### Importing required libraries

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from warnings import filterwarnings
filterwarnings('ignore')
```

In [2]:

```
plt.rcParams['figure.figsize'] = [15,8]
```

### Reading the dataset and viewing the first 5 rows of it.

In [3]:

```
df = pd.read_excel('Concrete_Data.xls')

df.head()
```

Out[3]:

| | 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) | Coarse Aggregate (component 6)(kg in a m^3 mixture) | Fine Aggregate (component 7)(kg in a m^3 mixture) | Age (day) | Concrete compressive strength(MPa, megapascals) |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 540.0 | 0.0 | 0.0 | 162.0 | 2.5 | 1040.0 | 676.0 | 28 | 79.986111 |
| **1** | 540.0 | 0.0 | 0.0 | 162.0 | 2.5 | 1055.0 | 676.0 | 28 | 61.887366 |
| **2** | 332.5 | 142.5 | 0.0 | 228.0 | 0.0 | 932.0 | 594.0 | 270 | 40.269535 |
| **3** | 332.5 | 142.5 | 0.0 | 228.0 | 0.0 | 932.0 | 594.0 | 365 | 41.052780 |
| **4** | 198.6 | 132.4 | 0.0 | 192.0 | 0.0 | 978.4 | 825.5 | 360 | 44.296075 |

### Checking the shape/dimension of the dataset

In [4]:

```
print(f'The dataset has {df.shape[0]} rows and {df.shape[1]} columns')
print(f'The dimension of the dataset is {df.ndim}')
```

```
The dataset has 1030 rows and 9 columns
The dimension of the dataset is 2
```

### Checking the datatype, number of non null values and name of each variable in the dataset.

In [5]:

```
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 (component 1)(kg in a m^3 mixture)              1030 non-null   float64
 1   Blast Furnace Slag (component 2)(kg in a m^3 mixture)  1030 non-null   float64
 2   Fly Ash (component 3)(kg in a m^3 mixture)             1030 non-null   float64
 3   Water  (component 4)(kg in a m^3 mixture)              1030 non-null   float64
 4   Superplasticizer (component 5)(kg in a m^3 mixture)    1030 non-null   float64
 5   Coarse Aggregate  (component 6)(kg in a m^3 mixture)   1030 non-null   float64
 6   Fine Aggregate (component 7)(kg in a m^3 mixture)      1030 non-null   float64
 7   Age (day)                                              1030 non-null   int64  
 8   Concrete compressive strength(MPa, megapascals)        1030 non-null   float64
dtypes: float64(8), int64(1)
memory usage: 72.5 KB
```

### Checking for the missing values. Displaying number of missing values per column

In [6]:

```
missing_values = pd.DataFrame({'Number of missing values': df.isnull().sum(),
                               'Percenatge of missing values': round((df.isnull().sum() / len(df)) * 100,2)})

missing_values
```

Out[6]:

| | Number of missing values | Percenatge of missing values |
|---|---|---|
| **Cement (component 1)(kg in a m^3 mixture)** | 0 | 0.0 |
| **Blast Furnace Slag (component 2)(kg in a m^3 mixture)** | 0 | 0.0 |
| **Fly Ash (component 3)(kg in a m^3 mixture)** | 0 | 0.0 |
| **Water (component 4)(kg in a m^3 mixture)** | 0 | 0.0 |
| **Superplasticizer (component 5)(kg in a m^3 mixture)** | 0 | 0.0 |
| **Coarse Aggregate (component 6)(kg in a m^3 mixture)** | 0 | 0.0 |
| **Fine Aggregate (component 7)(kg in a m^3 mixture)** | 0 | 0.0 |
| **Age (day)** | 0 | 0.0 |
| **Concrete compressive strength(MPa, megapascals)** | 0 | 0.0 |

From above table it is clearly evident that there are no missing values present in the given dataset

## Checking for the summary statistics of the dataset

In [7]:

```
df.describe().T
```

Out[7]:

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| **Cement (component 1)(kg in a m^3 mixture)** | 1030.0 | 281.165631 | 104.507142 | 102.000000 | 192.375000 | 272.900000 | 350.000000 | 540.000000 |
| **Blast Furnace Slag (component 2)(kg in a m^3 mixture)** | 1030.0 | 73.895485 | 86.279104 | 0.000000 | 0.000000 | 22.000000 | 142.950000 | 359.400000 |
| **Fly Ash (component 3)(kg in a m^3 mixture)** | 1030.0 | 54.187136 | 63.996469 | 0.000000 | 0.000000 | 0.000000 | 118.270000 | 200.100000 |
| **Water (component 4)(kg in a m^3 mixture)** | 1030.0 | 181.566359 | 21.355567 | 121.750000 | 164.900000 | 185.000000 | 192.000000 | 247.000000 |
| **Superplasticizer (component 5)(kg in a m^3 mixture)** | 1030.0 | 6.203112 | 5.973492 | 0.000000 | 0.000000 | 6.350000 | 10.160000 | 32.200000 |
| **Coarse Aggregate (component 6)(kg in a m^3 mixture)** | 1030.0 | 972.918592 | 77.753818 | 801.000000 | 932.000000 | 968.000000 | 1029.400000 | 1145.000000 |
| **Fine Aggregate (component 7) (kg in a m^3 mixture)** | 1030.0 | 773.578883 | 80.175427 | 594.000000 | 730.950000 | 779.510000 | 824.000000 | 992.600000 |
| **Age (day)** | 1030.0 | 45.662136 | 63.169912 | 1.000000 | 7.000000 | 28.000000 | 56.000000 | 365.000000 |
| **Concrete compressive strength(MPa, megapascals)** | 1030.0 | 35.817836 | 16.705679 | 2.331808 | 23.707115 | 34.442774 | 46.136287 | 82.599225 |

## Univariate Analysis

In [8]:

```
df.columns.to_list()
```

Out[8]:

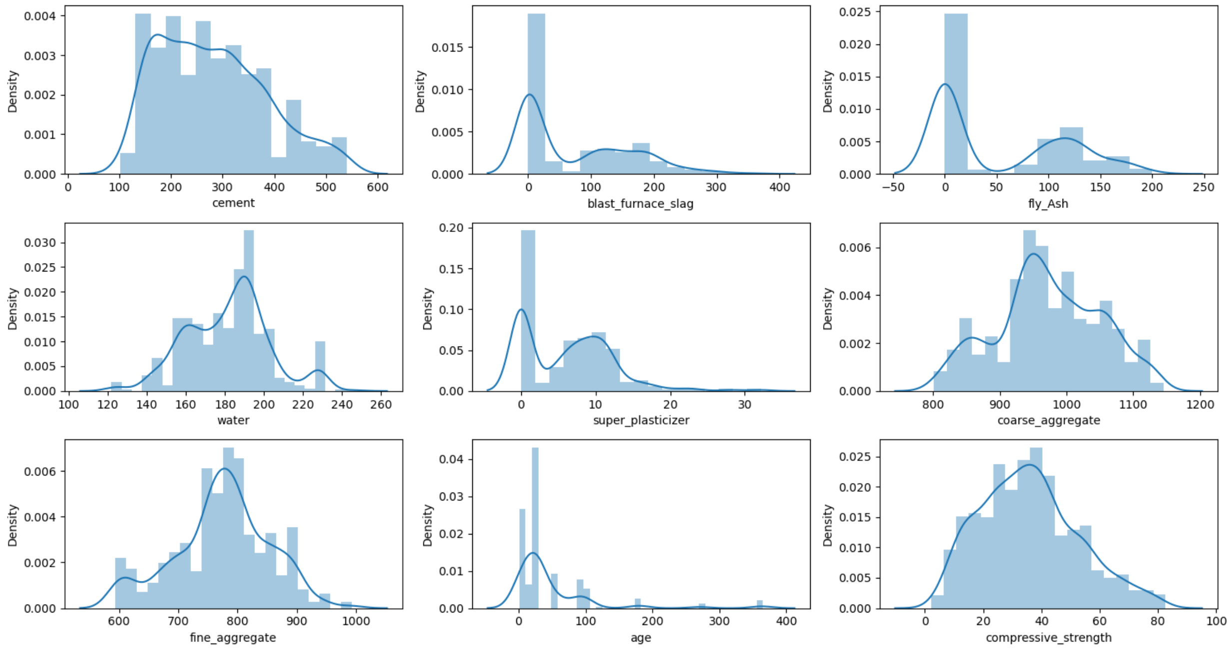
```
['Cement (component 1)(kg in a m^3 mixture)',
 'Blast Furnace Slag (component 2)(kg in a m^3 mixture)',
 'Fly Ash (component 3)(kg in a m^3 mixture)',
 'Water  (component 4)(kg in a m^3 mixture)',
 'Superplasticizer (component 5)(kg in a m^3 mixture)',
 'Coarse Aggregate  (component 6)(kg in a m^3 mixture)',
 'Fine Aggregate (component 7)(kg in a m^3 mixture)',
 'Age (day)',
 'Concrete compressive strength(MPa, megapascals) ']
```

In [9]:

```
# Renaming columns

df.rename(columns = {'Cement (component 1)(kg in a m^3 mixture)' : 'cement',
                     'Blast Furnace Slag (component 2)(kg in a m^3 mixture)' : 'blast_furnace_slag',
                     'Fly Ash (component 3)(kg in a m^3 mixture)' : 'fly_Ash',
                     'Water  (component 4)(kg in a m^3 mixture)' : 'water',
                     'Superplasticizer (component 5)(kg in a m^3 mixture)' : 'super_plasticizer',
                     'Coarse Aggregate  (component 6)(kg in a m^3 mixture)' : 'coarse_aggregate',
                     'Fine Aggregate (component 7)(kg in a m^3 mixture)' : 'fine_aggregate',
                     'Age (day)' : 'age',
                     'Concrete compressive strength(MPa, megapascals) ' : 'compressive_strength'},inplace = Tru
```

In [10]:

```
f , ax = plt.subplots(3,3)

for i ,v in zip(df.columns , ax.flatten()):
    sns.distplot(df[i] , ax = v)

plt.tight_layout()
plt.show()
```




## Bivariate Analysis

In [11]:

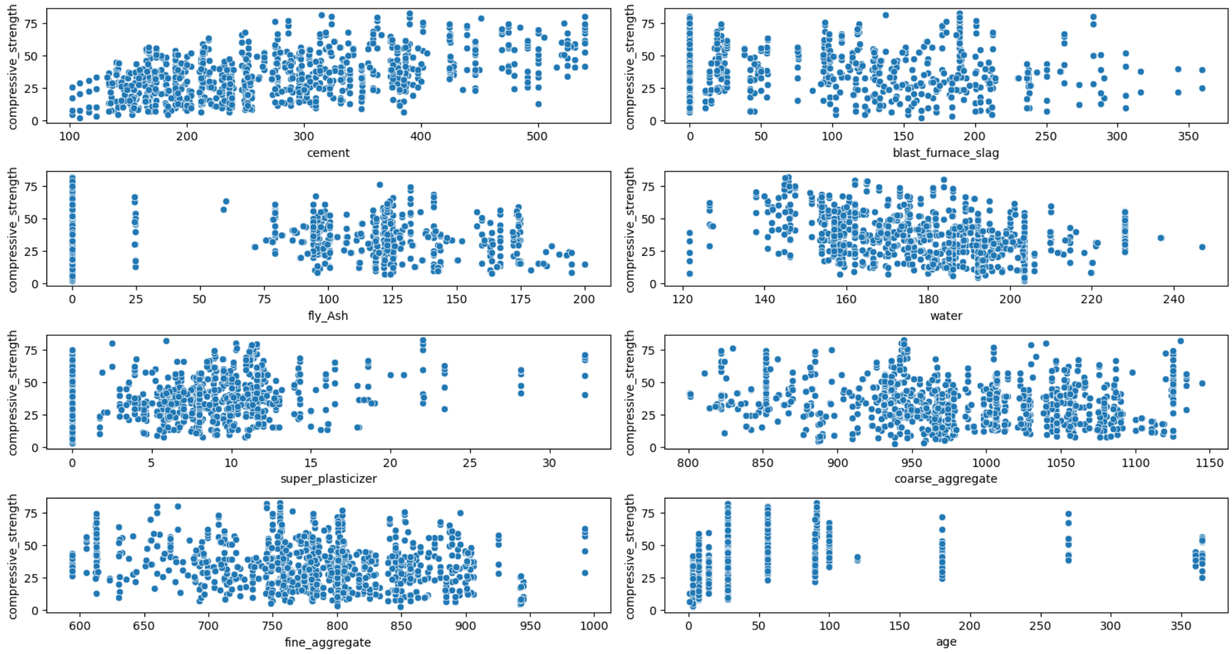
```
df.columns
```

Out[11]:

```
Index(['cement', 'blast_furnace_slag', 'fly_Ash', 'water', 'super_plasticizer',
       'coarse_aggregate', 'fine_aggregate', 'age', 'compressive_strength'],
      dtype='object')
```

In [12]:

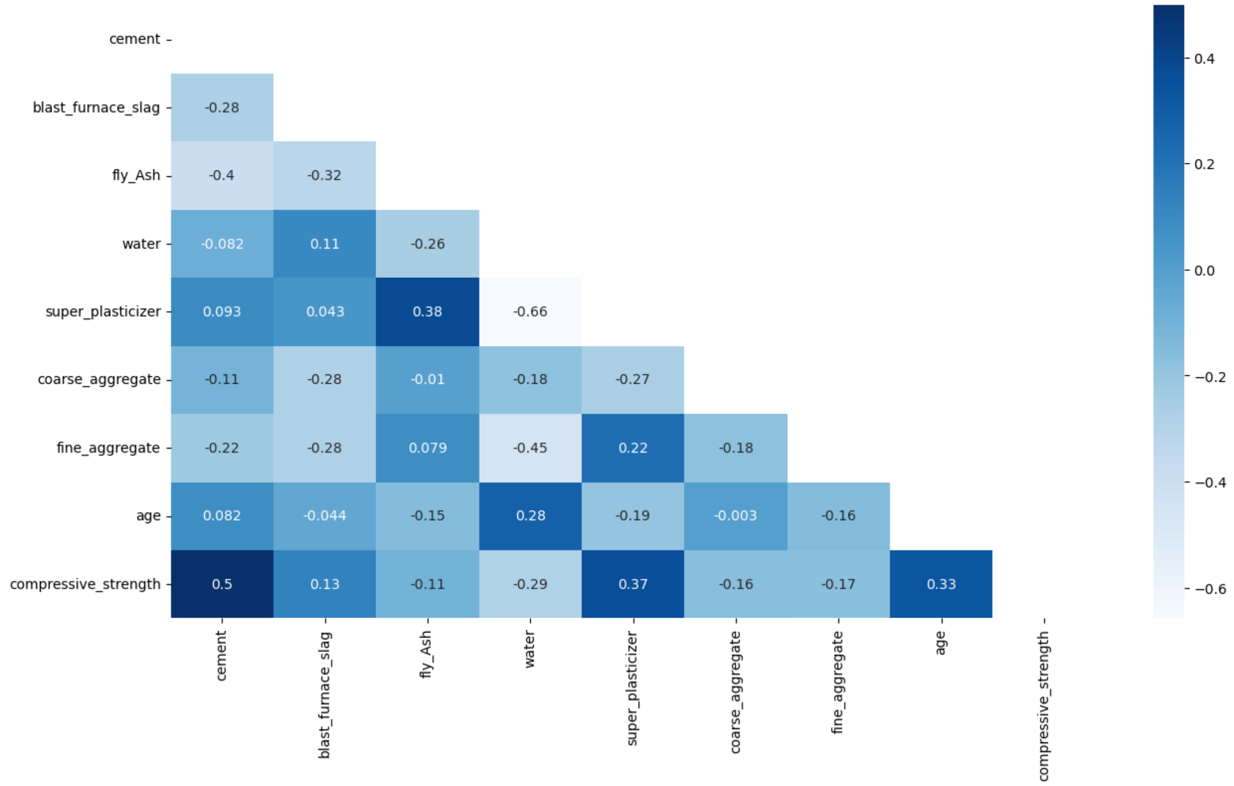
```python
f , ax = plt.subplots(4,2)

for i , v in zip(df.drop(columns = 'compressive_strength').columns , ax.flatten()):
    sns.scatterplot(x = df[i] , y = df['compressive_strength'] , ax = v)

plt.tight_layout()
plt.show()
```




## Multivariate Analysis

In [13]:

```python
sns.heatmap(df.corr() , annot = True , mask = np.triu(df.corr()) , cmap = 'Blues')
plt.show()
```



### Performing hypothesis testing to find the significant variables

Hypothesis :

H0 (Null Hypothesis) : There is no significant relationship between the variables being tested.

Ha (Alternative Hypothesis) : There is a significant relationship between the variables being tested

Consider significance level as 0.05

In [14]:

```
from scipy import stats
```

In [15]:

```
num_cols = df.select_dtypes(include = np.number).columns.to_list()

num_cols.remove('compressive_strength')

print(num_cols)
```

```
['cement', 'blast_furnace_slag', 'fly_Ash', 'water', 'super_plasticizer', 'coarse_aggregate', 'fine_
aggregate', 'age']
```

In [16]:

```
# Num vs Num - f_oneway test

statistical_result = pd.DataFrame(columns = ['Columns','Pvalue','Remarks'])

for i in num_cols:

    stat , pval = stats.pearsonr(df[i] , df['compressive_strength'])

    statistical_result = statistical_result.append({'Columns': i,
                                                    'Pvalue':pval,
                                                    'Remarks':'Reject H0' if pval <= 0.05 else 'Failed to rejec
                                                   ignore_index = True)

statistical_result
```

Out[16]:

| | Columns | Pvalue | Remarks |
|---|---|---|---|
| **0** | cement | 1.323458e-65 | Reject H0 |
| **1** | blast_furnace_slag | 1.414575e-05 | Reject H0 |
| **2** | fly_Ash | 6.752836e-04 | Reject H0 |
| **3** | water | 2.366073e-21 | Reject H0 |
| **4** | super_plasticizer | 5.079089e-34 | Reject H0 |
| **5** | coarse_aggregate | 1.019597e-07 | Reject H0 |
| **6** | fine_aggregate | 6.694681e-08 | Reject H0 |
| **7** | age | 2.103144e-27 | Reject H0 |

#### Insights from statistical test

From above performed statistical tests we can conclude that all of the columns has rejected null hypotheses which means the columns are significant to the target variable compressive_strength.

### Splitting the dataset randomly into train and test dataset using ratio of 70:30

In [17]:

```
from sklearn.model_selection import train_test_split
```

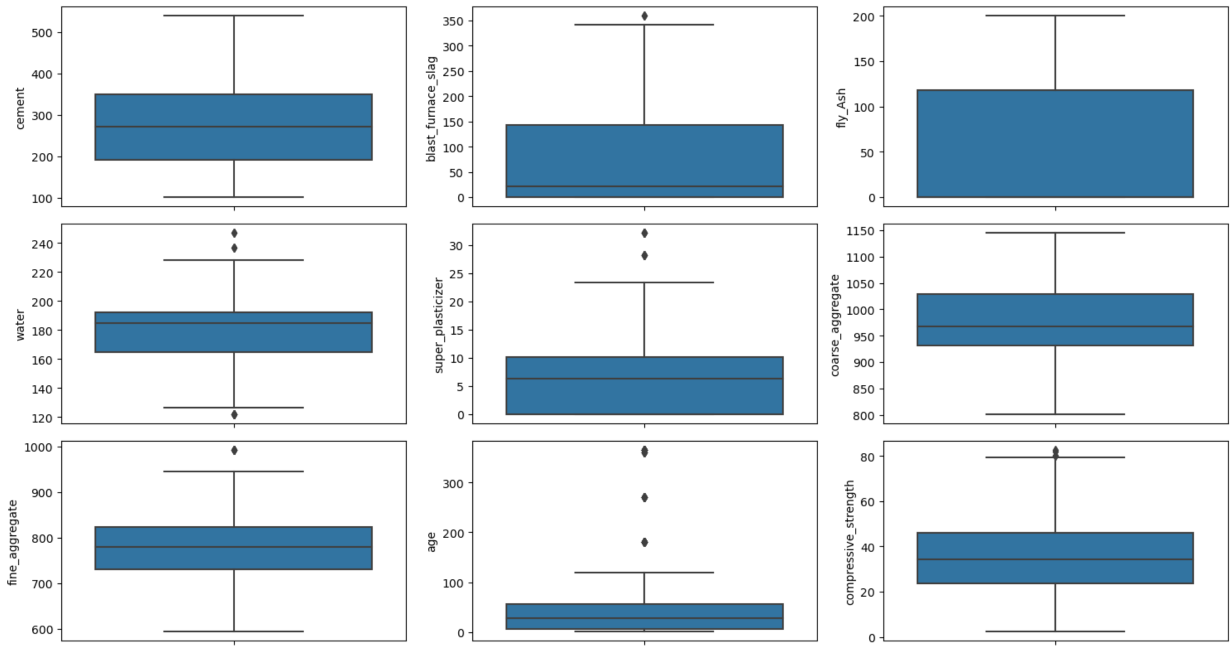
In [18]:

```
x = df.drop(columns = 'compressive_strength')
y = df['compressive_strength']

xtrain , xtest , ytrain , ytest = train_test_split(x , y , test_size = 0.30)
```

### Checking and treating of outliers

In [19]:

```
f , ax = plt.subplots(3,3)

for i , v in zip(df.columns , ax.flatten()):
    sns.boxplot(y = df[i] , ax = v)

plt.tight_layout()
plt.show()
```




From above boxplot it is clearly evident that there outliers present in water , super_plasticizer , age , fine_aggregate. By doing IQR method we tend to lose data so we go forward by performing PowerTransformer technique

In [20]:

```
from sklearn.preprocessing import PowerTransformer
```

In [21]:

```
out_cols = ['water' , 'super_plasticizer' , 'age' ,'fine_aggregate']

pt = PowerTransformer(standardize=False)

for i in out_cols :
    variable = pt.fit(xtrain[[i]])

    xtrain[i] = variable.transform(xtrain[[i]])
    xtest[i] = variable.transform(xtest[[i]])
```

In [22]:

```
xtrain
```

Out[22]:

| | cement | blast_furnace_slag | fly_Ash | water | super_plasticizer | coarse_aggregate | fine_aggregate | age |
|---|---|---|---|---|---|---|---|---|
| **62** | 310.00 | 0.0 | 0.00 | 130.753841 | 0.000000 | 971.0 | 68775.124744 | 1.380188 |
| **345** | 213.74 | 0.0 | 174.74 | 107.413653 | 3.222040 | 1053.5 | 58721.686648 | 2.684814 |
| **50** | 332.50 | 142.5 | 0.00 | 152.947925 | 0.000000 | 932.0 | 36955.248943 | 5.113320 |
| **719** | 166.80 | 250.2 | 0.00 | 137.881300 | 0.000000 | 975.6 | 48198.720921 | 4.446632 |
| **740** | 297.00 | 0.0 | 0.00 | 127.020037 | 0.000000 | 1040.0 | 53290.820050 | 2.065722 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... |
| **155** | 362.60 | 189.0 | 0.00 | 113.800652 | 3.436068 | 944.7 | 56058.584808 | 3.991404 |
| **370** | 218.85 | 0.0 | 124.13 | 109.755354 | 3.397428 | 1078.7 | 61171.655343 | 2.684814 |
| **659** | 108.30 | 162.4 | 0.00 | 137.881300 | 0.000000 | 938.2 | 68551.435774 | 4.446632 |
| **796** | 500.00 | 0.0 | 0.00 | 135.716004 | 0.000000 | 1125.0 | 39023.379454 | 4.446632 |
| **677** | 102.00 | 153.0 | 0.00 | 130.753841 | 0.000000 | 887.0 | 82060.005424 | 2.065722 |

721 rows × 8 columns

## Building a base model

Building a base model using Linear Regression as it is having the highest explanatory power compared to other models

In [23]:

```
import statsmodels.api as sma
```

In [24]:

```
model_lr = sma.OLS(ytrain , sma.add_constant(xtrain)).fit()

model_lr
```

Out[24]:

```
<statsmodels.regression.linear_model.RegressionResultsWrapper at 0x29441adff70>
```

#### Checking for summary

In [25]:

```
model_lr.summary()
```

Out[25]:

OLS Regression Results

| | | | |
|---|---|---|---|
| **Dep. Variable:** | compressive_strength | **R-squared:** | 0.816 |
| **Model:** | OLS | **Adj. R-squared:** | 0.814 |
| **Method:** | Least Squares | **F-statistic:** | 395.2 |
| **Date:** | Mon, 19 Jun 2023 | **Prob (F-statistic):** | 5.55e-256 |
| **Time:** | 20:30:08 | **Log-Likelihood:** | -2425.5 |
| **No. Observations:** | 721 | **AIC:** | 4869. |
| **Df Residuals:** | 712 | **BIC:** | 4910. |
| **Df Model:** | 8 | | |
| **Covariance Type:** | nonrobust | | |

| | coef | std err | t | P>\|t\| | [0.025 | 0.975] |
|---|---|---|---|---|---|---|
| **const** | -67.3042 | 18.572 | -3.624 | 0.000 | -103.766 | -30.842 |
| **cement** | 0.1269 | 0.007 | 18.401 | 0.000 | 0.113 | 0.140 |
| **blast_furnace_slag** | 0.1048 | 0.008 | 12.695 | 0.000 | 0.089 | 0.121 |
| **fly_Ash** | 0.0722 | 0.011 | 6.623 | 0.000 | 0.051 | 0.094 |
| **water** | -0.1724 | 0.049 | -3.534 | 0.000 | -0.268 | -0.077 |
| **super_plasticizer** | 1.4835 | 0.308 | 4.813 | 0.000 | 0.878 | 2.089 |
| **coarse_aggregate** | 0.0306 | 0.007 | 4.213 | 0.000 | 0.016 | 0.045 |
| **fine_aggregate** | 0.0002 | 6.5e-05 | 3.732 | 0.000 | 0.000 | 0.000 |
| **age** | 9.4084 | 0.247 | 38.161 | 0.000 | 8.924 | 9.892 |

| | | | |
|---|---|---|---|
| **Omnibus:** | 13.816 | **Durbin-Watson:** | 1.988 |
| **Prob(Omnibus):** | 0.001 | **Jarque-Bera (JB):** | 17.016 |
| **Skew:** | 0.232 | **Prob(JB):** | 0.000202 |
| **Kurtosis:** | 3.592 | **Cond. No.** | 4.23e+06 |

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 4.23e+06. This might indicate that there are
strong multicollinearity or other numerical problems.

In [26]:

```
from sklearn.metrics import mean_squared_error , r2_score

pred_train = model_lr.predict(sma.add_constant(xtrain))
pred_test = model_lr.predict(sma.add_constant(xtest))

rmse_train = np.sqrt(mean_squared_error(ytrain , pred_train))
rmse_test = np.sqrt(mean_squared_error(ytest , pred_test))

print('RMSE Train',rmse_train)
print('RMSE Test',rmse_test)
```

```
RMSE Train 6.994490794438262
RMSE Test 7.246478362374875
```

In [27]:

```python
# Creating a Dataframe to store metrics of train and test

performance_df = pd.DataFrame(columns = ['Model','R2_train','R2_test','RMSE_train','RMSE_test','Remarks'])

performance_df
```

Out[27]:

| Model | R2_train | R2_test | RMSE_train | RMSE_test | Remarks |
|---|---|---|---|---|---|

In [28]:

```python
# Appending values of base model in dataframe

performance_df = performance_df.append({'Model':'Linear Regression',
                                        'R2_train': r2_score(ytrain,pred_train),
                                        'R2_test': r2_score(ytest,pred_test),
                                        'RMSE_train':rmse_train,
                                        'RMSE_test':rmse_test,
                                        'Remarks' : 'Base'} , ignore_index=True)

performance_df
```

Out[28]:

| | Model | R2_train | R2_test | RMSE_train | RMSE_test | Remarks |
|---|---|---|---|---|---|---|
| **0** | Linear Regression | 0.816179 | 0.827372 | 6.994491 | 7.246478 | Base |

## Building different models and evaluating using appropriate technique

In [29]:

```python
# Creating a user defined function to predict and to store metrics of train and test


def model_performance(model,name):
    global performance_df

    pred_train = model.predict(xtrain)
    pred_test = model.predict(xtest)

    rmse_train = np.sqrt(mean_squared_error(ytrain,pred_train))
    rmse_test = np.sqrt(mean_squared_error(ytest,pred_test))
    r2_train = r2_score(ytrain , pred_train)
    r2_test = r2_score(ytest , pred_test)

    def remark(r2_train,r2_test,rmse_test):
        if abs(r2_train - r2_test) > 0.1 or rmse_test > 7.5:
            return 'Over Fit'

        else:
            return 'Good Fit'

    performance_df = performance_df.append({'Model':name,
                                            'R2_train':r2_train,
                                            'R2_test':r2_test,
                                            'RMSE_train':rmse_train,
                                            'RMSE_test':rmse_test,
                                            'Remarks':remark(r2_train,r2_test,rmse_test)} , ignore_index=True)
```

In [30]:

```
# Decision tree

from sklearn.tree import DecisionTreeRegressor

model_dt = DecisionTreeRegressor().fit(xtrain , ytrain)

model_performance(model_dt,'DecisionTree')
```

In [31]:

```
# RandomForest

from sklearn.ensemble import RandomForestRegressor

model_rf = RandomForestRegressor().fit(xtrain,ytrain)

model_performance(model_rf , 'Random Forest')
```

In [32]:

```
# KNN

from sklearn.neighbors import KNeighborsRegressor

model_knn = KNeighborsRegressor().fit(xtrain , ytrain)

model_performance(model_knn , 'KNN')
```

In [33]:

```
# AdaBoost

from sklearn.ensemble import AdaBoostRegressor

model_ab = AdaBoostRegressor().fit(xtrain , ytrain)

model_performance(model_ab , 'AdaBoost')
```

In [34]:

```
# Gradient Boosting

from sklearn.ensemble import GradientBoostingRegressor

model_gb = GradientBoostingRegressor().fit(xtrain , ytrain)

model_performance(model_gb , 'Gradient Boosting')
```

In [35]:

```
# Xgboost-RandomForest

from xgboost import XGBRFRegressor

model_xgbrf = XGBRFRegressor().fit(xtrain,ytrain)

model_performance(model_xgbrf , 'XGBoost Random Forest')
```

In [36]:

```
# Neural network

from sklearn.neural_network import MLPRegressor

model_nn = MLPRegressor().fit(xtrain , ytrain)

model_performance(model_nn , 'Neural Network')
```

In [37]:

```
# Creating a user defined function to highlight the rows which are good fit

def highlight_row(df):
    color_green = ['background-color : lightgreen']*len(df)
    color_white = ['background-color : white']*len(df)

    if df['Remarks'] == 'Good Fit':
        return color_green

    else:
        return color_white
```

In [38]:

```
performance_df.style.apply(highlight_row,axis = 1)
```

Out[38]:

| | Model | R2_train | R2_test | RMSE_train | RMSE_test | Remarks |
|---|---|---|---|---|---|---|
| **0** | Linear Regression | 0.816179 | 0.827372 | 6.994491 | 7.246478 | Base |
| **1** | DecisionTree | 0.997155 | 0.846114 | 0.870098 | 6.841828 | Over Fit |
| **2** | Random Forest | 0.983320 | 0.895372 | 2.106967 | 5.641513 | Good Fit |
| **3** | KNN | 0.570371 | 0.318708 | 10.693146 | 14.395894 | Over Fit |
| **4** | AdaBoost | 0.803477 | 0.753640 | 7.232119 | 8.656791 | Over Fit |
| **5** | Gradient Boosting | 0.951440 | 0.898429 | 3.595007 | 5.558483 | Good Fit |
| **6** | XGBoost Random Forest | 0.920989 | 0.850711 | 4.585666 | 6.738858 | Good Fit |
| **7** | Neural Network | 0.339765 | 0.356861 | 13.255861 | 13.986996 | Over Fit |

After assessing various models, it was observed that some models exhibited a significant drop in performance when applied to unseen data, indicating overfitting. However, there were models that consistently performed well on both training and unseen data. Notably, the Gradient boosting model outperformed other models in terms of high r2 and low rmse. Hence, based on its superior performance and generalization ability, we can confidently consider the Gradient boosting model as our final choice.