SWX1 Results Analysis B

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

```
# 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/mai
n/Data/swx1_cost_power.txt', sep='\t')
df_time = pd.read_csv('https://raw.githubusercontent.com/wilsongis/3DP_Experiments/mai
n/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', 'ai
d', 'arw', 'awt', 'rep', 'cost']
List of Time column names :  ['trial', 'lh', 'ps', 'id', 'rw', 'wt', 'alh', 'aps', 'ai
d', 'arw', 'awt', 'time']
```

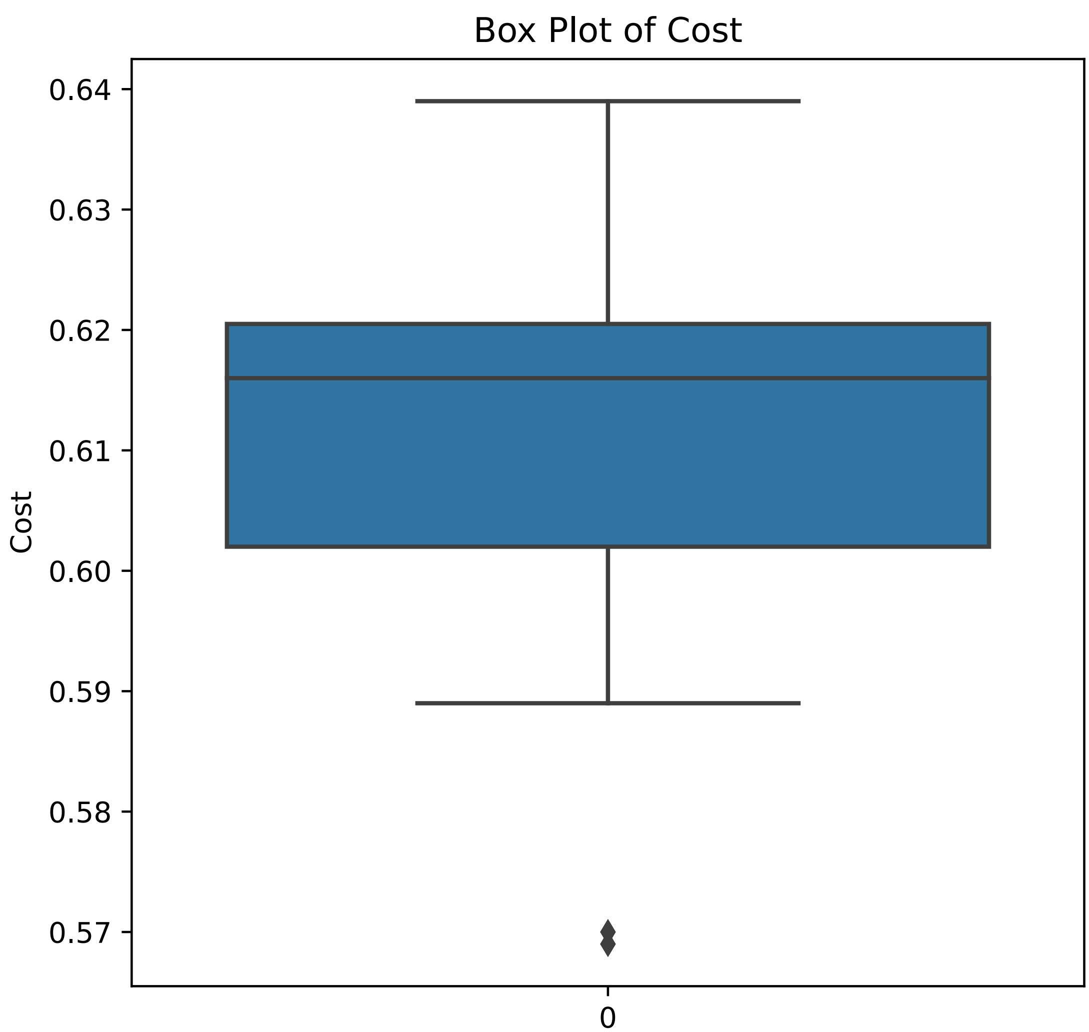
```
display((Markdown("### Statistics for Cost")))
df_cost.cost.describe()
```

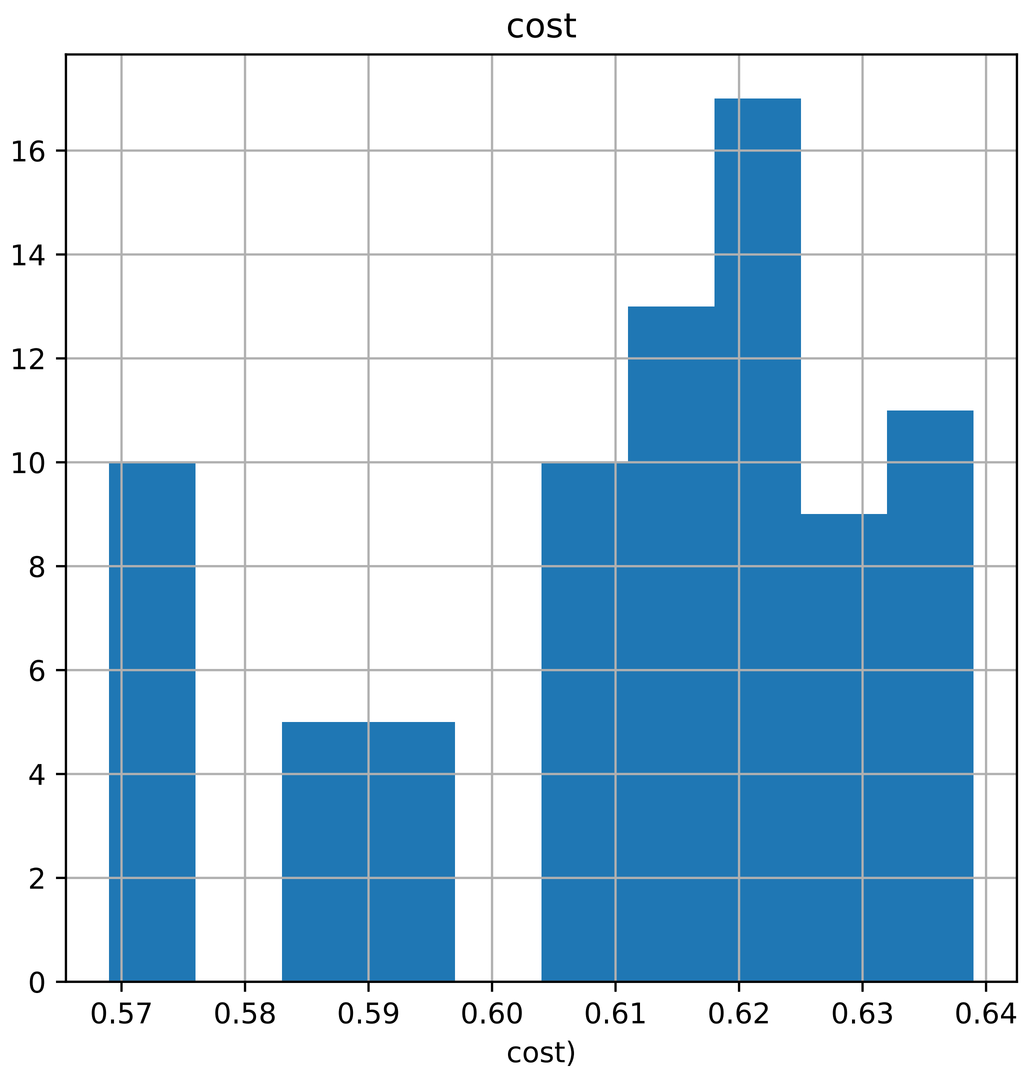
Statistics for Cost

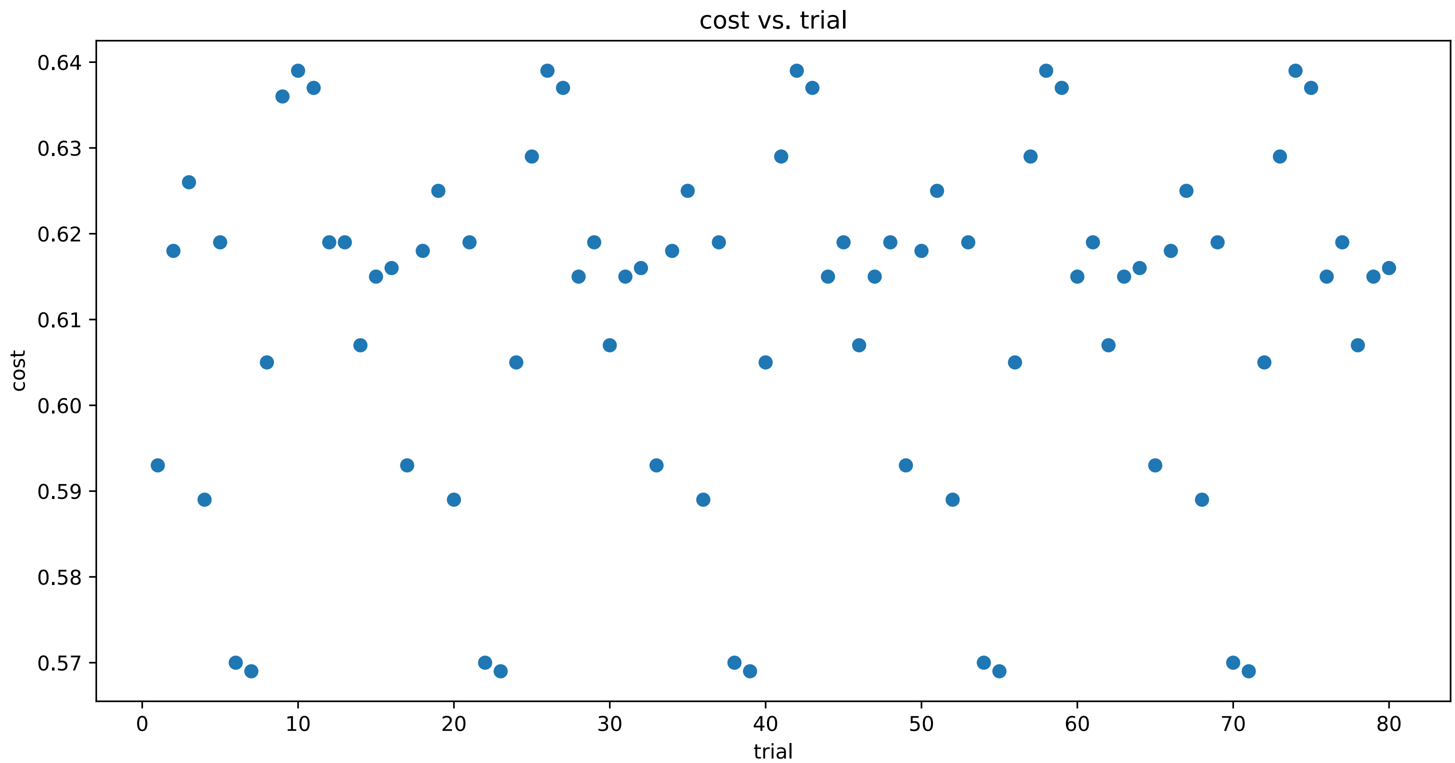
```
count    80.000000
mean      0.610500
std       0.020436
