AML Report part-2

by Sadikshya Duwadi

Submission date: 10-May-2024 11:14PM (UTC+0545)
Submission ID: 2376197864
File name: R_report_part-2.docx (3.54M)
Word count: 1438
Character count: 7974

"Data Analysis on Concrete Strength Part -2"

PART II

Sadikshya Duwadi
BSc (hons) Computing, 2024

Summary

The first part of the report was started by combining the two concrete strength datasets 'test' and 'train' followed by executing preliminary tasks. These tasks encompassed the following: Exploratory Data Analysis (EDA) to uncover hidden patterns and trends in the data, Data Visualization and Principal Component Analysis (PCA) to reduce dimensionality. Data pre-processing was done to find missing values and overall missingness in the data and the missing values were handled using mean imputation. Finally, the outliers were detected and removed. The respective means of the variables were used to replace any missing values and the variables were scaled. The final dataset contained 1030 rows and 10 columns.

Modelling

Equipped with the pre-processed data, five different machine learning algorithms were developed to predict the concrete strength.

Data frame of combined scaled numeric data with non-numeric data

```
> # Extracting non-numeric columns
> dataset <- final_imputed_data_without_outliers[, !sapply(final_imputed_data_without_outliers, is.numeric)]
> str(final_imputed_data_without_outliers)
'data.frame':	1030 obs. of  10 variables:
 $ Cement            : num  540 540 332 199 266 ...
 $ Blast.Furnace.Slag: num  0 0 142 132 114 ...
 $ Fly.Ash           : num  0 0 0 0 0 0 0 0 0 0 ...
 $ Water             : num  162 162 228 192 228 228 228 192 192 228 ...
 $ Superplasticizer  : num  2.5 2.5 0 0 0 0 0 0 0 0 ...
 $ Coarse.Aggregate  : num  1040 1055 932 978 932 ...
 $ Fine.Aggregate    : num  676 676 NA 826 670 ...
 $ Age               : int  28 28 NA NA 90 28 28 90 28 NA ...
 $ Strength          : num  NA 61.9 40.3 44.3 47 ...
 $ isTrain           : chr  "train" "train" "train" "train" ...
>
> # Combining scaled numeric data with non-numeric data
> final_data <- cbind(num_data, dataset)
> str(final_data)
'data.frame':	1030 obs. of  10 variables:
 $ Cement            : num  540 540 332 199 266 ...
 $ Blast.Furnace.Slag: num  0 0 142 132 114 ...
 $ Fly.Ash           : num  0 0 0 0 0 0 0 0 0 0 ...
 $ Water             : num  162 162 228 192 228 228 228 192 192 228 ...
 $ Superplasticizer  : num  2.5 2.5 0 0 0 0 0 0 0 0 ...
 $ Coarse.Aggregate  : num  1040 1055 932 978 932 ...
 $ Fine.Aggregate    : num  676 676 NA 826 670 ...
 $ Age               : int  28 28 NA NA 90 28 28 90 28 NA ...
 $ Strength          : num  NA 61.9 40.3 44.3 47 ...
 $ dataset           : chr  "train" "train" "train" "train" ...
>
> #Splitting the dataset
> strength_train<-final_data [final_data $isTrain=="train",]
> strength_test<-final_data [final_data $isTrain=="test", ]
>
```

Figure 1.1 Structure of Combined numeric data and dataset

Data frame of test and train data after removing the column is_Train.

```
> #removing variable isTrain from both train and test
> strength_train$isTrain<-NULL
> strength_test$isTrain<-NULL
> View(strength_train)
> View(strength_test)
> str(strength_train)
'data.frame':	642 obs. of  9 variables:
 $ Cement            : num  540 266 266 199 199 ...
 $ Blast.Furnace.Slag: num  0 114 114 132 132 ...
 $ Fly.Ash           : num  0 0 0 0 0 0 94 0 0 0 ...
 $ Water             : num  162 228 228 192 192 228 228 228 192 192 ...
 $ Superplasticizer  : num  2.5 0 0 0 0 0 0 0 0 0 ...
 $ Coarse.Aggregate  : num  1055 932 932 978 978 ...
 $ Fine.Aggregate    : num  676 670 670 826 826 ...
 $ Age               : int  28 90 28 90 28 90 28 90 100 28 ...
 $ Strength          : num  61.9 47 45.9 38.1 28 ...
> str(strength_test)
'data.frame':	281 obs. of  9 variables:
 $ Cement            : num  349 140 313 425 475 ...
 $ Blast.Furnace.Slag: num  0 209 262 114 119 ...
 $ Fly.Ash           : num  0 0 0 0 0 0 0 0 0 0 ...
 $ Water             : num  192 192 176 151 181 ...
 $ Superplasticizer  : num  0 0 8.6 18.6 8.9 12.1 16.5 11.6 10.3 15.9 ...
 $ Coarse.Aggregate  : num  1047 1047 1047 936 852 ...
 $ Fine.Aggregate    : num  807 807 612 804 782 ...
 $ Age               : int  3 7 3 3 3 3 3 3 1 3 ...
 $ Strength          : num  15.1 14.6 28.8 36.3 37.8 ...
```

