Predicting Race Outcomes in Formula 1

Tyler Campbell

11/20/2019

## Project Introduction

This report contains findings and steps-taken during Stage 2 (of 3) of a project in **MIS 545 - Data Mining for Business Intelligence**, taught by Dr. Bin Zhang at The University of Arizona.

The requirements for this stage were for team members to independently test different data subsets or algorithms, or to try different approaches altogether (as compared to Stage 1). In Stage 3, the team will regroup and build off eachother’s analyses.

## Problem Description

Apart from the extreme financial demands of Formula 1, one of the great challenges of the sport is for team principals and strategists to decide which of their two drivers to prioritize in a race. Sometimes it is obvious which driver is faster (thus deserving preferential treatment) and other times it is not (which often leads to conflict). Being able to better predict race outcomes would allow for more effective team strategizing.

## Team vs Individual Approach

In Stage 1 of the project, the team set out to predict order of podium finishers (i.e. top 3) in a given race based off of individual lap data from 2011 to 2017. We incorporated C5.0, SVM, and Naive Bayes classifer algorithms. The findings produced fair accuracies, but their usefulness was limited due to only considering outcomes of 3 drivers and only building off data from a 6 year window. Having the granularity of lap times as opposed to race results also resulted in problematic redundancies.

Given these shortfalls, for Stage 2 I chose to re-structure the dataset to include race results of all drivers dating back to 1950. The process required extensive cleaning, but ultimately allowed me to predict whether or not a racer would finish ‘in the points’ (i.e. top 10) or ‘out of the points’, which seemed like a more useful finding. Once the cleaning had been performed, I went back and incorporated an additional dataset to obtain driver home town geocordinates. In the end, the algorithms that I tested were as follows:

* **K-Means** - on where drivers are from geographicaly (values were assigned back to the dataset)
* **Naive Bayes** - to predict whether or not a racer finished ‘in the points’

## Dataset Descriptions

Data for this project was obtained from **Kaggle** (<https://www.kaggle.com/cjgdev/formula-1-race-data-19502017>). It was originally sourced from **Ergast Developer API** (<https://ergast.com/mrd/>) at the conclusion of the 2017 season. Data on ergast is gathered and published to the public domain by Chris Newell.

The raw data is comprised of the below 13 files, where code chunks contain table dimensions and summary statistics. The ‘laptimes’ table is the largest file, containing over 400,000 records. Due to my decision to change the grain to race outcome per driver (as opposed to lap times per driver, per race), the ‘results’ file dictated the overall number of observations at just under 30,000.

**circuits:** every circuit name, location, and wiki page url

## [1] 73 9

## circuitId circuitRef name location   
## Min. : 1 Length:73 Length:73 Length:73   
## 1st Qu.:19 Class :character Class :character Class :character   
## Median :37 Mode :character Mode :character Mode :character   
## Mean :37   
## 3rd Qu.:55   
## Max. :73   
##   
## country lat lng alt   
## Length:73 Min. :-37.85 Min. :-118.189 Min. :10   
## Class :character 1st Qu.: 33.58 1st Qu.: -9.394 1st Qu.:10   
## Mode :character Median : 41.37 Median : 3.931 Median :10   
## Mean : 33.87 Mean : 1.723 Mean :10   
## 3rd Qu.: 47.22 3rd Qu.: 14.765 3rd Qu.:10   
## Max. : 57.27 Max. : 144.968 Max. :10   
## NA's :72   
## url   
## Length:73   
## Class :character   
## Mode :character   
##

**constructorResults:** aggregated constructor points earned per race

## [1] 11142 5

## constructorResultsId raceId constructorId points   
## Min. : 1 Min. : 1.0 Min. : 1.00 Min. : 0.000   
## 1st Qu.: 2786 1st Qu.:259.2 1st Qu.: 6.00 1st Qu.: 0.000   
## Median : 5572 Median :440.0 Median : 23.00 Median : 0.000   
## Mean : 7364 Mean :456.4 Mean : 41.25 Mean : 3.193   
## 3rd Qu.:12823 3rd Qu.:628.0 3rd Qu.: 50.00 3rd Qu.: 4.000   
## Max. :15639 Max. :988.0 Max. :210.00 Max. :66.000   
## status   
## Length:11142   
## Class :character   
## Mode :character   
##

* ‘D’ in ‘status’ column represents ‘Disqualified’ due to Spygate scandal in 2007 season (<https://en.wikipedia.org/wiki/2007_Formula_One_espionage_controversy>)

**constructors:** every constructor name, nationality, and wiki page url

## [1] 208 6

## constructorId constructorRef name nationality   
## Min. : 1.00 Length:208 Length:208 Length:208   
## 1st Qu.: 53.75 Class :character Class :character Class :character   
## Median :105.50 Mode :character Mode :character Mode :character   
## Mean :105.51   
## 3rd Qu.:157.25   
## Max. :210.00   
## url X   
## Length:208 Mode:logical   
## Class :character NA's:208   
## Mode :character   
##

* constructors are teams (e.g. Ferrari, Williams, Red Bull, etc.)
* in modern racing era there are 10 constructors with 2 drivers each

**constructorStandings:** running/accumulated ‘points’ and ‘wins’ for constructors in a given season

## [1] 11896 8

## constructorStandingsId raceId constructorId points   
## Min. : 1 Min. : 1.0 Min. : 1.00 Min. : 0.00   
## 1st Qu.: 8245 1st Qu.:274.0 1st Qu.: 6.00 1st Qu.: 0.00   
## Median :17760 Median :463.0 Median : 25.00 Median : 6.00   
## Mean :15201 Mean :469.1 Mean : 45.67 Mean : 27.14   
## 3rd Qu.:23302 3rd Qu.:658.0 3rd Qu.: 56.00 3rd Qu.: 25.00   
## Max. :26932 Max. :988.0 Max. :210.00 Max. :765.00   
## position positionText wins X   
## Min. : 1.000 Length:11896 Min. : 0.0000 Mode:logical   
## 1st Qu.: 4.000 Class :character 1st Qu.: 0.0000 NA's:11896   
## Median : 7.000 Mode :character Median : 0.0000   
## Mean : 7.441 Mean : 0.6443   
## 3rd Qu.:10.000 3rd Qu.: 0.0000   
## Max. :22.000 Max. :19.0000

**drivers:** every driver name, number, dob, nationality, and wiki page url

## [1] 842 9

## driverId driverRef number code   
## Min. : 1.0 Length:842 Min. : 2.00 Length:842   
## 1st Qu.:211.2 Class :character 1st Qu.:10.25 Class :character   
## Median :421.5 Mode :character Median :21.50 Mode :character   
## Mean :421.5 Mean :30.50   
## 3rd Qu.:631.8 3rd Qu.:38.25   
## Max. :843.0 Max. :99.00   
## NA's :804   
## forename surname dob   
## Length:842 Length:842 Length:842   
## Class :character Class :character Class :character   
## Mode :character Mode :character Mode :character   
##   
##   
## nationality url   
## Length:842 Length:842   
## Class :character Class :character   
## Mode :character Mode :character   
##

**driverStandings:** accumulated driver points and wins for a given season

## [1] 31726 7

## driverStandingsId raceId driverId points   
## Min. : 1 Min. : 1.0 Min. : 1.0 Min. : 0.00   
## 1st Qu.:18503 1st Qu.:329.0 1st Qu.: 84.0 1st Qu.: 0.00   
## Median :47758 Median :566.0 Median :205.0 Median : 0.00   
## Mean :39545 Mean :537.9 Mean :280.2 Mean : 10.62   
## 3rd Qu.:56742 3rd Qu.:769.0 3rd Qu.:437.0 3rd Qu.: 8.00   
## Max. :68608 Max. :988.0 Max. :843.0 Max. :397.00   
## position positionText wins   
## Min. : 1.00 Length:31726 Min. : 0.000   
## 1st Qu.: 9.00 Class :character 1st Qu.: 0.000   
## Median : 17.00 Mode :character Median : 0.000   
## Mean : 20.57 Mean : 0.252   
## 3rd Qu.: 27.00 3rd Qu.: 0.000   
## Max. :108.00 Max. :13.000

**lapTimes:** lap time and position for each driver in each lap of each race

## [1] 426633 6

## raceId driverId lap position   
## Min. : 1.0 Min. : 1.0 Min. : 1.00 Min. : 1.000   
## 1st Qu.:100.0 1st Qu.: 14.0 1st Qu.:14.00 1st Qu.: 5.000   
## Median :205.0 Median : 26.0 Median :29.00 Median : 9.000   
## Mean :423.1 Mean :186.5 Mean :29.83 Mean : 9.647   
## 3rd Qu.:881.0 3rd Qu.: 71.0 3rd Qu.:44.00 3rd Qu.:14.000   
## Max. :988.0 Max. :843.0 Max. :78.00 Max. :24.000   
## time milliseconds   
## Length:426633 Min. : 67411   
## Class :character 1st Qu.: 82382   
## Mode :character Median : 90800   
## Mean : 95802   
## 3rd Qu.: 102738   
## Max. :7507547

* file is missing lap times from at least races 400-800 (by raceID)
* lap 25 of raceId 847 (2011 Canadian Grand Prix had “torrential rains” that caused a single lap to clock in at over 2 hrs)

**pitStops:** stop number and stop duration/milliseconds of each pitstop at a given time of day on a given lap by a given driver

## [1] 6251 7

## raceId driverId stop lap   
## Min. :841.0 Min. : 1.0 Min. :1.000 Min. : 1.00   
## 1st Qu.:868.0 1st Qu.: 13.0 1st Qu.:1.000 1st Qu.:13.00   
## Median :902.0 Median :807.0 Median :2.000 Median :24.00   
## Mean :907.9 Mean :429.5 Mean :1.836 Mean :24.81   
## 3rd Qu.:950.0 3rd Qu.:821.0 3rd Qu.:2.000 3rd Qu.:35.00   
## Max. :988.0 Max. :843.0 Max. :6.000 Max. :74.00   
## time duration milliseconds   
## Length:6251 Length:6251 Min. : 12897   
## Class :character Class :character 1st Qu.: 21768   
## Mode :character Mode :character Median : 23340   
## Mean : 47744   
## 3rd Qu.: 25547   
## Max. :2011266

* file excludes races before 2011 and after 2017

**qualifying:** qualifying times (and final qualifying position) for each driver of each race

## [1] 7516 9

## qualifyId raceId driverId constructorId   
## Min. : 1 Min. : 1.0 Min. : 1.0 Min. : 1.00   
## 1st Qu.:1880 1st Qu.: 91.0 1st Qu.: 14.0 1st Qu.: 4.00   
## Median :3760 Median :259.0 Median : 30.0 Median : 9.00   
## Mean :3763 Mean :449.8 Mean :203.8 Mean : 34.76   
## 3rd Qu.:5639 3rd Qu.:887.0 3rd Qu.:111.0 3rd Qu.: 19.00   
## Max. :7539 Max. :988.0 Max. :843.0 Max. :210.00   
## number position q1 q2   
## Min. : 0.00 Min. : 1.00 Length:7516 Length:7516   
## 1st Qu.: 7.00 1st Qu.: 6.00 Class :character Class :character   
## Median :12.00 Median :11.00 Mode :character Mode :character   
## Mean :15.29 Mean :11.47   
## 3rd Qu.:20.00 3rd Qu.:17.00   
## Max. :99.00 Max. :28.00   
## q3   
## Length:7516   
## Class :character   
## Mode :character   
##

* the process of qualifying determines grid position at start of race
* the slowest drivers (e.g. bottom 5) get knocked out after each round of qualifying
* ‘position’ should be the same as grid in results file

**races:** race name, date, and time for each seasons, and wiki page url

## [1] 997 8

## raceId year round circuitId   
## Min. : 1 Min. :1950 Min. : 1.000 Min. : 1.00   
## 1st Qu.: 250 1st Qu.:1974 1st Qu.: 4.000 1st Qu.: 9.00   
## Median : 499 Median :1990 Median : 8.000 Median :18.00   
## Mean : 500 Mean :1989 Mean : 8.234 Mean :21.76   
## 3rd Qu.: 748 3rd Qu.:2005 3rd Qu.:12.000 3rd Qu.:30.00   
## Max. :1009 Max. :2018 Max. :21.000 Max. :73.00   
## name date time   
## Length:997 Length:997 Length:997   
## Class :character Class :character Class :character   
## Mode :character Mode :character Mode :character   
##   
## url   
## Length:997   
## Class :character   
## Mode :character   
##

* race time is empty from 1950 to 2005
* remove ‘name’ which will be redundant after merge (circuits table contains ref)

**results:** results of every race (*critical file containing dependent variables*)

## [1] 23777 18

## resultId raceId driverId constructorId   
## Min. : 1 Min. : 1.0 Min. : 1.0 Min. : 1.00   
## 1st Qu.: 5945 1st Qu.:273.0 1st Qu.: 55.0 1st Qu.: 6.00   
## Median :11889 Median :478.0 Median :154.0 Median : 25.00   
## Mean :11889 Mean :487.2 Mean :226.5 Mean : 46.28   
## 3rd Qu.:17833 3rd Qu.:718.0 3rd Qu.:314.0 3rd Qu.: 57.00   
## Max. :23781 Max. :988.0 Max. :843.0 Max. :210.00   
##   
## number grid position positionText   
## Min. : 0.00 Min. : 0.00 Min. : 1.000 Length:23777   
## 1st Qu.: 7.00 1st Qu.: 5.00 1st Qu.: 4.000 Class :character   
## Median : 15.00 Median :11.00 Median : 7.000 Mode :character   
## Mean : 16.97 Mean :11.27 Mean : 7.782   
## 3rd Qu.: 23.00 3rd Qu.:17.00 3rd Qu.:11.000   
## Max. :208.00 Max. :34.00 Max. :33.000   
## NA's :6 NA's :10550   
## positionOrder points laps time   
## Min. : 1.00 Min. : 0.000 Min. : 0.00 Length:23777   
## 1st Qu.: 7.00 1st Qu.: 0.000 1st Qu.: 20.00 Class :character   
## Median :13.00 Median : 0.000 Median : 52.00 Mode :character   
## Mean :13.08 Mean : 1.601 Mean : 45.27   
## 3rd Qu.:19.00 3rd Qu.: 1.000 3rd Qu.: 66.00   
## Max. :39.00 Max. :50.000 Max. :200.00   
##   
## milliseconds fastestLap rank fastestLapTime   
## Min. : 1474899 Min. : 2.00 Min. : 0.0 Length:23777   
## 1st Qu.: 5442948 1st Qu.:29.00 1st Qu.: 5.0 Class :character   
## Median : 5859428 Median :44.00 Median :11.0 Mode :character   
## Mean : 6303313 Mean :41.06 Mean :10.6   
## 3rd Qu.: 6495440 3rd Qu.:53.00 3rd Qu.:16.0   
## Max. :15090540 Max. :78.00 Max. :24.0   
## NA's :17774 NA's :18394 NA's :18246   
## fastestLapSpeed statusId   
## Length:23777 Min. : 1.00   
## Class :character 1st Qu.: 1.00   
## Mode :character Median : 11.00   
## Mean : 18.24   
## 3rd Qu.: 16.00   
## Max. :136.00   
##

* ‘time’ is inconsistently represented as minutes:seconds beyond 1 hr for first place driver, then the gap for next several drivers
* no time is recorded for racers greater than 1 lap behind the winner
* ‘position’ = raceFinish, ‘positionText’ = raceFinish or descriptor of retired/disqualified etc., ‘positionOrder’ = raceFinish or order of retired/disqualified
* positionText: D=Disqualified, E=Excluded, F=Did Not (/failed to) Qualify, N=Not Classified, R=Retired, W=Withrew
* fastest lap data is empty or NA from 1950 to 2004
* ‘grid’ is mostly a duplicate of position in qualifying table, but is likely more reliable

**seasons:** year and wiki page url of each season

## [1] 69 2

## year url   
## Min. :1950 Length:69   
## 1st Qu.:1967 Class :character   
## Median :1984 Mode :character   
## Mean :1984   
## 3rd Qu.:2001   
## Max. :2018

* somewhat irrelevant file unless wanting to scrape wiki pages

**status:** key and description of race results (e.g. finished, +1 Lap, collision, etc.)

## [1] 134 2

## statusId status   
## Min. : 1.00 Length:134   
## 1st Qu.: 34.25 Class :character   
## Median : 69.50 Mode :character   
## Mean : 68.71   
## 3rd Qu.:102.75   
## Max. :136.00

## Data Preprocessing

The preprocessing phase took place in two stages:

1. **Individual file cleaning** - For this phase I treated every file as if it could be utilized at some point (e.g. during Stage 3), despite my personal decision to exclude records for lap times, pitStops, qualifying times, etc. This is when most of the field renaming, deleting, and data type converting took place.
2. **Merged file cleaning** - After the individual files were cleaned, I merged the files through a series of left joins beginning on ‘results’, which contained the dependent variable(s). During this stage I continued to clean data, e.g. correcting typos, removing outliers, handling NAs, and additional feature creation, etc. I also merged latitude, longitude data for drivers’ home towns.

The entire process involved several interations of exploration, cleaning, and data transformation. The specifics of what was performed is outlined in the comments of the code chunks.

### Individual File Cleaning

# The below code was used for each and every section (where df = data frame) to inspect for completeness. It is not displayed repeatedly for space-saving purposes.  
  
# df(df == "" | df == "NULL" | is.na(df))

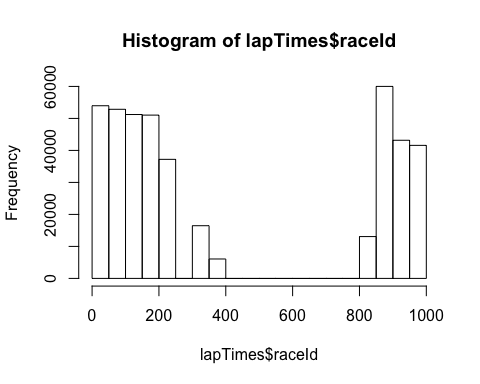
## circuits -----------------------------------------------------------------  
  
# remove columns: 'name', 'alt' ('name' is dirtier and lengthier than 'circuitRef' & 'alt' appears to have no meaning)  
# keep columns: 'url', 'lat', 'long' in case wanting to scrape wiki pages and/or calculate distance (e.g. from home)  
circuits <- subset(circuits, select = -c(name, alt))  
  
# clean dirty location names (caused by accent characters)  
circuits[c(4,18,20), "location"] <- c("Montmelo", "Sao Paulo", "Nurburg")  
  
# rename columns  
circuits <- circuits %>%  
 rename("circuit\_name" = "circuitRef", "circuit\_city" = "location", "circuit\_country" = "country", "circuit\_lat" = "lat", "circuit\_long" = "lng", "circuit\_url" = "url"  
 )  
  
  
## constructorResults -------------------------------------------------------  
  
# 'D' in 'status' column represents 'Disqualified' due to Spygate scandal in 2007 season (https://en.wikipedia.org/wiki/2007\_Formula\_One\_espionage\_controversy)  
  
# inspect distribution of 'status' (see above description of 'D') & remove column  
table(constructorResults$status)

##   
## D NULL   
## 17 11125

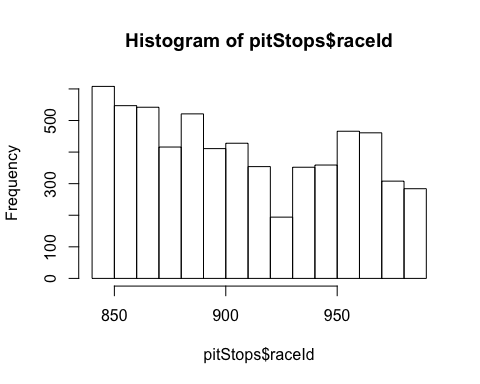
constructorResults <- subset(constructorResults, select = -status)  
  
# convert 'points' to int type  
constructorResults$points <- as.integer(constructorResults$points)  
  
# rename columns  
constructorResults <- constructorResults %>%  
 rename("constructorResult\_pointsPerRace" = "points")  
  
  
## constructors -------------------------------------------------------------  
  
# again keeping 'url' column incase of desire to scrape  
  
# remove columns: 'name', 'X' ('X' is entirely NA values)  
constructors <- subset(constructors, select = -c(X, name))  
  
# rename columns  
constructors <- constructors %>%  
 rename("constructor\_name" = "constructorRef", "constructor\_nationality" = "nationality", "constructor\_url" = "url"  
 )  
  
  
## constructorStandings -----------------------------------------------------  
  
# remove column: 'positionText' ('positionText' is identical to 'position' with the exception of 'E' for excluded due to "Spygate" scandal (see constructorResults section))  
# remove column: 'X' ('X' is entirely NA values)  
constructorStandings <- subset(constructorStandings, select = -c(positionText, X))  
  
# convert 'points' to int type  
constructorStandings$points <- as.integer(constructorStandings$points)  
  
# rename columns  
constructorStandings <- constructorStandings %>%  
 rename("constructorStanding\_runningTotalPointsInSeason" = "points", "constructorStanding\_runningPositionInSeason" = "position", "constructorStanding\_runningTotalWinsInSeason" = "wins"  
 )  
  
  
## drivers ------------------------------------------------------------------  
  
# remove column: 'number' ('number' is arbitrary driver/car number)  
# remove columns: 'code' 'forename', 'surname' ('code' serves no purpose $ forename' and 'surname' are messier and lengthier than 'driverRef')  
drivers <- subset(drivers, select = -c(number, code, forename, surname))  
  
# clean dob column and convert to date type variable using lubridate function  
drivers[415, "dob"] <- "12/08/1993"  
drivers$dob <- dmy(drivers$dob)

## Warning: 7 failed to parse.

# manually handle the 7 records that failed to parse and re-run lubridate  
drivers[c(590, 704, 742, 751, 761, 787, 792), "dob"] <-  
 c("1899-08-03", "1898-11-01", "1896-12-28", "1899-10-15", "1899-10-13", "1898-06-09", "1898-10-18")  
drivers$dob <- ymd(drivers$dob)  
  
# rename columns  
drivers <- drivers %>%  
 rename("driver\_name" = "driverRef", "driver\_dob" = "dob", "driver\_nationality" = "nationality", "driver\_url" = "url"  
 )  
  
  
## driverStandings ----------------------------------------------------------  
  
# remove column: 'positionText' ('positionText' is identical to 'position' with the exception of 'D' for disqualification from the championship due to intentional collision)  
driverStandings <- subset(driverStandings, select = -positionText)  
  
# convert 'points' to int type  
driverStandings$points <- as.integer(driverStandings$points)  
  
# rename columns  
driverStandings <- driverStandings %>%  
 rename("driverStanding\_runningTotalPointsInSeason" = "points", "driverStanding\_runningPositionInSeason" = "position", "driverStanding\_runningTotalWinsInSeason" = "wins"  
 )  
  
  
## lapTimes -----------------------------------------------------------------  
  
# inspect distribution of lap times  
hist(lapTimes$raceId)



# remove column: 'time' ('time' == 'milliseconds')  
lapTimes <- subset(lapTimes, select = -time)  
  
# create an average lap time column  
lapTimes <- lapTimes %>%  
 group\_by(raceId, driverId) %>%  
 mutate(lapTime\_avgMillisec = as.integer(mean(milliseconds)))  
  
# ungroup columns and convert back to data frame  
lapTimes <- lapTimes %>%  
 ungroup() %>%  
 as.data.frame()  
  
# rename columns  
lapTimes <- lapTimes %>%  
 rename("lapTime\_positionInLap" = "position", "lapTime\_millisec" = "milliseconds")  
  
  
## pitStops -----------------------------------------------------------------  
  
# inspect distribution of pit stops  
hist(pitStops$raceId)



# remove column: 'duration' ('duration' == 'milliseconds')  
pitStops <- subset(pitStops, select = -duration)  
  
# create a total number of stops per race column  
pitStops <- pitStops %>%  
 group\_by(raceId, driverId) %>%  
 arrange(raceId, driverId, stop) %>%  
 mutate(pitStop\_totalStops = last(stop))  
  
# create an average pitstop duration column  
pitStops <- pitStops %>%  
 group\_by(raceId, driverId) %>%  
 mutate(pitStop\_avgDuration = as.integer(mean(milliseconds)))  
  
# ungroup columns and convert back to data frame  
pitStops <- pitStops %>%  
 ungroup() %>%  
 as.data.frame()  
  
# rename columns  
pitStops <- pitStops %>%  
 rename("pitStop\_stopNum" = "stop", "pitStop\_lap" = "lap", "pitStop\_timeOfDay" = "time", "pitStop\_durationMillisec" = "milliseconds"  
 )  
  
  
## qualifying ---------------------------------------------------------------  
  
# remove column: 'number' ('number' is arbitrary driver/car number)  
qualifying <- subset(qualifying, select = -number)  
  
# correct erroneous record in q2  
qualifying[5663, "q2"] <- "1:48.552"  
  
# convert 'q1':'q3' to milliseconds  
qualifying <- qualifying %>%  
 separate(q1, into = c("q1\_minuteSecond", "q1\_decimal"), sep = "\\.") %>%  
 separate(q1\_minuteSecond, into = c("q1\_minute", "q1\_second"), convert = TRUE) %>%  
 mutate(q1\_milliseconds = paste0(((q1\_minute \* 60) + q1\_second), q1\_decimal))  
  
qualifying <- qualifying %>%  
 separate(q2, into = c("q2\_minuteSecond", "q2\_decimal"), sep = "\\.") %>%  
 separate(q2\_minuteSecond, into = c("q2\_minute", "q2\_second"), convert = TRUE) %>%  
 mutate(q2\_milliseconds = paste0(((q2\_minute \* 60) + q2\_second), q2\_decimal))  
  
qualifying <- qualifying %>%  
 separate(q3, into = c("q3\_minuteSecond", "q3\_decimal"), sep = "\\.") %>%  
 separate(q3\_minuteSecond, into = c("q3\_minute", "q3\_second"), convert = TRUE) %>%  
 mutate(q3\_milliseconds = paste0(((q3\_minute \* 60) + q3\_second), q3\_decimal))  
  
# convert 'NANA's (caused by concatenation) to NAs  
qualifying[qualifying == "NANA"] <- NA  
  
# remove columns: 'q1\_minute' through 'q3\_decimal' (temp columns created for parsing)  
qualifying <- subset(qualifying, select = -c(q1\_minute:q3\_decimal))  
  
# rename columns  
qualifying <- qualifying %>%  
 rename("qualifying\_finishPosition" = "position", "qualifying\_q1Millisec" = "q1\_milliseconds", "qualifying\_q2Millisec" = "q2\_milliseconds", "qualifying\_q3Millisec" = "q3\_milliseconds"  
 )  
  
  
## races --------------------------------------------------------------------  
  
# convert 'date' to date type  
races$date <- ymd(races$date)  
  
# convert 'time' to time type  
races$time <- times(races$time)  
  
# rename columns  
races <- races %>%  
 rename("race\_year" = "year", "race\_round" = "round", "race\_name" = "name", "race\_date" = "date", "race\_time" = "time", "race\_url" = "url"  
 )  
  
  
## results ------------------------------------------------------------------  
  
# remove columns: 'position', 'number', 'time', 'fastestLap', 'rank', 'fastestLapTime', 'fastestLapSpeed'  
# ('position' is redundant, 'number' is arbitrary driver/car number, 'time' is represented more cleanly in 'milliseconds')  
results <- subset(results, select = -c(position, number, time))  
  
# convert positionText column to finish description  
results <- mutate(results, result\_finishDescription =   
 ifelse(results$positionText == "D", "Disqualified",  
 ifelse(results$positionText == "E", "Excluded",  
 ifelse(results$positionText == "F", "FailedToFinish",  
 ifelse(results$positionText == "N", "NotClassified",  
 ifelse(results$positionText == "R", "Retired",  
 ifelse(results$positionText == "W", "Withdrew",  
 "Finished"  
 )))))))  
results <- subset(results, select = -positionText)  
  
# convert fastestLapTime to milliseconds  
results <- results %>%  
 separate(fastestLapTime, into = c("fLT\_minuteSecond", "fLT\_decimal"), sep = "\\.") %>%  
 separate(fLT\_minuteSecond, into = c("fLT\_minute", "fLT\_second"), convert = TRUE) %>%  
 mutate(fLT\_milliseconds = paste0(((fLT\_minute \* 60) + fLT\_second), fLT\_decimal, "00"))  
  
# convert 'NANA00's (caused by concatenation) to NAs  
results$fLT\_milliseconds[results$fLT\_milliseconds == "NANA00"] <- NA  
  
# convert 'points', 'fastestLapSpeed', and 'fLT\_milliseconds' to appropriate number types  
results$points <- as.integer(results$points)  
results$fLT\_milliseconds <- as.integer(results$fLT\_milliseconds)  
results$fastestLapSpeed <- as.double(results$fastestLapSpeed)

## Warning: NAs introduced by coercion

# remove columns: 'q1\_minute' through 'q3\_decimal' (temp columns created for parsing)  
results <- subset(results, select = -c(fLT\_minute:fLT\_decimal))  
  
# rename columns  
results <- results %>%  
 rename("result\_startingGridPosition" = "grid", "result\_finishOrder" = "positionOrder", "result\_pointsEarned" = "points", "result\_lapsCompleted" = "laps", "result\_finishTimeMillisec" = "milliseconds", "result\_fastestLapTimeMillisec" = "fLT\_milliseconds", "result\_fastestLap" = "fastestLap", "result\_fastestLapRank" = "rank", "result\_fastestLapSpeed" = "fastestLapSpeed"  
 )  
  
  
## seasons ------------------------------------------------------------------  
  
# convert year to date type  
seasons$year <- as.character(seasons$year)  
seasons$year <- as.Date(paste(seasons$year, 1, 1, sep = "-")) # beginning of year  
seasons$year <- as.Date(paste(seasons$year, 12, 31, sep = "-")) # end of year  
  
# rename columns  
seasons <- seasons %>%  
 rename("season\_year" = "year", "season\_url" = "url")  
  
  
## status ------------------------------------------------------------------  
  
# rename column  
colnames(status) [colnames(status) == "status"] <- "status\_description"

### Merge and Additional Cleaning

# read in country Codes  
countryCodes <- read.csv("../Raw Source Data/countryCodes.csv", stringsAsFactors = FALSE)  
  
# manually correct errors caused by accent marks  
countryCodes[1214, 1] <- "Monegasque"  
  
# join countryCodes to drivers and constructors to convert nationality to country  
drivers <- drivers %>%  
 left\_join(countryCodes, by = c("driver\_nationality" = "Nationality")) %>%  
 select(-driver\_nationality) %>%  
 rename("driver\_homeCountry" = "Country")  
  
constructors <- constructors %>%  
 left\_join(countryCodes, by = c("constructor\_nationality" = "Nationality")) %>%  
 select(-constructor\_nationality) %>%  
 rename("constructor\_homeCountry" = "Country")  
  
# manually correct errors cause by compound nationalities (e.g. 'East German') or similar causes  
drivers[496, 5] <- "Italy"  
drivers[578, 5] <- "Italy"  
drivers[714, 5] <- "Germany"  
drivers[715, 5] <- "Germany"  
drivers[718, 5] <- "Germany"  
  
constructors[100, 4] <- "Belgium"  
constructors[146, 4] <- "Germany"  
  
# join countriesLatLong to drivers to obtain lat, long coordinates  
# source: https://github.com/knowitall/chunkedextractor/blob/master/src/main/resources/edu/knowitall/chunkedextractor/demonyms.csv  
countriesLatLong <- read.csv("../Raw Source Data/countries\_LatLong.csv", stringsAsFactors = FALSE)  
  
drivers <- drivers %>%  
 left\_join(countriesLatLong, by = c("driver\_homeCountry" = "name")) %>%  
 rename("driver\_lat" = "latitude", "driver\_long" = "longitude")  
  
# clean missing lat, long for "Rhodesia"  
# arrange rows by driver\_homeCountry  
drivers <- drivers %>%  
 arrange(driver\_homeCountry)  
  
# manually update missing lat, longs  
drivers[445:448, 6] <- -19.0154381  
drivers[445:448, 7] <- 29.1548576  
  
# join countriesLatLong to constructors to obtain lat, long coordinates  
# source: https://github.com/knowitall/chunkedextractor/blob/master/src/main/resources/edu/knowitall/chunkedextractor/demonyms.csv  
constructors <- constructors %>%  
 left\_join(countriesLatLong, by = c("constructor\_homeCountry" = "name")) %>%  
 rename("constructor\_lat" = "latitude", "constructor\_long" = "longitude")  
  
# manually update missing lat, longs  
constructors[93, 5] <- -19.0154381  
constructors[93, 6] <- 29.1548576  
  
# join source tables beginning from 'results' in reverse alphabetical order  
# confirm primary key  
results %>%  
 count(resultId) %>%  
 filter(n > 1)

## # A tibble: 0 x 2  
## # … with 2 variables: resultId <int>, n <int>

# join status  
resultsHistorical <- results %>%  
 left\_join(status, by = "statusId") %>%  
 select(-statusId)  
  
# exclude seasons  
  
# join races  
resultsHistorical <- resultsHistorical %>%  
 left\_join(races, by = "raceId") %>%  
 select(-race\_name, -race\_year, -race\_time, -race\_url)  
  
# exclude qualifying  
  
# exclude pitStops  
  
# exclude lapTimes  
  
# join driverStandings  
# confirm primary key  
driverStandings %>%  
 count(raceId, driverId) %>%  
 filter(n > 1)

## # A tibble: 0 x 3  
## # … with 3 variables: raceId <int>, driverId <int>, n <int>

resultsHistorical <- resultsHistorical %>%  
 left\_join(driverStandings, by = c("raceId", "driverId")) %>%  
 select(-driverStandingsId)  
  
# join drivers  
resultsHistorical <- resultsHistorical %>%  
 left\_join(drivers, by = "driverId") %>%  
 select(-driverId, -driver\_url)  
  
# join constructorStandings  
# confirm primary key  
constructorStandings %>%  
 count(raceId, constructorId) %>%  
 filter(n > 1)

## # A tibble: 0 x 3  
## # … with 3 variables: raceId <int>, constructorId <int>, n <int>

resultsHistorical <- resultsHistorical %>%  
 left\_join(constructorStandings, by = c("raceId", "constructorId")) %>%  
 select(-constructorStandingsId)  
  
# join constructors  
resultsHistorical <- resultsHistorical %>%  
 left\_join(constructors, by = "constructorId") %>%  
 select(-constructor\_url)  
  
# join constructorResults  
# confirm primary key  
constructorResults %>%  
 count(raceId, constructorId) %>%  
 filter(n > 1)

## # A tibble: 2 x 3  
## raceId constructorId n  
## <int> <int> <int>  
## 1 75 7 2  
## 2 903 3 2

# remove duplicate records  
constructorResults <- constructorResults[-c(9632, 10355),]  
  
resultsHistorical <- resultsHistorical %>%  
 left\_join(constructorResults, by = c("raceId", "constructorId")) %>%  
 select(-constructorResultsId, -constructorId)  
  
# join circuits  
resultsHistorical <- resultsHistorical %>%  
 left\_join(circuits, by = "circuitId") %>%  
 select(-circuitId, -circuit\_url, -circuit\_lat, -circuit\_long)  
  
  
#### Clean Newly Merged Data Frame ####  
  
# inspect df (below 4 lines commented out for space-saving purposes)  
# glimpse(resultsHistorical)  
# summary(resultsHistorical)  
# colSums(is.na(resultsHistorical))  
# head(resultsHistorical)  
  
#normalize laps completed field  
resultsHistorical <- resultsHistorical %>%  
 group\_by(raceId) %>%  
 mutate(maxLaps = max(result\_lapsCompleted),  
 result\_percentOfRaceCompleted = (result\_lapsCompleted / maxLaps) \* 100,  
 result\_lapsCompleted = NULL,  
 maxLaps = NULL  
 )  
  
# ungroup columns  
resultsHistorical <- resultsHistorical %>%  
 ungroup() %>%  
 as.data.frame()  
  
# remove resultId column which is no longer needed  
resultsHistorical <- subset(resultsHistorical, select = -resultId)  
  
# remove result\_finishTimeMillisec due to difficult to handle NAs  
resultsHistorical <- subset(resultsHistorical, select = -result\_finishTimeMillisec)  
  
# remove columns relating to fastest lap data which contain many untreatable NAs  
resultsHistorical <- subset(resultsHistorical, select = -c(result\_fastestLap:result\_fastestLapSpeed, result\_fastestLapTimeMillisec))  
  
# fill NAs in 'points' & 'wins' related columns with 0 and remove 'position' related columns  
 # NAs in 'driverStandings.' and 'consturctorStandings.' caused by failures to qualify in early season races  
 # attempt later -- [fill NAs in 'position' columns with \*max runningPositionsInSeason + 1\* for each season]  
 # use grouped mutate (window function)  
resultsHistorical$driverStanding\_runningTotalPointsInSeason[is.na(resultsHistorical$driverStanding\_runningTotalPointsInSeason)] <- 0  
  
resultsHistorical$driverStanding\_runningTotalWinsInSeason[is.na(resultsHistorical$driverStanding\_runningTotalWinsInSeason)] <- 0  
  
resultsHistorical <- subset(resultsHistorical, select = -driverStanding\_runningPositionInSeason)  
  
resultsHistorical$constructorStanding\_runningTotalPointsInSeason[is.na(resultsHistorical$constructorStanding\_runningTotalPointsInSeason)] <- 0  
  
resultsHistorical$constructorStanding\_runningTotalWinsInSeason[is.na(resultsHistorical$constructorStanding\_runningTotalWinsInSeason)] <- 0  
  
resultsHistorical$constructorResult\_pointsPerRace[is.na(resultsHistorical$constructorResult\_pointsPerRace)] <- 0  
  
resultsHistorical <- subset(resultsHistorical, select = -constructorStanding\_runningPositionInSeason)  
  
# create driver age at time of race feature and convert to int  
# remove 'driver\_dob' (consider re-adding in future to analyze race results on closeness to birthday)  
resultsHistorical <- resultsHistorical %>%  
 mutate(driverAge = (race\_date - driver\_dob) / 365,  
 driver\_dob = NULL  
 )  
  
resultsHistorical$driverAge <- as.integer(resultsHistorical$driverAge)  
  
# correct outlier  
resultsHistorical[17012, "driverAge"] = 33  
  
  
# convert race date to month and year columns (for better factorization)  
# remove 'race\_date'  
resultsHistorical <- resultsHistorical %>%  
 mutate(race\_month = month(race\_date),  
 race\_year = year(race\_date),  
 race\_date = NULL  
 )  
  
# create running total points going into race for driver and constructor and remove existing columns  
resultsHistorical <- resultsHistorical %>%  
 mutate(driver\_preRaceTotPoints = driverStanding\_runningTotalPointsInSeason - result\_pointsEarned,  
 constructor\_preRaceTotPoints = constructorStanding\_runningTotalPointsInSeason - constructorResult\_pointsPerRace,  
 )  
  
# correct for inaccurate observations (i.e. negative points) caused by NAs from disqualification, failure to qualify, etc.  
# 1 error in driverPoints  
resultsHistorical$driver\_preRaceTotPoints <- ifelse(resultsHistorical$driver\_preRaceTotPoints < 0, 0, resultsHistorical$driver\_preRaceTotPoints)  
  
# ~50 errors in constructor points  
resultsHistorical$constructor\_preRaceTotPoints <- ifelse(resultsHistorical$constructor\_preRaceTotPoints < 0, 0, resultsHistorical$constructor\_preRaceTotPoints)  
  
# remove unnecessary columns  
resultsHistorical <- subset(resultsHistorical, select = -c(driverStanding\_runningTotalPointsInSeason, constructorStanding\_runningTotalPointsInSeason))  
  
# create running total wins going into race for driver and constructor and remove existing columns  
resultsHistorical <- resultsHistorical %>%  
 mutate(driver\_preRaceTotWins = ifelse(result\_finishOrder == 1, driverStanding\_runningTotalWinsInSeason - 1, driverStanding\_runningTotalWinsInSeason),  
 driverStanding\_runningTotalWinsInSeason = NULL  
 )  
  
resultsHistorical <- resultsHistorical %>%  
 group\_by(raceId, constructor\_name) %>%  
 mutate(constructor\_preRaceTotWins = ifelse(min(result\_finishOrder) == 1, constructorStanding\_runningTotalWinsInSeason - 1, constructorStanding\_runningTotalWinsInSeason),  
 constructorStanding\_runningTotalWinsInSeason = NULL  
 )  
  
# ungroup columns  
resultsHistorical <- resultsHistorical %>%  
 ungroup() %>%  
 as.data.frame()  
  
# correct for inaccurate observations (i.e. negative points) caused by NAs from disqualification, failure to qualify, etc.  
# >200 in constructorTotWins  
resultsHistorical$constructor\_preRaceTotWins <- ifelse(resultsHistorical$constructor\_preRaceTotWins < 0, 0, resultsHistorical$constructor\_preRaceTotWins)  
  
# remove unnecessary columns  
resultsHistorical <- subset(resultsHistorical, select = -c(raceId, constructorResult\_pointsPerRace))  
  
# create features which contain previous race results and remove current race result fields  
resultsHistorical <- resultsHistorical %>%  
 arrange(driver\_name, race\_year, race\_round) %>%  
 mutate(result\_percentOfPreviousRaceCompleted = ifelse(driver\_name == lag(driver\_name) & !is.na(lag(driver\_name)), lag(result\_percentOfRaceCompleted), 0),  
 result\_previousFinishDescrip = ifelse(driver\_name == lag(driver\_name) & !is.na(lag(driver\_name)), lag(result\_finishDescription), "First race"),  
 status\_previousDescrip = ifelse(driver\_name == lag(driver\_name) & !is.na(lag(driver\_name)), lag(status\_description), "First race"),  
 result\_percentOfRaceCompleted = NULL,  
 result\_finishDescription = NULL,  
 status\_description = NULL  
 )  
  
# convert columns (driverAge, percentOfPreviousRaceCompleted, preRaceTotPoints, preRaceTotWins) to integers  
resultsHistorical$driverAge <- as.integer(resultsHistorical$driverAge)  
resultsHistorical$result\_percentOfPreviousRaceCompleted <- as.integer(resultsHistorical$result\_percentOfPreviousRaceCompleted)  
resultsHistorical$driver\_preRaceTotPoints <- as.integer(resultsHistorical$driver\_preRaceTotPoints)  
resultsHistorical$driver\_preRaceTotWins <- as.integer(resultsHistorical$driver\_preRaceTotWins)  
  
# convert pointsEarned to boolean  
resultsHistorical <- resultsHistorical %>%  
 mutate(result\_inThePoints = ifelse(result\_pointsEarned > 0, TRUE, FALSE),  
 result\_pointsEarned = NULL  
 )  
  
# arrange rows  
resultsHistorical <- resultsHistorical %>%  
 arrange(race\_year, race\_round, result\_finishOrder, result\_startingGridPosition)  
  
# arrange columns  
resultsHistorical <- resultsHistorical[, c("result\_finishOrder", "result\_inThePoints", "result\_startingGridPosition", "driver\_name", "constructor\_name", "circuit\_name", "circuit\_city", "race\_month", "race\_year", "race\_round", "driverAge", "driver\_homeCountry", "driver\_lat", "driver\_long", "constructor\_homeCountry", "constructor\_lat", "constructor\_long", "result\_percentOfPreviousRaceCompleted", "result\_previousFinishDescrip", "status\_previousDescrip", "driver\_preRaceTotPoints", "constructor\_preRaceTotPoints", "driver\_preRaceTotWins", "constructor\_preRaceTotWins")]  
  
# remove columns due to appeared unreliability  
resultsHistorical <- subset(resultsHistorical, select = -c(constructor\_preRaceTotPoints, constructor\_preRaceTotWins))

## Algorithms & Execution

### K-Means

Once the data had been cleaned and consolidated, I wanted to check for hidden patterns in the dataset via clustering. I chose to test K-Means on driver home country (lat, long) because the sport is widely recognized to be biased towards European drivers and races. Could it be that drivers from a particular region are actually that much better?

The process of determining the correct number of clusters (k) involved plotting an elbow curve to measure levels of distortion. On several passes however, the kmeans.ani() function failed to identify the provided number of clusters. It seemed most consistent (as well as most visually logical) at 4 clusters.

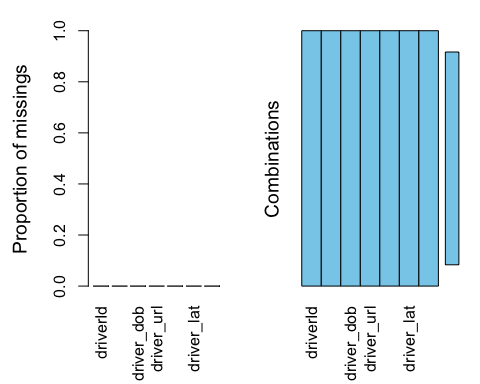
### Naive Bayes

After running K-Means, I first assigned the results back to the dataset and proceeded to test Naive Bayes, assigning the dependent variable as the binarized ‘result\_inThePoints’ feature. I kept most of the columns to use as predictors with the exception of lat, long fields which were now represented categorically in the results of the clutering. I also removed ‘driver\_finishOrder’ which had been retained as an alternate dependent variable (factor with ~ 25 levels).

The decision to perform Naive Bayes over other algorithms like Random Forest or Support Vector Machine was somewhat through process of elimination. For example, Random Forest can not handle factors of over ~50 levels, which would’ve meant more cleaning (e.g. # of circuits is over 70). Meanwhile, SVM did not appeal due its known slow processing speed. On first attempt with Naive Bayes, I obtained very reasonable results so I did not continue to explore other classifiers.

Once the model was tested and confirmed improvement over baseline, I decided to test it on results from a race that took place just last weekend, the 2019 Brazilian Grand Prix. This required translating results data (via Ergast Developer API) into the appropriate format required for the model.

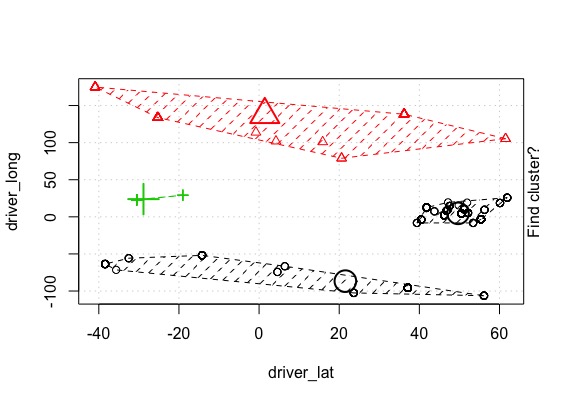
#### Explore data ####  
  
# confirm no missing values  
aggr(drivers)



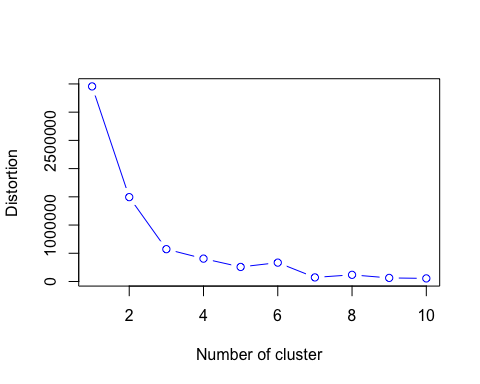
## K-means -----------------------------------------------------------------  
  
# incorporate lat long of driver homes to get clusters of where they come from  
# then use cluster association as new feature for prediction  
  
# partition the data  
driversLatLong <- drivers[,c(2, 6:7)]  
  
# convert driver name to factor  
driversLatLong$driver\_name <- as.factor(driversLatLong$driver\_name)  
  
# set seed for randmozing data  
set.seed(20)  
  
# run k-means  
clusters <- kmeans(driversLatLong[,c(2:3)], 4)  
  
# inspect cluters  
str(clusters)

## List of 9  
## $ cluster : int [1:848] 1 1 1 1 1 1 1 1 1 1 ...  
## $ centers : num [1:4, 1:2] 21.53 49.66 -28.85 1.42 -86.67 ...  
## ..- attr(\*, "dimnames")=List of 2  
## .. ..$ : chr [1:4] "1" "2" "3" "4"  
## .. ..$ : chr [1:2] "driver\_lat" "driver\_long"  
## $ totss : num 3456348  
## $ withinss : num [1:4] 272829 47434 586 84304  
## $ tot.withinss: num 405152  
## $ betweenss : num 3051196  
## $ size : int [1:4] 248 521 27 52  
## $ iter : int 2  
## $ ifault : int 0  
## - attr(\*, "class")= chr "kmeans"

# plot clusters (attempt multiple times if necessary)  
num\_clusters <- 4  
  
# driverClust <- kmeans.ani(driversLatLong[,c(2:3)], num\_clusters)  
# above line commented out for report compilation purposes



# save the cluster number in the dataset as column 'driverHomeCluster'  
driversLatLong <- driversLatLong %>%  
 mutate(driverCluster = as.factor(clusters$cluster))  
  
# create a function that returns the value of totwithinss, and takes inputdataset and number of clusters  
# credit: Dr. Binh Zhang, The University of Arizona  
kmeans.totwithinss.k <- function(dataset, number\_of\_centers){  
 km <- kmeans(dataset, number\_of\_centers)  
 km$tot.withinss  
 }  
  
# create a function that returns a series of totwithinss values, and takes input maxk  
# vec is a vector that contains totwithinss values associated with k from 1 to maxk  
# credit: Dr. Binh Zhang, The University of Arizona  
kmeans.distortion <- function(dataset, maxk){  
 vec <- as.vector(1:maxk)  
 vec[1:maxk] <- sapply(1:maxk, kmeans.totwithinss.k, dataset = dataset)  
 return (vec)  
 }  
  
# plot elbow curve  
maxk <- 10  
  
dist\_vect <- kmeans.distortion(driversLatLong[,c(2:3)], maxk)  
  
plot(1:maxk, # horizontal axis  
 dist\_vect, # vertical axis  
 type= 'b', # curve  
 col = 'blue',  
 xlab = "Number of cluster",  
 ylab = "Distortion"  
 )



# reintroduce cluster results to resultsHistorical  
driversLatLong <- subset(driversLatLong, select = -c(driver\_lat, driver\_long))  
  
resultsHistorical <- resultsHistorical %>%  
 left\_join(driversLatLong, by = "driver\_name")  
  
  
#### Predict ####  
  
## Naive Bayes ------------------------------------------------------------  
  
# review the data (below 2 lines commented out for space-saving purposes)  
# summary(resultsHistorical)  
# glimpse(resultsHistorical)  
  
# clean dataset  
resultsNB <- subset(resultsHistorical, select = -c(  
 result\_finishOrder,driver\_lat, driver\_long, constructor\_lat, constructor\_long)  
 )  
  
# partition data for training and testing  
sample\_size <- floor(0.7 \* nrow(resultsNB))  
  
training\_index <- sample(nrow(resultsNB), size = sample\_size, replace = FALSE)  
train <- resultsNB[training\_index,]  
test <- resultsNB[-training\_index,]  
  
# check dimensions of partitions  
dim(train)

## [1] 17326 18

dim(test)

## [1] 7426 18

# fit model  
results.model <- naiveBayes(result\_inThePoints ~ . , data = train)  
  
# review model (below line commented out for space-saving purposes)  
# results.model  
  
# test model  
results.predict <- predict(results.model, test, type = 'class')  
  
# test model against recent Brazil Grand Prix results  
testBrazil <- read.csv("../Processed Data/brazil2019.csv")  
  
# factorize driverCluster dimension  
testBrazil$driverCluster <- as.factor(testBrazil$driverCluster)  
  
results.predict2 <- predict(results.model, testBrazil, type = 'class')  
  
resultsNB\_output2 <- data.frame(driver\_name = testBrazil[,'driver\_name'], predicted = results.predict2, actual = testBrazil[,'result\_inThePoints'])

## Results

### K-Means

Unfortunately the results from K-Means were somewhat underwhelming. In essence, all that happened is that drivers were clustered according to continents. Obviously this data could have been easily supplemented without the use of an advanced algorithm. Nvertheless, it was still interesting to view the distribution on an animated plot chart.

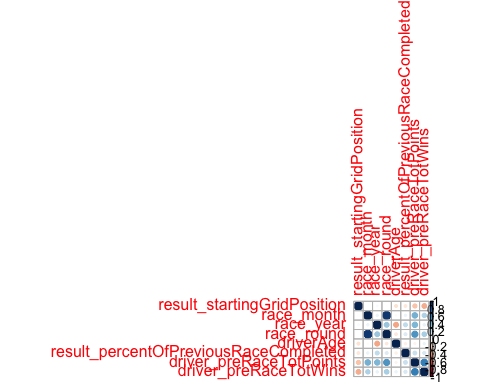
### Naive Bayes

Results from Naive Bayes were more interesting and encouraging. The model acheived an accuracy of 77.97%, which is an improvement of 6.28% from baseline. The specificity rate was also quite impressive at 86.62%, however it is somewhat understandable given that racers in the early years of F1 would often compete in qualifying and then not actually race (thus resulting in a high FALSE rate, ie. ‘Not in the points’).

Results when tested against the 2019 Brazilian Grand Prix outcomes were very encouraging, producing an accuracy of 65% (a 15% improvement over baseline). The reason the baseline is higher in this scenario is because now it is always the case that 10 out of 20 drivers finish in the points (i.e. 50% conditional probability). Whereas in the past, points were allocated differently and the number of drivers in the field varied.

Additional evaluation metrics are listed below in the code chunk (on next page).

#### Analyze results ####  
  
# inspect correlation of variables, which could impact model (i.e. assumed independence)  
train %>%  
 filter(result\_inThePoints == TRUE) %>%  
 select\_if(is.numeric) %>%  
 cor() %>%  
 corrplot::corrplot()



# create confusion matrix  
resultsNB\_output <- data.frame(actual = test[,'result\_inThePoints'], predicted = results.predict)  
  
table(resultsNB\_output)

## predicted  
## actual FALSE TRUE  
## FALSE 4583 676  
## TRUE 973 1194

# assign 'True Positive', 'False Positive', 'True Negative', 'False Negative'  
TP <- 1161  
FP <- 715  
TN <- 4629  
FN <- 921  
  
# calculate evaluation metrics  
errorRate <- (FP + FN) / (TP + FP + TN + FN)  
accuracy <- (TP + TN) / (TP + FP + TN + FN)  
sensitivity <- TP / (TP + FN)  
specificity <- TN / (TN + FP)  
precision <- TP / (TP + FP)  
falsePositiveRate <- FP / (TN + FP)  
  
  
## Overall Results -------------------------------------------------------  
  
  
# error rate = .2203  
# model accuracy = .7797  
 # baseline accuracy = .7169  
 # accuracy improvement = .0628  
# sensitivity (true positive) = .5576  
# specificity (true negative) = .8662  
# precision = .6189  
# false positive rate = .1338  
  
  
# results of 2019 Brazilian Grand Prix predictions  
table(resultsNB\_output2[,c(2,3)])

## actual  
## predicted FALSE TRUE  
## FALSE 4 2  
## TRUE 6 8

# accuracy = .65  
# baseline accuracy = .5  
  
resultsNB\_output2

## driver\_name predicted actual  
## 1 max\_verstappen TRUE TRUE  
## 2 vettel TRUE FALSE  
## 3 hamilton TRUE TRUE  
## 4 leclerc TRUE FALSE  
## 5 bottas TRUE FALSE  
## 6 albon TRUE FALSE  
## 7 gasly TRUE TRUE  
## 8 grosjean FALSE FALSE  
## 9 raikkonen TRUE TRUE  
## 10 kevin\_magnussen FALSE FALSE  
## 11 norris TRUE TRUE  
## 12 ricciardo TRUE TRUE  
## 13 giovinazzi FALSE TRUE  
## 14 hulkenberg TRUE FALSE  
## 15 perez TRUE TRUE  
## 16 kvyat FALSE TRUE  
## 17 stroll TRUE FALSE  
## 18 russell FALSE FALSE  
## 19 kubica FALSE FALSE  
## 20 sainz TRUE TRUE

## Conclusion/reflection

Results from this analysis will undoubtedly help the team going forward. Naive Bayes appears to be an effective model that has still has room for improvement if highly correlated variables are removed. If nothing else, the dataset is now much more robust and easier to work with. It will provide a good platform for testing more advanced deep learning algorithms that may require more structured data.

Apart from this advancement, I was very pleased at how well the model performed when tested against auxillary data (i.e. 2019 Grand Prix results). Being able to succesfully predict with 65% accuracy whether or not a driver will finish better than 10 out of 20 is not bad! It is especially impressive considering the model actually chose more than 10 drivers to finish in the points which is no longer relevant. Perhaps the next improvement can be to limit the model to predict only a certain number of ‘points finishers’ depending on the race season in question.

Regardless, I really enjoyed this project and look forward to testing the model on the race next weekend in Abu Dhabi!