Module 2 Assignment 1: House Prices

Aaron Chen, Nikhil Prabhu, Elvis Matos, Reese Mayer

Introduction:

In this assignment, our group explored two different datasets: the primary dataset, named 'train', and the secondary dataset known as 'test'. The training dataset served as our foundation, providing the same columns of data as the test dataset, except with one additional column: SalesPrice. With these datasets, we created different regression models, as well as indicators, dichotomous, and piecewise model components, to generate a predictive model. Through continuous construction, revision, and refinement, we were able to generate a working predictive model using key columns identified from the training dataset to create accurate predictions of sales prices within the test dataset.

EDA:

Within the Exploratory data analysis portion we grabbed specific columns such as Id, TotalBsmtSF, 1stFlrSF, 2ndFlrSF, LotFrontage, LotArea, and SalePrice for the training data. We collected the same columns and were planning on using these variables to help predict the Sales Price. Since we took out a lot of columns many Null values were taken out so we used a command just to count the amount of null values in both tables to make sure. Furthermore, we saw when creating a bar plot based on the training data that the values are skewed to the left which represents many outliers. Another example is the addition of boxplot which was able to show that the data had several outliers placed father in Sales Price greater than $700,000

Goodness of Fit:

R-squared measures the proportion of variance in the dependent variable (sales prices) that is predictable from the independent variable (square footage and lot size). A moderate R-square value (0.60) was found in the training data which may suggest not an ideal fit between the model and the training data. Similarly, the MSE and RMSE were found to be very high indicating that the model's predictions have larger errors, meaning the model is less accurate.

Module 2 Assignment 1: House Prices

Aaron Chen, Nikhil Prabhu, Elvis Matos, Reese Mayer

Normally we want the MSE or MSE on the validation and test data to be consistent with the training sets. If the MSE or RMSE on the validation data is significantly higher than the training data, it could indicate that the model is overfitting and not generalizing well. We found the validation set MSE to be significantly lower, taking into account that synthetic data had to be generated.

Discuss the Model:

Our model, when evaluated by Kaggle gave us a score of 0.56 out of a perfect 1.0 rating. While this score may not seem particularly impressive at first glance, there are several factors to consider. One of the most important factors to consider is that we chose what variables to include in our predictive model. Out of the 79 factors that were initially provided to us, we decided to include about 7 of them in our analysis. This decision impacted our results in the end because we reduced the amount of information that our model could have potentially had access to. Therefore, despite our 0.56, we believe that our model did quite well given our chosen information.

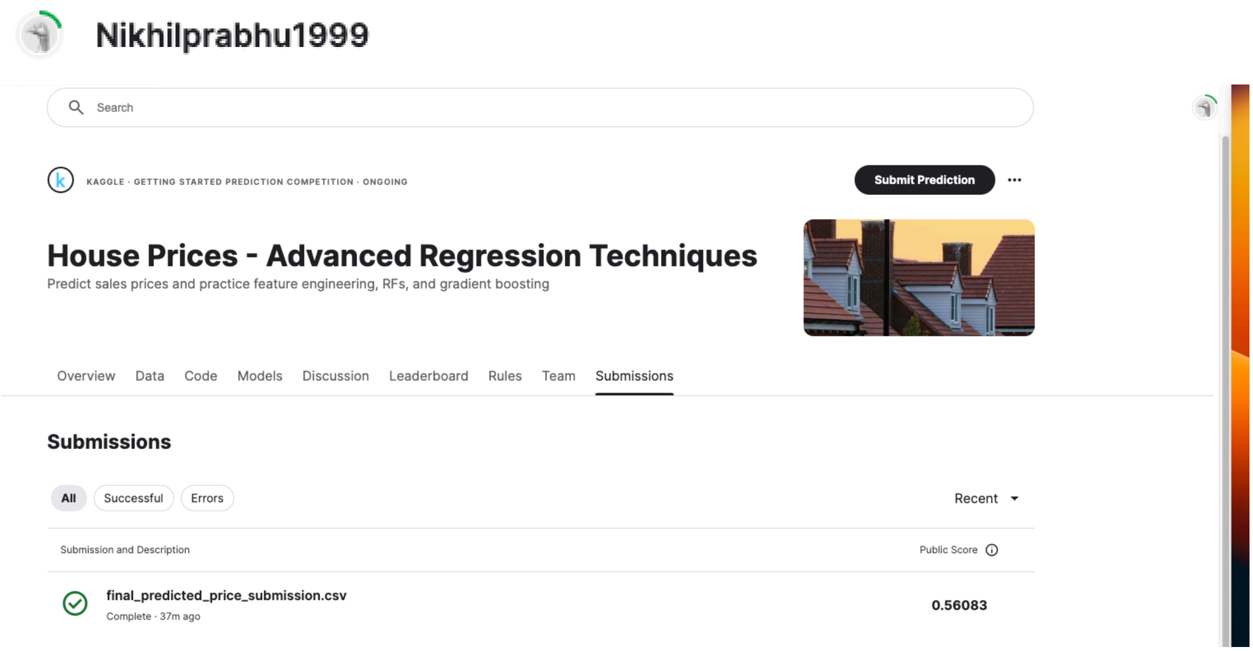
Conclusion:

In concluding our group project, we embarked on an analytical journey, diving into ‘train’ and ‘test’ datasets to forge a predictive model for real estate sales prices. Through careful exploratory data analysis and strategic selection of key variables, we addressed data integrity and honed in on potent predictors despite limiting ourselves to a fraction of the available data. Our model’s moderate R-squared value and a Kaggle score of 0.56 – which suggests room for improvement – provided valuable learning in data analytics and the significance of variable selection.