Figure 1.2 Structure of Train and Test Datasets after removing is_Train

1. Linear Regression

The value of an independent variable is used to predict the value of a dependent variable using linear regression analysis. A straight line or surface that minimises the differences between the expected and actual output values is fitted using linear regression.

```
> # Training the model
> model <- train(formula, data = strength_train, method = "lm")
> summary(model)

Call:
lm(formula = .outcome ~ ., data = dat)

Residuals:
     Min       1Q   Median       3Q      Max 
-25.6247  -4.8739  -0.2046   5.2629  30.3648 

Coefficients:
                    Estimate Std. Error t value Pr(>|t|)    
(Intercept)        33.879180  26.309307   1.288  0.19831    
Cement              0.102604   0.008149  12.591  < 2e-16 ***
Blast.Furnace.Slag  0.079614   0.009785   8.136 2.16e-15 ***
Fly.Ash             0.047584   0.012100   3.933 9.33e-05 ***
Water              -0.228637   0.040834  -5.599 3.21e-08 ***
Superplasticizer    0.343216   0.104494   3.285  0.00108 ** 
Coarse.Aggregate   -0.001508   0.009262  -0.163  0.87070    
Fine.Aggregate     -0.007078   0.010872  -0.651  0.51524    
Age                 0.319834   0.011453  27.925  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 8.145 on 633 degrees of freedom
Multiple R-squared:  0.766,	Adjusted R-squared:  0.763 
F-statistic:   259 on 8 and 633 DF,  p-value: < 2.2e-16
```

Figure 1.1.1 Summary

```
> # Predicting on the test set
> predictions <- predict(model, newdata = strength_test)
> predictions
      731       733       737       738       739       740       742       743       744       745       746       747 
19.458509 15.923932 44.775750 52.188975 47.865245 42.654671 49.178813 46.594405 45.345685 50.091262 41.052972 47.377286 
      748       749       750       752       753       754       755       756       757       758       760       761 
46.192371 51.190600 52.855279 43.934006 51.190600 47.873740 50.190654 46.981377 33.723872 57.907107 55.861086 57.907107 
      762       763       764       765       767       768       769       770       771       772       773       775 
54.255013 58.087103 63.049134 66.862449 63.545588 74.260222 74.826780 77.256396 78.213759 75.425678 74.739766 69.198332 
      776       777       778       779       780       781       782       783       784       785       786       787 
12.903864 43.927727 15.718257 20.195928 29.151270 43.223950 33.330849 48.249487 18.652486 17.543091 22.899354 27.377025 
      788       789       790       791       793       794       795       796       797       798       799       800 
36.332367 13.723426 30.674609 46.600657 57.976886 31.697941 45.605699 16.338742 32.985562 24.541399 20.067693 28.063534 
      801       802       803       804       805       806       807       809       810       811       812       813 
28.894538 37.594357 21.556955 18.518760 29.552796 17.077868 27.817356 26.127535 27.688858 21.690671 29.686512 38.641854 
      814       815       816       817       818       819       820       821       822       823       824       825 
58.593047 34.044761 45.344926 33.557744 56.769799 45.983342 37.064498 29.475013 60.778023 34.823430 44.293596 58.366277 
      826       827       828       829       830       831       832       833       834       835       836       837 
37.965452 40.034630 41.241348 64.152151 32.123606 34.750091 49.074789 28.915265 31.154101 47.628385 23.589850 40.541033 
      838       839       840       841       842       843       844       845       846       847       848       849 
56.005340 54.668320 51.617591 42.123532 25.011826 47.284179 50.954843 22.378146 24.945546 19.791282 24.167812 24.845899 
      850       851       852       853       854       855       856       857       858       859       860       861 
17.235973 22.550186 24.959833 29.512708 27.787123 32.843155 23.507130 33.608590 23.365465 34.187156 36.742465 41.798497 
      862       863       864       865       866       867       868       869       870       871       872       873 
42.563932 32.320807 54.739648 50.815145 56.935148 52.663938 48.047827 55.532706 47.993106 63.218773 42.890626 58.252181 
      874       875       876       877       878       879       880       881       882       883       884       885 
```

Figure 1.1.2 Predictions of Linear Regression

Above Figure 1.1.1 shows the summary of the linear regression model and Figure 1.1.2 shows the prediction made by linear regression model in test dataset.

```
> # Printing the metrics
> print(paste("R2: ", r2))
[1] "R2:  0.747564646718755"
> print(paste("Adjusted R2: ", adj_r2))
[1] "Adjusted R2:  0.740140077504601"
> print(paste("MSE: ", mse))
[1] "MSE:  69.8687558871473"
> print(paste("RMSE: ", rmse))
[1] "RMSE:  8.35875324956702"
> print(paste("MAE: ", mae))
[1] "MAE:  6.48433978334516"
> # Creating a data frame with the actual and predicted values
> comparison <- data.frame(Actual = strength_test$Strength, Predicted = predictions)
> # Creating a scatter plot
> ggplot(comparison, aes(x = Actual, y = Predicted)) +
+   geom_point() +
+   geom_smooth(method = lm, se = FALSE, color = "blue") +
+   labs(title = "Actual vs Predicted", x = "Actual", y = "Predicted") +
+   theme_minimal()
`geom_smooth()` using formula = 'y ~ x'
```

Figure 1.1.3 Evaluation Metrics

The value for evaluating the metrics is printed in the above fig.1.1.3. Different performance metrics: R-squared (R2), Adjusted R-squared (Adj. R2), Mean Squared Error (MSE), Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE) values are obtained for linear regression model.

