Real Estate Price Prediction in the UAE Using Machine Learning and Web-Scraped Data.

M00977896 Nishita Chaudhary

Importing Libraries for Sales dataset

In [108…

```
#For data manipulate
import pandas as pd
import numpy as np
#For visualization
import seaborn as sns
import matplotlib.pyplot as plt

#lable encoding
from sklearn.preprocessing import LabelEncoder, MinMaxScaler

# For Model building
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import Lasso
import xgboost as xgb
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor
from tensorflow import keras
from tensorflow.keras import layers
# Evalution Metrics
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.model_selection import GridSearchCV
```

In [109…

```
# Loading the dataset and using pandas for Dataframe
df_s= pd.read_csv('sale.csv')
df_s.head()
```

```
<ipython-input-109-04506be8f20f>:2: DtypeWarning: Columns (2,5,16,24) have mixed types. Specify dtype option on import or set low_memory=False.
  df_s= pd.read_csv('sale.csv')
```

Out[109…

	Unnamed: 0	title	URL	latitude	longitude	URL.1	Locality	Region	Country	price
0	0.0	INVESTOR`S DEAL |CORNER UNIT | DOWN TOWN VIEW.	https://www.bayut.com/property/details-9237326...	25.179186	55.301452	https://images.bayut.com/thumbnails/731154289-...	Sobha Hartland	Dubai	UAE	1,480,000
1	1.0	STUDIO /LOCATION FOR MODERN LIVING /LUXURY THO...	https://www.bayut.com/property/details-9345716...	24.840326	55.136496	https://images.bayut.com/thumbnails/731152910-...	Dubai South	Dubai	UAE	624,000
2	2.0	Furnished | Low Floor | Renovated | Fountain View	https://www.bayut.com/property/details-6904176...	25.197159	55.274491	https://images.bayut.com/thumbnails/728769791-...	Downtown Dubai	Dubai	UAE	3,300,000
3	3.0	Prime Location / Well Establish Community / Af...	https://www.bayut.com/property/details-9345002...	24.986096	55.390471	https://images.bayut.com/thumbnails/731141372-...	DAMAC Hills 2 (Akoya by DAMAC)	Dubai	UAE	1,870,000
4	4.0	Charming View |Unfurnished Modern 1BR | High F...	https://www.bayut.com/property/details-9059540...	24.491796	54.395088	https://images.bayut.com/thumbnails/707102154-...	Al Reem Island	Abu Dhabi	UAE	900,000

5 rows × 25 columns

In [110…

```
#Check the column & rows
df_s.shape
```

Out[110… (89917, 25)

Data Cleaning

Data cleaning involves 1. Checking for duplicates, 2. Checking Unwanted rows, 3. Checking the column is in write format 4. Checking the datatypes 5. Checking for Null Values

In [111…

```
#Checking the rows in this data set there are 89917 records initially.
len(df_s)
```

Out[111… 89917

Checking Duplicates

In [112…
```
#Chaecking duplicates 36595 coloumns were removed as they consist of duplicates
#Removing Duplicates
df_s = df_s.drop_duplicates()
df_s.duplicated().sum()
```

Out[112… 0

In [113…
```
#Again Checking the length after removing duplicates is 53322
len(df_s)
```

Out[113… 53322

Manipulating sales columns

In [114…
```
#Converting the object datatype of price to numeric, removing "," from the price and filling the nan values to 0.
df_s['price'] = pd.to_numeric(df_s['price'].str.replace(',', ''), errors='coerce')
df_s['price'] = df_s['price'].fillna(df_s['price'].mean())
```

In [115…
```
#Replace with 0 as it have only studio room can be seen in name column
# Extract the number of beds or 'Studio'
df_s['beds'] = df_s['beds'].str.extract(r'(\d+|Studio)', expand=False)
df_s['beds'] = df_s['beds'].replace('Studio', 0) # Replace 'Studio' with 0
df_s['beds'] = pd.to_numeric(df_s['beds'], errors='coerce')
df_s['baths'] = df_s['baths'].str.extract(r'(\d+)', expand=False)
```

In [116…
```
#Remove "sqrt" suffix data from area column

#Converting the ares to numeric.
def extract_last_number(text):
    numbers = [num.replace(',', '') for num in text.split() if num.replace(',', '').isdigit()]
    return numbers[-1] if numbers else None
df_s['area'] = df_s['area'].astype(str)

df_s['area'] = df_s['area'].apply(extract_last_number)

df_s['area'] = pd.to_numeric(df_s['area'], errors='coerce')

print(df_s[['area']])
```

```
        area
0      775.0
1      358.0
2      826.0
3     2352.0
4      904.0
...      ...
89903  470.0
89905  745.0
89910  563.0
89911  317.0
89914  686.0

[53322 rows x 1 columns]
```

Checking Null values and Handling Null values

In [117…

```python
#Checking null values greater than 80%
null_percentage = df_s.isnull().mean() * 100  # Calculate percentage of null values for each column

#filtering more 80% null values
columns_with_high_nulls = null_percentage[null_percentage > 80].index
print(f"Columns with more than 80% null values:\n{null_percentage[null_percentage > 80]}")
```

```
Columns with more than 80% null values:
year_of_completion     88.738232
Country.1             100.000000
dtype: float64
```

In [118…

```python
#Initially There were 22 columns by scrapping so dropping columns which are more than 80% null values and which are inconsist for analysis.
unwanted_columns = ['URL', 'URL.1', 'Country.1', 'other_details','Unnamed: 0', 'Reference', 'year_of_completion', 'description']
df_s = df_s.drop(columns=unwanted_columns)
```

In [119…

```python
#Checking null values
df_s.isnull().sum()
```

Out[119…

```
title                    0
latitude                 0
longitude                0
Locality                 0
Region                   0
Country                  0
price                    0
type                    23
address                 16
beds                  1096
baths                 1095
completion_status       23
furnishing            9624
post_date               23
area                  2406
agency_name             31
purpose                  0
dtype: int64
```

In [120…

```python
df_s['type'].fillna(df_s['type'].mode()[0], inplace=True)
df_s['beds'].fillna(df_s['beds'].mode()[0], inplace=True)
df_s['baths'].fillna(df_s['baths'].mode()[0], inplace=True)
df_s['furnishing'].fillna(df_s['furnishing'].mode()[0], inplace=True)
df_s['completion_status'].fillna(df_s['completion_status'].mode()[0], inplace=True)
df_s['agency_name'].fillna(df_s['agency_name'].mode()[0], inplace=True)
df_s['address'].fillna(df_s['address'].mode()[0], inplace=True)
df_s['post_date'].fillna(method='ffill', inplace=True)
df_s['area'].fillna(df_s['area'].mean(), inplace=True)
```

```
<ipython-input-120-5959eb69f78f>:8: FutureWarning: Series.fillna with 'method' is deprecated and will raise in a future version. Use obj.ffill() or obj.bfill() i
nstead.
  df_s['post_date'].fillna(method='ffill', inplace=True)
```

In [121…

```python
#Checking the null values again
df_s.isnull().sum()
```

Out[121…

```
title                0
latitude             0
longitude            0
Locality             0
Region               0
Country              0
price                0
type                 0
address              0
beds                 0
baths                0
completion_status    0
furnishing           0
post_date            0
area                 0
agency_name          0
purpose              0
dtype: int64
```

Checking data types

In [122…

```python
#Check the datatypes
df_s.dtypes
```

Out[122…

```
title                 object
latitude             float64
longitude            float64
Locality              object
Region                object
Country               object
price                float64
type                  object
address               object
beds                 float64
baths                 object
completion_status     object
furnishing            object
post_date             object
area                 float64
agency_name           object
purpose               object
dtype: object
```

Converting the datatype

In [123…

```python
#As from the above cell postdate is converted to datetime and address converted to string.
# Converting the datatypes using mean median mode
from datetime import date
df_s['address'] = df_s['address'].astype(str)
df_s['post_date'] = pd.to_datetime(df_s['post_date'])
df_s['beds'] = df_s['beds'].astype(int)
df_s['baths'] = df_s['baths'].astype(int)
df_s.dtypes
```

Out[123…

```
title                        object
latitude                    float64
longitude                   float64
Locality                     object
Region                       object
Country                      object
price                       float64
type                         object
address                      object
beds                          int32
baths                         int32
completion_status            object
furnishing                   object
post_date            datetime64[ns]
area                        float64
agency_name                  object
purpose                      object
dtype: object
```

In [124…

```python
columns_to_analyze = ['price', 'area', 'beds', 'baths']

# Statistical analysis for sales columns
statistical_summary = df_s[columns_to_analyze].agg(['count', 'mean', 'median', 'std', 'min', 'max']).T
```

```
statistical_summary.columns = ['number', 'mean', 'median', 'sd', 'min', 'max']

# Display the statistical summary
print("Statistical Summary:")
print(statistical_summary)
```

```
Statistical Summary:
        number          mean     median            sd    min          max
price  53322.0  3.627715e+06  1876000.0  7.269015e+06  210.0  383250000.0
area   53322.0  2.102827e+03     1302.0  3.664222e+03   83.0     385000.0
beds   53322.0  2.147875e+00        2.0  1.591168e+00    0.0         11.0
baths  53322.0  2.993399e+00        2.0  1.826774e+00    1.0         11.0
```

Feature Extraction

It improves the efficiency and accuracy of machine learning models through extracting the necessary information from a data set.

In [125…

```
# Extracting year, month, and day
df_s['year'] = df_s['post_date'].dt.year
df_s['month'] = df_s['post_date'].dt.month
df_s['day'] = df_s['post_date'].dt.day
df_s['quarter'] = df_s['post_date'].dt.quarter
```

In [126…

```
#Extracting building name from address
df_s.loc[:, 'building_name'] = df_s['address'].str.split(',', expand=True)[0]
```

In [127…

```
# Extracting price_per_sq_unit using price and area
df_s['price_per_sq_unit'] = (df_s['price'] / df_s['area']).round(2)
df_s['price_per_sq_unit']
```

Out[127…

```
0        1909.68
1        1743.02
2        3995.16
3         795.07
4         995.58
          ...
89903    2234.04
89905    2147.65
89910    2060.39
89911    3470.03
89914    3644.31
Name: price_per_sq_unit, Length: 53322, dtype: float64
```

In [128…

```
df_s.dtypes
```

Out[128...
```
title                         object
latitude                     float64
longitude                    float64
Locality                      object
Region                        object
Country                       object
price                        float64
type                          object
address                       object
beds                           int32
baths                          int32
completion_status             object
furnishing                    object
post_date             datetime64[ns]
area                         float64
agency_name                   object
purpose                       object
year                           int32
month                          int32
day                            int32
quarter                        int32
building_name                 object
price_per_sq_unit            float64
dtype: object
```

In [129...
```python
# Categorize the Properties into 4 sections affordable, midrange, premium and Luxury
def cat_property(price):
    if price < 2000000:
        return 'Affordable'
    elif 2000000 <= price < 4000000:
        return 'Mid-range'
    elif 4000000 <= price < 6000000:
        return 'Premium'
    else:
        return 'Luxury'
df_s.loc[:, 'price_category'] = df_s['price'].apply(cat_property)
```

Storing the dataset for data visualization clean_df_s = df_s.copy() clean_df_s.to_csv('sale_eda.csv', index=False) #clean_df_s.to_excel('sale_data.xlsx', index=False)

In [130...
```python
#From data visualization this anomalywas detected so, Replace 'Al Napooca' with 'Ajman' in the 'Region' column
df_s['Region'] = df_s['Region'].replace('Al Napoca', 'Ajman')
df_s['Region'] = df_s['Region'].replace('Al Ain', 'Abu Dhabi')
```

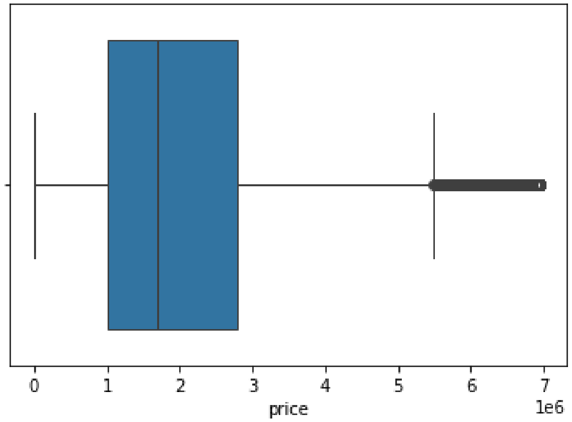
Removing Outliers

In [131...
```python
#Checking the outliers for price coloumn using iqr method
Q1 = df_s['price'].quantile(0.25)
Q3 = df_s['price'].quantile(0.75)
IQR = Q3 - Q1
df_s = df_s[(df_s['price'] >= (Q1 - 1.5 * IQR)) & (df_s['price'] <= (Q3 + 1.5 * IQR))]
len(df_s)
```

Out[131...
```
48176
```

In [132…

```
#using box plott to visualize the price outlier.
import seaborn as sns
import matplotlib.pyplot as plt
sns.boxplot(x=df_s['price'])
plt.show()
```



In [133…

```
# dropping other columns
dcolumn = [ 'title','address', 'purpose', 'Country', 'post_date','agency_name', 'building_name'] #
df_s= df_s.drop(columns=dcolumn)
```

In [134…

```
# From the below analysis year month is the import coloumns and cannot be doropped
#This encoding helps improve model performance by accurately reflecting the relationships between months and quarters, allowing for
#better predictions analysis.
df_s['month'] = np.sin(2 * np.pi * df_s['month'] / 12)
df_s['month'] = np.cos(2 * np.pi * df_s['month'] / 12)

# Cyclic encoding for quarter
df_s['quarter'] = np.sin(2 * np.pi * df_s['quarter'] / 4)
df_s['quarter'] = np.cos(2 * np.pi * df_s['quarter'] / 4)
```

In [135…

```
# As the price value are in millions so taking the log of price.
df_s['price'] = np.log1p(df_s['price'])
```

Normalization of price data by np.log1p(df_s['price']) enhances the performance of regression models. Logarithmic transformation reduces variance and stabilizes the distribution; therefore, it makes the model more robust to outliers. Besides, due to the fact that in a linear regression the coefficients are percentage changes, not an absolute one, it improves interpretability.

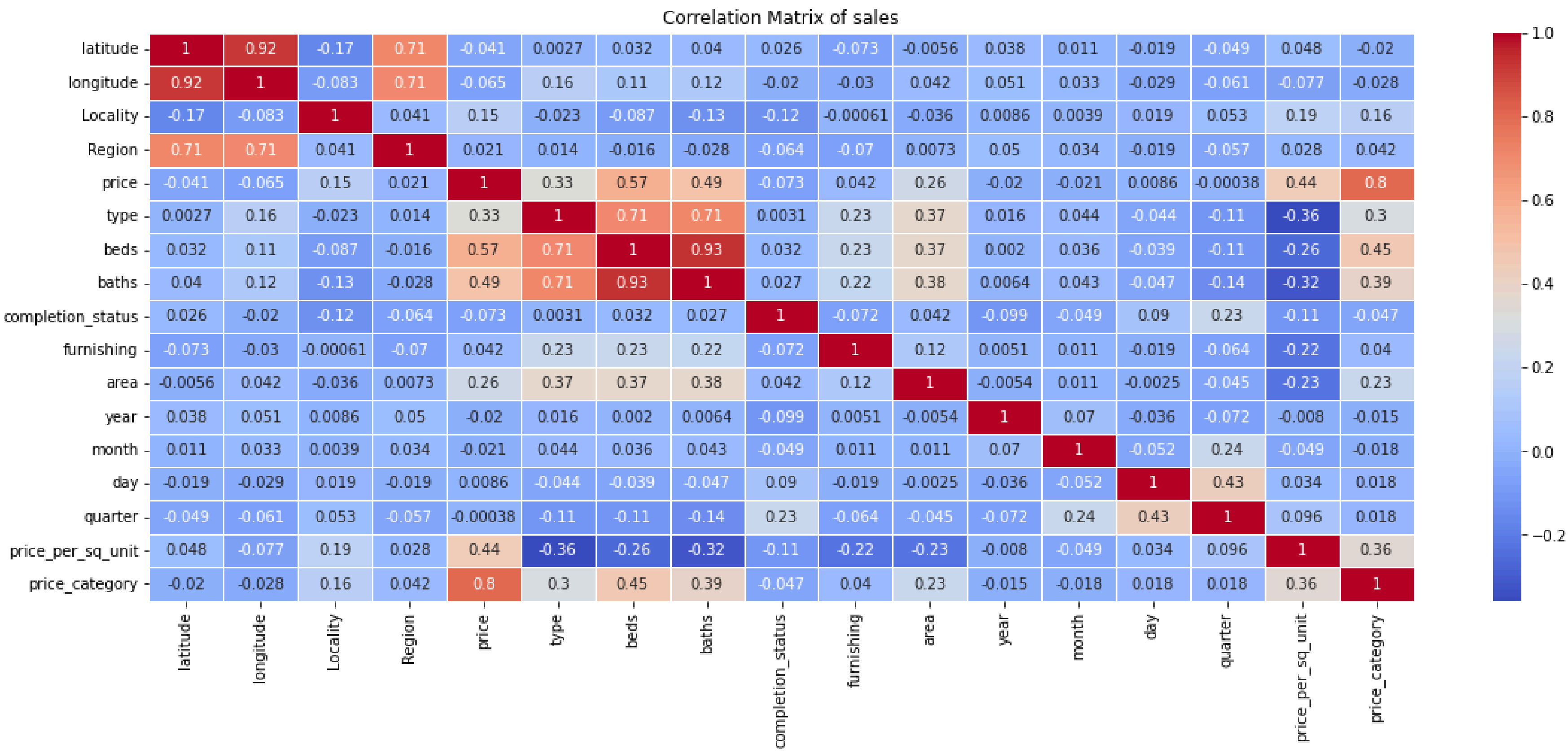
In [136…

```
#Initialize LabelEncoder
le = LabelEncoder()
scaler = MinMaxScaler()

#Apply LabelEncoder for categorical columns
categorical_columns = [ 'furnishing',  'completion_status', 'Region', 'type', 'price_category', 'Locality']
for column in categorical_columns:
    df_s[column] = le.fit_transform(df_s[column].astype(str))
```

```
#Apply StandardScaler for numerical columns
numerical_columns = [  'area', 'beds','baths', 'latitude', 'year', 'month', 'day', 'price_per_sq_unit', 'longitude' ]
df_s[numerical_columns] = scaler.fit_transform(df_s[numerical_columns])
```

In [137…

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
# using corr() function to check correlations
corr_matrix = df_s.corr()

# Optionally, you can visualize it using a heatmap
plt.figure(figsize=(19, 7))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', linewidths=0.55)
plt.title('Correlation Matrix of sales')
plt.show()
```



As beds and bath are important feature for analysis so cannont be removed, also longitute and latitude are important features and cannot be removed.

In [138…

