MT

In [1]:
```
from IPython.display import set_matplotlib_formats
set_matplotlib_formats('pdf', 'svg')
```

In [2]:
```
import pandas as pd
from sklearn.preprocessing import StandardScaler
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
import scipy.cluster.hierarchy as sch
from sklearn.decomposition import PCA
from sklearn.manifold import MDS
from sklearn.preprocessing import MinMaxScaler
%matplotlib
```

```
Using matplotlib backend: Qt5Agg
```

In [ ]:

In [3]:
```
temp = pd.read_table("slump_test.data", sep=",")
temp2 = pd.read_table("slump_test.names")
display(temp)
#drop the No column along with the target variables other than 280day Compressive Strength
temp = temp.drop(columns=["No", "SLUMP(cm)", "FLOW(cm)"])
temp.head()
```

	No	Cement	Slag	Fly ash	Water	SP	Coarse Aggr.	Fine Aggr.	SLUMP(cm)	FLOW(cm)	Compressive Strength (28-day)(Mpa)
0	1	273.0	82.0	105.0	210.0	9.0	904.0	680.0	23.0	62.0	34.99
1	2	163.0	149.0	191.0	180.0	12.0	843.0	746.0	0.0	20.0	41.14
2	3	162.0	148.0	191.0	179.0	16.0	840.0	743.0	1.0	20.0	41.81
3	4	162.0	148.0	190.0	179.0	19.0	838.0	741.0	3.0	21.5	42.08
4	5	154.0	112.0	144.0	220.0	10.0	923.0	658.0	20.0	64.0	26.82
...	...	...	...	...	...	...	...	...	...	...	...
98	99	248.3	101.0	239.1	168.9	7.7	954.2	640.6	0.0	20.0	49.97
99	100	248.0	101.0	239.9	169.1	7.7	949.9	644.1	2.0	20.0	50.23
100	101	258.8	88.0	239.6	175.3	7.6	938.9	646.0	0.0	20.0	50.50
101	102	297.1	40.9	239.9	194.0	7.5	908.9	651.8	27.5	67.0	49.17
102	103	348.7	0.1	223.1	208.5	9.6	786.2	758.1	29.0	78.0	48.77

103 rows × 11 columns

Out[3]:

	Cement	Slag	Fly ash	Water	SP	Coarse Aggr.	Fine Aggr.	Compressive Strength (28-day)(Mpa)
0	273.0	82.0	105.0	210.0	9.0	904.0	680.0	34.99
1	163.0	149.0	191.0	180.0	12.0	843.0	746.0	41.14
2	162.0	148.0	191.0	179.0	16.0	840.0	743.0	41.81
3	162.0	148.0	190.0	179.0	19.0	838.0	741.0	42.08
4	154.0	112.0	144.0	220.0	10.0	923.0	658.0	26.82

Basic EDA

In [4]:
```
temp.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 103 entries, 0 to 102
Data columns (total 8 columns):
 #   Column                              Non-Null Count  Dtype  
---  ------                              --------------  -----  
 0   Cement                              103 non-null    float64
 1   Slag                                103 non-null    float64
 2   Fly ash                             103 non-null    float64
 3   Water                               103 non-null    float64
 4   SP                                  103 non-null    float64
 5   Coarse Aggr.                        103 non-null    float64
 6   Fine Aggr.                          103 non-null    float64
 7   Compressive Strength (28-day)(Mpa)  103 non-null    float64
dtypes: float64(8)
memory usage: 6.6 KB
```

In [5]:
```
temp.describe().transpose()
```

Out[5]:

	count	mean	std	min	25%	50%	75%	max
Cement	103.0	229.894175	78.877230	137.00	152.00	248.00	303.900	374.00
Slag	103.0	77.973786	60.461363	0.00	0.05	100.00	125.000	193.00
Fly ash	103.0	149.014563	85.418080	0.00	115.50	164.00	235.950	260.00
Water	103.0	197.167961	20.208158	160.00	180.00	196.00	209.500	240.00
SP	103.0	8.539806	2.807530	4.40	6.00	8.00	10.000	19.00
Coarse Aggr.	103.0	883.978641	88.391393	708.00	819.50	879.00	952.800	1049.90
Fine Aggr.	103.0	739.604854	63.342117	640.60	684.50	742.70	788.000	902.00
Compressive Strength (28-day)(Mpa)	103.0	36.039417	7.838232	17.19	30.90	35.52	41.205	58.53

Observations:

1. This dataset has no null values
2. The data is not scaled and if we were to perform PCA on the dataset then we would have to scale and/or standardize the data
3. Columns like Cement, Slag, Fly ash, Water are somewhat evenly distributed with few outliers since the 75% mark is close to the maximum value of the column. However, we can confirm this using boxplots.

