F1 Prediction

FORMULA 1 CHAMPIONSHIP EDA AND WINNER PREDICTION

Loading libraries

```
library(ggplot2)
library(dplyr)
library(rpart)
library(corrplot)
library(treemap)
library(treemapify)
library(tree)
library(randomForest)
library(caret)
library(e1071)
library(rpart.plot)
library(car)
```

Data loading

```
df = read.csv("~/Library/Mobile Documents/com~apple~CloudDocs/UNICATT/Data analysis techniques and tools
str(df)
```

```
## 'data.frame':    23693 obs. of  25 variables:
##  $ raceId          : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ driverId        : int  10 15 16 17 18 2 20 21 22 3 ...
##  $ constructorId   : int  7 7 10 9 23 2 9 10 23 3 ...
##  $ circuitId       : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ resultId        : int  7557 7556 7562 7565 7554 7563 7566 7564 7555 7559 ...
##  $ number          : int  10 9 20 14 22 6 15 21 23 16 ...
##  $ grid            : int  19 20 16 8 1 9 3 15 2 5 ...
##  $ positionOrder   : int  4 3 9 12 1 10 13 11 2 6 ...
##  $ points          : num  5 6 0 0 10 0 0 0 8 3 ...
##  $ laps            : int  58 58 58 57 58 58 56 58 58 58 ...
##  $ fastestLapSpeed : num  216 215 215 216 217 ...
##  $ status          : chr  "Finished" "Finished" "Finished" "+1 Lap" ...
##  $ dob             : chr  "1982-03-18" "1974-07-13" "1983-01-11" "1976-08-27" ...
##  $ driv_nationality: chr  "German" "Italian" "German" "Australian" ...
##  $ fullname        : chr  "Timo Glock" "Jarno Trulli" "Adrian Sutil" "Mark Webber" ...
##  $ round           : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ date            : chr  "2009-03-29" "2009-03-29" "2009-03-29" "2009-03-29" ...
```

```
##  $ const_name      : chr  "Toyota" "Toyota" "Force India" "Red Bull" ...
##  $ name            : chr  "Albert Park Grand Prix Circuit" "Albert Park Grand Prix Circuit" "Albert
##  $ const_points    : num  11 11 0 0 18 0 0 0 18 3 ...
##  $ const_wins      : int  0 0 0 0 1 0 0 0 1 0 ...
##  $ driver_age      : int  27 35 26 33 29 32 22 36 37 24 ...
##  $ fastestLap_ms   : num  88416 88916 88943 88508 88020 ...
##  $ winner          : int  0 0 0 0 1 0 0 0 0 0 ...
##  $ wins            : int  1 1 1 1 2 1 1 1 1 1 ...
```

```
head(df)
```

```
##   raceId driverId constructorId circuitId resultId number grid positionOrder
## 1      1       10             7         1     7557     10   19             4
## 2      1       15             7         1     7556      9   20             3
## 3      1       16            10         1     7562     20   16             9
## 4      1       17             9         1     7565     14    8            12
## 5      1       18            23         1     7554     22    1             1
## 6      1        2             2         1     7563      6    9            10
##   points laps fastestLapSpeed   status        dob driv_nationality
## 1      5   58         215.920 Finished 1982-03-18           German
## 2      6   58         214.706 Finished 1974-07-13          Italian
## 3      0   58         214.640 Finished 1983-01-11           German
## 4      0   57         215.695   +1 Lap 1976-08-27       Australian
## 5     10   58         216.891 Finished 1980-01-19          British
## 6      0   58         216.245 Finished 1977-05-10           German
##        fullname round       date  const_name                           name
## 1    Timo Glock     1 2009-03-29      Toyota Albert Park Grand Prix Circuit
## 2  Jarno Trulli     1 2009-03-29      Toyota Albert Park Grand Prix Circuit
## 3  Adrian Sutil     1 2009-03-29 Force India Albert Park Grand Prix Circuit
## 4   Mark Webber     1 2009-03-29    Red Bull Albert Park Grand Prix Circuit
## 5 Jenson Button     1 2009-03-29       Brawn Albert Park Grand Prix Circuit
## 6 Nick Heidfeld     1 2009-03-29  BMW Sauber Albert Park Grand Prix Circuit
##   const_points const_wins driver_age fastestLap_ms winner wins
## 1           11          0         27         88416      0    1
## 2           11          0         35         88916      0    1
## 3            0          0         26         88943      0    1
## 4            0          0         33         88508      0    1
## 5           18          1         29         88020      1    2
## 6            0          0         32         88283      0    1
```

Our starting dataframe `df` is composed by 25 columns of features that define for each row a Driver performance in a specific Race of a specific year. So for each row:

- 5 columns describe the unique observation: `raceId`, `driverId`, `constructorId`, `circuitId`, `resultId`. They were useful in the initial part of the data processing to merge multiple features from other datasets into this one main dataframe that include all the features that we later use for data analysis and prediction.
- 2 columns define the perfomance of the driver in the specific race, mainly `positionOrder` and the binary `winner`. These are the target variables of the regression and classification methods.
- 11 columns are numerical features used:
  - `number`, number of each driver's car

  - `grid`, starting position for the driver
  - `points`, number of points earned in the race
  - `laps`, number of laps in the race
  - `fastestLapSpeed`
  - `round`, number of the Race in the season
  - `const_points`, number of points earned by the team before this race
  - `const_wins`, number of wins of the team before the race
  - `driver_age`, age of each driver
  - `fastestLap_ms`, fastest lap in the qualification race
  - `wins`, number of wins before the race for the driver

- 5 columns that describe categorical features:

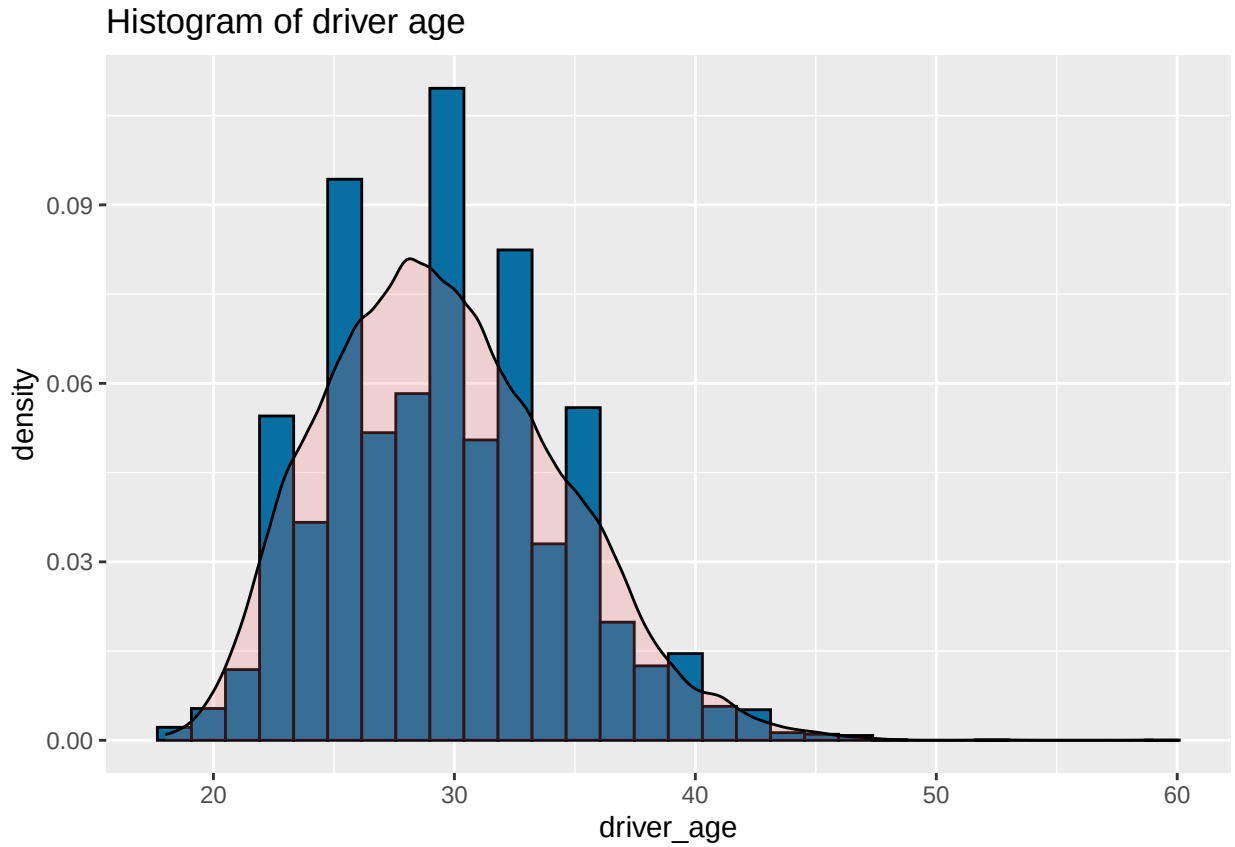
  - `status`, what happened in the race for the driver
  - `driv_nationality`
  - `fullname`, name of the driver
  - `const_name`, name of the team
  - `name`, name of the circuit in which the race takes place

- 2 columns for dates: `dob` is birth date for each driver, `date` is the date of the Race.