min       0.569000
25%       0.602000
50%       0.616000
75%       0.620500
max       0.639000
Name: cost, dtype: float64
```

```
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
16
14
12
10
8
6
4
2
0
0.57
0.58
0.59
0.60
0.61
0.62
0.63
0.64
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

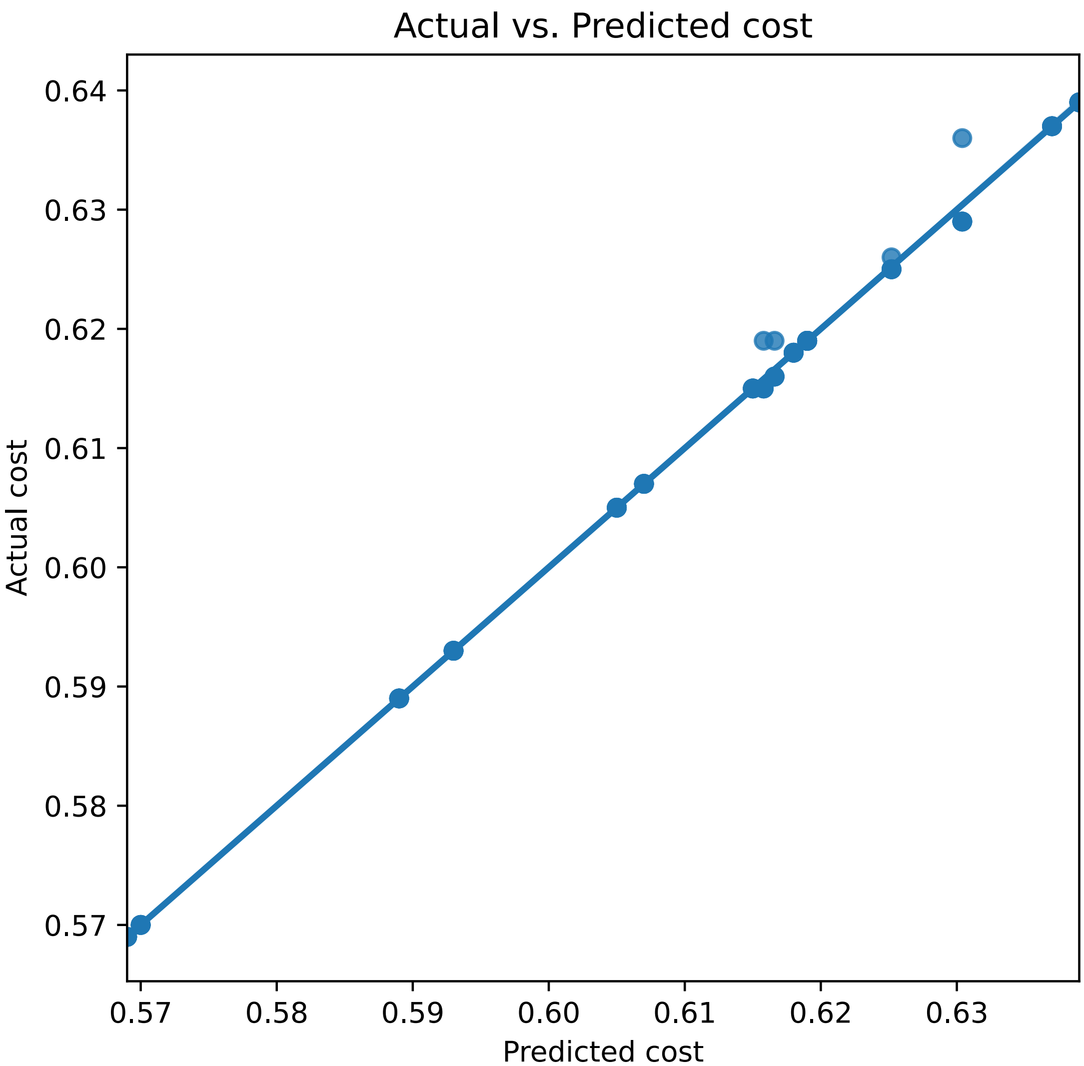
```
f = 'cost ~ (alh+aps+aid+arw+awt)**2'
y, X = patsy.dmatrices(f, df_cost, return_type='dataframe')
print(y[:5])
print(X[:5])
```

```
    cost
0  0.593
1  0.618
2  0.626
3  0.589
4  0.619
   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:                       0.998
Model:                            OLS   Adj. R-squared:                  0.998
Method:                 Least Squares   F-statistic:                     2342.
