Part 1: Linear Regression, kNN Regression, Decision Trees Regression

Group 6: Jiarui Groves, Md Hassan, Frederick Horn, Sanjana Jadhav, Anushka Karthikeyan
Part 1: Regression Notebook (Sanjana Jadhav)

This example looks at the **Fuel Consumption 2000-2022** data set found on Kaggle. Here is the **LINK**! Other Sources: **PREVIOUS HW** (some explanations of functions), **PROFESSOR'S GITHUB REPO**, Slides, & Textbook

This notebook will use Linear Regression, kNN Regression, and Decision Tree Regression to analyze the impact of various predictors on CO2 Emissions and compare the performance of these three algorithms.

Data Exploration

read.csv() Function

The function "read.csv" is utilized to read the contents of a file, with the file path provided as input, and store it in a data frame named "df".

```
df <- read.csv("Fuel_Consumption_2000-2022.csv")
```

Data Cleaning

This code applies the "sapply()" function to the data frame "df" to execute a function on all the columns. * "function(x)" integrates "sum()" and "is.na()" to calculate the number of missing values in each column. * The final result is a vector displaying the missing values for all the columns in "df". –> there are no missing values

```
sapply(df,function(x) sum(is.na(x)==TRUE))
```

```
##             YEAR             MAKE            MODEL    VEHICLE.CLASS
##                0                0                0                0
##      ENGINE.SIZE        CYLINDERS     TRANSMISSION             FUEL
##                0                0                0                0
## FUEL.CONSUMPTION   HWY..L.100.km.  COMB..L.100.km.       COMB..mpg.
##                0                0                0                0
##        EMISSIONS
##                0
```

str() Function

The "str()" function is used to display the structure of "df." We can see the column names and their data types.

```
str(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 ...
##  $ HWY..L.100.km.  : num  6.7 6.5 7.4 9.2 7 6.8 7 9.2 9.1 7.2 ...
##  $ COMB..L.100.km. : num  8.1 7.6 10 11.5 8.6 8.2 8.3 11.6 11.7 9.5 ...
##  $ COMB..mpg.      : int  35 37 28 25 33 34 34 24 24 30 ...
##  $ EMISSIONS       : int  186 175 230 264 198 189 191 267 269 218 ...
```

factor() Function

To make it easier to represent categories, the "as.factor()" function is utilized to convert a column's data type into a factor variable. The following variables will be converted to factors to better represent their categories:
YEAR, MAKE, MODEL, VEHICLE.CLASS, CYLINDERS, TRANSMISSION, FUEL

```
df$YEAR <- as.factor(df$YEAR)
df$MAKE <- as.factor(df$MAKE)
df$MODEL <- as.factor(df$MODEL)
df$VEHICLE.CLASS <- as.factor(df$VEHICLE.CLASS)
df$CYLINDERS <- as.factor(df$CYLINDERS)
df$TRANSMISSION <- as.factor(df$TRANSMISSION)
df$FUEL <- as.factor(df$FUEL)
```

Divide into Train/Test (80/20)

- set.seed(1234): ensures that the train/test data remains the same across multiple runs of the code
- 80% of df = training
- 20% of df = testing
- replace=FALSE ensures that the data points in the training and testing sets do not overlap

```
set.seed(1234)
i <- sample(1:nrow(df), nrow(df)*0.80, replace=FALSE)
train <- df[i,]
test <- df[-i,]
```

summary() Function

- The purpose of the "summary()" function is to present statistical information regarding the train data frame, including metrics like the maximum/minimum for numerical data types and the number of occurrences for each category in factor variables.

- We can utilize str() once more to verify whether these columns have been converted into factor variables: YEAR, MAKE, MODEL, VEHICLE.CLASS, CYLINDERS, TRANSMISSION, FUEL

```
summary(train)
```

```
##       YEAR                  MAKE              MODEL
##  2016   :  893   CHEVROLET    : 1388   JETTA     :   70
##  2008   :  880   BMW          : 1192   MUSTANG   :   69
##  2015   :  880   GMC          : 1072   RANGER    :   50
##  2009   :  870   FORD         : 1019   SENTRA    :   50
##  2011   :  870   MERCEDES-BENZ:  736   A4 QUATTRO:   45
##  2012   :  870   DODGE        :  644   SILVERADO :   45
##  (Other):12781   (Other)      :11993   (Other)   :17715
##                  VEHICLE.CLASS   ENGINE.SIZE      CYLINDERS     TRANSMISSION
##  COMPACT               :2113   Min.   :0.800   4      :6474   A4     :2821
##  SUV                   :2113   1st Qu.:2.300   6      :6423   AS6    :2264
##  MID-SIZE              :1837   Median :3.000   8      :4136   M6     :2069
##  PICKUP TRUCK - STANDARD:1346   Mean   :3.353   5      : 393   M5     :1685
##  SUBCOMPACT            :1257   3rd Qu.:4.200   12     : 354   A6     :1582
##  FULL-SIZE             : 866   Max.   :8.400   10     : 121   AS8    :1380
##  (Other)               :8512                   (Other): 143   (Other):6243
##  FUEL     FUEL.CONSUMPTION HWY..L.100.km.   COMB..L.100.km.   COMB..mpg.
##  D: 250   Min.   : 3.50    Min.   : 3.200   Min.   : 3.60   Min.   :11.0
##  E: 868   1st Qu.:10.40    1st Qu.: 7.300   1st Qu.: 9.10   1st Qu.:22.0
##  N:  29   Median :12.30    Median : 8.400   Median :10.50   Median :27.0
##  X:9525   Mean   :12.75    Mean   : 8.914   Mean   :11.03   Mean   :27.4
##  Z:7372   3rd Qu.:14.70    3rd Qu.:10.200   3rd Qu.:12.70   3rd Qu.:31.0
##           Max.   :30.60    Max.   :20.900   Max.   :26.10   Max.   :78.0
##
##    EMISSIONS
##  Min.   : 83.0
##  1st Qu.:209.0
##  Median :242.0
##  Mean   :249.8
##  3rd Qu.:288.0
##  Max.   :608.0
##
```

```
str(df)
```

```
## 'data.frame':    22556 obs. of  13 variables:
##  $ YEAR           : Factor w/ 23 levels "2000","2001",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ MAKE           : Factor w/ 87 levels "Acura","ACURA",..: 2 2 2 2 2 2 2 2 2 8 ...
##  $ MODEL          : Factor w/ 4242 levels "1 SERIES M COUPE",..: 2 2 62 63 2182 2182 2184 2724 2724
##  $ VEHICLE.CLASS  : Factor w/ 32 levels "Compact","COMPACT",..: 2 2 6 6 22 22 22 22 22 2 ...
##  $ ENGINE.SIZE    : num  1.6 1.6 3.2 3.5 1.8 1.8 1.8 3 3.2 1.8 ...
##  $ CYLINDERS      : Factor w/ 9 levels "2","3","4","5",..: 3 3 5 5 3 3 3 5 5 3 ...
```

```
##  $ TRANSMISSION    : Factor w/ 30 levels "A10","A3","A4",..: 3 28 16 3 3 28 28 15 29 4 ...
##  $ FUEL            : Factor w/ 5 levels "D","E","N","X",..: 4 4 5 5 4 4 5 5 5 5 ...
##  $ FUEL.CONSUMPTION: num  9.2 8.5 12.2 13.4 10 9.3 9.4 13.6 13.8 11.4 ...
##  $ HWY..L.100.km.  : num  6.7 6.5 7.4 9.2 7 6.8 7 9.2 9.1 7.2 ...
##  $ COMB..L.100.km. : num  8.1 7.6 10 11.5 8.6 8.2 8.3 11.6 11.7 9.5 ...
##  $ COMB..mpg.      : int  35 37 28 25 33 34 34 24 24 30 ...
##  $ EMISSIONS       : int  186 175 230 264 198 189 191 267 269 218 ...
```

table() Function

The "table()" function is utilized to display the distinct categories present within a vector, along with their corresponding frequencies.
An example is shown below for the FUEL column.
* X = Regular gasoline
* Z = Premium gasoline
* D = Diesel
* E = Ethanol (E85)
* N = Natural Gas

```
table(train$FUEL)
```

```
##
##    D    E    N    X    Z
##  250  868   29 9525 7372
```

mean() Function

Here, the "mean()" function is used to find the average of the CO2 emissions (g/km3) column.

```
mean(train$EMISSIONS)
```

```
## [1] 249.8008
```

cor() Function

The "cor()" function finds the correlation between two variables in "df".
Here, we are trying to calculate if the "ENGINE SIZE" of a car is correlated to the amount of "EMISSIONS" produced.

```
cor(train$ENGINE.SIZE, train$EMISSIONS)
```

```
## [1] 0.8220037
```

These variables are strongly and positively correlated as 0.8220037 is close to 1.

