CR6 Results Analysis C

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/cr6_cost_rework.txt', sep='\t')
df_time = pd.read_csv('https://raw.githubusercontent.com/wilsongis/3DP_Experiments/main/Data/cr6_time_rework.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', 'rep', 'time']
```

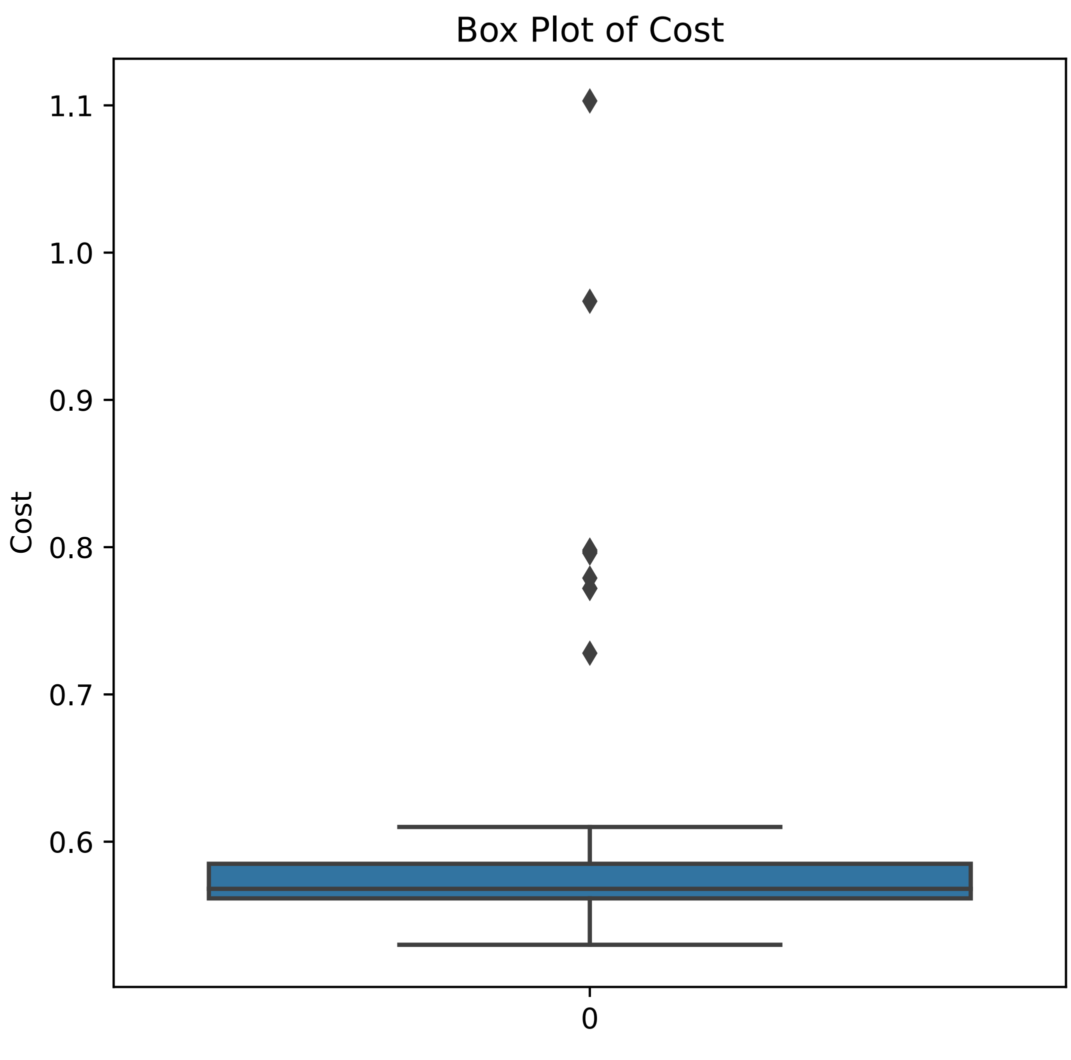
```python
