Predicting Compressive Strength of Concrete

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import itertools
%matplotlib inline
```

In [ ]:
```
pip install pyforest
```

```
Requirement already satisfied: pyforest in /usr/local/lib/python3.7/dist-pac
kages (1.1.0)
```

In [ ]:
```
from pyforest import*
lazy_imports()
```

Out[ ]:
```
['from sklearn.decomposition import PCA',
 'import statsmodels.api as sm',
 'from fbprophet import Prophet',
 'import gensim',
 'import plotly.graph_objs as go',
 'import xgboost as xgb',
 'from dask import dataframe as dd',
 'from sklearn.model_selection import StratifiedKFold',
 'import tensorflow as tf',
 'import tqdm',
 'from sklearn.linear_model import RidgeCV',
 'from sklearn.ensemble import GradientBoostingClassifier',
 'import sys',
 'import os',
 'from sklearn.preprocessing import StandardScaler',
 'from sklearn.linear_model import LassoCV',
 'from scipy import signal as sg',
 'from sklearn.preprocessing import OneHotEncoder',
 'from sklearn.manifold import TSNE',
 'from sklearn.linear_model import ElasticNet',
 'from scipy import stats',
 'from sklearn.preprocessing import RobustScaler',
 'from sklearn.preprocessing import LabelEncoder',
 'from sklearn.preprocessing import MinMaxScaler',
 'import glob',
 'import dash',
 'from pyspark import SparkContext',
 'from sklearn.model_selection import RandomizedSearchCV',
 'from sklearn.linear_model import LogisticRegression',
 'from sklearn.linear_model import LinearRegression',
 'from sklearn.preprocessing import PolynomialFeatures',
 'from sklearn.linear_model import Lasso',
 'from sklearn.linear_model import ElasticNetCV',
 'from sklearn.feature_extraction.text import TfidfVectorizer',
 'import textblob',
 'import spacy',
 'from pathlib import Path',
 'import awswrangler as wr',
 'import statistics',
 'import datetime as dt',
 'from xlrd import open_workbook',
 'import torch',
 'import skimage',
 'from sklearn.model_selection import GridSearchCV',
 'from sklearn.ensemble import RandomForestClassifier',
 'import sklearn',
 'from sklearn.linear_model import Ridge',
 'from sklearn import svm']
```

In [ ]:
```
df = pd.read_csv('/content/compresive_strength_concrete+2 (1).csv')
```

In [ ]:
```
df.head()
```

Out[ ]:

	Cement (component 1)(kg in a m^3 mixture)	Blast Furnace Slag (component 2)(kg in a m^3 mixture)	Fly Ash (component 3)(kg in a m^3 mixture)	Water (component 4)(kg in a m^3 mixture)	Superplasticizer (component 5) (kg in a m^3 mixture)	Agg (com 6) m
0	540.0	0.0	0.0	162.0	2.5	
1	540.0	0.0	0.0	162.0	2.5	
2	332.5	142.5	0.0	228.0	0.0	
3	332.5	142.5	0.0	228.0	0.0	
4	198.6	132.4	0.0	192.0	0.0	

Observation

1. It shows that there are eight independent variables(cement, slag, ash,water,superplastic,coarseagg,fineagg,age)and one dependent variable(strength)
2. All the records are numeric

In [ ]:
```
df = df.rename(columns={'Cement (component 1)(kg in a m^3 mixture)':'cement'
                        'Blast Furnace Slag (component 2)(kg in a m^3 mixtur
                        'Fly Ash (component 3)(kg in a m^3 mixture)':'ash',
                        'Water (component 4)(kg in a m^3 mixture)':'water',
                        "Superplasticizer (component 5)(kg in a m^3 mixture)
                        'Coarse Aggregate (component 6)(kg in a m^3 mixture)
                        'Fine Aggregate (component 7)(kg in a m^3 mixture)':
                        'Age (day)':'age',
                        'Concrete compressive strength(MPa, megapascals)':'s
})
```

Alternatively:

In [ ]:
```
df.columns = ['cement', 'slag', 'ash', 'Water','superplastic','coarseagg','f
```

In [ ]:
```
df.head()
```

Out[ ]:

	cement	slag	ash	Water	superplastic	coarseagg	fineagg	age	strengtl
0	540.0	0.0	0.0	162.0	2.5	1040.0	676.0	28	79.9
1	540.0	0.0	0.0	162.0	2.5	1055.0	676.0	28	61.8
2	332.5	142.5	0.0	228.0	0.0	932.0	594.0	270	40.2
3	332.5	142.5	0.0	228.0	0.0	932.0	594.0	365	41.0
4	198.6	132.4	0.0	192.0	0.0	978.4	825.5	360	44.3

In [ ]:
```
df.dtypes
```

Out[ ]:
```
cement          float64
slag            float64
ash             float64
Water           float64
superplastic    float64
coarseagg       float64
fineagg         float64
age               int64
strength        float64
dtype: object
```

In [ ]:
```
df.shape
```

Out[ ]:
```
(1030, 9)
```

In [ ]:
```
#Checking for missing values

df.isnull().sum()
```

Out[ ]:
```
cement          0
slag            0
ash             0
Water           0
superplastic    0
coarseagg       0
fineagg         0
age             0
strength        0
dtype: int64
```

In [ ]:
```
df.describe().T
```

Out[ ]:

	count	mean	std	min	25%	50%	75%
cement	1030.0	281.167864	104.506364	102.00	192.375	272.900	350.00
slag	1030.0	73.895825	86.279342	0.00	0.000	22.000	142.95
ash	1030.0	54.188350	63.997004	0.00	0.000	0.000	118.30
Water	1030.0	181.567282	21.354219	121.80	164.900	185.000	192.00
superplastic	1030.0	6.204660	5.973841	0.00	0.000	6.400	10.20
coarseagg	1030.0	972.918932	77.753954	801.00	932.000	968.000	1029.40
fineagg	1030.0	773.580485	80.175980	594.00	730.950	779.500	824.00
age	1030.0	45.662136	63.169912	1.00	7.000	28.000	56.00
strength	1030.0	35.817961	16.705742	2.33	23.710	34.445	46.13

Exploratory Data Analysis

CEMENT

In [ ]:
```
#Quartiles
from scipy import stats

Q1=df['cement'].quantile(q=0.25)
Q3=df['cement'].quantile(q=0.75)

print('1st Quartile (Q1) is: ',Q1)
print('3rd Quartile (Q3) is: ',Q3)
print('Interquartile range (IQR) is ', stats.iqr(df['cement']))
```

```
1st Quartile (Q1) is:  192.375
3rd Quartile (Q3) is:  350.0
Interquartile range (IQR) is  157.625
```

In [ ]:
```
#Outlier detection from Interquartile range (IQR) in original data


L_outliers=Q1-1.5*(Q3-Q1)
U_outliers=Q3+1.5*(Q3-Q1)
print('Lower outlier limit in cement: ',L_outliers)
print('Upper outlier limit in cement: ',U_outliers)
```

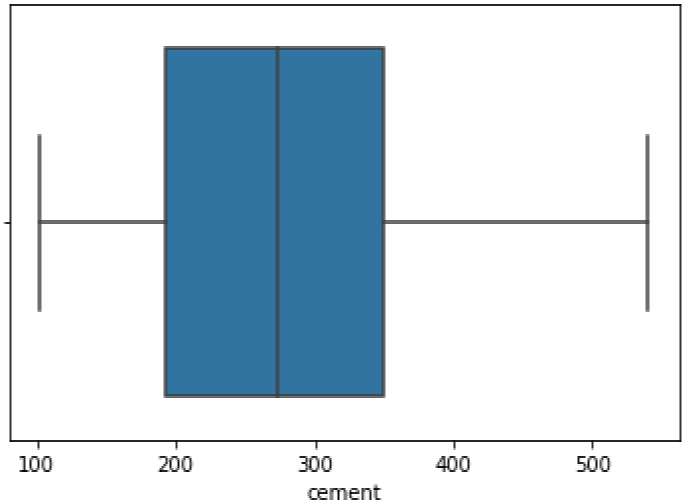
```
Lower outlier limit in cement:  -44.0625
Upper outlier limit in cement:  586.4375
```

In [ ]:
```
#Checking for presenece of outliers with the upper and lower limits

print('Number of outliers in cement upper: ', df[df['cement']>586.4375]['cem
print('Number of outliers in cement lower: ', df[df['cement']<-44.0625]['cem

# print('% of Outlier in cement upper: ', round(df[df['cement']>586.4375]['c
# print('% of Outlier in cement lower: ', round(df[df['cement']<-44.0625]['c
```

```
Number of outliers in cement upper:  0
Number of outliers in cement lower:  0
```

In [ ]:
```
#Distribution of CEMENT

