CR6 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_l
ows[col]))
            # don't need to cast to float here, if either are not a float except
ion will 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: ac
tual_highs[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, wors
t_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_power.txt', sep='\t')
df_time = pd.read_csv('https://raw.githubusercontent.com/wilsongis/3DP_Experiments/main/Data/cr6_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', 'rep', 'time']
```

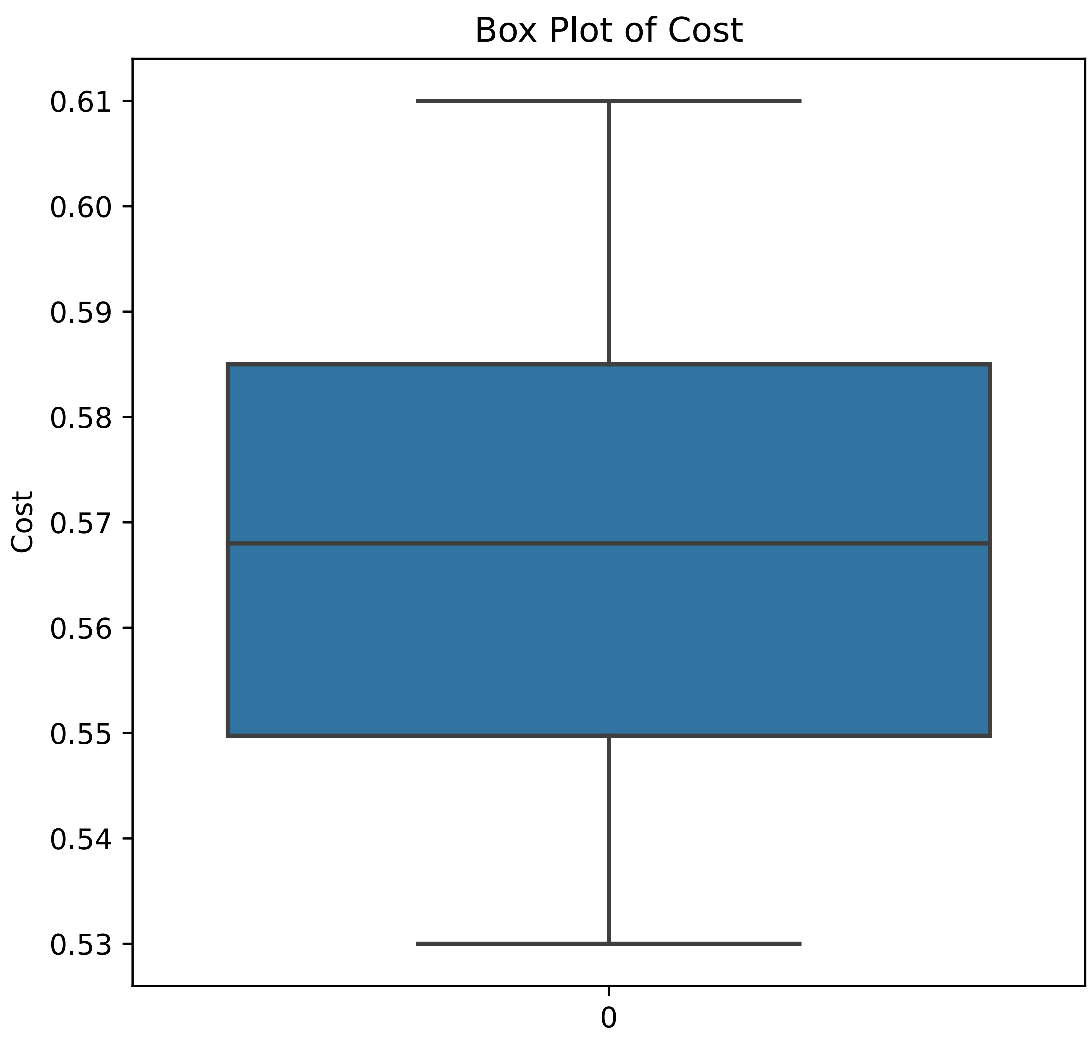
```python
display((Markdown("### Statistics for Cost")))
df_cost.cost.describe()
```

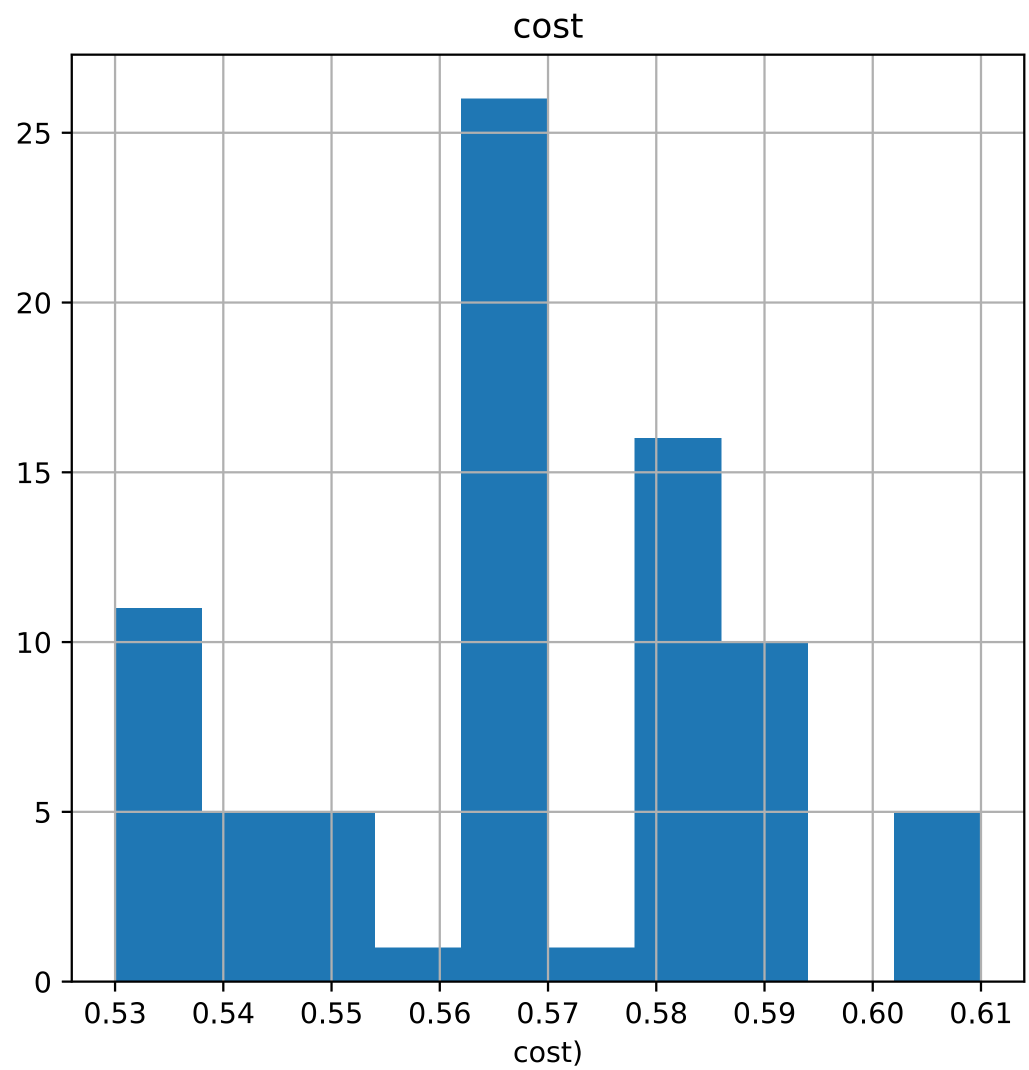
Statistics for Cost

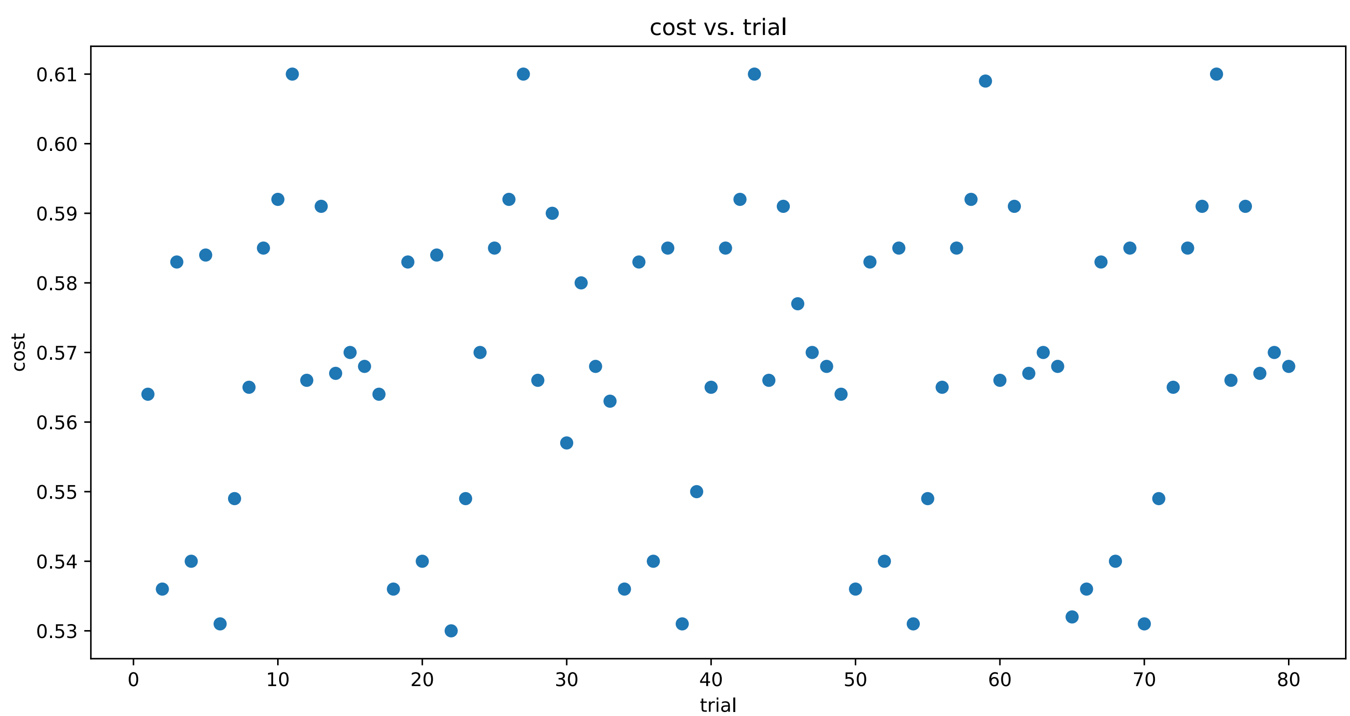
```
count    80.000000
mean      0.568588
std       0.021820
min       0.530000
25%       0.549750
50%       0.568000
75%       0.585000
max       0.610000
Name: cost, dtype: float64
```