In [6]:
```
#Confirming the distributions
#names of the columns
col_names = temp.columns
col_names

#plotting boxplots
plt.figure(figsize=(25,30))
for i in range(len(col_names)):
  plt.subplot(4,3,i+1)
  sns.boxplot(y=temp[col_names[i]])
  plt.xlabel(col_names[i])
  plt.ylabel("")
```

As expected, most columns, except SP and the target variable are evenly distributed.

In [7]:
```
#We can also plot them on a single plot since they all have the same units (kg of component in 1 M^3 of concrete )
plt.subplots(figsize=(12, 8))
ax = sns.boxplot(data=temp)
ax.set_xticklabels(ax.get_xticklabels(), rotation=90);
```

In [8]:
```
#Understanding the correlation between the variables
r = sns.PairGrid(temp)
r.map_upper(plt.scatter)
r.map_diag(sns.kdeplot, lw=3, legend=True)
r.map_lower(plt.plot)
```

Out[8]:
```
<seaborn.axisgrid.PairGrid at 0x20c988cf988>
```

Observation:

1. Cement, Slag, Fly Ash all seem to have 2 peaks hence they have a bimodal distribution which would make sense intuitively.
2. SP also has 2 peaks but one of them is higher than the other one.
3. Cement and Fly Ash seem to have a positive correlation with compressive strength (our target variable)
4. Water and Coarse Aggr. have a weak negative correlation

In [9]:
```
#Plotting Heat Maps to get a more accurate
sns.heatmap(temp.corr(), annot = True)
```

Out[9]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x20c9964e6c8>
```

Observations:

1. As we predicted, Cement and Fly Ash have a positive correlation to Compressive Strength while we learn that Slag and Water are negatively correlated to Compressive Strength. Sp, Coarse Aggr, and Fine Aggr. are all weakly correlated to Compressive Strength.
2. Cement and Fly-ash have a strong negative correlation
3. Water and Coarse Aggr. have a strong negative correlation.

In [ ]:
```

```

PCA

While we have made some predictions, I would like to try another method to see if we can learn something more about the dataset

In [10]:
```
temp.columns
```

Out[10]:
```
Index(['Cement', 'Slag', 'Fly ash', 'Water', 'SP', 'Coarse Aggr.',
       'Fine Aggr.', 'Compressive Strength (28-day)(Mpa)'],
      dtype='object')
```

In [11]:
```
#Step 1: Extract the features from the dataset
data = temp.drop(columns=('Compressive Strength (28-day)(Mpa)'), inplace=False)

#Step 2: Standardize the data
x = StandardScaler().fit_transform(X = data.values)
display(x.shape, np.mean(x), np.std(x))

#convert the normalized features back into a dataframe
feature_cols = data.columns
concrete_norml = pd.DataFrame(data=x, columns=feature_cols)
concrete_norml.tail()
```

```
(103, 7)

-2.0572232467395412e-16

1.0
```

Out[11]:

	Cement	Slag	Fly ash	Water	SP	Coarse Aggr.	Fine Aggr.
98	0.234489	0.382704	1.059799	-1.405679	-0.300589	0.798321	-1.570661
99	0.230667	0.382704	1.069210	-1.395734	-0.300589	0.749436	-1.515135
100	0.368258	0.166639	1.065681	-1.087427	-0.336382	0.624381	-1.484993
101	0.856197	-0.616180	1.069210	-0.157533	-0.372174	0.283322	-1.392979
102	1.513577	-1.294291	0.871569	0.563508	0.379472	-1.111610	0.293416

In [12]:
```
#using the sklearn library to perform pca on the normalised dataset

pca_concrete = PCA(n_components=7)
principalComp_concrete = pca_concrete.fit_transform(x)

print('The proportion of variance explained by each PC: {}'.format(pca_concrete.explained_variance_ratio_))
```

