Table of Contents

Project title

Housing market price prediction

Tutor name - Mr. Hao Zhang

Group Identities

IVAN SOO HOO (103798094)

CHHUNHEANG CHHORN (104849320)

AARYAN BHATI (104189110)

Introduction

Project description – The goal of this machine learning research is to help people and organizations overcome the difficulties they encounter in comprehending and forecasting Melbourne real estate values. The housing market is influenced by numerous factors such as location, property features, socio-economic conditions, and local amenities. This complexity often makes it difficult for potential homebuyers, real estate investors, and policymakers to make informed decisions.

These intended users are:

1. **Homebuyers**: People wishing to acquire residential homes who want accurate price forecasts in order to make wise investment decisions.
2. **Real estate investors**: These are investors looking to reduce market volatility risks and pinpoint high-return locations.
3. **Real estate professionals** :are consultants and agents who need correct data in order to provide clients with well-informed advice.
4. **Policymakers**: Those with an emphasis on urban planning and affordable housing who stand to gain from understanding market dynamics.

Problem Framing

The research intends to address the difficulty of precisely projecting property values in Melbourne, a market that is impacted by a variety of dynamic elements, including property attributes, location, socioeconomic circumstances, and ease of access to facilities. Existing approaches, such as historical trend analysis and simple statistical models, frequently fail to capture these intricate interdependencies, resulting in forecasts that are not as good as they may be.

A machine learning approach is more suitable because it can analyze vast amounts of data, identify hidden patterns and learn from multiple features to provide more accurate predictions. By leveraging machine learning models, this project aims to develop a robust predictive tool that can deliver high-accuracy price forecasts and trend analyses, offering users better insights into the housing market dynamics. For various consumers like Home buyers, real estate agents and investors.

Data Collection

This project's data was gathered from two primary sources to ensure a thorough analysis of the Melbourne property market. The first source, the Crime Statistics Agency of Victoria, gave specific crime statistics by location, which is critical for understanding the socioeconomic elements that may affect property prices. This dataset contains information on crime occurrences and victimization rates from several local government areas, which adds a valuable layer to the housing data study.

The second source, Kaggle's Melbourne Housing Snapshot, provides a complete dataset of property sales in Melbourne, containing parameters such as suburb, address, number of rooms, house type, price, and other property-specific features. This information was integrated to produce a cohesive image, allowing for an in-depth investigation of how various factors such as crime rates and property features affect housing prices.

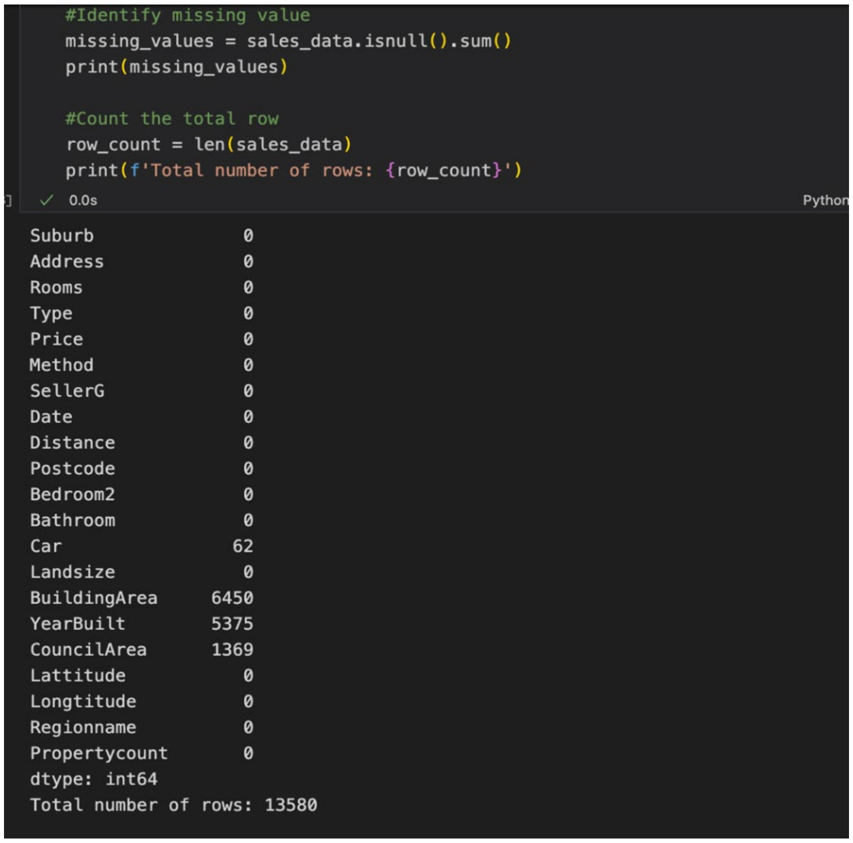
During data gathering, there were very few public data available for our use as their 3-4 companies that own all data and use it for the own use or provide paid API tools to use their data set. But there are government data that are available but difficult to locate without certain information. Which we when find there were discrepancies and formatting difficulties between the two sources. To solve these issues, data cleaning and transformation techniques were used, such as resolving missing values, standardizing formats, and merging datasets based on common variables such as suburb and date. This ensured that the final dataset was relevant, consistent, and sufficient for developing an accurate prediction model. Detailed info about data Analysis is in the Appendix item 1.

Data Processing

1. Melbourne Housing Data Set

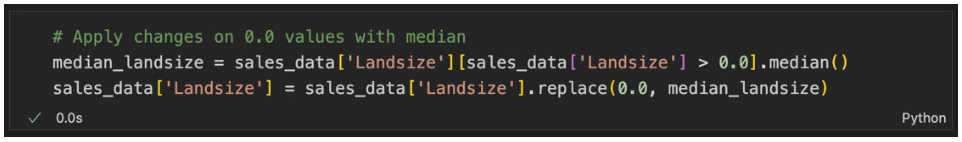
a. Handling missing values:

- There are missing value columns such as Building Area, Car, Year Built. Since these columns have too null value, we decided to drop it because we thought that we have other columns that have enough value that could influence the model for price prediction.
- We also fill the missing value of Council Area with the word "Unknown" since we will use that columns to merge with crime dataset.
- We also identified that there are some 0 value in the Land size column and we decided to replace those value with median value method.



