Table of Contents

Regression - Assignment 3

Data and Package Import

In [227]:
```
%matplotlib inline
import numpy as np
import pandas as pd
import pylab as plt
```

In [228]:
```
df = pd.read_excel('data/impurity_dataset-training.xlsx')

def is_real_and_finite(x):
    if not np.isreal(x):
        return False
    elif not np.isfinite(x):
        return False
    else:
        return True

all_data = df[df.columns[1:]].values
numeric_map = df[df.columns[1:]].applymap(is_real_and_finite)
real_rows = numeric_map.all(axis = 1).copy().values
X = np.array(all_data[real_rows, :-5], dtype = 'float')
y = np.array(all_data[real_rows, -3], dtype = 'float')
y = y.reshape(-1, 1)

print('X matrix dimensions: {}'.format(X.shape))
print('y matrix dimensions: {}'.format(y.shape))
```

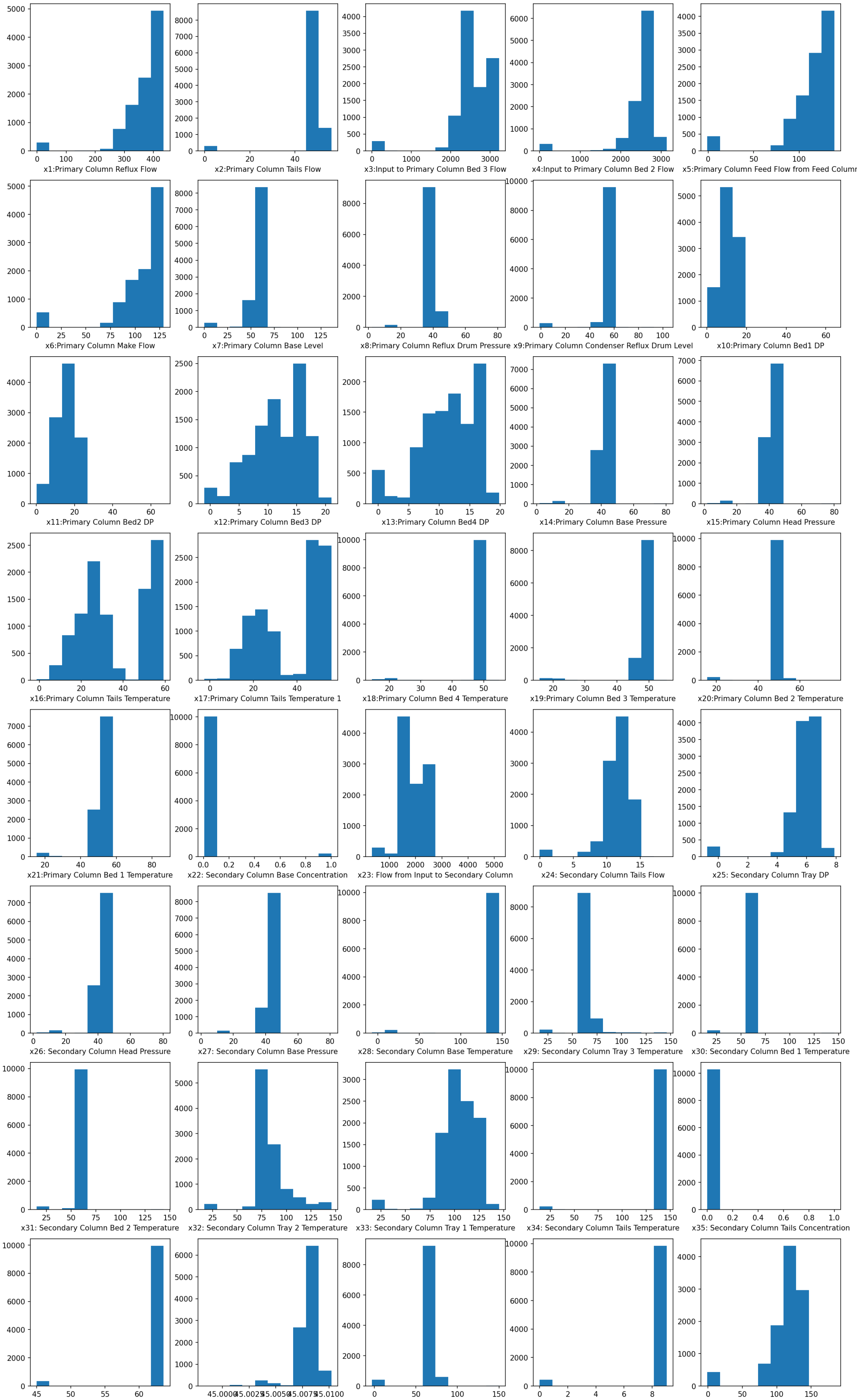
```
X matrix dimensions: (10297, 40)
y matrix dimensions: (10297, 1)
```

Distribution of Features

Plot histograms of all 40 features.