```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
0
5
10
15
20
25
0.53
0.54
0.55
0.56
0.57
0.58
0.59
0.60
0.61
cost)


cost vs. trial
cost
0.53
0.54
0.55
0.56
0.57
0.58
0.59
0.60
0.61
0
10
20
30
40
50
60
70
80
trial

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

Statistics for Time

```
count       80.00000
mean     12681.40000
std       3360.13591
min       8480.00000
25%       9464.25000
50%      12989.00000
75%      15460.75000
max      18098.00000
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])
```

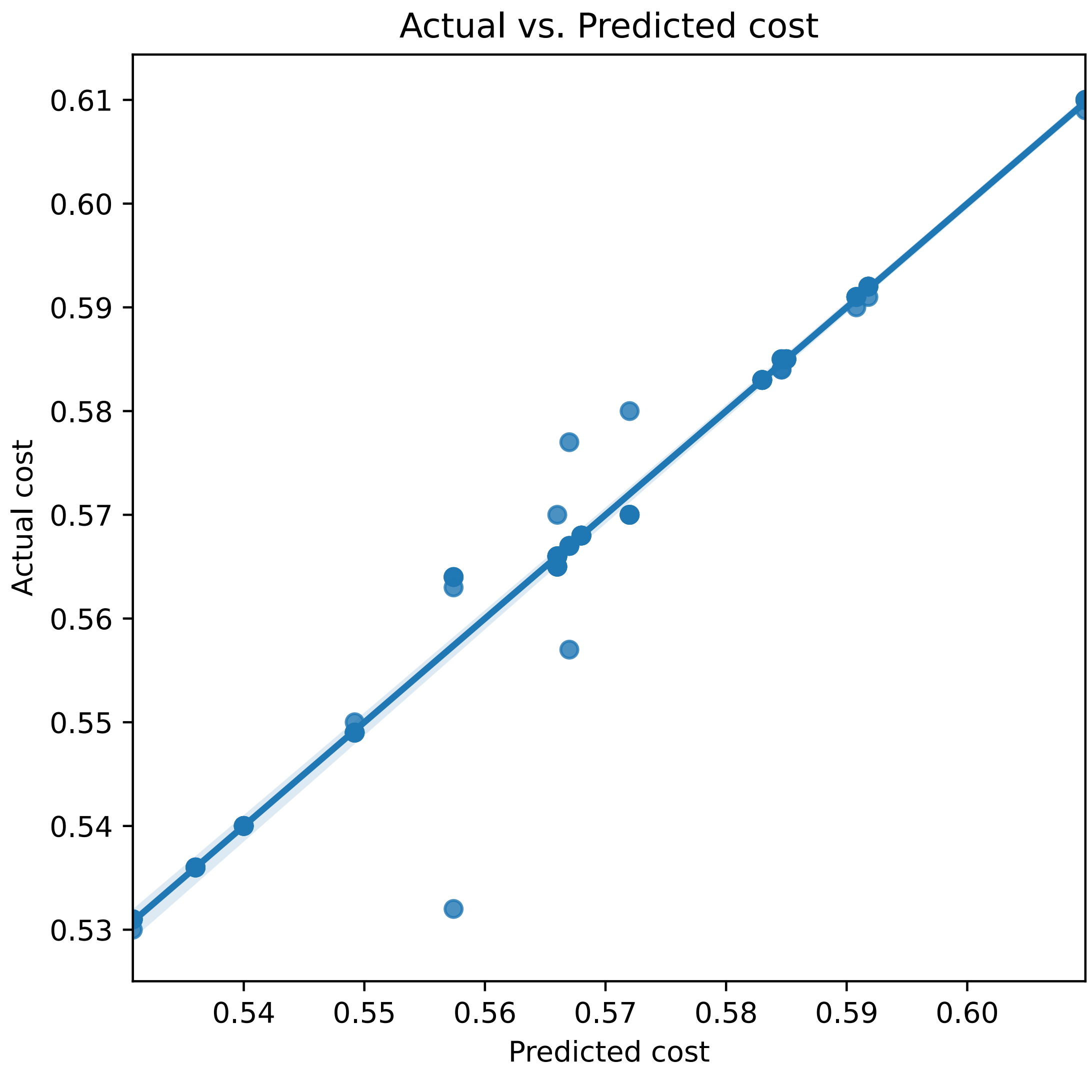
```
    cost
0  0.564
```

```
1  0.536
2  0.583
3  0.540
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:                       0.970
Model:                            OLS   Adj. R-squared:                  0.963
Method:                 Least Squares   F-statistic:                     140.0
Date:                Sat, 31 Jul 2021   Prob (F-statistic):           6.42e-43
Time:                        20:28:12   Log-Likelihood:                 333.82
No. Observations:                  80   AIC:                            -635.6
Df Residuals:                      64   BIC:                            -597.5
Df Model:                          15
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      0.4577      0.045     10.181      0.000       0.368       0.548
```

```
alh            -0.2758      0.102     -2.707      0.009     -0.479     -0.072
aps            -0.0024      0.001     -3.156      0.002     -0.004     -0.001
aid            -0.1823      0.121     -1.502      0.138     -0.425      0.060
arw             0.3551      0.031     11.471      0.000      0.293      0.417