EDA and plots.

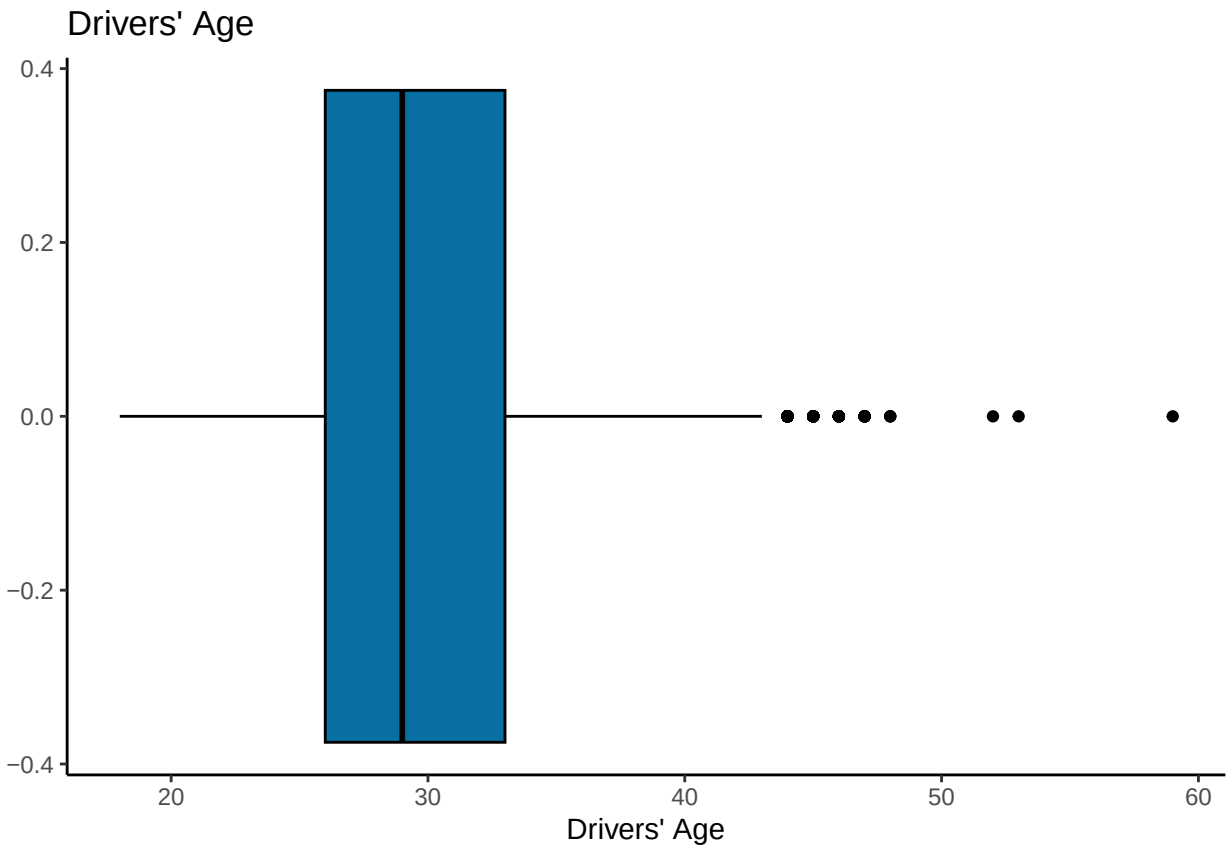
```
ggplot(df, aes(x=driver_age)) +
 geom_histogram(aes(y=after_stat(density)), colour="black", fill= '#076fa2')+
 geom_density(alpha=.2, fill="#FF6666") +
 labs(title = 'Histogram of driver age')
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



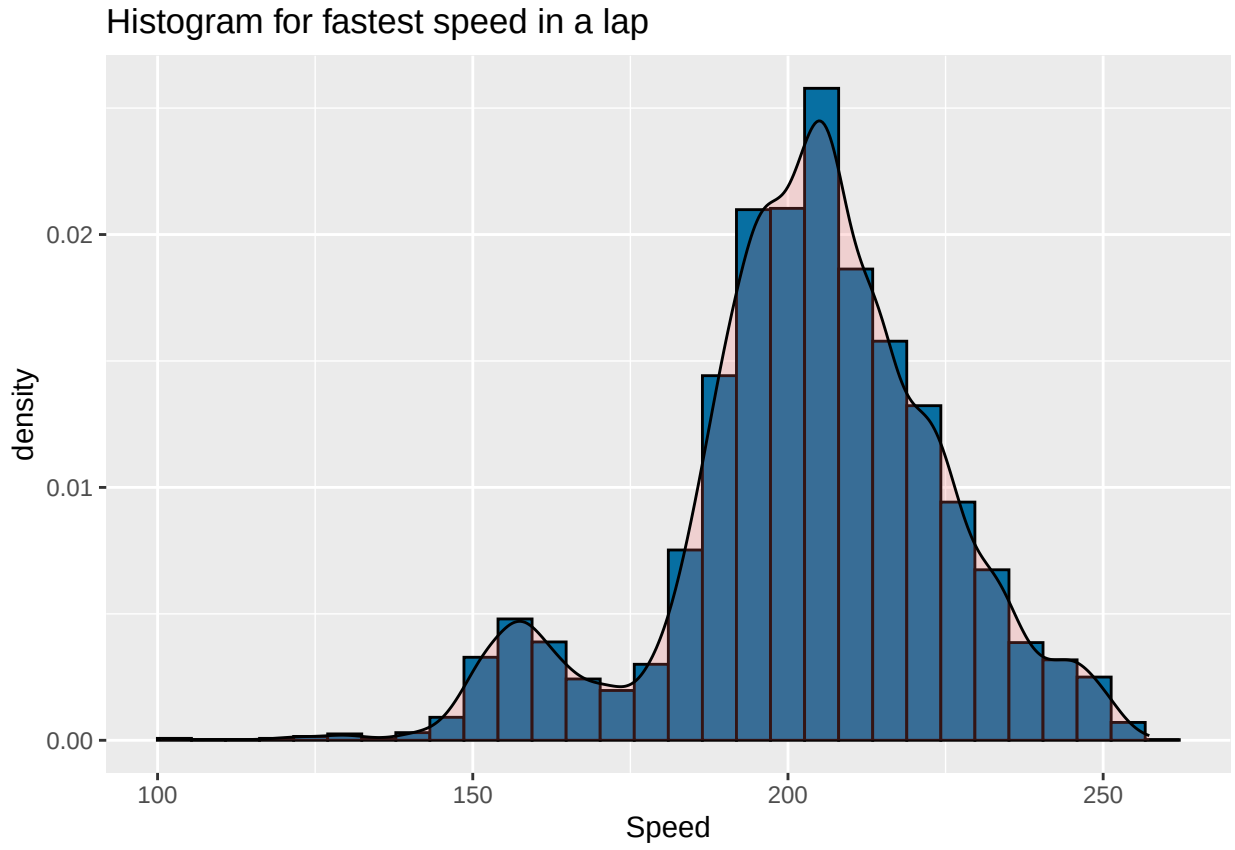
The frequency of the drivers' age from 1950 to 2022 can be traced to a normal distribution, with a peak around 30 years old. Since the column driver_age from the data frame collects also the various age during which the drivers have raced, we can interpret this histogram also like an indicator of the retirement age of a pilot: we have a peak around 30 years old and then as the age increase its frequency decrease, so we have less and less pilot with an higher age.

