# ACT_3

October 9, 2023

# 1 ACT 3

#### 1.0.1 Team Number: 7

#### 1.0.2 Team Members: Jacob Silva

#### 1.0.3 Juliana Steele

#### 1.0.4 Joel Hurtado

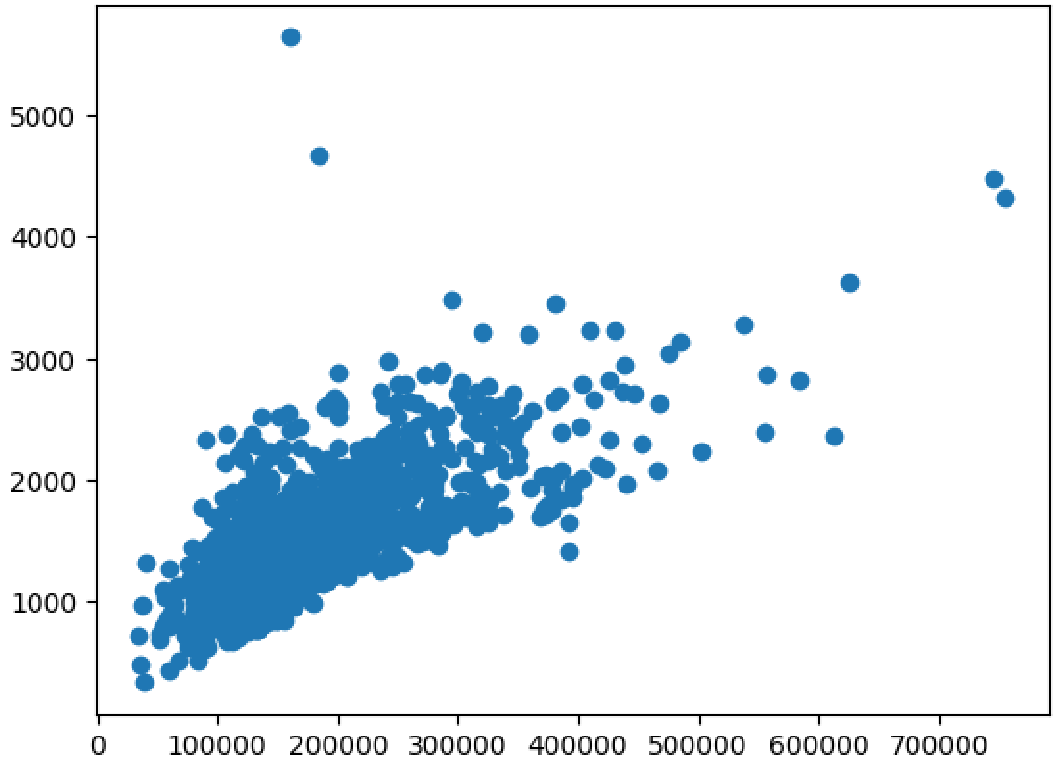
#### 1.0.5 Read the data into your software system

[ ]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm
from sklearn.linear_model import Lasso
import seaborn as sns

# the file is saved in the main folder. You should be able to run by adding the
↪file to the envrionment
data = pd.read_excel('ACT_03_Data.xlsx')

data
```

[ ]:
```
        Id MSZoning FireplaceQu  GAR  SalePrice  BATH  Age   TSF
0      614       RL         NaN    0     147000   1.0    0  1120
1     1454       RL         NaN    0      84500   1.0    0  1140
2      429       RL         NaN  628     195400   2.0    0  1208
3      508       FV         NaN  676     208300   2.0    0  1218
4     1022       RL         NaN  632     194000   2.0    0  1220
…      …        …           …   …        …    …   …    …
1455  1133       RM         NaN  205     117500   2.0  127  2210
1456   305       RM          Ex  870     295000   3.0  128  3493
1457   748       RM          Gd  864     265979   1.5  129  2640
1458  1138       RL         NaN    0      94000   1.0  135  1020
1459  1350       RM         NaN    0     122000   2.0  136  2153