awt             0.2113      0.030      7.159      0.000      0.152      0.270
alh:aps         0.0038      0.002      2.440      0.017      0.001      0.007
alh:aid        -0.7625      0.155     -4.908      0.000     -1.073     -0.452
alh:arw         0.1906      0.039      4.908      0.000      0.113      0.268
alh:awt        -0.0677      0.039     -1.743      0.086     -0.145      0.010
aps:aid         0.0117      0.002      6.248      0.000      0.008      0.015
aps:arw        -0.0030      0.000     -6.463      0.000     -0.004     -0.002
aps:awt         0.0011      0.000      2.440      0.017      0.000      0.002
aid:arw         0.4313      0.047      9.252      0.000      0.338      0.524
aid:awt        -0.4987      0.047    -10.700      0.000     -0.592     -0.406
arw:awt        -0.2541      0.012    -21.802      0.000     -0.277     -0.231
==============================================================================
Omnibus:                       98.343   Durbin-Watson:                   2.123
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             2409.437
Skew:                          -3.570   Prob(JB):                         0.00
Kurtosis:                      28.920   Cond. No.                     3.13e+04
==============================================================================

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

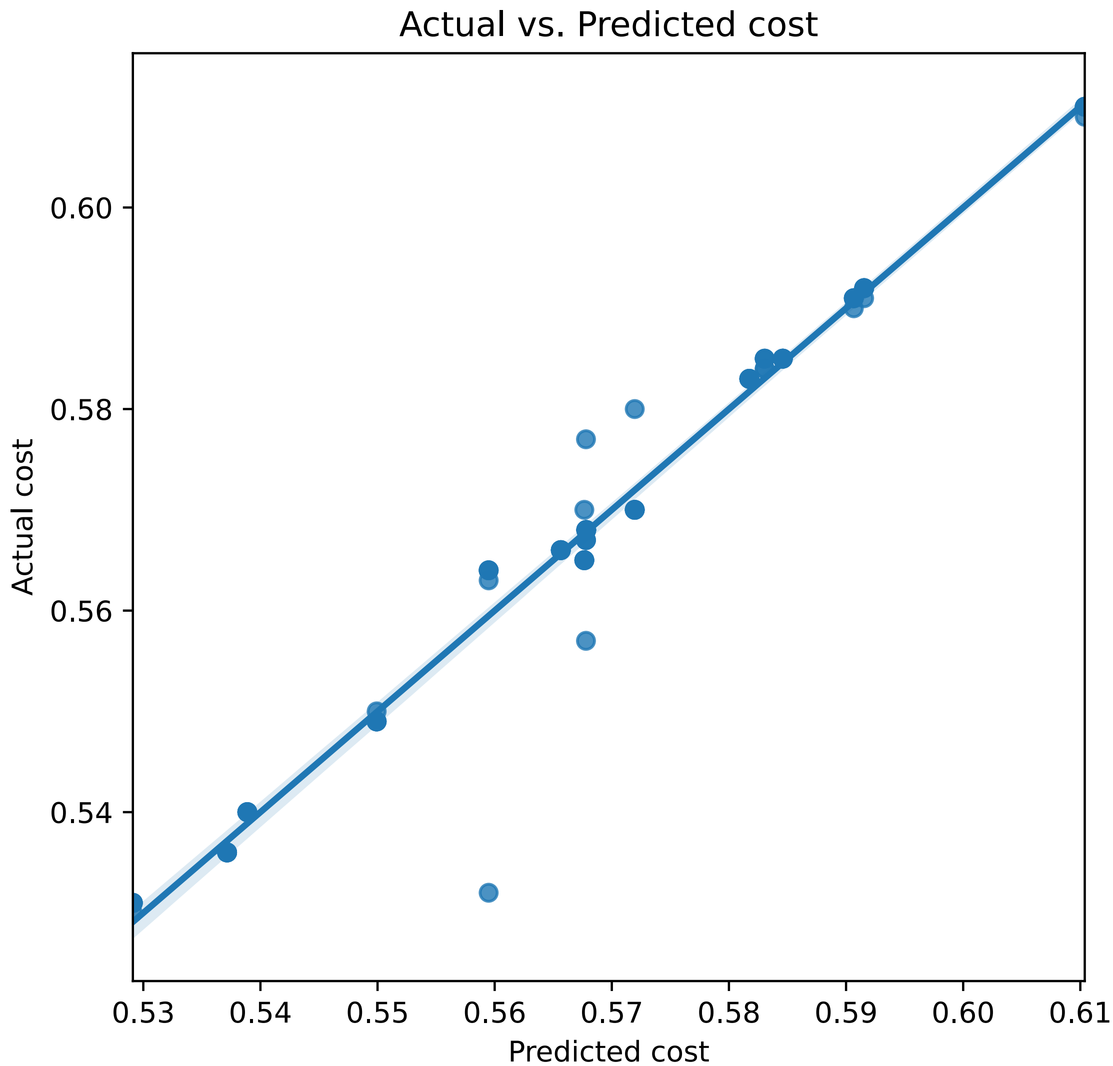
```
Drop aid                                with p-value 0.138052
Drop alh:awt                            with p-value 0.089003
['alh', 'aps', 'arw', 'awt', 'alh:aps', 'alh:aid', 'alh:arw', 'aps:aid', 'aps:ar
w', 'aps:awt', 'aid:arw', 'aid:awt', 'arw:awt']
```

