Predicting Compressive strength of Concrete

1. INDTRODUCTION

1.1 Overview

Compressive strength can be defined as **the capacity of concrete to withstand loads before failure**. Of the many tests applied to the concrete, the compressive strength test is the most important, as it gives an idea about the characteristics of the concrete. The compressive strength of concrete is found by considering several materials as per the requirements or amount.

1.2 Purpose

To determine the **Compressive strength of concrete** it takes about 28 days to evaluate. This makes the civil works take a larger amount of time and in turn there will be a delay in the process. So, we are using machine learning to predict the outcome of the strength in least time, which makes the input items also to be manageable and also the overall time required for the material to build and use it also can be derived in a least amount of time.

2. LITERATURE SURVEY

1. Yeh [22] investigated the potential of using neural networks to determine the effect of fly ash replacements on early and late compressive strength of low and high-strength concretes. The research shows the need for a much smaller number of experiments to obtain meaningful data, and the high correlations between the compressive strength and the component composition of concrete can be developed using ANNs. The percentage of the strength of concrete containing fly ash to the strength of concrete without fly ash at the same age is significantly reduced as the fly ash replacement increases.

2. Gandomi et al. [24] proposed a new design equation for the prediction of shear strength of reinforced concrete beams without stirrups using a linear genetic programming methodology. The shear strength was formulated in terms of several effective parameters such as shear span to depth ratio, concrete cylinder strength at the date of testing, amount of longitudinal reinforcement, lever arm, and maximum specified size of coarse aggregate. The results indicate that a derived model is an effective tool for the estimation of the shear capacity of members without stirrups

3. Bayazidi et al. [25] presented a new multigene genetic programming (MGGP) approach for the estimation of the elastic modulus of concrete. The technique models the elastic modulus behaviour by integrating the capabilities of standard genetic programming and classical regression. The research aims to derive precise relationships between the tangent elastic modulus of normal and high strength concrete and the corresponding compressive strength values. Another contribution of this study is to develop a generalized prediction model for the elastic modulus of both normal and high strength concrete. A comprehensive comparative study was conducted to verify the performance of the models. The proposed models perform superior to the existing traditional models, as well as those derived using other powerful soft computing tools.

4. Babanajad et al. [26] proposed a model to correlate the concrete true-triaxial strength to mix design parameters and principal stresses, with no need of conducting any time-

consuming laboratory experiments. A comprehensive true triaxial database was obtained from the literature to build the proposed models, subsequently implemented for verification purposes. The study demonstrates superior performance in comparison to other existing empirical and analytical models.

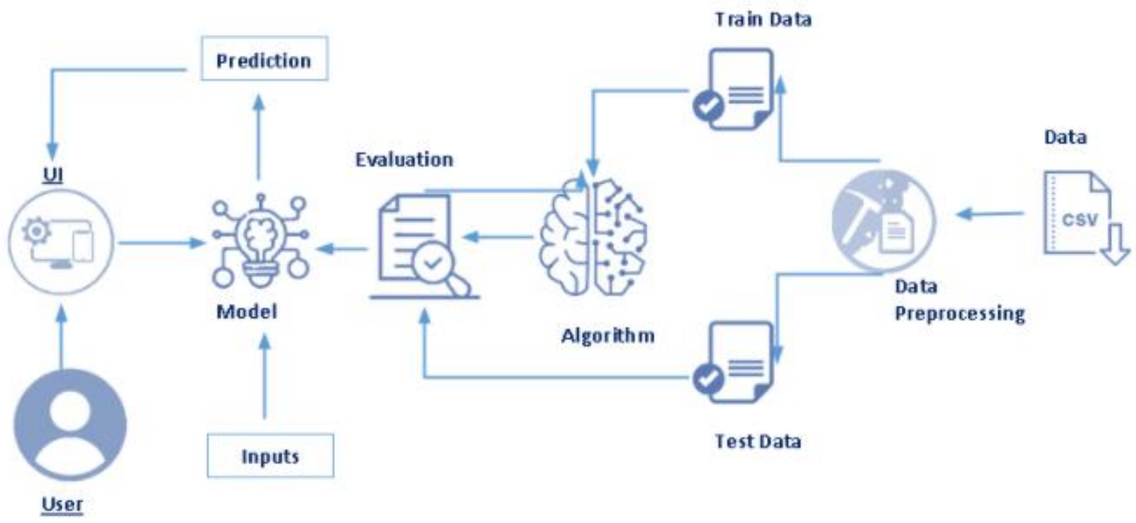
Proposed Solution:

The gradient boosting algorithm (gbm) can be most easily explained by first introducing the AdaBoost Algorithm. The AdaBoost Algorithm begins by training a decision tree in which each observation is assigned an equal weight. After evaluating the first tree, we increase the weights of those observations that are difficult to classify and lower the weights for those that are easy to classify. The second tree is therefore grown on this weighted data. Here, the idea is to improve upon the predictions of the first tree. Our new model is therefore *Tree 1 + Tree 2*. We then compute the classification error from this new 2-tree ensemble model and grow a third tree to predict the revised residuals. We repeat this process for a specified number of iterations. Subsequent trees help us to classify observations that are not well classified by the previous trees. Predictions of the final ensemble model is therefore the weighted sum of the predictions made by the previous tree models.

Gradient Boosting trains many models in a gradual, additive and sequential manner. The major difference between AdaBoost and Gradient Boosting Algorithm is how the two algorithms identify the shortcomings of weak learners (e.g., decision trees). While the AdaBoost model identifies the shortcomings by using high weight data points, gradient boosting performs the same by using gradients in the loss function (*y=ax+b+e, e needs a special mention as it is the error term)*. The loss function is a measure indicating how good are model's coefficients are at fitting the underlying data. A logical understanding of loss function would depend on what we are trying to optimise. For example, if we are trying to predict the sales prices by using a regression, then the loss function would be based off the error between true and predicted house prices. Similarly, if our goal is to classify credit defaults, then the loss function would be a measure of how good our predictive model is at classifying bad loans. One of the biggest motivations of using gradient boosting is that it allows one to optimise a user specified cost function, instead of a loss function that usually offers less control and does not essentially correspond with real world applications.

THEORITICAL ANALYSIS:

ARCHITECTURE:



REQUIREMENTS:

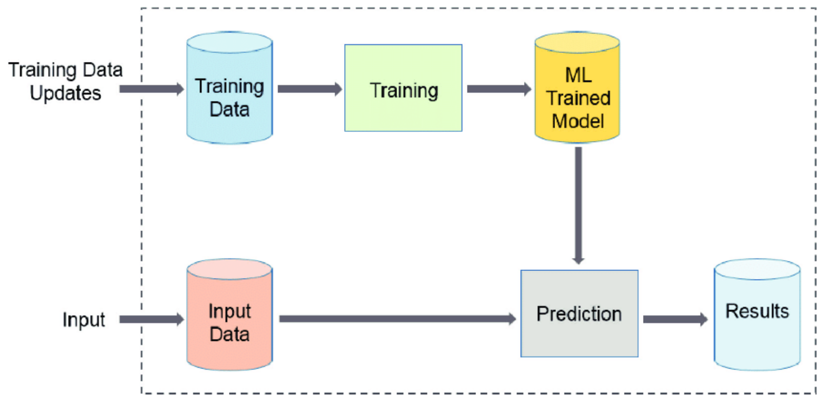
To complete this project, you need the following:

- Jupyter Notebook
- Spyder
- Python Library's Included:
    - Pandas
    - NumPy
    - Matplotlib
    - Flask
    - Sklearn

Experimental Investigation:

The analysis shows that the compressive strength of concrete depends on 8 different factors, which has nonlinear dependency. Concrete is the most important material in civil engineering. The concrete compressive strength is a highly nonlinear function of age and ingredients. These ingredients include cement, blast furnace slag, fly ash, water, superplasticizer, coarse aggregate, and fine aggregate.

