Assignment Cover Sheet

Unit Code: 4483

Semester/Year: Semester 1 2023

Lecturer/Tutor Name: Dr Girija Chetty / Linda Ma

Workshop /Tutorial Day & Time: Wednesday 4:30 PM

Assignment Number and Title: ST1 Capstone Project

Word Count: n/a

Date & Time Submitted:

By submitting this assignment I/We declare that this assignment is solely my/our own work, except where due acknowledgements are made. I/We acknowledge that the assessor of this assignment may provide a copy of this assignment to another member of the University, and/or to a plagiarism checking service whilst assessing this assignment. I/We have read and understood the University policies in respect of Student Academic Honesty.

Name: Adam Seaton

Table of Contents

Assignment Cover Sheet 1
Table of Contents 2
Introduction 3
Stage 1: Algorithm Design 4
- **Exploratory Data Analysis and Visualization** 4
- **Data Preparation and Predictive Model Development** 14

Stage 2: Algorithm Implementation 22
- **Software Testing** 22

Conclusion 23
- **References** 23

Appendix | Logbook/Journal 24
Appendix | Python Code 26

Introduction

This report details the design process and methods used to complete my Software Technology Capstone Project. The outline of this report is as follows: an introduction to the report and the dataset; a stage 1 section covering algorithm development, including: exploratory data analysis, data visualization, as well as predictive model preparation and development; a stage 2 section covering the algorithm implementation as a python Tkinter program; and finally, a conclusion and an appendix with a logbook detailing my weekly progress with the project. The main objective of this project is to gain experience in data analysis, visualization, prediction, and deployment using some of the following packages: Pandas, NumPy, Matplotlib, Seaborn, Plotly, Scikit-learn, PyTorch, Keras, TkInter.

The dataset I have chosen for this project is a snapshot of Melbourne housing information available on Kaggle that uses publicly available data sourced from Domain.com.au, a popular real-estate marketing website. Details of the houses include: location, property type, land size, number of total rooms, bathrooms, bedrooms, carports, and much more. The data is from houses sold in Melbourne from January 2016 to September 2017.

Utilizing this dataset, it was my aim to create a Tkinter application wherein a user would be able to input house information, such as: rooms, bedrooms, bathrooms, car ports, land size, building area, location, and then the tool would predict the house price. A tool that could accurately predict house prices would be of benefit to a wide range of stakeholders; including, buyers, sellers, real-estate agencies, and banks.

One final note: this dataset required regression to build a predictive model rather than classification, therefore this report does not contain a section where I have predicted different classes as it was not necessary.

Methodology

The methodology used for developing the software tool involved 2 stages as outlined below:

1. Design and development of decision support algorithms based on exploratory data analysis and predictive analytics, for identifying the best performing algorithm for solving a real world problem.

2. Implementation of the best performing algorithm as a desktop Tkinter software tool.

Stage 1: Algorithm Design

Exploratory Data Analysis and Visualization

The first step in creating a predictive model for house prices is to first understand the data we are working with using exploratory data analysis (EDA) techniques. Google Colab was chosen as the development environment on the recommendation of the lecturer and tutor for its use of cloud processing. Before the EDA can begin, we must conduct the preliminary steps of importing the required python modules:

```
[ ] #Mount Google Drive
    from google.colab import drive
    drive.mount("/content/drive")

    #Import Required Packages for EDA
    import os
    import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt
    import seaborn as sns
    import missingno as msno
    import plotly.graph_objects as go
    import plotly.express as px
    %matplotlib inline
    import warnings
    warnings.filterwarnings('ignore')

    #Import Required Packages for Regression Prediction
    from sklearn.preprocessing import StandardScaler
    from sklearn.model_selection import train_test_split
    from sklearn.model_selection import KFold
    from sklearn.model_selection import cross_val_score
    from sklearn.model_selection import GridSearchCV
    from sklearn.linear_model import LinearRegression
    from sklearn.linear_model import Lasso
    from sklearn.linear_model import ElasticNet
    from sklearn.tree import DecisionTreeRegressor
    from sklearn.neighbors import KNeighborsRegressor
    from sklearn.svm import SVR
    from sklearn.pipeline import Pipeline
    from sklearn.ensemble import RandomForestRegressor
    from sklearn.ensemble import GradientBoostingRegressor
    from sklearn.ensemble import ExtraTreesRegressor
    from sklearn.ensemble import AdaBoostRegressor
    from sklearn.metrics import mean_squared_error

    #Read the dataset
    df = pd.read_csv("/content/drive/MyDrive/Software Tech/melb_data.csv")
```

The EDA can now start with understanding the basic description of the data:

```
[ ] #First5 rows
    df.head()
```

	Suburb	Address	Rooms	Type	Price	Method	SellerG	Date	Distance	Postcode	...	Bathroom	Car	Landsize	BuildingArea	YearBuilt	CouncilArea	Lattitude	Longtitude	Regionname	Propertycount
0	Abbotsford	85 Turner St	2	h	1480000.0	S	Biggin	3/12/2016	2.5	3067.0	...	1.0	1.0	202.0	NaN	NaN	Yarra	-37.7996	144.9984	Northern Metropolitan	4019.0
1	Abbotsford	25 Bloomburg St	2	h	1035000.0	S	Biggin	4/02/2016	2.5	3067.0	...	1.0	0.0	156.0	79.0	1900.0	Yarra	-37.8079	144.9934	Northern Metropolitan	4019.0
2	Abbotsford	5 Charles St	3	h	1465000.0	SP	Biggin	4/03/2017	2.5	3067.0	...	2.0	0.0	134.0	150.0	1900.0	Yarra	-37.8093	144.9944	Northern Metropolitan	4019.0
3	Abbotsford	40 Federation La	3	h	850000.0	PI	Biggin	4/03/2017	2.5	3067.0	...	2.0	1.0	94.0	NaN	NaN	Yarra	-37.7969	144.9969	Northern Metropolitan	4019.0
4	Abbotsford	55a Park St	4	h	1600000.0	VB	Nelson	4/06/2016	2.5	3067.0	...	1.0	2.0	120.0	142.0	2014.0	Yarra	-37.8072	144.9941	Northern Metropolitan	4019.0

5 rows × 21 columns

```
[ ] #Last 5 rows
    df.tail()
```

	Suburb	Address	Rooms	Type	Price	Method	SellerG	Date	Distance	Postcode	...	Bathroom	Car	Landsize	BuildingArea	YearBuilt	CouncilArea	Lattitude	Longtitude	Regionname	Propertycount
13575	Wheelers Hill	12 Strada Cr	4	h	1245000.0	S	Barry	26/08/2017	16.7	3150.0	...	2.0	2.0	652.0	NaN	1981.0	NaN	-37.90562	145.16761	South-Eastern Metropolitan	7392.0
13576	Williamstown	77 Merrett Dr	3	h	1031000.0	SP	Williams	26/08/2017	6.8	3016.0	...	2.0	2.0	333.0	133.0	1995.0	NaN	-37.85927	144.87904	Western Metropolitan	6380.0
13577	Williamstown	83 Power St	3	h	1170000.0	S	Raine	26/08/2017	6.8	3016.0	...	2.0	4.0	436.0	NaN	1997.0	NaN	-37.85274	144.88738	Western Metropolitan	6380.0
13578	Williamstown	96 Verdon St	4	h	2500000.0	PI	Sweeney	26/08/2017	6.8	3016.0	...	1.0	5.0	866.0	157.0	1920.0	NaN	-37.85908	144.89299	Western Metropolitan	6380.0
13579	Yarraville	6 Agnes St	4	h	1285000.0	SP	Village	26/08/2017	6.3	3013.0	...	1.0	1.0	362.0	112.0	1920.0	NaN	-37.81188	144.88449	Western Metropolitan	6543.0

5 rows × 21 columns

```
[ ] #Rows and columns data shape(attributes & samples)
    df.shape
```

```
(13580, 21)
```

```
[ ] #Name of the attributes
    df.columns
```

```
Index(['Suburb', 'Address', 'Rooms', 'Type', 'Price', 'Method', 'SellerG',
       'Date', 'Distance', 'Postcode', 'Bedroom2', 'Bathroom', 'Car',
       'Landsize', 'BuildingArea', 'YearBuilt', 'CouncilArea', 'Lattitude',
       'Longtitude', 'Regionname', 'Propertycount'],
      dtype='object')
```

```
[ ] #Number of unique values for each attribute
    df.nunique()
```

```
Suburb             314
Address          13378
Rooms                9
Type                 3
Price             2204
Method               5
SellerG            268
Date                58
Distance           202
Postcode           198
Bedroom2            12
Bathroom             9
Car                 11
Landsize          1448
BuildingArea       602
YearBuilt          144
CouncilArea         33
Lattitude         6503
Longtitude        7063
Regionname           8
Propertycount      311
dtype: int64
```

```
[ ] #Complete info about data frame
    df.info()
```

