Concrete Compressive Strength Prediction

Report submitted for the course

"Regression Analysis and Time Series Models": 2023-2024

Prof. Buddhananda Banerjee

Department of Mathematics

Indian Institute of Technology Kharagpur

15.04.2024

Aakansh Singh (21CE3FP20)

Aadit Shah (21MI31001)

Anand Kumar (21MI3FP25)

Samyak Singhvi (21MI3FP11)

Table of Contents

Abstract:

The most crucial component in civil engineering is concrete. Age and composition have a very nonlinear effect on the compressive strength of concrete. These components include cement, fly ash, blast furnace slag, Superplasticizer, water, coarse and fine aggregate, and water.

Sources:

The dataset is collected from Kaggle wherein actual author and owner of the dataset is

Prof. I-Cheng Yeh

Department of Information Management

Chung-Hua University,

Hsin Chu, Taiwan 30067, R.O.C.

E-mail: icyeh@chu.edu.tw

Dataset Characteristics:

The actual concrete compressive strength (MPa) for a given mixture under a specific age (days) was determined from laboratory. Data is in raw form (not scaled).

Summary Statistics:

Number of instances (observations): 1030
Number of Attributes: 9
Attribute breakdown: 8 quantitative input variables, and 1 quantitative output variable
Missing Attribute Values: None

Importing Modules:

To implement the linear regression and to handle the data we have to import essential libraries.

```
# Importing essential libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from sklearn.linear_model import LinearRegression
import warnings
warnings.filterwarnings("ignore")
```

So, we import pandas for data analysis, NumPy for calculating N-dimensional array, seaborn and matplotlib to visualize the data

Reading data:

The dataset is in CSV file so we used to pandas (pd.read_csv('')) to read the data

	Cement	Blast Furnace Slag	Fly Ash	Water	Superplasticizer	Coarse Aggregate	Fine Aggregate	Age (day)	strength
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
...	...	...	...	...	...	...	...	...	...
1025	276.4	116.0	90.3	179.6	8.9	870.1	768.3	28	44.284354
1026	322.2	0.0	115.6	196.0	10.4	817.9	813.4	28	31.178794
1027	148.5	139.4	108.6	192.7	6.1	892.4	780.0	28	23.696601
1028	159.1	186.7	0.0	175.6	11.3	989.6	788.9	28	32.768036
1029	260.9	100.5	78.3	200.6	8.6	864.5	761.5	28	32.401235

1030 rows × 9 columns

The dataset contains 1030 examples having 9 columns in which 8 is features and 1 column (strength) is target variable.

Prepare dataset:

Separated 8 features and target variable as shown in the code.

```
# Separating dependent and independent variable.
X_raw = concrete_df.drop(columns=['strength '], axis=1)
Y = concrete_df['strength ']
```

Written a function to prepare the data to make suitable for fitting linear regression and then printed the data which shows all the column names are replaced by X variable.

```
# Preparing the data to fit Linear regression
def Prepare_data(X):
    X.columns = ['X_1', 'X_2', 'X_3', 'X_4', 'X_5', 'X_6', 'X_7', 'X_8']
    X_0 = pd.DataFrame({'X_0' : [1]*1030})
    df = pd.concat([X_0,X],axis = 1, join = 'inner')
    arr=np.array(df)
    return df,arr
```

```
df, X = Prepare_data(X_raw)
print("_____The prepared data is_____")
df
```

```
_____The prepared data is_____
```

	X_0	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8
0	1	540.0	0.0	0.0	162.0	2.5	1040.0	676.0	28
1	1	540.0	0.0	0.0	162.0	2.5	1055.0	676.0	28
2	1	332.5	142.5	0.0	228.0	0.0	932.0	594.0	270
3	1	332.5	142.5	0.0	228.0	0.0	932.0	594.0	365
4	1	198.6	132.4	0.0	192.0	0.0	978.4	825.5	360
...	...	...	...	...	...	...	...	...	...
1025	1	276.4	116.0	90.3	179.6	8.9	870.1	768.3	28
1026	1	322.2	0.0	115.6	196.0	10.4	817.9	813.4	28
1027	1	148.5	139.4	108.6	192.7	6.1	892.4	780.0	28
1028	1	159.1	186.7	0.0	175.6	11.3	989.6	788.9	28
1029	1	260.9	100.5	78.3	200.6	8.6	864.5	761.5	28

1030 rows × 9 columns

Parameter estimation:

A function is written to estimate the weight parameters.

```
# function to estimate parameters
def Parameter_est( X ,y):
    Transpose = X.T
    mal = np.matmul(X.T,X)
    inv = np.linalg.inv(mal)
    b_hat = np.matmul(np.matmul(inv,X.T),y)
    return b_hat

# Assign the estimated parameters to b_hat
b_hat = Parameter_est( X ,Y)
print("_____The weight parameters are_____")
b_hat
```

The **weight parameters** are found as:

```
_____The weight parameters are_____
array([-2.31637558e+01,  1.19785255e-01,  1.03847249e-01,  8.79430817e-02,
       -1.50297904e-01,  2.90686943e-01,  1.80301836e-02,  2.01544557e-02,
        1.14225620e-01])
```

Fitting Linear Regression:

Coded a linear regression function from scratch and fitted the linear regression and then predicted the target variable using the fitted equation.

```
# Function the Linear Regression
def linearRegression(beta):
    print ( "_____The fitted equation is_____")
    print (f"y_hat={b_hat[0]:.3f}X_0+{b_hat[1]:.3f}X_1+{b_hat[2]:.3f}X_2+{b_hat[3]:.3f}X_3+{b_hat[4]:.3f}X_4+
           {b_hat[5]:.3f}X_5+{b_hat[6]:.3f}X_6+{b_hat[7]:.3f}X_7+{b_hat[8]:.3f}X_8}}")
    print ( "_____")

# Fitting the equation:
linearRegression(b_hat)


# predicting the dependent variable using the fitted equation
y_hat = X@b_hat
print("_____The Predicted Values are_____")
print(y_hat)
```

The **fitted equation** and predicted values are printed as:

```
_____The fitted equation is_____
y_hat=-23.164X_0+0.120X_1+0.104X_2+0.088X_3+-0.150X_4+0.291X_5+0.018X_6+0.020X_7+0.114X_8}
_____
_____The Predicted Values are_____
[53.4728591  53.74331185 56.81194746 ... 26.47099254 29.11564722
 31.89398622]
```

Error calculation:

The function written to determine **Mean Square error** and **Root mean square error** is as

```
#Error calculation:
def Errors(y_true, y_pred):
    e = y_true - y_pred
    mse = ((e**2).sum())/X.shape[0]
    return mse
mse = Errors(Y,y_hat)

print("Mean squared error is: ", mse)
rmse = np.sqrt(mse)
print("Root mean squared error is: {}".format(rmse))
```

The Mean square error and Root mean square error is printed as

```
Mean squared error is:  107.21180273479736
Root mean squared error is: 10.354313243030527
```

One can see that the error is huge for the fitted equation.

Hypothesis Testing (ANOVA):

A function is coded to create ANOVA table using the equation and formulas taught in class.

```
# Function to create ANOVA Table
def Anova(y_true,y_pred):
    # Total variation
    SSt = ((y_true-y_true.mean())**2).sum()
    degree_t = X.shape[0] - 1
    # Residual variation
    SSres= ((y_true - y_pred)**2).sum()
    degree_res = X.shape[0] - X.shape[1]
    MSres = SSres/degree_res
    # variation due to regression
    SSreg = SSt-SSres
    degree_reg = X.shape[1]-1
    MSreg= SSreg/degree_reg
    F = MSreg/MSres
    return degree_res,degree_reg, SSres,SSreg,MSres,MSreg, F

degree_res, degree_reg, SSres,SSreg,MSres,MSreg, F = Anova(Y,y_hat)
```

The ANOVA table generated from the code is printed as:

```
# Creating ANOVA Table
anova_dict = {'DF':[degree_reg, degree_res, degree_reg+degree_res], 'SS':[SSreg, SSres, SSreg+SSres],
              'MS':[MSreg, MSres, MSreg+MSres], 'F':[F]}
Anova_df = pd.DataFrame(anova_dict,index=["Regression","Residual","Total"])
print("_________________ANOVA Table_____________________")
Anova_df
```

```
_________________ANOVA Table_____________________
```

	DF	SS	MS	F
Regression	8	176744.871659	22093.108957	204.269137
Residual	1021	110428.156817	108.156863	204.269137
Total	1029	287173.028476	22201.265820	204.269137

The ANOVA null hypothesis testing is performed as shown and it is found that the hypothesis is rejected.

```
# Testing Null hypothesis
print("_____________________________________________________________________________")
print(f"H0: b0=b1=.............bk-1=0 against \nH1: bi!=0 for i=0 to k-1")
print("If F>F{alpha, k-1, n-k}, The H0 is rejested")
print("_____________________________________________________________________________")
import scipy.stats
from scipy.stats import f
alpha = 0.05
q = 1 - alpha
f = f.ppf(q, degree_reg, degree_res)
print(f"The calcylated f value is: {f}" )
print(f"The observed f value is {F}")
if(abs(F)>f):
        print("The null hypothesis H0 is rejected")
else:
        print("The null hypothesis H0 is accepted")
```

```
_____________________________________________________________________________
H0: b0=b1=.............bk-1=0 against
H1: bi!=0 for i=0 to k-1
If F>F{alpha, k-1, n-k}, The H0 is rejested
_____________________________________________________________________________
The calcylated f value is: 1.9474558084667661
The observed f value is 204.2691365659187
The null hypothesis H0 is rejected
```

Since the hypothesis testing is rejected which suggest the all the features are significant, then proceeded to **Partial (Marginal) test.**

Predicting R^2 and R^2_{adj}:

The R^2 and R^2_{adj} value is determined of the fitted equation and it is found that R^2 value is 61.55% which is not decent.

```
# Predict R^2 value and adjusted R^2 value:
R_sq = (SSreg/ (SSres+SSreg) ) * 100
AdjR_sq = (1- MSres/(MSres+MSreg)) * 100
print(f"The R square value is:{R_sq}")
print(f"Adjusted R square value is: {AdjR_sq}")
```

```
The R square value is:61.54647342657952
Adjusted R square value is: 99.51283470241574
```

Since the score of the fitted equation is not satisfying so tested the significant of each regressor individually by Partial (Marginal) test.

