Contents

TABLE OF FIGURES

DATA PREPARATION AND RESEARCH QUESTION

The data used for this analytical study is a derivative data from Open Canada Data where it represents the fuel consumption of cars from the year 2000 until 2022. The data contains over 22000 observations with 13 variables. Moreover, the variables consist of categorical and numerical variables that provide detailed information about each of the cars. The following data dictionary showcases the data variables and types:

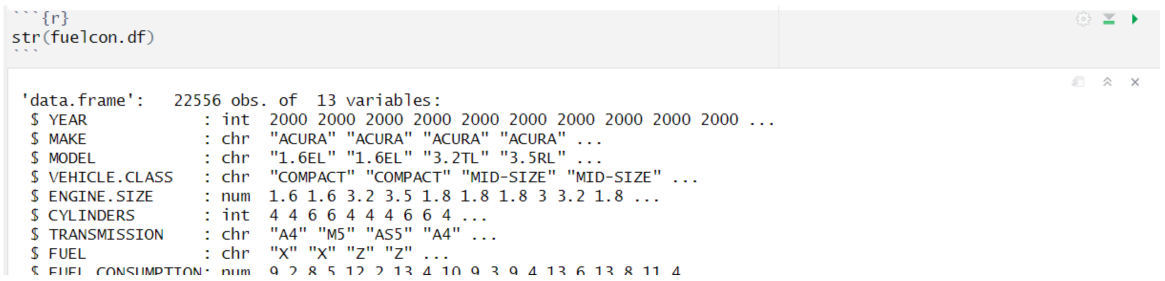
Variable Name	Description	Type
Year	No of years from where the data has been taken	Integer
Make	The company that makes the car	Categorical
Model	Model of the cars/Vehicles	Categorical
Vehicle class	Vehicle class	Categorical
Engine size	Engine size	Continuous
Cylinders	No of Cylinders	Integer
Transmission	Type of Transmission	Categorical
Fuel	Type of Fuel	Categorical
Fuel consumption	Fuel consumption by the vehicle	Numerical
HWY (L/100km)	Highway fuel consumption 100km per liter	Continuous
COMB (L/100km)	Combined rating of both city and highway	Continuous
COMB (mpg)	Combined rating (miles per gallon)	Continuous
EMISSIONS	CO2 Emissions (grams per kilometer)	Continuous

Table 1: Table for variables and its Type in the dataset

The research question is to predict CO2 Emissions in Cars given various variables from 2016 to 2022.

DATA PREPARATION AND CLEANING

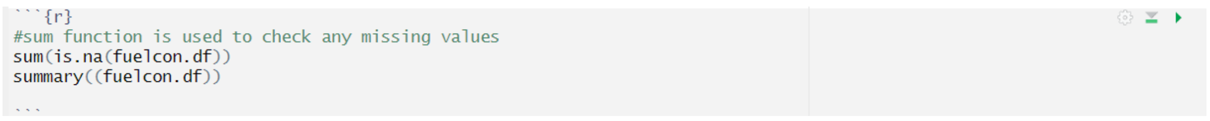
Initially, we examine the structure of the fuelcon.df dataset to understand the variables and their respective values using the str() function. The dataset contains a mix of numerical and categorical variables, which will be used for training and testing our machine learning model.



~~~
```{r}
str(fuelcon.df)
```
~~~

```
'data.frame':   22556 obs. of  13 variables:
 $ YEAR            : int  2000 2000 2000 2000 2000 2000 2000 2000 2000 2000 ...
 $ MAKE            : chr  "ACURA" "ACURA" "ACURA" "ACURA" ...
 $ MODEL           : chr  "1.6EL" "1.6EL" "3.2TL" "3.5RL" ...
 $ VEHICLE.CLASS   : chr  "COMPACT" "COMPACT" "MID-SIZE" "MID-SIZE" ...
 $ ENGINE.SIZE     : num  1.6 1.6 3.2 3.5 1.8 1.8 1.8 3 3.2 1.8 ...
 $ CYLINDERS       : int  4 4 6 6 4 4 4 6 6 4 ...
 $ TRANSMISSION    : chr  "A4" "M5" "AS5" "A4" ...
 $ FUEL            : chr  "X" "X" "Z" "Z" ...
 $ FUEL CONSUMPTION: num  9 2 8 5 12 2 13 4 10 9 3 9 4 13 6 13 8 11 4
```

Figure 1: Table for structure of the dataset



~~~
```{r}
#sum function is used to check any missing values
sum(is.na(fuelcon.df))
summary((fuelcon.df))

```
~~~

Figure 2: R code to check missing values

We then check for missing values in the dataset using the sum() function along with is.na(). The results indicate that the dataset does not have any missing values. We proceed to explore the categorical variables (MAKE, MODEL, VEHICLE.CLASS, FUEL, and TRANSMISSION) by converting them to upper case and removing duplicate values, which improves the readability and consistency of the dataset. Next, we visualize the numerical variables using boxplots to identify potential outliers. We use the boxplot() function for each numerical variable and loop through the columns to create a boxplot for each column. We identify the rows containing outliers using the which() function and save the outlier values for each variable in separate variables (outliers, outliers1, etc.).

