Stat Computing Final Code

Ishaan Aditya Janhavi Yaswanth Lokesh

2025-04-16

Load all the packages

```
# For model-1
library(readxl)    # for reading Excel files
library(lubridate) # Date/time handling

##
## Attaching package: 'lubridate'

## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union

library(caret) # Modeling utility (train-test split, confusion matrix)

## Loading required package: ggplot2

## Loading required package: lattice

library(pROC) # ROC and AUC calculations

## Type 'citation("pROC")' for a citation.

##
## Attaching package: 'pROC'

## The following objects are masked from 'package:stats':
##
##     cov, smooth, var

#library(Metrics)
library(writexl) # For exporting data

## Warning: package 'writexl' was built under R version 4.4.3

library(reshape2) # For reshaping data (used for correlation heatmap)
library(patchwork) # For arranging multiple plots

## Warning: package 'patchwork' was built under R version 4.4.3

# For model-2
library(dplyr)        # For data manipulation

##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

library(tidyr)        # For tidying data

##
## Attaching package: 'tidyr'

## The following object is masked from 'package:reshape2':
##
##     smiths

library(ggplot2)      # For creating visualizations
library(factoextra)   # For PCA visualization tools

## Warning: package 'factoextra' was built under R version 4.4.3

## Welcome! Want to learn more? See two factoextra-related books at
https://goo.gl/ve3WBa

library(ggrepel)      # For non-overlapping text labels in plots

## Warning: package 'ggrepel' was built under R version 4.4.3

library(scales)       # For scale functions in visualizations
#library(fmsb)




# For Model-3
library(zoo) # Rolling means

## Warning: package 'zoo' was built under R version 4.4.3

##
## Attaching package: 'zoo'

## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric

library(xgboost) # XGBoost model

## Warning: package 'xgboost' was built under R version 4.4.3

##
## Attaching package: 'xgboost'
```

```
## The following object is masked from 'package:dplyr':
##
##     slice

library(tibble)
library(forecast)

## Warning: package 'forecast' was built under R version 4.4.3

## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo

library(grid)
library(gridExtra) # Arranging plots

## Warning: package 'gridExtra' was built under R version 4.4.3

##
## Attaching package: 'gridExtra'

## The following object is masked from 'package:dplyr':
##
##     combine

library(cli) # Command-line styled outputs

# ================================================
# FUNCTION DEFINITIONS FOR DATA LOADING
# ================================================

# (a) Function to load a specific sheet from an Excel file.
load_dataset <- function(file_path, sheet) {
  # Read the specified sheet from the given Excel file.
  data <- read_excel(file_path, sheet = sheet)
  return(data)
}

# (b) Function to remove timezone attributes from POSIXct columns.
remove_timezone <- function(df) {
  # Loop over each column in the data frame.
  for (col in names(df)) {
    # If the column is of POSIXct type and contains a timezone attribute,
    # set the timezone to UTC and then remove the attribute.
    if (inherits(df[[col]], "POSIXct") && !is.null(attr(df[[col]], "tzone")))
{
      df[[col]] <- as.POSIXct(df[[col]], tz = "UTC")
      attr(df[[col]], "tzone") <- NULL
    }
  }
  return(df)
}
```

```r
# (c) Function to process the Results sheet.
preprocess_results_data <- function(results) {
  # If the "Abbreviation" column exists, create a new "Driver" column as
character.
  if ("Abbreviation" %in% colnames(results)) {
    results <- results %>% mutate(Driver = as.character(Abbreviation))
  }
  # Standardize key identifier columns as character.
  results <- results %>% mutate(
    Year = as.character(Year),
    RoundNumber = as.character(RoundNumber),
    EventName = as.character(EventName)
  )
  # Mark race winners with a binary label: 1 if Position equals 1, else 0.
  results <- results %>% mutate(Winner = ifelse(Position == 1, 1, 0))

  # Remove any timezone attributes (if present).
  results <- remove_timezone(results)
  return(results)
}

# (d) Function to process the Laps sheet.
preprocess_laps_data <- function(laps) {
  # Convert key columns to character for consistency.
  laps <- laps %>% mutate(
    Driver = as.character(Driver),
    Year = as.character(Year),
    RoundNumber = as.character(RoundNumber),
    EventName = as.character(EventName)
  )
  # If LapTime_sec does not exist but LapTime does, convert LapTime to
numeric.
  if (!"LapTime_sec" %in% colnames(laps) && "LapTime" %in% colnames(laps)) {
    # This conversion assumes LapTime is directly interpretable as numeric.
    laps <- laps %>% mutate(LapTime_sec = as.numeric(LapTime))
  }
  return(laps)
}

# (e) Function to process the Weather sheet.
preprocess_weather_data <- function(weather) {
  # Standardize key identifier columns.
  weather <- weather %>% mutate(
    Year = as.character(Year),
    RoundNumber = as.character(RoundNumber),
    EventName = as.character(EventName)
  )
  # Remove timezone attributes.
  weather <- remove_timezone(weather)
```

```r
  return(weather)
}

# (f) Function to summarize lap statistics by event and driver.
summarize_lap_stats <- function(laps) {
  # Filter out rows with missing lap time values.
  laps_valid <- laps %>% filter(!is.na(LapTime_sec))

  # Group by Year, RoundNumber, EventName, and Driver; then calculate:
  #   - Average lap time, Best lap time, and Total number of laps.
  laps_summary <- laps_valid %>%
    group_by(Year, RoundNumber, EventName, Driver) %>%
    summarise(
      AvgLapTime = mean(LapTime_sec, na.rm = TRUE),
      BestLapTime = min(LapTime_sec, na.rm = TRUE),
      NumLaps = n(),
      .groups = "drop"
    )

  # Remove any timezone attributes from the summary.
  laps_summary <- remove_timezone(laps_summary)
  return(laps_summary)
}

# (g) Function to summarize weather by event.
summarize_weather <- function(weather) {
  # Group by event identifiers and take the first available record for each
event.
  weather_summary <- weather %>%
    group_by(Year, RoundNumber, EventName) %>%
    slice_head(n=1) %>%  # take the first available reading per event
    ungroup()

  # Remove timezone attributes.
  weather_summary <- remove_timezone(weather_summary)
  return(weather_summary)
}

# (h) Function to merge Results, Lap summary, and Weather data.
merge_for_analysis <- function(results, laps_summary, weather_summary) {
  # Perform a left join on the Results data with lap statistics using common
keys,
  # followed by joining the Weather summary data.
  analysis_data <- results %>%
    left_join(laps_summary, by = c("Year", "RoundNumber", "EventName",
"Driver")) %>%
    left_join(weather_summary, by = c("Year", "RoundNumber", "EventName"))
  return(analysis_data)
}
```

```r
# (i) Function to filter complete records for modeling.
filter_complete_records <- function(analysis_data) {
  # Only keep records with non-missing values for critical variables.
  analysis_data_complete <- analysis_data %>%
    filter(!is.na(Winner) &
             !is.na(GridPosition) &
             !is.na(AvgLapTime) &
             !is.na(BestLapTime) &
             !is.na(NumLaps) &
             !is.na(AirTemp) &
             !is.na(TrackTemp))
  return(analysis_data_complete)
}

# ---- Additional Common Preprocessing Function ----
# This function can be used for various sheets by specifying the sheet_type.
preprocess_common <- function(data, sheet_type = "Generic") {
  # Convert common keys to character, if available.
  common_keys <- c("Year", "RoundNumber", "EventName")
  data <- data %>% mutate(across(any_of(common_keys), as.character))

  if(sheet_type == "EventSchedule") {
    # Attempt to parse date/time if a "Date/Time" column exists.
    if("Date/Time" %in% names(data)) {
      data <- data %>% mutate(DateTime = as.POSIXct(`Date/Time`, format =
"%Y-%m-%d %H:%M:%S", tz = "UTC"))
    }
  } else if(sheet_type == "Results") {
    # Convert driver information to character using common field names.
    if("Driver" %in% names(data)) {
      data <- data %>% mutate(Driver = as.character(Driver))
    } else if("Driver Name" %in% names(data)) {
      data <- data %>% mutate(Driver = as.character(`Driver Name`))
    }
    # Create the Winner variable based on finishing position.
    if("Finishing Positions" %in% names(data)) {
      data <- data %>% mutate(Winner = ifelse(`Finishing Positions` == 1, 1,
0))
    } else if("Position" %in% names(data)) {
      data <- data %>% mutate(Winner = ifelse(Position == 1, 1, 0))
    }
  } else if(sheet_type == "Laps") {
    # Ensure the Driver column is character and convert LapTime to numeric if
needed.
    if("Driver" %in% names(data)) {
      data <- data %>% mutate(Driver = as.character(Driver))
    }
    if(!"LapTime_sec" %in% names(data) && "LapTime" %in% names(data)) {
      data <- data %>% mutate(LapTime_sec = as.numeric(LapTime))
```

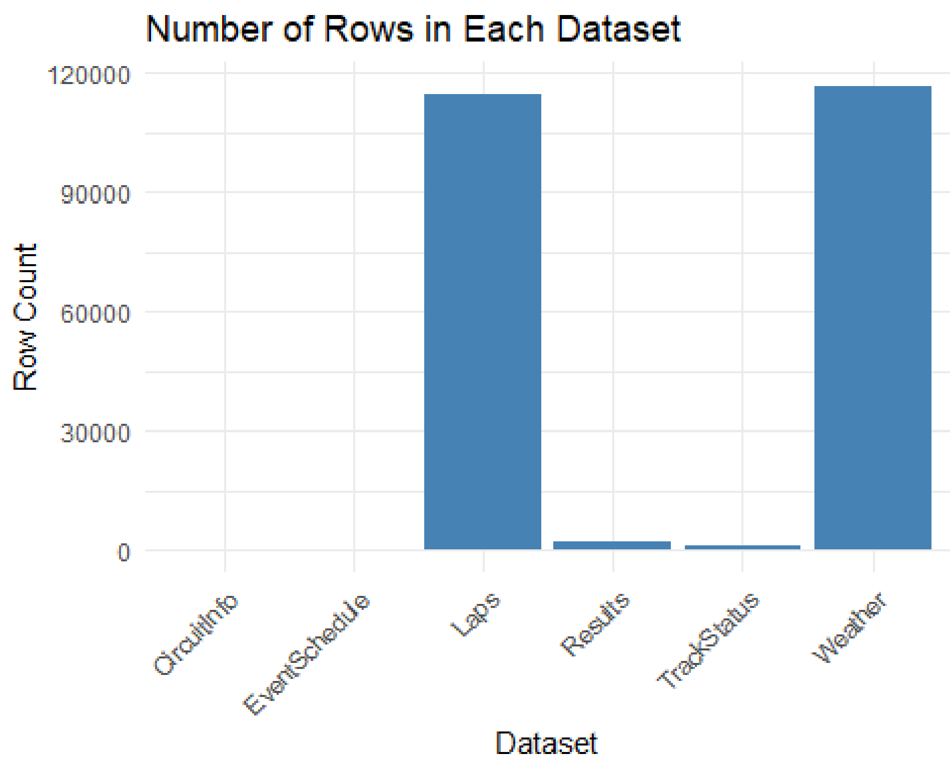
```
    }
  } else if(sheet_type == "Weather") {
    # Additional weather-specific processing can be added here if needed.
    data <- data
  } else if(sheet_type == "TrackStatus") {
    # Convert "Track Status" to a factor.
    if("Track Status" %in% names(data)) {
      data <- data %>% mutate(`Track Status` = as.factor(`Track Status`))
    }
  } else if(sheet_type == "CircuitInfo") {
    data <- data  # Additional processing for circuit info can be added here.
  }

  # Remove timezone attributes.
  data <- remove_timezone(data)
  return(data)
}

# ================================================
# IMPORTING DATA & PREPROCESSING
# ================================================

# Define the path to your Excel file.
file_path <- "fastf1_full_data_2020_2024.xlsx"

# Load all sheets (adjust sheet names if necessary) using the load_dataset
function.
event_schedule_raw <- load_dataset(file_path, sheet = "EventSchedule")
results_raw        <- load_dataset(file_path, sheet = "Results")
laps_raw           <- load_dataset(file_path, sheet = "Laps")
weather_raw        <- load_dataset(file_path, sheet = "Weather")
trackstatus_raw    <- load_dataset(file_path, sheet = "TrackStatus")
circuitinfo_raw    <- load_dataset(file_path, sheet = "CircuitInfo")

# Apply common preprocessing to each sheet.
event_schedule_clean <- preprocess_common(event_schedule_raw, sheet_type =
"EventSchedule")
results_clean        <- preprocess_common(results_raw, sheet_type =
"Results")
laps_clean           <- preprocess_common(laps_raw, sheet_type = "Laps")
weather_clean        <- preprocess_common(weather_raw, sheet_type =
"Weather")
trackstatus_clean    <- preprocess_common(trackstatus_raw, sheet_type =
"TrackStatus")
circuitinfo_clean    <- preprocess_common(circuitinfo_raw, sheet_type =
"CircuitInfo")


# For the Laps sheet, retain only rows with valid LapTime_sec values.
```

```r
laps_clean <- laps_clean %>% filter(!is.na(LapTime_sec))

# ==================================================
# EXPORT THE FULLY PREPROCESSED DATA
# ==================================================

# Write the preprocessed sheets into a new Excel workbook.
export_file <- "fully_preprocessed_data.xlsx"
write_xlsx(
  list(
    EventSchedule = event_schedule_clean,
    Results       = results_clean,
    Laps          = laps_clean,
    Weather       = weather_clean,
    TrackStatus   = trackstatus_clean,
    CircuitInfo   = circuitinfo_clean
  ),
  path = export_file
)
cat("Fully preprocessed data has been saved to", export_file, "\n")
```

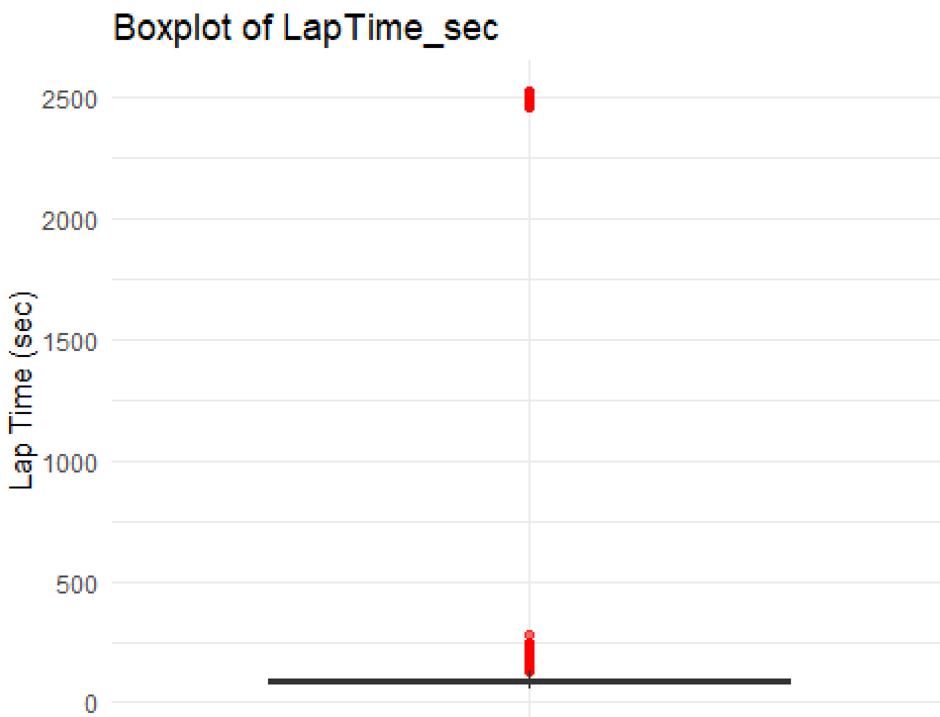
```
## Fully preprocessed data has been saved to fully_preprocessed_data.xlsx
```

```r
# For the Laps sheet, keep only rows with valid lap time values
laps_clean <- laps_clean %>% filter(!is.na(LapTime_sec))

# ==================================================
# EDA PLOTS & ARRANGEMENT
# ==================================================
# Reload the cleaned sheets
file_path = "fully_preprocessed_data.xlsx"
df_event_schedule <- load_dataset(file_path, sheet = "EventSchedule")
df_results        <- load_dataset(file_path, sheet = "Results")
df_laps           <- load_dataset(file_path, sheet = "Laps")
df_weather        <- load_dataset(file_path, sheet = "Weather")
df_track_status   <- load_dataset(file_path, sheet = "TrackStatus")
df_circuit_info   <- load_dataset(file_path, sheet = "CircuitInfo")

# (A) Dataset Sizes
df_sizes <- data.frame(
  Dataset  =
c("EventSchedule","Results","Laps","Weather","TrackStatus","CircuitInfo"),
  RowCount = c(
    nrow(df_event_schedule),
    nrow(df_results),
    nrow(df_laps),
    nrow(df_weather),
    nrow(df_track_status),
    nrow(df_circuit_info)
  )
```

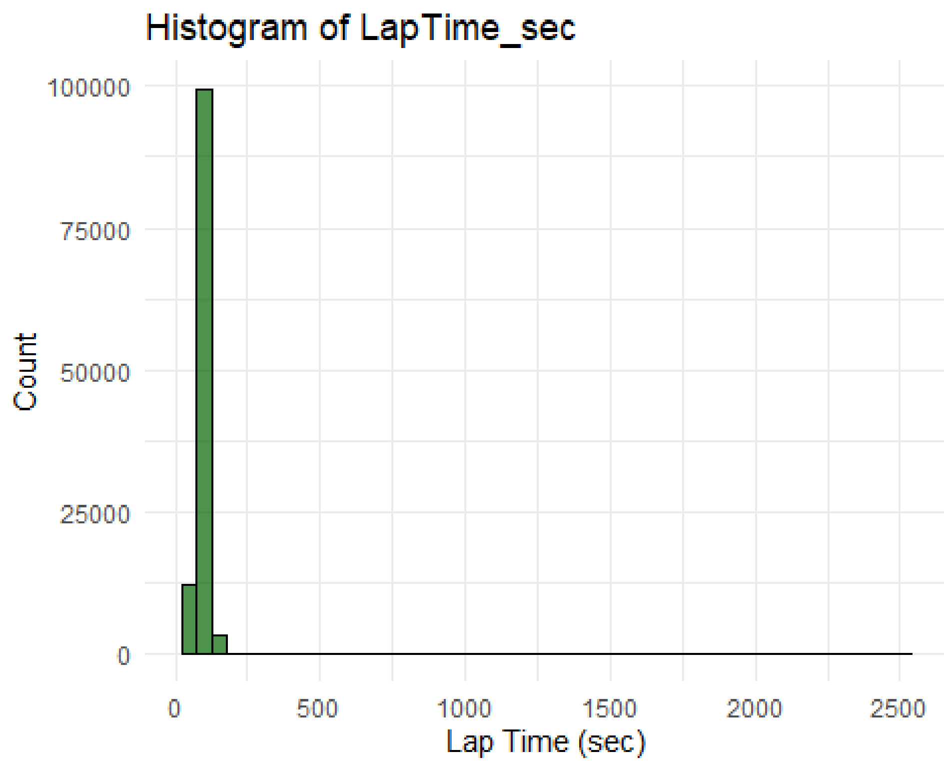
```
)
p_dataset_sizes <- ggplot(df_sizes, aes(Dataset, RowCount)) +
  geom_bar(stat="identity", fill="steelblue") +
  labs(title="Number of Rows in Each Dataset", x="Dataset", y="Row Count") +
  theme_minimal() + theme(axis.text.x=element_text(angle=45,hjust=1))

print(p_dataset_sizes)
```



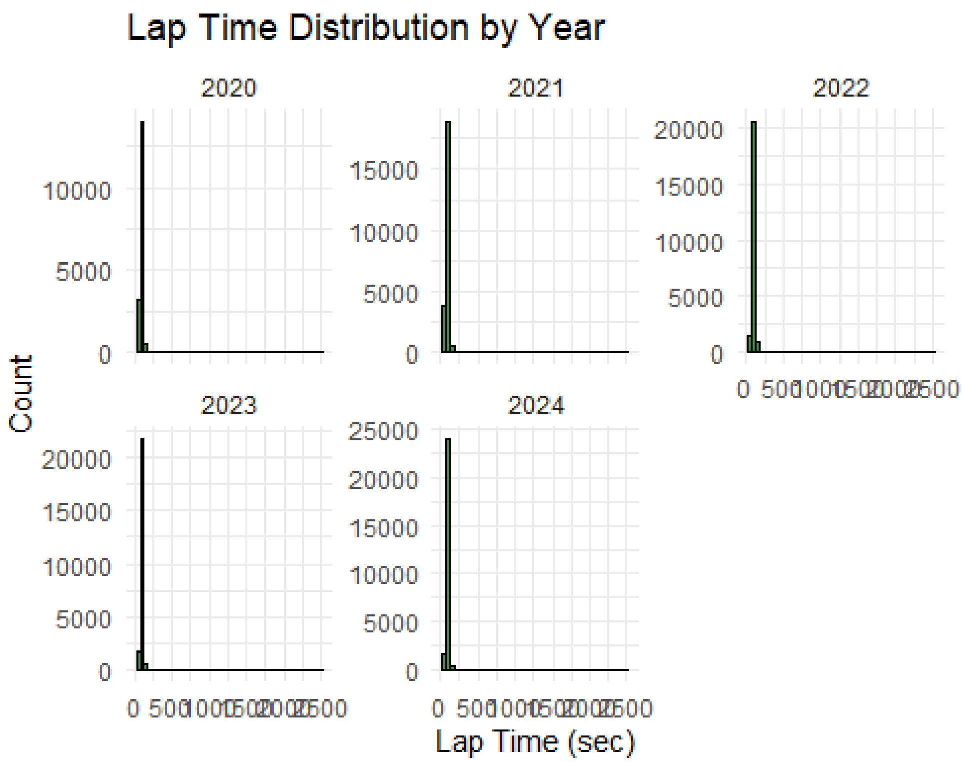

```
ggsave("EDA_plots/Dataset_Sizes.png", p_dataset_sizes, width=8, height=6)

# (B1) Boxplot of LapTime_sec
p_box_laptime <- ggplot(df_laps, aes("", LapTime_sec)) +
  geom_boxplot(outlier.color="red", outlier.alpha=0.6) +
  labs(title="Boxplot of LapTime_sec", x=NULL, y="Lap Time (sec)") +
  theme_minimal()

print(p_box_laptime)
```



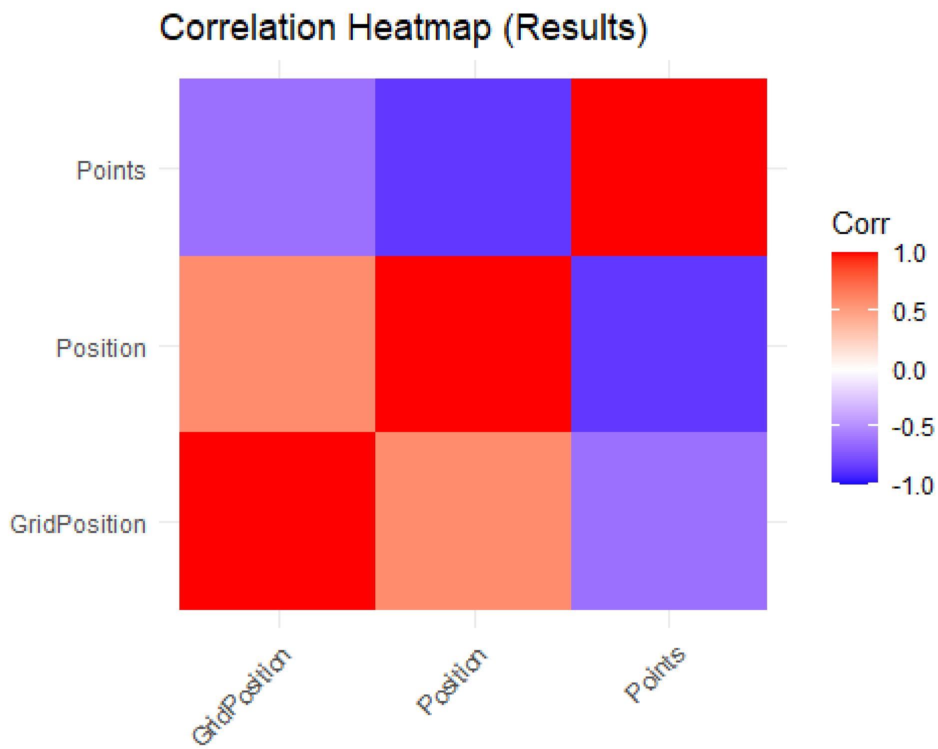

```r
ggsave("EDA_plots/LapTime_Boxplot.png", p_box_laptime, width=6, height=4)

# (B2) Histogram of LapTime_sec
p_hist_laptime <- ggplot(df_laps, aes(LapTime_sec)) +
  geom_histogram(bins=50, fill="darkgreen", alpha=0.7, color="black") +
  labs(title="Histogram of LapTime_sec", x="Lap Time (sec)", y="Count") +
  theme_minimal()

print(p_hist_laptime)
```



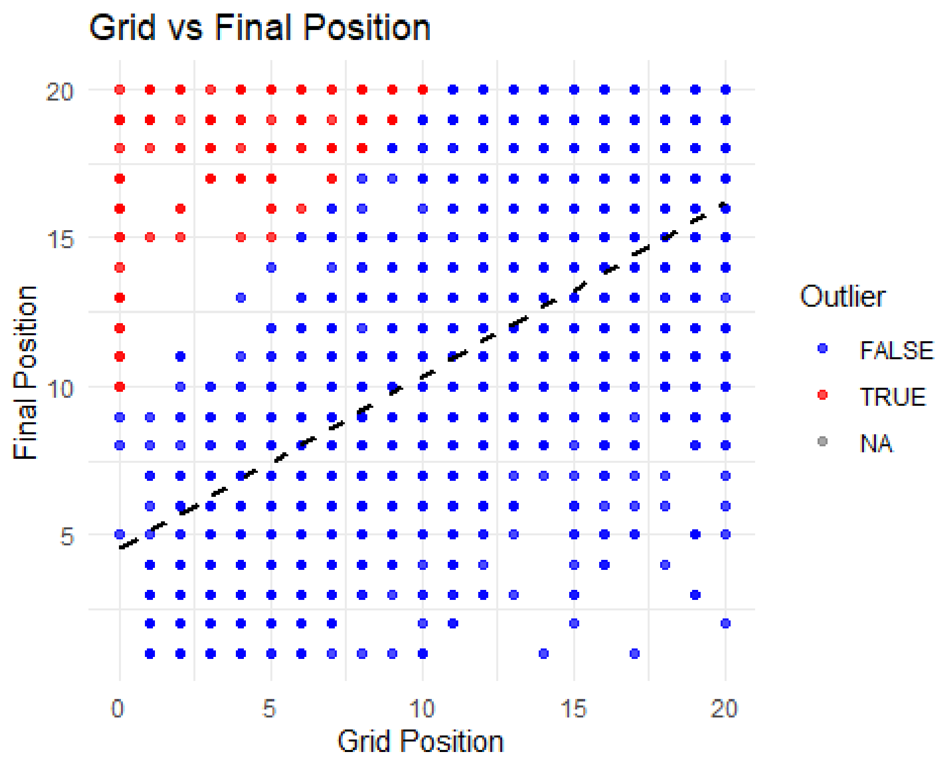

```
ggsave("EDA_plots/LapTime_Histogram.png", p_hist_laptime, width=6, height=4)

# (B3) Faceted Histogram by Year
p_lap_time_year <- ggplot(df_laps, aes(LapTime_sec)) +
  geom_histogram(bins=50, fill="darkgreen", alpha=0.7, color="black") +
  facet_wrap(~ Year, scales="free_y") +
  labs(title="Lap Time Distribution by Year", x="Lap Time (sec)", y="Count")
+
  theme_minimal()

print(p_lap_time_year)
```



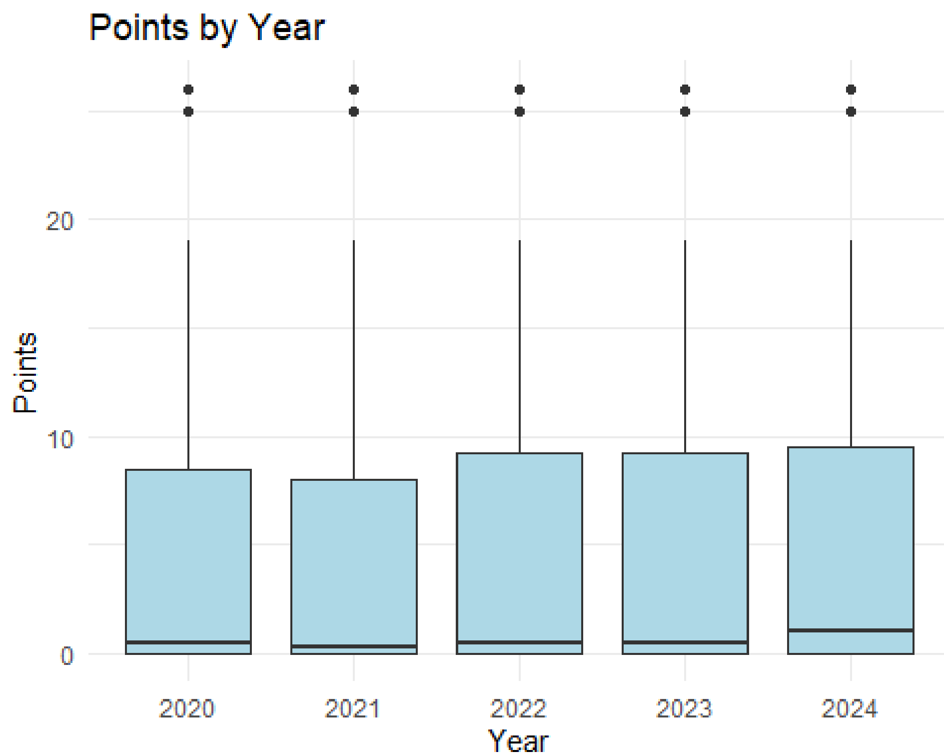

```r
ggsave("EDA_plots/LapTime_By_Year.png", p_lap_time_year, width=8, height=6)

# (C1) Correlation Heatmap (Results)
num_df    <- df_results %>% select(GridPosition, Position, Points) %>%
filter(complete.cases(.))
cor_matrix<- cor(num_df)
melted_cor<- reshape2::melt(cor_matrix)
p_corr <- ggplot(melted_cor, aes(Var1, Var2, fill=value)) +
  geom_tile() +
  scale_fill_gradient2(low="blue", high="red", mid="white", midpoint=0,
limit=c(-1,1)) +
  labs(title="Correlation Heatmap (Results)", fill="Corr") +
  theme_minimal() + theme(axis.text.x=element_text(angle=45,hjust=1),
                          axis.title=element_blank())

print(p_corr)
```



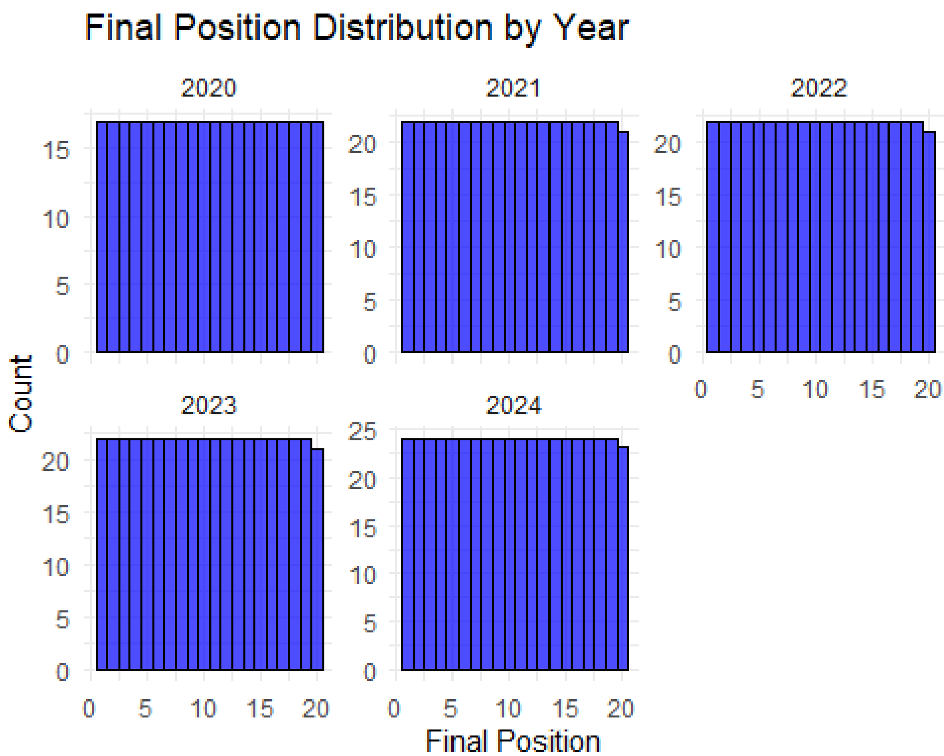

```
ggsave("EDA_plots/Results_Correlation_Heatmap.png", p_corr, width=6,
height=5)

# (C2) Grid vs Final Position
p_grid_vs_position <- df_results %>%
  mutate(Outlier = (Position - GridPosition >= 10)) %>%
  ggplot(aes(GridPosition, Position, color=Outlier)) +
    geom_point(alpha=0.7) +
    geom_smooth(method="lm", se=FALSE, color="black", linetype="dashed") +
    scale_color_manual(values=c("FALSE"="blue","TRUE"="red")) +
    labs(title="Grid vs Final Position", x="Grid Position", y="Final
Position") +
    theme_minimal()

print(p_grid_vs_position)

## `geom_smooth()` using formula = 'y ~ x'

## Warning: Removed 3 rows containing non-finite outside the scale range
## (`stat_smooth()`).

## Warning: Removed 3 rows containing missing values or values outside the
scale range
## (`geom_point()`).
```



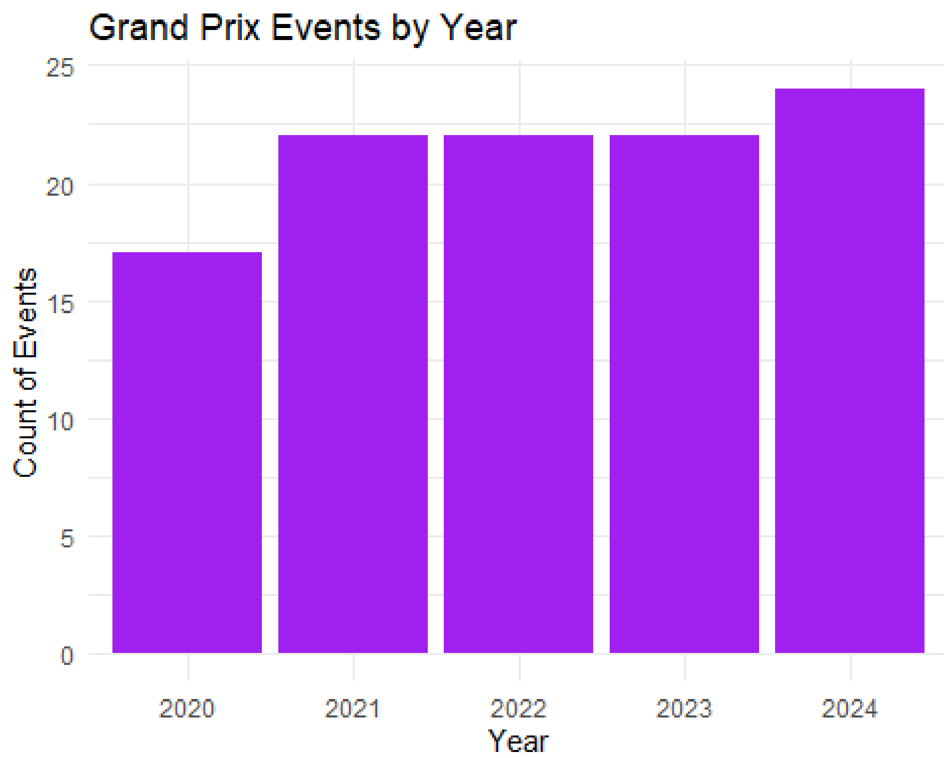

```
ggsave("EDA_plots/Grid_vs_FinalPosition.png", p_grid_vs_position, width=6,
height=4)

## `geom_smooth()` using formula = 'y ~ x'

## Warning: Removed 3 rows containing non-finite outside the scale range
(`stat_smooth()`).
## Removed 3 rows containing missing values or values outside the scale range
## (`geom_point()`).

# (C3) Points by Year
p_box_points_year <- ggplot(df_results, aes(factor(Year), Points)) +
  geom_boxplot(fill="lightblue") +
  labs(title="Points by Year", x="Year", y="Points") +
  theme_minimal()

print(p_box_points_year)
```



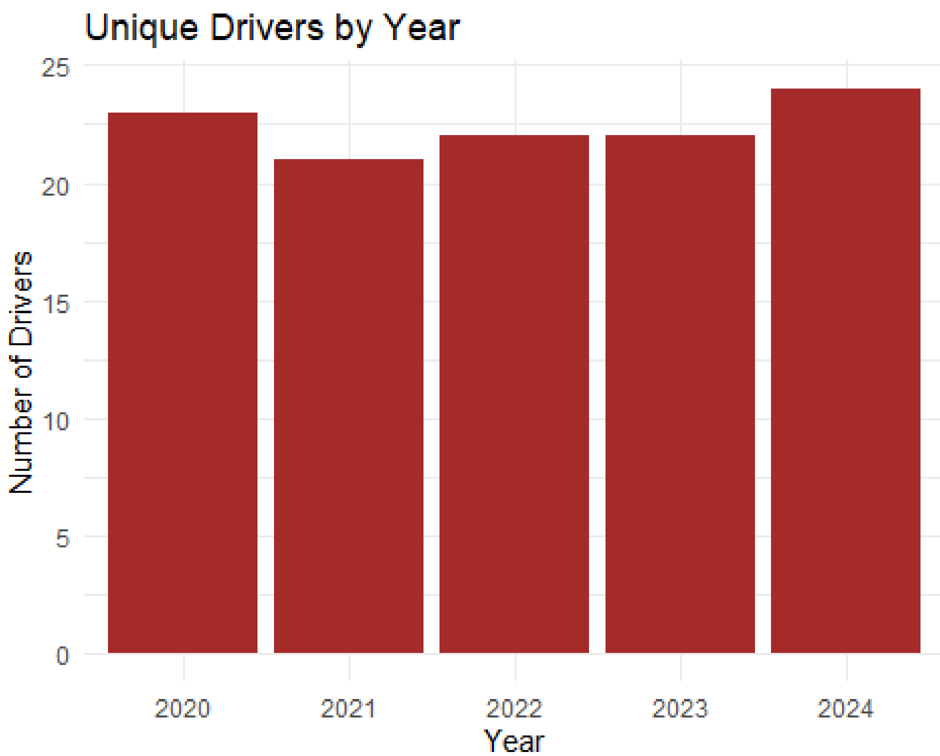

```
ggsave("EDA_plots/Points_By_Year_Boxplot.png", p_box_points_year, width=6,
height=4)

# (C4) Final Position Dist by Year
p_finish_pos <- ggplot(df_results, aes(as.numeric(Position))) +
  geom_histogram(binwidth=1, fill="blue", alpha=0.7, color="black") +
  facet_wrap(~ Year, scales="free_y") +
  labs(title="Final Position Distribution by Year", x="Final Position",
y="Count") +
  theme_minimal()

print(p_finish_pos)

## Warning: Removed 3 rows containing non-finite outside the scale range
## (`stat_bin()`).
```



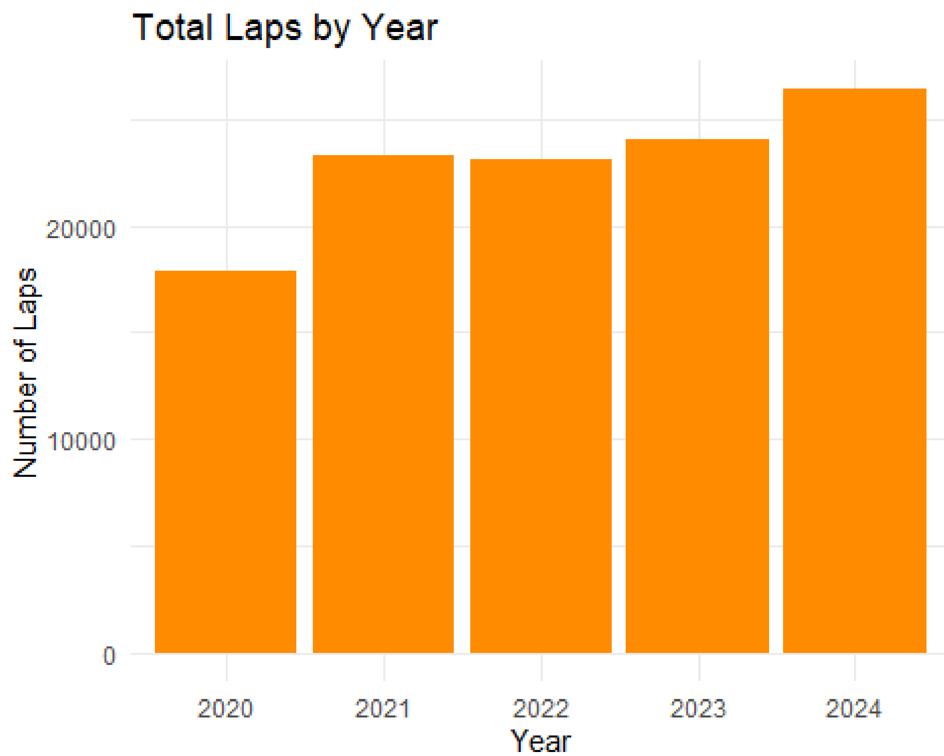

```
ggsave("EDA_plots/Final_Position_Distribution_By_Year.png", p_finish_pos,
width=8, height=6)

## Warning: Removed 3 rows containing non-finite outside the scale range
## (`stat_bin()`).

# (D1) Grand Prix Events by Year
df_event_count <- df_event_schedule %>% group_by(Year) %>%
summarize(Count=n_distinct(EventName))
p_event_count <- ggplot(df_event_count, aes(factor(Year), Count)) +
  geom_bar(stat="identity", fill="purple") +
  labs(title="Grand Prix Events by Year", x="Year", y="Count of Events") +
  theme_minimal()

print(p_event_count)
```



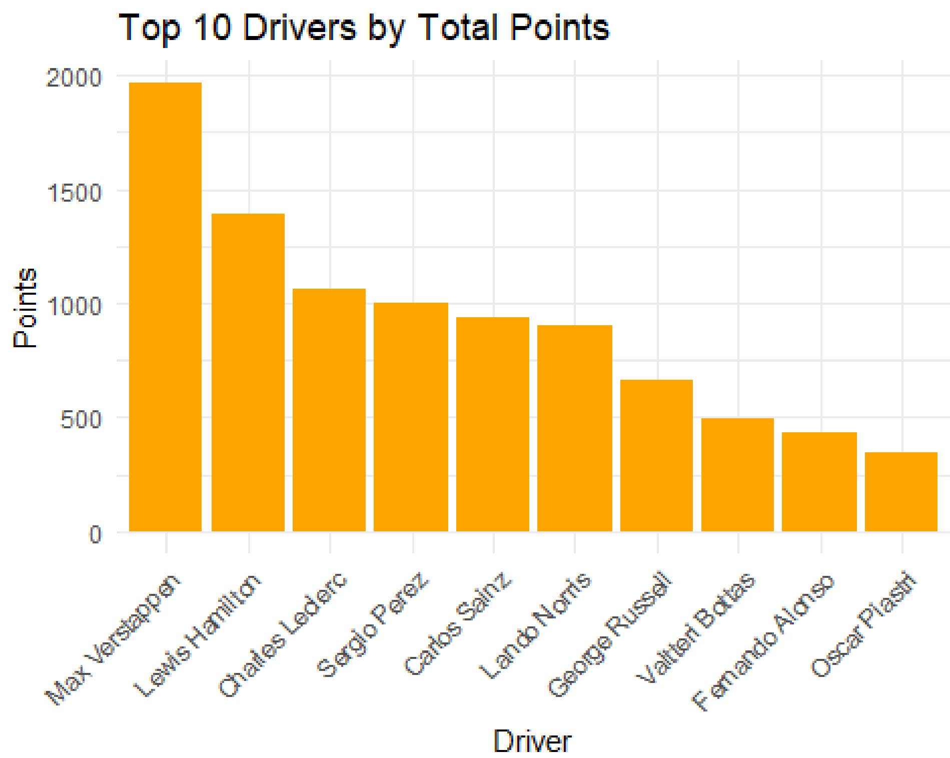

```
ggsave("EDA_plots/GrandPrix_Events_By_Year.png", p_event_count, width=6,
height=4)

# (D2) Unique Drivers by Year
df_unique_drivers <- df_results %>% group_by(Year) %>%
summarize(Unique=n_distinct(FullName))
p_unique_drivers <- ggplot(df_unique_drivers, aes(factor(Year), Unique)) +
  geom_bar(stat="identity", fill="brown") +
  labs(title="Unique Drivers by Year", x="Year", y="Number of Drivers") +
  theme_minimal()

print(p_unique_drivers)
```




```r
ggsave("EDA_plots/Unique_Drivers_By_Year.png", p_unique_drivers, width=6,
height=4)

# (D3) Total Laps by Year
df_lap_count <- df_laps %>% group_by(Year) %>% summarize(TotalLaps=n())
p_total_laps <- ggplot(df_lap_count, aes(factor(Year), TotalLaps)) +
  geom_bar(stat="identity", fill="darkorange") +
  labs(title="Total Laps by Year", x="Year", y="Number of Laps") +
  theme_minimal()

print(p_total_laps)
```




```
ggsave("EDA_plots/Total_Laps_By_Year.png", p_total_laps, width=6, height=4)

# (D4) Top 10 Drivers by Total Points
df_driver_points <- df_results %>%
  mutate(Points = as.numeric(Points)) %>%
  group_by(FullName) %>%
  summarize(Total=sum(Points,na.rm=TRUE)) %>%
  arrange(desc(Total)) %>%
  slice_head(n=10)

p_top10_drivers <- ggplot(df_driver_points, aes(reorder(FullName, -Total),
Total)) +
  geom_bar(stat="identity", fill="orange") +
  labs(title="Top 10 Drivers by Total Points", x="Driver", y="Points") +
  theme_minimal() +
  theme(axis.text.x=element_text(angle=45,hjust=1))

print(p_top10_drivers)
```

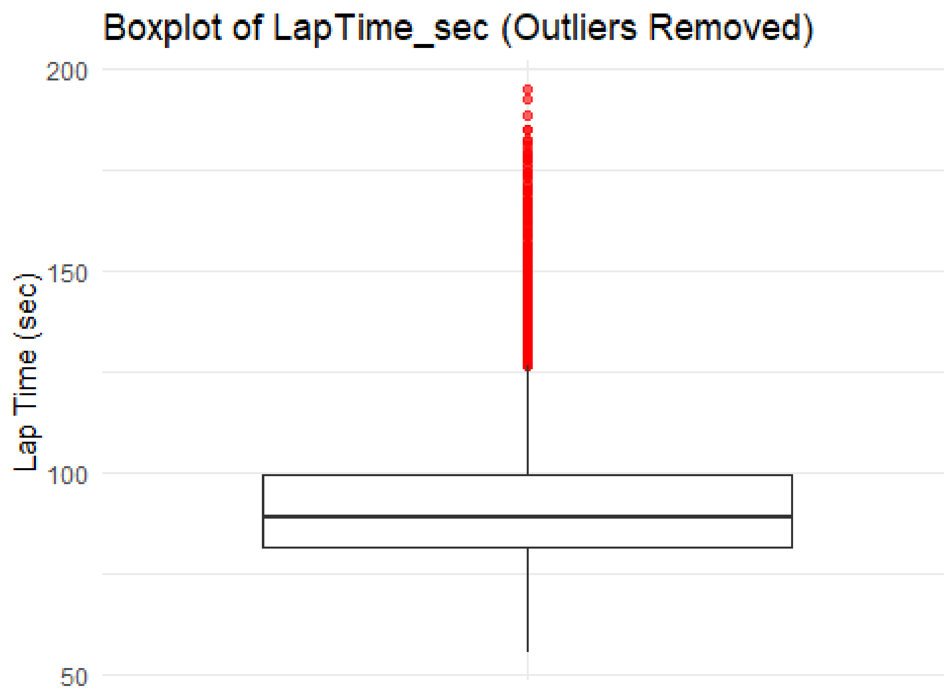



```
ggsave("EDA_plots/Top10_Drivers_By_Total_Points.png", p_top10_drivers,
width=8, height=6)

# ====================================================
# REMOVE OUTLIERS IN LAPTIME_SEC AND RE-PLOT
# ====================================================

# 1. Identify outliers using IQR method
# ------------------------------------
df_laps_filtered <- df_laps %>%
  filter(LapTime_sec >= 40, LapTime_sec <= 200)

cat("Number of laps remaining after applying 40-200 sec filter:",
    nrow(df_laps_filtered), "\n")

## Number of laps remaining after applying 40-200 sec filter: 114586

# 3. Remove corresponding weather records.
# Assume the Weather data has a LapNumber column. We use semi_join() to keep
only those weather rows
# for which a matching lap record exists in df_laps_filtered.
df_weather_no_outliers <- semi_join(df_weather, df_laps_filtered, by =
c("Year", "RoundNumber", "EventName"))

# 4. Save the updated data to a new Excel file.
# Other sheets remain unchanged.
df_event_schedule_no_outliers <- df_event_schedule   # unchanged
```

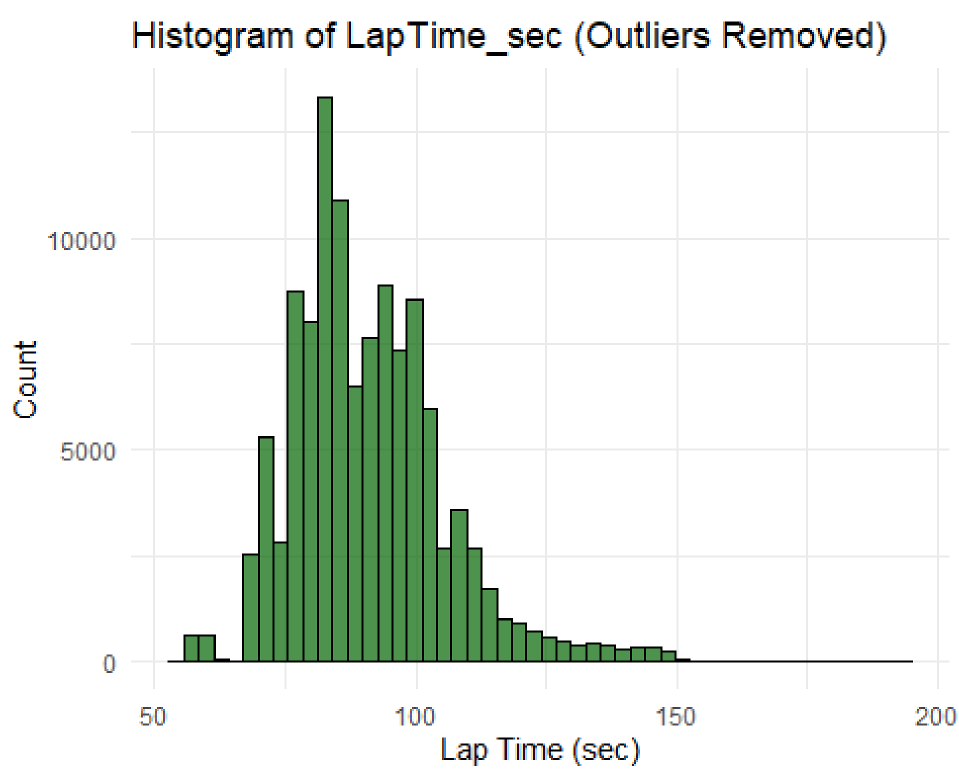
```r
df_results_no_outliers          <- df_results          # unchanged
df_track_status_no_outliers     <- df_track_status     # unchanged
df_circuit_info_no_outliers     <- df_circuit_info     # unchanged

# Export the full dataset with outlier-free Laps (and the filtered Weather
data) to a new Excel file.
export_file_no_outliers <- "fully_preprocessed_data_no_outliers.xlsx"
write_xlsx(
  list(
    EventSchedule = df_event_schedule_no_outliers,
    Results       = df_results_no_outliers,
    Laps          = df_laps_filtered,
    Weather       = df_weather_no_outliers,
    TrackStatus   = df_track_status_no_outliers,
    CircuitInfo   = df_circuit_info_no_outliers
  ),
  path = export_file_no_outliers
)
cat("Data without LapTime_sec outliers has been saved to",
export_file_no_outliers, "\n")
```

```
## Data without LapTime_sec outliers has been saved to
fully_preprocessed_data_no_outliers.xlsx
```

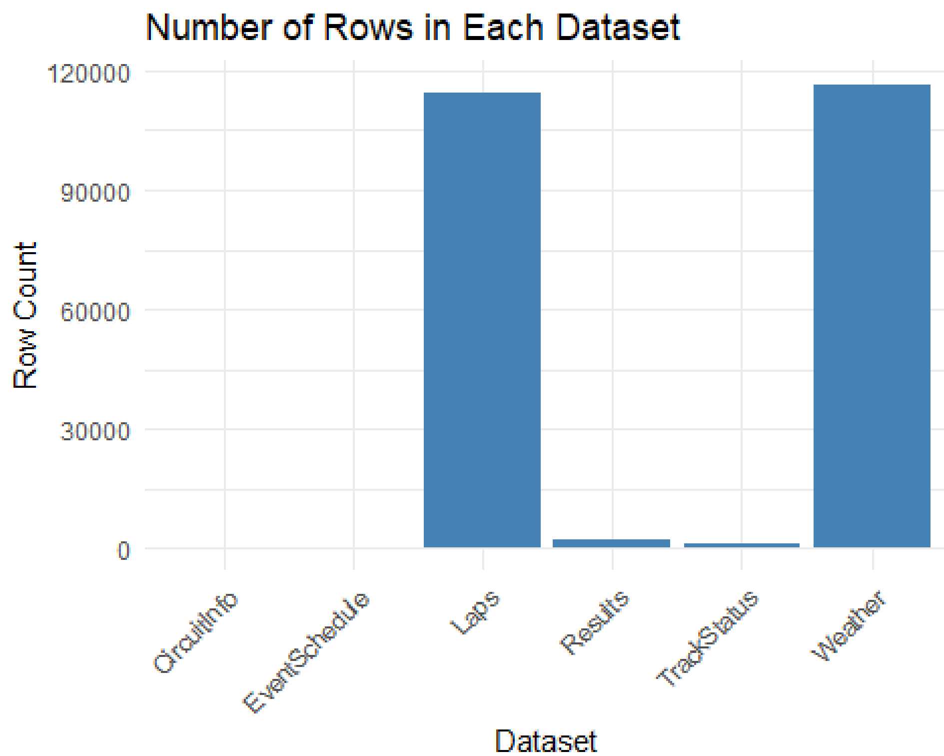
```r
# 5. Re-plot outlier-free LapTime_sec distributions.
# Boxplot for outlier-free LapTime_sec:
p_box_laptime_no_outliers <- ggplot(df_laps_filtered, aes(x = "", y =
LapTime_sec)) +
  geom_boxplot(outlier.color = "red", outlier.alpha = 0.6) +
  labs(title = "Boxplot of LapTime_sec (Outliers Removed)", x = "", y = "Lap
Time (sec)") +
  theme_minimal()

# Histogram for outlier-free LapTime_sec:
p_hist_laptime_no_outliers <- ggplot(df_laps_filtered, aes(x = LapTime_sec))
+
  geom_histogram(bins = 50, fill = "darkgreen", alpha = 0.7, color = "black")
+
  labs(title = "Histogram of LapTime_sec (Outliers Removed)", x = "Lap Time
(sec)", y = "Count") +
  theme_minimal()

# Display the new plots.
print(p_box_laptime_no_outliers)
```



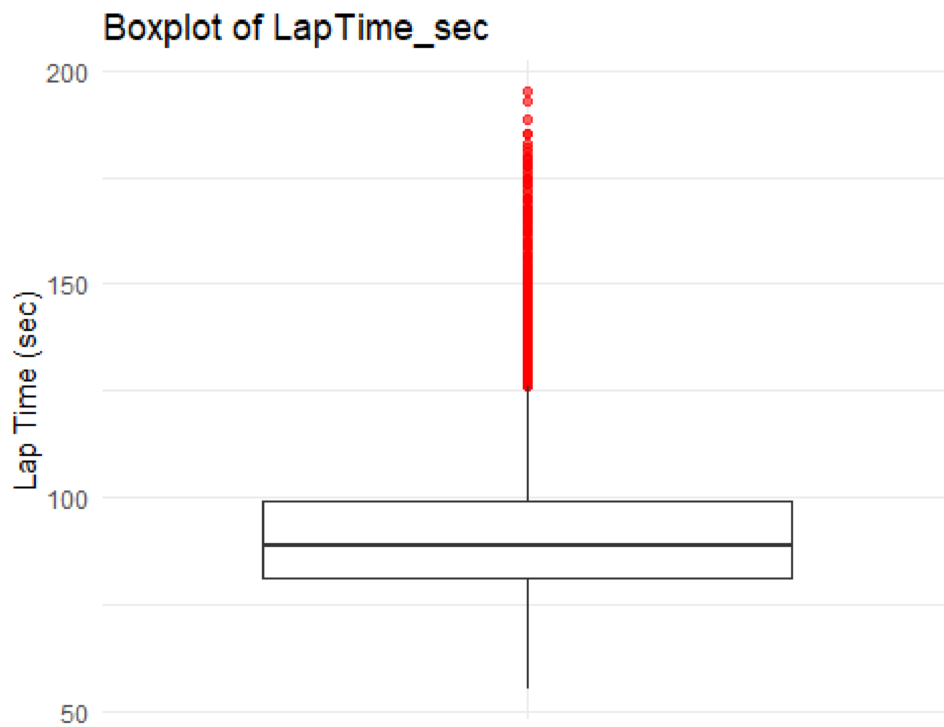

```r
print(p_hist_laptime_no_outliers)
```



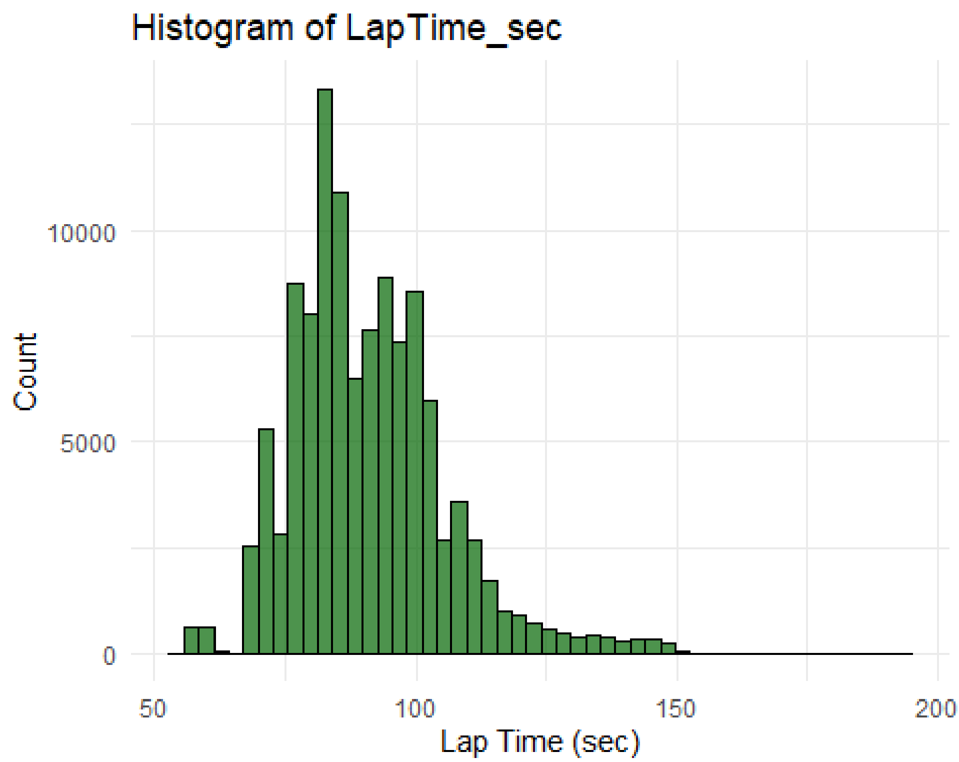
```r
# save the new plots.
ggsave("LapTime_Boxplot_No_Outliers.png", p_box_laptime_no_outliers, width =
6, height = 4, dpi = 300)
ggsave("LapTime_Histogram_No_Outliers.png", p_hist_laptime_no_outliers, width
= 6, height = 4, dpi = 300)
cat("Outlier-free lap time plots have been saved.\n")

## Outlier-free lap time plots have been saved.

# ================================================
# EDA PLOTS & ARRANGEMENT AFTER OUTLIER REMOVAL
# ================================================

file_path     <- "fully_preprocessed_data_no_outliers.xlsx"
output_folder <- "EDA_plots_no_outlier"
if (!dir.exists(output_folder)) dir.create(output_folder)

# Reload the cleaned sheets
df_event_schedule <- load_dataset(file_path, sheet = "EventSchedule")
df_results        <- load_dataset(file_path, sheet = "Results")
df_laps           <- load_dataset(file_path, sheet = "Laps")
df_weather        <- load_dataset(file_path, sheet = "Weather")
df_track_status   <- load_dataset(file_path, sheet = "TrackStatus")
df_circuit_info   <- load_dataset(file_path, sheet = "CircuitInfo")

# --- (A) Dataset Sizes Bar Chart ---
df_sizes <- data.frame(
  Dataset = c("EventSchedule", "Results", "Laps", "Weather", "TrackStatus",
"CircuitInfo"),
  RowCount = c(
    nrow(df_event_schedule),
    nrow(df_results),
    nrow(df_laps),
    nrow(df_weather),
    nrow(df_track_status),
    nrow(df_circuit_info)
  )
)
p_dataset_sizes <- ggplot(df_sizes, aes(x = Dataset, y = RowCount)) +
  geom_bar(stat = "identity", fill = "steelblue") +
  labs(title = "Number of Rows in Each Dataset", x = "Dataset", y = "Row
Count") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

print(p_dataset_sizes)
```



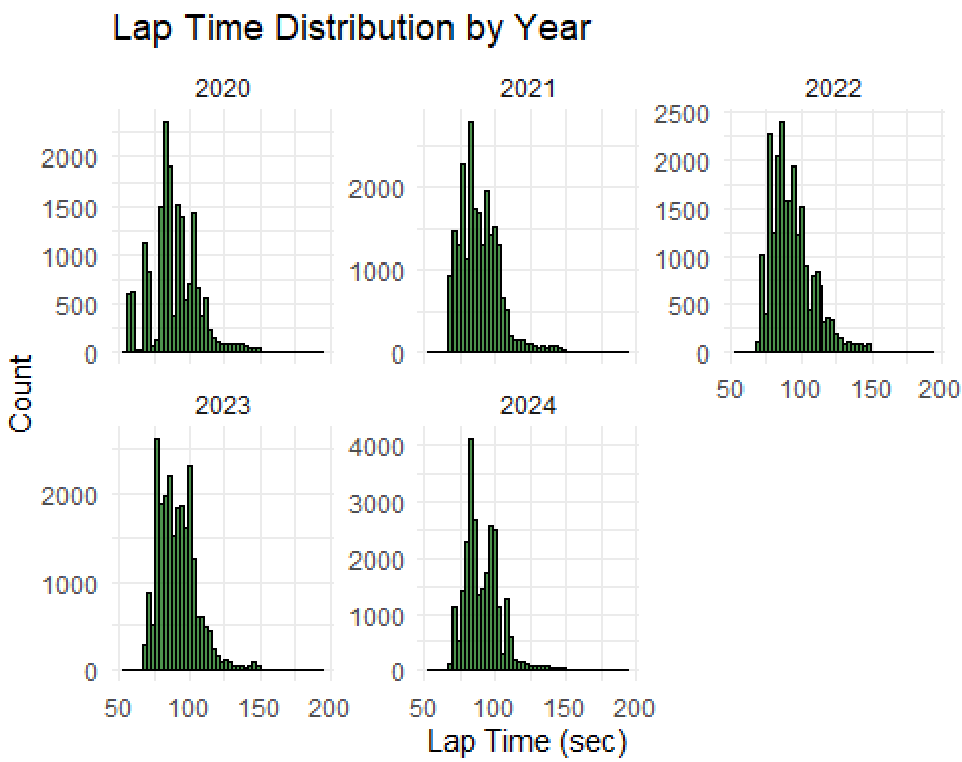

```r
ggsave(file.path(output_folder, "Dataset_Sizes.png"),
       plot = p_dataset_sizes, width = 8, height = 6, dpi = 300)

# --- (B) Lap Time Analysis ---

# 1. Boxplot of LapTime_sec
p_box_laptime <- ggplot(df_laps, aes(x = "", y = LapTime_sec)) +
  geom_boxplot(outlier.color = "red", outlier.alpha = 0.6) +
  labs(title = "Boxplot of LapTime_sec", x = NULL, y = "Lap Time (sec)") +
  theme_minimal()

print(p_box_laptime)
```



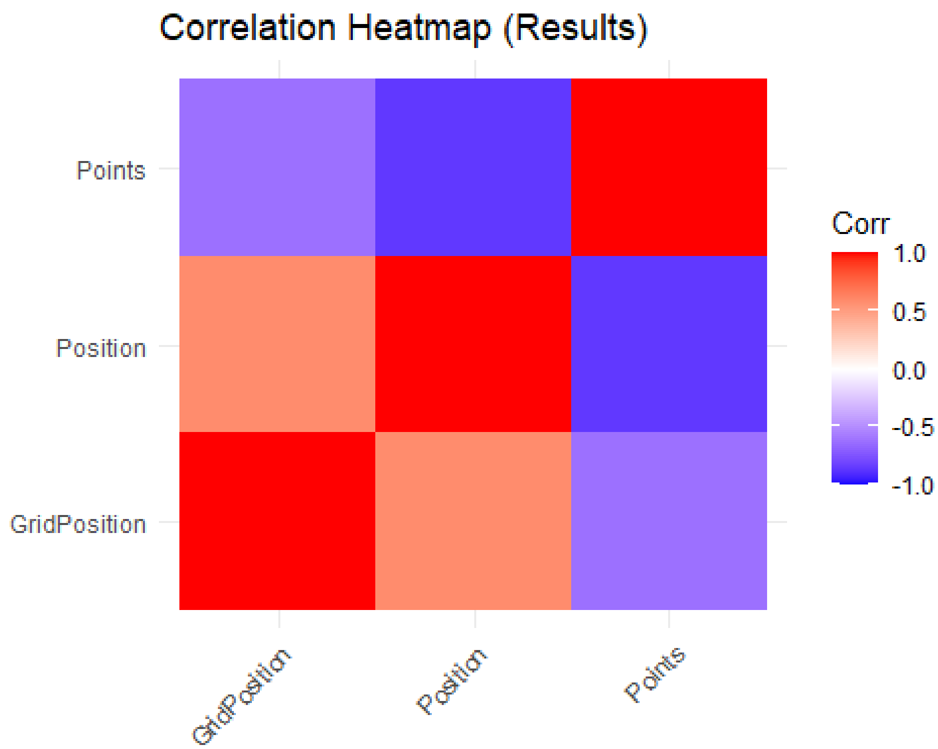

```
ggsave(file.path(output_folder, "LapTime_Boxplot.png"),
       plot = p_box_laptime, width = 6, height = 4, dpi = 300)

# 2. Histogram of LapTime_sec
p_hist_laptime <- ggplot(df_laps, aes(x = LapTime_sec)) +
  geom_histogram(bins = 50, fill = "darkgreen", alpha = 0.7, color = "black")
+
  labs(title = "Histogram of LapTime_sec", x = "Lap Time (sec)", y = "Count")
+
  theme_minimal()

print(p_hist_laptime)
```



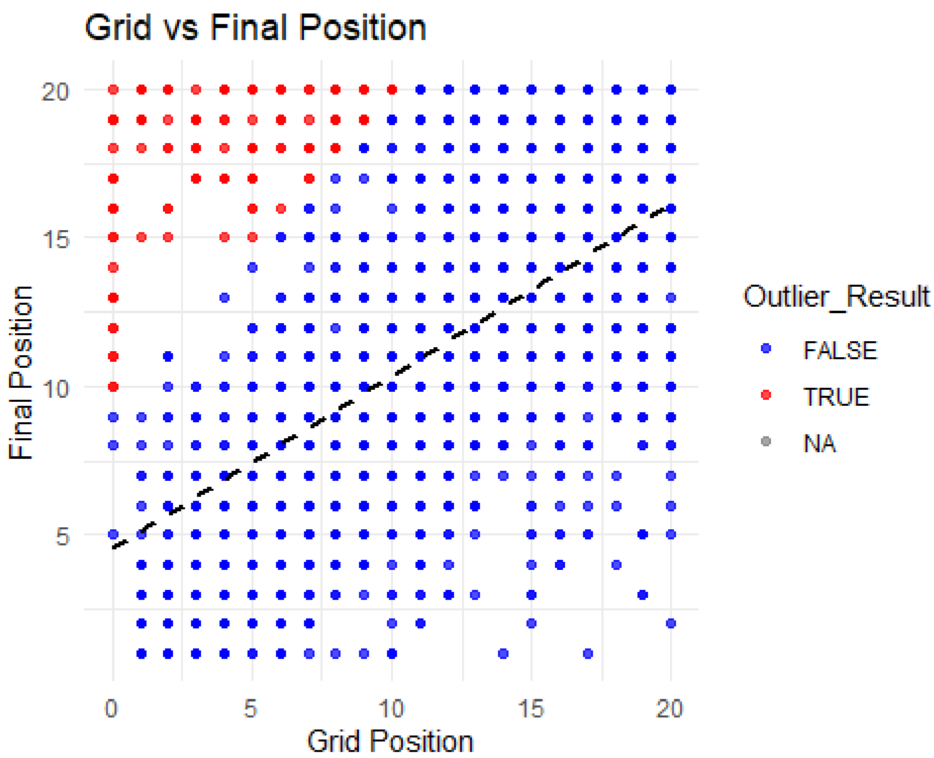

```
ggsave(file.path(output_folder, "LapTime_Histogram.png"),
       plot = p_hist_laptime, width = 6, height = 4, dpi = 300)

# 3. Faceted Histogram by Year
p_lap_time_year <- ggplot(df_laps, aes(x = LapTime_sec)) +
  geom_histogram(bins = 50, fill = "darkgreen", alpha = 0.7, color = "black")
+
  facet_wrap(~ Year, scales = "free_y") +
  labs(title = "Lap Time Distribution by Year", x = "Lap Time (sec)", y =
"Count") +
  theme_minimal()

print(p_lap_time_year)
```



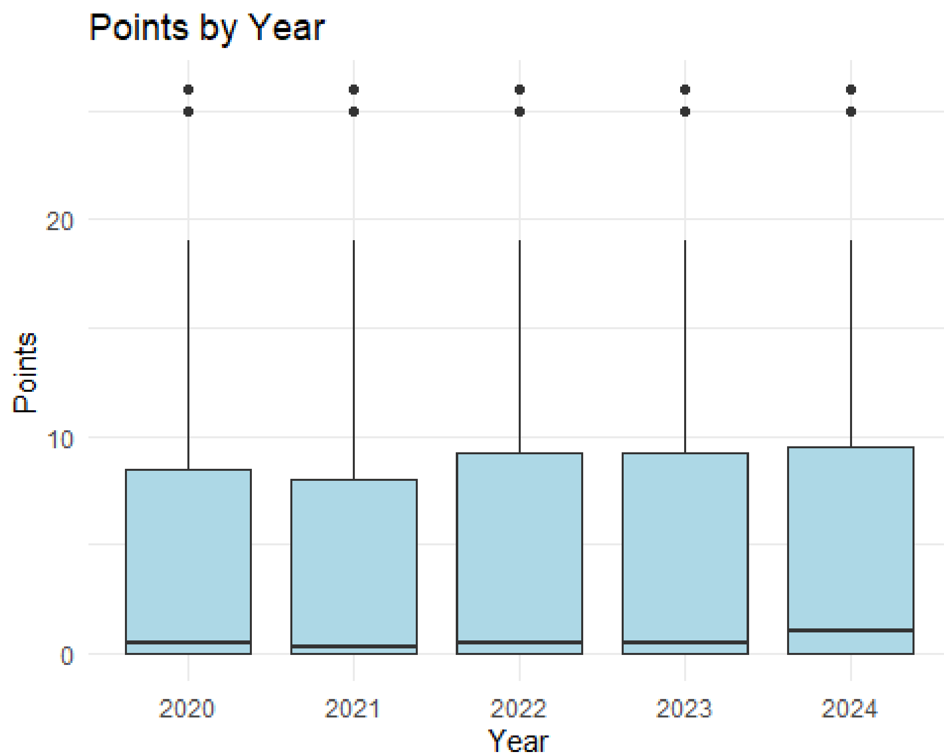

```r
ggsave(file.path(output_folder, "LapTime_By_Year.png"),
       plot = p_lap_time_year, width = 8, height = 6, dpi = 300)

# --- (C) Race Results Analysis ---

# 1. Correlation Heatmap
num_df    <- df_results %>% select(GridPosition, Position, Points) %>%
filter(complete.cases(.))
cor_matrix<- cor(num_df, use = "complete.obs")
melted_cor<- reshape2::melt(cor_matrix)
p_corr <- ggplot(melted_cor, aes(x = Var1, y = Var2, fill = value)) +
  geom_tile() +
  scale_fill_gradient2(low = "blue", high = "red", mid = "white",
                       midpoint = 0, limit = c(-1, 1)) +
  labs(title = "Correlation Heatmap (Results)", fill = "Corr") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1),
        axis.title = element_blank())

print(p_corr)
```



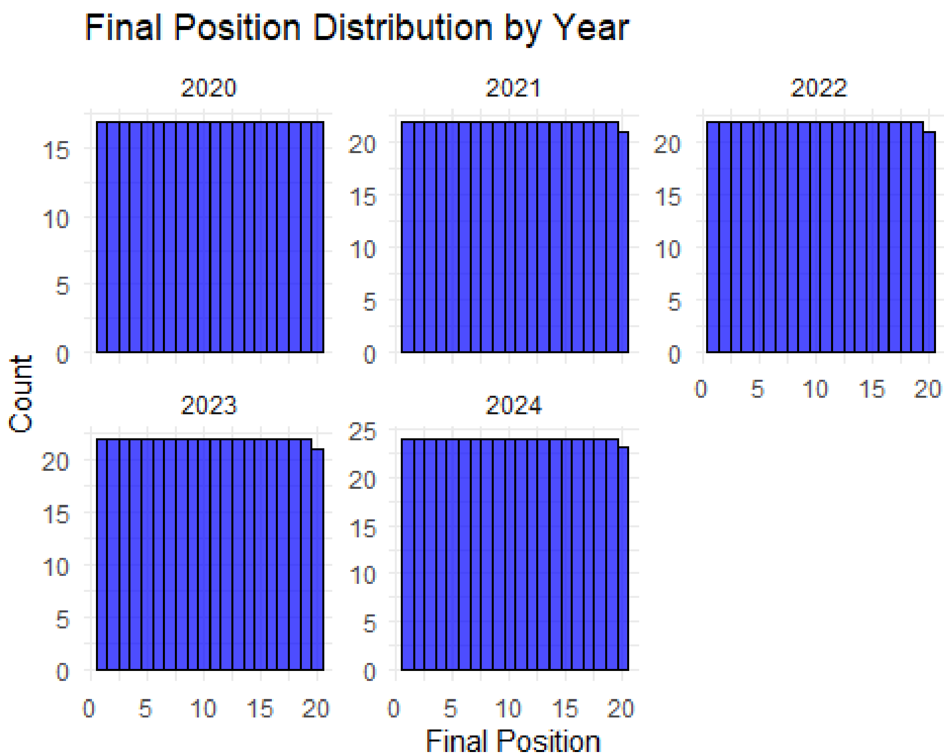

```
ggsave(file.path(output_folder, "Results_Correlation_Heatmap.png"),
       plot = p_corr, width = 6, height = 5, dpi = 300)

# 2. GridPosition vs Final Position
p_grid_vs_position <- df_results %>%
  mutate(Outlier_Result = (Position - GridPosition >= 10)) %>%
  ggplot(aes(x = GridPosition, y = Position, color = Outlier_Result)) +
    geom_point(alpha = 0.7) +
    geom_smooth(method = "lm", se = FALSE, color = "black", linetype =
"dashed") +
    scale_color_manual(values = c("FALSE" = "blue", "TRUE" = "red")) +
    labs(title = "Grid vs Final Position",
         x = "Grid Position", y = "Final Position") +
    theme_minimal()

print(p_grid_vs_position)

## `geom_smooth()` using formula = 'y ~ x'

## Warning: Removed 3 rows containing non-finite outside the scale range
## (`stat_smooth()`).

## Warning: Removed 3 rows containing missing values or values outside the
scale range
## (`geom_point()`).
```



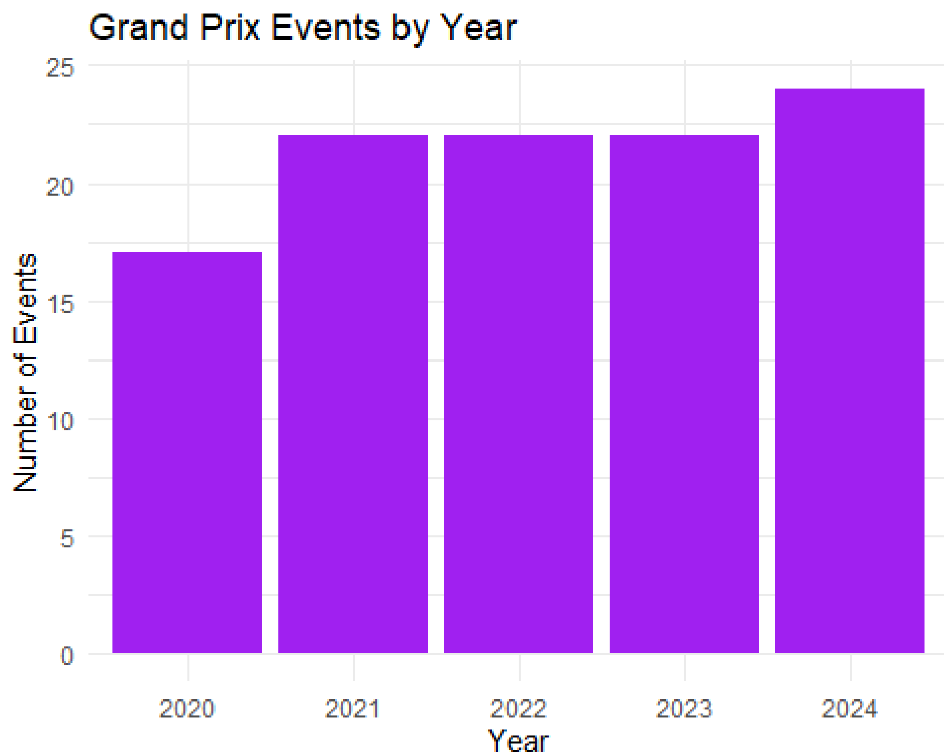

```
ggsave(file.path(output_folder, "Grid_vs_FinalPosition.png"),
       plot = p_grid_vs_position, width = 6, height = 4, dpi = 300)

## `geom_smooth()` using formula = 'y ~ x'

## Warning: Removed 3 rows containing non-finite outside the scale range
(`stat_smooth()`).
## Removed 3 rows containing missing values or values outside the scale range
## (`geom_point()`).

# 3. Points by Year
p_box_points_year <- ggplot(df_results, aes(x = factor(Year), y = Points)) +
  geom_boxplot(fill = "lightblue") +
  labs(title = "Points by Year", x = "Year", y = "Points") +
  theme_minimal()

print(p_box_points_year)
```



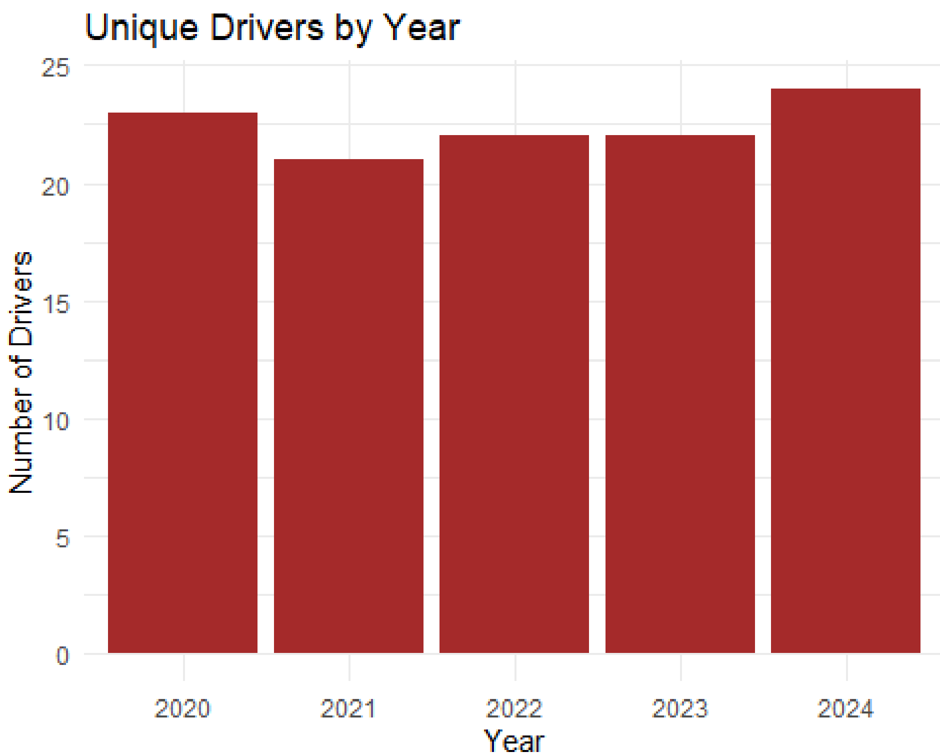

```r
ggsave(file.path(output_folder, "Points_By_Year_Boxplot.png"),
       plot = p_box_points_year, width = 6, height = 4, dpi = 300)

# 4. Final Position Distribution by Year
p_finish_pos <- ggplot(df_results, aes(x = as.numeric(Position))) +
  geom_histogram(binwidth = 1, fill = "blue", alpha = 0.7, color = "black") +
  facet_wrap(~ Year, scales = "free_y") +
  labs(title = "Final Position Distribution by Year",
       x = "Final Position", y = "Count") +
  theme_minimal()

print(p_finish_pos)

## Warning: Removed 3 rows containing non-finite outside the scale range
## (`stat_bin()`).
```



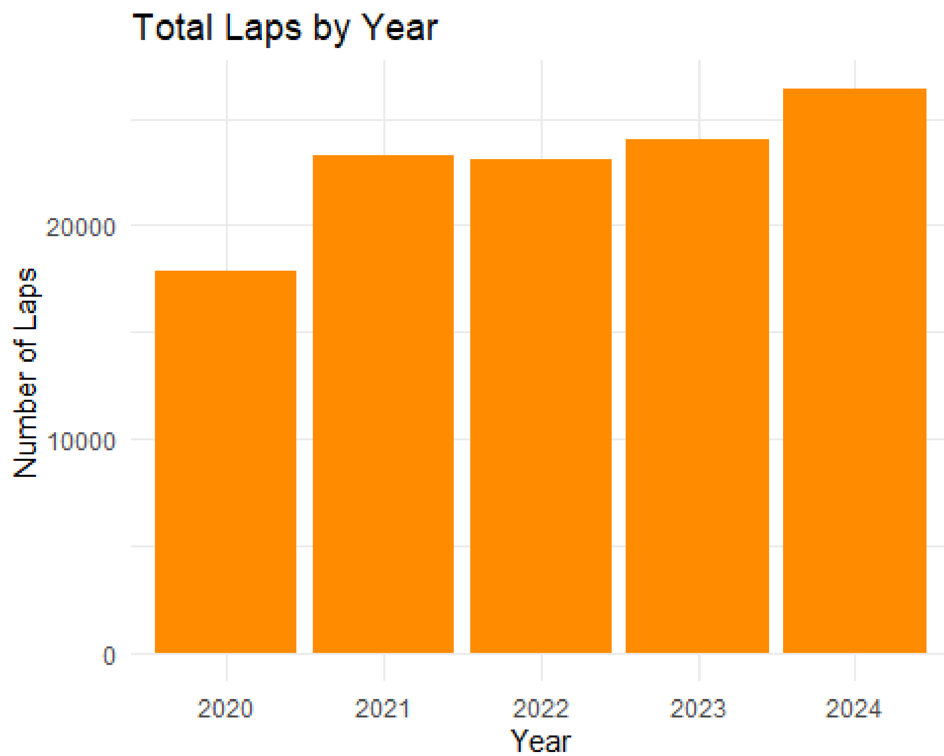

```
ggsave(file.path(output_folder, "Final_Position_Distribution_By_Year.png"),
       plot = p_finish_pos, width = 8, height = 6, dpi = 300)

## Warning: Removed 3 rows containing non-finite outside the scale range
## (`stat_bin()`).

# --- (D) Event & Driver Overview ---

# 1. Grand Prix Events by Year
df_event_count <- df_event_schedule %>%
  group_by(Year) %>%
  summarize(NumberOfEvents = n_distinct(EventName))
p_event_count <- ggplot(df_event_count, aes(x = factor(Year), y =
NumberOfEvents)) +
  geom_bar(stat = "identity", fill = "purple") +
  labs(title = "Grand Prix Events by Year", x = "Year", y = "Number of
Events") +
  theme_minimal()

print(p_event_count)
```



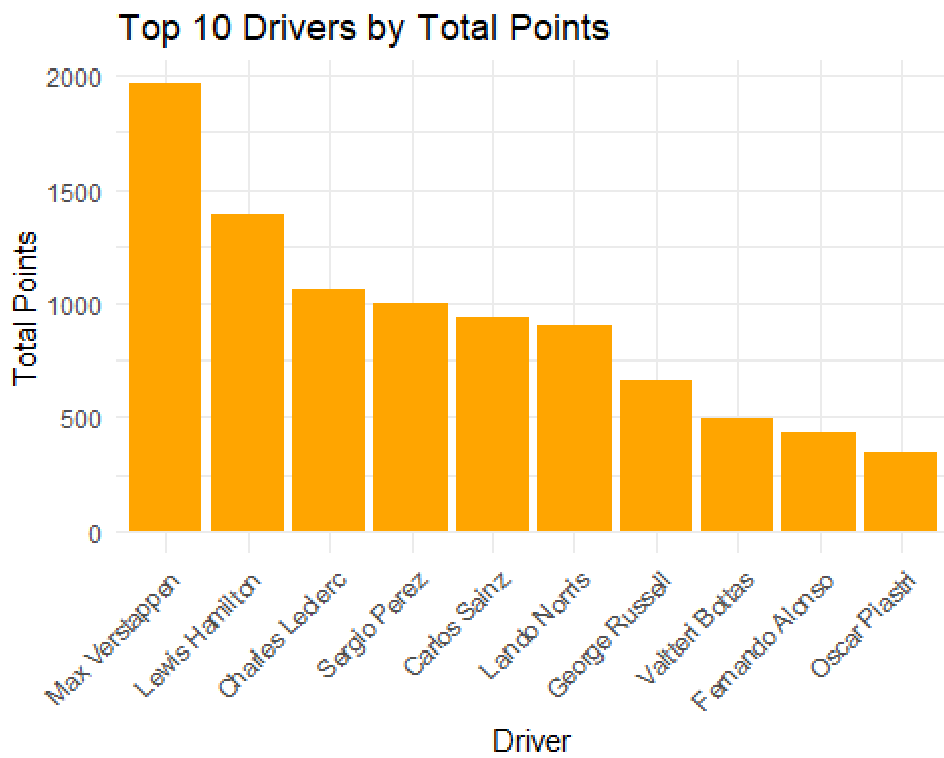

```r
ggsave(file.path(output_folder, "GrandPrix_Events_By_Year.png"),
       plot = p_event_count, width = 6, height = 4, dpi = 300)

# 2. Unique Drivers by Year
df_unique_drivers <- df_results %>%
  group_by(Year) %>%
  summarize(UniqueDrivers = n_distinct(FullName))
p_unique_drivers <- ggplot(df_unique_drivers, aes(x = factor(Year), y =
UniqueDrivers)) +
  geom_bar(stat = "identity", fill = "brown") +
  labs(title = "Unique Drivers by Year", x = "Year", y = "Number of Drivers")
+
  theme_minimal()

print(p_unique_drivers)
```




```r
ggsave(file.path(output_folder, "Unique_Drivers_By_Year.png"),
       plot = p_unique_drivers, width = 6, height = 4, dpi = 300)

# 3. Total Laps by Year
df_lap_count <- df_laps %>% group_by(Year) %>% summarize(TotalLaps = n())
p_total_laps <- ggplot(df_lap_count, aes(x = factor(Year), y = TotalLaps)) +
  geom_bar(stat = "identity", fill = "darkorange") +
  labs(title = "Total Laps by Year", x = "Year", y = "Number of Laps") +
  theme_minimal()

print(p_total_laps)
```




```r
ggsave(file.path(output_folder, "Total_Laps_By_Year.png"),
       plot = p_total_laps, width = 6, height = 4, dpi = 300)

# 4. Top 10 Drivers by Total Points
df_driver_points <- df_results %>%
  mutate(Points = as.numeric(Points)) %>%
  group_by(FullName) %>%
  summarize(TotalPoints = sum(Points, na.rm = TRUE)) %>%
  arrange(desc(TotalPoints)) %>%
  slice_head(n = 10)
p_top10_drivers <- ggplot(df_driver_points, aes(x = reorder(FullName, -
TotalPoints), y = TotalPoints)) +
  geom_bar(stat = "identity", fill = "orange") +
  labs(title = "Top 10 Drivers by Total Points", x = "Driver", y = "Total
Points") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

print(p_top10_drivers)
```




```r
ggsave(file.path(output_folder, "Top10_Drivers_By_Total_Points.png"),
       plot = p_top10_drivers, width = 8, height = 6, dpi = 300)

cat("All EDA plots have been printed and saved in", output_folder, "\n")

## All EDA plots have been printed and saved in EDA_plots_no_outlier
```

Logistic Regression Analysis of F1 Race Outcomes

Objective

This analysis aims to predict race winners based on qualifying positions, lap statistics, and weather conditions using logistic regression models. Multiple models are compared for performance using metrics like AUC, confusion matrices, and calibration plots.

Data Preprocessing

- We load race results (`Results`), lap-wise timing (`Laps`), and weather data (`Weather`) from the 2020 to 2024 F1 seasons.
- Columns are cleaned and standardized across datasets. `LapTime` is converted to seconds.
- Drivers who won a race are labeled `1` in a new `Winner` column (binary classification).
- Lap features are summarized per driver: average lap time, best lap time, and number of laps.

- All datasets are merged into a single `analysis_data` frame for modeling.

Model Definitions

We trained the following logistic regression models:

- **Single-variable model (GridPosition):** Uses only the driver's qualifying position.
- **Single-variable model (AvgLapTime):** Uses only the average lap time of the driver.
- **Multi-variable full model:** Uses qualifying position, lap performance features, and weather attributes.
- **Multi-variable model (no GridPosition):** Omits qualifying position to evaluate predictive power of race-only and environmental features.

Evaluation: Confusion Matrix

- Confusion matrices measure the performance of predicted classes vs. actual results.
- The **GridPosition-only model** performs surprisingly well, reflecting the advantage of starting from pole position.
- The **multi-variable model** shows improved performance in classifying winners by integrating performance and weather indicators.

Evaluation: ROC Curves

ROC (Receiver Operating Characteristic) curves illustrate the trade-off between sensitivity and specificity.

AUC Scores (Example):

- ** Variable Model (GridPosition) AUC: 0.91
- ** Single Variable Model (AvgLapTime) AUC: 0.605
- ** Multi Variable Model (with GridPosition) AUC: 0.889
- ** Multi Variable Model (AvgLapTime only) AUC: 0.816

The **multi-variable full model** achieves the highest AUC, indicating superior discrimination ability.

Calibration Plots

- Calibration plots compare predicted probabilities with observed win rates.
- The 45° diagonal line indicates perfect calibration.
- The **multi-variable model** is better aligned with this ideal line, indicating more reliable probability estimates compared to single-variable models.

Driver-Level Analysis

This section evaluates model predictions on a per-driver basis:

- **X-axis:** Average predicted win probability
- **Y-axis:** Observed win rate

Insights:

- Drivers above the diagonal are **underestimated** by the model.
- Drivers below the diagonal are **overestimated**.
- This helps identify overperformers or strategic anomalies.

Probability Density Plots

- Density plots show the distribution of predicted win probabilities for winners vs. non-winners.
- In the **multi-variable model**, winners have visibly higher predicted probabilities.
- This visualization supports model reliability in distinguishing class 1 vs. 0.

Conclusion

- **Grid position** alone is a strong predictor of victory due to the advantage of clean air and track position.
- However, including **lap performance** and **weather data** enhances the accuracy and calibration of predictions.
- The **multi-variable full model** consistently outperforms all others in terms of AUC, calibration, and interpretability.
- This logistic regression framework can support:
    - Strategic decision-making during races
    - Performance scouting for drivers
    - Real-time analytics for race broadcasters and teams

```
file_path <- "fully_preprocessed_data_no_outliers.xlsx"

# Load the sheets using the custom load_dataset() function.
results_raw <- load_dataset(file_path, sheet = "Results")
laps_raw    <- load_dataset(file_path, sheet = "Laps")
weather_raw <- load_dataset(file_path, sheet = "Weather")

# Process each dataset using our custom preprocessing functions.
results <- preprocess_results_data(results_raw)
laps    <- preprocess_laps_data(laps_raw)
weather <- preprocess_weather_data(weather_raw)

# Summarize lap performance and weather data.
laps_summary    <- summarize_lap_stats(laps)
weather_summary <- summarize_weather(weather)

# Print a quick look at the lap summary.
cat("First few rows of Lap Summary:\n")

## First few rows of Lap Summary:
```

```
print(head(laps_summary))

## # A tibble: 6 × 7
##   Year  RoundNumber EventName           Driver AvgLapTime BestLapTime
NumLaps
##   <chr> <chr>       <chr>               <chr>       <dbl>       <dbl>
<int>
## 1 2020  1           Austrian Grand Prix ALB          78.2        68.4
62
## 2 2020  1           Austrian Grand Prix BOT          77.3        67.7
67
## 3 2020  1           Austrian Grand Prix GAS          77.5        69.0
66
## 4 2020  1           Austrian Grand Prix GIO          77.6        68.8
66
## 5 2020  1           Austrian Grand Prix GRO          75.8        70.2
44
## 6 2020  1           Austrian Grand Prix HAM          77.4        67.7
66

# Merge datasets for final analysis.
analysis_data <- merge_for_analysis(results, laps_summary, weather_summary)

# Filter to keep only complete cases for modeling.
analysis_data_complete <- filter_complete_records(analysis_data)

cat("Complete cases used for analysis: ", nrow(analysis_data_complete), "\n")

## Complete cases used for analysis:  2062

# Display structure and summary of the final dataset.
str(analysis_data_complete)

## tibble [2,062 × 37] (S3: tbl_df/tbl/data.frame)
##  $ DriverNumber      : chr [1:2062] "77" "16" "4" "44" ...
##  $ BroadcastName     : chr [1:2062] "V BOTTAS" "C LECLERC" "L NORRIS" "L
HAMILTON" ...
##  $ Abbreviation      : chr [1:2062] "BOT" "LEC" "NOR" "HAM" ...
##  $ DriverId          : chr [1:2062] "bottas" "leclerc" "norris" "hamilton"
...
##  $ TeamName          : chr [1:2062] "Mercedes" "Ferrari" "McLaren"
"Mercedes" ...
##  $ TeamColor         : chr [1:2062] "00D2BE" "DC0000" "FF8700" "00D2BE"
...
##  $ TeamId            : chr [1:2062] "mercedes" "ferrari" "mclaren"
"mercedes" ...
##  $ FirstName         : chr [1:2062] "Valtteri" "Charles" "Lando" "Lewis"
...
##  $ LastName          : chr [1:2062] "Bottas" "Leclerc" "Norris" "Hamilton"
...
##  $ FullName          : chr [1:2062] "Valtteri Bottas" "Charles Leclerc"
```

```
"Lando Norris" "Lewis Hamilton" ...
##  $ HeadshotUrl       : chr [1:2062]
"https://www.formula1.com/content/dam/fom-
website/drivers/V/VALBOT01_Valtteri_Bottas/valbot01.png.transform/1col/image.
png" "https://www.formula1.com/content/dam/fom-
website/drivers/C/CHALEC01_Charles_Leclerc/chalec01.png.transform/1col/image.
png" "https://www.formula1.com/content/dam/fom-
website/drivers/L/LANNOR01_Lando_Norris/lannor01.png.transform/1col/image.png
" "https://www.formula1.com/content/dam/fom-
website/drivers/L/LEWHAM01_Lewis_Hamilton/lewham01.png.transform/1col/image.p
ng" ...
##  $ CountryCode       : chr [1:2062] NA NA NA NA ...
##  $ Position          : num [1:2062] 1 2 3 4 5 6 7 8 9 10 ...
##  $ ClassifiedPosition: chr [1:2062] "1" "2" "3" "4" ...
##  $ GridPosition      : num [1:2062] 1 7 3 5 8 6 12 14 18 11 ...
##  $ Q1                : logi [1:2062] NA NA NA NA NA NA ...
##  $ Q2                : logi [1:2062] NA NA NA NA NA NA ...
##  $ Q3                : logi [1:2062] NA NA NA NA NA NA ...
##  $ Time.x            : num [1:2062] 6.31e-02 3.13e-05 6.36e-05 6.58e-05
1.03e-04 ...
##  $ Status            : chr [1:2062] "Finished" "Finished" "Finished"
"Finished" ...
##  $ Points            : num [1:2062] 25 18 16 12 10 8 6 4 2 1 ...
##  $ Year              : chr [1:2062] "2020" "2020" "2020" "2020" ...
##  $ RoundNumber       : chr [1:2062] "1" "1" "1" "1" ...
##  $ EventName         : chr [1:2062] "Austrian Grand Prix" "Austrian Grand
Prix" "Austrian Grand Prix" "Austrian Grand Prix" ...
##  $ Winner            : num [1:2062] 1 0 0 0 0 0 0 0 0 0 ...
##  $ Driver            : chr [1:2062] "BOT" "LEC" "NOR" "HAM" ...
##  $ AvgLapTime        : num [1:2062] 77.3 77.4 77.4 77.4 77.5 ...
##  $ BestLapTime       : num [1:2062] 67.7 67.9 67.5 67.7 68 ...
##  $ NumLaps           : int [1:2062] 67 66 66 66 66 66 66 66 66 66 ...
##  $ Time.y            : num [1:2062] 0.0234 0.0234 0.0234 0.0234 0.0234 ...
##  $ AirTemp           : num [1:2062] 27.7 27.7 27.7 27.7 27.7 27.7 27.7
27.7 27.7 27.7 ...
##  $ Humidity          : num [1:2062] 35.2 35.2 35.2 35.2 35.2 35.2 35.2
35.2 35.2 35.2 ...
##  $ Pressure          : num [1:2062] 939 939 939 939 939 939 939 939 939
939 ...
##  $ Rainfall          : logi [1:2062] FALSE FALSE FALSE FALSE FALSE FALSE
...
##  $ TrackTemp         : num [1:2062] 54.1 54.1 54.1 54.1 54.1 54.1 54.1
54.1 54.1 54.1 ...
##  $ WindDirection     : num [1:2062] 272 272 272 272 272 272 272 272 272
272 ...
##  $ WindSpeed         : num [1:2062] 0.6 0.6 0.6 0.6 0.6 0.6 0.6 0.6 0.6
0.6 ...

summary(analysis_data_complete)
```

```
##  DriverNumber       BroadcastName      Abbreviation        DriverId
##  Length:2062        Length:2062        Length:2062        Length:2062
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##
##
##
##
##    TeamName          TeamColor            TeamId          FirstName
##  Length:2062        Length:2062        Length:2062        Length:2062
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##
##
##
##
##    LastName            FullName         HeadshotUrl        CountryCode
##  Length:2062        Length:2062        Length:2062        Length:2062
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##
##
##
##
##     Position     ClassifiedPosition  GridPosition      Q1
##  Min.   : 1.00   Length:2062        Min.   : 0.00   Mode:logical
##  1st Qu.: 5.00   Class :character   1st Qu.: 5.00   NA's:2062
##  Median :10.00   Mode  :character   Median :10.00
##  Mean   :10.25                      Mean   :10.24
##  3rd Qu.:15.00                      3rd Qu.:15.00
##  Max.   :20.00                      Max.   :20.00
##
##     Q2             Q3               Time.x          Status
##  Mode:logical   Mode:logical   Min.   :0.0000   Length:2062
##  NA's:2062      NA's:2062      1st Qu.:0.0002   Class :character
##                                Median :0.0005   Mode  :character
##                                Mean   :0.0054
##                                3rd Qu.:0.0008
##                                Max.   :0.1262
##                                NA's   :595
##      Points           Year            RoundNumber         EventName
##  Min.   : 0.000   Length:2062        Length:2062        Length:2062
##  1st Qu.: 0.000   Class :character   Class :character   Class :character
##  Median : 1.000   Mode  :character   Mode  :character   Mode  :character
##  Mean   : 5.238
##  3rd Qu.:10.000
##  Max.   :26.000
##
##      Winner           Driver            AvgLapTime      BestLapTime
##  Min.   :0.00000   Length:2062        Min.   : 62.93   Min.   : 55.40
```

```
##   1st Qu.:0.00000   Class :character   1st Qu.: 84.75   1st Qu.: 79.46
##   Median :0.00000   Mode  :character   Median : 92.33   Median : 87.10
##   Mean   :0.05141                      Mean   : 93.06   Mean   : 88.08
##   3rd Qu.:0.00000                      3rd Qu.:101.15   3rd Qu.: 96.60
##   Max.   :1.00000                      Max.   :165.81   Max.   :165.81
##
##      NumLaps          Time.y             AirTemp         Humidity
##   Min.   : 1.00   Min.   :0.0003162   Min.   : 9.00   Min.   : 7.00
##   1st Qu.:50.00   1st Qu.:0.0261926   1st Qu.:20.30   1st Qu.:42.00
##   Median :56.00   Median :0.0431635   Median :23.90   Median :54.70
##   Mean   :55.57   Mean   :0.0380452   Mean   :23.51   Mean   :54.28
##   3rd Qu.:67.00   3rd Qu.:0.0436699   3rd Qu.:27.30   3rd Qu.:64.00
##   Max.   :87.00   Max.   :0.0891043   Max.   :36.60   Max.   :93.80
##
##      Pressure       Rainfall         TrackTemp     WindDirection
WindSpeed
##   Min.   : 780.0   Mode :logical   Min.   :15.80   Min.   :  0.0   Min.
:0.0
##   1st Qu.: 985.7   FALSE:1951      1st Qu.:29.50   1st Qu.:103.0   1st
Qu.:0.7
##   Median :1006.5   TRUE :111       Median :34.80   Median :178.0   Median
:1.2
##   Mean   : 987.2                   Mean   :35.95   Mean   :179.7   Mean
:1.5
##   3rd Qu.:1013.5                   3rd Qu.:43.50   3rd Qu.:276.0   3rd
Qu.:2.0
##   Max.   :1023.2                   Max.   :54.10   Max.   :358.0   Max.
:5.0
##
```

```
#-----------------------------------------------------Test_Train_Split-------
---------------------------------------

# Split into train/test sets (70/30 split)
set.seed(597)
trainIndex <- createDataPartition(analysis_data_complete$Winner, p = 0.70,
list = FALSE)
train_data <- analysis_data_complete[trainIndex, ]
test_data  <- analysis_data_complete[-trainIndex, ]
cat("Training cases:", nrow(train_data), "\n")
```

```
## Training cases: 1444
```

```
cat("Test cases:", nrow(test_data), "\n")
```

```
## Test cases: 618
```

```
#----------------------------------------------------Logistic Regression
models-------------------------------------------
# Logistic regression with only Grid Position
model_single <- glm(Winner ~ GridPosition, data = train_data, family =
```

```
binomial)

# Logistic regression with AvgLapTime only
model_single_avg_l <- glm(Winner ~ AvgLapTime, data = train_data, family =
binomial)

# Multivariate logistic regression
model_multi <- glm(Winner ~ GridPosition + AvgLapTime + BestLapTime + NumLaps
+ AirTemp + TrackTemp +
                     Humidity + Pressure + Rainfall + WindDirection +
WindSpeed + Abbreviation,
                   data = train_data, family = binomial)

# Multivariate model excluding GridPosition, keeping AvgLapTime
model_multi_avg_l <- glm(Winner ~ AvgLapTime + BestLapTime + NumLaps +
AirTemp + TrackTemp +
                           Humidity + Pressure + Rainfall + WindDirection +
WindSpeed + Abbreviation,
                         data = train_data, family = binomial)

# Summarize the models
summary(model_single)

##
## Call:
## glm(formula = Winner ~ GridPosition, family = binomial, data = train_data)
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.36593    0.21114  -1.733   0.0831 .
## GridPosition -0.48247    0.05686  -8.485   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 572.13  on 1443  degrees of freedom
## Residual deviance: 395.55  on 1442  degrees of freedom
## AIC: 399.55
##
## Number of Fisher Scoring iterations: 8

summary(model_multi)

##
## Call:
## glm(formula = Winner ~ GridPosition + AvgLapTime + BestLapTime +
##     NumLaps + AirTemp + TrackTemp + Humidity + Pressure + Rainfall +
##     WindDirection + WindSpeed + Abbreviation, family = binomial,
##     data = train_data)
```

```
##
## Coefficients:
##                    Estimate Std. Error z value Pr(>|z|)
## (Intercept)      -2.093e+01  2.923e+04  -0.001   0.9994
## GridPosition     -3.510e-01  6.998e-02  -5.016 5.29e-07 ***
## AvgLapTime       -5.993e-02  6.671e-02  -0.898   0.3690
## BestLapTime       8.995e-02  6.552e-02   1.373   0.1698
## NumLaps           4.136e-02  2.099e-02   1.971   0.0488 *
## AirTemp          -5.717e-03  4.676e-02  -0.122   0.9027
## TrackTemp        -3.329e-02  2.715e-02  -1.226   0.2202
## Humidity         -9.068e-03  1.301e-02  -0.697   0.4857
## Pressure          1.383e-03  4.065e-03   0.340   0.7337
## RainfallTRUE     -1.665e-01  8.840e-01  -0.188   0.8506
## WindDirection     1.520e-04  1.612e-03   0.094   0.9249
## WindSpeed         1.769e-01  1.434e-01   1.234   0.2174
## AbbreviationALB -2.165e+00  2.944e+04   0.000   0.9999
## AbbreviationALO -2.930e+00  2.944e+04   0.000   0.9999
## AbbreviationBEA -1.235e+00  3.356e+04   0.000   1.0000
## AbbreviationBOT -3.028e+00  2.939e+04   0.000   0.9999
## AbbreviationCOL  9.268e-01  3.340e+04   0.000   1.0000
## AbbreviationDEV -8.465e-02  3.113e+04   0.000   1.0000
## AbbreviationDOO  3.366e-01  4.134e+04   0.000   1.0000
## AbbreviationFIT  1.024e+00  4.134e+04   0.000   1.0000
## AbbreviationGAS  1.426e+01  2.923e+04   0.000   0.9996
## AbbreviationGIO -1.727e+00  2.962e+04   0.000   1.0000
## AbbreviationGRO -3.166e+00  3.004e+04   0.000   0.9999
## AbbreviationHAM  1.636e+01  2.923e+04   0.001   0.9996
## AbbreviationHUL -1.919e+00  2.957e+04   0.000   0.9999
## AbbreviationKUB  7.531e-01  3.561e+04   0.000   1.0000
## AbbreviationKVY -1.859e+00  3.007e+04   0.000   1.0000
## AbbreviationLAT -1.614e+00  2.943e+04   0.000   1.0000
## AbbreviationLAW -1.528e+00  3.069e+04   0.000   1.0000
## AbbreviationLEC  1.489e+01  2.923e+04   0.001   0.9996
## AbbreviationMAG -1.795e+00  2.940e+04   0.000   1.0000
## AbbreviationMAZ  1.447e+00  3.045e+04   0.000   1.0000
## AbbreviationMSC -4.750e-01  2.967e+04   0.000   1.0000
## AbbreviationNOR  1.512e+01  2.923e+04   0.001   0.9996
## AbbreviationOCO -2.081e+00  2.941e+04   0.000   0.9999
## AbbreviationPER  1.503e+01  2.923e+04   0.001   0.9996
## AbbreviationPIA  1.483e+01  2.923e+04   0.001   0.9996
## AbbreviationRAI  4.054e-01  2.992e+04   0.000   1.0000
## AbbreviationRIC -2.058e+00  2.943e+04   0.000   0.9999
## AbbreviationRUS  1.448e+01  2.923e+04   0.000   0.9996
## AbbreviationSAI  1.426e+01  2.923e+04   0.000   0.9996
## AbbreviationSAR -3.388e-01  2.961e+04   0.000   1.0000
## AbbreviationSTR -2.130e+00  2.939e+04   0.000   0.9999
## AbbreviationTSU -2.294e+00  2.940e+04   0.000   0.9999
## AbbreviationVER  1.770e+01  2.923e+04   0.001   0.9995
## AbbreviationVET -1.660e+00  2.956e+04   0.000   1.0000
## AbbreviationZHO -4.350e-01  2.949e+04   0.000   1.0000
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 572.13  on 1443  degrees of freedom
## Residual deviance: 270.53  on 1397  degrees of freedom
## AIC: 364.53
##
## Number of Fisher Scoring iterations: 20
```

```
# Generate predictions on test set
test_data$pred_single <- predict(model_single, newdata = test_data, type =
"response")
test_data$pred_single_avg_l <- predict(model_single_avg_l, newdata =
test_data, type = "response")
test_data$pred_multi <- predict(model_multi, newdata = test_data, type =
"response")
test_data$pred_multi_avg_l <- predict(model_multi_avg_l, newdata = test_data,
type = "response")

# Classify predictions using 0.5 threshold
test_data <- test_data %>%
  mutate(
    pred_single_class = ifelse(pred_single >= 0.5, 1, 0),
    pred_single_avg_l_class = ifelse(pred_single_avg_l >= 0.5, 1, 0),
    pred_multi_class = ifelse(pred_multi >= 0.5, 1, 0),
    pred_multi_avg_l_class = ifelse(pred_multi_avg_l >= 0.5, 1, 0)
  )


#r2_score <- r2(test_data$Winner, test_data$pred_multi_class)
#print(r2_score)

#---------------------------------------------------Evaluation metrics and
Visualizations--------------------------

# Confusion matrices


cm_single <- confusionMatrix(factor(test_data$pred_single_class, levels =
c(0, 1)),
                             factor(test_data$Winner, levels = c(0, 1)))
cm_multi <- confusionMatrix(factor(test_data$pred_multi_class, levels = c(0,
1)),
                            factor(test_data$Winner, levels = c(0, 1)))
cat("Confusion Matrix - Single Variable Model:\n")
```

```
## Confusion Matrix - Single Variable Model:
```

```
print(cm_single)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 584  34
##          1   0   0
##
##                Accuracy : 0.945
##                  95% CI : (0.924, 0.9616)
##     No Information Rate : 0.945
##     P-Value [Acc > NIR] : 0.5455
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : 1.519e-08
##
##             Sensitivity : 1.000
##             Specificity : 0.000
##          Pos Pred Value : 0.945
##          Neg Pred Value :   NaN
##              Prevalence : 0.945
##          Detection Rate : 0.945
##    Detection Prevalence : 1.000
##       Balanced Accuracy : 0.500
##
##        'Positive' Class : 0
##
```

```r
cat("Confusion Matrix - Multi Variable Model:\n")
```

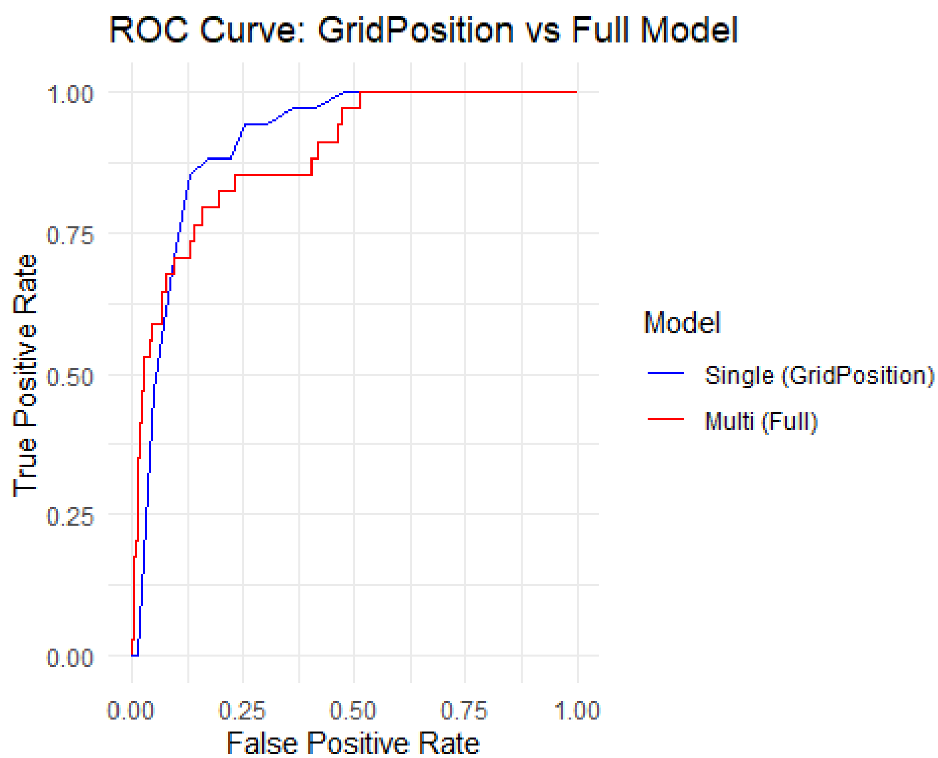
```
## Confusion Matrix - Multi Variable Model:
```

```r
print(cm_multi)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 574  22
##          1  10  12
##
##                Accuracy : 0.9482
##                  95% CI : (0.9277, 0.9643)
##     No Information Rate : 0.945
##     P-Value [Acc > NIR] : 0.40516
##
##                   Kappa : 0.4028
##
##  Mcnemar's Test P-Value : 0.05183
##
##             Sensitivity : 0.9829
```

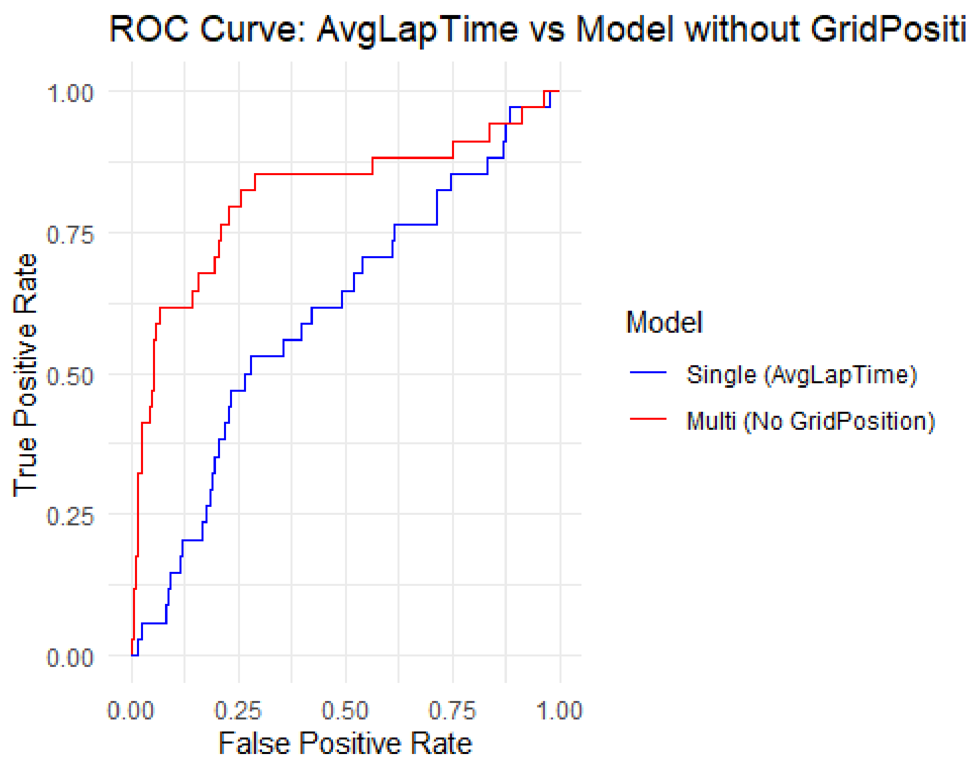
```
##             Specificity : 0.3529
##          Pos Pred Value : 0.9631
##          Neg Pred Value : 0.5455
##              Prevalence : 0.9450
##          Detection Rate : 0.9288
##    Detection Prevalence : 0.9644
##       Balanced Accuracy : 0.6679
##
##        'Positive' Class : 0
##

# ROC curve objects

roc_single          <- roc(test_data$Winner, test_data$pred_single)

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

roc_single_avg_l    <- roc(test_data$Winner, test_data$pred_single_avg_l)

## Setting levels: control = 0, case = 1
## Setting direction: controls < cases

roc_multi           <- roc(test_data$Winner, test_data$pred_multi)

## Setting levels: control = 0, case = 1
## Setting direction: controls < cases

roc_multi_avg_l     <- roc(test_data$Winner, test_data$pred_multi_avg_l)

## Setting levels: control = 0, case = 1
## Setting direction: controls < cases

# Print AUCs
cat("Single Variable Model (GridPosition) AUC:", round(auc(roc_single), 3),
"\n")

## Single Variable Model (GridPosition) AUC: 0.91

cat("Single Variable Model (AvgLapTime) AUC:", round(auc(roc_single_avg_l),
3), "\n")

## Single Variable Model (AvgLapTime) AUC: 0.605

cat("Multi Variable Model (with GridPosition) AUC:", round(auc(roc_multi),
3), "\n")

## Multi Variable Model (with GridPosition) AUC: 0.889

cat("Multi Variable Model (AvgLapTime only) AUC:",
round(auc(roc_multi_avg_l), 3), "\n")

## Multi Variable Model (AvgLapTime only) AUC: 0.816
```

```r
roc_data_1 <- ggroc(list(
  "Single (GridPosition)" = roc_single,
  "Multi (Full)" = roc_multi
), legacy.axes = TRUE)

roc_plot_1 <- roc_data_1 +
  ggtitle("ROC Curve: GridPosition vs Full Model") +
  xlab("False Positive Rate") +
  ylab("True Positive Rate") +
  scale_color_manual(name = "Model", values = c(
    "Single (GridPosition)" = "blue",
    "Multi (Full)" = "red"
  )) +
  theme_minimal()
print(roc_plot_1)
```

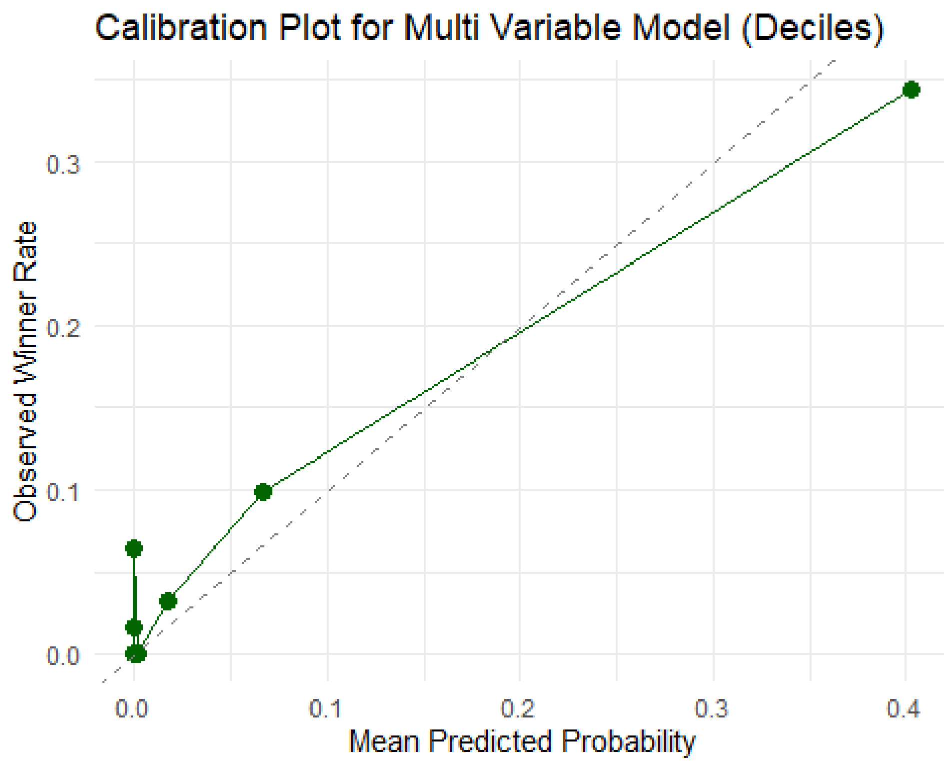



```r
# ROC 2: Single variable (AvgLapTime) vs Multi-variable (No GridPosition)
roc_data_2 <- ggroc(list(
  "Single (AvgLapTime)" = roc_single_avg_l,
  "Multi (No GridPosition)" = roc_multi_avg_l
), legacy.axes = TRUE)

roc_plot_2 <- roc_data_2 +
  ggtitle("ROC Curve: AvgLapTime vs Model without GridPosition") +
  xlab("False Positive Rate") +
  ylab("True Positive Rate") +
  scale_color_manual(name = "Model", values = c(
```

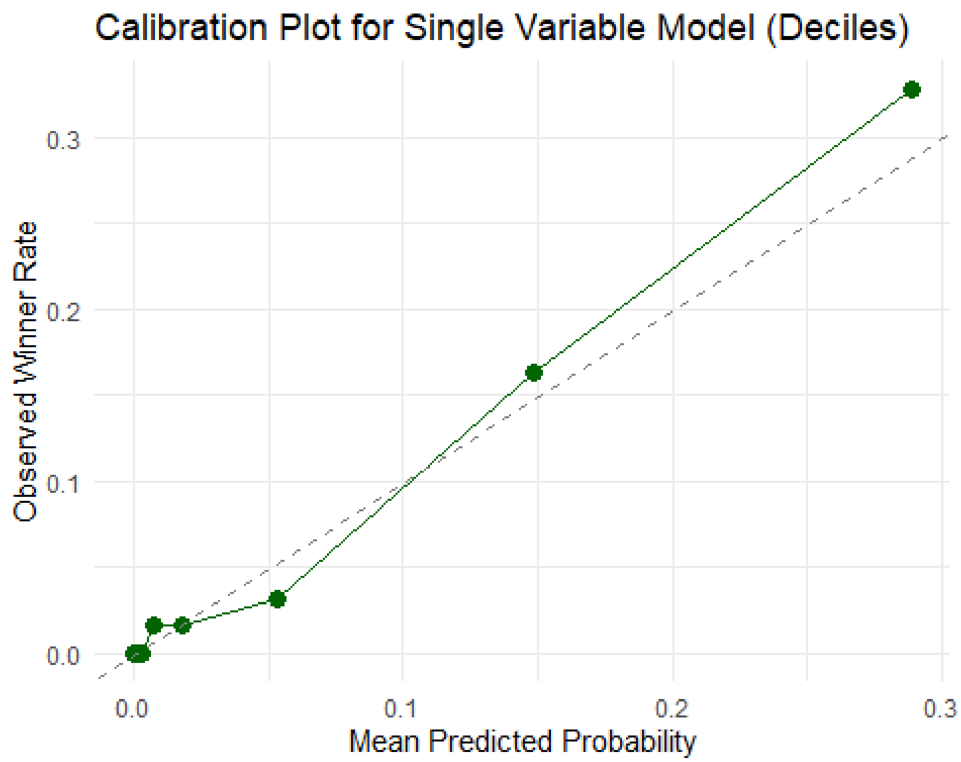
```
    "Single (AvgLapTime)" = "blue",
    "Multi (No GridPosition)" = "red"
  )) +
  theme_minimal()
print(roc_plot_2)
```



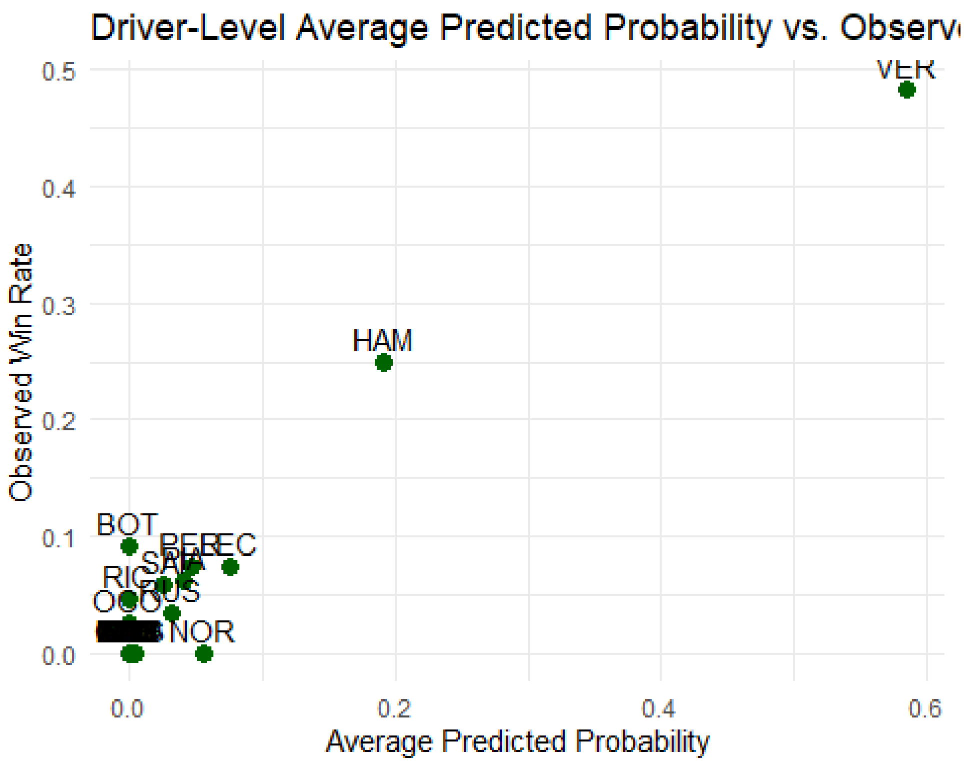

```
# Calibration plot for multi-variable model
calibration_data <- test_data %>%
  mutate(decile = ntile(pred_multi, 10)) %>%
  group_by(decile) %>%
  summarise(mean_pred = mean(pred_multi),
            obs_rate = mean(Winner),
            count = n(),
            .groups = "drop")

cal_plot <- ggplot(calibration_data, aes(x = mean_pred, y = obs_rate)) +
  geom_point(size = 3, color = "darkgreen") +
  geom_line(color = "darkgreen") +
  geom_abline(intercept = 0, slope = 1, linetype = "dashed", color =
"grey50") +
  labs(title = "Calibration Plot for Multi Variable Model (Deciles)",
       x = "Mean Predicted Probability",
       y = "Observed Winner Rate") +
  theme_minimal()
print(cal_plot)
```



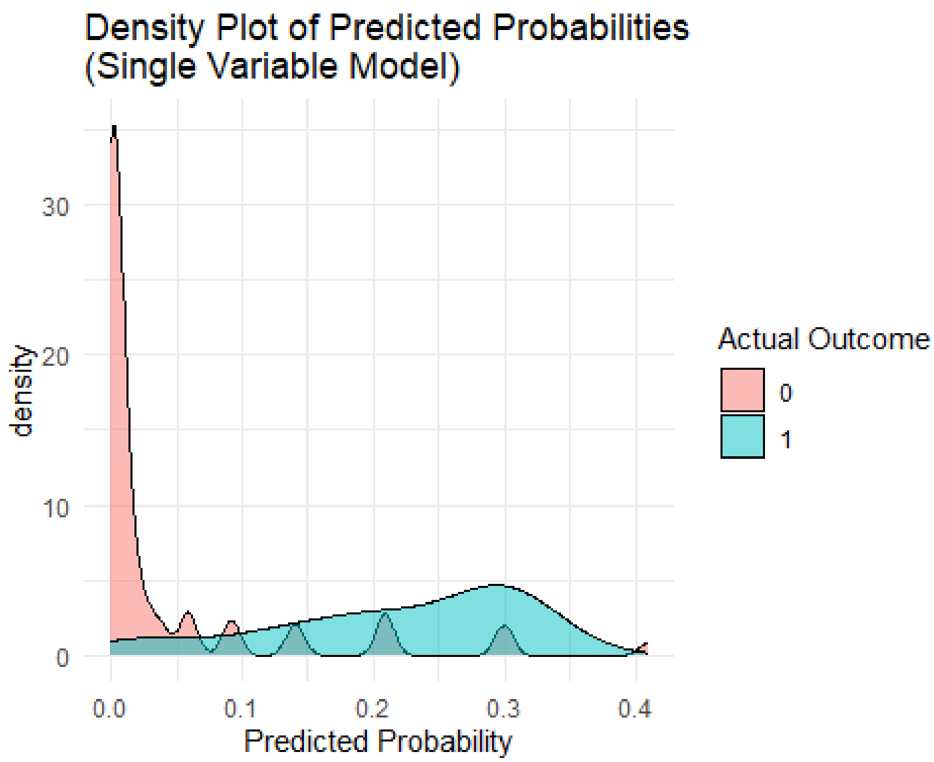

```r
# Calibration plot for single-variable model
calibration_data <- test_data %>%
  mutate(decile = ntile(pred_single, 10)) %>%
  group_by(decile) %>%
  summarise(mean_pred = mean(pred_single),
            obs_rate = mean(Winner),
            count = n(),
            .groups = "drop")

cal_plot <- ggplot(calibration_data, aes(x = mean_pred, y = obs_rate)) +
  geom_point(size = 3, color = "darkgreen") +
  geom_line(color = "darkgreen") +
  geom_abline(intercept = 0, slope = 1, linetype = "dashed", color =
"grey50") +
  labs(title = "Calibration Plot for Single Variable Model (Deciles)",
       x = "Mean Predicted Probability",
       y = "Observed Winner Rate") +
  theme_minimal()
print(cal_plot)
```



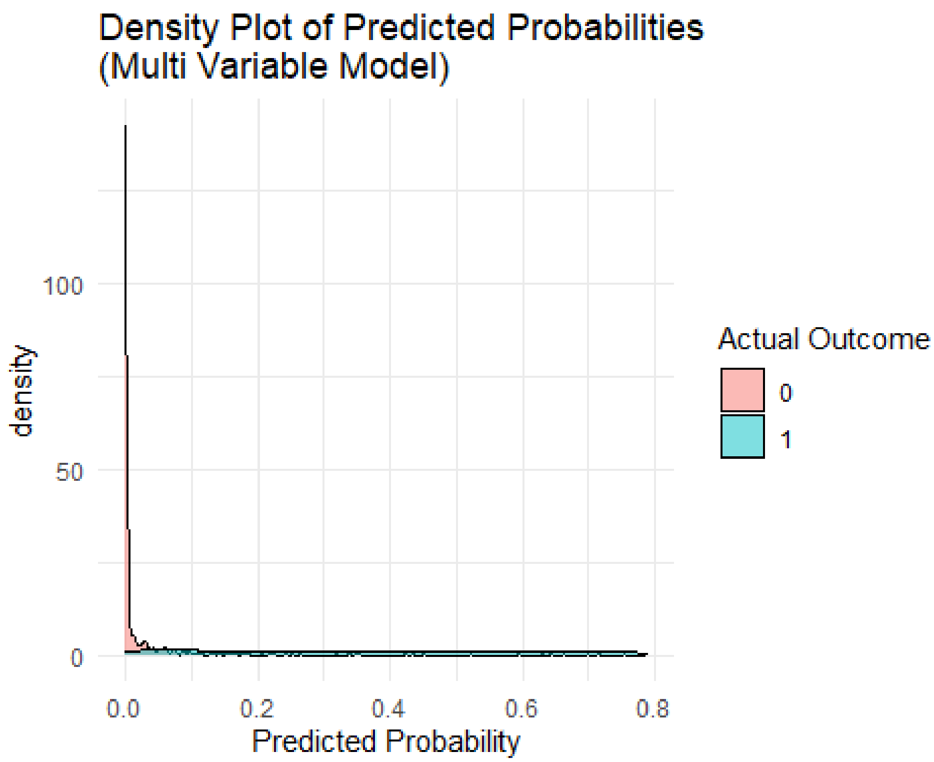

```r
# Driver-level summary plot for multi-variable predictions
driver_summary <- test_data %>%
  group_by(Driver) %>%
  summarise(mean_pred = mean(pred_multi, na.rm = TRUE),
            win_rate = mean(Winner, na.rm = TRUE),
            count = n(),
            .groups = "drop")
ggplot(driver_summary, aes(x = mean_pred, y = win_rate, label = Driver)) +
  geom_point(color = "darkgreen", size = 3) +
  geom_text(vjust = -0.5, size = 4) +
  labs(title = "Driver-Level Average Predicted Probability vs. Observed Win
Rate",
       x = "Average Predicted Probability",
       y = "Observed Win Rate") +
  theme_minimal()
```




```
# Density plots for both models
print(
  ggplot(test_data, aes(x = pred_single, fill = factor(Winner))) +
    geom_density(alpha = 0.5) +
    labs(title = "Density Plot of Predicted Probabilities\n(Single Variable
Model)",
         x = "Predicted Probability",
         fill = "Actual Outcome") +
    theme_minimal()
)
```




```
print(
  ggplot(test_data, aes(x = pred_multi, fill = factor(Winner))) +
    geom_density(alpha = 0.5) +
    labs(title = "Density Plot of Predicted Probabilities\n(Multi Variable
Model)",
         x = "Predicted Probability",
         fill = "Actual Outcome") +
    theme_minimal()
)
```




```r
obs <- test_data$Winner

#-- TOTAL SUM OF SQUARES (same denominator for both models)
sst <- sum((obs - mean(obs))^2)

#-- Single-variable model metrics
pred1       <- test_data$pred_single
pred1_cls   <- test_data$pred_single_class

# RMSE on probabilities
rmse1 <- sqrt(mean((obs - pred1)^2))

# R² on binary classes: 1 – SSE/SST
sse1  <- sum((obs - pred1_cls)^2)


#-- Multi-variable model metrics
predm      <- test_data$pred_multi
predm_cls  <- test_data$pred_multi_class

rmse_m <- sqrt(mean((obs - predm)^2))
sse_m  <- sum((obs - predm_cls)^2)
t

## function (x)
## UseMethod("t")
```

```
## <bytecode: 0x0000011cd43c1ce8>
## <environment: namespace:base>

#-- Print results
cat("Single-Variable Model (GridPosition):\n")

## Single-Variable Model (GridPosition):

cat("  RMSE (probabilities):", round(rmse1, 4), "\n")

##   RMSE (probabilities): 0.2063

cat("Multi-Variable Model (full):\n")

## Multi-Variable Model (full):

cat("  RMSE (probabilities):", round(rmse_m, 4), "\n")

##   RMSE (probabilities): 0.1999
```

F1 Driver Clustering Analysis (2020–2024)

Objective

This analysis clusters Formula 1 drivers based on their performance over the 2020–2024 seasons. The goal is to group drivers into meaningful categories—such as top performers, midfield, and developing talents—based on qualifying positions, finishing positions, total points, and lap time performance using **Principal Component Analysis (PCA)** and **K-Means Clustering**.

Data Preprocessing

- We load two sheets from the dataset:
  - `Results`: Contains race-wise positions, points, driver info
  - `Laps`: Contains lap-by-lap timing for each driver
- Race outcomes are filtered to exclude non-finishers (e.g., `DNF`, `DNS`, `DQ`).
- Driver statistics are **aggregated per season**, calculating:
  - Average **grid position**
  - Average **finish position**
  - **Total points**
  - **Average lap time**
- The two datasets are then merged using `Year` and `Driver` as keys.

Dimensionality Reduction (PCA)

- The four features (`AvgGridPosition`, `AvgFinishPosition`, `TotalPoints`, and `AvgLapTime`) are **standardized**.

- **Principal Component Analysis (PCA)** is used to reduce dimensionality to 2 axes:
    - **PC1:** Captures overall performance and speed.
    - **PC2:** Captures consistency and race execution.

Clustering (K-Means)

- We apply **K-Means Clustering** with `k = 3` on the PCA-reduced feature space.
- The resulting clusters are interpreted as:
    - **Top Performers:** Drivers with fast lap times, consistent finishes, and high points.
    - **Midfield Performers:** Competent drivers with occasional standout races.
    - **Developing Talents:** Drivers with inconsistent results or limited race data.

PCA Visualization

- A scatter plot of PC1 vs. PC2 is generated, showing driver clusters in a 2D plane.
- Text labels identify standout seasons (e.g., `Verstappen '21`, `Hamilton '20`).
- Confidence ellipses show the spread of each cluster.
- The **left-most area** (low PC1) generally corresponds to high-performing drivers.

Top Drivers per Cluster

- The top 3 drivers from each cluster (based on **total points**) are shown in a faceted bar plot.
- This highlights that:
    - **Verstappen** frequently ranks as a top performer across years.
    - **Norris** and **Sainz** represent midfield standouts.
    - **Gasly** and **Vettel** lead the developing category in certain seasons.

Radar Chart Comparison

- We compute the **mean of four key metrics** for each cluster:
    - Average Grid Position
    - Average Finish Position
    - Average Lap Time
    - Total Points
- These metrics are **normalized to 0–1** and visualized on a **radar chart**.
- Interpretation:
    - **Top Performers** have the **lowest grid/finish positions** (better ranks) and highest points.
    - **Developing Talents** rank the lowest in most metrics.
    - **Midfield Performers** sit between the two extremes, showcasing a mix of reliability and potential.

Conclusion

- This unsupervised learning approach allows us to categorize F1 drivers beyond pure rankings.
- **PCA** helps reduce complexity while preserving meaningful variance in driver performance.
- **K-means clustering** effectively segments drivers into insightful performance tiers.
- These insights can support:
    - Talent identification and scouting
    - Performance benchmarking across seasons
    - Data-driven storytelling for F1 analysts and fans

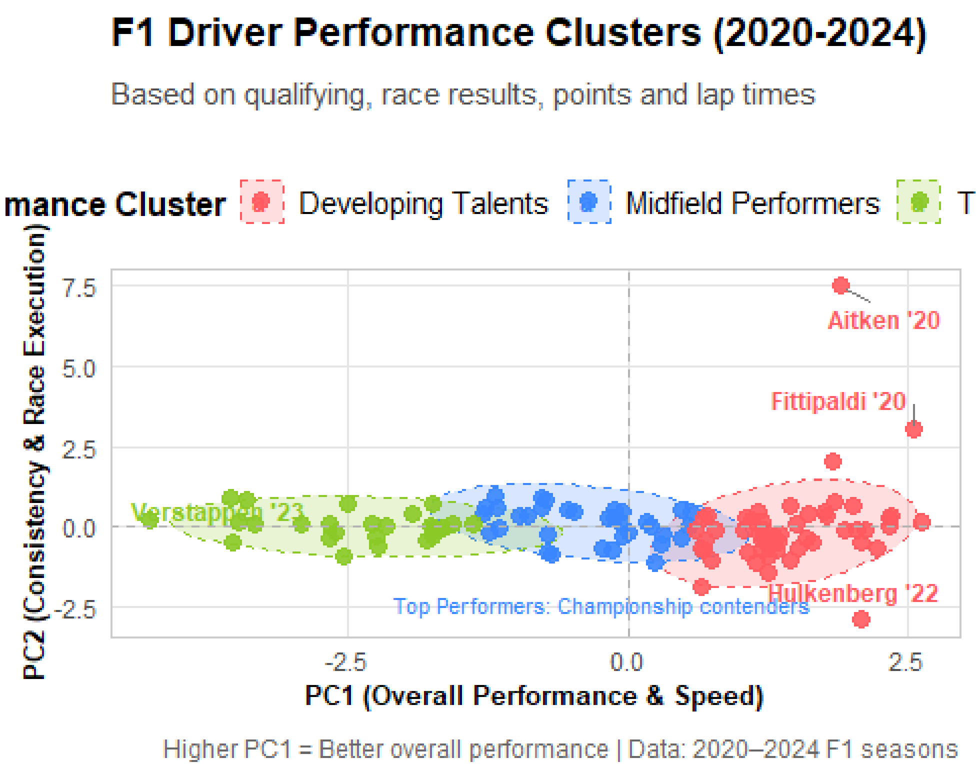
PCS + Clustering

```r
file_path <- "fully_preprocessed_data_no_outliers.xlsx"

# Load data using our custom loader.
results_df <- load_dataset(file_path, sheet = "Results")
laps_df    <- load_dataset(file_path, sheet = "Laps")

# Clean and summarize race results
results_clean <- results_df %>%
  filter(!is.na(FullName), !is.na(GridPosition), !is.na(ClassifiedPosition),
!is.na(Points)) %>%
  filter(!ClassifiedPosition %in% c("R", "NC", "D", "W", "DQ", "DNS", "DNF",
"DSQ")) %>%
  mutate(ClassifiedPosition = as.numeric(ClassifiedPosition)) %>%
  filter(!is.na(ClassifiedPosition)) %>%
  group_by(Year, FullName) %>%
  summarise(
    AvgGridPosition = mean(GridPosition, na.rm = TRUE),
    AvgFinishPosition = mean(ClassifiedPosition, na.rm = TRUE),
    TotalPoints = sum(Points, na.rm = TRUE),
    .groups = "drop"
  )

# Add driver names to lap data
laps_enriched <- laps_df %>%
  left_join(results_df %>%
              select(Year, DriverNumber, EventName, FullName),
            by = c("Year", "DriverNumber", "EventName"))

# Clean and summarize lap data
laps_clean <- laps_enriched %>%
  filter(!is.na(FullName), !is.na(LapTime_sec)) %>%
  group_by(Year, FullName) %>%
```

```r
  summarise(AvgLapTime = mean(LapTime_sec, na.rm = TRUE), .groups = "drop")

# Merge results with lap times
performance_df <- results_clean %>%
  left_join(laps_clean, by = c("Year", "FullName")) %>%
  drop_na()

# Prepare data for PCA
features <- performance_df %>%
  select(AvgGridPosition, AvgFinishPosition, TotalPoints, AvgLapTime)
features_scaled <- scale(features)

# Run PCA
pca_res <- prcomp(features_scaled, center = TRUE, scale. = TRUE)

# Add first 2 principal components
pca_df <- as.data.frame(pca_res$x[, 1:2])
colnames(pca_df) <- c("PC1", "PC2")
performance_df <- bind_cols(performance_df, pca_df)

# Run K-means clustering
set.seed(42)
kmeans_res <- kmeans(features_scaled, centers = 3)
performance_df$Cluster <- as.factor(kmeans_res$cluster)

# Rename clusters
cluster_names <- c("Midfield Performers", "Top Performers", "Developing
Talents")
performance_df$ClusterName <-
cluster_names[as.numeric(performance_df$Cluster)]

# Create short labels
performance_df <- performance_df %>%
  mutate(
    LastName = sub("^.* ", "", FullName),
    ShortLabel = paste0(LastName, " '", substr(Year, 3, 4))
  )

# Define custom colors
custom_palette <- c("#FF5A5F", "#3A86FF", "#8AC926")

# Plot PCA with clusters
plot <- ggplot(performance_df, aes(x = PC1, y = PC2, color = ClusterName)) +
  stat_ellipse(aes(fill = ClusterName), geom = "polygon", alpha = 0.2,
               linetype = 2, size = 0.5) +
  geom_point(size = 3, alpha = 0.9) +
  geom_text_repel(
    aes(label = ShortLabel),
    size = 3.2, fontface = "bold",
```

```r
    box.padding = 0.5, point.padding = 0.3,
    force = 10, max.overlaps = 15,
    segment.color = "grey50", segment.size = 0.25
  ) +
  scale_color_manual(values = custom_palette) +
  scale_fill_manual(values = custom_palette) +
  labs(
    title = "F1 Driver Performance Clusters (2020-2024)",
    subtitle = "Based on qualifying, race results, points and lap times",
    x = "PC1 (Overall Performance & Speed)",
    y = "PC2 (Consistency & Race Execution)",
    color = "Performance Cluster",
    fill = "Performance Cluster",
    caption = "Higher PC1 = Better overall performance | Data: 2020–2024 F1
seasons"
  ) +
  theme_minimal() +
  theme(
    legend.position = "top",
    legend.box = "horizontal",
    legend.text = element_text(size = 11),
    legend.title = element_text(face = "bold", size = 12),
    plot.title = element_text(face = "bold", size = 14, margin = margin(b =
10)),
    plot.subtitle = element_text(size = 11, color = "grey30", margin =
margin(b = 20)),
    plot.caption = element_text(size = 9, color = "grey40", margin = margin(t
= 10)),
    axis.title = element_text(face = "bold", size = 10),
    panel.grid.major = element_line(color = "grey90"),
    panel.grid.minor = element_blank(),
    panel.border = element_rect(color = "grey80", fill = NA)
  ) +
  annotate("text", x = max(performance_df$PC1) - 1, y =
min(performance_df$PC2) + 0.5,
           label = "Top Performers: Championship contenders",
           size = 2.8, hjust = 1, color = custom_palette[2]) +
  geom_hline(yintercept = 0, linetype = "dashed", color = "grey70", size =
0.5) +
  geom_vline(xintercept = 0, linetype = "dashed", color = "grey70", size =
0.5)

## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## ℹ Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.

# Show plot
print(plot)
```

```
## Warning: ggrepel: 108 unlabeled data points (too many overlaps). Consider
## increasing max.overlaps
```

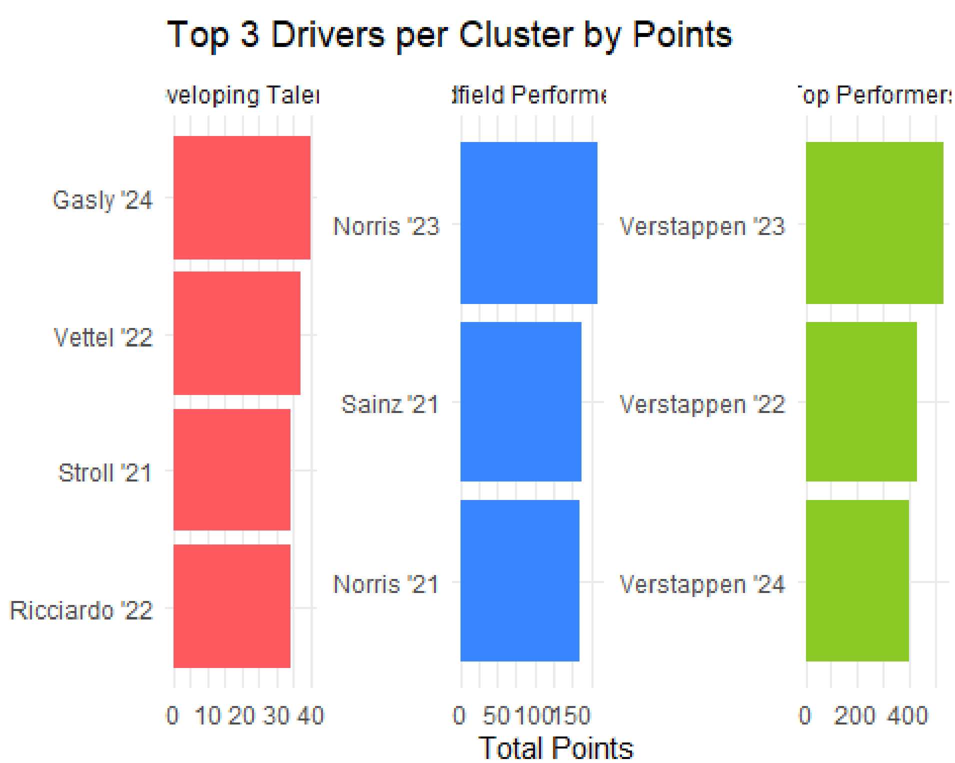



```
# Save plot
ggsave("F1_Driver_Performance_Clusters.png", plot, width = 10, height = 8,
dpi = 300)

## Warning: ggrepel: 55 unlabeled data points (too many overlaps). Consider
## increasing max.overlaps

# Show driver-cluster assignments
driver_clusters <- performance_df %>%
  arrange(ClusterName, Year, LastName) %>%
  select(Year, FullName, ClusterName) %>%
  print(n = Inf)

## # A tibble: 112 × 3
##     Year  FullName           ClusterName
##     <chr> <chr>              <chr>
##   1 2020  Jack Aitken        Developing Talents
##   2 2020  Pietro Fittipaldi  Developing Talents
##   3 2020  Antonio Giovinazzi Developing Talents
##   4 2020  Romain Grosjean    Developing Talents
##   5 2020  Nicholas Latifi    Developing Talents
##   6 2020  Kevin Magnussen    Developing Talents
##   7 2020  George Russell     Developing Talents
##   8 2020  Kimi Räikkönen     Developing Talents
```

```
##    9 2021  Antonio Giovinazzi Developing Talents
##   10 2021  Robert Kubica      Developing Talents
##   11 2021  Nicholas Latifi    Developing Talents
##   12 2021  Nikita Mazepin     Developing Talents
##   13 2021  George Russell     Developing Talents
##   14 2021  Kimi Räikkönen     Developing Talents
##   15 2021  Mick Schumacher    Developing Talents
##   16 2021  Lance Stroll       Developing Talents
##   17 2021  Yuki Tsunoda       Developing Talents
##   18 2022  Alexander Albon    Developing Talents
##   19 2022  Pierre Gasly       Developing Talents
##   20 2022  Nico Hulkenberg    Developing Talents
##   21 2022  Nicholas Latifi    Developing Talents
##   22 2022  Kevin Magnussen    Developing Talents
##   23 2022  Daniel Ricciardo   Developing Talents
##   24 2022  Mick Schumacher    Developing Talents
##   25 2022  Lance Stroll       Developing Talents
##   26 2022  Yuki Tsunoda       Developing Talents
##   27 2022  Sebastian Vettel   Developing Talents
##   28 2022  Guanyu Zhou        Developing Talents
##   29 2023  Alexander Albon    Developing Talents
##   30 2023  Valtteri Bottas    Developing Talents
##   31 2023  Nico Hulkenberg    Developing Talents
##   32 2023  Liam Lawson        Developing Talents
##   33 2023  Kevin Magnussen    Developing Talents
##   34 2023  Daniel Ricciardo   Developing Talents
##   35 2023  Logan Sargeant     Developing Talents
##   36 2023  Yuki Tsunoda       Developing Talents
##   37 2023  Nyck De Vries      Developing Talents
##   38 2023  Guanyu Zhou        Developing Talents
##   39 2024  Alexander Albon    Developing Talents
##   40 2024  Oliver Bearman     Developing Talents
##   41 2024  Valtteri Bottas    Developing Talents
##   42 2024  Franco Colapinto   Developing Talents
##   43 2024  Jack Doohan        Developing Talents
##   44 2024  Pierre Gasly       Developing Talents
##   45 2024  Liam Lawson        Developing Talents
##   46 2024  Kevin Magnussen    Developing Talents
##   47 2024  Esteban Ocon       Developing Talents
##   48 2024  Daniel Ricciardo   Developing Talents
##   49 2024  Logan Sargeant     Developing Talents
##   50 2024  Lance Stroll       Developing Talents
##   51 2024  Yuki Tsunoda       Developing Talents
##   52 2024  Guanyu Zhou        Developing Talents
##   53 2020  Alexander Albon    Midfield Performers
##   54 2020  Pierre Gasly       Midfield Performers
##   55 2020  Nico Hulkenberg    Midfield Performers
##   56 2020  Daniil Kvyat       Midfield Performers
##   57 2020  Charles Leclerc    Midfield Performers
##   58 2020  Lando Norris       Midfield Performers
```

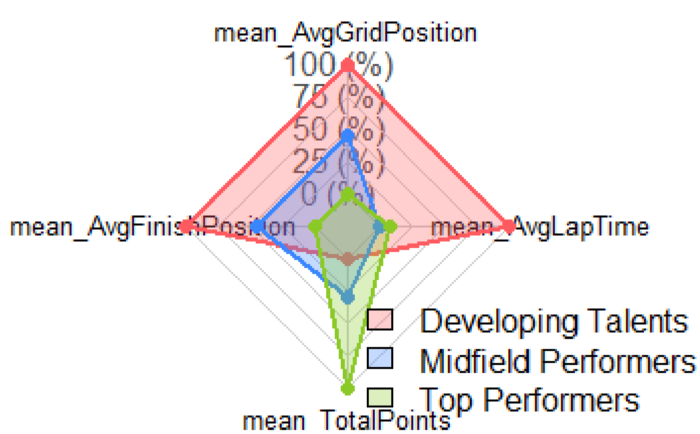
```
##  59 2020  Esteban Ocon        Midfield Performers
##  60 2020  Sergio Perez        Midfield Performers
##  61 2020  Daniel Ricciardo    Midfield Performers
##  62 2020  Carlos Sainz        Midfield Performers
##  63 2020  Lance Stroll        Midfield Performers
##  64 2020  Sebastian Vettel    Midfield Performers
##  65 2021  Fernando Alonso     Midfield Performers
##  66 2021  Pierre Gasly        Midfield Performers
##  67 2021  Charles Leclerc     Midfield Performers
##  68 2021  Lando Norris        Midfield Performers
##  69 2021  Esteban Ocon        Midfield Performers
##  70 2021  Daniel Ricciardo    Midfield Performers
##  71 2021  Carlos Sainz        Midfield Performers
##  72 2021  Sebastian Vettel    Midfield Performers
##  73 2022  Fernando Alonso     Midfield Performers
##  74 2022  Valtteri Bottas     Midfield Performers
##  75 2022  Lando Norris        Midfield Performers
##  76 2022  Esteban Ocon        Midfield Performers
##  77 2022  Nyck De Vries       Midfield Performers
##  78 2023  Pierre Gasly        Midfield Performers
##  79 2023  Lando Norris        Midfield Performers
##  80 2023  Esteban Ocon        Midfield Performers
##  81 2023  Oscar Piastri       Midfield Performers
##  82 2023  George Russell      Midfield Performers
##  83 2023  Lance Stroll        Midfield Performers
##  84 2024  Fernando Alonso     Midfield Performers
##  85 2024  Nico Hulkenberg     Midfield Performers
##  86 2024  Sergio Perez        Midfield Performers
##  87 2020  Valtteri Bottas     Top Performers
##  88 2020  Lewis Hamilton      Top Performers
##  89 2020  Max Verstappen      Top Performers
##  90 2021  Valtteri Bottas     Top Performers
##  91 2021  Lewis Hamilton      Top Performers
##  92 2021  Sergio Perez        Top Performers
##  93 2021  Max Verstappen      Top Performers
##  94 2022  Lewis Hamilton      Top Performers
##  95 2022  Charles Leclerc     Top Performers
##  96 2022  Sergio Perez        Top Performers
##  97 2022  George Russell      Top Performers
##  98 2022  Carlos Sainz        Top Performers
##  99 2022  Max Verstappen      Top Performers
## 100 2023  Fernando Alonso     Top Performers
## 101 2023  Lewis Hamilton      Top Performers
## 102 2023  Charles Leclerc     Top Performers
## 103 2023  Sergio Perez        Top Performers
## 104 2023  Carlos Sainz        Top Performers
## 105 2023  Max Verstappen      Top Performers
## 106 2024  Lewis Hamilton      Top Performers
## 107 2024  Charles Leclerc     Top Performers
## 108 2024  Lando Norris        Top Performers
```

```
## 109 2024  Oscar Piastri      Top Performers
## 110 2024  George Russell     Top Performers
## 111 2024  Carlos Sainz       Top Performers
## 112 2024  Max Verstappen     Top Performers

top_by_cluster <- performance_df %>%
  group_by(ClusterName) %>%
  top_n(3, TotalPoints)

ggplot(top_by_cluster, aes(x = reorder(ShortLabel, TotalPoints), y =
TotalPoints, fill = ClusterName)) +
  geom_bar(stat = "identity", show.legend = FALSE) +
  facet_wrap(~ ClusterName, scales = "free") +
  coord_flip() +
  labs(title = "Top 3 Drivers per Cluster by Points", x = NULL, y = "Total
Points") +
  scale_fill_manual(values = custom_palette) +
  theme_minimal()
```




```
#Radar chat
library(fmsb)

## Warning: package 'fmsb' was built under R version 4.4.3

## Registered S3 methods overwritten by 'fmsb':
##   method    from
```

```
##   print.roc pROC
##   plot.roc  pROC

##
## Attaching package: 'fmsb'

## The following object is masked from 'package:pROC':
##
##     roc

cluster_summary <- performance_df %>%
  group_by(ClusterName) %>%
  summarise(across(c(AvgGridPosition, AvgFinishPosition, TotalPoints,
AvgLapTime), mean, .names = "mean_{col}"))

# Normalize between 0-1 for radar
radar_data <- as.data.frame(lapply(cluster_summary[-1], scales::rescale))
radar_data <- rbind(rep(1, ncol(radar_data)), rep(0, ncol(radar_data)),
radar_data)
rownames(radar_data) <- c("Max", "Min", cluster_summary$ClusterName)

radarchart(radar_data, axistype = 1, pcol = custom_palette, pfcol =
scales::alpha(custom_palette, 0.3),
           plwd = 2, plty = 1, cglcol = "grey80", cglty = 1, axislabcol =
"grey30", vlcex = 0.8)
legend("bottomright", legend = cluster_summary$ClusterName, fill =
scales::alpha(custom_palette, 0.3), bty = "n")
```



```r
detach("package:fmsb", unload = TRUE)
```

Verstappen Lap Time Forecasting using XGBoost

Objective

The goal of this analysis is to forecast Max Verstappen's lap times using an XGBoost regression model. This approach utilizes real-time race telemetry, tire usage data, and weather metrics to generate accurate lap time predictions. It also helps simulate pit stop strategy by forecasting the next few laps.