sns.boxplot(x='cement',data=df, orient='h')
```

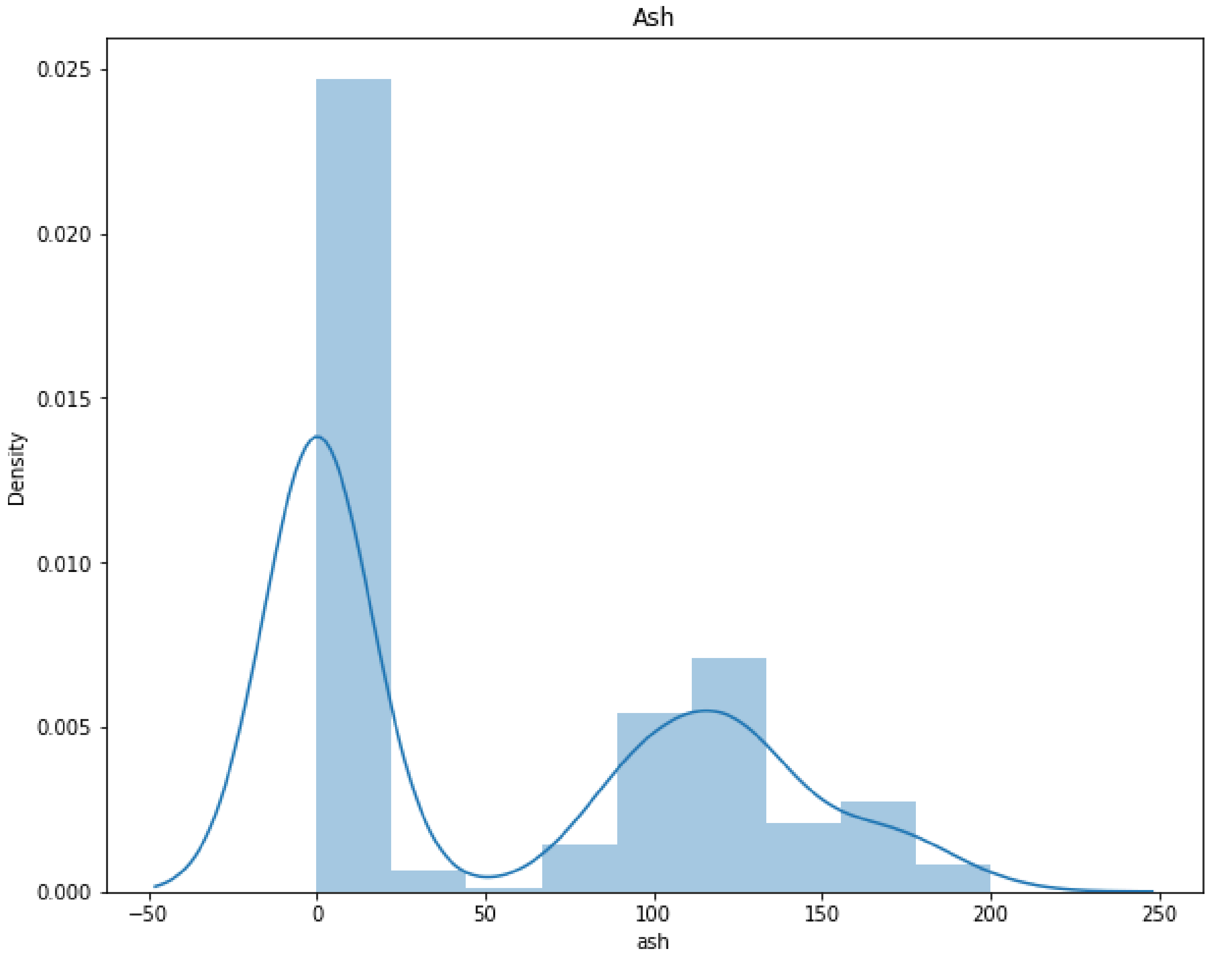
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7fbc4d92cd10>



In [ ]:

```
#displot

plt.figure(figsize=(10,8))
sns.distplot(df['ash']).set_title('Ash')
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: Future
Warning:

`distplot` is a deprecated function and will be removed in a future version.
Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).
```



Water

In [ ]:
```
#Quartiles

w_Q1=df['Water'].quantile(q=0.25)
w_Q3=df['Water'].quantile(q=0.75)

print('1st Quartile (Q1) is: ', w_Q1)
print('3rd Quartile (Q3) is: ', w_Q3)
print('Interquartile range (IQR) is: ', stats.iqr(df['Water']))
```

```
1st Quartile (Q1) is:  164.9
3rd Quartile (Q3) is:  192.0
Interquartile range (IQR) is:  27.099999999999994
```

In [ ]:
```
#Outlier detection from Interquartile range (IQR) in original data

WL_outliers=w_Q1-1.5*(w_Q3-w_Q1)
WU_outliers=w_Q3+1.5*(w_Q3-w_Q1)

print('Lower outlier in water: ',WL_outliers)
print('Upper outlier in water: ',WU_outliers)
```

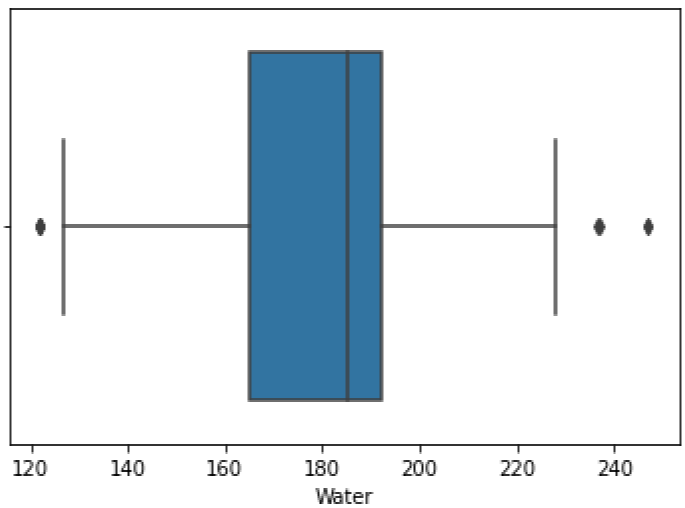
```
Lower outlier in water:  124.25000000000001
Upper outlier in water:  232.64999999999998
```

In [ ]:
```
#Checking for presenece of outliers with the upper and lower limits

print('Number of outliers in water upper: ', df[df['Water']>232.649999999999
print('Number of outliers in water lower: ', df[df['Water']<124.250000000000

# print('% of Outlier in water upper: ', round(df[df['Water']>232.6499999999
# print('% of Outlier in water lower: ', round(df[df['Water']<124.2500000000
```

```
Number of outliers in water upper:  4
Number of outliers in water lower:  5
```

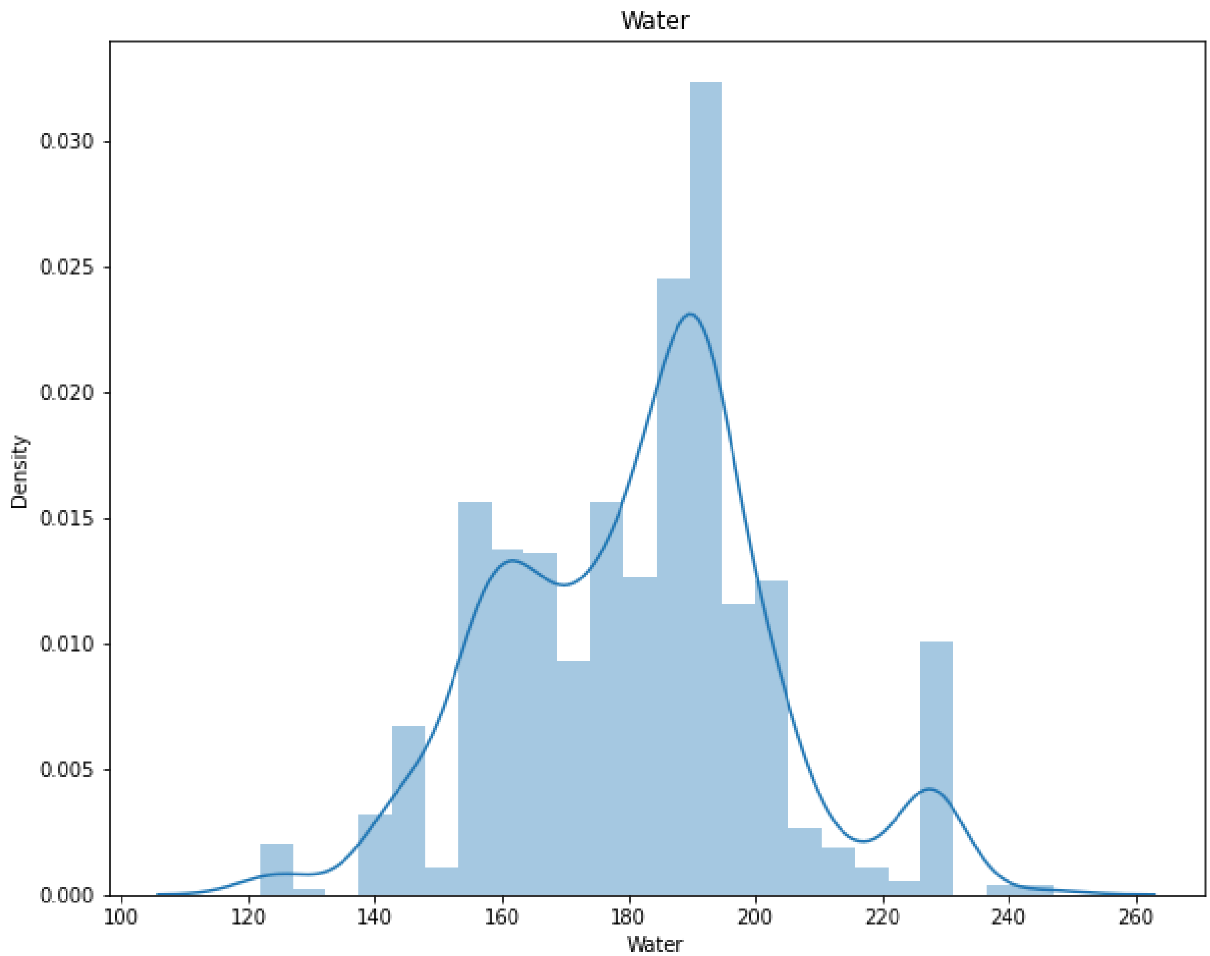
In [ ]:
```
#Distribution of WATER

sns.boxplot(x='Water', data=df, orient='h')
```

Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7fbc4d7c8a90>



In [ ]:
```
plt.figure(figsize=(10,8))
sns.distplot(df['Water']).set_title('Water')
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: Future
Warning:

`distplot` is a deprecated function and will be removed in a future version.
Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).
```



Slag

1st Quartile (Q1) is: 0.0

3st Quartile (Q3) is: 142.95

Interquartile range (IQR) is 142.95

In [ ]:
```
Q1=df['slag'].quantile(q=0.25)
Q3=df['slag'].quantile(q=0.75)
```

In [ ]:
```
#Outlier detection from Interquartile range (IQR) in original data

L_outliers=Q1-1.5*(Q3-Q1)
U_outliers=Q3+1.5*(Q3-Q1)

print('Lower outlier in water: ',L_outliers)
print('Upper outlier in water: ',U_outliers)
```

