	3.671885	0.171891	
21	20	13.36231	777.0477	151.0073	-34.0361	20/06/200	2	-0.00275	2234	80.2769	0.132836	1575	0.045589	3.281713	4.64155	0.636228	3
22	21	12.51905	742.9089	150.8376	-33.7681	24/08/200	3	0.039628	2766	220.0644	0.20692	841	0.309028		9.18828	0.383044	0
23	22	12.9867	971.8822	150.9991	-34.0522	09/07/200	3	-0.01984	2233	80.2769	0.132836	1409	0.114695	4.723052	3.728193	0.063423	2
24	23	12.63839	692.6026	151.2784	-33.7693	17/08/200	3	0.166422	2100	110.1008	0.129077	980	0.073323	1.447253	4.197033	0.51718	9
25	24	12.55116	600	150.8541	-34.0461	13/11/200	4	0.025555	2560	116.2035	0.249854	971	0.099985		10.63154	0.183404	1
26	25	12.50009	505	150.7413	-34.0544	11/04/200	2	0.140969	2567	33.41461	0.153509	1330	0.187552		13.65176	0.522073	2
27	26	12.67922	806	150.7703	-33.7935	28/09/200	3	0.050504	2759	90.44356	0.212955	1148	0.307205		10.42508	0.352061	3
28	27	13.33154	806.0218	151.0405	-33.7339	24/08/200	3	0.015999	2126	37.79187	0.120721	1651	0.14697	9.045588	1.310153	0.083202	2
29	28	12.25151	607	150.9055	-33.9371	10/01/200	1	0.027326	2170	97.21553	0.193314	709	0.206831	5.568327	1.941259	0.390359	1
30	29	13.25583	747.4111	151.1131	-33.932	26/06/200	2	0.031235	2206	545.6869	0.140951	946	0.040832	1.996628	7.245236	0.367329	0
31	30	13.19324	440	151.1007	-33.8916	16/01/200	1	0.007619	2136	981.4476	0.239182	1007		1.189723	5.753582	0.370217	

Figure2.1.2: Dataset with missing values

2.2 Describe the data

The data has 37676 rows, representing 37676 transaction records. Each record represents a unique set of properties. Meanwhile, the data has 39 columns, each of which represents an attribute. That said, there are 39 different underlying factors that affect property prices.

The data is in csv format. I purposely export the data to csv format for later use.

[4]Specifically, this data frame contains the following columns:

--logSalePrice: The natural logarithm of the property price. In log(AUD).

--lotSize: The size of the property. In square meters.

--longitude: The longitude of the property's location.

--latitude: The latitude of the property's location.

--saleDate: The sale date of the property.

--saleQtr: The sale quarter of the property.

--infRate: Inflation rate.

--postCode: The postcode of the property's location.

--crimeDensity: The crime density of the property's location.

--crimeRate: The crime rate of the property's location.

--income: The per capita income of the property's location. Calculated as weekly salary.

--distToBusStop: The distance of the property to the nearest bus stop. in kilometers.

--distToCoastline: The distance of the property from the coastline. in kilometers.

--distToNatPark: The distance of the property to the nearest national park. in kilometers.

--distToPark: The distance of the property to the nearest park. in kilometers.

--distToRailLine: The distance of the property from the rail line. in kilometers.

--distToRailStation: The distance of the property from the nearest rail station. in kilometers.

--distToHighway: The distance of the property to the nearest highway. in kilometers.

--distToFreeway: The distance of the property from the nearest freeway. in kilometers.

--distToTunnel: The distance of the property from the undersea tunnel. in kilometers.

--distToMainRoad: The distance of the property from the main road. in kilometers.

--distToSealedRoad: The distance of the property from Sealed Road. in kilometers.

--distToUnsealedRoad: The distance of the property from Unsealed Road. in kilometers.

--airNoise: The air noise of the property's location.

--foreignerRatio: The proportion of foreigners in the location where the property is located.

--distToGPO: The distance of the property from the Sydney GPO. in kilometers.

--NO: Nitric oxide concentration at the location of the property.

--NO2: The concentration of nitrogen dioxide at the location of the property.

--ozone: The ozone concentration at the location of the property.

--neph: Total particulate matter concentration at the location of the property.

--PM10: The concentration of particulate matter less than 10 microns in diameter at the location of the property.

--SO2: The concentration of sulfur dioxide at the location of the property.

--distToAmbulance: The distance of the property from the nearest ambulance station. in kilometers.

--distToFactory: The distance of the property to the nearest factory. in kilometers.

--distToFerry: The distance of the property from the nearest ferry terminal. in kilometers.

--distToHospital: The distance of the property from the nearest hospital. in kilometers.

--distToMedical: The distance of the property to the nearest medical facility. in kilometers.

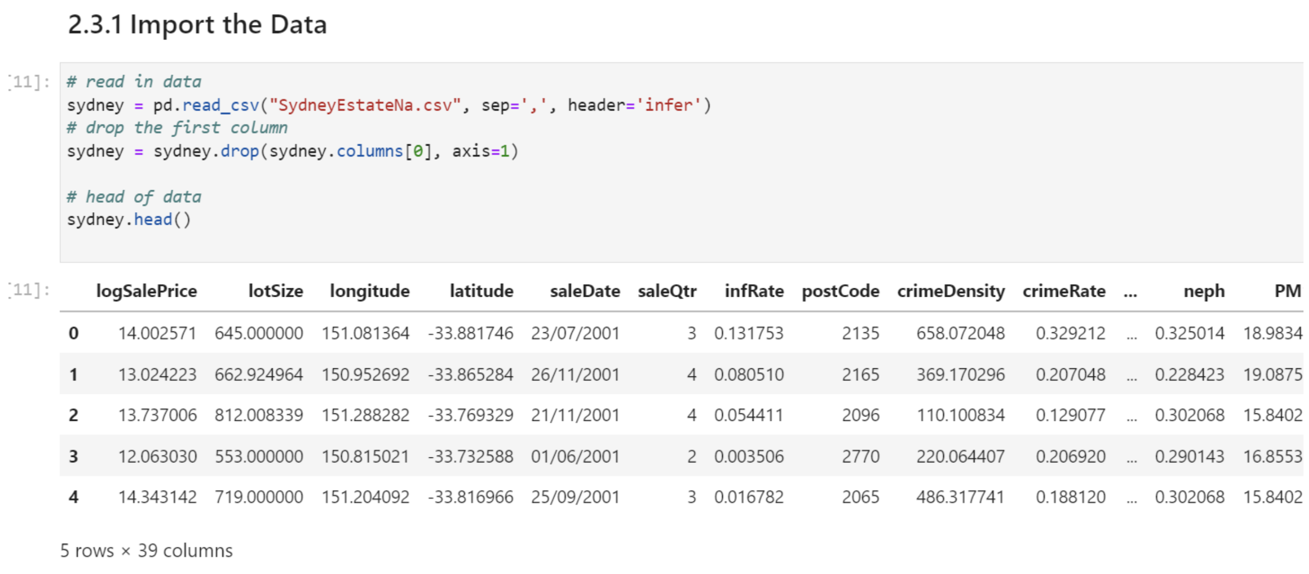
--distToSchool: The distance of the property to the nearest school. in kilometers.

--distToUniversity: The distance of the property to the nearest university. in kilometers.

2.3 Explore the data

2.3.1 Import the Data

The dataset is "SydneyEstateNa.csv" in the same directory as the source code. Read this csv file using pandas and print the first 5 lines.



2.3.1 Import the Data

```
[11]: # read in data
      sydney = pd.read_csv("SydneyEstateNa.csv", sep=',', header='infer')
      # drop the first column
      sydney = sydney.drop(sydney.columns[0], axis=1)

      # head of data
      sydney.head()
```

[11]:

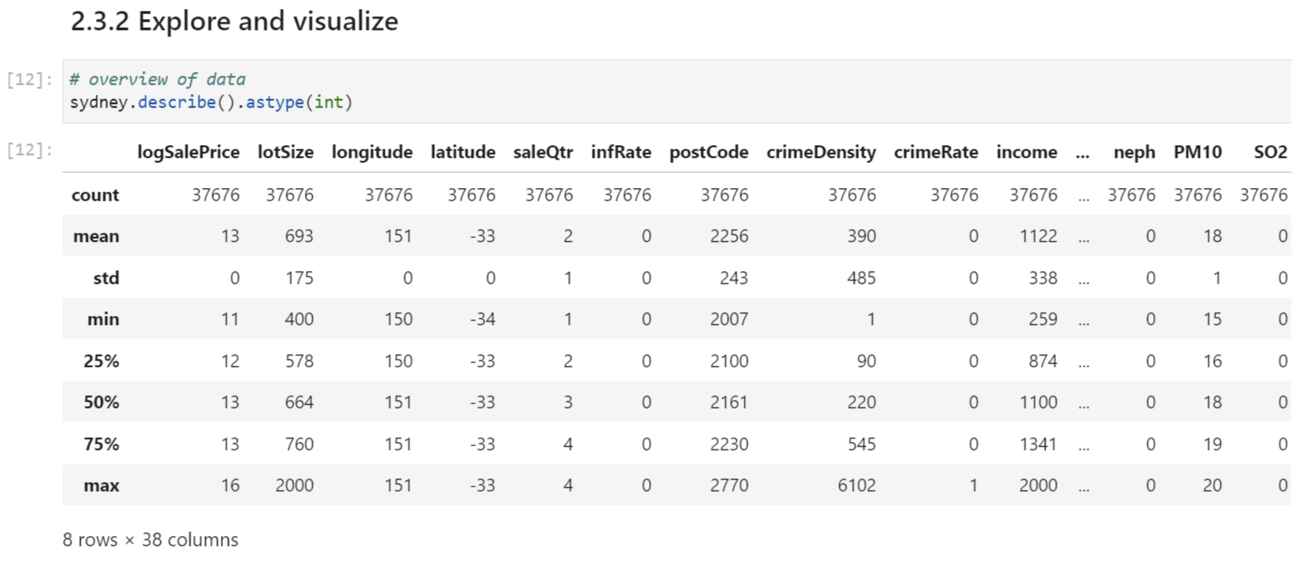
	logSalePrice	lotSize	longitude	latitude	saleDate	saleQtr	infRate	postCode	crimeDensity	crimeRate	...	neph	PM
0	14.002571	645.000000	151.081364	-33.881746	23/07/2001	3	0.131753	2135	658.072048	0.329212	...	0.325014	18.9834
1	13.024223	662.924964	150.952692	-33.865284	26/11/2001	4	0.080510	2165	369.170296	0.207048	...	0.228423	19.0875
2	13.737006	812.008339	151.288282	-33.769329	21/11/2001	4	0.054411	2096	110.100834	0.129077	...	0.302068	15.8402
3	12.063030	553.000000	150.815021	-33.732588	01/06/2001	2	0.003506	2770	220.064407	0.206920	...	0.290143	16.8553
4	14.343142	719.000000	151.204092	-33.816966	25/09/2001	3	0.016782	2065	486.317741	0.188120	...	0.302068	15.8402

5 rows × 39 columns

Figure2.3.1: Data Import

2.3.2 Explore and visualize

First let's focus on the overview of data by the describe() method.



2.3.2 Explore and visualize

```
[12]: # overview of data
      sydney.describe().astype(int)
```

[12]:

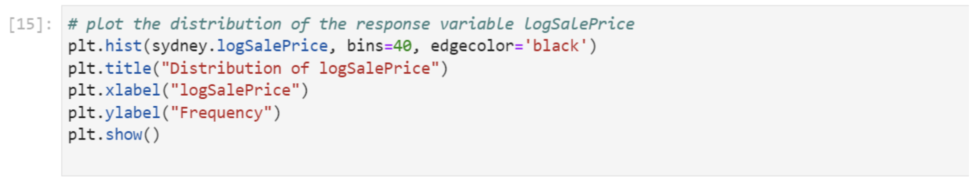
	logSalePrice	lotSize	longitude	latitude	saleQtr	infRate	postCode	crimeDensity	crimeRate	income	...	neph	PM10	SO2
count	37676	37676	37676	37676	37676	37676	37676	37676	37676	37676	...	37676	37676	37676
mean	13	693	151	-33	2	0	2256	390	0	1122	...	0	18	0
std	0	175	0	0	1	0	243	485	0	338	...	0	1	0
min	11	400	150	-34	1	0	2007	1	0	259	...	0	15	0
25%	12	578	150	-33	2	0	2100	90	0	874	...	0	16	0
50%	13	664	151	-33	3	0	2161	220	0	1100	...	0	18	0
75%	13	760	151	-33	4	0	2230	545	0	1341	...	0	19	0
max	16	2000	151	-33	4	0	2770	6102	1	2000	...	0	20	0

8 rows × 38 columns

Figure2.3.2: Data Overview

We can get to know the distribution of the data from the results. For example, the minimum lotSize is 400 square meters and the maximum is 2000 square meters (after rounding to integer).

Let's look at the distribution of each variable. The first is the response variable, the house price. Here, house prices are log-transformed.



```
[15]: # plot the distribution of the response variable logSalePrice
      plt.hist(sydney.logSalePrice, bins=40, edgecolor='black')
      plt.title("Distribution of logSalePrice")
      plt.xlabel("logSalePrice")
      plt.ylabel("Frequency")
      plt.show()
```

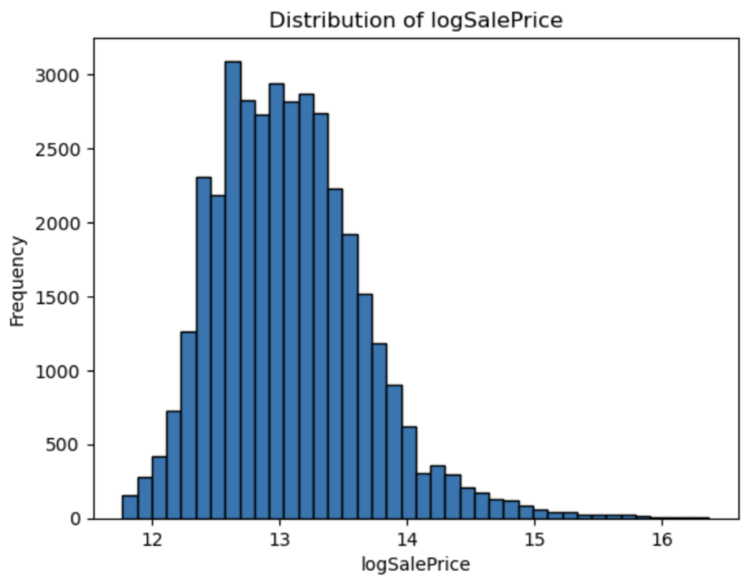
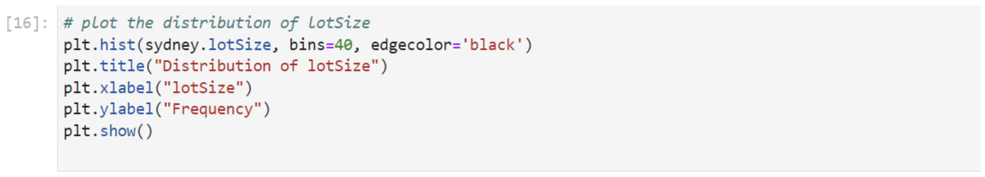


Figure2.3.3: Logged House Price

The housing price is a right-skewed data. Right-skewed data has a long tail that extends to the right. In this case, both the mean and the median are larger than the mode. This is what I expected: everything related to economics is skewed to the right, because there are always some rich people who amass most of the wealth. These outrageous houses are for them.

Next let's look at the lot size.



```
[16]: # plot the distribution of lotSize
      plt.hist(sydney.lotSize, bins=40, edgecolor='black')
      plt.title("Distribution of lotSize")
      plt.xlabel("lotSize")
      plt.ylabel("Frequency")
      plt.show()
```

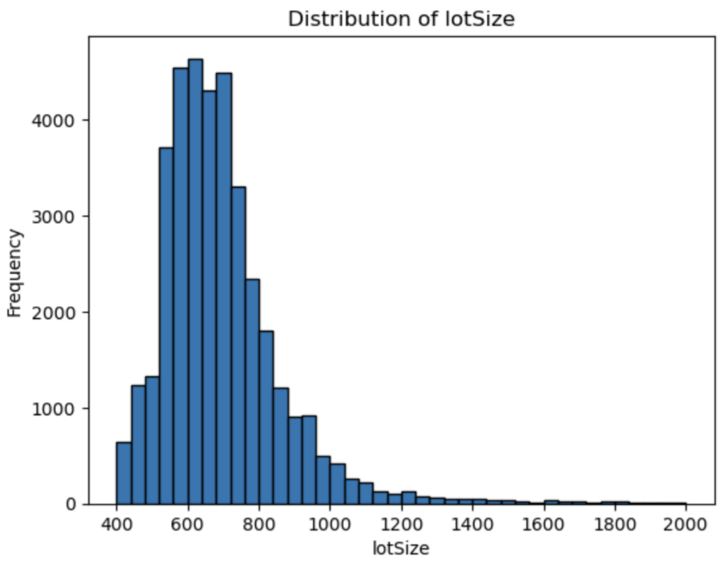


Figure2.3.4: Lot Size

The vast majority of properties in Sydney are between 500 and 1000 square meters. What big houses they are! The largest number of properties are concentrated between 550 square meters and 750 square meters. When calculating the price of a property later, the area is definitely an important point: because we may need to calculate the price of the unit area.

Next, let's focus on when these properties were sold.

```
[19]: # plot the distribution of saleQtr
      plt.hist(sydney.saleQtr, bins=20, edgecolor='black')
      plt.title("Distribution of saleQtr")
      plt.xlabel("saleQtr")
      plt.ylabel("Frequency")
      plt.show()
```

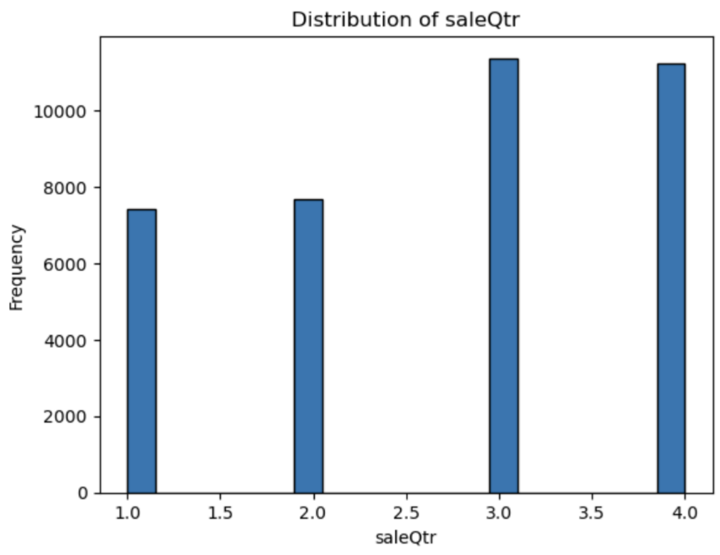


Figure2.3.5: Sale Quarter

The highest number of homes sold in the third quarter, and the least in the first quarter.