In [229]:

```
fig, axes = plt.subplots(8, 5, figsize = (20, 35), dpi = 150)

x_nameList = [str(x) for x in df.columns[1:41]]
y_nameList = str(df.columns[-3]) #last three columns of excel file

N = X.shape[-1]
n = int(np.sqrt(N))
ax_list = axes.ravel()
for i in range(N):
    ax_list[i].hist(X[:,i])
    ax_list[i].set_xlabel(x_nameList[i])
```



Name a feature that is approximately normally distributed.

You may use visual inspection to answer the following questions.

The features that look approximately normally distributed are Primary Column Bed3 DP (12), Primary Column Bed4 DP (13), Secondary Column Tray 1 Temperature (33).

Name a feature that is approximately bimodally distributed.

The features that appear to be bimodally distributed with two distinct peaks are Input to Primary Column Bed 3 Flow (3), Primary Column Tails Temperature (16), Primary Column Tails Temperature 1 (17).

Name a feature that has significant outliers.

The features with prominent outliers have a large cluster of data points in one area and a large separating other data points in another area. Examples of this include Primary Column Make Flow (6), Primary Column Feed Flow from Feed Column (5), and Feed Column Tails Flow (40).

Feature Scaling

Down-sample the dataset by selecting every 10th data point.

In [204]:

```
x_tenth = X[::10]
y_tenth = y[::10]
print(x_tenth.shape)
print(y_tenth.shape)
```

```
(1030, 40)
(1030, 1)
```

Do a train/test split with `test_size=0.3`.

In [205]:

```
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x_tenth, y_tenth, test_si

print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(721, 40)
(309, 40)
(721, 1)
(309, 1)
```

Use the standard scaler and make the standardized dataset.

In [206]:

```
from sklearn.preprocessing import StandardScaler

scalerVal = StandardScaler().fit(x_train)
X_scaledStand = scalerVal.transform(x_train)
X_scaledStandTest = scalerVal.transform(x_test)

print(np.mean(x_train))
print(np.mean(X_scaledStand))

#print(x_train) #data before scaling
#print(X_scaledStand) #data after using Standard Scalar ()

print(X_scaledStand.mean(axis=0))
print(X_scaledStand.std(axis=0))

print(x_train.shape)
print(X_scaledStand.shape)
print(y_train.shape)
```