```
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:                       0.968
Model:                            OLS   Adj. R-squared:                  0.962
Method:                 Least Squares   F-statistic:                     153.5
Date:                Sat, 31 Jul 2021   Prob (F-statistic):           5.23e-44
Time:                        20:28:14   Log-Likelihood:                 330.64
No. Observations:                  80   AIC:                            -633.3
Df Residuals:                      66   BIC:                            -599.9
Df Model:                          13
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.4362      0.038     11.546      0.000       0.361       0.512
alh           -0.3304      0.096     -3.438      0.001      -0.522      -0.139
aps           -0.0019      0.001     -2.715      0.008      -0.003      -0.000
arw            0.3584      0.032     11.324      0.000       0.295       0.422
awt            0.2018      0.029      7.027      0.000       0.144       0.259
alh:aps        0.0038      0.002      2.382      0.020       0.001       0.007
alh:aid       -0.8282      0.153     -5.422      0.000      -1.133      -0.523
alh:arw        0.1906      0.040      4.789      0.000       0.111       0.270
aps:aid        0.0093      0.001      9.088      0.000       0.007       0.011
aps:arw       -0.0030      0.000     -6.307      0.000      -0.004      -0.002
aps:awt        0.0011      0.000      2.382      0.020       0.000       0.002
aid:arw        0.4151      0.046      8.932      0.000       0.322       0.508
aid:awt       -0.5256      0.044    -11.920      0.000      -0.614      -0.438
arw:awt       -0.2541      0.012    -21.278      0.000      -0.278      -0.230
==============================================================================
Omnibus:                      112.032   Durbin-Watson:                   2.091
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             3244.806
Skew:                          -4.374   Prob(JB):                         0.00
Kurtosis:                      32.949   Cond. No.                     2.86e+04
==============================================================================

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

```
[2] The condition number is large, 2.86e+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 = 0.43619166666666676 + -0.3303957650272835 * model.X1 + -0.001888414207650
4296 * model.X2 + 0.3583517213114778 * model.X4 + 0.2018153688524602 * model.X5
+ 0.0037916666666636 * model.X1*model.X2 + -0.8282295081967748 * model.X1*model
.X3 + 0.19062500000000449 * model.X1*model.X4 + 0.009283737704918856 * model.X2*
model.X3 + -0.0030125000000000516 * model.X2*model.X4 + 0.0011375000000000004 * mo
del.X2*model.X5 + 0.4151163934426205 * model.X3*model.X4 + -0.5256393442622931 *
 model.X3*model.X5 + -0.25406250000000025 * model.X4*model.X5
```

Time Analysis

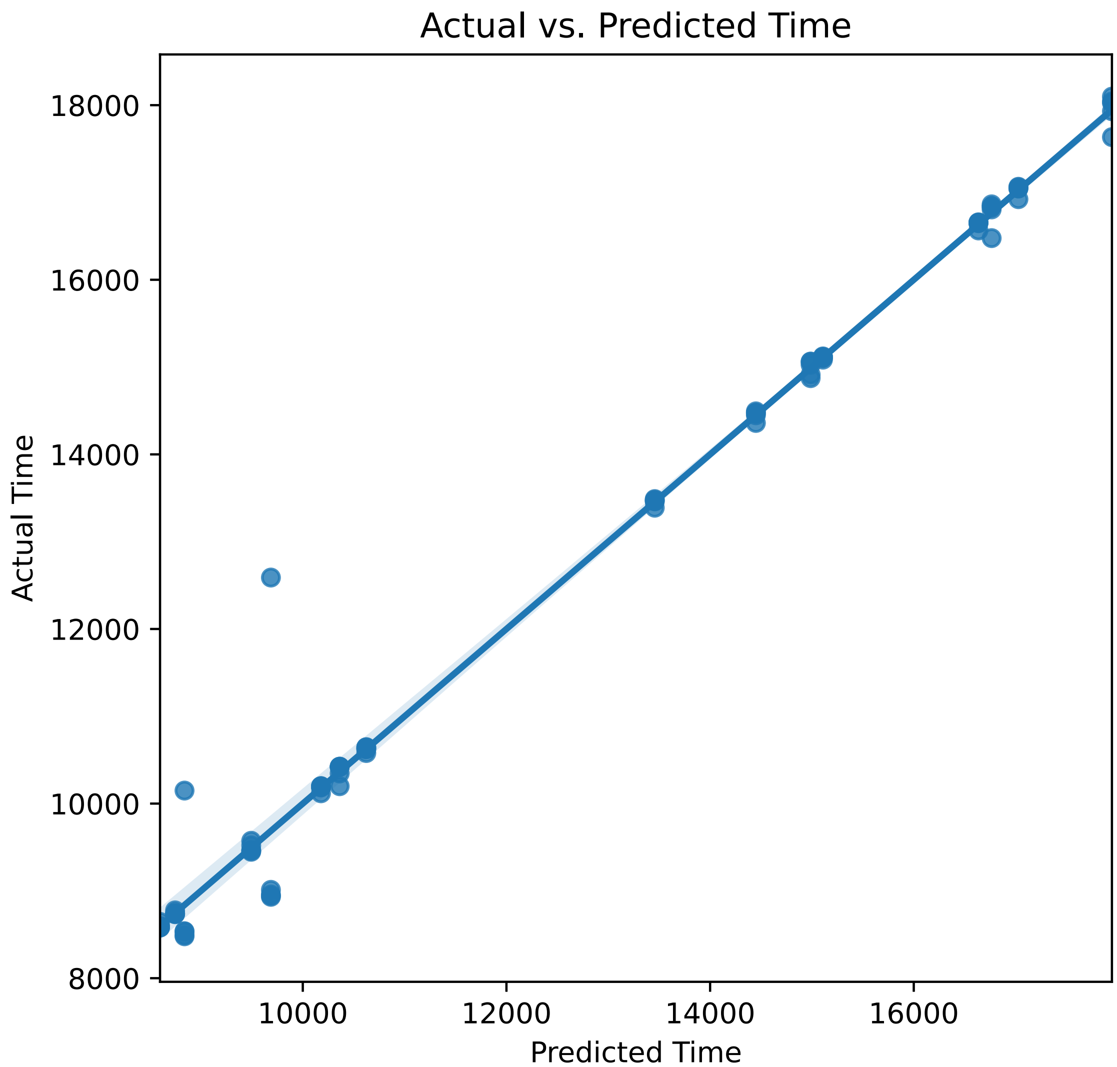
```python
f = 'time ~ (alh+aps+aid+arw+awt)**2'
y, X = patsy.dmatrices(f, df_time, return_type='dataframe')
print(y[:5])
print(X[:5])
```

```
      time
0  18098.0
1   8741.0
2  14493.0
3  10191.0
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
```

```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 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):           1.17e-52
Time:                        20:28:16   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   3.886e+04   4869.801      7.980      0.000    2.91e+04    4.86e+04
alh        -9.373e+04    1.1e+04     -8.494      0.000   -1.16e+05   -7.17e+04
aps         -101.9583     81.061     -1.258      0.213    -263.896      59.980
aid         6178.6667   1.31e+04      0.470      0.640   -2.01e+04    3.24e+04
arw         2096.5000   3353.230      0.625      0.534   -4602.347    8795.347
awt        -8507.9167   3197.582     -2.661      0.010   -1.49e+04   -2120.012
alh:aps      302.8333    168.294      1.799      0.077     -33.372     639.039
alh:aid    -4.994e+04   1.68e+04     -2.968      0.004   -8.36e+04   -1.63e+04
alh:arw     1.009e+04   4207.345      2.399      0.019    1688.613    1.85e+04
alh:awt     2.909e+04   4207.345      6.913      0.000    2.07e+04    3.75e+04
aps:aid      301.9000    201.953      1.495      0.140    -101.547     705.347
aps:arw     -137.1250     50.488     -2.716      0.008    -237.987     -36.263
aps:awt      -18.6750     50.488     -0.370      0.713    -119.537      82.187
aid:arw     5830.0000   5048.814      1.155      0.252   -4256.165    1.59e+04
aid:awt    -1.288e+04   5048.814     -2.552      0.013    -2.3e+04   -2798.835
arw:awt     1626.2500   1262.203      1.288      0.202    -895.291    4147.791
==============================================================================
Omnibus:                      117.964   Durbin-Watson:                   1.973
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             3766.256
Skew:                           4.734   Prob(JB):                         0.00
Kurtosis:                      35.252   Cond. No.                     3.13e+04
==============================================================================

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

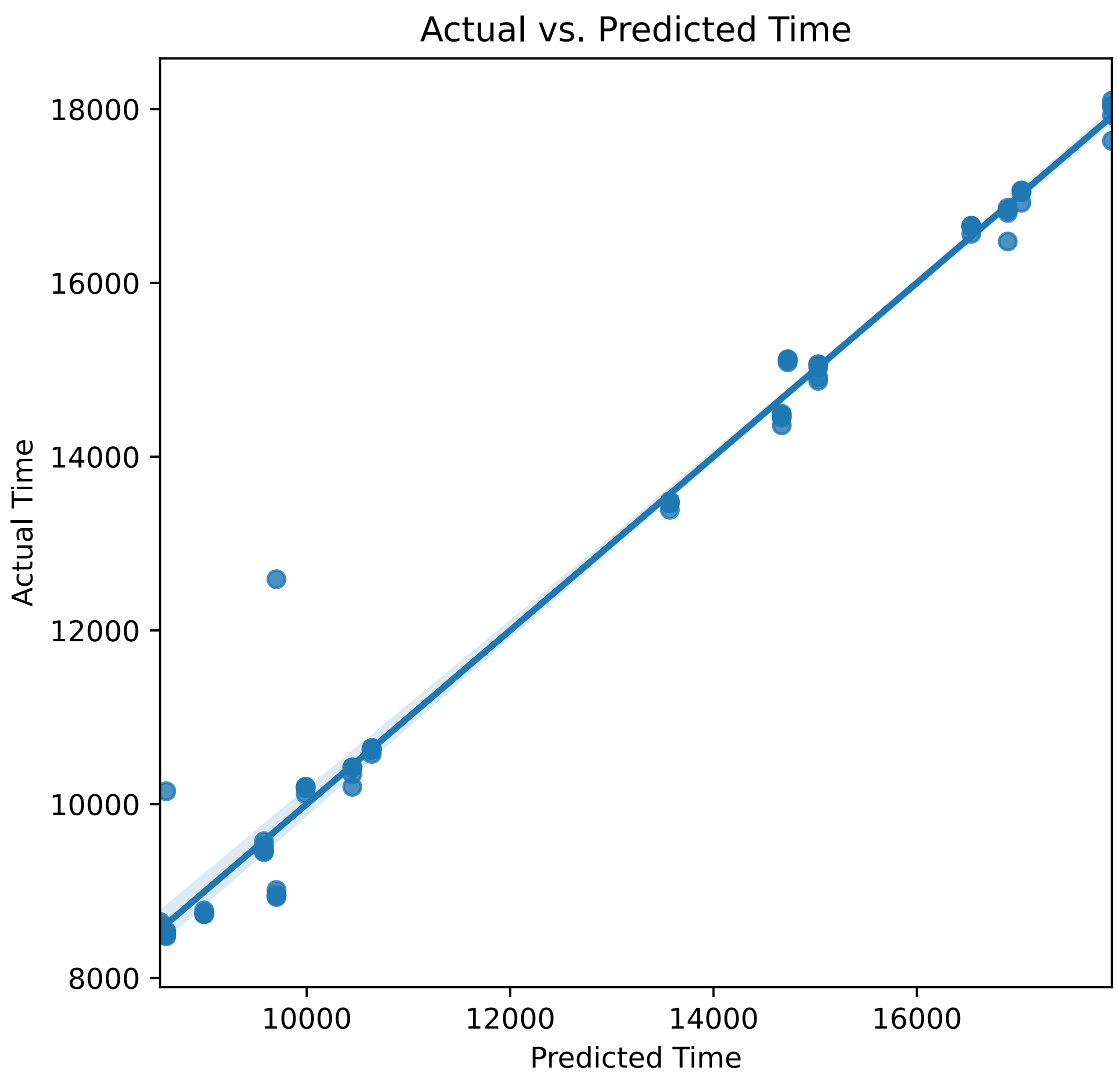
```
Drop aps:awt                    with p-value 0.712687
Drop aid                        with p-value 0.63764
Drop arw                        with p-value 0.549529
Drop aid:arw                    with p-value 0.12172