display((Markdown("### Statistics for Cost")))
df_cost.cost.describe()
```

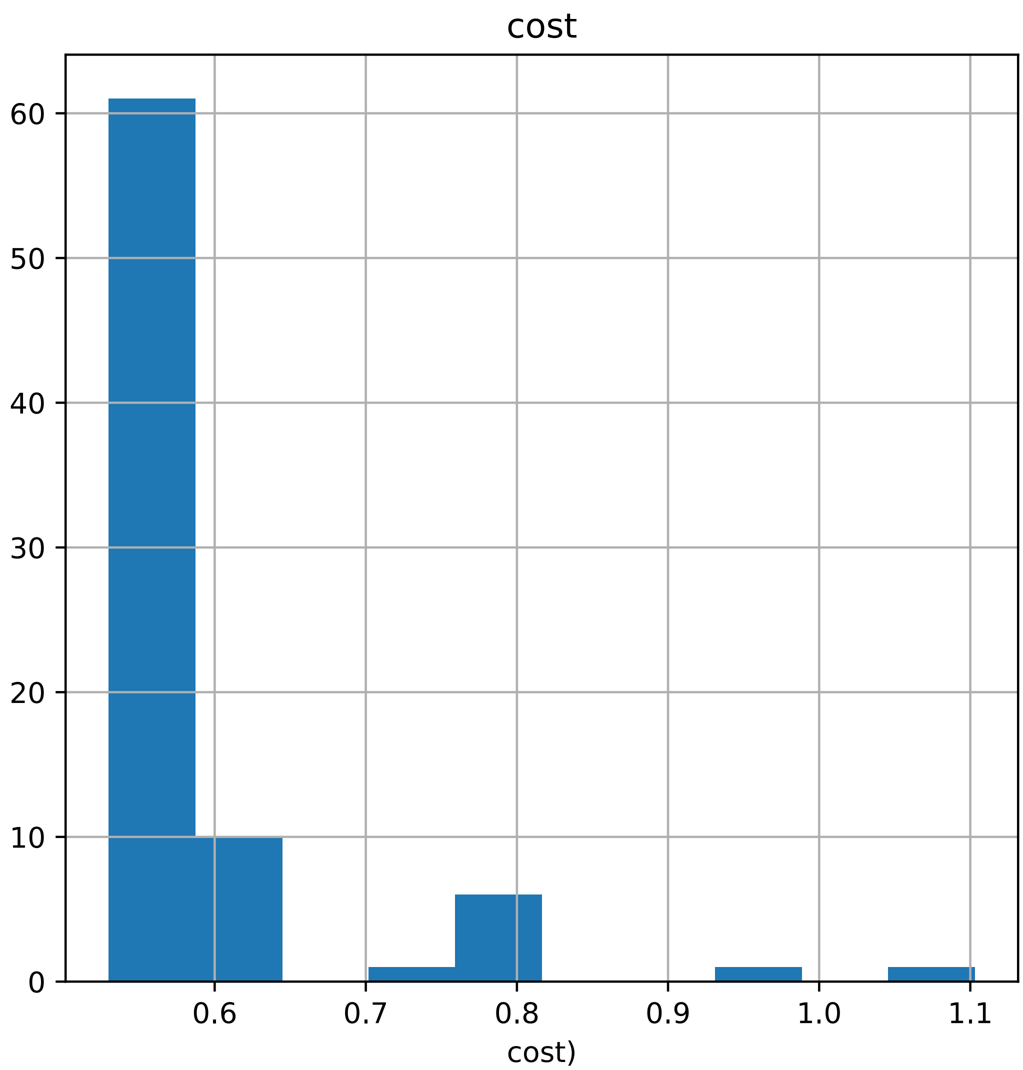
Statistics for Cost

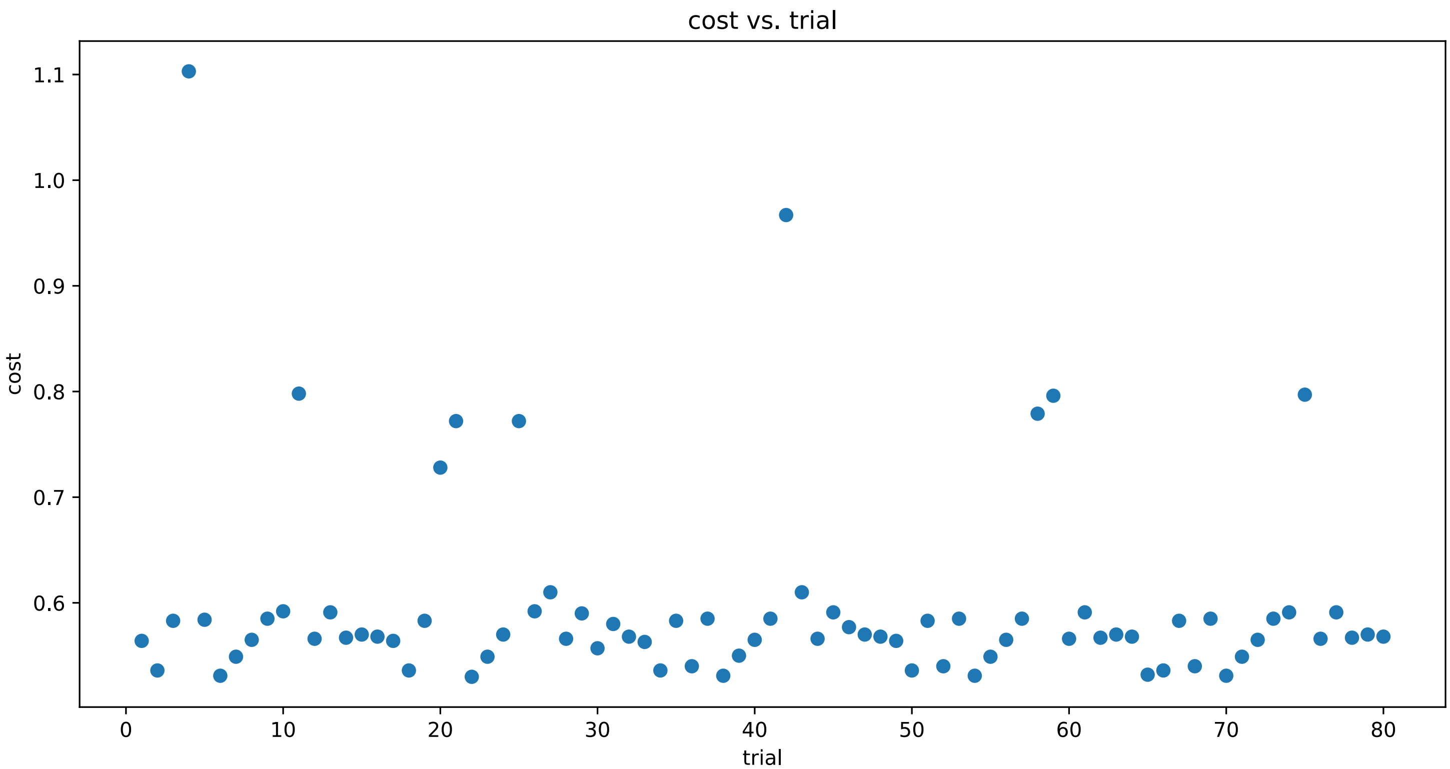
```
count    80.000000
mean      0.596712
std       0.095422
min       0.530000
25%       0.561500
50%       0.568000
75%       0.585000
max       1.103000
Name: cost, dtype: float64
```