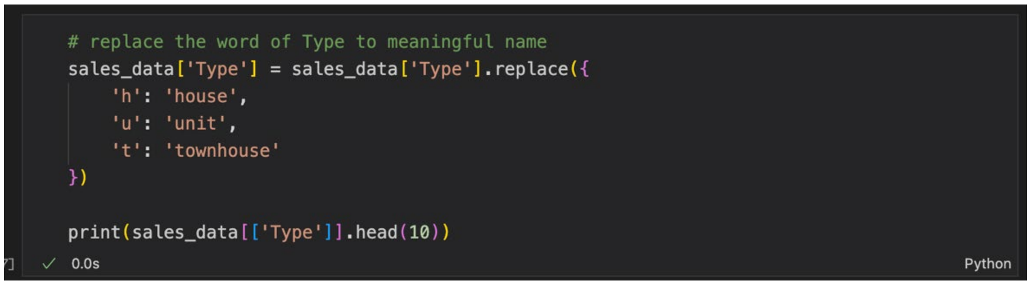
```
#Identify missing value
missing_values = sales_data.isnull().sum()
print(missing_values)

#Count the total row
row_count = len(sales_data)
print(f'Total number of rows: {row_count}')
```

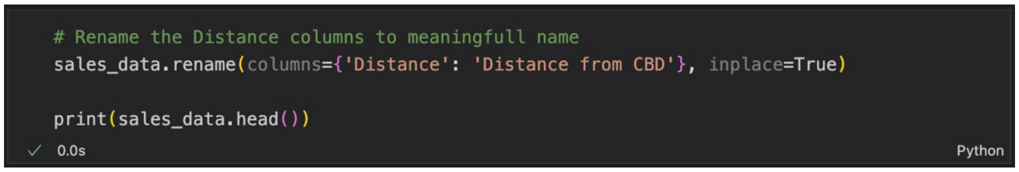
```
Suburb              0
Address             0
Rooms               0
Type                0
Price               0
Method              0
SellerG             0
Date                0
Distance            0
Postcode            0
Bedroom2            0
Bathroom            0
Car                62
Landsize            0
BuildingArea     6450
YearBuilt        5375
CouncilArea      1369
Lattitude           0
Longtitude          0
Regionname          0
Propertycount       0
dtype: int64
Total number of rows: 13580
```


```
# Apply changes on 0.0 values with median
median_landsize = sales_data['Landsize'][sales_data['Landsize'] > 0.0].median()
sales_data['Landsize'] = sales_data['Landsize'].replace(0.0, median_landsize)
```
✓ 0.0s Python

b. Formatting:

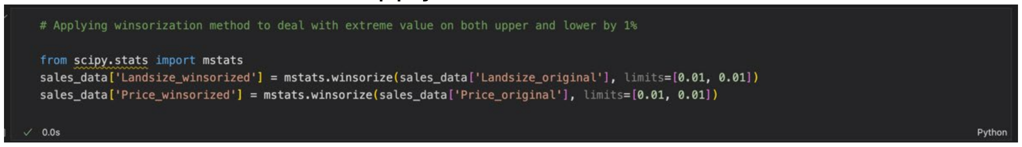
- We also rename a Distance and Type columns to a more meaningful name


```
# replace the word of Type to meaningful name
sales_data['Type'] = sales_data['Type'].replace({
    'h': 'house',
    'u': 'unit',
    't': 'townhouse'
})

print(sales_data[['Type']].head(10))
```
✓ 0.0s Python

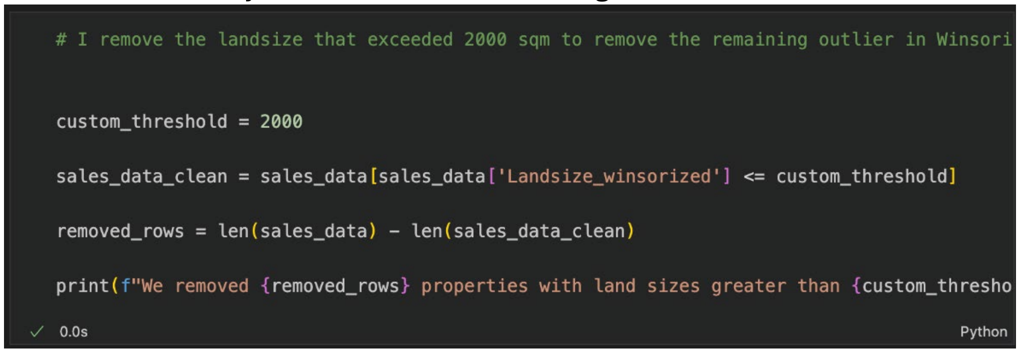

```
# Rename the Distance columns to meaningfull name
sales_data.rename(columns={'Distance': 'Distance from CBD'}, inplace=True)

print(sales_data.head())
```
✓ 0.0s Python

c. Data Transformation:

- There are also some extreme values of Land Size and Price in our dataset, so I decided to use a method to mitigate outliner called **Winsorization** method to apply on our Land Size and Price.

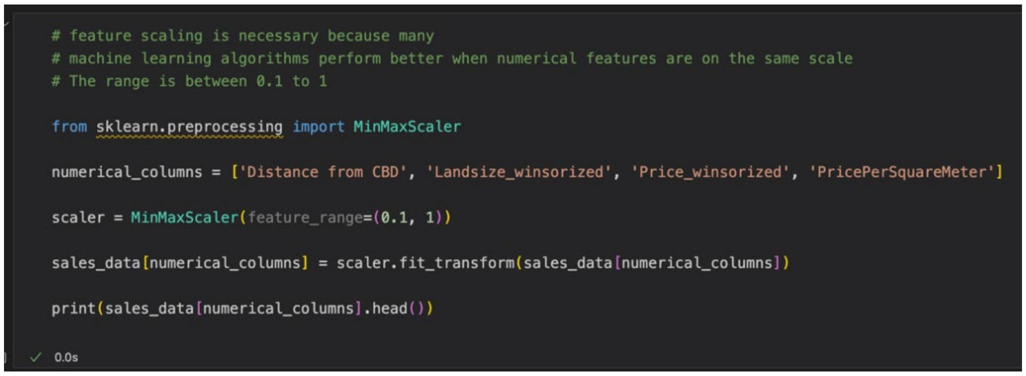

```
# Applying winsorization method to deal with extreme value on both upper and lower by 1%

from scipy.stats import mstats
sales_data['Landsize_winsorized'] = mstats.winsorize(sales_data['Landsize_original'], limits=[0.01, 0.01])
sales_data['Price_winsorized'] = mstats.winsorize(sales_data['Price_original'], limits=[0.01, 0.01])
```
✓ 0.0s Python