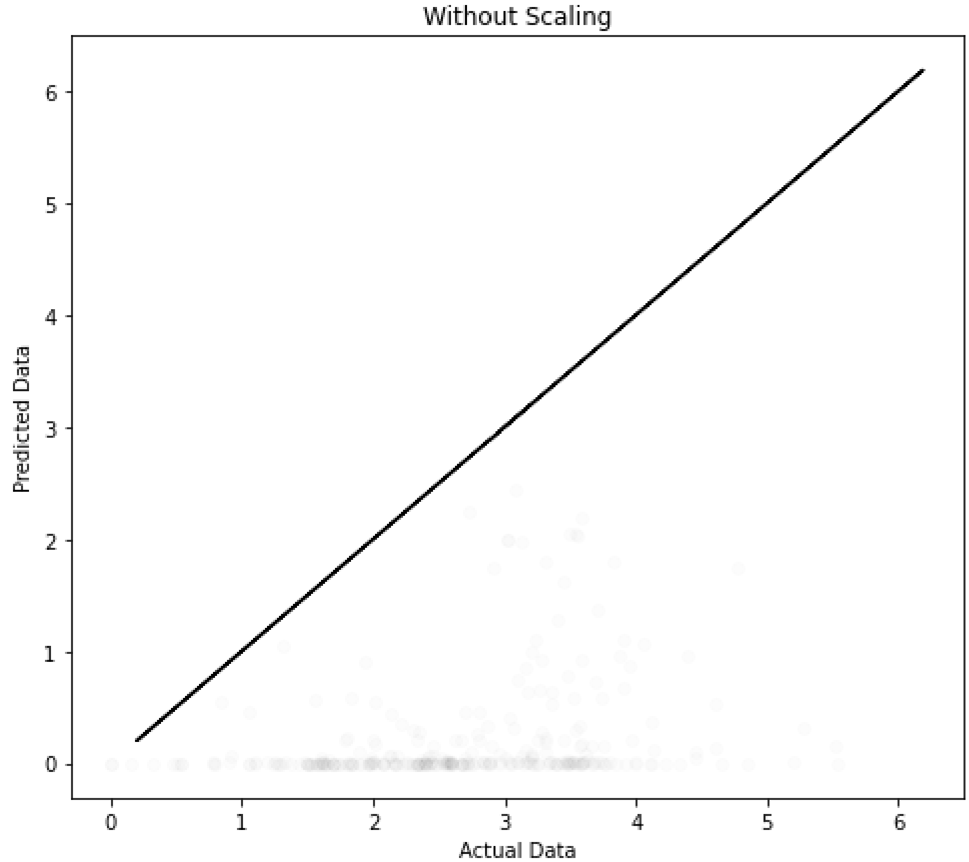
```
227.14895766739045
2.387648285279808e-12
[-6.91926097e-16  1.92302639e-15 -1.87502194e-15 -1.07488375e-15
  1.79229416e-15  8.60268860e-16  4.02596357e-15 -1.27031996e-14
 -4.81534215e-15  2.70025950e-15 -2.18164214e-15  4.56407912e-16
 -7.41123914e-16 -1.70512010e-15 -5.95520705e-15 -2.01021970e-15
 -1.37045561e-15 -6.74260309e-15  7.52056762e-16  9.58753034e-15
 -1.17433032e-14 -1.97484193e-17  4.24995222e-17  2.80758619e-15
 -3.30641663e-16  6.11130810e-15  5.30643490e-15 -2.50169742e-15
 -2.38674853e-15 -2.65088845e-15 -8.38299225e-15  6.35521840e-15
 -5.33338206e-15  6.66703555e-15  3.90175640e-16 -1.48364908e-14
  9.55465272e-11  5.91940582e-15 -6.44699273e-15  1.58872761e-16]
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
(721, 40)
(721, 40)
(721, 1)
```

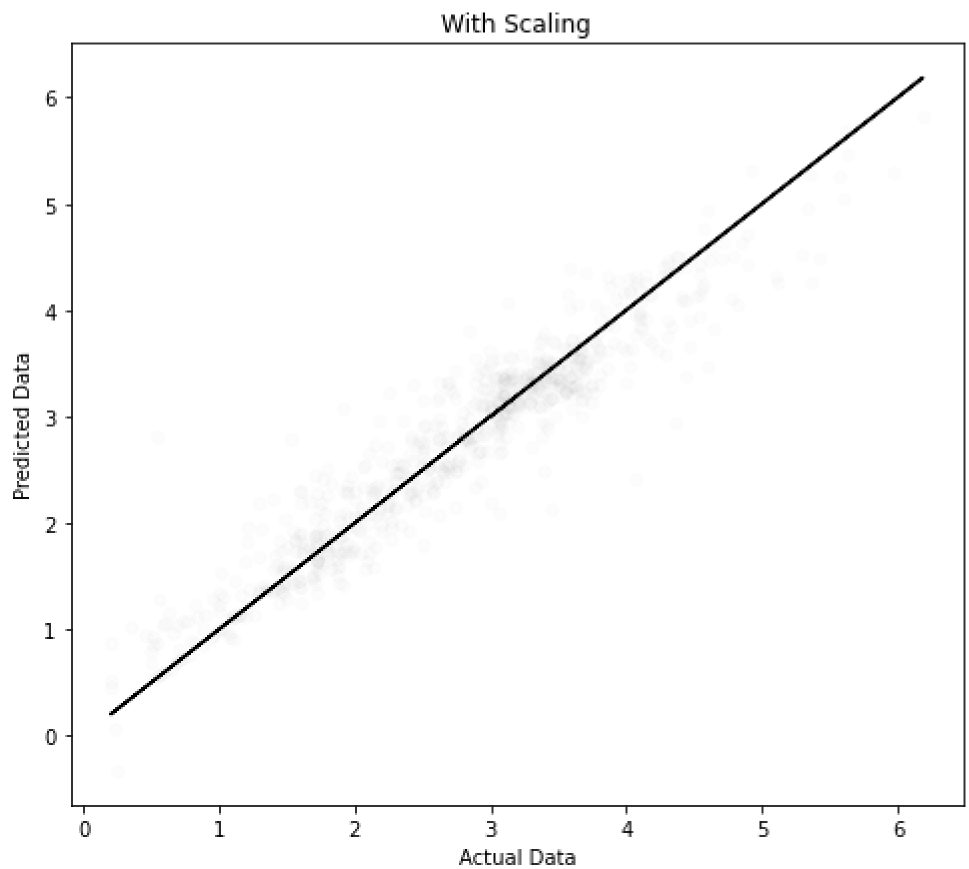
Build a KRR model on the Dow dataset with and without scaling.