Date:                Sat, 31 Jul 2021   Prob (F-statistic):           1.34e-81
Time:                        20:30:59   Log-Likelihood:                 450.61
No. Observations:                  80   AIC:                            -869.2
Df Residuals:                      64   BIC:                            -831.1
Df Model:                          15
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      0.2615      0.010     25.046      0.000       0.241       0.282
alh            0.0721      0.024      3.046      0.003       0.025       0.119
aps           -0.0002      0.000     -1.227      0.224      -0.000       0.000
aid            0.4563      0.028     16.192      0.000       0.400       0.513
arw            0.3490      0.007     48.541      0.000       0.335       0.363
awt            0.3199      0.007     46.662      0.000       0.306       0.334
alh:aps        0.0013      0.000      4.388      0.000       0.001       0.002
alh:aid       -0.0083      0.036     -0.231      0.818      -0.080       0.064
alh:arw        0.0063      0.009      0.693      0.491      -0.012       0.024
alh:awt       -0.2104      0.009    -23.325      0.000      -0.228      -0.192
aps:aid       -0.0008      0.000     -2.309      0.024      -0.002      -0.000
aps:arw     4.167e-05   9.02e-05      0.462      0.646      -0.000       0.000
aps:awt       -0.0002   9.02e-05     -2.309      0.024      -0.000   -2.81e-05
aid:arw        0.0150      0.011      1.386      0.171      -0.007       0.037
aid:awt       -0.2500      0.011    -23.094      0.000      -0.272      -0.228
arw:awt       -0.2963      0.003   -109.466      0.000      -0.302      -0.291
==============================================================================
Omnibus:                      102.425   Durbin-Watson:                   1.994
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1875.983
Skew:                           4.018   Prob(JB):                         0.00
Kurtosis:                      25.321   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)
```

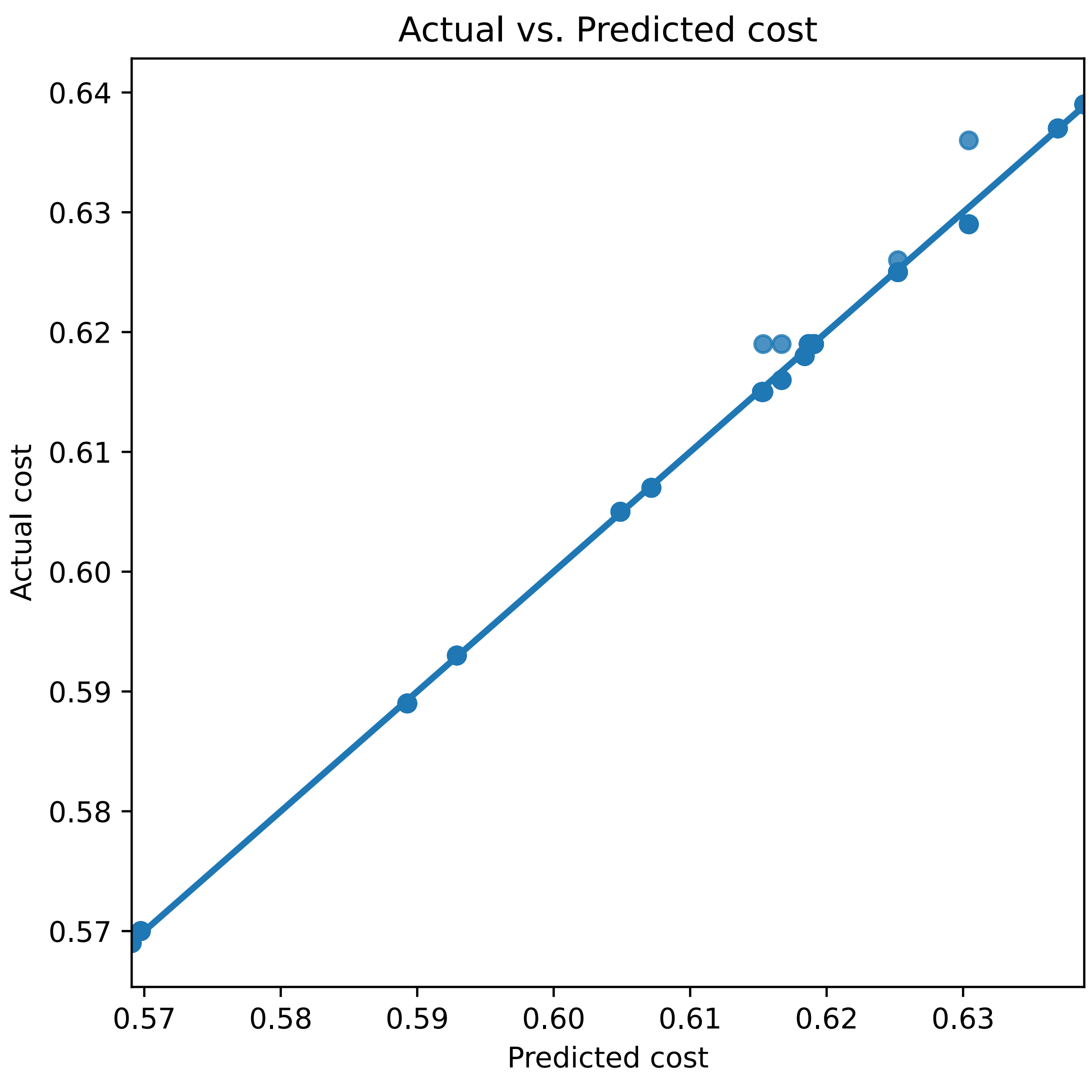
```
Drop alh:aid                        with p-value 0.818098
Drop aps:arw                        with p-value 0.643285
Drop alh:arw                        with p-value 0.485085
Drop aps                            with p-value 0.251449
Drop aid:arw                        with p-value 0.164261
['alh', 'aid', 'arw', 'awt', 'alh:aps', 'alh:awt', 'aps:aid', 'aps:awt', 'aid:awt', 'a
rw:awt']
```

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

```python
## 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:                       0.998
Model:                            OLS   Adj. R-squared:                  0.998
Method:                 Least Squares   F-statistic:                     3566.