- After the Winsorization method, there are still few extreme values in the Land Size column. Our solution is to remove any row in our data set that have Land Size exceeding 2000 square per meter in order to maintain accuracy when it comes to training the model.


```
# I remove the landsize that exceeded 2000 sqm to remove the remaining outlier in Winsori


custom_threshold = 2000

sales_data_clean = sales_data[sales_data['Landsize_winsorized'] <= custom_threshold]

removed_rows = len(sales_data) - len(sales_data_clean)

print(f"We removed {removed_rows} properties with land sizes greater than {custom_thresho
```
✓ 0.0s Python

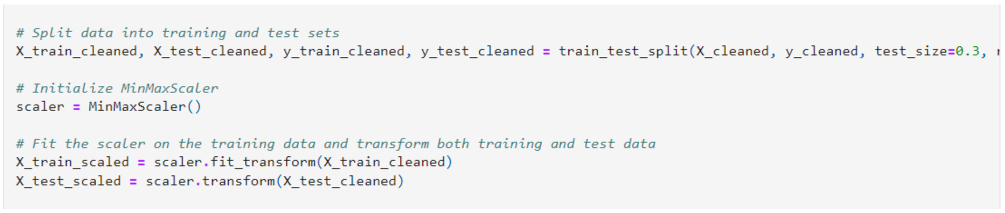
d. Normalization:

- We decided to use a normalization technique called Min-Max scaling that transforms Distance From CBD, Land Size and Price and Price Per Square Meter columns to a fixed range between 0.1 to 1. We do this because machine learning algorithms are sensitive to the range of data value



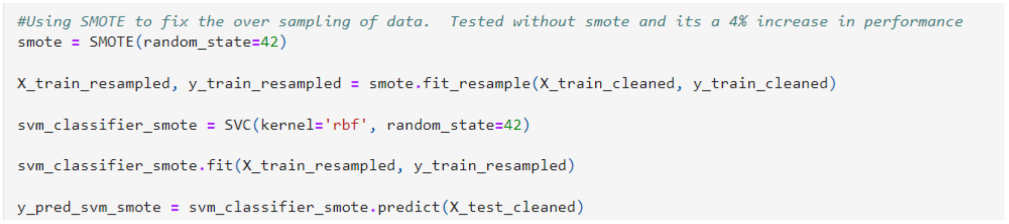
```python
# feature scaling is necessary because many
# machine learning algorithms perform better when numerical features are on the same scale
# The range is between 0.1 to 1

from sklearn.preprocessing import MinMaxScaler

numerical_columns = ['Distance from CBD', 'Landsize_winsorized', 'Price_winsorized', 'PricePerSquareMeter']

scaler = MinMaxScaler(feature_range=(0.1, 1))

sales_data[numerical_columns] = scaler.fit_transform(sales_data[numerical_columns])

print(sales_data[numerical_columns].head())
```
✓ 0.0s

- We also used min max scaling to transform the data that we are going to use for testing and training.



```python
# Split data into training and test sets
X_train_cleaned, X_test_cleaned, y_train_cleaned, y_test_cleaned = train_test_split(X_cleaned, y_cleaned, test_size=0.3, 

# Initialize MinMaxScaler
scaler = MinMaxScaler()

# Fit the scaler on the training data and transform both training and test data
X_train_scaled = scaler.fit_transform(X_train_cleaned)
X_test_scaled = scaler.transform(X_test_cleaned)
```

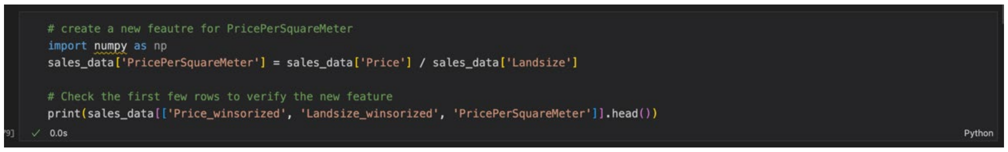
- We also use SMOTE in our implementation of SVC so that it increases the performance of the model due to issues with over sampling this model has had previously



```python
#Using SMOTE to fix the over sampling of data.  Tested without smote and its a 4% increase in performance
smote = SMOTE(random_state=42)

X_train_resampled, y_train_resampled = smote.fit_resample(X_train_cleaned, y_train_cleaned)

svm_classifier_smote = SVC(kernel='rbf', random_state=42)

svm_classifier_smote.fit(X_train_resampled, y_train_resampled)

y_pred_svm_smote = svm_classifier_smote.predict(X_test_cleaned)
```

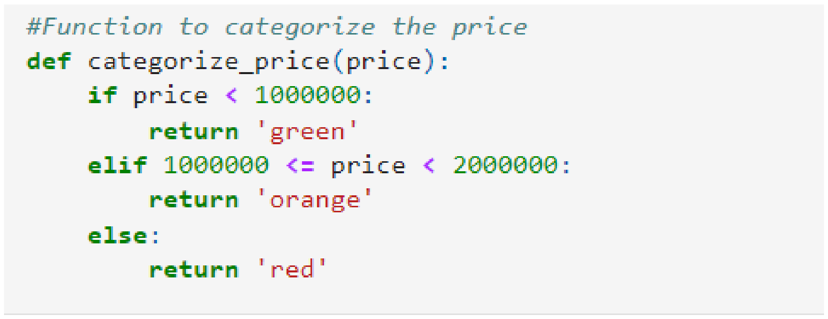
e. Feature Engineering:

- We also make a new feature called Price Per Square Meter to used it for our model training



```python
# create a new feautre for PricePerSquareMeter
import numpy as np
sales_data['PricePerSquareMeter'] = sales_data['Price'] / sales_data['Landsize']

# Check the first few rows to verify the new feature
print(sales_data[['Price_winsorized', 'Landsize_winsorized', 'PricePerSquareMeter']].head())
```
✓ 0.0s

- We changed the Price variable into a categorical data for training our classifiers



```python
#Function to categorize the price
def categorize_price(price):
    if price < 1000000:
        return 'green'
    elif 1000000 <= price < 2000000:
        return 'orange'
    else:
        return 'red'
```

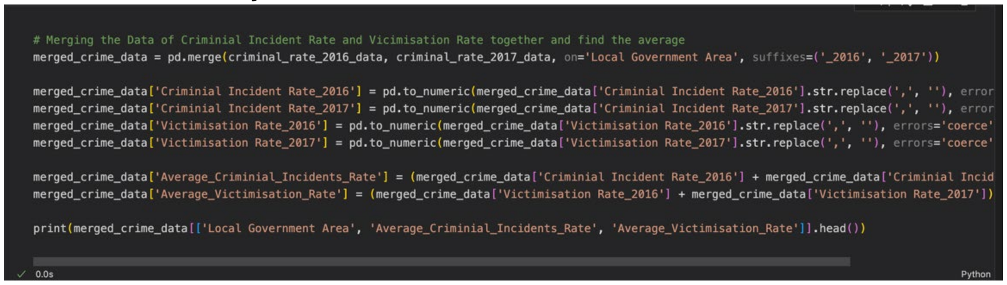
2. Crime Data Set

a. Dropping value:

- For the crime data set, we don't really need to do any cleaning because we only need the crime rate and the Local council area. We decide to drop all the unnecessary columns, so that we can merge the data.

