Table of Contents

Regression - Assignment 3

Data and Package Import

In [2]:
```
%matplotlib inline
import numpy as np
import pandas as pd
import pylab as plt
```

In [3]:
```
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))
```

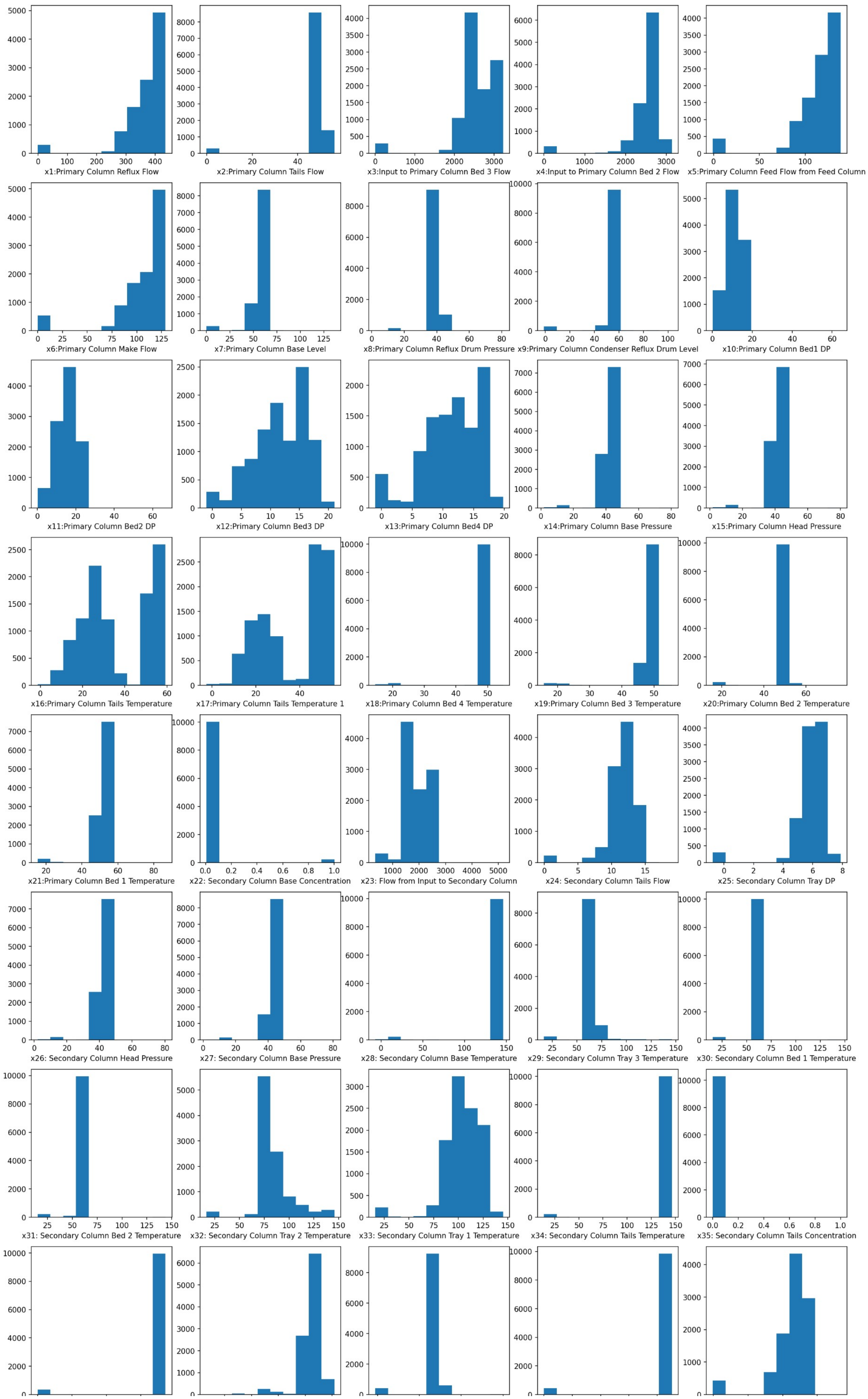
```
X matrix dimensions: (10297, 40)
y matrix dimensions: (10297, 1)
```

Distribution of Features

Plot histograms of all 40 features.

