0kuuou0vg

January 25, 2025

[ ]:
```
#     FINAL PROJECT REPORT


#  Melbourne Housing Market Prediction Project Report


#  Project Overview

# - Predicted house prices in Melbourne using Linear Regression model
# - Dataset: Melbourne Housing Market dataset
# - Features: Rooms, Distance, Postcode
# - Target: Price


# Methodology

#1. Data Preprocessing:
#    - Handled missing values using median imputation
#    - Standardized 'Price' column using StandardScaler
# 2. Model Selection:
#    - Chosen Linear Regression model for prediction
# 3. Model Evaluation:
#    - Split data into training (80%) and testing sets (20%)
#    - Evaluated model using Mean Absolute Error (MAE), Mean Squared Error␣
 ↪(MSE), Root Mean Squared Error (RMSE), R-squared (R2)


# Results

# - Model Performance Metrics:
#   - MAE: 0.507
#   - MSE: 0.590
#   - RMSE: 0.766
#   - R2: 0.262
#   - Accuracy: 26.20
#   - Predicted Values Plot:
#   - Actual vs predicted house values
```

```
#   - Actual vs Predicted values plot with best fit line

#           Conclusion
# - Linear Regression model predicted house prices with reasonable accuracy (24.
↪1%)
# - Model performance can be improved by feature engineering, hyperparameter␣
↪tuning, and exploring other algorithms
# - Project demonstrates ability to predict continuous outcomes using linear␣
↪regression analysis
```

[162]:
```
#importing the all the packages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error,  r2_score
```

[80]:
```
#loading the datset
data = pd.read_csv(r"C:\Users\91703\Downloads\ML - 1 - DATASET.csv")
```

[81]:
```
#to get the top rows in the data
data.head()
```

[81]:
```
         Suburb           Address  Rooms Type      Price Method   SellerG  \
0    Abbotsford      49 Lithgow St      3    h  1490000.0      S    Jellis
1    Abbotsford      59A Turner St      3    h  1220000.0      S  Marshall
2    Abbotsford      119B Yarra St      3    h  1420000.0      S    Nelson
3    Aberfeldie         68 Vida St      3    h  1515000.0      S     Barry
4  Airport West  92 Clydesdale Rd      2    h   670000.0      S    Nelson

         Date  Postcode             Regionname  Propertycount  Distance  \
0  01-04-2017      3067  Northern Metropolitan           4019       3.0
1  01-04-2017      3067  Northern Metropolitan           4019       3.0
2  01-04-2017      3067  Northern Metropolitan           4019       3.0
3  01-04-2017      3040   Western Metropolitan           1543       7.5
4  01-04-2017      3042   Western Metropolitan           3464      10.4

                 CouncilArea
0          Yarra City Council
1          Yarra City Council
2          Yarra City Council
3  Moonee Valley City Council
4  Moonee Valley City Council
```

[82]:
```
#to get the total information of the data
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 63021 entries, 0 to 63020
Data columns (total 13 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Suburb         63021 non-null  object
 1   Address        63021 non-null  object
 2   Rooms          63021 non-null  int64
 3   Type           63021 non-null  object
 4   Price          48432 non-null  float64
 5   Method         63021 non-null  object
 6   SellerG        63021 non-null  object
 7   Date           63021 non-null  object
 8   Postcode       63021 non-null  int64
 9   Regionname     63021 non-null  object
 10  Propertycount  63021 non-null  int64
 11  Distance       63021 non-null  float64
 12  CouncilArea    63021 non-null  object
dtypes: float64(2), int64(3), object(8)
memory usage: 6.3+ MB
```

[83]:
```
#to get the statistical columns of the data
data.describe()
```

[83]:

	Rooms	Price	Postcode	Propertycount	Distance
count	63021.000000	4.843200e+04	63021.000000	63021.000000	63021.000000
mean	3.110614	9.978980e+05	3125.674727	7617.791895	12.684930
std	0.957556	5.935050e+05	125.628659	4424.477446	7.592042
min	1.000000	8.500000e+04	3000.000000	39.000000	0.000000
25%	3.000000	6.200000e+05	3056.000000	4380.000000	7.000000
50%	3.000000	8.300000e+05	3107.000000	6795.000000	11.400000
75%	4.000000	1.220000e+06	3163.000000	10412.000000	16.700000
max	31.000000	1.120000e+07	3980.000000	21650.000000	64.100000