Set γ=0.01 and α=0.01.

In [207]:
```
from sklearn.kernel_ridge import KernelRidge

gamma = 0.01
alpha = 0.01 #LOW ALPHA = LIGHT/TRANSPARENT POINTS

KRR = KernelRidge(alpha = alpha, kernel = 'rbf', gamma = gamma)

#WITHOUT SCALING ########################
KRR.fit(x_train, y_train)
r2_withoutScale = KRR.score(x_test, y_test)
yhat_KRR = KRR.predict(x_test)

fig, ax = plt.subplots(figsize = (8,7))
ax.scatter(y_test, yhat_KRR, alpha = alpha, c = 'black')
ax.plot(y_train, y_train, 'k')
ax.set_xlabel('Actual Data')
ax.set_ylabel('Predicted Data')
ax.set_title('Without Scaling')

#WITH SCALING ###########################
KRR.fit(X_scaledStand, y_train)
r2_withScale = KRR.score(X_scaledStand, y_train)
yhat_KRR = KRR.predict(X_scaledStand)

fig, ax = plt.subplots(figsize = (8,7))
ax.scatter(y_train, yhat_KRR, alpha = alpha, marker = 'o', color = 'black')
ax.plot(y_train, y_train, 'k')
ax.set_xlabel('Actual Data')
ax.set_ylabel('Predicted Data')
ax.set_title('With Scaling')
```

Out[207]:
```
Text(0.5, 1.0, 'With Scaling')
```


Without Scaling
Predicted Data
Actual Data
0
1
2
3
4
5
6


With Scaling
Predicted Data
Actual Data
0
1
2
3
4
5
6

Compare the r^2 score on the test set of the two approaches.

In [208]:
```
print(r2_withoutScale)
print(r2_withScale)
```

```
-6.253292482533182
0.9000403429704119
```

LASSO Regression

Scale the feature matrix using the standard scaler.

In [209]:
```
X_scaled = (X - X.mean(axis = 0))/X.std(axis =0)
print('Minimum:{}, Maximimum: {}'.format(X.min(), X.max()))
print('Maximum scaled: {}, Maxmium scaled: {}'.format(X_scaled.min(),X_scaled

covar = np.cov(X_scaled.T)
corr = np.corrcoef(X.T)
np.isclose(corr, covar, 1e-4).all()

print(X_scaled.shape)
print(y.shape)
```

```
Minimum:-6.91425, Maximimum: 5176.74
Maximum scaled: -8.12009681442378, Maxmium scaled: 38.10583689480496
(10297, 40)
(10297, 1)
```

Shuffle the data.

In [210]:
```
from sklearn.linear_model import Lasso
from sklearn.model_selection import GridSearchCV
from sklearn.metrics.pairwise import rbf_kernel

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

In [211]:
```
from sklearn.utils import shuffle

x_scale_shuffle, y_scale_shuffle = shuffle(X_scaled, y)
```

Build a `GridSearchCV` model that optimizes the hyperparameters of a LASSO model.

Search over $\alpha \in$ [1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1].

Use 3-fold cross-validation.

In [234]:
```
alphas = np.array([1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1])

listAlp = []

gamma = 0.01

lasso = Lasso()
lasso = Lasso(max_iter = 100000, tol = 0.005)
param_grid = {'alpha':alphas}
lasso_search = GridSearchCV(lasso, param_grid, cv=3)
lasso_search.fit(x_scale_shuffle, y_scale_shuffle)
print(lasso_search.best_estimator_, lasso_search.best_score_)

r2 = lasso_search.best_estimator_.score(x_scale_shuffle, y_scale_shuffle)
listAlp.append(r2)
```

```
Lasso(alpha=0.001, max_iter=100000, tol=0.005) 0.6966856517221712
```

Evaluate the performance of the best model.

Print the optimized α as well as the r^2 score.

In [235]:
```
print(lasso_search.best_estimator_, lasso_search.best_score_)
```

```
Lasso(alpha=0.001, max_iter=100000, tol=0.005) 0.6966856517221712
```

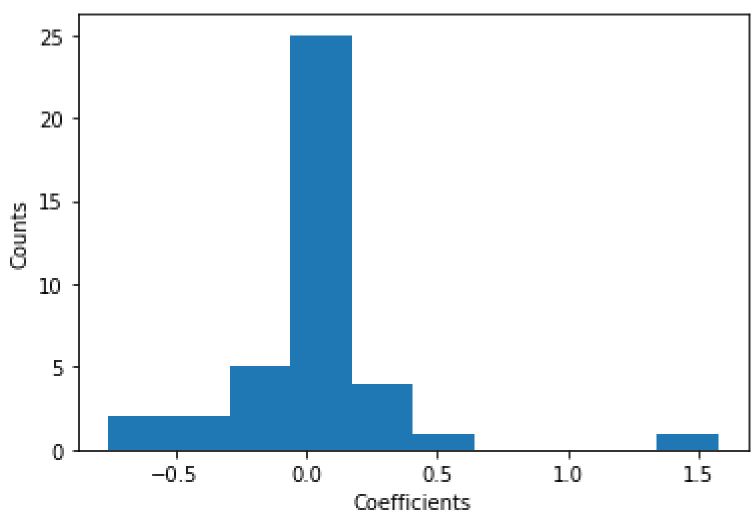
Describe which features (if any) were dropped.

Dropped features have coefficients equal to zero.

In [236]:

```
coeffs = lasso_search.best_estimator_.coef_
coeffs.shape

for i in range(len(coeffs)):
    if np.isclose(coeffs[i],0):
        print(x_nameList[i])

fig, ax = plt.subplots()
ax.hist(coeffs)
ax.set_xlabel('Coefficients')
ax.set_ylabel('Counts')

nonzero = [f for f in np.isclose(coeffs, 0) if f == False]
print('Total number of nonzero parameters: {}'.format(len(nonzero)))
```

```
x14:Primary Column Base Pressure
x15:Primary Column Head Pressure
x18:Primary Column Bed 4 Temperature
x21:Primary Column Bed 1 Temperature
x26: Secondary Column Head Pressure
x27: Secondary Column Base Pressure
x35: Secondary Column Tails Concentration
Total number of nonzero parameters: 33
```



Principal Component and Forward Selection

Use the eigenvalues of the covariance matrix to perform PCA on the scaled feature matrix.

Hint: You can check your answers using PCA from `scikit-learn` *or other packages if you want*

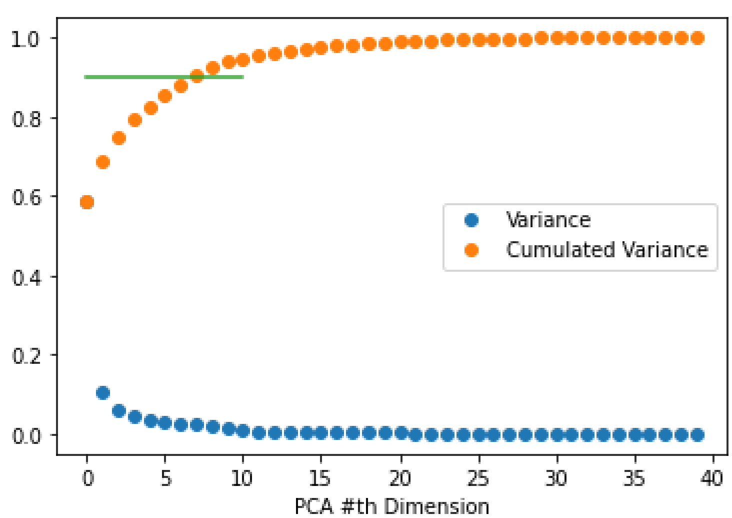
In [237]:

```
from scipy.linalg import eigvals, eig

eigvals,eigvecs = eig(corr)

PCvals, PCvecs = eigvals, eigvecs
total_variance = np.sum(np.real(PCvals))
exp_var = np.real(PCvals)/total_variance
print(total_variance)
print(exp_var)

fig,ax = plt.subplots()
ax.plot(exp_var, 'o')
ax.plot(np.cumsum(exp_var),'o')
ax.plot([0, 10], [0.9, 0.9])
ax.set_xlabel('PCA #th Dimension')
ax.legend(['Variance','Cumulated Variance'])
```