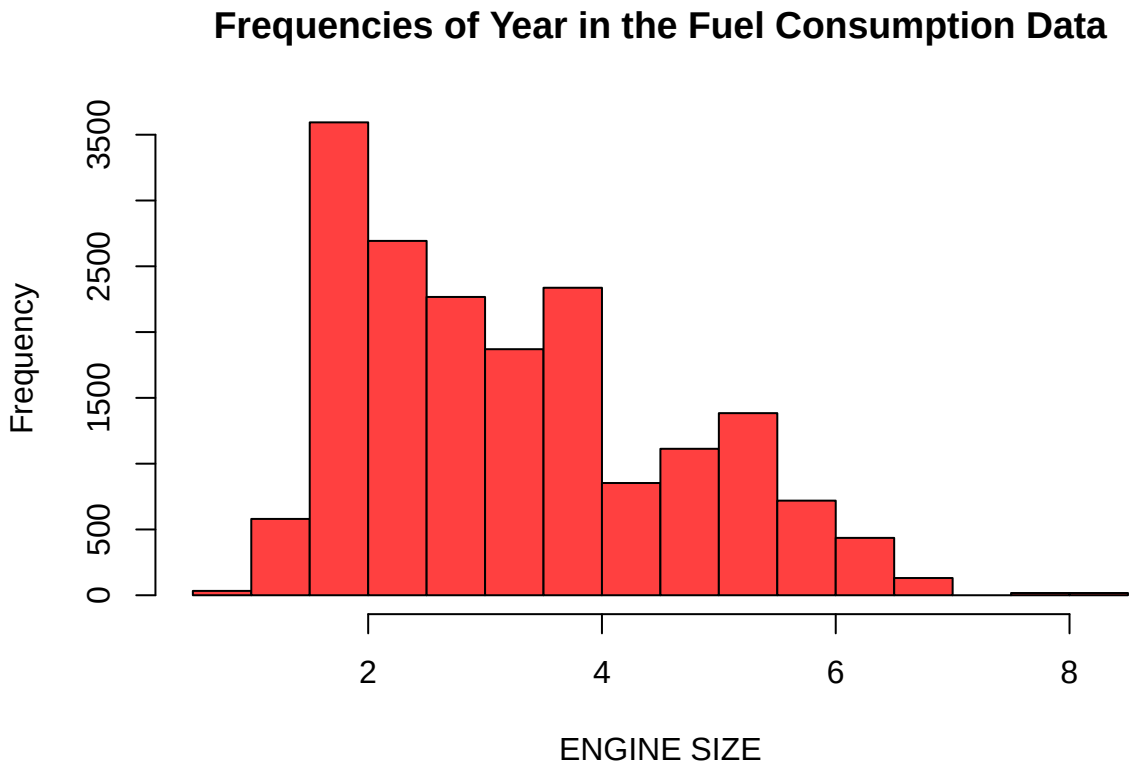
Data Visualization

Data Visualization assists us in identifying patterns within the data.

hist() Function

For instance, this is a **histogram** that shows the frequencies of different **ENGINE SIZES** in our data. As we can see, the data is a bit skewed to the right with the average being 3.353414.

```
hist(train$ENGINE.SIZE, col="brown1",
     main="Frequencies of Year in the Fuel Consumption Data", xlab="ENGINE SIZE")
```




```
mean(train$ENGINE.SIZE)
```

```
## [1] 3.353414
```

pie() Function

We can start by using the "table()" function to inspect the different values in the **CYLINDERS** column and their corresponding frequencies.

```
table(train$CYLINDERS)
```

```
##
##    2    3    4    5    6    8   10   12   16
##   15  117 6474  393 6423 4136  121  354   11
```

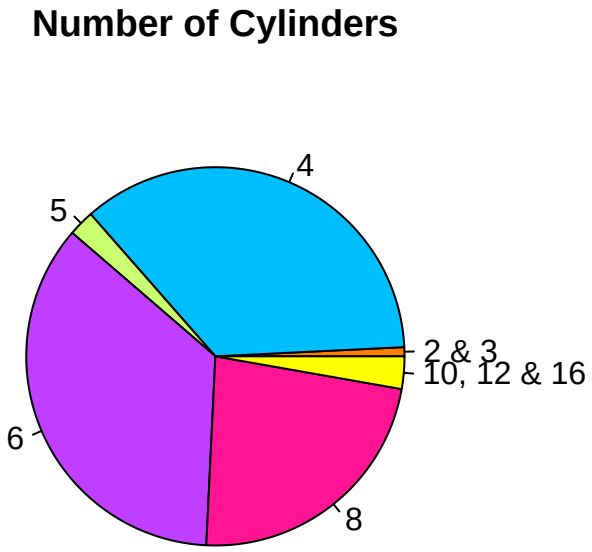
- We create a vector called "slices" to create sections of the pie chart. In this case, we are combining "2 & 3" and "10, 12 & 16" as they hold relatively low percentages of the pie chart.

- Next, we define another vector called “lbls” to label each of the sections of the pie chart.
- Finally, we use the “pie()” function to create the pie chart.

```
slices <- c(sum(df$CYLINDERS %in% c(2, 3), na.rm = TRUE),
            sum(df$CYLINDERS==4, na.rm = TRUE),
            sum(df$CYLINDERS==5, na.rm = TRUE),
            sum(df$CYLINDERS==6, na.rm = TRUE),
            sum(df$CYLINDERS==8, na.rm = TRUE),
            sum(df$CYLINDERS %in% c(10, 12, 16), na.rm = TRUE))

lbls <- c("2 & 3", "4", "5", "6", "8", "10, 12 & 16")

pie(slices, labels=lbls, main="Number of Cylinders", col=c("darkorange1", "deepskyblue1",
                                                           "darkolivegreen1",
                                                           "darkorchid1",
                                                           "deeppink1",
                                                           "yellow"))
```



Linear Regression

- this algorithm tries to find a linear “best-fit line” that shows how predictors influence a target variable.

Simple Linear Regression Model

- uses only 1 predictor

Building the Linear Model

In the code below, we are using the "train" data to create a Simple Linear Regression model named "lm1" to see how **FUEL CONSUMPTION** can be used to predict **EMISSIONS**.

```
lm1 <- lm(EMISSIONS~FUEL.CONSUMPTION, data=train)
summary(lm1)
```

```
##
## Call:
## lm(formula = EMISSIONS ~ FUEL.CONSUMPTION, data = train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -134.997   -7.235    0.585   12.212   84.793
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)      51.08890    0.65953   77.46   <2e-16 ***
## FUEL.CONSUMPTION 15.58147    0.04987  312.45   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 23.46 on 18042 degrees of freedom
## Multiple R-squared:  0.844,  Adjusted R-squared:  0.844
## F-statistic: 9.763e+04 on 1 and 18042 DF,  p-value: < 2.2e-16
```