Drop alh:aps                    with p-value 0.0744513
Drop arw:awt                    with p-value 0.0725949
['alh', 'aps', 'awt', 'alh:aid', 'alh:arw', 'alh:awt', 'aps:aid', 'aps:arw', 'ai
d: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:                     453.8
Date:                Sat, 31 Jul 2021   Prob (F-statistic):           2.19e-58
Time:                        20:28:18   Log-Likelihood:                -599.26
No. Observations:                  80   AIC:                             1219.
Df Residuals:                      70   BIC:                             1242.
Df Model:                           9
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       3.942e+04   1154.479     34.142      0.000    3.71e+04    4.17e+04
alh        -7.914e+04   6029.887    -13.124      0.000   -9.12e+04   -6.71e+04
aps         -128.1914     25.154     -5.096      0.000    -178.360     -78.023
awt        -8860.8498   1369.537     -6.470      0.000   -1.16e+04   -6129.395
alh:aid    -4.626e+04   1.65e+04     -2.798      0.007   -7.92e+04   -1.33e+04
alh:arw      1.23e+04   4097.512      3.002      0.004    4127.272    2.05e+04
alh:awt     2.909e+04   4317.386      6.737      0.000    2.05e+04    3.77e+04
aps:aid      434.5856    102.753      4.229      0.000     229.652     639.519
aps:arw      -57.7179     16.919     -3.411      0.001     -91.462     -23.974
aid:awt    -1.138e+04   4760.148     -2.390      0.020   -2.09e+04   -1883.394
==============================================================================
Omnibus:                      103.336   Durbin-Watson:                   1.908
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             2137.913
Skew:                           3.999   Prob(JB):                         0.00
Kurtosis:                      27.030   Cond. No.                     2.14e+04
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
 specified.
[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 = 39415.87499999855 + -79138.51046393832 * model.X1 + -128.19137670175314 * model.X2 + -8860.849838411767 * model.X5 + -46255.95567867106 * model.X1*model.X3 + 12299.502666120085 * model.X1*model.X4 + 29085.416666666522 * model.X1*model.X5 + 434.58559556786804 * model.X2*model.X3 + -57.717904019688284 * model.X2*model.X4 + -11377.20914127423 * model.X3*model.X5
```

Equations

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

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

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

Cost =

