-MACHINE LEARNING REGRESSION PROJECT-

CONCRETE COMPRESSIVE STRENGTH

TEAM-2

G. MADHU KIRAN

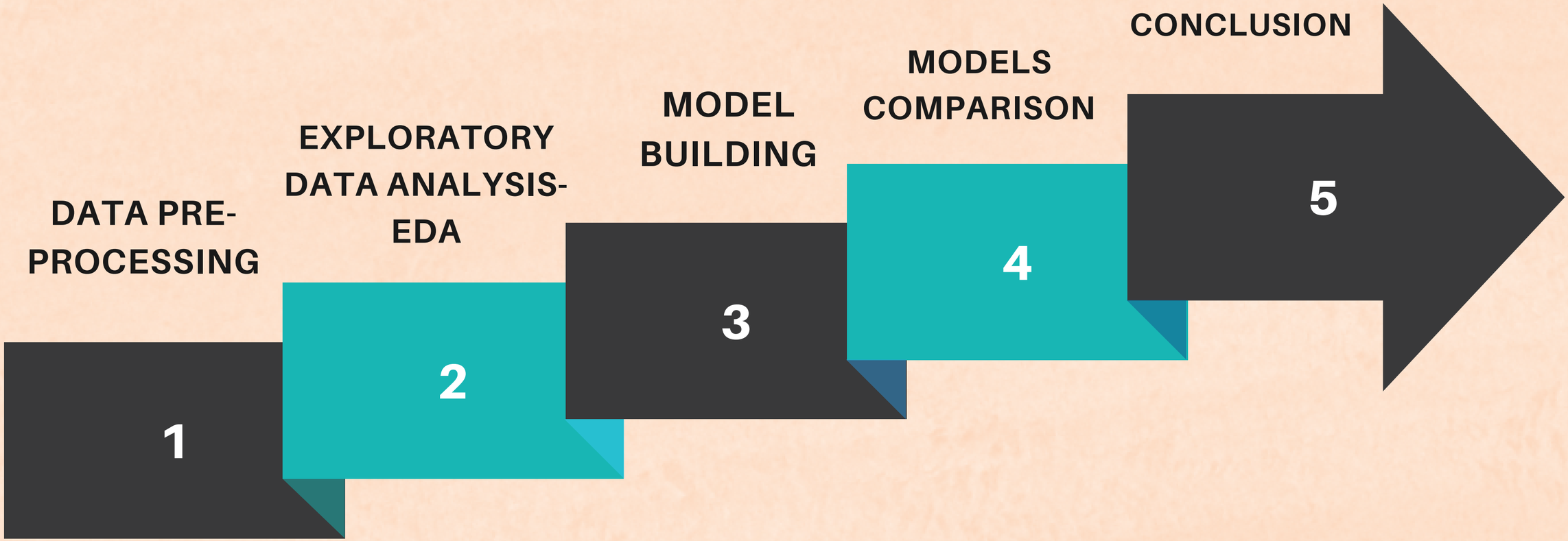
CHIRAMJEEVI

P. SHRAVANI

SATHVIKA REDDY

Bhavan's Vivekananda College | BSC Hons Data Science

MACHINE LEARNING PROJECT STAGES



ABOUT THE DATA:

The dataset is about concrete strength and the variables which effect the **"TARGET_concretestrength"**.

OBJECTIVE:

Concrete is the most used material for construction in the world! There are some components that should be combined to make the concrete. These components can affect the compressive strength of the concrete. We have to come up with ML solution to predict the strenght of the concrete which helps reducing risks during construction.

THE PATH:

Implementing multiple regression models to fit the best model for the dataset.

DATA AND DATA QUALITY CHECK:

INTRODUCTION:

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

VARIABLES:

Variable Name	Variable Description
TARGET_concretestrength	a critical measure of its ability to withstand applied forces and loads.
Cement	The amount of cement in the concrete mixture
BlastFurnaceSlag	a byproduct of iron , added to the concrete mix to enhance certain properties
Water	The amount of water used in the concrete mix
CoarseAggregate	gravel or crushed stone, providing strength and stability
FineAggregate	typically sand, contributing to the overall strength of the concrete mixture.
Age	indicating the duration since the initial mixing

DATA PRE-PROCESSING:

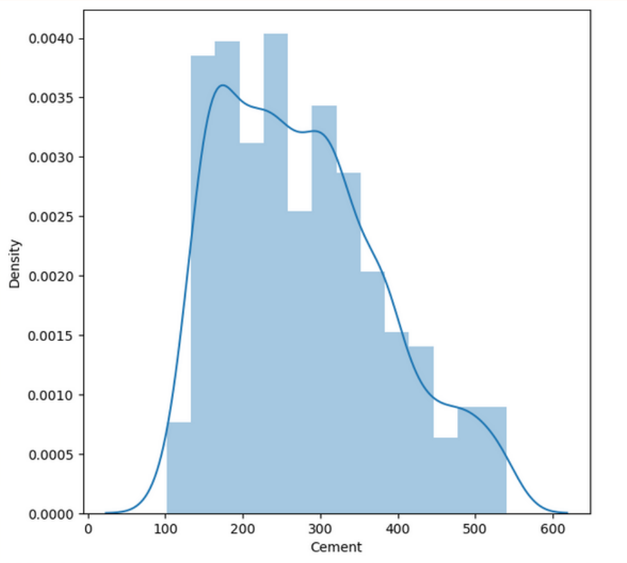
1. There are 1030 rows and 9 columns.

2. No null values.

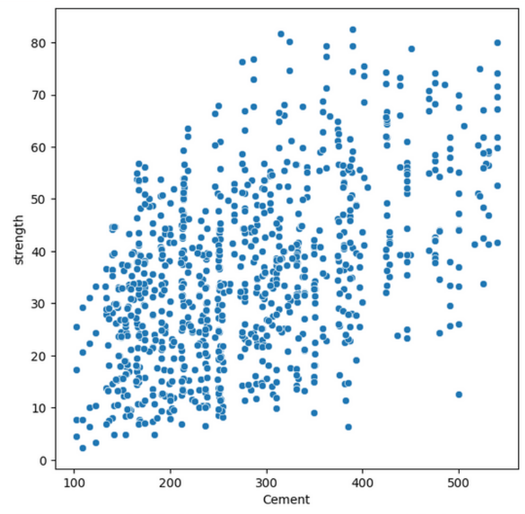
3. Number of duplicate values = 25

4. After overcoming the duplicate values, the resultant is obtained to 1005 rows and 9 columns.

EXPLORATORY DATA ANALYSIS:

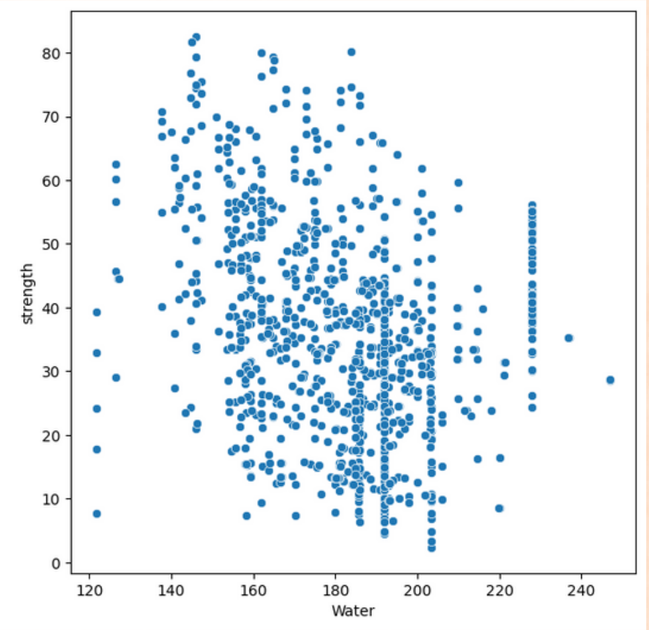


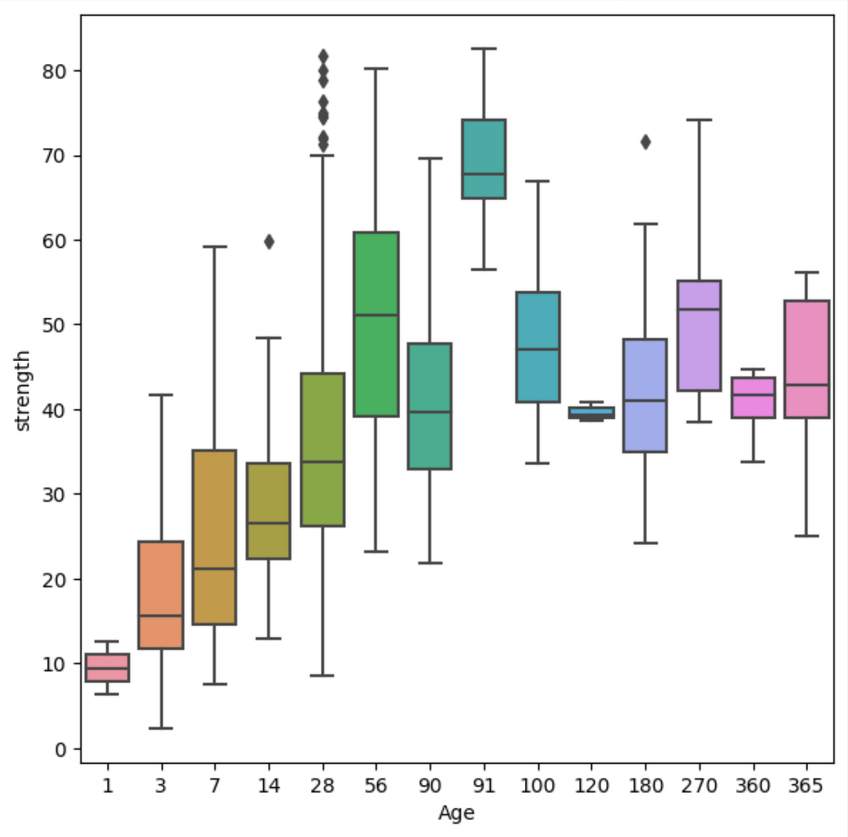
The graph follows normal distribution. It is equally skewed



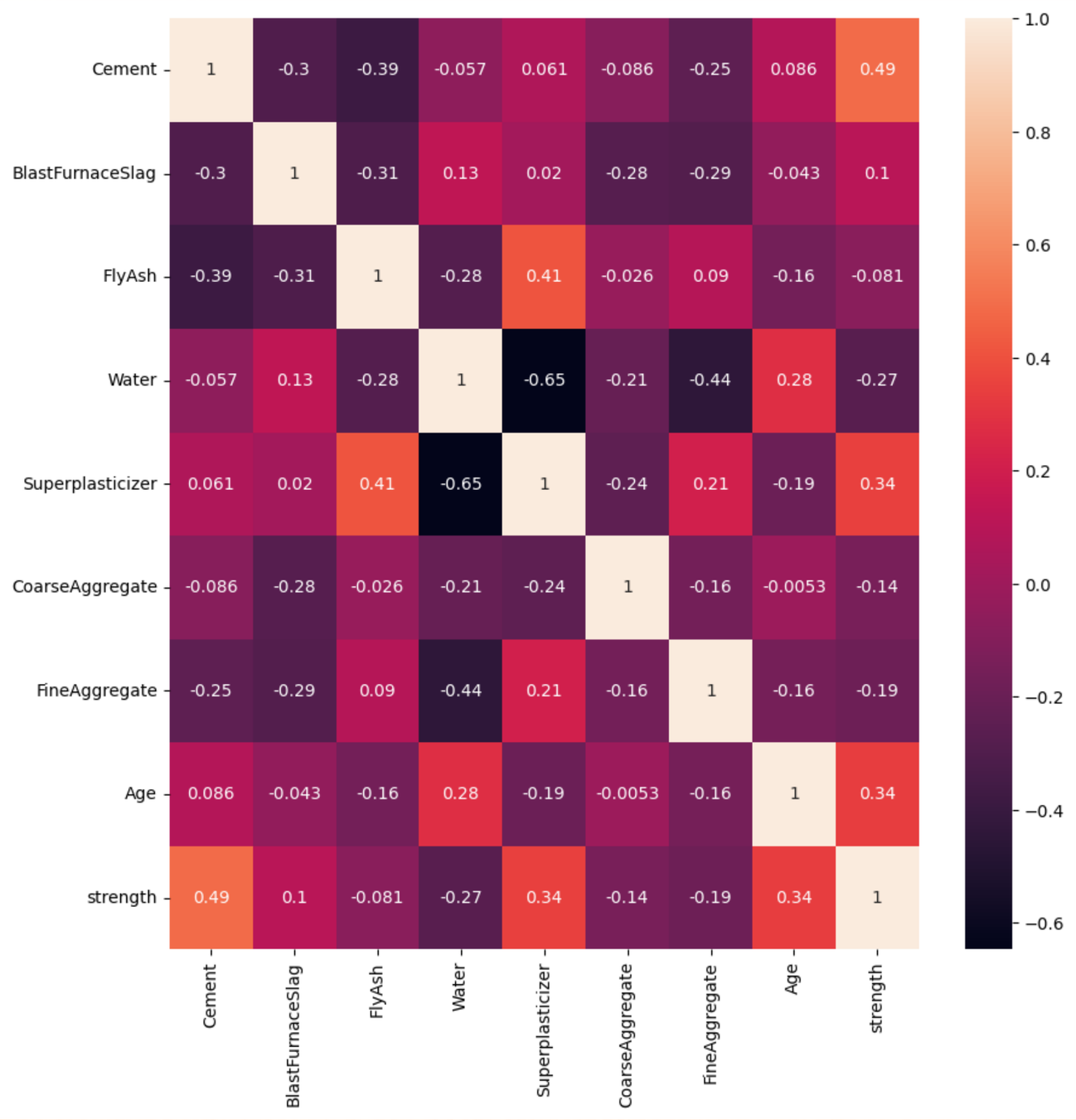
The scatter plot shows correlation between cement and TARGET_strength

The scatter plot shows correlation between water and TARGET_strength. A negative correlation is formed.





This is a Box Plot to demonstrate the outliers in the dataset.



Heatmap represents the correlation between different variables. There is no strong positive correlation between any two variables

VARIANCE INFLATION FACTOR MULTICOLINEARITY ANALYSIS

VARIABLES	VIF
Cement	15.1
BlastFurnaceSlag	3.2
FlyAsh	4.1
Water	81.3
Superplasticizer	5.1
CoarseAggregate	84.7
FineAggregate	72.4
Age	1.6

MODEL BUILDING:

1. LINEAR REGRESSION

TRAIN_TEST RATIO	R sq value	MAE
65-35	0.5705	8.91
70-30	0.5608	8.98
75-25	0.5644	8.93
80-20	**0.5801**	**8.89**

2. RANDOM FOREST

TRAIN_TEST RATIO	R sq value	MAE
65-35	0.8979	3.82
70-30	0.8963	3.81
75-25	0.8973	3.74
80-20	**0.9111**	**3.46**

3. K NEAREST REGRESSOR

TRAIN_TEST RATIO	R sq value	MAE
65-35	0.6274	7.84
70-30	0.6391	7.68
75-25	0.6369	7.69
80-20	**0.6996**	**7.23**

4. SUPPORT VECTOR REGRESSOR

TRAIN_TEST RATIO	R sq value	MAE
65-35	0.1980	12.25
70-30	**0.2027**	**12.12**
75-25	0.1986	12.18
80-20	0.2021	12.31

5. DECISION TREE REGRESSOR

TRAIN_TEST RATIO	R sq value	MAE
65-35	0.8129	4.66
70-30	**0.8482**	**4.43**
75-25	0.8513	4.17
80-20	0.8656	3.79

6. GRADIENT BOOSTING

TRAIN_TEST	learning rate	n_estimators	MAE
65-35	0.1	100	4.07
65-35	0.1	200	3.567
70-30	0.1	100	4.15
70-30	0.1	200	3.71
80-20	0.1	100	4.09
80-20	**0.1**	**200**	**3.53**

7. ADA BOOST

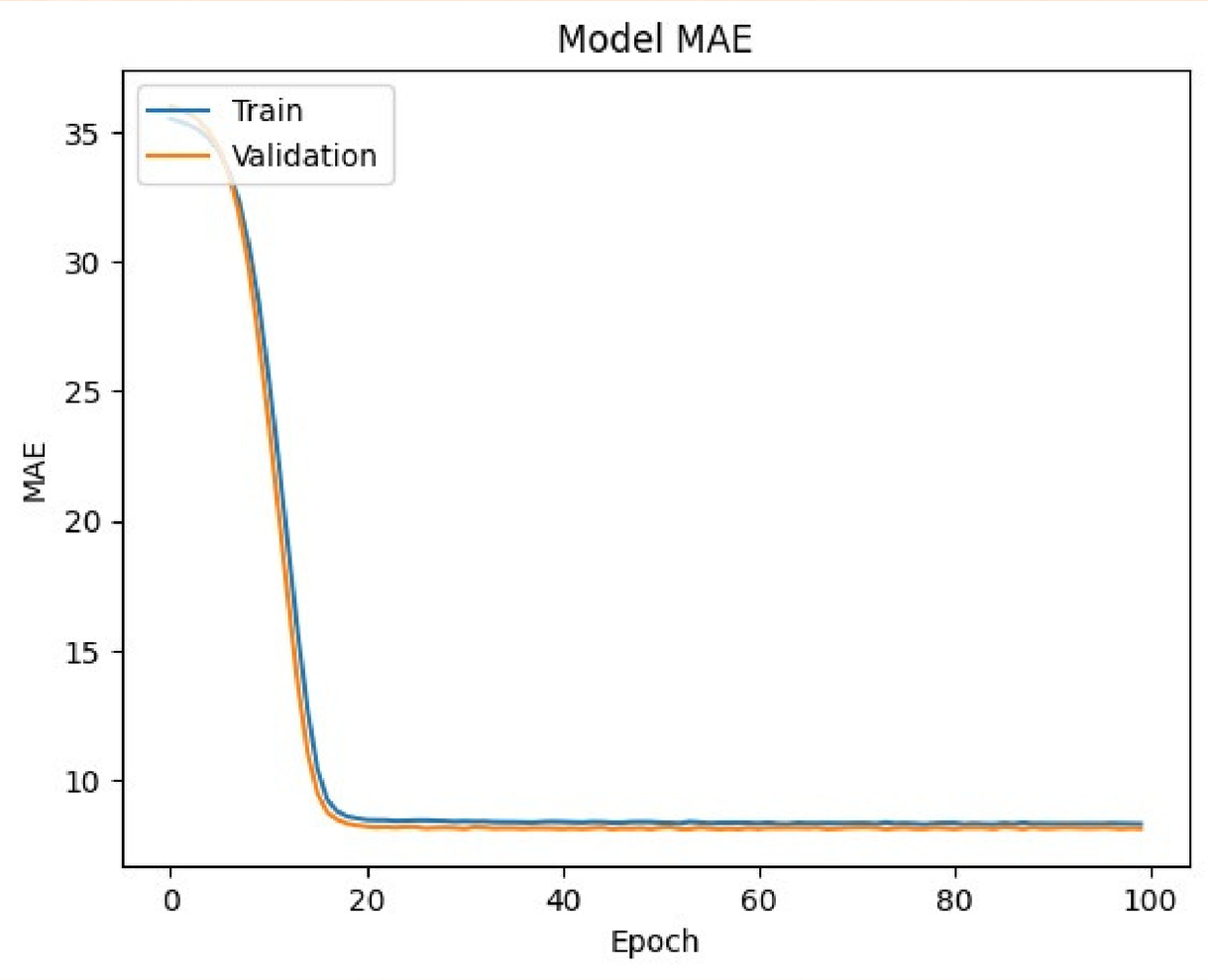
TRAIN_TEST	learning rate	n_estimators	MAE
65-35	0.1	100	7.08
65-35	0.1	200	6.89
70-30	0.1	100	6.86
70-30	0.1	200	6.83
80-20	0.1	100	6.80
80-20	**0.1**	**200**	**6.68**

8. XG BOOST

TRAIN_TEST	learning rate	n_estimators	MAE
65-35	0.1	100	3.33
65-35	0.1	200	3.19
70-30	0.1	100	3.38
70-30	0.1	200	3.14
80-20	0.1	100	2.93
80-20	**0.1**	**200**	**2.69**

9. NEURAL NETWORK

TRAIN_TEST	Architecture	Optimizer	Epochs	MAE
65-35	64-64-1	RMSprop	100	8.52
65-35	27-20-4	Adam	150	8.27
70-30	64-64-1	RMSprop	100	8.42
70-30	**27-20-4**	**Adam**	**150**	**7.89**
80-20	64-64-1	RMSprop	100	8.40
80-20	27-20-1	Adam	150	8.42



Train_Test	**70-30**
Architecture	27-20-1
Optimizer	Adam
mae	7.89
Epochs	150

COMPARISON OF MODELS:

MODEL	MAE(Test Size 80-20)
Linear Rregression	8.89
Random forest	3.46
K Nearest Regressor	7.23
Support Vector Regressor	12.31
Decision Tree	3.79
Gradient Boosting	3.53
Ada Boost	6.68
XG Boost	**2.69**
Neural Network	7.89

CONCLUSION:

1. For this concrete dataset, we performed 8 machine learning algorithms: Linear Regression, Random Forest, KNN, SVR, Decision Tree, Gradient Boosting, Ada Boost, XG Boost.

2. From all the analysis and implementations of the algorithm, we found the best results in XG Boost algorithm i.e. in the Extreme Gradient Boosting algorithm for 75-25 train-test ratio with Mean Absolute Error (MAE) of 2.89 which is the least error value among all the other errors.

3. So, here we can conclude that Extreme Gradient Boosting Algorithm can be considered the best model for the given concrete dataset.

THANK YOU

PRESENTED BY
G. MADHU KIRAN
CHIRAMJEEVI
P. SHRAVANI
SATHVIKA REDDY

APPENDIX

TRAINING AND TESTING :

```
#Split the dataset into features (X) and target variable (y)
X = df.drop('strength', axis=1)
y = df['strength']

#Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

LINEAR REGRESSION :

```
#Creating a linear regression model
model = LinearRegression()
```

```
#Fitting the model to the training data
model.fit(X_train, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
#Making predictions on the test data
y_pred = model.predict(X_test)
```

```
Mean Squared Error: 125.265338276725
Mean Absolute Error: 8.896027820458997
Root Mean Squared Error: 11.19219988548833
R-squared (R²): 0.580108901995383
```

RANDOM FOREST :

```
# Create a random forest regression model
model = RandomForestRegressor()
```

+ Code

```
# Fit the model to the training data
model.fit(X_train, y_train)
```

```
RandomForestRegressor
RandomForestRegressor()
```

```
# Make predictions on the test data
y_pred = model.predict(X_test)
```

```
Mean Squared Error: 25.261948569531203
Mean Absolute Error: 3.4175690807694963
Root Mean Squared Error: 5.026126597045801
R-squared (R²): 0.9153216087664731
```

K NEAREST NEIGHBOR :

```
model = KNeighborsRegressor(n_neighbors=5)
```

```
model.fit(X_train, y_train)
```

```
▾ KNeighborsRegressor
KNeighborsRegressor()
```

```
y_pred = model.predict(X_test)
```

```
Mean Squared Error: 89.59711037466813
Mean Absolute Error: 7.233026334203217
Root Mean Squared Error: 9.465575015532238
R-squared (R²): 0.6996692814563655
```

SUPPORT VECTOR REGRESSOR:

```
model = SVR()

model.fit(X_train, y_train)
```

▾ SVR
SVR()

```
y_pred = model.predict(X_test)
```

```
Mean Squared Error: 238.03118187441322
Mean Absolute Error: 12.311563862046931
Root Mean Squared Error: 15.428259197797178
R-squared (R²): 0.20211627820147926
```

GRADIENT BOOSTING :

```
# Split the data into training and testing sets
from sklearn.ensemble import GradientBoostingRegressor
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
X_train.shape, X_test.shape
```

```
((703, 8), (302, 8))
```

```
# Create a Gradient Boosting Regressor model with desired hyperparameters
gbr = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1, max_depth=3, random_state=0)
gbr.fit(X_train, y_train)
```

```
GradientBoostingRegressor
GradientBoostingRegressor(random_state=0)
```

```
y_pred = gbr.predict(X_test)
y_pred
```

```
Mean Absolute Error: 3.696727039233802
Mean Squared Error: 25.60731969774223
Root Mean Squared Error: 5.060367545716638
R-squared (R²): 0.9089598228410433
```

DECISION TREE :

```
from sklearn.tree import DecisionTreeRegressor
```

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.35)
X_train.shape, X_test.shape
```

```
((653, 8), (352, 8))
```

```
# Create a Decision Tree Regressor model
regressor = DecisionTreeRegressor()
regressor.fit(X_train, y_train)
```

```
▾ DecisionTreeRegressor
DecisionTreeRegressor()
```

```
# Make predictions on the test set
y_pred = regressor.predict(X_test)
y_pred
```

```
Mean Absolute Error: 5.516685514869916
Mean Squared Error: 69.28056497946498
Root Mean Squared Error: 8.323494757580194
R-squared (R²): 0.7530972683291066
```

ADA BOOST :

```
from sklearn.ensemble import AdaBoostRegressor
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

adaboost_regressor = AdaBoostRegressor(n_estimators=50, random_state=42)

# Train the AdaBoost regressor
adaboost_regressor.fit(X_train, y_train)
```

```
Mean Absolute Error: 6.374131390683759
Mean Squared Error: 62.14896629272236
Root Mean Squared Error: 7.883461567910531
R-squared (R²): 0.7916758294393089
```

XG BOOST :

```
from xgboost import XGBRegressor
# Assuming you have a dataset (X, y), replace this with your actual dataset
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the XGBoost regressor
xgb_regressor = XGBRegressor(n_estimators=100, learning_rate=0.1, random_state=42)

# Train the XGBoost regressor
xgb_regressor.fit(X_train, y_train)
```

```
Mean Absolute Error: 2.9327385864113493
Mean Squared Error: 20.758514502114636
Root Mean Squared Error: 4.556151281741491
R-squared (R²): 0.9304171803058371
```

NEURAL NETWORK :

```
def build_model():
    model = models.Sequential([
        layers.Dense(64, activation='linear', input_shape=(X_train.shape[1],)),
        layers.Dense(64, activation='linear'),
        layers.Dense(1)  # No activation for regression
    ])

    model.compile(optimizer='rmsprop', loss='mse', metrics=['mae'])
    return model

model = build_model()
```

```
history = model.fit(X_train, y_train, epochs=150, validation_split=0.2, verbose=2)
```

VARIANCE INFLATION FACTOR :

```
# Import library for VIF
from statsmodels.stats.outliers_influence import variance_inflation_factor

def calc_vif(X):

    # Calculating VIF
    vif = pd.DataFrame()
    vif["variables"] = X.columns
    vif["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]

    return(vif)
calc_vif(X)
```

	variables	VIF
0	Cement	15.155077
1	BlastFurnaceSlag	3.261879
2	FlyAsh	4.171453
3	Water	81.395278
4	Superplasticizer	5.171809
5	CoarseAggregate	84.738626
6	FineAggregate	72.495779
7	Age	1.696228