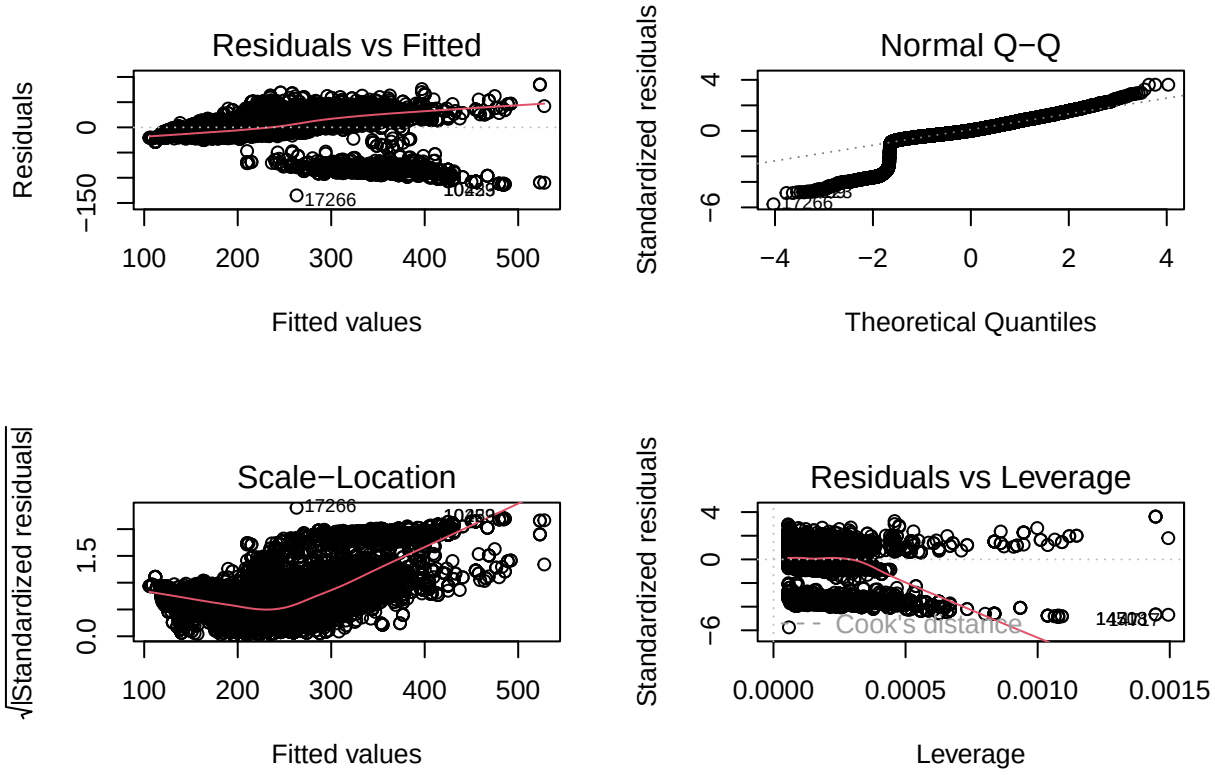
Interpretation of the Summary:

- **Residuals:** The residuals measure the distance between the observed data points and the predicted values from the regression line. The min is -134.997 and the max is 84.793.
- **Coefficients:** The y-intercept is 51.08890 and for every 1 unit of increase (L/100km) in **FUEL CONSUMPTION**, CO2 **EMISSIONS** increased by 15.58147 g/km3.
- **Residual Standard Error:** The RSE is 23.46, which shows us the model is a good fit as the value is very low.
- **Multiple R-squared & Adjusted R-Squared:** Both values are 0.844 as there is only one predictor. 84% of the variance in this model can be explained by the predictor.
- **F-Statistic:** The F-statistic tests for the presence of a significant relationship between the predictor and the target Ideally, a higher F-statistic value indicates that the model is a better fit for the data. In this case, the F-statistic of 9.763e+04 is very high, which suggests that the model is a good fit for the data.
- **P-Value:** Our p-value is very low, which means we can reject the null-hypothesis and our predictor significantly affects our target.

Plotting the Residuals

- par(mfrow=c(2,2)): prints 4 plots in one window
- plot() function is used to plot the linear model

```
par(mfrow=c(2,2))
plot(lm1)
```



Interpretation of the Residual Plots:

- **Residuals vs. Fitted**: The goal is to have a relatively horizontal line and our plot demonstrates a linear relationship between **FUEL CONSUMPTION** and CO2 **EMISSIONS**.
- **Normal Q-Q**: To indicate a normal distribution, it is necessary to have a fairly straight line, but our plot shows a slight deviation from this line.
- **Scale-Location**: We want to see a relatively horizontal line with points evenly dispersed around it. However, even though the points appear evenly distributed around the line, the line itself is not horizontal, which suggests the absence of a pattern.
- **Residuals vs. Leverage**: This enables us to identify any points that may influence our regression line, but there are no such points present in our plot.

Multiple Linear Regression Model

- uses multiple predictors

Building the Linear Model

In the code below, we are using the "train" data to create a Multiple Linear Regression model named "lm2" to see how **FUEL CONSUMPTION, ENGINE SIZE, CYLINDERS, & FUEL** can be used to predict **EMISSIONS**.

```
lm2 <- lm(EMISSIONS~FUEL.CONSUMPTION + ENGINE.SIZE + CYLINDERS + FUEL, data=train)
summary(lm2)
```

```
##
## Call:
## lm(formula = EMISSIONS ~ FUEL.CONSUMPTION + ENGINE.SIZE + CYLINDERS +
##     FUEL, data = train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -65.360  -6.146  -1.003   5.110  56.686
##
## Coefficients:
##                    Estimate Std. Error  t value Pr(>|t|)
## (Intercept)        42.98364    2.49271   17.244  < 2e-16 ***
## FUEL.CONSUMPTION   19.25195    0.04231  454.993  < 2e-16 ***
## ENGINE.SIZE        -0.23073    0.16155   -1.428  0.15324
## CYLINDERS3          5.17569    2.53652    2.040  0.04132 *
## CYLINDERS4          1.66657    2.39195    0.697  0.48597
## CYLINDERS5         -4.25695    2.43629   -1.747  0.08060 .
## CYLINDERS6         -0.31061    2.40406   -0.129  0.89720
## CYLINDERS8         -1.42380    2.44994   -0.581  0.56114
## CYLINDERS10        -7.04313    2.61012   -2.698  0.00697 **
## CYLINDERS12        -9.95152    2.53476   -3.926 8.67e-05 ***
## CYLINDERS16        -0.86730    3.80066   -0.228  0.81949
## FUELE            -145.92545    0.72383 -201.601  < 2e-16 ***
## FUELN             -94.63991    1.82562  -51.840  < 2e-16 ***
## FUELX             -31.41870    0.59555  -52.755  < 2e-16 ***
## FUELZ             -34.34719    0.59910  -57.332  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 9.214 on 18029 degrees of freedom
## Multiple R-squared:  0.976,  Adjusted R-squared:  0.9759
## F-statistic: 5.229e+04 on 14 and 18029 DF,  p-value: < 2.2e-16
```

Interpretation of the Summary:

- **Residuals:** The residual measures the distance between the observed data points and the predicted values from the regression line. The min is -65.360 and the max is 56.686.

- **Coefficients:** The y-intercept is 42.98364 and we can see how different predictors impact CO2 **EMISSIONS** by looking at their corresponding coefficients.

- **Residual Standard Error:** The RSE is 9.214, which shows us the model is a good fit as the value is very low.

- **Multiple R-squared & Adjusted R-Squared:** Multiple R-squared is 0.976 & Adjusted R-Squared is 0.9759. Almost 98% of the variance in this model can be explained by the predictor.

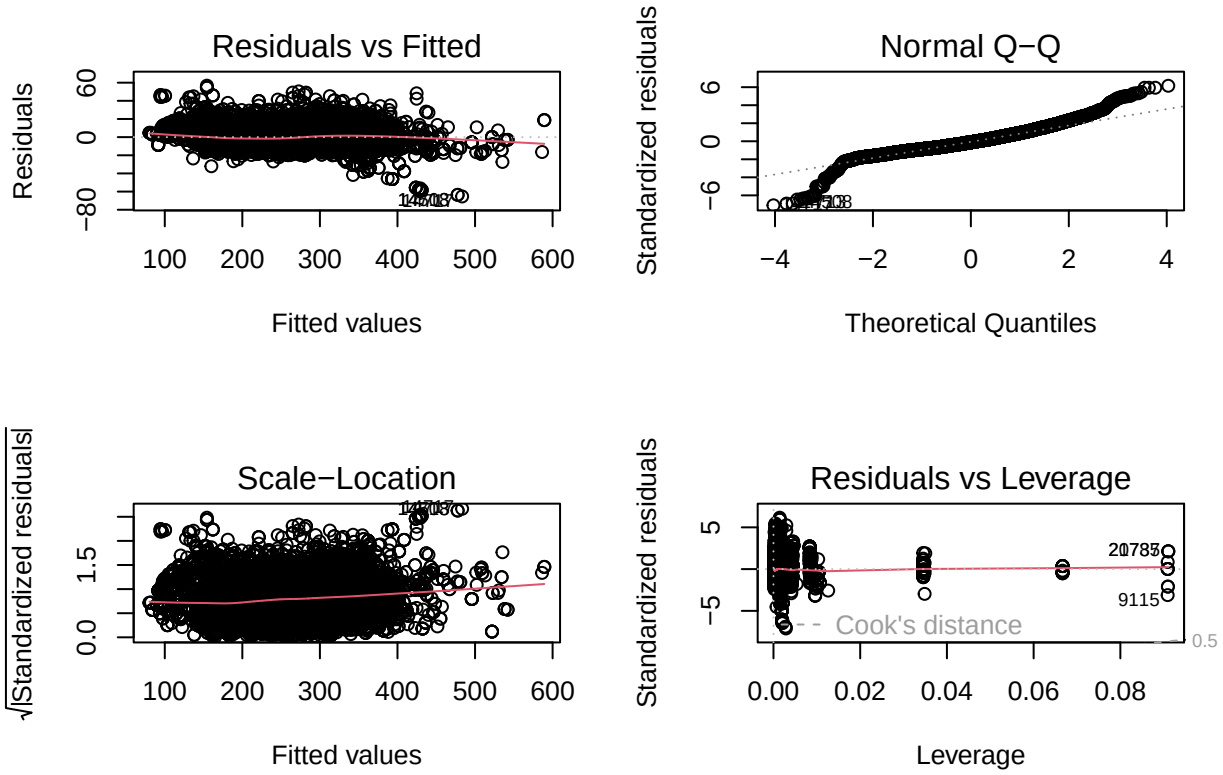
- **F-Statistic:** The F-statistic tests for the presence of a significant relationship between the predictors and the target Ideally, a higher F-statistic value indicates that the model is a better fit for the data. In this case, the F-statistic of 5.229e+04 is very high, which suggests that the model is a good fit for the data.