```
Lower outlier in water:  -214.42499999999998
Upper outlier in water:  357.375
```

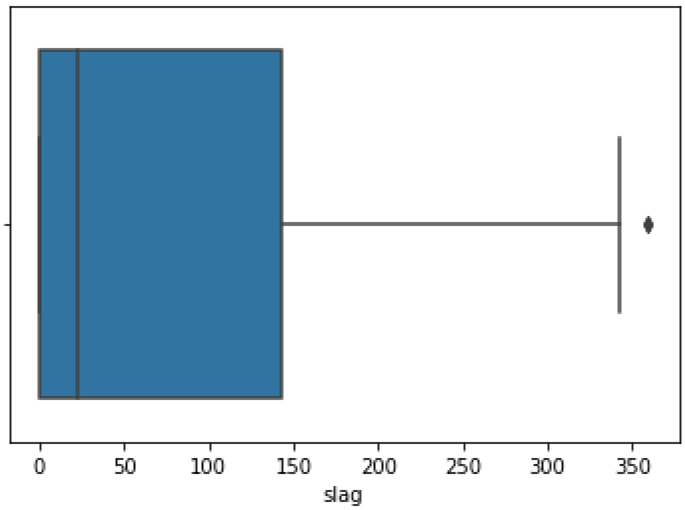
In [ ]:
```
#Checking for presenece of outliers with the upper and lower limits

print('Number of outliers in slag upper: ', df[df['slag']>357.375]['slag'].c
print('Number of outliers in slag lower: ', df[df['slag']<-214.425]['slag'].
```

```
# print('% of Outlier in slag upper: ', round(df[df['slag']>357.375]['slag']
# print('% of Outlier in slag lower: ', round(df[df['slag']<-214.425]['slag'
```

```
Number of outliers in slag upper:  2
Number of outliers in slag lower:  0
```

In [ ]:
```
#Distribution of SLAG

sns.boxplot(x='slag', data=df, orient='h')
```

Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7fbc4d6f5e50>



Age

Minimum age: 1

Maximum age: 365

Mean value: 45.662135922330094

Median value: 28.0

Standard deviation: 63.16991581033155

Null values: False

In [ ]:
```
Q1=df['age'].quantile(q=0.25)
Q3=df['age'].quantile(q=0.75)
```

In [ ]:
```
#Outlier detection from Interquartile range (IQR) in original data

L_outliers=Q1-1.5*(Q3-Q1)
U_outliers=Q3+1.5*(Q3-Q1)

print('Lower outlier in age: ',L_outliers)
print('Upper outlier in age: ',U_outliers)
```

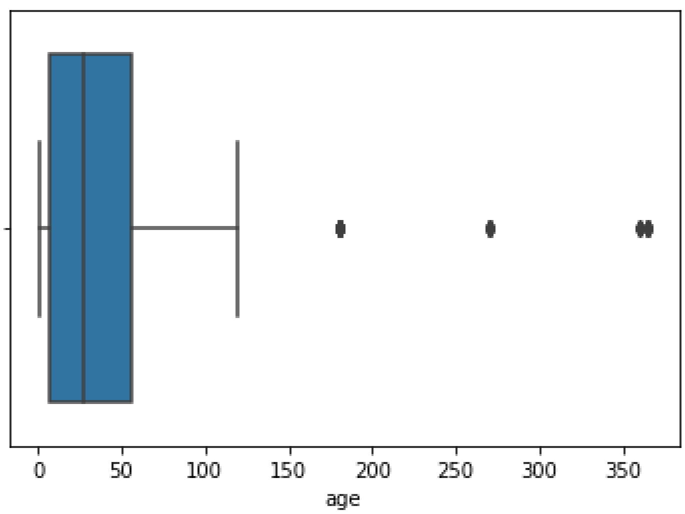
```
Lower outlier in age:  -66.5
Upper outlier in age:  129.5
```

In [ ]:
```
#Checking for presenece of outliers with the upper and lower limits

print('Number of outliers in age upper: ', df[df['age']>129.5]['age'].count(
print('Number of outliers in age lower: ', df[df['age']<-66.5]['age'].count(
```

```
Number of outliers in age upper:  59
Number of outliers in age lower:  0
```

In [ ]:
```
#Distribution of AGE

sns.boxplot(x='age', data=df, orient='h')
```

Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7fbc4d670250>



ASH

In [ ]:
```
Q1=df['ash'].quantile(q=0.25)
Q3=df['ash'].quantile(q=0.75)
```

In [ ]:
```
#Outlier detection from Interquartile range (IQR) in original data

L_outliers=Q1-1.5*(Q3-Q1)
U_outliers=Q3+1.5*(Q3-Q1)

print('Lower outlier in ash: ',L_outliers)
print('Upper outlier in ash: ',U_outliers)
```

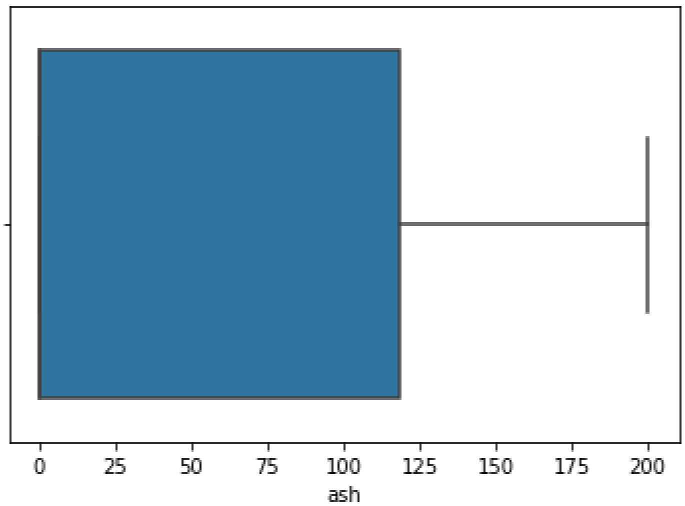
```
Lower outlier in ash:  -177.45
Upper outlier in ash:  295.75
```

In [ ]:
```
#Checking for presenece of outliers with the upper and lower limits

print('Number of outliers in ash upper: ', df[df['ash']>295.75]['ash'].count
print('Number of outliers in ash lower: ', df[df['ash']<-177.45]['ash'].cour
```

```
Number of outliers in ash upper:  0
Number of outliers in ash lower:  0
```

In [ ]:
```
#Distribution of AGE

sns.boxplot(x='ash', data=df, orient='h')
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fbc4d5db0d0>
```



In [ ]:

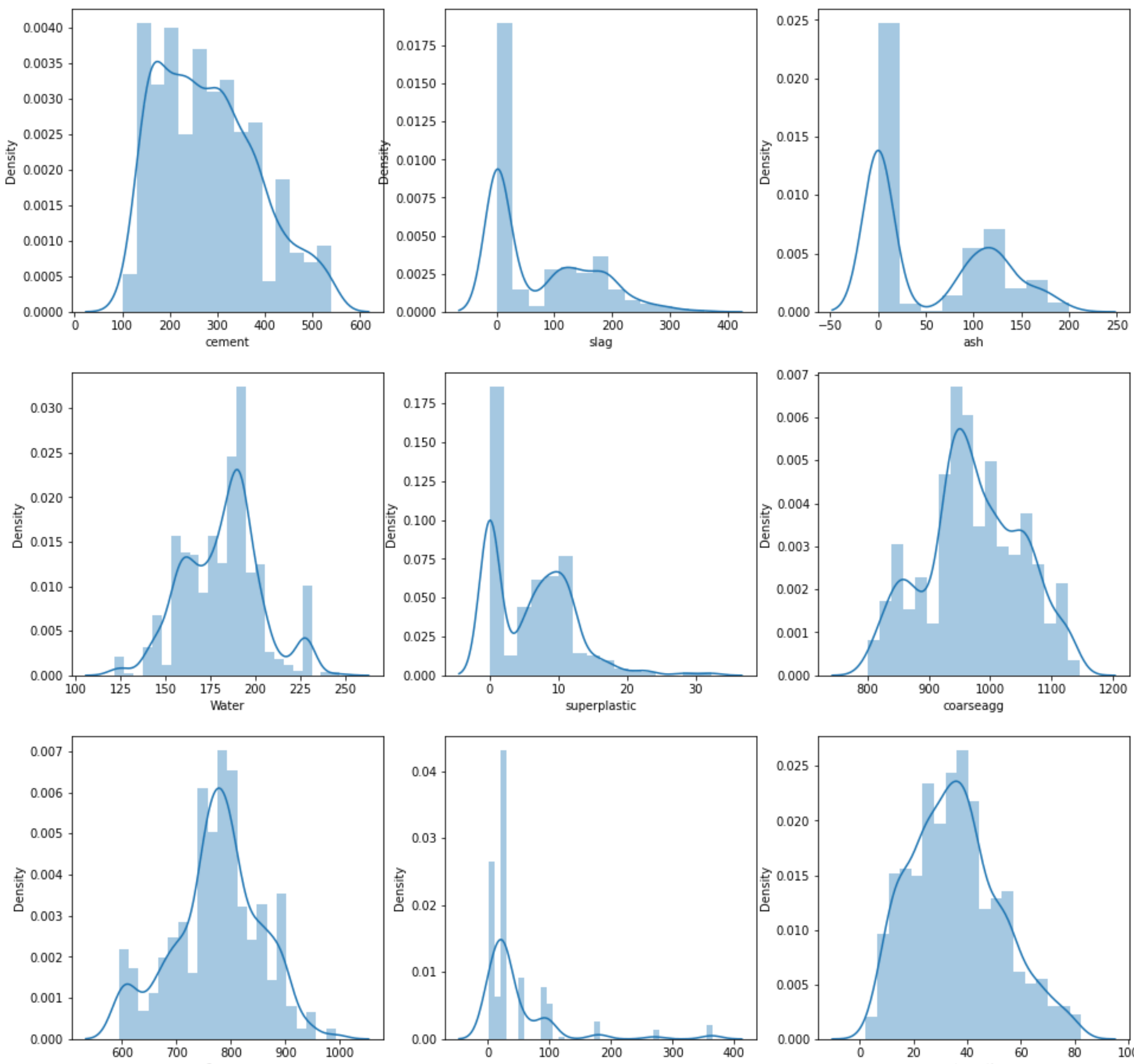
MultiVaariate Analysis

In [ ]:
```
#Displot

fig,ax2 = plt.subplots(3,3,figsize=(16,16))
sns.distplot(df['cement'],ax=ax2[0][0])
sns.distplot(df['slag'],ax=ax2[0][1])
sns.distplot(df['ash'],ax=ax2[0][2])
sns.distplot(df['Water'],ax=ax2[1][0])
sns.distplot(df['superplastic'],ax=ax2[1][1])
sns.distplot(df['coarseagg'],ax=ax2[1][2])
sns.distplot(df['fineagg'],ax=ax2[2][0])
sns.distplot(df['age'],ax=ax2[2][1])
sns.distplot(df['strength'],ax=ax2[2][2])
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: Future
Warning:

`distplot` is a deprecated function and will be removed in a future version.
Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: Future
Warning:

`distplot` is a deprecated function and will be removed in a future version.
Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: Future
Warning:

`distplot` is a deprecated function and will be removed in a future version.
Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: Future
Warning:

`distplot` is a deprecated function and will be removed in a future version.
Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: Future
Warning:

`distplot` is a deprecated function and will be removed in a future version.
Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: Future
Warning:

`distplot` is a deprecated function and will be removed in a future version.
Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: Future
Warning:

`distplot` is a deprecated function and will be removed in a future version.
Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: Future
Warning:

`distplot` is a deprecated function and will be removed in a future version.
Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: Future
Warning:

`distplot` is a deprecated function and will be removed in a future version.
Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fbc4d3a6c50>
```



Observation

We can see observe that :

cement is almost normal.

slag has three gausssians and rightly skewed.

ash has two gaussians and rightly skewed.

water has three guassians and slighly left skewed.

superplastic has two gaussians and rightly skewed.

coarseagg has three guassians and almost normal.

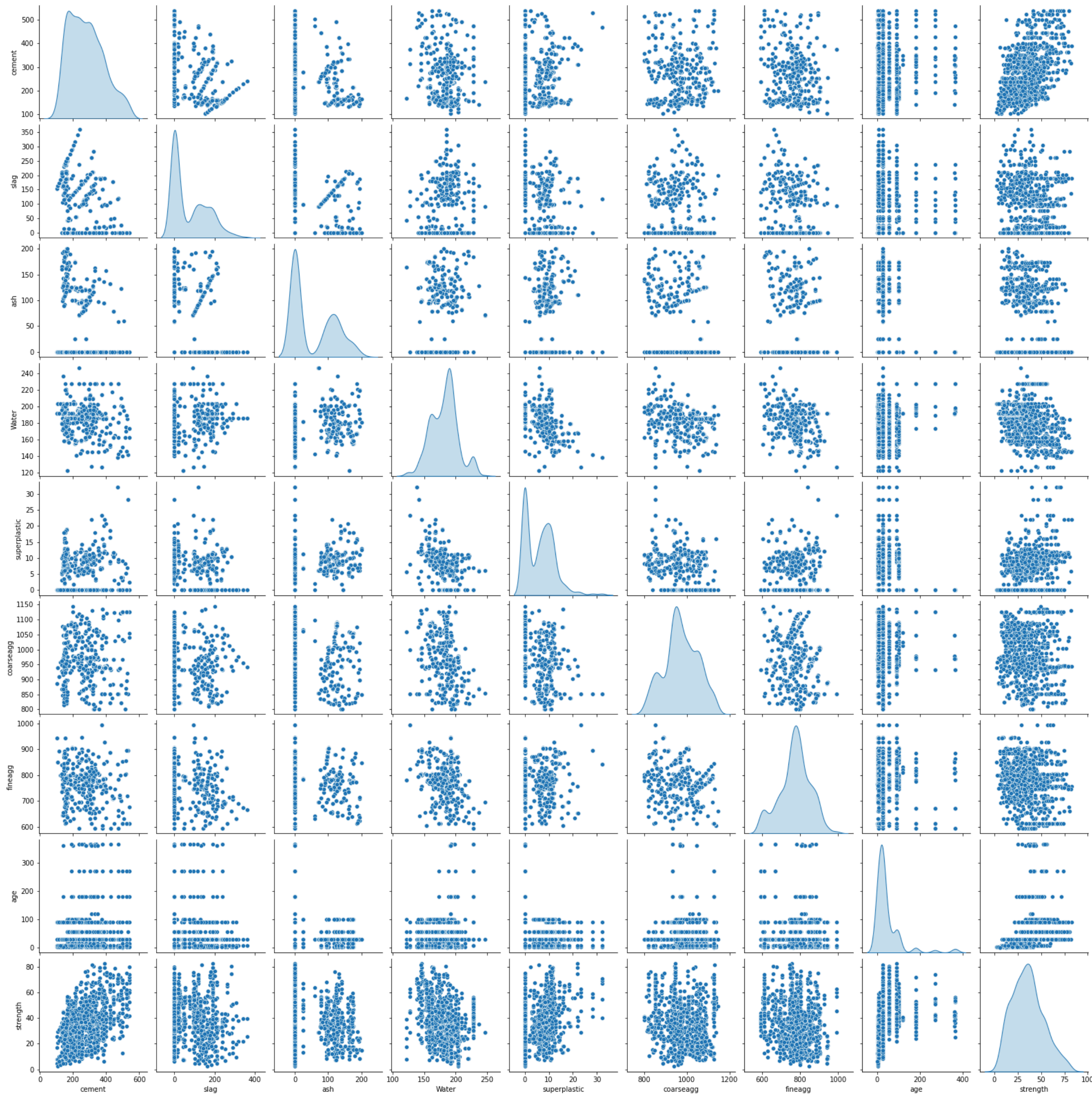
fineagg has almost two guassians and looks like normal.

age has multiple guassians and rightly skewed.

Pairplot

In [ ]:
```
# pairplot.
#plot density curve instead of histogram in the diagonals

sns.pairplot(df, diag_kind='kde')
```

Out[ ]: <seaborn.axisgrid.PairGrid at 0x7fbc4d54e310>



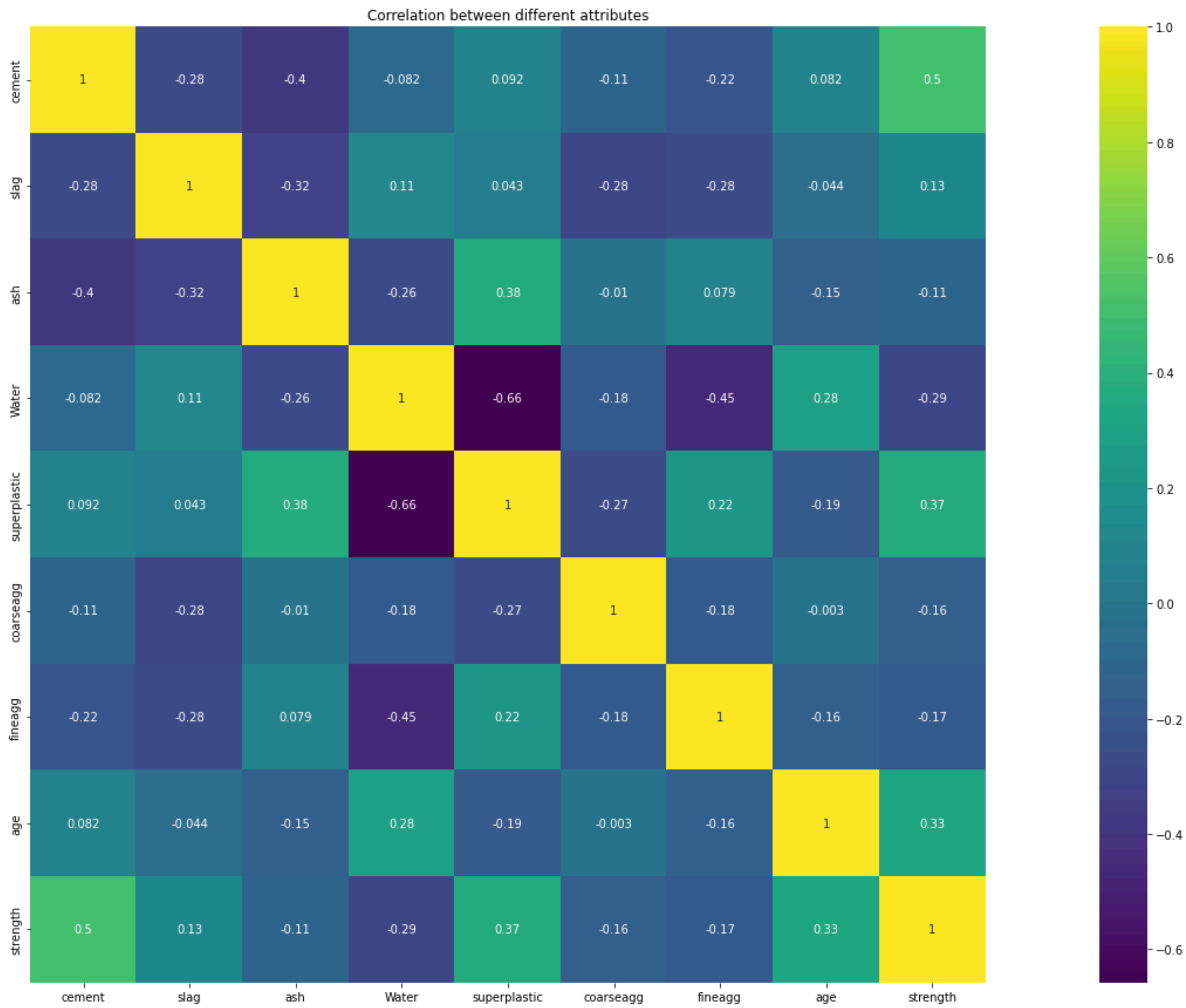
Correlation between variables