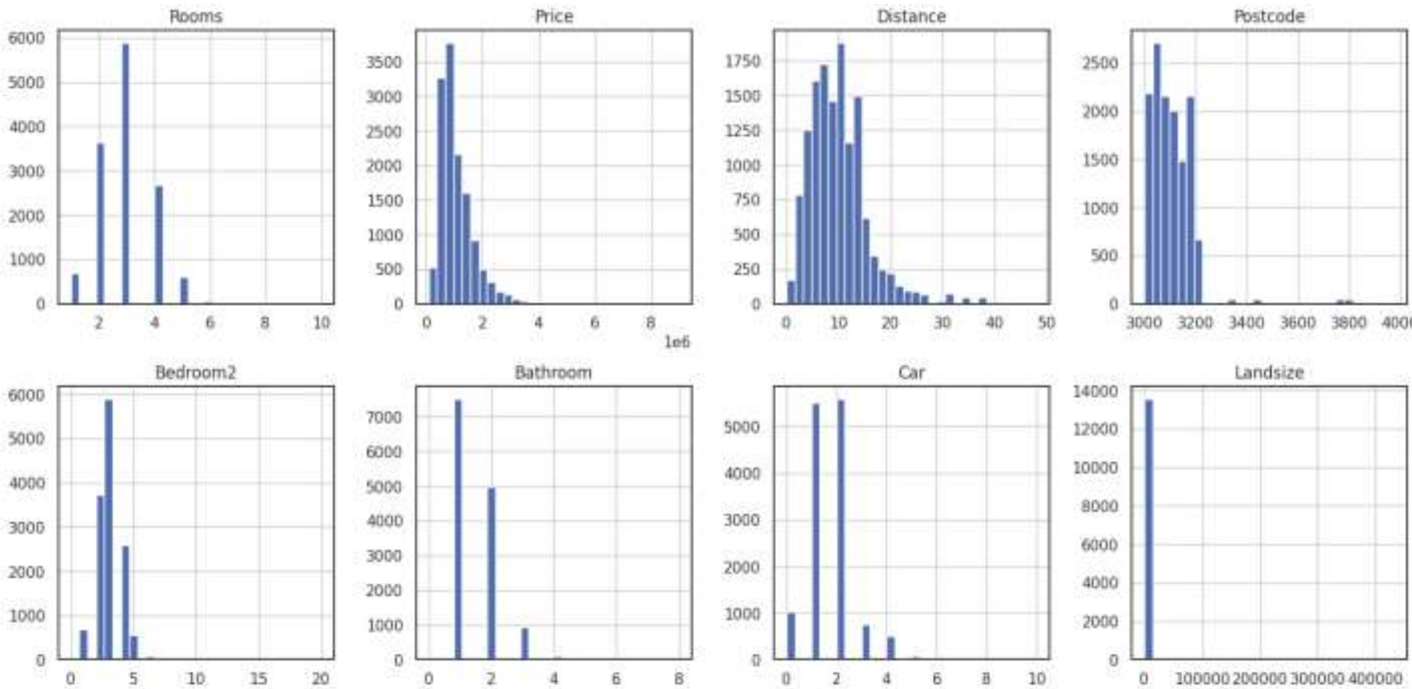
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13580 entries, 0 to 13579
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Suburb         13580 non-null  object
 1   Address        13580 non-null  object
 2   Rooms          13580 non-null  int64
 3   Type           13580 non-null  object
 4   Price          13580 non-null  float64
 5   Method         13580 non-null  object
 6   SellerG        13580 non-null  object
 7   Date           13580 non-null  object
 8   Distance       13580 non-null  float64
 9   Postcode       13580 non-null  float64
 10  Bedroom2       13580 non-null  float64
 11  Bathroom       13580 non-null  float64
 12  Car            13518 non-null  float64
 13  Landsize       13580 non-null  float64
 14  BuildingArea   7130 non-null   float64
 15  YearBuilt      8205 non-null   float64
 16  CouncilArea    12211 non-null  object
 17  Lattitude      13580 non-null  float64
 18  Longtitude     13580 non-null  float64
 19  Regionname     13580 non-null  object
 20  Propertycount  13580 non-null  float64
dtypes: float64(12), int64(1), object(8)
memory usage: 2.2+ MB
```

```
[ ] #Description of the data (mean, standard deviation etc.)
    df.describe()
```

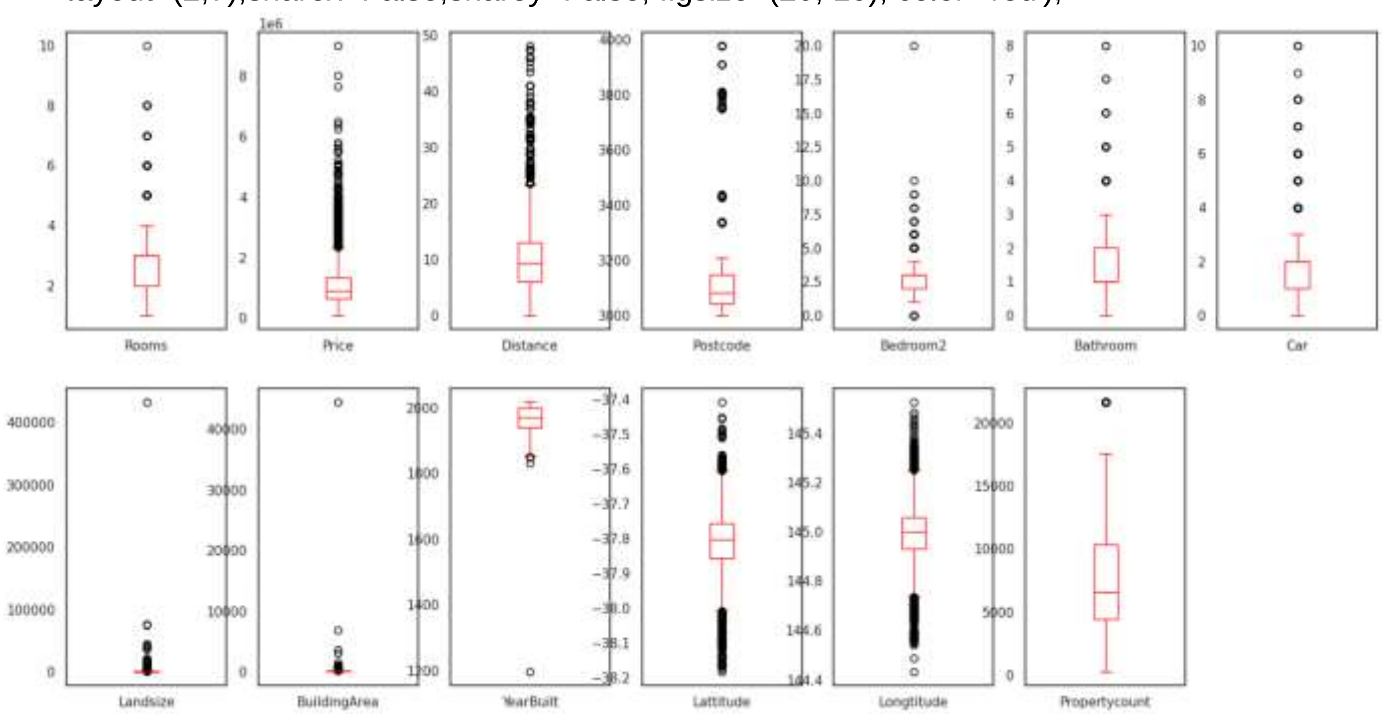
	Rooms	Price	Distance	Postcode	Bedroom2	Bathroom	Car	Landsize	BuildingArea	YearBuilt	Lattitude	Longtitude
count	13580.000000	1.358000e+04	13580.000000	13580.000000	13580.000000	13580.000000	13518.000000	13580.000000	7130.000000	8205.000000	13580.000000	13580.000000
mean	2.937997	1.075684e+06	10.137776	3105.301915	2.914728	1.534242	1.610075	558.416127	151.967650	1964.684217	-37.809203	144.995216
std	0.955748	6.393107e+05	5.868725	90.676964	0.965921	0.691712	0.962634	3990.669241	541.014538	37.273762	0.079260	0.103916
min	1.000000	8.500000e+04	0.000000	3000.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1196.000000	-38.182550	144.431810
25%	2.000000	6.500000e+05	6.100000	3044.000000	2.000000	1.000000	1.000000	177.000000	93.000000	1940.000000	-37.856822	144.929600
50%	3.000000	9.030000e+05	9.200000	3084.000000	3.000000	1.000000	2.000000	440.000000	126.000000	1970.000000	-37.802355	145.000100
75%	3.000000	1.330000e+06	13.000000	3148.000000	3.000000	2.000000	2.000000	651.000000	174.000000	1999.000000	-37.756400	145.058305
max	10.000000	9.000000e+06	48.100000	3977.000000	20.000000	8.000000	10.000000	433014.000000	44515.000000	2018.000000	-37.408530	145.526350

Now that we have a basic understanding of the data we can begin to visualize it to gain a deeper understanding:

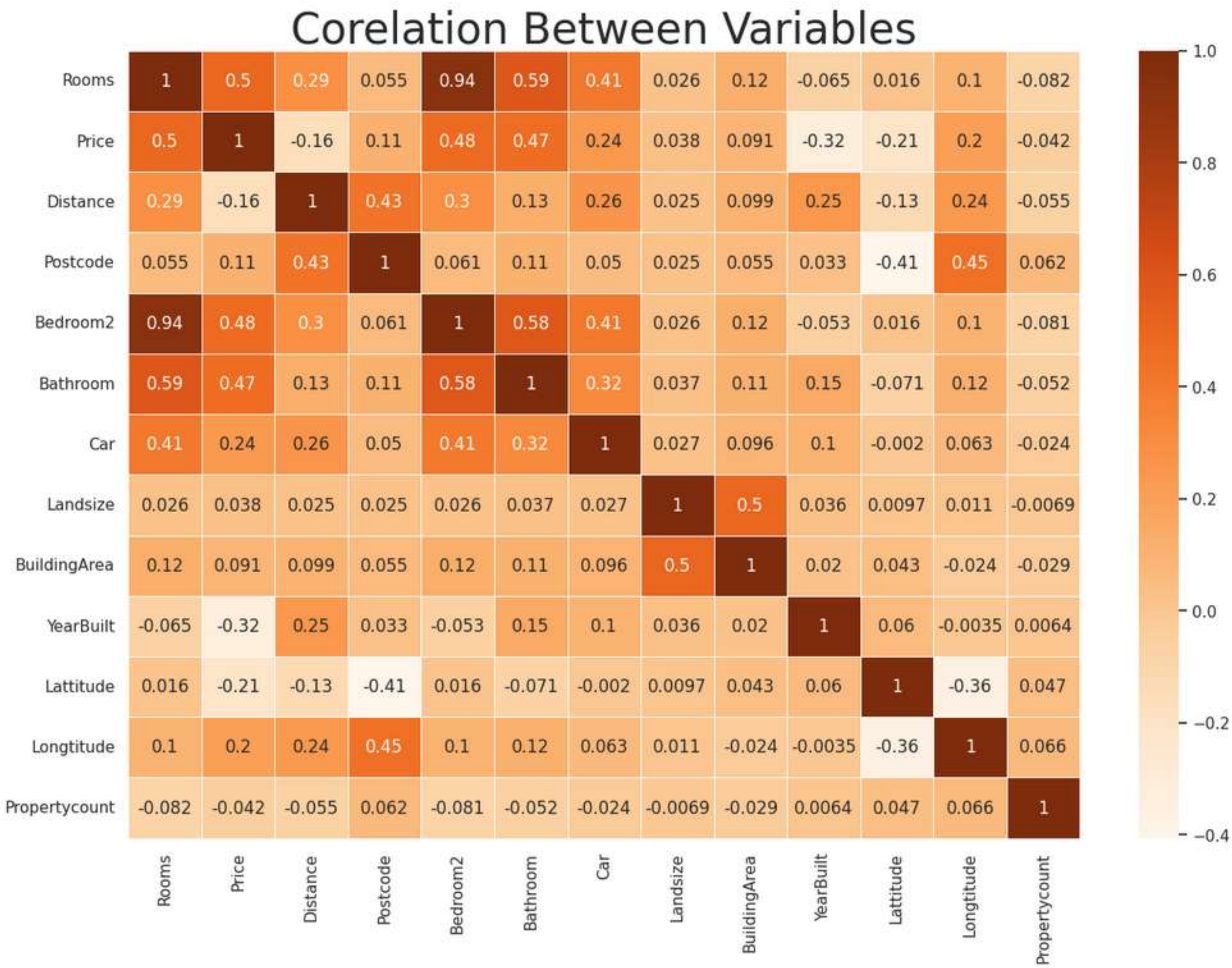
```
# Visualizing data distribution
fig = plt.figure(figsize =(18,18))
ax=fig.gca()
df.hist(ax=ax,bins =30)
```