[84]:
```
#to get the shape of the data
data.shape
```

[84]: (63021, 13)

[85]:
```
#to get the columns in the dataset
data.columns
```

[85]:
```
Index(['Suburb', 'Address', 'Rooms', 'Type', 'Price', 'Method', 'SellerG',
       'Date', 'Postcode', 'Regionname', 'Propertycount', 'Distance',
```

```
       'CouncilArea'],
      dtype='object')
```

[86]:

```
#to get the null values
data.isnull()
```

[86]:

```
       Suburb  Address  Rooms   Type  Price  Method  SellerG   Date  Postcode  \
0       False    False  False  False  False   False    False  False     False
1       False    False  False  False  False   False    False  False     False
2       False    False  False  False  False   False    False  False     False
3       False    False  False  False  False   False    False  False     False
4       False    False  False  False  False   False    False  False     False
…         …      …    …      …      …      …       …      …       …
63016   False    False  False  False  False   False    False  False     False
63017   False    False  False  False  False   False    False  False     False
63018   False    False  False  False  False   False    False  False     False
63019   False    False  False  False   True   False    False  False     False
63020   False    False  False  False   True   False    False  False     False

       Regionname  Propertycount  Distance  CouncilArea
0           False          False     False        False
1           False          False     False        False
2           False          False     False        False
3           False          False     False        False
4           False          False     False        False
…             …              …       …          …
63016       False          False     False        False
63017       False          False     False        False
63018       False          False     False        False
63019       False          False     False        False
63020       False          False     False        False

[63021 rows x 13 columns]
```

[87]:

```
#to get the count of the null values in each column
data.isnull().sum()
```

[87]:

```
Suburb               0
Address              0
Rooms                0
Type                 0
Price            14589
Method               0
SellerG              0
Date                 0
Postcode             0
Regionname           0
```

```
Propertycount        0
Distance             0
CouncilArea          0
dtype: int64
```

[32]:
```
#after getting the count we get the 14589 null values in the price column
#Since Price is a numerical column, we'll use a strategy to fill null values
 ↪logically. Options:

#1. Mean Imputation: Fill null values with mean Price value.
#2. Median Imputation: Fill null values with median Price value.
#3. Regression Imputation: Predict missing Price values using regression model.

#mean_price = data['Price'].mean()
#data['Price'].fillna(mean_price, inplace=True)

#For Price column, Median Imputation might suit better because:
#1. Prices can have outliers (extremely high/low values).
#2. Median represents a more typical middle value.

median_price = data['Price'].median()
data['Price'].fillna(median_price, inplace=True)

#to unshown the warnings we use this code
import warnings
warnings.filterwarnings('ignore')
```

[30]:
```
data.isnull().sum()
```

[30]:
```
Suburb           0
Address          0
Rooms            0
Type             0
Price            0
Method           0
SellerG          0
Date             0
Postcode         0
Regionname       0
Propertycount    0
Distance         0
CouncilArea      0
dtype: int64
```

[88]:
```
scaler = StandardScaler()
data['Price'] = scaler.fit_transform(data[['Price']]).flatten()
```

[89]:
```
#Now we are free with null values, so we can now use this data to train
#splitting the data into feauters(x) and target
X = data[['Rooms', 'Distance', 'Postcode']]
y = data['Price']

#Target Variable (y):
#Price: This is what we want to predict using Linear Regression. House price␣
↪depends on various factors, so we'll model these factors to forecast prices.
#Feature Variables (X):
#Rooms: Number of rooms in a house. More rooms typically increase price.
#Distance: Distance from city center. Closer proximity often raises property␣
↪value.
#Postcode: Area code representing location. Different postcodes have varying␣
↪price trends.
#We choose these features because they:
#1. Directly influence house prices
#2. Are numerical or categorical data types suitable for Linear Regression
#3. Are available in our dataset
#By selecting these columns, we create a relationship between:
#Independent variables (Rooms, Distance, Postcode) -> Features (X)
#Dependent variable (Price) -> Target (y)
```

[93]:
```
data['Price'].fillna(data['Price'].mean(), inplace=True)
```

[94]:
```
y.head()
```

[94]:
```
0    0.829154
1    0.374225
2    0.711209
3    0.871277
4   -0.552483
Name: Price, dtype: float64
```

[167]:
```
#Split data into training and testing sets
#we can change the test_size and random_state based on the result we can choose␣
↪the best one
#test_szie = 0.2, random_state = 42
#X_train, X_test, y_train, y_test = train_test_split(X, y, test_size= 0.2,␣
↪random_state=42)
#test_size = 0.2, random_state = 72
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size= 0.2,␣
↪random_state=72)
#test_size = 0.3, random_state = 60
#X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
↪random_state=60)
```

[168]:
```
#Training Linear Regression model
model = LinearRegression()
model.fit(X_train, y_train)
```

[168]: LinearRegression()

[169]:
```
#Making Predections on test data
y_pred = model.predict(X_test)
print("Predicted House Prices:", y_pred)
```

```
Predicted House Prices: [ 0.55758047  0.40863947 -0.44233004 …  0.52362203
0.15057692
 -0.24003049]
```

[170]:
```
# Calculate metrics
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = (mean_squared_error(y_test, y_pred)) ** 0.5
r2 = r2_score(y_test, y_pred)
#Coefficients
coefficients = model.coef_
intercept = model.intercept_

# Print results
print("Mean Absolute Error (MAE):", mae)
print("Mean Squared Error (MSE):", mse)
print("Root Mean Squared Error (RMSE):", rmse)
print("R-squared (R2):", r2)
print("Coefficients:", coefficients)
print("Intercept:", intercept)
```

```
Mean Absolute Error (MAE): 0.5072731929655269
Mean Squared Error (MSE): 0.5909231323176708
Root Mean Squared Error (RMSE): 0.7687152478763972
R-squared (R2): 0.26216527313646076
Coefficients: [ 0.41393791 -0.04949711  0.0012052 ]
Intercept: -4.429320633764733
```

[152]:
```
#FOR TEST_SCORE = 0.2, RANDOM_STATE = 42
0.257 * 100
```

[152]: 25.7

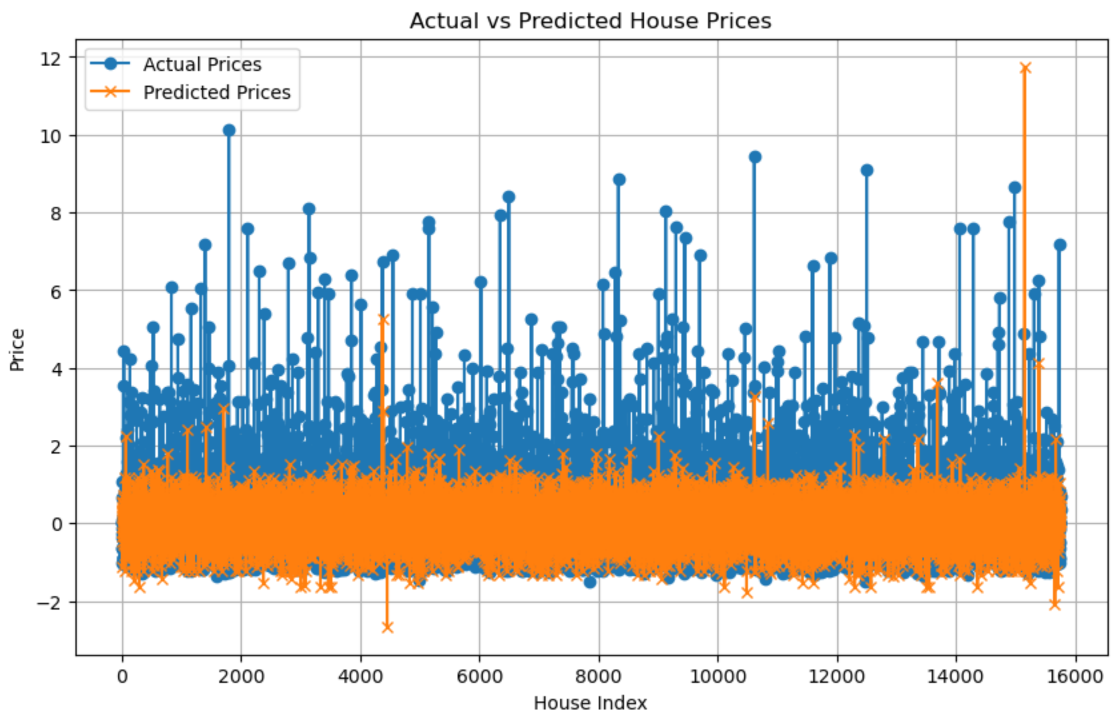
[151]:
```
#FOR TEST_SCORE = 0.3, RANDOM_STATE = 60
0.254 * 100
```

[151]: 25.4

[150]:
```
#FOR TEST_SCORE = 0.2, RANDOM_STATE = 72
#we are using this , because this has the r2 score high which when we calculate␣
↪the r2 with 100,
# then we get the accuracy
#this one has the highest accuracy among the others
0.262 * 100
```

[150]: 26.200000000000003

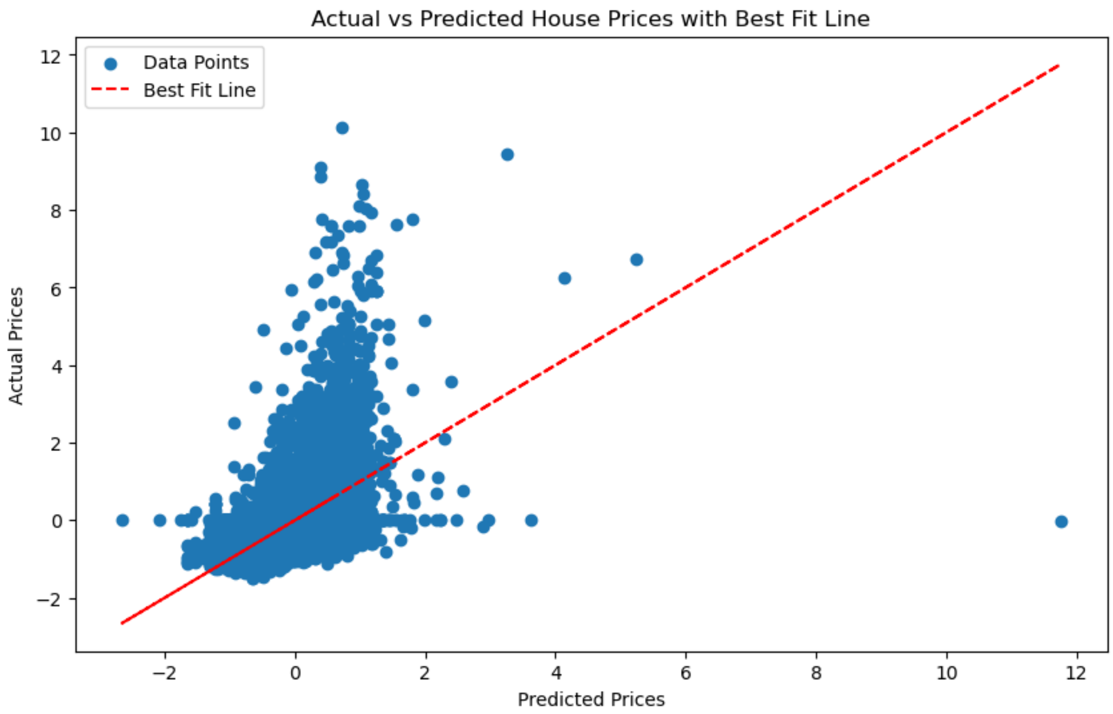
[163]:
```
plt.figure(figsize=(10,6))
plt.plot(range(len(y_test)), y_test, label='Actual Prices', marker='o')
plt.plot(range(len(y_pred)), y_pred, label='Predicted Prices', marker='x')
plt.xlabel('House Index')
plt.ylabel('Price')
plt.title('Actual vs Predicted House Prices')
plt.legend()
plt.grid(True)
plt.show()
```



[ ]:
```
#This is the graphical representation of how the predected values are at and␣
↪actual values are at.
```

[166]:
```
plt.figure(figsize=(10,6))
plt.scatter(y_pred, y_test, label='Data Points')
```

```python
plt.xlabel('Predicted Prices')
plt.ylabel('Actual Prices')
plt.title('Actual vs Predicted House Prices with Best Fit Line')
plt.plot(y_pred, y_pred, "r--", label='Best Fit Line')
plt.legend()
plt.show()
```



[ ]:

```python

```