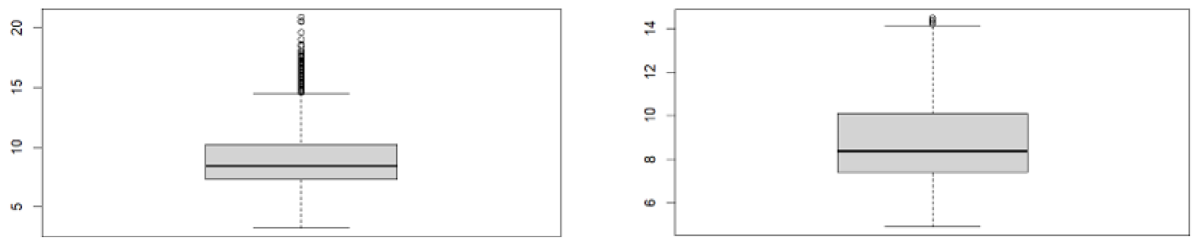


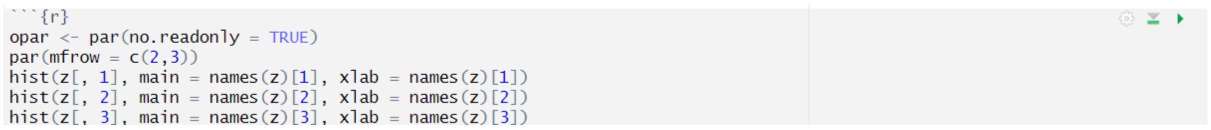
Figure 3: Boxplot of HWY..L.100.km before and after outlier removal

We then combine all the outlier row indexes into a single vector (all_outliers) and remove all outlier rows from the dataset using the new variable z. This comprehensive data preparation and cleaning process ensures that the dataset is ready for further analysis and modeling.

EDA (EXPLORATORY DATA ANALYSIS)

This code is performing an Exploratory Data Analysis (EDA) on a cleaned dataset called `z` (z is a variable which is storing the outlier free data). EDA is a crucial step in data analysis, as it allows us to understand the structure, relationships, and patterns in the data. In this case, the code focuses on visualizations and correlation analysis. I'll break down the code into sections and discuss the EDA aspects.

1. Visualizing the distribution of variables using histograms:



~~~
```{r}
opar <- par(no.readonly = TRUE)
par(mfrow = c(2,3))
hist(z[, 1], main = names(z)[1], xlab = names(z)[1])
hist(z[, 2], main = names(z)[2], xlab = names(z)[2])
hist(z[, 3], main = names(z)[3], xlab = names(z)[3])
~~~

Figure 4: R code to visualise the distribution of variables using histograms

The code sets up a plotting environment with a 2x3 grid and then plots the histograms for all the numerical variables in the dataset. Histograms provide an overview of the frequency distribution of each variable, which helps in understanding the shape, center, and spread of the data.

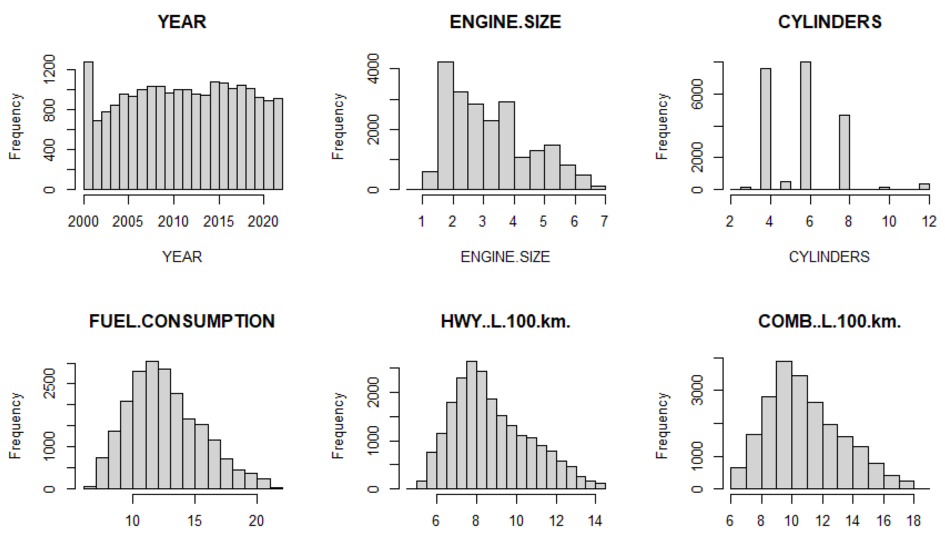


Figure 5: A sample image of 6 variables along with its frequency represented in histogram

2. Visualizing variables using boxplots:

This section plots boxplots for all 8 variables in the dataset. Boxplots display the median, quartiles, and potential outliers of each variable, providing an idea of the data's spread and skewness.

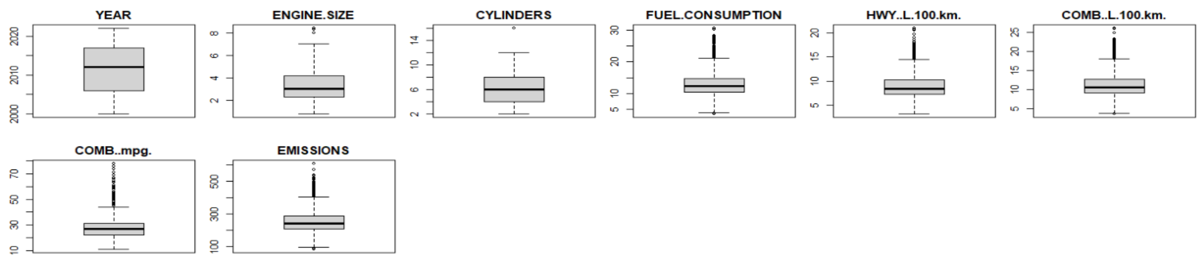
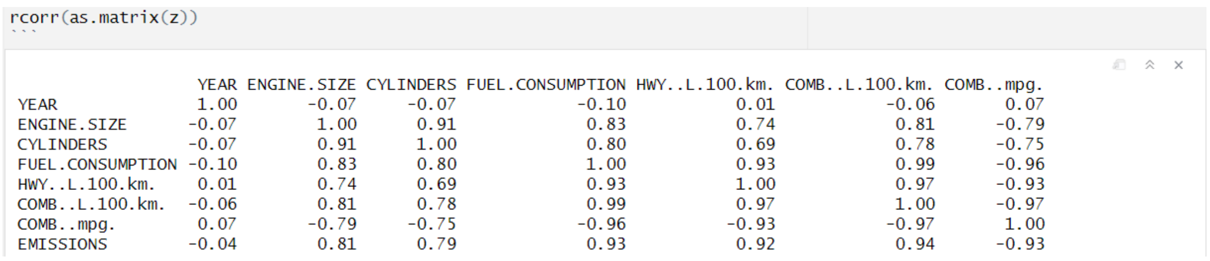


Figure 6: Boxplots for all 8 variables

3. Calculating p-value:

The code uses the `rcorr()` function to calculate the p-values for the variables in the dataset. The p-values indicate the significance of the correlations.



```
rcorr(as.matrix(z))
```

```
                 YEAR ENGINE.SIZE CYLINDERS FUEL.CONSUMPTION HWY..L.100.km. COMB..L.100.km. COMB..mpg.
YEAR              1.00       -0.07     -0.07            -0.10           0.01           -0.06       0.07
ENGINE.SIZE      -0.07        1.00      0.91             0.83           0.74            0.81      -0.79
CYLINDERS        -0.07        0.91      1.00             0.80           0.69            0.78      -0.75
FUEL.CONSUMPTION -0.10        0.83      0.80             1.00           0.93            0.99      -0.96
HWY..L.100.km.    0.01        0.74      0.69             0.93           1.00            0.97      -0.93
COMB..L.100.km.  -0.06        0.81      0.78             0.99           0.97            1.00      -0.97
COMB..mpg.        0.07       -0.79     -0.75            -0.96          -0.93           -0.97       1.00
EMISSIONS        -0.04        0.81      0.79             0.93           0.92            0.94      -0.93
```

Figure 7: Calculating p-value to see whether the correlation is significant

4. Visualizing the correlation matrix using corrplot:

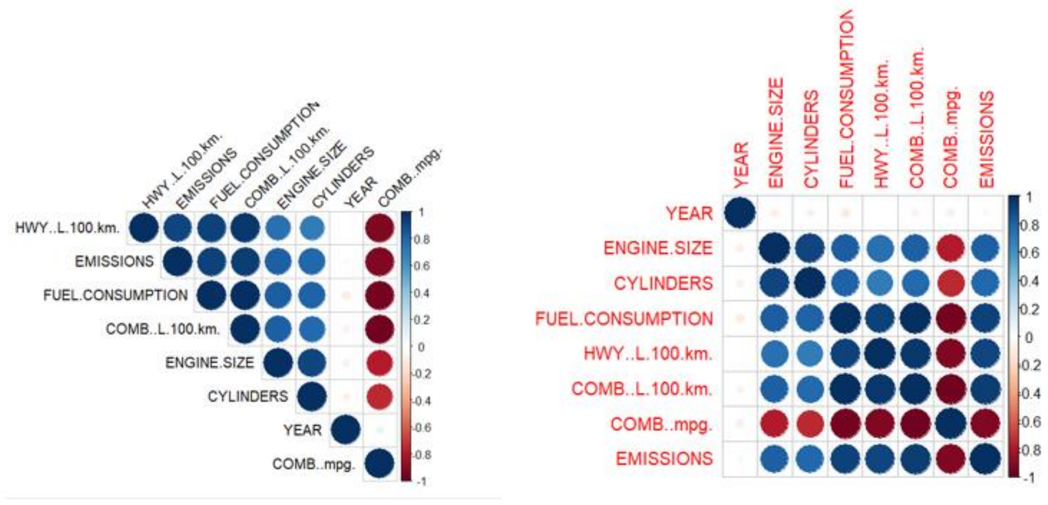


Figure 8: Visualising the correlation. it plots a hierarchical clustered heatmap of the upper-triangle correlation matrix using res

The `corrplot()` function is used to visualize the correlation matrix using circle plots. Circle plots help to identify the strength and direction of relationships between variables, allowing for further analysis and feature selection. Overall, the code provides a comprehensive EDA of the cleaned dataset `z`, helping to understand the data's structure, relationships, and patterns. By exploring the correlation between variables, it enables better feature selection and insights for further analysis.

CLUSTERING ANALYSIS:

The clustering analysis code provided performs k-means clustering on the mobile dataset to group similar data points. The process starts by selecting relevant variables and scaling them to standardize the values. Then, Principal Component Analysis (PCA) is applied to reduce the dataset's dimensionality, making it more manageable for clustering. To find the optimal number of clusters, the within-cluster sum of squares (WCSS) is calculated for different cluster sizes, and the elbow method is employed. Once the ideal number of clusters is determined, k-means clustering is carried out using that number. The results, including the cluster assignments and visualizations, provide insights into how the data points are grouped based on their similarities.

1. **Data preprocessing and PCA:** The selected variables from the `z` dataset are scaled to have zero mean and unit variance. Then PCA is applied to the scaled data, reducing its dimensionality and retaining the most significant components.

2. **Determining the optimal number of clusters:** The code calculates the WCSS for cluster sizes ranging from 1 to 10. It then plots the WCSS against the number of clusters to visually identify the elbow point, which represents the optimal number of clusters. In this example, the optimal number of clusters is 3.

3. **K-means clustering:** With the optimal number of clusters determined, k-means clustering is performed on the PCA-transformed data using the `kmeans()` function. The algorithm assigns each data point to one of the three clusters based on their similarity in the PCA-transformed space.

4. **Visualizing the clustered data points:** The code plots the clustered data points in the PCA-transformed space, with different colors representing each cluster. Additionally, the cluster centroids are plotted using the `points()` function.

5. **Cluster assignments:** Finally, the code prints the cluster assignments for each data point, providing a clear representation of which cluster each point belongs to. By conducting this clustering analysis, the data points in the mobile dataset can be effectively grouped based on their similarities, revealing underlying patterns and relationships that can be used for further analysis or decision-making processes.

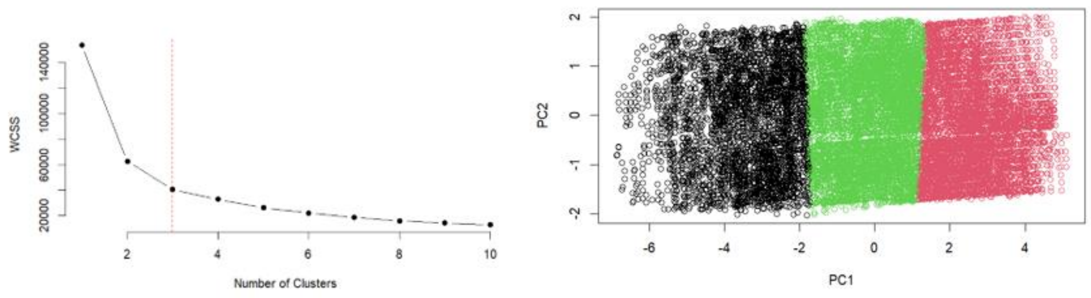
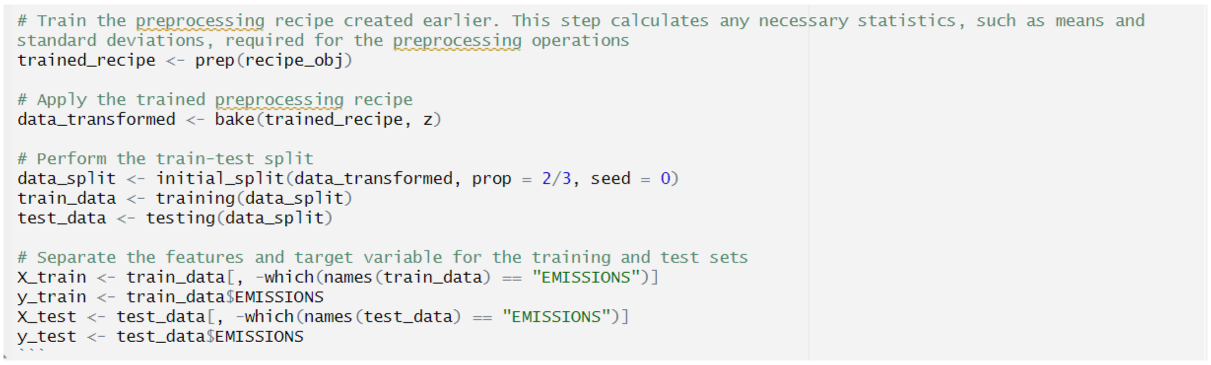


Figure 9: Plot of cluster data points and centriods (right), Plot the WCSS against the number of clusters (left)

MACHINE LEARNING:

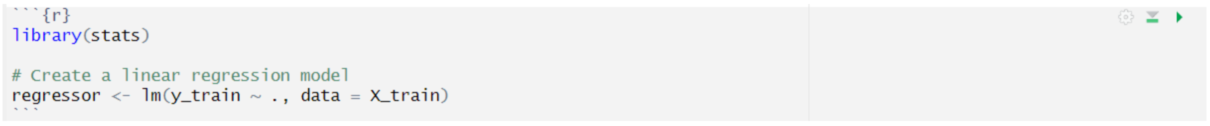
In this R project, the objective is to predict vehicle emissions using linear regression and polynomial regression models. The dataset is first preprocessed and split into training and testing sets for model evaluation.



```
# Train the preprocessing recipe created earlier. This step calculates any necessary statistics, such as means and
standard deviations, required for the preprocessing operations
trained_recipe <- prep(recipe_obj)

# Apply the trained preprocessing recipe
data_transformed <- bake(trained_recipe, z)

# Perform the train-test split
data_split <- initial_split(data_transformed, prop = 2/3, seed = 0)
train_data <- training(data_split)
test_data <- testing(data_split)

# Separate the features and target variable for the training and test sets
X_train <- train_data[, -which(names(train_data) == "EMISSIONS")]
y_train <- train_data$EMISSIONS
X_test <- test_data[, -which(names(test_data) == "EMISSIONS")]
y_test <- test_data$EMISSIONS
```

Figure 10: Sample R code for splitting the data into training and testing data before implementing machine learning model

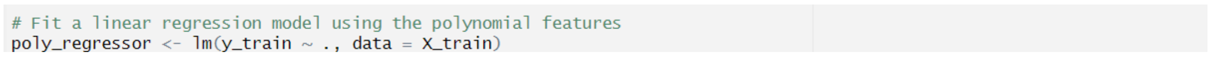
LINEAR REGRESSION: The linear regression model is built using the "stats" package. The model's coefficients and intercept are extracted, and predictions are made for the test dataset. Performance metrics like Mean Absolute Error (MAE), R-squared (R2) score, and accuracy are calculated to evaluate the model.



~~~
```{r}
library(stats)

# Create a linear regression model
regressor <- lm(y_train ~ ., data = X_train)
```
~~~

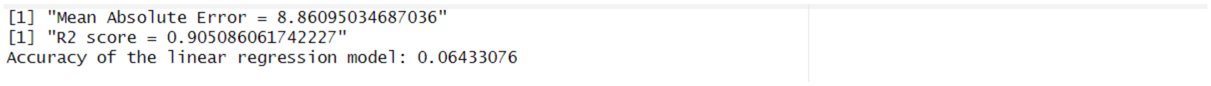
Figure 11: R code for linear regression model

POLYNOMIAL REGRESSION: A polynomial regression model with degree 2 is built, extending the linear regression model by adding polynomial features. The model is trained, and predictions are made for the test dataset. The performance metrics, such as MAE, R2 score, and accuracy, are calculated and compared with the linear regression model.



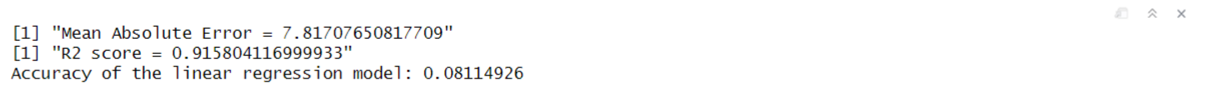
```
# Fit a linear regression model using the polynomial features
poly_regressor <- lm(y_train ~ ., data = X_train)
```

Figure 12: R code used for analysis for polynomial regression



```
[1] "Mean Absolute Error = 8.86095034687036"
[1] "R2 score = 0.905086061742227"
Accuracy of the linear regression model: 0.06433076
```

Figure 13: Mean Absolute Error, R-squared value and accuracy for the linear regression model



```
[1] "Mean Absolute Error = 7.81707650817709"
[1] "R2 score = 0.915804116999933"
Accuracy of the linear regression model: 0.08114926
```

Figure 14: Mean Absolute Error, R-squared value and accuracy for the polynomial regression model

Graphical representations of the actual and predicted emissions for both models are plotted to visualize their performance. Using ggplot2, scatter plots are created that compare the actual and predicted emissions for the linear and polynomial regression models. The true emissions are plotted along the x-axis, while the predicted emissions are plotted along the y-axis. Data points for the linear regression model are colored blue, and those for the polynomial regression model are colored red.

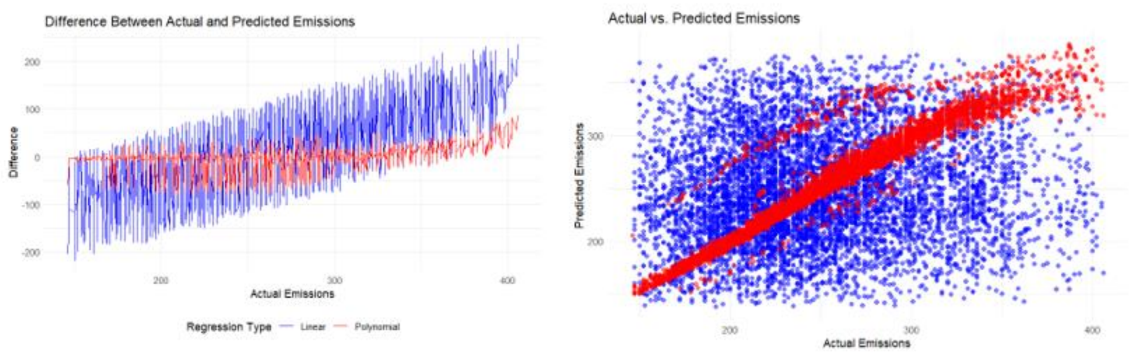
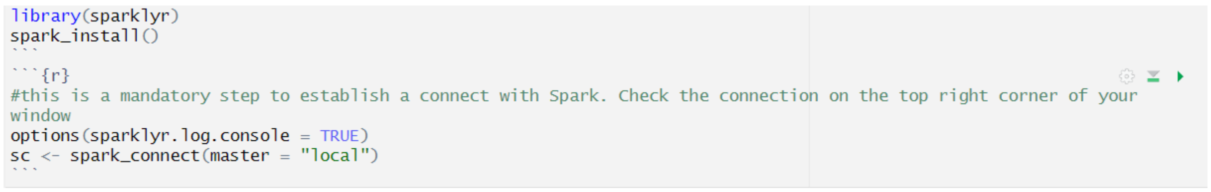


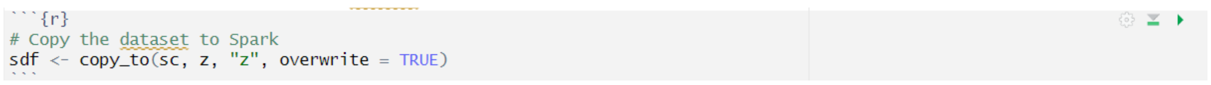
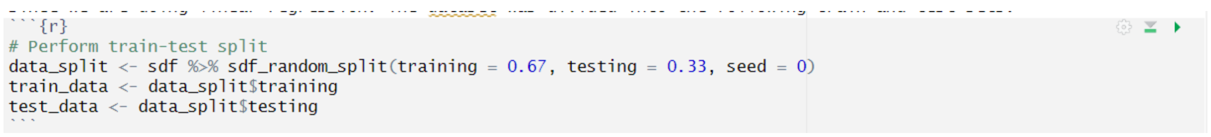
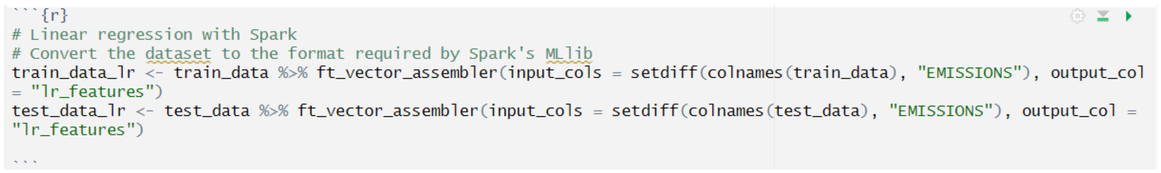
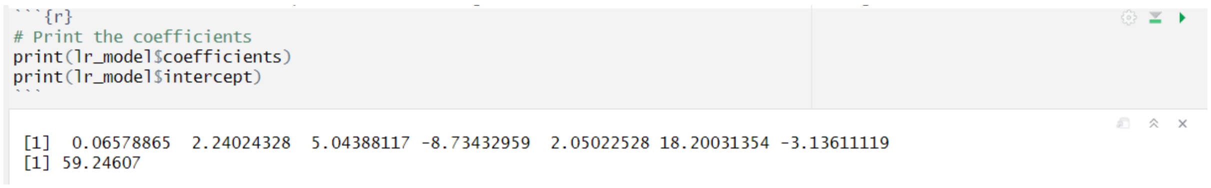
Figure 15: Comparison of linear regression model with polynomial regression model

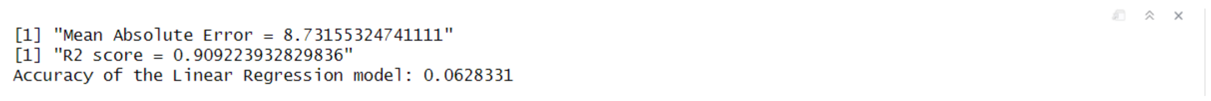
This visualization allows for an easy comparison of the models' performance. As seen in the plot, the polynomial regression model's predictions are generally closer to the actual emissions, indicating better performance in predicting vehicle emissions compared to the linear regression model.

HPCI (HIGH PERFORMANCE COMPUTATIONAL IMPLEMENTATION)

In this code, a High-Performance Computational technique for distributed data analysis using Apache Spark is implemented. Before proceeding, make sure to install the proper Java version (8 or above) and the Spark package in R.



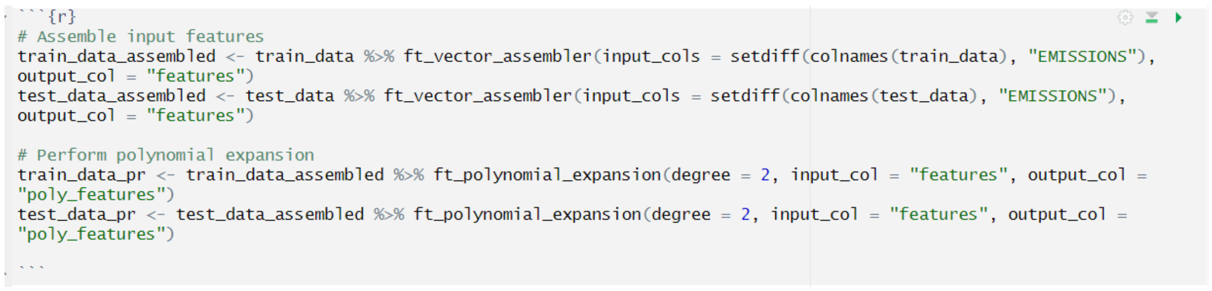
```
library(sparklyr)
spark_install()
```
```{r}
#this is a mandatory step to establish a connect with Spark. Check the connection on the top right corner of your
window
options(sparklyr.log.console = TRUE)
sc <- spark_connect(master = "local")
```
```

Figure 16: R code to install spark library and connect with spark cluster

The `sparklyr` library is loaded, and the Spark installation is done using the `spark_install()` function. The connection between R and Spark is established using `spark_connect()`. The dataset, named `z`, containing all numerical data without outliers is imported into the Spark cluster for further distributed data analysis using the `copy_to()` function.



```
```{r}
# Copy the dataset to Spark
sdf <- copy_to(sc, z, "z", overwrite = TRUE)
```
```

Figure 17: R code which imports the dataset free from outliers to the spark cluster

The dataset is divided into train and test sets using `sdf_random_split()`. Next, the data is converted into a format compatible with Spark MLlib using `ft_vector_assembler()`. The dependent variable, EMISSIONS, is omitted from the dataset to find the correlation between all independent variables.



```
```{r}
# Perform train-test split
data_split <- sdf %>% sdf_random_split(training = 0.67, testing = 0.33, seed = 0)
train_data <- data_split$training
test_data <- data_split$testing
```
```

Figure 18: R code in spark cluster to split the dataset into test and train set for linear and polynomial regression

A **linear regression** model based on Spark MLlib is trained with the `ml_linear_regression()` function. The coefficients and intercept of the linear regression model are printed, and predictions are made using the `ml_predict()` function. The mean absolute error (MAE) and R2 score are calculated using the `ml_regression_evaluator()` function.



```
```{r}
# Linear regression with Spark
# Convert the dataset to the format required by Spark's MLlib
train_data_lr <- train_data %>% ft_vector_assembler(input_cols = setdiff(colnames(train_data), "EMISSIONS"), output_col
= "lr_features")
test_data_lr <- test_data %>% ft_vector_assembler(input_cols = setdiff(colnames(test_data), "EMISSIONS"), output_col =
"lr_features")

```
```

Figure 19: R code to transform the dataset into a format compatible with Spark MLib



```
```{r}
# Print the coefficients
print(lr_model$coefficients)
print(lr_model$intercept)
```
```

```
[1]  0.06578865  2.24024328  5.04388117 -8.73432959  2.05022528 18.20031354 -3.13611119
[1] 59.24607
```

Figure 20: Coefficient and intercept of Linear regression model using Spark
```



```
[1] "Mean Absolute Error = 8.73155324741111"
[1] "R2 score = 0.909223932829836"
Accuracy of the Linear Regression model: 0.0628331
```

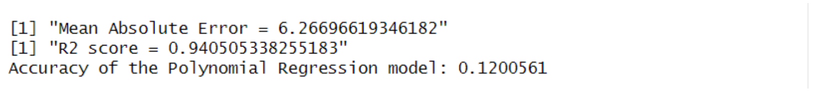
Figure 21: Mean Absolute Error, R-squared value and accuracy for the linear regression model

For the **polynomial regression** model, the train and test data are prepared in a similar manner. A polynomial expansion with a degree of 2 is applied using `ft_polynomial_expansion()`. The `ml_linear_regression()` function is used again, this time with the polynomial features as input. Predictions are made using the `ml_predict()` function, and the MAE and R2 score are calculated using the `ml_regression_evaluator()` function.



~~~
```{r}
# Assemble input features
train_data_assembled <- train_data %>% ft_vector_assembler(input_cols = setdiff(colnames(train_data), "EMISSIONS"),
output_col = "features")
test_data_assembled <- test_data %>% ft_vector_assembler(input_cols = setdiff(colnames(test_data), "EMISSIONS"),
output_col = "features")

# Perform polynomial expansion
train_data_pr <- train_data_assembled %>% ft_polynomial_expansion(degree = 2, input_col = "features", output_col =
"poly_features")
test_data_pr <- test_data_assembled %>% ft_polynomial_expansion(degree = 2, input_col = "features", output_col =
"poly_features")

```
~~~

Figure 22: R code to assemble input features and perform polynomial expansion in Spark for polynomial regression



