PREDICTING HOUSE PRICES USING MLR AND KNN REGRESSION

Rajkumar Ramu

Rustam Adigozalov

Hoang Phuong Duy Nguyen

Natarajan Govindarajan

Henil Shah

IMPORTING THE REQUIRED LIBRARIES

In [1]:
```
from pathlib import Path
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import matplotlib.pylab as plt
from dmba import regressionSummary, classificationSummary
from dmba import backward_elimination, AIC_score, stepwise_selection
```

READ THE DATA

In [2]:
```
HousingData = pd.read_csv('kc_house_data.csv')
HousingData.info()
HousingData.describe()   # for numeric columns
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   id             21613 non-null  int64  
 1   date           21613 non-null  object 
 2   price          21613 non-null  float64
 3   bedrooms       21613 non-null  int64  
 4   bathrooms      21613 non-null  float64
 5   sqft_living    21613 non-null  int64  
 6   sqft_lot       21613 non-null  int64  
 7   floors         21613 non-null  float64
 8   waterfront     21613 non-null  int64  
 9   view           21613 non-null  int64  
 10  condition      21613 non-null  int64  
 11  grade          21613 non-null  int64  
 12  sqft_above     21611 non-null  float64
 13  sqft_basement  21613 non-null  int64  
 14  yr_built       21613 non-null  int64  
 15  yr_renovated   21613 non-null  int64  
 16  zipcode        21613 non-null  int64  
 17  lat            21613 non-null  float64
 18  long           21613 non-null  float64
 19  sqft_living15  21613 non-null  int64  
 20  sqft_lot15     21613 non-null  int64  
dtypes: float64(6), int64(14), object(1)
memory usage: 3.5+ MB
```

Out[2]:

	id	price	bedrooms	bathrooms	sqft_living	sqft_lo
count	2.161300e+04	2.161300e+04	21613.000000	21613.000000	21613.000000	2.161300e+0
mean	4.580302e+09	5.400881e+05	3.370842	2.114757	2079.899736	1.510697e+0
std	2.876566e+09	3.671272e+05	0.930062	0.770163	918.440897	4.142051e+0
min	1.000102e+06	7.500000e+04	0.000000	0.000000	290.000000	5.200000e+0
25%	2.123049e+09	3.219500e+05	3.000000	1.750000	1427.000000	5.040000e+0
50%	3.904930e+09	4.500000e+05	3.000000	2.250000	1910.000000	7.618000e+0
75%	7.308900e+09	6.450000e+05	4.000000	2.500000	2550.000000	1.068800e+0
max	9.900000e+09	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+0

In [3]:
```
HousingData.head(10)
```

Out[3]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	fl
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650	
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000	
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000	
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080	
5	7237550310	20140512T000000	1225000.0	4	4.50	5420	101930	
6	1321400060	20140627T000000	257500.0	3	2.25	1715	6819	
7	2008000270	20150115T000000	291850.0	3	1.50	1060	9711	
8	2414600126	20150415T000000	229500.0	3	1.00	1780	7470	
9	3793500160	20150312T000000	323000.0	3	2.50	1890	6560	

10 rows × 21 columns

PREPARING THE DATA

In [4]:
```
# Convert date to datetime
HousingData['date'] = pd.to_datetime(HousingData['date'])

# Convert zipcode and categorical/ordinal columns to category
HousingData['zipcode'] = HousingData['zipcode'].astype('category')
for col in ['waterfront', 'view', 'condition', 'grade']:
    HousingData[col] = HousingData[col].astype('category')

# Check updated datatypes
print(HousingData.dtypes)
```

```
id                        int64
date             datetime64[ns]
price                   float64
bedrooms                  int64
bathrooms               float64
sqft_living               int64
sqft_lot                  int64
floors                  float64
waterfront             category
view                   category
condition              category
grade                  category
sqft_above              float64
sqft_basement             int64
yr_built                  int64
yr_renovated              int64
zipcode                category
lat                     float64
long                    float64
sqft_living15             int64
sqft_lot15                int64
dtype: object
```