Partial Test or Marginal test:

```
# Test on individual regression coefficient (Partial test or Marginat Test)
Corr = np.linalg.inv(X.T@X)
from scipy.stats import t
def Marginal_test(beta, C, MSres,X):
    n = X.shape[1]
    for i in range(n):
        T = beta[i]/np.sqrt(MSres*C[i][i])
        if(abs(T)>t.ppf(1-0.05, degree_res)):
            print(f"H0:b{i}=0 is rejected")
        else:
            print(f"HØ:b{i}=0 is accepted")

Marginal_test(b_hat,Corr,MSres,X)
```

```
HØ:b0=0 is accepted
H0:b1=0 is rejected
H0:b2=0 is rejected
H0:b3=0 is rejected
H0:b4=0 is rejected
H0:b5=0 is rejected
H0:b6=0 is rejected
H0:b7=0 is rejected
H0:b8=0 is rejected
```

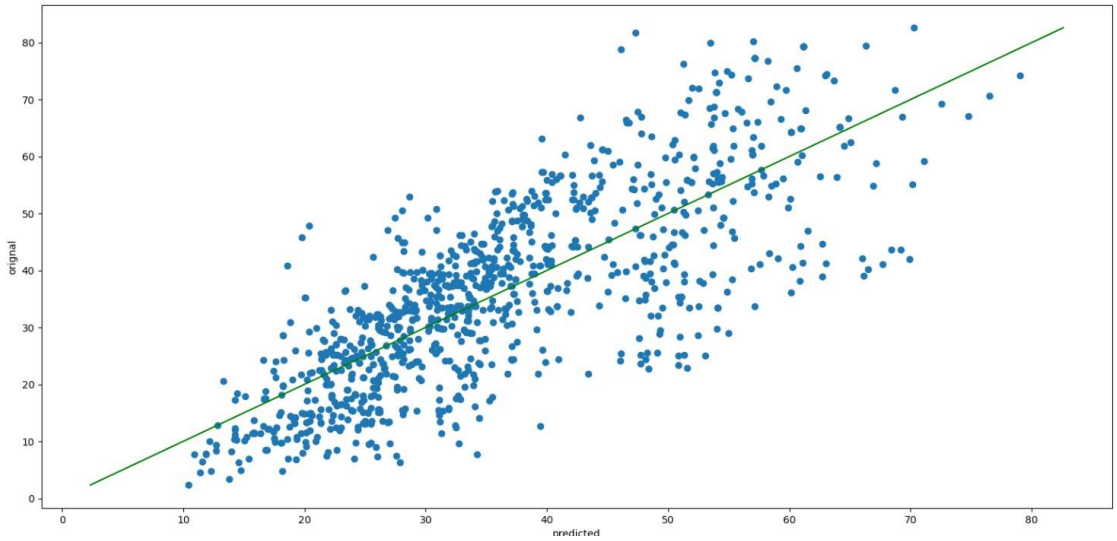
Notice that the null hypothesis for $\beta_0 = 0$ is accepted which means β_0 is not a significant parameter, therefore X_0 is not a significant regressor.

Further experimented fitting the linear regression discarding X_0 using similar functions as described above but the performance of the equation didn't change the errors much and score was almost the same as the previous one.

Plot of the equation:

Plotted the fitted equation using matplotlib library from python and the plot is shown how the line is fitted to the scattered data.

```
# Plot the regression Line fitted by the function made
plt.figure(figsize=[19 , 9])
plt.scatter( y_hat, Y)
plt.plot([Y.min(), Y.max()], [Y.min(), Y.max()], color = 'green')
plt.xlabel('predicted')
plt.ylabel('orignal')
plt.show()
```



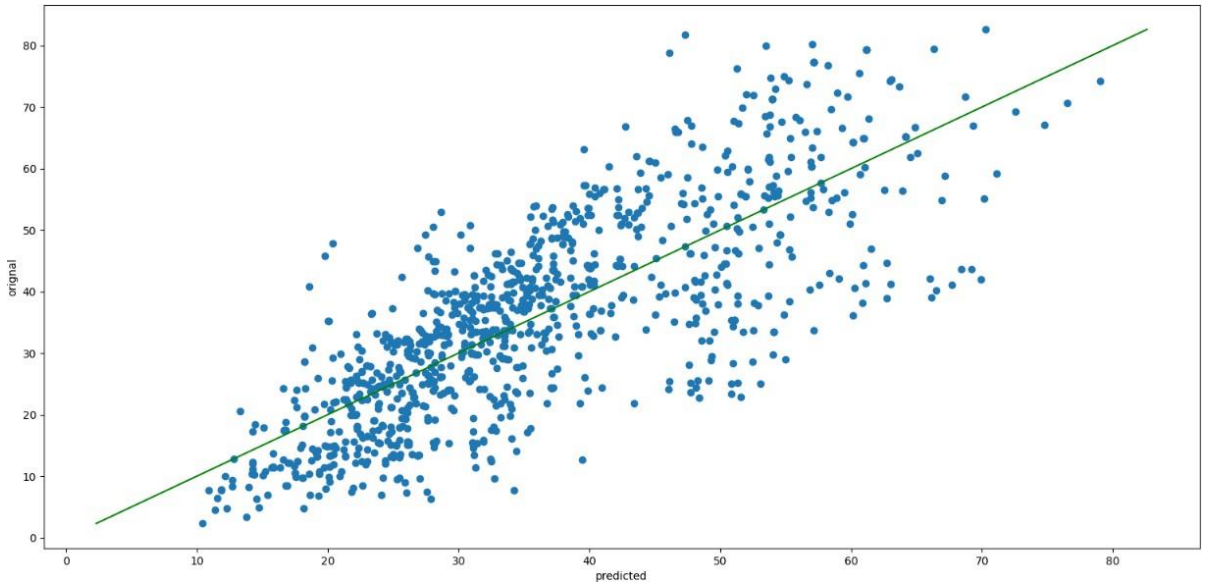
The performance of the created function from scratch was not satisfactory so tried python inbuilt regression function to check the credibility of the created function.

Using python inbuilt regression:

```
# trying the inbuilt regression functionn
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
lr = LinearRegression()
fit = lr.fit(X,Y)
print( '.....................................................................')
y_predict = lr.predict(X)
print( 'mean_ squred_error is ==',mean_squared_error(Y, y_predict))
rms = np.sqrt(mean_squared_error(Y,y_predict))
print( 'root mean squared error is == {}'.format(rms))
print("_____________________________________________The predicted values are_____________________________________________")
y_predict
```

```
.....................................................................
mean_ squred_error is == 107.21180273479739
root mean squared error is == 10.354313243030528
_____________________________________________The predicted values are_____________________________________________

array([53.4728591 , 53.74331185, 56.81194746, ..., 26.47099254,
       29.11564722, 31.89398622])
```



It is found that the error and the predicted value obtained from the created function from scratch and from the python inbuilt function is exactly same. And the plot of the regression is also found the same as shown above. This proves that that the function created from scratch is accurate and less performance of linear regression gives hint to try other regression model like Polynomial Regression.

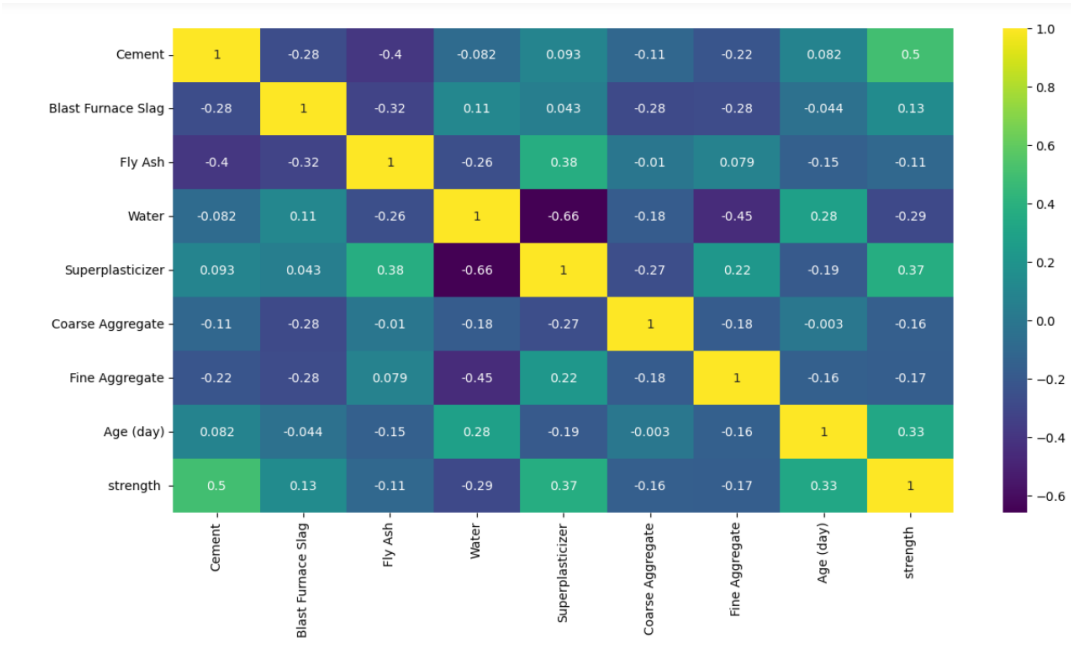
Before applying polynomial regression, we first checked for Multicollinearity and then cleaned the data to improve the performance of the model.

Multicollinearity Check:

Using seaborn, plotted the heatmap which shows the correlation of each variable with one another.

```
# Plot heatmap to check multicollinearity
plt.figure(figsize = (14,7))
sns.heatmap(concrete_df.corr( ), annot=True, cmap='viridis')
```

```
<Axes: >
```



It is noticed that the none of the variables have correlation greater than 0.5 which means the variables are not strongly correlated therefore the problem of Multicollinearity is not severe.

Data Cleaning:

First checked for the duplicates in the data, because duplicated data may bias the model.

```
# check for any duplicate values in the data
duplicates = concrete_df.duplicated()
concrete_df[duplicates]
duplicates.value_counts( )
```

```
False    1005
True       25
Name: count, dtype: int64
```

There are 25 duplicates are found as shown in the code block.

All the duplicates are dropped and then we printed the data frame.

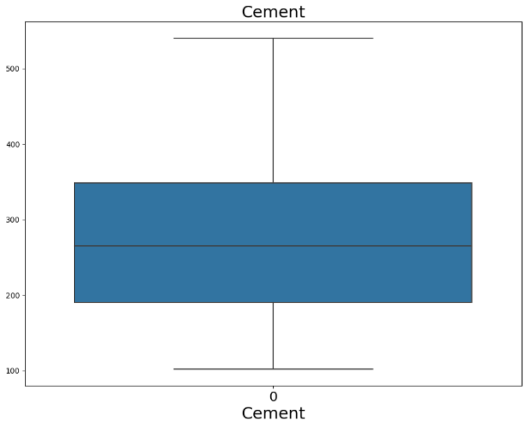
```
# Dropping duplicate values.
concrete_df=concrete_df.drop_duplicates()
concrete_df
```

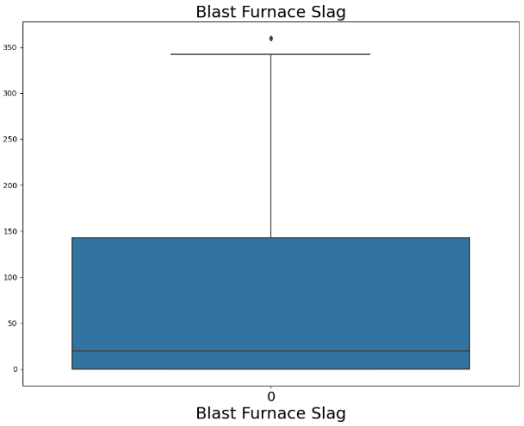
	Cement	Blast Furnace Slag	Fly Ash	Water	Superplasticizer	Coarse Aggregate	Fine Aggregate	Age (day)	strength
0	540.0	0.0	0.0	162.0	2.5	1040.0	676.0	28	79.986111
1	540.0	0.0	0.0	162.0	2.5	1055.0	676.0	28	61.887366
2	332.5	142.5	0.0	228.0	0.0	932.0	594.0	270	40.269535
3	332.5	142.5	0.0	228.0	0.0	932.0	594.0	365	41.052780
4	198.6	132.4	0.0	192.0	0.0	978.4	825.5	360	44.296075
...	...	...	...	...	...	...	...	...	...
1025	276.4	116.0	90.3	179.6	8.9	870.1	768.3	28	44.284354
1026	322.2	0.0	115.6	196.0	10.4	817.9	813.4	28	31.178794
1027	148.5	139.4	108.6	192.7	6.1	892.4	780.0	28	23.696601
1028	159.1	186.7	0.0	175.6	11.3	989.6	788.9	28	32.768036
1029	260.9	100.5	78.3	200.6	8.6	864.5	761.5	28	32.401235

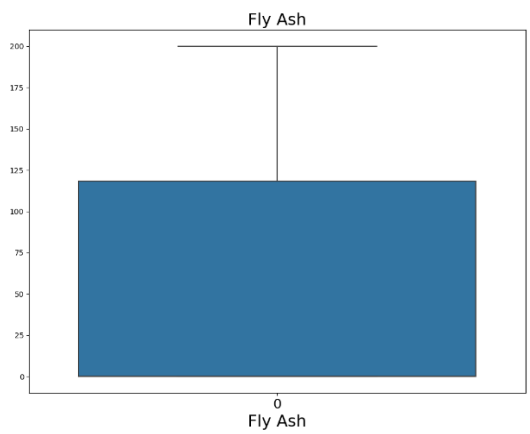
1005 rows × 9 columns

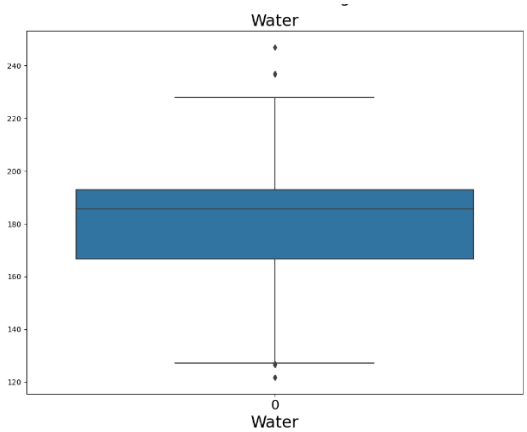
Note that 25 duplicates are dropped and the number of example changed from 1030 to 1005.