```
0.43619166666666676 + -0.3303957650272835 * model.X1 + -0.0018884142076504296 *
model.X2 + 0.3583517213114778 * model.X4 + 0.2018153688524602 * model.X5 + 0.003
79166666666636 * model.X1*model.X2 + -0.8282295081967748 * model.X1*model.X3 + 0
.19062500000000449 * model.X1*model.X4 + 0.009283737704918856 * model.X2*model.X
3 + -0.0030125000000000516 * model.X2*model.X4 + 0.0011375000000000004 * model.X2*
model.X5 + 0.4151163934426205 * model.X3*model.X4 + -0.5256393442622931 * model.
X3*model.X5 + -0.25406250000000025 * model.X4*model.X5
------
```

Time =

```
39415.87499999855 + -79138.51046393832 * model.X1 + -128.19137670175314 * model.
X2 + -8860.849838411767 * model.X5 + -46255.95567867106 * model.X1*model.X3 + 12
299.502666120085 * model.X1*model.X4 + 29085.416666666522 * model.X1*model.X5 +
434.58559556786804 * model.X2*model.X3 + -57.717904019688284 * model.X2*model.X4
 + -11377.20914127423 * model.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 == (0.43619166666666676 + -0.33039576502
72835 * model.X1 + -0.0018884142076504296 * model.X2 + 0.3583517213114778 * mode
l.X4 + 0.2018153688524602 * model.X5 + 0.0037916666666636 * model.X1*model.X2 +
 -0.8282295081967748 * model.X1*model.X3 + 0.19062500000000449 * model.X1*model.
X4 + 0.009283737704918856 * model.X2*model.X3 + -0.0030125000000000516 * model.X
2*model.X4 + 0.0011375000000000004 * model.X2*model.X5 + 0.4151163934426205 * mode
l.X3*model.X4 + -0.5256393442622931 * model.X3*model.X5 + -0.25406250000000025 *
 model.X4*model.X5))
model.C_f2 = Constraint(expr = model.f2 == (39415.87499999855 + -79138.510463938
32 * model.X1 + -128.19137670175314 * model.X2 + -8860.849838411767 * model.X5 +
 -46255.95567867106 * model.X1*model.X3 + 12299.502666120085 * model.X1*model.X4
 + 29085.416666666522 * model.X1*model.X5 + 434.58559556786804 * model.X2*model.
X3 + -57.717904019688284 * model.X2*model.X4 + -11377.20914127423 * model.X3*mod
el.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()
```

```python
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=minimi
ze)
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]

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.sl
ack), value(model.f2))
```

```
( X1 , X2, X3, X4, X5 ) = ( 0.2800000007742209 , 50.000001589671186 , 0.24999989
70310959 , 0.400000005503882 , 0.8000000876753305 )
f1 = 0.5268908923649562
f2 = 10416.123742705258
( X1 , X2, X3, X4, X5 ) = ( 0.28000000999993935 , 60.000000599972694 , 0.2500000
099939384 , 0.8000000096771722 , 1.2000000119993277 )
f1 = 0.5656472935545355
f2 = 8557.622612553765
```

```python

```