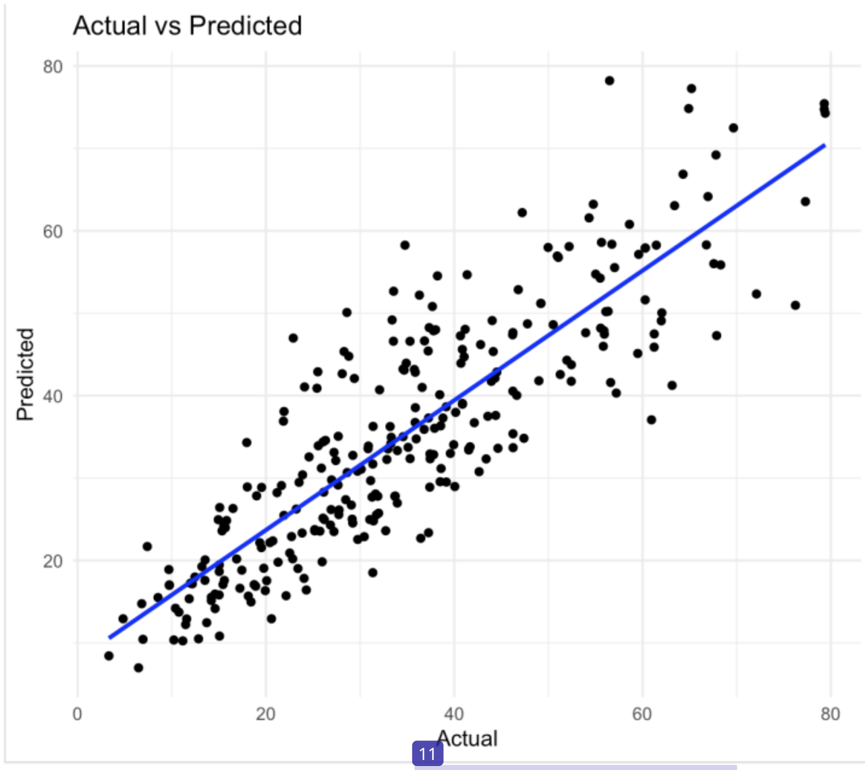

Figure 1.1.4 Scatter Plot for Linear Regression with actual vs predicted values

The predicted values of a linear regression model is shown against the actual values in the above scatter plot.

2. Random Forest

A popular machine learning approach called Random Forest mixes the outputs of several decision trees to get a single outcome. Its versatility and ease of use, combined with its ability to handle both regression and classification issues, makes it more flexible to use.

```
> # Training the model
> model <- randomForest(formula, data = strength_train)
> plot(model)
> # Printing the model summary
> print(summary(model))
                Length Class  Mode
call              3    -none- call
type              1    -none- character
predicted       642    -none- numeric
mse             500    -none- numeric
rsq             500    -none- numeric
oob.times       642    -none- numeric
importance        8    -none- numeric
importanceSD      0    -none- NULL
localImportance   0    -none- NULL
proximity         0    -none- NULL
ntree             1    -none- numeric
mtry              1    -none- numeric
forest           11    -none- list
coefs             0    -none- NULL
y               642    -none- numeric
test              0    -none- NULL
inbag             0    -none- NULL
terms             3    terms  call
```

Figure 1.2.1 Summary of Random Forest

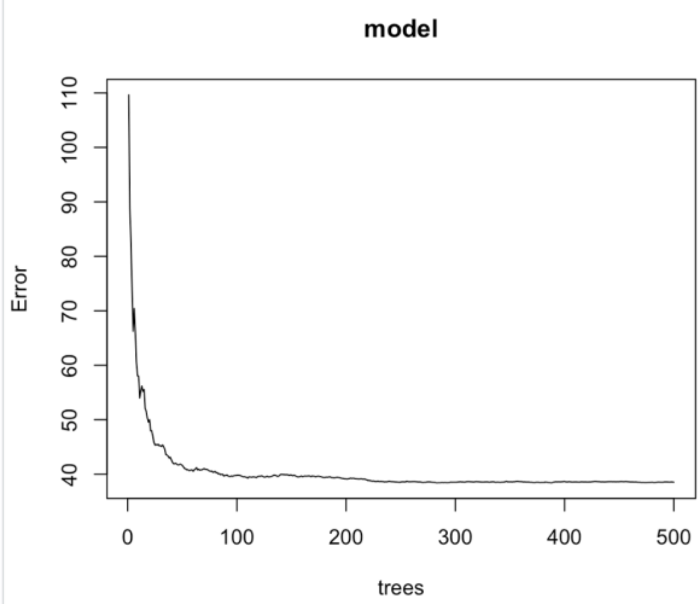

Figure 1.2.2 Plotting the Model value

The model value after using randomForest() on Strength_Train data is plotted and is shown in fig.1.2.2, where the plot of the errors in y-axis is from 0-110 as the number of trees rises from 0 to 500 in x-axis using a random forest model trained on train data.

```
> # Printing the metrics
> print(paste("R2: ", r2))
[1] "R2:  0.875868771262608"
> print(paste("Adjusted R2: ", adj_r2))
[1] "Adjusted R2:  0.876310519407581"
> print(paste("MSE: ", mse))
[1] "MSE:  37.3859024943196"
> print(paste("RMSE: ", rmse))
[1] "RMSE:  6.11440123759634"
> print(paste("MAE: ", mae))
[1] "MAE:  4.50949076974893"
```

Figure 1.2.3 Evaluation Metrics

The value for evaluating the metrics is printed in the above fig.1.2.3. Different performance metrics: R-squared (R2), Adjusted R-squared (Adj. R2), Mean Squared Error (MSE), Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE) values are obtained for Random Forest model.

```
> # Creating a data frame with the actual and predicted values
> comparison <- data.frame(Actual = strength_test$Strength, Predicted = predictions)
> # Creating a scatter plot
> ggplot(comparison, aes(x = Actual, y = Predicted)) +
+   geom_point() +
+   geom_smooth(method = lm, se = FALSE, color = "blue") +
+   labs(title = "Actual vs Predicted", x = "Actual", y = "Predicted") +
+   theme_minimal()
`geom_smooth()` using formula = 'y ~ x'
```

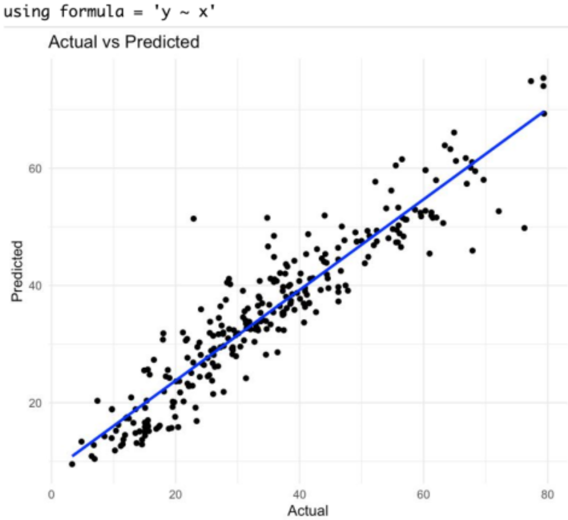

Figure 1.2.4 Scatter Plot for Random Forest with actual vs predicted values

The predicted of a *Random Forest* model is shown against the actual values in the above scatter plot. Here we can see the data seem to be more closely packed around the diagonal line than in Linear Regression Model suggesting that the *Random Forest* model would offer a much better fit to the data.

3. Decision Tree

In a decision tree, each attribute test is represented by each internal node, the test result is represented by each branch, and the class label is held by each leaf node, or terminal node, which resembles a flowchart. The decision tree works by splitting the data into smaller and smaller subsets based on certain features.

```
> # Plot the decision tree
> fancyRpartPlot(model, sub = '')
> # Plot the decision tree
> fancyRpartPlot(model, sub = '')
>
```