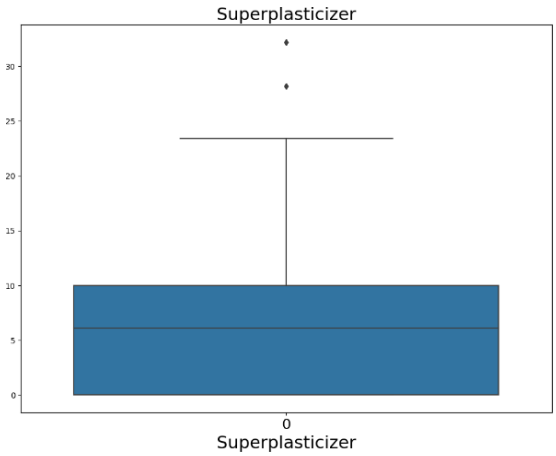
Any outliers present in data is checked using boxplot method the plots are shown below:

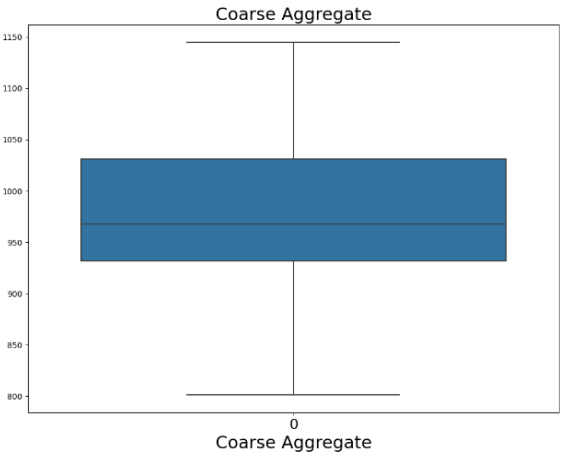


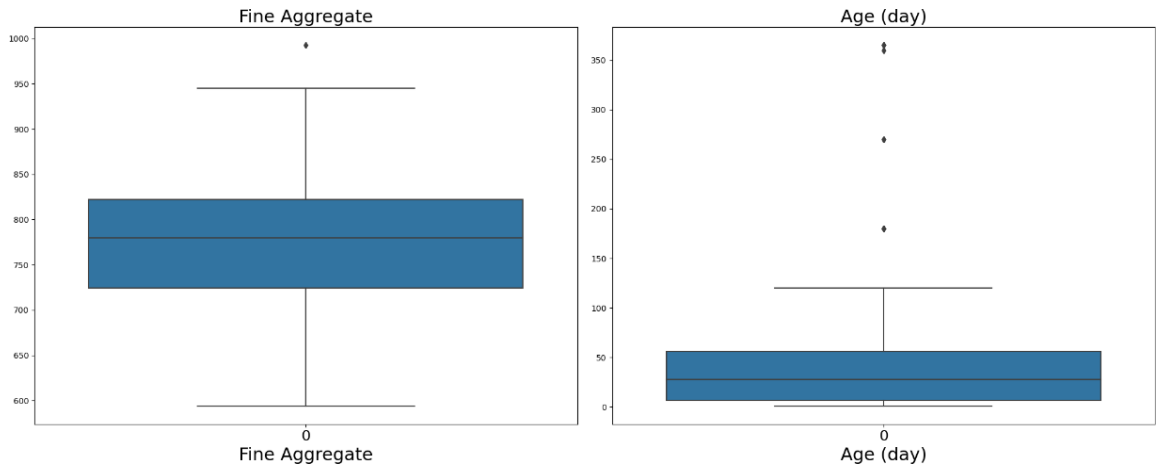












It is noted that some outliers are present in “slag”, “water”, “fine aggregate”, “superplastic” and "age". Further created a function to remove outlier and removed the outlier from each variable whereit was present.

The function to remove outlier is shown below:

```
|: # Function to remove outlier
   def remove_outlier(col):
       sorted(col)
       Q1,Q3=col.quantile([0.25,0.75])
       IQR=Q3-Q1
       lower_range=Q1-(1.5*IQR)
       upper_range=Q3+(1.5*IQR)
       return lower_range, upper_range
```

After preprocessing and cleaning the whole data once again, fitted the linear regression to the data using the function created from scratch.

The observation are as follows:

```
_____The weight parameters are_____

: 0    0.114994
  1    0.090383
  2    0.067183
  3   -0.176913
  4    0.266261
  5    0.005134
  6    0.010068
  7    0.307632
  dtype: object
```

```
_____The fitted equation is_____
y_hat=0.115X_1+0.090X_2+0.067X_3+-0.177X_4+0.266X_5+0.005X_6+0.010X_7+0.308X_8}

_____The Predicted Values are_____
1       54.938918
5         39.7736
7       31.327257
8       20.700388
9       33.665311
          ...
1025    39.747941
1026    33.914072
1027    25.553784
1028    28.750184
1029    31.866135
Length: 911, dtype: object
```

The errors are found as:

```
Mean squared error is:  60.91449792340837
Root mean squared error is: 7.804774046915668
```

It is notable that after cleaning and preprocessing the data. The mean square error has reduced from **107.21** to **60.91** and rmse has reduced from **10.354** to **7.805** . Though decrement in error is not very profound but it proves that the cleaning and preprocessing has improved the performance of the model.

Polynomial Regression:

Having tried various technique it is observed that the performance of the model has not increased very significantly, therefore using the polynomial regression to check if it helps in improving the performance of model.

Imported the polynomial features function from sklearn.preprocessing and then fitted the model on preprocessed and cleaned data as:

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn import linear_model
poly = PolynomialFeatures ( degree=3, interaction_only=False , include_bias=True, order= 'C' )
x = poly.fit_transform(X)
poly_clf = linear_model.LinearRegression()
poly_clf.fit (x, Y)
print(poly_clf.score(x,Y))
```

```
0.9127464768959767
```

Printed the error value determined for the model as shown:

```
print( '.....................................................................')
y_predict = poly_clf.predict(x)
print( 'mean_squared_error is ==' , mean_squared_error(Y,y_predict) )
rms = np.sqrt(mean_squared_error(Y,y_predict))
print( 'root mean squared error is == {} '.format(rms))
```

```
.....................................................................
mean_squared_error is == 21.90218383458496
root mean squared error is == 4.6799769053473925
```

Note that the mean square error has drastically decreased from **107** to **21.9** and root mean squared error decreased from **10.35** to **4.68** which is remarkable.

Created the ANOVA table for the polynomial regression using the created function, the table is:

ANOVA Table

	DF	SS	MS	F
Regression	7	173184.070842	24740.581549	402.585944
Residual	903	55493.107608	61.454161	402.585944
Total	910	228677.178451	24802.035710	402.585944

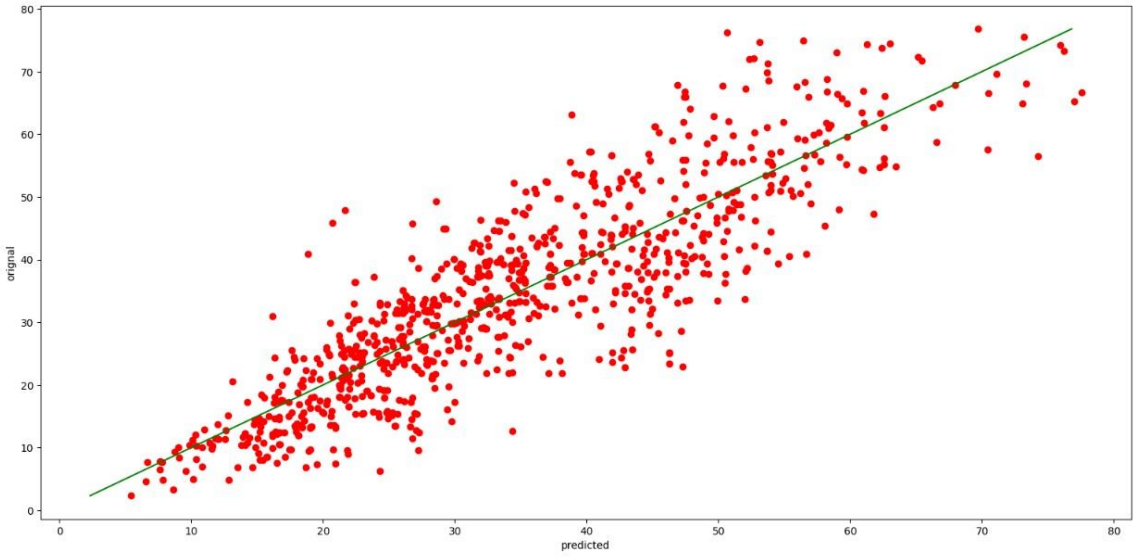
Further printed the R^2 and R^2_{adj} value for the polynomial regression as shown:

```
# Predict R^2 value and adjusted R^2 value:
R_sq = (SSreg /(SSreg + SSres)) * 100
AdjR_sq = (1-MSres/(MSres+MSreg)) * 100
print(f"The R square value is: {R_sq}")
print(f"Adjusted R square value is: {AdjR_sq}")
```

```
The R square value is: 75.73299269117922
Adjusted R square value is: 99.75222130164006
```

It is worth to note that the value of R^2 has boosted from **61.5%** to **75.7%** which is remarkable, thus the polynomial regression is the best fit for the data.

The plot for the polynomial regression model is found as:



Note that the plot looks more fitting to the data than the linear regression model.

Results:

The weights parameters of the polynomial regression are printed as:

```
poly_clf.coef_
```

```
array([ 1.17606455e+06, -1.21797733e+02, -9.77272573e+01, -3.53238111e+02,
       -3.76676743e+02,  7.75298952e+02, -1.18724136e+02, -1.19179471e+02,
        6.76041298e+01,  4.58569683e-02,  9.58166304e-02,  2.45367884e-01,
        2.12243922e-01, -5.60072212e-01,  8.08168849e-02,  1.11840965e-01,
       -2.67501695e-02,  3.75291867e-02,  2.29013504e-01,  1.98304860e-01,
       -3.92699355e-01,  3.46025890e-02,  1.13145075e-01, -1.88148404e-02,
        2.29256932e-01,  6.41643543e-01, -4.57355725e-02,  2.31437232e-01,
        3.36810366e-01, -9.65166160e-02,  3.98257257e-01, -1.86652967e+00,
        3.21019872e-01,  2.71458223e-01, -1.92018557e-01, -4.76295214e-01,
       -2.95132720e-01, -9.73777622e-01,  5.63309521e-01,  4.51403166e-02,
        7.60088285e-02, -6.04224988e-02,  4.35843077e-02, -3.54123114e-02,
        8.47042601e-03, -6.06690449e-06, -2.20257439e-05, -4.76341834e-05,
       -2.63209791e-05,  1.38002560e-04, -1.53867488e-05, -2.34148459e-05,
        2.89178895e-06, -2.38578358e-05, -9.52082411e-05, -6.92818356e-05,
        2.02832857e-04, -2.33398681e-05, -5.41903069e-05,  7.61163967e-06,
       -7.55986215e-05, -1.91994405e-04,  1.47060437e-04, -7.83991785e-05,
       -1.23243164e-04,  3.86209393e-05, -7.56230069e-05,  6.22996491e-04,
       -7.63595393e-05, -1.11817284e-04,  3.69674159e-05,  1.42900063e-04,
        1.13123157e-04,  3.26173952e-04, -3.76857311e-04, -1.38788017e-05,
       -3.44716823e-05,  9.40923818e-06, -2.42730236e-05,  1.26710662e-05,
       -1.86757958e-05, -4.97889407e-06, -4.65634622e-05, -1.95807905e-05,
        1.57180910e-04, -4.83034385e-06, -2.69114087e-05, -3.52807168e-06,
       -7.93913694e-05, -1.84597322e-04,  6.19072050e-05, -5.91607543e-05,
       -1.27288675e-04,  2.64163429e-05, -1.14313037e-04,  2.62856617e-04,
       -3.53889653e-05, -1.23903297e-04,  3.98773406e-05, -2.99252062e-04,
        1.05529347e-04,  2.23772913e-04, -2.23501057e-04, -1.78748337e-07,
       -2.47185019e-05,  6.14690295e-06, -2.84803050e-05,  8.87638963e-06,
       -2.67067311e-05, -4.75932313e-05, -2.07327678e-04, -1.49503068e-04,
       -6.97120488e-05, -1.14466678e-04,  3.70893451e-05, -2.51396202e-04,
        4.58598045e-04, -2.05987040e-04, -3.45137377e-04,  9.02872960e-05,
        2.56145967e-04, -1.17654804e-04,  9.06443595e-05, -2.76394334e-04,
       -3.75746448e-05, -1.13619475e-04,  3.06418525e-05, -7.19746603e-05,
        4.61589425e-05, -2.17619235e-05, -9.22903603e-05,  1.12007320e-03,
       -2.14705714e-04, -1.38522589e-04,  1.02401524e-04, -2.17934993e-03,
        1.68743282e-04,  1.40964584e-03, -2.74292131e-04, -7.58364327e-05,
       -8.54057714e-05,  9.10085563e-05, -5.93615726e-05,  5.62802645e-05,
        5.38778679e-06,  2.62786175e-05, -5.74894183e-05,  8.53079265e-04,
       -1.88916884e-08,  3.27364966e-06,  3.03961969e-04, -2.07485180e-04,
        2.02738758e-04, -2.17015434e-04, -2.99990484e-05, -5.14818501e-06,
       -1.41635045e-05,  1.48010070e-05, -1.19784130e-05,  1.43269037e-05,
       -9.54195496e-06, -4.54048116e-06,  3.98642397e-06, -1.09146830e-05,
        7.88325444e-05])
```

The weight parameters of the linear regression are printed as:

```
_____________________________The weight parameters are_____________________________
[ 0.11335388  0.09623361  0.07931894 -0.18223602  0.26473371  0.01029339
  0.01133186  0.11399624]
```

Having experimented, the results of different fitted regression models are summarized below in the given table:

	Mean Squared Error	Root mean squared error	R^2 %	R^2_{adj} %
Linear Regression	60.914	7.805	61.55	99.51
Polynomial Regression	21.902	4.680	75.73	99.75

Conclusion:

In this report, we have analyzed the compressive strength prediction of concrete using linear regression and ANOVA hypothesis testing. We used a dataset consisting of 1030 examples having 8 features and 1 target variable. We prepared the data by separating the features and the target variable and then applied linear regression to fit the model. We calculated the mean square error and root mean square error for the model, and we performed ANOVA hypothesis testing to determine if all the features were significant or not. The null hypothesis was rejected, which means that all the features are significant. However, we also performed a partial (marginal) test and it was found that one of the regressors is not significant. We further experimented with fitting the linear regression model by discarding the non-significant regressor and found that the performance of the model did not change significantly. We then checked for multicollinearity and cleaned the data by removing both duplicates and outliers. We also checked the performance of the model using the Python inbuilt regression function and found that the performance was similar to the model we created from scratch. Overall, we concluded that the performance of the linear regression model was not satisfactory and this prompted us to try other regression models like polynomial regression.

Trying polynomial regression has resulted to be the best fit for the data. The model has boosted the performance and reduced the errors drastically.