[1460 rows x 8 columns]
```

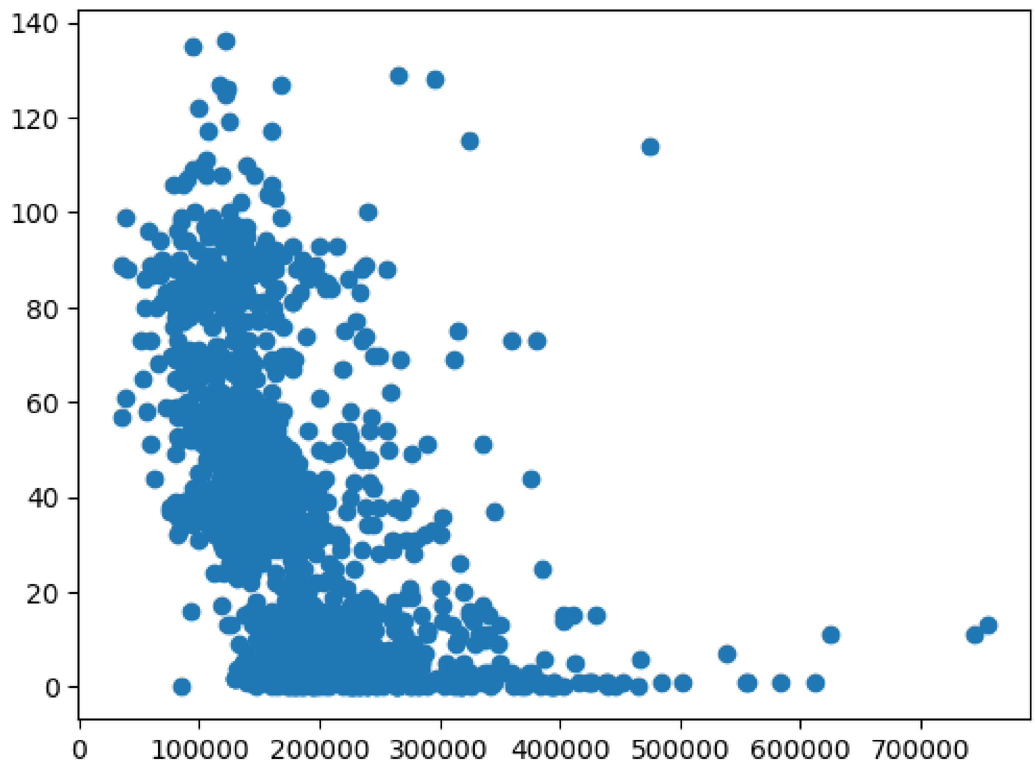
### 1.0.6 Produce a scatter plot of SalePrice and TSF.

[ ]:
```python
plt.scatter(data['SalePrice'], data['TSF'])
plt.show()
```

