In [1]:
```python
import numpy as np
import pandas as pd
import seaborn as sns
import datetime
import plotly.graph_objs as go
import plotly.offline as py
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.metrics import accuracy_score, classification_report
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score, mean_absolute_percentage_erro
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from plotly.offline import download_plotlyjs, init_notebook_mode, i
from statsmodels.tsa.stattools import acf, pacf
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from sklearn.metrics import mean_absolute_error
py.init_notebook_mode(connected = True)
dataf= pd.read_csv ( "melbdata.csv")
dataf.head()
```

Out[1]:

	Suburb	Address	Rooms	Type	Price	Method	SellerG	Date	Distance
0	Abbotsford	85 Turner St	2	h	1480000.0	S	Biggin	3/12/2016	2.5
1	Abbotsford	25 Bloomburg St	2	h	1035000.0	S	Biggin	4/02/2016	2.5
2	Abbotsford	5 Charles St	3	h	1465000.0	SP	Biggin	4/03/2017	2.5
3	Abbotsford	40 Federation La	3	h	850000.0	PI	Biggin	4/03/2017	2.5
4	Abbotsford	55a Park St	4	h	1600000.0	VB	Nelson	4/06/2016	2.5

5 rows × 21 columns

In [2]:
```
# Check for missing values in the dataset
missing_values = dataf.isnull().sum()
print(missing_values)
```

```
Suburb               0
Address              0
Rooms                0
Type                 0
Price                0
Method               0
SellerG              0
Date                 0
Distance             0
Postcode             0
Bedroom2             0
Bathroom             0
Car                 62
Landsize             0
BuildingArea      6450
YearBuilt         5375
CouncilArea       1369
Lattitude            0
Longtitude           0
Regionname           0
Propertycount        0
dtype: int64
```

In [3]:
```
dataf.info
```

Out[3]:
```
<bound method DataFrame.info of               Suburb           Add
ress  Rooms Type      Price Method  \
0          Abbotsford      85 Turner St      2    h  1480000.0
S
1          Abbotsford   25 Bloomburg St      2    h  1035000.0
S
2          Abbotsford      5 Charles St      3    h  1465000.0
SP
3          Abbotsford  40 Federation La      3    h   850000.0
PI
4          Abbotsford       55a Park St      4    h  1600000.0
VB
...               ...               ...    ...  ...        ...    .
..
13575  Wheelers Hill      12 Strada Cr      4    h  1245000.0
S
13576   Williamstown     77 Merrett Dr      3    h  1031000.0
SP
13577   Williamstown       83 Power St      3    h  1170000.0
S
13578   Williamstown      96 Verdon St      4    h  2500000.0
PI
13579     Yarraville        6 Agnes St      4    h  1285000.0
SP

        SellerG        Date  Distance  Postcode  ...  Bathroom  Ca
r  Landsize  \
0        Biggin   3/12/2016       2.5    3067.0  ...       1.0  1.
```

```
0     202.0
1        Biggin   4/02/2016       2.5    3067.0  ...       1.0  0.
0     156.0
2        Biggin   4/03/2017       2.5    3067.0  ...       2.0  0.
0     134.0
3        Biggin   4/03/2017       2.5    3067.0  ...       2.0  1.
0      94.0
4        Nelson   4/06/2016       2.5    3067.0  ...       1.0  2.
0     120.0
...         ...         ...       ...       ...  ...       ...  ..
.       ...
13575     Barry  26/08/2017      16.7    3150.0  ...       2.0  2.
0     652.0
13576  Williams  26/08/2017       6.8    3016.0  ...       2.0  2.
0     333.0
13577     Raine  26/08/2017       6.8    3016.0  ...       2.0  4.
0     436.0
13578   Sweeney  26/08/2017       6.8    3016.0  ...       1.0  5.
0     866.0
13579   Village  26/08/2017       6.3    3013.0  ...       1.0  1.
0     362.0

       BuildingArea  YearBuilt  CouncilArea Lattitude  Longtitude
\
0               NaN        NaN        Yarra -37.79960   144.99840
1              79.0     1900.0        Yarra -37.80790   144.99340
2             150.0     1900.0        Yarra -37.80930   144.99440
3               NaN        NaN        Yarra -37.79690   144.99690
4             142.0     2014.0        Yarra -37.80720   144.99410
...             ...        ...          ...       ...         ...
13575           NaN     1981.0          NaN -37.90562   145.16761
13576         133.0     1995.0          NaN -37.85927   144.87904
13577           NaN     1997.0          NaN -37.85274   144.88738
13578         157.0     1920.0          NaN -37.85908   144.89299
13579         112.0     1920.0          NaN -37.81188   144.88449

                       Regionname Propertycount
0           Northern Metropolitan        4019.0
1           Northern Metropolitan        4019.0
2           Northern Metropolitan        4019.0
3           Northern Metropolitan        4019.0
4           Northern Metropolitan        4019.0
...                           ...           ...
13575  South-Eastern Metropolitan        7392.0
13576          Western Metropolitan        6380.0
13577          Western Metropolitan        6380.0
13578          Western Metropolitan        6380.0
13579          Western Metropolitan        6543.0

[13580 rows x 21 columns]>
```

In [4]:

```
# Replace missing values with zero
dataf.fillna(0, inplace=True)
```

In [5]:
```python
# Check for missing values in the dataset
missing_values = dataf.isnull().sum()
print(missing_values)
```

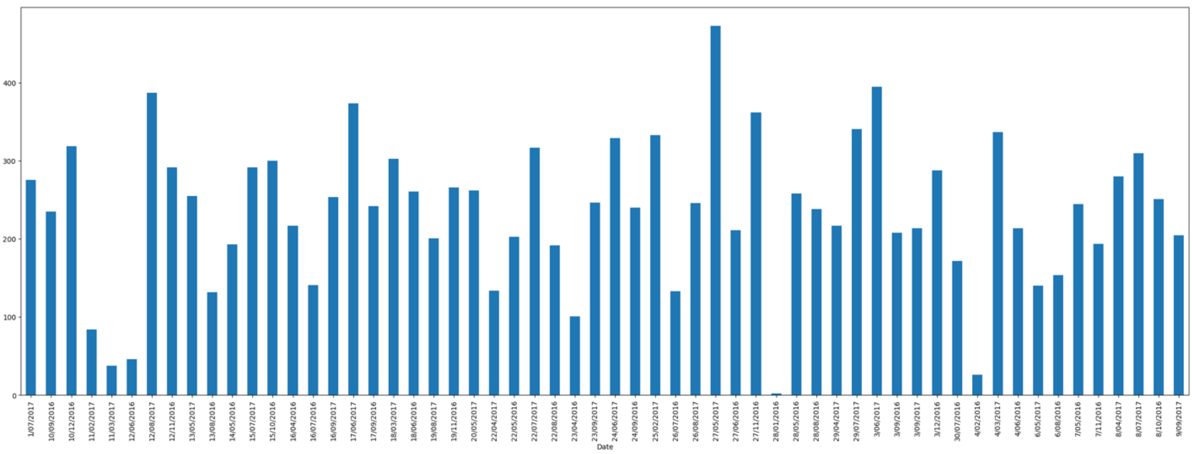
```
Suburb           0
Address          0
Rooms            0
Type             0
Price            0
Method           0
SellerG          0
Date             0
Distance         0
Postcode         0
Bedroom2         0
Bathroom         0
Car              0
Landsize         0
BuildingArea     0
YearBuilt        0
CouncilArea      0
Lattitude        0
Longtitude       0
Regionname       0
Propertycount    0
dtype: int64
```