Finally, we draw a graph with the distribution of all variables.

```
[16]: # a plot of all variables
      sydney.hist(figsize=(20, 20), edgecolor='black')
      plt.show()
```

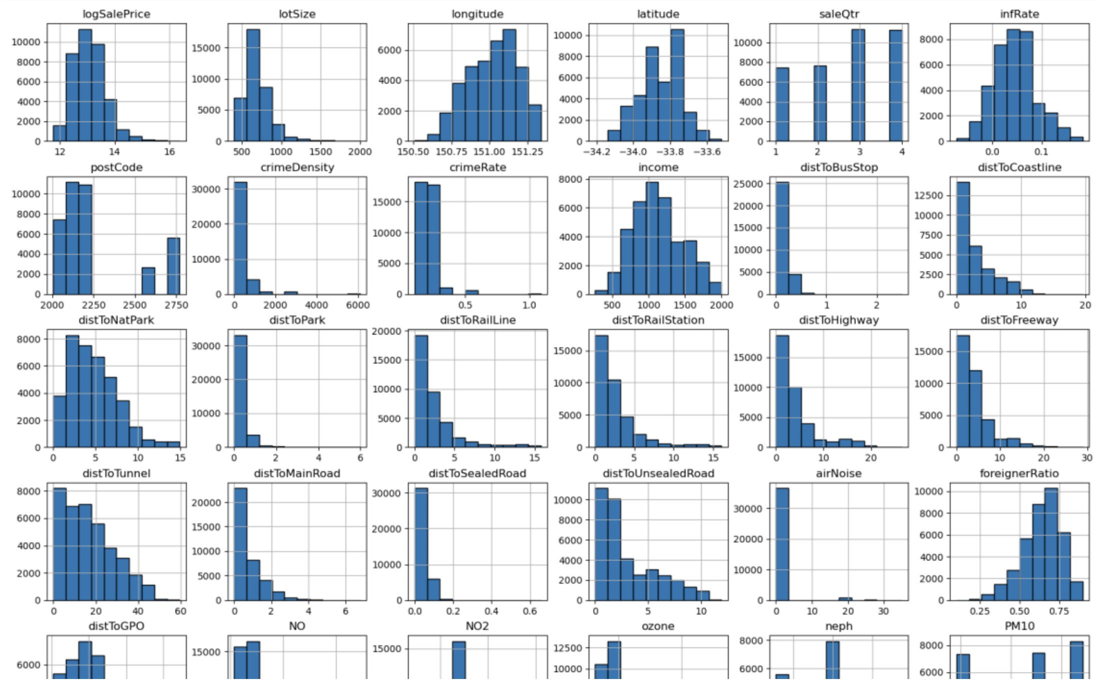
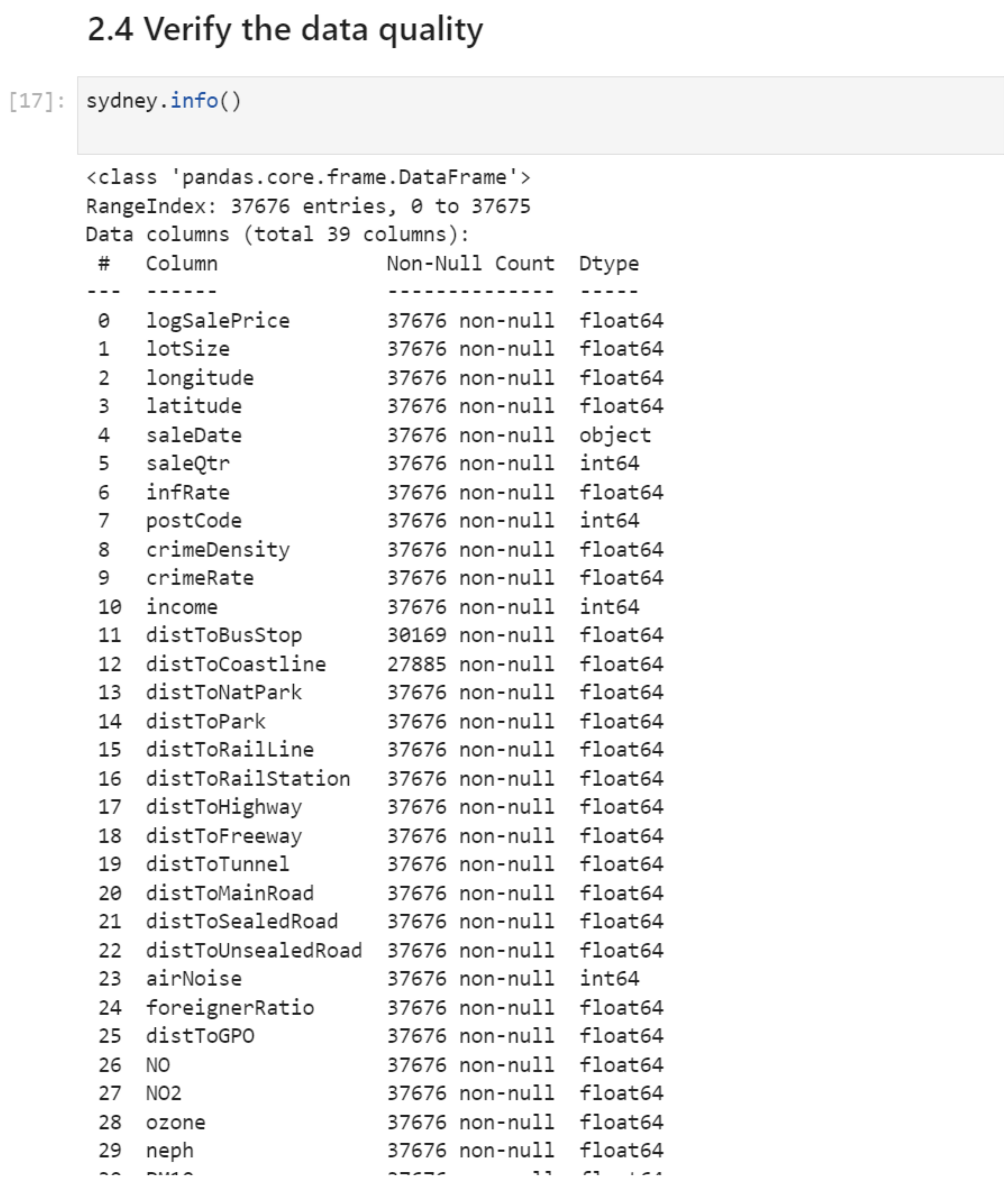


Figure2.3.6: Data Distribution

2.4 Verify the data quality

Here I used the info() method to check the data type.



2.4 Verify the data quality

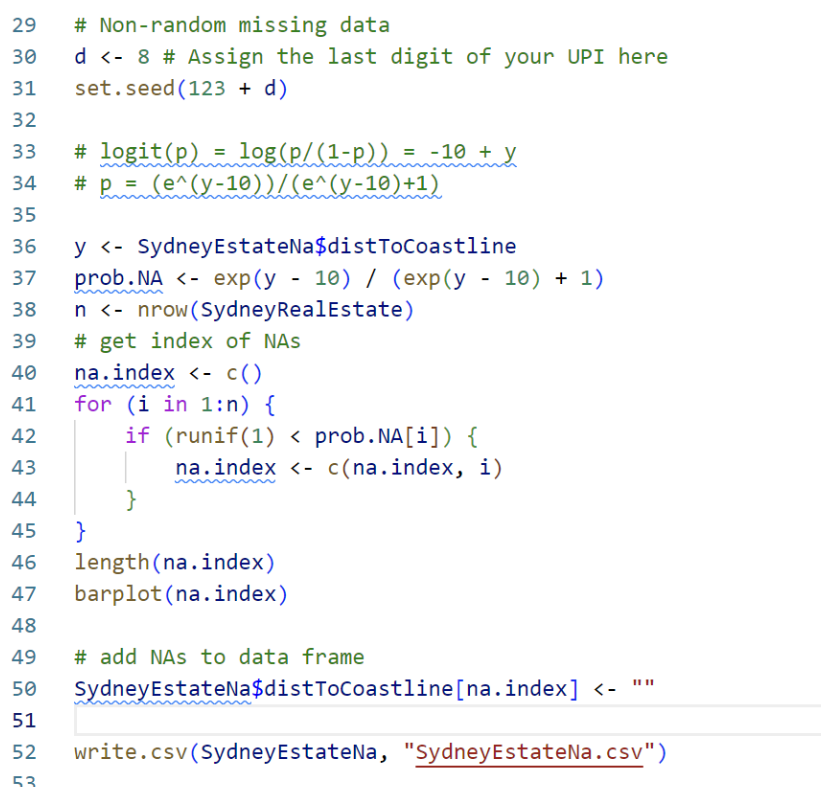
```
[17]: sydney.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 37676 entries, 0 to 37675
Data columns (total 39 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   logSalePrice       37676 non-null  float64
 1   lotSize            37676 non-null  float64
 2   longitude          37676 non-null  float64
 3   latitude           37676 non-null  float64
 4   saleDate           37676 non-null  object
 5   saleQtr            37676 non-null  int64
 6   infRate            37676 non-null  float64
 7   postCode           37676 non-null  int64
 8   crimeDensity       37676 non-null  float64
 9   crimeRate          37676 non-null  float64
 10  income             37676 non-null  int64
 11  distToBusStop      30169 non-null  float64
 12  distToCoastline    27885 non-null  float64
 13  distToNatPark      37676 non-null  float64
 14  distToPark         37676 non-null  float64
 15  distToRailLine     37676 non-null  float64
 16  distToRailStation  37676 non-null  float64
 17  distToHighway      37676 non-null  float64
 18  distToFreeway      37676 non-null  float64
 19  distToTunnel       37676 non-null  float64
 20  distToMainRoad     37676 non-null  float64
 21  distToSealedRoad   37676 non-null  float64
 22  distToUnsealedRoad 37676 non-null  float64
 23  airNoise           37676 non-null  int64
 24  foreignerRatio     37676 non-null  float64
 25  distToGPO          37676 non-null  float64
 26  NO                 37676 non-null  float64
 27  NO2                37676 non-null  float64
 28  ozone              37676 non-null  float64
 29  neph               37676 non-null  float64
```

Figure2.4.1: Data Quality

Almost all variables are considered as numeric variables, because the variables in the dataset are all described numerically. There are some that should be considered categorical, though, such as saleQtr: the quarter of the property sold, as it has only four possible unique values.

The dataset was a "clean" dataset. Two of the explanatory variables have missing values, which I added earlier with R.

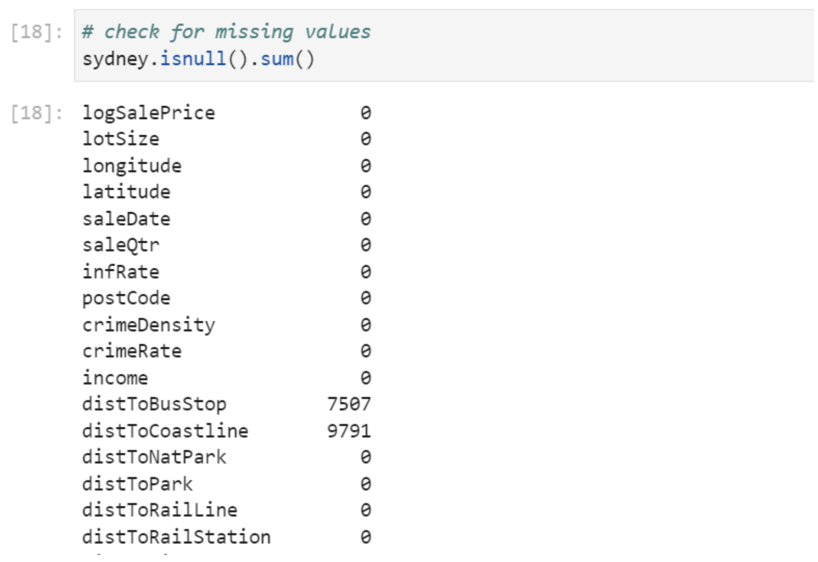


```r
# Non-random missing data
d <- 8 # Assign the last digit of your UPI here
set.seed(123 + d)

# logit(p) = log(p/(1-p)) = -10 + y
# p = (e^(y-10))/(e^(y-10)+1)

y <- SydneyEstateNa$distToCoastline
prob.NA <- exp(y - 10) / (exp(y - 10) + 1)
n <- nrow(SydneyRealEstate)
# get index of NAs
na.index <- c()
for (i in 1:n) {
    if (runif(1) < prob.NA[i]) {
        na.index <- c(na.index, i)
    }
}
length(na.index)
barplot(na.index)

# add NAs to data frame
SydneyEstateNa$distToCoastline[na.index] <- ""

write.csv(SydneyEstateNa, "SydneyEstateNa.csv")
```

Figure2.4.2: Some R code to Add Missing Values

There are two variables with incomplete data (Null Value) that we need to deal with. For the distToBusStop variable, I randomly lost 20% of the data; for distToCoastline, I used the loss probability of logit(p) = log(p/(1-p)) = -10 + y, where y refers to the value of distToCoastline value. So distToBusStop is Missing At Random (MAR) , while distToCoastline is Missing Not At Random (MNAR).

Checking variables with missing values in python:



```
[18]: # check for missing values
      sydney.isnull().sum()

[18]: logSalePrice          0
      lotSize               0
      longitude             0
      latitude              0
      saleDate              0
      saleQtr               0
      infRate               0
      postCode              0
      crimeDensity          0
      crimeRate             0
      income                0
      distToBusStop      7507
      distToCoastline    9791
      distToNatPark         0
      distToPark            0
      distToRailLine        0
      distToRailStation     0
```

Figure2.4.3: Missing Values

As expected, only two variables, distToBusStop and distToCoastline, have missing values.

In addition, there are some outliers and extreme values detected in the data. In later analysis, I will deduce whether they are really outliers or extreme values.

```
[26]: # plot outliers in logSalePrice
      plt.boxplot(sydney.logSalePrice)
```

```
[26]: {'whiskers': [<matplotlib.lines.Line2D at 0x1a82557c0a0>,
        <matplotlib.lines.Line2D at 0x1a82557f4c0>],
       'caps': [<matplotlib.lines.Line2D at 0x1a8254a3c10>,
        <matplotlib.lines.Line2D at 0x1a8254a3ee0>],
       'boxes': [<matplotlib.lines.Line2D at 0x1a82557d390>],
       'medians': [<matplotlib.lines.Line2D at 0x1a8254b41f0>],
       'fliers': [<matplotlib.lines.Line2D at 0x1a8254b44c0>],
       'means': []}
```

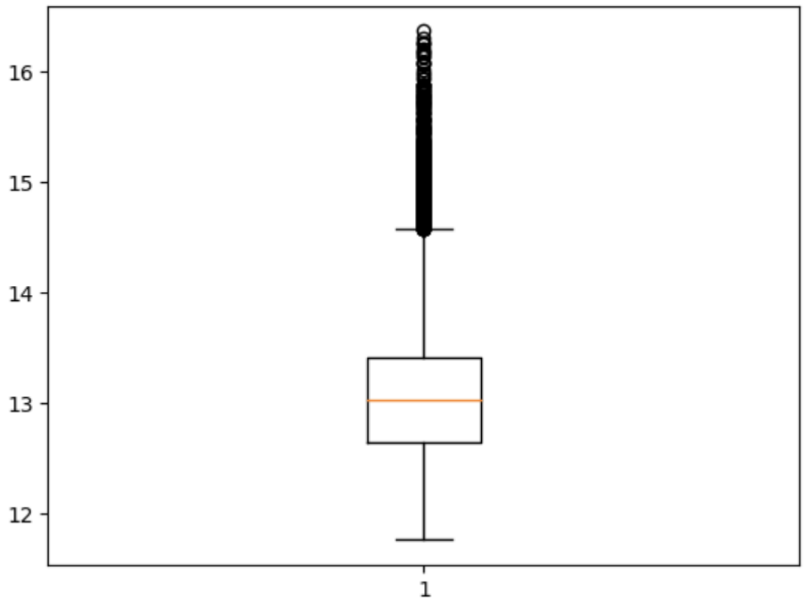
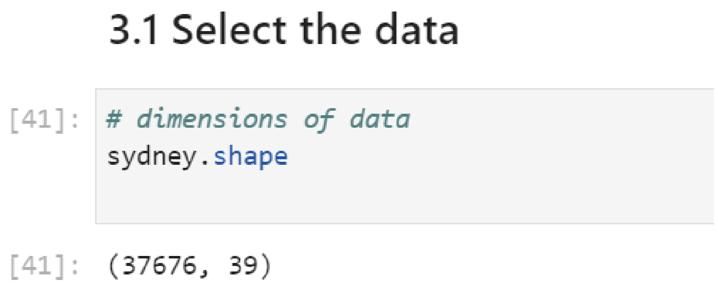


Figure2.4.4: Potential Outliers

3. Data preparation

3.1 Select the data

The current data has 37676 rows and 39 columns. Each row is a record of a property transaction, and each column represents a factor that may affect the price of a property. Having 39 columns means we have 38 explanatory variables and one response variable.



3.1 Select the data

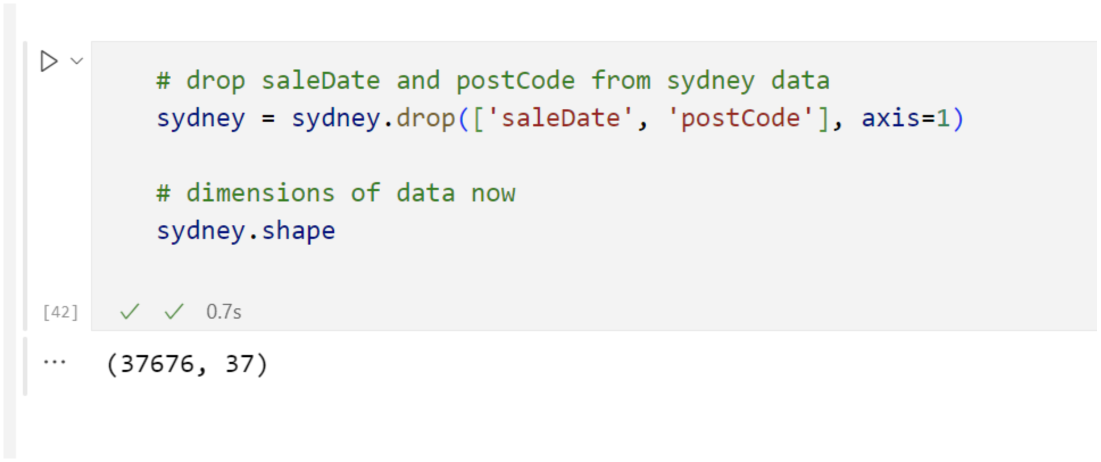
```
[41]: # dimensions of data
      sydney.shape

[41]: (37676, 39)
```

Figure3.1.1: Dimensions of Data

Select rows. Each row of data contains a real estate transaction information. Our study is about all Sydney property transactions, and more data leads to a better fit, so I decided to keep data for all rows.

Select columns. All the columns here are potentially valuable variables that may be helpful in predicting real estate prices. Since the data does not contain sensitive customer information, all columns can be preserved here.

In addition, there are some variables that are more suitable to be treated as categorical variables, such as zip code and transaction date. However, when used as categorical variables, they will have too many levels, which is not conducive to us building a stable model, and will also bring trouble to interpret the statistical significance of the model, so I choose to discard them.



```
# drop saleDate and postCode from sydney data
sydney = sydney.drop(['saleDate', 'postCode'], axis=1)

# dimensions of data now
sydney.shape
```

[42] ✓ 0.7s

```
(37676, 37)
```

Figure3.1.2: Column Filter

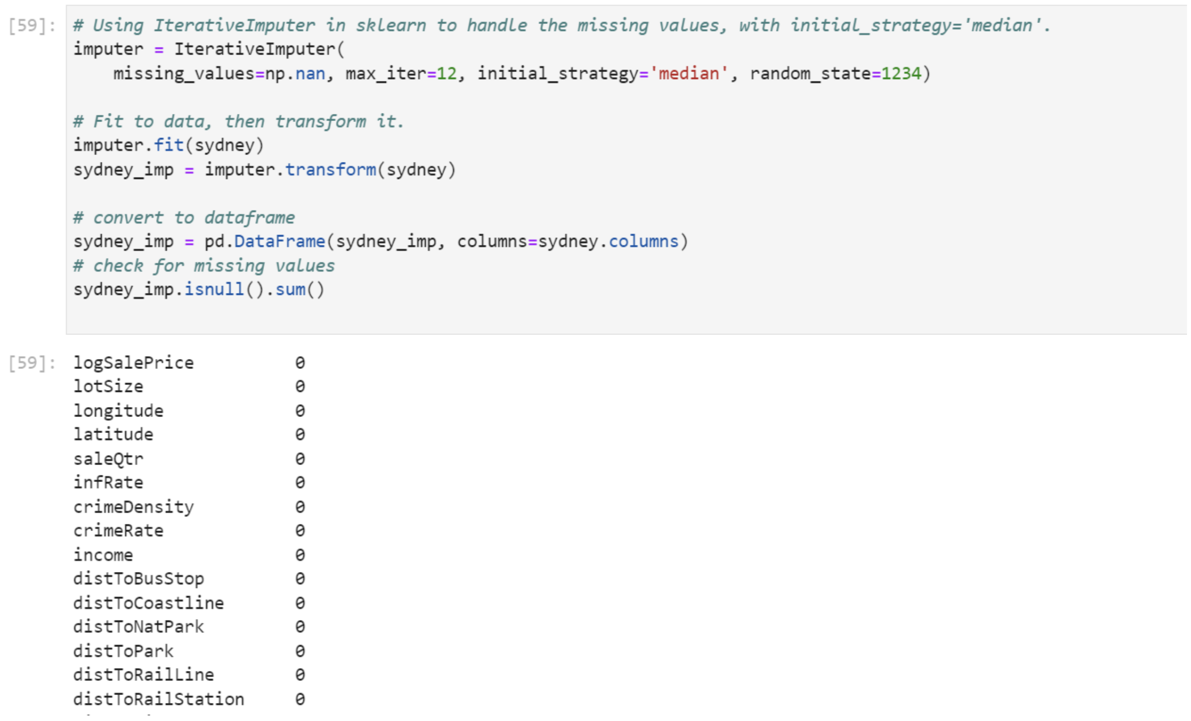
3.2 Clean the data

As previously described, only two columns in the dataset contain missing data. Unfortunately, based on my intuition, they are in two columns that are important for predicting property prices, and simply dropping those two columns is not a good idea.