```
# Defining X and y for prediction
#Price is my target variable
X = df_s.drop('price', axis=1)
y = df_s['price']
```

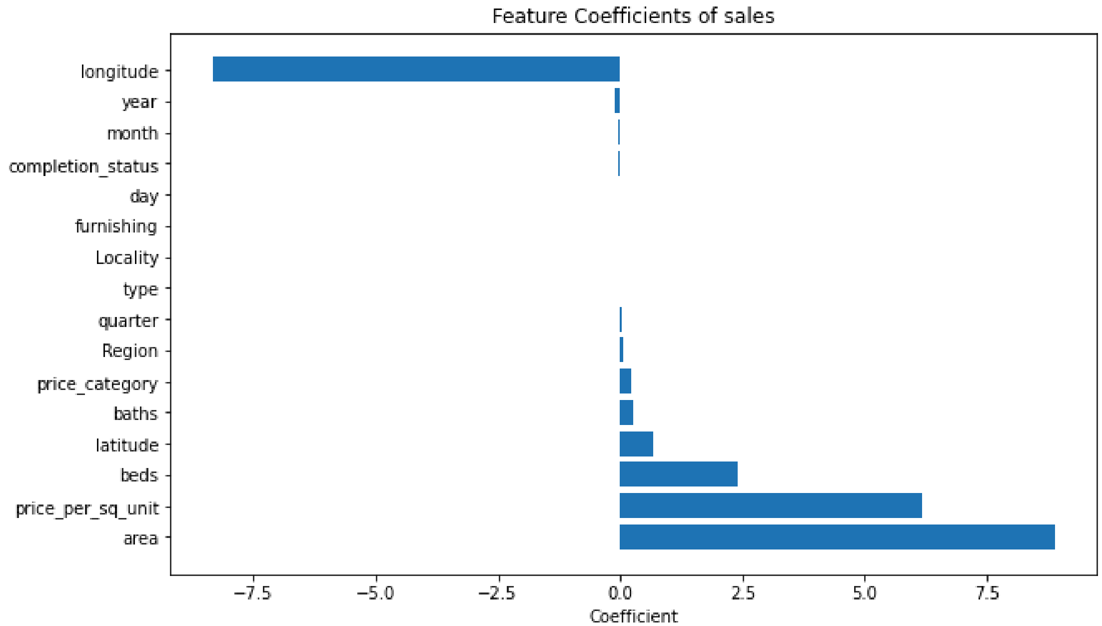
```python
# Splitting the data into xtrain and text and y train & text by splitting 30% for testing and 70% for training
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

Feature Importance

In [139…

```python
#Fiting a linear regression model
lin_model = LinearRegression()
lin_model.fit(X, y)

# Getting all feature coefficients
coef = lin_model.coef_
#For tabluar data
coef_importance = pd.DataFrame({'Feature': X.columns, 'Coefficient': coef})
coef_importance = coef_importance.sort_values(by='Coefficient', ascending=False)

print(coef_importance)

# Plotting the coefficients
plt.figure(figsize=(10, 6))
plt.barh(coef_importance['Feature'], coef_importance['Coefficient'])
plt.xlabel('Coefficient')
plt.title('Feature Coefficients of sales')
plt.show()
```

```
              Feature  Coefficient
9                area     8.891789
14  price_per_sq_unit     6.156396
5                beds     2.410648
0            latitude     0.662654
6               baths     0.260655
15     price_category     0.232776
3              Region     0.075551
13            quarter     0.023956
4                type     0.003899
2            Locality     0.000513
8          furnishing    -0.002802
12                day    -0.004485
7   completion_status    -0.034464
11              month    -0.044869
10               year    -0.092338
1           longitude    -8.329032
```



Area: For every additional unit of area, the price is more by approximately 8.94 units. Price Per Square Unit: For every unit increase in price per square unit, the total price increases by approximately 6.08 units. Beds: For every additional bedroom, it contributes to about 2.39 units in prices. Baths: For every additional bathroom, there is an approximate increase of 0.32 units to the price of a house. Latitude: With every unit increase in latitude, prices increase by approximately 0.72 units. Price Category: Each category contributes around 0.23 units to the price. Region: Region contributes to the pricing by about 0.07 units. Quarter: The quarterly variable adds about 0.02 units to the price. Type: Property type contributes minimally towards pricing at 0.005 units. Furnishing: The level of furnishing contributes about 0.002 units toward pricing. Locality: Locality has a negligible effect on pricing at 0.0004 units. Day: Price decreases a little with an increase in the day of the month by 0.0027 units. Completion Status: Properties that are not completed are valued lower by approximately 0.03 units. Month: Price slightly decreases as the month increases by 0.0439 units. Year: Each additional year roughly contributes to about a decrease in price by about 0.10 units. Longitude: The further east/west a property is, the far lower in price, with a huge negative contribution of -9.34 units.

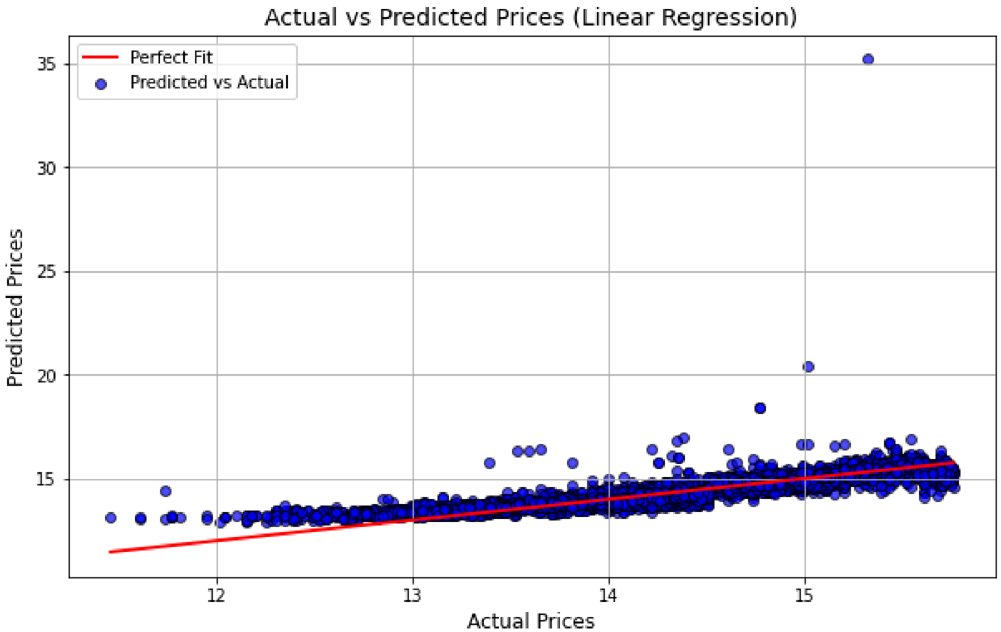
1. LinearRegression

In [140…

```
# Training Linear regression model for sales
model_linear_s = LinearRegression()

model_linear_s.fit(X_train, y_train)

#predicting using the testing data for sales
y_pred = model_linear_s.predict(X_test)

# clculate regression metrics of sales for sales
mae_linear = mean_absolute_error(y_test, y_pred)
mse_linear = mean_squared_error(y_test, y_pred)
rmse_linear = np.sqrt(mse_linear)
R_2_s_linear = r2_score(y_test, y_pred)


y_train_pred = model_linear_s.predict(X_train)

# Calculate regression metrics for training set
r2_train_l_s = r2_score(y_train, y_train_pred)
print(f"Training R2 Score of sale: {r2_train_l_s}")
```

```python
#display the results of sales
print(f"R² Score: {R_2_s_linear}")
print(f"Mean Absolute Error of linear regression  for sales: {mae_linear}")
print(f"Mean Squared Error of linear regression  for sales: {mse_linear}")
print(f"Root Mean Squared Error Linear regression  for sales: {rmse_linear}")
import matplotlib.pyplot as plt

#plot the Actual vs Predicted values of sales
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, color='blue', edgecolors='k', alpha=0.7, label='Predicted vs Actual')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', linewidth=2, label='Perfect Fit')

plt.title('Actual vs Predicted Prices (Linear Regression)', fontsize=14)
plt.xlabel('Actual Prices', fontsize=12)
plt.ylabel('Predicted Prices', fontsize=12)
plt.legend()
plt.grid(True)
plt.show()
```

```
Training R2 Score of sale: 0.8351844383285107
R² Score: 0.7873270337916629
Mean Absolute Error of linear regression  for sales: 0.20315595472671413
Mean Squared Error of linear regression  for sales: 0.1045769145109135
Root Mean Squared Error Linear regression  for sales: 0.32338354087818616
```



2. lasso

In [141…

```python
# Initialize and train the Lasso Regression model and using random alpha for regularization strength
model_lasso = Lasso(alpha= 0.1)
model_lasso.fit(X_train, y_train)
```

```
y_pred = model_lasso.predict(X_test)

#model evaluating of sales
mse_lasso = mean_squared_error(y_test, y_pred)
mae_lasso = mean_absolute_error(y_test, y_pred)
r2_lasso = r2_score(y_test, y_pred)
actual_prices_mean = y_test.mean()

# Calculate regression metrics of sales
mae_l = mean_absolute_error(y_test, y_pred)
mse_l = mean_squared_error(y_test, y_pred)
rmse_l = np.sqrt(mse_lasso)
R_2_s_l = r2_score(y_test, y_pred)


y_train_pred = model_lasso.predict(X_train)

# Calculate regression metrics for training set
r2_train_la_s = r2_score(y_train, y_train_pred)
print(f"Training R2 Score of sale: {r2_train_la_s}")

# Display the results of sales lasso
print(f"R2 Score: {R_2_s_l}")
print(f"Mean Absolute Error of lasso with random alpha  for sales: {mae_l}")
print(f"Mean Squared Error of lasso with random alpha  for sales: {mse_l}")
print(f"Root Mean Squared Error lasso with random alpha  for sales: {rmse_l}")
```

```
Training R2 Score of sale: 0.6336449876326321
R2 Score: 0.6344111627744164
Mean Absolute Error of lasso with random alpha  for sales: 0.3104858042352185
Mean Squared Error of lasso with random alpha  for sales: 0.1797696870378505
Root Mean Squared Error lasso with random alpha  for sales: 0.4239925554038072
```

In [142…

```
# Define the grid of alpha values to search over of sales
param_grid = {'alpha': [0.01,  0.0001, 0.1, 1]}

# Initialize the GridSearchCV object with cross validation = 5
grid_search = GridSearchCV(estimator=Lasso(), param_grid=param_grid, cv=5)

#fit the model to find the best alpha using gridsearch
grid_search.fit(X_train, y_train)

# Get the best model from the grid search of sales
best_lasso_model = grid_search.best_estimator_
best_alpha = grid_search.best_params_['alpha']

#predict on the test set using the best model of sales
y_pred_best = best_lasso_model.predict(X_test)

y_train_pred = best_lasso_model.predict(X_train)

# Calculate regression metrics for training set
r2_train_a_s = r2_score(y_train, y_train_pred)
print(f"Training R2 Score of sale: {r2_train_a_s}")
```

```python
# evalutation the lasso with best alpha
r2_lasso_grid = r2_score(y_test, y_pred_best)
mse_best_lasso_grid = mean_squared_error(y_test, y_pred_best)
mae_lasso_grid = mean_absolute_error(y_test, y_pred_best)

# Print results of sales
print(f"Best alpha value selected lasso for sales: {best_alpha}")
print(f'R2 (Lasso)  for sales: {r2_lasso_grid}')
print(f"Mean Squared Error with best alpha  for sales: {mse_best_lasso_grid}")
print(f"Mean Absolute Error for sales: {mae_lasso_grid}")
```

```
Training R2 Score of sale: 0.8337570023470391
Best alpha value selected lasso for sales: 0.0001
R2 (Lasso)  for sales: 0.8105140629435301
Mean Squared Error with best alpha  for sales: 0.09317523987116877
Mean Absolute Error for sales: 0.20567278139378503
```

XGBOOST

In [143…

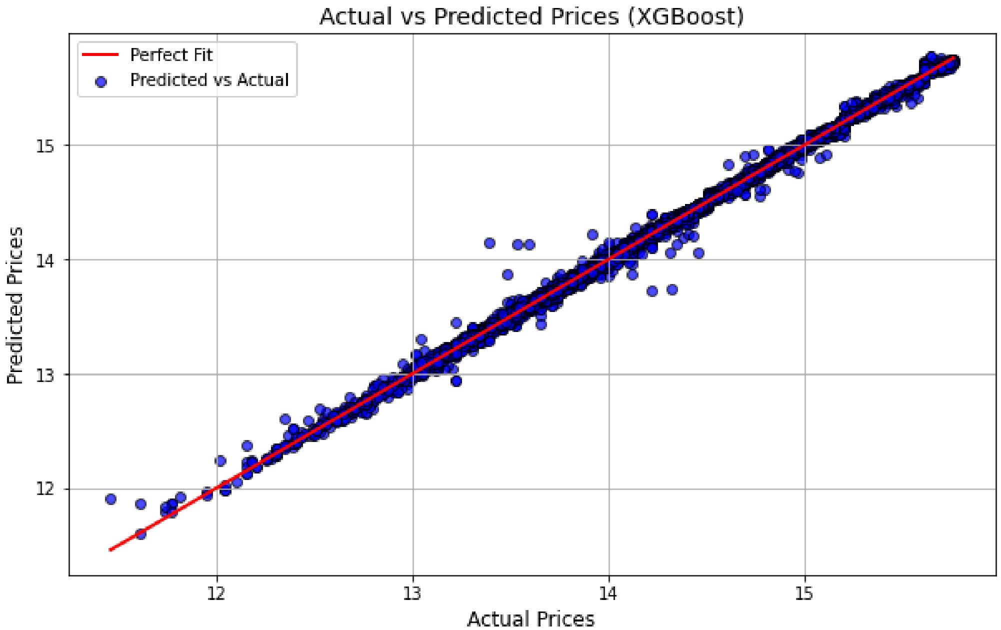
```python
# XGB WITHOUT PARAMETER
```

In [144…

```python
# initailze the xbg model
model_xgb_sale = xgb.XGBRegressor(objective='reg:squarederror')
model_xgb_sale.fit(X_train, y_train)
y_pred = model_xgb_sale.predict(X_test)
#Metrics
mae_x_sale = mean_absolute_error(y_test, y_pred)
mse_x_sale = mean_squared_error(y_test, y_pred)
r2_x_sale = r2_score(y_test, y_pred)
rmse_x_sale = np.sqrt(mse_x_sale)


y_train_pred = model_xgb_sale.predict(X_train)

# Calculate regression metrics for training set
r2_train_xgb_s = r2_score(y_train, y_train_pred)
print(f"Training R2 Score of sale: {r2_train_xgb_s}")


print(f"R2 Score XGBoost for sales: {r2_x_sale}")
print(f"Mean Absolute Error XGBoost for sales: {mae_x_sale}")
print(f"Mean Squared Error XGBoost for sales: {mse_x_sale}")
print(f"Root Mean Squared Error XGBoost for sales: {rmse_x_sale}")

# Plotting the Actual vs Predicted values
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, color='blue', edgecolors='k', alpha=0.7, label='Predicted vs Actual')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', linewidth=2, label='Perfect Fit')

# Adding titles and labels
plt.title('Actual vs Predicted Prices (XGBoost)', fontsize=14)
plt.xlabel('Actual Prices', fontsize=12)
plt.ylabel('Predicted Prices', fontsize=12)
```

```
plt.legend()

# Displaying the plot
plt.grid(True)
plt.show()
```

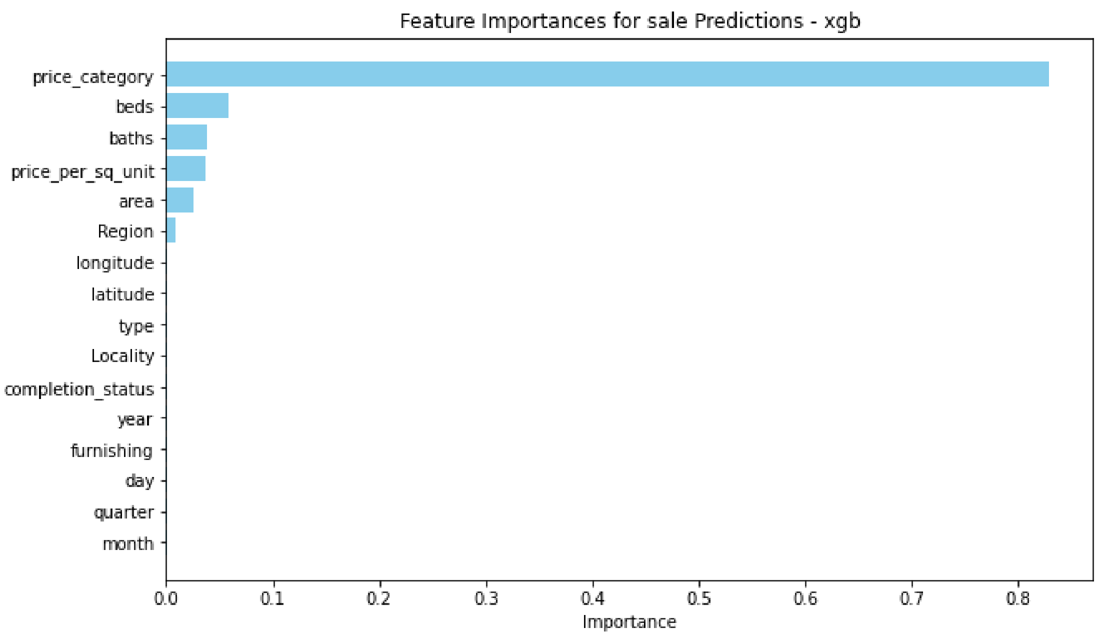
```
Training R2 Score of sale: 0.999236445811853
R2 Score XGBoost for sales: 0.9984267439419624
Mean Absolute Error XGBoost for sales: 0.016967793071900023
Mean Squared Error XGBoost for sales: 0.0007736115558947283
Root Mean Squared Error XGBoost for sales: 0.027813873442847335
```



In [145…

```
feature_importance_rent = model_xgb_sale.feature_importances_

#creating a DataFrame for a tabular display of features and their importance
coef_importance_rent = pd.DataFrame({'Feature': X.columns, 'Importance': feature_importance_rent})
coef_importance_rent = coef_importance_rent.sort_values(by='Importance', ascending=False)

print(coef_importance_rent)

#Plotting the feature importances
plt.figure(figsize=(10, 6))
plt.barh(coef_importance_rent['Feature'], coef_importance_rent['Importance'], color='skyblue')
plt.xlabel('Importance')
plt.title('Feature Importances for sale Predictions - xgb')
plt.gca().invert_yaxis()  # To display the highest importance at the top
plt.show()
```

```
              Feature  Importance
15     price_category    0.828957
5                beds    0.058111
6               baths    0.037992
14  price_per_sq_unit    0.036792
9                area    0.025375
3              Region    0.008083
1           longitude    0.001603
0            latitude    0.001034
4                type    0.000578
2            Locality    0.000501
7   completion_status    0.000244
10               year    0.000184
8          furnishing    0.000145
12                day    0.000144
13            quarter    0.000134
11              month    0.000123
```



In [147…

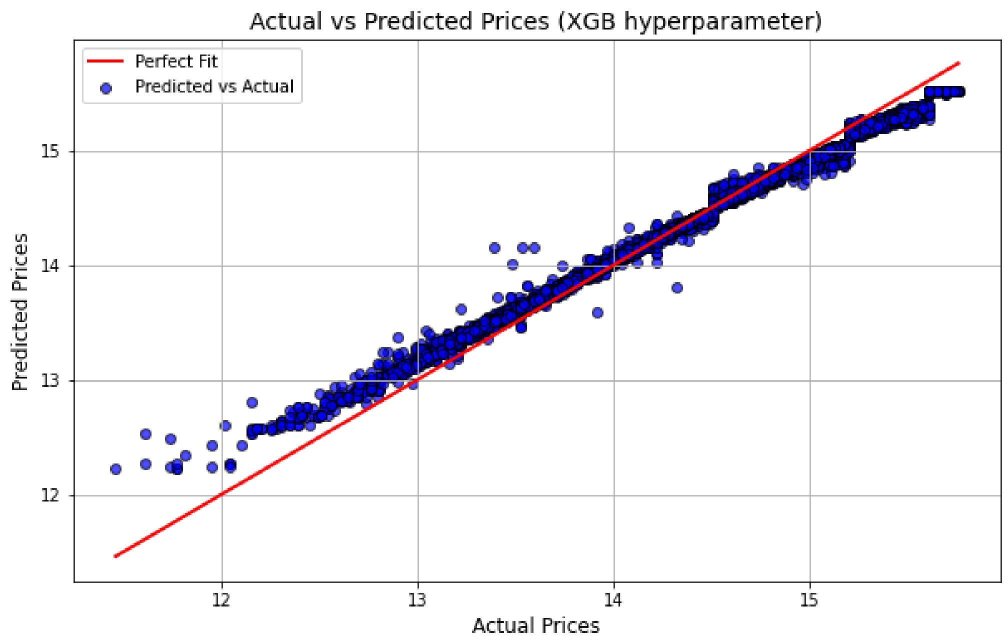
```
# XGB WITH PARAMETERS
```

In [148…

```
# Create and train the XGBoost model with hyperparameter
model_xgb_s = xgb.XGBRegressor(objective='reg:squarederror', n_estimators=20, learning_rate=0.1, max_depth=10)
model_xgb_s.fit(X_train, y_train)

#predict xgb
y_pred = model_xgb_s.predict(X_test)

#evaluatee xbg
mae_x_s = mean_absolute_error(y_test, y_pred)
mse_x_s = mean_squared_error(y_test, y_pred)
r2_x_s = r2_score(y_test, y_pred)
```

```python
rmse_x_s = np.sqrt(mse_x_s)

y_train_pred = model_xgb_s.predict(X_train)

# Calculate regression metrics for training set
r2_train_xgbh_s = r2_score(y_train, y_train_pred)
print(f"Training R2 Score of sale: {r2_train_xgbh_s}")


print(f"R2 Score xgb  for sales: {r2_x_s}")
print(f"Mean Absolute Error xbg hyper parameter for sales: {mae_x_s}")
print(f"Mean Squared Error xgb hyperparamter for sales: {mse_x_s}")
print(f"Root Mean Squared Error xgb hyper paramet for sales: {rmse_x_s}")
# Plotting the Actual vs Predicted values
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, color='blue', edgecolors='k', alpha=0.7, label='Predicted vs Actual')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', linewidth=2, label='Perfect Fit')

# Adding titles and labels
plt.title('Actual vs Predicted Prices (XGB hyperparameter)', fontsize=14)
plt.xlabel('Actual Prices', fontsize=12)
plt.ylabel('Predicted Prices', fontsize=12)
plt.legend()

# Displaying the plot
plt.grid(True)
plt.show()
```

```
Training R2 Score of sale: 0.981437524937552
R2 Score xgb  for sales: 0.9811286402749706
Mean Absolute Error xbg hyper parameter for sales: 0.07523551603197816
Mean Squared Error xgb hyperparamter for sales: 0.00927954599897674
Root Mean Squared Error xgb hyper paramet for sales: 0.09633040018071523
```



Random forest

In [149…

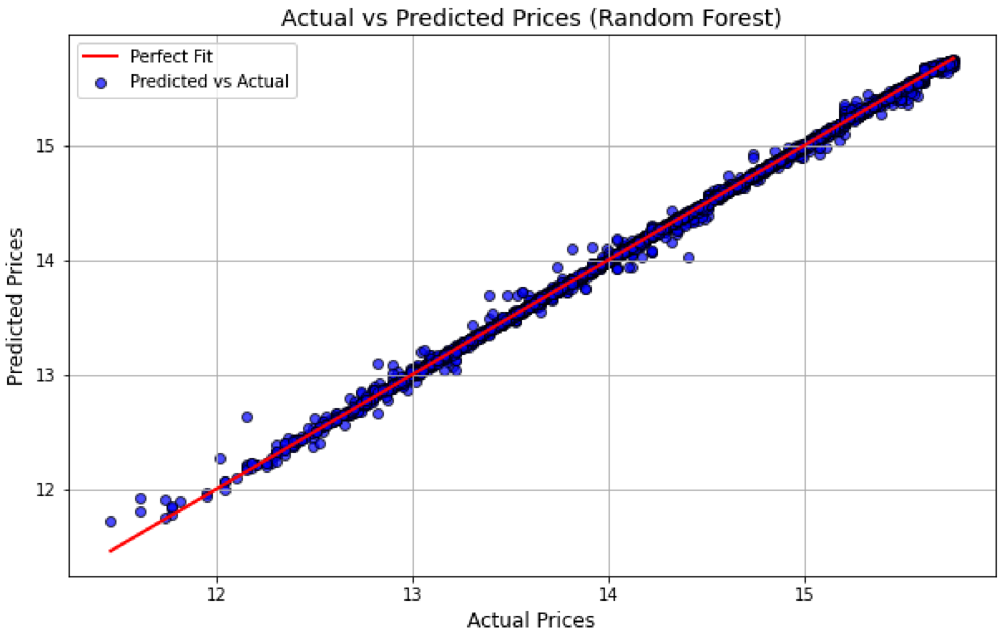
```
# Without hyperparameter
```

In [150…

```
#intialize the random forest model to sales data
model_random_sale = RandomForestRegressor()
model_random_sale.fit(X_train, y_train)

y_pred = model_random_sale.predict(X_test)

#evaluate the model for sales data
mse_random_sale = mean_squared_error(y_test, y_pred)
mae_random_sale = mean_absolute_error(y_test, y_pred)
r2_sale = r2_score(y_test, y_pred)


y_train_pred = model_random_sale.predict(X_train)

# Calculate regression metrics for training set
r2_train_rm_s = r2_score(y_train, y_train_pred)
print(f"Training R2 Score of sale: {r2_train_rm_s}")


#calculate the mean of actual prices for sales data
actual_prices_mean_random_sale = y_test.mean()

#calculate Root Mean Squared Error
rmse_random_sale = np.sqrt(mse_random_sale)
```

```
#display the results for random forest for sales data
print(f"R2 Score random forest: {r2_sale}")
print(f"Mean Absolute Error random forest: {mae_random_sale}")
print(f"Mean Squared Error random forest: {mse_random_sale}")
print(f"Root Mean Squared Error random forest: {rmse_random_sale}")

#plotting the Actual vs Predicted values of sales
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, color='blue', edgecolors='k', alpha=0.7, label='Predicted vs Actual')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', linewidth=2, label='Perfect Fit')
plt.title('Actual vs Predicted Prices (Random Forest)', fontsize=14)
plt.xlabel('Actual Prices', fontsize=12)
plt.ylabel('Predicted Prices', fontsize=12)
plt.legend()
plt.grid(True)
plt.show()
```

```
Training R2 Score of sale: 0.9996122579716925
R2 Score random forest: 0.9994640864823852
Mean Absolute Error random forest: 0.00588338922363119
Mean Squared Error random forest: 0.0002635228309269395
Root Mean Squared Error random forest: 0.016233386304987
```

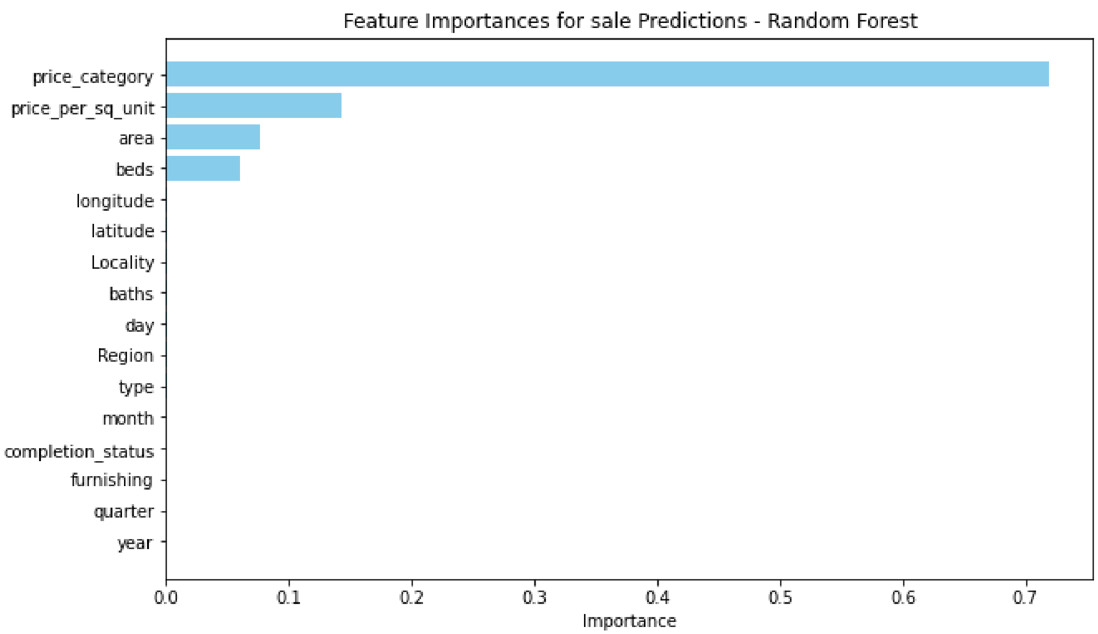


In [151…

```
feature_importance_rent = model_random_sale.feature_importances_

#creating a DataFrame for a tabular display of features and their importance
coef_importance_rent = pd.DataFrame({'Feature': X.columns, 'Importance': feature_importance_rent})
coef_importance_rent = coef_importance_rent.sort_values(by='Importance', ascending=False)

print(coef_importance_rent)
```

```python
#Plotting the feature importances
plt.figure(figsize=(10, 6))
plt.barh(coef_importance_rent['Feature'], coef_importance_rent['Importance'], color='skyblue')
plt.xlabel('Importance')
plt.title('Feature Importances for sale Predictions - Random Forest')
plt.gca().invert_yaxis()  # To display the highest importance at the top
plt.show()
```

	Feature	Importance
15	price_category	0.719094
14	price_per_sq_unit	0.142393
9	area	0.076852
5	beds	0.059902
1	longitude	0.000524
0	latitude	0.000404
2	Locality	0.000210
6	baths	0.000156
12	day	0.000129
3	Region	0.000128
4	type	0.000076
11	month	0.000049
7	completion_status	0.000030
8	furnishing	0.000023
13	quarter	0.000020
10	year	0.000010