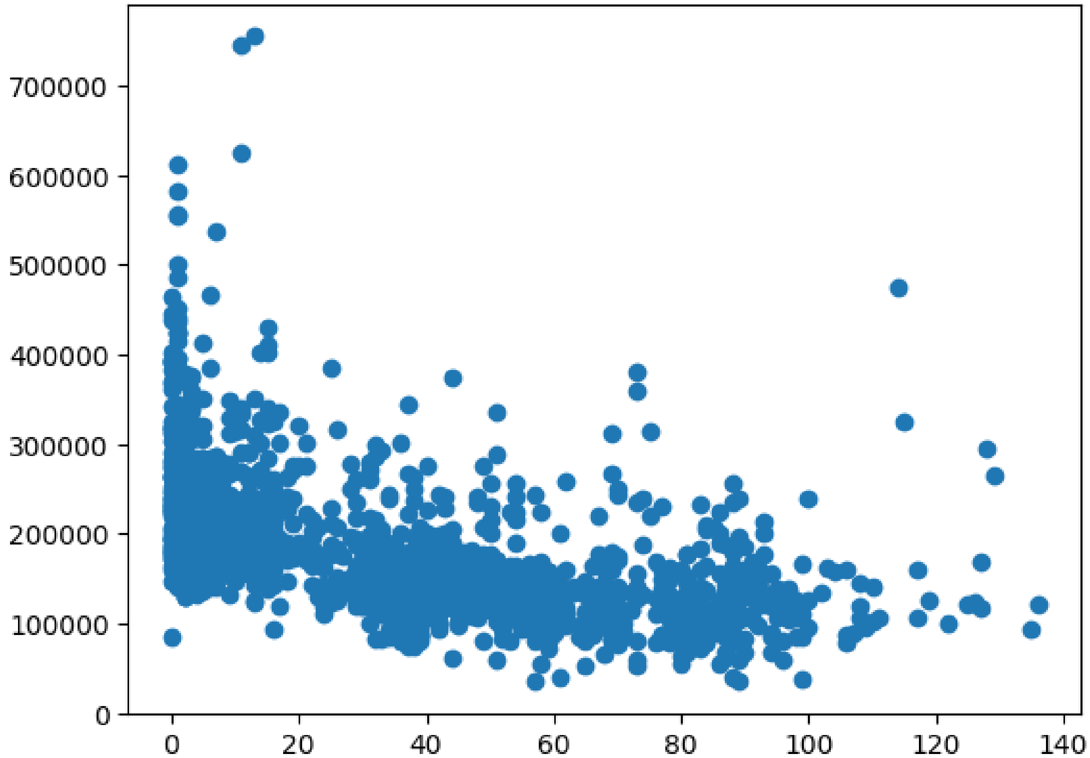



### 1.0.7 Produce a scatter plot of SalePrice and Age.

[ ]:
```python
plt.scatter(data['SalePrice'], data['Age'])
plt.show()
```

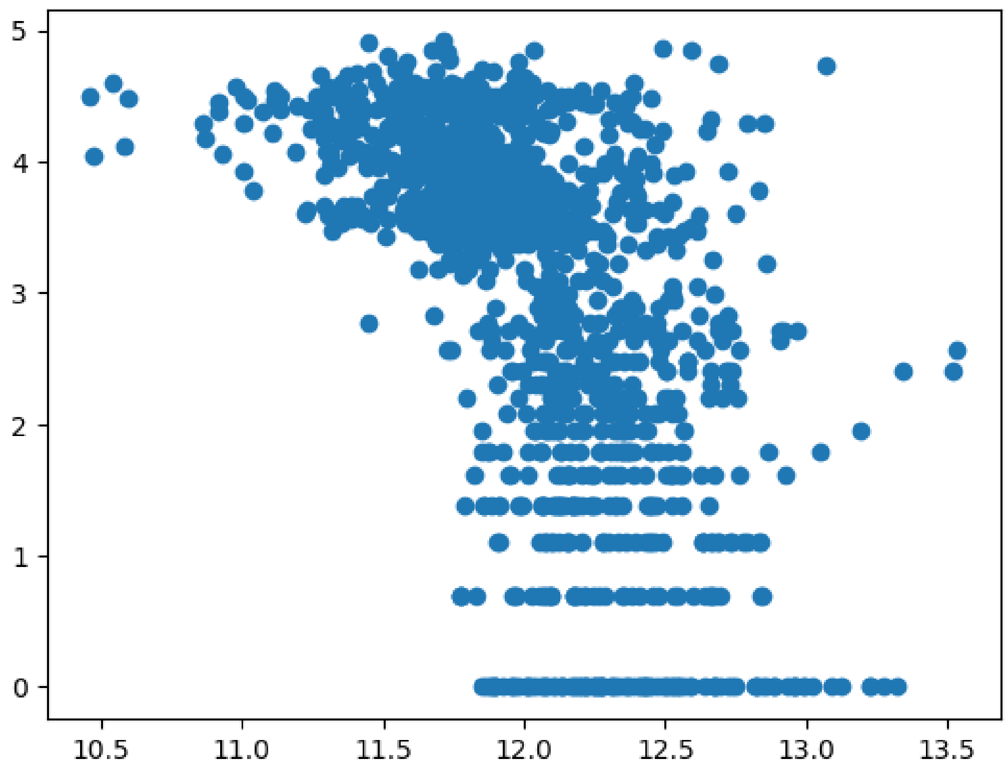