In [6]:
```python
# Check for duplicate rows
duplicate_rows = dataf.duplicated()
print(duplicate_rows.sum())
# Remove duplicate rows
dataf = dataf.drop_duplicates ()
```

```
0
```

In [7]:
```python
dataf.groupby(['Date'])['Price'].count().plot(kind = 'bar', figsize
```

Out[7]:
```
<AxesSubplot:xlabel='Date'>
```



```
In [8]: df_date_idx = dataf.set_index('Date')
```

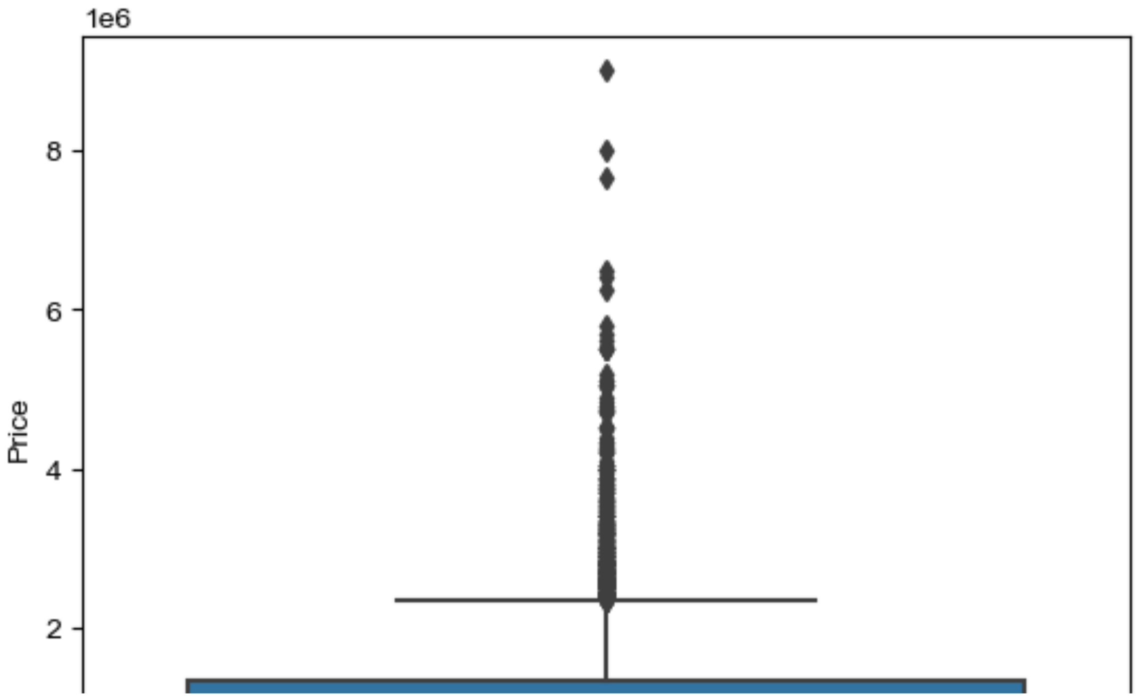
```
In [9]: dsc = df_date_idx['Price'].describe()
        print('Basic level stats about the price data\n', dsc)
```

```
Basic level stats about the price data
 count    1.358000e+04
mean     1.075684e+06
std      6.393107e+05
min      8.500000e+04
25%      6.500000e+05
50%      9.030000e+05
75%      1.330000e+06
max      9.000000e+06
Name: Price, dtype: float64
```

```
In [10]: dsc = df_date_idx['Price'].describe()
         min_dsc = dsc[3]
         med_dsc = dsc[5]
         max_dsc = int(med_dsc+(1.5*(dsc[6]-dsc[4])))
         box_plot = sns.boxplot(y = df_date_idx.Price, data = df_date_idx)

         x_tick = box_plot.get_xticks()
         box_plot.text(x_tick, med_dsc, med_dsc,horizontalalignment='center'

         sns.set(rc={'figure.figsize':(10,10)}, font_scale = 1)
```



Beyond the 75th quartile figure of
1.33*million*, *itappearsthattherearea fewoutliers*. *Thestartingprice forapropei*
and the median cost of a home is 903, 000.*Ihavetoconcedethat*85,000 is a
remarkably low price for a home in Melbourne. Permit me to look for homes in these
three categories.

In [11]:
```
dataf[dataf['Price'].isin([85000,903000,1.330000e+06])].sort_values
```
Out[11]:

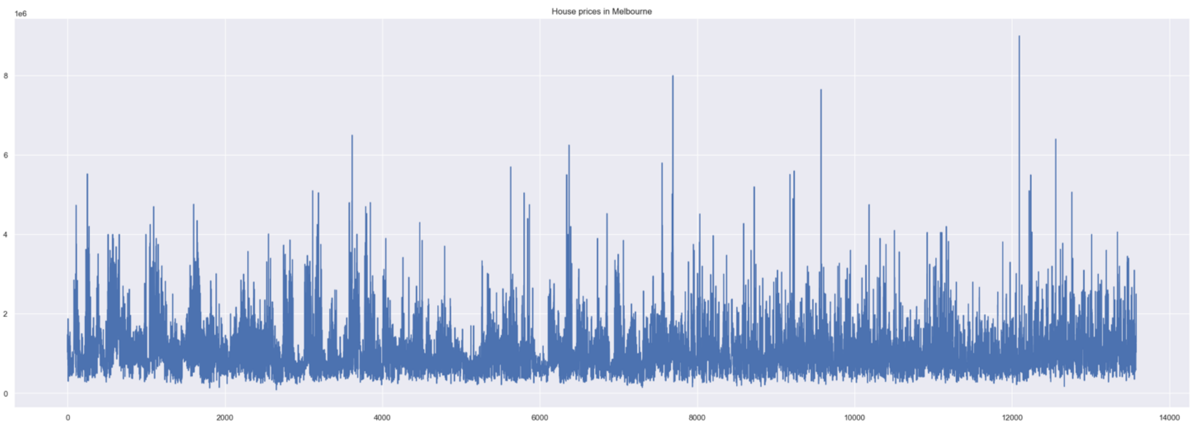
	Suburb	Address	Rooms	Type	Price	Method	SellerG	Da
2652	Footscray	202/51 Gordon St	1	u	85000.0	PI	Burnham	3/09/20
6238	Templestowe Lower	208 Templestowe Rd	4	h	903000.0	S	Barry	27/06/20
1242	Brighton East	2/33 Cluden St	2	u	903000.0	S	Hodges	27/11/20
1502	Bulleen	28 Flinders St	2	h	903000.0	S	Barry	26/07/20
1958	Coburg	53 Hawthorn St	2	h	903000.0	S	Nelson	15/10/20
8640	Kensington	33 Chelmsford St	2	h	903000.0	S	Rendina	22/04/20
7311	Glen Huntly	8 Augusta St	3	h	1330000.0	S	Woodards	10/09/20
7431	Aberfeldie	32 Fawkner St	4	h	1330000.0	S	Nelson	6/05/20
8338	Richmond	3 Kimber St	3	h	1330000.0	S	Jellis	22/04/20
8519	Williamstown	240 Coogee La	3	h	1330000.0	S	Greg	29/04/20
20	Abbotsford	3/72 Charles St	4	h	1330000.0	PI	Kay	18/03/20
9432	Bentleigh East	52 Paloma St	4	h	1330000.0	PI	hockingstuart	17/06/20
10119	Balwyn North	2/10 Bolinda Rd	3	t	1330000.0	S	Marshall	27/05/20
10294	Glen Iris	1a Pascoe St	3	u	1330000.0	S	Fletchers	27/05/20
10601	Brunswick East	219 Glenlyon Rd	3	h	1330000.0	PI	Woodards	8/07/20
6513	Williamstown	112 Crofton Dr	4	h	1330000.0	S	Williams	12/11/20
8898	Newport	34 Thorpe St	3	h	1330000.0	S	RT	1/07/20
5487	Seddon	58 Station Rd	3	h	1330000.0	S	Sweeney	7/11/20
12707	Doncaster East	5 Ryder Ct	5	h	1330000.0	S	Parkes	16/09/20
5009	Preston	11 Inverloch St	3	h	1330000.0	S	Barry	27/11/20
4354	North Melbourne	331 Flemington Rd	4	h	1330000.0	S	McDonald	10/09/20
4290	Niddrie	51 Garnet St	1	h	1330000.0	S	Brad	3/12/20

3645	Kew	2/37 Wills St	3	t	1330000.0	PI	Jellis	19/11/20
2448	Essendon	8 Buckley St	3	h	1330000.0	S	Barry	26/07/20
1855	Caulfield South	42 Poplar St	3	h	1330000.0	S	Gary	26/07/20
1684	Carlton North	40 Ogrady St	2	h	1330000.0	S	Nelson	15/10/20
1670	Carlton North	527 Nicholson St	3	h	1330000.0	VB	Nelson	3/09/20
1414	Brunswick West	81 Whitby St	3	h	1330000.0	S	Nelson	12/11/20
5891	Strathmore	27 Henshall Rd	3	h	1330000.0	S	Considine	10/09/20
13571	Wantirna South	15 Mara Cl	4	h	1330000.0	S	Barry	26/08/20

30 rows × 21 columns

In [12]:
```
dataf['Price'].plot(kind = 'line', title = 'House prices in Melbour
```

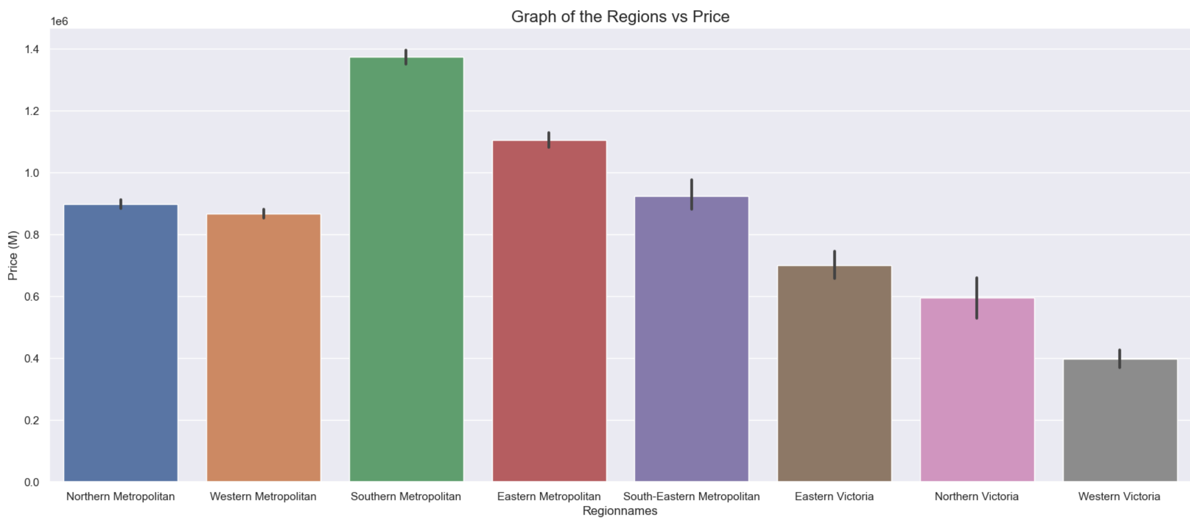
Out[12]:
```
<AxesSubplot:title={'center':'House prices in Melbourne'}>
```



This is so intriguing! Auto-Correlation technologies can be used to analyse the data and find recurring trends, including seasonality. We've already established a date-time index for the time series data before to applying the ACF. We eliminated the dataset's trend to establish stationarity, ensuring the ACF's efficacy. We can now detect any seasonality in the data over time with a stationary dataset.

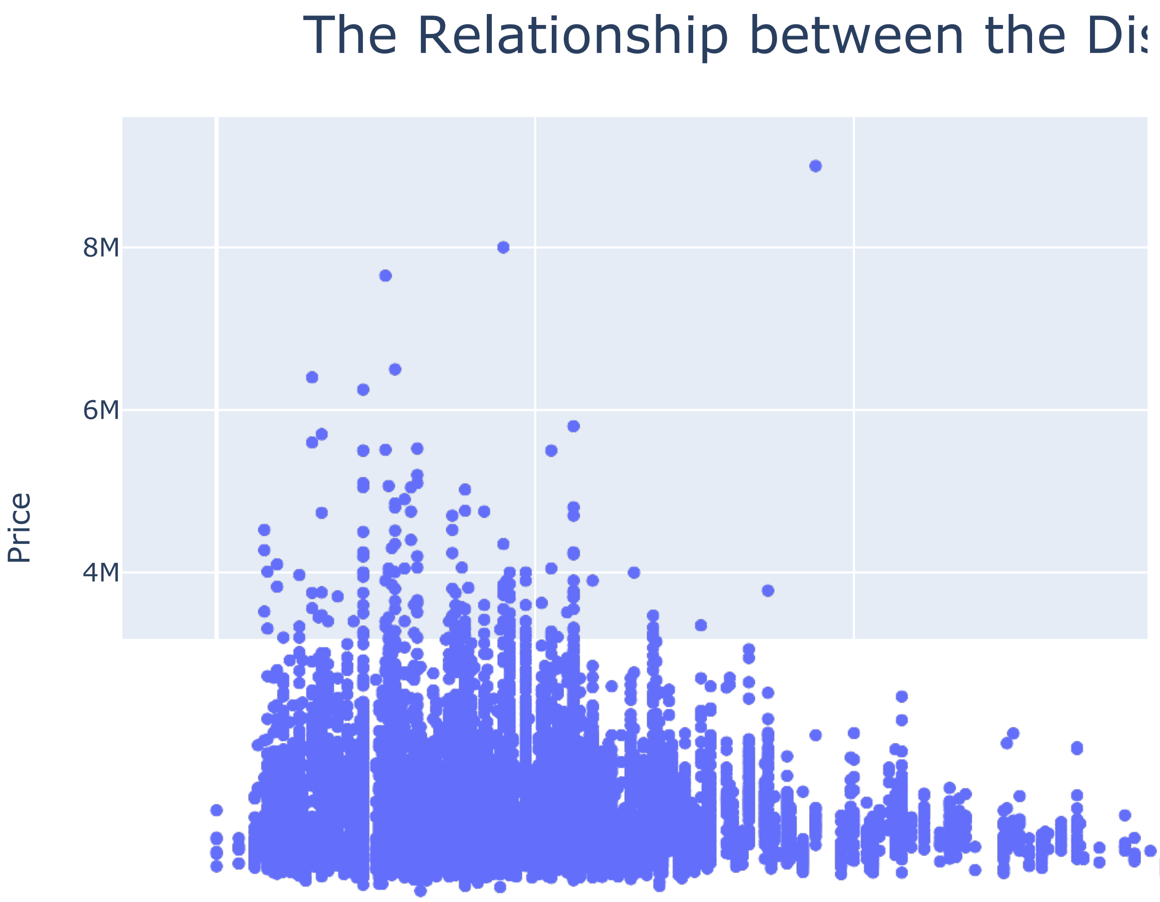
In [13]:
```python
plt.figure(figsize=(20, 8))
sns.barplot(x=dataf['Regionname'], y=dataf['Price'])
plt.title("Graph of the Regions vs Price", fontsize=16)
plt.ylabel('Price (M)', fontsize=12)
plt.xlabel('Regionnames', fontsize=12)
```

Out[13]:
```
Text(0.5, 0, 'Regionnames')
```



A graph of regions vs home prices is shown in the figure. The Y-axis displays housing prices, and the X-axis lists the names of the regions. Despite having the highest total cost of housing, the Southern Metropolitan region does not necessarily have the most costly homes. The Southern Metropolitan has the largest density, according to the region names' density. The Southern Metropolitan region may not actually be the most expensive due to its highest pricing.

In [14]:
```
import plotly.express as px
fig = px.scatter(dataf, x='Distance', y='Price', hover_data=['Price
fig.update_layout(title='The Relationship between the Distance and
fig.show()
```



The Y-axis in the scatter plot above reflects house prices, and the X-axis the distance to the Central Business Sub District. The information demonstrates that lower distances are where the majority of house prices and distances are concentrated. This pattern makes sense because individuals want homes near the Central Business Sub District, which increases demand and drives up housing costs.

In [15]:
```
fig = px.scatter(dataf, x='Regionname', y='Price', hover_data=['Pri
fig.update_layout(title='The Relationship between the Regionname an
fig.show()
```

The X-axis of the scatter plot shows the names of the regions, and the Y-axis shows house prices. The South-Eastern Metropolitan area is home to the most costly residence, which costs $9 million.

The Southern Metropolitan region has the highest total housing values, according to an analysis of the data using a bar plot. It is important to keep in mind, though, that just because one area has the highest total prices doesn't indicate it's where the most costly homes are. The Southern Metropolitan is the region with the highest cost, according to the scatter plot.

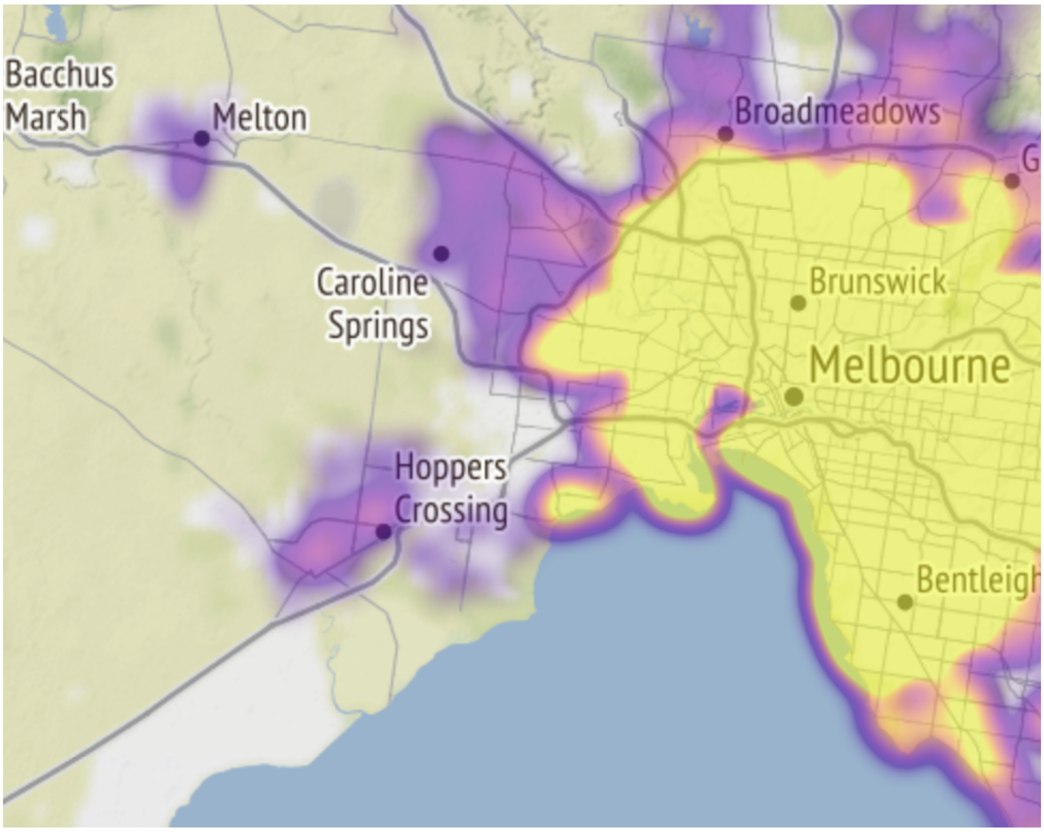
On the other side, Western Victoria is the most affordable region for homes, and the least costly home, which costs $85K, is situated in the Western Metropolitan area.

In [16]:
```
total_rooms = dataf['Rooms'] + dataf['Bedroom2'] + dataf['Bathroom'
fig = px.scatter(dataf, x=total_rooms, y='Price', hover_data=['Pric
fig.update_layout(title='The Relationship between the Regionname an
fig.update_layout(xaxis_title='Total Number of Rooms')
fig.show()
```

The Total Number of Rooms (which includes bedrooms and baths) is represented by the X-axis in this scatter plot, and the House Prices are represented by the Y-axis. The two variables have a significant positive correlation up to 15 Total Number of Rooms. Beyond 15, the Total Number of Rooms does not continue to correlate with this relationship.

In [17]:
```
fig = px.density_mapbox(dataf, lat='Lattitude', lon='Longtitude', z
                        center=dict(lat=-37.823002, lon=144.998001)
                        mapbox_style="stamen-terrain",
                        radius=20,
                        opacity=0.5)
fig.update_layout(title_text='Melbourne Heatmap of the House Prices
fig.show()
```

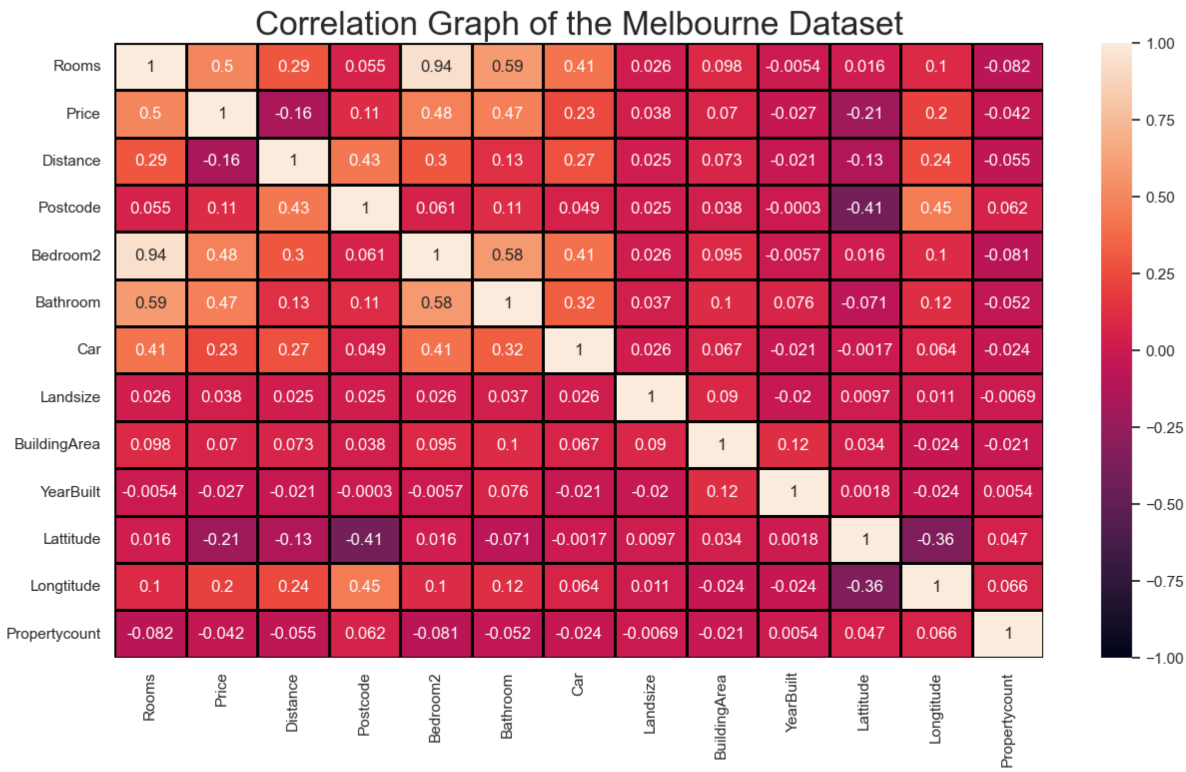
Melbourne Heatmap of tl



The Melbourne House Prices are displayed on the heatmap. Zoom as needed for a more thorough examination. The highest values, which are reasonable for this investigation, are found in South-Eastern Metropolitan.

In [18]:
```
plt.figure(figsize=(15, 8))
correlation = sns.heatmap(dataf.corr(), vmin=-1, vmax=1, annot=True
correlation.set_title('Correlation Graph of the Melbourne Dataset',
```

Out[18]:
```
Text(0.5, 1.0, 'Correlation Graph of the Melbourne Dataset')
```



The Melbourne Dataset's Heatmap clearly shows that the Rooms feature has a significant positive link with house prices. This conclusion is supported by the scatter plot of these features.

On the other hand, in line with their scatter plot, House Price and Distance show a negative association. It suggests that the price of homes tends to rise as the distance from the CBD decreases.

A strong correlation between house prices and the features of the bedroom and bathroom has also been discovered. The scatter plot of these factors and this observation line up, proving the importance of these variables.

I could choose the best characteristics for the machine learning model by using this technique.

Fixing Landsize Data Distribution

It is best to remove the variable from the dataset in light of the mistakes discovered in the recorded landsize data. The following factors formed the basis for this choice:

One of the dataset's properties with the largest land size (389 Gore St., Fitzroy) contained errors. Because of point 1, there is little trust in the precision of all other data values. A more efficient model development procedure will result from the removal of erroneous data.

In [19]:
```python
missing_values = dataf.isnull().sum()
print(missing_values)
```

```
Suburb           0
Address          0
Rooms            0
Type             0
Price            0
Method           0
SellerG          0
Date             0
Distance         0
Postcode         0
Bedroom2         0
Bathroom         0
Car              0
Landsize         0
BuildingArea     0
YearBuilt        0
CouncilArea      0
Lattitude        0
Longtitude       0
```

In [20]:
```python
#Drop variables which are not required
dataf=dataf.drop (['Postcode', 'Method', 'SellerG', 'Date', 'Counci
'Address',],axis=1)
dataf.head( )
```

Out[20]:

	Suburb	Rooms	Type	Price	Distance	Bedroom2	Bathroom	Car	Landsize	Bu
0	Abbotsford	2	h	1480000.0	2.5	2.0	1.0	1.0	202.0	
1	Abbotsford	2	h	1035000.0	2.5	2.0	1.0	0.0	156.0	
2	Abbotsford	3	h	1465000.0	2.5	3.0	2.0	0.0	134.0	
3	Abbotsford	3	h	850000.0	2.5	3.0	2.0	1.0	94.0	
4	Abbotsford	4	h	1600000.0	2.5	3.0	1.0	2.0	120.0	

In [21]:
```python
# Normalize numerical features using StandardScaler
scaler = StandardScaler()
numerical_features = ['Rooms','Distance','Bedroom2', 'Bathroom','Ca
'YearBuilt','Lattitude','Longtitude','Propertycount']
dataf[numerical_features]= scaler.fit_transform(dataf[numerical_fea
```

In [22]:
```python
one_hot_encoded_df = pd.get_dummies(dataf, columns=['Type'])

print("One-Hot Encoded DataFrame:")
print(one_hot_encoded_df)
print()
```

```
One-Hot Encoded DataFrame:
              Suburb     Rooms      Price  Distance  Bedroom2  B
athroom  \
0         Abbotsford -0.981463  1480000.0 -1.301485 -0.947035 -0
.772376
1         Abbotsford -0.981463  1035000.0 -1.301485 -0.947035 -0
.772376
2         Abbotsford  0.064876  1465000.0 -1.301485  0.088284  0
.673367
3         Abbotsford  0.064876   850000.0 -1.301485  0.088284  0
.673367
4         Abbotsford  1.111216  1600000.0 -1.301485  0.088284 -0
.772376
...              ...       ...        ...       ...       ...
...
13575  Wheelers Hill  1.111216  1245000.0  1.118210  1.123604  0
.673367
13576   Williamstown  0.064876  1031000.0 -0.568761  0.088284  0
.673367
```

In [23]:
```python
X = one_hot_encoded_df.drop(columns=['Price','Suburb','Regionname']
y = one_hot_encoded_df['Price']                # Target variable
```

In [24]:
```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
```

In [25]:
```python
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
```

In [26]:
```python
rf_model.fit(X_train, y_train)
```

Out[26]:
```
RandomForestRegressor(random_state=42)
```

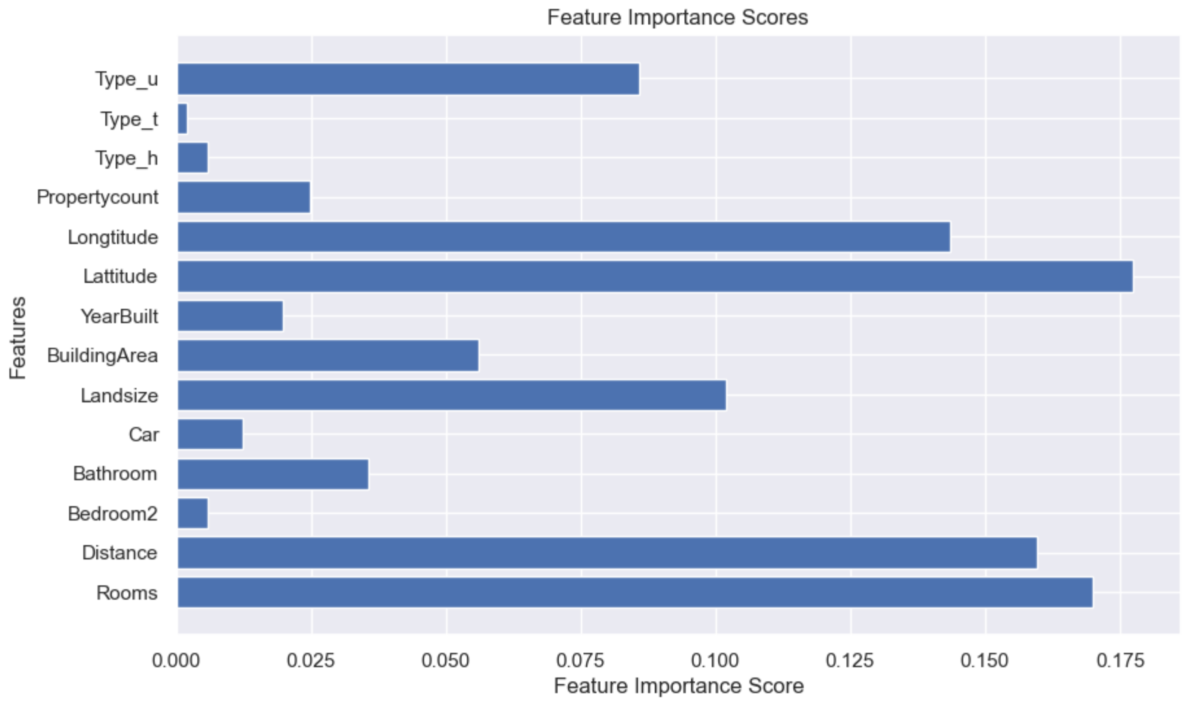
In [27]:
```python
feature_importances = rf_model.feature_importances_
```