3. Joining and Merging Process

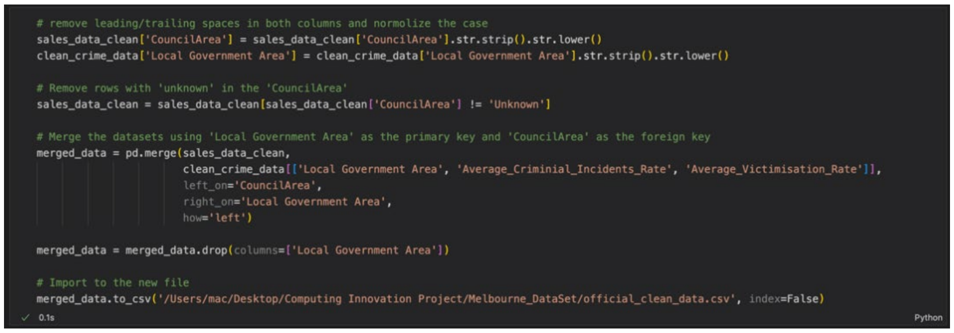
a. First, we merged our Criminal Incident Rate and Victimisation Rate between two data set of crime into average by using common column called Local Government Area. After the merging process, we around up the average rate value to allow only 2 decimal values.



```python
# Merging the Data of Criminial Incident Rate and Vicimisation Rate together and find the average
merged_crime_data = pd.merge(criminal_rate_2016_data, criminal_rate_2017_data, on='Local Government Area', suffixes=('_2016', '_2017'))

merged_crime_data['Criminial Incident Rate_2016'] = pd.to_numeric(merged_crime_data['Criminial Incident Rate_2016'].str.replace(',', ''), error
merged_crime_data['Criminial Incident Rate_2017'] = pd.to_numeric(merged_crime_data['Criminial Incident Rate_2017'].str.replace(',', ''), error
merged_crime_data['Victimisation Rate_2016'] = pd.to_numeric(merged_crime_data['Victimisation Rate_2016'].str.replace(',', ''), errors='coerce'
merged_crime_data['Victimisation Rate_2017'] = pd.to_numeric(merged_crime_data['Victimisation Rate_2017'].str.replace(',', ''), errors='coerce'

merged_crime_data['Average_Criminial_Incidents_Rate'] = (merged_crime_data['Criminial Incident Rate_2016'] + merged_crime_data['Criminial Incid
merged_crime_data['Average_Victimisation_Rate'] = (merged_crime_data['Victimisation Rate_2016'] + merged_crime_data['Victimisation Rate_2017'])

print(merged_crime_data[['Local Government Area', 'Average_Criminial_Incidents_Rate', 'Average_Victimisation_Rate']].head())
```

✓ 0.0s Python

b. After that, we merged the crime data set that have Local Government Area, Average Criminal Incidents Rate and Average Victimisation Rate columns into our main data set by Local Government Area from Crime dataset as the primary key and Council Area as a foreign key.

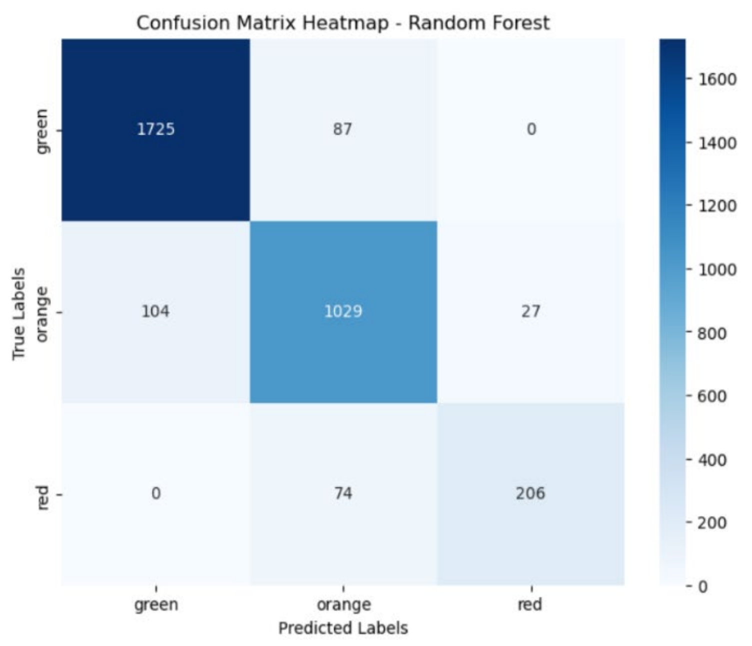


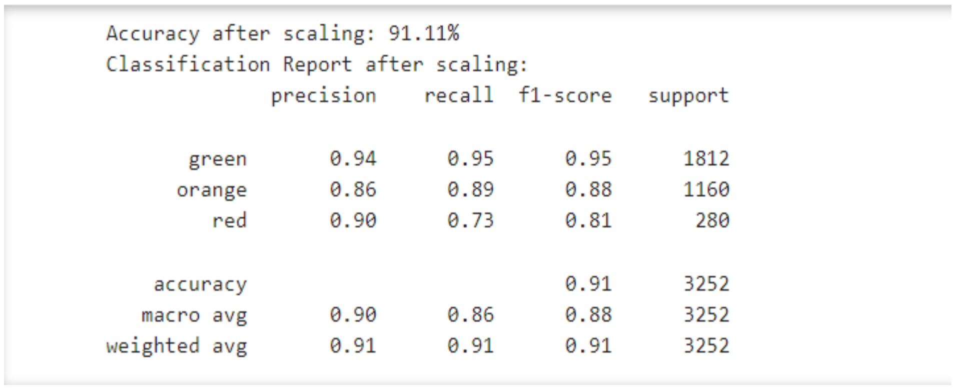
```python
# remove leading/trailing spaces in both columns and normolize the case
sales_data_clean['CouncilArea'] = sales_data_clean['CouncilArea'].str.strip().str.lower()
clean_crime_data['Local Government Area'] = clean_crime_data['Local Government Area'].str.strip().str.lower()

# Remove rows with 'unknown' in the 'CouncilArea'
sales_data_clean = sales_data_clean[sales_data_clean['CouncilArea'] != 'Unknown']

# Merge the datasets using 'Local Government Area' as the primary key and 'CouncilArea' as the foreign key
merged_data = pd.merge(sales_data_clean,
                       clean_crime_data[['Local Government Area', 'Average_Criminial_Incidents_Rate', 'Average_Victimisation_Rate']],
                       left_on='CouncilArea',
                       right_on='Local Government Area',
                       how='left')

merged_data = merged_data.drop(columns=['Local Government Area'])

# Import to the new file
merged_data.to_csv('/Users/mac/Desktop/Computing Innovation Project/Melbourne_DataSet/official_clean_data.csv', index=False)
```
✓ 0.1s Python