- **P-Value:** Our p-value is very low for **FUEL CONSUMPTION, CYLINDERS (10, 12), and FUEL (all categories).** This means we can reject the null-hypothesis and these predictors significantly affects our target.

Plotting the Residuals

- par(mfrow=c(2,2)): prints 4 plots in one window
- plot() function is used to plot the linear model

```
par(mfrow=c(2,2))
plot(lm2)
```



Interpretation of the Residual Plots:

- **Residuals vs. Fitted**: The goal is to have a relatively horizontal line and our plot demonstrates a linear relationship between **FUEL CONSUMPTION, ENGINE SIZE, CYLINDERS, & FUEL** and CO2 **EMISSIONS**.
- **Normal Q-Q**: To indicate a normal distribution, it is necessary to have a fairly straight line and our plot demonstrates that.
- **Scale-Location**: We want to see a relatively horizontal line with points evenly dispersed around it and our plot indicates that this is the case.
- **Residuals vs. Leverage**: This enables us to identify any points that may influence our regression line. As we can see, some points are influencing the regression line on the right. **Overall, this model performed better than the previous model!**

Let's change our predictors!

Building the Linear Model

In the code below, we are using the "train" data to create a Multiple Linear Regression model named "lm3" to see how **FUEL CONSUMPTION, COMB..L.100.km., & FUEL** can be used to predict **EMISSIONS**.

```
lm3 <- lm(EMISSIONS~FUEL.CONSUMPTION + COMB..L.100.km. + FUEL, data=train)
summary(lm3)
```

```
##
## Call:
## lm(formula = EMISSIONS ~ FUEL.CONSUMPTION + COMB..L.100.km. +
##     FUEL, data = train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -64.482  -1.831  -0.904   1.376  51.608
##
## Coefficients:
##                    Estimate Std. Error  t value Pr(>|t|)
## (Intercept)        35.61291    0.35616   99.991  < 2e-16 ***
## FUEL.CONSUMPTION   -0.59621    0.09727   -6.129 9.02e-10 ***
## COMB..L.100.km.    23.40532    0.11685  200.309  < 2e-16 ***
## FUELE            -145.76598    0.39346 -370.476  < 2e-16 ***
## FUELN             -99.35436    1.02744  -96.701  < 2e-16 ***
## FUELX             -31.39224    0.33440  -93.875  < 2e-16 ***
## FUELZ             -30.67221    0.33775  -90.814  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.2 on 18037 degrees of freedom
## Multiple R-squared:  0.9923, Adjusted R-squared:  0.9923
## F-statistic: 3.894e+05 on 6 and 18037 DF,  p-value: < 2.2e-16
```

Interpretation of the Summary:

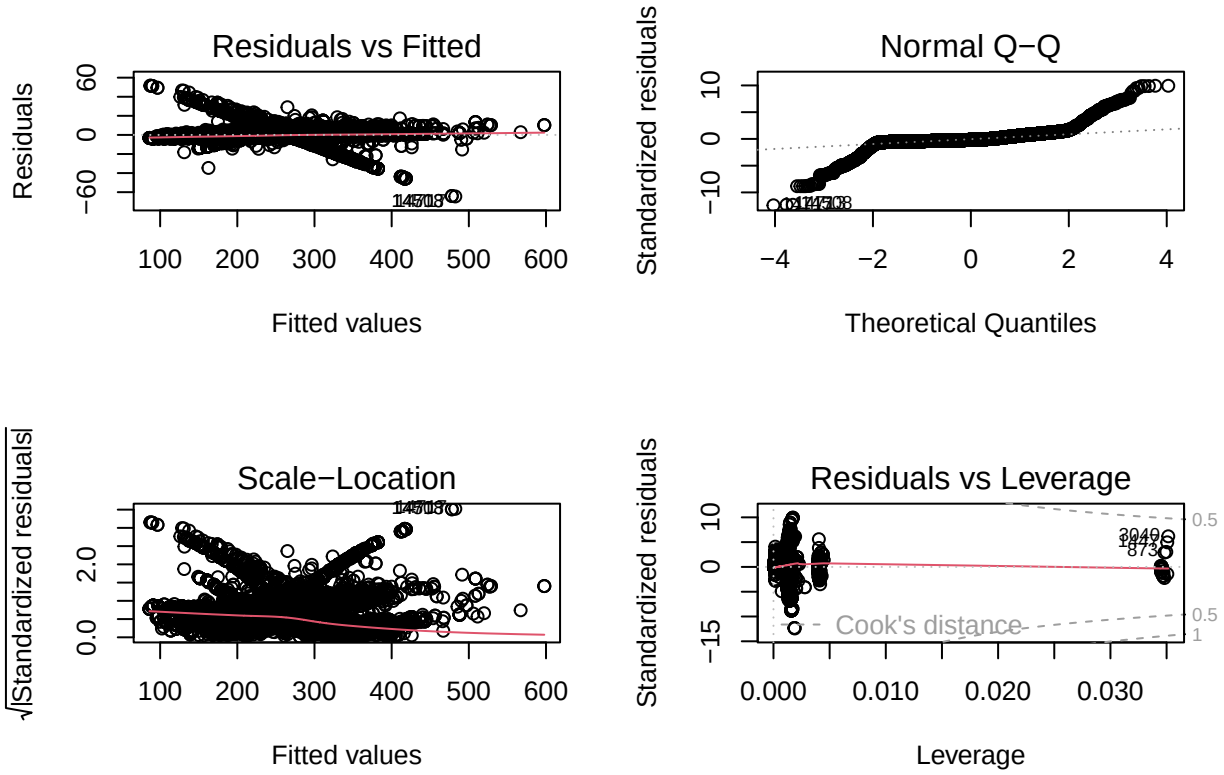
- **Residuals:** The residual measures the distance between the observed data points and the predicted values from the regression line. The min is -64.482 and the max is 51.608.

- **Coefficients:** The y-intercept is 35.61291 and we can see how different predictors impact CO2 **EMISSIONS** by looking at their corresponding coefficients.
- **Residual Standard Error:** The RSE is 5.2, which shows us the model is a good fit as the value is very low.
- **Multiple R-squared & Adjusted R-Squared:** Both values are 0.9923. 99% of the variance in this model can be explained by the predictor.
- **F-Statistic:** The F-statistic tests for the presence of a significant relationship between the predictor and the target Ideally, a higher F-statistic value indicates that the model is a better fit for the data. In this case, the F-statistic of 3.894e+05 is very high, which suggests that the model is a good fit for the data.
- **P-Value:** Our p-value is very low for all predictors. This means we can reject the null-hypothesis and these predictors significantly affects our target.

Plotting the Residuals

- par(mfrow=c(2,2)): prints 4 plots in one window
- plot() function is used to plot the linear model

```
par(mfrow=c(2,2))
plot(lm3)
```



Interpretation of the Residual Plots:

- **Residuals vs. Fitted**: The goal is to have a relatively horizontal line and our plot demonstrates a linear relationship between **FUEL CONSUMPTION, COMB..L.100.km., & FUEL** and CO2 **EMISSIONS**.