SELECT THE IMPORTANT VARIAVLES TO PREDICT PRICE OF THE HOUSES

In [5]:
```
# Select the most relevant variables for predicting house price
selected_vars = [
    'bedrooms',
    'bathrooms',
    'sqft_living',
    'sqft_lot',
    'floors',
    'waterfront',
    'view',
    'condition',
    'grade',
    'sqft_above',
    'sqft_basement',
    'yr_built',
    'yr_renovated',
    'zipcode',
    'lat',
    'long',
    'sqft_living15',
    'sqft_lot15'
]

# see how many missing values each column has
HousingData[selected_vars].isnull().sum()
```

Out[5]:
```
bedrooms         0
bathrooms        0
sqft_living      0
sqft_lot         0
floors           0
waterfront       0
view             0
condition        0
grade            0
sqft_above       2
sqft_basement    0
yr_built         0
yr_renovated     0
zipcode          0
lat              0
long             0
sqft_living15    0
sqft_lot15       0
dtype: int64
```

CLEANING AND SPLITTING THE DATA

In [6]:
```
# Remove rows with null values in column 'sqft_above'
HousingData = HousingData.dropna(subset=['sqft_above'])

# Possible predictor variables
X = HousingData[selected_vars]

# Target variable (the thing we want to predict)
y = HousingData['price']
# splitting data for testing and training
train_X, valid_X, train_y, valid_y = train_test_split(
    X, y, test_size=0.2, random_state=1
)
```

PERFORM VARIABLE SELECTION METHODS TO SELECT THE BEST 6 PREDICTORS FOR PREDICTING THE PRICE

In [7]:
```
#Backward elimination
def train_model(variables):
    model = LinearRegression()
    model.fit(train_X[list(variables)], train_y)
    return model
def score_model(model, variables):
    return AIC_score(train_y, model.predict(train_X[variables]), model)
allVariables = train_X.columns
best_model, best_variables = backward_elimination(allVariables, train_model,
score_model, verbose=True)
print(best_variables)
regressionSummary(valid_y, best_model.predict(valid_X[best_variables]))
```

```
Variables: bedrooms, bathrooms, sqft_living, sqft_lot, floors, waterfront, view, con
dition, grade, sqft_above, sqft_basement, yr_built, yr_renovated, zipcode, lat, lon
g, sqft_living15, sqft_lot15
Start: score=471391.52
Step: score=471389.52, remove sqft_living
Step: score=471389.52, remove None
['bedrooms', 'bathrooms', 'sqft_lot', 'floors', 'waterfront', 'view', 'condition',
'grade', 'sqft_above', 'sqft_basement', 'yr_built', 'yr_renovated', 'zipcode', 'la
t', 'long', 'sqft_living15', 'sqft_lot15']

Regression statistics

                    Mean Error (ME) : 4077.1651
     Root Mean Squared Error (RMSE) : 200155.2491
          Mean Absolute Error (MAE) : 125626.8727
        Mean Percentage Error (MPE) : -3.0831
Mean Absolute Percentage Error (MAPE) : 25.7103
```