```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
60
50
40
30
20
10
0
0.6
0.7
0.8
0.9
1.0
1.1
cost)




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

Statistics for Time

```
count      80.000000
mean    12685.900000
std      3360.237079
min      8480.000000
25%      9464.250000
50%     12989.000000
75%     15482.500000
max     18098.000000
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

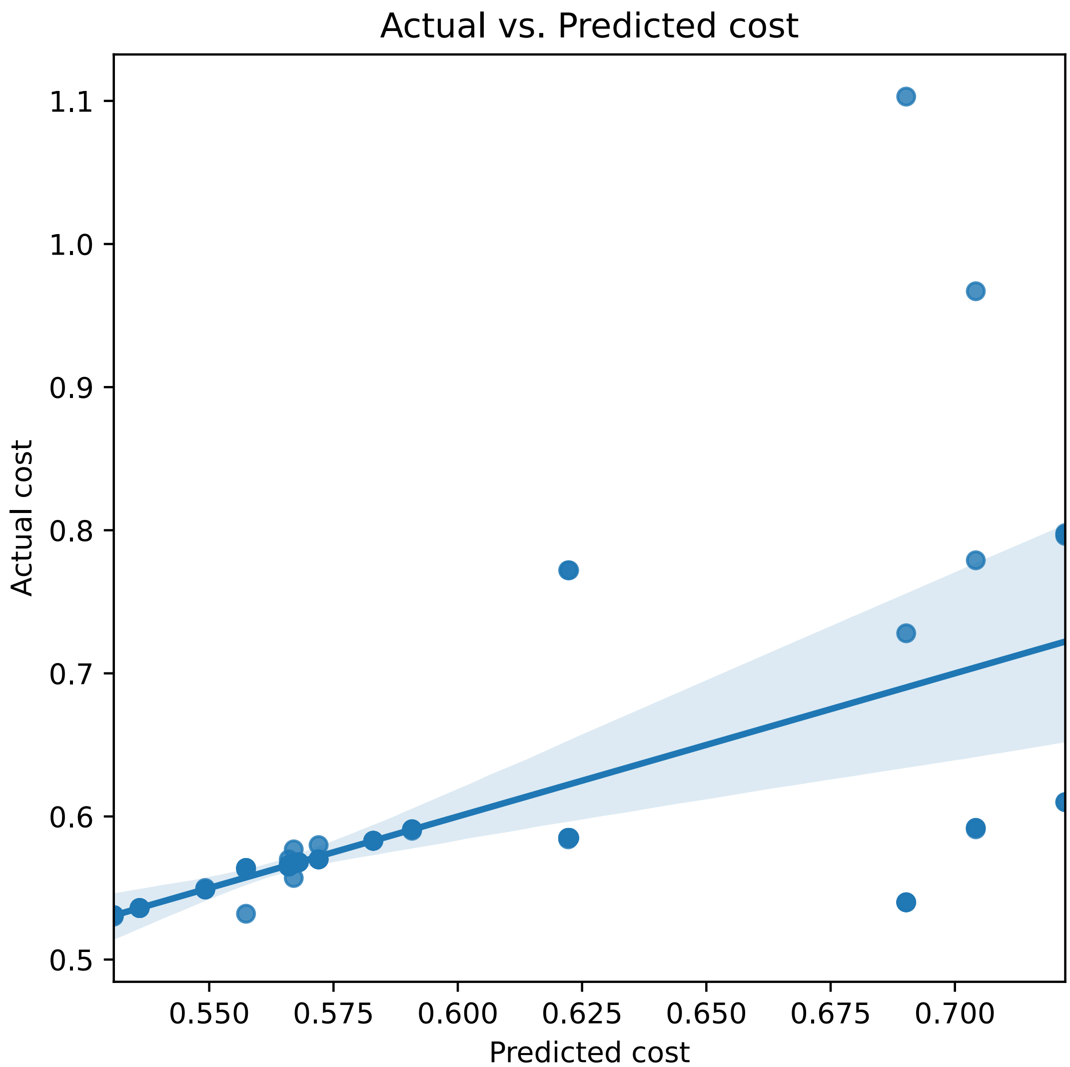
```
f = 'cost ~ (alh+aps+aid+arw+awt)**2'
y, X = patsy.dmatrices(f, df_cost, return_type='dataframe')
print(y[:5])
print(X[:5])
```

```
    cost
0  0.564
1  0.536
2  0.583
3  1.103
4  0.584
   Intercept   alh   aps   aid  arw  awt  alh:aps  alh:aid  alh:arw  alh:awt  \
0        1.0  0.16  50.0  0.25  0.4  0.8      8.0    0.040    0.064    0.128
1        1.0  0.28  50.0  0.25  0.4  1.2     14.0    0.070    0.112    0.336
2        1.0  0.16  60.0  0.25  0.4  1.2      9.6    0.040    0.064    0.192
3        1.0  0.28  60.0  0.25  0.4  0.8     16.8    0.070    0.112    0.224
4        1.0  0.16  50.0  0.15  0.4  1.2      8.0    0.024    0.064    0.192

   aps:aid  aps:arw  aps:awt  aid:arw  aid:awt  arw:awt
0     12.5     20.0     40.0     0.10     0.20     0.32
1     12.5     20.0     60.0     0.10     0.30     0.48
2     15.0     24.0     72.0     0.10     0.30     0.48
3     15.0     24.0     48.0     0.10     0.20     0.32
4      7.5     20.0     60.0     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:                       0.373
Model:                            OLS   Adj. R-squared:                  0.226
Method:                 Least Squares   F-statistic:                     2.534
Date:                Sat, 31 Jul 2021   Prob (F-statistic):            0.00518
Time:                        18:19:29   Log-Likelihood:                 93.594
No. Observations:                  80   AIC:                            -155.2
Df Residuals:                      64   BIC:                            -117.1
Df Model:                          15
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     -0.9161      0.906     -1.012      0.316      -2.725       0.893
alh            0.8129      2.052      0.396      0.693      -3.287       4.912
aps            0.0107      0.015      0.713      0.479      -0.019       0.041
aid           -0.6871      2.444     -0.281      0.780      -5.570       4.196
arw            0.9958      0.624      1.597      0.115      -0.250       2.241
awt            1.6889      0.595      2.840      0.006       0.501       2.877
alh:aps        0.0039      0.031      0.124      0.902      -0.059       0.066
alh:aid        2.3708      3.129      0.758      0.451      -3.881       8.623
alh:arw       -0.5906      0.782     -0.755      0.453      -2.154       0.972
alh:awt       -1.2406      0.782     -1.586      0.118      -2.804       0.322
aps:aid        0.0493      0.038      1.311      0.194      -0.026       0.124
aps:arw       -0.0124      0.009     -1.319      0.192      -0.031       0.006
aps:awt       -0.0129      0.009     -1.378      0.173      -0.032       0.006
aid:arw        1.3663      0.939      1.455      0.150      -0.509       3.242
aid:awt       -2.8437      0.939     -3.029      0.004      -4.719      -0.968
arw:awt       -0.3709      0.235     -1.580      0.119      -0.840       0.098
==============================================================================
Omnibus:                       65.477   Durbin-Watson:                   2.187
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              524.930
Skew:                           2.338   Prob(JB):                    1.03e-114
Kurtosis:                      14.646   Cond. No.                     3.13e+04
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly speci
fied.
[2] The condition number is large, 3.13e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```