Date:                Sat, 31 Jul 2021   Prob (F-statistic):           1.77e-89
Time:                        20:31:01   Log-Likelihood:                 448.21
No. Observations:                  80   AIC:                            -874.4
Df Residuals:                      69   BIC:                            -848.2
Df Model:                          10
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.2475      0.003     71.944      0.000       0.241       0.254
alh            0.0853      0.019      4.391      0.000       0.047       0.124
aid            0.4780      0.023     21.015      0.000       0.433       0.523
arw            0.3561      0.003    130.018      0.000       0.351       0.362
awt            0.3245      0.006     58.700      0.000       0.313       0.335
alh:aps        0.0012      0.000      4.432      0.000       0.001       0.002
alh:awt       -0.2104      0.009    -23.503      0.000      -0.228      -0.193
aps:aid       -0.0011      0.000     -3.489      0.001      -0.002      -0.000
aps:awt       -0.0003   6.63e-05     -4.179      0.000      -0.000      -0.000
aid:awt       -0.2500      0.011    -23.270      0.000      -0.271      -0.229
arw:awt       -0.2963      0.003   -110.300      0.000      -0.302      -0.291
==============================================================================
Omnibus:                       99.172   Durbin-Watson:                   1.988
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1596.490
Skew:                           3.889   Prob(JB):                         0.00
Kurtosis:                      23.456   Cond. No.                     1.62e+04
==============================================================================

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

```
[2] The condition number is large, 1.62e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```




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

```
Cost = 0.24752499999999666 + 0.08528056112224291 * model.X1 + 0.4780490981963954 * mod
el.X3 + 0.3561250000000017 * model.X4 + 0.32446325985304236 * model.X5 + 0.00115105210
4208572 * model.X1*model.X2 + -0.21041666666666942 * model.X1*model.X5 + -0.0010537742
150968782 * model.X2*model.X3 + -0.00027722110888844477 * model.X2*model.X5 + -0.250000
0000000011 * model.X3*model.X5 + -0.29625000000000157 * 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])
```

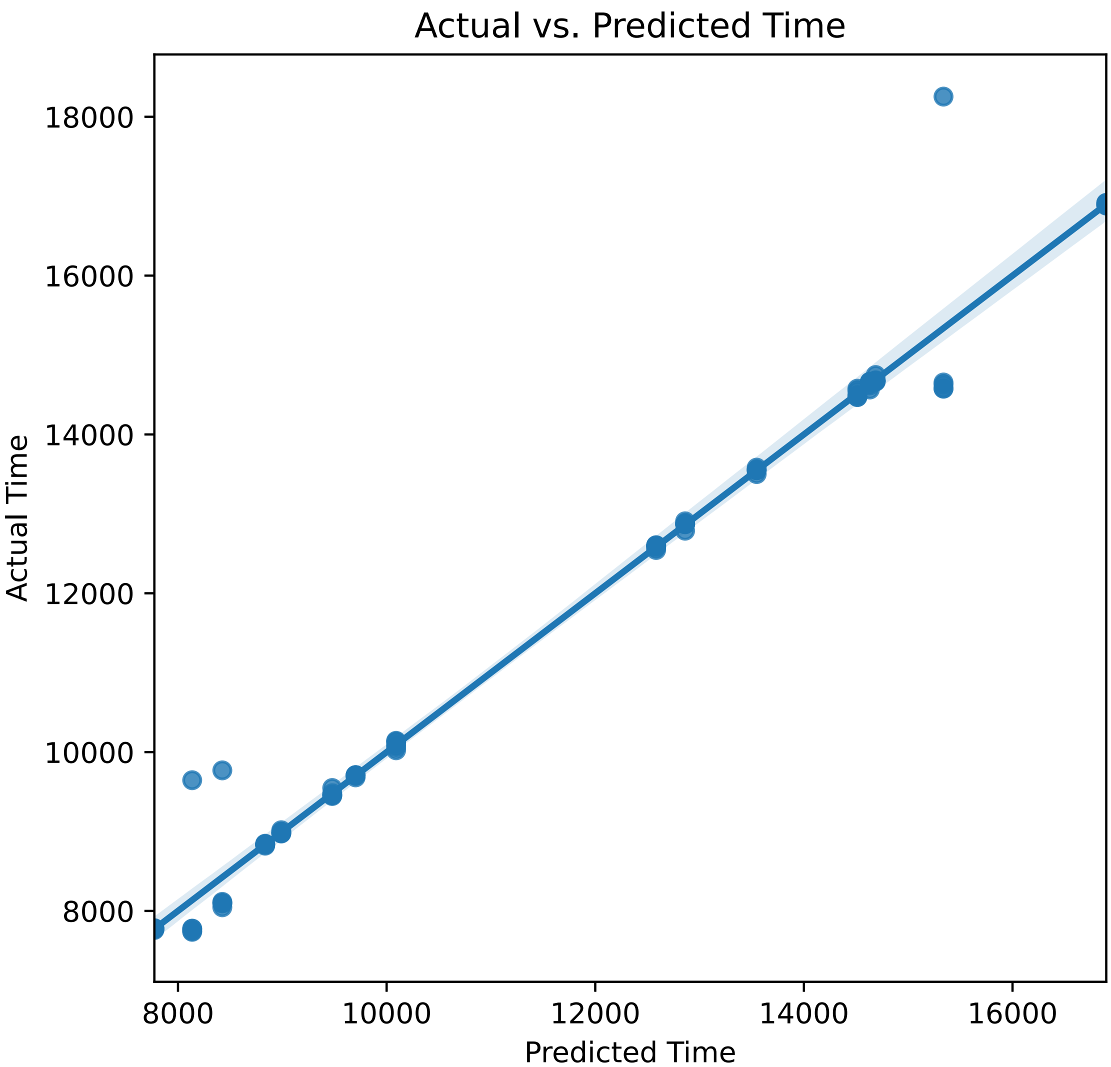
```
print(X[:5])
```

```
      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:                        20:31:02   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

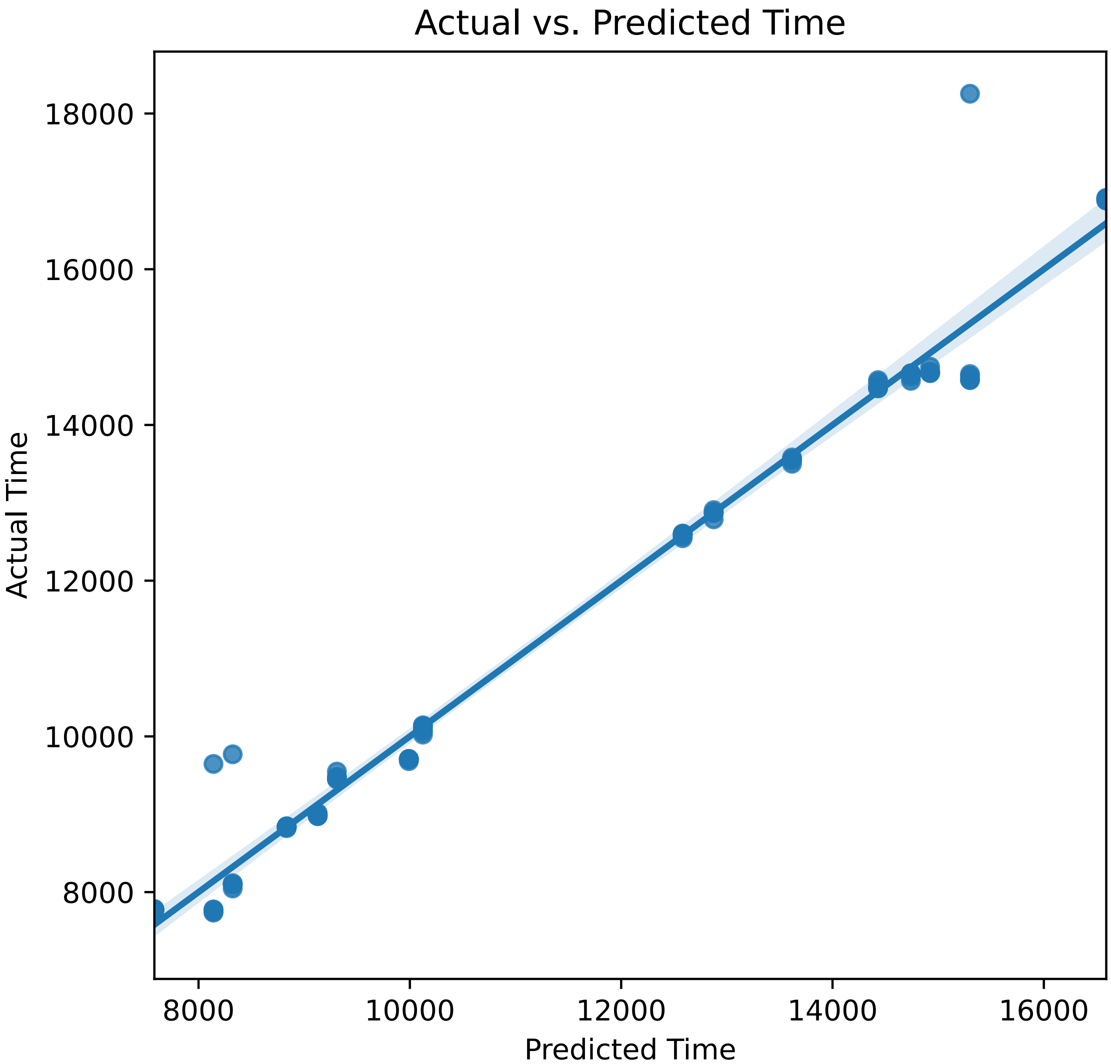
```
time_included = backward_regression(X,y,.05)
time_included.pop(0)
print(time_included)
```

```
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:                        20:31:05   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 =