Module 2 Assignment 1: House Prices

Aaron Chen, Nikhil Prabhu, Elvis Matos, Reese Mayer

APPENDIX:



Module 2 Assignment 1 - House Regression

April 7, 2024

[1]:
```
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.model_selection import KFold, cross_val_score
from sklearn.linear_model import LinearRegression
from sklearn.datasets import load_iris
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import PolynomialFeatures
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
import scipy.stats as stats
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
```

1 Research Question

The goal is to establish and research the variables of house square footage and lot size that will impact the final price of the home.

2 Q1

Conduct your analysis using a cross-validation design.

[3]:
```
train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')

train_p = train[['Id','TotalBsmtSF','1stFlrSF','2ndFlrSF','LotFrontage',␣
↪'LotArea','SalePrice']]
test_p = test[['Id','TotalBsmtSF','1stFlrSF','2ndFlrSF','LotFrontage',␣
↪'LotArea']]
```

[4]:
```
train_p = train_p.dropna()
train_p
```

[4]:
```
        Id  TotalBsmtSF  1stFlrSF  2ndFlrSF  LotFrontage  LotArea  SalePrice
0        1          856       856       854         65.0     8450     208500
1        2         1262      1262         0         80.0     9600     181500
2        3          920       920       866         68.0    11250     223500
3        4          756       961       756         60.0     9550     140000
4        5         1145      1145      1053         84.0    14260     250000
…     …           …        …         …            …       …        …
1455  1456          953       953       694         62.0     7917     175000
1456  1457         1542      2073         0         85.0    13175     210000
1457  1458         1152      1188      1152         66.0     9042     266500
1458  1459         1078      1078         0         68.0     9717     142125
1459  1460         1256      1256         0         75.0     9937     147500

[1201 rows x 7 columns]
```

[5]:
```python
test_p = test_p.dropna()
test_p1 = test_p.drop(test_p.index[1:31])
test_p1
```

[5]:
```
        Id  TotalBsmtSF  1stFlrSF  2ndFlrSF  LotFrontage  LotArea
0     1461        882.0       896         0         80.0    11622
32    1493       1208.0      1494         0         39.0    15410
33    1494       1231.0      1251      1098         85.0    13143
34    1495       1390.0      1402       823         88.0    11134
35    1496       1488.0      1488         0         25.0     4835
…     …           …        …         …            …       …
1454  2915        546.0       546       546         21.0     1936
1455  2916        546.0       546       546         21.0     1894
1456  2917       1224.0      1224         0        160.0    20000
1457  2918        912.0       970         0         62.0    10441
1458  2919        996.0       996      1004         74.0     9627

[1201 rows x 6 columns]
```

[6]:
```python
#Initialize the Linear Regression Model
model = LinearRegression()
```

[7]:
```python
# Perform 5-Fold Cross-Validation
scores = cross_val_score(model, train_p, test_p1, cv = 5)
scores
```

[7]:
```
array([0.15800698, 0.14853421, 0.15953752, 0.1632745 , 0.14710362])
```

3 Q2

Conduct EDA and provide appropriate visualizations in the process.

[9]:
```
train_p.isnull().sum()
```

[9]:
```
Id             0
TotalBsmtSF    0
1stFlrSF       0
2ndFlrSF       0
LotFrontage    0
LotArea        0
SalePrice      0
dtype: int64
```

[10]:
```
test_p.isnull().sum()
```

[10]:
```
Id             0
TotalBsmtSF    0
1stFlrSF       0
2ndFlrSF       0
LotFrontage    0
LotArea        0
dtype: int64
```

[11]:
```
print(train_p.describe())
```

```
                Id  TotalBsmtSF     1stFlrSF     2ndFlrSF  LotFrontage  \
count  1201.000000  1201.000000  1201.000000  1201.000000  1201.000000
mean    727.037469  1059.384679  1158.437968   346.073272    70.049958
std     421.038976   448.307125   386.257235   435.143451    24.284752
min       1.000000     0.000000   334.000000     0.000000    21.000000
25%     366.000000   784.000000   876.000000     0.000000    59.000000
50%     724.000000   990.000000  1082.000000     0.000000    69.000000
75%    1092.000000  1309.000000  1383.000000   728.000000    80.000000
max    1460.000000  6110.000000  4692.000000  2065.000000   313.000000

             LotArea      SalePrice
count    1201.000000    1201.000000
mean     9951.698585  180770.480433
std      7924.353975   83389.519866
min      1300.000000   34900.000000
25%      7420.000000  127500.000000
50%      9262.000000  159500.000000
75%     11249.000000  213500.000000
max    215245.000000  755000.000000
```

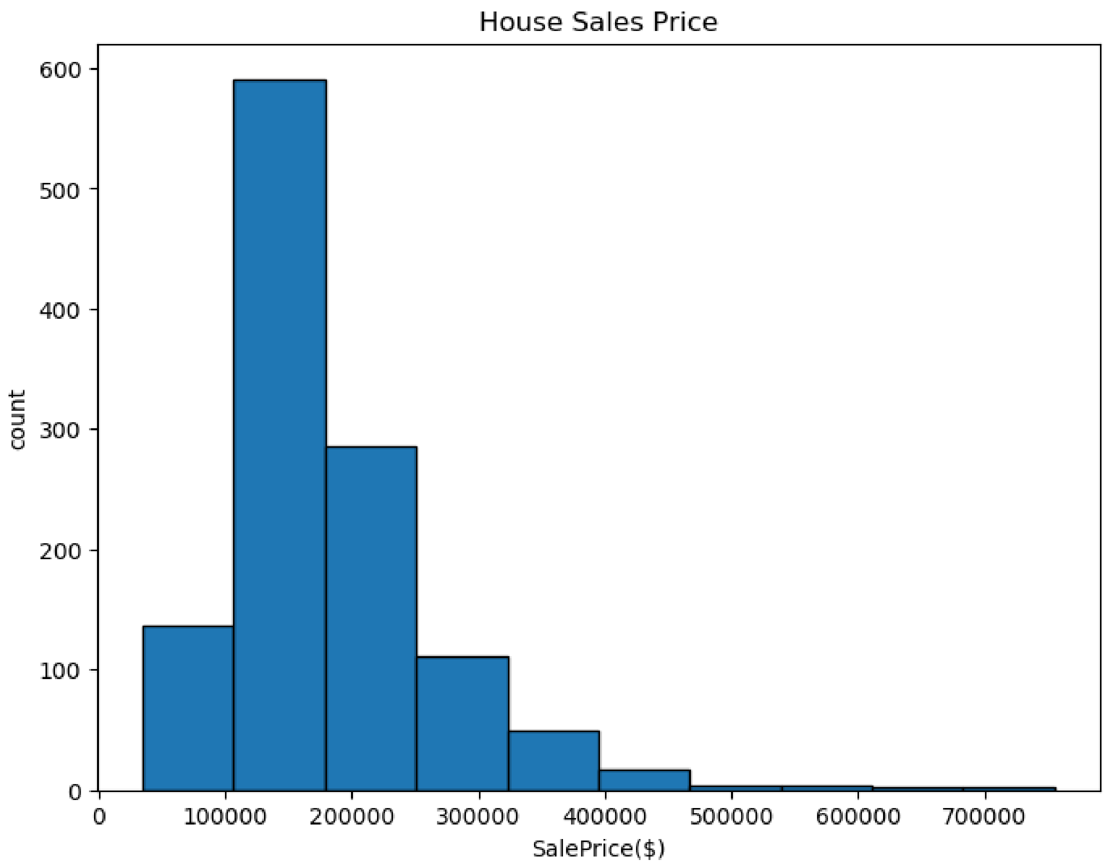
[12]:
```
print(test_p.describe())
```

```
                Id  TotalBsmtSF     1stFlrSF     2ndFlrSF  LotFrontage  \
count  1231.000000  1231.000000  1231.000000  1231.000000  1231.000000
mean   2191.450853  1042.466288  1148.472786   317.662063    68.555646
std     425.025588   452.839348   408.934484   408.077429    22.369112
```

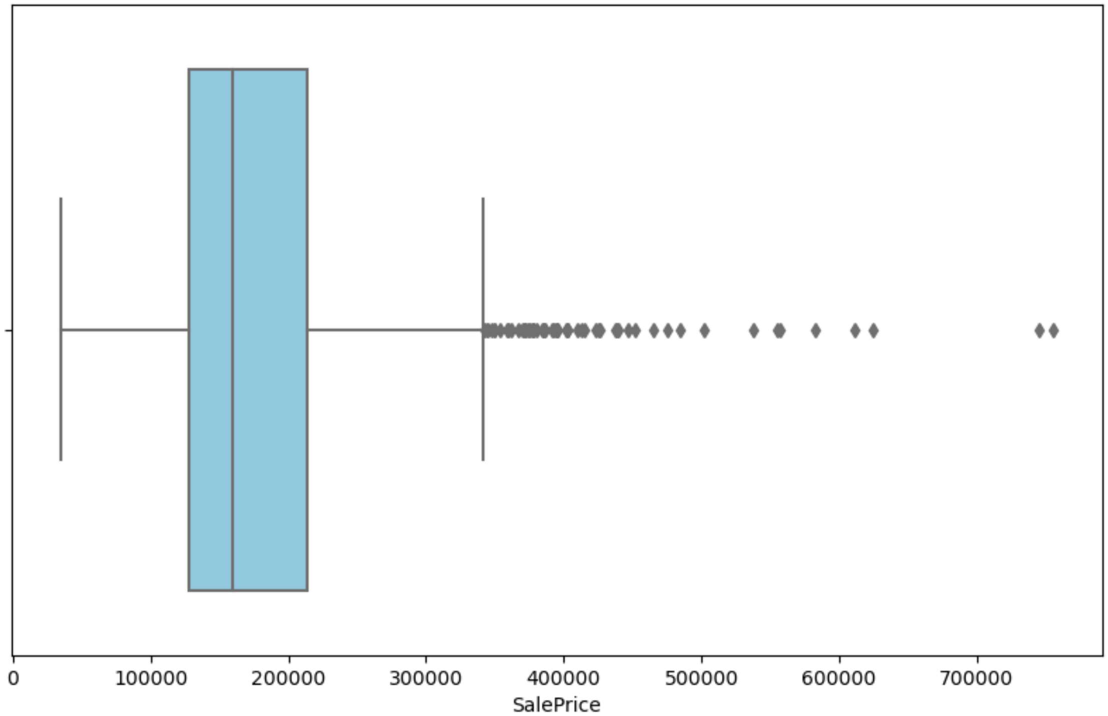
```
min     1461.000000     0.000000   407.000000     0.000000    21.000000
25%     1821.500000   768.000000   864.000000     0.000000    58.000000
50%     2198.000000   976.000000  1064.000000     0.000000    67.000000
75%     2554.500000  1309.000000  1377.500000   660.500000    80.000000
max     2919.000000  5095.000000  5095.000000  1862.000000   200.000000

            LotArea
count   1231.000000
mean    9509.337124
std     4502.130836
min     1470.000000
25%     7200.000000
50%     9260.000000
75%    11135.000000
max    51974.000000
```

[13]:

```
# histogram of train data
plt.figure(figsize=(8,6))
plt.hist(train_p['SalePrice'], edgecolor='black')
plt.title('House Sales Price')
plt.xlabel('SalePrice($)')
plt.ylabel('count')
plt.show()
```



[14]:
```
# Boxplot for train data
plt.figure(figsize=(10 , 6))
sns.boxplot(x='SalePrice', data=train_p,
color='skyblue')
plt.show()
```



4 Q3

Build a minimum of two separate regression models using the training set.

[15]:
```
# Create a Linear Regression Model
model = LinearRegression()
```

[16]:
```
train_p
```

[16]:

	Id	TotalBsmtSF	1stFlrSF	2ndFlrSF	LotFrontage	LotArea	SalePrice
0	1	856	856	854	65.0	8450	208500
1	2	1262	1262	0	80.0	9600	181500
2	3	920	920	866	68.0	11250	223500
3	4	756	961	756	60.0	9550	140000
4	5	1145	1145	1053	84.0	14260	250000
…	…	…	…	…	…	…	…
1455	1456	953	953	694	62.0	7917	175000
1456	1457	1542	2073	0	85.0	13175	210000
1457	1458	1152	1188	1152	66.0	9042	266500
1458	1459	1078	1078	0	68.0	9717	142125
1459	1460	1256	1256	0	75.0	9937	147500

[1201 rows x 7 columns]

[17]:
```python
train_x = train_p.drop('SalePrice', axis=1)
train_y = train_p['SalePrice']

# Fit the model
model.fit(train_x, train_y)
```

[17]:
```
LinearRegression()
```

[18]:
```python
# Random Forest Regression Model
rdm_frst = RandomForestRegressor(n_estimators = 100)
rdm_frst.fit(train_x, train_y)
```

[18]:
```
RandomForestRegressor()
```

5 Q4

Evaluate polynomial, indicator, dichotomous, & piecewise model components.

[19]:
```python
# Polynomial Features
degree = 2
poly_features = PolynomialFeatures(degree=degree)
X_poly_train = poly_features.fit_transform(train_x)
X_poly_train
```

[19]:
```
array([[1.0000000e+00, 1.0000000e+00, 8.5600000e+02, …, 4.2250000e+03,
         5.4925000e+05, 7.1402500e+07],
       [1.0000000e+00, 2.0000000e+00, 1.2620000e+03, …, 6.4000000e+03,
         7.6800000e+05, 9.2160000e+07],
       [1.0000000e+00, 3.0000000e+00, 9.2000000e+02, …, 4.6240000e+03,
         7.6500000e+05, 1.2656250e+08],
       …,
       [1.0000000e+00, 1.4580000e+03, 1.1520000e+03, …, 4.3560000e+03,
         5.9677200e+05, 8.1757764e+07],
       [1.0000000e+00, 1.4590000e+03, 1.0780000e+03, …, 4.6240000e+03,
         6.6075600e+05, 9.4420089e+07],
       [1.0000000e+00, 1.4600000e+03, 1.2560000e+03, …, 5.6250000e+03,
         7.4527500e+05, 9.8743969e+07]])
```

[20]:
```python
categorical_cols = train[['MSZoning', 'BldgType']]

# Indicator Variables

indicator_df = pd.get_dummies(categorical_cols)
indicator_df
```

[20]:
```
      MSZoning_C (all)  MSZoning_FV  MSZoning_RH  MSZoning_RL  MSZoning_RM  \
0                    0            0            0            1            0
```

```
1                    0             0             0            1             0
2                    0             0             0            1             0
3                    0             0             0            1             0
4                    0             0             0            1             0
…                  …           …           …           …          …
1455                 0             0             0            1             0
1456                 0             0             0            1             0
1457                 0             0             0            1             0
1458                 0             0             0            1             0
1459                 0             0             0            1             0

      BldgType_1Fam  BldgType_2fmCon  BldgType_Duplex  BldgType_Twnhs  \
0                 1                0                0               0
1                 1                0                0               0
2                 1                0                0               0
3                 1                0                0               0
4                 1                0                0               0
…               …              …              …             …
1455              1                0                0               0
1456              1                0                0               0
1457              1                0                0               0
1458              1                0                0               0
1459              1                0                0               0

      BldgType_TwnhsE
0                   0
1                   0
2                   0
3                   0
4                   0
…                 …
1455                0
1456                0
1457                0
1458                0
1459                0

[1460 rows x 10 columns]
```

[22]:
```python
# Dichotomous Variables

dichotomous_var = pd.get_dummies(categorical_cols)
dichotomous_var
```