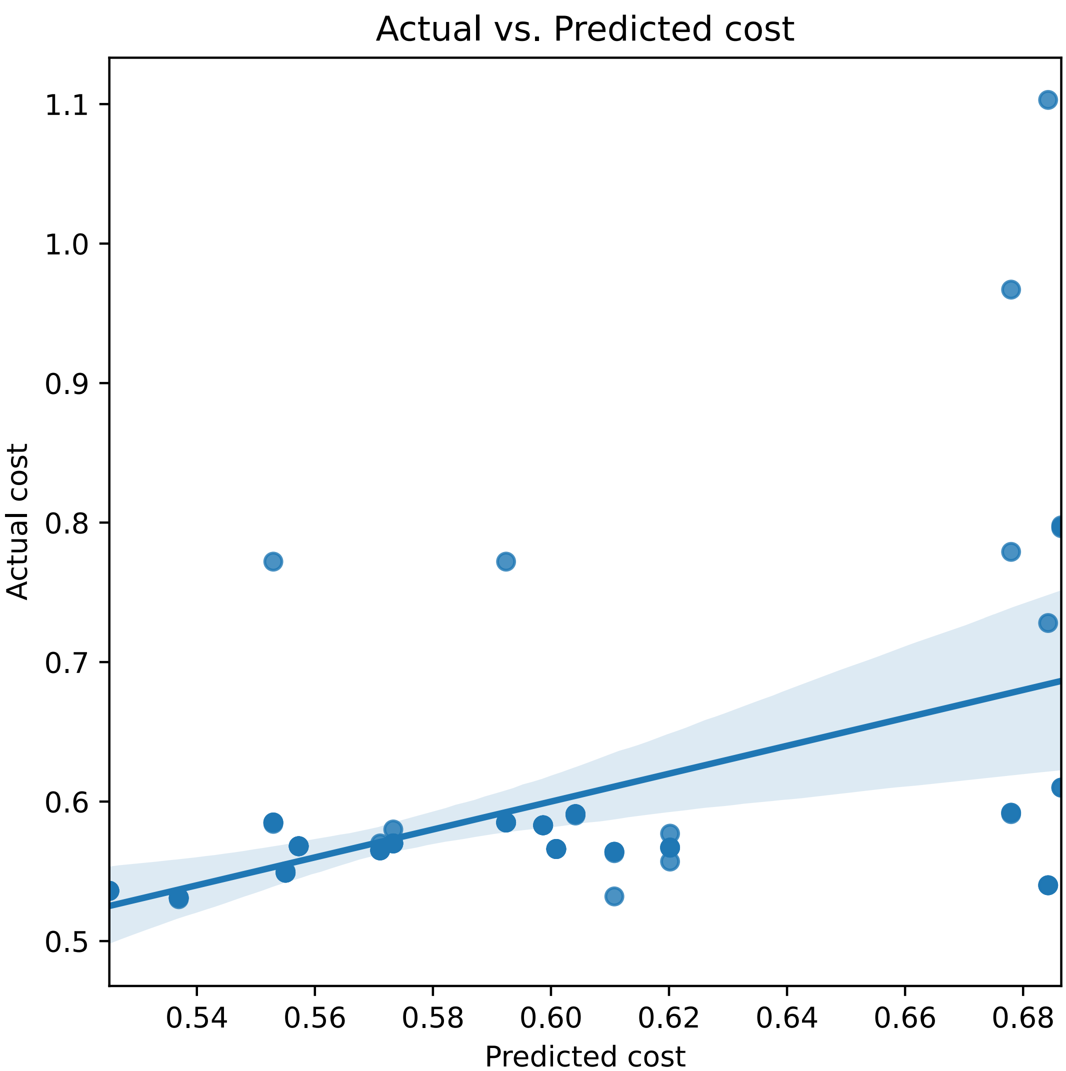
Reduced Cost Model

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

```
Drop alh:aps                           with p-value 0.901841
Drop aid                               with p-value 0.77787
Drop alh:aid                           with p-value 0.475609
Drop alh:arw                           with p-value 0.444672
Drop aps                               with p-value 0.294638
Drop aps:awt                           with p-value 0.351045
Drop alh                               with p-value 0.146827
Drop alh:awt                           with p-value 0.362181
Drop aid:arw                           with p-value 0.11887
Drop arw:awt                           with p-value 0.116793
['arw', 'awt', 'aps:aid', 'aps:arw', 'aid:awt']
```

```python
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:                       0.266
Model:                            OLS   Adj. R-squared:                  0.216
Method:                 Least Squares   F-statistic:                     5.364
Date:                Sat, 31 Jul 2021   Prob (F-statistic):           0.000291
Time:                        18:19:31   Log-Likelihood:                 87.314
No. Observations:                  80   AIC:                            -162.6
Df Residuals:                      74   BIC:                            -148.3
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0222      0.164      0.135      0.893      -0.305       0.349
arw            0.9805      0.288      3.408      0.001       0.407       1.554
awt            0.4210      0.160      2.631      0.010       0.102       0.740
aps:aid        0.0554      0.014      3.950      0.000       0.027       0.083
aps:arw       -0.0162      0.005     -3.149      0.002      -0.027      -0.006
aid:awt       -2.5398      0.765     -3.322      0.001      -4.063      -1.016
==============================================================================
Omnibus:                       64.265   Durbin-Watson:                   2.150
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              376.325
Skew:                           2.458   Prob(JB):                     1.91e-82
Kurtosis:                      12.420   Cond. No.                     3.20e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly speci
fied.
[2] The condition number is large, 3.2e+03. 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 = 0.022179997348888314 + 0.9805362539766707 * model.X4 + 0.4210264448568395 * mod
el.X5 + 0.055398409331918294 * model.X2*model.X3 + -0.016249522799575436 * model.X2*mo
del.X4 + -2.5398197242841984 * model.X3*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])
```

