SWX1 Results Analysis

Python Imports

```
import numpy as np
import pandas as pd
from prettypandas import PrettyPandas
import patsy
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
import statsmodels.formula.api
from pyomo.environ import *

from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures

from IPython.display import display, Markdown, HTML

%matplotlib inline
PlotWidth = 6

import warnings
warnings.filterwarnings('ignore')
```

```
# helper functions for this notebook

# use SVG for matplotlib-based figures
%matplotlib inline
%config InlineBackend.figure_format = 'svg'

def coded_to_actual(coded_data, actual_lows, actual_highs):
    """Converts a pandas DataFrame from coded units to actuals."""
    actual_data = coded_data.copy()
    for col in actual_data.columns:
        if not (col in actual_highs and col in actual_lows):
            continue
        try:
            # convert continuous variables to their actual value
            actual_data[col] *= 0.5 * (float(actual_highs[col]) - float(actual_lows[co
l]))
            # don't need to cast to float here, if either are not a float exception wi
ll have been thrown
            actual_data[col] += 0.5 * (actual_highs[col] + actual_lows[col])
        except ValueError:
            # assume 2 level categorical
```

```
            actual_data[col] = actual_data[col].map({-1: actual_lows[col], 1: actual_h
ighs[col]})
    return actual_data

def get_tick_labels(key, lows, highs, units):
    """Returns a list of low/high labels with units (e.g. [8mm, 10mm])"""
    return [str(lows[key]) + units[key], str(highs[key]) + units[key]]

def backward_regression(X, y,
                           threshold_out,
                           verbose=True):
    included=list(X.columns)
    while True:
        changed=False
        model = sm.OLS(y, sm.add_constant(pd.DataFrame(X[included]))).fit()
        # use all coefs except intercept
        pvalues = model.pvalues.iloc[1:]
        worst_pval = pvalues.max() # null if pvalues is empty
        if worst_pval > threshold_out:
            changed=True
            worst_feature = pvalues.idxmax()
            included.remove(worst_feature)
            if verbose:
                print('Drop {:30} with p-value {:.6}'.format(worst_feature, worst_pval
))
        if not changed:
            break

    return included

def build_model(X,values,verbose=True):
    X = [sub.replace('alh', 'model.X1') for sub in X]
    X = [sub.replace('aps', 'model.X2') for sub in X]
    X = [sub.replace('aid', 'model.X3') for sub in X]
    X = [sub.replace('arw', 'model.X4') for sub in X]
    X = [sub.replace('awt', 'model.X5') for sub in X]
    X = [sub.replace(':', '*') for sub in X]
    model = str(values[0])
    i=1
    for v in X:
        model  += " + " + str(values[i]) + " * " + v
        i += 1
    if verbose:
        print(model)
    return model
```

Process CSV Files

```python
# importing the pandas library
import pandas as pd

# reading the csv file using read_csv
# storing the data frame in variable called df
df_cost = pd.read_csv('https://raw.githubusercontent.com/wilsongis/3DP_Experiments/main/Data/swx1_cost_raw.txt', sep='\t')
df_time = pd.read_csv('https://raw.githubusercontent.com/wilsongis/3DP_Experiments/main/Data/swx1_time_raw.txt', sep='\t')

# creating a list of column names by
# calling the .columns
list_of_columns_cost = list(df_cost.columns)
list_of_columns_time = list(df_time.columns)

# displaying the list of column names
print('List of Cost column names : ',
      list_of_columns_cost)
print('List of Time column names : ',
      list_of_columns_time)
```

```
List of Cost column names :  ['trial', 'lh', 'ps', 'id', 'rw', 'wt', 'alh', 'aps', 'aid', 'arw', 'awt', 'rep', 'cost']
List of Time column names :  ['trial', 'lh', 'ps', 'id', 'rw', 'wt', 'alh', 'aps', 'aid', 'arw', 'awt', 'time']
```

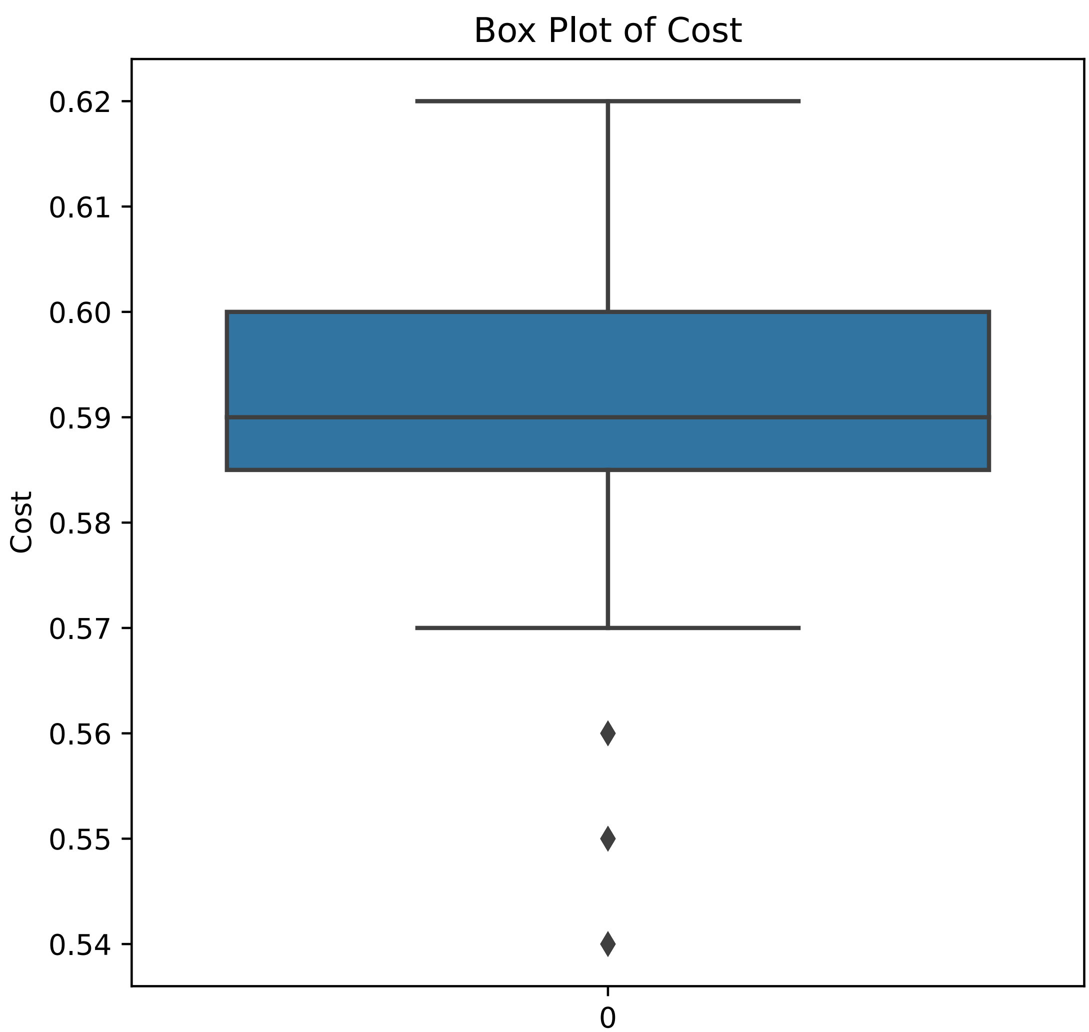
```python
display((Markdown("### Statistics for Cost")))
df_cost.cost.describe()
```