Machine Learning, Model Selection and Implementation Evaluation

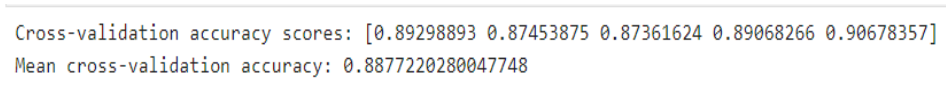
1. **Random Forest Classifier**
    a. We used this model due to the complexity of the housing market and how many different influences there are on the market itself such as crime rates, location from CBD, number of rooms and neighborhood characteristics.
    b. Random Forest is perfect for this problem due to how it uses an ensemble of decision trees each capturing different data providing a more complete understanding of the relationship between the features and prices
    c. This dataset contains a mixture of numerical and categorical features and random forest can naturally handle both types of data which makes it ideal.
    d. Its robustness to missing values and outliers is critical in the real estate domain, where data may have quality issues.
    e. The model provides good generalization while being scalable, making it suitable for the dynamic and evolving housing market.

2. **Model evaluation**





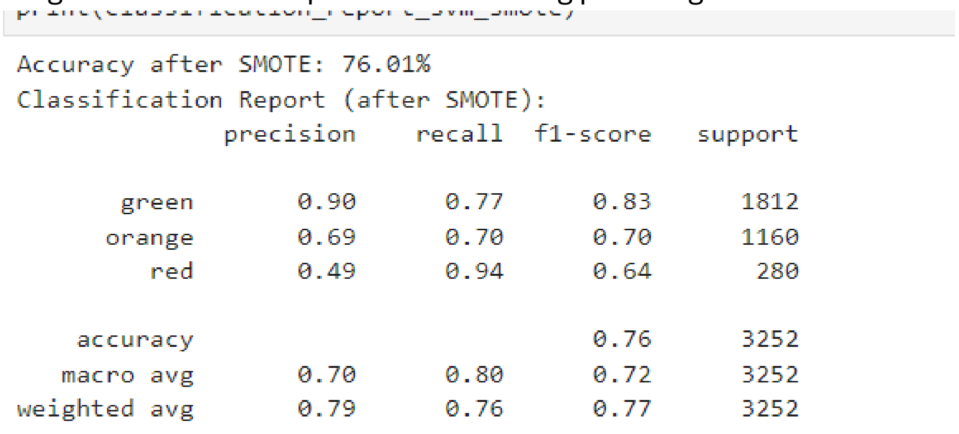
```
Accuracy after scaling: 91.11%
Classification Report after scaling:
              precision    recall  f1-score   support

       green       0.94      0.95      0.95      1812
      orange       0.86      0.89      0.88      1160
         red       0.90      0.73      0.81       280

    accuracy                           0.91      3252
   macro avg       0.90      0.86      0.88      3252
weighted avg       0.91      0.91      0.91      3252
```



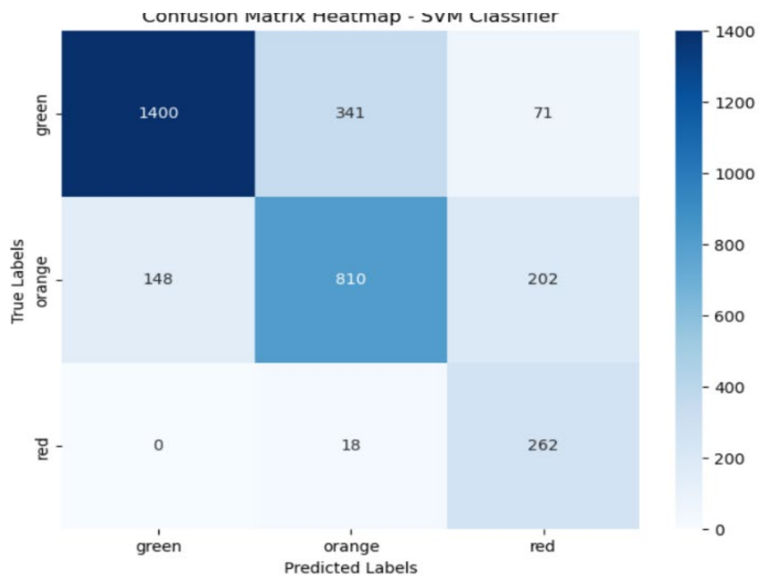
```
Cross-validation accuracy scores: [0.89298893 0.87453875 0.87361624 0.89068266 0.90678357]
Mean cross-validation accuracy: 0.8877220280047748
```

3. **SVM (Support Vector Machine)**
    a. SVC was chosen for its ability to handle high-dimensional data and non-linear relationships effectively, which are inherent characteristics of housing market datasets.
    b. Its robustness against outliers, ability to generalize well, and the use of kernel functions make it a strong candidate for predicting price categories, particularly when housing prices depend on multiple complex and interacting features.
    c. Handling of class imbalance, through methods like SMOTE, ensures that the model provides good performance across all price segments, making it a reliable tool for the housing market analysis.

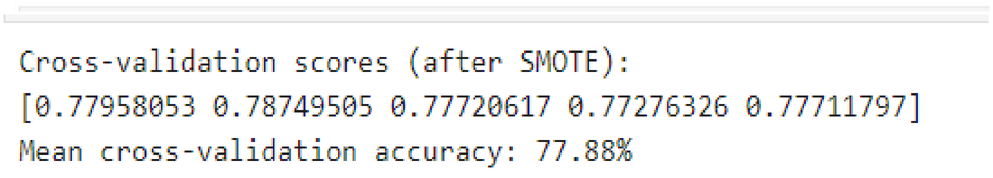
d. Optimal decision boundaries and generalization capability help SVC achieve accurate classification for properties across different price categories, making it suitable for the problem of housing price segmentation



```
Accuracy after SMOTE: 76.01%
Classification Report (after SMOTE):
              precision    recall  f1-score   support

       green       0.90      0.77      0.83      1812
      orange       0.69      0.70      0.70      1160
         red       0.49      0.94      0.64       280

    accuracy                           0.76      3252
   macro avg       0.70      0.80      0.72      3252
weighted avg       0.79      0.76      0.77      3252
```



e.



```
Cross-validation scores (after SMOTE):
[0.77958053 0.78749505 0.77720617 0.77276326 0.77711797]
Mean cross-validation accuracy: 77.88%
```

f.