In [9]:
```
fig, axes = plt.subplots(8, 5, figsize = (20, 35), dpi = 150)
x_names = [str(x) for x in df.columns[1:41]];
y_name = str(df.columns[-3])

print('X dimensions: {}'.format(X.shape))
print('Feature names: {}'.format(x_names))
N = X.shape[-1]
n = int(np.sqrt(N))
#fig, axes = plt.subplots(n, n+1, figsize = (5*n, 5*n))
ax_list = axes.ravel()
for i in range(N):
    ax_list[i].hist(X[:,i])
    ax_list[i].set_xlabel(x_names[i])
```

```
X dimensions: (10297, 40)
Feature names: ['x1:Primary Column Reflux Flow', 'x2:Primary Column Tails Flow',
'x3:Input to Primary Column Bed 3 Flow', 'x4:Input to Primary Column Bed 2 Flow
', 'x5:Primary Column Feed Flow from Feed Column', 'x6:Primary Column Make Flow
', 'x7:Primary Column Base Level', 'x8:Primary Column Reflux Drum Pressure', 'x
9:Primary Column Condenser Reflux Drum Level', 'x10:Primary Column Bed1 DP', 'x1
1:Primary Column Bed2 DP', 'x12:Primary Column Bed3 DP', 'x13:Primary Column Bed
4 DP', 'x14:Primary Column Base Pressure', 'x15:Primary Column Head Pressure', '
x16:Primary Column Tails Temperature', 'x17:Primary Column Tails Temperature 1',
'x18:Primary Column Bed 4 Temperature', 'x19:Primary Column Bed 3 Temperature',
'x20:Primary Column Bed 2 Temperature', 'x21:Primary Column Bed 1 Temperature',
'x22: Secondary Column Base Concentration', 'x23: Flow from Input to Secondary C
olumn', 'x24: Secondary Column Tails Flow', 'x25: Secondary Column Tray DP', 'x2
6: Secondary Column Head Pressure', 'x27: Secondary Column Base Pressure', 'x28:
Secondary Column Base Temperature', 'x29: Secondary Column Tray 3 Temperature',
'x30: Secondary Column Bed 1 Temperature', 'x31: Secondary Column Bed 2 Temperat
ure', 'x32: Secondary Column Tray 2 Temperature', 'x33: Secondary Column Tray 1
Temperature', 'x34: Secondary Column Tails Temperature', 'x35: Secondary Column
Tails Concentration', 'x36: Feed Column Recycle Flow', 'x37: Feed Column Tails F
low to Primary Column', 'x38: Feed Column Calculated DP', 'x39: Feed Column Stea
m Flow', 'x40: Feed Column Tails Flow']
```


x1:Primary Column Reflux Flow
x2:Primary Column Tails Flow
x3:Input to Primary Column Bed 3 Flow
x4:Input to Primary Column Bed 2 Flow
x5:Primary Column Feed Flow from Feed Column
x6:Primary Column Make Flow
x7:Primary Column Base Level
x8:Primary Column Reflux Drum Pressure
x9:Primary Column Condenser Reflux Drum Level
x10:Primary Column Bed1 DP
x11:Primary Column Bed2 DP
x12:Primary Column Bed3 DP
x13:Primary Column Bed4 DP
x14:Primary Column Base Pressure
x15:Primary Column Head Pressure
x16:Primary Column Tails Temperature
x17:Primary Column Tails Temperature 1
x18:Primary Column Bed 4 Temperature
x19:Primary Column Bed 3 Temperature
x20:Primary Column Bed 2 Temperature
x21:Primary Column Bed 1 Temperature
x22: Secondary Column Base Concentration
x23: Flow from Input to Secondary Column
x24: Secondary Column Tails Flow
x25: Secondary Column Tray DP
x26: Secondary Column Head Pressure
x27: Secondary Column Base Pressure
x28: Secondary Column Base Temperature
x29: Secondary Column Tray 3 Temperature
x30: Secondary Column Bed 1 Temperature
x31: Secondary Column Bed 2 Temperature
x32: Secondary Column Tray 2 Temperature
x33: Secondary Column Tray 1 Temperature
x34: Secondary Column Tails Temperature
x35: Secondary Column Tails Concentration

