

Predicting Compressive Strength Of Concrete Using Machine Learning

Introduction:

1.1 Project overviews :

This project involves predicting the compressive strength of concrete based on its components using machine learning techniques. The dataset includes various features such as the quantities of cement, slag, fly ash, water, superplasticizer, coarse aggregate, and fine aggregate used in the concrete mix, along with the age of the concrete. The goal is to build a predictive model that accurately estimates the concrete's compressive strength.

1.2 Objectives :

The primary objective of this project is to develop and deploy a machine learning model that accurately predicts the compressive strength of concrete based on its mix composition, curing conditions, and other relevant parameters. The process involves developing and training a machine learning model using a chosen algorithm, evaluating its performance using metrics like MAE and MSE, and implementing it into a user- friendly interface for real-time compressive strength predictions in construction projects.

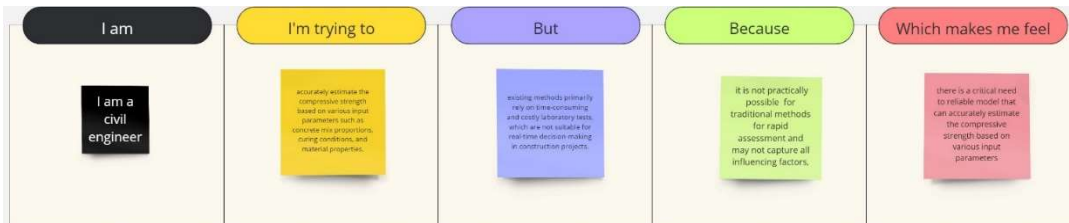
1. Project Initialization and Planning Phase

2.1 Define Problem Statement:

One of the primary aspects that govern the durability and safety of structures in construction is concrete's compressive strength. Sadly, it is hard to come by a precise anticipation on this strength; as various factors take part in determining it, and their relationships are not only multiple but intertwined complexly—such as mix proportions entangled with material properties and curing conditions. the techniques used at present mostly depend on

tests conducted in labs that take both time and money; they do not help decision makers know what to do when as they cannot support any real-time initiative towards effective project management.

Customer Problem Statement Template:



Refer : [Problem statement template](#)

2.2 Project Proposal (Proposed Solution) :

The proposed solution is to use the GradientBoostingRegressor from the scikit-learn library to build the predictive model. The model will be trained on the preprocessed dataset and optimized through hyperparameter tuning. The final model will be deployed using a Flask web application, allowing users to input concrete mix values and receive compressive strength predictions.

Refer: [Project proposal Template](#)

2.3 Initial Project Planning : [Project flow](#)

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members	Sprint Start Date	Sprint End Date (Planned)
Sprint-1	Define Problem / Problem Understanding	CS-1	Identify Business objective .	1	Medium	Rakesh	12-07-24	12-07-24
Sprint-1	Define Problem / Problem Understanding	CS-2	Understanding required features to solve the problem.	1	Low	Jayanth	12-07-24	12-07-24
Sprint-1	Define Problem / Problem Understanding	CS-3	Formulate the Problem Statement	2	Medium	Dattu	13-07-24	13-07-24
Sprint-2	Data collection	CS-4	Collect dataset	2	Medium	Manvitha	14-07-24	14-07-24
Sprint-3	Data Pre-Processing	CS-6	Importing Dataset and Required Libraries	1	Low	Dattu	14-07-24	14-07-24
Sprint-3	Data Pre-Processing	CS-7	Finding and Handling Missing Values	2	Medium	Manvitha	14-07-24	14-07-24
Sprint-3	Data Pre-Processing	CS-8	Data Visualization	1	Low	Jayanth	14-07-24	14-07-24
Sprint-3	Data Pre-Processing	CS-9	Splitting Dataset into Train and Test sets	1	Low	Rakesh	15-07-24	15-07-24
Sprint-4	Model Building	CS-10	Training and Testing Model	2	Medium	Rakesh	15-07-24	15-07-24
Sprint-4	Model Building	CS-11	Evaluate Model and save model	1	Low	Manvitha	16-07-24	16-07-24
Sprint-5	Application Building	CS-12	Creating HTML pages	2	Medium	Jayanth	16-07-24	16-07-24
Sprint-5	Application Building	CS-13	Build Python Flask Code	2	Medium	Manvitha	17-07-24	17-07-24
Sprint-5	Application Building	CS-14	Run App	1	Low	Dattu	17-07-24	17-07-24

3. Data Collection and Preprocessing Phase

3.1 Data Collection Plan and Raw Data Sources Identified :

Assemble a comprehensive dataset with tangible sample records containing the following data: proportions of each sample's coarse, fine, and water-based aggregates. The duration in days since the concrete was poured. Measurements of the target variable, or compressive strength, for every sample.

The raw data sources for this project include datasets obtained from Kaggle, the popular platforms for data science competitions and repositories. The dataset contains components of concrete mixture like cement, blast furnace slag, fly ash, water, superplasticizer, coarse aggregate, fine aggregate, and the age of the concrete.

Dataset used in Project:

<https://www.kaggle.com/datasets/elikplim/concrete-compressive-strength-data-set>

Refer : [Data collection Plan Template](#)

3.2 Data Quality Report :

- The dataset contains no missing values.
- All features are numerical.
- There are potential outliers in some features

Refer: [Data report](#)

3.3 Data Exploration and Preprocessing :

Data Overview:

- **Basic statistics:** Mean, median, standard deviation, min, max values for each feature.

```
data.describe()
```

	cement	blast_furnace_slag	fly_ash	water	superplasticizer	coarse_aggregate	fine_aggregate	age	concrete_compressive_strength
count	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000
mean	281.167864	73.895825	54.188350	181.567282	6.204660	972.918932	773.580485	45.662136	35.817961
std	104.506364	86.279342	63.997004	21.354219	5.973841	77.753954	80.175980	63.169912	16.705742
min	102.000000	0.000000	0.000000	121.800000	0.000000	801.000000	594.000000	1.000000	2.330000
25%	192.375000	0.000000	0.000000	164.900000	0.000000	932.000000	730.950000	7.000000	23.710000
50%	272.900000	22.000000	0.000000	185.000000	6.400000	968.000000	779.500000	28.000000	34.445000
75%	350.000000	142.950000	118.300000	192.000000	10.200000	1029.400000	824.000000	56.000000	46.135000
max	540.000000	359.400000	200.100000	247.000000	32.200000	1145.000000	992.600000	365.000000	82.600000

- **Dimensions:** The dataset contains 1030 samples and 9 features.
- **Structure:** All features are numerical, with the target variable being the compressive strength of concrete.

```
: data.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   blast_furnace_slag                   1030 non-null   float64
2   fly_ash                              1030 non-null   float64
3   water                                1030 non-null   float64
4   superplasticizer                     1030 non-null   float64
5   coarse_aggregate                     1030 non-null   float64
6   fine_aggregate                       1030 non-null   float64
7   age                                  1030 non-null   int64
8   concrete_compressive_strength        1030 non-null   float64
dtypes: float64(8), int64(1)
```

Null values:
There are no null values in the dataset.

```
data.isnull().sum()

cement          0
blast_furnace_slag  0
fly_ash         0
water           0
superplasticizer  0
coarse_aggregate  0
fine_aggregate   0
age             0
concrete_compressive_strength  0
dtype: int64
```

Data Analysis & Visualization:
Univariate Analysis:

```
data.describe()
```

	cement	blast_furnace_slag	fly_ash	water	superplasticizer	coarse_aggregate	fine_aggregate	age	concrete_compressive_strength
count	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000
mean	281.167864	73.895825	54.188350	181.567282	6.204660	972.918932	773.580485	45.662136	35.817961
std	104.506364	86.279342	63.997004	21.354219	5.973841	77.753954	80.175980	63.169912	16.705742
min	102.000000	0.000000	0.000000	121.800000	0.000000	801.000000	594.000000	1.000000	2.330000
25%	192.375000	0.000000	0.000000	164.900000	0.000000	932.000000	730.950000	7.000000	23.710000
50%	272.900000	22.000000	0.000000	185.000000	6.400000	968.000000	779.500000	28.000000	34.445000
75%	350.000000	142.950000	118.300000	192.000000	10.200000	1029.400000	824.000000	56.000000	46.135000
max	540.000000	359.400000	200.100000	247.000000	32.200000	1145.000000	992.600000	365.000000	82.600000

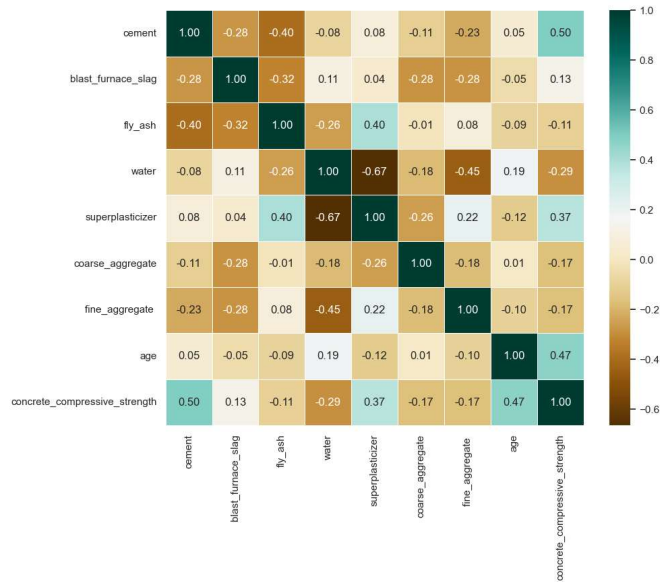
Bivariate Analysis:
Correlation matrix:
Identifies the strength of relationships between pairs of variables

```
data.corr()
```

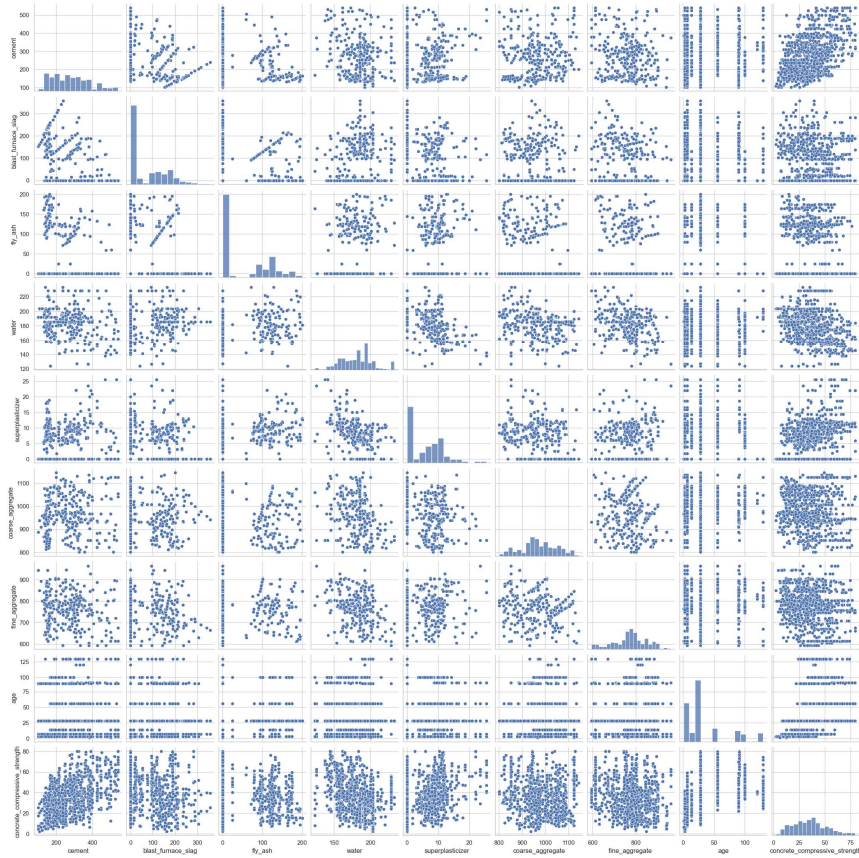
	cement	blast_furnace_slag	fly_ash	water	superplasticizer	coarse_aggregate	fine_aggregate	age	concrete_compressive_strength
cement	1.000000	-0.275239	-0.397467	-0.081617	0.079619	-0.109349	-0.225311	0.054325	0.498008
blast_furnace_slag	-0.275239	1.000000	-0.323590	0.107031	0.043724	-0.284023	-0.283263	-0.052703	0.134621
fly_ash	-0.397467	-0.323590	1.000000	-0.258379	0.395319	-0.009961	0.080962	-0.092704	-0.105577
water	-0.081617	0.107031	-0.258379	1.000000	-0.665260	-0.180249	-0.448979	0.189637	-0.290969
superplasticizer	0.079619	0.043724	0.395319	-0.665260	1.000000	-0.261624	0.216947	-0.122913	0.366235
coarse_aggregate	-0.109349	-0.284023	-0.009961	-0.180249	-0.261624	1.000000	-0.176542	0.012295	-0.165250
fine_aggregate	-0.225311	-0.283263	0.080962	-0.448979	0.216947	-0.176542	1.000000	-0.096565	-0.169601
age	0.054325	-0.052703	-0.092704	0.189637	-0.122913	0.012295	-0.096565	1.000000	0.469807
concrete_compressive_strength	0.498008	0.134621	-0.105577	-0.290969	0.366235	-0.165250	-0.169601	0.469807	1.000000

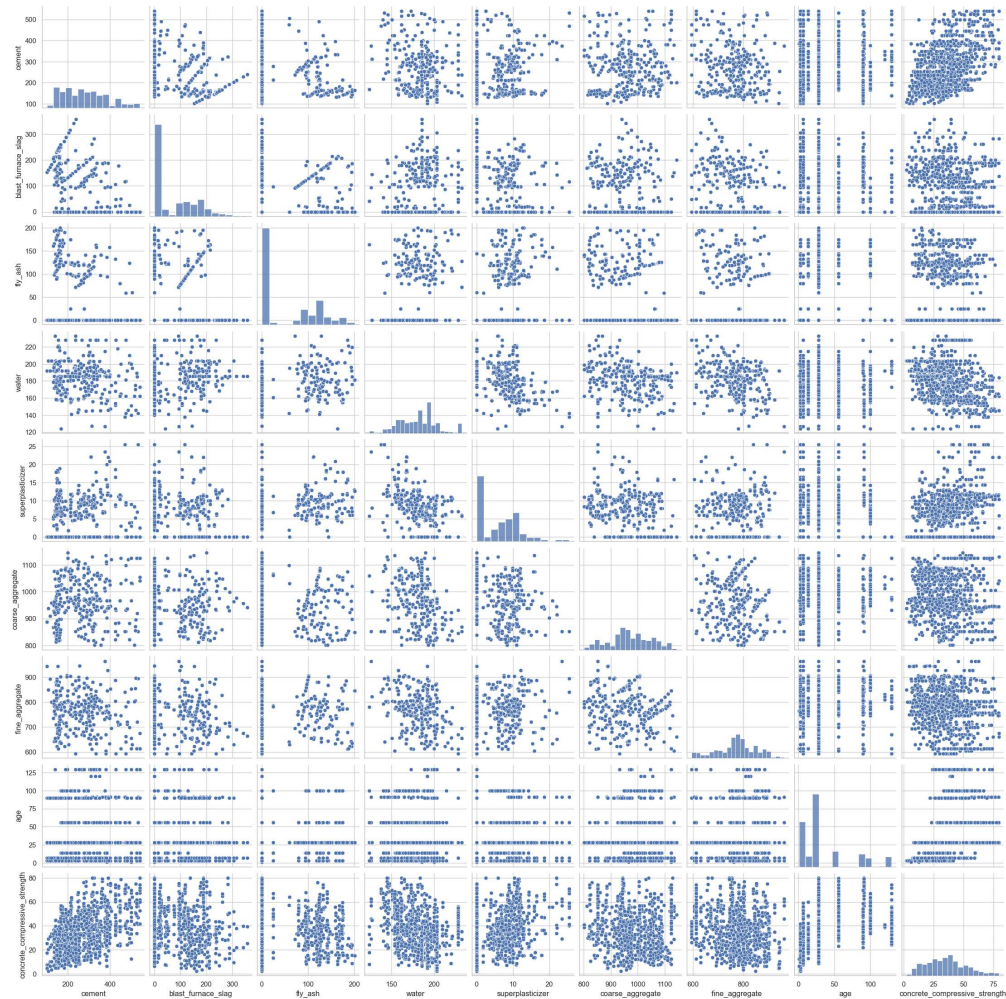
Multivariate Analysis:

Heatmap of correlations: Visualize the correlation matrix



Pair plots: Explore relationships among multiple variables.

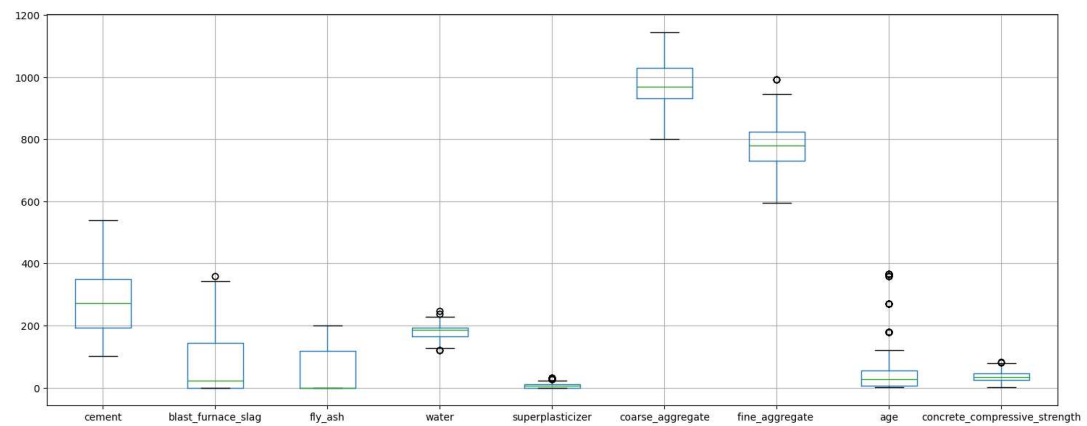




Outliers Treatment:

There are outliers in the dataset columns. We can treat outliers by IQR method.

Outliers:

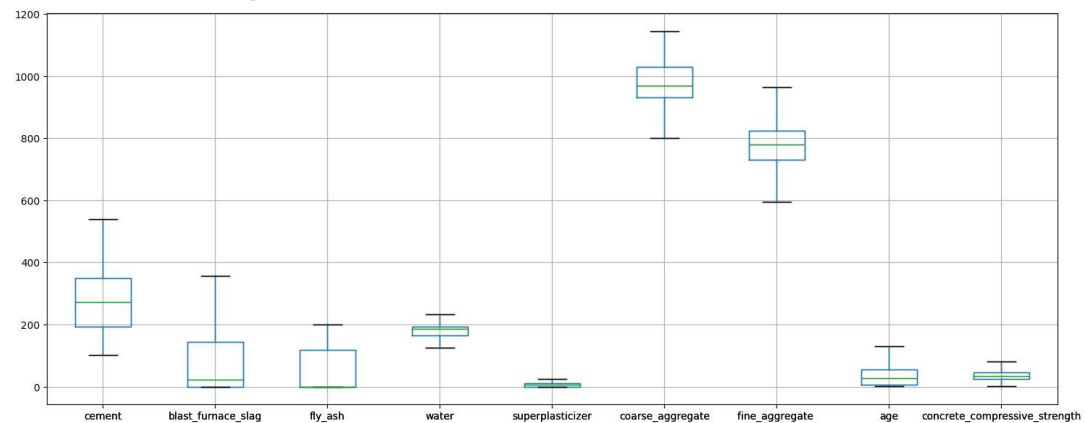


Outliers treatment:

```
# Define a function to handle outliers using the IQR method
def handle_outliers(df):
    for column in df.columns:
        if df[column].dtype != 'object':
            Q1 = df[column].quantile(0.25)
            Q3 = df[column].quantile(0.75)
            IQR = Q3 - Q1
            lower_bound = Q1 - 1.5 * IQR
            upper_bound = Q3 + 1.5 * IQR
            df[column] = np.where(df[column] < lower_bound, lower_bound, df[column])
            df[column] = np.where(df[column] > upper_bound, upper_bound, df[column])
    return df
```

```
# Apply the outlier treatment function to the data
data = handle_outliers(data)
```

After Removing Outliers:



Now data is pre-processed .

Refer: [Data Pre-processing Report](#)

4. Model Development Phase

Feature Selection Report :

All features are selected for model training as they contribute to the prediction of the target variable.

It includes the following features:

- Cement
- Slag
- Fly Ash
- Water
- Superplasticizer
- Coarse Aggregate
- Fine Aggregate
- Age
- Concrete Compressive Strength (target variable)

Refer: [Feature Selection Report](#)

3.4 Model Selection Report :

The GradientBoosting Regressor is used to predict the compressive strength of concrete by iteratively improving weak prediction models through gradient descent. It combines multiple decision trees to minimize prediction errors and enhance accuracy.

Refer: [Model selection Report](#)

3.5 Initial Model Training Code, Model Validation and Evaluation Report :

The GradientBoostingRegressor is chosen for its ability to handle non-linearity and its robustness in predictive performance.

```
gb = GradientBoostingRegressor()
gb.fit(x_train,y_train)
y_pre=gb.predict(x_test)
print(y_pre)
print("R2 score :",r2_score(y_test,y_pre))
```

Refer : [Initial Model](#)

5. Model Optimization and Tuning Phase

3.6 Hyperparameter Tuning Documentation :

Grid Search with Cross-Validation: GridSearchCV performs an exhaustive search over the parameter grid, using 5-fold cross-validation to evaluate each combination

```
param_grid = { 'n_estimators': [100, 200, 300], 'learning_rate': [0.01, 0.05, 0.1], 'max_depth': [3, 4, 5], 'subsample': [0.8, 0.9, 1.0], 'min_samples_split': [2, 5, 10], 'min_samples_leaf': [1, 2, 4] }
```

- **n_estimators:** The number of boosting stages to be run.
- **learning_rate:** The learning rate shrinks the contribution of each tree.
- **max_depth:** The maximum depth of the individual regression estimators.
- **subsample:** The fraction of samples to be used for fitting the individual base learners.
- **min_samples_split:** The minimum number of samples required to split an internal node.
- **min_samples_leaf:** The minimum number of samples required to be at a leaf node.

3.7 Performance Metrics Comparison Report

Before Hyperparameter tuning:

Baseline Metric:

R2 score : 0.8863409219454843

After Hyperparameter tuning:

Model Evaluation

```
print("Mean Absolute Error:", mean_absolute_error(y_test,y_pred))
print("Mean Squared Error:", mean_squared_error(y_test,y_pred))
print("Root Mean Square Error:", np.sqrt(mean_squared_error(y_test,y_pred)))
print("R2 score :", r2_score(y_test,y_pred))
```

Mean Absolute Error: 3.2827032042535604

Mean Squared Error: 22.38162773605611

Root Mean Square Error: 1.8118231713535293

R2 score : 0.9137489588786115

3.8 Final Model Selection Justification

We selected the GradientBoostingRegressor for its ability to handle non-linearity and provide robust predictions by combining multiple weak learners. GridSearchCV was utilized to systematically explore the hyperparameter space and identify the best parameter combination, enhancing the model's performance and generalization ability. This combination ensures optimal accuracy and efficiency for our predictive task. The accuracy of prediction increased from 88% to 91%.

Refer: [Model Optimization and Tuning Phase](#)

4 Results:

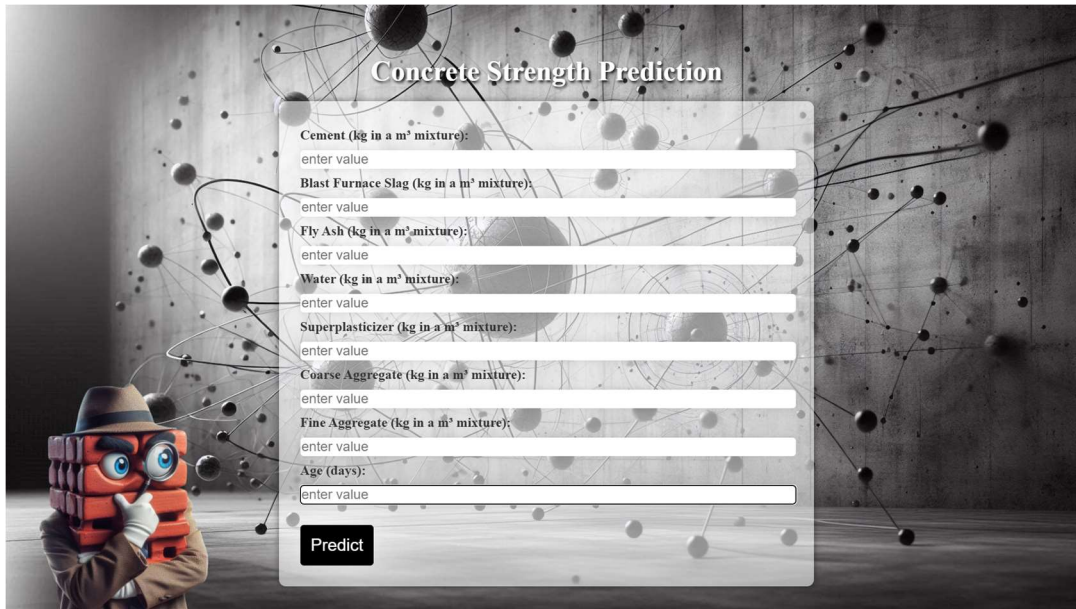
Saved the best model using pickle library and test the model

4.2 Output Screenshots

Home page:



Input Page:



The input page features a dark, abstract background with a network of black lines and spheres. On the left, a cartoon character made of red bricks, wearing a brown trench coat and a fedora, stands with its hand on its chin. A semi-transparent white form is centered on the page, containing the following fields:

Concrete Strength Prediction

Cement (kg in a m³ mixture):
enter value

Blast Furnace Slag (kg in a m³ mixture):
enter value

Fly Ash (kg in a m³ mixture):
enter value

Water (kg in a m³ mixture):
enter value

Superplasticizer (kg in a m³ mixture):
enter value

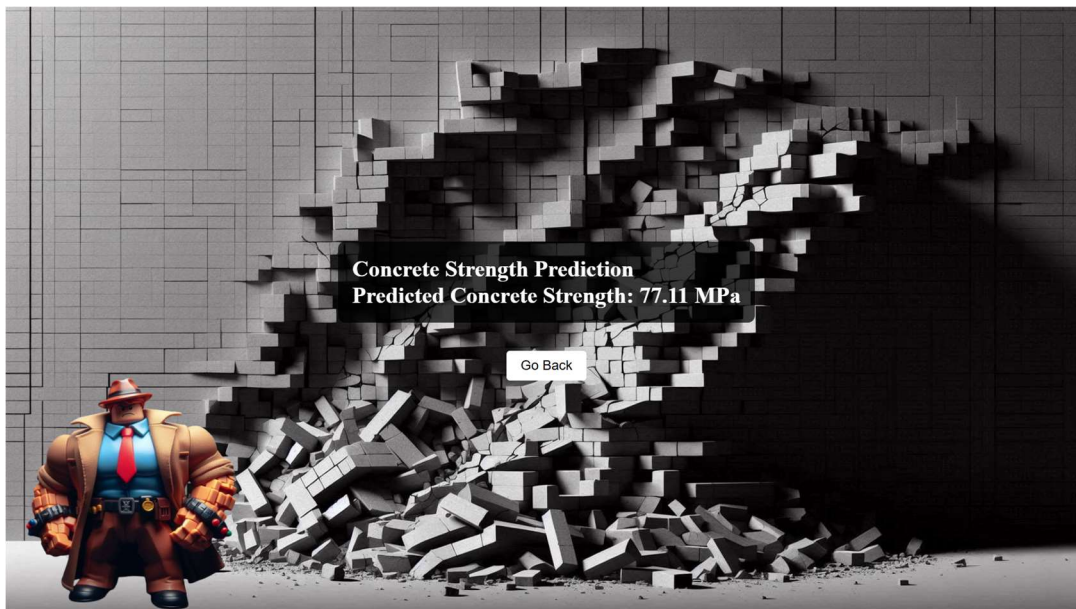
Coarse Aggregate (kg in a m³ mixture):
enter value

Fine Aggregate (kg in a m³ mixture):
enter value

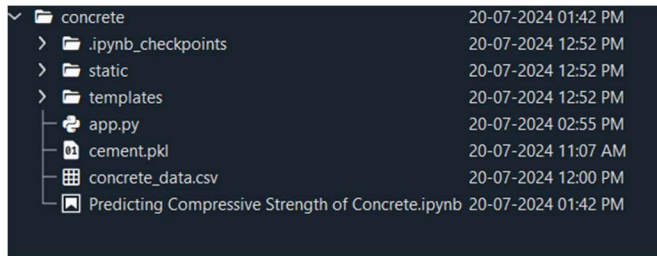
Age (days):
enter value

Predict

Result Page:



Project Folder Structure:

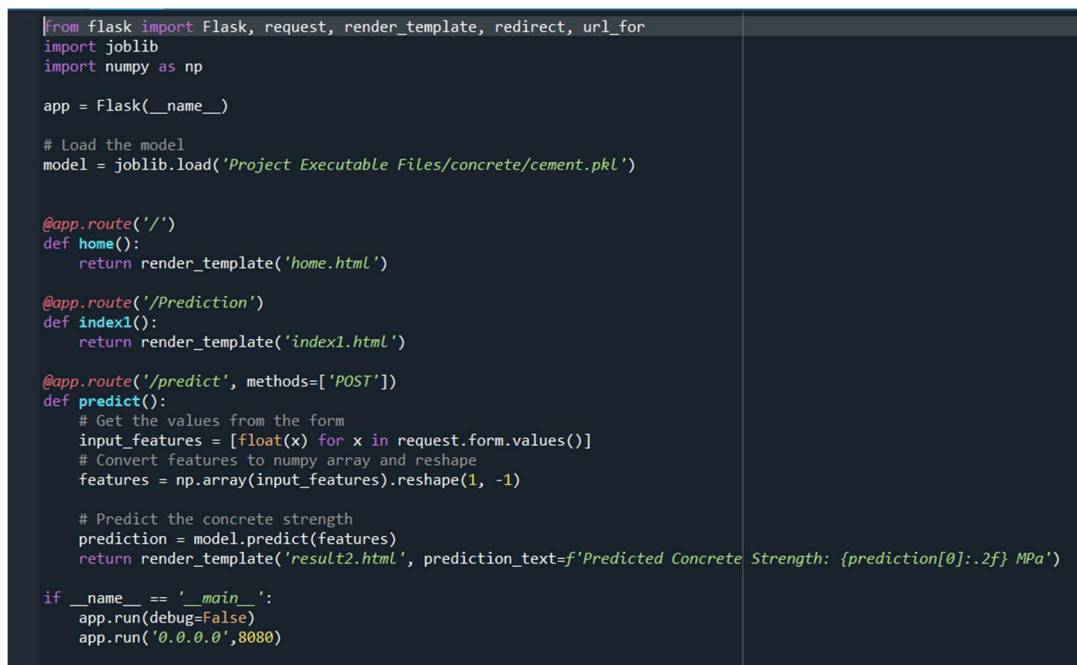


concrete	20-07-2024 01:42 PM
> .ipynb_checkpoints	20-07-2024 12:52 PM
> static	20-07-2024 12:52 PM
> templates	20-07-2024 12:52 PM
app.py	20-07-2024 02:55 PM
cement.pkl	20-07-2024 11:07 AM
concrete_data.csv	20-07-2024 12:00 PM
Predicting Compressive Strength of Concrete.ipynb	20-07-2024 01:42 PM

Static contains background images for html.

Templates contains HTML pages .

App.py file contains the code to load the files and deploy the website. Make predictions from cement.pkl file and give result.



```
from flask import Flask, request, render_template, redirect, url_for
import joblib
import numpy as np

app = Flask(__name__)

# Load the model
model = joblib.load('Project Executable Files/concrete/cement.pkl')

@app.route('/')
def home():
    return render_template('home.html')

@app.route('/Prediction')
def index1():
    return render_template('index1.html')

@app.route('/predict', methods=['POST'])
def predict():
    # Get the values from the form
    input_features = [float(x) for x in request.form.values()]
    # Convert features to numpy array and reshape
    features = np.array(input_features).reshape(1, -1)

    # Predict the concrete strength
    prediction = model.predict(features)
    return render_template('result2.html', prediction_text=f'Predicted Concrete Strength: {prediction[0]:.2f} MPa')

if __name__ == '__main__':
    app.run(debug=False)
    app.run('0.0.0.0', 8080)
```

Running the code generates the server side website address to type in the browser,

Deployed result: Local deployment,


```

PS C:\Users\rakes\Desktop\Predicting Compressive Strength Of Concrete Using Machine Learning> & "C:/Program Files/Python312/python.exe" "c:/Users/rakes/Desktop/Predicting Compressive Strength Of Concrete Using Machine Learning/Project Executable Files/concrete/app.py"
C:\Users\rakes\AppData\Roaming\Python\Python312\site-packages\sklearn\base.py:376: InconsistentVersionWarning: Trying to unpickle estimator DummyRegressor from version 1.4.2 when using version 1.5.1. This might lead to breaking code or invalid results. Use at your own risk. For more info please refer to:
https://scikit-learn.org/stable/model_persistence.html#security-maintainability-limitations
warnings.warn(
C:\Users\rakes\AppData\Roaming\Python\Python312\site-packages\sklearn\base.py:376: InconsistentVersionWarning: Trying to unpickle estimator DecisionTreeRegressor from version 1.4.2 when using version 1.5.1. This might lead to breaking code or invalid results. Use at your own risk. For more info please refer to:
https://scikit-learn.org/stable/model_persistence.html#security-maintainability-limitations
warnings.warn(
C:\Users\rakes\AppData\Roaming\Python\Python312\site-packages\sklearn\base.py:376: InconsistentVersionWarning: Trying to unpickle estimator GradientBoostingRegressor from version 1.4.2 when using version 1.5.1. This might lead to breaking code or invalid results. Use at your own risk. For more info please refer to:
https://scikit-learn.org/stable/model_persistence.html#security-maintainability-limitations
warnings.warn(
* Serving Flask app 'app'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit

```

Use <http://127.0.0.1:5000> ,it directs to the concrete strength prediction website .

5 Advantages :

- **Better Precision:** Machine learning models have the ability to attain a level of precision that far surpasses what traditional empirical formulas can achieve because these models are able to apprehend intricate patterns and interrelationships within the data.
- **Prediction Through Data:** With machine learning models being trained on extensive sets of data, they have the capacity to derive knowledge from countless numbers of actual concrete mixtures and circumstances. This leads to predictions that are stronger in quality.
- **Adaptation:** ML models are flexible and able to be adjusted for different types of data inputs, such as various mix designs or curing conditions. They can even accommodate non-linear relationships.
- **Speed:** After the training phase, ML models can promptly forecast the strength of concrete from given input factors. This has the potential to lessen time and workload needed for testing and analysis efforts.
- **Insight Generation:** Machine learning models have the ability to generate information about what specific factors majorly affect the strength of concrete— thus helping in coming up

with an optimal mix design that enhances quality of the entire structure.

Disadvantages :

- **Data Dependency:** ML models require large amounts of high-quality data for training. Insufficient or biased data can lead to inaccurate predictions.
- **Complexity:** Developing and fine-tuning ML models can be complex and may require expertise in both concrete engineering and machine learning.
- **Interpretability:** Some ML models, especially complex ones like neural networks, are often considered "black boxes," making it challenging to interpret how they arrive at predictions.
- **Longer Training Time:** The model can take a long time to train, particularly with large datasets and when tuning multiple hyperparameters.
- **Model Maintenance:** ML models need periodic updates and retraining as new data becomes available or as concrete materials and practices evolve.

6. Conclusion :

In closing, this work has shown that the machine learning can be used to predict concrete's compressive strength quite accurately. In this project, we successfully developed a machine learning model to predict the compressive strength of concrete based on its mix proportions. The process involved data preprocessing, outlier treatment, model building, and hyperparameter tuning using Gradient Boosting Regressor. By deploying the model using a Flask web application, we created an interactive platform where users can input concrete mix values and get real-time predictions of concrete strength.

6 Future Scope :

Feature Engineering: The scope of Multi-Parameter Optimization can extend beyond predicting compressive strength to also include other critical parameters like flexural strength, durability, and

shrinkage; this approach involves a comprehensive concrete performance prediction.

Real-Time Data Integration: Integrating real-time data collection and prediction into the application could provide immediate insights and recommendations for concrete mixing.

Cross-Industry Applications: The approach used in this project can be adapted to other industries and applications, such as predicting the properties of other construction materials or manufacturing processes.

Industry Standards must not be lost sight of in the process of predictive models—aligning these models with industry standards and practices ensures smooth adoption and use in the already existent construction flow.

By focusing on these areas, we can further enhance the practical utility and robustness of our model, making it more effective and widely applicable in various real-world scenarios.

8 Appendix :

a. Source Code

importing libraries:

```
import pandas as pd
import numpy as np
```

```
import matplotlib.pyplot as plt
import seaborn as sns
import pickle
```

```
from sklearn.metrics import
mean_squared_error,accuracy_score,mean_absolute_error,
confusion_matrix,r2_score
from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.ensemble import GradientBoostingRegressor
import joblib
import warnings
warnings.filterwarnings("ignore")
```

Loading Data

```
data=pd.read_csv('concrete_data.csv')
```

Data overview

```
data.head()
data.tail()
data.columns
```

```
data.shape
```

Null values:

```
data.isnull().sum()
```

Removing outliers:

```
# Define a function to handle outliers using the IQR method
```

```
def handle_outliers(df):
```

```
    for column in df.columns:
```

```
        if df[column].dtype != 'object':
```

```
            Q1 = df[column].quantile(0.25)
```

```
            Q3 = df[column].quantile(0.75)
```

```
            IQR = Q3 - Q1
```

```
            lower_bound = Q1 - 1.5 * IQR
```

```
            upper_bound = Q3 + 1.5 * IQR
```

```
            df[column] = np.where(df[column] < lower_bound, lower_bound,  
df[column])
```

```
            df[column] = np.where(df[column] > upper_bound, upper_bound,  
df[column])
```

```
    return df
```

Univariate analysis:

```
data.describe()
```

```
# Set the aesthetic style of the plots
```

```
sns.set(style="whitegrid")
```

```
# Plotting the distribution of each feature
```

```
fig, axes = plt.subplots(nrows=3, ncols=3, figsize=(15, 15))
```

```
for i, column in enumerate(data.columns):
```

```
    sns.histplot(data[column], ax=axes[i//3, i%3], kde=True)
```

```
    axes[i//3, i%3].set_title(f'Distribution of {column}')
```

```
plt.tight_layout()
```

```
plt.show()
```

Bivariate Analysis:

```
sns.pairplot(data)
```

```
plt.show()
```

Multivariate analysis:

```
data.corr()
```

```
# Set up the matplotlib figure
```

```
plt.figure(figsize=(10, 8))
```

```
# Create a heatmap using seaborn
```

```
sns.heatmap(data.corr(), annot=True, cmap='BrBG', fmt=".2f",
linewidths=0.5)
plt.show()
```

splitting data:

```
x=pd.DataFrame(data,columns=data.columns[:8])
y=pd.DataFrame(data,columns=data.columns[8:])
```

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,
random_state=1)
```

Model Building

Before Hyperparameter Tuning:

```
gb = GradientBoostingRegressor()
gb.fit(x_train,y_train)
y_pre=gb.predict(x_test)
print(y_pre)
print("R2 score :",r2_score(y_test,y_pre))
```

After Hyperparameter Tuning:

```
gbr = GradientBoostingRegressor(random_state=42)
param_grid = {
    'n_estimators': [100, 200, 300],
    'learning_rate': [0.01, 0.05, 0.1],
    'max_depth': [3, 4, 5],
    'subsample': [0.8, 0.9, 1.0],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
```

Initialize Grid Search with Cross-Validation

```
grid_search = GridSearchCV(estimator=gbr, param_grid=param_grid, cv=5,
n_jobs=-1, verbose=2, scoring='neg_mean_squared_error')
```

Fit Grid Search to the data

```

grid_search.fit(x_train, y_train)

# Get the best parameters from the Grid Search
best_params = grid_search.best_params_
print(f"Best Parameters: {best_params}")

# Train the model using the best parameters
best_model = GradientBoostingRegressor(**best_params, random_state=42)
best_model.fit(x_train, y_train)

# Make predictions on the testing set
y_pred = best_model.predict(x_test)

Model Evaluation:

print("Mean Absolute Error:", mean_absolute_error(y_test,y_pred))
print("Mean Squared Error:", mean_squared_error(y_test,y_pred))
print("Root Mean Square Error:", np.sqrt(mean_absolute_error(y_test,y_pred)))
print("R2 score :",r2_score(y_test,y_pred))

Save model:

pickle.dump(best_model,open('cement.pkl','wb'))

```

b. GitHub & Project Demo Link

<https://github.com/JayanthSrinivas06/Predicting-Compressive-Strength-Of-Concrete-Using-Machine-Learning.git>

Demo link:

[Demonstration Link](#)