We could delete all rows with data, but I'd like to try to avoid doing that. Doing so will lose a lot of data, potentially making our model less robust; one of the columns is missing non-randomly (MNAR), and deleting the corresponding row may lead to bias in our final model predictions.[5]

Now, we need to apply an appropriate imputation method for missing values. Using a measure of central tendency for the attribute seems to be a good way, but whether it is mean, median or most frequent has certain limitations. Here, I used IterativeImputer in sklearn to deal with missing values. It estimates each feature from all the others. In the first iteration, I chose the “median” strategy.

Run checking again, all missing values have been filled.



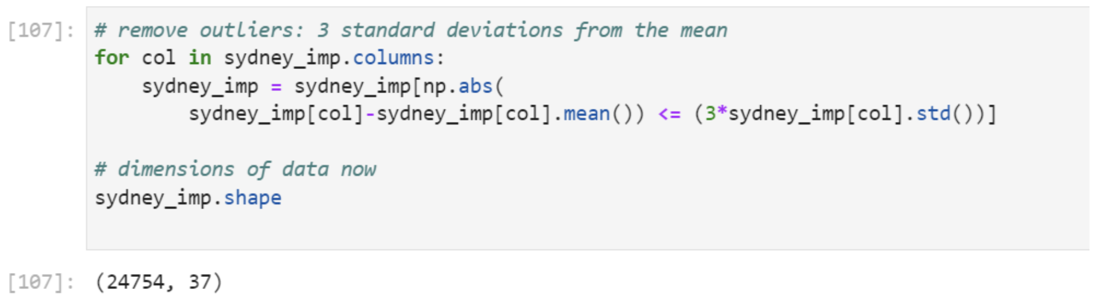
```
[59]: # Using IterativeImputer in sklearn to handle the missing values, with initial_strategy='median'.
      imputer = IterativeImputer(
          missing_values=np.nan, max_iter=12, initial_strategy='median', random_state=1234)

      # Fit to data, then transform it.
      imputer.fit(sydney)
      sydney_imp = imputer.transform(sydney)

      # convert to dataframe
      sydney_imp = pd.DataFrame(sydney_imp, columns=sydney.columns)
      # check for missing values
      sydney_imp.isnull().sum()
```

```
[59]: logSalePrice         0
      lotSize              0
      longitude            0
      latitude             0
      saleQtr              0
      infRate              0
      crimeDensity         0
      crimeRate            0
      income               0
      distToBusStop        0
      distToCoastline      0
      distToNatPark        0
      distToPark           0
      distToRailLine       0
      distToRailStation    0
```

Figure3.2.1: Data Imputation

To remove outliers and outliers, we only kept data within plus or minus 3 standard deviations of the mean. Within the normal distribution assumptions, this would contain 99.73% of the data. Any data outside this is likely to be outliers or extreme values.

After removing possible outliers for all variables, the number of rows of the data is reduced.

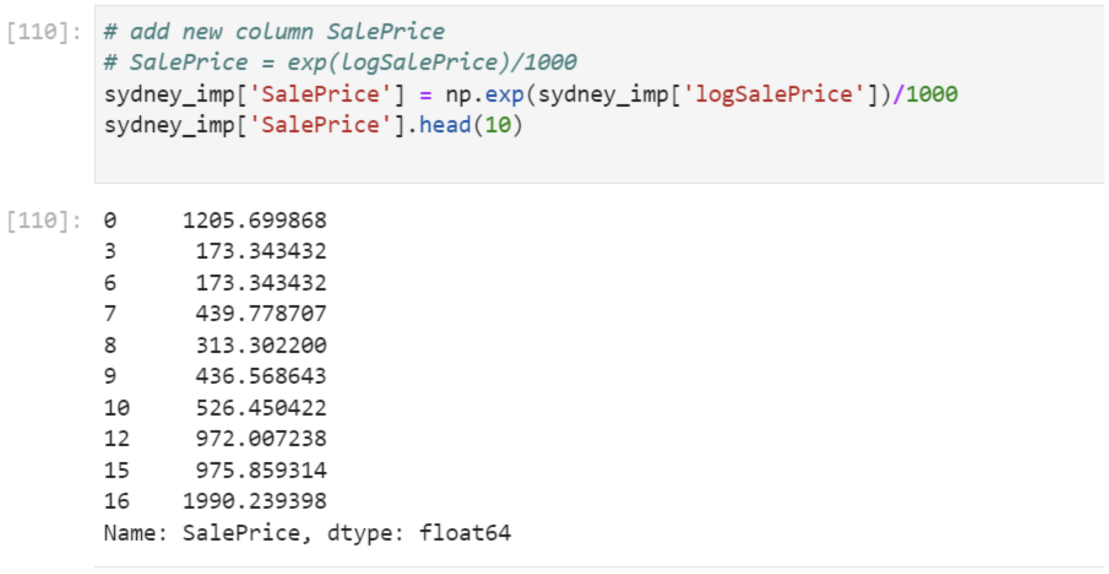


```
[107]: # remove outliers: 3 standard deviations from the mean
       for col in sydney_imp.columns:
           sydney_imp = sydney_imp[np.abs(
               sydney_imp[col]-sydney_imp[col].mean()) <= (3*sydney_imp[col].std())]

       # dimensions of data now
       sydney_imp.shape

[107]: (24754, 37)
```

Figure3.2.2: Outlier Removal

3.3 Construct the data

The real estate prices in the original data are on the log-scale. This can be useful when modeling, but is less intuitive when interpreting. So, I'm adding a new variable here: SalePrice(thousands of AUD). This variable is obtained by taking the exponent of "logSalePrice" and dividing it by 1000.



```
[110]: # add new column SalePrice
       # SalePrice = exp(logSalePrice)/1000
       sydney_imp['SalePrice'] = np.exp(sydney_imp['logSalePrice'])/1000
       sydney_imp['SalePrice'].head(10)

[110]: 0     1205.699868
       3      173.343432
       6      173.343432
       7      439.778707
       8      313.302200
       9      436.568643
       10     526.450422
       12     972.007238
       15     975.859314
       16    1990.239398
       Name: SalePrice, dtype: float64
```

Figure3.3.1: Generate New Column

It can be seen from the plot that the restored real estate prices show an exponential relation to prices in log-scale.

```
[114]: # plot SalePrice vs logSalePrice
       plt.scatter(sydney_imp['logSalePrice'], sydney_imp['SalePrice'])
       plt.title("SalePrice vs logSalePrice")
       plt.xlabel("logSalePrice")
       plt.ylabel("SalePrice(in $1000)")
       plt.show()
```

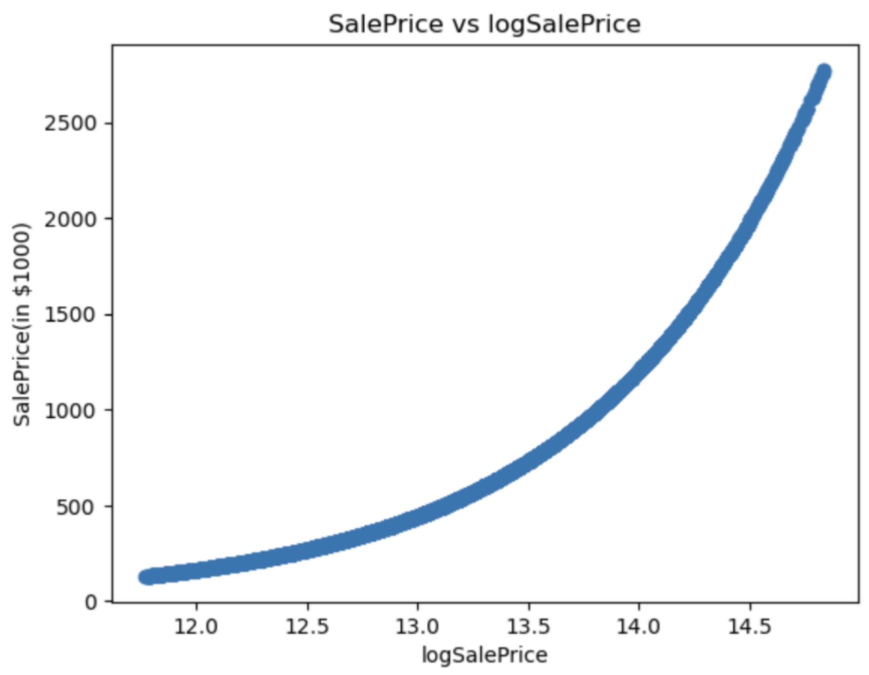
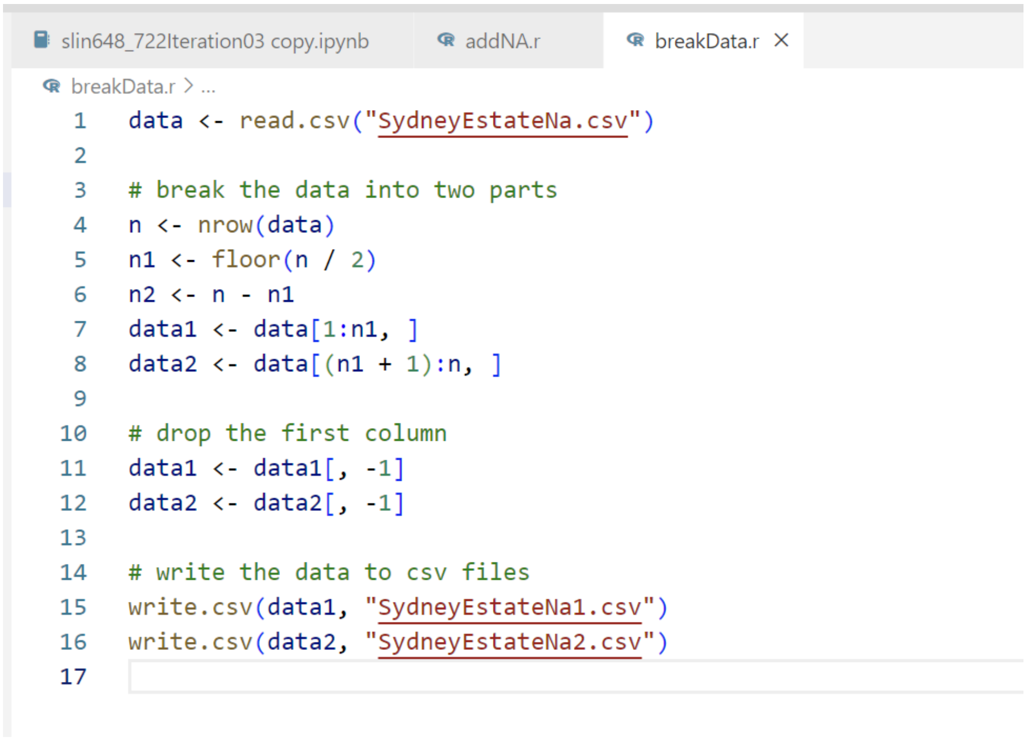


Figure3.3.2: Restored Prices

3.4 Integrate various data sources

Our data comes from a single CSV file, and the newly added variables have already been integrated into the current data frame, so here I have to break the data into halves and merge it back.

I used the R code below to split the csv file into two halves:

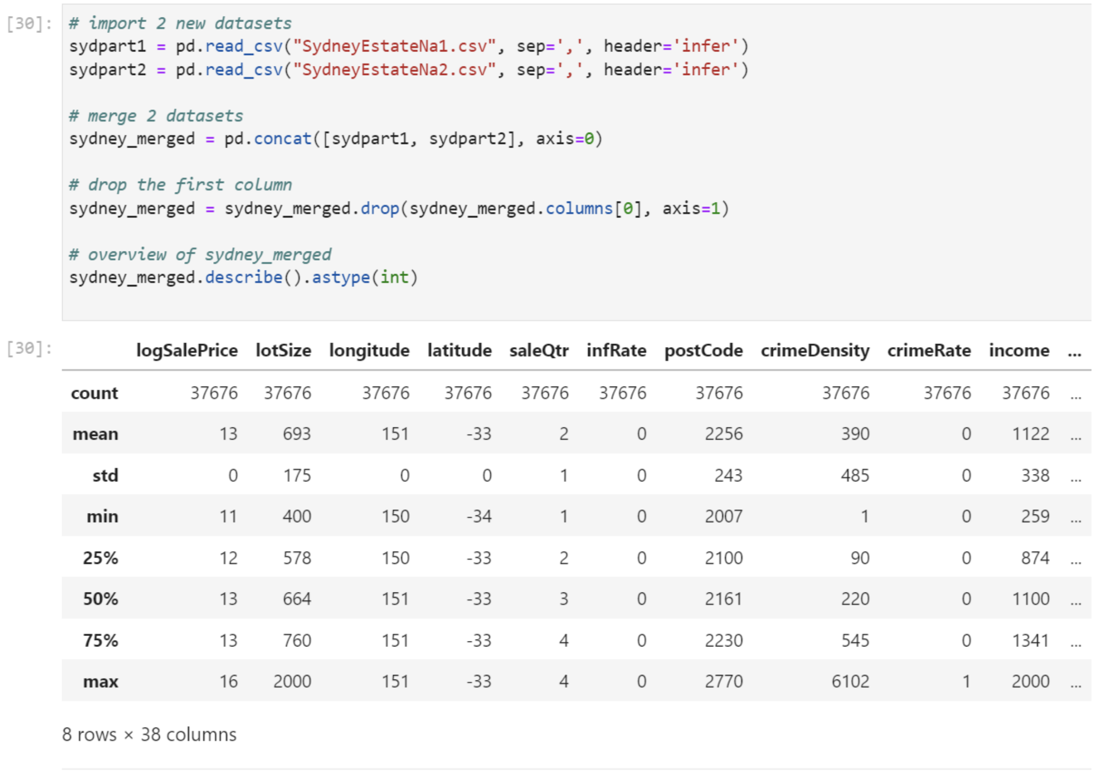


```r
data <- read.csv("SydneyEstateNa.csv")

# break the data into two parts
n <- nrow(data)
n1 <- floor(n / 2)
n2 <- n - n1
data1 <- data[1:n1, ]
data2 <- data[(n1 + 1):n, ]

# drop the first column
data1 <- data1[, -1]
data2 <- data2[, -1]

# write the data to csv files
write.csv(data1, "SydneyEstateNa1.csv")
write.csv(data2, "SydneyEstateNa2.csv")
```

Figure3.4.1: Breaking Data

The merged dataset is the same as the original dataset.

3.4 Integrate various data sources



```python
[30]: # import 2 new datasets
sydpart1 = pd.read_csv("SydneyEstateNa1.csv", sep=',', header='infer')
sydpart2 = pd.read_csv("SydneyEstateNa2.csv", sep=',', header='infer')

# merge 2 datasets
sydney_merged = pd.concat([sydpart1, sydpart2], axis=0)

# drop the first column
sydney_merged = sydney_merged.drop(sydney_merged.columns[0], axis=1)

# overview of sydney_merged
sydney_merged.describe().astype(int)
```

[30]:

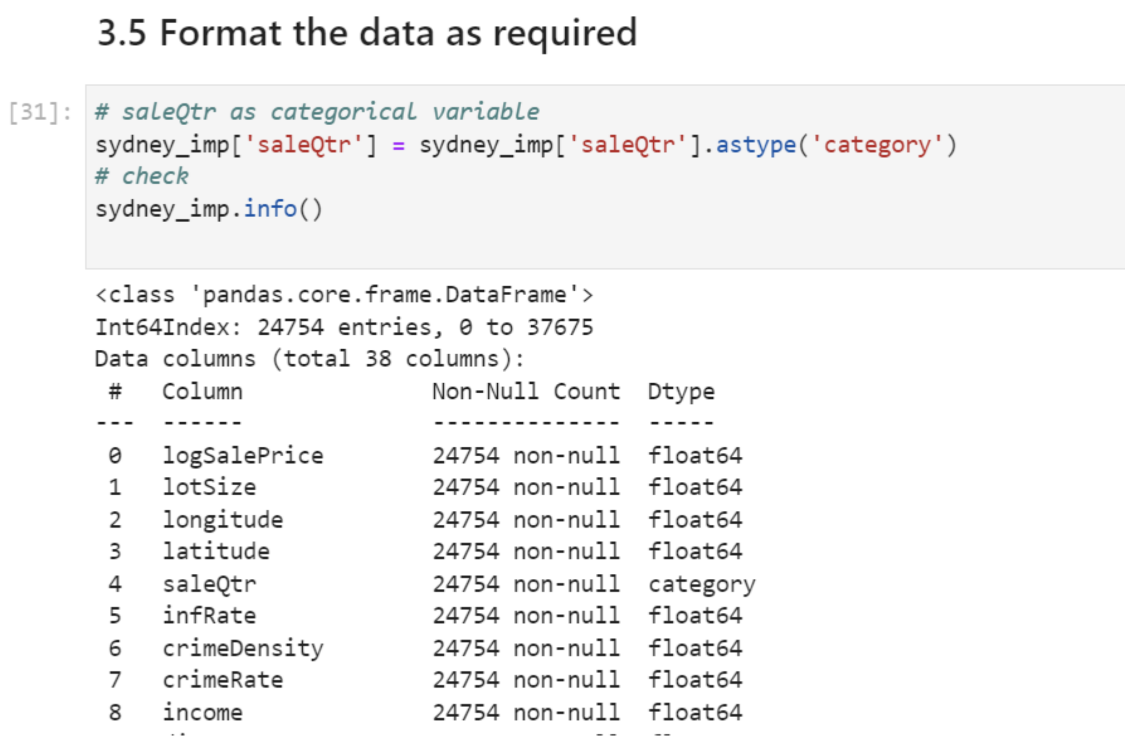
	logSalePrice	lotSize	longitude	latitude	saleQtr	infRate	postCode	crimeDensity	crimeRate	income	...
count	37676	37676	37676	37676	37676	37676	37676	37676	37676	37676	...
mean	13	693	151	-33	2	0	2256	390	0	1122	...
std	0	175	0	0	1	0	243	485	0	338	...
min	11	400	150	-34	1	0	2007	1	0	259	...
25%	12	578	150	-33	2	0	2100	90	0	874	...
50%	13	664	151	-33	3	0	2161	220	0	1100	...
75%	13	760	151	-33	4	0	2230	545	0	1341	...
max	16	2000	151	-33	4	0	2770	6102	1	2000	...

8 rows × 38 columns

Figure3.4.2: Merged Dataset

3.5 Format the data as required

Financial quarter is more suitable to be considered a categorical variable because it is discrete and has only 4 levels.