```
# Visualizing outliers
df.plot(kind='box', subplots=True,
        layout=(2,7),sharex=False,sharey=False, figsize=(20, 10), color='red');
```



```
# Visualizing the correlation between variables
sns.set(style="white")
plt.rcParams['figure.figsize'] = (15, 10)
sns.heatmap(df.corr(), annot = True, linewidths=.5, cmap="Oranges")
plt.title('Corelation Between Variables', fontsize = 30)
plt.show()
```

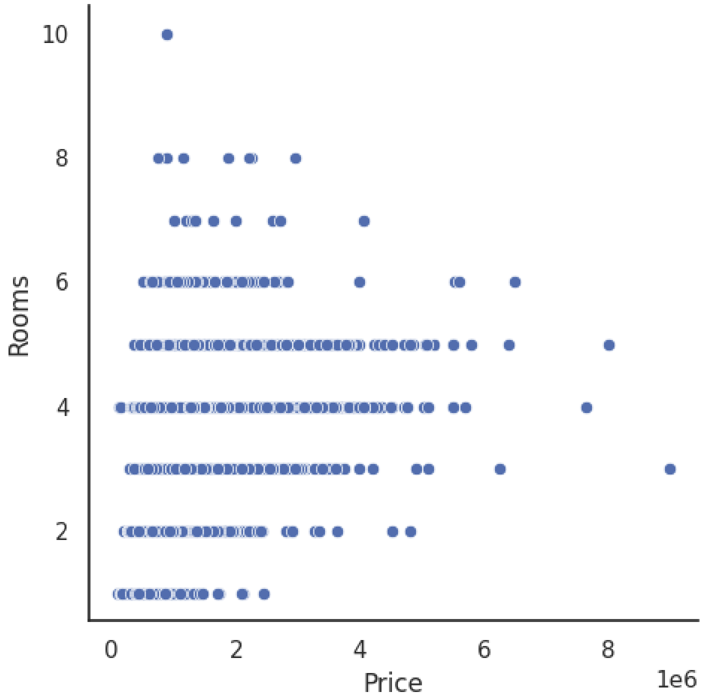


	Rooms	Price	Distance	Postcode	Bedroom2	Bathroom	Car	Landsize	BuildingArea	YearBuilt	Lattitude	Longtitude	Propertycount
Rooms	1	0.5	0.29	0.055	0.94	0.59	0.41	0.026	0.12	-0.065	0.016	0.1	-0.082
Price	0.5	1	-0.16	0.11	0.48	0.47	0.24	0.038	0.091	-0.32	-0.21	0.2	-0.042
Distance	0.29	-0.16	1	0.43	0.3	0.13	0.26	0.025	0.099	0.25	-0.13	0.24	-0.055
Postcode	0.055	0.11	0.43	1	0.061	0.11	0.05	0.025	0.055	0.033	-0.41	0.45	0.062
Bedroom2	0.94	0.48	0.3	0.061	1	0.58	0.41	0.026	0.12	-0.053	0.016	0.1	-0.081
Bathroom	0.59	0.47	0.13	0.11	0.58	1	0.32	0.037	0.11	0.15	-0.071	0.12	-0.052
Car	0.41	0.24	0.26	0.05	0.41	0.32	1	0.027	0.096	0.1	-0.002	0.063	-0.024
Landsize	0.026	0.038	0.025	0.025	0.026	0.037	0.027	1	0.5	0.036	0.0097	0.011	-0.0069
BuildingArea	0.12	0.091	0.099	0.055	0.12	0.11	0.096	0.5	1	0.02	0.043	-0.024	-0.029
YearBuilt	-0.065	-0.32	0.25	0.033	-0.053	0.15	0.1	0.036	0.02	1	0.06	-0.0035	0.0064
Lattitude	0.016	-0.21	-0.13	-0.41	0.016	-0.071	-0.002	0.0097	0.043	0.06	1	-0.36	0.047
Longtitude	0.1	0.2	0.24	0.45	0.1	0.12	0.063	0.011	-0.024	-0.0035	-0.36	1	0.066
Propertycount	-0.082	-0.042	-0.055	0.062	-0.081	-0.052	-0.024	-0.0069	-0.029	0.0064	0.047	0.066	1

Relational plots

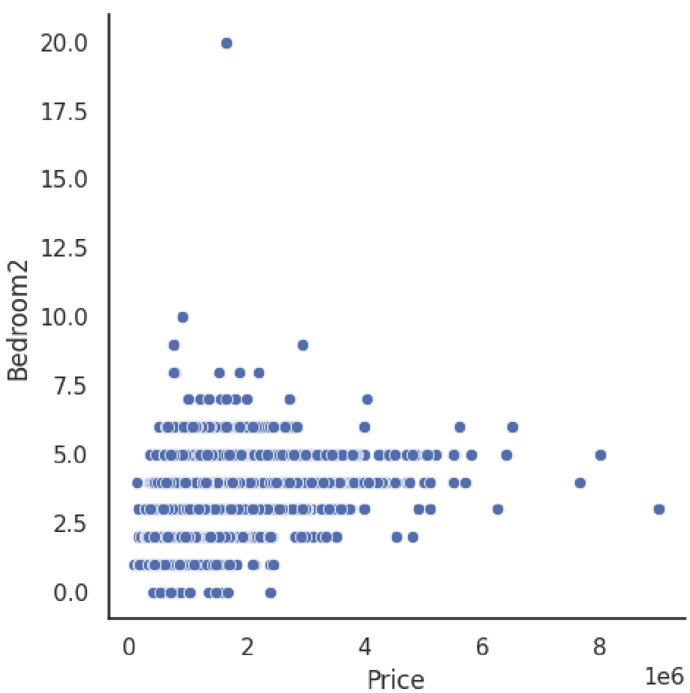
```
[ ] sns.relplot(x="Price", y="Rooms", data=df)
```

<seaborn.axisgrid.FacetGrid at 0x7fab62ece950>



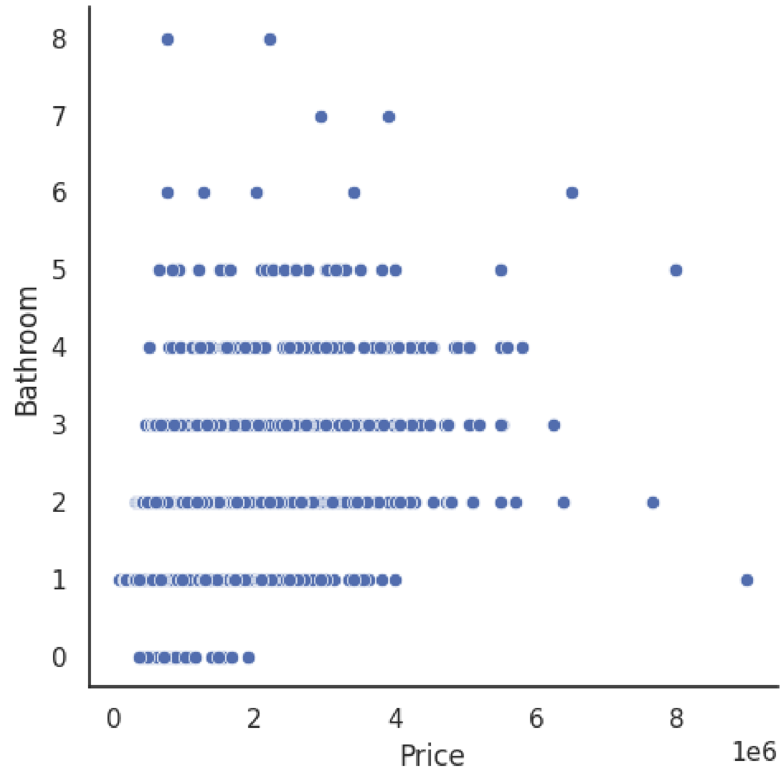