In [28]:
```python
feature_importance_df = pd.DataFrame({'Feature': X.columns, 'Import
```

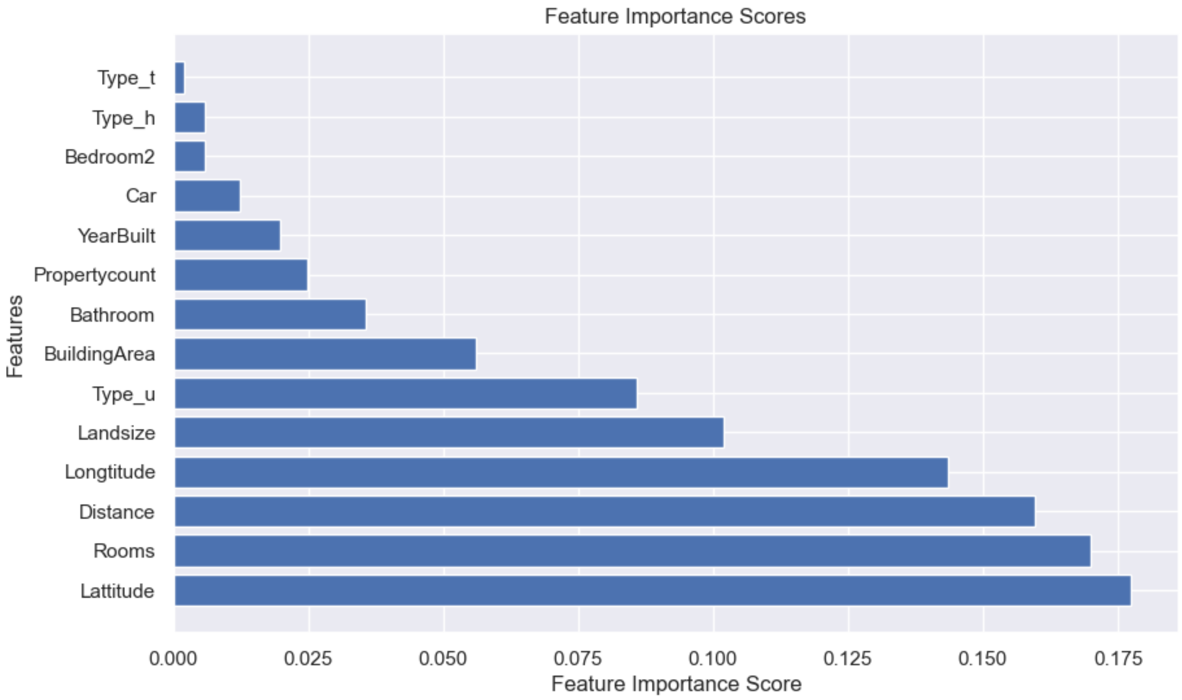
In [29]:
```python
print(feature_importance_df)
```

```
         Feature  Importance
0          Rooms    0.169899
1       Distance    0.159619
2       Bedroom2    0.005774
3       Bathroom    0.035555
4            Car    0.012273
5       Landsize    0.101973
6   BuildingArea    0.056132
7      YearBuilt    0.019685
8      Lattitude    0.177360
9     Longtitude    0.143517
10 Propertycount    0.024827
11        Type_h    0.005717
12        Type_t    0.001942
13        Type_u    0.085726
```

In [30]:
```python
plt.figure(figsize=(10, 6))
plt.barh(feature_importance_df['Feature'], feature_importance_df['I
plt.xlabel('Feature Importance Score')
plt.ylabel('Features')
plt.title('Feature Importance Scores')
plt.show()
```



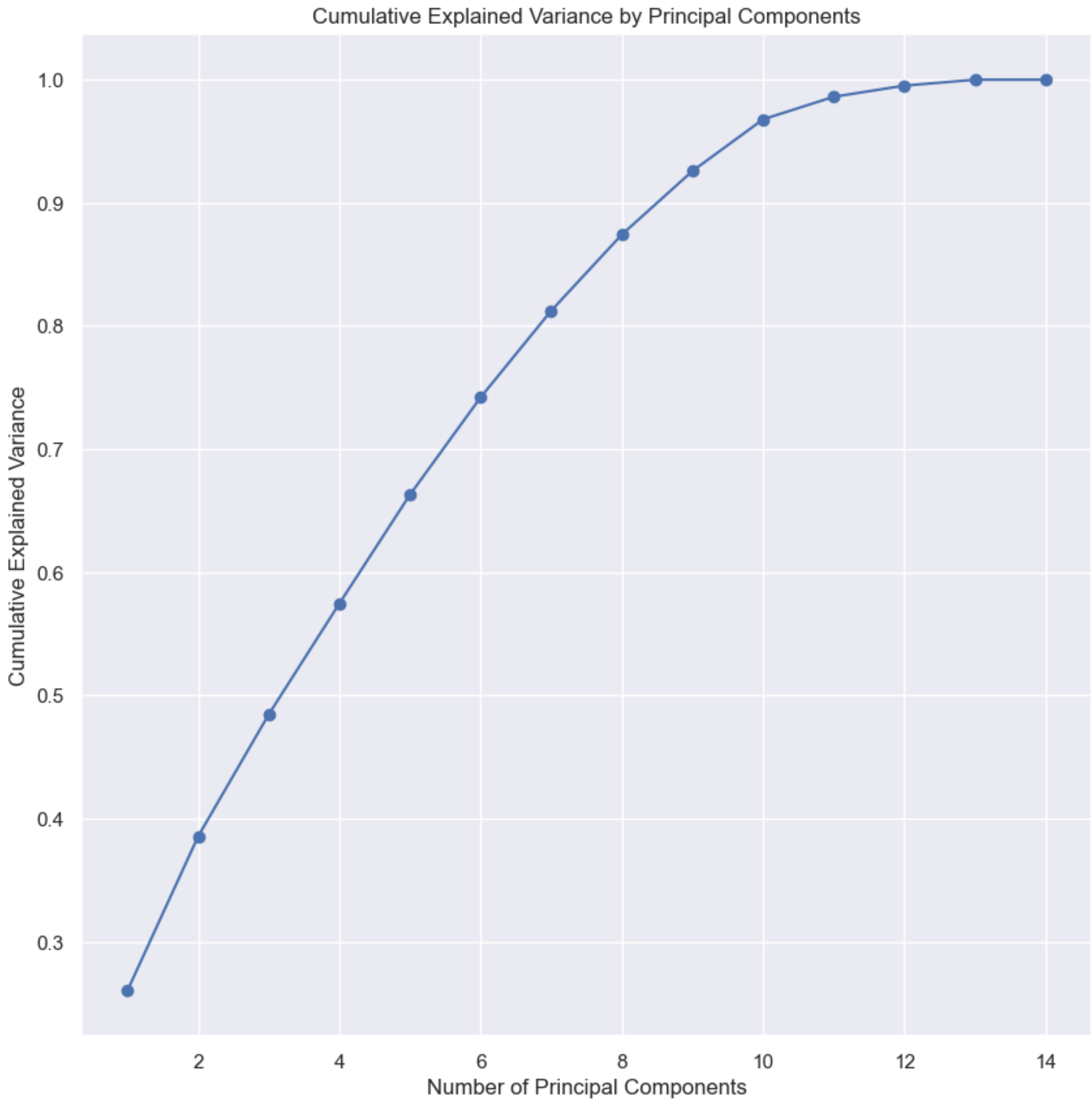
In [31]:
```
feature_importance_df = feature_importance_df.sort_values(by='Impor
plt.figure(figsize=(10, 6))
plt.barh(feature_importance_df['Feature'], feature_importance_df['I
plt.xlabel('Feature Importance Score')
plt.ylabel('Features')
plt.title('Feature Importance Scores')
plt.show()
```



In [32]:
```
newdf = one_hot_encoded_df.drop(columns=['Price','Suburb','Regionna
pca = PCA()
principal_components = pca.fit_transform(newdf)
```

In [33]:
```
explained_variance_ratio = pca.explained_variance_ratio_
cumulative_explained_variance = np.cumsum(explained_variance_ratio)
```

In [34]:
```
plt.plot(np.arange(1, min(principal_components.shape)+1), cumulativ
plt.xlabel('Number of Principal Components')
plt.ylabel('Cumulative Explained Variance')
plt.title('Cumulative Explained Variance by Principal Components')
plt.grid(True)
plt.show()
```



Data reduction techniques like Principal Component Analysis (PCA) are unsupervised methods. It enables us to represent data in a low-dimensional manner while retaining as much feature diversity as feasible. The first six components, which make up almost 80% of the variance of all predictors, are extracted for further analysis after applying PCA to the train data using cross-validation. demonstrates the screen plot of the total percentage of variance across the primary components that can be explained.

In [35]:
```
dataf
```

Out[35]:

	Suburb	Rooms	Type	Price	Distance	Bedroom2	Bathroom	(
0	Abbotsford	-0.981463	h	1480000.0	-1.301485	-0.947035	-0.772376	-0.623(
1	Abbotsford	-0.981463	h	1035000.0	-1.301485	-0.947035	-0.772376	-1.658:
2	Abbotsford	0.064876	h	1465000.0	-1.301485	0.088284	0.673367	-1.658:
3	Abbotsford	0.064876	h	850000.0	-1.301485	0.088284	0.673367	-0.623(
4	Abbotsford	1.111216	h	1600000.0	-1.301485	0.088284	-0.772376	0.411(
...	...	...	...	...	...	...	...	
13575	Wheelers Hill	1.111216	h	1245000.0	1.118210	1.123604	0.673367	0.411(

Random Forest Regressor Model

In [36]:
```
#Using target variable as price for property price prediction
target_column = 'Price'
X = dataf.drop(columns=[target_column])
y = dataf[target_column]
```

In [37]:

```
# One-hot encode categorical features
# Handle missing values
X_imputed = X.fillna(X.mean())
categorical_features = X_imputed.select_dtypes(include=['object']).
encoder = OneHotEncoder(handle_unknown='ignore', sparse=False)
X_encoded = pd.DataFrame(encoder.fit_transform(X_imputed[categorica
X_encoded.index = X_imputed.index
X_encoded.columns = encoder.get_feature_names(categorical_features)
X_numerical = X_imputed.drop(columns=categorical_features)
X_processed = pd.concat([X_numerical, X_encoded], axis=1)
```

```
/var/folders/zj/v2cmcjkn0_3gvj4qx06vp3p00000gn/T/ipykernel_40777/2
761222775.py:3: FutureWarning:

Dropping of nuisance columns in DataFrame reductions (with 'numeri
c_only=None') is deprecated; in a future version this will raise T
ypeError.  Select only valid columns before calling the reduction.

/Users/jeetpatel499/opt/anaconda3/lib/python3.9/site-packages/skle
arn/utils/deprecation.py:87: FutureWarning:

Function get_feature_names is deprecated; get_feature_names is dep
recated in 1.0 and will be removed in 1.2. Please use get_feature_
names_out instead.
```

In [38]:

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_processed, y,

# Train the Random Forest Regressor
rf_regressor = RandomForestRegressor(random_state=42)
rf_regressor.fit(X_train, y_train)
```

Out[38]:

```
RandomForestRegressor(random_state=42)
```

In [39]:

```
# Make predictions on the test set
y_pred = rf_regressor.predict(X_test)
```

In [40]:
```
# Create a DataFrame to display actual and predicted prices side by
predictions_df = pd.DataFrame({'Actual Price': y_test, 'Predicted P

# Sort DF by actual price vs predicted price
predictions_df.sort_values(by='Actual Price', inplace=True)

# Display the DataFrame
print(predictions_df)
```

```
       Actual Price  Predicted Price
8860       210000.0        225885.00
5512       215000.0        313680.00
2455       250000.0        332960.00
7168       250000.0        289580.00
11156      257500.0        276542.00
...             ...              ...
1602      4760000.0       3057705.00
3614      4850000.0       3672040.00
6345      5500000.0       4243560.00
251       5525000.0       4107828.88
3616      6500000.0       3523850.00

[2716 rows x 2 columns]
```

In [41]:
```
predictions_df.head()
```

Out[41]:

	Actual Price	Predicted Price
8860	210000.0	225885.0
5512	215000.0	313680.0
2455	250000.0	332960.0
7168	250000.0	289580.0
11156	257500.0	276542.0

In [42]:
```
# Calculate R-squared (R^2) score
r2 = r2_score(y_test, y_pred)

# Calculate Mean Absolute Percentage Error (MAPE)
mape = mean_absolute_percentage_error(y_test, y_pred) * 100

print(f"R-squared (R^2) Score: {r2:.2f}")
print(f"Mean Absolute Percentage Error (MAPE): {mape:.2f}%")
```

```
R-squared (R^2) Score: 0.82
Mean Absolute Percentage Error (MAPE): 15.00%
```

In [43]:
```
dataf
```

Out[43]:

	Suburb	Rooms	Type	Price	Distance	Bedroom2	Bathroom	Ca
0	Abbotsford	-0.981463	h	1480000.0	-1.301485	-0.947035	-0.772376	-0.62360
1	Abbotsford	-0.981463	h	1035000.0	-1.301485	-0.947035	-0.772376	-1.65825
2	Abbotsford	0.064876	h	1465000.0	-1.301485	0.088284	0.673367	-1.65825
3	Abbotsford	0.064876	h	850000.0	-1.301485	0.088284	0.673367	-0.62360
4	Abbotsford	1.111216	h	1600000.0	-1.301485	0.088284	-0.772376	0.41104
...	...	...	...	...	...	...	...	..
13575	Wheelers Hill	1.111216	h	1245000.0	1.118210	1.123604	0.673367	0.41104
13576	Williamstown	0.064876	h	1031000.0	-0.568761	0.088284	0.673367	0.41104
13577	Williamstown	0.064876	h	1170000.0	-0.568761	0.088284	0.673367	2.48033
13578	Williamstown	1.111216	h	2500000.0	-0.568761	1.123604	-0.772376	3.51498
13579	Yarraville	1.111216	h	1285000.0	-0.653961	1.123604	-0.772376	-0.62360

13580 rows × 15 columns

Random Forest Classifier for Price Prediction Class

In [44]:
```
# Calculate the mean price to determine the threshold for binary cl
threshold = dataf['Price'].mean()
# Convert 'Price' to binary classes 'low' and 'high'
dataf['Price_Class'] = dataf['Price'].apply(lambda x: 'low' if x <
```

In [45]:
```
# Separate the features (X) and target variable (y)
X = dataf.drop(columns=['Price_Class'])
y = dataf['Price_Class']
```

In [46]:

```python
# Label encode categorical features
label_encoder = LabelEncoder()
categorical_features = X.select_dtypes(include=['object']).columns
for feature in categorical_features:
    X[feature] = label_encoder.fit_transform(X[feature])
```

In [47]:

```python
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
```

In [48]:

```python
# Train the Random Forest Classifier
rf_classifier = RandomForestClassifier(random_state=42)
rf_classifier.fit(X_train, y_train)
```

Out[48]:

```
RandomForestClassifier(random_state=42)
```

In [49]:

```python
# Make predictions on the test set
y_pred = rf_classifier.predict(X_test)
# Create a DataFrame to display the predicted price classes
predicted_table = X_test.copy()
predicted_table['Predicted_Price_Class'] = y_pred
# Display the predicted table
print(predicted_table)
```

```
       Suburb     Rooms  Type      Price  Distance  Bedroom2  Bath
room  \
1061       41  0.064876     0  2600000.0  0.181004  0.088284  0.67
3367
6482      300 -0.981463     2   620000.0 -1.403726 -0.947035 -0.77
2376
8395      266  0.064876     2  1000000.0 -0.688041  0.088284 -0.77
2376
4659      229  0.064876     1   430000.0 -0.040517  0.088284  0.67
3367
7386      256 -0.981463     0   392250.0 -0.176838 -0.947035 -0.77
2376
...       ...       ...   ...        ...       ...       ...
...
10455     227  0.064876     0  1415000.0  1.936135  0.088284 -0.77
2376
3616      170  3.203895     0  6500000.0 -0.773242  3.194242  6.45
6335
577        21  2.157555     0  2450000.0 -0.074598  1.123604  0.67
3367
12620     293  3.203895     0  1155000.0  0.777408  3.194242  2.11
9109
4993      234  0.064876     0  1040000.0 -0.227958  0.088284 -0.77
2376
```

```
            Car  Landsize  BuildingArea  YearBuilt  Lattitude  Lon
gtitude  \
1061   1.445688  0.007414     -0.199838  -1.234959  -1.507721   -0
.030951
6482  -0.623608 -0.139936      0.008043   0.847829   0.001296   -0
.415893
8395  -1.658256 -0.139936      0.088190   0.772924  -0.789802   -0
.178191
4659  -0.623608 -0.103349      0.065649   0.851991   1.186051   -0
.583343
7386   0.411040 -0.024161     -0.199838   0.814538  -0.301517   -1
.190589
...         ...       ...           ...        ...        ...
...
10455  0.411040 -0.003362      0.163328   0.793731  -2.385499    0
.767515
3616   1.445688  0.194356      0.714338   0.731310   0.079523    0
.302986
577    0.411040  0.105897      0.451356   0.804134   0.042933    0
.649434
12620  1.445688  0.049012     -0.199838  -1.234959  -0.912567    2
.259743
4993   0.411040 -0.039949      0.103218   0.783327   0.916043    0
.106666

       Regionname  Propertycount Predicted_Price_Class
1061            5       0.713632                  high
6482            2      -1.193220                   low
8395            5       1.321386                   low
4659            2       0.006985                   low
7386            6      -1.477797                   low
...           ...            ...                   ...
10455           4      -0.540701                  high
3616            5       0.656991                  high
577             5      -0.404807                  high
12620           0      -0.085057                  high
4993            2       1.626747                   low

[2716 rows x 16 columns]
```

In [50]:
```
# Calculate the accuracy report
accuracy_report = classification_report(y_test, y_pred)

# Print the accuracy report
print("Accuracy Report:")
print(accuracy_report)
```

```
Accuracy Report:
              precision    recall  f1-score   support

        high       1.00      1.00      1.00      1041
         low       1.00      1.00      1.00      1675

    accuracy                           1.00      2716
   macro avg       1.00      1.00      1.00      2716
weighted avg       1.00      1.00      1.00      2716
```

In [51]:
```
# Create a count plot to visualize the predicted classes
sns.countplot(x='Predicted_Price_Class', data=predicted_table)

# Add labels and title
plt.xlabel('Predicted Price Class')
plt.ylabel('Count')
plt.title('Distribution of Predicted Price Classes')

# Show the plot
plt.show()
```

In [52]:
```
dataf
```

Out[52]:

	Suburb	Rooms	Type	Price	Distance	Bedroom2	Bathroom	Ca
0	Abbotsford	-0.981463	h	1480000.0	-1.301485	-0.947035	-0.772376	-0.62360
1	Abbotsford	-0.981463	h	1035000.0	-1.301485	-0.947035	-0.772376	-1.65825
2	Abbotsford	0.064876	h	1465000.0	-1.301485	0.088284	0.673367	-1.65825
3	Abbotsford	0.064876	h	850000.0	-1.301485	0.088284	0.673367	-0.62360
4	Abbotsford	1.111216	h	1600000.0	-1.301485	0.088284	-0.772376	0.41104
...	...	...	...	...	...	...	...	..
13575	Wheelers Hill	1.111216	h	1245000.0	1.118210	1.123604	0.673367	0.41104
13576	Williamstown	0.064876	h	1031000.0	-0.568761	0.088284	0.673367	0.41104
13577	Williamstown	0.064876	h	1170000.0	-0.568761	0.088284	0.673367	2.48033
13578	Williamstown	1.111216	h	2500000.0	-0.568761	1.123604	-0.772376	3.51498
13579	Yarraville	1.111216	h	1285000.0	-0.653961	1.123604	-0.772376	-0.62360

13580 rows × 16 columns

Neural Network

In [ ]:
```python
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers


# Separate features (X) and target (y)
X = dataf.drop(['Price'], axis=1)
y = dataf['Price']

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
```

In [ ]:
```
# List of numeric features
numeric_features = ['Rooms', 'Distance', 'Bedroom2', 'Bathroom', 'C

# Standardize numeric features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train[numeric_features])
X_test_scaled = scaler.transform(X_test[numeric_features])
```

In [ ]:
```
# Build and train the neural network model
model = keras.Sequential([
    layers.Input(shape=(X_train_scaled.shape[1],)),  # Input layer
    layers.Dense(64, activation='relu'),             # Hidden layer
    layers.Dense(32, activation='relu'),             # Additional h
    layers.Dense(1)                                 # Output layer
])
```

In [ ]:
```
# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')

# Train the model
model.fit(X_train_scaled, y_train, epochs=100, batch_size=32, valid
```

In [ ]:
```
from sklearn.metrics import mean_squared_error

# Calculate RMSE on the test set
y_pred = model.predict(X_test_scaled)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
print(f"Root Mean Squared Error (RMSE): {rmse}")
```

In [ ]:
```
# Print the first few predicted prices along with corresponding act
num_samples_to_display = 10
for i in range(num_samples_to_display):
    print(f"Predicted Price: {predictions[i][0]:.2f} - Actual Price

# Show the model's summary
model.summary()
```

In [ ]:

```python
# Extract the first ten predicted and actual prices
predicted_prices = predictions[:10]
actual_prices = y_test.iloc[:10]

# Create a scatter plot
plt.figure(figsize=(10, 6))
plt.scatter(actual_prices, predicted_prices, color='blue')
plt.plot([min(actual_prices), max(actual_prices)], [min(actual_pric
plt.xlabel('Actual Price')
plt.ylabel('Predicted Price')
plt.title('Actual Price vs. Predicted Price')
plt.show()
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: