WESTERN SYDNEY UNIVERSITY

SCHOOL of: Computer, Data and Mathematical Sciences

ASSIGNMENT COVER SHEET

STUDENT DETAILS

Name: Nguyễn Hoàng Nhật Linh **Student ID:** 22084016

SUBJECT AND TUTORIAL DETAILS

Subject Name: Analytics Programming **Subject code:** COMP1013

Tutorial Group: **Day:** Sunday **Time:** 3:15 PM - 6:45 PM

Lecturer or Tutor name: Prof. Nguyen Tan Luy

ASSIGNMENT DETAILS

Title: Assignment **Length:** 20 pages

Due Date: 21/11/2025 **Date submitted: 21/11/2025**

Home campus: Vietnam

DECLARATION

By submitting your work using this link you are certifying that:

☒ You hold a copy of this submission if the original is lost or damaged.

☒ No part of this submission has been copied from any other student's work or from any other third party (including generative AI) except where due acknowledgment is made in the submission.

☒ No part of this submission has been submitted by you in another (previous or current) assessment, except where appropriately referenced, and with prior permission from the teacher/tutor/supervisor/ Subject Coordinator for this subject.

☒ No part of this submission has been written/produced for you by any other person or technology except where collaboration has been authorised by the teacher/tutor/ supervisor/Subject Coordinator either in the assessment resources section of the Learning Guide for this assessment task, in the instructions for this assessment task, or through vUWS.

☒ You are aware that this submission will be reproduced and submitted to detection software programs for the purpose of investigating possible breaches of the Student Misconduct Rule, for example, plagiarism, contract cheating, or unauthorised use of generative AI. Turnitin or other tools of investigation may retain a copy of the submission for the purposes of future investigation.

☒ You will not make this submission available to any other person unless required by the University.

***Instructions**: Please complete the requested details in the form, save it and convert to PDF before adding your signature below*

Student signature:

Note: An examiner or lecturer/tutor has the right to not mark this assignment if the above declaration has not been completed. Staff may contact you for permission to share a de-identified extract or copy of your submission with students or staff for teaching purposes, following guidelines for requesting and sharing exemplar assessment tasks.

Repository submission link (GitHub):

https://github.com/21000847/AP_A1_Nguyen-Hoang-Nhat-Linh_22084016

Assignment

Nguyen Hoang Nhat Linh

2025-11-23

Load Data

First of all, we begin by reading the datasets from its respective CSV files. Afterwards, we inspect on each datasets to see the total rows and columns. This allow us to know the original dataset characteristics.

```
# Read csv files with the directory well set beforehand
engine <- read.csv("Engine.csv")
auto <- read.csv("Automobile.csv")
maint <- read.csv("Maintenance.csv")

# Inspect 'Engine'' data
print("Engines:")
```

```
## [1] "Engines:"
```

```
str(engine)
```

```
## 'data.frame':    88 obs. of  8 variables:
##  $ EngineModel : chr  "E-0001" "E-0002" "E-0003" "E-0004" ...
##  $ EngineType  : chr  "dohc" "ohcv" "ohc" "ohc" ...
##  $ NumCylinders: chr  "four" "six" "four" "five" ...
##  $ EngineSize  : int  130 152 109 136 136 131 131 108 164 164 ...
##  $ FuelSystem  : chr  "mpfi" "mpfi" "mpfi" "mpfi" ...
##  $ Horsepower  : chr  "111" "154" "102" "115" ...
##  $ FuelTypes   : chr  "gas" "gas" "gas" "gas" ...
##  $ Aspiration  : chr  "std" "std" "std" "std" ...
```

```
summary(engine)
```

```
##  EngineModel         EngineType        NumCylinders         EngineSize
##  Length:88          Length:88          Length:88          Min.   : 60.0
##  Class :character   Class :character   Class :character   1st Qu.:108.0
##  Mode  :character   Mode  :character   Mode  :character   Median :121.0
##                                                           Mean   :134.1
##                                                           3rd Qu.:151.2
##                                                           Max.   :320.0
##   FuelSystem         Horsepower         FuelTypes          Aspiration
##  Length:88          Length:88          Length:88          Length:88
##  Class :character   Class :character   Class :character   Class :character
```

```
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##
##
##
```

```r
# Inspect 'Automobile' data
print("Automobile:")
```

```
## [1] "Automobile:"
```

```r
str(auto)
```

```
## 'data.frame':    204 obs. of  13 variables:
##  $ PlateNumber   : chr  "53N-001" "53N-002" "53N-003" "53N-004" ...
##  $ Manufactures  : chr  "Alfa-romero" "Alfa-romero" "Audi" "Audi" ...
##  $ BodyStyles    : chr  "convertible" "hatchback" "sedan" "sedan" ...
##  $ DriveWheels   : chr  "rwd" "rwd" "fwd" "4wd" ...
##  $ EngineLocation: chr  "front" "front" "front" "front" ...
##  $ WheelBase     : num  88.6 94.5 99.8 99.4 99.8 ...
##  $ Length        : num  169 171 177 177 177 ...
##  $ Width         : num  64.1 65.5 66.2 66.4 66.3 71.4 71.4 71.4 67.9 64.8 ...
##  $ Height        : num  48.8 52.4 54.3 54.3 53.1 55.7 55.7 55.9 52 54.3 ...
##  $ CurbWeight    : int  2548 2823 2337 2824 2507 2844 2954 3086 3053 2395 ...
##  $ EngineModel   : chr  "E-0001" "E-0002" "E-0003" "E-0004" ...
##  $ CityMpg       : int  21 19 24 18 19 19 19 17 16 23 ...
##  $ HighwayMpg    : int  27 26 30 22 25 25 25 20 22 29 ...
```

```r
summary(auto)
```

```
##  PlateNumber        Manufactures        BodyStyles        DriveWheels
##  Length:204         Length:204         Length:204         Length:204
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##
##
##
##  EngineLocation       WheelBase          Length          Width
##  Length:204         Min.   : 86.60   Min.   :141.1   Min.   :60.30
##  Class :character   1st Qu.: 94.50   1st Qu.:166.3   1st Qu.:64.08
##  Mode  :character   Median : 97.00   Median :173.2   Median :65.50
##                     Mean   : 98.81   Mean   :174.1   Mean   :65.92
##                     3rd Qu.:102.40   3rd Qu.:183.2   3rd Qu.:66.90
##                     Max.   :120.90   Max.   :208.1   Max.   :72.30
##      Height        CurbWeight   EngineModel           CityMpg
##  Min.   :47.80   Min.   :1488   Length:204         Min.   :10.00
##  1st Qu.:52.00   1st Qu.:2145   Class :character   1st Qu.:19.00
##  Median :54.10   Median :2414   Mode  :character   Median :24.00
##  Mean   :53.75   Mean   :2556                      Mean   :25.23
##  3rd Qu.:55.50   3rd Qu.:2939                      3rd Qu.:30.00
##  Max.   :59.80   Max.   :4066                      Max.   :50.00
##    HighwayMpg
##  Min.   :15.00
```

```
##  1st Qu.:25.00
##  Median :30.00
##  Mean   :30.76
##  3rd Qu.:34.50
##  Max.   :55.00
```

```
# Inspect 'Maintenance' data
print("Maintenance:")
```

```
## [1] "Maintenance:"
```

```
str(maint)
```

```
## 'data.frame':    374 obs. of  7 variables:
##  $ ID         : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ PlateNumber: chr  "53N-001" "53N-001" "53N-001" "53N-001" ...
##  $ Date       : chr  "15/02/2024" "16/03/2024" "15/04/2024" "15/05/2024" ...
##  $ Troubles   : chr  "Break system" "Transmission" "Suspected clutch" "Ignition (finding)" ...
##  $ ErrorCodes : int  -1 -1 -1 1 -1 1 1 0 -1 -1 ...
##  $ Price      : int  110 175 175 180 85 1000 180 0 180 180 ...
##  $ Methods    : chr  "Replacement" "Replacement" "Adjustment" "Adjustment" ...
```

```
summary(maint)
```

```
##        ID         PlateNumber           Date             Troubles
##  Min.   :  1.00   Length:374         Length:374         Length:374
##  1st Qu.: 94.25   Class :character   Class :character   Class :character
##  Median :187.50   Mode  :character   Mode  :character   Mode  :character
##  Mean   :187.50
##  3rd Qu.:280.75
##  Max.   :374.00
##    ErrorCodes          Price          Methods
##  Min.   :-1.00000   Min.   :   0.0   Length:374
##  1st Qu.:-1.00000   1st Qu.:  85.0   Class :character
##  Median : 0.00000   Median : 120.0   Mode  :character
##  Mean   : 0.04813   Mean   : 204.8
##  3rd Qu.: 1.00000   3rd Qu.: 180.0
##  Max.   : 1.00000   Max.   :1000.0
```

Using "str()", the overall structure of each datasets are well displayed.From the summaries above, we can see that the dataset overview can be seen as follow:

- **Engine**: 8 variables with 88 rows
- **Automobile**: 13 variables with 204 rows
- **Maintenance**: 7 variables with 374 rows

This is the overall structure when run the datasets in R. From here, we would proceed to solve each tasks.

Task 1:

1.1: Write the code to inspect the data structure and present the data: The missing values in the dataset were written as "?", replace any "?" with NA; Write code to check: after replacing, how many rows were affected in total? Does this change alter the data distribution?

Count "?" and replace it with "NA"

In R, the question mark **'?'** is not recognize by as a missing value. This could cause errors in statistical analysis. Therefore, by replacing **'?' to 'NA'** we can code to find out the missing values.

In this question, using the *"mutate()"* function with *"(across(where(is.character)))"*,we can scan all character columns and replace all the '?' to 'NA'.

```
# Count "?" and replace it with "NA"
NA_replace <- function(df, name) {
  # Count rows with "?"
  rows_with_q <- df %>%
    filter(if_any(everything(), ~ . == "?")) %>%
    # counts total number of rows that contain at least one instance '?' (everything)
    nrow()

  # Replace "?" with NA
  df_clean <- df %>%
    mutate(across(where(is.character), ~na_if(., "?")))
  # scan all character columns and replace all the '?' to 'NA'

  print(paste("Number of rows affected in", name, ":", rows_with_q))
  return(df_clean)
}

# Apply cleaning
engines <- NA_replace(engine, "Engines")
```

```
## [1] "Number of rows affected in Engines : 6"
```

```
autos <- NA_replace(auto, "Automobiles")
```

```
## [1] "Number of rows affected in Automobiles : 0"
```

```
maint <- NA_replace(maint, "Maintenance")
```

```
## [1] "Number of rows affected in Maintenance : 0"
```

In total, there are 6 rows being affected (all in 'Engine').

```
# Total NA
print(paste("Total NAs in Engine:", sum(is.na(engines))))
```

```
## [1] "Total NAs in Engine: 6"
```

```r
print(paste("Total NAs in Automobile:", sum(is.na(autos))))
```

```
## [1] "Total NAs in Automobile: 0"
```

```r
print(paste("Total NAs in Engine:", sum(is.na(maint))))
```

```
## [1] "Total NAs in Engine: 28"
```

As we can see, there are 6 '?' and when we replace it, we end up with **34 NAs in total**.

Here, when we convert it to NA, we are simply marking the missing spots properly so R can handle them. All the real numbers and categories stays exactly the same, so the overall distribution of the dataset does not change.

1.2 Convert categorical variables BodyStyles, FuelTypes, ErrorCodes to factors

```r
# Convert categorical variables to factors

# Converting 'BodyStyles' from character to factor
autos$BodyStyles <- as.factor(autos$BodyStyles)

# Converting 'FuelTypes' from character to factor
engines$FuelTypes <- as.factor(engines$FuelTypes)

# Converting 'ErrorCodes' from character to factor
maint$ErrorCodes <- as.factor(maint$ErrorCodes)
```

By using **"as.factor"** we transform the 'BodyStyles', 'FuelTypes', and 'ErrorCodes' from characters types to factors. This is to ensure the that variables are recognised as a factor rather than numbers or simple text. This way, R can read them correctly to group data when plotting and analysing.

1.3 Replace the missing values in column Horsepower with the median horsepower

```r
# Convert Horsepower to numeric
engines$Horsepower <- as.numeric(engines$Horsepower)

# This is just an extra step to make sure it is also numeric
engines$EngineSize <- as.numeric(engines$EngineSize)

# since the Horsepower column is numeric
hp_median <- median(engines$Horsepower, na.rm = TRUE)

# Replace NA with the median
engines <- engines %>%
  mutate(Horsepower = ifelse(is.na(Horsepower), hp_median, Horsepower))
# checks every row in the Horsepower column.
# If a value is missing "(is.na())", it replaces it with the calculated 'hp_median'
```

```r
# otherwise it keeps the original value

print(paste("Missing Horsepower values remaining:", sum(is.na(engines$Horsepower))))
```

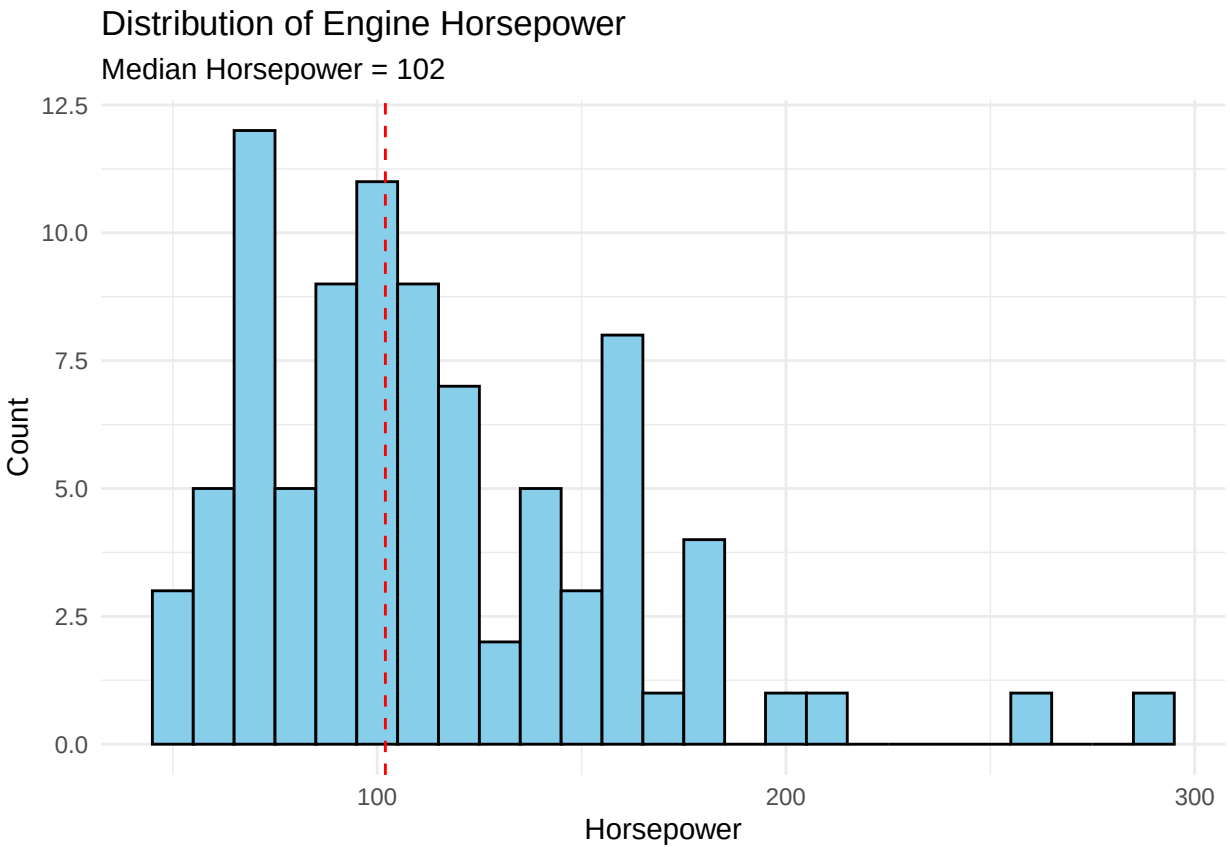
```
## [1] "Missing Horsepower values remaining: 0"
```

In this situation, 'Horsepower' is converted to numeric so calculations work. Then, imputing the missing values in 'Horsepower' column with **median** would offer a better better than using mean. As median is less sensitive to outliers than mean, making it a better option to chose considering the central tendency of this variable.

Additionally, **"mutate()"** and **"ifelse()"** were used to select only the NAs to replace with the calculated "hp_median" above. This would eventually preserves the integrity of the existing data.

1.4 Select the appropriate chart type and display: horsepower distribution

```r
# Histogram for distribution
ggplot(engines, aes(x = Horsepower)) +
  geom_histogram( # this creates a histogram
    binwidth = 10, fill = "skyblue", color = "black") +
  geom_vline(aes(xintercept = median(Horsepower)), color="red", linetype="dashed") +
  # the line above creates an abline that shows the median for 'Horsepower'
  labs(title = "Distribution of Engine Horsepower", # this is the name for the histogram
       subtitle = paste("Median Horsepower =", hp_median), # this adds a sub-title
       x = "Horsepower", y = "Count") + # names the x and y axis
  theme_minimal() # removes the default gray background
```



As the variable 'Horsepower' is a numeric variable, its distribution can be visualized using a histogram. The abline (red line) shows where the middle value is. This makes the shape and central tendency of horsepower becomes much easier to understand. Our red line here shows that the median horsepower is around 102.

Task 2:

2.1 The code to analyse the distribution of the horsepower across the engine types.

Here, we would want to see the 'Horsepower' (a numeric variable) and we would want to see it across engine categories (in our case is Engine Types).

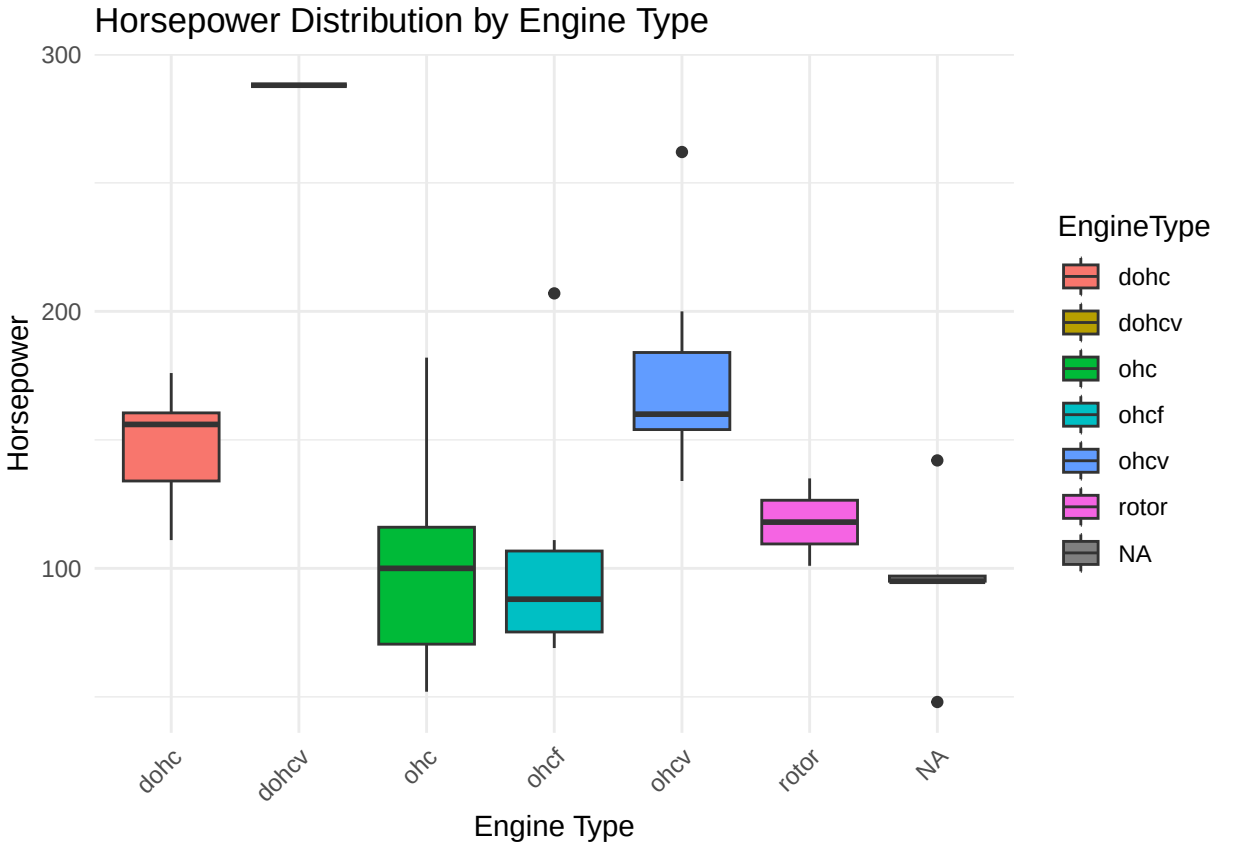
A boxplot would show the median, this helps us to quickly identify patterns, such as which engine types are generally more powerful or more consistent in their performance. Most importantly, it shows the overall spread of horsepower values, and any extreme points that stand out as outliers.

Therefore, employing a box plot would allow us to have a clearer understanding of how horsepower is distributed across the different engine designs.

```
# To use boxplots to analyse the distribution and
# compare continuous variable (in our case, horsepower) across multiple categories.

ggplot(engines, aes(x = EngineType, y = Horsepower, fill = EngineType)) + # this defines the x and y
  geom_boxplot() + # this creates a box plot
  labs(title = "Horsepower Distribution by Engine Type", # title for the box plot
       x = "Engine Type", y = "Horsepower") + # this names the x and y axis
```

```
  theme_minimal() + # removes the default gray background
  # the one below rotates the labels 45 degrees to prevent overlapping
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



From the boxplot above, we can see that two 'dohc' (red) and 'ohcv' (blue) generally produces a higher median horsepower compared to other types. However, the 'ohc' (green) shows a great range of horsepower.

We can easily spot several outliers that indicates an "irregular" in what is generally shown. For instance, while most 'ohc' (pink) engines are low-power, some specific models have much higher output.

2.2 Write the code to investigate the distribution of the horsepower across the groups of the engine sizes (e.g., 60-90, 91-190, 191-299, 300+). Henceforth, visualise the findings.

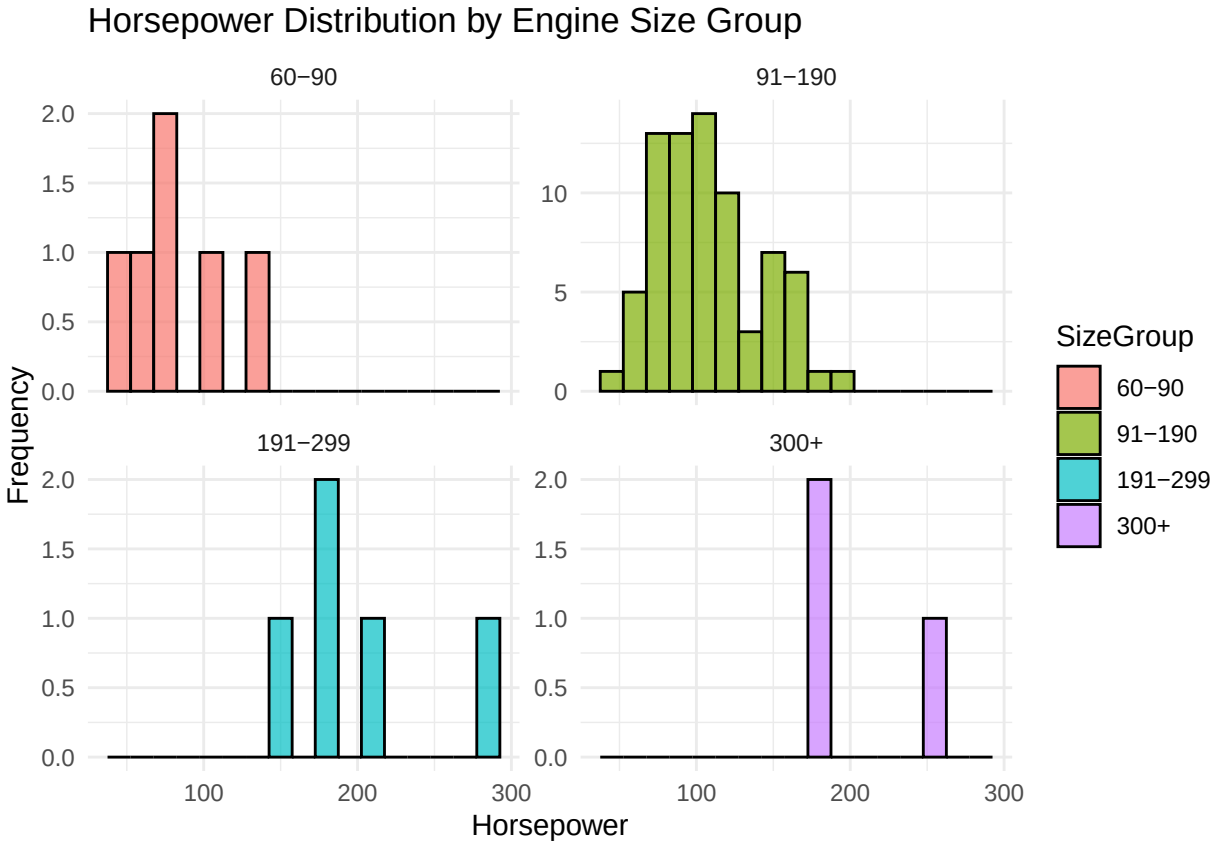
To know the distribution of horsepower across groups of engines sizes, we will need to create bins that represent each groups (60-90, 91-190, 191-299, 300+). Here, we know that 'EngineSize' is a continuous variable.

To create bins, the function **"mutate()"** was used to create a new column 'SizeGroup' and **"cut()"** was used to categorize rows base on their 'EngineSize' variable.

```
# Create bins: 60-90, 91-190, 191-299, 300+
engines <- engines %>%
  mutate(SizeGroup # create a new column 'SizeGroup'
```

```
        = cut(EngineSize, # categorize rows base on their 'EngineSize'
                          breaks = c(59, 90, 190, 299, Inf),
                          labels = c("60-90", "91-190", "191-299", "300+")))


# Visualisations
ggplot(engines, aes(x = Horsepower, fill = SizeGroup)) + # defines the x and y axis
  geom_histogram( # Creates a histogram
    binwidth = 15, color = "black", alpha = 0.7) +
  # groups horsepower into bins 15 units wide, black colored with 0.7 opacity
  facet_wrap(~SizeGroup, scales = "free_y") +
  # splits the single chart into separate mini-charts for each 'SizeGroup'
  labs(title = "Horsepower Distribution by Engine Size Group", # chart's title
       x = "Horsepower", y = "Frequency") +  # this names the x and y axis
  theme_minimal() # removes the default gray background
```



We generally expect a positive correlation such as larger engine size groups (300+), should correlate with the higher horsepower ranges in the histogram.

Here, the histogram confirms it. We can see a positive correlation between engine sizes and horsepower as follow:

- **Small engines (60-90)** and **(91-190)**: Strictly concentrated at the lower end of the horsepower scale.
- **Medium engines (191-299)**: Interestingly, while most fall in the average range, some outliers does shows the highest horsepower.

- **Large engines (300+)**: Here, it is reasonable when large engines fall mostly in high horsepower ranges. However, the distribution only falls in 2 specific level of horsepower.

As we can observe, most of the times the larger the engines the horsepower would be higher as well. However, some outliers (special cases) would offer a different horsepower outcome. This also means that, not all engine size larger than 300+ would bring back the best 'Horsepower' range.

Task 3:

Clean data (eliminates duplications)

Before we perform the analysis, it is important to ensure that the 'Engine' dataset does not contain duplicate. Since if duplicates exist in the engines table, merging it with the autos table would result in a "many-to-many" join.

```
# Clean data
print("Duplicate Engine Models found (original Engine data):")
```

```
## [1] "Duplicate Engine Models found (original Engine data):"
```

```
duplicate_engines <- engines %>% # taking the 'engines' data frame
  group_by(EngineModel) %>% # creating groups for every unique value in 'EngineModel'
  filter(n() > 1) # counts the number of rows in the current group, count groups greater than 1
print(duplicate_engines)
```

```
## # A tibble: 8 x 9
## # Groups:   EngineModel [4]
##   EngineModel EngineType NumCylinders EngineSize FuelSystem Horsepower FuelTypes
##   <chr>       <chr>      <chr>             <dbl> <chr>           <dbl> <fct>
## 1 E-0010      ohc        six                 164 mpfi              121 gas
## 2 E-0010      ohc        six                 209 mpfi              182 gas
## 3 E-0028      rotor      two                  70 4bbl              101 gas
## 4 E-0028      rotor      two                  80 mpfi              135 gas
## 5 E-0041      ohc        four                110 spdi              116 gas
## 6 E-0041      ohc        four                110 spdi              116 gas
## 7 E-0081      ohc        four                141 mpfi              114 gas
## 8 E-0081      ohc        four                141 mpfi              114 gas
## # i 2 more variables: Aspiration <chr>, SizeGroup <fct>
```

As we can see here, these are the duplicates within engine.

```
# Create a unique version to merge reliably (without duplicates)
engines_unique <- engines %>%
  distinct(EngineModel, .keep_all = TRUE) # ensure each engine model appears only once

# Merge only Automobile and unique Engine data
car_engine_merged <- inner_join(autos, engines_unique, by = "EngineModel")
#combine 'Automobile' (Mpg) and 'engines_unique data' (FuelTypes)
```

By applying distinct **"(EngineModel, .keep_all = TRUE)"**, we can ensure that each engine model appears only once, guaranteeing a clean one-to-one relationship when joining with the autos dataset.

We then use **"inner_join"** to combine the 'Automobile' data (for MPG) and the 'engines_unique data' (for FuelTypes), creating the normalized dataset, where each car has one unique set of engine specifications (this is essential for statistical testing).

3.1 Do diesel cars have higher average CityMpg than gasoline cars? Provide statistical evidence

To compare the fuel efficiency of diesel and gasoline cars, a two-Sample t-test was performed. This statistical test would allow us to compare how the mean in 'CityMPG' differs between two independent groups (diesel vs. gas).

Here, 'CityMpg' would act as a measurement to compare these two.

```
# T-test to compare the means of CityMpg and FuelTypes (Gas & Diesel).
print("--- T-Test: Diesel vs Gas CityMpg ---") # the title
```

```
## [1] "--- T-Test: Diesel vs Gas CityMpg ---"
```

```
t_test_result <- t.test(CityMpg ~ FuelTypes, data = car_engine_merged) # employing t-test
print(t_test_result)
```

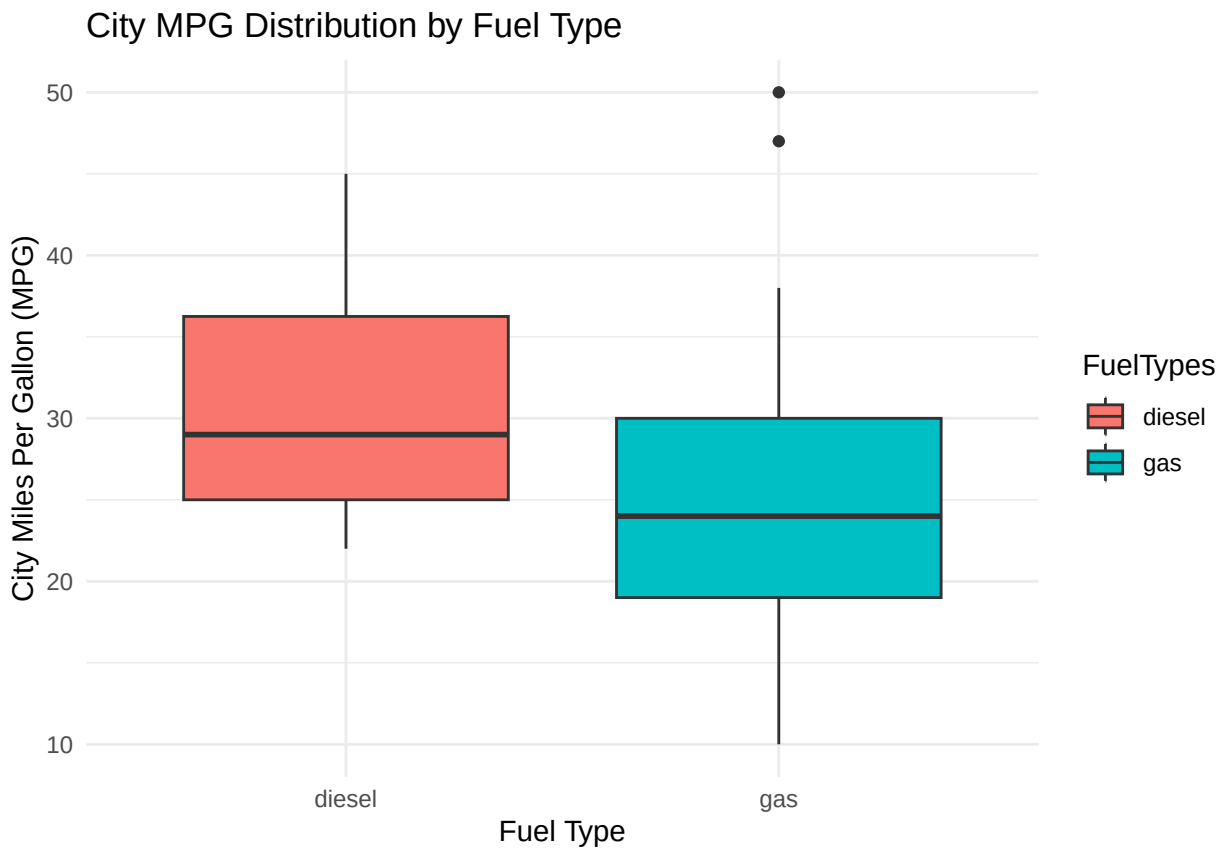
```
##
##  Welch Two Sample t-test
##
## data:  CityMpg by FuelTypes
## t = 3.6237, df = 23.015, p-value = 0.001424
## alternative hypothesis: true difference in means between group diesel and group gas is not equal to 0
## 95 percent confidence interval:
##  2.412141 8.829163
## sample estimates:
## mean in group diesel    mean in group gas
##             30.30000             24.67935
```

In our t-test, we can see that it showed a p-value of **0.0014**, which is far below the standard significance threshold of 0.05 (α). This indicates strong statistical evidence that the average 'CityMpg' is different between diesel and gasoline cars.

More importantly, the mean in group shows that diesel average **30.3 MPG**, while gas averaged **24.68 MPG**. This means that diesel cars have a higher mean and we can say that diesel cars really do bring back better fuel efficiency than gasoline.

Visualisation

```
# Visualization
ggplot(car_engine_merged, aes(x = FuelTypes, y = CityMpg, fill = FuelTypes)) +
  geom_boxplot() + # this creates a box plot
  labs(title = "City MPG Distribution by Fuel Type", # the title
       y = "City Miles Per Gallon (MPG)",  # names the y axis
       x = "Fuel Type") +  # names the x axis
  theme_minimal() # removes the default gray background
```



To better see the differences, a boxplot displaying these 2 numeric variables. Here, we can see that diesel cars have a highe median 'CityMpg', and the majority distribution sits above gasoline's.

3.2 How does DriveWheels affect fuel efficiency (CityMpg and HighwayMpg)?

To understand how the vehicle's drive configuration influences fuel efficiency, MPG values were grouped. In this problem, **"group_by()?" and "summarise()"** were used for both 'CityMpg' and HighwayMpg to compute indicators of efficiency for each type.

In short, the **"group_by()"** will separate the data by drive type and the **"summarise()"** will compute the mean Mpg for each group ('CityMpg' and 'HighwayMpg'). This would allow direct comparison of fuel efficiency across different types of drive.

```
# Group by DriveWheels and calculate mean MPG metrics to quantify the relationship.
efficiency_summary <- car_engine_merged %>%
  group_by(DriveWheels) %>% # separates the data by drive type
  summarise( # compute the mean Mpg for each group
   gitAvg_CityMpg = mean(CityMpg, na.rm = TRUE), # mean of CityMpg
    Avg_HwyMpg = mean(HighwayMpg, na.rm = TRUE) # mean of HighwayMpg
  )

print("Average City and Highway MPG by Drive Wheels:")
```

```
## [1] "Average City and Highway MPG by Drive Wheels:"
```

```r
print(efficiency_summary)
```

```
## # A tibble: 3 x 3
##   DriveWheels gitAvg_CityMpg Avg_HwyMpg
##   <chr>                <dbl>      <dbl>
## 1 4wd                   23.1       27.2
## 2 fwd                   28.3       34.2
## 3 rwd                   20.5       25.6
```

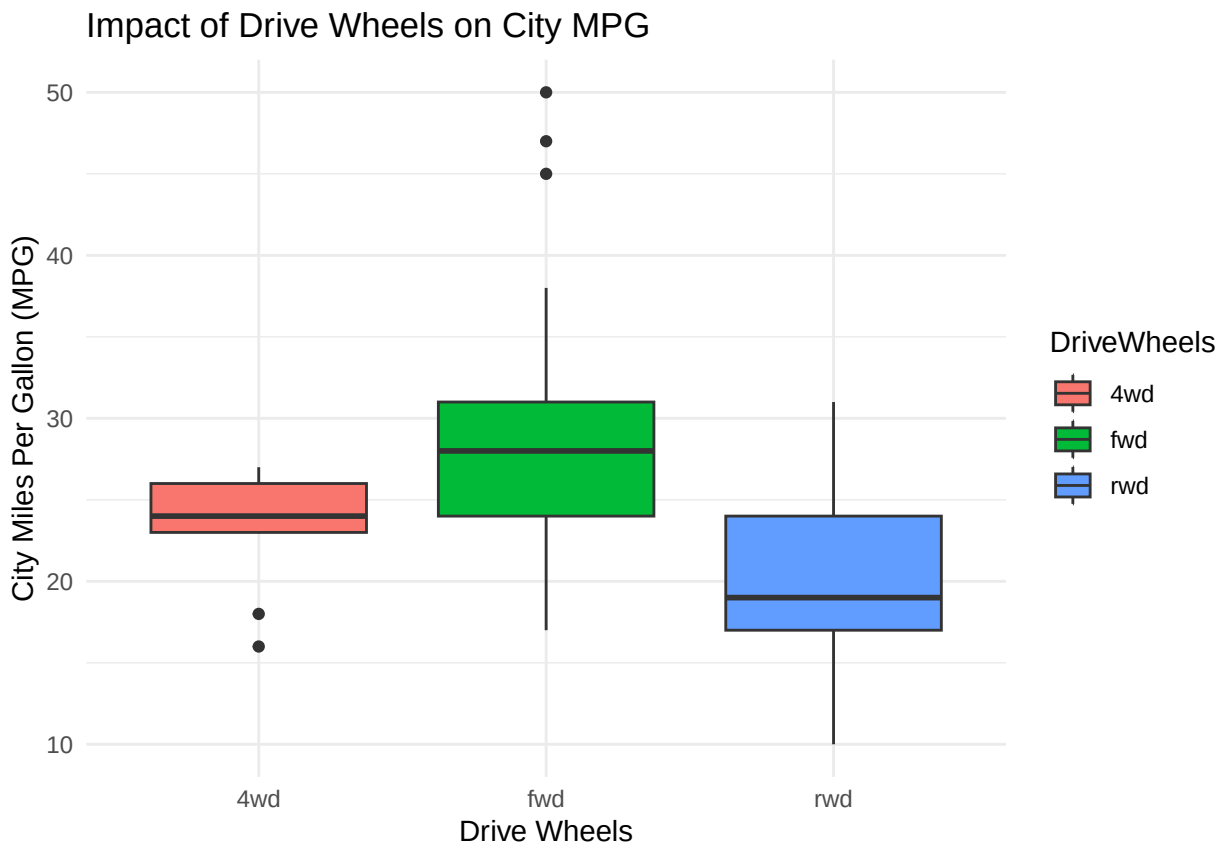
From the table above, we can see the following trends:

- **4wd** vehicles (at 27.22) shows lower Mpg than Fwd vehicles , and slightly higher than Rwd. Even so, it is still reasonable to say that 4wd systems does add significant weight and require more engine output to move both axles.
- **Fwd** vehicles (34.23) has the highest Mpg in both city and highway conditions. Here, we can see that Fwd vehicles offer a much more fuel-efficient type of vehicle.
- **Rwd** vehicles, on the other hand, shows lowest fuel efficiency (at 25.64).This indicates that regardless, Rwd would be the type of vehicles that cost energy the most.

Visualisation

To put this comparison into better visualisation, we can use boxplot to see the entire distribution of 'CityMpg' across the three ordered 'DriveWheels' groups.

```r
# For CityMpg
ggplot(car_engine_merged, aes(x = DriveWheels, y = CityMpg, fill = DriveWheels)) +
  geom_boxplot() +  # This creates a box plot
  labs(title = "Impact of Drive Wheels on City MPG",
       y = "City Miles Per Gallon (MPG)",
       x = "Drive Wheels") +
  theme_minimal() # removes the default gray background
```



The boxplot shows the exact trends the were found previously. Fwd vehicles show the highest median and widest overall distribution, indicating a diverse set of fuel-efficient models.

In contrast, Rwd and 4wd vehicles cluster toward lower MPG values. The visualization makes it evident that Drive Wheels plays an important role in vehicle efficiency.

3.3 Filter out those engines in the dataset that have trouble or are suspected of having trouble

To focus specifically on vehicles with mechanical issues, the Maintenance dataset was filtered to include only records with **ErrorCodes** $\neq$ **0**.

First of all, we filer as it is important as it allows us to analyze only vehicles that genuinely exhibit issues. As a code = 0 means "No error," while negative or positive values indicate faults or suspected problems.

```
# Merge all three tables and filter out 'No error' records (ErrorCodes != "0")
# to focus solely on vehicles confirmed or suspected of having a failure.

full_data <- maint %>%
  filter(ErrorCodes != "0") %>% # filters to keep rows where ErrorCodes is not equal "0"
  inner_join(autos, by = "PlateNumber")%>%
  # merges filtered 'maint' with 'autos' based on the matching vehicle 'PlateNumber'
  # to link the trouble records to specific vehicles
  inner_join(engines_unique, by = "EngineModel")
# merges the resulting data with 'engines_unique' base on 'EngineModel'
```

Using **"(filter(ErrorCodes !="0"))", we exclude the records that are classified as "No error"** (0).

After filtering, the 'Maintenance' records were joined with the 'Automobile' and 'Engine' datasets using inner joins. By using inner join, we ensure that only vehicles that appear in all three datasets are included, creating a clean dataset that represents only problematic vehicles.

3.4 Top 5 common troubles are related to

Here, the resulting **"full_data"** contains only confirmed or suspected trouble vehicles. We first use **"count()"** to automatically group all indentical **"Troubles"** descriptions and summarize their occurrences, sorting the results to put the most frequent one first.

Finally, as the final step, we use **"head(5)"** to select and display only the five troubles with the highest counts.

```
# Find the top 5 common troubles by aggregating 'Troubles' to find the most frequent descriptions.
top_troubles <- full_data %>%
  count(Troubles, sort = TRUE) %>%
  # groups the data by every unique entry in the Troubles column
  # then counts how many times each unique trouble appears,
  # then sorts the results from highest to lowest.
  head(5) # selects and keeps only the top 5 rows from the sorted list

print("Top 5 Common Troubles are related to:")
```

```
## [1] "Top 5 Common Troubles are related to:"
```

```
print(top_troubles)
```

```
##             Troubles  n
## 1          Cylinders 38
## 2            Chassis 25
## 3 Ignition (finding) 22
## 4    Noise (finding) 19
## 5         Worn tires 16
```

Here we can see the 5 most common troubles are:

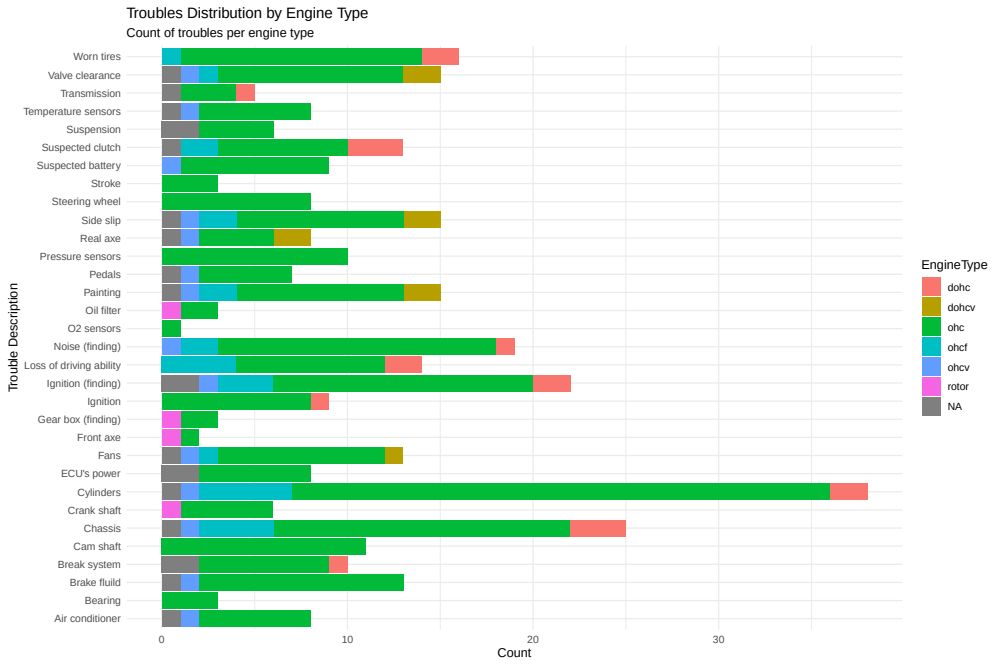
1. Cylinders
2. Chassis
3. Ignition (finding)
4. Noise (finding)
5. Worn tires

3.5 Do the troubles differ between engine types?

To explore whether certain problems are more common in specific engine types, a bar stacked chart would offer best as it would show both the total frequency of each trouble and how those issues are distributed across engine types.

In this visualiation, **"coord_flip()"** was applied to rotate the chart, making long trouble names easier to read and improving interpretability.

```
# A visualization of stacked bar chart to compare the count of troubles across Engine Types.
ggplot(full_data, aes(x = Troubles, fill = EngineType)) +
  # sets 'Troubles' as category to be counted and 'EngineType' to fill (all using 'full_data')
  geom_bar(position = "stack") + # this creates a stack barchat
  coord_flip() + # rotates the chart, making long names easier to read
  labs(title = "Troubles Distribution by Engine Type", # title
       subtitle = "Count of troubles per engine type", # sub-title (to explain further)
       x = "Trouble Description", y = "Count") +  # x and y axis
  theme_minimal() # removes the default gray background
```



From the chart above, we can see that some trouble types are strongly associated with particular engine types. For example, how 'ohc' engines appear to have the largest share of issues across many categories.

Overall, with the purpose to compare between different engine types, it is best if we employ a visualisation to better spot the differences.

Task 4:

Pre-requisite fix

Before any analysis in Task 4, we explicitly assign **"trouble_data <- full_data"**. As previously, 'full_data' was used to filter the maintenance records and then joined. This step is to make sure that we can use a consistent set where only vehicles with confirmed or suspected faults.

```r
# Previously we created "full_data" that filters out the "No error"
# Here, we would want to define the "trouble_data" to ensure the rest of the code runs smoothly.
trouble_data <- full_data
```

Doing this prevents accidental re-use of the full unfiltered maintenance table, clarifies intent to readers of the script, and makes the flow much smoother.

4.1 Most frequent error codes

Here, to find the most frequent code we use the frequency count, **"count(ErrorCodes, sort = TRUE)"** to aggregate and order ErrorCodes from most to least common (with most common at the top). Along with **"slice(1)"** function to isolate and pick the most frequent code away from the rest.

```r
# By grouping error codes and count them to identify the most relevant issue
frequent_error <- trouble_data %>%
  count(ErrorCodes, sort = TRUE) %>% # aggregates the data and sorts them from most to least common
  slice(1) # extracts the single most frequent code

print(paste("Most frequent error code:", frequent_error$ErrorCodes))
```

```
## [1] "Most frequent error code: 1"
```

In our case, the most frequent error code is **'1'**. Practically, this means that among the vehicles flagged as having problems, the diagnostic labeled as 1 appears most frequently.

Because we previously removed **'0'** entries, this is not a "no issue, it just genuinely identifies the single most common problem type that mechanics recorded ("Error Code 1").

4.2 Factors influcencing maintenance methods

As we can see, Maintenance method here are categorized as Adjustment, Replacement, or Urgent care based on the severity.

Therefore, by analysing the vehicle's characteristics (**'BodyStyles'** and **'FuelTypes'**), we can understand if vehicle specifications that influence the repair methods.

Factor 1: BodyStyles

```r
# Analysing 'BodyStyles' vs 'Maintenance'
print("BodyStyles vs Maintenance Methods")
```

```
## [1] "BodyStyles vs Maintenance Methods"
```

```r
# Check the contingency table
## This creates a table to summarize BodyStyles and Methods
table_body_method <- table(trouble_data$BodyStyles, trouble_data$Methods)
print(table_body_method)
```

```
##
##               Adjustment Replacement Urgent care
##   convertible          7           8           0
##   hardtop              2           4           2
##   hatchback           48          63           9
##   sedan               59          89          14
##   wagon               15          24           2
```

The contingency table table **"(trouble_data**$BodyStyles, trouble_data$**Methods)"** lists the raw counts of how many times each maintenance method occurs for each body style. Raw counts let us see absolute workload differences, which is useful for statistical test and the visualization later on.

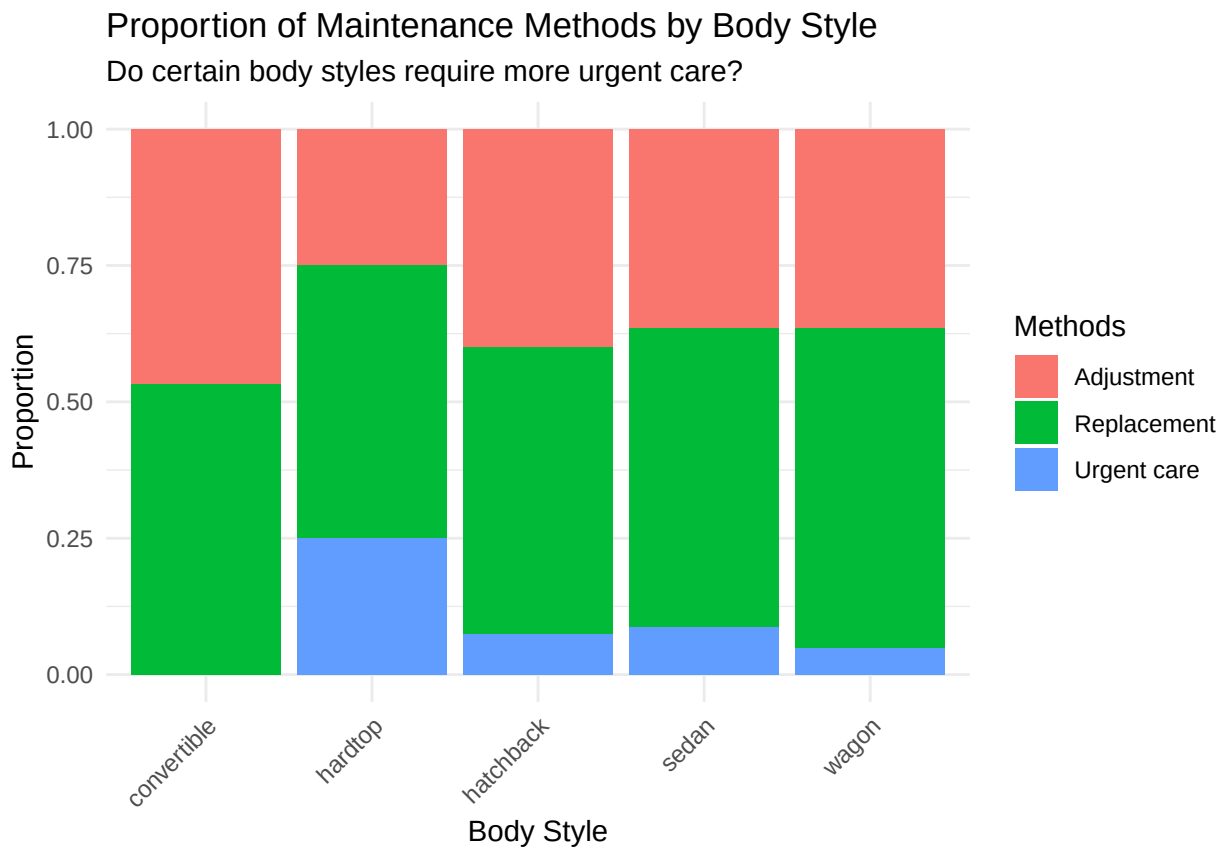
From the table above, we can also see that **"Sedan"** and **"Hatchback"** accounts for the majority of maintenance events (especially those in urgent care). This reflects their higher maintenance frequency in overall vehicle population.

A filled barchart to visualise the proportion of 'Methods' by 'BodyStyle'

In order for us to better visualise the porpotions of maintenance methods over body style, it is best to use a filled bar chart.

This is better to use when the table of comparison is not complex and distribution within the same total are required. As it transforms counts into proportions, allowing us to have a fair comparison across body styles with different sample sizes.

```r
# Plotting Proportion of Maintenance 'Methods' by 'BodyStyle'
ggplot(trouble_data, aes(x = BodyStyles, fill = Methods)) +
  # places 'BodyStyles' on the x axis and fill of the bar segments based on the 'Methods'
  geom_bar(position = "fill") +
  # creates a filled bar chart, allows us to compare proportions across groups of different sizes
  labs(title = "Proportion of Maintenance Methods by Body Style",
       subtitle = "Do certain body styles require more urgent care?",
       y = "Proportion", x = "Body Style") +
  theme_minimal() + # removes the default gray background
  # the one below rotates the labels 45 degrees to prevent overlapping
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



From the filled barchart above, we can see and confirm the following elements:

- **Sedans** and **hatchbacks** show slightly larger proportional shares of 'Replacement' and 'Urgent care' than other body styles. That implies that these styles have somewhat greater severity in maintenance.
- **Convertibles** show no urgent care in this dataset. Meaning that, either they experience fewer severe failures, or there are too few convertibles to observe.
- **Hardtops** show a relatively high urgent-care proportion relative to their small counts. This could indicate that when hardtops fail, the problems are more often severe.

Factor 2: FuelTypes

```
# Analysing 'FuelTypes' vs 'Maintenance'
print("FuelTypes vs Maintenance Methods")
```

```
## [1] "FuelTypes vs Maintenance Methods"
```

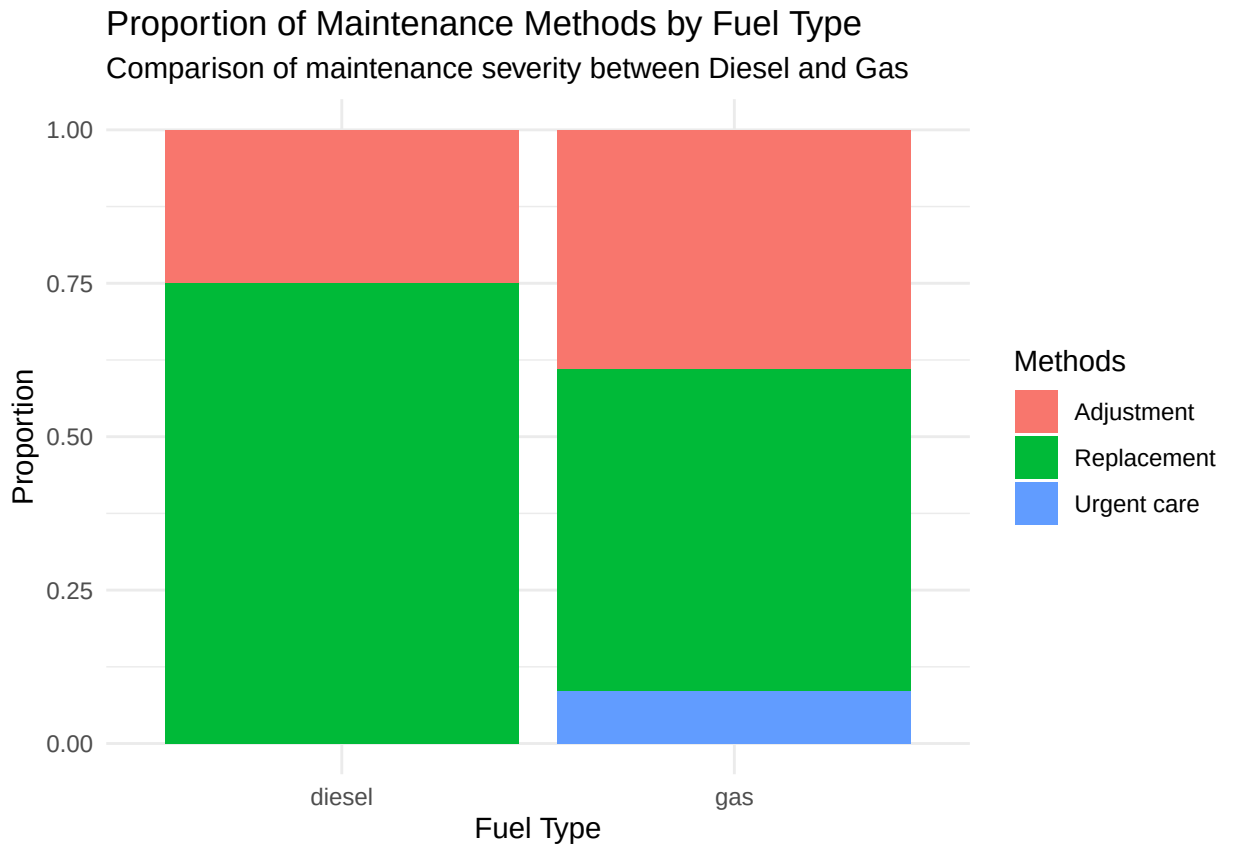
```
# Check the contingency table
table_fuel_method <- table(trouble_data$FuelTypes, trouble_data$Methods)
# creates the tabel for 'FuelTypes' and 'Methods' in 'trouble_data'
print(table_fuel_method)
```

```
##
##          Adjustment Replacement Urgent care
##   diesel          7          21           0
##   gas           124         167          27
```

The table above regarding fuel types, shows us counts of repair methods split by fuel type. As we can see, gas vehicles overwhelmingly account for the total number of record.

Because gas vehicles are numerically more common, raw counts alone will bias any interpretation. By converting to proportions using **"(position ="fill")"** (creates a filled bar chart) will help us to have a better side by side comparison between diesel and gas vehicles.

```
# Plotting Proportion of Maintenance 'Methods' by 'FuelTypes'
ggplot(trouble_data, aes(x = FuelTypes, fill = Methods)) +
  # places 'FuelTypes' on the x axis and fill of the bar segments based on the 'Methods'
  geom_bar(position = "fill") + # creates the fill bar chart
  labs(title = "Proportion of Maintenance Methods by Fuel Type",
       # title
       subtitle = "Comparison of maintenance severity between Diesel and Gas",
       # sub-title to explain the chart
       y = "Proportion", x = "Fuel Type") + # names the x and y axis
  theme_minimal() # removes the default gray background
```



Similar to Factor 1, this proportional plot is used to compare the relative severity of maintenance required between the two fuel types, compensating for the large difference in sample size. From this chart, we can see the following trends:

- **Diesel vehicles** shows no urgent care in the sample and a larger share of replacements. This pattern may reflect that when diesel engines fail, they often owing to planned replacement rather than emergency repairs.
- **Gasoline vehicles**, on the other hand, include a small but visible proportion of urgent care events. This indicates that gas cars may experience more unexpected failures that would require urgent care.

Overall conclusion

From these patterns, we can say that diesel engines are typically built for durability and operate with different characteristics, which lead to different failure modes than gasoline engines. Also, diesel cars maintenance severity does tends to be lower compared to gasoline cars owing to their mechanical build.

Task 5

Submit Git

Process used to submit:

1. Cloning the repository to local desktop
2. Entering the project directory on Git Bash
3. Include all assignment files (RMarkdown, PDF, Scripts). Before that, I need to copy paste the files into the local folder then proceed the following code on Git Bash
4. Commit the changes with a message to note the changes
5. Push the submission to GitHub

```
git clone [https://github.com/21000847/AP_A1_Nguyen-Hoang-Nhat-Linh_22084016.git]

cd AP_A1_Nguyen-Hoang-Nhat-Linh_22084016

git add .

git commit -m "Submitting assignment files"

git push
```

To manage the version control for this assignment, the following Git commands in Git Bash shows my progress for each task:

```
# Initial setup
git init
git add .
git commit -m "Initial commit: Load datasets and setup RMarkdown"

# Task 1: Data Cleaning
git add .
git commit -m "Task 1.1: Inspect data, find '?' and replace with NA"
git add .
git commit -m "Loading datasets"
git add .
```

```
git commit -m "Task 1.1: Count ? and replace with NA".
git add .
git commit -m "Convert categorical variables BodyStyles, FuelTypes, ErrorCodes to factors"
git add .
git commit -m "Replace the missing values in column Horsepower with the median horsepower"
git add
git commit -m "Horsepower distribution visualisation"
git push

# Task 2: Engine Analysis
git add .
git commit -m "Task 2.1:  1 The code to analyse the distribution of the horsepower across the engine ty
git add .
git commit -m "Task 2.2: The distribution of the horsepower across the groups"
git add .
git push

# Task 3: Fuel & Troubles Analysis
git add .
git commit -m "Task 3: Clean data"
git add .
git commit -m "Task 3.2: Do diesel cars have higher average CityMpg than gasoline cars? Provide statisti
git add .
git commit -m "Task 3.2: How does DriveWheels affect fuel efficiency (CityMpg and HighwayMpg)?"
git add .
git commit -m "Task 3.3: Filter out those engines in the dataset that have trouble or are suspected of
git add .
git commit -m "Task 3.4: Top 5 common troubles are related to"
git add .
git commit -m "Task 3.5: Do the troubles differ between engine types?"
git push

# Task 4: Maintenance Analysis
git add .
git commit -m "Task 4.1: Most frequent error codes"
git add .
git commit -m "Task 4.2: 2 Factors influcencing maintenance methods"
git push

# Task 5: Final Submission
git add .
git commit -m "Insert Git, Knit and submission"
git push
```