[22]:
```
      MSZoning_C (all)  MSZoning_FV  MSZoning_RH  MSZoning_RL  MSZoning_RM  \
0                    0            0            0            1            0
1                    0            0            0            1            0
```

```
2                          0             0             0             1             0
3                          0             0             0             1             0
4                          0             0             0             1             0
…                       …           …           …           …           …
1455                       0             0             0             1             0
1456                       0             0             0             1             0
1457                       0             0             0             1             0
1458                       0             0             0             1             0
1459                       0             0             0             1             0

      BldgType_1Fam  BldgType_2fmCon  BldgType_Duplex  BldgType_Twnhs  \
0                 1                0                0               0
1                 1                0                0               0
2                 1                0                0               0
3                 1                0                0               0
4                 1                0                0               0
…              …                …                …               …
1455              1                0                0               0
1456              1                0                0               0
1457              1                0                0               0
1458              1                0                0               0
1459              1                0                0               0

      BldgType_TwnhsE
0                   0
1                   0
2                   0
3                   0
4                   0
…                …
1455                0
1456                0
1457                0
1458                0
1459                0

[1460 rows x 10 columns]
```

[23]:
```python
#Piecewise Function
new_train = train[['SalePrice', 'OverallCond']].copy()
new_train
```

[23]:
```
      SalePrice  OverallCond
0        208500            5
1        181500            8
2        223500            5
3        140000            5
```

```
4       250000          5
...       ...         ...
1455    175000          5
1456    210000          6
1457    266500          9
1458    142125          6
1459    147500          6

[1460 rows x 2 columns]
```

[24]:
```
breakpoint = 5


# Split the data into two segments based on the breakpoint
X1 = new_train[new_train['OverallCond'] < breakpoint][['OverallCond']]
X2 = new_train[new_train['OverallCond'] >= breakpoint][['OverallCond']]
y1 = new_train[new_train['OverallCond'] < breakpoint]['SalePrice']
y2 = new_train[new_train['OverallCond'] >= breakpoint]['SalePrice']

# Fitting linear regression models for each segment
model1 = LinearRegression().fit(X1, y1)
model2 = LinearRegression().fit(X2, y2)



# Predictions for each segment
y_pred1 = model1.predict(X1)
y_pred2 = model2.predict(X2)
```

6 Q5

Create at least one feature from the data set.

[25]:
```
train_p['TotalSF'] = train_p['TotalBsmtSF'] + train_p['1stFlrSF'] +␣
 ↪train_p['2ndFlrSF']
test_p['TotalSF'] = test_p['TotalBsmtSF'] + test_p['1stFlrSF'] +␣
 ↪test_p['2ndFlrSF']

train_p.head()
```

[25]:
```
   Id  TotalBsmtSF  1stFlrSF  2ndFlrSF  LotFrontage  LotArea  SalePrice  \
0   1          856       856       854         65.0     8450     208500
1   2         1262      1262         0         80.0     9600     181500
2   3          920       920       866         68.0    11250     223500
3   4          756       961       756         60.0     9550     140000
4   5         1145      1145      1053         84.0    14260     250000
```

```
   TotalSF
0     2566
1     2524
2     2706
3     2473
4     3343
```

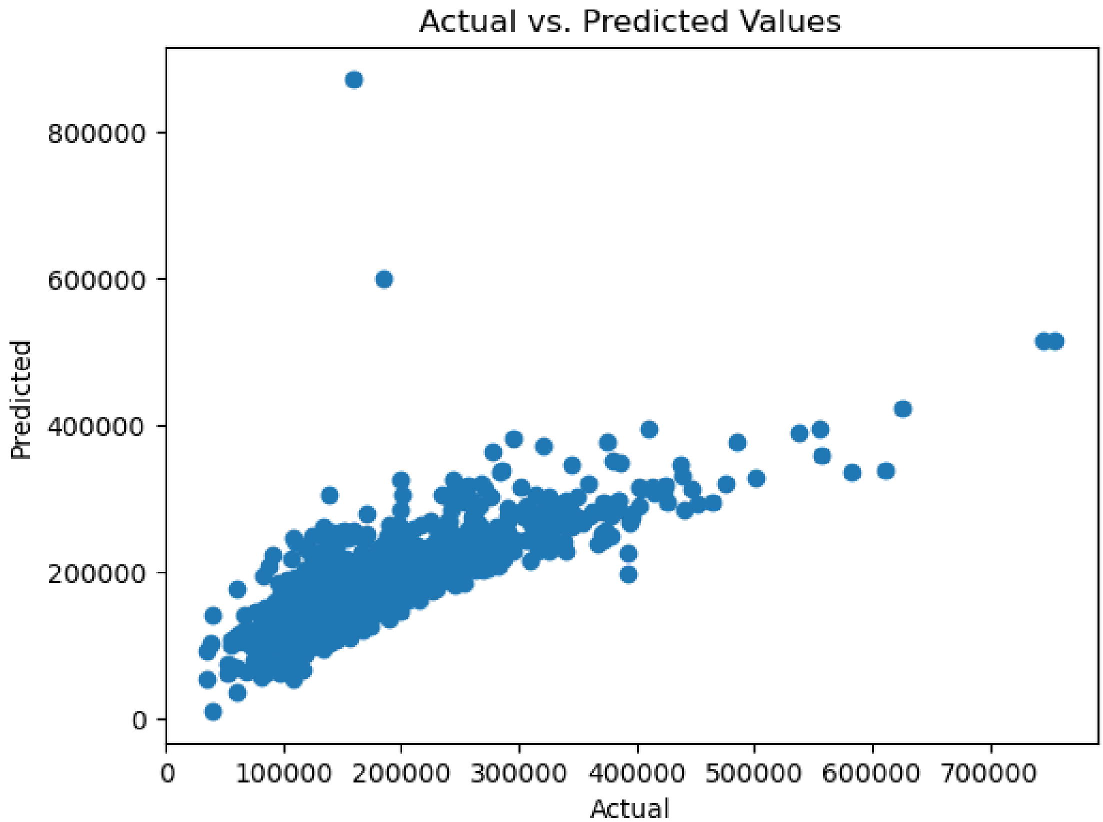
[26]:
```
train_p['TotalLotSize'] = train_p['LotFrontage'] + train_p['LotArea']
test_p['TotalLotSize'] = test_p['LotFrontage'] + test_p['LotArea']

train_p.head()
```