```
ggplot(df, aes(x=driver_age)) +
  geom_boxplot(fill='#076fa2', color="black")+
  theme_classic() +
  labs(title = "Drivers' Age", x = "Drivers' Age")
```



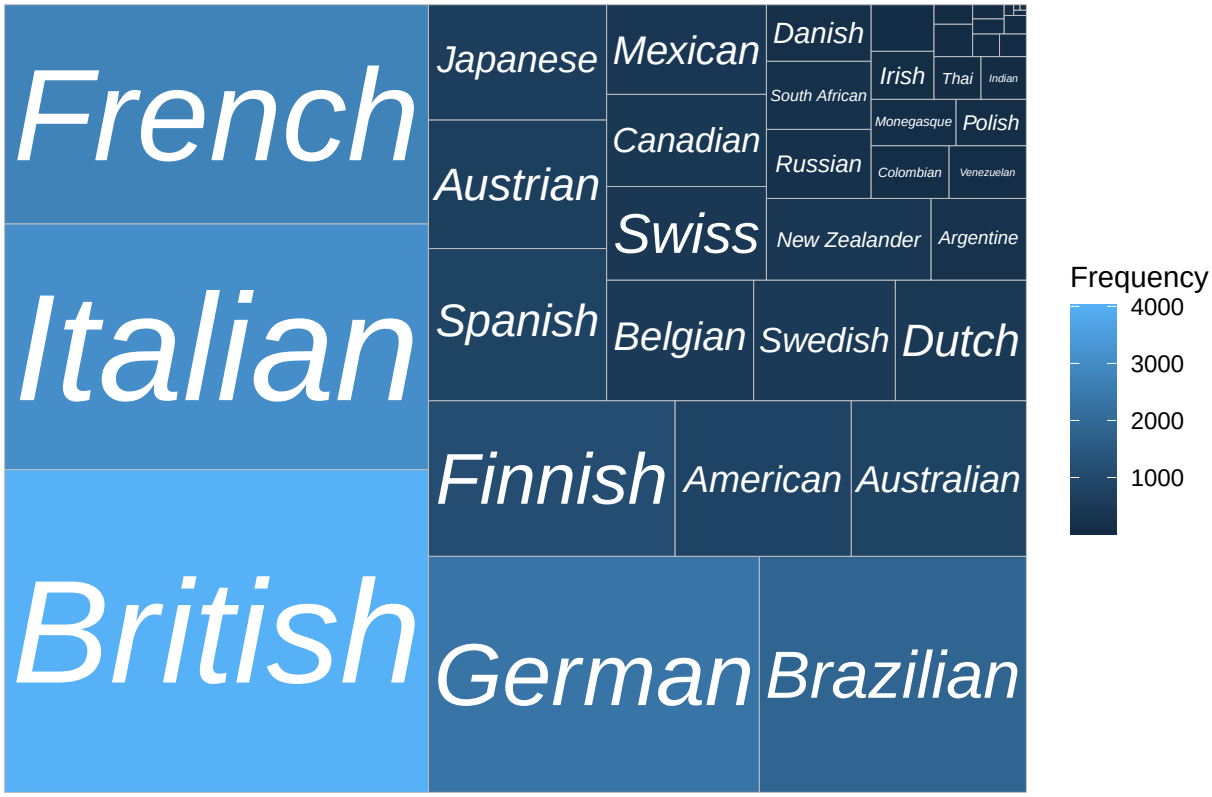

```
ggplot(subset(df, df$fastestLapSpeed > 100), aes(x=fastestLapSpeed)) +
 geom_histogram(aes(y=after_stat(density)), colour="black", fill= '#076fa2')+
 geom_density(alpha=.2, fill="#FF6666") +
 labs(title = 'Histogram for fastest speed in a lap', x = 'Speed')
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



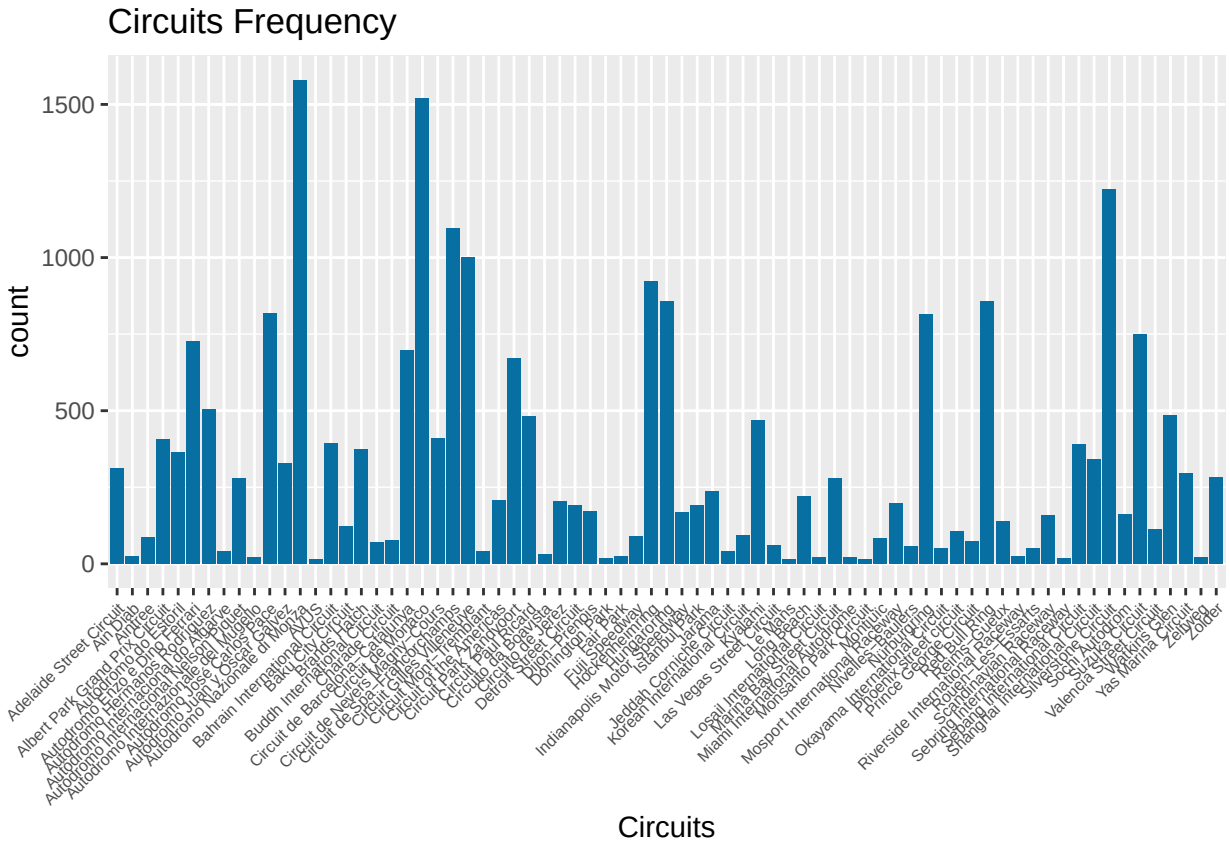

```
nationality = data.frame(table(df$driv_nationality))
nationality$Var1 = as.character(nationality$Var1)
ggplot(nationality, aes(area = Freq, fill = Freq, label = Var1)) +
  geom_treemap() +
  geom_treemap_text(fontface = "italic", colour = "white", place = "centre",
                    grow = TRUE) +
  labs(title = "Treemap of the drivers' nationality", fill = "Frequency")
```

Treemap of the drivers' nationality



This histogram shows the frequency of every circuit over time.

```
ggplot(df, aes(name)) +
  labs(x = 'Circuits') +
  ggtitle('Circuits Frequency') +
  geom_bar(fill="#076fa2", position = position_dodge(0.7)) +
  theme(axis.text.x = element_text(angle = 45, vjust = 1, hjust = 1, size = 6))
```

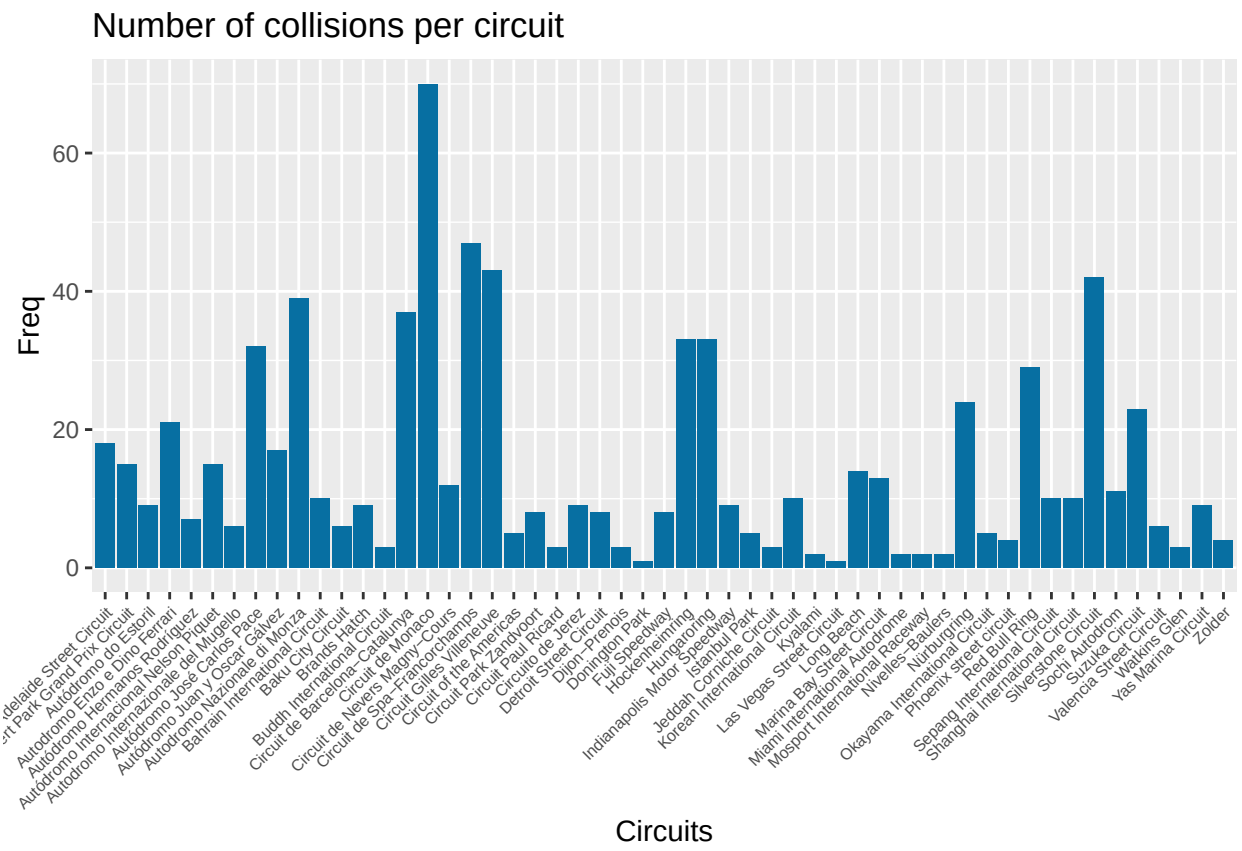


\\

Number of collisions per circuit.