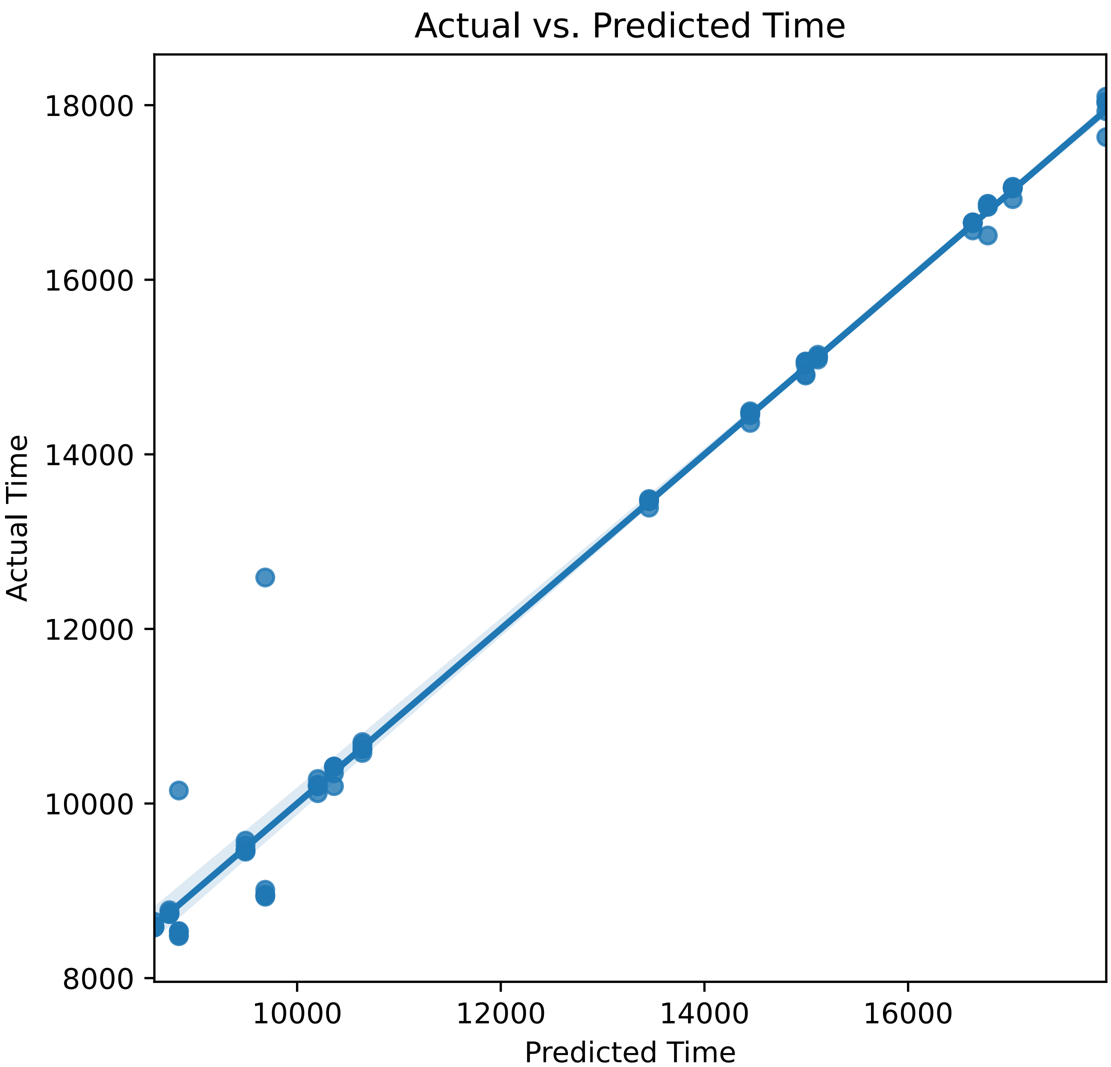
```
      time
0  18098.0
```

```
1   8741.0
2  14493.0
3  10281.0
4  14914.0
   Intercept   alh   aps   aid  arw  awt  alh:aps  alh:aid  alh:arw  alh:awt  \
0        1.0  0.16  50.0  0.25  0.4  0.8      8.0    0.040    0.064    0.128
1        1.0  0.28  50.0  0.25  0.4  1.2     14.0    0.070    0.112    0.336
2        1.0  0.16  60.0  0.25  0.4  1.2      9.6    0.040    0.064    0.192
3        1.0  0.28  60.0  0.25  0.4  0.8     16.8    0.070    0.112    0.224
4        1.0  0.16  50.0  0.15  0.4  1.2      8.0    0.024    0.064    0.192

   aps:aid  aps:arw  aps:awt  aid:arw  aid:awt  arw:awt
0     12.5     20.0     40.0     0.10     0.20     0.32
1     12.5     20.0     60.0     0.10     0.30     0.48
2     15.0     24.0     72.0     0.10     0.30     0.48
3     15.0     24.0     48.0     0.10     0.20     0.32
4      7.5     20.0     60.0     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:                       0.985
Model:                            OLS   Adj. R-squared:                  0.982
Method:                 Least Squares   F-statistic:                     287.3
Date:                Sat, 31 Jul 2021   Prob (F-statistic):           1.18e-52
Time:                        18:19:33   Log-Likelihood:                -593.61
No. Observations:                  80   AIC:                             1219.
Df Residuals:                      64   BIC:                             1257.
Df Model:                          15
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept   3.864e+04   4869.987      7.934      0.000    2.89e+04    4.84e+04
alh        -9.356e+04    1.1e+04     -8.477      0.000   -1.16e+05   -7.15e+04
aps          -99.8583     81.064     -1.232      0.223    -261.803      62.086
```

```
aid          6098.6667   1.31e+04      0.464      0.644   -2.02e+04    3.24e+04
arw          2199.0000   3353.358      0.656      0.514   -4500.103    8898.103
awt         -8271.6667   3197.704     -2.587      0.012   -1.47e+04   -1883.518
alh:aps       302.8333    168.300      1.799      0.077     -33.385     639.052
alh:aid     -4.944e+04   1.68e+04     -2.938      0.005   -8.31e+04   -1.58e+04
alh:arw      9968.7500   4207.505      2.369      0.021    1563.292    1.84e+04
alh:awt       2.89e+04   4207.505      6.868      0.000    2.05e+04    3.73e+04
aps:aid       307.9000    201.960      1.525      0.132     -95.562     711.362
aps:arw      -138.6250     50.490     -2.746      0.008    -239.490     -37.760
aps:awt       -20.9250     50.490     -0.414      0.680    -121.790      79.940
aid:arw      5980.0000   5049.006      1.184      0.241   -4106.550    1.61e+04
aid:awt     -1.326e+04   5049.006     -2.626      0.011   -2.33e+04   -3173.450
arw:awt      1607.5000   1262.252      1.274      0.207    -914.137    4129.137
==============================================================================
Omnibus:                      117.962   Durbin-Watson:                   1.969
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             3764.993
Skew:                           4.734   Prob(JB):                         0.00
Kurtosis:                      35.247   Cond. No.                     3.13e+04
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly speci
fied.
[2] The condition number is large, 3.13e+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)
```