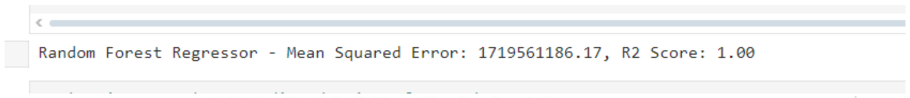
4. Random Forest Regression Model

a. Random Forest Regression was chosen for its ability to handle non-linear relationships, its robustness to outliers and noise, and its versatility in dealing with high-dimensional and mixed-type data.

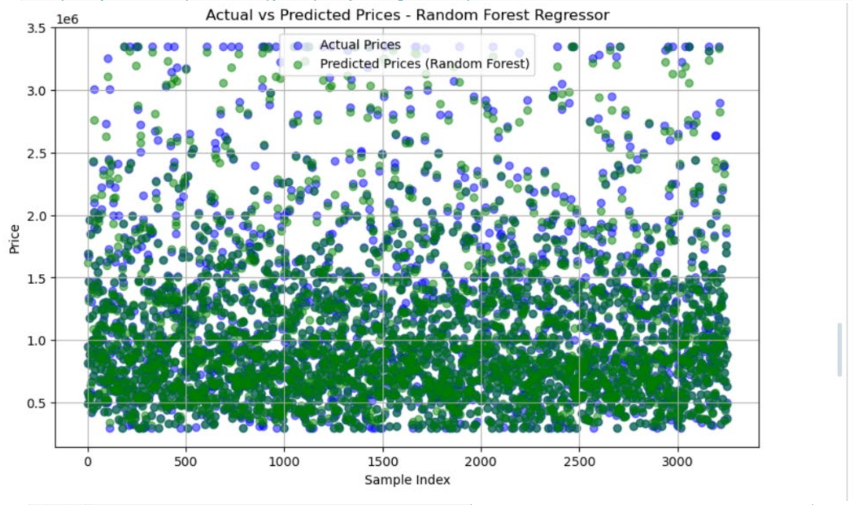
b. It is well-suited to the problem context due to its capability to capture complex interactions between features, provide feature importance, and generalize well across different types of properties in the housing market.

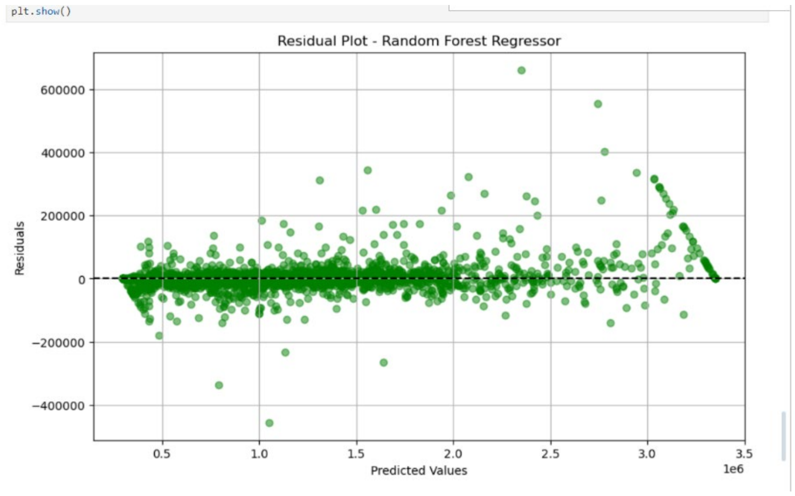
c. The bagging approach helps reduce overfitting and ensures stability in predictions, which is crucial for developing a reliable model for the real estate sector.

d. Model Evaluation

e. 

```
Random Forest Regressor - Mean Squared Error: 1719561186.17, R2 Score: 1.00
```



f. 

```
plt.show()
```

g.

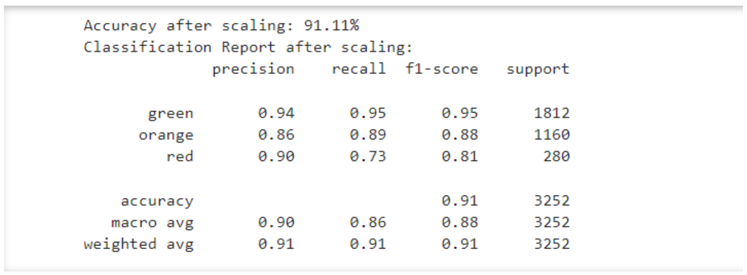
Model Comparison

Classifiers. SVC (Support Vector Classifier) Vs **RF ()** VS **SGD** VS **MLP**

1. We chose Random Forest and SVC as our implementations for classifiers in the end due to the accuracy in which SGD and MLP predict. We performed a comparison of these 4 models using the classification report as well as Accuracy Scores to determine which classifiers we wanted to pick.

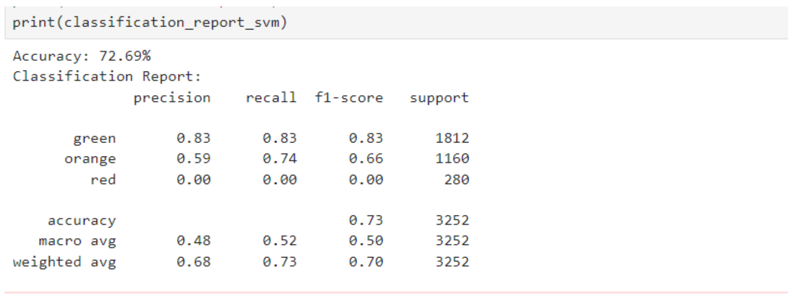
2. **Classifiers** simplify the housing price prediction problem by dividing it into **price categories**, making it more interpretable and actionable for real estate stakeholders.
3. **Handling Outliers and Noise**: Classifiers are less affected by extreme values and data noise, leading to more stable predictions, which is particularly useful in the housing market where outliers and noisy features are common.
4. **Adaptability and Reduced Complexity**: Classifiers provide **high-level segmentation** that can adapt well to **market changes**, reducing the complexity of predictions while still offering valuable insights.
5. **Facilitated Market Analysis** and **User Experience**: Classifiers make it easier to conduct market analyses and provide a **user-friendly interaction** through web applications by focusing on segmented insights rather than complex, continuous values.