```
The proportion of variance explained by each PC: [0.31779551 0.21620366 0.15837992 0.14373075 0.09732565 0.0661284
 0.00043611]
```

In [13]:
```
#creating a scree plot to determine th number of components to be used
PC_values = np.arange(pca_concrete.n_components_) + 1
plt.plot(PC_values, pca_concrete.explained_variance_ratio_, 'ro-', linewidth=2)
plt.title('Scree Plot')
plt.xlabel('Principal Component')
plt.ylabel('Proportion of Variance Explained')
plt.show()
```

Ideally, PCA is performed on datasets with a large number of columns, which might mean that a big proportion of them are irrelevant. In this case, we only have 7 columns out of which 4 are the key components that make up concrete:

1. Water
2. Coarse aggr.
3. Fine aggr.
4. Cement

However, we can see in the scree plot that the first 4 PCs explain 83.6% of the variance. We can make an educated guess that the part discarded could be a combination of the additional chemicals added to concrete such as superplasticizer, fly ash which we saw had a positive correlation with strength of the cement and could contribute to it durability.

In [14]:
```
sum(pca_concrete.explained_variance_ratio_[:4])
```

Out[14]:
```
0.8361098405885852
```

In [15]:
```
#creating a dataframe with the pca components of the all the samples
pc_concrete_df = pd.DataFrame(data=principalComp_concrete, columns=['pc_'+str(i) for i in range(1,8)])
pc_concrete_df = pc_concrete_df.drop(columns=["pc_5","pc_6","pc_7"])
pc_concrete_df
```

Out[15]:

	pc_1	pc_2	pc_3	pc_4
0	0.267519	0.048998	-0.944112	0.746884
1	-0.488953	1.844713	0.832270	-0.288779
2	-0.452798	2.699056	0.902533	-0.783674
3	-0.398914	3.341436	0.950733	-1.136015
4	-0.490634	0.849283	0.053376	1.469176
...	...	...	...	...
98	-2.013182	0.087107	-0.849752	0.523874
99	-1.966723	0.083172	-0.798182	0.502728
100	-1.710631	-0.174662	-0.748766	0.584401
101	-0.943042	-1.026062	-0.638782	0.746259
102	1.099560	-1.353299	0.336996	-0.344661

103 rows × 4 columns

This dataframe can be further used for regression. (PC Regression)

Modelling

In [16]:
```
from sklearn.linear_model import Lasso
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn import metrics
```

In [17]:
```
temp.columns
```

Out[17]:
```
Index(['Cement', 'Slag', 'Fly ash', 'Water', 'SP', 'Coarse Aggr.',
       'Fine Aggr.', 'Compressive Strength (28-day)(Mpa)'],
      dtype='object')
```

In [18]:
```
train_conc,test_conc = train_test_split(temp,test_size =.3,random_state = 23)
train_X = train_conc[[x for x in train_conc.columns if x not in ["Compressive Strength (28-day)(Mpa)"]]]
train_Y = train_conc["Compressive Strength (28-day)(Mpa)"]
test_X  = test_conc[[x for x in test_conc.columns if x not in ["Compressive Strength (28-day)(Mpa)"]]]
test_Y  = test_conc["Compressive Strength (28-day)(Mpa)"]
```

In [19]:
```
#trying the lasso model

lasso_reg = Lasso(alpha=0.1)
lasso_model = lasso_reg.fit(train_X, train_Y)
y_pred = lasso_reg.predict(test_X)

lasso_score=lasso_model.score(test_X, test_Y)
rmse=np.sqrt(mean_squared_error(y_pred, test_Y))

print('Accuracy of model is', lasso_model.score(test_X, test_Y))
print('Mean Absolute Error:', metrics.mean_absolute_error(test_Y, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(test_Y, y_pred))
print('Root Mean Squared Error:', np.sqrt(mean_squared_error(test_Y, y_pred)))
```

```
Accuracy of model is 0.9111405896672247
Mean Absolute Error: 1.8841709624028053
Mean Squared Error: 5.876247451247834
Root Mean Squared Error: 2.42409724459392

C:\Users\Mitsy\anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:476: ConvergenceWarning: Objec
tive did not converge. You might want to increase the number of iterations. Duality gap: 1.2105567425626873, toleranc
e: 0.4204890644444444
  positive)
```

In [20]:
```
#trying the Linear regression model for comparison

lm_reg = LinearRegression()
lm_model = lm_reg.fit(train_X, train_Y)
y_pred2 = lm_reg.predict(test_X)
lm_score =lm_reg.score(test_X, test_Y)
rmse_lm =np.sqrt(mean_squared_error(test_Y, y_pred2))
print('Accuracy of model is', lm_model.score(test_X, test_Y))
print('Mean Absolute Error:', metrics.mean_absolute_error(y_pred2, test_Y))
print('Mean Squared Error:', metrics.mean_squared_error(y_pred2, test_Y))
print('Root Mean Squared Error:', np.sqrt(mean_squared_error(test_Y, y_pred2)))
```

```
Accuracy of model is 0.9051803290283622
Mean Absolute Error: 1.9386730361407418
Mean Squared Error: 6.270397786667847
Root Mean Squared Error: 2.5040762341965244
```

Thus we can see that the lasso model does marginally better than the linear regression model when tested on the same dataset.

In [ ]:

```

```