[26]:
```
   Id  TotalBsmtSF  1stFlrSF  2ndFlrSF  LotFrontage  LotArea  SalePrice  \
0   1          856       856       854         65.0     8450     208500
1   2         1262      1262         0         80.0     9600     181500
2   3          920       920       866         68.0    11250     223500
3   4          756       961       756         60.0     9550     140000
4   5         1145      1145      1053         84.0    14260     250000

   TotalSF  TotalLotSize
0     2566        8515.0
1     2524        9680.0
2     2706       11318.0
3     2473        9610.0
4     3343       14344.0
```

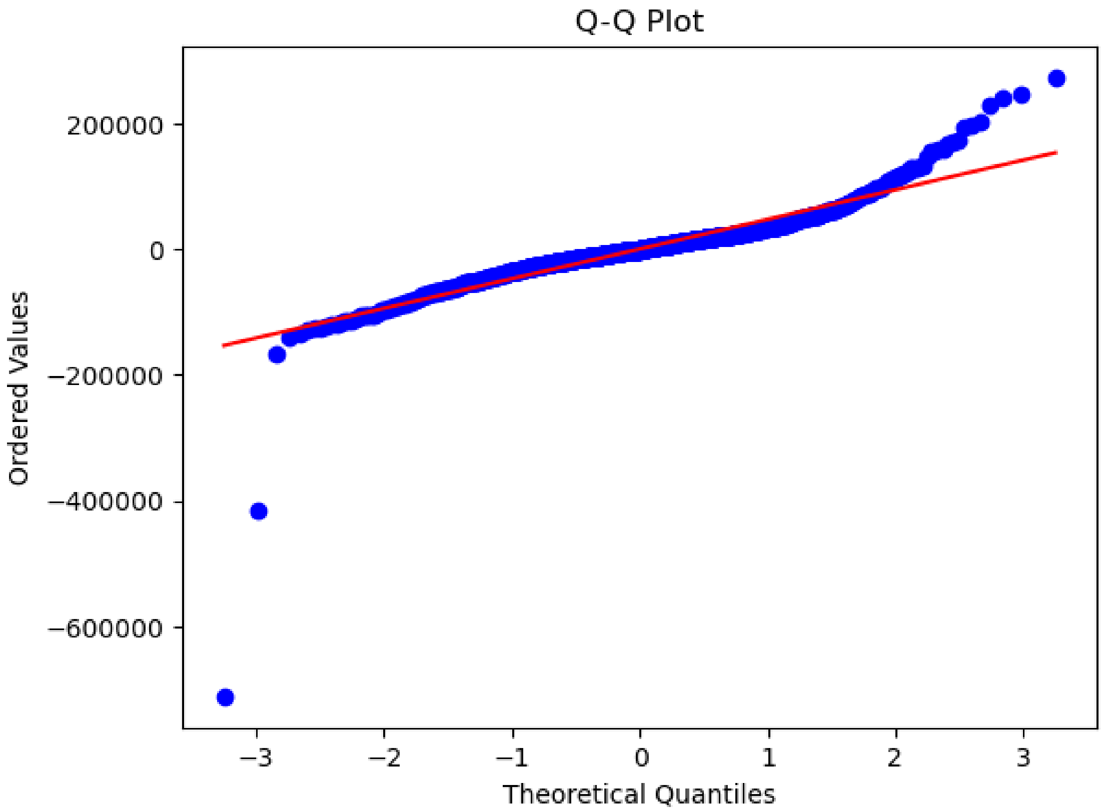
7 Q6

Evaluate the models' assumptions.

[27]:
```
#Linearity Plot
predicted = model.predict(train_x)
plt.scatter(train_y, predicted)
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title('Actual vs. Predicted Values')
plt.show()
#underfitted plot due to the simplicity of the model.
```



[28]:
```
# Generate the Q-Q plot
predictions = model.predict(train_x)

residuals = train_y - predictions


stats.probplot(residuals, dist="norm", plot=plt)
plt.title('Q-Q Plot')
plt.xlabel('Theoretical Quantiles')
plt.ylabel('Ordered Values')
plt.show()
```



[29]:
```
# Independence (Durbin-Watson Test)

print('Durbin-Watson:', sm.stats.durbin_watson(residuals))
##A value of 2.0 suggests no autocorrelation.
##The residuals are uncorrelated, and the assumption of independence is met.
```

```
Durbin-Watson: 1.9973818636096454
```

8 Q7

Evaluate goodness of fit metrics on the training and validation sets.

[30]:
```
print("Segment 1 (OverallCond < 5):")
print("Intercept:", model1.intercept_)
print("Coefficient:", model1.coef_[0])
print("R-squared:", model1.score(X1, y1))
```

```
Segment 1 (OverallCond < 5):
Intercept: 92320.91958887543
Coefficient: 6560.245465538092
R-squared: 0.007174299232783898
```

[31]:
```python
print("Segment 2 (OverallCond >= 5):")
print("Intercept:", model2.intercept_)
print("Coefficient:", model2.coef_[0])
print("R-squared:", model2.score(X2, y2))
```

```
Segment 2 (OverallCond >= 5):
Intercept: 277483.7162944825
Coefficient: -16195.61343675314
R-squared: 0.04252833129614908
```

[96]:
```python
#The model with the higher R^2 signifies a better fit.
```

[32]:
```python
train_x
```

[32]:

	Id	TotalBsmtSF	1stFlrSF	2ndFlrSF	LotFrontage	LotArea
0	1	856	856	854	65.0	8450
1	2	1262	1262	0	80.0	9600
2	3	920	920	866	68.0	11250
3	4	756	961	756	60.0	9550
4	5	1145	1145	1053	84.0	14260
…	…	…	…	…	…	…
1455	1456	953	953	694	62.0	7917
1456	1457	1542	2073	0	85.0	13175
1457	1458	1152	1188	1152	66.0	9042
1458	1459	1078	1078	0	68.0	9717
1459	1460	1256	1256	0	75.0	9937

[1201 rows x 6 columns]

[33]:
```python
train_predictions = model.predict(train_x)
train_predictions
```

[33]:
```
array([187594.48494719, 177219.04149697, 199285.98848029, …,
       253094.31001597, 141597.3369366 , 168437.82360463])
```

[34]:
```python
# goodness of fit metrics on the training data

train_mse = mean_squared_error(train_y, train_predictions)
train_rmse = np.sqrt(train_mse)
train_mae = mean_absolute_error(train_y, train_predictions)
train_r2 = r2_score(train_y, train_predictions)

print("Training Set Metrics:")
print(f"MSE: {train_mse}")
print(f"RMSE: {train_rmse}")
print(f"MAE: {train_mae}")
print(f"R-squared: {train_r2}")
```

```
Training Set Metrics:
```

```
MSE: 2682183412.527564
RMSE: 51789.80027503064
MAE: 32910.846098982875
R-squared: 0.6139644620334612
```

[51]:

```python
# Generate synthetic data for validation sets

np.random.seed(0)
num_samples = 100
square_footage = np.random.randint(1000, 3000, num_samples)

# Square footage in sqft
lot_size = np.random.randint(2000, 10000, num_samples)

# Lot size in sqft
house_prices = 100 * square_footage + 50 * lot_size + np.random.
 ↪randn(num_samples) * 10000

# True relationship: price = 100 * sqft + 50 * lot_size + noise

# Combine features into X and target variable into y
X = np.column_stack((square_footage, lot_size))
y = house_prices

# Split data into training and validation sets
X_train,X_val, y_train, y_val = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=0)

# Train a linear regression model
model_1 = LinearRegression()
model_1.fit(X_train, y_train)

# Predict on validation sets

y_val_pred = model_1.predict(X_val)

# Evaluate goodness of fit using Mean Squared Error (MSE) for validation sets

mse_val = mean_squared_error(y_val, y_val_pred)
val_rmse = np.sqrt(mse_val)
val_r2 = r2_score(y_val, y_val_pred)

print(f"Mean Squared Error (MSE) on Validation sets: {mse_val:.2f}")
print(f"RMSE on Validation sets: {val_rmse}")
print(f"MAE on Validation sets: {mse_val}")
print(f"R-squared on Validation sets: {val_r2}")
```

```
Mean Squared Error (MSE) on Validation sets: 77949865.43
```

```
RMSE on Validation sets: 8828.92209882864
MAE on Validation sets: 77949865.42718472
R-squared on Validation sets: 0.9937832134596482
```

9 8

Submit predictions for the unseen test set available on Kaggle.com.

[38]:
```python
train_x = train_p.drop('SalePrice', axis=1)

# Fit the model
model.fit(train_x, train_y)
```

[38]:
```
LinearRegression()
```

[39]:
```python
test_predictions = model.predict(test_p)
test_predictions
```

[39]:
```
array([112449.90487929, 180855.73981829, 178707.40365498, …,
       151939.52068175, 113579.40199189, 204608.54950575])
```

[41]:
```python
test_p["SalePrice"] = test_predictions.round(2)
Submission = test[['Id','TotalBsmtSF','1stFlrSF','2ndFlrSF','LotFrontage',␣
 ↪'LotArea']]
Submission['TotalSF'] = Submission['TotalBsmtSF'] + Submission['1stFlrSF'] +␣
 ↪Submission['2ndFlrSF']
Submission['TotalLotSize'] = Submission['LotFrontage'] + Submission['LotArea']
Submission
```

```
C:\Users\Aaron\AppData\Local\Temp\ipykernel_35704\1604618483.py:3:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  Submission['TotalSF'] = Submission['TotalBsmtSF'] + Submission['1stFlrSF'] +
Submission['2ndFlrSF']
C:\Users\Aaron\AppData\Local\Temp\ipykernel_35704\1604618483.py:4:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  Submission['TotalLotSize'] = Submission['LotFrontage'] + Submission['LotArea']
```