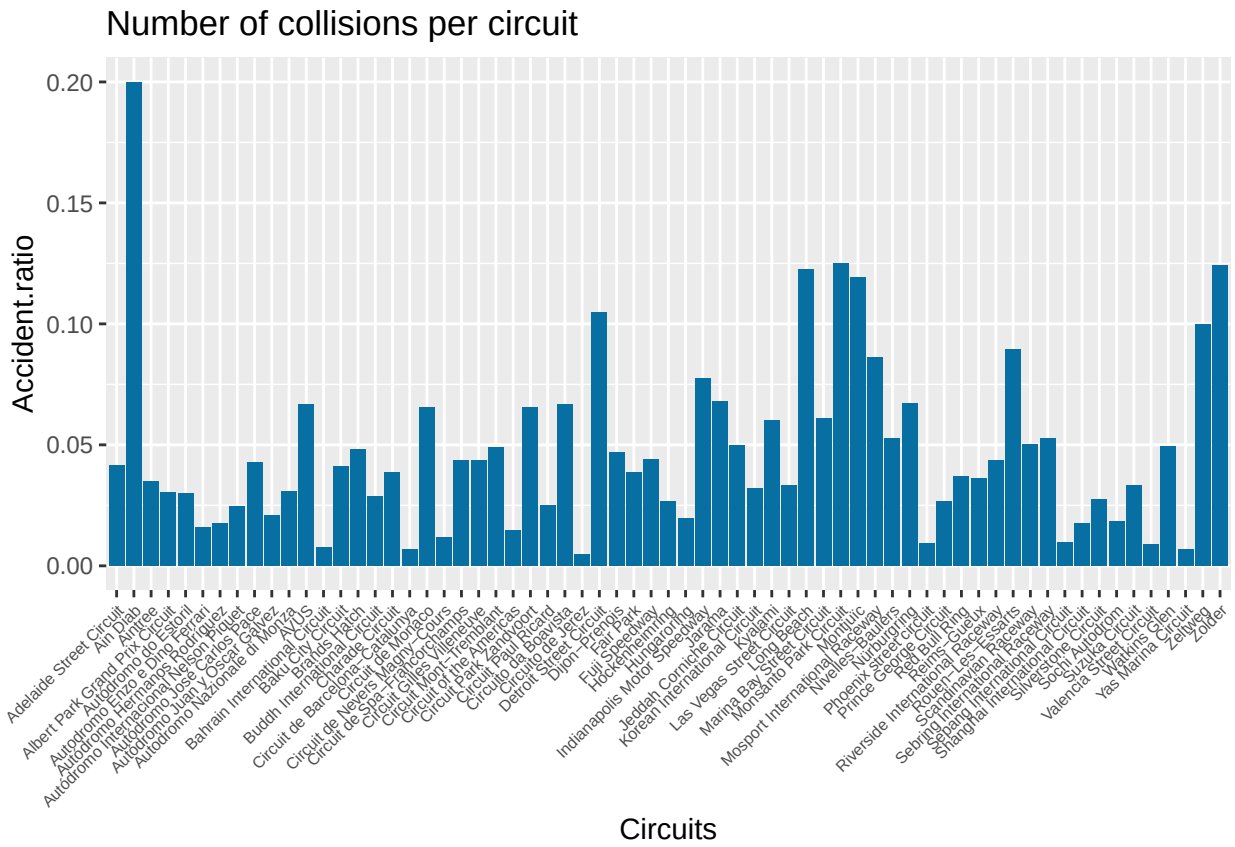
```
#barplot showing number of collisions for each circuit
n_collision <- table(df[df$status == 'Collision', 'name'])
n_collision <- as.data.frame(n_collision)

ggplot(n_collision, aes(x = Var1, y = Freq)) +
  labs(x = 'Circuits') +
  ggtitle('Number of collisions per circuit') +
  geom_bar(fill="#076fa2", position = position_dodge(0.7), stat = 'identity') +
  theme(axis.text.x = element_text(angle = 45, vjust = 1, hjust = 1, size = 6))
```



Accident ratio per circuit.

```
n_accident = read.csv("~/Library/Mobile Documents/com~apple~CloudDocs/UNICATT/Data analysis techniques a
ggplot(n_accident, aes(x = as.character(Accident.Names), y = Accident.ratio)) +
  labs(x = 'Circuits') +
  ggtitle('Number of collisions per circuit') +
  geom_bar(fill="#076fa2", position = position_dodge(0.7), stat = 'identity') +
  theme(axis.text.x = element_text(angle = 45, vjust = 1, hjust = 1, size = 6))
```



Winning driver age overtime

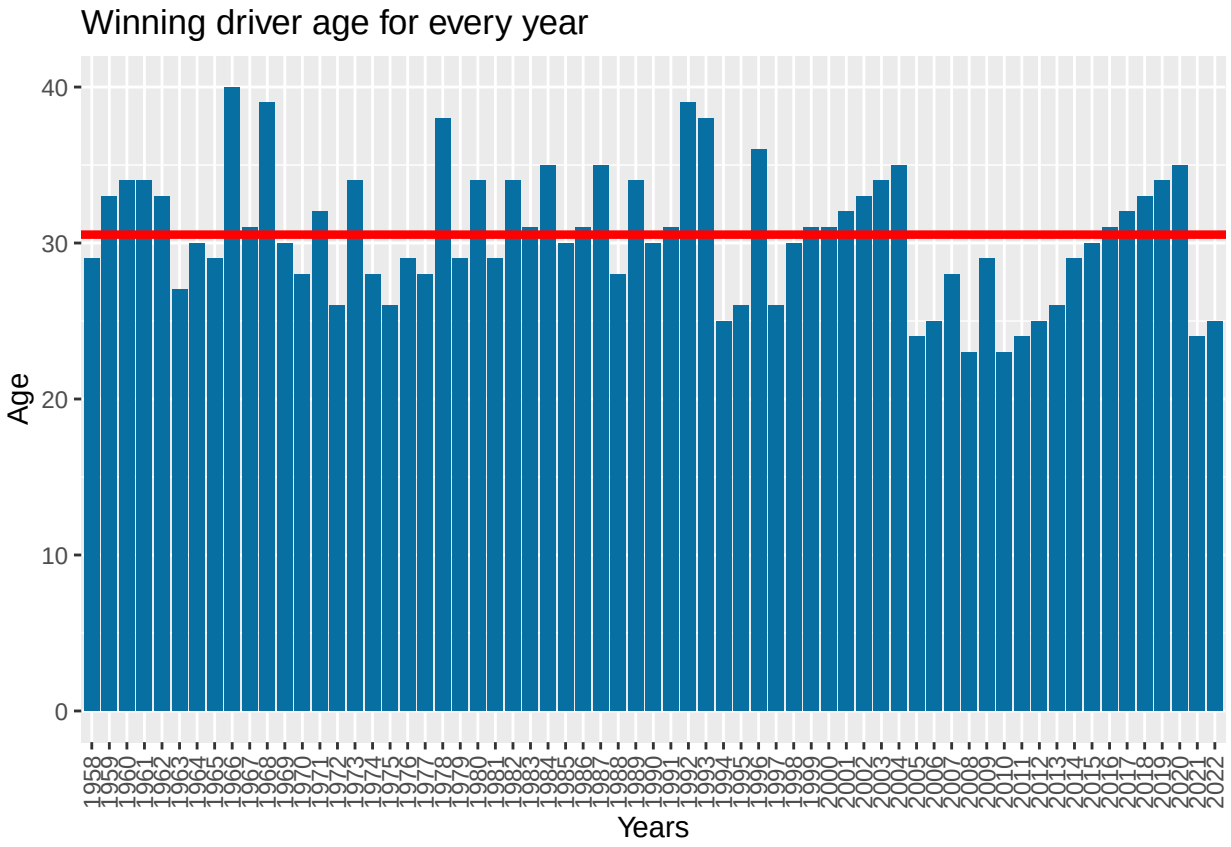
```
df_points = read.csv("~/Library/Mobile Documents/com~apple~CloudDocs/UNICATT/Data analysis techniques ap
```

```
winner_Age = function(){
  df_year_winner = matrix(nrow = 1, ncol=5)
  years = seq(1958, 2022, 1)
  for (y in years){
    df_year = df_points[df_points$year == y, c(2,4,9,10)]
    id_winner = df_year[which.max(df_year$points), ]
    df_year_winner <- rbind(df_year_winner, c(y, id_winner$driverId,
                    id_winner$points, id_winner$fullname, id_winner$driver_age))
  }
  df_year_winner = data.frame(df_year_winner)
  df_year_winner = df_year_winner[-1,]
  return(df_year_winner)
}
df_year_winner = winner_Age()
names(df_year_winner) = c('year', 'driverId', 'final_points', 'fullname', 'age')
df_year_winner$age = as.numeric(df_year_winner$age)
head(df_year_winner)
```

```
##   year driverId final_points      fullname age
## 2 1958      578           42 Mike Hawthorn  29
## 3 1959      356           31  Jack Brabham  33
```

```
## 4 1960      356           43  Jack Brabham  34
## 5 1961      403           34     Phil Hill  34
## 6 1962      289           42   Graham Hill  33
## 7 1963      373           54     Jim Clark  27
```

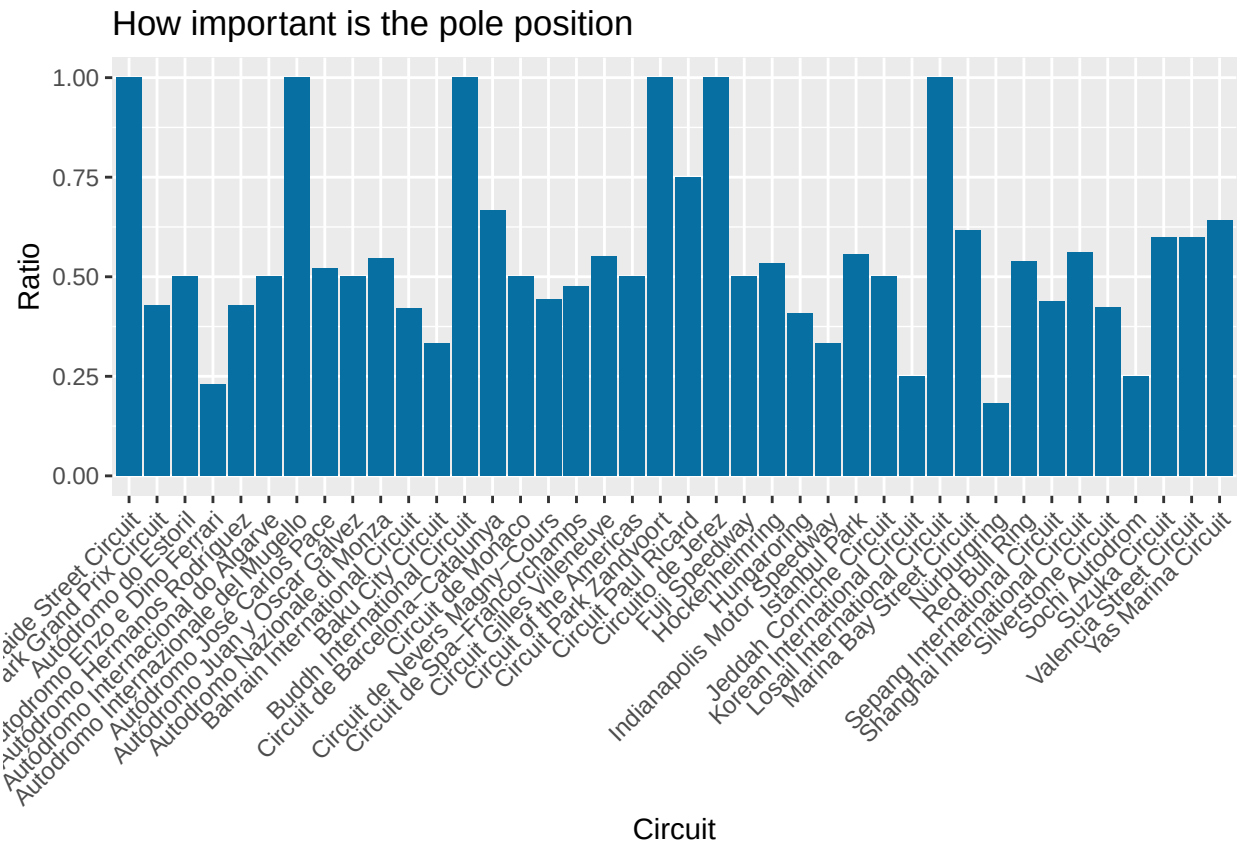
```
#plot for winner's age over time
ggplot(df_year_winner, aes(x = year, y = age)) +
  labs(x = 'Years', y = 'Age') +
  ggtitle('Winning driver age for every year') +
  geom_bar(stat = "identity") +
  geom_col(fill = "#076fa2") +
  theme(axis.text.x = element_text(angle = 90, vjust=0.5, hjust = 1)) +
  geom_abline(slope = 0, intercept = mean(df_year_winner$age),
              color = 'red', linewidth = 1.5)
```



How important is the pole position in each circuit to win the race? Each column is the ratio between how many times a driver have won a race after he has qualified first.

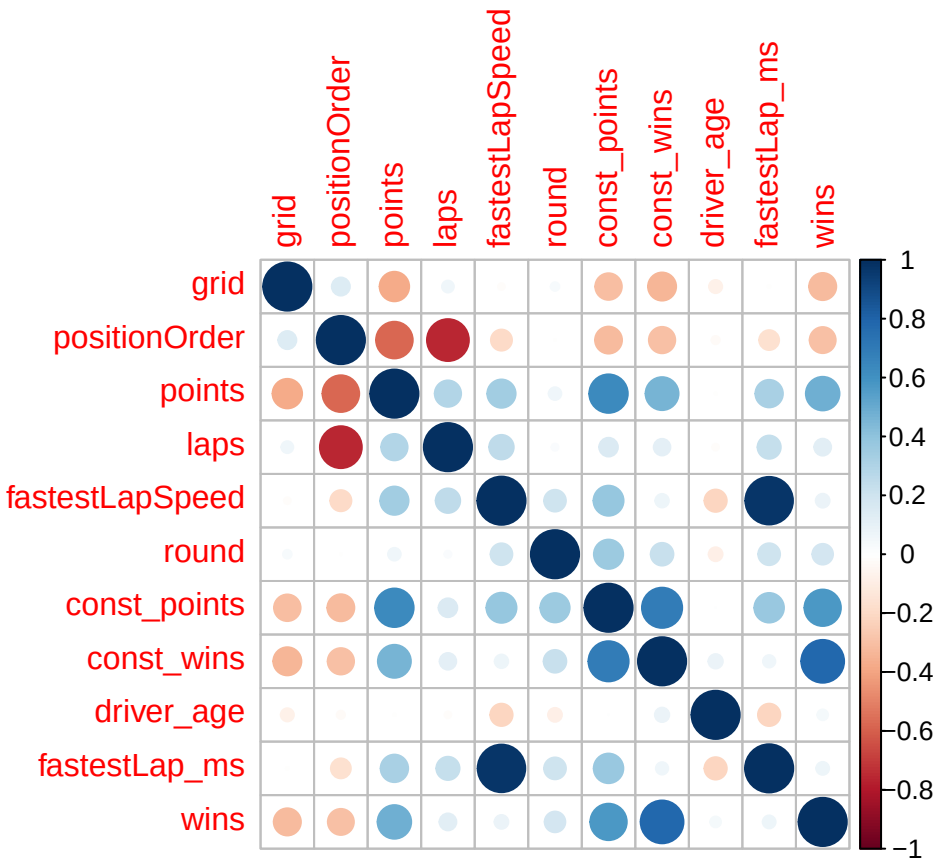
```
pole_ratio = read.csv("~/Library/Mobile Documents/com~apple~CloudDocs/UNICATT/Data analysis techniques a
```

```
ggplot(pole_ratio, aes(x = X1, y = X2)) +
  labs(x = 'Circuit', y = 'Ratio') +
  ggtitle('How important is the pole position') +
  geom_bar(stat = "identity") +
  geom_col(fill = "#076fa2") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



Correlation

```
#CORRELATION MATRIX for numerical features
corr_matrix = cor(df[c(7,8,9,10,11,16,20,21,22,23,25)])
corrplot(corr_matrix)
```



DEFINING AND TRAINING ML MODELS FOR PREDICTION.