Model Overview

- **Features Used:** Lap number, tire compound, tire life, weather (pressure, wind), track status, and lag/rolling lap metrics.
- **Model:** XGBoost regression (`reg:squarederror`)
- **Train/Test Split:** 80/20 chronological split to simulate real-time forecasting
- **Evaluation Metrics:**
  - RMSE: `3.42 sec`
  - R^2 Score: `0.916`

Feature Importance

- This bar chart shows the **relative contribution of each feature** to the final model.
- **Lag_1** (previous lap time) is the most important predictor, showing strong autocorrelation in lap behavior.
- Rolling averages like **RollingMean_3** and **RollingMean_2** help capture momentum or degradation patterns.
- Tire-related features like **TyreLife** and **Compound** also rank high, highlighting their influence on lap time.

Actual vs. Predicted Lap Times (by Grand Prix)

- This faceted plot shows **actual vs. predicted lap times** for multiple 2021 races.
- **Blue lines** represent real lap times; **Red lines** represent predicted lap times.
- The predictions closely track the actual times, including spikes due to pit stops or safety car deployments.
- Some deviations occur in laps with anomalies (e.g., out-laps, incidents), which is expected in time-series forecasting.

Smoothed Lap Time Trends

- A **LOESS-smoothed version** of the same lap time trends helps visualize **underlying patterns**.
- The model generally **tracks the fatigue curve**, showing gradual increases in lap time as tire wear increases.

- Divergences between predicted and actual in early or late laps may reflect strategy shifts or weather variability.

Scatter Plot: Actual vs. Predicted Lap Times

- Each point corresponds to a single lap.
- The **closer to the diagonal**, the better the prediction.
- Most predictions are tightly clustered around the line, reinforcing the model's **strong linear correspondence**.
- **Outliers** indicate laps with unexpected disturbances (e.g., yellow flags, safety cars).

Track Status Impact Visualization

- This plot overlays lap times with **track status conditions** (coded numerically).
- Laps with certain flags (e.g., **TrackStatus = 4 or 12**) show significant spikes in lap time.
- The model is able to partially accommodate these disruptions, thanks to one-hot encoding of track status.

Forecasting the Next 3 Laps

RMSE: 3.42

R^2 Score: 0.916

Conclusion

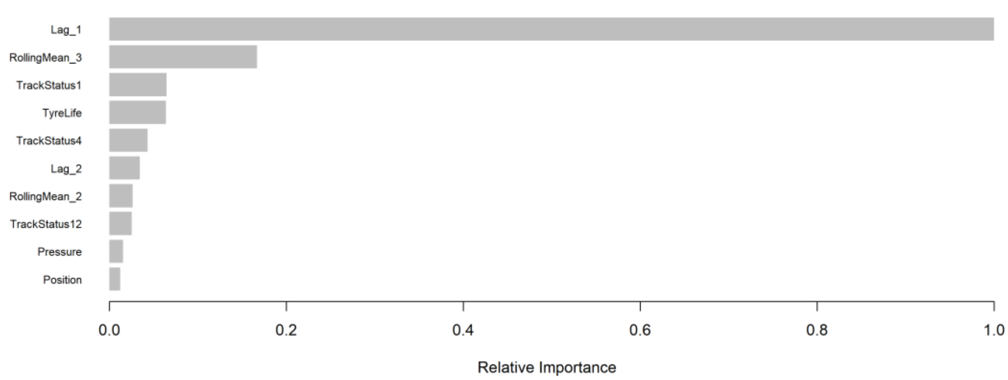
- The XGBoost regression model demonstrates **strong predictive power** in forecasting Max Verstappen's lap times with an **R^2 of 0.916** and an **RMSE of ~3.42 seconds**.
- Feature importance analysis confirms that **recent lap history (Lag_1, rolling averages)** and **tire condition (TyreLife, Compound)** are key to accurate forecasting.
- The model's predictions **closely match actual lap behavior**, even across varying track conditions and races.
- **Smoothed lap time trends** and **scatter plots** confirm that the model captures broader performance dynamics while remaining resilient to noise.
- **Track status visualization** validates that disruptions such as yellow flags or safety car conditions are reflected in the predictions.
- Forecasting the next few laps reveals valuable insights into **tire degradation**, supporting decisions like whether to **pit or stay out**.

This approach lays the foundation for **real-time strategy optimization**, **driver monitoring**, and **telemetry-driven analytics** in modern motorsport.

```r
# Load sheets
data_laps <- read_excel("fastf1_full_data_2020_2024.xlsx", sheet = "Laps")
data_weather <- read_excel("fastf1_full_data_2020_2024.xlsx", sheet =
"Weather")

# Rename weather columns to avoid conflicts
names(data_weather) <- paste0("weather_", names(data_weather))
data <- bind_cols(data_laps, data_weather)

# Drop unwanted columns
cols_to_drop <- c("Time", "LapTime", "PitOutTime", "PitInTime",
"Sector1Time", "Sector2Time", "Sector3Time",
                  "Sector1SessionTime", "Sector2SessionTime",
"Sector3SessionTime", "SpeedI1", "SpeedI2",
                  "SpeedFL", "SpeedST", "LapStartTime", "LapStartDate",
"Deleted", "DeletedReason",
                  "FastF1Generated", "IsAccurate", "Driver",
"IsPersonalBest", "Team", "EventName")
data <- data %>% select(-any_of(cols_to_drop))
names(data) <- gsub("weather_", "", names(data))


# Keep relevant columns
final_columns <- c("DriverNumber", "LapNumber", "Stint", "Compound",
"TyreLife", "FreshTyre", "TrackStatus",
                   "Position", "Year", "RoundNumber", "LapTime_sec",
"AirTemp", "Humidity", "Pressure",
                   "Rainfall", "TrackTemp", "WindDirection", "WindSpeed",
"EventName")
data <- data %>% select(any_of(final_columns))

# removed outliers in laptime
data <- data %>% filter(LapTime_sec > 40 & LapTime_sec < 200)
# Encode categorical variables
tyre_map <- c("SOFT" = 0, "MEDIUM" = 1, "HARD" = 2, "INTERMEDIATE" = 3, "WET"
= 4, "UNKNOWN" = -1)
data <- data %>%
  mutate(
    Compound = recode(Compound, !!!tyre_map),
    FreshTyre = as.integer(FreshTyre),
    Rainfall = as.integer(Rainfall)
  )

# Feature engineering
data <- na.omit(data)
data <- data %>%
  group_by(DriverNumber, Year, EventName) %>%
  arrange(LapNumber, .by_group = TRUE) %>%
  mutate(
```

```r
    Lag_1 = lag(LapTime_sec, 1),
    Lag_2 = lag(LapTime_sec, 2),
    RollingMean_2 = lag(rollmean(LapTime_sec, k = 2, fill = NA, align =
"right")),
    RollingMean_3 = lag(rollmean(LapTime_sec, k = 3, fill = NA, align =
"right"))
  ) %>%
  ungroup()

data <- drop_na(data)



# Convert DriverNumber and TrackStatus to categorical
data$DriverNumber <- as.factor(data$DriverNumber)
data$TrackStatus <- as.factor(data$TrackStatus)

# Filter Verstappen data
max_data <- data %>% filter(DriverNumber %in% c(33, 1))

event_names <- max_data$EventName
recent_avg <- tail(max_data$RollingMean_3, 1)

# Use same feature subset as Python
selected_features <- c("TrackStatus", "Pressure", "Year", "RoundNumber",
"FreshTyre",
                       "Position", "LapNumber", "TyreLife", "Compound",
"Stint",
                       "Lag_1", "Lag_2", "RollingMean_2", "RollingMean_3")

max_data <- max_data %>%
  arrange(Year, RoundNumber, LapNumber)

# Select and encode features
X_raw <- max_data %>% select(all_of(selected_features))
X <- model.matrix(~ . -1, data = X_raw)  # One-hot encode
y <- max_data$LapTime_sec


row_indices <- 1:nrow(max_data)
# Train-test split
split_index <- floor(0.9 * nrow(X))
X_train <- X[1:split_index, ]
X_test  <- X[(split_index + 1):nrow(X), ]
y_train <- y[1:split_index]
y_test  <- y[(split_index + 1):length(y)]
test_indices <- row_indices[(split_index + 1):nrow(X)]
```

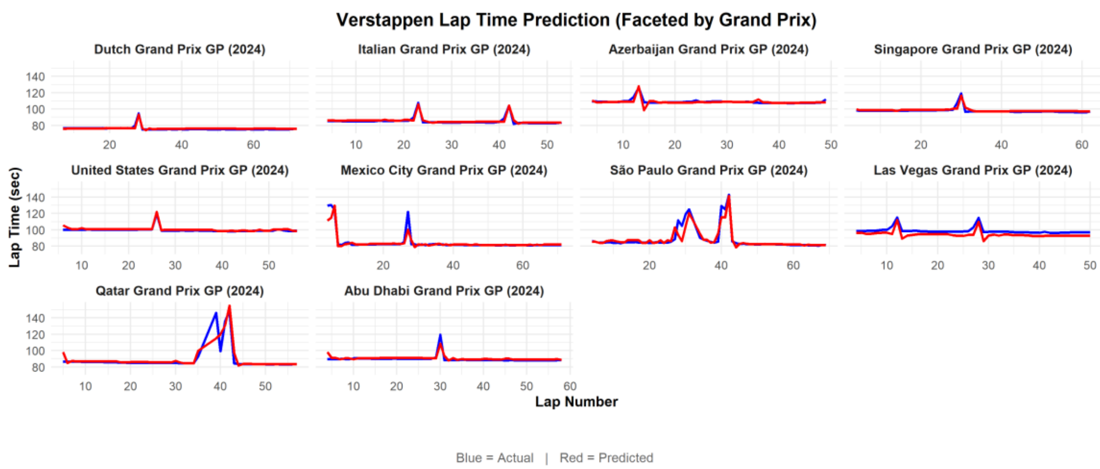
```r
year_train <- X_raw[1:split_index, "Year"]
year_test <- X_raw[(split_index + 1):nrow(X_raw), "Year"]


# Create DMatrix
dtrain <- xgb.DMatrix(data = X_train, label = y_train)
dtest <- xgb.DMatrix(data = X_test, label = y_test)

# Parameters
params <- list(
  objective = "reg:squarederror",
  eta = 0.2,             # Reduce learning rate
  max_depth = 3,         # Lower depth to avoid overfitting
  subsample = 0.9,
  colsample_bytree = 0.8
)

# Train model
set.seed(42)
model <- xgboost(params = params, data = dtrain, nrounds = 300, verbose = 0)
importance <- xgb.importance(model = model)
xgb.plot.importance(importance, top_n = 10, rel_to_first = TRUE, xlab =
"Relative Importance")
```



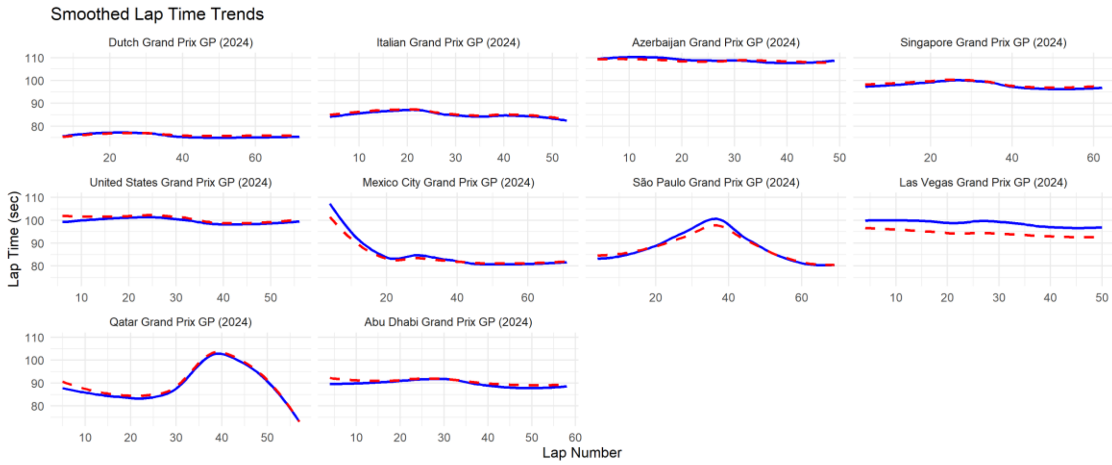

```r
# Predict and evaluate
y_pred <- predict(model, dtest)
rmse <- sqrt(mean((y_test - y_pred)^2))
r2 <- 1 - sum((y_test - y_pred)^2) / sum((y_test - mean(y_test))^2)
cat("RMSE:", rmse, "\n")

## RMSE: 3.421272

cat("R² Score:", r2, "\n")

## R² Score: 0.9163451
```

```r
# Plot results
n_test <- length(y_test)
test_indices <- (nrow(max_data) - n_test + 1):nrow(max_data)

lap_df <- max_data[test_indices, ] %>%
  mutate(
    Actual = y_test,
    Predicted = y_pred,
    Race = paste0(EventName, " GP (", Year, ")")
  )


# Order Race by RoundNumber and Year
race_order <- lap_df %>%
  distinct(Race, Year, RoundNumber) %>%
  arrange(Year, RoundNumber) %>%
  pull(Race)

# Convert Race to ordered factor
lap_df$Race <- factor(lap_df$Race, levels = unique(race_order))

facet_wrap(~ Race, scales = "free_x")

## <ggproto object: Class FacetWrap, Facet, gg>
##     compute_layout: function
##     draw_back: function
##     draw_front: function
##     draw_labels: function
##     draw_panels: function
##     finish_data: function
##     init_scales: function
##     map_data: function
##     params: list
##     setup_data: function
##     setup_params: function
##     shrink: TRUE
##     train_scales: function
##     vars: function
##     super:  <ggproto object: Class FacetWrap, Facet, gg>

# Plot
p <- ggplot(lap_df, aes(x = LapNumber)) +
  geom_line(aes(y = Actual), color = "blue", size = 1) +
  geom_line(aes(y = Predicted), color = "red", size = 1) +
  facet_wrap(~ Race, scales = "free_x") +
  labs(
    title = "Verstappen Lap Time Prediction (Faceted by Grand Prix)",
    x = "Lap Number",
    y = "Lap Time (sec)"
  ) +
```

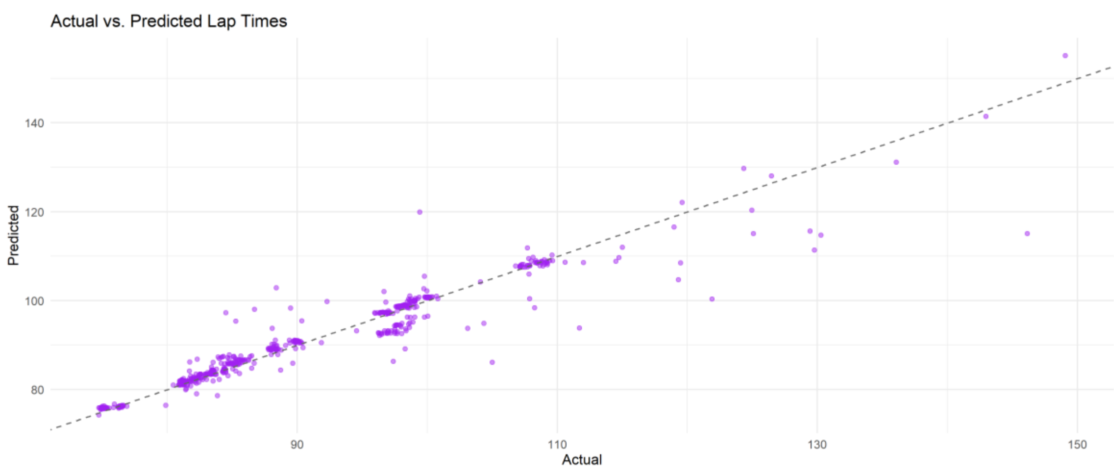
```
  theme_minimal() +
  theme(
    strip.text = element_text(face = "bold", size = 10),
    plot.title = element_text(face = "bold", hjust = 0.5, size = 14),
    axis.title = element_text(face = "bold", size = 11),
    plot.margin = margin(10, 10, 30, 10)  # more space for caption
  )

# Caption (legend) at the bottom
caption <- textGrob("Blue = Actual    |    Red = Predicted",
                    gp = gpar(col = "gray40", fontsize = 10))

# Combine plot and caption
grid.arrange(p, bottom = caption)
```




```
#Smoothed Laptiming trends
ggplot(lap_df, aes(x = LapNumber)) +
  geom_smooth(aes(y = Actual), color = "blue", se = FALSE) +
  geom_smooth(aes(y = Predicted), color = "red", linetype = "dashed", se =
FALSE) +
  facet_wrap(~ Race, scales = "free_x") +
  labs(title = "Smoothed Lap Time Trends",
       x = "Lap Number", y = "Lap Time (sec)") +
  theme_minimal()

## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```

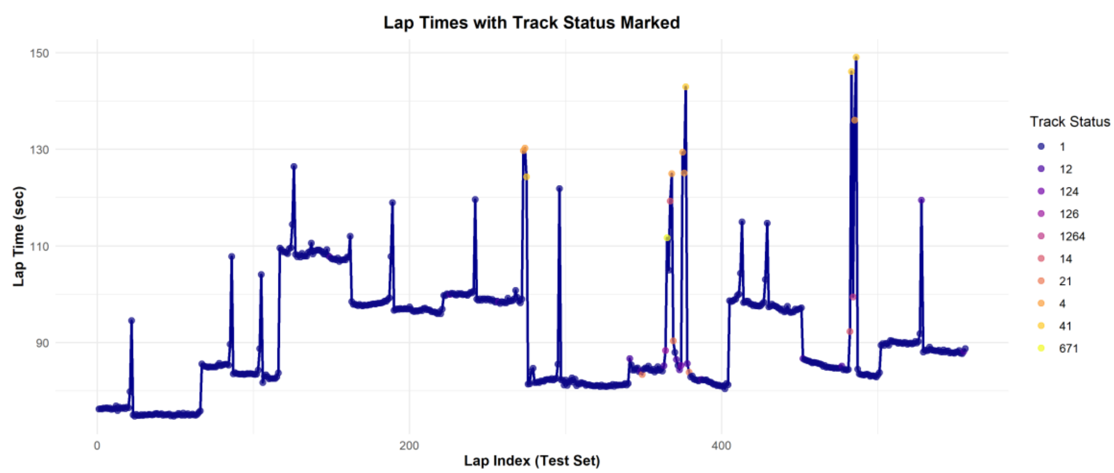



```r
# Using Scatter plot to compare actual vs predicted value. Closer to
diagonal, better prediction
ggplot(lap_df, aes(x = Actual, y = Predicted)) +
  geom_point(alpha = 0.5, color = "purple") +
  geom_abline(slope = 1, intercept = 0, linetype = "dashed", color =
"gray40") +
  labs(title = "Actual vs. Predicted Lap Times", x = "Actual", y =
"Predicted") +
  theme_minimal()
```




```r
#Ploting How trackstatus had effect on the laptiming

# Use only test data portion
n_test <- length(y_test)
test_indices <- (nrow(max_data) - n_test + 1):nrow(max_data)
lap_df <- max_data[test_indices, ] %>%
  mutate(Row = row_number())  # Use index for plotting

# Plot lap times with track status markers
ggplot(lap_df, aes(x = Row, y = LapTime_sec)) +
  geom_line(color = "darkblue", size = 1) +
```

```r
  geom_point(aes(color = TrackStatus), size = 2, alpha = 0.7) +
  scale_color_viridis_d(option = "C") +
  labs(
    title = "Lap Times with Track Status Marked",
    x = "Lap Index (Test Set)",
    y = "Lap Time (sec)",
    color = "Track Status"
  ) +
  theme_minimal() +
  theme(
    plot.title = element_text(face = "bold", hjust = 0.5),
    axis.title = element_text(face = "bold")
  )
```




```r
# Predict next 3 laps
latest <- X_test[nrow(X_test), , drop = FALSE]
predicted_next_3 <- c()
temp_X <- latest

for (i in 1:3) {
  next_pred <- predict(model, xgb.DMatrix(as.matrix(temp_X)))
  predicted_next_3 <- c(predicted_next_3, next_pred)

  temp_X[, "Lag_2"] <- temp_X[, "Lag_1"]
  temp_X[, "Lag_1"] <- next_pred
  temp_X[, "RollingMean_2"] <- mean(c(temp_X[, "Lag_1"], temp_X[, "Lag_2"]))
  temp_X[, "RollingMean_3"] <- mean(c(temp_X[, "Lag_1"], temp_X[, "Lag_2"],
latest[, "Lag_2"]))
  temp_X[, "TyreLife"] <- temp_X[, "TyreLife"] + 1
  temp_X[, "LapNumber"] <- temp_X[, "LapNumber"] + 1
}

avg_next_3 <- mean(predicted_next_3)
```

```r
# Styled output
cli::cli_h1("Next 3 Lap Time Predictions")

##
## ── Next 3 Lap Time Predictions
──────────────────────────────────────────────────

cli::cli_alert_info("Most recent 3-lap average: {.strong {round(recent_avg,
2)}} sec")

## ℹ Most recent 3-lap average: 99.87 sec

cli::cli_alert_info("Predicted next 3-lap average: {.strong
{round(avg_next_3, 2)}} sec")

## ℹ Predicted next 3-lap average: 89.08 sec

cat("\n")

cli::cli_text("Predicted lap times:")

## Predicted lap times:

cli::cli_ul()
for (i in 1:3) {
  cli::cli_li("{.emph Lap {i}} → {round(predicted_next_3[i], 2)} sec")
}

## • Lap 1 → 88.83 sec
## • Lap 2 → 89.28 sec
## • Lap 3 → 89.13 sec

cli::cli_end()

cat("\n")

if (!is.na(recent_avg) && avg_next_3 > recent_avg + 3) {
  cli::cli_alert_danger("PIT STOP SUGGESTED — Avg lap time is dropping!")
} else {
  cli::cli_alert_success("Stay Out — Tyres holding up well.")
}

## ✔ Stay Out — Tyres holding up well.
```