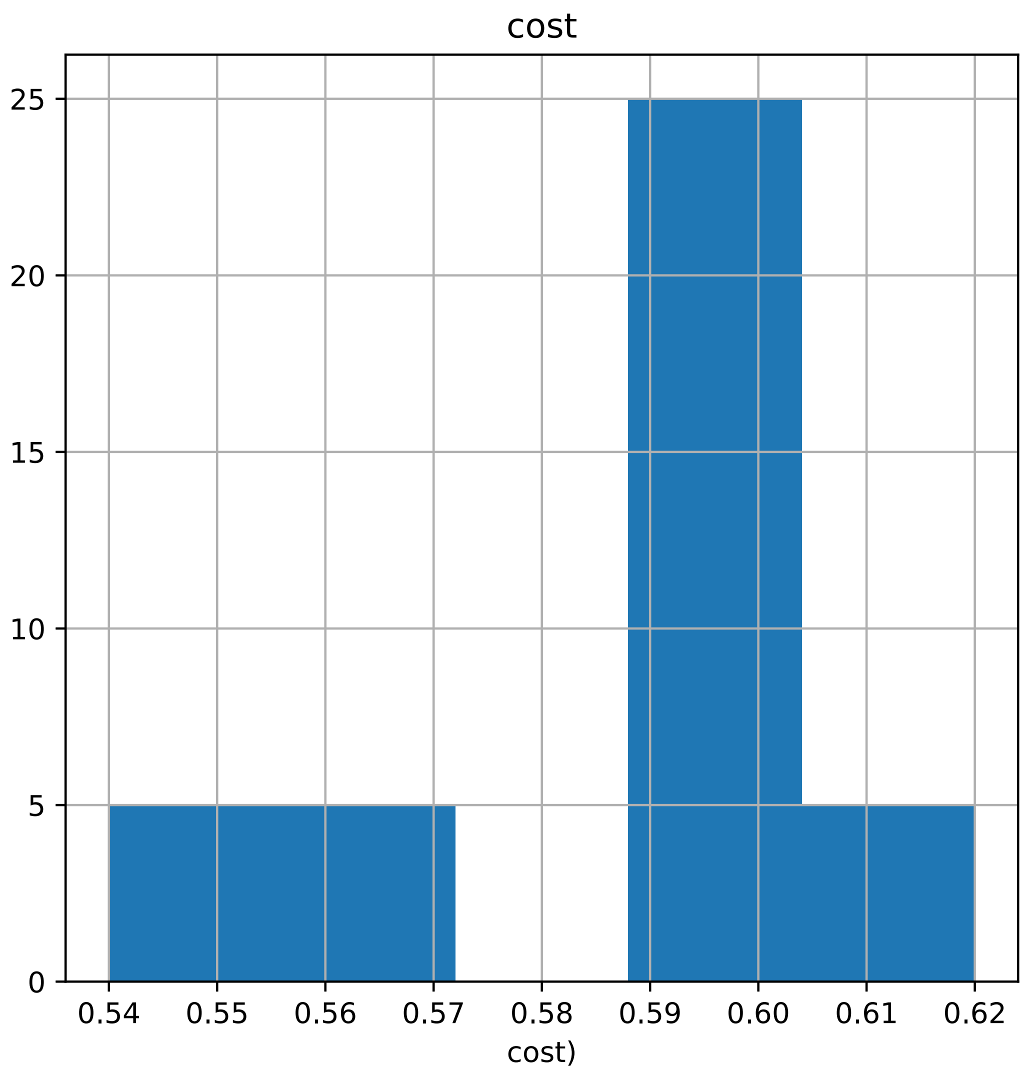
Statistics for Cost

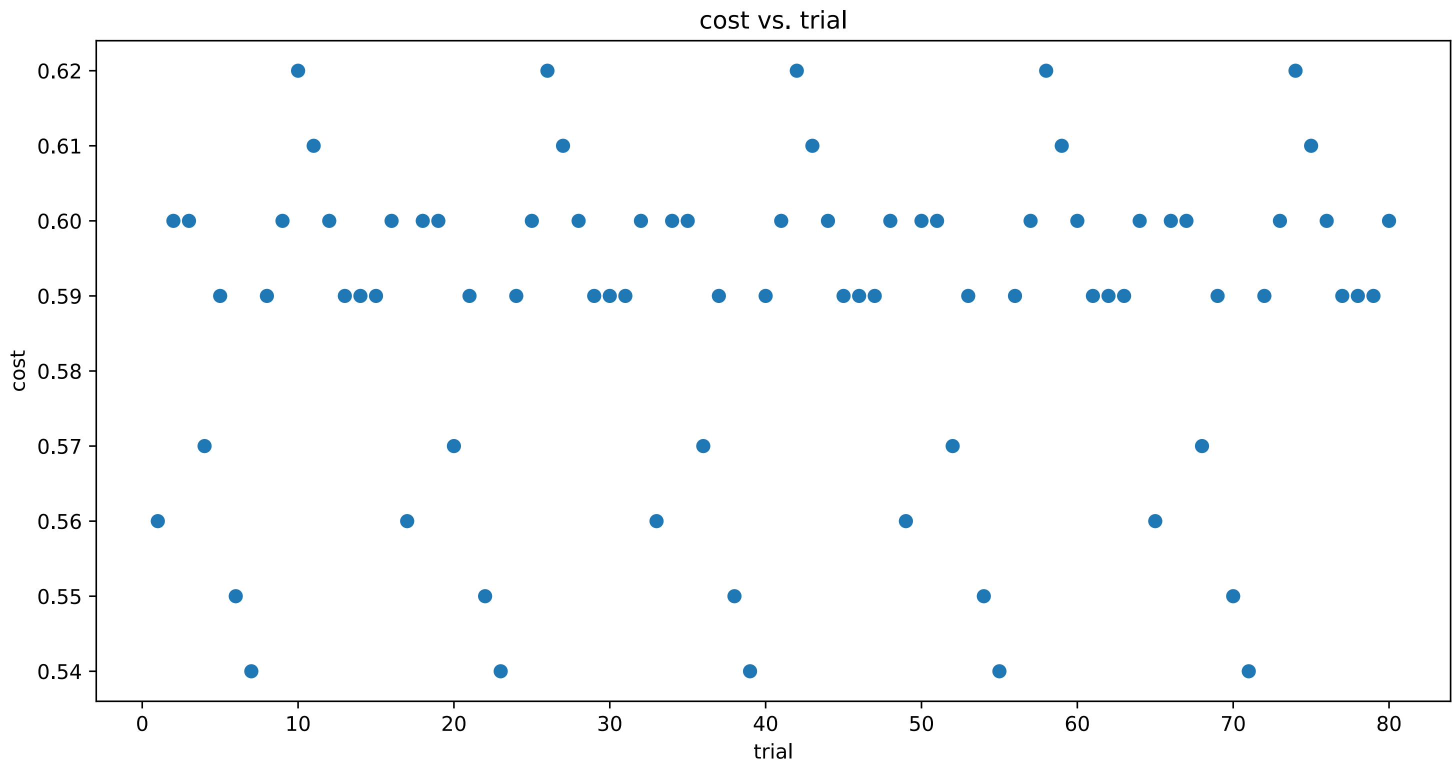
```
count    80.000000
mean      0.587500
std       0.021198
min       0.540000
25%       0.585000
50%       0.590000
75%       0.600000
max       0.620000
Name: cost, dtype: float64
```

```python
plt.figure(figsize=(PlotWidth, PlotWidth))
sns.boxplot(data=df_cost['cost'])
plt.title('Box Plot of Cost')
plt.ylabel('Cost')
plt.show()
```

```
plt.figure(figsize=(PlotWidth, PlotWidth))
df_cost['cost'].hist()
plt.title('cost')
plt.xlabel('cost)')
plt.show()

plt.figure(figsize=(PlotWidth*2, PlotWidth))
plt.scatter(df_cost['trial'], df_cost['cost'])
plt.title('cost vs. trial')
plt.xlabel('trial')
plt.ylabel('cost')
plt.show()
```




cost
25
20
15
10
5
0
0.54
0.55
0.56
0.57
0.58
0.59
0.60
0.61
0.62
cost)




```
display((Markdown("### Statistics for Time")))
df_time.time.describe()
```

Statistics for Time

```
count       80.000000
mean     11655.975000
std       2979.686374
min       7737.000000
25%       8981.000000
50%      11343.500000
75%      14568.500000
max      18254.000000
Name: time, dtype: float64
```

plt.figure(figsize=(PlotWidth, PlotWidth))
sns.boxplot(data=df_time['time'])
plt.title('Box Plot of Time')
plt.ylabel('Time')
plt.show()

plt.figure(figsize=(PlotWidth, PlotWidth))
df_time['time'].hist()
plt.title('time')

plt.xlabel(‘time)’)
plt.show()

plt.figure(figsize=(PlotWidth*2, PlotWidth))
plt.scatter(df_time[‘trial’], df_time[‘time’])
plt.title(‘time vs. trial’)
plt.xlabel(‘trial’)
plt.ylabel(‘time’)
plt.show()

Cost Analysis

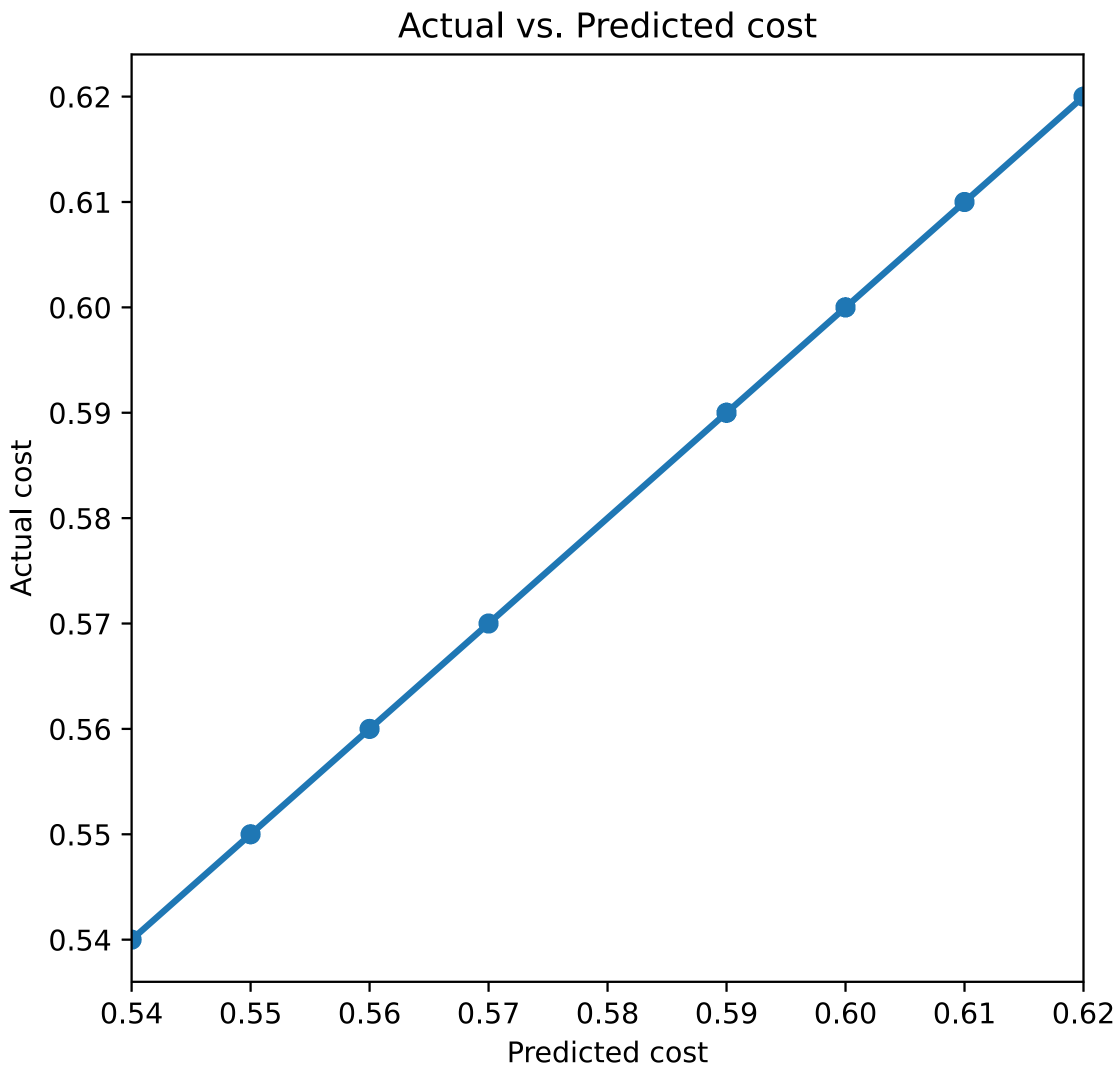
```
f = 'cost ~ (alh+aps+aid+arw+awt)**2'
y, X = patsy.dmatrices(f, df_cost, return_type='dataframe')
print(y[:5])
print(X[:5])
```

```
   cost
0  0.56
1  0.60
2  0.60
3  0.57
4  0.59
   Intercept   alh   aps   aid  arw  awt  alh:aps  alh:aid  alh:arw  alh:awt  \
0        1.0  0.16  60.0  0.25  0.4  0.8     9.60    0.040    0.064    0.128
1        1.0  0.28  60.0  0.25  0.4  1.2    16.80    0.070    0.112    0.336
2        1.0  0.16  72.0  0.25  0.4  1.2    11.52    0.040    0.064    0.192
3        1.0  0.28  72.0  0.25  0.4  0.8    20.16    0.070    0.112    0.224
4        1.0  0.16  60.0  0.15  0.4  1.2     9.60    0.024    0.064    0.192

   aps:aid  aps:arw  aps:awt  aid:arw  aid:awt  arw:awt
0     15.0     24.0     48.0     0.10     0.20     0.32
1     15.0     24.0     72.0     0.10     0.30     0.48
2     18.0     28.8     86.4     0.10     0.30     0.48
3     18.0     28.8     57.6     0.10     0.20     0.32
4      9.0     24.0     72.0     0.06     0.18     0.48
```

```

## An intercept is not added by default, so we need to add that here
X = sm.add_constant(X)
results = sm.OLS(y, X).fit()
results.summary()

print(results.summary())

plt.figure(figsize=(PlotWidth, PlotWidth))
sns.regplot(x=results.predict(X), y=y)
```

```
plt.xlabel('Predicted cost')
plt.ylabel('Actual cost')
plt.title('Actual vs. Predicted cost')
plt.show()
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                   cost   R-squared:                       1.000
Model:                            OLS   Adj. R-squared:                  1.000
Method:                 Least Squares   F-statistic:                 4.742e+26
Date:                Sat, 31 Jul 2021   Prob (F-statistic):               0.00
Time:                        00:15:09   Log-Likelihood:                 2594.2
No. Observations:                  80   AIC:                            -5156.
Df Residuals:                      64   BIC:                            -5118.
Df Model:                          15
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      0.1900   2.41e-14   7.89e+12      0.000       0.190       0.190
alh            0.2500   5.46e-14   4.58e+12      0.000       0.250       0.250
aps         -1.11e-16   3.34e-16     -0.332      0.741   -7.79e-16    5.57e-16
aid            0.4000    6.5e-14   6.15e+12      0.000       0.400       0.400
arw            0.3750   1.66e-14   2.26e+13      0.000       0.375       0.375
awt            0.3208   1.58e-14   2.03e+13      0.000       0.321       0.321
alh:aps     1.665e-16   6.94e-16      0.240      0.811   -1.22e-15    1.55e-15
alh:aid     1.599e-14   8.33e-14      0.192      0.848    -1.5e-13    1.82e-13
alh:arw     3.553e-15   2.08e-14      0.171      0.865    -3.8e-14    4.51e-14
alh:awt       -0.2083   2.08e-14     -1e+13      0.000      -0.208      -0.208
aps:aid     6.765e-17   8.33e-16      0.081      0.935    -1.6e-15    1.73e-15
aps:arw     3.816e-17   2.08e-16      0.183      0.855   -3.78e-16    4.54e-16
aps:awt     9.064e-17   2.08e-16      0.435      0.665   -3.25e-16    5.06e-16
aid:arw     3.608e-15    2.5e-14      0.144      0.886   -4.63e-14    5.35e-14
aid:awt       -0.2500    2.5e-14     -1e+13      0.000      -0.250      -0.250
arw:awt       -0.3125   6.24e-15     -5e+13      0.000      -0.313      -0.312
==============================================================================
Omnibus:                       13.228   Durbin-Watson:                   0.231
Prob(Omnibus):                  0.001   Jarque-Bera (JB):                3.783
Skew:                          -0.059   Prob(JB):                        0.151
Kurtosis:                       1.941   Cond. No.                     3.75e+04
==============================================================================


Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly speci
fied.
[2] The condition number is large, 3.75e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```



Reduced Cost Model

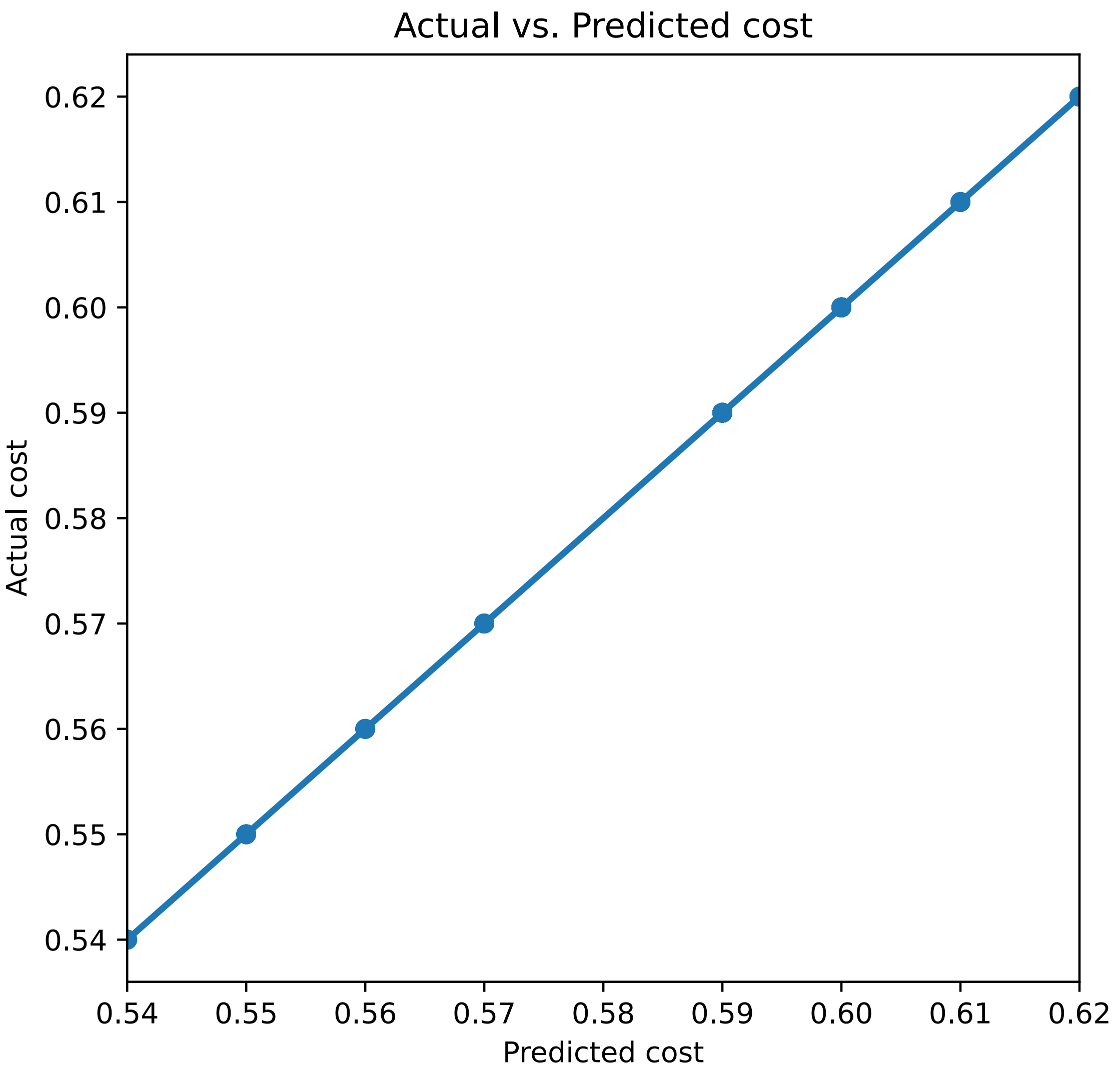
```
cost_included = backward_regression(X,y,.05)
cost_included.pop(0)
print(cost_included)
```

```
Drop aps:aid                            with p-value 0.93549
Drop alh:aid                            with p-value 0.99566
Drop aps:arw                            with p-value 0.970373
Drop aps:awt                            with p-value 0.994181
Drop alh:arw                            with p-value 0.949303
Drop alh:aps                            with p-value 0.930312
Drop aid:arw                            with p-value 0.679975
Drop aps                                with p-value 0.787536
['alh', 'aid', 'arw', 'awt', 'alh:awt', 'aid:awt', 'arw:awt']
```

```
y = df_cost['cost']
```

```
#y = df_cost['time']
X = X[cost_included]

## An intercept is not added by default, so we need to add that here
X = sm.add_constant(X)
results = sm.OLS(y, X).fit()
results.summary()

print(results.summary())

plt.figure(figsize=(PlotWidth, PlotWidth))
sns.regplot(x=results.predict(X), y=y)
plt.xlabel('Predicted cost')
plt.ylabel('Actual cost')
plt.title('Actual vs. Predicted cost')
plt.show()
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                   cost   R-squared:                       1.000
Model:                            OLS   Adj. R-squared:                  1.000
Method:                 Least Squares   F-statistic:                 5.386e+28
Date:                Sat, 31 Jul 2021   Prob (F-statistic):               0.00
Time:                        00:15:11   Log-Likelihood:                 2748.3
No. Observations:                  80   AIC:                            -5481.
Df Residuals:                      72   BIC:                            -5462.
Df Model:                           7
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.1900     1.1e-15   1.73e+14      0.000       0.190       0.190
alh            0.2500   2.92e-15   8.57e+13      0.000       0.250       0.250
aid            0.4000     3.5e-15   1.14e+14      0.000       0.400       0.400
arw            0.3750   8.75e-16   4.29e+14      0.000       0.375       0.375
awt            0.3208   1.08e-15   2.98e+14      0.000       0.321       0.321
alh:awt       -0.2083   2.86e-15  -7.29e+13      0.000      -0.208      -0.208
aid:awt       -0.2500   3.43e-15  -7.29e+13      0.000      -0.250      -0.250
arw:awt       -0.3125   8.58e-16  -3.64e+14      0.000      -0.313      -0.312
==============================================================================
Omnibus:                        1.172   Durbin-Watson:                   2.607
Prob(Omnibus):                  0.556   Jarque-Bera (JB):                1.172
Skew:                          -0.179   Prob(JB):                        0.556
Kurtosis:                       2.527   Cond. No.                         249.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly speci
fied.
```




```
cost_eq = build_model(cost_included,results.params,False)
print("Cost = " + cost_eq)
```

```
Cost = 0.1900000000000014 + 0.24999999999999423 * model.X1 + 0.3999999999999986 * mode
l.X3 + 0.37499999999999967 * model.X4 + 0.320833333333332 * model.X5 + -0.208333333333
32477 * model.X1*model.X5 + -0.25000000000000133 * model.X3*model.X5 + -0.312499999999
9999 * model.X4*model.X5
```

Time Analysis

```
f = 'time ~ (alh+aps+aid+arw+awt)**2'
y, X = patsy.dmatrices(f, df_time, return_type='dataframe')
print(y[:5])
print(X[:5])
```

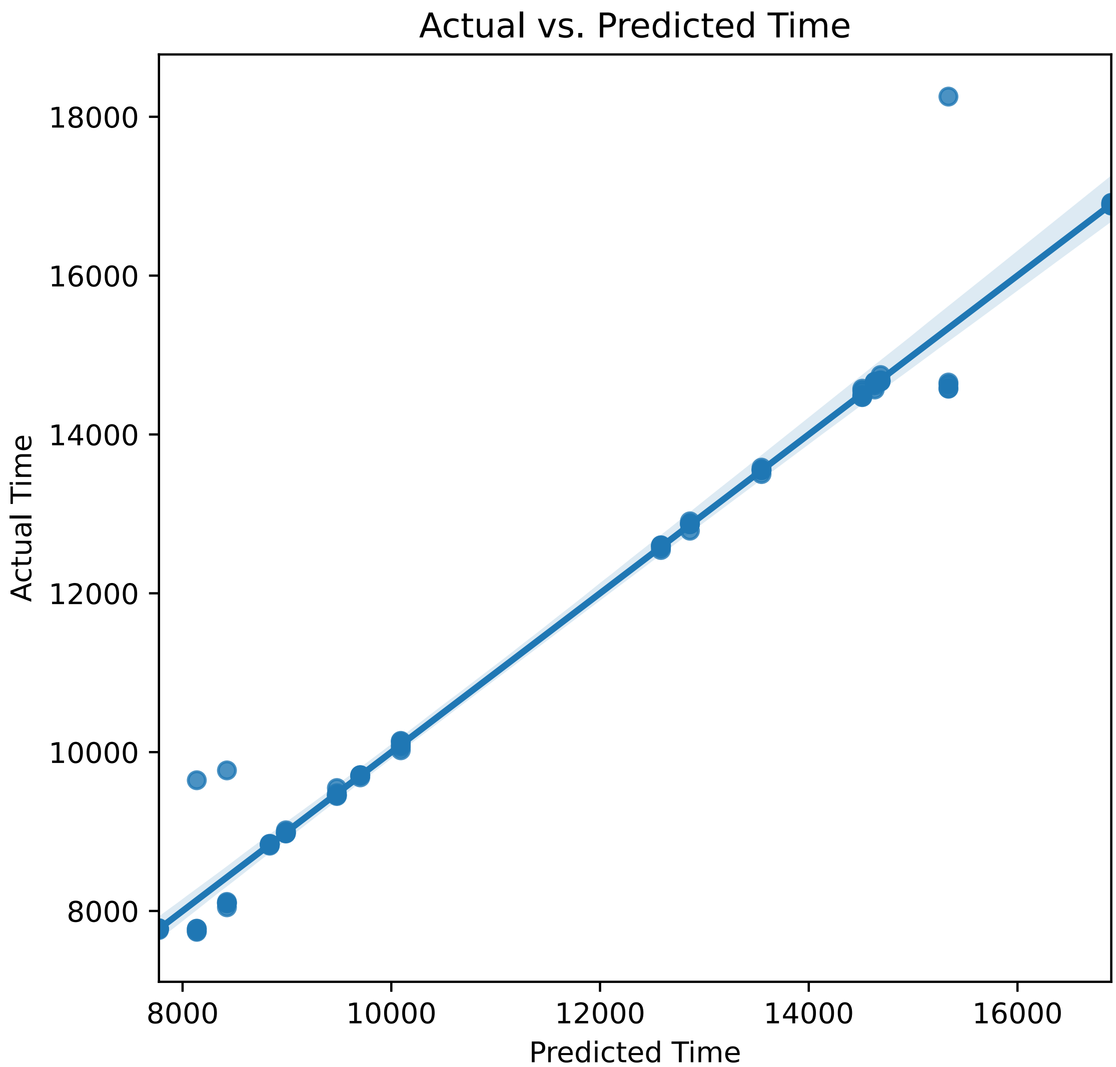
```
      time
0  16916.0
1   9016.0
```

```
2  12906.0
3   9711.0
4  14617.0
   Intercept   alh   aps   aid  arw  awt  alh:aps  alh:aid  alh:arw  alh:awt  \
0        1.0  0.16  60.0  0.25  0.4  0.8     9.60    0.040    0.064    0.128
1        1.0  0.28  60.0  0.25  0.4  1.2    16.80    0.070    0.112    0.336
2        1.0  0.16  72.0  0.25  0.4  1.2    11.52    0.040    0.064    0.192
3        1.0  0.28  72.0  0.25  0.4  0.8    20.16    0.070    0.112    0.224
4        1.0  0.16  60.0  0.15  0.4  1.2     9.60    0.024    0.064    0.192

   aps:aid  aps:arw  aps:awt  aid:arw  aid:awt  arw:awt
0     15.0     24.0     48.0     0.10     0.20     0.32
1     15.0     24.0     72.0     0.10     0.30     0.48
2     18.0     28.8     86.4     0.10     0.30     0.48
3     18.0     28.8     57.6     0.10     0.20     0.32
4      9.0     24.0     72.0     0.06     0.18     0.48
```

```
## An intercept is not added by default, so we need to add that here
X = sm.add_constant(X)
results = sm.OLS(y, X).fit()
results.summary()

print(results.summary())

plt.figure(figsize=(PlotWidth, PlotWidth))
sns.regplot(x=results.predict(X), y=y)
plt.xlabel('Predicted Time')
plt.ylabel('Actual Time')
plt.title('Actual vs. Predicted Time')
plt.show()
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                   time   R-squared:                       0.977
Model:                            OLS   Adj. R-squared:                  0.972
Method:                 Least Squares   F-statistic:                     185.2
Date:                Sat, 31 Jul 2021   Prob (F-statistic):           1.10e-46
Time:                        00:15:12   Log-Likelihood:                -601.25
No. Observations:                  80   AIC:                             1235.
Df Residuals:                      64   BIC:                             1273.
Df Model:                          15
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept   4.021e+04   5357.872      7.505      0.000    2.95e+04    5.09e+04
alh        -9.962e+04   1.21e+04     -8.204      0.000   -1.24e+05   -7.54e+04
aps         -162.2569     74.321     -2.183      0.033    -310.730     -13.784
aid         3.035e+04   1.45e+04      2.099      0.040    1459.283    5.92e+04
```

```
arw          -1.256e+04   3689.304     -3.404      0.001   -1.99e+04   -5188.518
awt          -2640.6667   3518.056     -0.751      0.456   -9668.792    4387.459
alh:aps        815.0694    154.301      5.282      0.000     506.818    1123.321
alh:aid       -2.18e+04   1.85e+04     -1.177      0.243   -5.88e+04    1.52e+04
alh:arw       5550.0000   4629.021      1.199      0.235   -3697.533    1.48e+04
alh:awt       1404.1667   4629.021      0.303      0.763   -7843.367    1.07e+04
aps:aid       -313.6667    185.161     -1.694      0.095    -683.568      56.235
aps:arw          5.8750     46.290      0.127      0.899     -86.600      98.350
aps:awt        -76.7083     46.290     -1.657      0.102    -169.184      15.767
aid:arw       3282.5000   5554.826      0.591      0.557   -7814.540    1.44e+04
aid:awt      -2557.5000   5554.826     -0.460      0.647   -1.37e+04    8539.540
arw:awt       8909.3750   1388.706      6.416      0.000    6135.115    1.17e+04
==============================================================================
Omnibus:                      104.143   Durbin-Watson:                   1.987
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             2030.811
Skew:                           4.092   Prob(JB):                         0.00
Kurtosis:                      26.286   Cond. No.                     3.75e+04
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly speci
fied.
[2] The condition number is large, 3.75e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```



Time Reduced Model

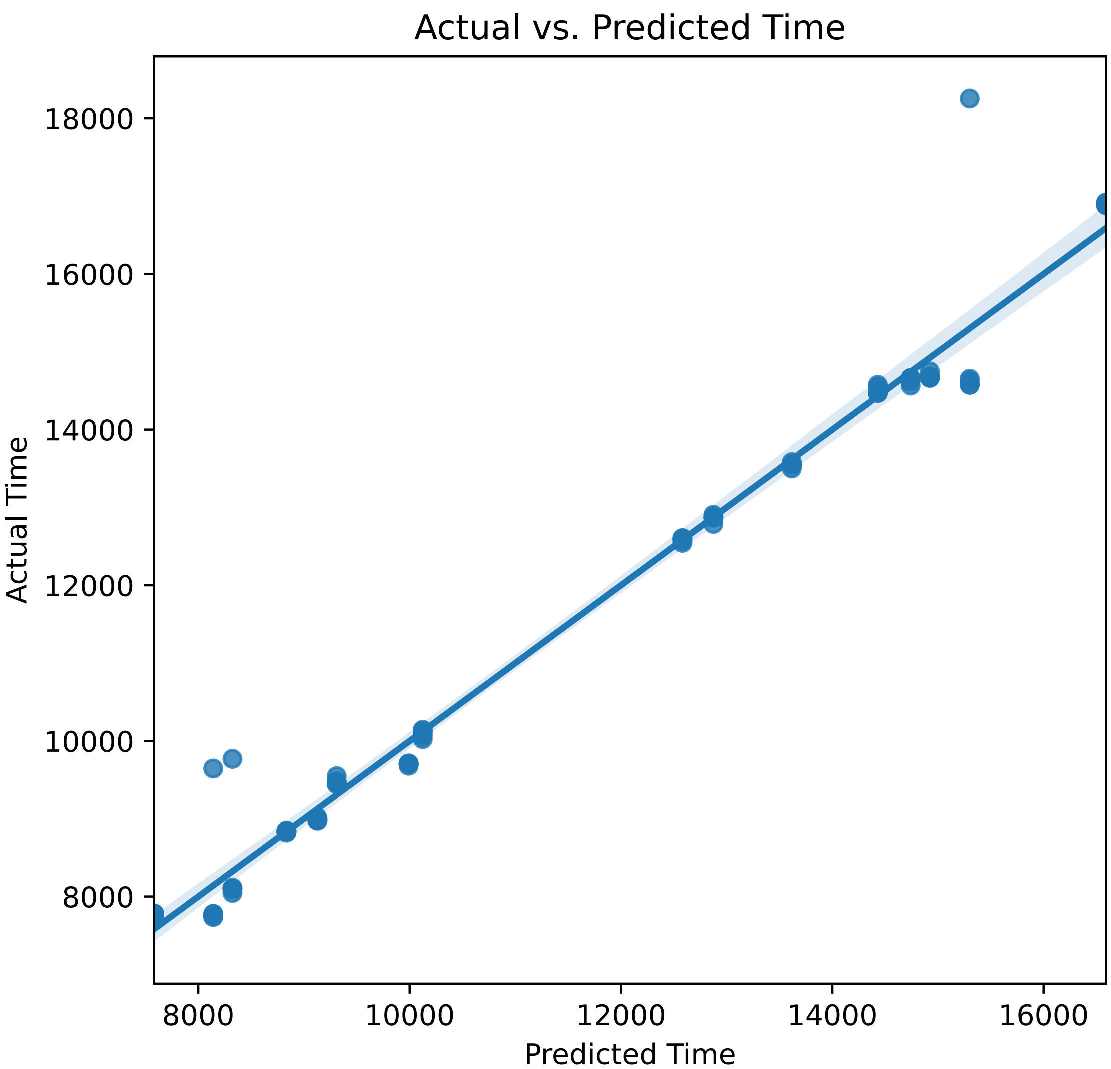
```
time_included = backward_regression(X,y,.05)
time_included.pop(0)
print(time_included)
```

```
Drop aps:arw                            with p-value 0.899404
Drop alh:awt                            with p-value 0.760839
Drop aid:awt                            with p-value 0.641928
Drop aid:arw                            with p-value 0.54847
Drop awt                                with p-value 0.362301
Drop alh:aid                            with p-value 0.23091
Drop alh:arw                            with p-value 0.223997
Drop aps:aid                            with p-value 0.0884672
['alh', 'aps', 'aid', 'arw', 'alh:aps', 'aps:awt', 'arw:awt']
```

```
y = df_time['time']
```

```
X = X[time_included]

## An intercept is not added by default, so we need to add that here
X = sm.add_constant(X)
results = sm.OLS(y, X).fit()
results.summary()

print(results.summary())

plt.figure(figsize=(PlotWidth, PlotWidth))
sns.regplot(x=results.predict(X), y=y)
plt.xlabel('Predicted Time')
plt.ylabel('Actual Time')
plt.title('Actual vs. Predicted Time')
plt.show()
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                   time   R-squared:                       0.975
Model:                            OLS   Adj. R-squared:                  0.973
Method:                 Least Squares   F-statistic:                     400.4
Date:                Sat, 31 Jul 2021   Prob (F-statistic):           5.59e-55
Time:                        00:15:14   Log-Likelihood:                -605.49
No. Observations:                  80   AIC:                             1227.
Df Residuals:                      72   BIC:                             1246.
Df Model:                           7
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       4.131e+04   2334.638     17.695      0.000    3.67e+04     4.6e+04
alh        -9.924e+04   1.02e+04     -9.762      0.000    -1.2e+05    -7.9e+04
aps         -181.6742     37.220     -4.881      0.000    -255.871    -107.478
aid         4263.5000   1104.486      3.860      0.000    2061.747    6465.253
arw        -9967.9370   1360.650     -7.326      0.000   -1.27e+04   -7255.531
alh:aps      815.0694    153.401      5.313      0.000     509.270    1120.868
aps:awt     -116.4994     12.715     -9.162      0.000    -141.846     -91.153
arw:awt     8583.8120   1332.338      6.443      0.000    5927.844    1.12e+04
==============================================================================
Omnibus:                       95.831   Durbin-Watson:                   1.951
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1447.100
Skew:                           3.713   Prob(JB):                         0.00
Kurtosis:                      22.467   Cond. No.                     1.80e+04
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly speci
fied.
[2] The condition number is large, 1.8e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```




```
time_eq = build_model(time_included,results.params,False)
print("Time = " + time_eq)
```

```
Time = 41311.84166666333 + -99242.9166666657 * model.X1 + -181.67424724344346 * model.X2 + 4263.500000000091 * model.X3 + -9967.93702290074 * model.X4 + 815.0694444444525 * model.X1*model.X2 + -116.4993638676806 * model.X2*model.X5 + 8583.812022900747 * model.X4*model.X5
```

Equations

```
display(Markdown("Cost = "))
print(cost_eq)

print("------")

display(Markdown("Time = "))
```

```
print(time_eq)
```

Cost =

```
0.1900000000000014 + 0.24999999999999423 * model.X1 + 0.3999999999999986 * model.X3 +
0.37499999999999967 * model.X4 + 0.320833333333332 * model.X5 + -0.20833333333332477 *
 model.X1*model.X5 + -0.25000000000000133 * model.X3*model.X5 + -0.3124999999999999 *
model.X4*model.X5
------
```

Time =

```
41311.84166666333 + -99242.9166666657 * model.X1 + -181.67424724344346 * model.X2 + 42
63.500000000091 * model.X3 + -9967.93702290074 * model.X4 + 815.0694444444525 * model.
X1*model.X2 + -116.4993638676806 * model.X2*model.X5 + 8583.812022900747 * model.X4*mo
del.X5
```

Optimization

```
model = ConcreteModel()

model.X1 = Var(within=NonNegativeReals)
model.X2 = Var(within=NonNegativeReals)
model.X3 = Var(within=NonNegativeReals)
model.X4 = Var(within=NonNegativeReals)
model.X5 = Var(within=NonNegativeReals)

model.C1 = Constraint(expr = model.X1 <= .28)
model.C2 = Constraint(expr = model.X2 <= 72)
model.C3 = Constraint(expr = model.X3 <= .25)
model.C4 = Constraint(expr = model.X4 <= .8)
model.C5 = Constraint(expr = model.X5 <= 1.2)

model.C6 = Constraint(expr = model.X1 >= .16)
model.C7 = Constraint(expr = model.X2 >= 60)
model.C8 = Constraint(expr = model.X3 >= .15)
model.C9 = Constraint(expr = model.X4 >= .4)
model.C10 = Constraint(expr = model.X5 >= .8)

model.f1 = Var()
model.f2 = Var()
model.C_f1 = Constraint(expr = model.f1 == (0.1900000000000014 + 0.24999999999999423 *
 model.X1 + 0.3999999999999986 * model.X3 + 0.37499999999999967 * model.X4 + 0.3208333
33333332 * model.X5 + -0.20833333333332477 * model.X1*model.X5 + -0.25000000000000133
* model.X3*model.X5 + -0.3124999999999999 * model.X4*model.X5))
model.C_f2 = Constraint(expr = model.f2 == (41311.84166666333 + -99242.9166666657 * mo
```

```
del.X1 + -181.67424724344346 * model.X2 + 4263.500000000091 * model.X3 + -9967.9370229
0074 * model.X4 + 815.0694444444525 * model.X1*model.X2 + -116.4993638676806 * model.X
2*model.X5 + 8583.812022900747 * model.X4*model.X5))
model.O_f1 = Objective(expr = model.f1, sense=minimize)
model.O_f2 = Objective(expr = model.f2, sense=minimize)

# max f1 separately
# install glpk solver:  sudo apt-get install glpk-utils
model.O_f2.deactivate()
solver = SolverFactory('ipopt')  #'cplex', 'ipopt'
solver.solve(model)

print('( X1 , X2, X3, X4, X5 ) = ( ' + str(value(model.X1)) + ' , ' + str(value(model.
X2)) + ' , ' + str(value(model.X3)) + ' , ' + str(value(model.X4)) + ' , ' + str(value
(model.X5)) + ' )')
print('f1 = ' + str(value(model.f1)))
print('f2 = ' + str(value(model.f2)))
f2_min = value(model.f2)

# max f2 separately
model.O_f2.activate()
model.O_f1.deactivate()
solver = SolverFactory('ipopt')  #'cplex', 'ipopt'
solver.solve(model)

print('( X1 , X2, X3, X4, X5 ) = ( ' + str(value(model.X1)) + ' , ' + str(value(model.
X2)) + ' , ' + str(value(model.X3)) + ' , ' + str(value(model.X4)) + ' , ' + str(value
(model.X5)) + ' )')
print('f1 = ' + str(value(model.f1)))
print('f2 = ' + str(value(model.f2)))
f2_max = value(model.f2)

# apply augmented $\epsilon$-Constraint
# max                   f1 + delta*s
# constraint    f2 - s = e
model.O_f1.activate()
model.O_f2.deactivate()

model.del_component(model.O_f1)
model.del_component(model.O_f2)

model.e = Param(initialize=0, mutable=True)
model.delta = Param(initialize=0.00001)
model.slack = Var(within=NonNegativeReals)
model.O_f1 = Objective(expr = model.f1 + model.delta * model.slack, sense=minimize)
model.C_e = Constraint(expr = model.f2 - model.slack == model.e)

n = 100
step = int((f2_max - f2_min) / n)
steps = list(range(int(f2_min),int(f2_max),step)) + [f2_max]
```

```python
x1_l, x2_l, x3_l, x4_l, x5_l = [], [], [], [], []
f1_l, f2_l = [], []
for i in steps:
    model.e = i
    solver.solve(model)
    x1_l.append(value(model.X1))
    x2_l.append(value(model.X2))
    x3_l.append(value(model.X3))
    x4_l.append(value(model.X4))
    x5_l.append(value(model.X5))
    f1_l.append(value(model.f1))
    f2_l.append(value(model.f2))
    # print(i, value(model.X1), value(model.X2), value(model.f1), value(model.slack), value(model.f2))
```

```
( X1 , X2, X3, X4, X5 ) = ( 0.16000001416311918 , 66.27123298035423 , 0.15000000195904373 , 0.40000000961690135 , 0.8000000093914478 )
f1 = 0.5400000040372802
f2 = 15258.427909188156
( X1 , X2, X3, X4, X5 ) = ( 0.28000000999993796 , 72.00000071997286 , 0.1499999900005892 , 0.39999999000704073 , 1.2000000119995038 )
f1 = 0.5900000002000078
f2 = 7582.113544009459
```

```python

```