Control Flow of the Solution:



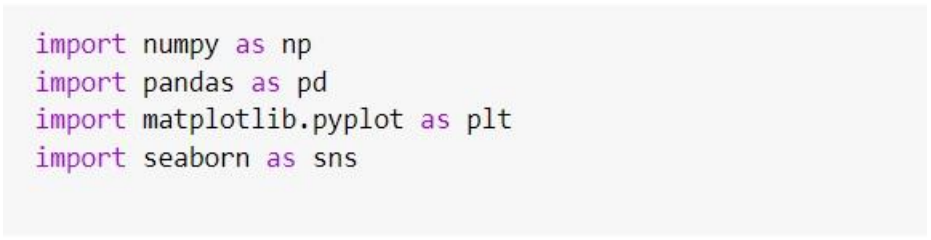
To start with the project, we project flow considered as:

- Data Collection
- Data Pre-processing
    - Importing Libraries
    - Reading the dataset
    - Processing the data
    - Finding the Missing data
    - Label Encoding
    - Data Visualization
    - Splitting the dataset in Dependent and Independent Variables
    - Splitting Data for testing and training
- Model Building

        - Training and testing the model
        - Evaluation of Model
    - Application Building
        - Creating the UI using HTML
        - Integrating the Flask for its backend to interact with our ML Model.

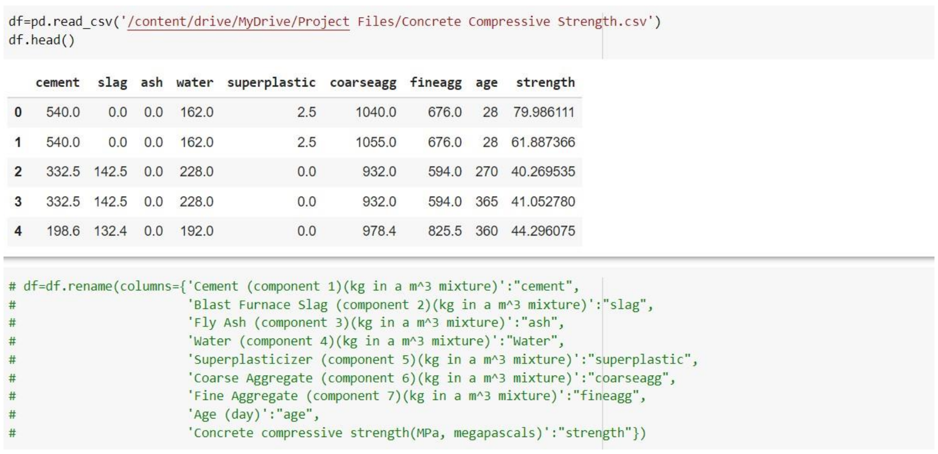
Data Preprocessing:

Importing Libraries



```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Reading and processing the data:

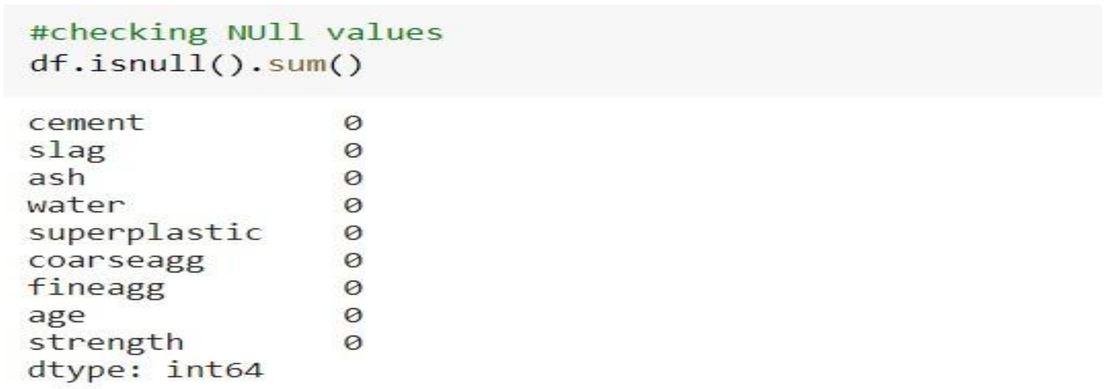


```
df=pd.read_csv('/content/drive/MyDrive/Project Files/Concrete Compressive Strength.csv')
df.head()
```

	cement	slag	ash	water	superplastic	coarseagg	fineagg	age	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

```
# df=df.rename(columns={'Cement (component 1)(kg in a m^3 mixture)':"cement",
#                        'Blast Furnace Slag (component 2)(kg in a m^3 mixture)':"slag",
#                        '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)':"superplastic",
#                        'Coarse Aggregate (component 6)(kg in a m^3 mixture)':"coarseagg",
#                        'Fine Aggregate (component 7)(kg in a m^3 mixture)':"fineagg",
#                        'Age (day)':"age",
#                        'Concrete compressive strength(MPa, megapascals)':"strength"})
```

Finding Missing values:

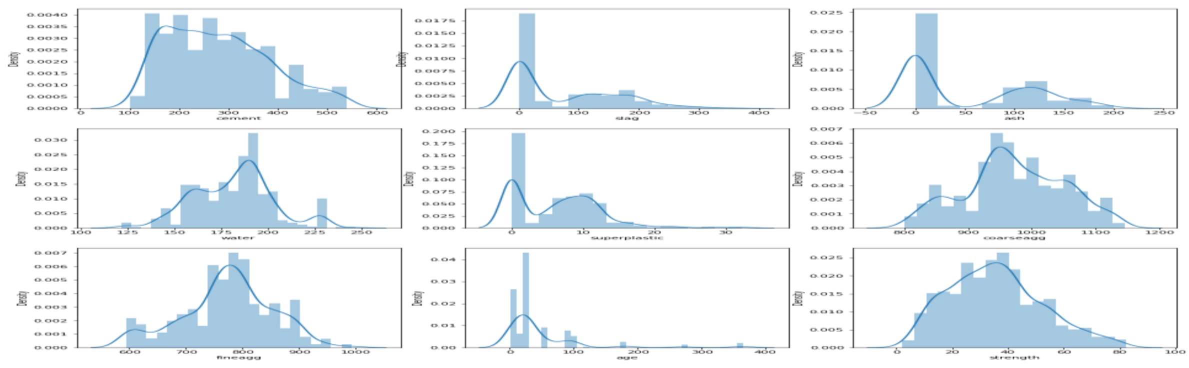


```
#checking NUll values
df.isnull().sum()
```

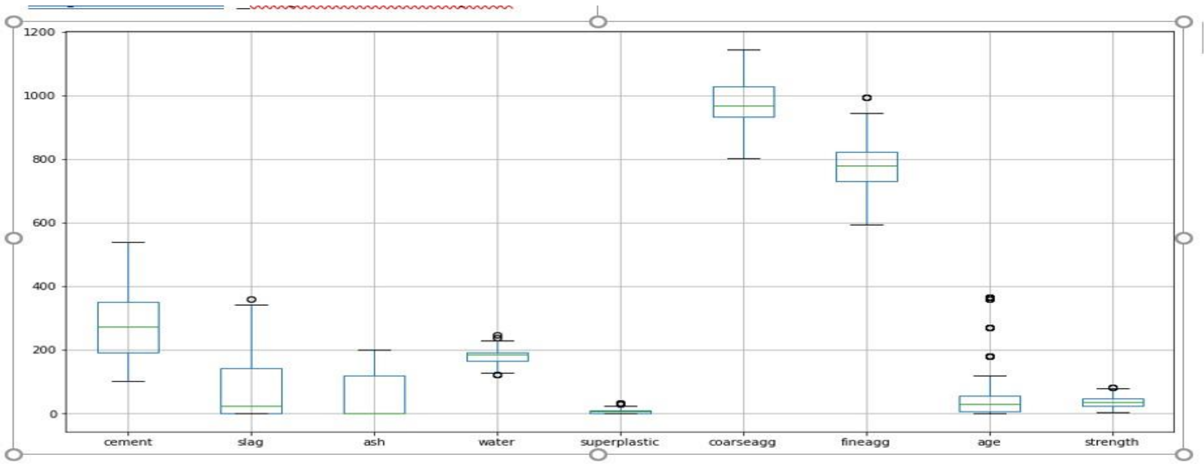
```
cement          0
slag            0
ash             0
water           0
superplastic    0
coarseagg       0
fineagg         0
age             0
strength        0
dtype: int64
```

Data visualization:

1.Multi variable analysis:

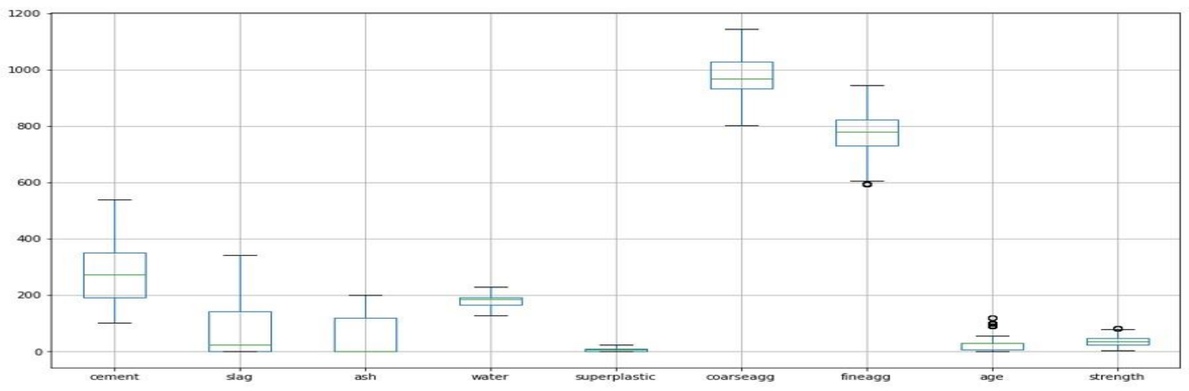


Finding outliers:



Since we have found outliers present and that may effect the actual outcome , so we are going to remove to minimalize the outlier by considering the median value of there respective column by considering the quantile

Reducing Outliers:

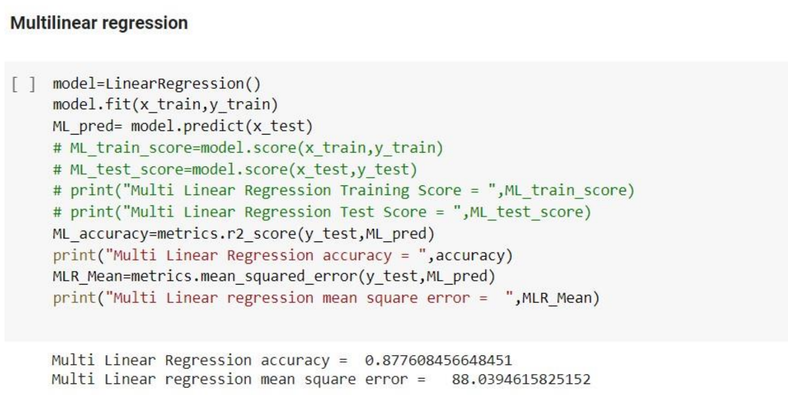


now we can see that the outliers are minimized

Splitting Data as Training and Testing:

```
]  x_train,x_test,y_train,y_test =train_test_split(xscaled,y,test_size=0.3,random_state=1)
```

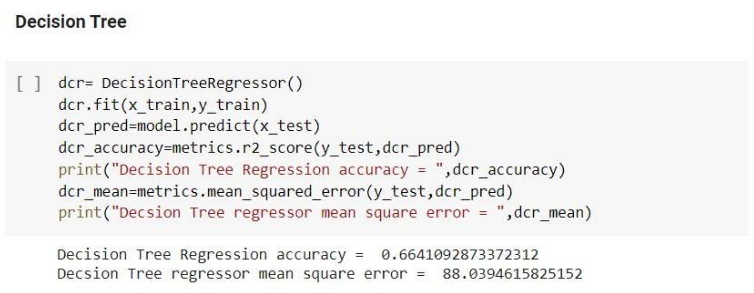
Training and picking the best model:



Multilinear regression