```
df$winner = as.factor(df$winner) #converting the column to a factor.
```

Partition the data for train and validation.

For partitioning the data, we decided to use a different approach. Instead of just randomly select rows from the original dataset (we recall that each row is a driver's performance in a specific race) we decided to create a block of rows for each race (with each block containing multiple observation, i.e. all the drivers that attained the race) and randomly select 80% blocks for the training sample and the rest for the testing sample.

We decided to partition the data in this way in order to maintain the integrity of observations for each race. Otherwise, we could have situations in which in the testing sets there is just one driver for a Race, and so his performance would be assessed in the lack of opponents.

```
races_list = unique(df$raceId)
races_train_idx = sample(length(races_list), length(races_list)*0.8)

races_train_list = races_list[races_train_idx]
races_test_list = races_list[-races_train_idx]
```

```
#creating train & test dataframes
df.train = df[df$raceId %in% races_train_list, ]
df.test = df[df$raceId %in% races_test_list, ]
```

Fixing `df.train` and `df.test` for categorical features by converting those columns in factors.

```
#converting TRAIN and TEST categorical features in factors
df.train$status = as.factor(df.train$status)
df.train$driv_nationality = as.factor(df.train$driv_nationality)
df.train$fullname = as.factor(df.train$fullname)
df.train$const_name = as.factor(df.train$const_name)
df.train$name = as.factor(df.train$name)
str(df.train)
```

```
## 'data.frame':    18908 obs. of  25 variables:
##  $ raceId          : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ driverId        : int  10 15 16 17 18 2 20 21 22 3 ...
##  $ constructorId   : int  7 7 10 9 23 2 9 10 23 3 ...
##  $ circuitId       : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ resultId        : int  7557 7556 7562 7565 7554 7563 7566 7564 7555 7559 ...
##  $ number          : int  10 9 20 14 22 6 15 21 23 16 ...
##  $ grid            : int  19 20 16 8 1 9 3 15 2 5 ...
##  $ positionOrder   : int  4 3 9 12 1 10 13 11 2 6 ...
##  $ points          : num  5 6 0 0 10 0 0 0 8 3 ...
##  $ laps            : int  58 58 58 57 58 58 56 58 58 58 ...
##  $ fastestLapSpeed : num  216 215 215 216 217 ...
##  $ status          : Factor w/ 124 levels "+1 Lap","+10 Laps",..: 61 61 61 1 61 61 37 61 61 61 ...
##  $ dob             : chr  "1982-03-18" "1974-07-13" "1983-01-11" "1976-08-27" ...
##  $ driv_nationality: Factor w/ 41 levels "American","American-Italian",..: 19 24 19 5 9 19 19 24 8 19
##  $ fullname        : Factor w/ 586 levels "Adrián Campos",..: 547 264 2 364 277 408 517 194 510 410
##  $ round           : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ date            : chr  "2009-03-29" "2009-03-29" "2009-03-29" "2009-03-29" ...
##  $ const_name      : Factor w/ 156 levels "AGS","Alfa Romeo",..: 149 149 57 125 22 14 125 57 22 154
##  $ name            : Factor w/ 69 levels "Adelaide Street Circuit",..: 4 4 4 4 4 4 4 4 4 4 ...
##  $ const_points    : num  11 11 0 0 18 0 0 0 18 3 ...
##  $ const_wins      : int  0 0 0 0 1 0 0 0 1 0 ...
##  $ driver_age      : int  27 35 26 33 29 32 22 36 37 24 ...
##  $ fastestLap_ms   : num  88416 88916 88943 88508 88020 ...
##  $ winner          : Factor w/ 2 levels "0","1": 1 1 1 1 2 1 1 1 1 1 ...
##  $ wins            : int  1 1 1 1 2 1 1 1 1 1 ...
```

```
df.test$status = as.factor(df.test$status)
df.test$driv_nationality = as.factor(df.test$driv_nationality)
df.test$fullname = as.factor(df.test$fullname)
df.test$const_name = as.factor(df.test$const_name)
df.test$name = as.factor(df.test$name)
str(df.test)
```

```
## 'data.frame':    4785 obs. of  25 variables:
##  $ raceId          : int  1005 1005 1005 1005 1005 1005 1005 1005 1005 1005 ...
##  $ driverId        : int  1 154 20 4 8 807 815 817 822 825 ...
##  $ constructorId   : int  131 210 6 1 6 4 10 9 131 210 ...
```

```
##  $ circuitId       : int  22 22 22 22 22 22 22 22 22 22 ...
##  $ resultId        : int  24103 24110 24108 24116 24107 24121 24109 24106 24104 24122 ...
##  $ number          : int  44 8 5 14 7 27 11 3 77 20 ...
##  $ grid            : int  1 5 8 18 4 16 9 15 2 12 ...
##  $ positionOrder   : int  1 8 6 14 5 19 7 4 2 20 ...
##  $ points          : num  25 4 8 0 10 0 6 12 18 0 ...
##  $ laps            : int  53 53 53 52 53 37 53 53 53 8 ...
##  $ fastestLapSpeed : num  225 221 226 223 222 ...
##  $ status          : Factor w/ 89 levels "+1 Lap","+10 Laps",..: 42 42 42 1 42 37 42 42 42 27 ...
##  $ dob             : chr  "1985-01-07" "1986-04-17" "1987-07-03" "1981-07-29" ...
##  $ driv_nationality: Factor w/ 38 levels "American","American-Italian",..: 8 17 18 34 16 18 27 4 16 
##  $ fullname        : Factor w/ 418 levels "Adrián Campos",..: 231 352 362 103 223 283 366 63 401 222
##  $ round           : int  17 17 17 17 17 17 17 17 17 17 ...
##  $ date            : chr  "2018-10-07" "2018-10-07" "2018-10-07" "2018-10-07" ...
##  $ const_name      : Factor w/ 125 levels "AGS","Alfa Romeo",..: 83 44 37 78 37 99 41 98 83 44 ...
##  $ name            : Factor w/ 53 levels "Adelaide Street Circuit",..: 50 50 50 50 50 50 50 50 50 50
##  $ const_points    : num  538 84 460 58 460 92 43 319 538 84 ...
##  $ const_wins      : int  9 0 5 0 5 0 0 3 9 0 ...
##  $ driver_age      : int  33 32 31 37 39 31 28 29 29 26 ...
##  $ fastestLap_ms   : num  92785 94786 92318 93943 94223 ...
##  $ winner          : Factor w/ 2 levels "0","1": 2 1 1 1 1 1 1 1 1 1 ...
##  $ wins            : int  10 1 6 1 1 1 1 3 1 1 ...
```

Fixing FACTOR LEVELS in the testing dataframe.

When partitioning the data, categories for nominal features are shuffled between df.train and df.test. This lead to a situation in which categories in df.test could not be present in df.train and so classification models trained on df.test would not consider other categories that are only present in df.test.

With the following lines of code we overcome this problem by setting factor levels of all categorical columns of df.test to be exactly the same as df.train. Otherwise, we would encounter problems in the classifications method when predicting the new data from df.test using the model trained on df.train.

```
levels(df.test$status) <- levels(df.train$status)
levels(df.test$driv_nationality) <- levels(df.train$driv_nationality)
levels(df.test$fullname) <- levels(df.train$fullname)
levels(df.test$const_name) <- levels(df.train$const_name)
levels(df.test$name) <- levels(df.train$name)
```

SCORING FUNCTIONS

The two scoring functions we created, simply calculate the accuracy for the prediction of the winner of each race.

For the regressions, the scoring functions sort the prediction of `positionOrder` for each race (by using `raceId`) and take the smallest value for the prediction (since in the `positionsOrder` the winner has position 1, hence the smallest position). If the same driver (a row of df.test) has the real value of `positionOrder` equal to 1, then the prediction is correct and the score is increased by 1. After scrolling through all races that are contained in df.test, the `accuracy_score` is defined as the ratio between score and the number of races.

For the classification, the scoring function works similarly but there is a difference. We are using the classifications models to predict `winner` defined in {0,1} but the models in action are going to assign 1 to multiple drivers in the same race (but we can have just one winner in a race). So instead we look for the greatest probability among the driver of a race to have the winner feature equal to 1. The driver with the

greatest probability of having the `winner` feature equal to 1 would be the driver with the greatest probability of winning the race and hence he is the winner our function pick. The definition of score works in the same way as in the scoring function for regressions.

Defining scoring function for Regression.

```
score_regression <- function(test, prediction){

  df.score.regres = data.frame(test, prediction)
  colnames(df.score.regres)[c(26)] = c('prediction')

  racelist = unique(df.score.regres$raceId)

  score = 0
  for (i in 1:length(racelist)){
    race = df.score.regres[df.score.regres$raceId %in% racelist[i], ]
    pred_winner_idx = which.min(race$prediction)
    if (race$positionOrder[pred_winner_idx] == 1){
      score = score + 1
    }
  }

  score_ratio = score/length(racelist)
  return(score_ratio)
}
```

Defining scoring function for Classification.

```
score_classification <- function(test, prediction){

  df.score.class = data.frame(test, prediction)
  colnames(df.score.class)[c(26,27)] = c('pred_0', 'pred_1')

  racelist = unique(df.score.class$raceId)

  score = 0
  for (i in 1:length(racelist)){
    race = df.score.class[df.score.class$raceId %in% racelist[i], ]
    pred_winner_idx = which.max(race$pred_1)
    if (race$positionOrder[pred_winner_idx] == 1){
      score = score + 1
    }
  }

  score_ratio = score/length(racelist)
  return(score_ratio)
}
```

REGRESSION

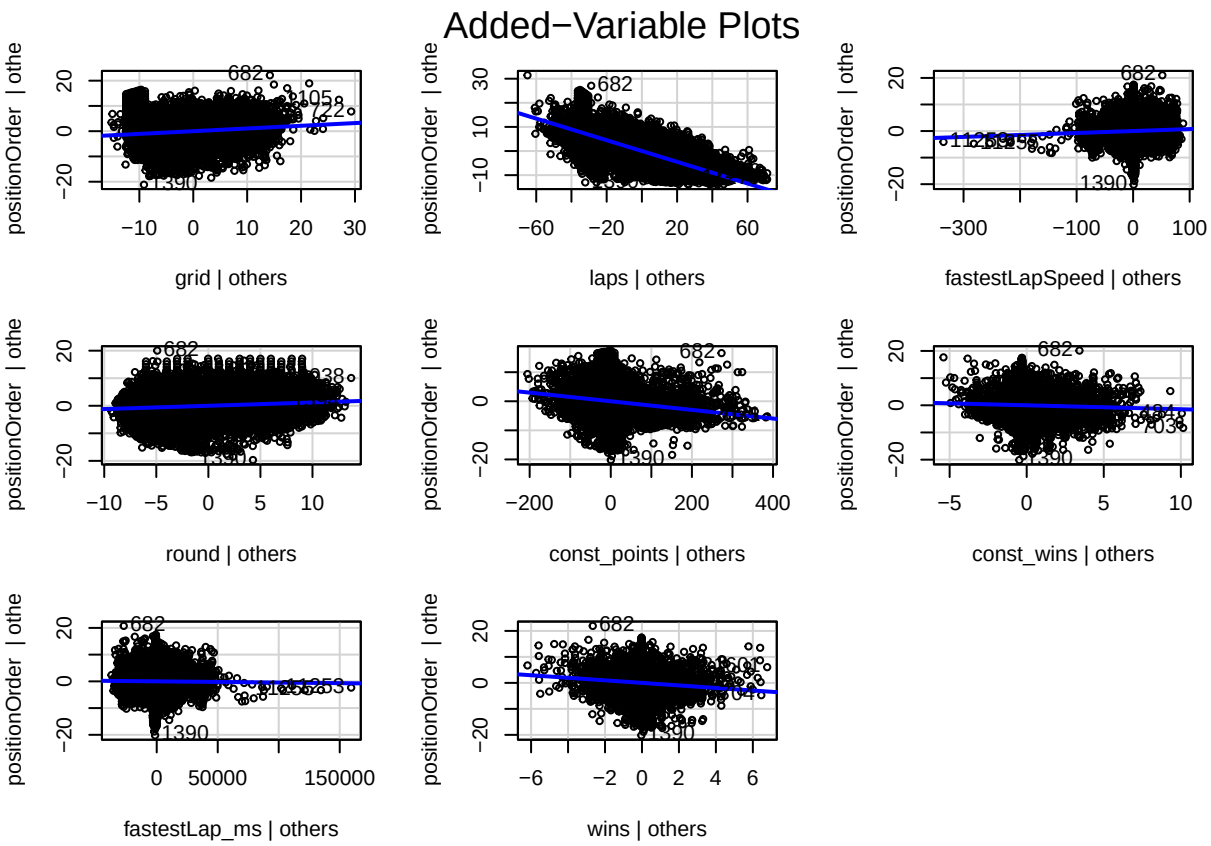
Training and testing linear regression models. For the regression model we decided to use as explanatory variables most of the numerical features as: grid, laps, fastestLapSpeed, round, const_points, const_wins, fastestLap_ms and wins that presumably would help predict and explain the final position of

the driver. Some variables were excluced as points, since its high correlation with the final position (higher number of points means a higher position, were 1 is the highest position).

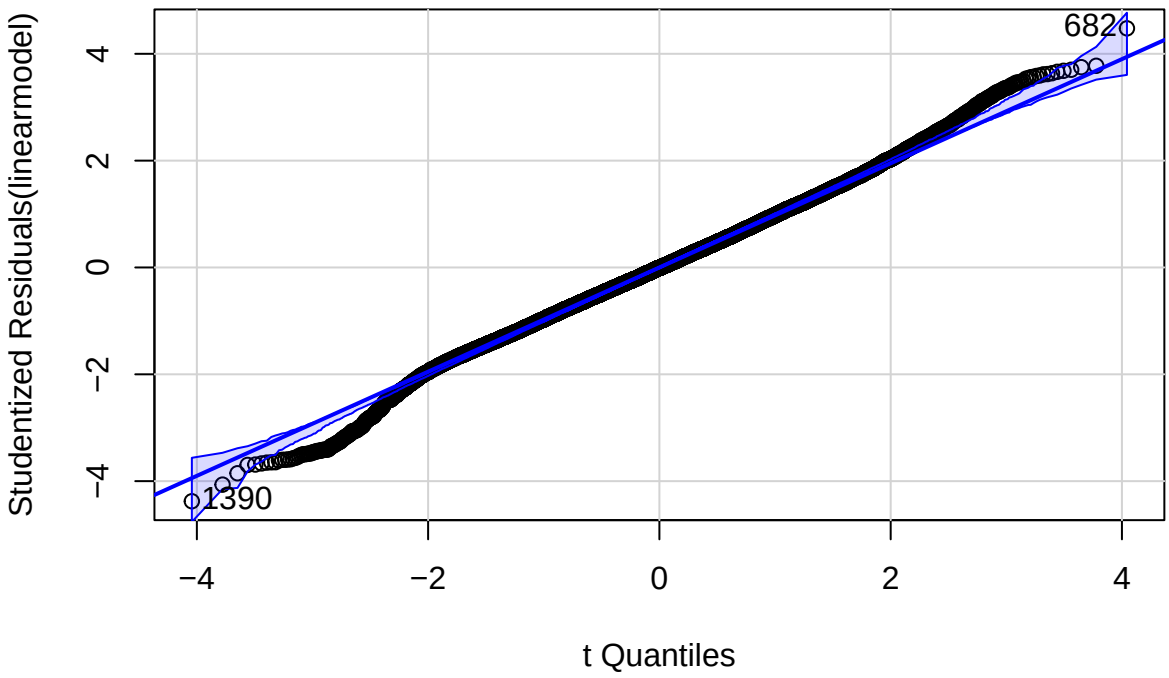
```
linearmodel = lm(positionOrder ~ grid + laps + fastestLapSpeed + round +
                   const_points + const_wins + fastestLap_ms + wins, data = df.train)
summary(linearmodel)
```

```
##
## Call:
## lm(formula = positionOrder ~ grid + laps + fastestLapSpeed +
##     round + const_points + const_wins + fastestLap_ms + wins,
##     data = df.train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -20.2098  -3.0047  -0.0309   2.9698  20.6172
##
## Coefficients:
##                   Estimate Std. Error  t value Pr(>|t|)
## (Intercept)      2.157e+01  1.130e-01  190.897  < 2e-16 ***
## grid             1.071e-01  5.160e-03   20.749  < 2e-16 ***
## laps            -2.242e-01  1.391e-03 -161.134  < 2e-16 ***
## fastestLapSpeed  7.377e-03  1.628e-03    4.531 5.91e-06 ***
## round            1.194e-01  7.442e-03   16.045  < 2e-16 ***
## const_points    -1.485e-02  7.077e-04  -20.976  < 2e-16 ***
## const_wins      -1.444e-01  3.392e-02   -4.257 2.08e-05 ***
## fastestLap_ms   -4.519e-06  3.593e-06   -1.258    0.209
## wins            -4.884e-01  4.828e-02  -10.116  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.621 on 18899 degrees of freedom
## Multiple R-squared:  0.6402, Adjusted R-squared:  0.6401
## F-statistic:  4204 on 8 and 18899 DF,  p-value: < 2.2e-16
```

```
avPlots(linearmodel)
```



As we can see the linear model obtain an R-squared of 65% that confirms a good capacity of the model to represent variability in the original data. Also, the majority of variables are strongly significant.




```
##  682 1390
##  582 1070
```

In order to test the significance of the whole model, we should first look at the normality of the residuals. From the qqplot presented above, it's clear how the residuals follow aproximately a normal distribution (they lie on a straight line as normal residuals should). Looking at the F-test value of the model, it's clear how the really low value suggests a strong significance of the whole regression. Hence we can say the model does a better job in describing the dataset than just the mean.

We can now try to run a prediction and recall 'score_regression' function to see the results of the prediction.

```
predict_lm = predict(linearmodel, df.test)
score_regression(df.test, predict_lm)
```

```
## [1] 0.6403941
```

Regression tree Decision trees are another technique that we have used both in regression and in classification. The model used in regression take as explanatory the same variables in the regression:

```
predict_rt = predict(tree(positionOrder ~ grid + number + laps + fastestLapSpeed
                          + round + const_points + const_wins + driver_age
                          + fastestLap_ms + wins, df.train), newdata = df.test)
score_regression(df.test, predict_rt) #47%
```

```
## [1] 0.4581281
```

CLASSIFICATION

Using the features class `winner` that we have created, we want to predict the probability of class=1 (winner) or class=0 (not winner). Then we are going to sort the probabilities and pick the greater probability of class=1, hence the driver with the greatest probability of being a winner.

For the classification methods, we decided to remove those categorical features as `status`, `fullname` and `const_name` that when converted in factors where causing problems in the application of the techniques, or drastically reduced performances due to their dimension (all of those 3 factors presented more than 100 levels that were not accepted by the functions or ruined the prediction accuracy).

```
df.nb <- naiveBayes(winner ~ . -number -positionOrder -resultId -points -status
                    -fullname -const_name, data = df.train)
prediction_nb <- predict(df.nb, newdata = df.test, type = 'raw')
score_classification(df.test, prediction_nb)
```

BAYES CLASSIFIER

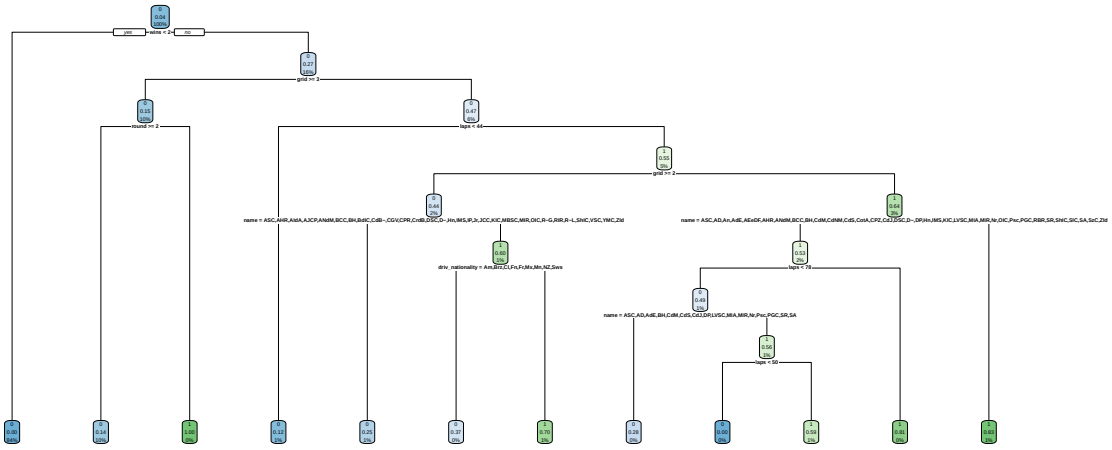
```
## [1] 0.5615764
```

```
df.dt = rpart(winner ~ . -number -positionOrder -resultId -points -status
              -fullname -const_name -dob -date, data = df.train, method="class")
prediction_dt = predict(df.dt, newdata = df.test, method="prob")
score_classification(df.test, prediction_dt) #58%
```

DECISION TREES

```
## [1] 0.5320197
```

```
rpart.plot(df.dt,faclen = 2)
```



```
df.rf <- randomForest(winner ~ . -number -points -positionOrder -resultId
                      -const_name -name -fullname -status, data = df.train,
                      ntree = 200)
prediction.rf <- predict(df.rf, df.test, type = 'prob')
prediction.rf[is.na(prediction.rf)] <- 0
score_classification(df.test, prediction.rf)
```

RANDOM FOREST

```
## [1] 0.5812808
```

```
df.lsvm = svm(winner ~ ., data = df.train[, -c(5,6,8,9,12,13,15,17)],
              kernel = 'linear', fitted = FALSE, probability = TRUE)
prediction_svm <- predict(df.lsvm, newdata = df.test[,-c(5,6,8,9,12,13,15,17)],
                          fitted = FALSE, probability = TRUE)
SVM_class = data.frame(attributes(prediction_svm)$probabilities)
score_classification(df.test, SVM_class)
```

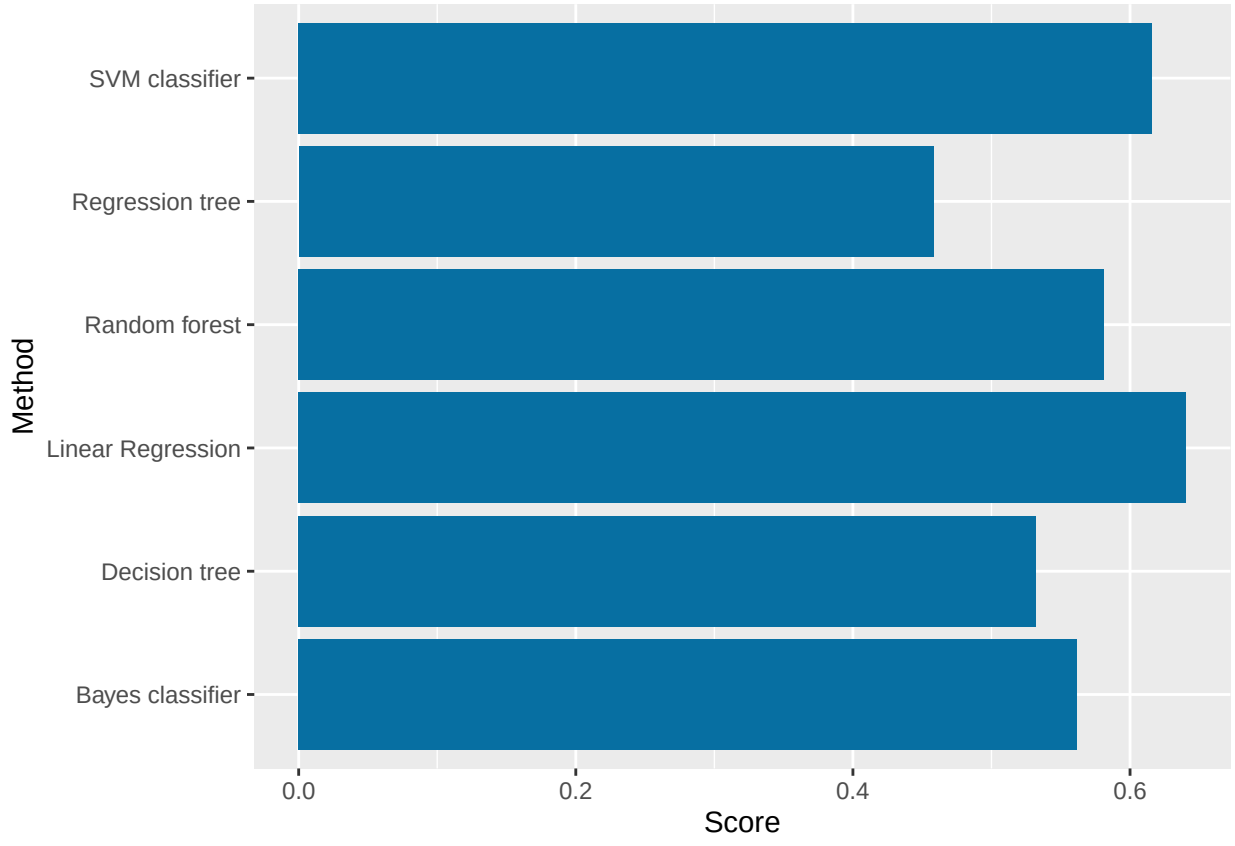
SVM classifier

```
## [1] 0.6157635
```

Visualize final prediction results

```
accuracy_results = data.frame(c(score_regression(df.test, predict_lm),
                                score_regression(df.test, predict_rt),
                                score_classification(df.test, prediction_nb),
                                score_classification(df.test, prediction_dt),
                                score_classification(df.test, prediction.rf),
                                score_classification(df.test, SVM_class)),
                              c('Linear Regression', 'Regression tree',
                                'Bayes classifier','Decision tree','Random forest',
                                'SVM classifier'))
colnames(accuracy_results) = c('score', 'method')

ggplot(accuracy_results, aes(x = score, y = method)) +
  labs(x = 'Score', y = 'Method') +
  geom_bar(stat = "identity") +
  geom_col(fill = "#076fa2")
```



CONCLUSIONS

Our initial goal wasn't to analyze deeply the data that characterize a Formula 1 Race. Our goal was to rather try to predict the winner of a Race with the least possible number of numerical and categorical features. If we look at the last chart with all the accuracy score computed from both regression and classification

models, we can see quite good results: all of the models predicted with a accuracy of about 60%, apart from regression tree and Bayes classifier. The first model is maybe too weak for such a big dataset like this one and for the second the explanation could be similar: the Bayes classifier works like a benchmark for other classifiers, so it is only an indicator of performance and maybe it is too weak too.