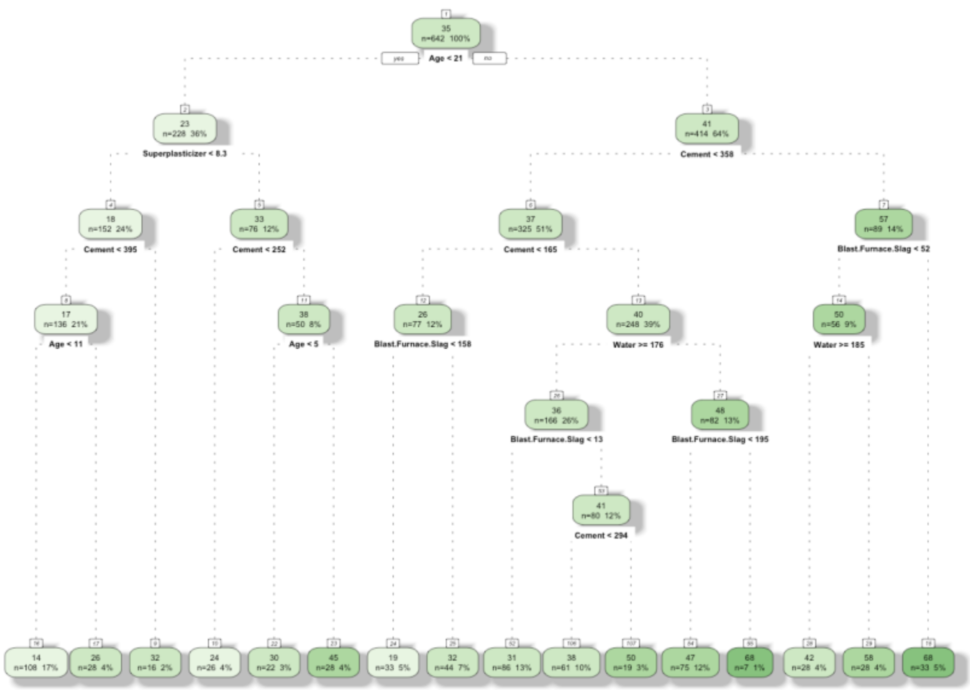

Figure 1.3.1 Decision Tree Model Plot

In fig.1.3.1, we can see a decision tree visualisation produced by using fancyRpartPlot() function to forecast the target variable. The tree divides the data into multiple categories trying to predict the compressive strength of concrete. Initially, the decision tree divides the data into two categories according on whether the concrete is older than 11 years. Next, it divides the data once again according to whether or not there is less than 158 blast furnace slag. The procedure carries on until the decision tree reaches a leaf node, which has a forecast regarding the concrete's compressive strength.

```
> # Predicting on the test set
> predictions <- predict(model, newdata = strength_test)
> predictions
     731      733      737      738      739      740      742      743      744      745 
14.18509 14.18509 29.66955 29.66955 29.66955 29.66955 29.66955 29.66955 29.66955 29.66955 
     746      747      748      749      750      752      753      754      755      756 
29.66955 44.58107 44.58107 44.58107 44.58107 44.58107 44.58107 44.58107 44.58107 44.58107 
     757      758      760      761      762      763      764      765      767      768 
44.58107 68.10303 68.10303 68.10303 67.84143 68.10303 68.10303 68.10303 68.10303 68.10303 
     769      770      771      772      773      775      776      777      778      779 
68.10303 68.10303 68.10303 68.10303 68.10303 46.66040 14.18509 30.57547 25.88750 30.57547 
     780      781      782      783      784      785      786      787      788      789 
30.57547 30.57547 46.66040 46.66040 14.18509 25.88750 24.15615 46.66040 46.66040 14.18509 
     790      791      793      794      795      796      797      798      799      800 
30.57547 30.57547 38.21016 30.57547 30.57547 14.18509 30.57547 30.57547 24.15615 46.66040 
     801      802      803      804      805      806      807      809      810      811 
46.66040 46.66040 14.18509 25.88750 46.66040 14.18509 46.66040 46.66040 46.66040 24.15615 
     812      813      814      815      816      817      818      819      820      821 
46.66040 46.66040 46.66040 46.66040 46.66040 46.66040 46.66040 46.66040 46.66040 24.15615 
```

Figure 1.3.2 Predictions of test dataset.

```
> # Printing the metrics
> print(paste("R2: ", r2))
[1] "R2:  0.694009737825136"
> print(paste("Adjusted R2: ", adj_r2))
[1] "Adjusted R2:  0.695098671142484"
> print(paste("MSE: ", mse))
[1] "MSE:  84.6328556995497"
> print(paste("RMSE: ", rmse))
[1] "RMSE:  9.19961171460784"
> print(paste("MAE: ", mae))
[1] "MAE:  7.29603865850811"
>
```

Figure 1.3.3 Evaluation Metrics

In the above fig.1.3.3, the data probability is displayed for each decision: R2, Adjusted R2, MSE, RMSE, and MAE.

```
> # Creating a data frame with the actual and predicted values
> comparison <- data.frame(Actual = strength_test$Strength, Predicted = predictions)
> # Creating a scatter plot with different colors for actual and predicted values
> ggplot(comparison, aes(x = Actual, y = Predicted)) +  # Initialize the plot with data
+   geom_point() +  # Add a layer of points
+   geom_smooth(method = lm, se = FALSE, color = "blue") +  # Add a layer of a linear regression line
+   labs(title = "Actual vs Predicted", x = "Actual", y = "Predicted") +  # Add labels
+   theme_minimal()
`geom_smooth()` using formula = 'y ~ x'
```

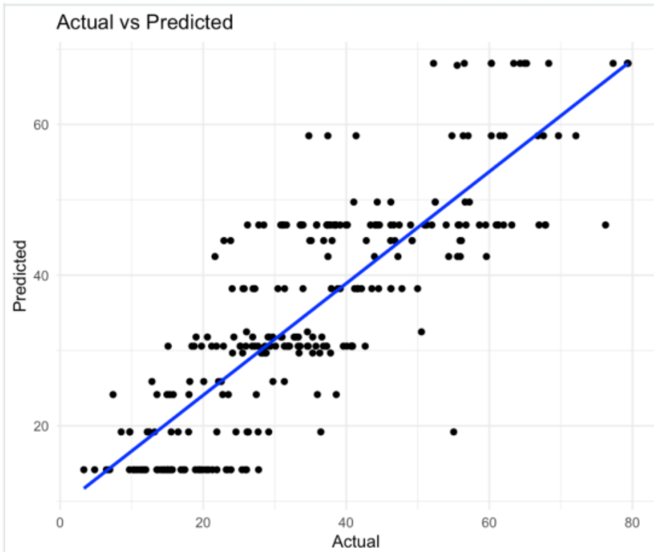

Figure 1.3.4 Scatter Plot for Decision Tree with actual vs predicted values

In the fig.1.3.4, scatter plot for Decision Tree with actual vs predicted values is shown to evaluate the model's accuracy and decision-making abilities.

4. k-Nearest Neighbours (KNN)

KNN is a member of the supervised learning domain. It is simple to grasp, nonparametric as it does not make any underlying assumptions about the distribution of the data and lazy. However, for huge datasets, it may be computationally costly.

```
> # Defining training control
> train_control <- trainControl(method = "cv", number = 10)
> model <- train(formula, data = strength_train, method = "knn", trControl = train_control)
> # Printing the model summary
> print(summary(model))
            Length Class      Mode
learn       2      -none-     list
k           1      -none-     numeric
theDots     0      -none-     list
xNames      8      -none-     character
problemType 1      -none-     character
tuneValue   1      data.frame list
obsLevels   1      -none-     logical
param       0      -none-     list
> # Predicting on the test set
> predictions <- predict(model, newdata = strength_test)
> predictions
  [1] 18.32200 17.78600 49.76400 58.08000 61.47800 44.38000 50.81500 43.54000 50.50800 51.77200 42.34000
 [12] 42.01667 49.76400 50.81500 58.08000 44.38000 50.81500 43.54000 62.27800 43.54000 25.07600 50.81500
 [23] 61.47800 50.81500 47.43800 51.77200 72.85800 60.65800 72.61800 75.54000 76.96667 63.32000 54.37600
 [34] 76.96667 76.96667 74.52714 19.69200 38.81400 19.89000 23.17600 32.50800 38.81400 32.36400 39.09800
 [45] 21.57500 18.94600 23.38000 26.46000 35.82200 18.95800 29.91200 49.42600 47.94800 30.83400 40.51000
 [56] 19.89000 32.66800 23.86400 20.43600 29.33400 29.19200 32.84600 17.70600 21.57500 29.61800 19.01800
 [67] 25.55200 23.56200 30.93800 14.45000 27.69167 36.44400 47.23200 32.71600 41.23000 31.74200 45.73800
 [78] 45.70200 31.74200 35.55200 49.38000 34.50800 36.76800 44.81200 31.71400 40.07400 36.42000 47.94800
 [89] 28.52200 28.52200 48.97800 34.22400 34.22400 52.52000 19.53800 40.92200 55.88800 49.36600 51.39400
[100] 43.53200 26.36800 45.52200 48.08800 34.86800 34.86800 17.57600 31.99000 31.27400 21.08600 20.27200
[111] 22.84600 40.70200 20.27200 40.74200 22.79800 31.71400 28.92800 33.49200 25.85500 45.97200 39.24600
[122] 32.36400 51.53600 39.98400 53.37800 46.30200 42.71800 48.02800 35.00800 55.37400 40.73600 41.15000
```

Figure 1.4.1 Summary and Predictions

Above fig.1.4.1 prints the summary of the KNN model and gives the prediction on the test set data.

```
> # Printing the metrics
> print(paste("R2: ", r2))
[1] "R2:  0.681509866180901"
> print(paste("Adjusted R2: ", adj_r2))
[1] "Adjusted R2:  0.682643283027232"
> print(paste("MSE: ", mse))
[1] "MSE:  87.6233608098789"
> print(paste("RMSE: ", rmse))
[1] "RMSE:  9.36073505713514"
> print(paste("MAE: ", mae))
[1] "MAE:  7.05612862226741"
>
```

Figure 1.4.2 Evaluation Metrics on

The value for evaluating the metrics is printed in the above fig.1.2.3. Different performance metrics: R-squared (R2), Adjusted R-squared (Adj. R2), Mean Squared Error (MSE), Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE) values are obtained for KNN model.

```
> # Creating a data frame with the actual and predicted values
> comparison <- data.frame(Actual = strength_test$Strength, Predicted = predictions)
> # Creating a scatter plot
> ggplot(comparison, aes(x = Actual, y = Predicted)) +
+   geom_point() +
+   geom_smooth(method = lm, se = FALSE, color = "green") +
+   labs(title = "Actual vs Predicted", x = "Actual Strength", y = "Predicted Strength") +
+   theme_minimal()
`geom_smooth()` using formula = 'y ~ x'
```

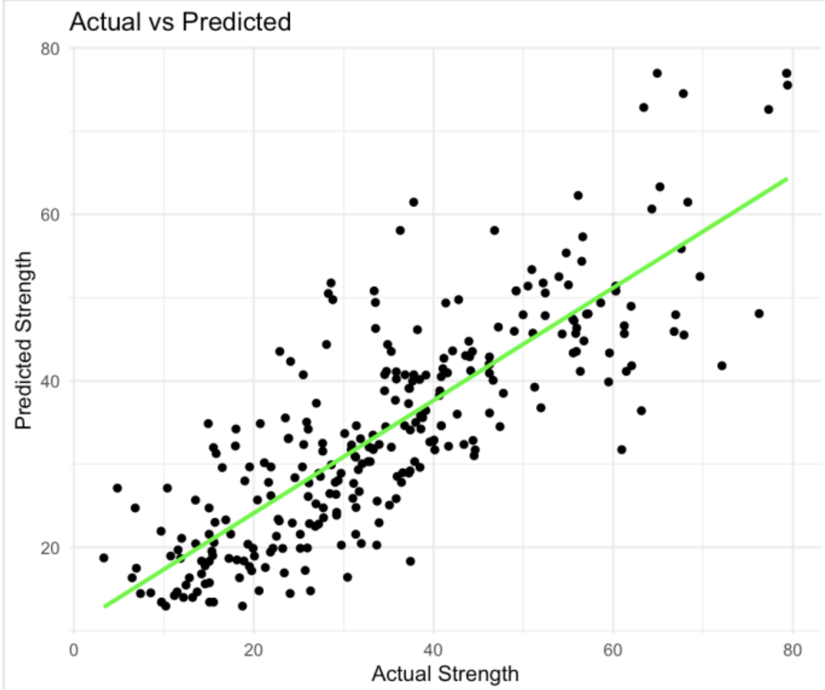

Figure 1.4.3 Scatter Plot for KNN with actual vs predicted strength

The above scatter plot showing the relationship between actual strength on the x-axis and predicted strength on the y-axis.

5. KNN Model with optimal value of k = 4

```
> # Printing the metrics
> print(paste("R2: ", r2))
[1] "R2:  0.679303876342072"
> print(paste("Adjusted R2: ", adj_r2))
[1] "Adjusted R2:  0.68044514368605"
> print(paste("MSE: ", mse))
[1] "MSE:  88.0285416568802"
> print(paste("RMSE: ", rmse))
[1] "RMSE:  9.38235267173859"
> print(paste("MAE: ", mae))
[1] "MAE:  7.0849709371293"
>
```

Figure 1.5.1 Evaluation Metrics

```
> # Creating a data frame with the actual and predicted values
> comparison <- data.frame(Actual = strength_test$Strength, Predicted = predictions)
> # Creating a scatter plot
> ggplot(comparison, aes(x = Actual, y = Predicted)) +
+   geom_point() +
+   geom_smooth(method = lm, se = FALSE, color = "green") +
+   labs(title = "Actual vs Predicted", x = "Actual Strength", y = "Predicted Strength") +
+   theme_minimal()
`geom_smooth()` using formula = 'y ~ x'
```

Figure 1.5.2 Scatter Plot for KNN=4 with actual vs predicted values

The above scatter plot showing the relationship between actual strength on the x-axis and predicted strength) on the y-axis with different KNN value i.e. KNN=4.