```
[ ]  model=LinearRegression()
     model.fit(x_train,y_train)
     ML_pred= model.predict(x_test)
     # ML_train_score=model.score(x_train,y_train)
     # ML_test_score=model.score(x_test,y_test)
     # print("Multi Linear Regression Training Score = ",ML_train_score)
     # print("Multi Linear Regression Test Score = ",ML_test_score)
     ML_accuracy=metrics.r2_score(y_test,ML_pred)
     print("Multi Linear Regression accuracy = ",accuracy)
     MLR_Mean=metrics.mean_squared_error(y_test,ML_pred)
     print("Multi Linear regression mean square error =  ",MLR_Mean)
```

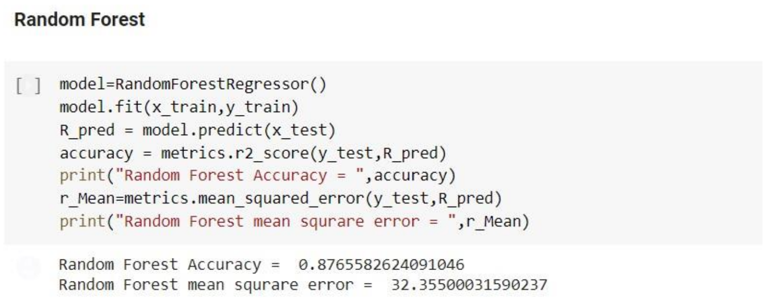
```
     Multi Linear Regression accuracy =  0.877608456648451
     Multi Linear regression mean square error =   88.0394615825152
```



Decision Tree

```
[ ]  dcr= DecisionTreeRegressor()
     dcr.fit(x_train,y_train)
     dcr_pred=model.predict(x_test)
     dcr_accuracy=metrics.r2_score(y_test,dcr_pred)
     print("Decision Tree Regression accuracy = ",dcr_accuracy)
     dcr_mean=metrics.mean_squared_error(y_test,dcr_pred)
     print("Decsion Tree regressor mean square error = ",dcr_mean)
```

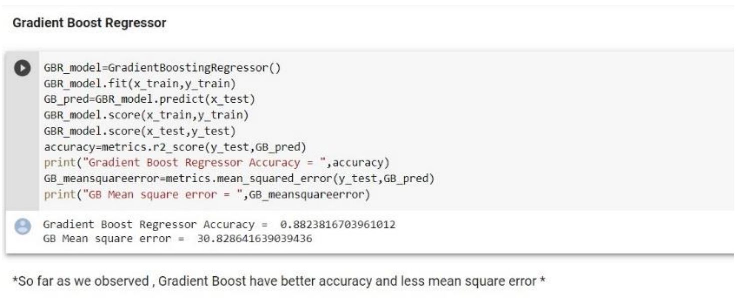
```
     Decision Tree Regression accuracy =  0.6641092873372312
     Decsion Tree regressor mean square error =  88.0394615825152
```



Random Forest

```
[ ]  model=RandomForestRegressor()
     model.fit(x_train,y_train)
     R_pred = model.predict(x_test)
     accuracy = metrics.r2_score(y_test,R_pred)
     print("Random Forest Accuracy = ",accuracy)
     r_Mean=metrics.mean_squared_error(y_test,R_pred)
     print("Random Forest mean squrare error = ",r_Mean)
```

```
     Random Forest Accuracy =  0.8765582624091046
     Random Forest mean squrare error =  32.35500031590237
```



Gradient Boost Regressor

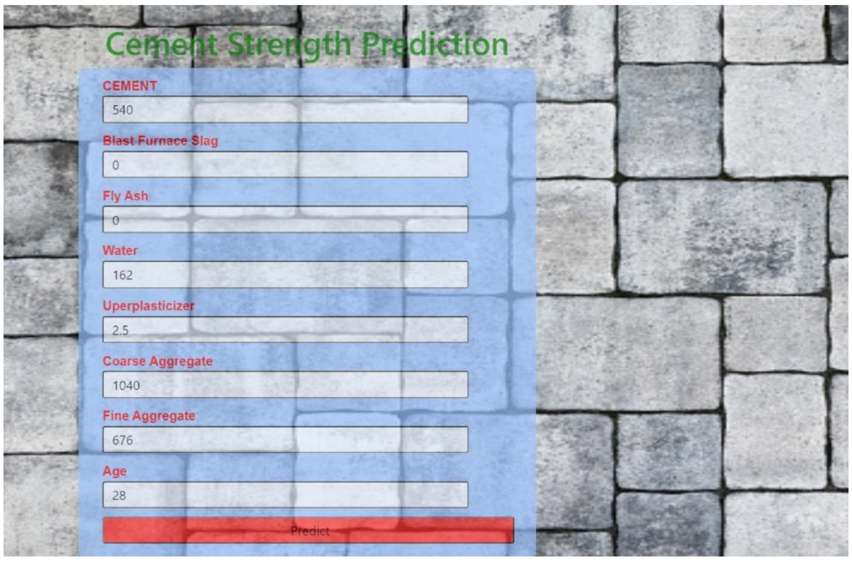
```
GBR_model=GradientBoostingRegressor()
GBR_model.fit(x_train,y_train)
GB_pred=GBR_model.predict(x_test)
GBR_model.score(x_train,y_train)
GBR_model.score(x_test,y_test)
accuracy=metrics.r2_score(y_test,GB_pred)
print("Gradient Boost Regressor Accuracy = ",accuracy)
GB_meansquareerror=metrics.mean_squared_error(y_test,GB_pred)
print("GB Mean square error = ",GB_meansquareerror)
```