```
[37]: plt.figure(figsize=(35,15))

sns.heatmap(df.corr(), vmax=1, square=True, annot=True,cmap='viridis')
plt.title('Correlation between different attributes')
plt.show()
```

...

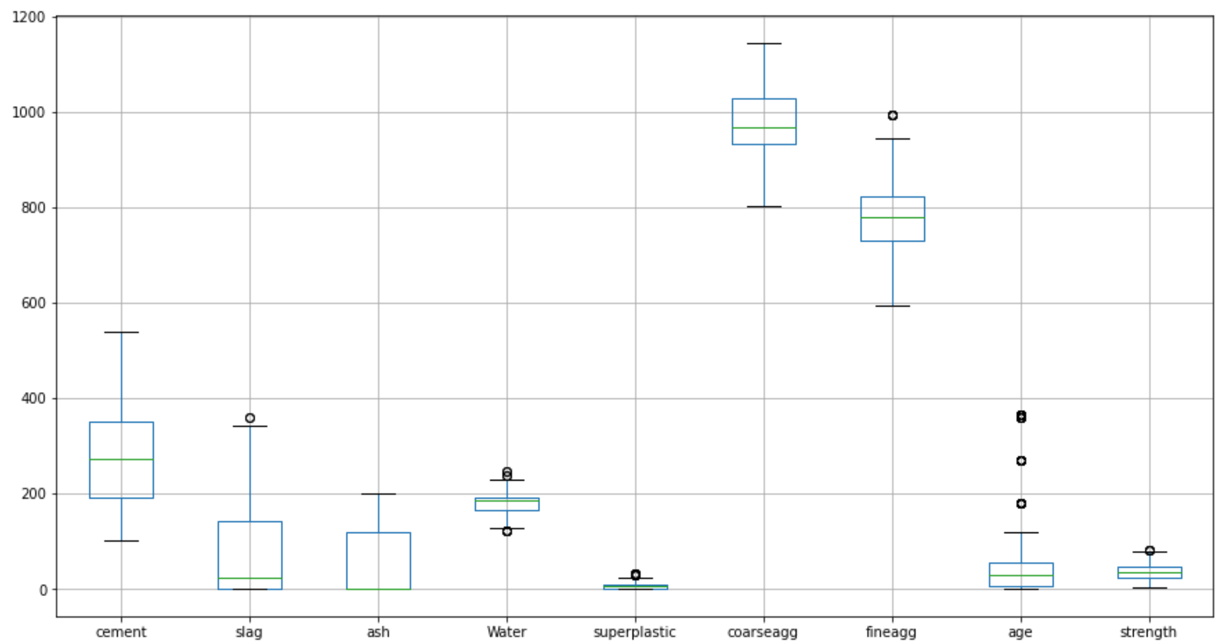
Correlation between variables

In [ ]:
```
plt.figure(figsize=(35,15))

sns.heatmap(df.corr(),vmax=1, square=True, annot=True, cmap='viridis')
plt.title('Correlation between different attributes')
plt.show()
```




```
In [ ]: df.boxplot(figsize=(15,8))
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7fbc4b1b6d10>
```



Checking for outliers

In [ ]:

```
print('Outliers in cement: ', df[((df.cement - df.cement.mean())/df.cement.s
print('Outliers in slag: ', df[((df.slag - df.slag.mean())/df.slag.std()).ab
print('Outliers in ash: ', df[((df.ash - df.ash.mean())/df.ash.std()).abs()>
print('Outliers in water: ', df[((df.Water - df.Water.mean())/df.Water.std()
print('Outliers in superplastic: ', df[((df.superplastic - df.superplastic.m
print('Outliers in coarseagg: ', df[((df.coarseagg - df.coarseagg.mean())/df
print('Outliers in fineagg: ', df[((df.fineagg - df.fineagg.mean())/df.finea
print('Outliers in age: ', df[((df.age - df.age.mean())/df.age.std()).abs()>
```

```
Outliers in cement:  0
Outliers in slag:  4
Outliers in ash:  0
Outliers in water:  2
Outliers in superplastic:  10
Outliers in coarseagg:  0
Outliers in fineagg:  0
Outliers in age:  33
```

Replacing the outliers by median

In [ ]:

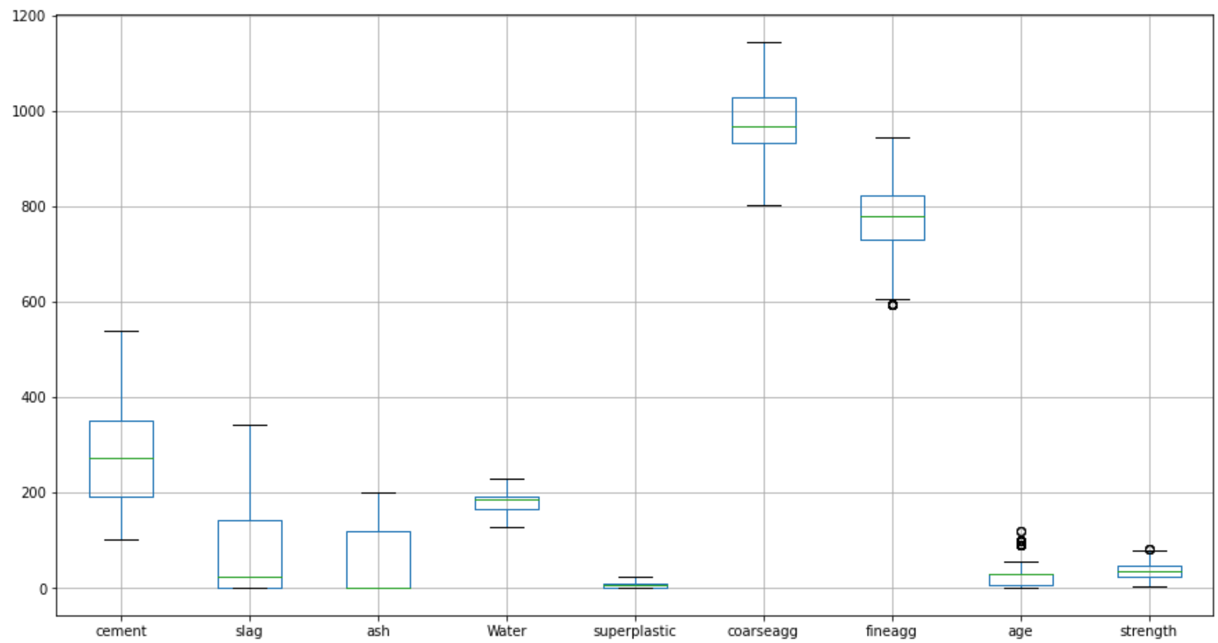
```
for cols in df.columns[:-1]:
  Q1 = df[cols].quantile(0.25)
  Q3 = df[cols].quantile(0.75)
  iqr = Q3 - Q1

  low = Q1-1.5*iqr
  high = Q3+1.5*iqr
  df.loc[(df[cols] < low) | (df[cols] > high), cols] = df[cols].median()
```

In [ ]:

```
df.boxplot(figsize=(15,8))
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fbc4af95650>
```



Feature Engineering and Model Building

In [ ]:
```
df.head()
```

Out[ ]:

	cement	slag	ash	Water	superplastic	coarseagg	fineagg	age	strengtl
0	540.0	0.0	0.0	162.0	2.5	1040.0	676.0	28	79.9
1	540.0	0.0	0.0	162.0	2.5	1055.0	676.0	28	61.8
2	332.5	142.5	0.0	228.0	0.0	932.0	594.0	28	40.2
3	332.5	142.5	0.0	228.0	0.0	932.0	594.0	28	41.0
4	198.6	132.4	0.0	192.0	0.0	978.4	825.5	28	44.3

In [ ]:
```
#Splitting the data into independent and dependent attributes


#independent and dependent variables
X = df.drop('strength', axis = 1)
y = df['strength']
```

In [ ]:
```
from scipy.stats import zscore

Xscaled = X.apply(zscore)
Xscaled_df = pd.DataFrame(Xscaled, columns=df.columns)
```

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(Xscaled,y, test_size= 0.
```

Building different Models

Random Forest

In [ ]:
```
model = RandomForestRegressor()
model.fit(X_train, y_train)
```

Out[ ]:
```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features='auto', max_leaf_nodes=N
one,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=100, n_jobs=None, oob_score=False,
                      random_state=None, verbose=0, warm_start=False)
```

In [ ]:
```
y_pred = model.predict(X_test)
```

In [ ]:
```
#Model Performance on Training Data

model.score(X_train, y_train)

# round(model.score(X_train, y_train)*100) #if you want to get the exact per
```

Out[ ]:
```
0.9811245610730347
```

In [ ]:
```
#Model Performance on Test Data

model.score(X_test, y_test)

# round(model.score(X_test, y_test)*100) #if you want to get the exact perce
```

Out[ ]:
```
0.8717535758767027
```

In [ ]:
```
#Same as above
acc_R=metrics.r2_score(y_test, y_pred)
acc_R
```

Out[ ]:
```
0.8717535758767028
```

In [ ]:
```
metrics.mean_squared_error(y_test, y_pred)
```

Out[ ]:
```
33.61431101044406
```

In [ ]:
```
#Store the accuracy results for each model in a dataframe for final comparis

results_1 = pd.DataFrame({'Algorithm': ['Random Forest'], 'accuracy': acc_R}
results = results_1[['Algorithm','accuracy']]
results
```

Out[ ]:

	Algorithm	accuracy
1	Random Forest	0.871754

KFold Cross Validation

In [ ]:
```
k = 20

kfold = KFold(n_splits=k, random_state=70)
K_results = cross_val_score(model, X, y, cv=kfold)
accuracy=np.mean(abs(K_results))
accuracy
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_split.py:29
6: FutureWarning:

Setting a random_state has no effect since shuffle is False. This will raise
an error in 0.24. You should leave random_state to its default (None), or se
t shuffle=True.
```

Out[ ]: 0.7589960174216859

In [ ]:
```
#Store the accuracy results for each model in a dataframe for final comparis

random_re = pd.DataFrame({'Algorithm': ['Random Forest Regressor k_fold'], '
results = pd.concat([results, random_re])
results = results[['Algorithm','accuracy']]
results
```

Out[ ]:

	Algorithm	accuracy
1	Random Forest	0.871754
2	Random Forest Regressor k_fold	0.758996

Gradient Boosting Regressor

In [ ]:
```
model = GradientBoostingRegressor()
model.fit(X_train, y_train)
```

```
Out[ ]: GradientBoostingRegressor(alpha=0.9, ccp_alpha=0.0, criterion='friedman_ms
e',
                          init=None, learning_rate=0.1, loss='ls', max_dept
h=3,
                          max_features=None, max_leaf_nodes=None,
                          min_impurity_decrease=0.0, min_impurity_split=Non
e,
                          min_samples_leaf=1, min_samples_split=2,
                          min_weight_fraction_leaf=0.0, n_estimators=100,
                          n_iter_no_change=None, presort='deprecated',
                          random_state=None, subsample=1.0, tol=0.0001,
                          validation_fraction=0.1, verbose=0, warm_start=Fa
lse)
```

```
In [ ]: y_pred = model.predict(X_test)
```

```
In [ ]: #Model Performance on Training Data

        model.score(X_train, y_train)
```

```
Out[ ]: 0.947736861039059
```

NB: output might change when you run the notebook at different times

```
In [ ]: #Model Performance on Test Data

        model.score(X_test, y_test)
```

```
Out[ ]: 0.8805238132226646
```

```
In [ ]: #Same as above, you can also store the above in a variable and use without c
        acc_G=metrics.r2_score(y_test, y_pred)
        acc_G
```

```
Out[ ]: 0.8805238132226646
```

```
In [ ]: metrics.mean_squared_error(y_test, y_pred)
```

```
Out[ ]: 31.315568665011156
```

```
In [ ]: #Store the accuracy results for each model in a dataframe for final comparis

        gradient_re = pd.DataFrame({'Algorithm': ['Gradient Boost Regressor'], 'accu
        results = pd.concat([results, gradient_re])
        results = results[['Algorithm','accuracy']]
        results
```