Random Forest Classifier

a. 

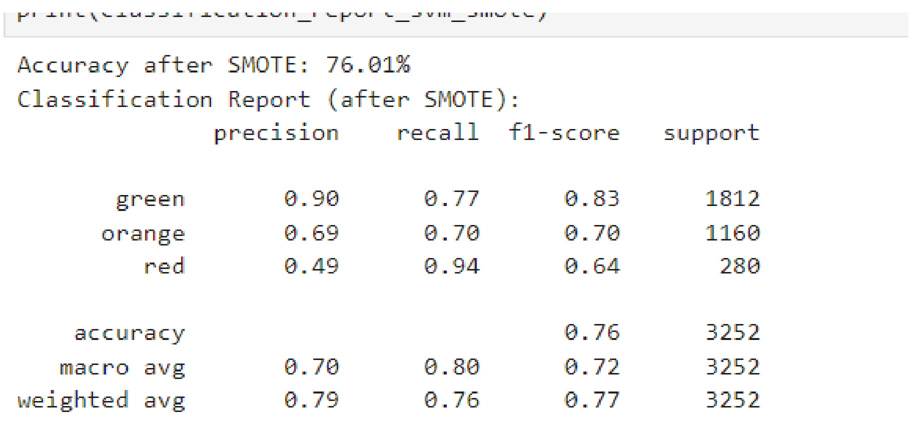
```
Accuracy after scaling: 91.11%
Classification Report after scaling:
              precision    recall  f1-score   support

       green       0.94      0.95      0.95      1812
      orange       0.86      0.89      0.88      1160
         red       0.90      0.73      0.81       280

    accuracy                           0.91      3252
   macro avg       0.90      0.86      0.88      3252
weighted avg       0.91      0.91      0.91      3252
```

SVC Classifier



```
print(classification_report_svm)
```

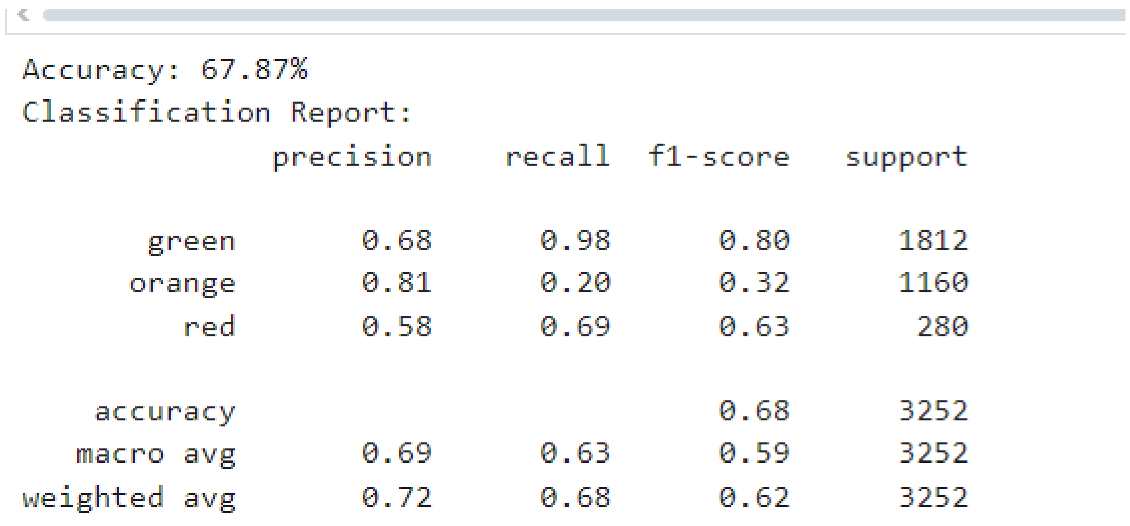
```
Accuracy: 72.69%
Classification Report:
              precision    recall  f1-score   support

       green       0.83      0.83      0.83      1812
      orange       0.59      0.74      0.66      1160
         red       0.00      0.00      0.00       280

    accuracy                           0.73      3252
   macro avg       0.48      0.52      0.50      3252
weighted avg       0.68      0.73      0.70      3252
```



```
Accuracy after SMOTE: 76.01%
Classification Report (after SMOTE):
              precision    recall  f1-score   support

       green       0.90      0.77      0.83      1812
      orange       0.69      0.70      0.70      1160
         red       0.49      0.94      0.64       280

    accuracy                           0.76      3252
   macro avg       0.70      0.80      0.72      3252
weighted avg       0.79      0.76      0.77      3252
```

SVM Classifier

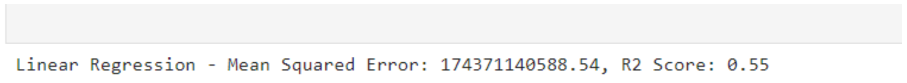
MLP Classifier



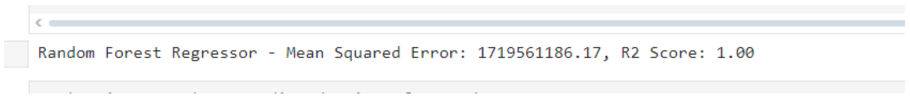
```
Accuracy: 67.87%
Classification Report:
              precision    recall  f1-score   support

       green       0.68      0.98      0.80      1812
      orange       0.81      0.20      0.32      1160
         red       0.58      0.69      0.63       280

    accuracy                           0.68      3252
   macro avg       0.69      0.63      0.59      3252
weighted avg       0.72      0.68      0.62      3252
```

From the images its very apparent that the Random Forest classifier and SVC Classifiers have the highest accuracy and best classifications reports. We further justify the picking of the Random Forest Classifier and SVC Classifiers with heatmaps to see how accurate they are when predicting price. We also performed cross validation scores to check accuracy, and both the random forest and SVC performed very well.

The last model we chose is the random forest regression model. Tests done on linear regression model vs Random Forest regression model. From what can be seen from using Mean squared error and R2 Score to check performance of Linear regression it has a lower R2 score, and it has a higher mean squared error number.