```
Gradient Boost Regressor Accuracy =  0.8823816703961012
GB Mean square error =  30.828641639039436
```

*So far as we observed , Gradient Boost have better accuracy and less mean square error *

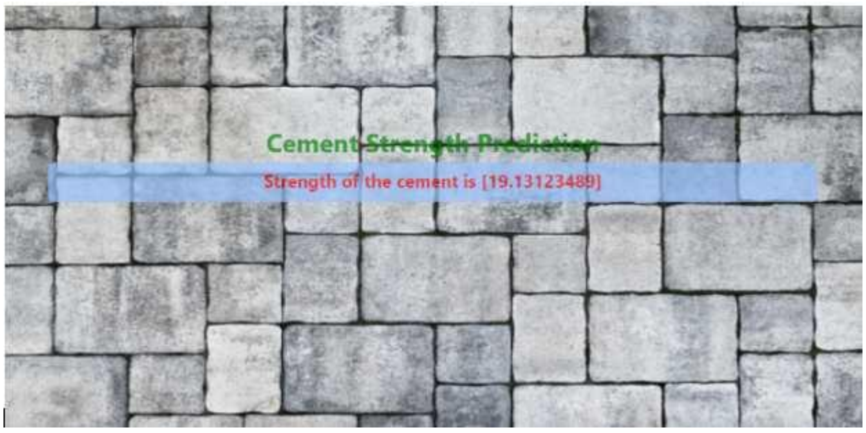
Result:

- user Interacts with the User Interface (i.e., Flask Application) to enter the input parameters.
- The Input values are given to our Trained Machine Learning Model and the model starts analyzing the input
- After the analysis is done by the model it gives a result and the User can able to see that Output in the Flask Application (UI) that we created.



Cement Strength Prediction

CEMENT
540
Blast Furnace Slag
0
Fly Ash
0
Water
162
Uperplasticizer
2.5
Coarse Aggregate
1040
Fine Aggregate
676
Age
28



Cement Strength Prediction

Strength of the cement is [19.13123489]

ADVANTAGES :

1. Time Management:

The proposed model can be able to predict the strength of the concrete over different age or time , just by giving the values as input , which inturn can be able to reduce the Time required to physically check the strength.

2. Cost Efficient:

Since we are directly putting the values and finding the strength of concrete , the overall cost and amount required to make the concrete can also be predicted , which makes the client or user to make decisions based on the amount available.

DISADVANTAGES :

1. Accuracy:

Since we are only considering the model based on the limited amount of data, the accuracy of the model to predict the accurate compressive strength of concrete is not equal. It is the from the model we trained cannot say that we are having 100 % accuracy, but the accuracy of the prediction is almost nearer.

Conclusion:

From this we can say that Gradient Boost Regressor has performed well in predicting the Compressive Strength of Concrete.

Future Scope:

If we can able to select an algorithm that reduces the error and increases the efficiency, then we can able to get the accurate Compressive strength of the concrete.

BIBILOGRAPHY:

- Towards Data science article
- MDN – For Frontend design