```
40.00000000000003
[5.85131103e-01 1.04616962e-01 5.82973742e-02 4.40659737e-02
 3.40839033e-02 3.01618510e-02 2.54587022e-02 2.26421290e-02
 1.95641723e-02 1.52615787e-02 8.68826826e-03 7.01138646e-03
 5.92122630e-03 5.41361439e-03 4.74920386e-03 4.01904250e-03
 3.72210430e-03 2.83302111e-03 2.64161866e-03 2.38163482e-03
 2.27195475e-03 1.88085134e-03 1.60501789e-03 1.51764287e-03
 1.28150640e-03 8.76163737e-04 7.83694293e-04 6.94054814e-04
 6.41230512e-04 5.28499069e-04 3.77574006e-04 3.12684559e-04
 2.53737597e-04 1.12340442e-04 9.25549278e-05 6.56928898e-05
 3.51662969e-05 2.35729919e-06 1.23238073e-06 1.17395139e-06]
```

Out[237]:

```
<matplotlib.legend.Legend at 0x13de17fe220>
```



Determine which principal component of the dataset is most linearly correlated with the impurity concentration.

Print the order of the principal component (e.g. 5th PC) and its r^2 score.

In [244]:
```
from sklearn.linear_model import LinearRegression

PC_projection = np.dot(X_scaled, PCvecs)

N = 5

model_PC = LinearRegression()
model_PC.fit(PC_projection[:,:N], y)
r2 = model_PC.score(PC_projection[:, :N], y)
print('r^2  (5th Order) PCA = {}'.format(r2))

model = LinearRegression()
model.fit(X_scaled[:,:N],y)
r2 = model.score(X_scaled[:,:N], y)
print('r^2 (5th Order) regular = {}'.format(r2))
```

```
r^2  (5th Order) PCA = 0.44544601313776533
r^2 (5th Order) regular = 0.4644174145725659
```

Determine which original feature of the dataset is most linearly correlated to the impurity concentration.

Print the name of the feature and its r^2 score.

In [247]:

```
scoreList = []
for j in range(PC_projection.shape[1]):
    model = LinearRegression()
    xj = PC_projection[:,j].reshape(-1,1)
    model.fit(xj, y)
    r2 = model.score(xj, y)
    scoreList.append([r2, j])
scoreList.sort()
scoreList.reverse()

for r, j in scoreList:
    print('{}: r^2 = {}'.format(j,r))
```

```
1: r^2 = 0.20685135722275394
0: r^2 = 0.1740523250716769
6: r^2 = 0.06122482871680679
7: r^2 = 0.06048989471356614
4: r^2 = 0.04417209773857633
25: r^2 = 0.017497338519021688
8: r^2 = 0.016205721751366142
5: r^2 = 0.013951580686418774
2: r^2 = 0.013223153135118348
16: r^2 = 0.013047707758553573
33: r^2 = 0.011755340770010725
18: r^2 = 0.009381481309652218
9: r^2 = 0.009144490126667848
15: r^2 = 0.008497745937736667
3: r^2 = 0.007147079969638592
21: r^2 = 0.006899441884438029
31: r^2 = 0.006459664342175042
22: r^2 = 0.005113590985049377
14: r^2 = 0.003515332215412892
11: r^2 = 0.0033082402426461988
39: r^2 = 0.0032105781156153146
38: r^2 = 0.002510692114693458
10: r^2 = 0.0023541786992695712
27: r^2 = 0.0022663723214013665
32: r^2 = 0.002216144465406411
13: r^2 = 0.0019406439983040702
37: r^2 = 0.0019400593063266802
20: r^2 = 0.0018165356203503347
28: r^2 = 0.00147546784014152
12: r^2 = 0.000987838964638721
36: r^2 = 0.000725629300603714
34: r^2 = 0.0006972577893930021
24: r^2 = 0.0006704453274830602
26: r^2 = 0.0005656925838350979
17: r^2 = 0.0004727962229361671
35: r^2 = 0.00044635061456155256
30: r^2 = 0.00035149123509436997
19: r^2 = 0.00020390636051270672
29: r^2 = 3.351129185191759e-05
23: r^2 = 1.63738252623169e-07
```

In [257]:
```
maxScore = max(scoreList)
j = maxScore[1]

print(x_nameList[j])
```

```
x2:Primary Column Tails Flow
```

In [258]:
```
print('r^2 = {}'.format(maxScore[0]))
```

```
r^2 = 0.20685135722275394
```

In [ ]:
```

```