```
[1] "Mean Absolute Error = 6.26696619346182"
[1] "R2 score = 0.940505338255183"
Accuracy of the Polynomial Regression model: 0.1200561
```

Figure 23: Mean Absolute Error, R-squared value and accuracy for the polynomial regression model

Finally, the accuracies of both the linear regression and polynomial regression models are calculated. This is done by creating a new column in the predictions dataframe to identify correct predictions, calculating the total number of correct predictions and the total number of predictions, and then finding the ratio of correct predictions to the total predictions. The calculated accuracies for both models are printed.

leveraging the power of Spark MLlib and R in implementing linear and polynomial regression models provides an efficient and scalable solution for analyzing large datasets. Comparing the performance metrics of these models enables data scientists and researchers to make informed decisions on which model best suits their specific use case. As data sizes continue to grow and become more complex, the integration of Spark MLlib with R offers a robust platform for tackling various data-driven challenges. Additionally, it's crucial to remember that model selection is not a one-size-fits-all approach. Depending on the nature of the data and the problem at hand, it may be necessary to explore other machine learning algorithms and techniques. By staying informed and adaptable, you can ensure that you are equipped to handle the rapidly evolving world of data analysis and make the most accurate predictions possible.

COMPARISON OF RESULTS

In this model study, the performance of machine learning models was compared using both traditional R-based libraries and Spark-based libraries, focusing on linear regression and polynomial regression models.

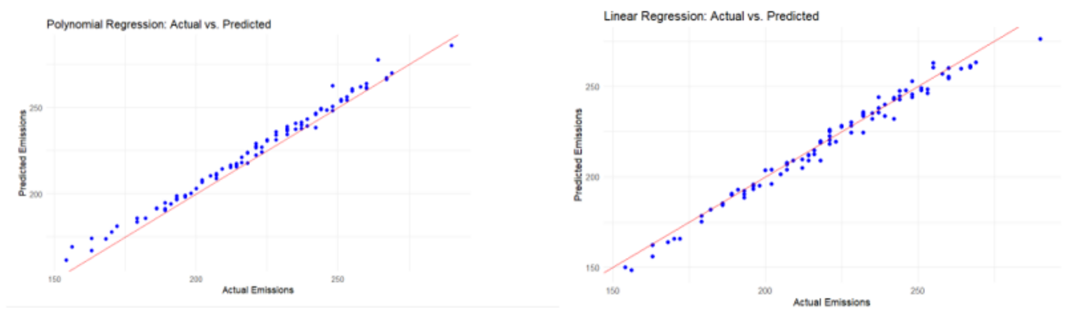


Figure 24: Comparison of Actual vs Predicted values for linear regression model with polynomial regression model in Spark

For the traditional R-based approach, linear regression and polynomial regression models were constructed using the caret library. The linear regression model achieved an accuracy of 0.06433076, a mean absolute error of 8.86095034687036, and an R-squared value of 0.905086061742227. The polynomial regression model reached an accuracy of 0.08114926 and a mean absolute error of 7.81707650817709, with an R-squared value of 0.915804116999933.

For the Spark-based approach, linear regression and polynomial regression models were built using the ml_linear_regression function in the sparklyr library. The linear regression model attained an accuracy of 0.0628331, a mean absolute error of 8.73155324741111, and an R-squared value of 0.909223932829836. The polynomial regression model achieved an accuracy of 0.1200561 and a mean absolute error of 6.26696619346182, with an R-squared value of 0.940505338255183.

Upon comparing the results of both approaches, it is evident that the accuracy, mean absolute error, and R-squared values of the linear regression model are quite similar in each case. However, the Spark-based polynomial regression model demonstrated a significant improvement in performance compared to the traditional R-based polynomial regression model. The mean absolute error is considerably lower, the R-squared score is higher, and the accuracy is noticeably better in the Spark-based model.

In conclusion, the traditional machine learning approach and the Spark-based approach can yield similar results for linear regression models, but the Spark-based approach may be more effective for polynomial regression models. Nevertheless, the Spark-based approach offers the advantage of scalability, rendering it a more appropriate choice for larger datasets that cannot be processed in memory.

DISCUSSIONS

This project aimed to compare the performance of machine learning models developed using traditional R-based libraries and Spark-based libraries, focusing specifically on linear regression and polynomial regression models. The comparison was carried out on the same dataset, allowing for a fair evaluation of the approaches' effectiveness and accuracy. The results provide valuable insights into the strengths and weaknesses of each method and offer guidance for selecting the most suitable approach for a given problem.

One key finding from the project is that the traditional R-based and Spark-based approaches produce comparable results for linear regression models. Both methods achieve similar accuracy, mean absolute error, and R-squared values, indicating that the choice between the two approaches for linear regression problems might be driven by factors other than performance, such as dataset size, computational resources, or familiarity with the libraries.

However, the Spark-based approach demonstrated a clear advantage for polynomial regression models, outperforming the traditional R-based method in terms of accuracy, mean absolute error, and R-squared value. This suggests that for more complex regression problems, the Spark-based approach may be more suitable due to its ability to deliver improved model performance.

Another important consideration is the scalability offered by the Spark-based approach. As datasets grow in size, traditional R-based methods may struggle to process the data in memory, resulting in increased computational time or even the inability to perform the analysis altogether. In contrast, the Spark-based approach can handle large-scale datasets more efficiently, making it a more appropriate choice for big data applications.

Despite the advantages of the Spark-based approach, it is crucial to recognize the learning curve associated with adopting a new library and adapting existing workflows to a distributed computing environment. For data scientists familiar with traditional R-based methods, there may be a period of adjustment required to become proficient with Spark-based libraries. Nevertheless, the benefits of scalability and improved performance for more complex regression problems may justify the investment in learning these new tools.

In conclusion, this project highlights the importance of selecting the appropriate machine learning approach based on the problem's complexity and dataset size. While traditional R-based methods may be sufficient for simple regression problems or smaller datasets, the Spark-based approach can offer improved performance and scalability for more complex problems or large-scale datasets. By considering these factors, data scientists can make informed decisions when choosing the best approach for their specific needs.