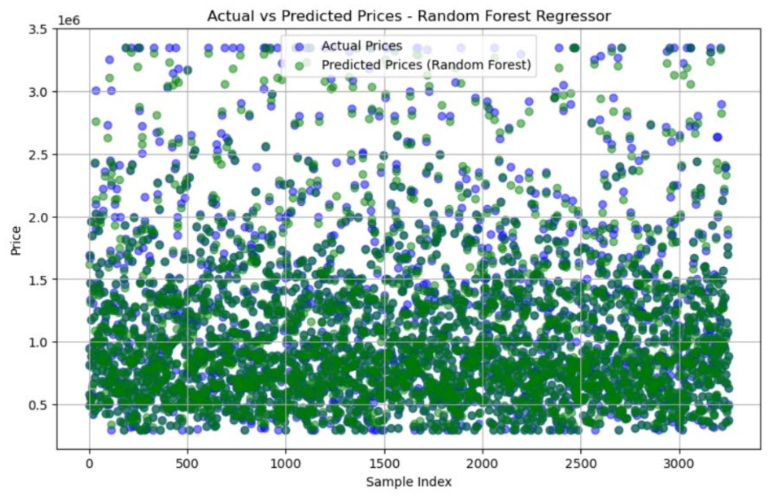


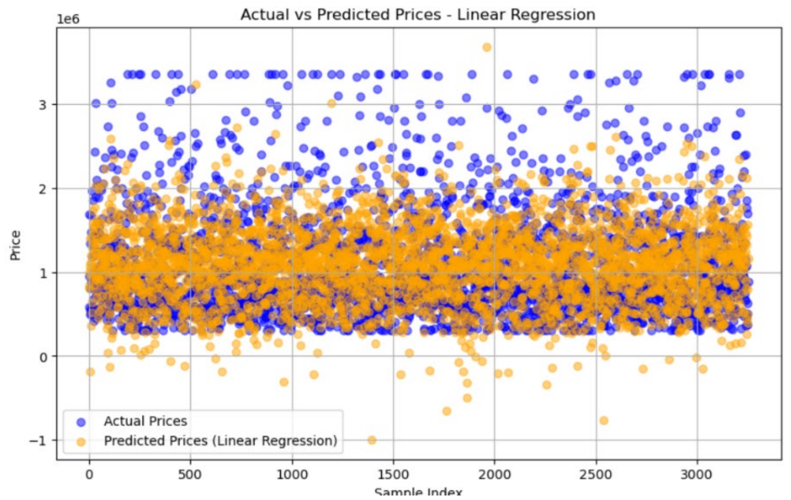
```
Linear Regression - Mean Squared Error: 174371140588.54, R2 Score: 0.55
```



```
Random Forest Regressor - Mean Squared Error: 1719561186.17, R2 Score: 1.00
```

From checking the Scatter plots it tends to congregate all plots into the center whereas there are multiple real data that is way above the predicted price. It also under predicts the prices more often than the regression model.


Actual vs Predicted Prices - Random Forest Regressor
1e6
3.5
3.0
2.5
2.0
1.5
1.0
0.5
Price
0
500
1000
1500
2000
2500
3000
Sample Index
Actual Prices
Predicted Prices (Random Forest)


Actual vs Predicted Prices - Linear Regression
1e6
3
2
1
0
−1
Price
0
500
1000
1500
2000
2500
3000
Sample Index
Actual Prices
Predicted Prices (Linear Regression)

Technical Implementation

In the implementation, the following libraries and tools were used:

1. **pandas**:
    - A powerful library for data manipulation and analysis. It is used for handling and analyzing structured data.
    - Typical operations involve reading data, data cleaning, and data frame manipulation.
2. **scikit-learn**:
    - A widely used machine learning library offering tools for data mining and analysis. It includes a variety of classification, regression, and clustering algorithms.
    - **Key modules used**:
        - train_test_split: Splits the dataset into training and test sets.
        - RandomForestClassifier: An ensemble classifier that uses multiple decision trees to improve accuracy and control over-fitting.
        - classification_report, accuracy_score, confusion_matrix: Metrics for evaluating classification models.
        - cross_val_score: Used to evaluate models using cross-validation.
        - SVC: Support Vector Classifier, part of the SVM (Support Vector Machine) family, used for classification tasks.
        - RandomForestRegressor, GradientBoostingRegressor: Ensemble learning models used for regression tasks.
        - mean_squared_error, r2_score: Metrics for evaluating regression models.
        - MLPClassifier: Multilayer Perceptron, a neural network-based model for classification tasks.
        - LabelEncoder, MinMaxScaler, StandardScaler: Tools for preprocessing data (label encoding and feature scaling).
3. **imbalanced-learn (imblearn)**:

    - A library to handle imbalanced datasets.
    - **SMOTE** (Synthetic Minority Over-sampling Technique): Used to address class imbalance by generating synthetic samples.

4. **matplotlib** and **seaborn**:
    - Visualization libraries used to create plots and graphs.
    - matplotlib is the core plotting library, while seaborn provides a high-level interface for drawing attractive statistical graphics.
5. **numpy**:
    - A fundamental package for scientific computing with Python. It is mainly used for handling arrays and mathematical operations.
6. **scipy**:
    - Another scientific computing library.
    - mstats from scipy.stats: Used for statistical operations.

This project was challenging primarily due to the extensive data wrangling required. As it was my first time working with AI, I faced difficulties in **data preprocessing and visualization**, particularly in handling missing values, outliers, and mixed data types. Understanding how to clean and prepare the data for modeling was a steep learning curve. Additionally, **training the model** and evaluating its performance were confusing at first due to the complexity of the data. However, these challenges provided valuable insights into how machine learning models work and improved my skills in AI and data analysis.

Conclusion

The research successfully produced a machine learning system for projecting Melbourne home values, taking into account the complex affects of geography, crime rates, property attributes, and socioeconomic factors. For price segmentation, the Random Forest and SVC classifiers outperformed Linear Regression, whereas Random Forest Regression produced more accurate continuous price predictions. Key lessons include the relevance of feature selection and dealing with outliers in boosting model accuracy. This project provides useful tools for homebuyers, investors, and legislators, allowing them to make smarter real estate decisions. Future work should look into adding more socioeconomic variables to improve forecast accuracy and model applicability.

Bibliography

1. Crime Statistics Agency Victoria (2024). *Latest crime data by area*. [online] www.crimestatistics.vic.gov.au. Available at: https://www.crimestatistics.vic.gov.au/crime-statistics/latest-crime-data-by-area

2. www.kaggle.com. (n.d.). *Melbourne Housing Snapshot*. [online] Available at: https://www.kaggle.com/datasets/dansbecker/melbourne-housing-snapshot

Appendix

1. Data_Analysis Link - CTIP
2. Data_Cleaning 1 Link - CTIP
3. https://github.com/IvanSooHoo/COS30049-Assignment2