- **Normal Q-Q**: To indicate a normal distribution, it is necessary to have a fairly straight line, but our plot shows a significant deviation from this line.

- **Scale-Location**: We want to see a relatively horizontal line with points evenly dispersed around it. However, even though the line is horizontal, the points are not evenly spread out, which suggests the absence of a pattern.

- **Residuals vs. Leverage**: This enables us to identify any points that may influence our regression line. As we can see, some points are influencing the regression line in the middle. **Overall, this model performed better than the previous 2 models!**

Summary

- **lm1:** simple linear regression model that uses **FUEL CONSUMPTION** to predict **EMISSIONS**.
- **lm2:** multiple linear regression model that uses **FUEL CONSUMPTION, ENGINE SIZE, CYLINDERS, & FUEL** to predict **EMISSIONS**.
- **lm3:** multiple linear regression model that uses **FUEL CONSUMPTION, COMB..L.100.km., & FUEL** to predict **EMISSIONS**.

  Overall, lm3 performed the best with the lowest Residual Standard Error (5.2), highest R-squared errors (0.9923), highest F-statistic value (3.894e+05), and low p-values for all predictors used in the linear model.

Evaluate on Test Data + Using Test Metrics

lm1

- **predictions**: Create a vector called "pred1" to store the predicted values. Call the "predict()" function with **lm1** and the **test vector** as input.

- **cor**: Shows us the correlation coefficient between predicted values and actual values of **EMISSIONS**. Our value is 0.920693906117968, which is close to 1 and indicates a strong and positive correlation.

- **mse**: Shows us the mean squared error (differences) between predicted values and actual values of **EMISSIONS**. Our value is 533.115826323111, which is very high and not preferred.

- **rmse**: Shows us the root mean squared error (differences) between predicted values and actual values of **EMISSIONS**. Our value is 23.0893011224487, which is very low and preferred. RMSE ignores outliers that may influence MSE.

```
pred1 <- predict(lm1, newdata = test)
cor1 <- cor(pred1, test$EMISSIONS)
print(paste('correlation for lm1:', cor1))
```

```
## [1] "correlation for lm1: 0.920693906117968"
```

```
mse1 <- mean((pred1-test$EMISSIONS)^2)
print(paste('mse for lm1:', mse1))
```

```
## [1] "mse for lm1: 533.115826323111"
```

```
rmse1 <- sqrt(mse1)
print(paste('rmse for lm1:', rmse1))
```

```
## [1] "rmse for lm1: 23.0893011224487"
```

lm2

- **predictions**: Create a vector called "pred2" to store the predicted values. Call the "predict()" function with **lm2** and the **test vector** as input.
- **cor**: Shows us the correlation coefficient between predicted values and actual values of **EMISSIONS**. Our value is 0.987795996255288, which is very close to 1 and indicates a strong and positive correlation.
- **mse**: Shows us the mean squared error (differences) between predicted values and actual values of **EMISSIONS**. Our value is 84.8851607209109, which is low and preferred.
- **rmse**: Shows us the root mean squared error (differences) between predicted values and actual values of **EMISSIONS**. Our value is 9.21331431792658, which is very low and preferred. RMSE ignores outliers that may influence MSE.
  The test metrics for lm2 performed better than lm1!

```
pred2 <- predict(lm2, newdata = test)
cor2 <- cor(pred2, test$EMISSIONS)
print(paste('correlation for lm2:', cor2))
```

```
## [1] "correlation for lm2: 0.987795996255288"
```

```
mse2 <- mean((pred2-test$EMISSIONS)^2)
print(paste('mse for lm2:', mse2))
```

```
## [1] "mse for lm2: 84.8851607209109"
```

```
rmse2 <- sqrt(mse2)
print(paste('rmse for lm2:', rmse2))
```

```
## [1] "rmse for lm2: 9.21331431792658"
```

lm3

- **predictions**: Create a vector called "pred3" to store the predicted values. Call the "predict()" function with **lm3** and the **test vector** as input.

- **cor**: Shows us the correlation coefficient between predicted values and actual values of **EMISSIONS**. Our value is 0.996362653202669, which is extremely close to 1 and indicates a strong and positive correlation.

- **mse**: Shows us the mean squared error (differences) between predicted values and actual values of **EMISSIONS**. Our value is 25.4089698366953, which is very low and preferred.

- **rmse**: Shows us the root mean squared error (differences) between predicted values and actual values of **EMISSIONS**. Our value is 5.04073108156895, which is very low and preferred. RMSE ignores outliers that may influence MSE.
  The test metrics for lm3 performed better than lm1 & lm2!

```
pred3 <- predict(lm3, newdata = test)
cor3 <- cor(pred3, test$EMISSIONS)
print(paste('correlation for lm3:', cor3))
```

```
## [1] "correlation for lm3: 0.996362653202669"
```

```
mse3 <- mean((pred3-test$EMISSIONS)^2)
print(paste('mse for lm3:', mse3))
```

```
## [1] "mse for lm3: 25.4089698366953"
```

```
rmse3 <- sqrt(mse3)
print(paste('rmse for lm3:', rmse3))
```

```
## [1] "rmse for lm3: 5.04073108156895"
```

knn Regression

- this algorithm uses the k parameter to find the k nearest neighbors to predict the target value.

Building the 1st kNN Model

In the code below, we are using the "train" data to create a kNN Regression model named "fit" to see how **ENGINE.SIZE, CYLINDERS, HWY..L.100.km, & COMB..L.100.km.** can be used to predict **EMISSIONS**.
* The k value is 3, meaning the model is using the 3 nearest neighbors.

```
library(caret)
```

```
## Loading required package: ggplot2
```

```
## Loading required package: lattice
```

```
fit <- knnreg(train[, c(5, 6, 10, 11)], train[, 13], k = 3)
```

Predictions for the 1st kNN model

- **predictions**: Create a vector called "pred_knn1" to store the predicted values. Call the "predict()" function with **fit** and the **test vector** (columns: 5, 6, 10, 11) as input.

- **cor**: Shows us the correlation coefficient between predicted values and actual values of **EMISSIONS**. Our value is 0.945901995612778, which is close to 1 and indicates a strong and positive correlation.

- **mse**: Shows us the mean squared error (differences) between predicted values and actual values of **EMISSIONS**. Our value is 380.23704635907, which is high and not preferred.

- **rmse**: Shows us the root mean squared error (differences) between predicted values and actual values of **EMISSIONS**. Our value is 19.499667852532, which is not low and not preferred. RMSE ignores outliers that may influence MSE.

```
pred_knn1 <- predict(fit, test[, c(5, 6, 10, 11)])
cor_knn1 <- cor(pred_knn1, test$EMISSIONS)
print(paste('correlation for pred_knn1:', cor_knn1))
```

```
## [1] "correlation for pred_knn1: 0.991056571954218"
```

```
mse_knn1 <- mean((pred_knn1-test$EMISSIONS)^2)
print(paste('mse for pred_knn1:', mse_knn1))
```

```
## [1] "mse for pred_knn1: 62.31091313928"
```

```
rmse_knn1 <- sqrt(mse_knn1)
print(paste('rmse for pred_knn1:', rmse_knn1))
```

```
## [1] "rmse for pred_knn1: 7.89372618851705"
```

Scaling the data

We need to scale the data to make sure that all features are weighted equally. source

Columns Used: **ENGINE.SIZE, CYLINDERS, HWY..L.100.km, & COMB..L.100.km.**

Steps:
* First, we need to convert the **CYLINDERS** column into a numeric variable (both train & test vectors).
* Calculate the mean and standard deviation of all the values in the **train** vector.
* **scale() function**: We can achieve train/test scaled vectors by subtracting the mean from each column and dividing by the standard deviation. source

```
train$CYLINDERS <- as.numeric(train$CYLINDERS)
test$CYLINDERS <- as.numeric(test$CYLINDERS)
train_scaled <- train[, c(5, 6, 10, 11)]
means <- sapply(train_scaled, mean)
stdvs <- sapply(train_scaled, sd)
train_scaled <- scale(train_scaled, center = means, scale = stdvs)
test_scaled <- scale(test[, c(5, 6, 10, 11)], center = means, scale = stdvs)
```

Building the 2nd kNN Model

In the code below, we are using the "train_scaled" data to create a kNN Regression model named "fit2" to see how the scaled **ENGINE.SIZE, CYLINDERS, HWY..L.100.km, & COMB..L.100.km.** columns can be used to predict **EMISSIONS**.
* The k value is 3, meaning the model is using the 3 nearest neighbors.

```
fit2 <- knnreg(train_scaled, train$EMISSIONS, k = 3)
```

Predictions for the 2nd kNN model

- **predictions**: Create a vector called "pred_knn2" to store the predicted values. Call the "predict()" function with **fit2** and the **test_scaled vector** (columns: 5, 6, 10, 11) as input.