```
0.24752499999999666 + 0.08528056112224291 * model.X1 + 0.4780490981963954 * model.X3 +
 0.3561250000000017 * model.X4 + 0.32446325985304236 * model.X5 + 0.001151052104208572
 * model.X1*model.X2 + -0.2104166666666942 * model.X1*model.X5 + -0.00105377421509687
82 * model.X2*model.X3 + -0.0002772211088844477 * model.X2*model.X5 + -0.2500000000000
011 * model.X3*model.X5 + -0.29625000000000157 * 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 == (0.24752499999999666 + 0.08528056112224291
* model.X1 + 0.4780490981963954 * model.X3 + 0.3561250000000017 * model.X4 + 0.3244632
5985304236 * model.X5 + 0.001151052104208572 * model.X1*model.X2 + -0.2104166666666694
2 * model.X1*model.X5 + -0.0010537742150968782 * model.X2*model.X3 + -0.00027722110888
```

```
44477 * model.X2*model.X5 + -0.2500000000000011 * model.X3*model.X5 + -0.2962500000000
0157 * 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)
```

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

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 ) = ( 0.27998582110954734 , 71.99995706607429 , 0.15000000239457
778 , 0.40000001103591754 , 0.8000000179710657 )
f1 = 0.5690547178020313
f2 = 9564.462829634651
( X1 , X2, X3, X4, X5 ) = ( 0.28000000999993796 , 72.00000071997286 , 0.14999999000058
92 , 0.3999999900070408 , 1.2000000119995038 )
f1 = 0.6048893778398489
f2 = 7582.113544009455
```

```python

```