Out[ ]:

	Algorithm	accuracy
1	Random Forest	0.871754
2	Random Forest Regressor k_fold	0.758996
3	Gradient Boost Regressor	0.880524

In [ ]:
```
k = 20

kfold = KFold(n_splits=k, random_state=70)
results_3 = cross_val_score(model, X, y, cv=kfold)
accuracy=np.mean(abs(results_3))
accuracy
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_split.py:29
6: FutureWarning:

Setting a random_state has no effect since shuffle is False. This will raise
an error in 0.24. You should leave random_state to its default (None), or se
t shuffle=True.
```

Out[ ]: 0.7693484121447565

In [ ]:
```
#Store the accuracy results for each model in a dataframe for final comparis

gradient_k = pd.DataFrame({'Algorithm': ['Gradient Boost Regressor k fold'],
results = pd.concat([results, gradient_k])
results = results[['Algorithm','accuracy']]
results
```

Out[ ]:

	Algorithm	accuracy
1	Random Forest	0.871754
2	Random Forest Regressor k_fold	0.758996
3	Gradient Boost Regressor	0.880524
4	Gradient Boost Regressor k fold	0.769348

Ada Boost Regressor

In [ ]:
```
from sklearn.ensemble import AdaBoostRegressor
```

In [ ]:
```
model = AdaBoostRegressor()
model.fit(X_train, y_train)
```

Out[ ]:
```
AdaBoostRegressor(base_estimator=None, learning_rate=1.0, loss='linear',
                  n_estimators=50, random_state=None)
```

In [ ]:
```
y_pred = model.predict(X_test)
```

In [ ]:
```
#Model Performance on Test Data, NB: check on train data

model.score(X_test, y_test)
```

Out[ ]: 0.7532979137277463

NB: the performance might vary when you run the notebook, that's a normal scenario

In [ ]:
```
#Same as above, you can also store the above in a variable and use without c
acc_Ada=metrics.r2_score(y_test, y_pred)
acc_Ada
```

Out[ ]:
```
0.7532979137277463
```

In [ ]:
```
#Store the accuracy results for each model in a dataframe for final comparis

acc_Ada = pd.DataFrame({'Algorithm': ['Ada Boost Regressor'], 'accuracy': ac
results = pd.concat([results, acc_Ada])
results = results[['Algorithm','accuracy']]
results
```

Out[ ]:

	Algorithm	accuracy
1	Random Forest	0.871754
2	Random Forest Regressor k_fold	0.758996
3	Gradient Boost Regressor	0.880524
4	Gradient Boost Regressor k fold	0.769348
5	Ada Boost Regressor	0.753298

K fold cross Validation for Ada Boost

In [ ]:
```
k = 20

kfold = KFold(n_splits=k, random_state=70)
results_4 = cross_val_score(model, X, y, cv=kfold)
accuracy=np.mean(abs(results_4))
accuracy
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_split.py:29
6: FutureWarning:

Setting a random_state has no effect since shuffle is False. This will raise
an error in 0.24. You should leave random_state to its default (None), or se
t shuffle=True.
```

Out[ ]:
```
0.5867887758796766
```

In [ ]:
```
#Store the accuracy results for each model in a dataframe for final comparis

acc_AdaC = pd.DataFrame({'Algorithm': ['Ada Boost Regressor k fold'], 'accur
results = pd.concat([results, acc_AdaC])
results = results[['Algorithm','accuracy']]
results
```

Out[ ]:

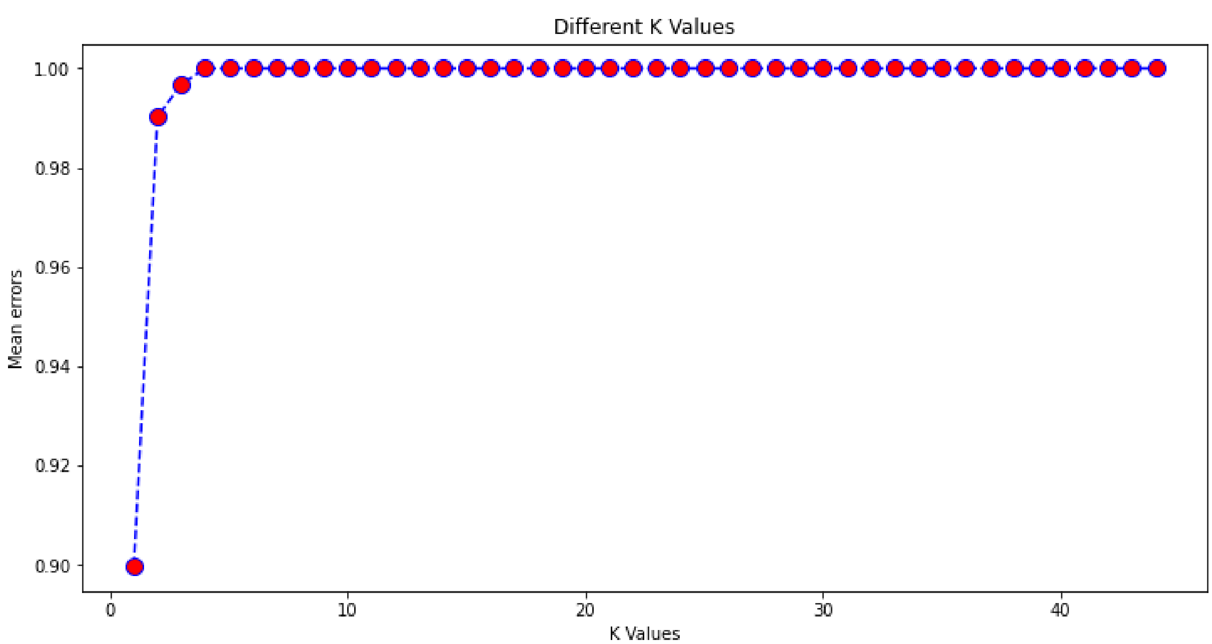
	Algorithm	accuracy
1	Random Forest	0.871754
2	Random Forest Regressor k_fold	0.758996
3	Gradient Boost Regressor	0.880524
4	Gradient Boost Regressor k fold	0.769348
5	Ada Boost Regressor	0.753298
6	Ada Boost Regressor k fold	0.586789

KNN Regressor

In [ ]:
```
#Checking for different values of neighbors to determine K
from sklearn.neighbors import KNeighborsRegressor


diff_k=[]
for i in range(1,45):
  knn = KNeighborsRegressor(n_neighbors=i)
  knn.fit(X_train, y_train)
  pred_i = knn.predict(X_test)
  diff_k.append(np.mean(pred_i != y_test))
```

In [ ]:
```
plt.figure(figsize=(12,6))
plt.plot(range(1,45),diff_k,color='blue',linestyle='dashed',marker='o',marke
plt.title('Different K Values')
plt.xlabel('K Values')
plt.ylabel('Mean errors')
```

Out[ ]: `Text(0, 0.5, 'Mean errors')`



In [ ]:
```
#k=3 is a better choice from the above plot

model = KNeighborsRegressor(n_neighbors=3)
model.fit(X_train, y_train)
```

Out[ ]:
```
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=None, n_neighbors=3, p=2,
                    weights='uniform')
```

In [ ]:
```
y_pred = model.predict(X_test)
```

In [ ]:
```
model.score(X_train, y_train)
```

Out[ ]:
```
0.9075702785732312
```

In [ ]:
```
acc_KNN=metrics.r2_score(y_test, y_pred)
acc_KNN
```

Out[ ]:
```
0.7539494934126327
```

In [ ]:
```
metrics.mean_squared_error(y_test, y_pred)
```

Out[ ]:
```
64.49160909744695
```

In [ ]:
```
KNN_df = pd.DataFrame({'Algorithm':['KNN Regressor'], 'accuracy': [acc_KNN]}
results = pd.concat([results, KNN_df])
results = results[['Algorithm','accuracy']]
results
```

Out[ ]:

	Algorithm	accuracy
1	Random Forest	0.871754
2	Random Forest Regressor k_fold	0.758996
3	Gradient Boost Regressor	0.880524
4	Gradient Boost Regressor k fold	0.769348
5	Ada Boost Regressor	0.753298
6	Ada Boost Regressor k fold	0.586789
7	KNN Regressor	0.753949

KFold Validation

In [ ]:
```
k = 20

kfold = KFold(n_splits=k, random_state=70)
results_5 = cross_val_score(model, X, y, cv=kfold)
accuracy=np.mean(abs(results_5))
accuracy
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_split.py:29
6: FutureWarning:

Setting a random_state has no effect since shuffle is False. This will raise
an error in 0.24. You should leave random_state to its default (None), or se
t shuffle=True.
```

Out[ ]:
```
0.6907106255855276
```

In [ ]:
```
#Store the accuracy results for each model in a dataframe for final comparis

KNNfold_df = pd.DataFrame({'Algorithm': ['KNN Regressor k fold'], 'accuracy'
results = pd.concat([results, KNNfold_df])
results = results[['Algorithm','accuracy']]
results
```

Out[ ]:

	Algorithm	accuracy
1	Random Forest	0.871754
2	Random Forest Regressor k_fold	0.758996
3	Gradient Boost Regressor	0.880524
4	Gradient Boost Regressor k fold	0.769348
5	Ada Boost Regressor	0.753298
6	Ada Boost Regressor k fold	0.586789
7	KNN Regressor	0.753949
8	KNN Regressor k fold	0.690711

Bagging Reggressor

In [ ]:
```
from sklearn.ensemble import BaggingRegressor

model = BaggingRegressor()
model.fit(X_train,y_train)
```

Out[ ]:
```
BaggingRegressor(base_estimator=None, bootstrap=True, bootstrap_features=Fa
lse,
                 max_features=1.0, max_samples=1.0, n_estimators=10,
                 n_jobs=None, oob_score=False, random_state=None, verbose=
0,
                 warm_start=False)
```