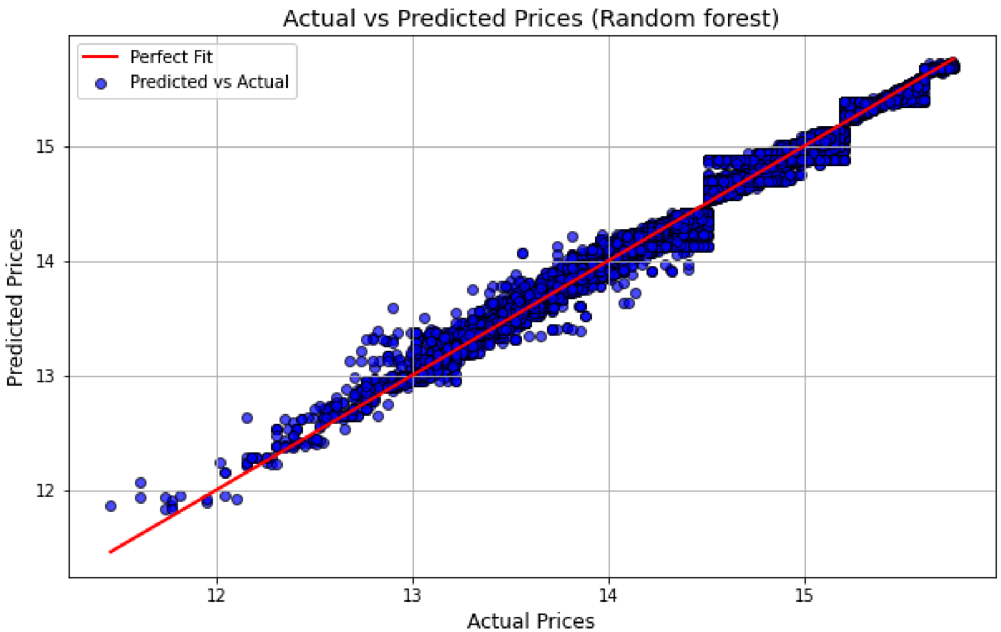


In [152…
```python
# With parameters
```

In [153…
```python
# Initialize and train the model with limited trees to remove overfitting
model_random_s = RandomForestRegressor(n_estimators=20, max_depth=8, random_state=42)
model_random_s.fit(X_train, y_train)
```

```python
# Predict on the test set random forest
y_pred = model_random_s.predict(X_test)

# Evaluate the random forest for sales data
mse_random_s = mean_squared_error(y_test, y_pred)
mae_random_s = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

y_train_pred = model_random_s.predict(X_train)

# Calculate regression metrics for training set
r2_train_rmh_s = r2_score(y_train, y_train_pred)
print(f"Training R2 Score of sale: {r2_train_rmh_s}")
# Calculate the mean of actual prices for sales data
actual_prices_mean_random_s = y_test.mean()

rmse_r_s = np.sqrt(mse_random_s)

# Display the results for random forest for sales data
print(f"R2 Score random forest: {r2}")
print(f"Mean Absolute Error random forest: {mae_random_s}")
print(f"Mean Squared Error random forest: {mse_random_s}")
print(f"Root Mean Squared Error Randon forest: {rmse_r_s}")

# Plotting the Actual vs Predicted values
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, color='blue', edgecolors='k', alpha=0.7, label='Predicted vs Actual')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', linewidth=2, label='Perfect Fit')

# Adding titles and labels
plt.title('Actual vs Predicted Prices (Random forest)', fontsize=14)
plt.xlabel('Actual Prices', fontsize=12)
plt.ylabel('Predicted Prices', fontsize=12)
plt.legend()

# Displaying the plot
plt.grid(True)
plt.show()
```

```
Training R2 Score of sale: 0.9838534538538241
R2 Score random forest: 0.9837517229163514
Mean Absolute Error random forest: 0.062127429682532175
Mean Squared Error random forest: 0.007989706984487201
Root Mean Squared Error Randon forest: 0.0893851608740914
```



Desicion tree

In [154…

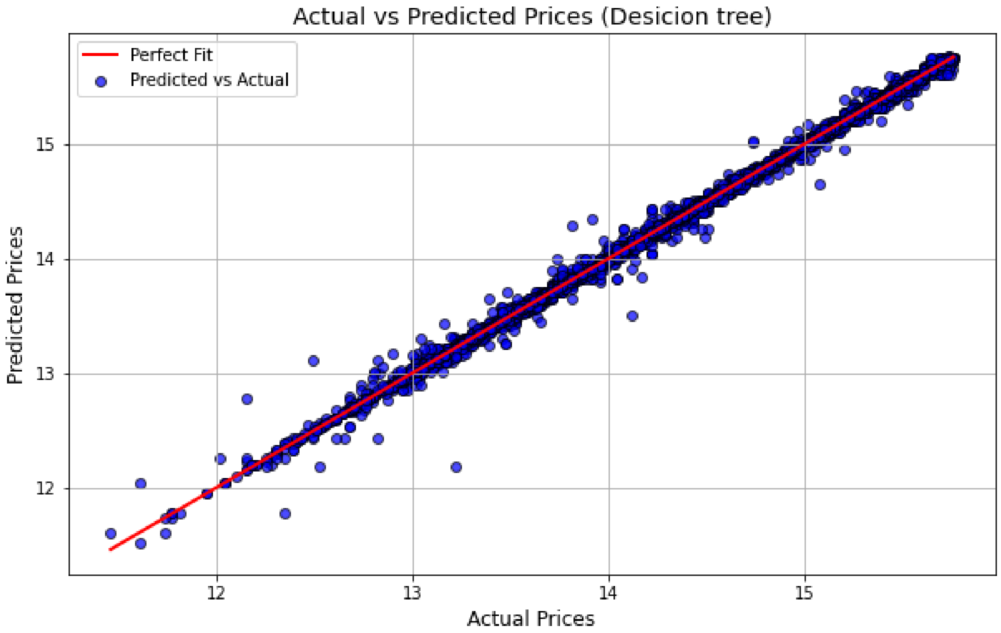
```
# Without parameters
```

In [155…

```
model_tree = DecisionTreeRegressor(random_state=42)

# Fitting the model on the training data
model_tree.fit(X_train, y_train)

# Predicting using the testing data for sales data
y_pred = model_tree.predict(X_test)

# Calculate metrcs for sales data
mae_tree = mean_absolute_error(y_test, y_pred)
mse_tree = mean_squared_error(y_test, y_pred)
rmse_tree = np.sqrt(mse_tree)
r2_tree = r2_score(y_test, y_pred)

y_train_pred = model_tree.predict(X_train)

# Calculate regression metrics for training set
r2_train_d_s = r2_score(y_train, y_train_pred)
print(f"Training R2 Score of sale: {r2_train_d_s}")

print(f"R2 Score  for sales: {r2_tree}")
print(f"Mean Absolute Error of Decision Tree for sales: {mae_tree}")
print(f"Mean Squared Error of Decision Tree for sales: {mse_tree}")
print(f"Root Mean Squared Error Decision Tree  for sales: {rmse_tree}")
```

```python
# Plotting the Actual vs Predicted values
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, color='blue', edgecolors='k', alpha=0.7, label='Predicted vs Actual')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', linewidth=2, label='Perfect Fit')

# Adding titles and labels
plt.title('Actual vs Predicted Prices (Desicion tree)', fontsize=14)
plt.xlabel('Actual Prices', fontsize=12)
plt.ylabel('Predicted Prices', fontsize=12)
plt.legend()

# Displaying the plot
plt.grid(True)
plt.show()
```

```
Training R2 Score of sale: 0.9999999999999959
R2 Score  for sales: 0.9984913786637591
Mean Absolute Error of Decision Tree for sales: 0.008759376056976841
Mean Squared Error of Decision Tree for sales: 0.0007418289560829468
Root Mean Squared Error Decision Tree  for sales: 0.027236537152930194
```

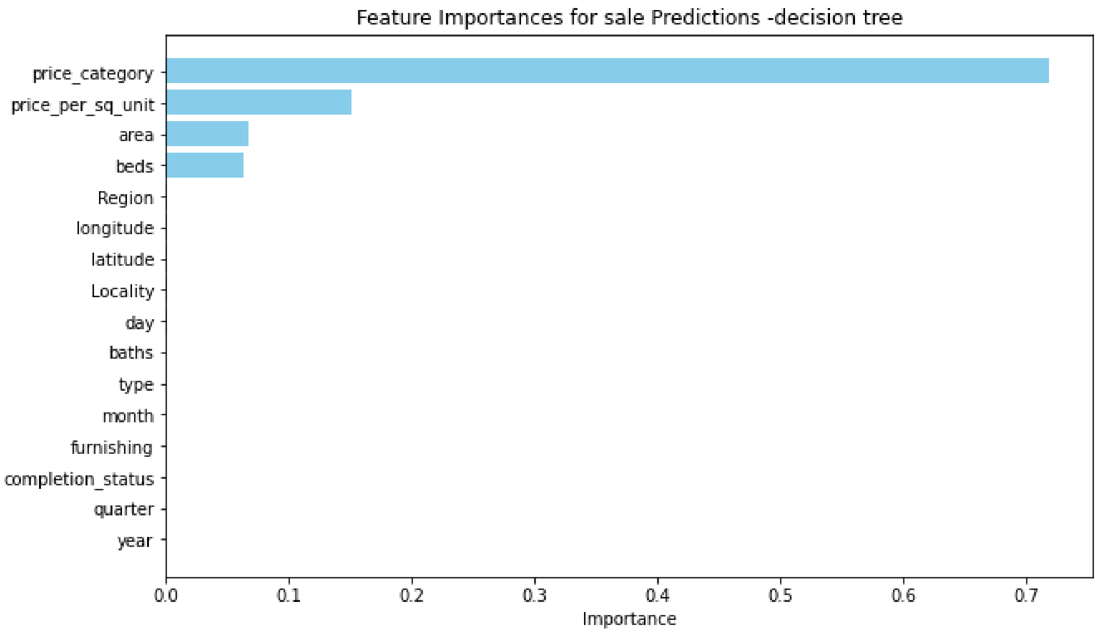


In [156…

```python
feature_importance_rent = model_tree.feature_importances_

#creating a DataFrame for a tabular display of features and their importance
coef_importance_rent = pd.DataFrame({'Feature': X.columns, 'Importance': feature_importance_rent})
coef_importance_rent = coef_importance_rent.sort_values(by='Importance', ascending=False)

print(coef_importance_rent)

#Plotting the feature importances
plt.figure(figsize=(10, 6))
plt.barh(coef_importance_rent['Feature'], coef_importance_rent['Importance'], color='skyblue')
```

```
plt.xlabel('Importance')
plt.title('Feature Importances for sale Predictions -decision tree')
plt.gca().invert_yaxis()  # To display the highest importance at the top
plt.show()
```

```
              Feature  Importance
15     price_category    0.718395
14  price_per_sq_unit    0.150331
9                area    0.066882
5                beds    0.062853
3              Region    0.000560
1           longitude    0.000359
0            latitude    0.000262
2            Locality    0.000110
12                day    0.000084
6               baths    0.000038
4                type    0.000035
11              month    0.000027
8          furnishing    0.000022
7   completion_status    0.000019
13            quarter    0.000017
10               year    0.000005
```