Name a feature that is approximately normally distributed.

You may use visual inspection to answer the following questions.

x12 and x33 have a distribution that is similar to a normal distribution, resembling the bell curve that they have.

Name a feature that is approximately bimodally distributed.

In [ ]:
```
x3: Input to Primary column bed flow is bimodally distributed, as there are two hig
h points in the histogram.
```

Name a feature that has significant outliers.

x36: Feed Column Recycle flow has a large outlier since all of the data is seen in one bar on the histogram except for the very small bar at the 45 tick mark.

Feature Scaling

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

In [40]:
```
X_ten = X[0::10]
y_ten = y[0::10]
```

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

In [41]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_ten, y_ten, test_size = 0.3)
```

Use the standard scaler and make the standardized dataset.

In [42]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_train)

X_scaled_test = scaler.transform(X_test)
```

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

Set γ=0.01 and α=0.01.

In [43]:
```
from sklearn.kernel_ridge import KernelRidge
alpha = 0.01
gamma = 0.01

#With scaling

KRR = KernelRidge(alpha = alpha, kernel = 'rbf', gamma = gamma)
KRR.fit(X_scaled, y_train)
yhat = KRR.predict(X_test)
r2_scaled = KRR.score(X_scaled_test,y_test)


#Without scaling
KRR = KernelRidge(alpha = alpha, kernel = 'rbf', gamma = gamma)
KRR.fit(X_train, y_train)
yhat = KRR.predict(X_test)
r2_unsc = KRR.score(X_test,y_test)
```

```
0.7966119235982759
-5.376112560367068
```

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

In [45]:
```
print('r2 of unscaled training set: {}'.format(r2_unsc))
print('r2 of scaled training set: {}'.format(r2_scaled))
print('The r2 of the scaled set is much closer to 1 than the unscaled set.')
```

```
r2 of unscaled training set: -5.376112560367068
r2 of scaled training set: 0.7966119235982759
The r2 of the scaled set is much closer to 1 than the unscaled set.
```

LASSO Regression

Scale the feature matrix using the standard scaler.

In [58]:
```
scaler = StandardScaler()
X_scaled_tot = scaler.fit_transform(X)

#X_scaled_test_tot = scaler.transform(X_test)
```

Shuffle the data.

In [59]:
```
from sklearn.utils import shuffle
X_shuffle, y_shuffle = shuffle(X_scaled_tot, 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 [71]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import Lasso
import warnings
warnings.simplefilter('ignore')

r2 = []
alphas = np.array([1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1])
lasso = Lasso(max_iter = 1000000, tol = 0.005)
param_grid = {'alpha':alphas}

lasso_search = GridSearchCV(lasso,param_grid, cv = 3)
lasso_search.fit(X_shuffle,y_shuffle)
print(lasso_search.best_estimator_, lasso_search.best_score_)
r2.append(lasso_search.best_score_)
```

```
Lasso(alpha=0.0001, max_iter=1000000, tol=0.002) 0.6729721137283695
```

Evaluate the performance of the best model.

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

In [66]:
```
print(lasso_search.best_estimator_, lasso_search.best_score_)
```

```
Lasso(alpha=0.0001, max_iter=1000000, tol=0.005) 0.6727408027494165
```

Describe which features (if any) were dropped.

Dropped features have coefficients equal to zero.

In [72]:
```
coeff = lasso_search.best_estimator_.coef_
coeff.shape
for x in range(len(coeff)):
    if np.isclose(coeff[x],0):
        print(x_names[x])
```

```
x27: Secondary Column Base Pressure
```

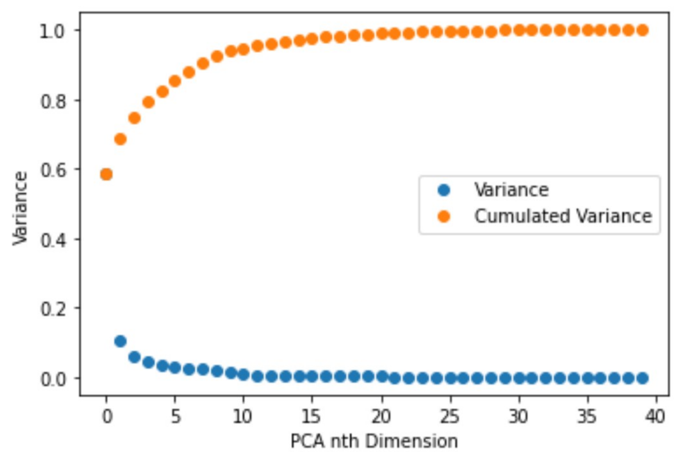
Principal Component and Forward Selection

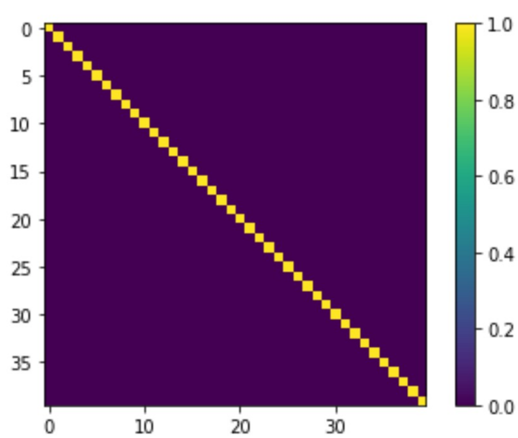
Use the eigenvalues of the covariance matrix to perform PCA on the scaled feature matrix.

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

In [107]:
```
from scipy.linalg import eigvals, eig
corr = np.corrcoef(X.T)
covar = np.cov(X_shuffle.T)
np.isclose(corr, covar, 1e-4).all()
eigvals, eigvecs = eig(cor)

PCvals, PCvecs = eigvals, eigvecs
tot_variance = np.sum(np.real(PCvals))
variance_exp = np.real(PCvals)/tot_variance

fig, ax = plt.subplots()
ax.plot(variance_exp,'o')
ax.plot(np.cumsum(variance_exp),'o')
ax.set_xlabel('PCA nth Dimension')
ax.set_ylabel('Variance')
ax.legend(['Variance', 'Cumulated Variance'])

PC_projection = np.dot(X_scaled_tot, PCvecs)
print(PC_projection.shape)

corr_PCs = np.corrcoef(PC_projection.T)
fig,ax = plt.subplots()
c = ax.imshow(corr_PCs)
fig.colorbar(c);
```

```
(10297, 40)
```





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 [92]:
```
from sklearn.linear_model import LinearRegression


PC_projection = np.dot(X_scaled_tot, PCvecs)

N = 5

model_PC = LinearRegression() #create a linear regression model instance
model_PC.fit(PC_projection[:, :N], y) #fit the model
r2 = model_PC.score(PC_projection[:, :N], y) #get the "score", which is equivalent
to r^2
print("5th order r^2 PCA = {}".format(r2))

model = LinearRegression() #create a linear regression model instance
model.fit(X_scaled_tot[:, :N], y) #fit the model
r2 = model.score(X_scaled_tot[:, :N], y) #get the "score", which is equivalent to r
^2
print("r^2 regular = {}".format(r2))
```

```
r^2 PCA = 0.4454460131377648
r^2 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 [106]:
```
score_list = []
max_score = 0
max_r2 = 0
for j in range(PC_projection.shape[1]):
    model = LinearRegression() #create a linear regression model instance
    xj = PC_projection[:,j].reshape(-1,1)
    model.fit(xj, y) #fit the model
    r2 = model.score(xj, y) #get the "score", which is equivalent to r^2
    score_list.append([r2, j])
    if r2 < 1:
        if r2 > max_r2:
            max_r2 = r2
            max_score = j
score_list.sort()
score_list.reverse()

print('The feature that is most linearaly correlated is ',x_names[max_score])
for r, j in score_list:
    print("{} : r^2 = {}".format(j, r))
print((score_list))
```

```
The feature that is most linearaly correlated is  x2:Primary Column Tails Flow
1 : r^2 = 0.20685135722275394
0 : r^2 = 0.1740523250716769
6 : r^2 = 0.061224828716807345
7 : r^2 = 0.06048989471356592
4 : r^2 = 0.044172097738576
25 : r^2 = 0.017497338519020134
8 : r^2 = 0.016205721751365476
5 : r^2 = 0.01395158068641944
2 : r^2 = 0.013223153135118126
16 : r^2 = 0.013047707758553129
33 : r^2 = 0.011755340770008949
18 : r^2 = 0.009381481309652218
9 : r^2 = 0.00914449012666807
15 : r^2 = 0.008497745937736445
3 : r^2 = 0.007147079969638592
21 : r^2 = 0.006899441884438695
31 : r^2 = 0.006459664342175042
22 : r^2 = 0.0051135909850487105
14 : r^2 = 0.003515332215413447
11 : r^2 = 0.0033082402426463098
39 : r^2 = 0.003210578115308449
38 : r^2 = 0.002510692115424873
10 : r^2 = 0.0023541786992695712
27 : r^2 = 0.0022663723214011444
32 : r^2 = 0.002216144465406522
13 : r^2 = 0.0019406439983039592
37 : r^2 = 0.0019400593063473304
20 : r^2 = 0.0018165356203506677
28 : r^2 = 0.0014754678401431853
12 : r^2 = 0.0009878389646386099
36 : r^2 = 0.0007256293006049352
34 : r^2 = 0.0006972577893956666
24 : r^2 = 0.0006704453274829492
26 : r^2 = 0.0005656925838339877
17 : r^2 = 0.00047279622293650014
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
[[0.20685135722275394, 1], [0.1740523250716769, 0], [0.061224828716807345, 6],
[0.06048989471356592, 7], [0.044172097738576, 4], [0.017497338519020134, 25],
[0.016205721751365476, 8], [0.01395158068641944, 5], [0.013223153135118126, 2],
[0.013047707758553129, 16], [0.011755340770008949, 33], [0.009381481309652218, 1
8], [0.00914449012666807, 9], [0.008497745937736445, 15], [0.007147079969638592,
3], [0.006899441884438695, 21], [0.006459664342175042, 31], [0.00511359098504871
05, 22], [0.003515332215413447, 14], [0.0033082402426463098, 11], [0.00321057811
5308449, 39], [0.002510692115424873, 38], [0.0023541786992695712, 10], [0.002266
3723214011444, 27], [0.002216144465406522, 32], [0.0019406439983039592, 13], [0.
0019400593063473304, 37], [0.0018165356203506677, 20], [0.0014754678401431853, 2
8], [0.0009878389646386099, 12], [0.0007256293006049352, 36], [0.000697257789395
6666, 34], [0.0006704453274829492, 24], [0.0005656925838339877, 26], [0.00047279
622293650014, 17], [0.00044635061456155256, 35], [0.00035149123509436997, 30],
[0.00020390636051270672, 19], [3.351129185191759e-05, 29], [1.63738252623169e-0
7, 23]]
```