```
[ ]: plt.scatter(data['Age'], data['SalePrice'])
     plt.show()
```




[ ]:
```
saleprice_data = data['SalePrice']     #taking the log was not useful
Age_data = data['Age']


log_saleprice_data = np.log(saleprice_data)
log_Age_data = np.log(Age_data)
plt.scatter(log_saleprice_data,log_Age_data)
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/pandas/core/arraylike.py:402:
RuntimeWarning: divide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
```




### 1.0.8 Split data into two subsets, a training set and a validation set. The training set has approximately 80% of the data.

[ ]:
```
#one way to split the data into testing and training

import sklearn.model_selection as ms

X = data
Y = data

XTrain, XTest, YTrain, YTest = ms.train_test_split(X, Y, test_size= 0.2,␣
↪random_state=7)
```

### 1.0.9 Build the first multiple regression model with four numerical predictors TSF, Age, BATH, and GAR (Model I)

[ ]:
```
from sklearn.model_selection import train_test_split, cross_val_predict
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
```

```
X = data[['TSF', 'Age', 'BATH', 'GAR']]
y = data['SalePrice']

X_train_1, X_test_1, y_train_1, y_test_1 = train_test_split(X, y, test_size=0.
↪2, random_state=42)

model = LinearRegression()
model.fit(X_train_1, y_train_1)

y_pred = model.predict(X_test_1)

y_pred_train = model.predict(X_train_1)

mse = mean_squared_error(y_test_1, y_pred)
r2 = r2_score(y_test_1, y_pred)

print(f'Mean Squared Error (MSE): {mse}')
print(f'R-squared (R2): {r2}')
print()
print()
print()

X_train_1 = sm.add_constant(X_train_1)
model = sm.OLS(y_train_1, X_train_1).fit()


print(model.summary())

kf = KFold(n_splits=5, shuffle=True, random_state=42)
mse_scores = []
r2_scores = []

for train_index, test_index in kf.split(X):
    X_train_cv, X_test_cv = X.iloc[train_index], X.iloc[test_index]
    y_train_cv, y_test_cv = y.iloc[train_index], y.iloc[test_index]

    X_train_cv = sm.add_constant(X_train_cv)
    model_cv = sm.OLS(y_train_cv, X_train_cv).fit()


    X_test_cv = sm.add_constant(X_test_cv)
    y_pred_cv = model_cv.predict(X_test_cv)


    mse_cv = mean_squared_error(y_test_cv, y_pred_cv)
    r2_cv = r2_score(y_test_cv, y_pred_cv)
```

```
    mse_scores.append(mse_cv)
    r2_scores.append(r2_cv)


mean_mse_cv = np.mean(mse_scores)
mean_r2_cv = np.mean(r2_scores)

# Define and train the regression model
model = LinearRegression()
model.fit(X_train_1, y_train_1)

# Make predictions on the testing set
# Calculate residuals for the training set

residuals_train = y_train_1 - y_pred_train

# Calculate ASE for the training set
sse_train = np.sum(residuals_train**2)
ase_scores = sse_train / len(y_train_1)
print(f'ASE for Training Set: {ase_scores}')

mean_ase_cv = np.mean(ase_scores)

data_ASE = {
    'K-fold': ['1','2','3','4','5'],
    'Cross-Validation MSE Scores':[␣
 ↪mse_scores[0],mse_scores[1],mse_scores[2],mse_scores[3],mse_scores[4]],
    'Mean Cross-Validation MSE': [mean_mse_cv]*5,
    'Cross-Validation R2 Scores':␣
 ↪[r2_scores[0],r2_scores[1],r2_scores[2],r2_scores[3],r2_scores[4]],
    'Mean Cross-Validation R2': [mean_r2_cv]*5,
    'Average Squared Error (ASE)': [ase_scores]*5
}

ASE_model_1 = pd.DataFrame(data_ASE)

model_1_table = pd.DataFrame(data_ASE)

style_dict = {
    'text-align': 'center',
    'border': '1px solid black'
}

model_1_table.reset_index(drop=True, inplace=True)
model_1_table = model_1_table.reset_index(drop=True)

styled_Table_model_1 = model_1_table.style.set_properties(**style_dict)
```

```
styled_Table_model_1
```

```
Mean Squared Error (MSE): 2632874083.439605
R-squared (R2): 0.5447171052964432