In [8]:

```
# Keep top 10 most frequent zip codes, combine the rest as 'Other'
top_zips = train_X['zipcode'].value_counts().nlargest(10).index
train_X['zipcode_grouped'] = train_X['zipcode'].apply(lambda x: x if x in top_zips
valid_X['zipcode_grouped'] = valid_X['zipcode'].apply(lambda x: x if x in top_zips

# Forward stepwise selection
# 17 predictors from backward elimination
b_predictors = ['bedrooms', 'bathrooms', 'sqft_lot', 'floors', 'view', 'condition',
# Only low-cardinality categorical variable like grouped zipcode and waterfront
categorical_vars = ['zipcode_grouped', 'waterfront']

# Create dummy variables for the categorical predictors
train_selected_X = pd.get_dummies(train_X[b_predictors + categorical_vars], columns
valid_selected_X = pd.get_dummies(valid_X[b_predictors + categorical_vars], columns

# Align columns so validation set matches train set
valid_selected_X = valid_selected_X.reindex(columns=train_selected_X.columns, fill_

def train_model(variables):
    model = LinearRegression()
    # If no variables, fit a model with just the mean
    if len(variables) == 0:
        # Create an array of zeros to allow model.fit
        model.fit([[0]]*len(train_y), train_y)
    else:
        model.fit(train_selected_X[variables], train_y)
    return model

def score_model(model, variables):
    # If no variables, predict mean
    if len(variables) == 0:
        y_pred = [train_y.mean()]*len(train_y)
    else:
        y_pred = model.predict(train_selected_X[variables])
    return AIC_score(train_y, y_pred, model)

best_model, best_variables = stepwise_selection(train_selected_X.columns, train_mod
print("best_predictors:", best_variables)
```

```
Variables: bedrooms, bathrooms, sqft_lot, floors, view, condition, grade, sqft_abov
e, sqft_basement, yr_built, yr_renovated, lat, long, sqft_living15, sqft_lot15, zipc
ode_grouped_98023, zipcode_grouped_98034, zipcode_grouped_98038, zipcode_grouped_980
42, zipcode_grouped_98052, zipcode_grouped_98103, zipcode_grouped_98115, zipcode_gro
uped_98117, zipcode_grouped_98118, zipcode_grouped_Other, waterfront_1
Start: score=491885.20, constant
Step: score=481704.20, add grade
Step: score=479293.65, add yr_built
Step: score=477748.56, add bathrooms
Step: score=476445.71, add waterfront_1
Step: score=474897.32, add lat
Step: score=473800.00, add sqft_above
Step: score=472570.38, add sqft_basement
Step: score=472104.71, add view
Step: score=471877.47, add bedrooms
Step: score=471772.65, add condition
Step: score=471706.20, add long
Step: score=471641.69, add zipcode_grouped_98023
Step: score=471598.32, add sqft_living15
Step: score=471562.67, add zipcode_grouped_98034
Step: score=471533.13, add yr_renovated
Step: score=471521.30, add zipcode_grouped_98115
Step: score=471510.92, add sqft_lot15
Step: score=471501.93, add zipcode_grouped_98042
Step: score=471492.91, add zipcode_grouped_98103
Step: score=471484.85, add sqft_lot
Step: score=471482.34, add zipcode_grouped_98118
Step: score=471477.08, add zipcode_grouped_Other
Step: score=471466.12, add zipcode_grouped_98052
Step: score=471462.04, add zipcode_grouped_98038
Step: score=471458.49, add zipcode_grouped_98117
Step: score=471457.97, add floors
Step: score=471457.97, unchanged None
best_predictors: ['grade', 'yr_built', 'bathrooms', 'waterfront_1', 'lat', 'sqft_abo
ve', 'sqft_basement', 'view', 'bedrooms', 'condition', 'long', 'zipcode_grouped_9802
3', 'sqft_living15', 'zipcode_grouped_98034', 'yr_renovated', 'zipcode_grouped_9811
5', 'sqft_lot15', 'zipcode_grouped_98042', 'zipcode_grouped_98103', 'sqft_lot', 'zip
code_grouped_98118', 'zipcode_grouped_Other', 'zipcode_grouped_98052', 'zipcode_grou
ped_98038', 'zipcode_grouped_98117', 'floors']
```

NORMALIZE THE PREDICTOR DATA

In [9]:
```
predictors = ['grade', 'yr_built', 'bathrooms', 'waterfront', 'lat', 'sqft_above']
train_X = train_X[predictors].copy()
valid_X = valid_X[predictors].copy()

scaler = StandardScaler()

# Fit only on training data
train_X = pd.DataFrame(scaler.fit_transform(train_X),
                       columns=train_X.columns,
                       index=train_X.index)

# Transform validation data
valid_X = pd.DataFrame(scaler.transform(valid_X),
```

```
                                columns=valid_X.columns,
                                index=valid_X.index)
```

In [10]:
```
# Use only the first 1000 records and select columns for regression analysis
HousingData = HousingData.iloc[0:1000]
outcome = 'price'
# Partition data into predictors (x) and output (y)
X = pd.get_dummies(HousingData[predictors], drop_first=True)
y = HousingData[outcome]
X.head(9)
```

Out[10]:

	yr_built	bathrooms	lat	sqft_above	grade_3	grade_4	grade_5	grade_6	grade_7
0	1955	1.00	47.5112	1180.0	False	False	False	False	True
1	1951	2.25	47.7210	2170.0	False	False	False	False	True
2	1933	1.00	47.7379	770.0	False	False	False	True	False
3	1965	3.00	47.5208	1050.0	False	False	False	False	True
4	1987	2.00	47.6168	1680.0	False	False	False	False	False
5	2001	4.50	47.6561	3890.0	False	False	False	False	False
6	1995	2.25	47.3097	1715.0	False	False	False	False	True
7	1963	1.50	47.4095	1060.0	False	False	False	False	True
8	1960	1.00	47.5123	1050.0	False	False	False	False	True