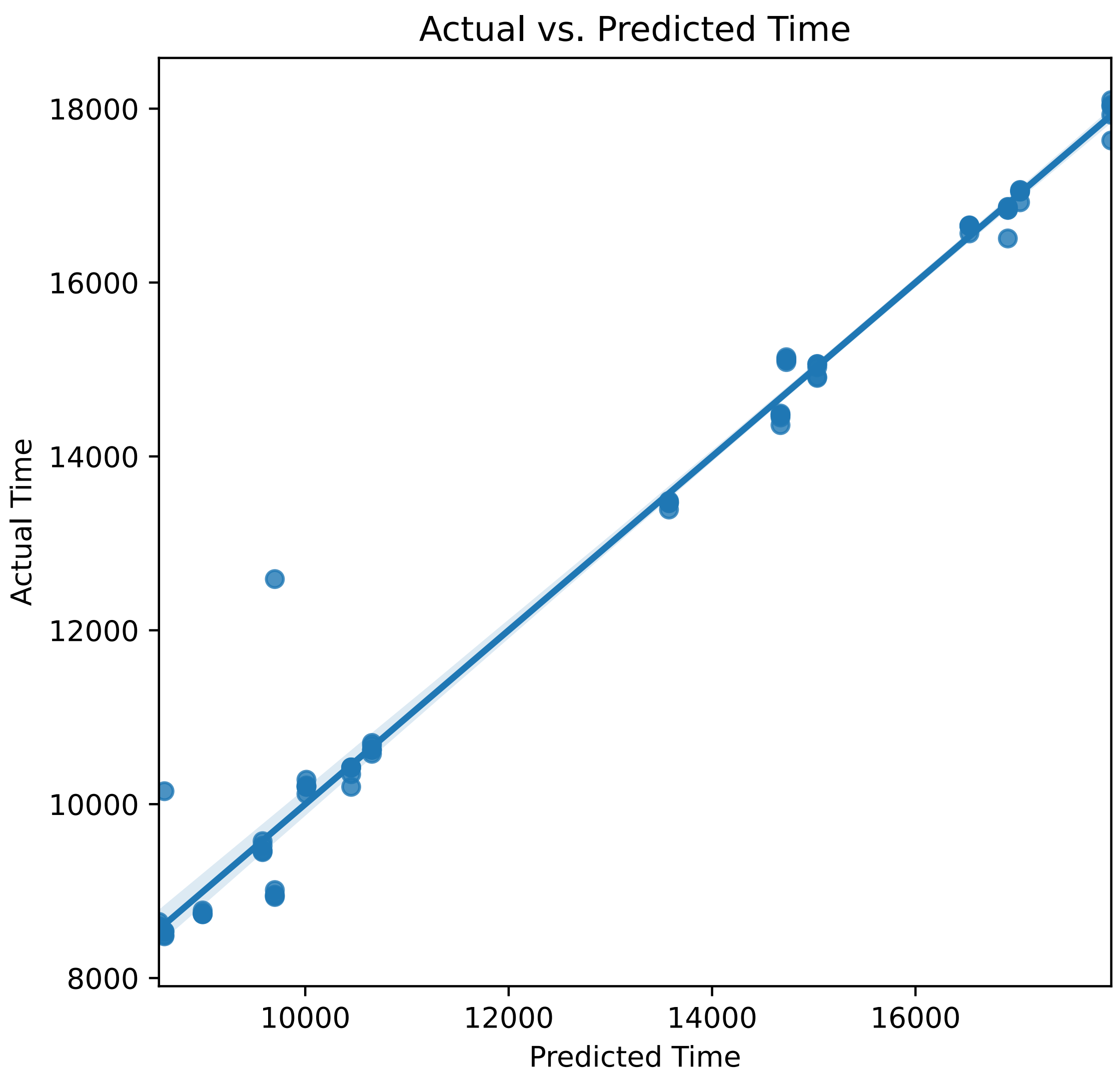
```
Drop aps:awt                            with p-value 0.679939
Drop aid                                with p-value 0.642097
Drop arw                                with p-value 0.529019
Drop aid:arw                            with p-value 0.112705
Drop alh:aps                            with p-value 0.0748668
Drop arw:awt                            with p-value 0.0727373
['alh', 'aps', 'awt', 'alh:aid', 'alh:arw', 'alh:awt', 'aps:aid', 'aps:arw', 'aid: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:                       0.983
Model:                            OLS   Adj. R-squared:                  0.981
Method:                 Least Squares   F-statistic:                     452.5
Date:                Sat, 31 Jul 2021   Prob (F-statistic):           2.41e-58
Time:                        18:19:35   Log-Likelihood:                -599.37
No. Observations:                  80   AIC:                             1219.
Df Residuals:                      70   BIC:                             1243.
Df Model:                           9
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       3.938e+04   1156.066     34.066      0.000     3.71e+04    4.17e+04
alh         -7.9e+04   6038.177    -13.083      0.000     -9.1e+04    -6.7e+04
aps        -129.4774     25.189     -5.140      0.000     -179.715     -79.240
awt       -8759.9115   1371.420     -6.387      0.000    -1.15e+04   -6024.701
alh:aid   -4.575e+04    1.66e+04     -2.764      0.007    -7.88e+04   -1.27e+04
alh:arw    1.223e+04   4103.145      2.980      0.004     4042.360    2.04e+04
alh:awt     2.89e+04   4323.322      6.684      0.000     2.03e+04    3.75e+04
aps:aid     440.7227    102.894      4.283      0.000      235.507     645.938
aps:arw     -57.3703     16.942     -3.386      0.001      -91.160     -23.580
aid:awt   -1.175e+04   4766.692     -2.465      0.016     -2.13e+04   -2243.783
==============================================================================
Omnibus:                      103.225   Durbin-Watson:                   1.901
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             2123.126
Skew:                           3.995   Prob(JB):                         0.00
Kurtosis:                      26.939   Cond. No.                     2.14e+04
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly speci
fied.
[2] The condition number is large, 2.14e+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 = 39383.12499999856 + -78995.06592013844 * model.X1 + -129.4773731249602 * model.
X2 + -8759.911472760796 * model.X5 + -45752.146814405125 * model.X1*model.X3 + 12225.8
25471698428 * model.X1*model.X4 + 28897.916666666522 * model.X1*model.X5 + 440.7227146
814413 * model.X2*model.X3 + -57.37028301886791 * model.X2*model.X4 + -11750.650969529
073 * model.X3*model.X5
```