In [ ]:
```
y_pred = model.predict(X_test)
```

In [ ]:
```
model.score(X_train, y_train)
```

Out[ ]:
```
0.9761360288792809
```

In [ ]:
```
model.score(X_test, y_test)
```

Out[ ]:
```
0.8688590692690799
```

In [ ]:
```
acc_BR=metrics.r2_score(y_test, y_pred)
acc_BR
```

Out[ ]:
```
0.8688590692690799
```

In [ ]:
```
metrics.mean_squared_error(y_test, y_pred)
```

Out[ ]:
```
34.37298202989392
```

In [ ]:
```
#Store the accuracy results for each model in a dataframe for final comparis

Bagging_df = pd.DataFrame({'Algorithm': ['Bagging Regressor'], 'accuracy': a
results = pd.concat([results, Bagging_df])
results = results[['Algorithm','accuracy']]
results
```

Out[ ]:

	Algorithm	accuracy
1	Random Forest	0.871754
2	Random Forest Regressor k_fold	0.758996
3	Gradient Boost Regressor	0.880524
4	Gradient Boost Regressor k fold	0.769348
5	Ada Boost Regressor	0.753298
6	Ada Boost Regressor k fold	0.586789
7	KNN Regressor	0.753949
8	KNN Regressor k fold	0.690711
9	Bagging Regressor	0.868859

KFold Validation

In [ ]:
```
k = 20

kfold = KFold(n_splits=k, random_state=70)
results_7 = cross_val_score(model, X, y, cv=kfold)
accuracy=np.mean(abs(results_7))
accuracy
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_split.py:29
6: FutureWarning:

Setting a random_state has no effect since shuffle is False. This will raise
an error in 0.24. You should leave random_state to its default (None), or se
t shuffle=True.
```

Out[ ]: 0.7282362195588703

In [ ]:
```
#Store the accuracy results for each model in a dataframe for final comparis

BaggingKFold_df = pd.DataFrame({'Algorithm': ['Bagging Regressor k fold'], '
results = pd.concat([results, BaggingKFold_df])
results = results[['Algorithm','accuracy']]
results
```

Out[ ]:

	Algorithm	accuracy
1	Random Forest	0.871754
2	Random Forest Regressor k_fold	0.758996
3	Gradient Boost Regressor	0.880524
4	Gradient Boost Regressor k fold	0.769348
5	Ada Boost Regressor	0.753298
6	Ada Boost Regressor k fold	0.586789
7	KNN Regressor	0.753949
8	KNN Regressor k fold	0.690711
9	Bagging Regressor	0.868859
10	KNN Regressor k fold	0.728236

In [ ]:

Support Vector Regressor

In [ ]:
```
from sklearn.svm import SVR
model = SVR(kernel='linear')
model.fit(X_train, y_train)
```

Out[ ]:
```
SVR(C=1.0, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma='scale',
    kernel='linear', max_iter=-1, shrinking=True, tol=0.001, verbose=False)
```

In [ ]:
```
y_pred = model.predict(X_test)
```

In [ ]:
```
model.score(X_train, y_train)
```

Out[ ]: 0.7296525761559518

```
In [ ]: acc_SVR=metrics.r2_score(y_test, y_pred)
        acc_SVR
```

```
Out[ ]: 0.6549962611822544
```

```
In [ ]: metrics.mean_squared_error(y_test, y_pred)
```

```
Out[ ]: 90.42796363067555
```

```
In [ ]: #Store the accuracy results for each model in a dataframe for final comparis

        SVR_df = pd.DataFrame({'Algorithm': ['Support Vector Regressor'], 'accuracy'
        results = pd.concat([results, SVR_df])
        results = results[['Algorithm','accuracy']]
        results
```

Out[ ]:

	Algorithm	accuracy
1	Random Forest	0.871754
2	Random Forest Regressor k_fold	0.758996
3	Gradient Boost Regressor	0.880524
4	Gradient Boost Regressor k fold	0.769348
5	Ada Boost Regressor	0.753298
6	Ada Boost Regressor k fold	0.586789
7	KNN Regressor	0.753949
8	KNN Regressor k fold	0.690711
9	Bagging Regressor	0.868859
10	KNN Regressor k fold	0.728236
10	Support Vector Regressor	0.654996

KFold for SVR

```
In [ ]: k = 20

        kfold = KFold(n_splits=k, random_state=70)
        results_8 = cross_val_score(model, X, y, cv=kfold)
        accuracy=np.mean(abs(results_8))
        accuracy
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_split.py:29
6: FutureWarning:

Setting a random_state has no effect since shuffle is False. This will raise
an error in 0.24. You should leave random_state to its default (None), or se
t shuffle=True.
```

Out[ ]: 0.6155301658292511

In [ ]:
```
#Store the accuracy results for each model in a dataframe for final comparis

SVRKFold_df = pd.DataFrame({'Algorithm': ['Support Vector Regressor k fold']
results = pd.concat([results, SVRKFold_df])
results = results[['Algorithm','accuracy']]
results
```

Out[ ]:

	Algorithm	accuracy
1	Random Forest	0.871754
2	Random Forest Regressor k_fold	0.758996
3	Gradient Boost Regressor	0.880524
4	Gradient Boost Regressor k fold	0.769348
5	Ada Boost Regressor	0.753298
6	Ada Boost Regressor k fold	0.586789
7	KNN Regressor	0.753949
8	KNN Regressor k fold	0.690711
9	Bagging Regressor	0.868859
10	KNN Regressor k fold	0.728236
10	Support Vector Regressor	0.654996
10	Support Vector Regressor k fold	0.615530

XGBoost Regressor

In [ ]:
```
import xgboost as xgb
from xgboost.sklearn import XGBRegressor
xgr = XGBRegressor()

xgr.fit(X_train, y_train)
```

```
[11:48:46] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:line
ar is now deprecated in favor of reg:squarederror.
```

Out[ ]:
```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0,
             importance_type='gain', learning_rate=0.1, max_delta_step=0,
             max_depth=3, min_child_weight=1, missing=None, n_estimators=10
0,
             n_jobs=1, nthread=None, objective='reg:linear', random_state=
0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
             silent=None, subsample=1, verbosity=1)
```

In [ ]:
```
y_pred = xgr.predict(X_test)
```

In [ ]:
```
xgr.score(X_train, y_train)
```

Out[ ]:
```
0.9441437766049378
```

In [ ]:
```
acc_XGB=metrics.r2_score(y_test, y_pred)
acc_XGB
```

Out[ ]:
```
0.8818060857485882
```

In [ ]:
```
metrics.mean_squared_error(y_test, y_pred)
```

Out[ ]:
```
30.979475804869445
```

In [ ]:
```
#Store the accuracy results for each model in a dataframe for final comparis

XGB_df = pd.DataFrame({'Algorithm': ['Support Vector Regressor'], 'accuracy'
results = pd.concat([results, XGB_df])
results = results[['Algorithm','accuracy']]
results
```

Out[ ]:

	Algorithm	accuracy
1	Random Forest	0.871754
2	Random Forest Regressor k_fold	0.758996
3	Gradient Boost Regressor	0.880524
4	Gradient Boost Regressor k fold	0.769348
5	Ada Boost Regressor	0.753298
6	Ada Boost Regressor k fold	0.586789
7	KNN Regressor	0.753949
8	KNN Regressor k fold	0.690711
9	Bagging Regressor	0.868859
10	KNN Regressor k fold	0.728236
10	Support Vector Regressor	0.654996
10	Support Vector Regressor k fold	0.615530
13	Support Vector Regressor	0.881806

DesionTreeRegressor

In [ ]:
```
from sklearn.tree import DecisionTreeRegressor

dec_model = DecisionTreeRegressor()
dec_model.fit(X_train, y_train)
```

Out[ ]:
```
DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=None,
                      max_features=None, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, presort='deprecated',
                      random_state=None, splitter='best')
```

In [ ]:
```
#printing the feature importance(that's features that are important and help
print('Feature importance: \n',pd.DataFrame(dec_model.feature_importances_,c
```

```
Feature importance:
                Importance
cement          0.308079
slag            0.059006
ash             0.008372
Water           0.122484
superplastic    0.049993
coarseagg       0.028604
fineagg         0.050500
age             0.372962
```

As we can see, **Cement, Age** and **Water** are the most important features

In [ ]:
```
y_pred = dec_model.predict(X_test)
```

In [ ]:
```
dec_model.score(X_train, y_train)
```

Out[ ]:
```
0.9930841416603411
```

In [ ]:
```
dec_model.score(X_test, y_test)
```

Out[ ]:
```
0.7539514842000445
```

In [ ]:
```
acc_DT=metrics.r2_score(y_test, y_pred)
acc_DT
```

Out[ ]:
```
0.7539514842000445
```

In [ ]:
```
#Store the accuracy results for each model in a dataframe for final comparis

DT_df = pd.DataFrame({'Algorithm': ['Decision Tree Regressor 1'], 'accuracy'
results = pd.concat([results, DT_df])
results = results[['Algorithm','accuracy']]
results
```

Out[ ]:

	Algorithm	accuracy
1	Random Forest	0.871754
2	Random Forest Regressor k_fold	0.758996
3	Gradient Boost Regressor	0.880524
4	Gradient Boost Regressor k fold	0.769348
5	Ada Boost Regressor	0.753298
6	Ada Boost Regressor k fold	0.586789
7	KNN Regressor	0.753949
8	KNN Regressor k fold	0.690711
9	Bagging Regressor	0.868859
10	KNN Regressor k fold	0.728236
10	Support Vector Regressor	0.654996
10	Support Vector Regressor k fold	0.615530
13	Support Vector Regressor	0.881806
14	Support Vector Regressor	0.753951

KFold for Decision Tree Regressor

In [ ]:

```
k = 20

kfold = KFold(n_splits=k, random_state=70)
results_9 = cross_val_score(dec_model, X, y, cv=kfold)
accuracy=np.mean(abs(results_9))
accuracy
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_split.py:29
6: FutureWarning:

Setting a random_state has no effect since shuffle is False. This will raise
an error in 0.24. You should leave random_state to its default (None), or se
t shuffle=True.
```

Out[ ]: 0.5972893593134021

In [ ]:

```
#Store the accuracy results for each model in a dataframe for final comparis

DCT_df = pd.DataFrame({'Algorithm': ['Decision Tree Regressor k fold'], 'acc
results = pd.concat([results, DCT_df])
results = results[['Algorithm','accuracy']]
results
```

Out[ ]:

	Algorithm	accuracy
1	Random Forest	0.871754
2	Random Forest Regressor k_fold	0.758996
3	Gradient Boost Regressor	0.880524
4	Gradient Boost Regressor k fold	0.769348
5	Ada Boost Regressor	0.753298
6	Ada Boost Regressor k fold	0.586789
7	KNN Regressor	0.753949
8	KNN Regressor k fold	0.690711
9	Bagging Regressor	0.868859
10	KNN Regressor k fold	0.728236
10	Support Vector Regressor	0.654996
10	Support Vector Regressor k fold	0.615530
13	Support Vector Regressor	0.881806
14	Support Vector Regressor	0.753951
15	Decision Tree Regressor k fold	0.597289

Feature Selection

In [ ]:
```
df2 = df.copy() #create a copy of df in order to drop the least important fe
```

In [ ]:
```
X = df2.drop(['strength','ash','coarseagg','fineagg'],axis=1)
y = df2['strength']
#Split the X and y into training and test set in 70:30 ratio
X_train,X_test, y_train,y_test = train_test_split(X,y, test_size=0.3,random_
```

In [ ]:
```
X_train = X_train.apply(zscore)
X_test = X_test.apply(zscore)
```

In [ ]:
```
decNew_Model = DecisionTreeRegressor()
decNew_Model.fit(X_train, y_train)
```

Out[ ]:
```
DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=None,
                      max_features=None, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, presort='deprecated',
                      random_state=None, splitter='best')
```

In [ ]:
```
#printing the feature importance(that's features that are important and help
print('Feature importance: \n',pd.DataFrame(decNew_Model.feature_importances
```

```
Feature importance:
               Importance
cement           0.350456
slag             0.074806
Water            0.138865
superplastic     0.058386
age              0.377487
```

In [ ]:
```
y_pred = decNew_Model.predict(X_test)
```

In [ ]:
```
decNew_Model.score(X_train, y_train)
```

Out[ ]:
```
0.9911889880235538
```

In [ ]:
```
decNew_Model.score(X_test, y_test)
```

Out[ ]:
```
0.7441978271113237
```

In [ ]:
```
acc_DT=metrics.r2_score(y_test, y_pred)
acc_DT
```

Out[ ]:
```
0.7441978271113237
```

In [ ]:
```
#Store the accuracy results for each model in a dataframe for final comparis

DT_df = pd.DataFrame({'Algorithm': ['Decision Tree Regressor 2'], 'accuracy'
results = pd.concat([results, DT_df])
results = results[['Algorithm','accuracy']]
results
```

Out[ ]:

	Algorithm	accuracy
1	Random Forest	0.871754
2	Random Forest Regressor k_fold	0.758996
3	Gradient Boost Regressor	0.880524
4	Gradient Boost Regressor k fold	0.769348
5	Ada Boost Regressor	0.753298
6	Ada Boost Regressor k fold	0.586789
7	KNN Regressor	0.753949
8	KNN Regressor k fold	0.690711
9	Bagging Regressor	0.868859
10	KNN Regressor k fold	0.728236
10	Support Vector Regressor	0.654996
10	Support Vector Regressor k fold	0.615530
13	Support Vector Regressor	0.881806
14	Support Vector Regressor	0.753951
15	Decision Tree Regressor k fold	0.597289
16	Decision Tree Regressor 2	0.744198

In [ ]:
```
#Let's create our training and testing data again since it has been override

X=df.drop('strength',axis=1)
y=df['strength']
```

In [ ]:
```
Xscaled=X.apply(zscore)
Xscaled_df=pd.DataFrame(Xscaled,columns=df.columns)
```

In [ ]:
```
#Split the X and y into training and test set in 70:30 ratio
X_train,X_test, y_train,y_test = train_test_split(Xscaled,y, test_size=0.3,r
```

In [ ]:
```
dec_prun_model=DecisionTreeRegressor(max_depth=4, random_state=1,min_samples
dec_prun_model.fit(X_train,y_train)
```

Out[ ]:
```
DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=4,
                      max_features=None, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=5, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, presort='deprecated',
                      random_state=1, splitter='best')
```

In [ ]:
```
#printing the feature importance(that's features that are important and help
print('Feature importance: \n',pd.DataFrame(dec_prun_model.feature_importanc
```

```
Feature importance:
               Importance
cement           0.355615
slag             0.000000
ash              0.000000
Water            0.106034
superplastic     0.035409
coarseagg        0.000000
fineagg          0.025055
age              0.477887
```

Plotting The Decision Tree

In [ ]:
```
!pip install graphviz
```

```
Requirement already satisfied: graphviz in /usr/local/lib/python3.7/dist-pac
kages (0.10.1)
```

In [ ]:
```
!pip install pydot
```

```
Requirement already satisfied: pydot in /usr/local/lib/python3.7/dist-packag
es (1.3.0)
Requirement already satisfied: pyparsing>=2.1.4 in /usr/local/lib/python3.7/
dist-packages (from pydot) (2.4.7)
```

In [ ]:
```
from sklearn.tree import export_graphviz
from sklearn.externals.six import StringIO
from IPython.display import Image
import graphviz
import pydot
```

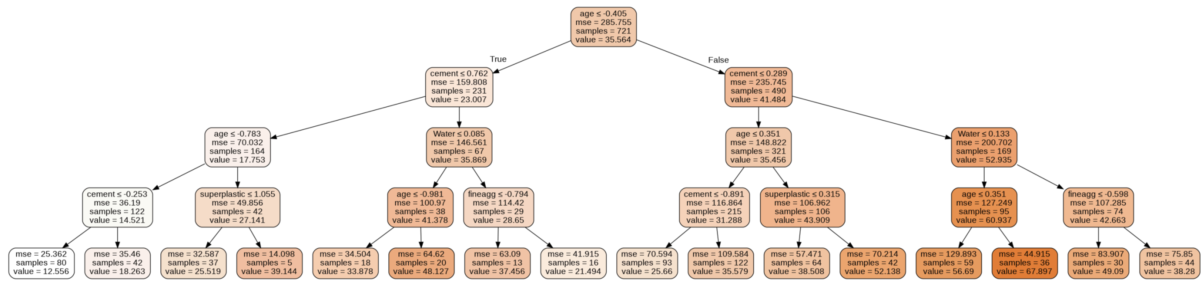
In [ ]:
```
Xscaled_df=Xscaled_df.drop('strength',axis=1)
feature_cols = Xscaled_df.columns
```

In [ ]:
```
feature_cols
```

Out[ ]:
```
Index(['cement', 'slag', 'ash', 'Water', 'superplastic', 'coarseagg',
       'fineagg', 'age'],
      dtype='object')
```

In [ ]:
```
dot_data = StringIO()
export_graphviz(dec_prun_model,out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True,
                feature_names = feature_cols,class_names=['0','1'])
(graph,) = pydot.graph_from_dot_data(dot_data.getvalue())
graph.write_png('concrete_pruned.png')
Image(graph.create_png())
```

Out[ ]:



In [ ]:

```
y_pred = dec_prun_model.predict(X_test)
```

In [ ]:

```
#On Training data
dec_prun_model.score(X_train, y_train)
```

Out[ ]: 0.7578225840644413

In [ ]:

```
#On testing data

dec_prun_model.score(X_test, y_test)
```

Out[ ]: 0.556820999525816

In [ ]:

```
acc_DecT=metrics.r2_score(y_test, y_pred)
acc_DecT
```

Out[ ]: 0.556820999525816

In [ ]:

```
metrics.mean_squared_error(y_test, y_pred)
```

Out[ ]: 116.16040647585388

In [ ]:

```
#Store the accuracy results for each model in a dataframe for final comparis

DecT_df = pd.DataFrame({'Algorithm': ['Pruned Decision Tree'], 'accuracy': [
results = pd.concat([results, DecT_df])
results = results[['Algorithm','accuracy']]
results
```

Out[ ]:

	Algorithm	accuracy
1	Random Forest	0.871754
2	Random Forest Regressor k_fold	0.758996
3	Gradient Boost Regressor	0.880524
4	Gradient Boost Regressor k fold	0.769348
5	Ada Boost Regressor	0.753298
6	Ada Boost Regressor k fold	0.586789
7	KNN Regressor	0.753949
8	KNN Regressor k fold	0.690711
9	Bagging Regressor	0.868859
10	KNN Regressor k fold	0.728236
10	Support Vector Regressor	0.654996
10	Support Vector Regressor k fold	0.615530
13	Support Vector Regressor	0.881806
14	Support Vector Regressor	0.753951
15	Decision Tree Regressor k fold	0.597289
16	Decision Tree Regressor 2	0.744198
17	Pruned Decision Tree	0.556821

KFold for Pruned Decision Tree

In [ ]:

```
k = 20

kfold = KFold(n_splits=k, random_state=70)
results_10 = cross_val_score(dec_prun_model, X, y, cv=kfold)
accuracy=np.mean(abs(results_10))
accuracy
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_split.py:29
6: FutureWarning:

Setting a random_state has no effect since shuffle is False. This will raise
an error in 0.24. You should leave random_state to its default (None), or se
t shuffle=True.
```

Out[ ]: 0.44792037352404224

In [ ]:

```
#Store the accuracy results for each model in a dataframe for final comparis

decKFold_df = pd.DataFrame({'Algorithm': ['Pruned Decision Tree k fold'], 'a
results = pd.concat([results, decKFold_df])
```

```python
results = results[['Algorithm','accuracy']]
results
```

Out[ ]:

	Algorithm	accuracy
1	Random Forest	0.871754
2	Random Forest Regressor k_fold	0.758996
3	Gradient Boost Regressor	0.880524
4	Gradient Boost Regressor k fold	0.769348
5	Ada Boost Regressor	0.753298
6	Ada Boost Regressor k fold	0.586789
7	KNN Regressor	0.753949
8	KNN Regressor k fold	0.690711
9	Bagging Regressor	0.868859
10	KNN Regressor k fold	0.728236
10	Support Vector Regressor	0.654996
10	Support Vector Regressor k fold	0.615530
13	Support Vector Regressor	0.881806
14	Support Vector Regressor	0.753951
15	Decision Tree Regressor k fold	0.597289
16	Decision Tree Regressor 2	0.744198
17	Pruned Decision Tree	0.556821
18	Pruned Decision Tree k fold	0.447920

Gradient Boost Regressor, Support Vector Regressor, Bagging Regressor and **Random Forest** seems to do well in the scenario. We can choose either of them.

NB: You can again drop the features that are not important and rebuild the models again(consider doing hyperparameter tuning using GridSearchCV)

In [ ]:

```python

```