In [157…

```
# With parameters
```

In [158…

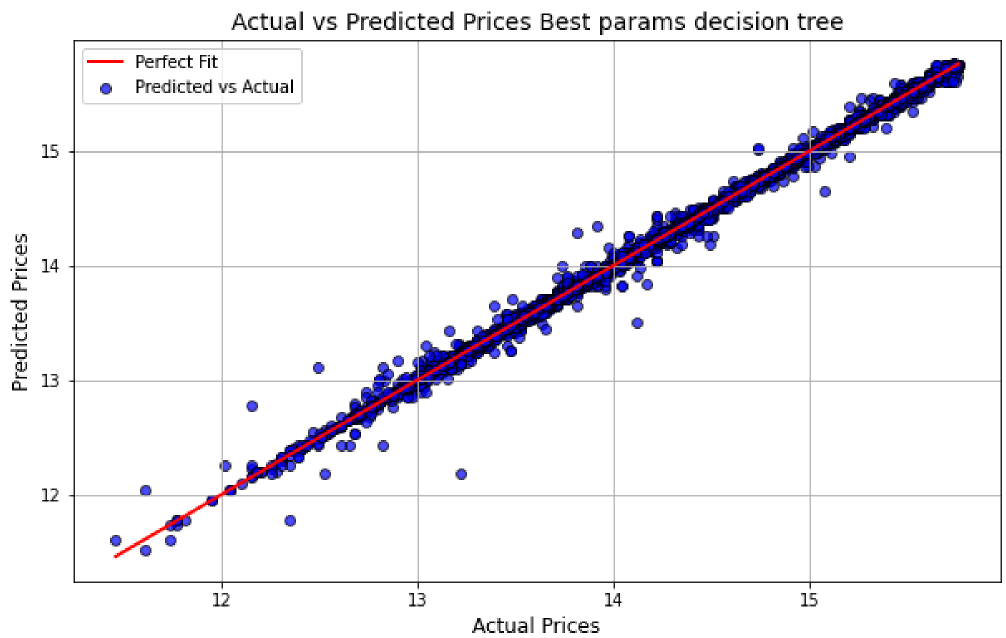
```
# Set up the parameter grid
param_grid = {
    'max_depth': [3, 5, 10],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 5, 10],
    'max_features': [None, 'sqrt', 'log2']
}
```

```python
# Initialize GridSearchCV with DecisionTreeRegressor for sale
grid_search = GridSearchCV(DecisionTreeRegressor(random_state=42), param_grid, cv=5, scoring='r2')

# Fit the grid search
grid_search.fit(X_train, y_train)

# Best parameters and model
print("Best parameters found: ", grid_search.best_params_)
best_tree = grid_search.best_estimator_

# Make predictions using the best model
y_pred_best_tree = best_tree.predict(X_test)

y_train_pred = best_tree.predict(X_train)

# Calculate regression metrics for training set
r2_train_dh_s = r2_score(y_train, y_train_pred)
print(f"Training R2 Score of sale: {r2_train_dh_s}")

# Evaluate the model
mae_tree_g = mean_absolute_error(y_test, y_pred_best_tree)
mse_tree_g = mean_squared_error(y_test, y_pred_best_tree)
rmse_tree_g = np.sqrt(mse_tree_g)
r2_tree_g = r2_score(y_test, y_pred_best_tree)

print(f"R2 Score for sale best : {r2_tree_g}")
print(f"Mean Absolute Error of Decision Tree for sale using gridsearch: {mae_tree_g}")
print(f"Mean Squared Error of Decision Tree for sale using gridsearch: {mse_tree_g}")
print(f"Root Mean Squared Error of Decision Tree for sale using gridsearch: {rmse_tree_g}")
```

```
Best parameters found:  {'max_depth': 10, 'max_features': None, 'min_samples_leaf': 1, 'min_samples_split': 5}
Training R2 Score of sale: 0.9906648984205506
R2 Score for sale best : 0.9899519953171332
Mean Absolute Error of Decision Tree for sale using gridsearch: 0.04791803790611975
Mean Squared Error of Decision Tree for sale using gridsearch: 0.004940869286113459
Root Mean Squared Error of Decision Tree for sale using gridsearch: 0.07029131728822173
```

In [159…

```python
# Plotting the Actual vs Predicted values
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, color='blue', edgecolors='k', alpha=0.7, label='Predicted vs Actual')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', linewidth=2, label='Perfect Fit')

# Adding titles and labels
plt.title('Actual vs Predicted Prices Best params decision tree', fontsize=14)
plt.xlabel('Actual Prices', fontsize=12)
plt.ylabel('Predicted Prices', fontsize=12)
plt.legend()

# Displaying the plot
plt.grid(True)
plt.show()
```



In [160…

```
# Created a dictionary to store metrics for each model
metrics = {
    'Model': [
        'Linear Regression',
        'Lasso (Random Alpha)',
        'Lasso Regression (Best Alpha)',
        'XGBoost',
        'XGBoost[Hyperparameter]',
        'Random Forest',
        'Random Forest[Hyperparameter]',
        'Decision Tree',
        'Decision Tree gridsearch'
    ],
    'R2 Score ': [
        R_2_s_linear,
        R_2_s_l,
        r2_lasso_grid,
        r2_x_sale,
        r2_x_s,
        r2_sale,
        r2,
        r2_tree,
        r2_tree_g
    ],
    'Mean Absolute Error': [
        mae_linear,
        mae_l,
        mae_lasso_grid,
        mae_x_sale,
        mae_x_s,
        mae_random_sale,
```

```
        mae_random_s,
        mae_tree,
        mae_tree_g
    ],
    'Mean Squared Error': [
        mse_linear,
        mse_l,
        mse_best_lasso_grid,
        mse_x_sale,
        mse_x_s,
        mse_random_sale,
        mse_random_s,
        mse_tree,
        mse_tree_g
    ],
    'Root Mean Squared Error': [
        rmse_linear,
        rmse_l,
        np.sqrt(mse_best_lasso_grid),
        rmse_x_sale,
        rmse_x_s,
        rmse_random_sale,
        rmse_r_s,
        rmse_tree,
        rmse_tree_g
    ]
}
metrics_df = pd.DataFrame(metrics)

# Display the metrics table for easy understanding
print(metrics_df)
```

```
                          Model  R2 Score   Mean Absolute Error  \
0             Linear Regression   0.787327             0.203156
1          Lasso (Random Alpha)   0.634411             0.310486
2  Lasso Regression (Best Alpha)  0.810514             0.205673
3                       XGBoost   0.998427             0.016968
4        XGBoost[Hyperparameter]  0.981129             0.075236
5                 Random Forest   0.999464             0.005883
6  Random Forest[Hyperparameter]  0.983752             0.062127
7                 Decision Tree   0.998491             0.008759
8      Decision Tree gridsearch   0.989952             0.047918

   Mean Squared Error  Root Mean Squared Error
0            0.104577                 0.323384
1            0.179770                 0.423993
2            0.093175                 0.305246
3            0.000774                 0.027814
4            0.009280                 0.096330
5            0.000264                 0.016233
6            0.007990                 0.089385
7            0.000742                 0.027237
8            0.004941                 0.070291
```

In [161…

```python
# Created figure and axis for the evalution table
fig, ax = plt.subplots(figsize=(12, 4))
ax.axis('tight')
ax.axis('off')
table = ax.table(cellText=metrics_df.values,
                 colLabels=metrics_df.columns,
                 cellLoc='center',
                 loc='center')
#Styling
table.auto_set_font_size(False)
table.set_fontsize(14)
table.scale(2, 4)

#header bold and increase its font size
for (i, j), cell in table.get_celld().items():
    if i == 0:  # Header row
        cell.set_text_props(fontweight='bold', fontsize=16)
    else:
        cell.set_fontsize(14)
plt.show()
```

Model	R2 Score	Mean Absolute Error	Mean Squared Error	Root Mean Squared Error
Linear Regression	0.7873270337916629	0.20315595472671413	0.1045769145109135	0.32338354087818616
Lasso (Random Alpha)	0.6344111627744164	0.3104858042352185	0.1797696870378505	0.4239925554038072
Lasso Regression (Best Alpha)	0.8105140629435301	0.20567278139378503	0.09317523987116877	0.30524619550646126
XGBoost	0.9984267439419624	0.016967793071900023	0.0007736115558947283	0.027813873442847335
XGBoost[Hyperparameter]	0.9811286402749706	0.07523551603197816	0.00927954599897674	0.09633040018071523
Random Forest	0.9994640864823852	0.00588338922363119	0.0002635228309269395	0.016233386304987
Random Forest[Hyperparameter]	0.9837517229163514	0.062127429682532175	0.007989706984487201	0.0893851608740914
Decision Tree	0.9984913786637591	0.008759376056976841	0.0007418289560829468	0.027236537152930194
Decision Tree gridsearch	0.9899519953171332	0.04791803790611975	0.004940869286113459	0.07029131728822173

Multilayer Perceptron

In [35]:

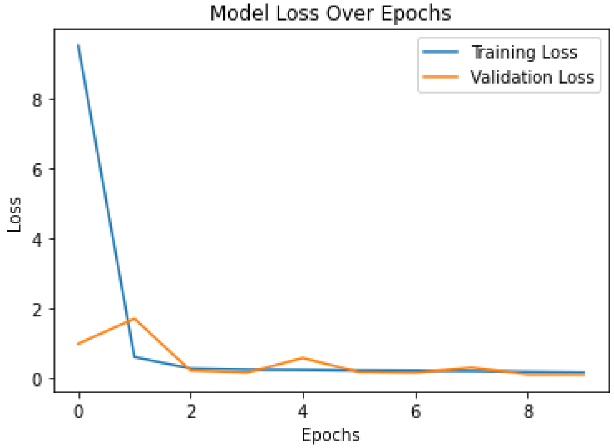
```python
import numpy as np

#building the model for sales
```

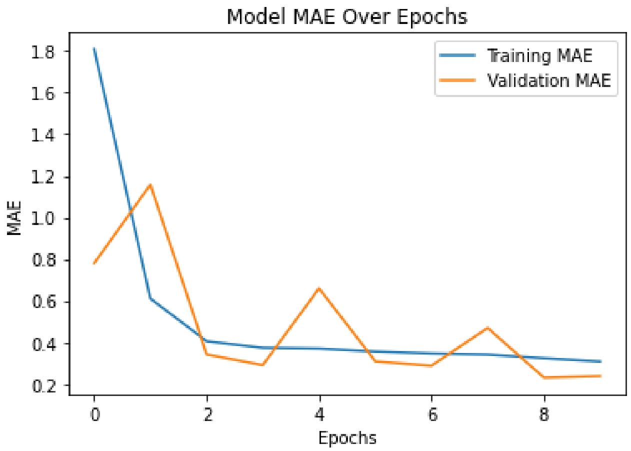
```python
model = keras.Sequential([
    layers.Dense(64, activation='relu', input_shape=(X_train.shape[1],)),  # Input layer
    layers.Dense(32, activation='relu'),  # Hidden layer
    layers.Dense(1)  # Output layer
])

model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mae'])

#Training model
h_sale = model.fit(X_train, y_train, epochs=10, batch_size=32, validation_split=0.2)

# Evaluating the model for sales
test_loss, test_mae = model.evaluate(X_test, y_test)
y_pred = model.predict(X_test).flatten()

# metrics mse
mse = np.mean((y_test - y_pred) ** 2)

# Calculate RMSE
rmse = np.sqrt(mse)

# Calculate AMSE
n = X_test.shape[0]  # Number of observations
p = X_test.shape[1]  # Number of features
amse = mse * (n / (n - p))

# Calculate R2 for sales
ss_res = np.sum((y_test - y_pred) ** 2)
ss_tot = np.sum((y_test - np.mean(y_test)) ** 2)
r2_score = 1 - (ss_res / ss_tot)

# Display all results
print(f'Test MAE: {test_mae}')
print(f'Test RMSE: {rmse}')
print(f'Test AMSE: {amse}')
print(f'R² Score: {r2_score}')
```

```
WARNING:tensorflow:From c:\users\nishu\appdata\local\programs\python\python39\lib\site-packages\keras\src\backend.py:873: The name tf.get_default_graph is deprecated. Please use tf.compat.v1.get_default_graph instead.

WARNING:tensorflow:From c:\users\nishu\appdata\local\programs\python\python39\lib\site-packages\keras\src\optimizers\__init__.py:309: The name tf.train.Optimizer is deprecated. Please use tf.compat.v1.train.Optimizer instead.

Epoch 1/10
WARNING:tensorflow:From c:\users\nishu\appdata\local\programs\python\python39\lib\site-packages\keras\src\utils\tf_utils.py:492: The name tf.ragged.RaggedTensorValue is deprecated. Please use tf.compat.v1.ragged.RaggedTensorValue instead.

WARNING:tensorflow:From c:\users\nishu\appdata\local\programs\python\python39\lib\site-packages\keras\src\engine\base_layer_utils.py:384: The name tf.executing_eagerly_outside_functions is deprecated. Please use tf.compat.v1.executing_eagerly_outside_functions instead.

844/844 [==============================] - 5s 4ms/step - loss: 9.5183 - mae: 1.8095 - val_loss: 1.0006 - val_mae: 0.7798
Epoch 2/10
844/844 [==============================] - 2s 2ms/step - loss: 0.6235 - mae: 0.6103 - val_loss: 1.7159 - val_mae: 1.1581
Epoch 3/10
844/844 [==============================] - 3s 3ms/step - loss: 0.2939 - mae: 0.4049 - val_loss: 0.2377 - val_mae: 0.3417
Epoch 4/10
844/844 [==============================] - 4s 4ms/step - loss: 0.2603 - mae: 0.3734 - val_loss: 0.1719 - val_mae: 0.2903
Epoch 5/10
844/844 [==============================] - 3s 3ms/step - loss: 0.2507 - mae: 0.3697 - val_loss: 0.5952 - val_mae: 0.6598
Epoch 6/10
844/844 [==============================] - 3s 3ms/step - loss: 0.2329 - mae: 0.3557 - val_loss: 0.1892 - val_mae: 0.3083
Epoch 7/10
844/844 [==============================] - 2s 3ms/step - loss: 0.2231 - mae: 0.3461 - val_loss: 0.1663 - val_mae: 0.2873
Epoch 8/10
844/844 [==============================] - 3s 3ms/step - loss: 0.2158 - mae: 0.3411 - val_loss: 0.3201 - val_mae: 0.4689
Epoch 9/10
844/844 [==============================] - 2s 3ms/step - loss: 0.1938 - mae: 0.3235 - val_loss: 0.1201 - val_mae: 0.2302
Epoch 10/10
844/844 [==============================] - 2s 3ms/step - loss: 0.1787 - mae: 0.3080 - val_loss: 0.1203 - val_mae: 0.2374
452/452 [==============================] - 1s 2ms/step - loss: 0.1394 - mae: 0.2443
452/452 [==============================] - 1s 1ms/step
Test MAE: 0.24434716999530792
Test RMSE: 0.37334716183598726
Test AMSE: 0.13954258199671074
R² Score: 0.7165332185293369
```

In [37]:
```python
import matplotlib.pyplot as plt

#Visualizing the training and validation loss over epochs
plt.plot(h_sale.h_sale['loss'], label='Training Loss')
plt.plot(h_sale.h_sale['val_loss'], label='Validation Loss')
plt.title('Model Loss Over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

#Visualizing the training and validation MAE over epochs
plt.plot(h_sale.h_sale['mae'], label='Training MAE')
plt.plot(h_sale.h_sale['val_mae'], label='Validation MAE')
plt.title('Model MAE Over Epochs')
plt.xlabel('Epochs')
```

```
plt.ylabel('MAE')
plt.legend()
plt.show()
```





The data splitting to 20% of the training set, that is to say, 20% of the 70% of the original data used for validation. Computation: 20% of 70% equals 14% of the original data that is reserved for validation. Model Loss Across Epochs (Upper Plot)

Epoch 0: Training Loss: starts off very high, at about 9.0, which indicates that the initial predictions are considerably far from the ground truth in the training data. Validation Loss: starts at about 2.0, a sign that this model does relatively better on validation data than the initial impression it left with the training data. This mainly occurs because of the model initialization or randomness in the early stages.

Epoch 1: Training Loss: It sharply drops from 9.0 to about 0.5. That big drop means the model starts to fit very fast already on the first epoch, which radically improves its predictions. Validation Loss: It goes down to about 0.4, which is good and reflects a decent decrease in error not only on unseen data but also it is not so drastic for the training set.

Epochs 2–3: Training Loss: Still goes down to around 0.1 at epoch 3, which means that the model continues at each epoch to fit the training data better. Validation Loss: It was generally a number around 0.2 to 0.3, but through these epochs, it fluctuates a bit. That would mean the model generalizes better but becomes a bit sensitive to the validation data.

Epochs 4–9: Training Loss: It goes down to almost zero at epoch 4 and stays like that throughout the rest of the training. That basically means the model fits the training data perfectly and has a really small error in it. Validation Loss: Stays quite close to 0.2, fluctuates around that with little spikes, but nothing too dramatic in increase. Overall, it means the model generalizes well, though the fact that validation loss is not decreasing anymore could mean that the model reached its full capacity in terms of generalizing.

Specific Insight: That would mean, if after a few epochs the model fits perfectly, the training loss would drop to zero, then the validation loss would flatten around 0.2; that would mean its performance on unseen data does not really improve after epoch 4, so that by this point it may have learned most of what it can learn from this data.

2. Model MAE Over Epochs (Bottom Plot)

Epoch 0: Training MAE: starts at around 1.8, which indicates that the model's initial absolute prediction error is very large-that is, on average, predictions are off by about 1.8 units, whatever unit the target variable represents. Validation MAE: starts at around 1.2, which means slightly better performance on the validation set compared to the training set at the beginning, and similar to what the loss curve did. Epoch 1: MAE: This also decreases, but not as abruptly; it stabilizes at 0.8. This means the performance of the model on unseen data improves, too, but not quite so abruptly.

Epochs 2–3: Training MAE: You can see that it decreases to about 0.3 until epoch 3, reflecting continued improvement in the prediction accuracy on the training data. Validation MAE: Decreases to around 0.5 but again fluctuates more than the training MAE. The error on the validation data is lower, though the fluctuations do hint at some sensitivity regarding the validation set.

Epochs 4–9: Training MAE: Reaches 0.2 or lower after epoch 4 and remains there. This means that on the training data, the model is making predictions always very close to the actual values. MAE Validation: During this period, it continues to oscillate between 0.3 and 0.4. While the MAE of this range is lower than in the earlier epochs, such oscillation suggests further optimization for better generalization may still be achieved by this model, which keeps doing fairly good predictions. Specific Insight: Its training MAE rapidly goes down to 0.2, suggesting that after a few epochs, this model already makes really accurate predictions on the training set. The validation MAE fluctuated a lot around the value of 0.3 to 0.4, which means there is some variability when generalizing to unseen data, but the error remains relatively small.

Thank you