3.5 Format the data as required

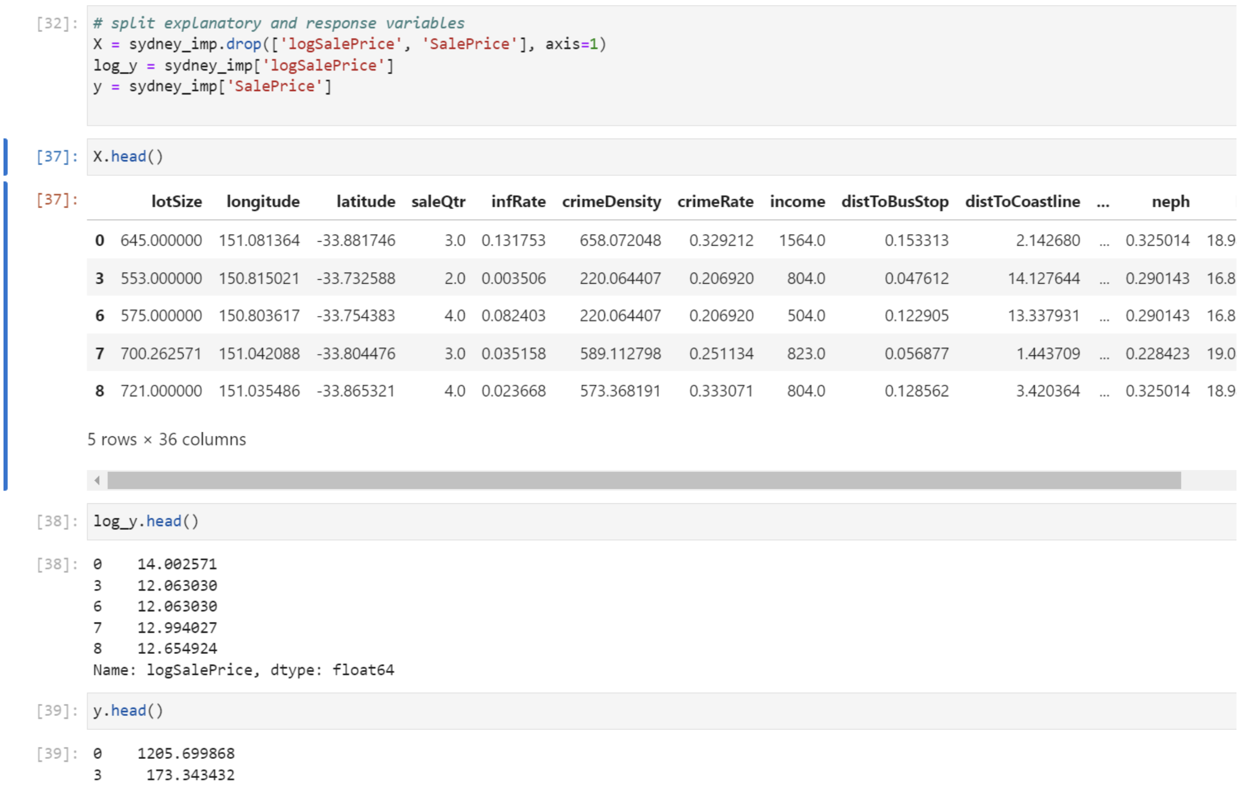
```
[31]: # saleQtr as categorical variable
      sydney_imp['saleQtr'] = sydney_imp['saleQtr'].astype('category')
      # check
      sydney_imp.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 24754 entries, 0 to 37675
Data columns (total 38 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   logSalePrice        24754 non-null  float64
 1   lotSize             24754 non-null  float64
 2   longitude           24754 non-null  float64
 3   latitude            24754 non-null  float64
 4   saleQtr             24754 non-null  category
 5   infRate             24754 non-null  float64
 6   crimeDensity        24754 non-null  float64
 7   crimeRate           24754 non-null  float64
 8   income              24754 non-null  float64
```

Figure3.5.1: saleQtr as Categorical

In addition, input and target should also be specified. input refers to the explanatory variable and target refers to the response variable.

Variables except logSalePrice and SalePrice are explanatory variables, logSalePrice and SalePrice are response variables.



```
[32]: # split explanatory and response variables
      X = sydney_imp.drop(['logSalePrice', 'SalePrice'], axis=1)
      log_y = sydney_imp['logSalePrice']
      y = sydney_imp['SalePrice']
```

```
[37]: X.head()
```

[37]:

	lotSize	longitude	latitude	saleQtr	infRate	crimeDensity	crimeRate	income	distToBusStop	distToCoastline	...	neph	
0	645.000000	151.081364	-33.881746	3.0	0.131753	658.072048	0.329212	1564.0	0.153313	2.142680	...	0.325014	18.9
3	553.000000	150.815021	-33.732588	2.0	0.003506	220.064407	0.206920	804.0	0.047612	14.127644	...	0.290143	16.8
6	575.000000	150.803617	-33.754383	4.0	0.082403	220.064407	0.206920	504.0	0.122905	13.337931	...	0.290143	16.8
7	700.262571	151.042088	-33.804476	3.0	0.035158	589.112798	0.251134	823.0	0.056877	1.443709	...	0.228423	19.0
8	721.000000	151.035486	-33.865321	4.0	0.023668	573.368191	0.333071	804.0	0.128562	3.420364	...	0.325014	18.9

5 rows × 36 columns

```
[38]: log_y.head()

[38]: 0    14.002571
      3    12.063030
      6    12.063030
      7    12.994027
      8    12.654924
      Name: logSalePrice, dtype: float64

[39]: y.head()

[39]: 0    1205.699868
      3     173.343432
```

Figure3.5.2: Explanatory and Response Variables

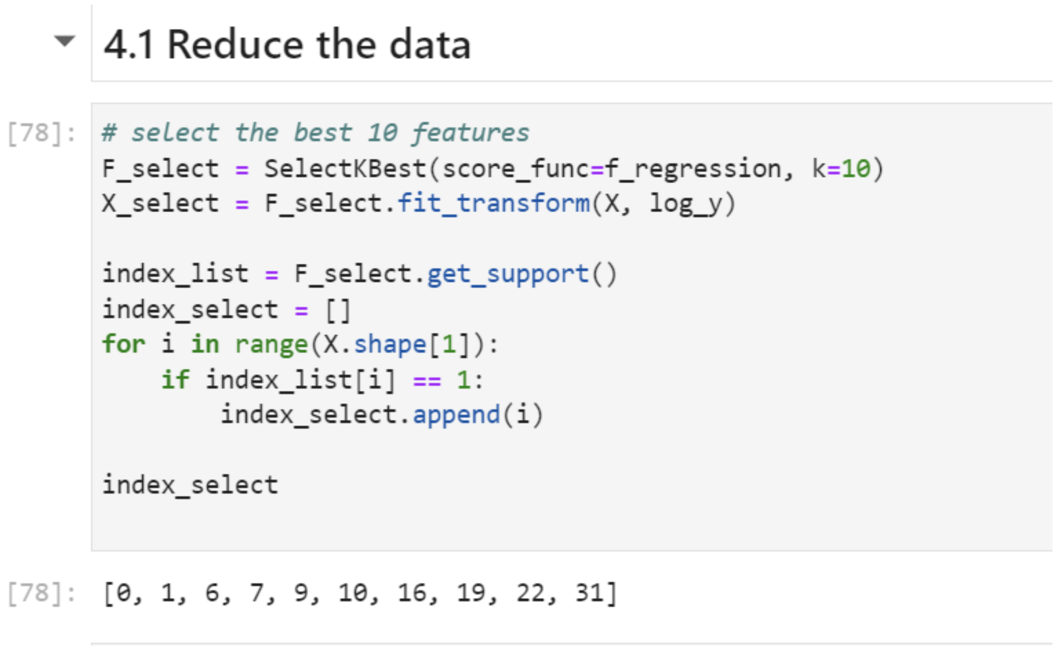
4. Data transformation

4.1 Reduce the data

Feature selection helps identify the most important fields when modeling and predicting outcomes. This is useful when there are many explanatory variables, because not every explanatory variable will be helpful for modeling and prediction, and useless explanatory variables will increase the computational cost and affect the stability of the model.

Here, I used Feature selection in sklearn. According to the method mentioned in Univariate feature selection, SelectKBest removes all features except the K features with the highest score. Since here it is a regression problem, we can use f_regression for Feature selection, which is a Feature selection method based on ANOVA F-value. This method calculates the ANOVA F-value of each feature. The larger the F-value, the greater the relationship between the feature and the prediction, so the feature is more "important".

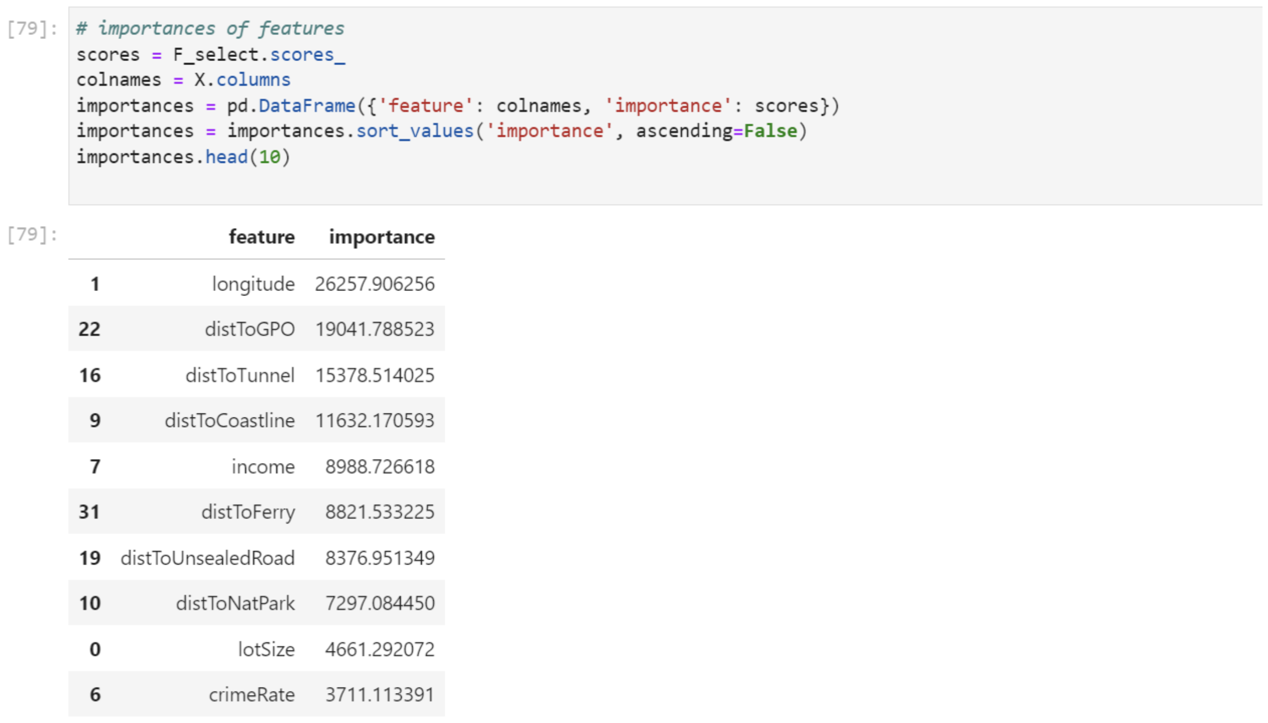
Here, I keep the top 10 important variables.


4.1 Reduce the data

```
[78]: # select the best 10 features
      F_select = SelectKBest(score_func=f_regression, k=10)
      X_select = F_select.fit_transform(X, log_y)

      index_list = F_select.get_support()
      index_select = []
      for i in range(X.shape[1]):
          if index_list[i] == 1:
              index_select.append(i)

      index_select

[78]: [0, 1, 6, 7, 9, 10, 16, 19, 22, 31]
```

Figure4.1.1: Feature Selection Index

Variables with index within [0, 1, 6, 7, 9, 10, 16, 19, 22, 31] are reserved. In order to more intuitively explain what these variables are, here is a table of variable importance in descending order:



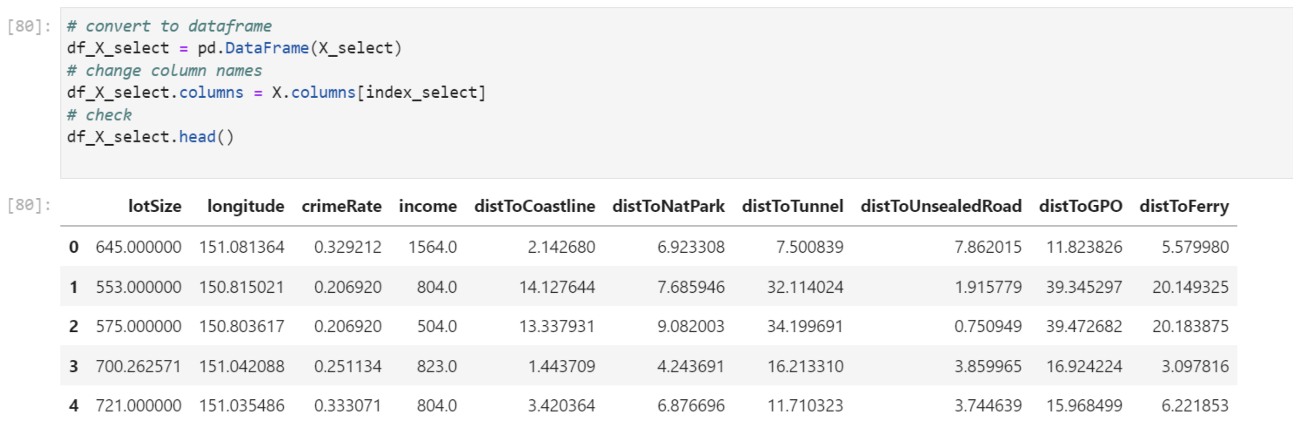
```
[79]: # importances of features
      scores = F_select.scores_
      colnames = X.columns
      importances = pd.DataFrame({'feature': colnames, 'importance': scores})
      importances = importances.sort_values('importance', ascending=False)
      importances.head(10)
```

[79]:

	feature	importance
1	longitude	26257.906256
22	distToGPO	19041.788523
16	distToTunnel	15378.514025
9	distToCoastline	11632.170593
7	income	8988.726618
31	distToFerry	8821.533225
19	distToUnsealedRoad	8376.951349
10	distToNatPark	7297.084450
0	lotSize	4661.292072
6	crimeRate	3711.113391

Figure4.1.2: Variable Importance Table

The dataset after feature selection is called df_X_select.



```
[80]: # convert to dataframe
      df_X_select = pd.DataFrame(X_select)
      # change column names
      df_X_select.columns = X.columns[index_select]
      # check
      df_X_select.head()
```

[80]:

	lotSize	longitude	crimeRate	income	distToCoastline	distToNatPark	distToTunnel	distToUnsealedRoad	distToGPO	distToFerry
0	645.000000	151.081364	0.329212	1564.0	2.142680	6.923308	7.500839	7.862015	11.823826	5.579980
1	553.000000	150.815021	0.206920	804.0	14.127644	7.685946	32.114024	1.915779	39.345297	20.149325
2	575.000000	150.803617	0.206920	504.0	13.337931	9.082003	34.199691	0.750949	39.472682	20.183875
3	700.262571	151.042088	0.251134	823.0	1.443709	4.243691	16.213310	3.859965	16.924224	3.097816
4	721.000000	151.035486	0.333071	804.0	3.420364	6.876696	11.710323	3.744639	15.968499	6.221853

Figure4.1.3: Dataset after Feature Selection

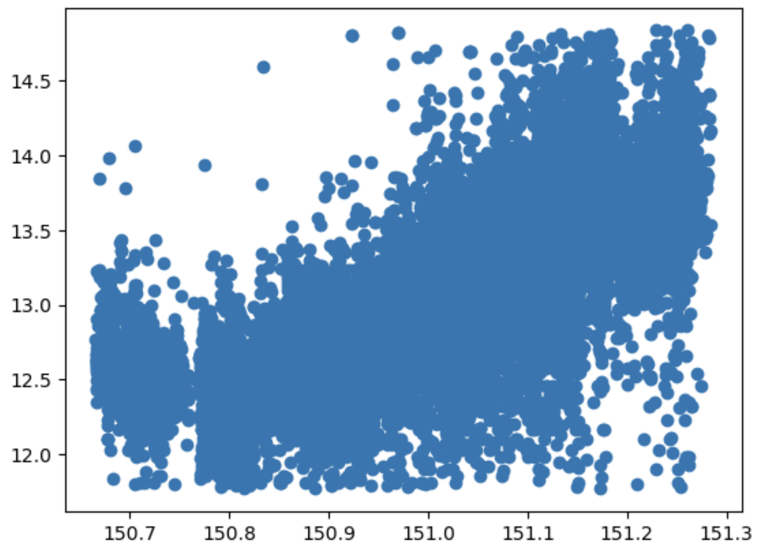
4.2 Project the data

The relationship between property prices and most of the explanatory variables is the inverse of the log (and its multiples). Also, when plotting against log-scaled price, we find that the data has a near-constant variance, whereas when plotting against price, the variance varies with level. Many statistical models require the assumption of constant variance, so it would be a better idea to use log-scale explanatory variables.[6]

4.2 Project the data

```
[81]: # plot longitude vs logSalePrice
      plt.scatter(df_X_select['longitude'], log_y)
```

```
[81]: <matplotlib.collections.PathCollection at 0x1f91c54fb50>
```



```
[82]: # plot longitude vs SalePrice
      plt.scatter(df_X_select['longitude'], y)
```

```
[82]: <matplotlib.collections.PathCollection at 0x1f91c73a1d0>
```

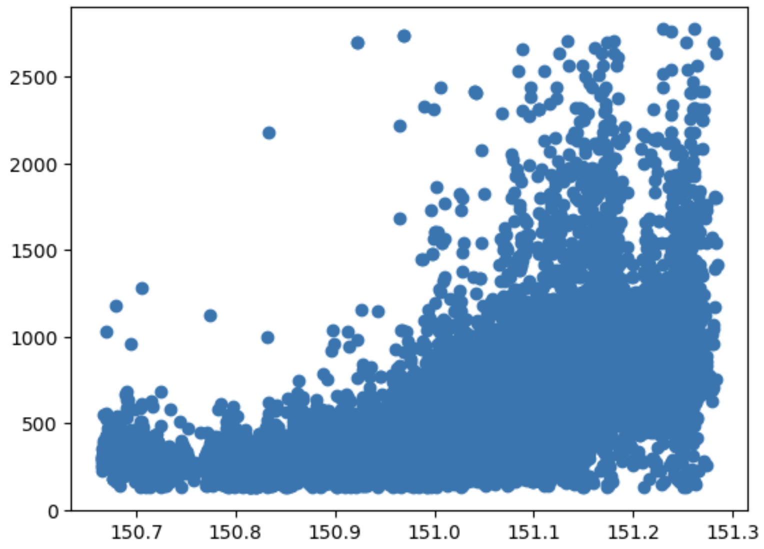


Figure4.2.1: Log-transformation

Log-transformed data generally better fit the model assumptions. Reciprocal-shaped distributions are difficult to interpret, whereas linear-like distributions are easier to fit and interpret.[6]

5. Data-mining algorithm(s) selection

5.1 Match and discuss the objectives of data mining to data mining methods

The goal of this research project is to study the factors that influence Sydney property prices and to predict property prices based on different explanatory variables in the dataset.

The data mining goals we focus on are:
Explanation: Identify important explanatory variables that affect property prices and explain the relationship between these variables and property prices.
Prediction: Predict the house price given the explanatory variables.

Desired Outcome: Get to know what factors affect house prices the most, and what's trivial, and make the model obtain a high prediction accuracy in the test set.

There are three main categories of data mining methods: **Classification, Clustering** and **Regression**.[7] we can implement all these algorithms in Python, and there are packages for these algorithms. The three types of data mining methods will be explained below and combined with our data mining goals to see why only regression is the appropriate data mining method for this project.

Classification is a supervised learning method. Classification models are suitable for datasets where the response variable is discrete.[7]

Clustering is an unsupervised learning method. Unsupervised learning is a training method of machine learning, which is essentially a statistical method, a training method in which some potential structures can be found in unlabeled data.[7] Clustering focuses on identifying similar data and labeling the data according to the group to which the data corresponds.

Regression is a common data analysis method. Regression models produce continuous response variables.[7] Common regression models include linear regression, generalized linear regression, random forest regression, and lasso regression.

5.2 Select the appropriate data-mining method(s) based on discussion

In this project, we will use a variety of models and data mining methods and come up with the most suitable model or models for prediction.

Based on the objectives of data mining,
1. For **classification**, the response variable we want to model and predict is house price (or

its logarithm), which is a continuous variable rather than a categorical variable, and if it were a categorical variable, there would be close to the number of rows of data levels, thus making it impossible to interpret the modeling results and make predictions. Therefore, **classification is not a method that meets our data mining goals.**

2. **Clustering** is an unsupervised learning method with no predefined outputs, is suitable when the original dataset does not contain a response variable. We were also unable to verify the predictive accuracy of the clustering, as there was no established criterion. This does not meet our data mining goals, so **clustering is not the data mining method we should choose here.**

3. For **regression**, the response variable here is house price (or its logarithm), which is a continuous variable, regression methods are suitable in this case. So, **I decided to use regression as the data mining method we will use.**

6. Data-mining algorithm(s) selection

6.1 Conduct exploratory analysis and discuss

As we discussed earlier, we will use regression as the data mining method. There are many data mining methods for regression, such as regression trees, random forests, Bayesian regression, etc.

Regression trees are basically decision trees for regression tasks that can be used to predict continuous valued outputs instead of discrete outputs.[12] The basic idea behind the algorithm is to find the point in the independent variable, split the dataset into 2 parts so that the mean squared error is minimum at that point. The algorithm does this in an iterative manner, forming a tree-like structure.[12]
Random forests grow many classification trees. To classify new objects in the input vector, place the input vector under each tree in the forest. Each tree is given a classification, and we say the tree "votes" for that class. The forest selects the class with the most votes (amongst all trees in the forest).[13]

There are two types of machine learning: supervised learning and unsupervised learning.
Supervised learning is a machine learning method defined by the use of labeled datasets. These datasets are designed to train or "supervised" algorithms to classify data or accurately predict outcomes. Using labeled inputs and outputs, the model can measure its accuracy and learn over time.[14]
Unsupervised learning uses machine learning algorithms to analyze and cluster unlabeled datasets. These algorithms discover hidden patterns in data without human intervention (thus, they are "unsupervised").[14]

The goal of this research is to study the factors that affect the price of real estate, build a model and use these factors to predict the price of real estate, so the input of the model is some explanatory variables, and the output is the house price. This is a scenario with a clear output target, so we should choose a supervised model.