[41]:
```
        Id  TotalBsmtSF  1stFlrSF  2ndFlrSF  LotFrontage  LotArea  TotalSF  \
0     1461        882.0       896         0         80.0    11622   1778.0
1     1462       1329.0      1329         0         81.0    14267   2658.0
2     1463        928.0       928       701         74.0    13830   2557.0
3     1464        926.0       926       678         78.0     9978   2530.0
4     1465       1280.0      1280         0         43.0     5005   2560.0
…      …            …        …        …            …       …        …
1454  2915        546.0       546       546         21.0     1936   1638.0
1455  2916        546.0       546       546         21.0     1894   1638.0
1456  2917       1224.0      1224         0        160.0    20000   2448.0
1457  2918        912.0       970         0         62.0    10441   1882.0
1458  2919        996.0       996      1004         74.0     9627   2996.0

      TotalLotSize
0          11702.0
1          14348.0
2          13904.0
3          10056.0
4           5048.0
…              …
1454        1957.0
1455        1915.0
1456       20160.0
1457       10503.0
1458        9701.0

[1459 rows x 8 columns]
```

[42]:
```
Submission.isnull().sum()
```

[42]:
```
Id                0
TotalBsmtSF       1
1stFlrSF          0
2ndFlrSF          0
LotFrontage     227
LotArea           0
TotalSF           1
TotalLotSize    227
dtype: int64
```

[43]:
```
Submission.fillna(0, inplace=True)
```

```
C:\Users\Aaron\AppData\Local\Temp\ipykernel_35704\43894959.py:1:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

```
Submission.fillna(0, inplace=True)
```

[44]:
```
Submission.isnull().sum()
```

[44]:
```
Id              0
TotalBsmtSF     0
1stFlrSF        0
2ndFlrSF        0
LotFrontage     0
LotArea         0
TotalSF         0
TotalLotSize    0
dtype: int64
```

[45]:
```
test_predictions = model.predict(Submission)
test_predictions
```

[45]:
```
array([112449.90487929, 180855.73981829, 178707.40365498, …,
       151939.52068175, 113579.40199189, 204608.54950575])
```

[46]:
```
Submission
```

[46]:
```
        Id  TotalBsmtSF  1stFlrSF  2ndFlrSF  LotFrontage  LotArea  TotalSF  \
0     1461        882.0       896         0         80.0    11622   1778.0
1     1462       1329.0      1329         0         81.0    14267   2658.0
2     1463        928.0       928       701         74.0    13830   2557.0
3     1464        926.0       926       678         78.0     9978   2530.0
4     1465       1280.0      1280         0         43.0     5005   2560.0
…     …             …        …         …            …        …
1454  2915        546.0       546       546         21.0     1936   1638.0
1455  2916        546.0       546       546         21.0     1894   1638.0
1456  2917       1224.0      1224         0        160.0    20000   2448.0
1457  2918        912.0       970         0         62.0    10441   1882.0
1458  2919        996.0       996      1004         74.0     9627   2996.0

      TotalLotSize
0          11702.0
1          14348.0
2          13904.0
3          10056.0
4           5048.0
…             …
1454        1957.0
1455        1915.0
1456       20160.0
1457       10503.0
1458        9701.0
```

```
[1459 rows x 8 columns]
```

[48]:
```
Submission['SalePrice'] = test_predictions.round(2)
Submission
```

```
C:\Users\Aaron\AppData\Local\Temp\ipykernel_35704\275483597.py:1:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  Submission['SalePrice'] = test_predictions.round(2)
```

[48]:
```
        Id  TotalBsmtSF  1stFlrSF  2ndFlrSF  LotFrontage  LotArea  TotalSF  \
0     1461        882.0       896         0         80.0    11622   1778.0
1     1462       1329.0      1329         0         81.0    14267   2658.0
2     1463        928.0       928       701         74.0    13830   2557.0
3     1464        926.0       926       678         78.0     9978   2530.0
4     1465       1280.0      1280         0         43.0     5005   2560.0
…     …            …        …         …            …         …       …
1454  2915        546.0       546       546         21.0     1936   1638.0
1455  2916        546.0       546       546         21.0     1894   1638.0
1456  2917       1224.0      1224         0        160.0    20000   2448.0
1457  2918        912.0       970         0         62.0    10441   1882.0
1458  2919        996.0       996      1004         74.0     9627   2996.0

      TotalLotSize  SalePrice
0          11702.0  112449.90
1          14348.0  180855.74
2          13904.0  178707.40
3          10056.0  174590.07
4           5048.0  172836.84
…             …         …
1454        1957.0   98323.64
1455        1915.0   98301.19
1456       20160.0  151939.52
1457       10503.0  113579.40
1458        9701.0  204608.55

[1459 rows x 9 columns]
```

[50]:
```
test_data = Submission.
↪drop(['TotalBsmtSF','1stFlrSF','2ndFlrSF','LotFrontage','LotArea','TotalSF','TotalLotSize'],
↪axis=1)
test_data
```

[50]:
```
        Id  SalePrice
0     1461  112449.90
1     1462  180855.74
2     1463  178707.40
3     1464  174590.07
4     1465  172836.84
…     …           …
1454  2915   98323.64
1455  2916   98301.19
1456  2917  151939.52
1457  2918  113579.40
1458  2919  204608.55

[1459 rows x 2 columns]
```

[113]:
```
test_data.to_csv('unseen_test_data.csv', index=False)
```