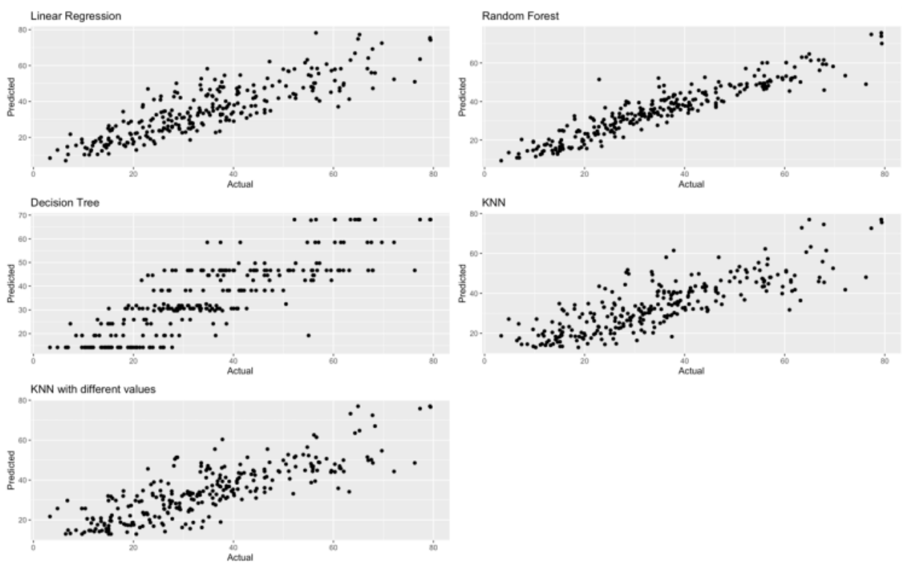

Figure 1.6 Scatter Plot for all five Models

The scatter plot for all five different models that were developed is shown to compare in the above fig.1.6. The scatter plot of the Random Forest and Linear Regression seems to be much more compacted and fit for the data than other models.

```
> # Print the metrics
> print(metrics)
              Model        R2    Adj_R2      MSE     RMSE      MAE
1 Linear Regression 0.7475646 0.7466599 69.86876 8.358753 6.484340
2     Random Forest 0.8745155 0.8740657 37.90529 6.156728 4.529004
3     Decision Tree 0.6940097 0.6929130 84.63286 9.199612 7.296039
4               KNN 0.6815099 0.6803683 87.62336 9.360735 7.056129
5         KNN (k=4) 0.6793039 0.6781544 88.02854 9.382353 7.084971
> # Select the best model based on the evaluation metrics
> best_model <- metrics[which.min(metrics$RMSE), ]
> print("Best Predicting Model based on RMSE:")
[1] "Best Predicting Model based on RMSE:"
> print(best_model)
          Model        R2    Adj_R2      MSE     RMSE      MAE
2 Random Forest 0.8745155 0.8740657 37.90529 6.156728 4.529004
>
```

Figure 1.7 Finding the Best Model

Above, the metrics for all five models are printed and the best predicting model based on evaluation metrics is shown in fig.1.7.

Model	R2	Adj. R2	MSE	RMSE	MAE
LR	0.7474646	0.7401400	69.8687558	8.3587532	6.4843397
RF	0.8758687	0.8763105	37.3859024	6.1144012	4.5094907
DECISION	0.6940097	0.6950986	84.6328556	9.1996117	7.2960386
KNN	0.6815098	0.6826432	87.6233608	9.3607350	7.0561286
KNN=4	0.6793038	0.6804451	88.0285416	9.3823526	7.0849709

An overview of the many regression models assessed using a range of performance metrics is given in this table which are: R-squared (R2), Adjusted R-squared (Adj. R2), Mean Squared Error (MSE), Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE). The five models which were assessed are: Linear Regression Model, Random Forest Model, k-Nearest Neighbours Model (KNN), Decision Tree Model, and a Modified KNN model (KNN=4). We may compare and choose the best model for the task at hand by observing each model's performance across these measures based on this summary.

Model Interpretation

We can observe that, in comparison to the other models, the Random Forest (RF) model has the highest R-squared (0.8758687) and the lowest values for MSE, RMSE, and MAE indicating a superior fit to the data with lower error and high accuracy rate. When extracting and plotting the importance of random forest in the code we can see the following:

	IncNodePurity
Blast.Furnace.Slag	11066.801
Water	24434.640
Coarse.Aggregate	10528.946
Age	50511.355
Cement	34792.586
Fly.Ash	9079.506
Superplasticizer	17187.622
Fine.Aggregate	12678.759

Based on the test results, the Random Forest model seems to be the most accurate predictor. A more thorough understanding of the variables affecting concrete strength may result from this knowledge, which might enhance decision-making and optimise the design and formulation of concrete mixes.

Conclusion

In conclusion, Among the five different models that was used in the regression problems, the most accurate one is the Random Forest as the model is consistent. In case of the regression model, MAE, MSE, EMSE, R2 and adjusted R2 we, R2 gave more variance between predictor variable and the target variable. The variance measured by R2 is the ratio of two different variables one being dependent variable which is predictable from the independent variable. Evaluation metrices help find or explain the variability in the data.

Bibliography

- Ibm.com. (2024). *What Is Linear Regression? | IBM*. [online] Available at: https://www.ibm.com/topics/linear-regression#:~:text=IBM-,What%20is%20linear%20regression%3F,is%20called%20the%20independent%20variable. [Accessed 10 May 2024].

- Ibm.com. (2024). *What Is Random Forest? | IBM*. [online] Available at: https://www.ibm.com/topics/random-forest [Accessed 10 May 2024].

- GeeksforGeeks (2017). *Decision Tree*. [online] GeeksforGeeks. Available at: https://www.geeksforgeeks.org/decision-tree/ [Accessed 10 May 2024].

AML Report part-2

ORIGINALITY REPORT

20%	16%	7%	16%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	dergipark.org.tr Internet Source	4%
2	Submitted to University of Bradford Student Paper	3%
3	Submitted to University of Surrey Student Paper	2%
4	user-jbdbkvi.cld.bz Internet Source	2%
5	Submitted to Liverpool John Moores University Student Paper	2%
6	Submitted to University of North Texas Student Paper	2%
7	escholarship.org Internet Source	1%
8	Submitted to Leeds Beckett University Student Paper	1%
9	Submitted to University of Wales Institute, Cardiff	1%

Student Paper

10	repository.effatuniversity.edu.sa Internet Source	1%
11	Submitted to University of Bolton Student Paper	1%
12	cs229.stanford.edu Internet Source	1%
13	Submitted to University of the Sunshine Coast Student Paper	<1%
14	fastercapital.com Internet Source	<1%
15	research.tue.nl Internet Source	<1%
16	Yuhui Lyu, Madura Pathirage, Elham Ramyar, Wing Kam Liu, Gianluca Cusatis. "Machine learning meta-models for fast parameter identification of the lattice discrete particle model", Computational Mechanics, 2023 Publication	<1%

Exclude quotes Off

Exclude bibliography Off

Exclude matches Off