- **cor**: Shows us the correlation coefficient between predicted values and actual values of **EMISSIONS**. Our value is 0.99133755124126, which is extremely close to 1 and indicates a strong and positive correlation. The correlation is stronger than the previous model's correlation.

- **mse**: Shows us the mean squared error (differences) between predicted values and actual values of **EMISSIONS**. Our value is 60.4290486275571, which is high and not preferred. However, it is lower than the previous model's mse.

- **rmse**: Shows us the root mean squared error (differences) between predicted values and actual values of **EMISSIONS**. Our value is 7.773612328098, which is low and preferred. RMSE ignores outliers that may influence MSE.

```
pred_knn2 <- predict(fit2, test_scaled)
cor_knn2 <- cor(pred_knn2, test$EMISSIONS)
print(paste('correlation for pred_knn2:', cor_knn2))
```

```
## [1] "correlation for pred_knn2: 0.99133755124126"
```

```
mse_knn2 <- mean((pred_knn2-test$EMISSIONS)^2)
print(paste('mse for pred_knn2:', mse_knn2))
```

```
## [1] "mse for pred_knn2: 60.4290486275571"
```

```
rmse_knn2 <- sqrt(mse_knn2)
print(paste('rmse for pred_knn2:', rmse_knn2))
```

```
## [1] "rmse for pred_knn2: 7.773612328098"
```

Finding the best k value

This code uses a for loop to iterate the **knnreg()** function 10 times and evaluate the performance of the k-nearest neighbor regression algorithm for different values of k. By analyzing the correlation and mean squared error (MSE) values obtained for each k value, we can determine the best value of k for the **test** data set.
* cor_k: vector that stores correlation values for all iterations
* mse_k: vector that stores mse values for all iterations

```
cor_k <- rep(0, 10)
mse_k <- rep(0, 10)
i <- 1
for (k in seq(1, 19, 2)){
  fit_k <- knnreg(train_scaled, train$EMISSIONS, k = k)
  pred_k <- predict(fit_k, test_scaled)
  cor_k[i] <- cor(pred_k, test$EMISSIONS)
  mse_k[i] <- mean((pred_k - test$EMISSIONS)^2)
  print(paste("k =", k, cor_k[i], mse_k[i]))
  i <- i + 1
}
```

```
## [1] "k = 1 0.994086190574364 41.2597902830024"
## [1] "k = 3 0.99133755124126 60.4290486275571"
## [1] "k = 5 0.989377123388745 74.2085197571979"
## [1] "k = 7 0.988255456180075 81.9934878154413"
## [1] "k = 9 0.987089447322153 89.9490104068337"
## [1] "k = 11 0.986275218405222 95.612176051431"
## [1] "k = 13 0.985021075065504 104.378490565303"
## [1] "k = 15 0.984400991950239 108.823226211365"
## [1] "k = 17 0.983964124156943 112.106199697465"
## [1] "k = 19 0.983613209648359 114.70423394623"
```

Plot

Here, we are creating a scatter plot of the values stored in **cor_k** and **mse_k** to determine the optimal k value.

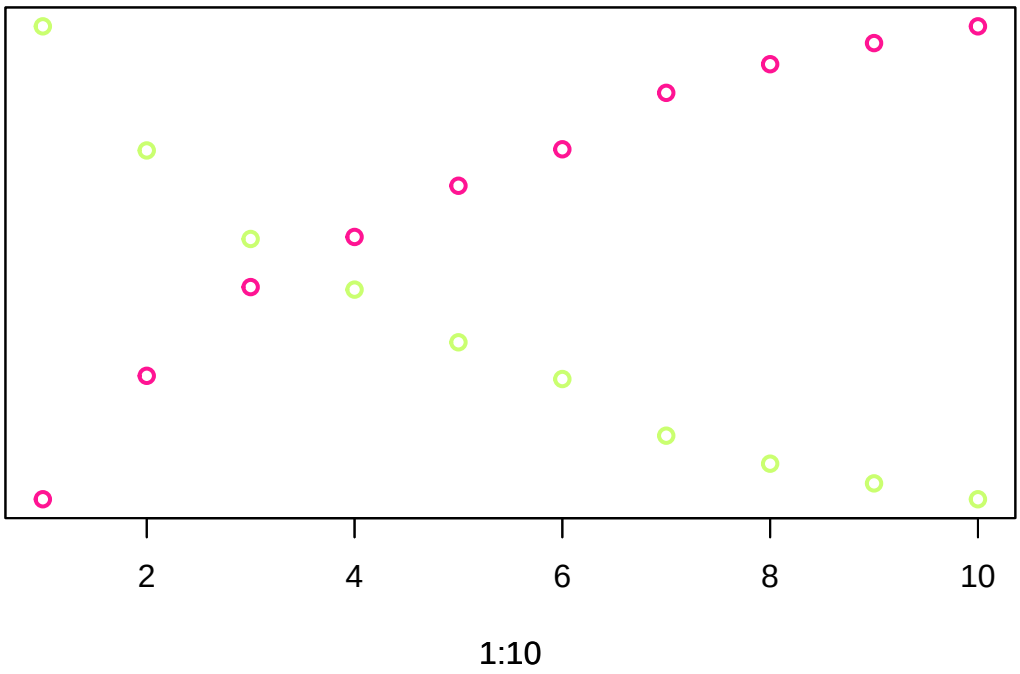
```
plot(1:10, cor_k, lwd=2, col='darkolivegreen1', ylab="", yaxt='n')
par(new=TRUE)
plot(1:10, mse_k, lwd=2, col='deeppink1', labels=FALSE, ylab="", yaxt='n')
```

```
## Warning in plot.window(...): "labels" is not a graphical parameter

## Warning in plot.xy(xy, type, ...): "labels" is not a graphical parameter

## Warning in box(...): "labels" is not a graphical parameter

## Warning in title(...): "labels" is not a graphical parameter
```



We need a high correlation value and a low mse value, so we used the **which.max()** function and the **which.min** function to determine the k value. As we can see, **k = 1** is the best k value for our data.

```
which.max(cor_k)
```

```
## [1] 1
```

```
which.min(mse_k)
```

```
## [1] 1
```

Building the 3rd kNN Model

In the code below, we are using the "train_scaled" data to create a kNN Regression model named "fit3" to see how the scaled **ENGINE.SIZE, CYLINDERS, HWY..L.100.km, & COMB..L.100.km.** columns can be used to predict **EMISSIONS**.
* The k value is 1, meaning the model is selecting the nearest neighbor.

```
fit3 <- knnreg(train_scaled, train$EMISSIONS, k = 1)
```

Predictions for the 3rd kNN model

- **predictions**: Create a vector called "pred_knn3" to store the predicted values. Call the "predict()" function with **fit3** and the **test_scaled vector** (columns: 5, 6, 10, 11) as input.

- **cor**: Shows us the correlation coefficient between predicted values and actual values of **EMISSIONS**. Our value is 0.994086190574364, which is extremely close to 1 and indicates a strong and positive correlation. The correlation is stronger than the previous model's correlation.

- **mse**: Shows us the mean squared error (differences) between predicted values and actual values of **EMISSIONS**. Our value is 41.2597902830024, which is the lowest mse value so far for kNN regression.