Equations

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

```
print("------")

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

Cost =

```
0.022179997348888314 + 0.9805362539766707 * model.X4 + 0.4210264448568395 * model.X5 +
 0.055398409331918294 * model.X2*model.X3 + -0.016249522799575436 * model.X2*model.X4
+ -2.5398197242841984 * model.X3*model.X5
------
```

Time =

```
39383.12499999856 + -78995.06592013844 * model.X1 + -129.4773731249602 * model.X2 + -8
759.911472760796 * model.X5 + -45752.146814405125 * model.X1*model.X3 + 12225.82547169
8428 * model.X1*model.X4 + 28897.916666666522 * model.X1*model.X5 + 440.7227146814413
* model.X2*model.X3 + -57.37028301886791 * model.X2*model.X4 + -11750.650969529073 * m
odel.X3*model.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 <= 60)
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 >= 50)
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 == (0.022179997348888314 + 0.9805362539766707
* model.X4 + 0.4210264448568395 * model.X5 + 0.055398409331918294 * model.X2*model.X3
```

```
+ -0.016249522799575436 * model.X2*model.X4 + -2.5398197242841984 * model.X3*model.X5)
)
model.C_f2 = Constraint(expr = model.f2 == (39383.12499999856 + -78995.06592013844 * m
odel.X1 + -129.4773731249602 * model.X2 + -8759.911472760796 * model.X5 + -45752.14681
4405125 * model.X1*model.X3 + 12225.825471698428 * model.X1*model.X4 + 28897.916666666
522 * model.X1*model.X5 + 440.7227146814413 * model.X2*model.X3 + -57.37028301886791 *
 model.X2*model.X4 + -11750.650969529073 * model.X3*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 ) = ( 0.22777655237840858 , 50.0000009450316 , 0.14999999321143795 , 0.4000000038314072 , 0.8000000479253332 )
f1 = 0.5369349015047474
f2 = 13472.566567809765
( X1 , X2, X3, X4, X5 ) = ( 0.2800000099999455 , 60.00000059996157 , 0.2500000099946454 4 , 0.800000009867982 , 1.200000011999305 )
f1 = 0.6008938634299353
f2 = 8561.47631366192
```

```python

```