```
[ ] sns.relplot(x="Price", y="Bedroom2", data=df)
```

<seaborn.axisgrid.FacetGrid at 0x7fab6a4f5c60>



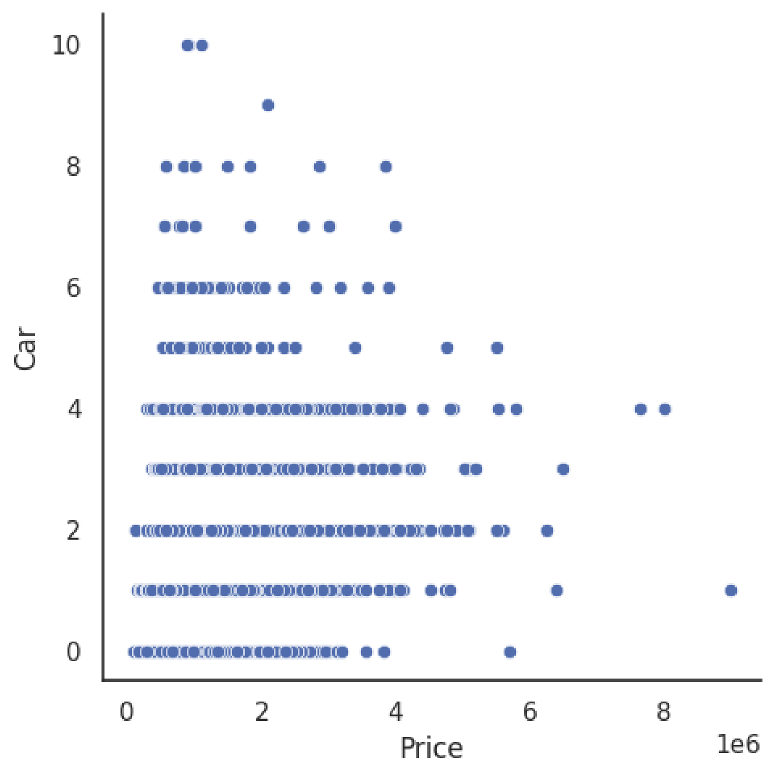
```
[ ]  sns.relplot(x="Price", y="Bathroom", data=df)
```

<seaborn.axisgrid.FacetGrid at 0x7fab6850be20>



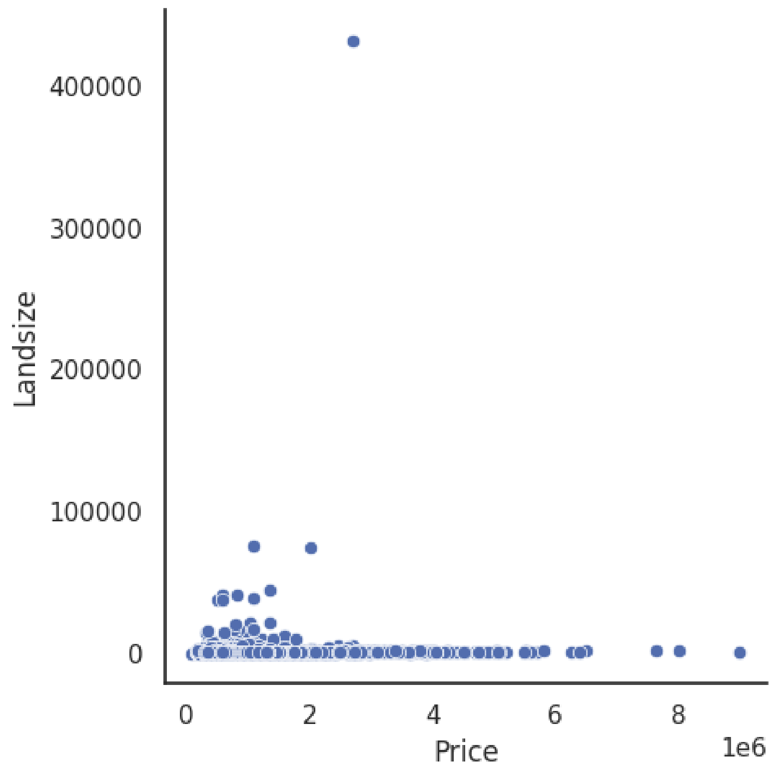

```
[ ]  sns.relplot(x="Price", y="Car", data=df)
```

<seaborn.axisgrid.FacetGrid at 0x7fab684124a0>



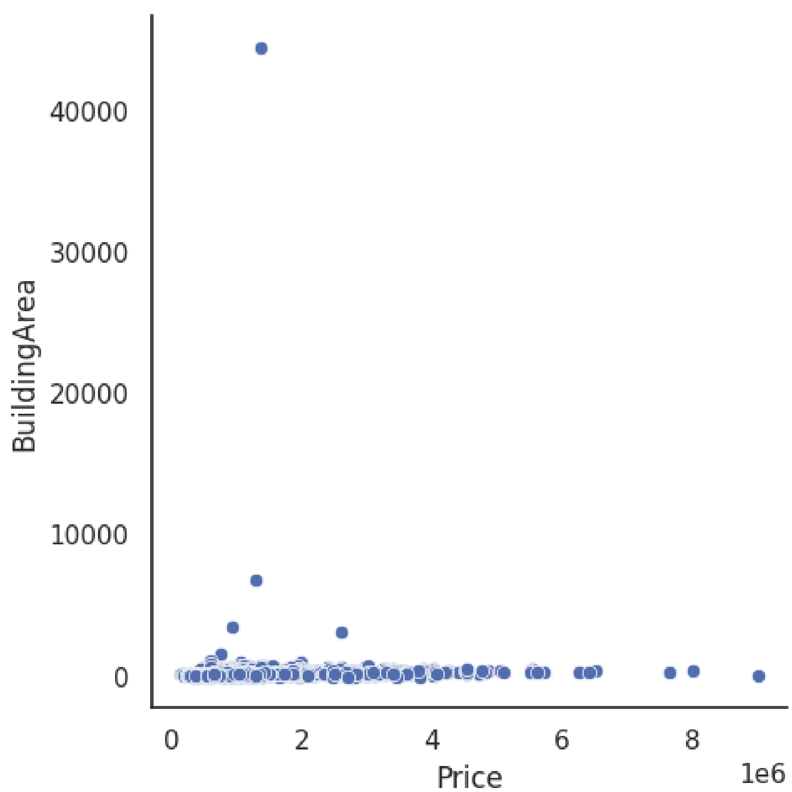
```
[ ] sns.relplot(x="Price", y="Landsize", data=df)
```

```
<seaborn.axisgrid.FacetGrid at 0x7fab62f9b4f0>
```

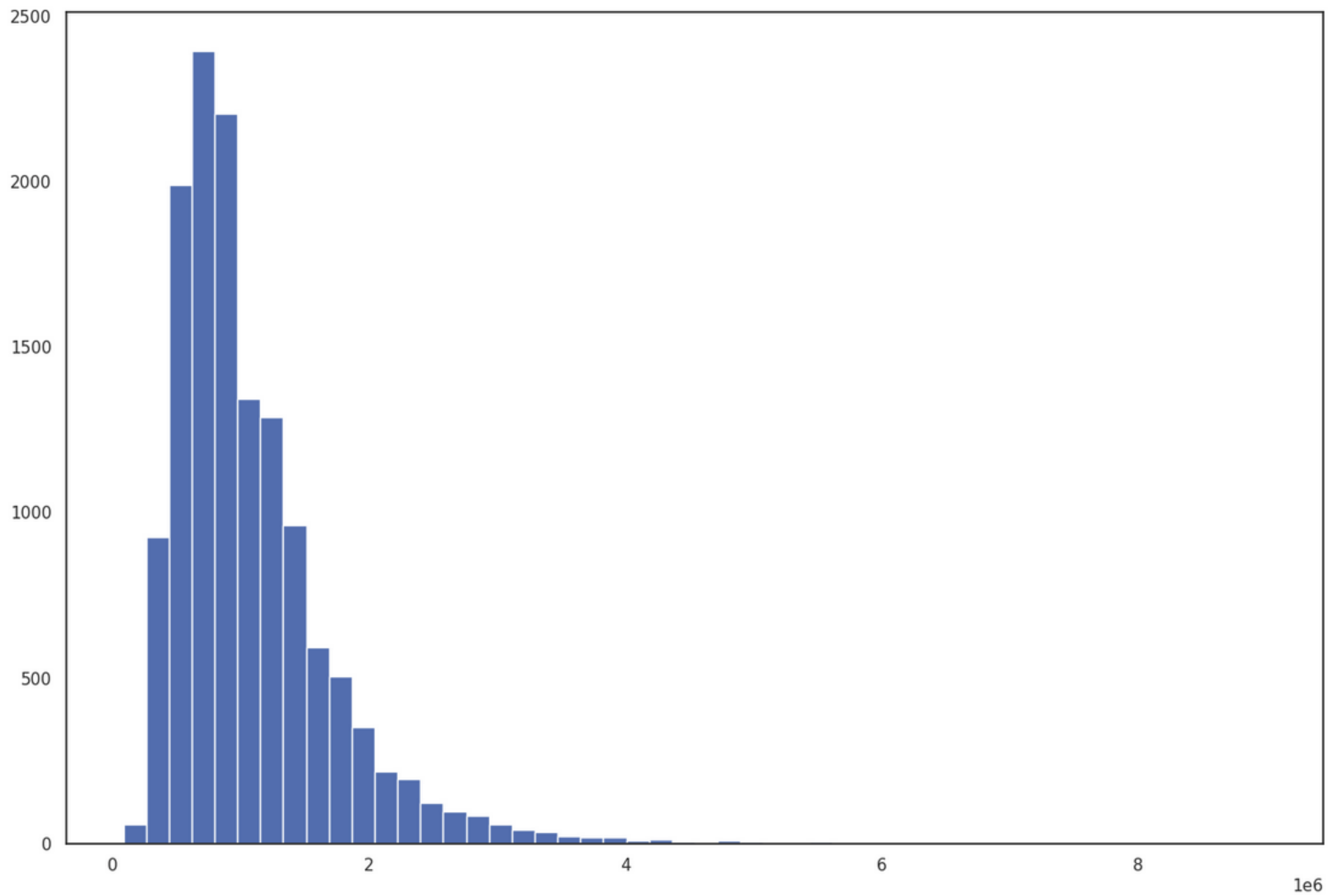



```
[ ] sns.relplot(x="Price", y="BuildingArea", data=df)
```

```
<seaborn.axisgrid.FacetGrid at 0x7fab62ca94b0>
```

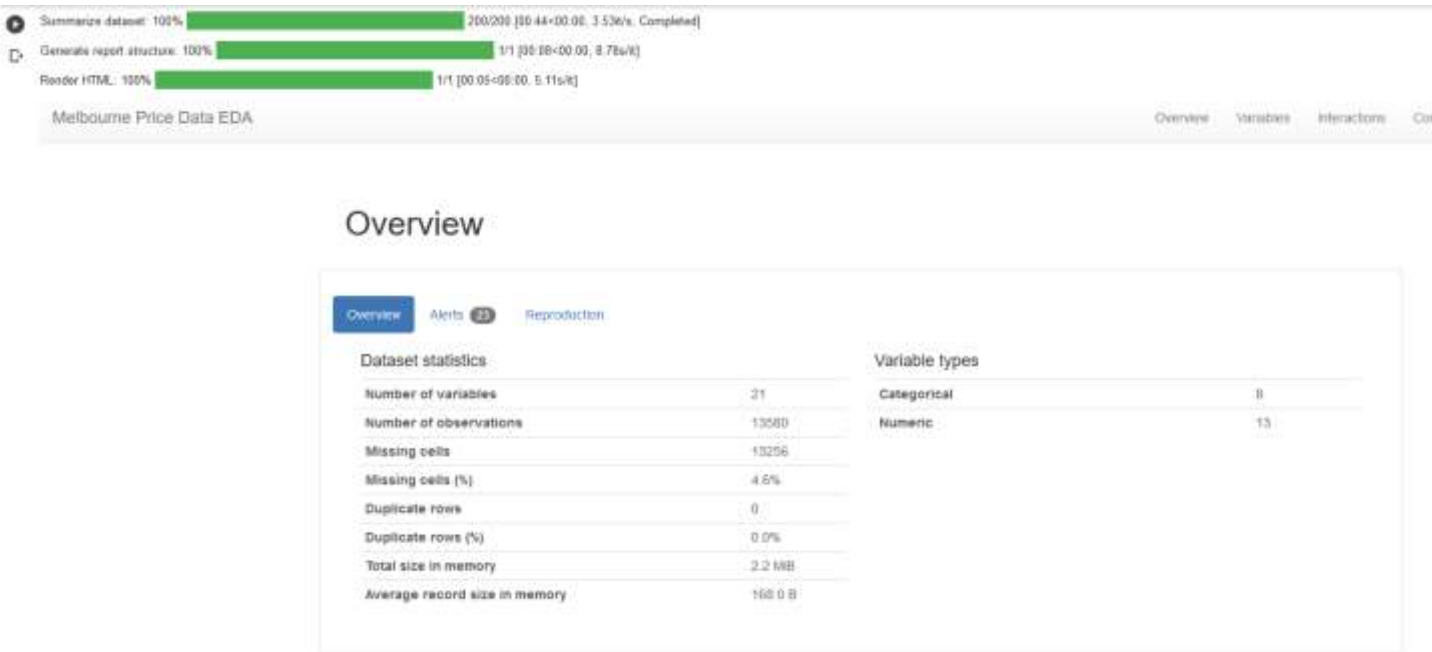


plt.hist(df["Price"], bins=50)



Pandas profiling report
!pip install https://github.com/pandas-profiling/pandas-profiling/archive/master.zip

```
[ ] #obtain full profiler report
    #restart kernel
    #re-run import libraries and data
    import pandas as pd
    import numpy as np
    from pandas_profiling import ProfileReport
    profile = ProfileReport(df,title="Melbourne Price Data EDA",
                            html={'style':{'full_width':True}})
    profile.to_notebook_iframe()
```



Summarize dataset: 100% 200/200 [00:44<00:00, 3.53it/s, Completed]
Generate report structure: 100% 1/1 [00:08<00:00, 8.78s/it]
Render HTML: 100% 1/1 [00:05<00:00, 5.11s/it]

Melbourne Price Data EDA

Overview Variables Interactions Corr

Overview

Overview Alerts 23 Reproduction

Dataset statistics

Number of variables	21
Number of observations	13580
Missing cells	13256
Missing cells (%)	4.6%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	2.2 MiB
Average record size in memory	168.0 B

Variable types

Categorical	8
Numeric	13

Data Preparation and Predictive Model Development

We can now move on to producing a predictive model. First, a subset of the data with the attributes we want to work with is created. This subset is then processed to remove any null values and outliers.

```
[ ] #Define subset of data we want to work with
    df2 = df[['Rooms', 'Bedroom2', 'Bathroom', 'Car', 'Landsize', 'BuildingArea','Lattitude', 'Longtitude', 'Price']]

    #Fill null values for car ports with 0
    df2['Car'] = df2['Car'].fillna(0)
    #Drop records with BuildingArea null values
    df2 = df2.dropna()

    #Show results of dropping null values
    df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 7130 entries, 1 to 13579
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Rooms         7130 non-null   int64
 1   Bedroom2      7130 non-null   float64
 2   Bathroom      7130 non-null   float64
 3   Car           7130 non-null   float64
 4   Landsize      7130 non-null   float64
 5   BuildingArea  7130 non-null   float64
 6   Lattitude     7130 non-null   float64
 7   Longtitude    7130 non-null   float64
 8   Price         7130 non-null   float64
dtypes: float64(8), int64(1)
memory usage: 557.0 KB
```

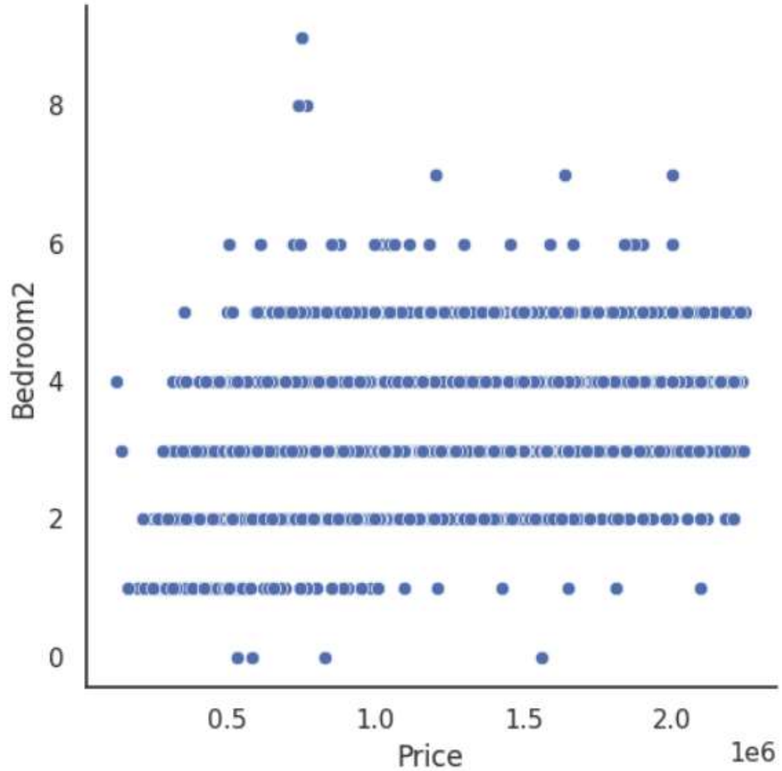
```
[ ] #Manually remove obvious outliers
    df2 = df2[df2['BuildingArea'] < 500]
    df2 = df2[df2['Bedroom2'] < 11]
    df2 = df2[df2['Landsize'] < 1250]
    df2 = df2[df2['Price'] < 2250000]

    #Show results of removing outliers
    df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6494 entries, 1 to 13579
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   Rooms         6494 non-null   int64  
 1   Bedroom2      6494 non-null   float64
 2   Bathroom      6494 non-null   float64
 3   Car           6494 non-null   float64
 4   Landsize      6494 non-null   float64
 5   BuildingArea  6494 non-null   float64
 6   Lattitude     6494 non-null   float64
 7   Longtitude    6494 non-null   float64
 8   Price         6494 non-null   float64
dtypes: float64(8), int64(1)
memory usage: 507.3 KB
```

Visualize the results of removing outliers

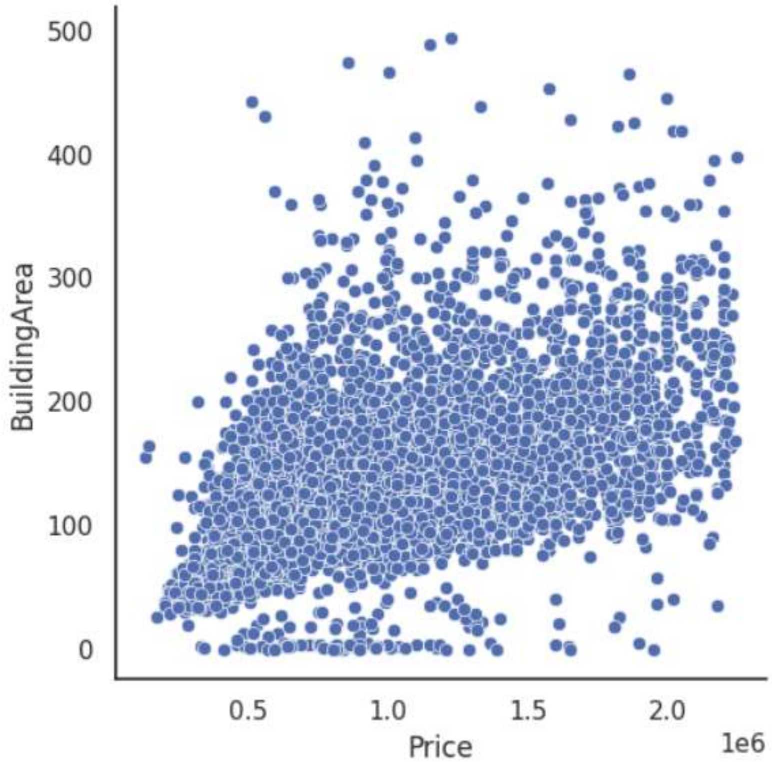
```
[ ]  sns.relplot(x="Price", y="Bedroom2", data=df2)
```

<seaborn.axisgrid.FacetGrid at 0x7fab625f2920>



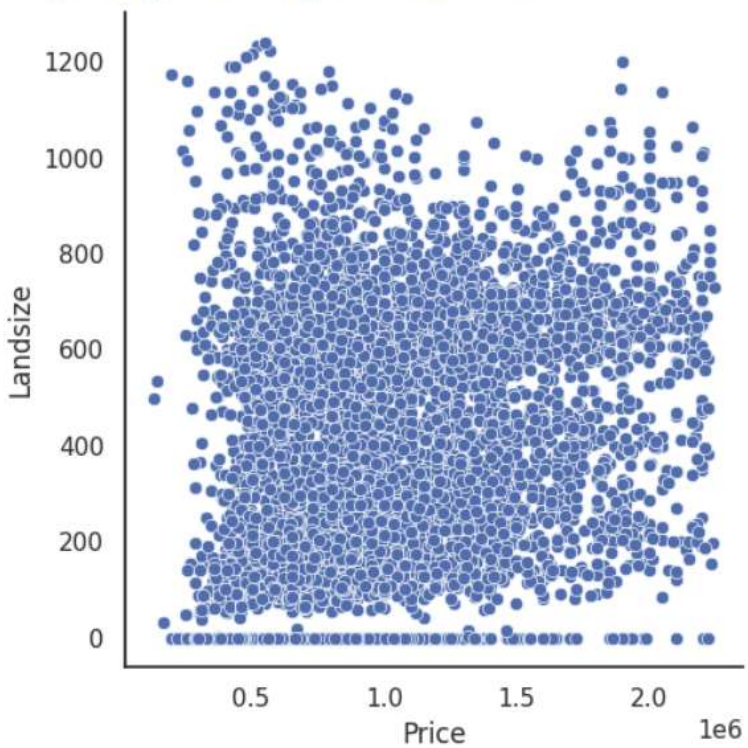
```
sns.relplot(x="Price", y="BuildingArea", data=df2)
```

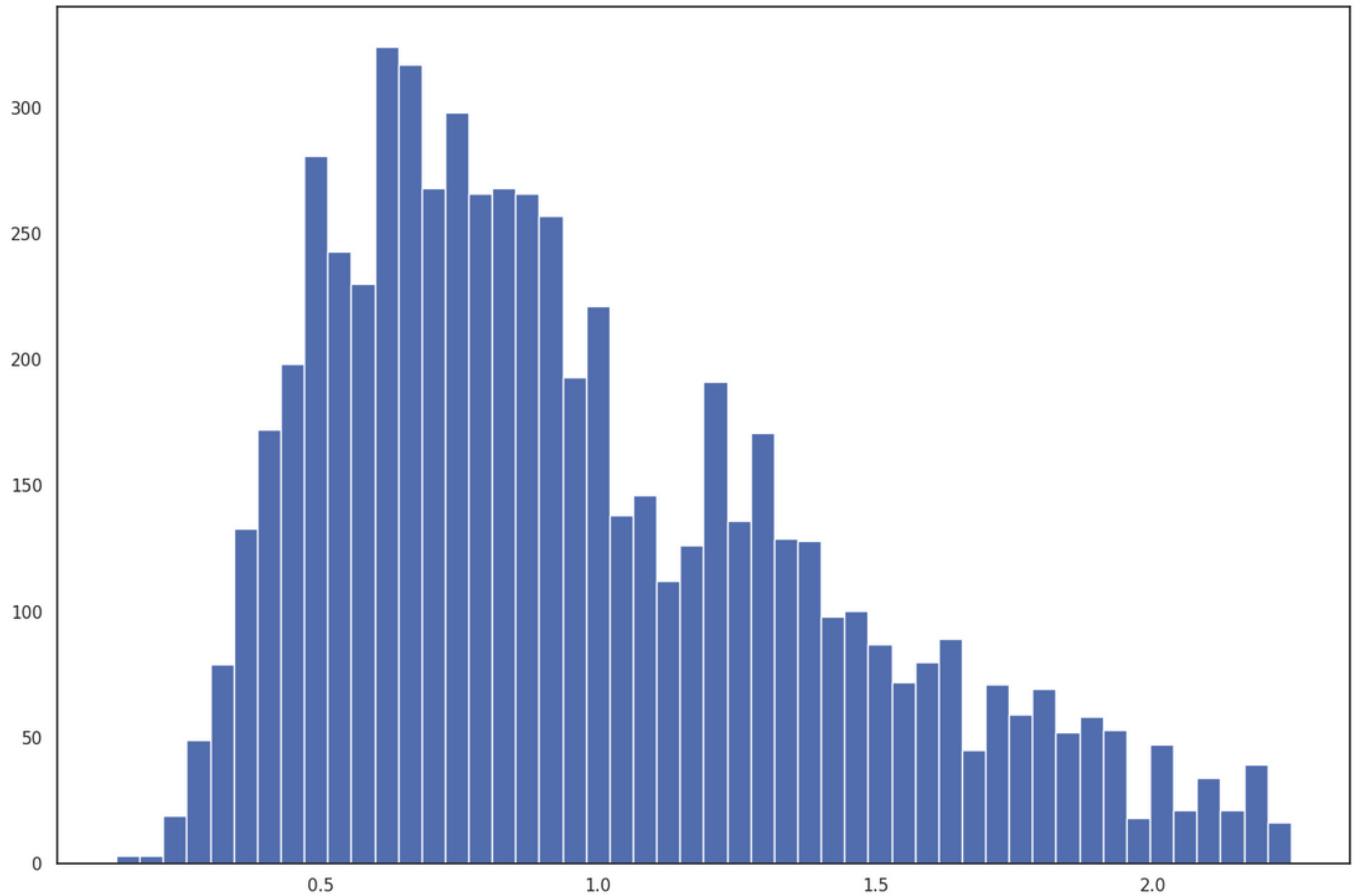
<seaborn.axisgrid.FacetGrid at 0x7fab625f31c0>




```
[ ] sns.relplot(x="Price", y="Landsize", data=df2)
```

<seaborn.axisgrid.FacetGrid at 0x7fab6246cdf0>





Now that the data is prepared, we can begin the process of developing a model. The data is first split into a training set and testing set (80% and 20% split) and then each algorithm is fitted with the training set and evaluated against the test set. The best performing algorithm is then selected to be the predictive model.

```
[ ] #Create X dataset
    X = df2[['Rooms', 'Bedroom2', 'Bathroom', 'Car', 'Landsize', 'BuildingArea','Lattitude', 'Longtitude']]
    #Create Y dataset with the variable we will be predicting for
    y = df2['Price']
```

```
[ ] X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=45)
```

```
#Linear Regression
lr = LinearRegression()
lr.fit(X_train, y_train)
print(f'Linear Regression: {lr.score(X_test, y_test)}')

#Random Forest Regression
rfr = RandomForestRegressor()
rfr.fit(X_train, y_train)
print(f'Random Forest Regression: {rfr.score(X_test, y_test)}')

#Gradient Boosting Regression
gbr = GradientBoostingRegressor()
gbr.fit(X_train, y_train)
print(f'Gradient Boosting Regression: {gbr.score(X_test, y_test)}')

#Ada Boost Regression
abr = AdaBoostRegressor()
abr.fit(X_train, y_train)
print(f'Ada Boost Regression: {abr.score(X_test, y_test)}')

#Extra Trees Regression
etr = ExtraTreesRegressor()
etr.fit(X_train, y_train)
print(f'Extra Trees Regression: {etr.score(X_test, y_test)}')

#K-nearest Neighbbour Regression
knn = KNeighborsRegressor()
knn.fit(X_train, y_train)
print(f'Linear Regression: {knn.score(X_test, y_test)}')

#Decision Tree Regression
cart = DecisionTreeRegressor()
cart.fit(X_train, y_train)
print(f'K-nearest Neighbbour Regression: {cart.score(X_test, y_test)}')

#Support Vector Regression
svr = SVR()
svr.fit(X_train, y_train)
print(f'Support Vector Regression: {svr.score(X_test, y_test)}')

#LASSO
lasso = Lasso()
lasso.fit(X_train, y_train)
print(f'LASSO: {lasso.score(X_test, y_test)}')

#ElasticNet
en = ElasticNet()
en.fit(X_train, y_train)
print(f'ElasticNet: {en.score(X_test, y_test)}')
```

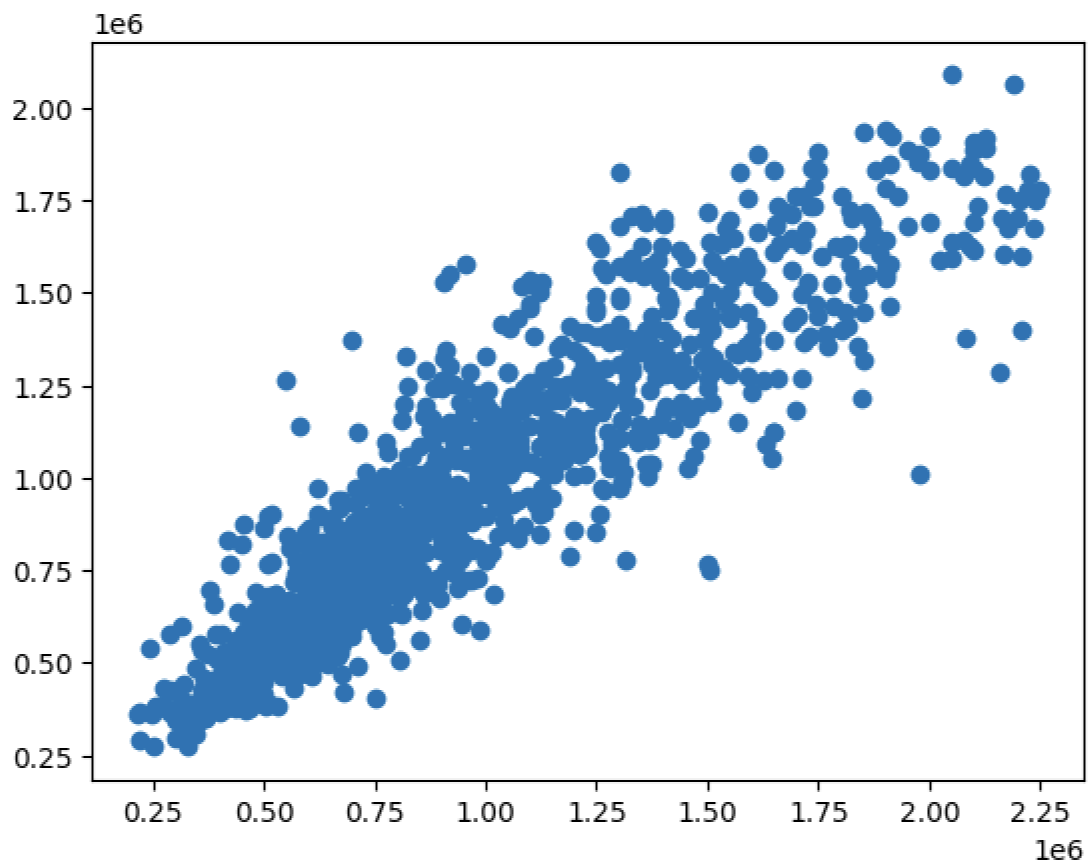
```
Linear Regression: 0.38530638265189177
Random Forest Regression: 0.8270414022361864
Gradient Boosting Regression: 0.7799629120023629
Ada Boost Regression: 0.5203509910014112
Extra Trees Regression: 0.8208666563018694
Linear Regression: 0.2992054634116156
K-nearest Neighbbour Regression: 0.6548156789755422
Support Vector Regression: -0.03401887997591668
LASSO: 0.3853042541726427
ElasticNet: 0.3109764714865888
```

```
[ ] #Select best performing algorithm
    model = RandomForestRegressor()
    model.fit(X_train, y_train)
    print(model.score(X_test, y_test))
```

```
0.8268668515665556
```

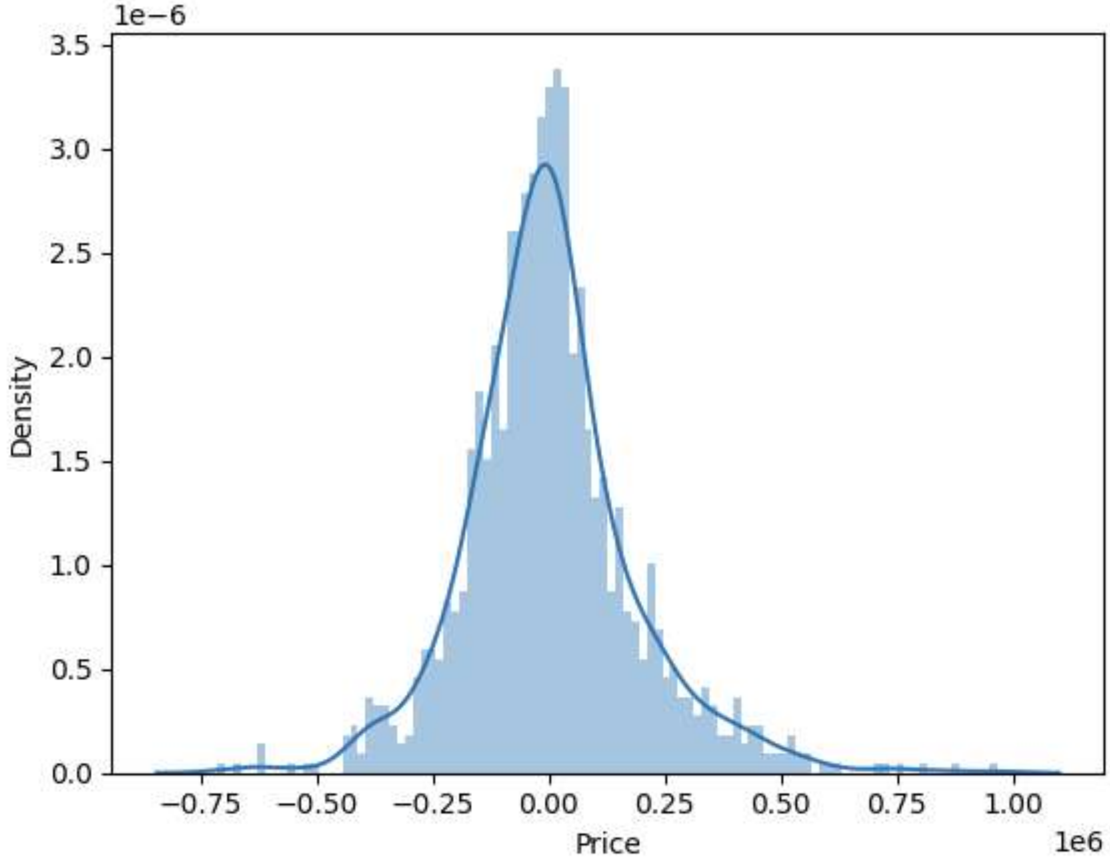
```
[ ] #Visualize predictions
    predictions = model.predict(X_test)
    plt.scatter(y_test, predictions)
```

```
<matplotlib.collections.PathCollection at 0x7f62c9163f70>
```



```
[ ] sns.distplot((y_test - predictions), bins=100)
```

<Axes: xlabel='Price', ylabel='Density'>




```
[ ] #Best Model export for deployment
    import pickle
    #Save the model to disk
    model_filename = 'best_regr_model.h5'
    pickle.dump(model, open(model_filename, 'wb'))
```

```
0.8251466537747794
```

```
[ ] #Check by Reloading saved model from disk using load function of pickle
    with open('best_regr_model.h5','rb') as file:
        loaded_model = pickle.load(file)
    #Validate the R sqaured value of test data, it should be same of the original model
    print(str(loaded_model.score(X_test, y_test)))
```

```
0.8251466537747794
```

Stage 2: Algorithm Implementation

Now that the best performing algorithm and machine learning model for predicting house prices has been selected from stage 1, it is now time to implement the algorithm as a desktop software tool using python and the Tkinter package.

The PyCharm project for the implementation (along with the regression model and Juptyer notebook) is available at this google drive link: https://drive.google.com/drive/folders/1QZR3rbf0GmCAjUdl-BUQDtVPRdRWobzf?usp=sharing

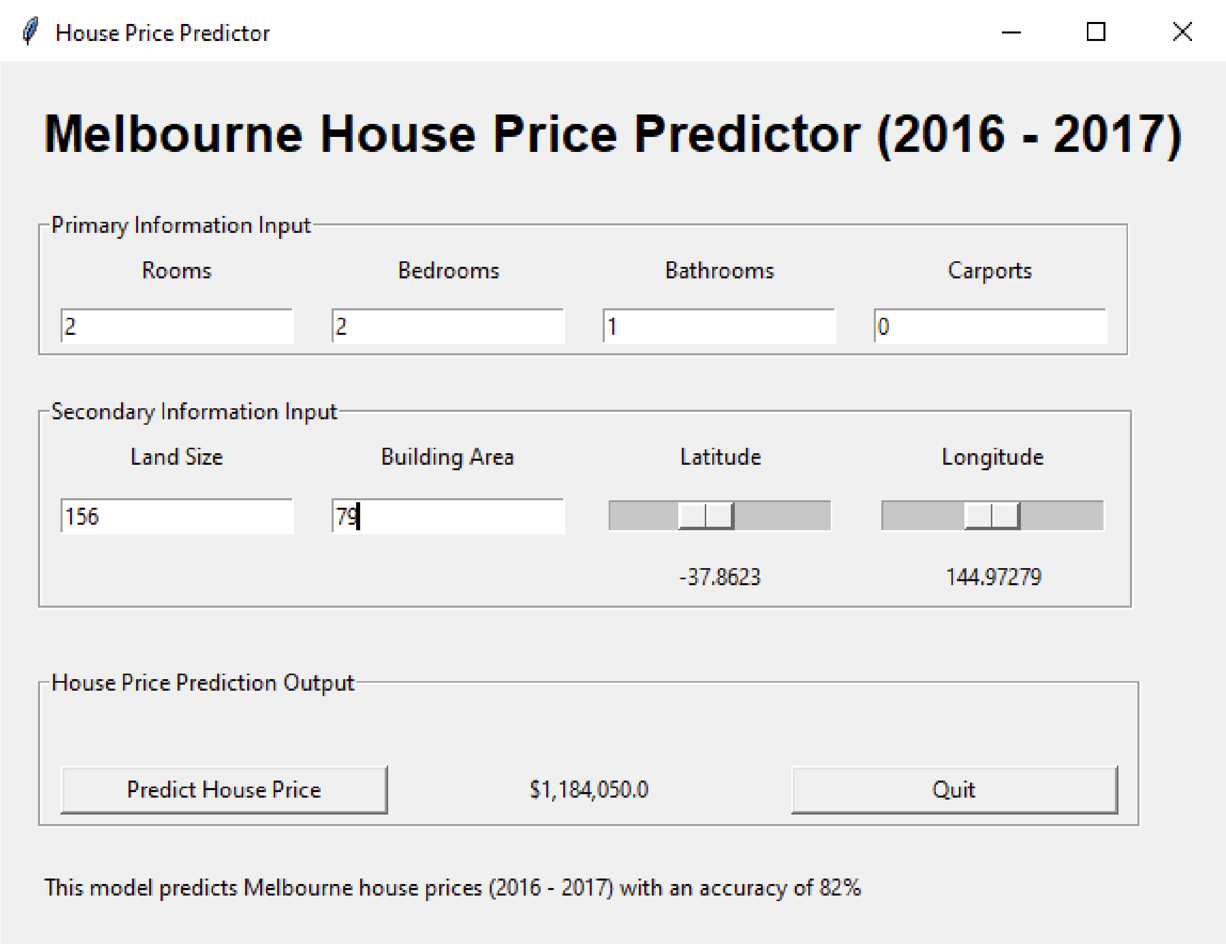
Software Testing

Test Number	Input	Expected Output	Actual Output	Pass/Fail
1	2, 2, 1, 0, 156, 79, -37.8079, 144.9934	$1,035,000 (+/- 18%)	$1,073,540	**Pass**
2	3, 3, 2, 1, 132, 159, -37.8415, 144.952	$1,800,000 (+/- 18%)	$1,671,120	**Pass**
3	1, 1, 1, 1, 0, 53, -37.8531, 145.0115	$440,000 (+/- 18%)	$422,995	**Pass**
4	3, 3, 2, 2, 532, 147, -37.7737, 144.904	$1,435,000 (+/- 18%)	$1,296,590	**Pass**
5	8, 9, 7, 4, 1472, 618, -37.8729, 145.0788	$2,950,000 (+/- 18%)	$1,839,910	**Fail**
6	4, 4, 2, 3, 628, 146, -37.7256, 144.8739	$900,000 (+/- 18%)	$952,228.35	**Pass**
7	3, 3, 2, 1, 352, 242, -37.87, 144.825	$520,000 (+/- 18%)	$904,580	**Fail**
8	2, 2, 1, 1, 154, 91, -37.8646, 144.8272	$535,600 (+/- 18%)	$511,664	**Pass**
9	2, 2, 1, 1, 214, 77, -37.7776, 144.9142	$710,000 (+/- 18%)	$741,090	**Pass**
10	4, 4, 2, 2, 757, 130, -37.9266, 145.0088	$2,310,000 (+/- 18%)	$1,652,090	**Fail**

The prediction tool works well on the average property but understandably struggles with outliers.

Conclusion

This report presents the design, development, and implementation processes utilized in completing the Melbourne house price prediction tool for the Software Technology 1 Capstone Project. This desktop software tool, developed using python and Tkinter, is useful for all stakeholders who require accurate prediction of house prices, which is undoubtedly indispensable for the healthy functioning of the housing market. Whilst the data set was missing a lot of values, especially in regards to building area, it was overall a serviceable data set for the purposes of learning data analysis, visualization, prediction, and deployment using python and various packages. Altogether, I am pleased with the results of the house price prediction tool and I believe it is accurate enough to predict house prices for the average property. Below is a screenshot of the working program:



References

1. https://uclearn.canberra.edu.au/courses/13571/assignments/105232
2. https://www.kaggle.com/datasets/dansbecker/melbourne-housing-snapshot

Appendix | Logbook/Journal

Log 1 | Week 10

In tutorials today (12/04/2023) our tutor tasked the class with finding a dataset to use for our Capstone Projects. I have no prior experience with data analysis, so I am not sure what makes a good data set. Since I have no idea how to evaluate data sets, I have chosen the 'IMDB' films data set because I am familiar with films. Overall, I spent all of my time in class looking at different data sets and trying to decide on one.

Log 2 | Week 11

After assessing the 'IMDB' dataset and learning more about data analysis, I have concluded that the dataset is not suitable for the Capstone Project task. Most 'IMDB' datasets on Kaggle do not have enough attributes or even records to create a predictive model. Therefore, I have decided to switch to the 'Melbourne Housing Snapshot' dataset. With this dataset it will be possible to create a predictive model for house prices and a Tkinter software tool that can estimate a market price for a property. I have communicated this with the tutor in class and I have posted a text submission to the 'dataset allocation' assignment on Canvas.

Log 3 | Week 12

The change in dataset has been approved by the lecturer/teaching team and I am now ready to start the algorithm design stage. I made quick progress through the data exploration and visualization stages of the project, but ran into an obstacle when it came to the predictive algorithms. When the algorithm results were evaluated based on their mean and standard deviation, my results were completely different from the example project (my results pictured below).

```
# build the model with training subset with each algorithm
# And evaluate each model using baseline performance metric
results = []
names = []
for name, model in models:
  kfold = KFold(n_splits=num_folds,shuffle=True, random_state=seed)
  cv_results = cross_val_score(model, X_train, Y_train, cv=kfold, scoring=scoring)
  results.append(cv_results)
  names.append(name)
  msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
  print(msg)
```

```
LR: -269196275534.951263 (24944359965.392467)
LASSO: -269196256168.478943 (24944037319.842979)
EN: -301324597032.750854 (26754518771.612930)
KNN: -323528219913.347595 (18082293794.716633)
CART: -174627124586.411774 (14366434772.595922)
SVR: -435822504407.291382 (20169455961.455032)
```

This forced me to diverge from the example project and try a new method. The results of my new approach are below:

```
print(f'K-nearest Neighbbour Regression: {cart.score(X_

#Support Vector Regression
svr = SVR()
svr.fit(X_train, y_train)
print(f'Support Vector Regression: {svr.score(X_test, y

#LASSO
lasso = Lasso()
lasso.fit(X_train, y_train)
print(f'LASSO: {lasso.score(X_test, y_test)}')

#ElasticNet
en = ElasticNet()
en.fit(X_train, y_train)
print(f'ElasticNet: {en.score(X_test, y_test)}')
```

```
Linear Regression: 0.3386553618596897
Random Forest Regression: 0.8078663764991502
Gradient Boosting Regression: 0.762530373711211
Ada Boost Regression: 0.4471395898992241
Extra Trees Regression: 0.8035667006351319
Linear Regression: 0.6984719290374
K-nearest Neighbbour Regression: 0.6092057541138809
Support Vector Regression: -0.02180585971469995
LASSO: 0.33866413178196264
ElasticNet: 0.03783250663267168
```

I have determined that the best performing algorithm is the Random Forest Regressor. I believe that I am now ready to move on to the model preparation stage.

Log 4 | Week 13

After completing the Google Colab data analysis, visualization, and predictive algorithm development stage, I began work on this report. It did not take that long to finish the report and soon I was on to the Tkinter program development stage. After reading through the example Tkinter deployment, I learnt how to import the prediction model using pickle, and then it was time to create the GUI. This again did not take too long, as I was able to reuse a lot of my code from the previous Software Technology 1 Programming Challenges assignment. However, this was the first time I used the Tkinter sliders or 'scales', which took some experimentation to get just how I wanted it. The sliders are a bit of a tradeoff because they are less accurate than an entry box, but I thought that this design better communicates to the user that they are limited to longitude and latitude that corresponds with Melbourne. All that is left to do now is to prepare for the presentation.

Appendix | Python Code

```
import pickle
import tkinter as tk

# Constants
LATITUDE_FACTOR = 10000
LONGITUDE_FACTOR = 100000

# Load the prediction model
with open('best_regr_model.h5', 'rb') as file:
   prediction_model = pickle.load(file)


# Functions
def predict_house_price(house_data):
   house_price_prediction = prediction_model.predict([house_data])
   return house_price_prediction


def custom_slider_value_display(load_bearing_argument):
   latitude_result_label.config(text=str(latitude_scale.get()      /
LATITUDE_FACTOR))
   longitude_result_label.config(text=str(longitude_scale.get()   /
LONGITUDE_FACTOR))


def process_house_price(house_price_raw):
   # Convert to string to strip unnecessary characters
   house_price_str = str(house_price_raw).strip('[].')
   # Convert to number and round
   house_price = round(float(house_price_str), 2)
   # Return processed house price value
   return house_price


def enter_info_and_display_price():
   # Collect primary information
   rooms = rooms_entry.get()
```

```
    bedrooms = bedrooms_entry.get()
    bathrooms = bathrooms_entry.get()
    carports = carports_entry.get()
    # Collect secondary information
    land = land_entry.get()
    building = building_entry.get()
    latitude = latitude_scale.get() / LATITUDE_FACTOR
    longitude = longitude_scale.get() / LONGITUDE_FACTOR

    # Collate house data
    house_data = (rooms, bedrooms, bathrooms, carports, land, building, latitude, longitude)
    # Predict the house price
    house_price_raw = predict_house_price(house_data)
    # Process house price
    house_price = process_house_price(house_price_raw)
    # Display the predicted house price
    result_label.config(text=f'${house_price:,}')


# Create the main window.
window = tk.Tk()
window.title("House Price Predictor")

# Create frame
base_frame = tk.Frame(window)
base_frame.pack()

# Create title label
title_label = tk.Label(base_frame, text="Melbourne House Price Predictor (2016 - 2017)", font="Arial 20 bold")
title_label.grid(row=0, column=0, padx=20, pady=(20, 0), sticky='WENS')

# Create primary information input frame
info_frame_1 = tk.LabelFrame(base_frame, text="Primary Information Input")
info_frame_1.grid(row=1, column=0, padx=20, pady=(20, 10), sticky='W')
```

```
# Rooms label
rooms_label = tk.Label(info_frame_1, text="Rooms")
rooms_label.grid(row=0, column=0)
# Rooms entry
rooms_entry = tk.Entry(info_frame_1)
rooms_entry.grid(row=1, column=0)

# Bedrooms label
bedrooms_label = tk.Label(info_frame_1, text="Bedrooms")
bedrooms_label.grid(row=0, column=1)
# Bedrooms entry
bedrooms_entry = tk.Entry(info_frame_1)
bedrooms_entry.grid(row=1, column=1)

# Bathrooms label
bathrooms_label = tk.Label(info_frame_1, text="Bathrooms")
bathrooms_label.grid(row=0, column=2)
# Bathrooms entry
bathrooms_entry = tk.Entry(info_frame_1)
bathrooms_entry.grid(row=1, column=2)

# Carports label
carports_label = tk.Label(info_frame_1, text="Carports")
carports_label.grid(row=0, column=3)
# Carports entry
carports_entry = tk.Entry(info_frame_1)
carports_entry.grid(row=1, column=3)

# Create secondary information input frame
info_frame_2 = tk.LabelFrame(base_frame, text="Secondary Information Input")
info_frame_2.grid(row=2, column=0, padx=20, pady=(10, 20), sticky='W')

# Land size label
land_label = tk.Label(info_frame_2, text="Land Size")
land_label.grid(row=0, column=0)
# land size entry
land_entry = tk.Entry(info_frame_2)
land_entry.grid(row=1, column=0)
```

```
# Building area label
building_label = tk.Label(info_frame_2, text="Building Area")
building_label.grid(row=0, column=1)
# Building area entry
building_entry = tk.Entry(info_frame_2)
building_entry.grid(row=1, column=1)

# Latitude label
latitude_label = tk.Label(info_frame_2, text="Latitude")
latitude_label.grid(row=0, column=2)
# Latitude scale
latitude_scale = tk.Scale(info_frame_2, length=119,
showvalue=False, from_=-381826, to=-374085, orient='horizontal',
command=custom_slider_value_display)
latitude_scale.grid(row=1, column=2,)
latitude_scale.set(-381826)
# Latitude result label
latitude_result_label = tk.Label(info_frame_2, text="-37.4085")
latitude_result_label.grid(row=2, column=2)

# Longitude label
longitude_label = tk.Label(info_frame_2, text="Longitude")
longitude_label.grid(row=0, column=3)
# Longitude scale
longitude_scale = tk.Scale(info_frame_2, length=119,
showvalue=False, from_=14443181, to=14552635, orient='horizontal',
command=custom_slider_value_display)
longitude_scale.grid(row=1, column=3,)
# Longitude result label
longitude_result_label = tk.Label(info_frame_2, text="144.43181")
longitude_result_label.grid(row=2, column=3)

# Create tertiary information input frame
info_frame_3 = tk.LabelFrame(base_frame, text="House Price
Prediction Output")
info_frame_3.grid(row=3, column=0, padx=20, pady=(10, 20),
sticky='W')

# Padding labels (ignore)
```

```
padding_label1 = tk.Label(info_frame_3, text="", width=24)
padding_label1.grid(row=0, column=0)
padding_label2 = tk.Label(info_frame_3, text="", width=24)
padding_label2.grid(row=0, column=1)
padding_label3 = tk.Label(info_frame_3, text="", width=24)
padding_label3.grid(row=0, column=2)

# Enter info button
predict_button = tk.Button(info_frame_3, text="Predict House Price", command=enter_info_and_display_price)
predict_button.grid(row=1, column=0, sticky="WENS")

# Result label
result_label = tk.Label(info_frame_3, text="")
result_label.grid(row=1, column=1, sticky="WENS")

# Quit button
quit_button = tk.Button(info_frame_3, text="Quit", command=window.destroy)
quit_button.grid(row=1, column=2, sticky="WENS")

# Accuracy label
accuracy_label = tk.Label(base_frame, text="This model predicts Melbourne house prices (2016 - 2017) with an accuracy of 82%")
accuracy_label.grid(row=4, column=0, padx=20, pady=(1, 20), sticky="W")

# Pad GUI elements
for widget in info_frame_1.winfo_children():
   widget.grid_configure(padx=10, pady=5)

for widget in info_frame_2.winfo_children():
   widget.grid_configure(padx=10, pady=5)

for widget in info_frame_3.winfo_children():
   widget.grid_configure(padx=10, pady=5)

# Enter the tkinter main loop.
tk.mainloop()
```