Importing all the required libraries

In [367]:
```
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression, Lasso, Ridge
from sklearn.metrics import mean_squared_error
from math import sqrt
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler
```

Importing dataset

In [368]:
```
df = pd.read_csv("concrete.csv")
# Component 1 - Cement (kg in a m^3 mixture)
# Component 2 - Blast furnace slag (kg in a m^3 mixture)
# Component 3 - Flyash (kg in a m^3 mixture)
# Component 4 - Water (kg in a m^3 mixture)
# Component 5 - Superplastisizer (kg in a m^3 mixture)
# Component 6 - Coarse Aggregate (kg in a m^3 mixture)
# Component 7 - Fine Aggregate (kg in a m^3 mixture)
# Component 8 - Age (days)


#The cement Strength can vary between min - 17Mpa , 18-28 Mpa and 28 - <
70 Mpa,
```

In [369]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1030 entries, 0 to 1029
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   cement        1030 non-null   float64
 1   slag          1030 non-null   float64
 2   ash           1030 non-null   float64
 3   water         1030 non-null   float64
 4   superplastic  1030 non-null   float64
 5   coarseagg     1030 non-null   float64
 6   fineagg       1030 non-null   float64
 7   age           1030 non-null   int64  
 8   strength      1030 non-null   float64
dtypes: float64(8), int64(1)
memory usage: 72.5 KB
```

In [370]:
```
df.head()
```

Out[370]:

	cement	slag	ash	water	superplastic	coarseagg	fineagg	age	strength
0	141.3	212.0	0.0	203.5	0.0	971.8	748.5	28	29.89
1	168.9	42.2	124.3	158.3	10.8	1080.8	796.2	14	23.51
2	250.0	0.0	95.7	187.4	5.5	956.9	861.2	28	29.22
3	266.0	114.0	0.0	228.0	0.0	932.0	670.0	28	45.85
4	154.8	183.4	0.0	193.3	9.1	1047.4	696.7	28	18.29

In [371]:
```
df.isnull().sum()
```

Out[371]:
```
cement          0
slag            0
ash             0
water           0
superplastic    0
coarseagg       0
fineagg         0
age             0
strength        0
dtype: int64
```

In [372]:

```
print(df[df['age'] > 365])
print(df[df['age'] < 1])
```

```
Empty DataFrame
Columns: [cement, slag, ash, water, superplastic, coarseagg, fineagg, a
ge, strength]
Index: []
Empty DataFrame
Columns: [cement, slag, ash, water, superplastic, coarseagg, fineagg, a
ge, strength]
Index: []
```

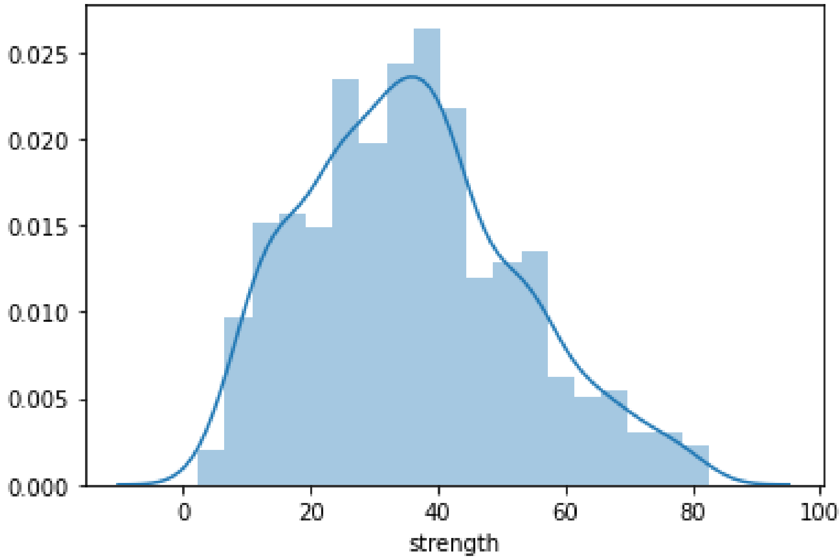
In [373]:

```
df.describe
# Inferences :

# Slag and ash, age has wide difference in mean and 50% values,indicatin
g mean > median, so being a right tailed skewness in data.

# Also, slag and ash, superplastic has min value as 0, which cannot be i
n he composition of cement.

# in cement., min = 102, std = 104, range is 102 - 540, does the nearing
value of std and min indicate something?

# In col Water , std is < min value, shuld we consider this variable for
further analysis

# considering the difference b/w ( Q1,median ) and (Q3 and median) in co
lumns slag,ash there are huge number of outliers.

# Other columns has very little diff b/w ( Q1,median ) and (Q3 and media
n), so there might be 1 or 2 outliers
```

Out[373]:
```
<bound method NDFrame.describe of       cement   slag    ash  water  su
perplastic  coarseagg  fineagg  age  \
0      141.3  212.0    0.0  203.5          0.0      971.8    748.5   2
8
1      168.9   42.2  124.3  158.3         10.8     1080.8    796.2   1
4
2      250.0    0.0   95.7  187.4          5.5      956.9    861.2   2
8
3      266.0  114.0    0.0  228.0          0.0      932.0    670.0   2
8
4      154.8  183.4    0.0  193.3          9.1     1047.4    696.7   2
8
...      ...    ...    ...    ...          ...        ...      ...
...
1025   135.0    0.0  166.0  180.0         10.0      961.0    805.0   2
8
1026   531.3    0.0    0.0  141.8         28.2      852.1    893.7
3
1027   276.4  116.0   90.3  179.6          8.9      870.1    768.3   2
8
1028   342.0   38.0    0.0  228.0          0.0      932.0    670.0  27
0
1029   540.0    0.0    0.0  173.0          0.0     1125.0    613.0
7

      strength
0        29.89
1        23.51
2        29.22
3        45.85
4        18.29
...        ...
1025     13.29
1026     41.30
1027     44.28
1028     55.06
1029     52.61

[1030 rows x 9 columns]>
```

In [374]:
```
sns.distplot(df['strength'])
```

Out[374]: <matplotlib.axes._subplots.AxesSubplot at 0x13a2a4280>

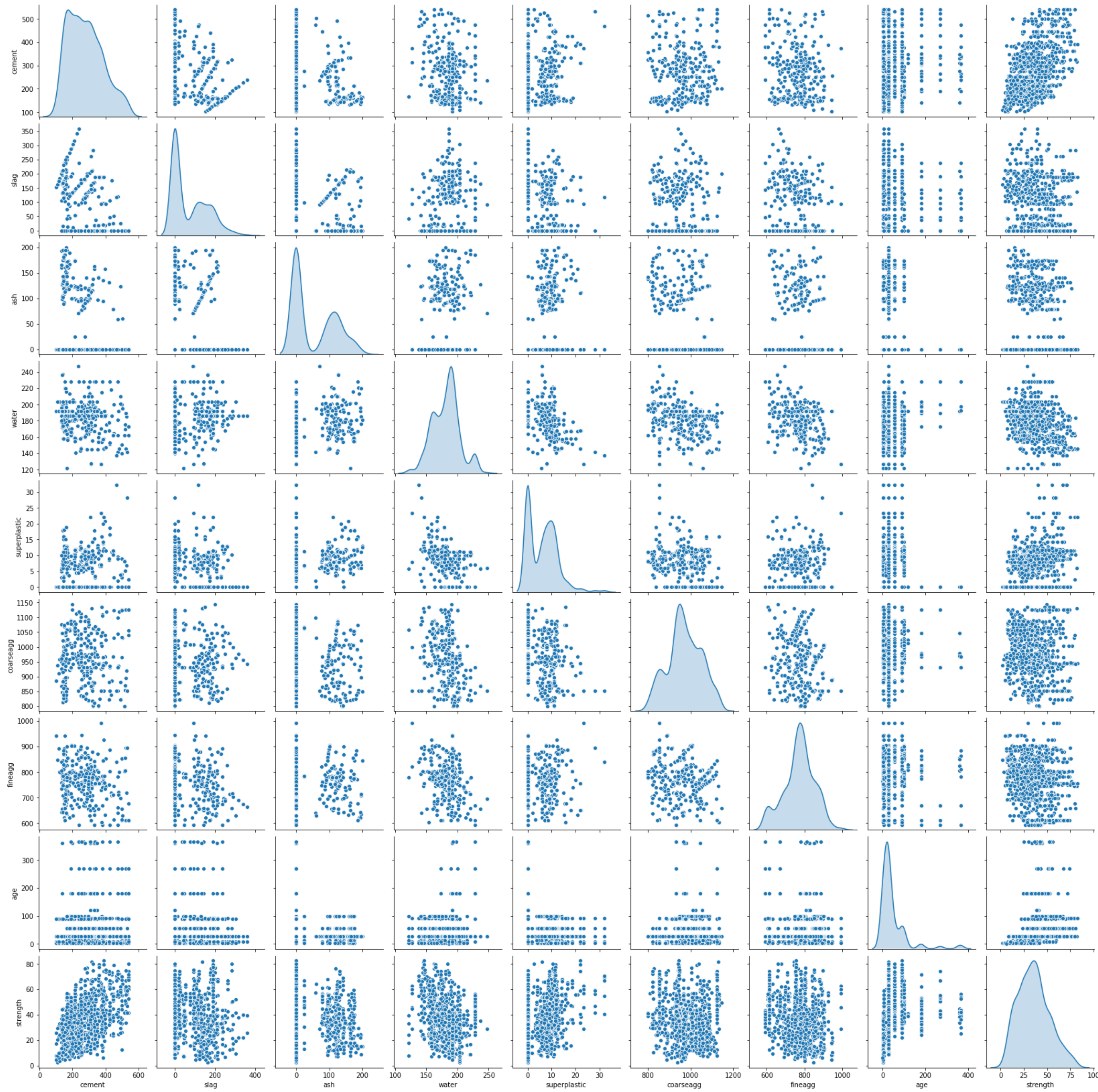


In [375]:
```
# The target variable has the distribution of data to be almost normal,
 but with lesser values on higher range, / higher values on lesser range
# Let us check the skewness:
# In probability theory and statistics, skewness is a measure of the asy
mmetry of the probability distribution of a real-valued random variable
 about its mean. The skewness value can be positive, zero, negative, or
 undefined.
print("Skewness = ",df['strength'].skew())

print('\nIt is positively, lightly skewed, as the skew value is < 0.5. F
or analysis, we can log transform this variable for better analysis, opt
ional')
```

```
Skewness =  0.41697728841071807

It is positively, lightly skewed, as the skew value is < 0.5. For analy
sis, we can log transform this variable for better analysis, optional
```

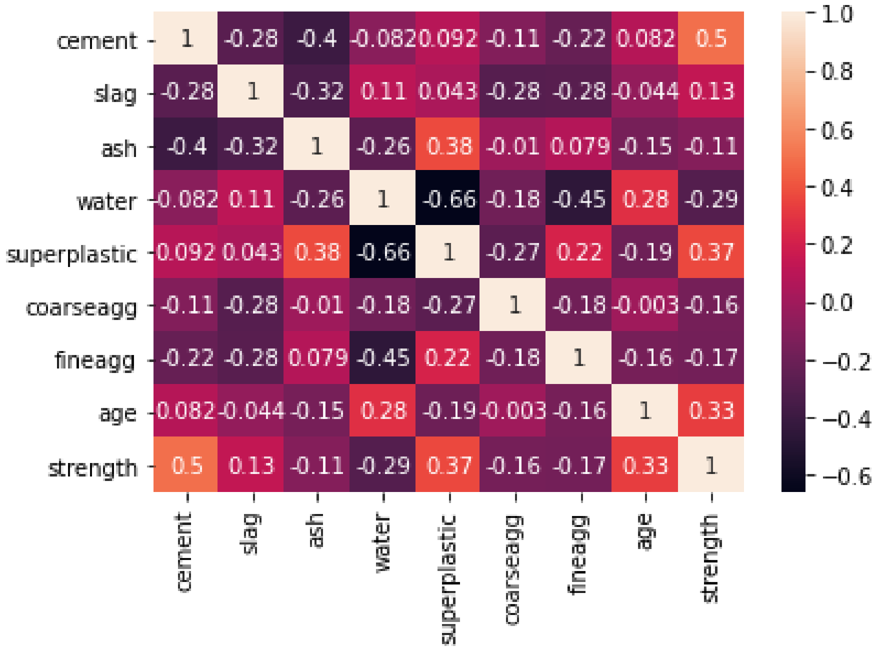
In [376]:
```
sns.pairplot(df, diag_kind='kde')
```

Out[376]: <seaborn.axisgrid.PairGrid at 0x13b355b20>



In [377]:
```
corr_DF = df.corr()
sns.heatmap(corr_DF, annot = True)
```

Out[377]: <matplotlib.axes._subplots.AxesSubplot at 0x13d5870a0>

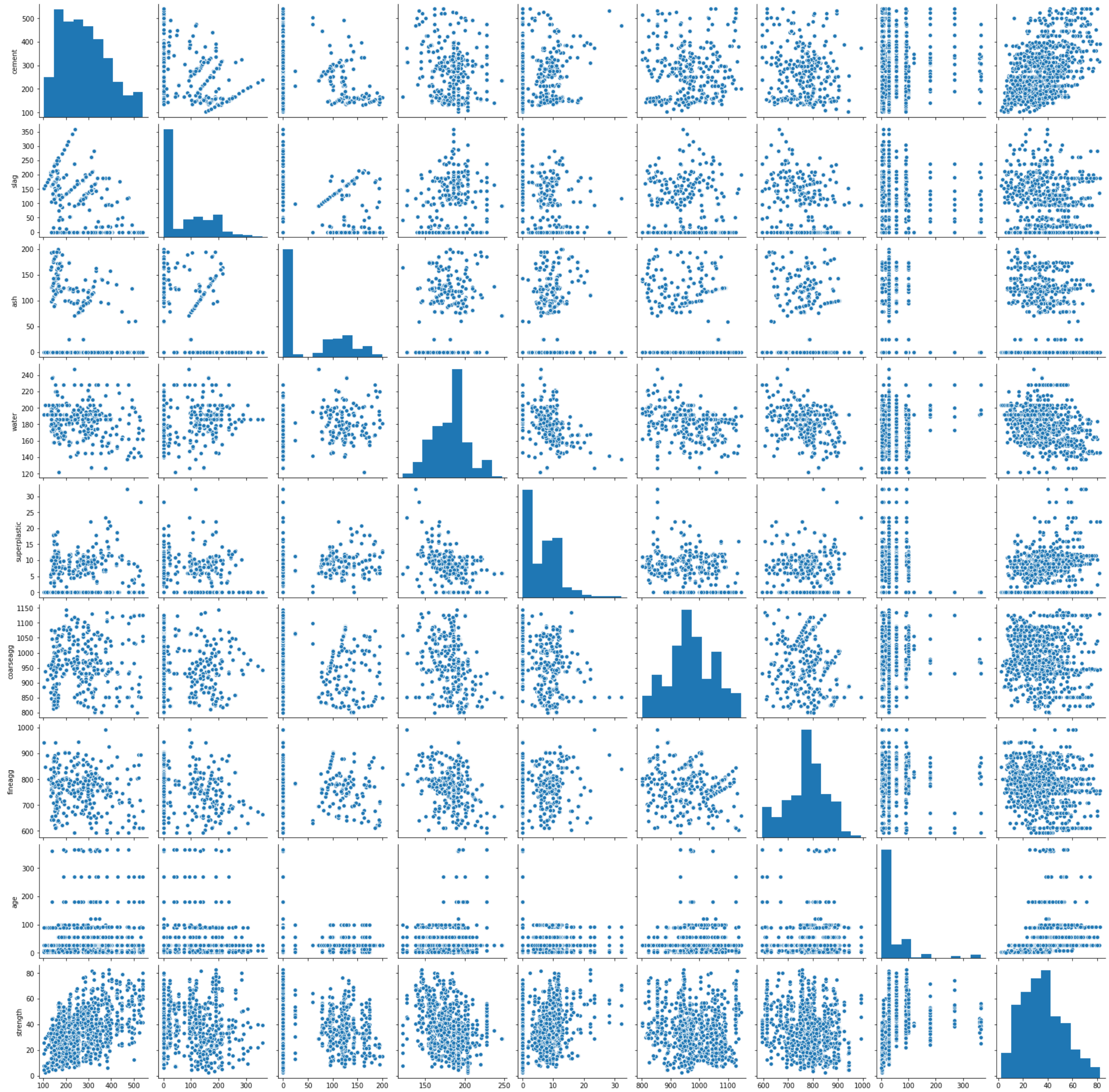


In [378]:
```
data_to_normalize = df.iloc[:,:-1]
scaler = MinMaxScaler()
scaler.fit_transform(data_to_normalize)
```

Out[378]:
```
array([[0.08972603, 0.58987201, 0.        , ..., 0.49651163, 0.3876066
2,
        0.07417582],
       [0.15273973, 0.11741792, 0.62118941, ..., 0.81337209, 0.5072754
6,
        0.03571429],
       [0.33789954, 0.        , 0.47826087, ..., 0.45319767, 0.6703462
1,
        0.07417582],
       ...,
       [0.39817352, 0.32276016, 0.45127436, ..., 0.20087209, 0.4372804
8,
        0.07417582],
       [0.54794521, 0.10573178, 0.        , ..., 0.38081395, 0.1906673
4,
        0.73901099],
       [1.        , 0.        , 0.        , ..., 0.94186047, 0.0476668
3,
        0.01648352]])
```

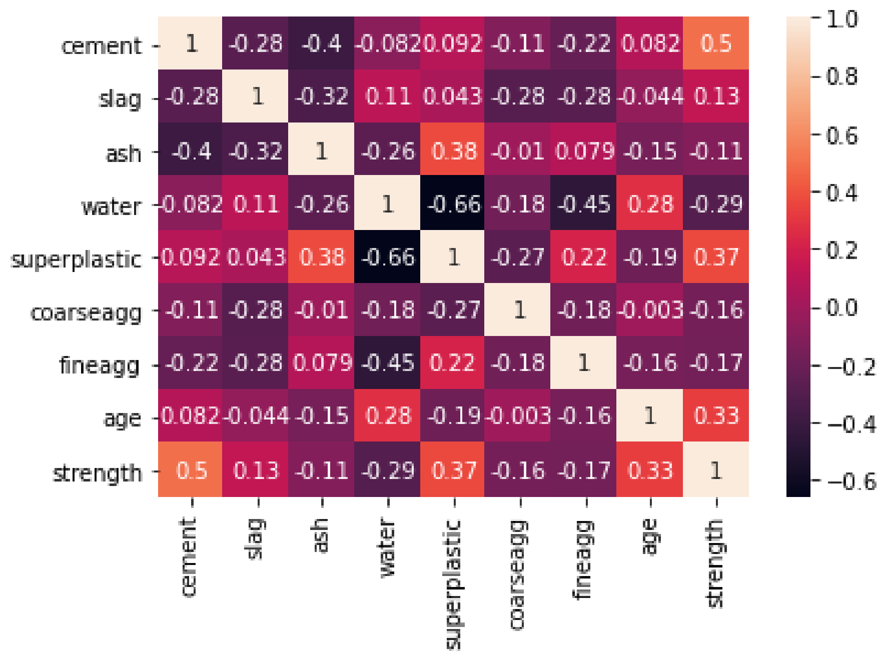
In [379]:
```
data_to_normalize['strength'] = df['strength']
sns.pairplot(data_to_normalize)
```

Out[379]: <seaborn.axisgrid.PairGrid at 0x13d64a340>



In [380]:
```python
corr_normalised_Data = data_to_normalize.corr()
sns.heatmap(corr_normalised_Data, annot = True)
```

Out[380]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x140152e80>
```

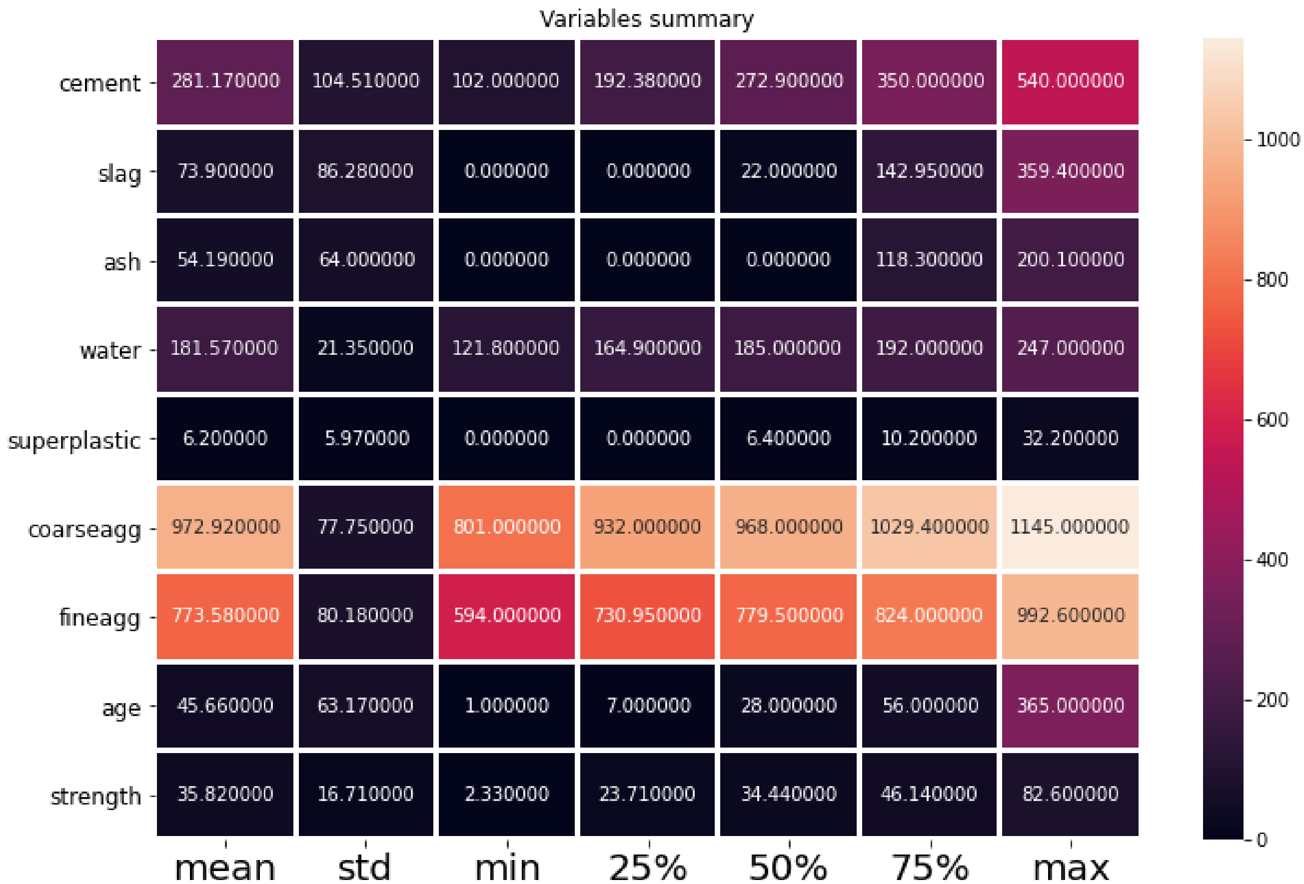


In [381]:
```python
#The cement Strength can vary between min - 17Mpa , 18-28 Mpa and 28 - < 70 Mpa,
#we can have df divided into three.
```

In [382]:
```python
print('\nIt is evident that cement and age, superplastic have good positive correlation with strength, cement being the strong predictor Water has a good negative correlation with Strength, where are there are other  variables with very less correlation there are good correlation amng variables : cement -> Slag = -ve relation and strength-> +ve Slag -> ash and fineaggg = -ve Water -> superplastic , fineagg strength = -ve superplastic -> ash and Strength positive correlation between water and slag age -> Strength and water\n')
```

```
It is evident that cement and age, superplastic have good positive correlation with strength, cement being the strong predictor Water has a good negative correlation with Strength, where are there are other variables with very less correlation there are good correlation amng variables : cement -> Slag = -ve relation and strength-> +ve Slag -> ash and fineaggg = -ve Water -> superplastic , fineagg strength = -ve superplastic -> ash and Strength positive correlation between water and slag age -> Strength and water
```

In [383]:
```
plt.figure(figsize=(12,8))
sns.heatmap(round(df.describe()[1:].transpose(),2),linewidth=2,annot=True,fmt="f")
plt.xticks(fontsize=20)
plt.yticks(fontsize=12)
plt.title("Variables summary")
plt.show()
```



In [ ]:

Splitting data in training set and test set

```
In [384]: shuffle_df = df.sample(frac=1)
          train_size = int(0.80 * len(df))

          train_set = shuffle_df[:train_size]
          test_set = shuffle_df[train_size:]

          X_train = np.array(train_set[["cement","slag",'ash','water','superplasti
          c','coarseagg','fineagg','age']])
          Y_train = np.array(train_set[["strength"]])
          X_test = np.array(test_set[["cement","slag",'ash','water','superplastic'
          ,'coarseagg','fineagg','age']])
          Y_test = np.array(test_set[["strength"]])

          #A dictionary to store RMSE corresponding the regression
          RMSE_dict = {}
```

Applying Linear regression and calculating RMSE

```
In [385]: regr = LinearRegression()
          regr.fit(X_train, Y_train)
          print("Linear regression score = ", regr.score(X_test, Y_test))

          lr_intercept = regr.intercept_[0]
          lr_coef_list = [regr.coef_[0][0], regr.coef_[0][1], regr.coef_[0][2], re
          gr.coef_[0][3], regr.coef_[0][4], regr.coef_[0][5], regr.coef_[0][6], re
          gr.coef_[0][7]]

          #RMSE in linear regression
          Y_pred = regr.predict(X_test)
          rms = sqrt(mean_squared_error(Y_test, Y_pred))
          print("RMSE for linear regression = ", rms)

          # for i in range(len(Y_test)):
          #     print(Y_test[i], Y_pred[i])
          RMSE_dict["Linear Regression"] = rms
```

```
Linear regression score =  0.6069624088661314
RMSE for linear regression =  10.345310637749185
```

Applying Ridge regression and calculating RMSE

In [386]:
```
ridge = Ridge()
ridge.fit(X_train, Y_train)
print("Ridge regression score = ", ridge.score(X_test, Y_test))

ridg_intercept = ridge.intercept_[0]
ridg_coef_list = [ridge.coef_[0][0], ridge.coef_[0][1], ridge.coef_[0][2
], ridge.coef_[0][3], ridge.coef_[0][4], ridge.coef_[0][5], ridge.coef_[
0][6], ridge.coef_[0][7]]

#RMSE in Ridge regression
Y_pred = ridge.predict(X_test)
rms = sqrt(mean_squared_error(Y_test, Y_pred))
print("RMSE for Ridge regression = ", rms)

# for i in range(len(Y_test)):
#     print(Y_test[i], Y_pred[i])
RMSE_dict["Ridge Regression"] = rms
```

```
Ridge regression score =  0.6069647354893649
RMSE for Ridge regression =  10.345280017680963
```

Applying Lasso regression and calculating RMSE

In [387]:
```
lasso = Lasso()
lasso.fit(X_train, Y_train)
print("Lasso regression score = ", lasso.score(X_test, Y_test))

lass_intercept = lasso.intercept_[0]
lass_coef_list = [lasso.coef_[0], lasso.coef_[1], lasso.coef_[2], lasso.
coef_[3], lasso.coef_[4], lasso.coef_[5], lasso.coef_[6], lasso.coef_[7
]]

#RMSE in Ridge regression
Y_pred = lasso.predict(X_test)
rms = sqrt(mean_squared_error(Y_test, Y_pred))
print("RMSE for Lasso regression = ", rms)

# for i in range(len(Y_test)):
#     print(Y_test[i], Y_pred[i])
RMSE_dict["Lasso Regression"] = rms
```

```
Lasso regression score =  0.6105729109785738
RMSE for Lasso regression =  10.297684223949583
```

Applying Random Forest regression and Calculating RMSE

In [388]:
```
rfr = RandomForestRegressor(n_estimators=100)
rfr.fit(X_train, Y_train)
print("Random Forest regression score = ", rfr.score(X_test, Y_test))

Y_pred = rfr.predict(X_test)
rms = sqrt(mean_squared_error(Y_test, Y_pred))
print("RMSE for Random Forest regression = ", rms)

# for i in range(len(Y_test)):
#     print(Y_test[i], Y_pred[i])
RMSE_dict["Random Forest Regression"] = rms
```

```
<ipython-input-388-5872acb0dd9b>:2: DataConversionWarning: A column-vec
tor y was passed when a 1d array was expected. Please change the shape
of y to (n_samples,), for example using ravel().
  rfr.fit(X_train, Y_train)

Random Forest regression score =  0.9364858371542695
RMSE for Random Forest regression =  4.158740100851902
```

Applying Decision Tree regression and Calculating RMSE

In [389]:
```
dtr = DecisionTreeRegressor()
dtr.fit(X_train, Y_train)
print("Decision Tree regression score = ", rfr.score(X_test, Y_test))

Y_pred = dtr.predict(X_test)
rms = sqrt(mean_squared_error(Y_test, Y_pred))
print("RMSE for Decison Tree regression = ", rms)

RMSE_dict["Decision Tree Regression"] = rms
```

```
Decision Tree regression score =  0.9364858371542695
RMSE for Decison Tree regression =  5.443718433970434
```

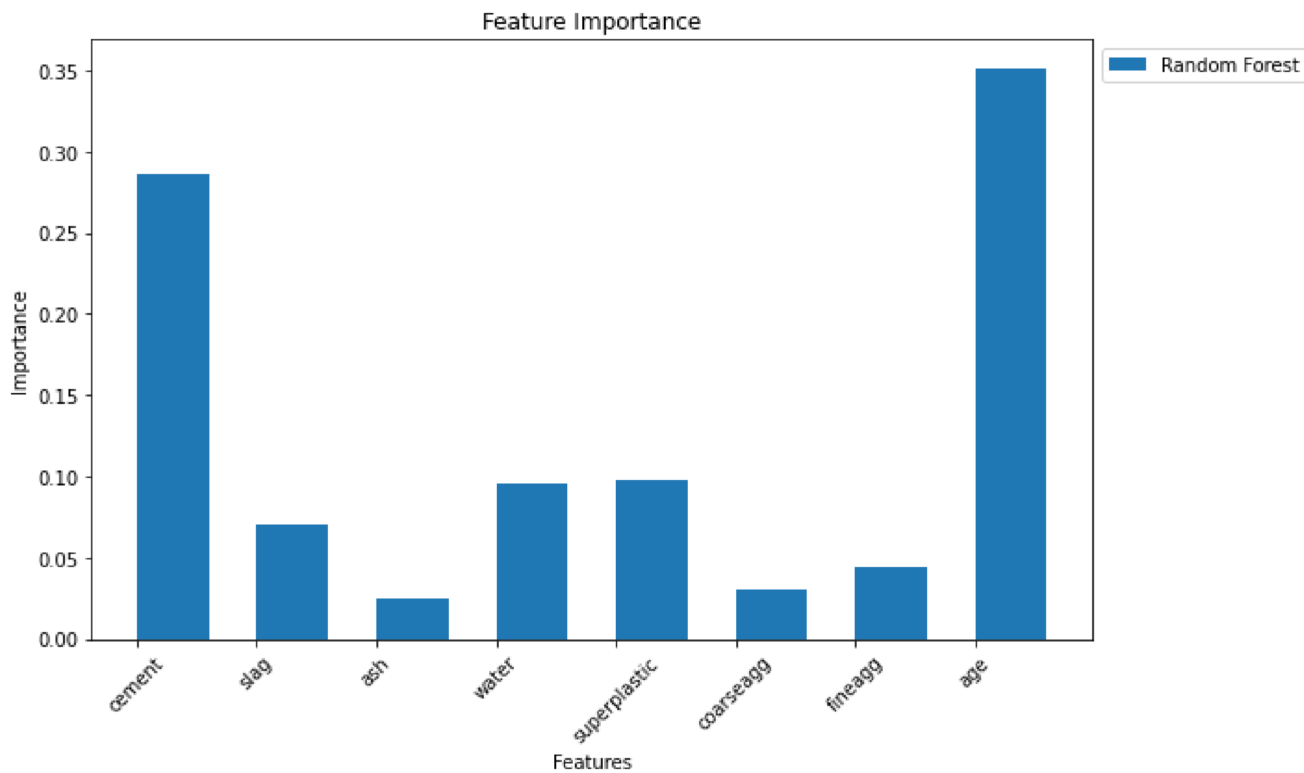
In [390]:
```
RMSE_dict
```

Out[390]:
```
{'Linear Regression': 10.345310637749185,
 'Ridge Regression': 10.345280017680963,
 'Lasso Regression': 10.297684223949583,
 'Random Forest Regression': 4.158740100851902,
 'Decision Tree Regression': 5.443718433970434}
```

In [391]:
```
#Random Forest Regression gives the minimum RMSE.
```

In [392]:
```
#Feature Importance or weight according to random forest regression
```

In [393]:
```
feature_rfr = rfr.feature_importances_
labels = df.columns[:-1]
x = np.arange(len(labels))
width = 0.6
fig, ax = plt.subplots(figsize=(10,6))
rects2 = ax.bar(x+(width/2), feature_rfr, width, label='Random Forest')
ax.set_ylabel('Importance')
ax.set_xlabel('Features')
ax.set_title('Feature Importance')
ax.set_xticks(x)
ax.set_xticklabels(labels, rotation=45)
ax.legend(loc="upper left", bbox_to_anchor=(1,1))
fig.tight_layout()
plt.show()
```



In [ ]:

In [ ]: