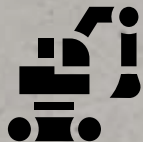




Concrete Cement Strength ×



Linear Regression

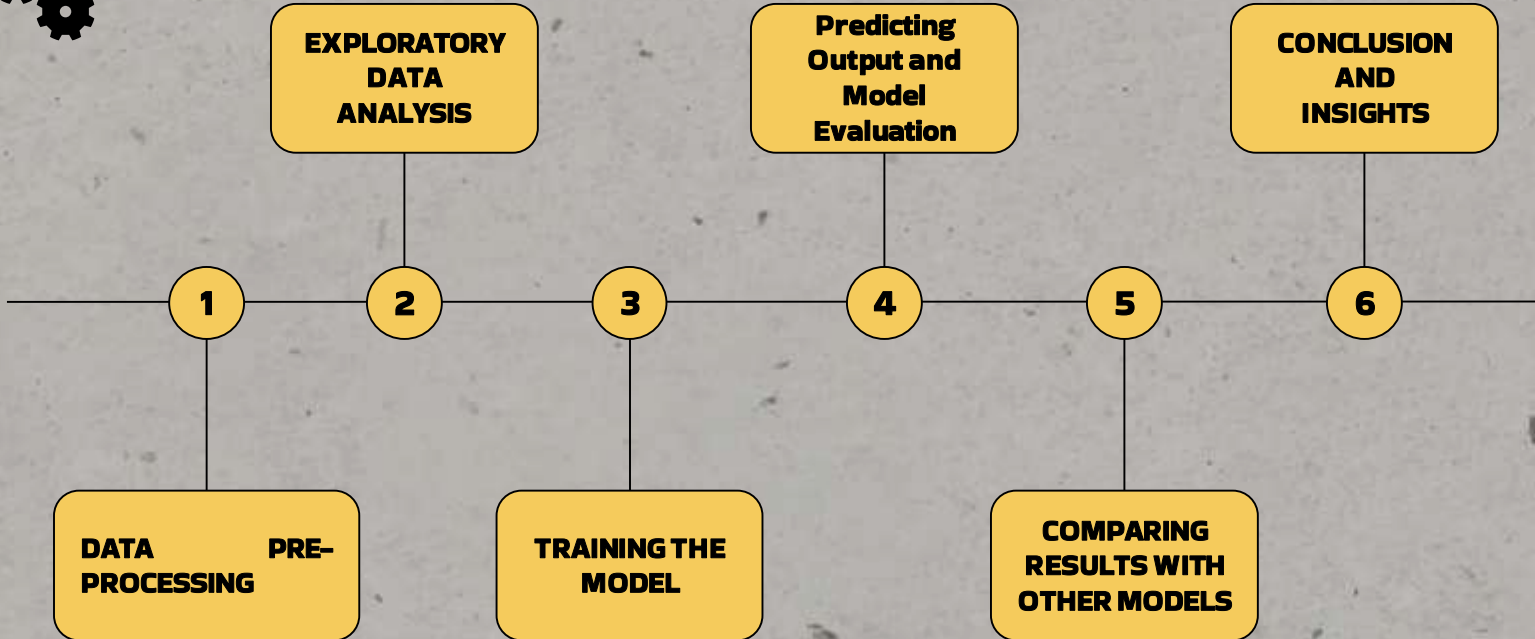


Group - 4

**Ch. Vinay Kumar
R. Srushtitha**



Machine Learning Analysis Timeline





»» About The Data:

The project will utilize a dataset comprising historical concrete strength test results, concrete mix compositions, and curing conditions. This dataset will be sourced from construction projects and testing laboratories, enabling the development of a robust predictive model for concrete strength

»» Objective:

To develop an accurate predictive model for concrete strength based on historical data and material characteristics. The goal is to enhance quality control in construction projects, optimize concrete mixtures, and improve structural performance through data-driven insights and advanced modeling techniques.

»» The Path:

Implementing multiple machine learning models to fit the best model for the dataset.



Data And Data Quality Check

»» Data Introduction:

The data consists of 9 columns and 1030 observations taken from UC Irvine Machine Learning Repository.

»» Features:

Concrete_compressive	Dependent variable, Concrete compressive strength is measured in megapascals (MPa)
Cement	powdered binder for construction and concrete.
Superplasticizer	enhances concrete workability with less water.
Blast_Furnace_Slag	Blast furnace slag is a byproduct used to improve concrete's strength and durability.
coarse_Aggregate	Coarse aggregate in concrete: larger particles for strength.



»» Missing Values:


The data is complete, with no missing values.

»» Duplicate Values:

I have dropped the duplicates, and there are now 25 duplicate values removed out of 1030 observations.

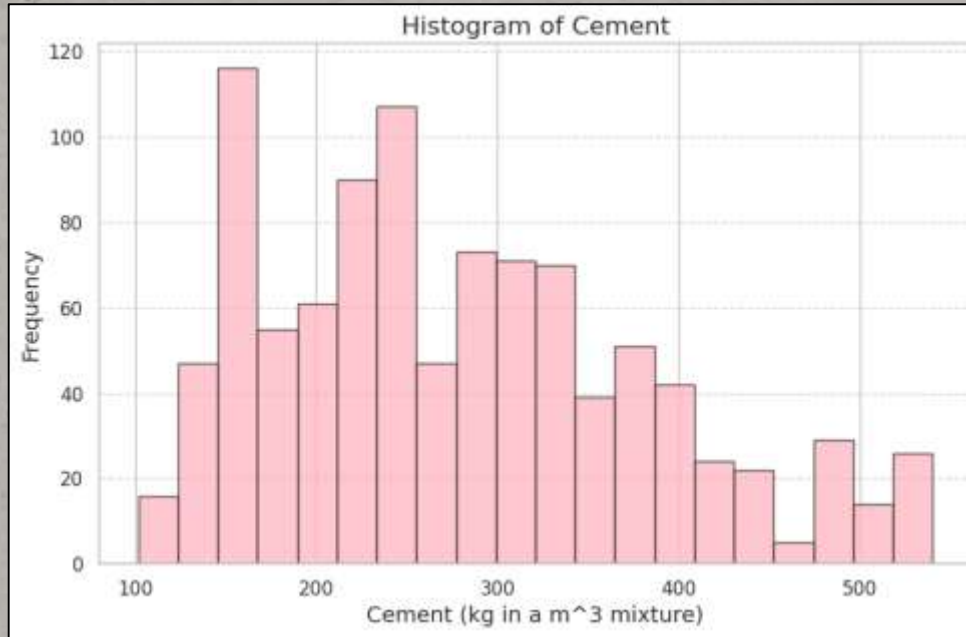
»» Dropped Columns:

There are no dropped columns from dataset, as each column have their respective importance for predicting the “Dependent Variable”.



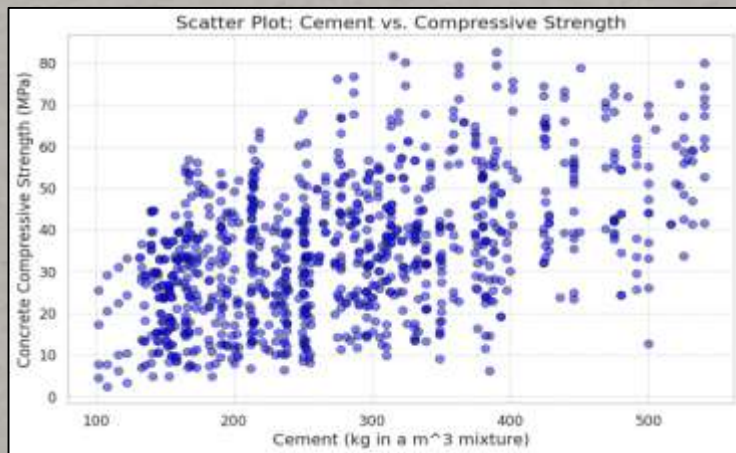


Data Visualization:



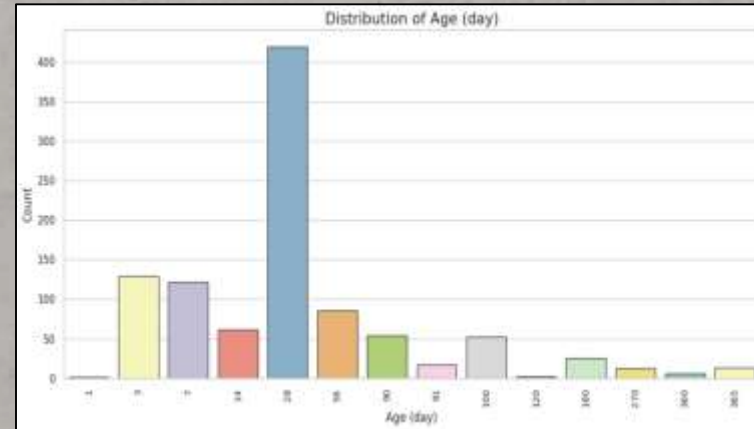
The histogram reveals that the highest frequency occurs within the interval of **140-160**, indicating that this range of cement usage is the most prevalent in the dataset.





The scatter plot reveals a straightforward positive relationship between the variables, indicating that as one variable increases, the other tends to increase as well.

The "Age vs Count" bar graph illustrates the distribution of ages in the dataset, emphasizing the prevalence of specific age groups.



Heat Map:



- **Cement, Superplasticizer, Age are moderately positively correlated with the target variable i.e Concrete Compressive.**
- **Blast_Furnance_Slay has almost zero correlation with the target variable i.e Concrete Compressive.**
- **Fly_Ash, Water, Coarse_Aggregate, Fine_Aggregate are moderately negatively Correlated with the target variable i.e Concrete Compressive.**

»» Algorithms:

»» Linear Regression:

Linear regression is a statistical method used to model the relationship between a dependent variable and one or more independent variables. It assumes a linear relationship and aims to find the best-fitting linear equation to predict the dependent variable. This technique is widely employed in various fields for tasks like prediction and trend analysis.

- »» **Feature Selection :** I have identified and selected all the features as all of them have a strong influence on the dependent variable.
- »» **Model Building :** Utilized linear regression to fit a linear equation between the selected independent variables and the dependent variable
- »» **Evaluation :** Assessed the model's performance using metrics like R-squared, MAE, and RMSE to understand its predictive capability and potential for further refinement

Train Test Ratio	R-square Value	MAE(Mean Absolute Error)
70 : 30	0.57	8.10
80 : 20	0.61	8.22
75 : 25	0.57	7.99
60 : 40	0.62	7.79

➤➤ In summary, the **60 : 40** split consistently showed the highest R-squared score **0.62** and the lowest mean absolute error **7.79** , making it the ideal choice for our model. This proportion strikes a balance between model fit and predictive accuracy, ensuring reliable performance for future deployment.

»» **Decision Tree Regression:**

Decision Tree regression is a supervised machine learning algorithm used for regression tasks. It models relationships between independent and dependent variables by recursively splitting the data into segments to create a tree-like structure. Each leaf node represents a predicted value, making it suitable for both simple and complex regression problems with interpretable results.

- »» **Feature Selection :** Identified significant features to determine how they influence the target variable.
- »» **Model Training :** Trained a Decision Tree regression model on the data to create a tree structure based on feature splits.
- »» **Hyperparameter Tuning:** Optimized the tree's depth, minimum samples per leaf, and other hyperparameters to prevent overfitting and improve model performance
- »» **Evaluation :** Assessed the model's accuracy using metrics like R-squared, MAE, and RMSE and visualized the tree structure for interpretability.

Train Test Ratio	R-square Value	MAE(Mean Absolute Error)
70 : 30	0.76	5.40
80 : 20	0.84	4.83
75 : 25	0.79	5.35
60 : 40	0.79	5.40

➤➤ In summary, the **80 : 20** split consistently showed the highest R-squared score **0.84** and the lowest mean absolute error **4.83**, making it the ideal choice for our model. This proportion strikes a balance between model fit and predictive accuracy, ensuring reliable performance for future deployment.

»» **Random Forest Regression:**

Random Forest regression is an ensemble machine learning technique used for regression tasks. It constructs multiple decision trees during training and combines their predictions to reduce overfitting and improve accuracy. By aggregating predictions from multiple trees, it provides robust and accurate predictions for complex regression problems.

»» **Hyperparameter Tuning :** Optimized hyperparameters such as the number of trees and maximum depth to enhance model performance.

»» **Evaluation :** Evaluated the model using regression metrics like Mean Absolute Error (MAE) and R-squared, and assessed feature importance to gain insights into the predictive factors.

Train Test Ratio	n_estimators	R-square Value	MAE(Mean Absolute Error)
70 : 30	200	0.897	3.61
80 : 20	200	0.901	3.63
75 : 25	200	0.907	3.39
60 : 40	200	0.873	3.93

➤➤ In summary, the **75 : 25** split consistently showed the highest R-squared score **0.907** and the lowest mean absolute error **3.39** , making it the ideal choice for our model. This proportion strikes a balance between model fit and predictive accuracy, ensuring reliable performance for future deployment.

»» **K – Nearest Neighbor Regression:**

K-Nearest Neighbors (KNN) regression is a non-parametric machine learning algorithm used for regression tasks. It predicts the target variable by averaging the values of its k-nearest neighboring data points. KNN regression is simple to understand and implement but can be sensitive to the choice of the k parameter, requiring careful tuning for optimal performance.

- »» Selecting K : Determined the optimal value for the K parameter by experimenting with different values and evaluating model performance.**
- »» Model Training : Trained the KNN regression model by storing the training dataset and using it to make predictions based on the k-nearest neighbors of each data point.**
- »» Evaluation : Assessed the model's performance using regression metrics such as Mean Absolute Error (MAE) or R-squared, and considered the implications of the chosen K value on the model's accuracy and computational efficiency**

Train Test Ratio	n_neighbors	R-square Value	MAE(Mean Absolute Error)
70 : 30	11	0.62	7,40
80 : 20	11	0.65	7.69
75 : 25	11	0.62	7.36
60 : 40	11	0.64	7.53

» In summary, the **80 : 20** split consistently showed the highest R-squared score **0.65** and the lowest mean absolute error **7.69** , making it the ideal choice for our model. This proportion strikes a balance between model fit and predictive accuracy, ensuring reliable performance for future deployment.

»» **Support Vector Machine:**

Support Vector Machine (SVM) regression is a machine learning technique used for regression tasks. It aims to find the optimal hyperplane that maximizes the margin between data points while minimizing prediction errors. SVM regression is effective in handling both linear and non-linear relationships between features and the target variable.

- »» **Kernel Selection :** Choose an appropriate kernel function (e.g., linear, polynomial, or radial basis function) to transform data and capture complex relationships.
- »» **Hyperparameter Tuning :** Optimized hyperparameters such as the regularization parameter (C) and kernel-specific parameters to enhance model accuracy.
- »» **Model Training :** Trained the SVM regression model to find the hyperplane that best fits the data while minimizing prediction errors.
- »» **Evaluation :** Assessed model performance using regression metrics like Mean Absolute Error (MAE) or R-squared, and fine-tuned the model as necessary to achieve optimal predictive results.

Train Test Ratio	Kernel	R-square Value	MAE(Mean Absolute Error)
70 : 30	Poly	0.72	6.40
80 : 20	Poly	0.76	6.47
75 : 25	Poly	0.70	6.45
60 : 40	Poly	0.73	6.31

»» In summary, the **80 : 20** split consistently showed the highest R-squared score **0.76** and the lowest mean absolute error **6.47** , making it the ideal choice for our model. This proportion strikes a balance between model fit and predictive accuracy, ensuring reliable performance for future deployment.


Algorithm	R-square Value	MAE(Mean Absolute Error)
Linear Regression	0.62	7.79
Decision Tree	0.84	4.83
Random Forest	0.90	3.39
KNN	0.65	7.69
SVM	0.76	6.47

Summary And Recommendations

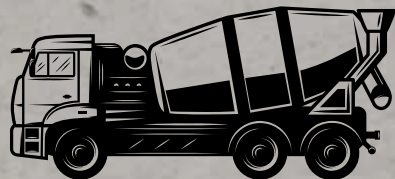
- For the concrete dataset, we performed five machine learning algorithms : Linear Regression, Bagging, K-nn, SVM.
- The visualization of data performed in exploratory data analysis showed clearly which of the features are more and less correlated with the target variable i.e Concrete Compressive .
- From all the analysis and implementations of the algorithm, we found the best results in Bagging algorithm i.e. in the Random Forest Bagging algorithm for **75 - 25** train-test ratio with a R-squared value of **0.907** and Mean Absolute Error (MAE) of **3.39** which is the least error value among all the other errors.
- So, here we can conclude that Random Forest Bagging Algorithm can be considered as the best model for the given concrete dataset.



Future insights that contribute to increase the concrete compressive strength include the following

- **The features like Cement, Superplasticizer and Age are interconnected with one another, as we have seen in the data visualization, and this has a significant impact on the target Concrete Compressive .**
 - **Therefore in order to improve the Strength of Concrete Compressive, it is necessary to raise the variable Cement, Superplasticizer and Age. Additionally we need to decrease the variables Fly_Ash, Water, Coarse_Aggregate, Fine_Aggregate will also help us to increase the strength of Concrete Compressive.**
 - **Mostly we need to manage the following features values i.e Fly_Ash, Water, Coarse_Aggregate, Fine_Aggregate in order to increase the strength of Concrete Compressive. And raising the remaining features values.**
- 

THANKS!



Presented by:

- **Ch Vinay Kumar**
- **R. Srushtitha**



Appendix x

Training And Testing:



```
1 x = data.drop("Concrete_compressive",axis=1)
2 y = data["Concrete_compressive"]
```

```
[ ] 1 x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.3, random_state =37)
```



```
1 print(x_train.shape)
2 print(x_test.shape)
3 print(y_train.shape)
4 print(y_test.shape)
```



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


Linear Regression:

```
[22] 1 regmodel = linear_model.LinearRegression()  
      2 regmodel.fit(x_train,y_train)
```

▼ LinearRegression
LinearRegression()

```
1 mse3 = mean_squared_error(y_test3,predictions3)  
2 mae3 = mean_absolute_error(y_test3,predictions3)  
3 rmse3 = math.sqrt(mean_squared_error(y_test3,predictions3))  
4 print("Mean Squared Error: {MSE3}",format(mse3,".2f"))  
5 print("Mean Absolute Error (MAE3):",format(mae3,".2f"))  
6 print("Root Mean Square Error (RMSE): {rmse3}",format(rmse3,".2f"))
```

➡ Mean Squared Error: {MSE3} 95.73
Mean Absolute Error (MAE3): 7.79
Root Mean Square Error (RMSE): {rmse3} 9.78

```
1 r2 = r2_score(y_test3,predictions3)  
2 print("R-squared:",format(r2,'.2f'))
```

R-squared: 0.62

Decision Tree Regression:

```
[45] 1 regressor = DecisionTreeRegressor(criterion = "squared_error", max_depth = 10, min_samples_split = 10, random_state = 90)
      2 ## we can use "friedman_mse", "squared_error", "poisson", "absolute_error" as values of criterion
```

```
1 regressor.fit(x_train, y_train)
```

```
DecisionTreeRegressor
DecisionTreeRegressor(max_depth=10, min_samples_split=10, random_state=90)
```

```
[55] 1 mse1 = mean_squared_error(y_test1,y_predict1)
      2 mae1 = mean_absolute_error(y_test1,y_predict1)
      3 rmse1 = math.sqrt(mean_squared_error(y_test1,y_predict1))
      4 print("Mean Squared Error: {MSE}",format(mse1,".2f"))
      5 print("Mean Absolute Error (MAE):",format(mae1,".2f"))
      6 print("Root Mean Square Error (RMSE): {rmse}",format(rmse1,".2f"))
```

```
Mean Squared Error: {MSE} 45.86
Mean Absolute Error (MAE): 4.83
Root Mean Square Error (RMSE): {rmse} 6.77
```

```
1 r2 = r2_score(y_test1,y_predict1)
2 print("R-squared:",format(r2,'.2f'))
```

```
R-squared: 0.84
```

Random Forest Regression:

```
[70] 1 rf_regressor = RandomForestRegressor(n_estimators = 200, random_state = 50) ##even for classifier we can use n_estimators.  
2 rf_regressor.fit(x_train, y_train)
```

RandomForestRegressor
RandomForestRegressor(n_estimators=200, random_state=50)

```
[71] 1 y_predict = rf_regressor.predict(x_test)
```

```
1 mse2 = mean_squared_error(y_test2,y_predict2)  
2 mae2 = mean_absolute_error(y_test2,y_predict2)  
3 rmse2 = math.sqrt(mean_squared_error(y_test2,y_predict2))  
4 print("Mean Squared Error: {MSE}",format(mse2,".2f"))  
5 print("Mean Absolute Error (MAE):",format(mae2,".2f"))  
6 print("Root Mean Square Error (RMSE): {rmse}",format(rmse2,".2f"))
```

```
➤ Mean Squared Error: {MSE} 23.18  
Mean Absolute Error (MAE): 3.39  
Root Mean Square Error (RMSE): {rmse} 4.81
```

```
[81] 1 r2 = r2_score(y_test2,y_predict2)  
2 print("R-squared:",format(r2,'.3f')) ##best
```

R-squared: 0.907

K – Nearest Neighbor Regression:

```
1 knn_regressor = KNeighborsRegressor(n_neighbors=11)
2 knn_regressor.fit(x_train, y_train)
```

```
• KNeighborsRegressor
KNeighborsRegressor(n_neighbors=11)
```

```
[87] 1 y_predict = knn_regressor.predict(x_test)
```

```
1 mse1 = mean_squared_error(y_test1, y_predict1)
2 mae1 = mean_absolute_error(y_test1, y_predict1)
3 rmse1 = math.sqrt(mean_squared_error(y_test1, y_predict1))
4 print("Mean Squared Error: {MSE}", format(mse1, ".2f"))
5 print("Mean Absolute Error (MAE):", format(mae1, ".2f"))
6 print("Root Mean Square Error (RMSE): {rmse}", format(rmse1, ".2f"))
```

```
Mean Squared Error: {MSE} 102.83
Mean Absolute Error (MAE): 7.69
Root Mean Square Error (RMSE): {rmse} 10.14
```

```
[93] 1 r2 = r2_score(y_test1, y_predict1) ##best
2 print("R-squared:", format(r2, '.2f'))
```

```
R-squared: 0.65
```


Support Vector Machine:

```
[102] 1 svr_regressor = SVR(kernel = "poly",C = 100)
      2 svr_regressor.fit(x_train, y_train)
```

• SVR:
SVR(C=100, kernel='poly')

```
1 y_predict = svr_regressor.predict(x_test)
```

```
[108] 1 mse1 = mean_squared_error(y_test1,y_predict1)
      2 mae1 = mean_absolute_error(y_test1,y_predict1)
      3 rmse1 = math.sqrt(mean_squared_error(y_test1,y_predict1))
      4 print("Mean Squared Error: {MSE}",format(mse1,".2f"))
      5 print("Mean Absolute Error (MAE):",format(mae1,".2f"))
      6 print("Root Mean Square Error (RMSE): {rmse}",format(rmse1,".2f"))
```

Mean Squared Error: {MSE} 70.40
Mean Absolute Error (MAE): 6.47
Root Mean Square Error (RMSE): {rmse} 8.39

```
1 r2 = r2_score(y_test1,y_predict1) ##best
2 print("R-squared:",format(r2,'.2f'))
```

R-squared: 0.76