6.2 Select data-mining algorithms based on discussion

Specifically, there are some models that are more traditional and straightforward, such as multivariate regression and linear models, which are helpful for our later explanations. There are models that use more modern statistical theory, which may perform better, but also incur greater computational overhead. Here, we will choose some models and proceed to the subsequent modeling steps.

In fact, all models available in Python are our potential models. This includes Regression, Generalized Linear, KNN, SVM, etc. We can fit all the models and compare their predictive performance on the input data, but fitting all the models is computationally expensive.

I decide to fit three models: **Linear model, Multivariate Regression model** and **Random Forest**.

Linear regression is the most traditional one of regression methods, which assumes a linear relationship between the explanatory variable and the response variable. Here, I will choose a single variable as the explanatory variable.

Multivariate Regression can tell the relationship between multiple explanatory variables and a single response variable. Here, I will put all explanatory variables (after the feature selection) into the model.

Random forest belongs to the Bagging algorithm. The general idea is to train multiple weak models and package them to form a strong model. The performance of the strong model is much better than that of a single weak model.[8] In the training phase, random forest uses bootstrap sampling to collect multiple different sub-training datasets from the input training dataset to train multiple different decision trees in turn; in the prediction phase, random forest averages the prediction results of multiple internal decision trees to obtain final result.[8]

6.3 Build/Select appropriate model(s) and choose relevant parameter(s)

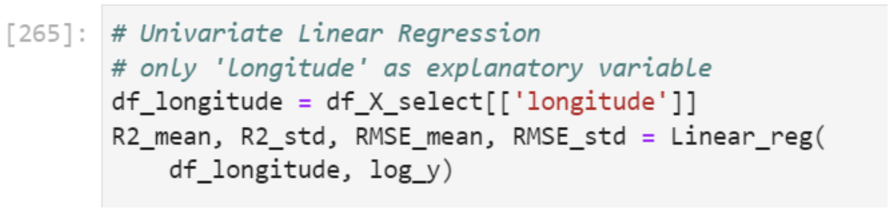
6.3.1 Linear regression model

For a linear model, the response variable is house price (in log-scale) and the explanatory variable will be one of the explanatory variables of the dataset.

In Section 4.1, we already know that the most important variable affecting house prices is longitude. Therefore, I will use longitude as the only explanatory variable in the linear regression model.

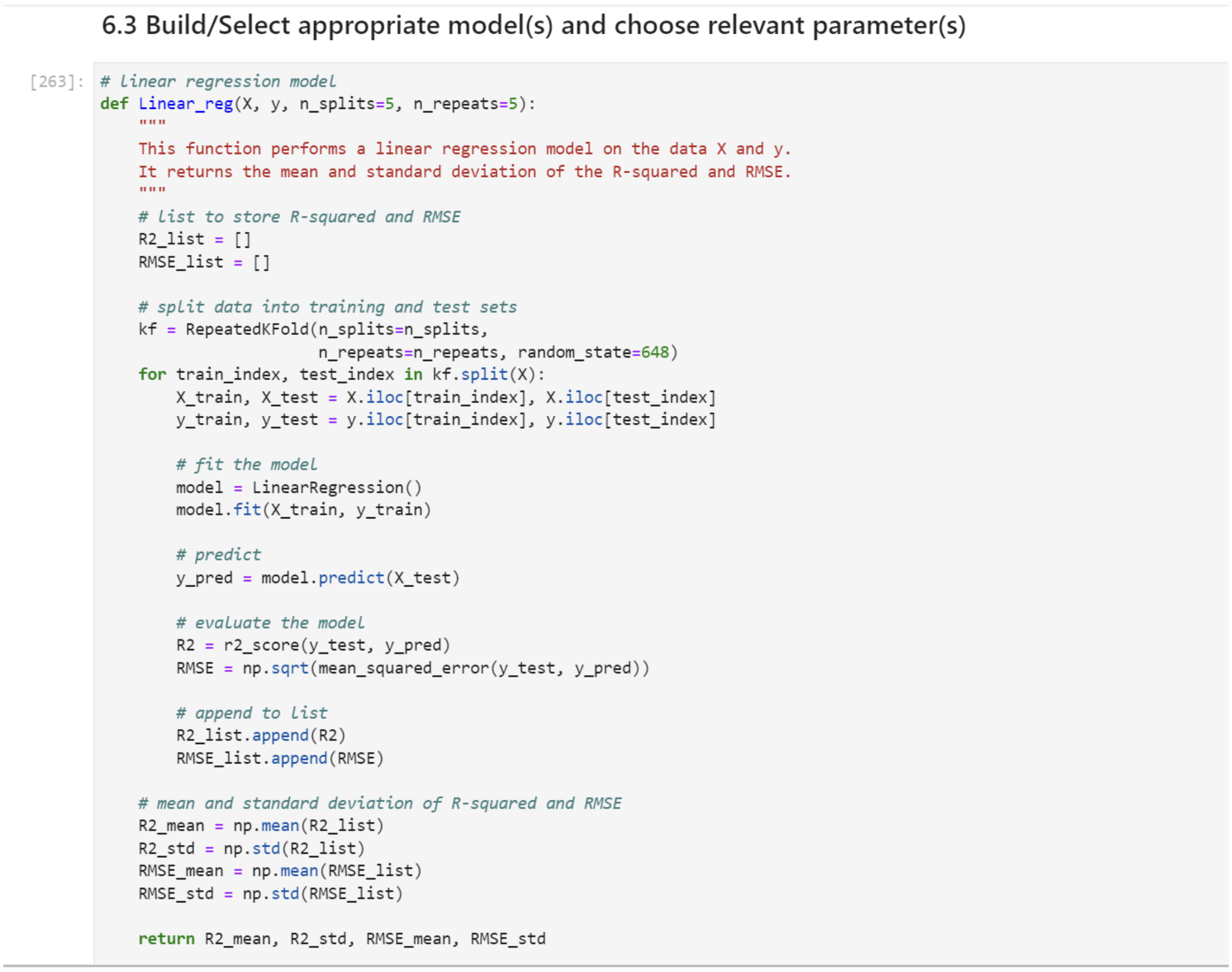
For linear regression, I used the LinearRegression method from sklearn.linear_model. It is a function in the sklearn package.

I wrote a function to do all the steps of the linear model. For the univariate linear model here, the parameter passed in is longitude, not the entire dataset.



```
[265]: # Univariate Linear Regression
       # only 'longitude' as explanatory variable
       df_longitude = df_X_select[['longitude']]
       R2_mean, R2_std, RMSE_mean, RMSE_std = Linear_reg(
           df_longitude, log_y)
```

Figure6.3.1: Linear Model Parameter Settings



6.3 Build/Select appropriate model(s) and choose relevant parameter(s)

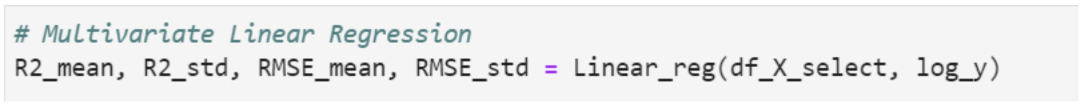
[263]:
```python
# linear regression model
def Linear_reg(X, y, n_splits=5, n_repeats=5):
    """
    This function performs a linear regression model on the data X and y.
    It returns the mean and standard deviation of the R-squared and RMSE.
    """
    # list to store R-squared and RMSE
    R2_list = []
    RMSE_list = []

    # split data into training and test sets
    kf = RepeatedKFold(n_splits=n_splits,
                       n_repeats=n_repeats, random_state=648)
    for train_index, test_index in kf.split(X):
        X_train, X_test = X.iloc[train_index], X.iloc[test_index]
        y_train, y_test = y.iloc[train_index], y.iloc[test_index]

        # fit the model
        model = LinearRegression()
        model.fit(X_train, y_train)

        # predict
        y_pred = model.predict(X_test)

        # evaluate the model
        R2 = r2_score(y_test, y_pred)
        RMSE = np.sqrt(mean_squared_error(y_test, y_pred))

        # append to list
        R2_list.append(R2)
        RMSE_list.append(RMSE)

    # mean and standard deviation of R-squared and RMSE
    R2_mean = np.mean(R2_list)
    R2_std = np.std(R2_list)
    RMSE_mean = np.mean(RMSE_list)
    RMSE_std = np.std(RMSE_list)

    return R2_mean, R2_std, RMSE_mean, RMSE_std
```

Figure6.3.2: Linear Model Function

6.3.2 Multivariate Regression model

For the Multivariate Regression model, I used all the explanatory variables previously selected as input, and the output (response) variable being house price (AUD) on log scale. This is obviously a more complex model, with a high probability that it will perform better than the linear model.

Compared with the previous univariate linear regression, the only difference is that the parameter X passed into the linear regression function is all explanatory variables here.

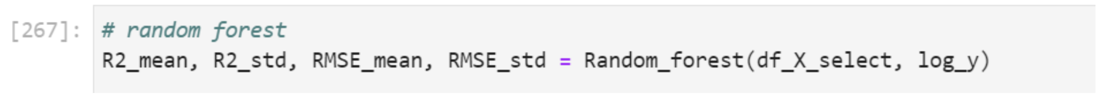


```python
# Multivariate Linear Regression
R2_mean, R2_std, RMSE_mean, RMSE_std = Linear_reg(df_X_select, log_y)
```

Figure6.3.3: Multivariate Regression model Parameter Settings

6.3.3 Random Forest model

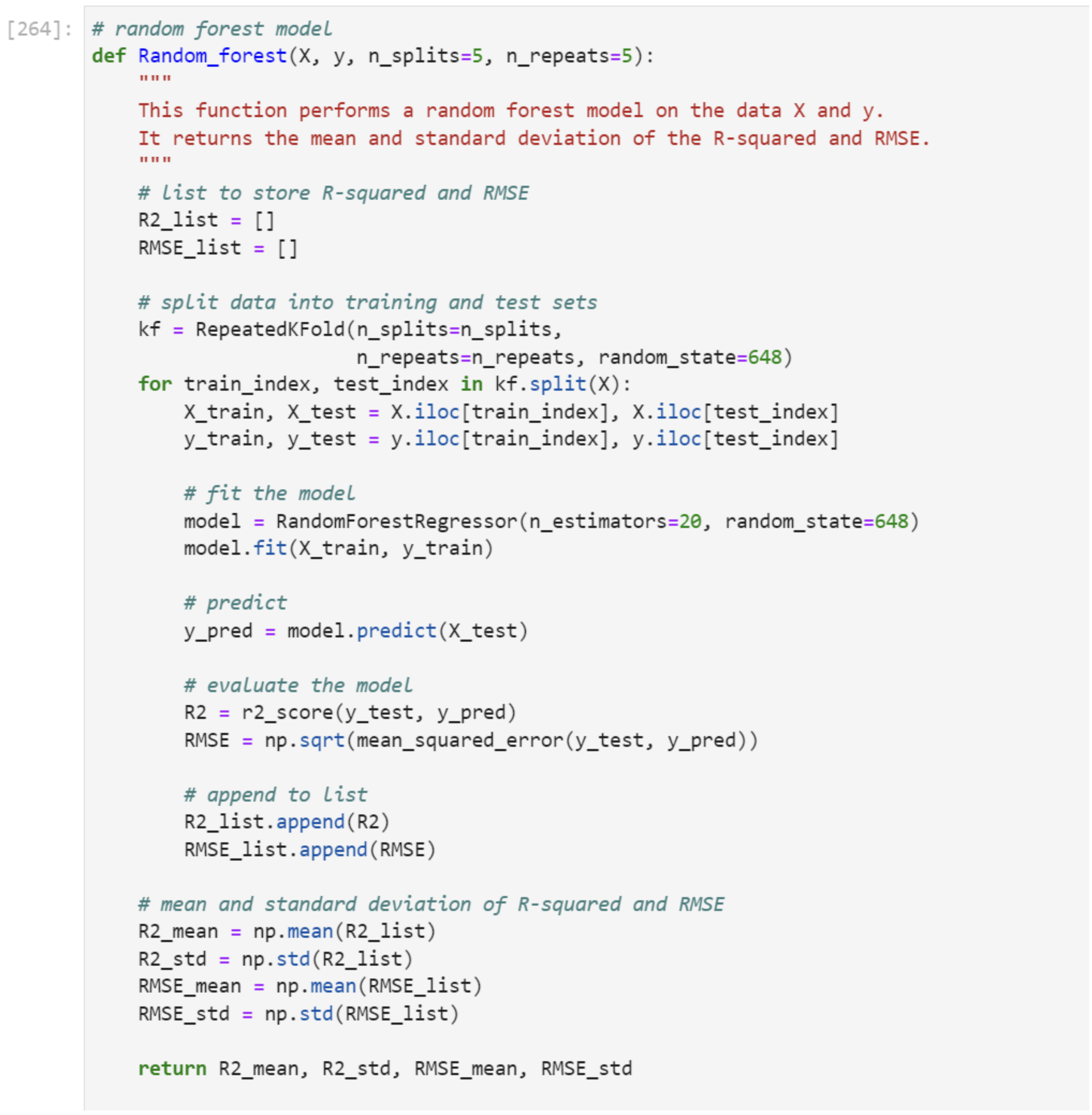
Similar to the multivariate regression model, our input variables are all explanatory variables (selected earlier), and the output explanatory variable is the log-scale house price (AUD).



```
[267]: # random forest
       R2_mean, R2_std, RMSE_mean, RMSE_std = Random_forest(df_X_select, log_y)
```

Figure6.3.4: Random Forest model Parameter Settings

We also need to set the number of trees used in the bagging process. I set it to 20 here. Larger values will give more stable results, but the runtime will be greatly increased.



```
[264]: # random forest model
       def Random_forest(X, y, n_splits=5, n_repeats=5):
           """
           This function performs a random forest model on the data X and y.
           It returns the mean and standard deviation of the R-squared and RMSE.
           """
           # list to store R-squared and RMSE
           R2_list = []
           RMSE_list = []

           # split data into training and test sets
           kf = RepeatedKFold(n_splits=n_splits,
                              n_repeats=n_repeats, random_state=648)
           for train_index, test_index in kf.split(X):
               X_train, X_test = X.iloc[train_index], X.iloc[test_index]
               y_train, y_test = y.iloc[train_index], y.iloc[test_index]

               # fit the model
               model = RandomForestRegressor(n_estimators=20, random_state=648)
               model.fit(X_train, y_train)

               # predict
               y_pred = model.predict(X_test)

               # evaluate the model
               R2 = r2_score(y_test, y_pred)
               RMSE = np.sqrt(mean_squared_error(y_test, y_pred))

               # append to list
               R2_list.append(R2)
               RMSE_list.append(RMSE)

           # mean and standard deviation of R-squared and RMSE
           R2_mean = np.mean(R2_list)
           R2_std = np.std(R2_list)
           RMSE_mean = np.mean(RMSE_list)
           RMSE_std = np.std(RMSE_list)

           return R2_mean, R2_std, RMSE_mean, RMSE_std
```

Figure6.3.5: Random Forest Function

In order to get reproducible results, for algorithms with randomness we need to set a seed. Here I set it to 648, which is the last three digits of my UPI.

So far, we have completed the construction and parameter setting of the models. In the next section, we'll run these models, and try to find conclusions that meet the data mining goals.

7. Data Mining

7.1 Create and justify test designs

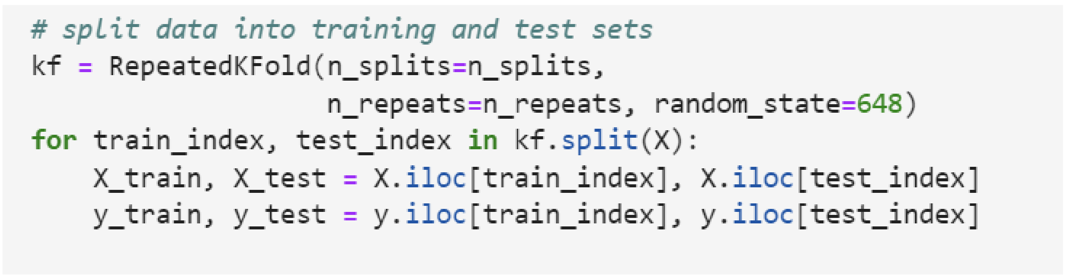
In statistical modeling practice, some methods of data partitioning are often used. This is based on the Golden rule of machine learning -- the test data cannot influence training the model in any way. [9] Therefore, we need to separate the training and test sets. Models are trained on the training set and evaluated on the test set. Usually we only care about the performance of the model on the test set, because over-optimizing on the training set can lead to overfitting, which can lead to large variance. In supervised learning, overfitting happens when our model captures the noise along with the underlying pattern in data. [10] To a large extent, the process of machine learning is a Bias-Variance Tradeoff process.

Specifically, we can use a 70/30 split, that is, use 70% of the data as the training set and the remaining 30% as the test set. This method is also known as Hold-out. Another popular solution is cross-validation. With higher accuracy and stability, it has a higher computational cost. Cross-validation randomly splits the dataset into "k" groups. One of the groups is used as the test set, and the remaining groups are used as the training set. The model is trained on the training set and scored on the test set. The process is then repeated until each unique group is used as a test set.[11]

Here, I decided to use 5-fold cross-validation, which is to use 20% of the data as the test set each time, and the rest of the data as the training set, and repeat it five times. This is because cross-validation has higher accuracy and stability; with higher folds it will perform better, but the run time will be beyond my tolerance.

7.2 Conduct data mining – classify, regress, cluster, etc.

First let's focus on the design of cross-validation and repeated experiments. For each algorithm, I run it 5 times, using 5-fold cross-validation each time. Both n_splits and n_repeats are 5 here.



```
# split data into training and test sets
kf = RepeatedKFold(n_splits=n_splits,
                   n_repeats=n_repeats, random_state=648)
for train_index, test_index in kf.split(X):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]
```

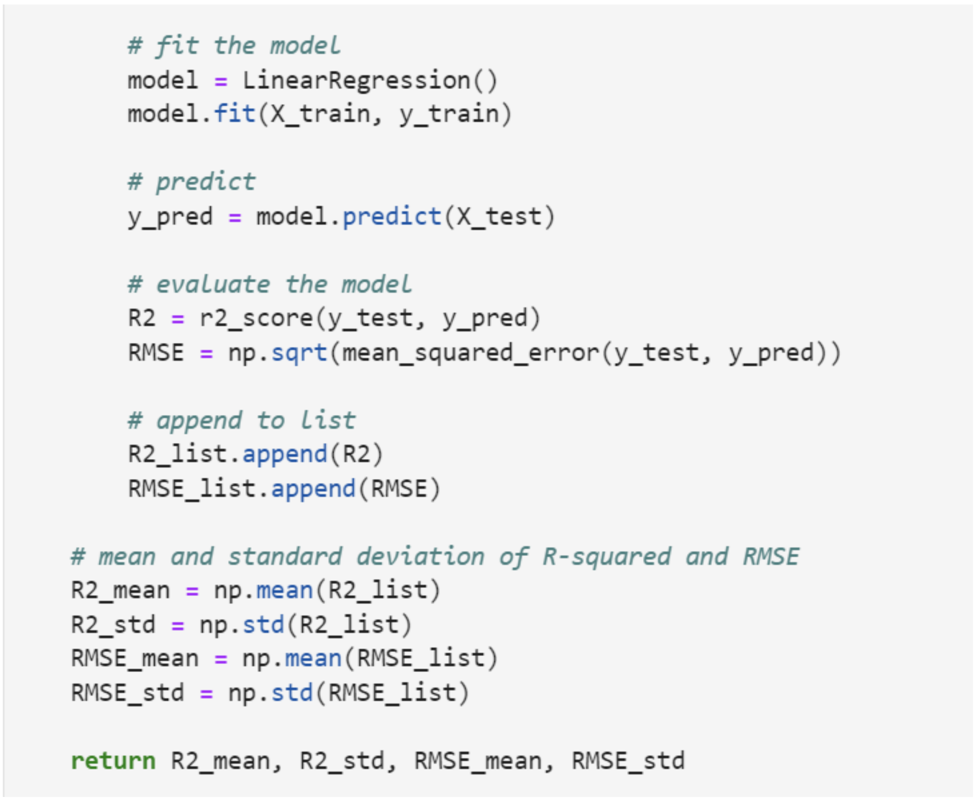
Figure7.2.1: CV and Repeated Experiments

For each model, I keep 4 performance measures, that is the mean and standard deviation of **R2** and **RMSE**.

R2 is a measure that tells us the proportion of variance in the response variable of the regression model that can be explained by the predictor variables. This value ranges from 0 to 1. The higher the R2 value, the better the model fits the dataset.[15]

RMSE is a metric that tells us the average distance of the predicted value from the observed value in the dataset. The lower the RMSE, the better the model fits the dataset.[15]

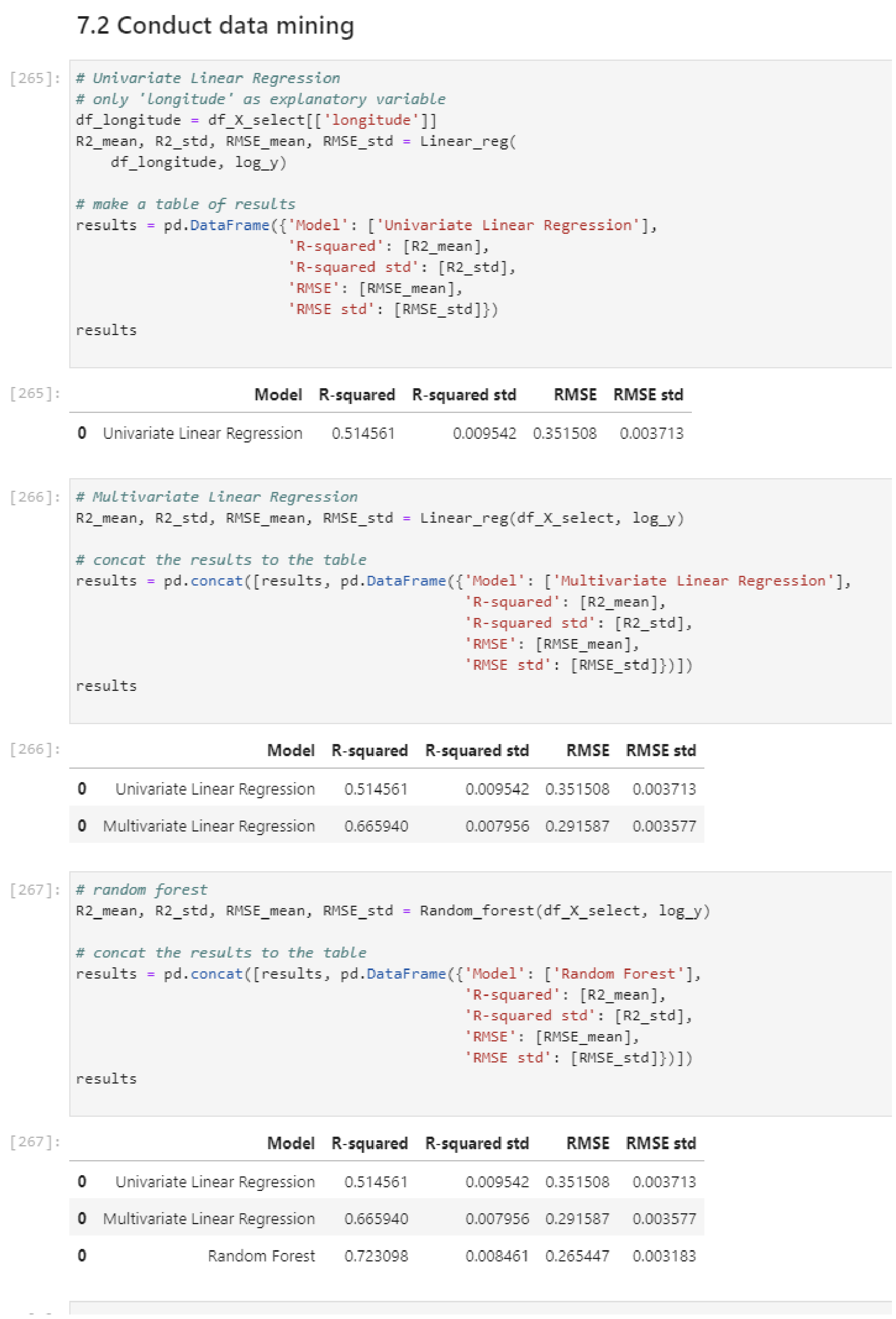
The mean value of R2 and RMSE reflects the average performance of the model fitting the test data, and the standard deviation of R2 and RMSE reflects the stability of the model. The smaller the standard deviation, the more stable the model is.



```
        # fit the model
        model = LinearRegression()
        model.fit(X_train, y_train)

        # predict
        y_pred = model.predict(X_test)

        # evaluate the model
        R2 = r2_score(y_test, y_pred)
        RMSE = np.sqrt(mean_squared_error(y_test, y_pred))

        # append to list
        R2_list.append(R2)
        RMSE_list.append(RMSE)

    # mean and standard deviation of R-squared and RMSE
    R2_mean = np.mean(R2_list)
    R2_std = np.std(R2_list)
    RMSE_mean = np.mean(RMSE_list)
    RMSE_std = np.std(RMSE_list)

    return R2_mean, R2_std, RMSE_mean, RMSE_std
```

Figure7.2.2: Model Performance Metrics

By calling the functions defined earlier, I ran the three algorithms mentioned earlier on the dataset and got some results. Here just to proof that I ran it successfully, and I'll explore these results in the next section.



7.2 Conduct data mining

```
[265]: # Univariate Linear Regression
       # only 'longitude' as explanatory variable
       df_longitude = df_X_select[['longitude']]
       R2_mean, R2_std, RMSE_mean, RMSE_std = Linear_reg(
           df_longitude, log_y)

       # make a table of results
       results = pd.DataFrame({'Model': ['Univariate Linear Regression'],
                               'R-squared': [R2_mean],
                               'R-squared std': [R2_std],
                               'RMSE': [RMSE_mean],
                               'RMSE std': [RMSE_std]})
       results
```

```
[265]:                           Model  R-squared  R-squared std      RMSE  RMSE std
       0  Univariate Linear Regression   0.514561       0.009542  0.351508  0.003713
```

```
[266]: # Multivariate Linear Regression
       R2_mean, R2_std, RMSE_mean, RMSE_std = Linear_reg(df_X_select, log_y)

       # concat the results to the table
       results = pd.concat([results, pd.DataFrame({'Model': ['Multivariate Linear Regression'],
                                                   'R-squared': [R2_mean],
                                                   'R-squared std': [R2_std],
                                                   'RMSE': [RMSE_mean],
                                                   'RMSE std': [RMSE_std]})])
       results
```

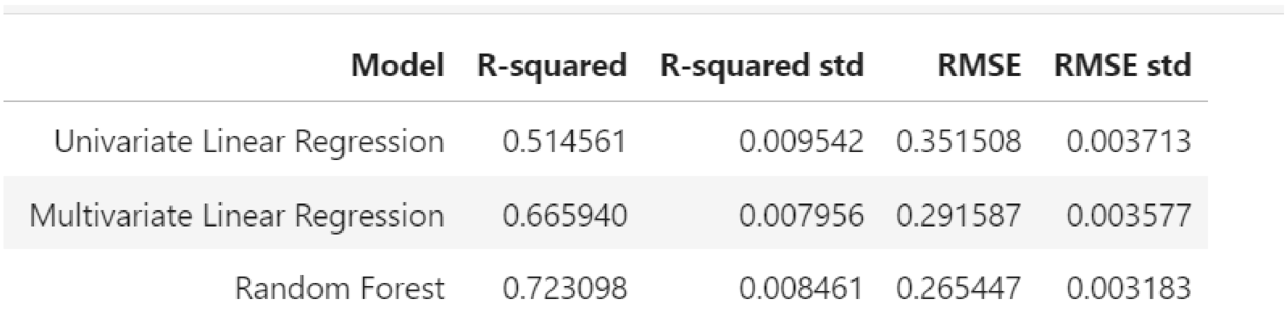
```
[266]:                             Model  R-squared  R-squared std      RMSE  RMSE std
       0    Univariate Linear Regression   0.514561       0.009542  0.351508  0.003713
       0  Multivariate Linear Regression   0.665940       0.007956  0.291587  0.003577
```

```
[267]: # random forest
       R2_mean, R2_std, RMSE_mean, RMSE_std = Random_forest(df_X_select, log_y)

       # concat the results to the table
       results = pd.concat([results, pd.DataFrame({'Model': ['Random Forest'],
                                                   'R-squared': [R2_mean],
                                                   'R-squared std': [R2_std],
                                                   'RMSE': [RMSE_mean],
                                                   'RMSE std': [RMSE_std]})])
       results
```

```
[267]:                             Model  R-squared  R-squared std      RMSE  RMSE std
       0    Univariate Linear Regression   0.514561       0.009542  0.351508  0.003713
       0  Multivariate Linear Regression   0.665940       0.007956  0.291587  0.003577
       0                   Random Forest   0.723098       0.008461  0.265447  0.003183
```

Figure7.2.3: Run 3 Algorithms

7.3 Search for patterns



Model	R-squared	R-squared std	RMSE	RMSE std
Univariate Linear Regression	0.514561	0.009542	0.351508	0.003713
Multivariate Linear Regression	0.665940	0.007956	0.291587	0.003577
Random Forest	0.723098	0.008461	0.265447	0.003183

Figure7.3.1: Model performance

```
[270]: # plot the results: R-squared
       plt.bar(results['Model'], results['R-squared'])
       plt.title('R-squared of 3 models')
       plt.ylabel('R-squared')
       plt.xticks(rotation=45)
       plt.show()
```

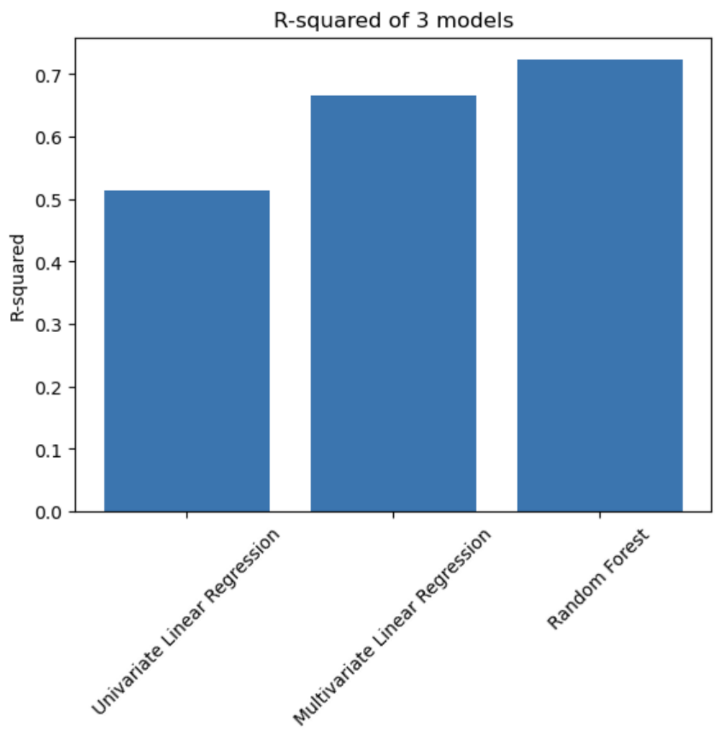
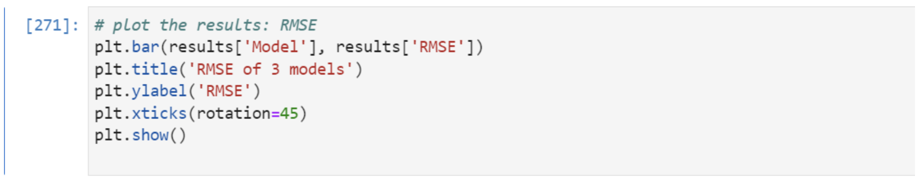


Figure7.3.2: R2



```
[271]: # plot the results: RMSE
       plt.bar(results['Model'], results['RMSE'])
       plt.title('RMSE of 3 models')
       plt.ylabel('RMSE')
       plt.xticks(rotation=45)
       plt.show()
```

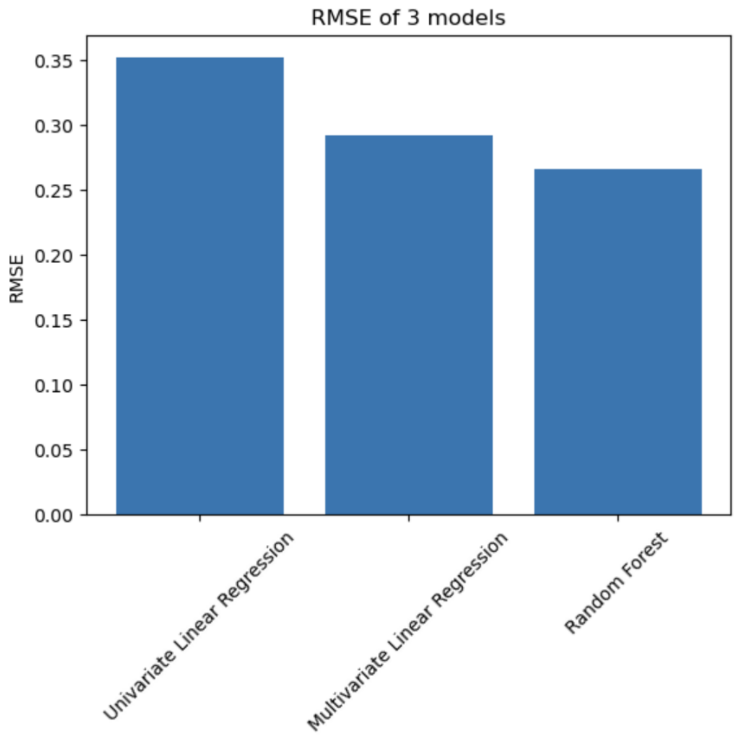


Figure7.3.3: RMSE

From the results, in terms of model stability, the standard deviation of the indicators of the univariate linear regression model is slightly larger, so its stability is slightly worse than the other two models. In terms of average prediction performance, the univariate linear regression model has the smallest R2 and the largest RMSE, which means its performance is the worst. The random forest has the largest R2 and the smallest RMSE, which means that its performance is the best among the three models. The performance of multivariate linear models is somewhere in between.

These results are not surprising: only one explanatory variable is used in the linear model, resulting in a model that is too simplistic and has a large bias. This is a typical underfitting model, which will not perform well on either the training set or the test set. Both multivariate regression models and random forests use the full dataset as explanatory variables and thus have much better predictive performance. Being a more complex algorithm, random forest has higher prediction performance than the other two, but also has the longest running time.

8. Interpretation

8.1 Study and discuss the mined patterns

In the previous steps, we identified a supervised method rather than an unsupervised method to achieve our data mining goals, and further confirmed that we used a regression method rather than a classification method.

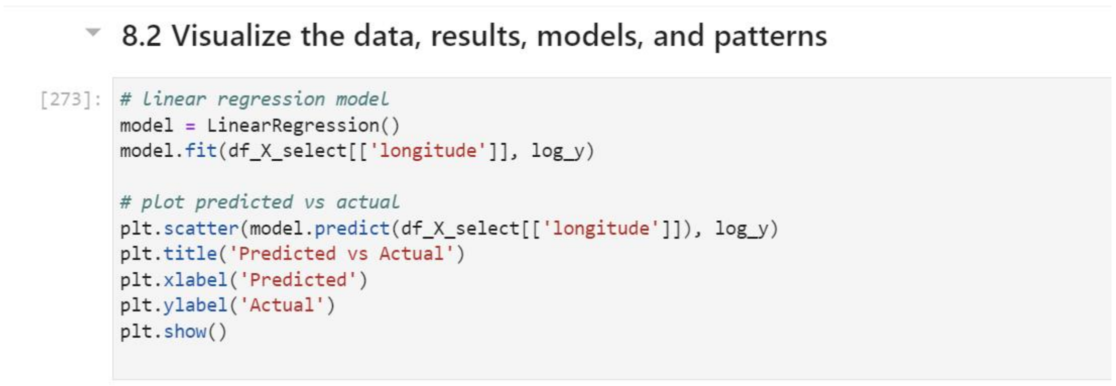
As mentioned in Section 7.3, random forests have the best predictive performance, while multivariate linear models also perform well and are more interpretable than random forests. We will discuss it in detail in subsequent subsections.

Before fitting a model, it is necessary to determine whether the data conform to the statistical assumptions of the model. Linear regression models assume that the relationship between the response variable and the explanatory variable is linear, and the residuals are independent, with a constant variance. We observe that the relationship between property prices and some important explanatory variables is not linear, and the variance is obviously not constant, but increasing with the increase of levels. This does not meet the assumptions of linear regression, so we need to convert prices to log scale.

Finally, we used the mean of R2 and RMSE to estimate the average predictive performance of the models, and the standard deviation of R2 and RMSE to estimate the stability of the models. The final conclusion is that univariate linear regression has the worst performance and random forest has the best performance.

8.2 Visualize the data, results, models, and patterns

In the predicted value -- actual value plot of the linear model, the linear relationship between the two is not very obvious. The scattered distribution of data points is shown in the plot.



8.2 Visualize the data, results, models, and patterns

```
[273]: # linear regression model
       model = LinearRegression()
       model.fit(df_X_select[['longitude']], log_y)

       # plot predicted vs actual
       plt.scatter(model.predict(df_X_select[['longitude']]), log_y)
       plt.title('Predicted vs Actual')
       plt.xlabel('Predicted')
       plt.ylabel('Actual')
       plt.show()
```

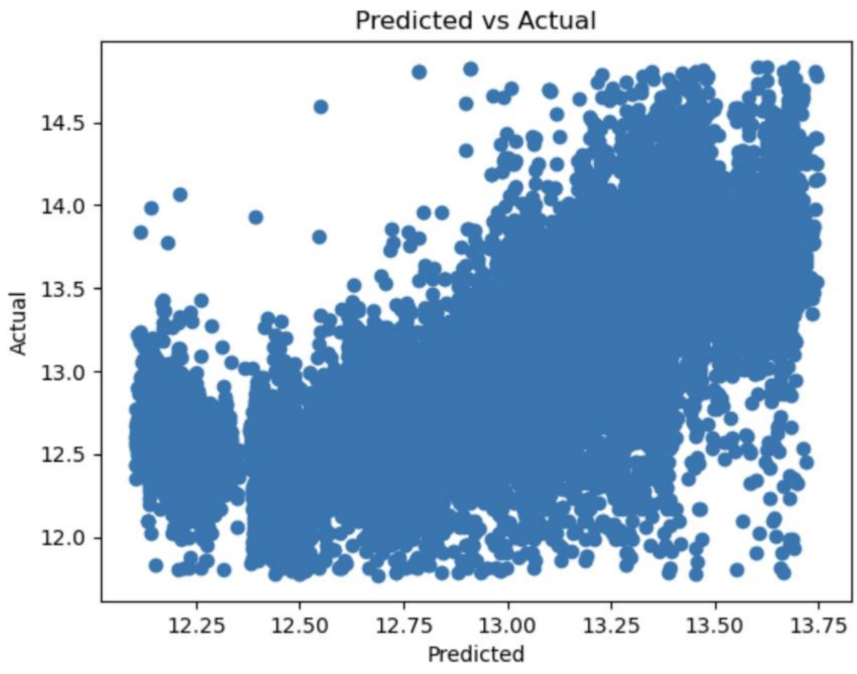


Figure8.2.1: Performance of Linear Model

In the plot of the multivariate regression model, a clear linear relationship can be observed.

```
[274]: # multiple linear regression model
model = LinearRegression()
model.fit(df_X_select, log_y)

# plot predicted vs actual
plt.scatter(model.predict(df_X_select), log_y)
plt.title('Predicted vs Actual')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

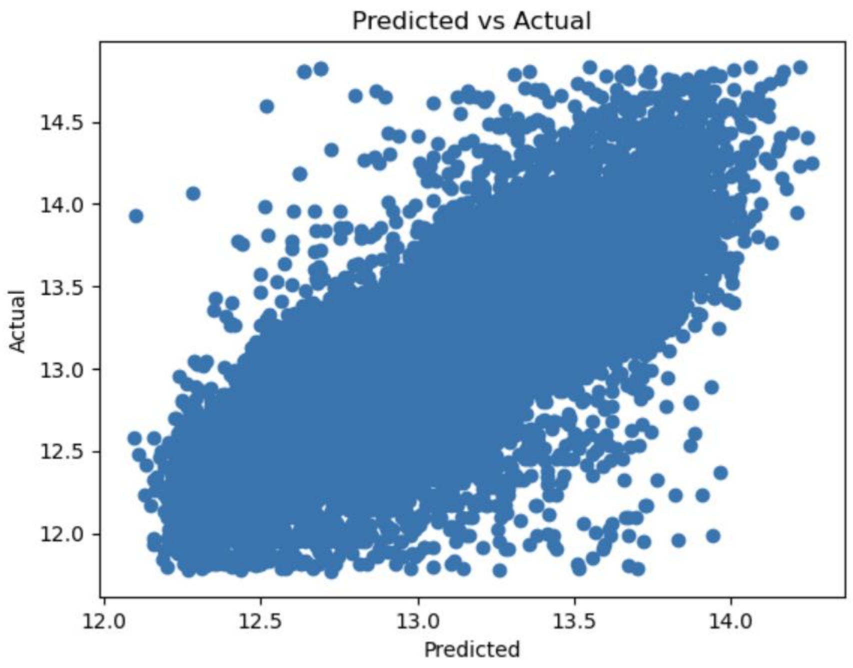


Figure8.2.2: Performance of Multi-regression Model

For random forest model, the linear relationship between the actual value and the predicted value is the most obvious among the three models.

```
[275]: # random forest model
model = RandomForestRegressor(n_estimators=20, random_state=648)
model.fit(df_X_select, log_y)

# plot predicted vs actual
plt.scatter(model.predict(df_X_select), log_y)
plt.title('Predicted vs Actual')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

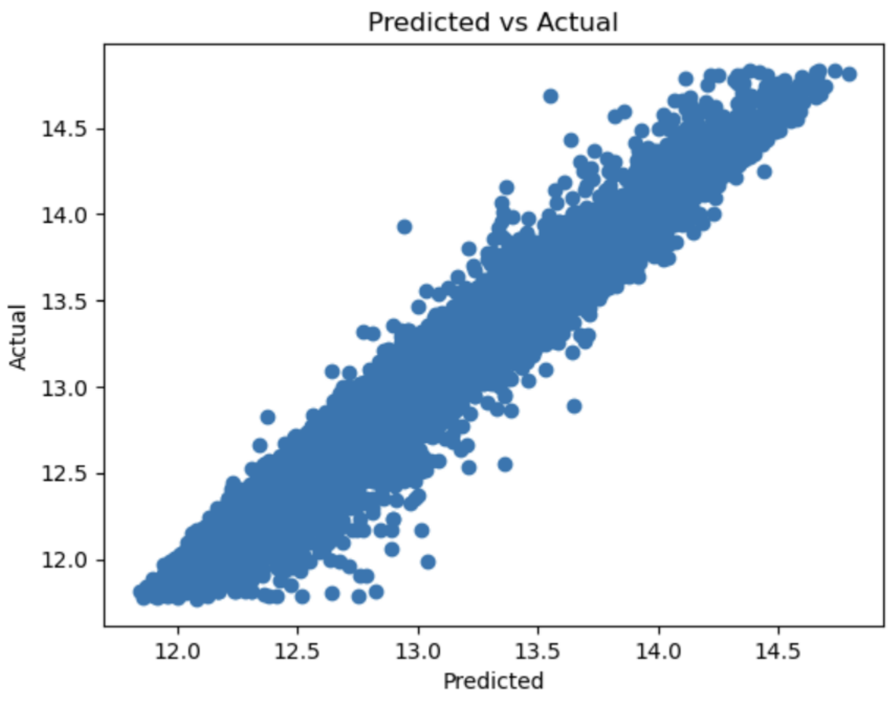
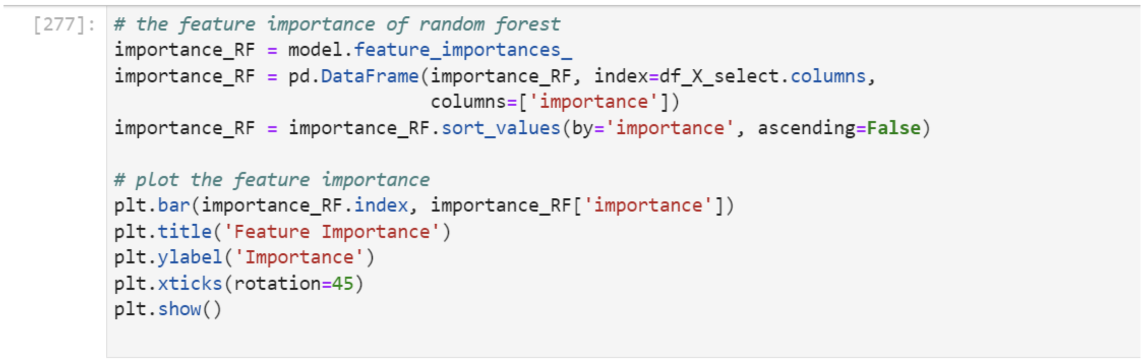


Figure8.2.3: Performance of RF Model

Random forest gives the importance ranking of explanatory variables.

For housing prices, the top four most important influencing factors are longitude, income, lotSize and distToCoastline. This is slightly different from the previous results get using SelectKBest.



```
[277]: # the feature importance of random forest
       importance_RF = model.feature_importances_
       importance_RF = pd.DataFrame(importance_RF, index=df_X_select.columns,
                                    columns=['importance'])
       importance_RF = importance_RF.sort_values(by='importance', ascending=False)

       # plot the feature importance
       plt.bar(importance_RF.index, importance_RF['importance'])
       plt.title('Feature Importance')
       plt.ylabel('Importance')
       plt.xticks(rotation=45)
       plt.show()
```

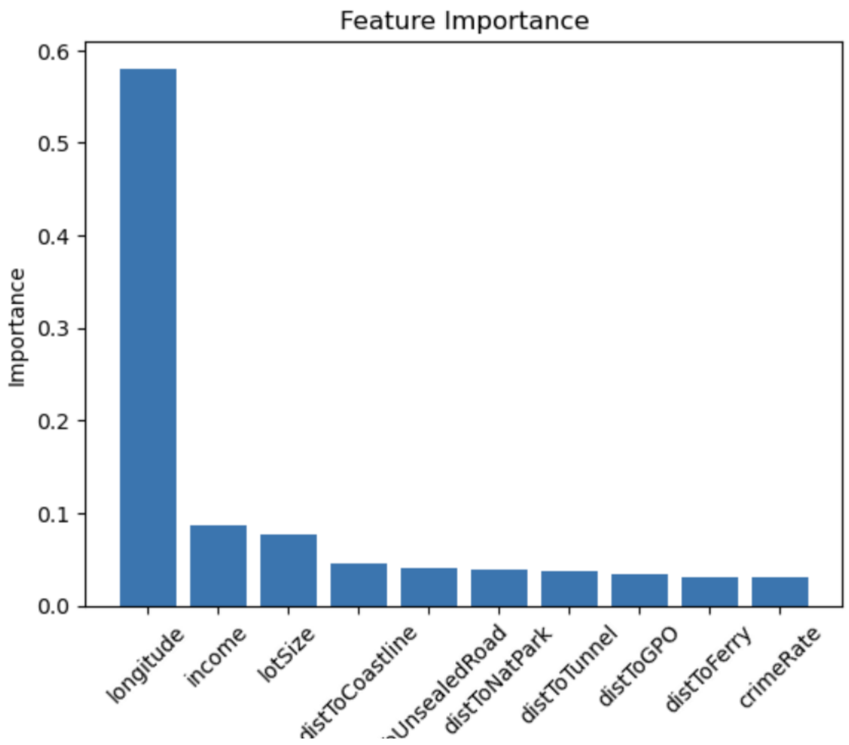


Figure8.2.4: Feature Importance from RF

Some variables are positively correlated with the logarithm of property prices, such as income level; others are negatively correlated with the logarithm of property prices, such as the distance from the property to the coastline.

[278]:
```python
# plot income vs logSalePrice
plt.scatter(df_X_select[['income']], log_y)
plt.title('Income vs logSalePrice')
plt.xlabel('Income')
plt.ylabel('logSalePrice')
plt.show()
```

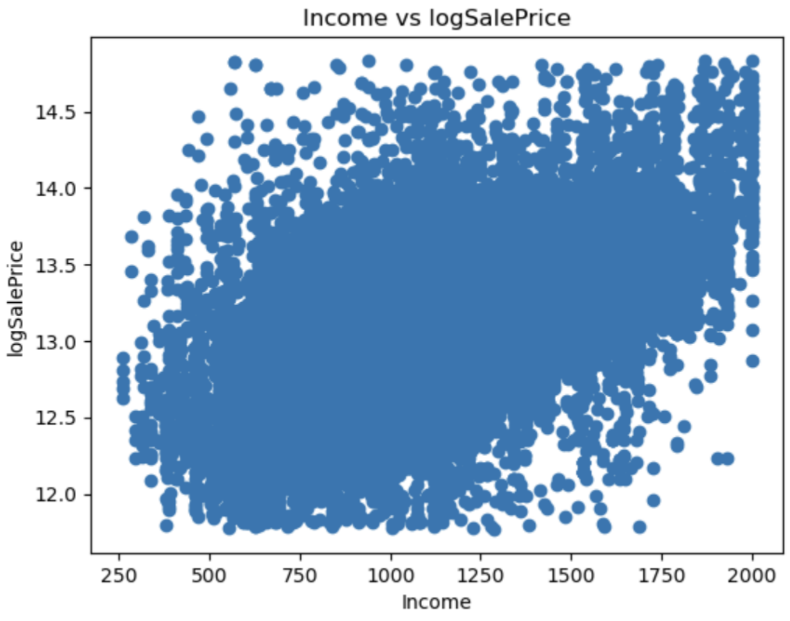


Figure8.2.5: Positive Correlation

[280]:
```python
# plot distToCoastline vs logSalePrice
plt.scatter(df_X_select[['distToCoastline']], log_y)
plt.title('distToCoastline vs logSalePrice')
plt.xlabel('distToCoastline')
plt.ylabel('logSalePrice')
plt.show()
```

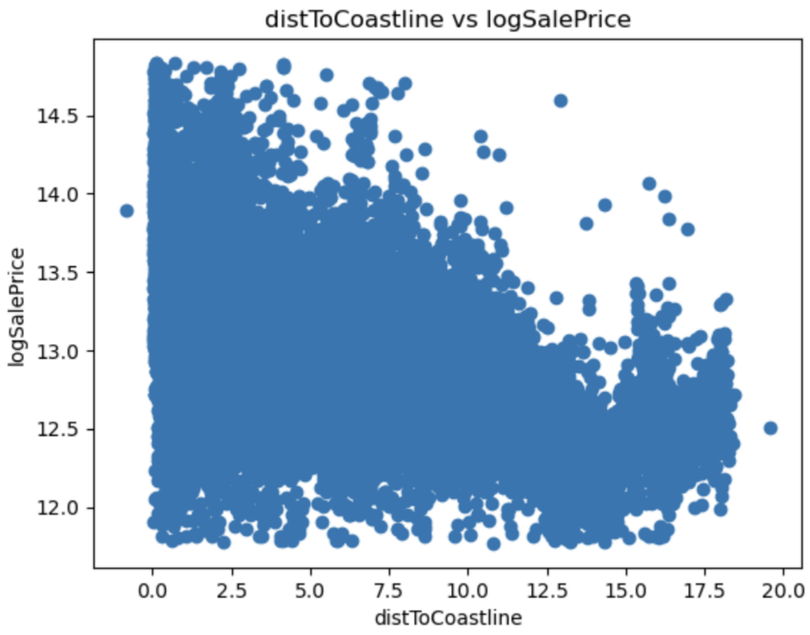


Figure8.2.6: Negative Correlation

I would expect a very clear positive correlation between longitude and property prices as it is considered the most important influencer on property prices in Sydney.

The result surprised me a little, it seems that the relationship between longitude and property prices is not monotonically positive: properties with a longitude of 150.75 appear to have the lowest prices.

```
[281]: # plot longitude vs logSalePrice
       plt.scatter(df_X_select[['longitude']], log_y)
       plt.title('longitude vs logSalePrice')
       plt.xlabel('longitude')
       plt.ylabel('logSalePrice')
       plt.show()
```

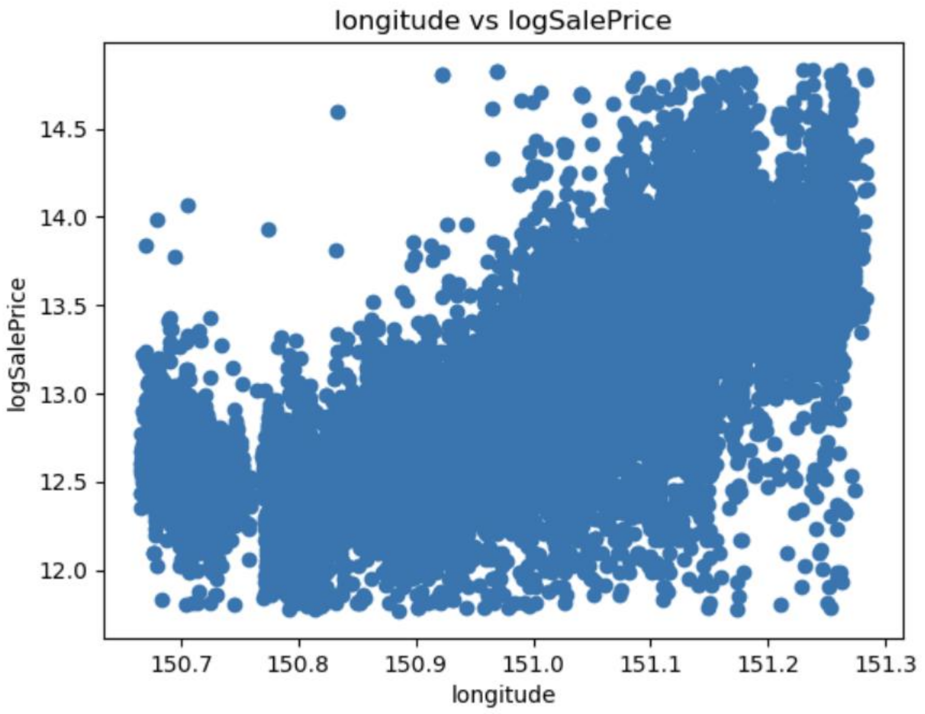
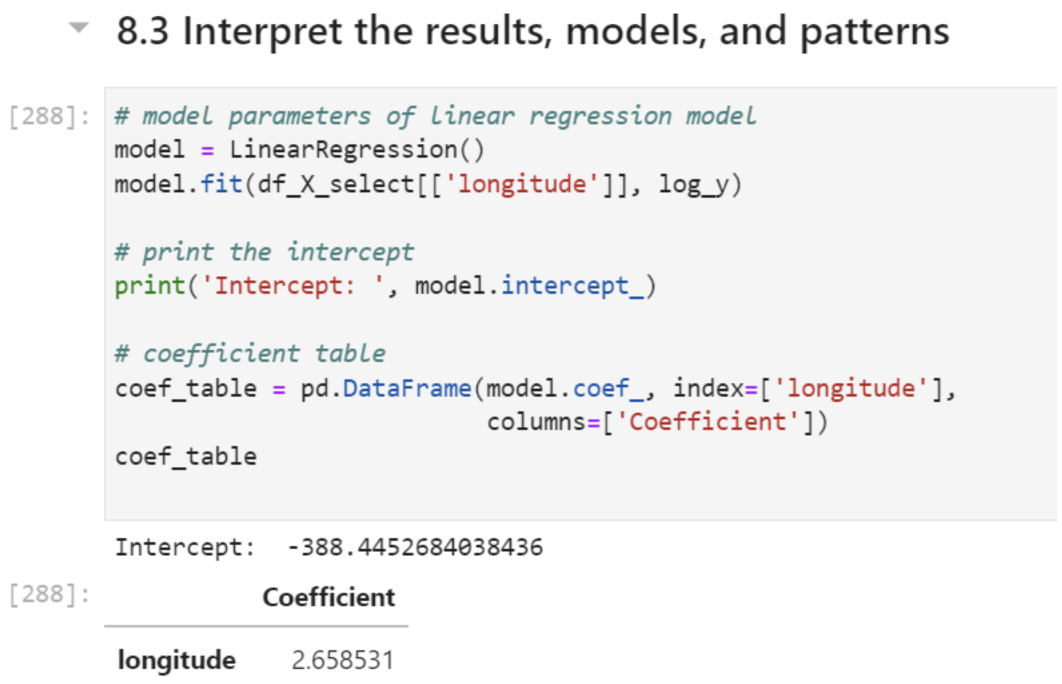


Figure8.2.7: Longitude vs logSalePrice

Specifically, when the longitude is less than 150.75, the more eastward the house price is lower; when the longitude is greater than 150.75, the more eastward the house price is higher.

8.3 Interpret the results, models, and patterns

The model parameters of the **Linear model** are as follows:



8.3 Interpret the results, models, and patterns

```
[288]: # model parameters of linear regression model
       model = LinearRegression()
       model.fit(df_X_select[['longitude']], log_y)

       # print the intercept
       print('Intercept: ', model.intercept_)

       # coefficient table
       coef_table = pd.DataFrame(model.coef_, index=['longitude'],
                                 columns=['Coefficient'])
       coef_table

       Intercept:  -388.4452684038436
```

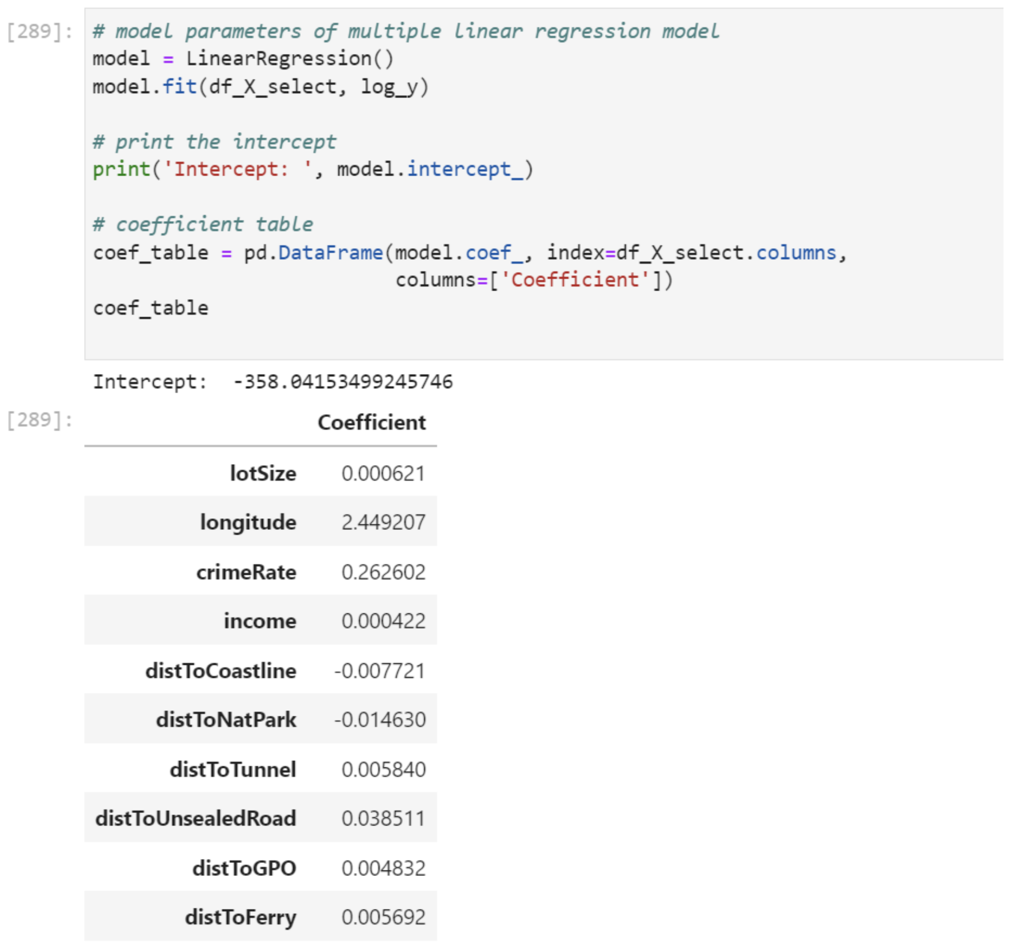
[288]:

	Coefficient
longitude	2.658531

Figure8.3.1: Linear Model Parameters

From the model parameters of the linear model, it can be concluded that each time the property location increases by one unit of longitude, the property price on the log scale will increase by about 2.66, that is to say, the property price will be the original exp (2.658531), which is about 14.275 times. However, the longitude changes within the city of Sydney will not be too large, so the actual number will not be that large.

The model parameters of the **Multivariate Regression model** are as follows:



```
[289]: # model parameters of multiple linear regression model
       model = LinearRegression()
       model.fit(df_X_select, log_y)

       # print the intercept
       print('Intercept: ', model.intercept_)

       # coefficient table
       coef_table = pd.DataFrame(model.coef_, index=df_X_select.columns,
                                 columns=['Coefficient'])
       coef_table

       Intercept:  -358.04153499245746
```

[289]:

	Coefficient
lotSize	0.000621
longitude	2.449207
crimeRate	0.262602
income	0.000422
distToCoastline	-0.007721
distToNatPark	-0.014630
distToTunnel	0.005840
distToUnsealedRoad	0.038511
distToGPO	0.004832
distToFerry	0.005692

Figure8.3.2: Multivariate Regression Model Parameters

The parameters of longitude have changed compared to previous model, because here we take all explanatory variables into the model. Some parameters can be difficult to interpret because not all parameters fit the assumption of a linear relationship with property prices on log-scale. For example, if you want to explain the relationship between lot size and house price, you can say that log(Price) increases by 0.000621 for every 1 square meter increase in lot size, that is, property price multiplied by exp(0.000621) = 1.00062119. This is obviously problematic because the relationship between property price and lot size can't be exponential. After transforming the response variable to log-scale, the model is more predictive, but less interpretable.

However, we can still see some trends from the parameter list. When the coefficient of an explanatory variable is greater than 0, it can be said to be positively correlated with the response variable, such as lot size and income; when the coefficient of an explanatory variable is less than 0, it can be said to be negatively correlated with the response variable, such as the distance to the coastline. This is not surprising, intuitively, houses with larger lot sizes are more expensive; people with higher incomes tend to buy more expensive houses; people are more Prefer to live closer to the coastline, so the farther from the coastline, the cheaper the property. The relationship between longitude and house prices may be explained by the location and coastline of Sydney city. The larger the longitude, the further east. Since Sydney is on the east coast of Australia, so the larger the longitude, the closer it is to the sea.

8.4 Assess and evaluate results, models, and patterns

We fit a total of three models: linear model, multiple regression model and random forest. The performance of the linear model is the worst, with severe underfitting. In actual forecasting, we cannot use such a model. Multiple regression and random forests perform similarly, with random forests slightly better than multiple regression models. However, the running time of the random forest model is significantly longer, and the cost of slightly better prediction performance is obvious. It is reasonable to believe that random forests will achieve better predictive performance if the number of bagging trees is increased. In actual data mining practice, data mining goals should be fully considered, and whether more complex, longer-running, and harder-to-interpret models should be used for small improvements. My answer tends to be no.

From the parameter coefficient of the multivariate linear model, the factors that lead to higher real estate prices are:
--larger lot size
-- Higher average weekly income of the suburb in which the house is located
-- larger longitude
. . . . . .
Factors that lead to lower property prices include:
-- farther from the coastline
-- farther from the national parks
. . . . . .

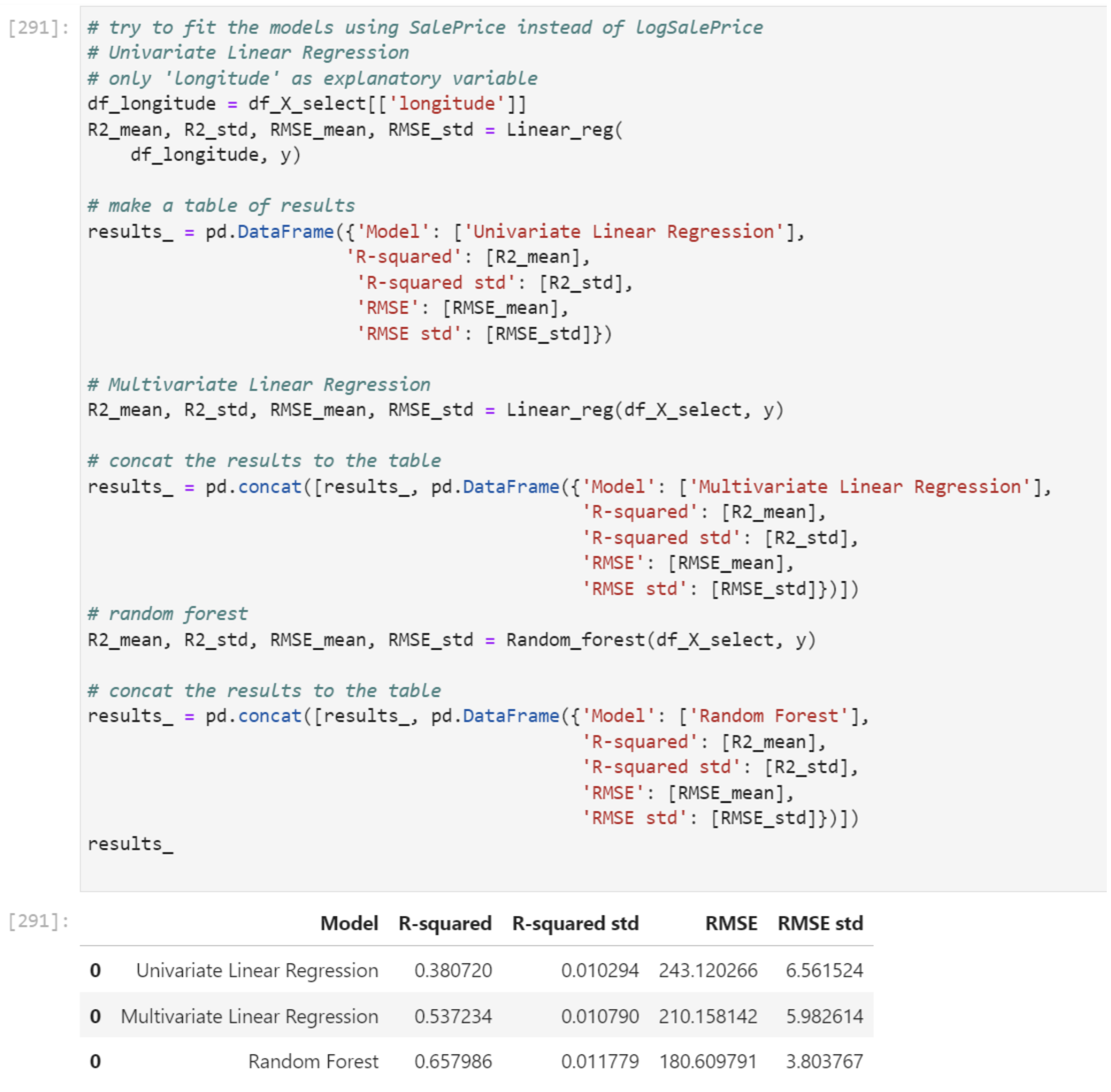
The multiple linear model also gave us some incredible conclusions, such as the higher the crime rate around the property, the higher the property price. A possible explanation is that the more expensive homes are located in the city center, where has a higher crime rate.

8.5 Iterate prior steps (1 – 7) as required

8.5.1 SalePrice instead of logSalePrice

Earlier we decided to use a log-scale response variable because such a response variable is more likely to satisfy the model's assumptions (linear relationship and constant variance). So what happens if we don't transform the response variable?

The result is worse model performance than before. There was a drop in R2 for each model and a clear rise in RMSE.



[291]:
```python
# try to fit the models using SalePrice instead of LogSalePrice
# Univariate Linear Regression
# only 'longitude' as explanatory variable
df_longitude = df_X_select[['longitude']]
R2_mean, R2_std, RMSE_mean, RMSE_std = Linear_reg(
    df_longitude, y)

# make a table of results
results_ = pd.DataFrame({'Model': ['Univariate Linear Regression'],
                        'R-squared': [R2_mean],
                         'R-squared std': [R2_std],
                         'RMSE': [RMSE_mean],
                         'RMSE std': [RMSE_std]})

# Multivariate Linear Regression
R2_mean, R2_std, RMSE_mean, RMSE_std = Linear_reg(df_X_select, y)

# concat the results to the table
results_ = pd.concat([results_, pd.DataFrame({'Model': ['Multivariate Linear Regression'],
                                             'R-squared': [R2_mean],
                                             'R-squared std': [R2_std],
                                             'RMSE': [RMSE_mean],
                                             'RMSE std': [RMSE_std]})])
# random forest
R2_mean, R2_std, RMSE_mean, RMSE_std = Random_forest(df_X_select, y)

# concat the results to the table
results_ = pd.concat([results_, pd.DataFrame({'Model': ['Random Forest'],
                                             'R-squared': [R2_mean],
                                             'R-squared std': [R2_std],
                                             'RMSE': [RMSE_mean],
                                             'RMSE std': [RMSE_std]})])
results_
```

[291]:

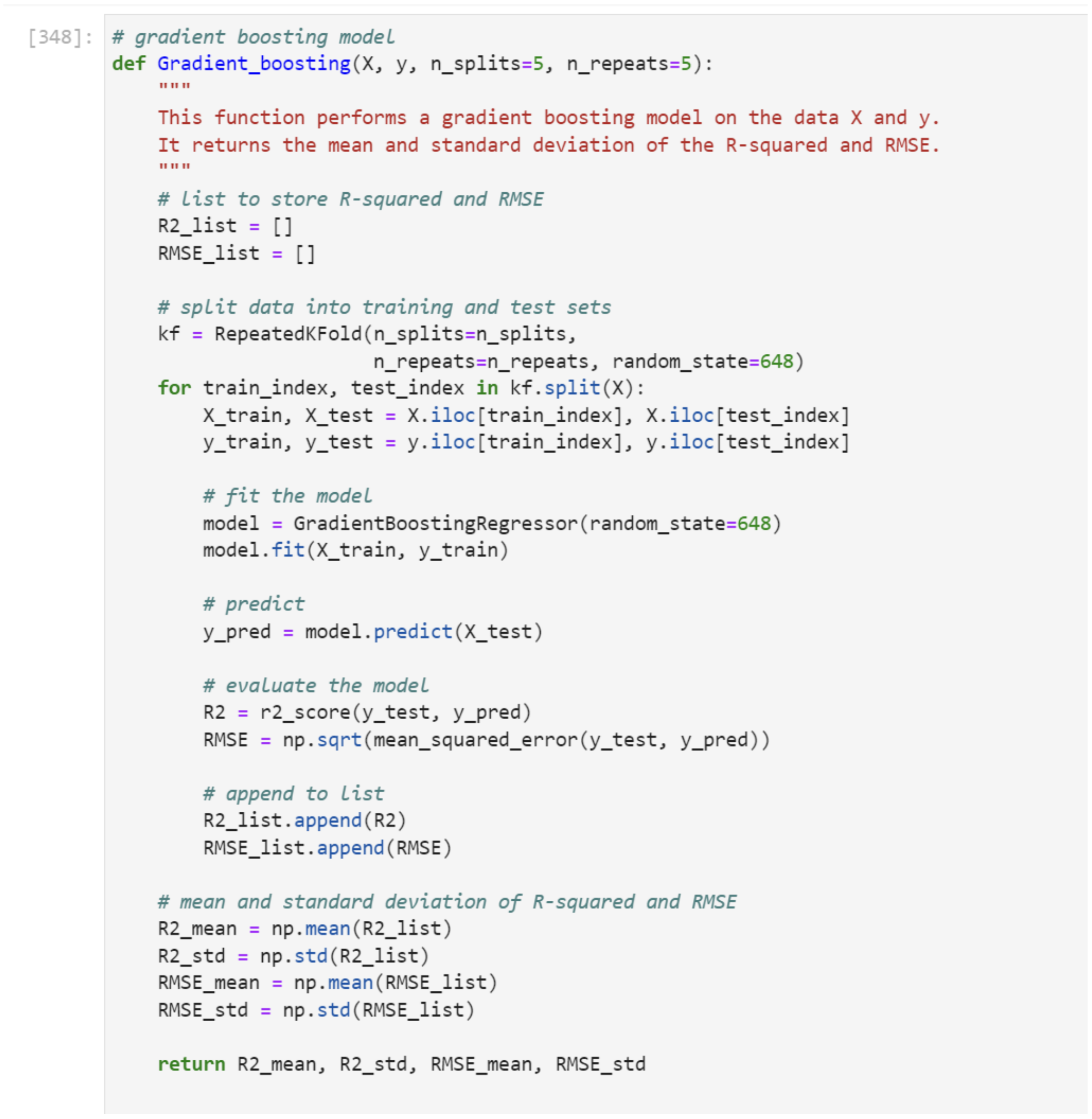
	Model	R-squared	R-squared std	RMSE	RMSE std
0	Univariate Linear Regression	0.380720	0.010294	243.120266	6.561524
0	Multivariate Linear Regression	0.537234	0.010790	210.158142	5.982614
0	Random Forest	0.657986	0.011779	180.609791	3.803767

Figure8.5.1: Bad Results

8.5.2 More models: Gradient Boosting

Gradient boosting is a machine learning technique used for regression and classification tasks, among others. It gives a predictive model in the form of a collection of weak predictive models, which are usually decision trees. When a decision tree is a weak learner, the resulting algorithm is called a gradient boosted tree; it usually outperforms random forests.[16]

I used the same test design as before, using the GradientBoostingRegressor model from sklearn.ensemble to model and predict log-scale response variables and all selected explanatory variables.

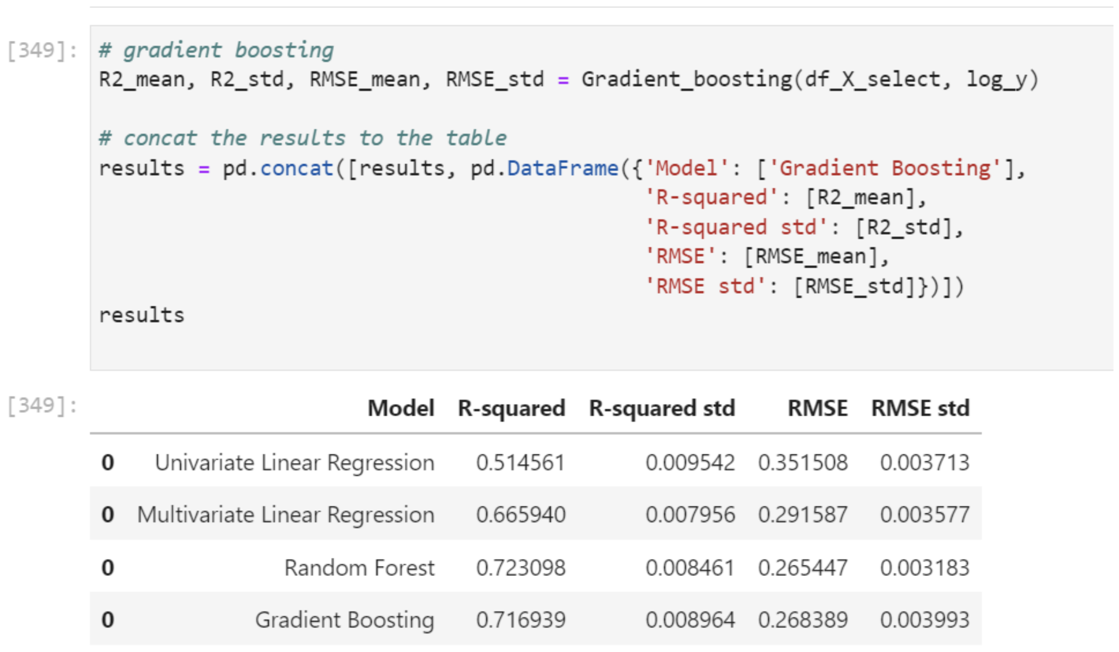


```
[348]: # gradient boosting model
def Gradient_boosting(X, y, n_splits=5, n_repeats=5):
    """
    This function performs a gradient boosting model on the data X and y.
    It returns the mean and standard deviation of the R-squared and RMSE.
    """
    # list to store R-squared and RMSE
    R2_list = []
    RMSE_list = []

    # split data into training and test sets
    kf = RepeatedKFold(n_splits=n_splits,
                       n_repeats=n_repeats, random_state=648)
    for train_index, test_index in kf.split(X):
        X_train, X_test = X.iloc[train_index], X.iloc[test_index]
        y_train, y_test = y.iloc[train_index], y.iloc[test_index]

        # fit the model
        model = GradientBoostingRegressor(random_state=648)
        model.fit(X_train, y_train)

        # predict
        y_pred = model.predict(X_test)

        # evaluate the model
        R2 = r2_score(y_test, y_pred)
        RMSE = np.sqrt(mean_squared_error(y_test, y_pred))

        # append to list
        R2_list.append(R2)
        RMSE_list.append(RMSE)

    # mean and standard deviation of R-squared and RMSE
    R2_mean = np.mean(R2_list)
    R2_std = np.std(R2_list)
    RMSE_mean = np.mean(RMSE_list)
    RMSE_std = np.std(RMSE_list)

    return R2_mean, R2_std, RMSE_mean, RMSE_std
```

Figure8.5.2: Gradient Boosting Function

Following the same experimental design, gradient boosting showed stronger performance than both linear regression methods. However, in my tests, gradient boosting was not stronger than random forest.

The specific results are as follows:



```
[349]: # gradient boosting
       R2_mean, R2_std, RMSE_mean, RMSE_std = Gradient_boosting(df_X_select, log_y)

       # concat the results to the table
       results = pd.concat([results, pd.DataFrame({'Model': ['Gradient Boosting'],
                                                   'R-squared': [R2_mean],
                                                   'R-squared std': [R2_std],
                                                   'RMSE': [RMSE_mean],
                                                   'RMSE std': [RMSE_std]})])
       results
```

[349]:

	Model	R-squared	R-squared std	RMSE	RMSE std
0	Univariate Linear Regression	0.514561	0.009542	0.351508	0.003713
0	Multivariate Linear Regression	0.665940	0.007956	0.291587	0.003577
0	Random Forest	0.723098	0.008461	0.265447	0.003183
0	Gradient Boosting	0.716939	0.008964	0.268389	0.003993

Figure8.5.2: Gradient Boosting Results

The R2 of gradient boosting is slightly lower than random forest, while the RMSE is slightly higher than random forest. Gradient boosting takes longer running time than random forest here.

References

[Aut], J. D. H. R. (2021a, November 23). *HRW: Datasets, Functions and Scripts for Semiparametric Regression Supporting Harezlak, Ruppert & Wand (2018)*. R Package Documentation. https://rdrr.io/cran/HRW/

[Aut], J. D. H. R. (2021b, November 23). *Sydney real estate*. R Package Documentation. https://rdrr.io/cran/HRW/man/SydneyRealEstate.html

Realestate, E. (2021, October 20). *Factors That Affect Housing Prices in Sydney*. Etch Real Estate. https://www.etchrealestate.com.au/factors-that-affect-housing-prices-in-sydney/

Sustainable cities and human settlements | Department of Economic and Social Affairs. (2018). United Nations. https://sdgs.un.org/topics/sustainable-cities-and-human-settlements

Yee, T. (Ed.). (2022). Introduction to Data Mining. In *STATS 784: Statistical Data Mining* (pp. 60–67).

Cialdella, L. (2020, August 30). *When do we log transform the response variable? Model assumptions, multiplicative combinations and log-linear models*. Casual Inference. https://lmc2179.github.io/posts/multiplicative.html

Agrawal, P., Gupta, C., Sharma, A., Madaan, V., & Joshi, N. (2022). *Machine Learning and Data Science: Fundamentals and Applications* (1st ed.). Wiley-Scrivener.

Education, I. C. (2021, January 26). *Random Forest*. IBM. https://www.ibm.com/cloud/learn/random-forest

Machine Learning and Data Mining. (2016). In *CPSC 340 UBC* (p. 19).

Singh, S. (2022, February 18). *Understanding the Bias-Variance Tradeoff - Towards Data Science*. Medium. https://towardsdatascience.com/understanding-the-bias-variance-tradeoff-165e6942b229

Allibhai, E. (2022, June 21). *Hold-out vs. Cross-validation in Machine Learning - Eijaz Allibhai*. Medium. https://medium.com/@eijaz/holdout-vs-cross-validation-in-machine-learning-7637112d3f8f

Prasad, A. (2022, January 6). Regression Trees | Decision Tree for Regression | Machine Learning. Medium. Retrieved September 23, 2022, from https://medium.com/analytics-vidhya/regression-trees-decision-tree-for-regression-machine-learning-e4d7525d8047

Random forests - classification description. (n.d.). Retrieved September 23, 2022, from https://www.stat.berkeley.edu/%7Ebreiman/RandomForests/cc_home.htm

Supervised vs. Unsupervised Learning: What's the Difference? (2021, March 12). IBM. Retrieved September 23, 2022, from https://www.ibm.com/cloud/blog/supervised-vs-unsupervised-learning

Gradient boosting. (2022, September 1). Wikipedia. Retrieved September 23, 2022, from https://en.wikipedia.org/wiki/Gradient_boosting