In [11]:
```
y.head(9)
```

Out[11]:
```
0     221900.0
1     538000.0
2     180000.0
3     604000.0
4     510000.0
5    1225000.0
6     257500.0
7     291850.0
8     229500.0
Name: price, dtype: float64
```

In [12]:
```
# Built the Linear Model based on the training data
HD_lm = LinearRegression()
HD_lm.fit(train_X, train_y)
# print coefficients
print(pd.DataFrame({'Predictor': train_X.columns, 'coefficient': HD_lm.coef_}))
```

```
    Predictor    coefficient
0       grade  164582.523973
1    yr_built -114464.411501
2   bathrooms   70755.656515
3  waterfront   68123.721630
4         lat   76484.611213
5  sqft_above   92012.223780
```

PERFORMING MLR AND PRINTING METRICS

In [13]:
```
# Built the Linear Model based on the training data
reg = LinearRegression()
reg.fit(train_X, train_y)
# Evaluate Perfomance
# Training
print("Training Data:")
regressionSummary(train_y, reg.predict(train_X))
r2 = r2_score(train_y, reg.predict(train_X))
print(f"{'R-squared (R²)':>35}   : {r2:.4f}")
```

```
Training Data:

Regression statistics

                      Mean Error (ME) : -0.0000
       Root Mean Squared Error (RMSE) : 216153.7804
            Mean Absolute Error (MAE) : 134219.5827
          Mean Percentage Error (MPE) : -5.2376
Mean Absolute Percentage Error (MAPE) : 27.0648
                    R-squared (R²)   : 0.6489
```

In [14]:
```
# Evaluate Perfomace
# Validation
print("\nValidation Data:")
regressionSummary(valid_y, reg.predict(valid_X))
r2 = r2_score(valid_y, reg.predict(valid_X))
print(f"{'R-squared (R²)':>35}   : {r2:.4f}")
```

```
Validation Data:

Regression statistics

                      Mean Error (ME) : 4679.7174
       Root Mean Squared Error (RMSE) : 216467.8979
            Mean Absolute Error (MAE) : 133910.7573
          Mean Percentage Error (MPE) : -4.1604
Mean Absolute Percentage Error (MAPE) : 27.3150
                    R-squared (R²)   : 0.6691
```