- **rmse**: Shows us the root mean squared error (differences) between predicted values and actual values of **EMISSIONS**. Our value is 6.42337841661243, which is low and preferred. RMSE ignores outliers that may influence MSE.
  Overall, the 3rd kNN model performed the best when k = 1!

```
pred_knn3 <- predict(fit3, test_scaled)
cor_knn3 <- cor(pred_knn3, test$EMISSIONS)
print(paste('correlation for pred_knn3:', cor_knn3))
```

```
## [1] "correlation for pred_knn3: 0.994086190574364"
```

```
mse_knn3 <- mean((pred_knn3-test$EMISSIONS)^2)
print(paste('mse for pred_knn3:', mse_knn3))
```

```
## [1] "mse for pred_knn3: 41.2597902830024"
```

```
rmse_knn3 <- sqrt(mse_knn3)
print(paste('rmse for pred_knn3:', rmse_knn3))
```

```
## [1] "rmse for pred_knn3: 6.42337841661243"
```

Decision Trees

- this algorithm recursively splits and organizes the data into a tree-structure with nodes

Building the Decision Tree Model

In the code below, we are using the "train" data to create a Decision Tree Model named "tree1" to see how **FUEL.CONSUMPTION, COMB..L.100.km., & FUEL** can be used to predict **EMISSIONS**. Additionally, the **summary()** function is used to see the results of the model.

```
library(tree)
tree1 <- tree(EMISSIONS~FUEL.CONSUMPTION + COMB..L.100.km. + FUEL, data = train)
summary(tree1)
```

```
##
## Regression tree:
## tree(formula = EMISSIONS ~ FUEL.CONSUMPTION + COMB..L.100.km. +
##     FUEL, data = train)
## Variables actually used in tree construction:
## [1] "COMB..L.100.km." "FUEL"
## Number of terminal nodes:  11
```

```
## Residual mean deviance:  190.6 = 3438000 / 18030
## Distribution of residuals:
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -91.33000  -7.93900   0.06094   0.00000   8.23500 204.70000
```

Interpretation of Summary

* The model has 11 nodes.
* **Residual Mean Deviance:** This is the sum of the differences between actual and predicted values. Our value is 190.6, which is high and not preferred.
* **Distribution of residuals:** Shows the distribution of differences between and actual and predicted values. Min: -91.33000, Mean: 0.00000, & Max: 204.70000.

Predictions for the Decision Tree model

- **predictions**: Create a vector called "pred_tree" to store the predicted values. Call the "predict()" function with **tree1** and the **test vector** (columns: 8, 9, 11) as input.
- **cor**: Shows us the correlation coefficient between predicted values and actual values of **EMISSIONS**. Our value is 0.97439442864124, which is extremely close to 1 and indicates a strong and positive correlation. However, we've seen higher values.
- **rmse**: Shows us the root mean squared error (differences) between predicted values and actual values of **EMISSIONS**. Our value is 13.2986769927383, but we've seen lower values.

```
pred_tree <- predict(tree1, newdata = test)
cor_tree <- cor(pred_tree, test$EMISSIONS)
print(paste('correlation for pred_tree:', cor_tree))
```

```
## [1] "correlation for pred_tree: 0.97439442864124"
```

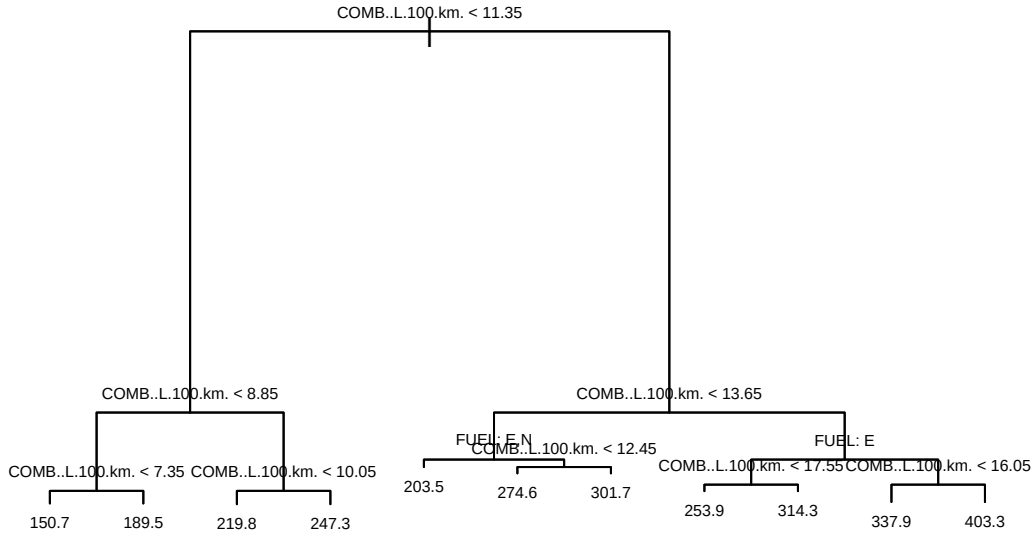
```
rmse_tree <- sqrt(mean((pred_tree-test$EMISSIONS)^2))
print(paste('rmse for pred_tree:', rmse_tree))
```

```
## [1] "rmse for pred_tree: 13.2986769927383"
```

Plotting the Decision Tree model

We can visualize the model's decision-making process for organizing the data and see the number of observations corresponding to each node for the **FUEL type and COMB..L.100.km.**

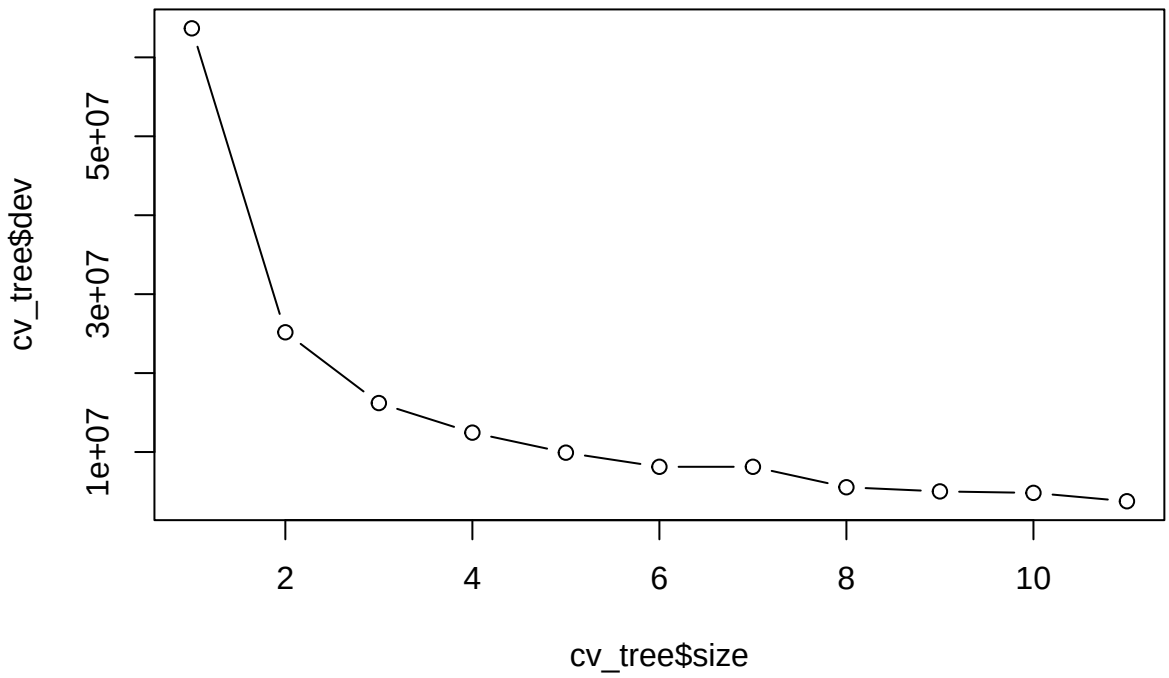
```
plot(tree1)
text(tree1, cex=0.5, pretty=0)
```



Cross Validation

The plot illustrates how the mean cross-validated deviance changes with increasing tree size (number of nodes). To avoid overfitting the model, it is beneficial to identify a "sweet spot" in the middle, where the tree size is neither too small nor too large.