                            OLS Regression Results
==============================================================================
Dep. Variable:              SalePrice   R-squared:                       0.716
Model:                            OLS   Adj. R-squared:                  0.715
Method:                 Least Squares   F-statistic:                     732.4
Date:                Tue, 10 Oct 2023   Prob (F-statistic):          7.37e-316
Time:                        01:27:52   Log-Likelihood:                -14112.
No. Observations:                1168   AIC:                         2.823e+04
Df Residuals:                    1163   BIC:                         2.826e+04
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       5.171e+04   6036.094      8.567      0.000    3.99e+04    6.36e+04
TSF          105.2914      4.066     25.893      0.000      97.313     113.270
Age         -920.4186     54.521    -16.882      0.000   -1027.389    -813.449
BATH        -1.91e+04   3419.567     -5.587      0.000   -2.58e+04   -1.24e+04
GAR           80.9210      7.361     10.993      0.000      66.479      95.363
==============================================================================
Omnibus:                      342.586   Durbin-Watson:                   1.953
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             6036.936
Skew:                           0.880   Prob(JB):                         0.00
Kurtosis:                      13.998   Cond. No.                     8.36e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 8.36e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
ASE for Training Set: 1828356162.4328048
```

[ ]: <pandas.io.formats.style.Styler at 0x7fa56e672e30>

#Build the second multiple regression model with four numerical predictors and two categorical predictors (Model II)

[ ]:
```
import statsmodels.api as sm
```

```
X_2 = pd.get_dummies(data[['TSF', 'Age', 'BATH', 'GAR', 'FireplaceQu',␣
↪'MSZoning']], columns=['MSZoning','FireplaceQu'], drop_first=True)
y = data['SalePrice']
X_train_2, X_test_2, y_train_2, y_test_2 = train_test_split(X_2, y, test_size=0.
↪2, random_state=42)

model_2 = LinearRegression()
model_2.fit(X_train_2, y_train_2)

y_pred_2 = model_2.predict(X_test_2)

# Calculate SSE for the training set
y_pred_train = model_2.predict(X_train_2)
sse_train = ((y_test_2 - y_pred_2) ** 2).sum()

# Calculate the number of data points
n_train = len(y_train_2)

# Calculate ASE for the test set and training set
ase_scores_2 = sse_train / n_train
print(f'ASE for Training Set: {ase_scores_2}')

mse_2 = mean_squared_error(y_test_2, y_pred_2)
r2_2 = r2_score(y_test_2, y_pred_2)

print(f'Mean Squared Error (MSE): {mse}')

X_2 = sm.add_constant(X_2)
model_2 = sm.OLS(y, X_2).fit()


summary = model_2.summary()

print(summary)


kf = KFold(n_splits=5, shuffle=True, random_state=42)
mse_scores_2 = []
r2_scores_2 = []

for train_index, test_index in kf.split(X_2):
    X_train_cv_2, X_test_cv_2 = X_2.iloc[train_index], X_2.iloc[test_index]
    y_train_cv_2, y_test_cv_2 = y.iloc[train_index], y.iloc[test_index]

    X_train_cv_2 = sm.add_constant(X_train_cv_2)
    model_cv_2 = sm.OLS(y_train_cv_2, X_train_cv_2).fit()
```

```
    X_test_cv_2 = sm.add_constant(X_test_cv_2)
    y_pred_cv_2 = model_cv_2.predict(X_test_cv_2)


    mse_cv_2 = mean_squared_error(y_test_cv_2, y_pred_cv_2)
    r2_cv_2 = r2_score(y_test_cv_2, y_pred_cv_2)

    mse_scores_2.append(mse_cv_2)
    r2_scores_2.append(r2_cv_2)


mean_mse_cv_2 = np.mean(mse_scores_2)
mean_r2_cv_2 = np.mean(r2_scores_2)

mean_ase_cv_2 = np.mean(ase_scores_2)

data_ASE_2 = {
    'K-fold': ['1','2','3','4','5'],
    'Cross-Validation MSE Scores':[␣
 ↪mse_scores_2[0],mse_scores_2[1],mse_scores_2[2],mse_scores_2[3],mse_scores_2[4]],
    'Mean Cross-Validation MSE': [mean_mse_cv_2]*5,
    'Cross-Validation R2 Scores':␣
 ↪[r2_scores_2[0],r2_scores_2[1],r2_scores_2[2],r2_scores_2[3],r2_scores_2[4]],
    'Mean Cross-Validation R2': [mean_r2_cv_2]*5,
    'Average Squared Error (ASE)': [ase_scores_2]*5
}