RESIDUAL PLOT(SIDE BY SIDE) HISTOGRAM

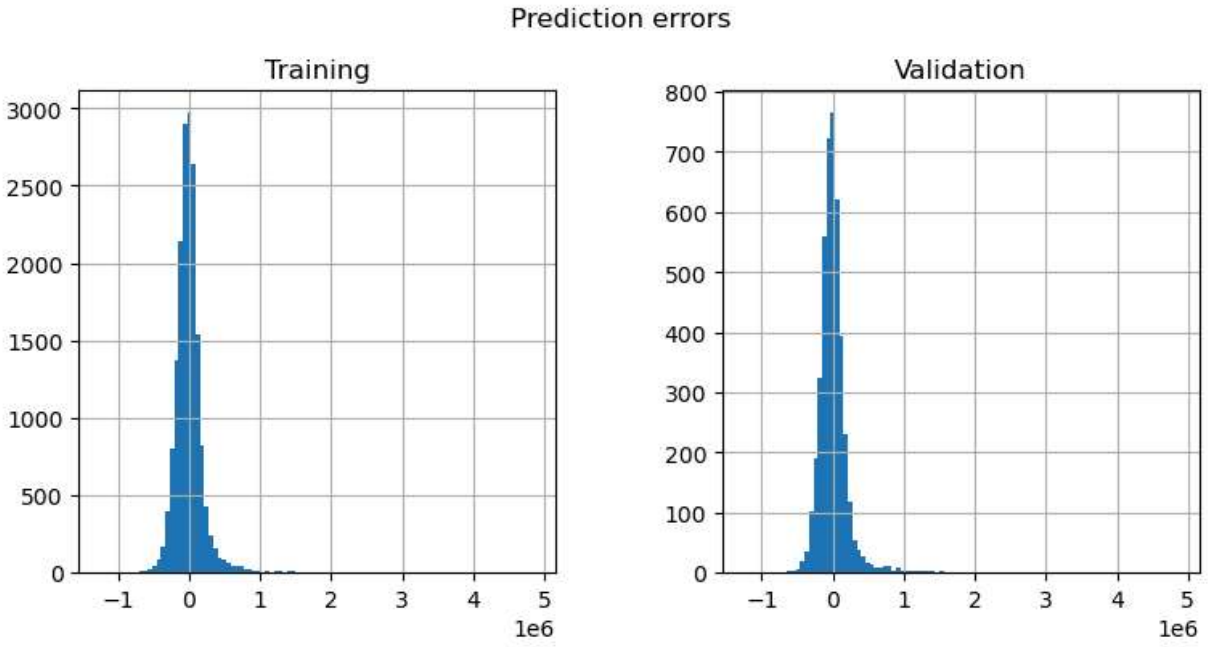
In [15]:
```
# training
train_e = train_y - reg.predict(train_X)

# validation
valid_e = valid_y - reg.predict(valid_X)
```

```
pred_error_train = pd.DataFrame({
'residual': train_e,
'dat set' :'training'
})
pred_error_valid = pd.DataFrame({
'residual': valid_e,
'dat set' :'validation'
})
boxdata_df = pd.concat([pred_error_train, pred_error_valid], ignore_index=True)
# Compute min/max across both train and valid errors
min_val = min(train_e.min(), valid_e.min())
max_val = max(train_e.max(), valid_e.max())

# Use updated range
common = {'bins': 100, 'range': [min_val, max_val]}

fig, axes = plt.subplots(nrows=1, ncols=2)
fig.set_size_inches(9, 4)
pred_error_train.hist(ax=axes[0], **common)
pred_error_valid.hist(ax=axes[1], **common)
axes[0].set_title('Training')
axes[1].set_title('Validation')
plt.suptitle('Prediction errors')
plt.subplots_adjust(bottom=0.1, top=0.85, wspace=0.35)
plt.show()
```



SCATTER PLOT: PREDICTED VS ACTUAL PRICES

In [16]:
```
import matplotlib.pyplot as plt
import numpy as np

# Select a random subset of 100 samples
subset_size = 100
train_idx = np.random.choice(len(train_X), size=subset_size, replace=False)
```

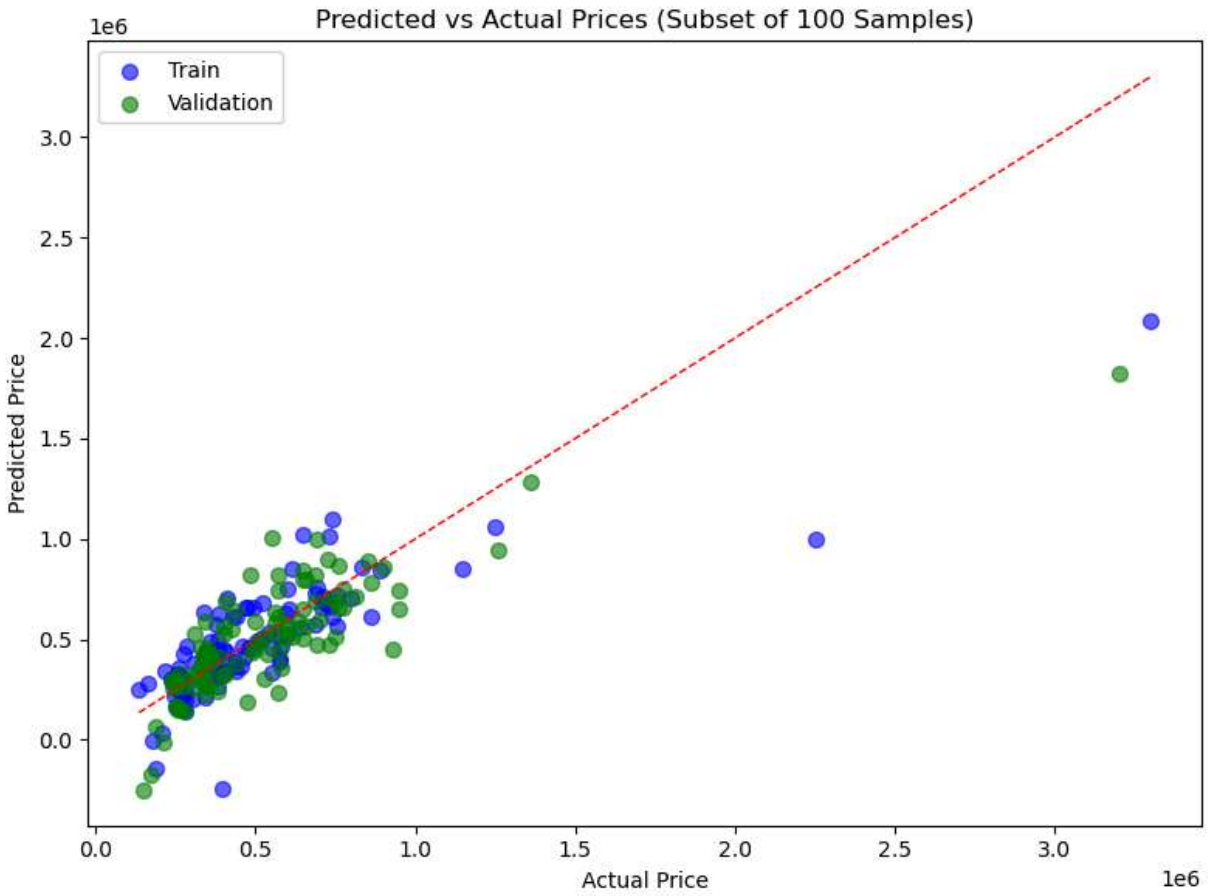
```
valid_idx = np.random.choice(len(valid_X), size=subset_size, replace=False)

# Get predictions for the subsets
train_pred_subset = reg.predict(train_X.iloc[train_idx])
valid_pred_subset = reg.predict(valid_X.iloc[valid_idx])
train_actual_subset = train_y.iloc[train_idx]
valid_actual_subset = valid_y.iloc[valid_idx]

# Plot scatter
plt.figure(figsize=(8, 6))
plt.scatter(train_actual_subset, train_pred_subset, label='Train', alpha=0.6, color
plt.scatter(valid_actual_subset, valid_pred_subset, label='Validation', alpha=0.6,

# Diagonal line (perfect predictions)
min_val = min(train_actual_subset.min(), valid_actual_subset.min())
max_val = max(train_actual_subset.max(), valid_actual_subset.max())
plt.plot([min_val, max_val], [min_val, max_val], color='red', linestyle='--', linew

plt.xlabel('Actual Price')
plt.ylabel('Predicted Price')
plt.title('Predicted vs Actual Prices (Subset of 100 Samples)')
plt.legend()
plt.tight_layout()
plt.show()
```



COMPARING ACTUAL AND PREDICTED PRICES TAKING 2 RANDOM SAMPLE USING MLR MODEL

In [17]:
```
# Take 2 new random samples each run
sample_houses = HousingData.sample(n=2, random_state=None)  # random_state=None for

# Select only the predictors
X_sample = sample_houses[predictors].copy()

# Scale using the same scaler as training
X_sample_scaled = pd.DataFrame(scaler.transform(X_sample),
                               columns=X_sample.columns,
                               index=X_sample.index)

# Predict prices
predicted_prices = HD_lm.predict(X_sample_scaled)

# Print actual vs predicted
print(f"{'Actual':>12}  {'Predicted':>12}")
for actual, pred in zip(sample_houses['price'], predicted_prices):
    print(f"{actual:12,.0f}  {pred:12,.0f}")
```

```
      Actual     Predicted
    221,900       307,039
    700,000       615,154
```

TRAINING KNN FOR K=5 AND K =10

In [18]:
```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# predictors already selected
predictors = ['grade', 'yr_built', 'bathrooms', 'waterfront', 'lat', 'sqft_above']

# Train KNN for K=5
knn5 = KNeighborsRegressor(n_neighbors=5)
knn5.fit(train_X, train_y)

# Train KNN for K=10
knn10 = KNeighborsRegressor(n_neighbors=10)
knn10.fit(train_X, train_y)
```

Out[18]:

▾ KNeighborsRegressor

▶ Parameters

PRINTING METRICS AND CHOOSING THE BEST K

In [19]:
```
def regression_stats(y_true, y_pred):
    ME = np.mean(y_true - y_pred)
    RMSE = np.sqrt(mean_squared_error(y_true, y_pred))
    MAE = mean_absolute_error(y_true, y_pred)
    MPE = np.mean((y_true - y_pred) / y_true) * 100
    MAPE = np.mean(np.abs((y_true - y_pred) / y_true)) * 100
    R2 = r2_score(y_true, y_pred)
    return ME, RMSE, MAE, MPE, MAPE, R2
```

```
# Predictions
train_pred5 = knn5.predict(train_X)
valid_pred5 = knn5.predict(valid_X)

train_pred10 = knn10.predict(train_X)
valid_pred10 = knn10.predict(valid_X)

# Stats
stats_train5 = regression_stats(train_y, train_pred5)
stats_valid5 = regression_stats(valid_y, valid_pred5)

stats_train10 = regression_stats(train_y, train_pred10)
stats_valid10 = regression_stats(valid_y, valid_pred10)

# Print metrics
def print_metrics(name, stats):
    labels = ['ME', 'RMSE', 'MAE', 'MPE (%)', 'MAPE (%)', 'R-squared']
    print(f"\n{name}")
    for label, value in zip(labels, stats):
        print(f"{label:15}: {value:12,.4f}")

# Show metrics for K=5
print_metrics("KNN (K=5) - Training", stats_train5)
print_metrics("KNN (K=5) - Validation", stats_valid5)

# Show metrics for K=10
print_metrics("KNN (K=10) - Training", stats_train10)
print_metrics("KNN (K=10) - Validation", stats_valid10)

# Pick the best K
best_knn = knn5 if stats_valid5[1] < stats_valid10[1] else knn10
print("Best K:", best_knn.n_neighbors)
```

```
KNN (K=5) - Training
ME              :   1,796.4900
RMSE            : 146,925.9017
MAE             :  80,692.9222
MPE (%)         :      -3.6034
MAPE (%)        :      14.7745
R-squared       :       0.8378

KNN (K=5) - Validation
ME              :   4,771.0709
RMSE            : 176,460.6028
MAE             :  96,738.6186
MPE (%)         :      -4.3671
MAPE (%)        :      17.6448
R-squared       :       0.7801

KNN (K=10) - Training
ME              :   2,663.0940
RMSE            : 158,630.3342
MAE             :  87,229.1409
MPE (%)         :      -4.2302
MAPE (%)        :      15.9660
R-squared       :       0.8109

KNN (K=10) - Validation
ME              :   4,847.3248
RMSE            : 172,744.0395
MAE             :  93,987.7225
MPE (%)         :      -4.5134
MAPE (%)        :      17.0916
R-squared       :       0.7893
Best K: 10
```

COMPARING ACTUAL AND PREDICTED PRICES TAKING 2 RANDOM SAMPLE USING KNN MODEL

In [20]:
```python
# Random 2-house sample
sample_houses = HousingData.sample(n=2, random_state=None)
X_sample = sample_houses[predictors].copy()

# Scale using same scaler
X_sample_scaled = pd.DataFrame(scaler.transform(X_sample),
                               columns=X_sample.columns,
                               index=X_sample.index)

# Predict using best KNN
predicted_prices_knn = best_knn.predict(X_sample_scaled)

print(f"{'Actual':>12}  {'KNN Predicted':>15}")
for actual, pred in zip(sample_houses['price'], predicted_prices_knn):
    print(f"{actual:12,.0f}  {pred:15,.0f}")
```

```
      Actual    KNN Predicted
  2,250,000        1,830,300
    268,000          263,050
```

COMPARE THE MLR AND KNN MODEL

In [21]:
```
# Predict using your MLR model
predicted_prices_mlr = HD_lm.predict(X_sample_scaled)

print(f"{'Actual':>12}  {'MLR Predicted':>15}  {'KNN Predicted':>15}")
for actual, pred_mlr, pred_knn in zip(sample_houses['price'], predicted_prices_mlr,
    print(f"{actual:12,.0f}  {pred_mlr:15,.0f}  {pred_knn:15,.0f}")
```

```
      Actual    MLR Predicted    KNN Predicted
   2,250,000        1,302,867        1,830,300
     268,000          215,177          263,050
```

In [ ]:
```

```