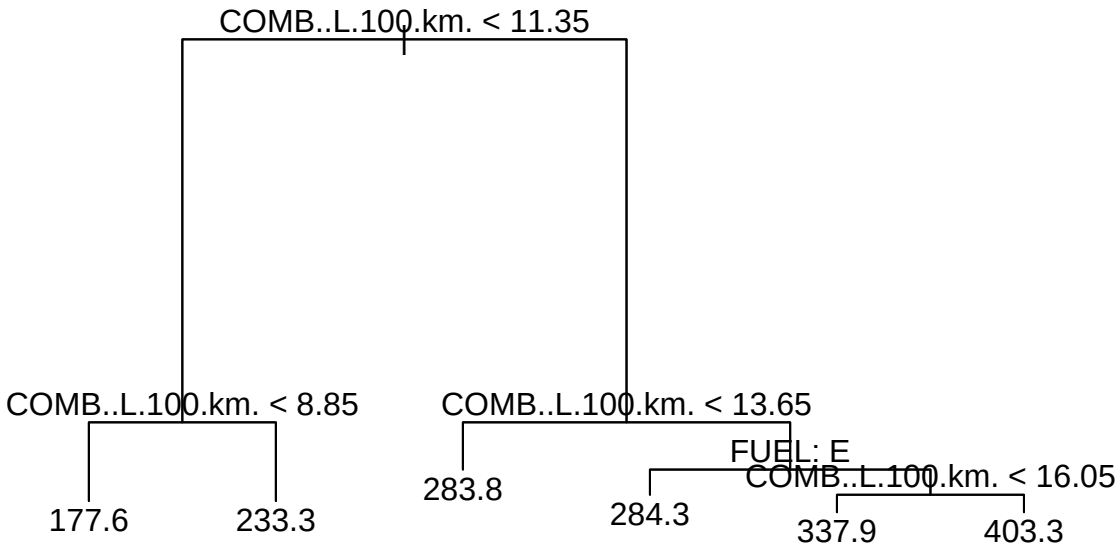
```
cv_tree <- cv.tree(tree1)
plot(cv_tree$size, cv_tree$dev, type='b')
```



Building the Pruned Decision Tree Model

In the code below, we are using the "train" data to create a pruned Decision Tree model named "tree_pruned" to see how **FUEL.CONSUMPTION, COMB..L.100.km., & FUEL** can be used to predict **EMISSIONS**. Additionally, the **plot()** function is used to visualize the pruned tree. The tree will have 6 nodes.

```
tree_pruned <- prune.tree(tree1, best=6)
plot(tree_pruned)
text(tree_pruned, pretty=0)
```



Predictions for the Decision Tree model (Pruned)

- **predictions**: Create a vector called "pred_pruned" to store the predicted values. Call the "predict()" function with **tree_pruned** and the **test vector** (columns: 8, 9, 11) as input.
- **cor**: Shows us the correlation coefficient between predicted values and actual values of **EMISSIONS**. Our value is 0.937474787533414, which is close to 1 and indicates a strong and positive correlation. However, the previous model had a higher correlation value.
- **rmse**: Shows us the root mean squared error (differences) between predicted values and actual values of **EMISSIONS**. Our value is 20.5894930291704, but the previous model had a lower value.

```
pred_pruned <- predict(tree_pruned, newdata=test)
cor_pruned <- cor(pred_pruned, test$EMISSIONS)
print(paste('correlation for pred_pruned:', cor_pruned))
```

```
## [1] "correlation for pred_pruned: 0.937474787533414"
```

```
rmse_pruned <- sqrt(mean((pred_pruned-test$EMISSIONS)^2))
print(paste('rmse for pred_pruned:', rmse_pruned))
```

```
## [1] "rmse for pred_pruned: 20.5894930291704"
```

Pruning did not help, as the correlation value decreased and the rmse value increased. However, the tree is easier to interpret.

Random Forest

Building the RF Model

- set.seed(1234): ensures that the train/test data remains the same across multiple runs of the code
- In the code below, we are using the "train" data to create a Random Forest model named "rf" to see how **FUEL.CONSUMPTION, COMB..L.100.km., & FUEL** can be used to predict **EMISSIONS.** This algorithm is a greedy approach that aims to identify the optimal decision tree that performs better than the others.
- importance=TRUE: prioritizes the importance of predictors in the model.

```
library(randomForest)
```

```
## randomForest 4.7-1.1
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
set.seed(1234)
rf <- randomForest(EMISSIONS~FUEL.CONSUMPTION + COMB..L.100.km. + FUEL, data=train, importance=TRUE)
rf
```

```
##
## Call:
##  randomForest(formula = EMISSIONS ~ FUEL.CONSUMPTION + COMB..L.100.km. +      FUEL, data = train, imp
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 1
##
##           Mean of squared residuals: 53.1068
##                     % Var explained: 98.5
```

Interpretation of Summary

- **Number of Trees**: 500 trees were checked to find the optimal one.
- **No. of variables tried at each split:** 1
- **Mean of squared residuals**: Shows us the differences between predicted values and actual values of **EMISSIONS**. Our value is 53.1068, but we've seen lower values.
- **% Var explained:** The model accounts for 98.5% of the variance.

Predictions for the Random Forest model

- **predictions**: Create a vector called "pred_rf" to store the predicted values. Call the "predict()" function with **rf** and the **test vector** (columns: 8, 9, 11) as input.

- **cor**: Shows us the correlation coefficient between predicted values and actual values of **EMISSIONS**. Our value is 0.995683972106648, which is extremely close to 1 and indicates a strong and positive correlation.
- **rmse**: Shows us the root mean squared error (differences) between predicted values and actual values of **EMISSIONS**. Our value is 7.02266093303241, which is low and preferred.

```
pred_rf <- predict(rf, newdata=test)
cor_rf <- cor(pred_rf, test$EMISSIONS)
print(paste('correlation for pred_rf:', cor_rf))
```

```
## [1] "correlation for pred_rf: 0.995683972106648"
```

```
rmse_rf <- sqrt(mean((pred_rf-test$EMISSIONS)^2))
print(paste('rmse for pred_rf:', rmse_rf))
```

```
## [1] "rmse for pred_rf: 7.02266093303241"
```

Bagging

- In the code below, we are using the "train" data to create a Random Forest model named "bag" to see how **FUEL.CONSUMPTION, COMB..L.100.km., & FUEL** can be used to predict **EMISSIONS.** This algorithm is a greedy approach that aims to identify the optimal decision tree that performs better than the others.
- importance=TRUE: prioritizes the importance of predictors in the model.
- mtry-2: samples 2 random predictors at a time

```
bag <- randomForest(EMISSIONS~FUEL.CONSUMPTION + COMB..L.100.km. + FUEL, data=train, mtry=2)
bag
```

```
##
## Call:
##  randomForest(formula = EMISSIONS ~ FUEL.CONSUMPTION + COMB..L.100.km. +      FUEL, data = train, mt
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##           Mean of squared residuals: 5.563235
##                     % Var explained: 99.84
```

Interpretation of Summary

- **Number of Trees**: 500 trees were checked to find the optimal one.
- **No. of variables tried at each split:** 2
- **Mean of squared residuals**: Shows us the differences between predicted values and actual values of **EMISSIONS**. Our value is 5.636477, which is low and preferred.
- **% Var explained:** The model accounts for 99.84% of the variance.

Predictions for the Random Forest model (Bagging)

- **predictions**: Create a vector called "pred_bag" to store the predicted values. Call the "predict()" function with **bag** and the **test vector** (columns: 8, 9, 11) as input.
- **cor**: Shows us the correlation coefficient between predicted values and actual values of **EMISSIONS**. Our value is 0.999189618456568, which is extremely close to 1 and indicates a strong and positive correlation.
- **rmse**: Shows us the root mean squared error (differences) between predicted values and actual values of **EMISSIONS**. Our value is 2.38168110053404, which is very low and preferred. **This model outperformed the previous random forest model!*

```
pred_bag <- predict(bag, newdata=test)
cor_bag <- cor(pred_bag, test$EMISSIONS)
print(paste('correlation for pred_bag:', cor_bag))
```

```
## [1] "correlation for pred_bag: 0.999189618456568"
```

```
rmse_bag <- sqrt(mean((pred_bag-test$EMISSIONS)^2))
print(paste('rmse for pred_bag:', rmse_bag))
```

```
## [1] "rmse for pred_bag: 2.38168110053404"
```

Analysis of all 3 Algorithms:

- Linear Regression is used to model linear relationships between predictor variables and target variables. However, it is not the best algorithm to use when there are non-linear relationships or categorical variables, as it is easily influenced by outliers. In this notebook, lm3 performed the best and had a correlation of 0.996362653202669 and rmse of 5.04073108156895.
- kNN Regression predicts the target based on k nearest neighbors. However, this algorithm is not effective if the most optimal k value is not chosen. Scaling the train/test vectors was beneficial and setting k = 1 yielded a correlation of 0.994086190574364 and rmse of 6.42337841661243.
- Decision Tree Regression recursively organizes the predictors into a tree structure with nodes. It can show non-linear relationships and trees are easy to interpret if they are pruned. The algorithm works best when predictors are randomly sampled using the "randomForest()" function to find the most optimal tree. The resulting correlation was 0.999189618456568 and rmse was 2.38168110053404.

Creating a Random Forest Model with randomly sampled predictors yielded the best model for our data!