ASE_model_2 = pd.DataFrame(data_ASE_2)

model_2_table = pd.DataFrame(data_ASE_2)

style_dict = {
    'text-align': 'center',
    'border': '1px solid black'
}

model_2_table.reset_index(drop=True, inplace=True)
model_2_table = model_2_table.reset_index(drop=True)

styled_Table_model_2 = model_2_table.style.set_properties(**style_dict)

styled_Table_model_2
```

```
ASE for Training Set: 612833834.7511606
Mean Squared Error (MSE): 2632874083.439605
                            OLS Regression Results
```

```
==============================================================================
Dep. Variable:              SalePrice   R-squared:                       0.706
Model:                            OLS   Adj. R-squared:                  0.703
Method:                 Least Squares   F-statistic:                     289.3
Date:                Tue, 10 Oct 2023   Prob (F-statistic):               0.00
Time:                        01:27:58   Log-Likelihood:                -17651.
No. Observations:                1460   AIC:                         3.533e+04
Df Residuals:                    1447   BIC:                         3.540e+04
Df Model:                          12
Covariance Type:            nonrobust
================================================================================
==
                     coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------------
--
const           1.988e+04   1.51e+04      1.317      0.188   -9738.277
4.95e+04
TSF               85.9443      3.688     23.301      0.000      78.709
93.179
Age             -805.6540     52.817    -15.254      0.000    -909.260
-702.048
BATH           -9457.0258   3062.585     -3.088      0.002   -1.55e+04
-3449.444
GAR               74.7712      6.815     10.971      0.000      61.402
88.140
MSZoning_FV     3.455e+04   1.52e+04      2.278      0.023    4793.530
6.43e+04
MSZoning_RH     2.465e+04   1.75e+04      1.404      0.160   -9776.439
5.91e+04
MSZoning_RL     3.791e+04    1.4e+04      2.712      0.007    1.05e+04
6.53e+04
MSZoning_RM      2.84e+04   1.41e+04      2.019      0.044     807.209
5.6e+04
FireplaceQu_Fa  2801.5755   7740.198      0.362      0.717   -1.24e+04
1.8e+04
FireplaceQu_Gd  2.287e+04   3001.796      7.620      0.000     1.7e+04
2.88e+04
FireplaceQu_Po -6004.0568   9825.249     -0.611      0.541   -2.53e+04
1.33e+04
FireplaceQu_TA   524.6274   3221.184      0.163      0.871   -5794.062
6843.317
==============================================================================
Omnibus:                      393.258   Durbin-Watson:                   1.501
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            25588.481
Skew:                           0.268   Prob(JB):                         0.00
Kurtosis:                      23.502   Cond. No.                     4.67e+04
==============================================================================
```

```
Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 4.67e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```

[ ]: <pandas.io.formats.style.Styler at 0x7fa56e517730>

#Build the third multiple regression model with all predictors using stepwise selection. (Model III)

[ ]:
```python
X = pd.get_dummies(
                      data[['TSF', 'Age', 'BATH', 'GAR', 'FireplaceQu',
 ↪'MSZoning']],
                      columns=['MSZoning','FireplaceQu']
                    )

y = data['SalePrice']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
 ↪random_state=42)

# step-wise with forward selection

def forward_selection(data, target):
    selected_features = []
    remaining_features = list(data.columns)

    while remaining_features:
        best_pvalue = float('inf')
        best_feature = None

        for feature in remaining_features:
            model = sm.OLS(target, sm.add_constant(data[selected_features +
 ↪[feature]])).fit()
            pvalue = model.pvalues[feature]

            if pvalue < best_pvalue:
                best_pvalue = pvalue
                best_feature = feature

        if best_pvalue < 0.05:  # Adjust the significance level as needed
            selected_features.append(best_feature)
            remaining_features.remove(best_feature)
        else:
            break

    return selected_features
```

```python
selected_features = forward_selection(X_train, y_train)
final_model = sm.OLS(y_train, sm.add_constant(X_train[selected_features])).fit()

summary = final_model.summary()

print(summary)
print()

residuals_train = y_train - final_model.predict(sm.
    ↪add_constant(X_train[selected_features]))
sse_train = np.sum(residuals_train**2)
ase_train = sse_train / len(y_train)
print(f'ASE for Training Set: {ase_train}')
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:              SalePrice   R-squared:                       0.740
Model:                            OLS   Adj. R-squared:                  0.739
Method:                 Least Squares   F-statistic:                     471.9
Date:                Tue, 10 Oct 2023   Prob (F-statistic):               0.00
Time:                        01:28:04   Log-Likelihood:                -14060.
No. Observations:                1168   AIC:                         2.814e+04
Df Residuals:                    1160   BIC:                         2.818e+04
Df Model:                           7
Covariance Type:            nonrobust
================================================================================
==
                     coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------------
--
const           4.606e+04   6440.229      7.152      0.000    3.34e+04
5.87e+04
TSF               92.9307      4.076     22.801      0.000      84.934
100.927
Age             -836.4845     54.283    -15.410      0.000    -942.988
-729.981
GAR               74.8727      7.079     10.577      0.000      60.984
88.761
FireplaceQu_Gd  2.284e+04   2873.402      7.948      0.000    1.72e+04
2.85e+04
FireplaceQu_Ex  7.231e+04   9879.562      7.319      0.000    5.29e+04
9.17e+04
BATH           -1.306e+04   3330.641     -3.923      0.000   -1.96e+04
-6529.908
MSZoning_RL     7870.8759   3101.107      2.538      0.011    1786.469
1.4e+04
```

```
==============================================================================
Omnibus:                      379.695   Durbin-Watson:                   1.968
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             7450.335
Skew:                           0.996   Prob(JB):                         0.00
Kurtosis:                      15.212   Cond. No.                     1.36e+04
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 1.36e+04. This might indicate that there are
strong multicollinearity or other numerical problems.

ASE for Training Set: 1672286859.904567
```

### 1.0.10 Backward Elimination

[ ]:
```python
def backward_elimination(X_train, y_train, significance_level=.6):
    num_predictors = X_train.shape[1]
    for i in range(num_predictors):
        model = sm.OLS(y_train, X_train).fit()
        p_values = model.pvalues[1:]
        max_p_value = p_values.max()
        if max_p_value > significance_level:
            max_p_value_index = p_values.idxmax()
            X_train = X_train.drop(max_p_value_index, axis=1)
        else:
            break
    return model


final_model = backward_elimination(X_train, y_train)

print(final_model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:              SalePrice   R-squared:                       0.741
Model:                            OLS   Adj. R-squared:                  0.739
Method:                 Least Squares   F-statistic:                     276.0
Date:                Tue, 10 Oct 2023   Prob (F-statistic):               0.00
Time:                        01:28:10   Log-Likelihood:                -14057.
No. Observations:                1168   AIC:                         2.814e+04
Df Residuals:                    1155   BIC:                         2.821e+04
Df Model:                          12
Covariance Type:            nonrobust
================================================================================
==
```

```
                     coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------------
--
TSF               92.7919      4.212     22.031      0.000      84.528
101.056
Age             -807.8424     56.161    -14.384      0.000    -918.032
-697.653
BATH           -1.396e+04   3366.568     -4.147      0.000   -2.06e+04
-7356.160
GAR               75.0921      7.124     10.540      0.000      61.114
89.070
MSZoning_C      2.242e+04   1.52e+04      1.476      0.140   -7386.549
5.22e+04
MSZoning_FV     5.511e+04   8477.244      6.500      0.000    3.85e+04
7.17e+04
MSZoning_RH     4.608e+04   1.21e+04      3.797      0.000    2.23e+04
6.99e+04
MSZoning_RL     5.369e+04   5812.316      9.236      0.000    4.23e+04
6.51e+04
MSZoning_RM      4.41e+04   6812.595      6.473      0.000    3.07e+04
5.75e+04
FireplaceQu_Ex  7.386e+04   1.01e+04      7.312      0.000     5.4e+04
9.37e+04
FireplaceQu_Fa  7232.4047   8285.344      0.873      0.383   -9023.606
2.35e+04
FireplaceQu_Gd  2.382e+04   3233.922      7.365      0.000    1.75e+04
3.02e+04
FireplaceQu_TA  2147.0836   3512.443      0.611      0.541   -4744.400
9038.567
==============================================================================
Omnibus:                      393.223   Durbin-Watson:                   1.974
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             7667.884
Skew:                           1.052   Prob(JB):                         0.00
Kurtosis:                      15.375   Cond. No.                     2.32e+04
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 2.32e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```

#Build the last multiple regression model with all predictors using LASSO selection (Model IV)

[ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
 ↪random_state=42)
```

```
alpha = 1.0
lasso_model = Lasso(alpha=alpha)
lasso_model.fit(X_train, y_train)

y_pred = lasso_model.predict(X_test)

y_pred_train = lasso_model.predict(X_train)
sse_train = ((y_train - y_pred_train) ** 2).sum()

# Calculate the number of data points
n_train = len(y_train)

# Calculate ASE for the test set and training set
ase_train = sse_train / n_train

print(f'ASE for Training Set: {ase_train}')

mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f"Mean Squared Error: {mse}")
print(f"R-squared: {r2}")

selected_features = [feature for feature, coef in zip(X.columns, lasso_model.
 ↪coef_) if coef != 0]
num_selected_predictors = np.sum(lasso_model.coef_ != 0)
print(f"Number of Selected Predictors: {num_selected_predictors}")

adjusted_r2 = 1 - ((1 - r2) * (n_train - 1) / (n_train -␣
 ↪num_selected_predictors - 1))
print(f"Adjusted R-squared: {adjusted_r2}")
```

```
ASE for Training Set: 1663397596.6416483
Mean Squared Error: 2319402961.3982716
R-squared: 0.5989232827762649
Number of Selected Predictors: 13
Adjusted R-squared: 0.5944050875215782
```

[ ]:
```
# Calculate residuals
residuals = y_test - y_pred

# Create a residual plot
plt.figure(figsize=(8, 6))
sns.scatterplot(x=y_pred, y=residuals)
plt.axhline(y=0, color='r', linestyle='--')  # Add a horizontal line at y=0 for␣
 ↪reference
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
```

```
plt.title('Residual Plot')
plt.show()
```


#After completion of this activity, complete the following table

[ ]:
```
data = {
    'R-square': ['0.716', '0.706', '0.741', '0.059'],
    'ASE': ['1.97e+09', '1.67e+09', '1.67e+09', '1.66e+09'],
    'ADJ R-square': ['0.715', '0.703', '0.739', '0.594'],
    'Model': ['OLS','OLS','STEP','LASSO']
}

Table = pd.DataFrame(data)

style_dict = {
    'text-align': 'center',
    'border': '1px solid black'
}

styled_Table = Table.style.set_properties(**style_dict).relabel_index(['1',
 ↪'2', '3', '4'], axis=0)
```

```
styled_Table
```

[ ]: <pandas.io.formats.style.Styler at 0x7fa56b6f6620>

## 2 What is the best model using the criterion of your choice?

Here we can compare the ASE between the models and typically we want to focus on the lower errors. Here model 2-4 have around the same ASE so we should hone in on them. As far as R^2, the higher the number, the better we can explain the variance of the dependant variable due to the independant variables. Here 3 has the slight advantage around 0.741 which means the around 74% of the variance of dependant variables can be explained by the independant variables. Since we are looking at 2-4, it looks like the Step-wise model is the best model.

## 3 Find the predicted value for each house in the table below:

[ ]:
```
data_points = [
    [3000, 10, 2.5, 600, 0, 0, 0, 1, 0, 1, 0, 0, 0],
    [3000, 10, 2.5, 600, 0, 0, 0, 1, 0, 0, 0, 0, 1],
    [3000, 10, 3, 600, 0, 0, 0, 0, 1, 1, 0, 0, 0],
    [3000, 15, 3, 600, 0, 0, 0, 0, 1, 0, 0, 0, 1],
    [3200, 20, 4, 800, 0, 0, 0, 0, 1, 1, 0, 0, 0]
]

predicted_prices = []

for data_point in data_points:
    predicted_sale_price = final_model.predict(sm.add_constant([data_point]))
    predicted_prices.append(predicted_sale_price[0])

for i, predicted_price in enumerate(predicted_prices):
    print(f'Predicted Sale Price for House {i + 1}: {predicted_price}')
```

```
Predicted Sale Price for House 1: 407992.24428484426
Predicted Sale Price for House 2: 336281.29243558727
Predicted Sale Price for House 3: 391426.84970698575
Predicted Sale Price for House 4: 315676.6859179327
Predicted Sale Price for House 5: 402963.80919305363
```