FORMULA 1 CHAMPIONSHIP'S GRAND PRIX WINNING PREDICITION

submitted in the partial fulfilment of the requirements

for the award of the degree in

BACHELOR OF SCIENCE

by

SOORIAN T.S

(193031101109)

DEPARTMENT OF COMPUTER SCIENCE

Dr. M.G.R.
EDUCATIONAL AND RESEARCH INSTITUTE
(Deemed to be University)
Maduravoyal, Chennai - 600 095. Tamilnadu. India.
(An ISO 9001 : 2015 Certified Institution)

MAY 2022

DECLARATION

I'm, Soorian T.S, hereby declare that the Project Report entitled **"Formula 1 Championship's Grand Prix Winning Prediction"** is done by us under the guidance of Dr./Prof./Mr./Ms. **Radha Jayalakshmi** (Internal) is submitted in partial fulfilment of the requirements for the award of the degree in BACHELOR OF SCIENCE

DATE:

PLACE:

SIGNATURE OF THE CANDIDATE

Dr. M.G.R.
EDUCATIONAL AND RESEARCH INSTITUTE
(Deemed to be University)
Maduravoyal, Chennai - 600 095. Tamilnadu. India.
(An ISO 9001 : 2015 Certified Institution)

DEPARTMENT OF COMPUTER SCIENCE

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of Mr. **SOORIAN T.S** Reg.No.193031101109 who carried out the project entitled "**FORMULA 1 CHAMPIONSHIP GRAND PRIX WINNING PREDICTION**" under our supervision from June 2019 to May 2022.

Internal Guide **Head of the department**

Submitted for Viva Voce Examination held on_____

Internal Examiner **External Examiner**

ACKNOWLEDGEMENT

We would first like to thank our beloved Chancellor **Thiru A.C. Shanmugam, B.A., B.L.** and President **Er. A.C.S. Arunkumar, B.Tech., M.B.A.,** also our secretary **Thiru. A. Ravikumar** for all the encouragement and support extended to us during the tenure of this project and also our years of studies in his wonderful University.

We express our heartfelt thanks to our Vice-Chancellor **Prof. Dr. S.Geetha Lakshmi** in providing all the support of our Project. We express our heart felt thanks to our Head of The Department **Prof. Dr. A.R. Arunachalam** who has been actively involved and very influential from the start till the completion of our project.

Our sincere thanks to our Project Coordinator **Dr. S. ISMAILKALILULAH** and Project guide **Ms. RADHA JEYALAKSHMI** for their continuous guidance and encouragement throughout this work, which has made the project a success.

We would also like to thank all the teaching and non-teaching staffs of Computer Science department, for their constant support and the encouragement given to us while we went about achieving our project

List of Contents

1 Introduction

1.1 Aim of the project
1.2 Project domain
1.3 Objective
1.4 Existing System
1.5 Proposed System

2 System Specifications

2.1 Software Specifications

3 Implementation and Testing

3.1 Data Collection
3.2 Data Analysis
3.3 Machine Learning Modelling
3.4 System code

4 Module Description

4.1 Data flow Diagram
4.2 Tasks
4.3 Limitations

5 Results and Discussions

5.1 Applications
5.2 Advantages

6 Conclusion and future enhancements

6.1 Conclusion
6.2 Future Enhancement

LIST OF FIGURES

S.NO	TITLES	PAGE
1	Race Info	13
2	Race Results	14
3	Driver Standings	15
4	Constructor Standings	16
5	Qualifying	17
6	Weather	18

LIST OF DATA ANALYSIS

S.NO	TITLES	PAGE
1	Tracks and Countries	19
2	Correlation	20
3	Most Dangerous Circuits	21
4	Teams Car Failures	22
5	Driver Crashes	23

ABSTRACT

Most of the people in this world had a known the Formula 1 championship , It's a highest clash of international open wheel single seated formula cars which are sanctioned by FEDERATION INTERNATIONAL OF AUTOMOBILE which is shortly known as FIA. The World Drivers' Championship, which became the FIA Formula One World Championship in 1981, has been one of the premier forms of racing around the world since its inaugural season in 1950. The word formula in the name refers to the set of rules to which all participants' cars must conform.[1] A Formula One season consists of a series of races, known as Grands Prix, which take place worldwide on both purpose-built circuits and closed public roads.

Formula 1 cars are the fastest road course racing cars which have maximum speed of 200 kmph in corners with high gravitational and downforce. these cars can achieve speed of 350kmph in a long track lane.

From the audience and fans point of view this formula 1 is just of all a racing, a driver who drives car at maximum speed to win his race. But apart from this, it is not only a racing championship, it the area where each and every driver risks their lives for to gain championship and also to entertain their fans. Most of drivers has lost their lives over Decades. And also this enthusiastic game not only focus on driver it goes through with each and every aspects of cars, drivers mind, team management ,car management , analysis of track, climatic condition, decision making of managers, modifying the car before race for good performance and etc. Car Engineering is also important in this part, from the front wing to rear exhaust the car should be safely inspected by team engineers, Car analysis is a big deal for engineers apart from others, because the Cars are being upgraded for every year, like Aerodynamics, Engine Displacement, Coolent systems, Power Units and etc. Even Cars are modified from track to track , According to track length, track condition or temperature.

So as I said its not only a racing its all about the various aspects which I have mentioned above. So these aspects are tends to make this championship as thrilling and enjoyable moments for fans.

FIA also consist of lower end CC championship which are Formula 2 and Formula 3, where these cars are smaller than F1 cars in basis of speed , power and technology and etc. Even FIA conducts Formula E championship, it’s also a single seated closed wheel car which are runs out in electronic motors instead of internal combustion engine(ICE).

CHAPTER-1

INTRODUCTION

Formula 1 championship , It's a highest clash of international open wheel single seated formula cars which are sanctioned by FEDERATION INTERNATIONAL OF AUTOMOBILE which is shortly known as FIA. The World Drivers' Championship, which became the FIA Formula One World Championship in 1981, has been one of the premier forms of racing around the world since its inaugural season in 1950. The word formula in the name refers to the set of rules to which all participants' cars must conform.[1] A Formula One season consists of a series of races, known as Grands Prix, which take place worldwide on both purpose-built circuits and closed public roads.

AIM OF THE PROJECT

So if a driver wants to win a race their team should go through and analyse all those aspects which I have said. So with help of all data's, I'm going to predict the winner of the next F1 Grand Prix using Data Science and Machine Learning.

First the data's will be collected from ergast database which have all the data's of F1 championships since 1950. But in this project I have just only taken the data's of previous year 2021. Once the data's are collected from database next it will be modified for some changes using python in jupyter notebook(one of the best IDE for python developments) and then I gather out race results, qualifying results, constructor championship, driver championship finally this data cleaning process is done. Then here come the main part Data Analysis of all the aspects that I have collected for data analysis I'm using seaborn and matplotlib which are most popular data analysing library functions in python, once completion of analysis the final part to build machine learning algorithm here's the algorithm is the most important part of this project to analyse all those datasets to give the predicted results.

Project Domain

So, this project is done by using Python 3 as developing language under Jupyter Notebook, an excellent Integrated Development Environment (IDE) for data science and machine learning projects. Data sets are downloaded as CSV files from Ergast Developers API , which is the archive of all Formula 1 data's since 1950's. so we can modify the data sets as per our convenient using Python 3 in Jupyter Notebook. So this projects focuses on three main agenda's first collection of datasets, analysing of dataset, machine learning modelling for collected and analysed dataset. And finally giving out the actual predicted results.

OBJECTIVE:

To predict the actual winner of grand prix using Data Science with Machine Learning modelling.

EXISTING SYSTEM:

I have gone through Arthur. Veronica Nigro, a Data Scientist with a background in Finance and economics. She have done very well analysis of data's and about collection of data she have gone through collecting the data's in forms of csv files. And directly downloaded into systems for analysis and some prediction models.

PROPOSED SYSTEM:

In this proposed system I have also collected some of data's as a csv format, but in her model there is some data repositories have been missing and its difficult to collect the data for analysis . so here I have gone through process of web scraping using python 3 for my analysis.

CHAPTER-2

SYSTEM SPECIFICATIONS

SOFTWARE SPECIFICATIONS :

Jupyter Notebook:

Project Jupyter is a project and community whose goal is to "develop open-source software, open-standards, and services for interactive computing across dozens of programming languages".[2] It was spun off from IPython in 2014 by Fernando Pérez[3] and Brian Granger.[4] Project Jupyter's name is a reference to the three core programming languages supported by Jupyter, which are Julia, Python and R, and also a homage to Galileo's notebooks recording the discovery of the moons of Jupiter. Project Jupyter has developed and supported the interactive computing products Jupyter Notebook, JupyterHub, and JupyterLab. Jupyter is financially sponsored by NumFOCUS.

- ### Python 3:
  - Python 3.0 (also called "Python 3000" or "Py3K") was released on December 3, 2008.[9] It was designed to rectify fundamental design flaws in the language—the changes required could not be implemented while retaining full backwards compatibility with the 2.x series, which necessitated a new major version number. The guiding principle of Python 3 was: "reduce feature duplication by removing old ways of doing things".
  - Python 3.0 was developed with the same philosophy as in prior versions. However, as Python had accumulated new and redundant ways to program the same task, Python 3.0 had an emphasis on removing duplicative constructs and modules, in keeping

with "There should be one— and preferably only one —obvious way to do it".

- Nonetheless, Python 3.0 remained a multi-paradigm language. Coders could still follow object-oriented, structured, and functional programming paradigms, among others, but within such broad choices, the details were intended to be more obvious in Python 3.0 than they were in Python 2.x.

- **Ergast Database:**

  The Ergast Developer API is an experimental web service which provides a historical record of motor racing data for non-commercial purposes. Please read the terms and conditions of use. The API provides data for the Formula One series, from the beginning of the world championships in 1950.Source code for a wide range of projects using the API can be found on the Ergast API Topic Page on GitHub. Example applications are showcased in the Application Gallery.Non-programmers can query the database using the manual interface or download the database tables in CSV format for import into spreadsheets or analysis software.

- **Microsoft Power Bi**
  - **Power BI** is an interactive data visualization software product developed by Microsoft with primary focus on business intelligence.[1] It is part of the Microsoft Power Platform. Power BI is a collection of software services, apps, and connectors that work together to turn unrelated sources of data into coherent, visually immersive, and interactive insights. Data may be input by reading directly from a database, webpage, or structured files such as spreadsheets, CSV, XML, and JSON.

Project Domain *:*

So, this project is done by using Python 3 as developing language under Jupyter Notebook, an excellent Integrated Development Environment (IDE) for data science and machine learning projects. Data sets are downloaded as CSV files from Ergast Developers API , which is the archive of all Formula 1 data's since 1950's. so we can modify the data sets as per our convenient using Python 3 in Jupyter Notebook. So this projects focuses on three main agenda's first collection of datasets, analysing of dataset, machine learning modelling for collected and analysed dataset. And finally giving out the actual predicted results.

LITRATURE SURVEY

Sl .NO	**PAPER TITLE**	**AUTHOR NAME**	**YEAR**	**DISADVANTAGES**
1	F1 Race Predictor	• Veronica Nirgo	2019	➢ It is proposed design ➢ It has security challenges ➢ Data holes in repository

CHAPTER-3

IMPLEMENTATION AND TESTING

Data Collection:

1. So in this first agenda I'm going to explain how I have collected all the data from browser.
2. As I said Data Collection, its very important to collect required data from browser or internet or form any other third parties. So in this project we can extract data's in two forms, we can directly download the datasets from the browser and convert in to CSV files or we can go through Web Scraping method using Python 3 to extract all entities and data's.

DATAFRAME – 1:

So our first data frame is to collect the datasets of all the races, tracks, location, season and url information.so for my data mining as I have said ergast developer API and also official formula 1 website, both have the relevant data but we have more accuracy during collection.So for this dataframe I have collected the last 2021 and current ongoing season of formula 1 championship.

In [23]:
```
races
```

Out[23]:

	season	round	circuit_id	country	date	url
0	2021	1	bahrain	Bahrain	2021-03-28	http://en.wikipedia.org/wiki/2021_Bahrain_Gran...
1	2021	2	imola	Italy	2021-04-18	http://en.wikipedia.org/wiki/2021_Emilia_Romag...
2	2021	3	portimao	Portugal	2021-05-02	http://en.wikipedia.org/wiki/2021_Portuguese_G...
3	2021	4	catalunya	Spain	2021-05-09	http://en.wikipedia.org/wiki/2021_Spanish_Gran...
4	2021	5	monaco	Monaco	2021-05-23	http://en.wikipedia.org/wiki/2021_Monaco_Grand...
5	2021	6	BAK	Azerbaijan	2021-06-06	http://en.wikipedia.org/wiki/2021_Azerbaijan_G...
6	2021	7	ricard	France	2021-06-20	http://en.wikipedia.org/wiki/2021_French_Grand...
7	2021	8	red_bull_ring	Austria	2021-06-27	http://en.wikipedia.org/wiki/2021_Styrian_Gran...
8	2021	9	red_bull_ring	Austria	2021-07-04	http://en.wikipedia.org/wiki/2021_Austrian_Gra...
9	2021	10	silverstone	UK	2021-07-18	http://en.wikipedia.org/wiki/2021_British_Gran...
10	2021	11	hungaroring	Hungary	2021-08-01	http://en.wikipedia.org/wiki/2021_Hungarian_Gr...
11	2021	12	spa	Belgium	2021-08-29	http://en.wikipedia.org/wiki/2021_Belgian_Gran...
12	2021	13	zandvoort	Netherlands	2021-09-05	http://en.wikipedia.org/wiki/2021_Dutch_Grand_...
13	2021	14	monza	Italy	2021-09-12	http://en.wikipedia.org/wiki/2021_Italian_Gran...
14	2021	15	sochi	Russia	2021-09-26	http://en.wikipedia.org/wiki/2021_Russian_Gran...

DATAFRAME -2: RACE RESULTS

For my second dataframe I iterated through each year and each round of my races file to query the Ergast API and get information about all the drivers' results. I included features such as grid and finishing position of each driver, their teams, and other less relevant variables such as date of birth, nationality and finishing status.

In [26]:
```
results
```

Out[26]:

	season	round	circuit_id	driver	date_of_birth	nationality	constructor	grid	laps	time
0	2021	1	bahrain	hamilton	1985-01-07	British	mercedes	2	56	5523897.0
1	2021	1	bahrain	max_verstappen	1997-09-30	Dutch	red_bull	1	56	5524642.0
2	2021	1	bahrain	bottas	1989-08-28	Finnish	mercedes	3	56	5561280.0
3	2021	1	bahrain	norris	1999-11-13	British	mclaren	7	56	5570363.0
4	2021	1	bahrain	perez	1990-01-26	Mexican	red_bull	0	56	5575944.0
...	...	...	...	...	...	...	...	...	...	...
435	2021	22	yas_marina	latifi	1995-06-29	Canadian	williams	16	50	NaN
436	2021	22	yas_marina	giovinazzi	1993-12-14	Italian	alfa	14	33	NaN
437	2021	22	yas_marina	russell	1998-02-15	British	williams	17	26	NaN
438	2021	22	yas_marina	raikkonen	1979-10-17	Finnish	alfa	18	25	NaN
439	2021	22	yas_marina	mazepin	1999-03-02	Russian	haas	20	0	NaN

440 rows × 13 columns

DATAFRAME-3: DRIVER STANDINGS

Points are awarded during the Championship based on where drivers and teams finish the race. Only the first 10 drivers finishing are awarded points, with the winner receiving 25 points. The Ergast API provides the number of points, wins and the standing position of each driver and team throughout the Championship. Because the points are awarded after the race, I had to create a lookup function to shift the points from previous races within the same Championship.

```
driver_standings
```

	season	round	driver	driver_points	driver_wins	driver_pos
0	2021	1	hamilton	0.0	0.0	0.0
1	2021	1	max_verstappen	0.0	0.0	0.0
2	2021	1	bottas	0.0	0.0	0.0
3	2021	1	norris	0.0	0.0	0.0
4	2021	1	perez	0.0	0.0	0.0
...	...	...	...	...	...	...
445	2021	22	latifi	7.0	0.0	17.0
446	2021	22	giovinazzi	3.0	0.0	18.0
447	2021	22	mick_schumacher	0.0	0.0	19.0
448	2021	22	kubica	0.0	0.0	20.0
449	2021	22	mazepin	0.0	0.0	21.0

DATAFRAME-4: CONSTRUCTORS STANDINGS

The Constructors Championship was awarded for the first time in 1958 so there is no data prior to that year. The data mining process is the same as the driver standings', eventually applying the same lookup function to get the data before the race.

In [21]: `constructor_standings[]`

	season	round	constructor	constructor_points	constructor_wins	constructor_pos
0	2021	1	mercedes	0.0	0.0	0.0
1	2021	1	red_bull	0.0	0.0	0.0
2	2021	1	mclaren	0.0	0.0	0.0
3	2021	1	ferrari	0.0	0.0	0.0
4	2021	1	alphatauri	0.0	0.0	0.0
...	...	...	...	...	...	...
215	2021	22	alphatauri	120.0	0.0	6.0
216	2021	22	aston_martin	77.0	0.0	7.0
217	2021	22	williams	23.0	0.0	8.0
218	2021	22	alfa	13.0	0.0	9.0
219	2021	22	haas	0.0	0.0	10.0

DATAFRAME-5: QUALIFYING

Getting the qualifying time data was the trickiest part, mainly because the Ergast data repository has some holes in the data and because qualifying rules changed so much over the years. Since 2006, qualifying takes place on a Saturday afternoon in a three-stage “knockout” system where the cars try to set their fastest lap time. In the past, qualifying would only consist of one or two sessions, causing missing data in my dataframe. I decided to consider only the best qualifying time for each driver, regardless of how many qualifying sessions were held in that year. The best qualifying time is reflected in the grid position, so I will later calculate the cumulative difference in times between the first qualified car and the others, hoping that it might give me an indication of how much faster a car is compared to the other ones.

```
In [17]: qualifying_results
```

Out[17]:

	grid	driver_name	car	qualifying_time	season	round
0	1	Max Verstappen VER	Red Bull Racing Honda	1:28.997	2021	1
1	2	Lewis Hamilton HAM	Mercedes	1:29.385	2021	1
2	3	Valtteri Bottas BOT	Mercedes	1:29.586	2021	1
3	4	Charles Leclerc LEC	Ferrari	1:29.678	2021	1
4	5	Pierre Gasly GAS	AlphaTauri Honda	1:29.809	2021	1
...	...	...	...	...	...	...
14	15	Sebastian Vettel VET	Aston Martin Mercedes	1:24.305	2021	22
15	16	Nicholas Latifi LAT	Williams Mercedes	1:24.338	2021	22
16	17	George Russell RUS	Williams Mercedes	1:24.423	2021	22
17	18	Kimi Räikkönen RAI	Alfa Romeo Racing Ferrari	1:24.779	2021	22
18	19	Mick Schumacher MSC	Haas Ferrari	1:24.906	2021	22

435 rows × 6 columns

DATAFRAME-6: WEATHER

Weather in Formula 1 plays a significant role on the choice of tyres, on the drivers' performance and on the overall teams' strategy. I decided to iterate through the wikipedia links of each race appended in the races_df and scrape the weather forecast. Since the wikipedia pages do not have a consistent html structure I need to look into a few different tables, and even at that point I still have many missing values. However, I noticed that I can find the remaining information in the corresponding pages in a different language. I then used selenium to click on the Italian page for each link and append the missing weather data. Eventually, I created a dictionary to categorise the weather forecasts and map my results.

In [19]:
```
weather_info
```

Out[19]:

	season	round	circuit_id	weather	weather_warm	weather_cold	weather_dry	weather_wet	weather_cloudy
0	2021	1	bahrain	Clear, 21 °C (70 °F)	0	0	0	0	0
1	2021	2	imola	Wet at start, drying during race, 9 °C (48 °F)	0	0	0	1	0
2	2021	3	portimao	Light cloud	0	0	0	0	0
3	2021	4	catalunya	Cloudy	0	0	0	0	1
4	2021	5	monaco	Sunny	1	0	0	0	0
5	2021	6	BAK	Sunny	1	0	0	0	0
6	2021	7	ricard	Windy and partly cloudy. Ambient temperature: ...	0	0	0	0	0
7	2021	8	red_bull_ring	Partly cloudy. Ambient temperature: 25 to 29 °...	0	0	0	0	0
8	2021	9	red_bull_ring	Partly cloudy	0	0	0	0	1
9	2021	10	silverstone	Clear. Ambient: 29 to 31 °C (84 to 88 °F); Sur...	0	0	0	0	0
10	2021	11	hungaroring	Rain and sunny. Ambient: 26 to 29 °C (79 to 84...	0	0	0	1	0

DATA ANALYSIS:

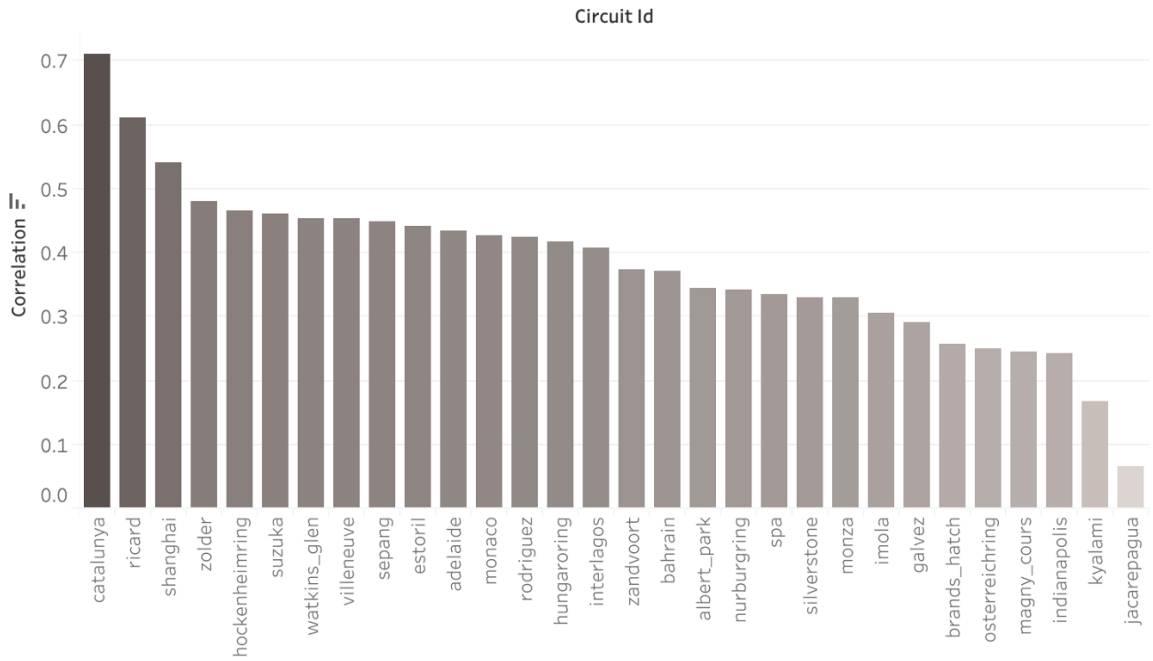
ANALYSIS-1:

The first drivers' world championship was held in 1950 at the British Grand Prix at Silverstone and comprised only seven races. The number of Grand Prix per season varied over the years, averaging 21 races in the latest seasons. The location of the races has also varied over time, depending on the suitability of the track and other financial reasons.

so the above world map image shows the location of race tracks in their country.

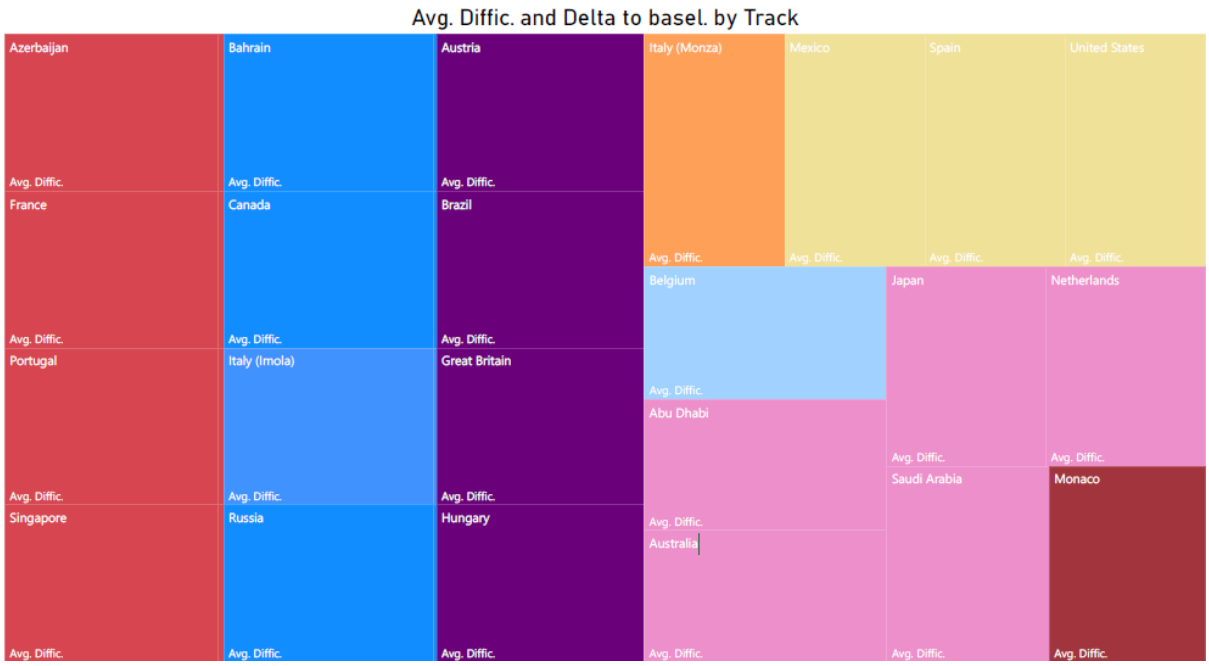
ANALYSIS-2: CORRELATION

During qualification sessions the drivers try to set their fastest time around the track and the grid position is determined by the drivers' best single lap, with the fastest on pole position. Starting on pole position is crucial in those circuits where overtaking is more difficult, in addition to having the advantage of staring a few meters ahead and on the normal racing line, which is usually cleaner and has more grip. The following graph shows the correlation between staring in pole position and winning the race in some of the most popular circuits.



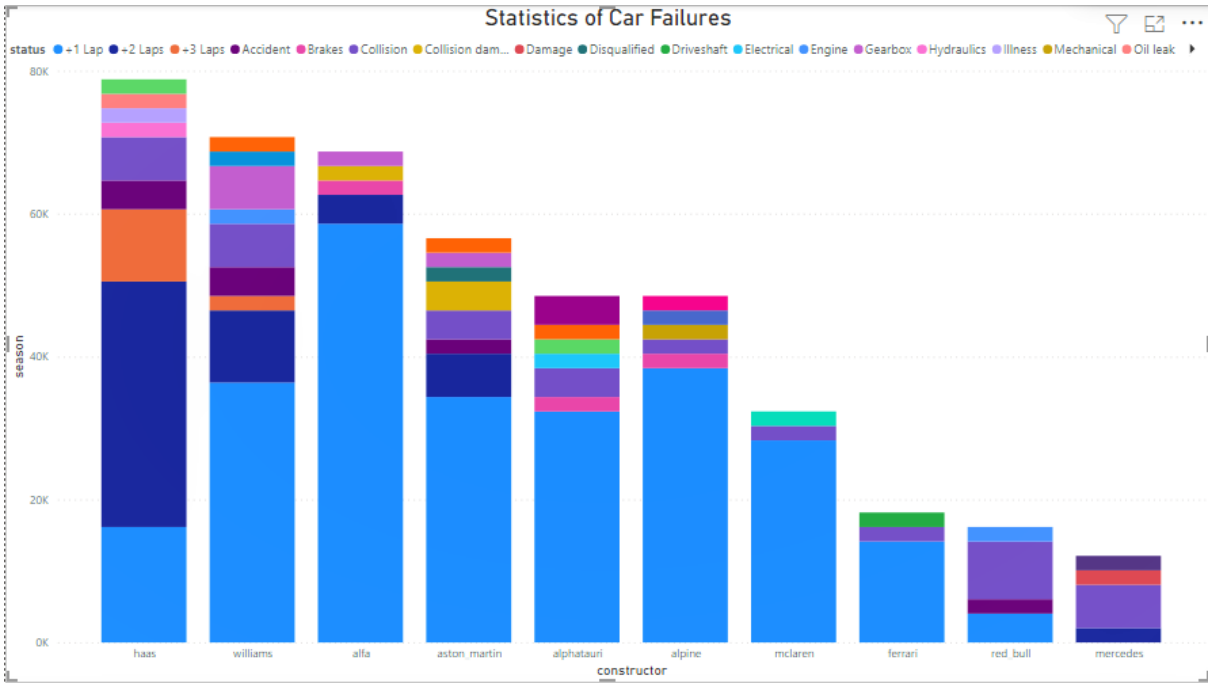
ANALYSIS-4: MOST DANGEROUS TRACKS

Some of the circuit layouts have been redesigned over the years to meet stricter safety requirements. Currently, most of the circuits are specifically constructed for competitions, in order to avoid long and fast straights or dangerous turns. However, some races are still held at street circuits, such as the Monaco Grand Prix, which is still in use mainly for its fame and history, despite not conforming with the latest strict measures. The following tree-map shows some of the most popular tracks by number of incidents or collisions.



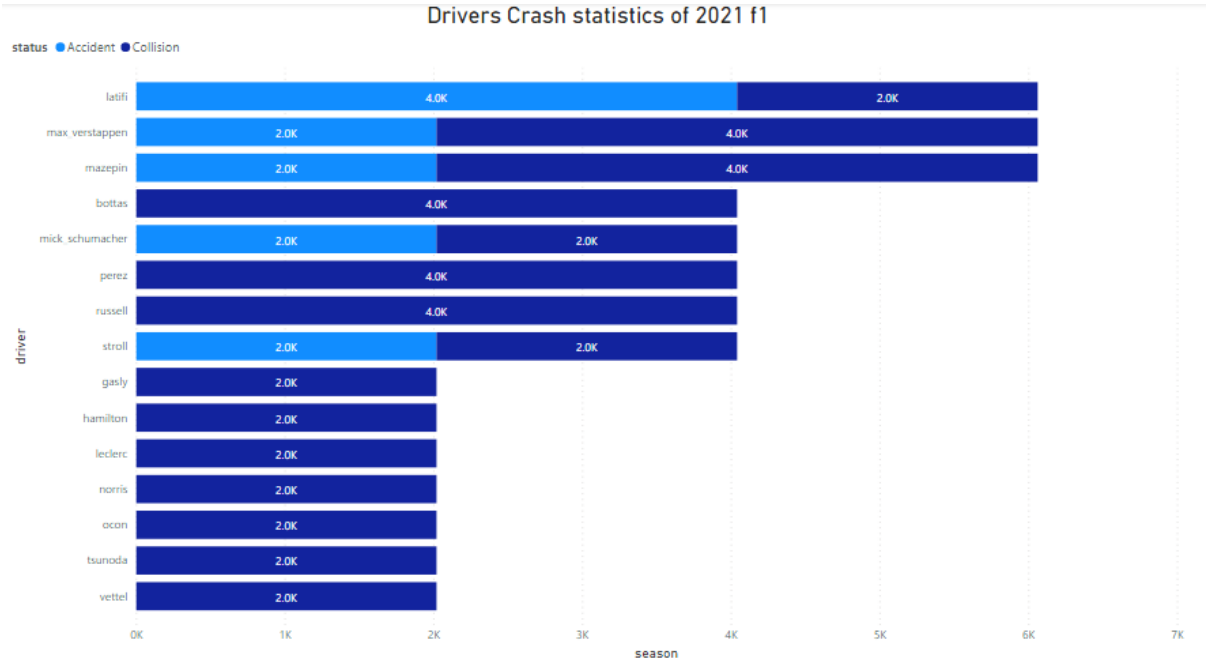
ANALYSIS-4: TEAM CAR FAILURES

The bar chart shows which teams that raced in the last few seasons experienced the highest number of car problems over the years, including engine failures, brakes, suspension or transmission problems



ANALYSIS-4: DRIVER'S CRASH

Cars in Formula 1 can reach top speeds of 375 km/h (233 mph) so crashes can ultimately terminate the race for the drivers. The chart below shows the ratio of crashes of some of the drivers



MACHINE LEARNING MODELLING

This last section will address the following topics: the **metrics** that I used to evaluate the best model, the process of **merging data** and eventually **Machine Learning modelling with neural networks**.

SUCCESS METRICES:

Precision score — percentage of correctly predicted winners in 2021 season

Odds comparison

DATA PREPARATION:

After collecting all the data, I end up with six different dataframe which I have to merge together using common keys. My final dataframe includes information of races, results, weather, driver and team standings and qualifying times from 2021.

REGRESSION OR CLASSIFICATION PROBLEM:

Since I want to predict the first place on the podium for each race in 2021, I can treat the target variable as either a regression or a classification.

When evaluating the precision score of a **regression**, I sort my predicted results in an ascending order and map the lowest value as the winner of the race. Eventually, I calculate the precision score between the actual values and predicted (mapped 1 and 0) and repeat for each race in 2021, until I get the percentage of correctly predicted races in that season.

This is what the prediction_df in the scoring function looks like for any race in 2021. The actual podium is mapped 0 and 1 (winner) and so are the predicted results after being sorted. In this case the model wrongly predicts Bottas as the winner of the race, so the model will have a score equal to 0.

OUTPUT:

driver	podium	proba_0	proba_1	actual	predicted
max_verstappen	1	0.651345	0.348655	1	1
hamilton	5	0.696430	0.303570	0	0
bottas	3	0.834249	0.165751	0	0
leclerc	2	0.879126	0.120874	0	0

SYSTEM CODE:

RACES:

```
import pandas as pd
import numpy as np
import requests
import bs4

races = {'season': [],
        'round': [],
        'circuit_id': [],
        'country': [],
        'date': [],
        'url': []
        }

for year in list(range(2021,2022)):

    url = 'https://ergast.com/api/f1/{}.json'
    r = requests.get(url.format(year))
    json = r.json()

    for item in json['MRData']['RaceTable']['Races']:
        try:
            races['season'].append(int(item['season']))
        except:
            races['season'].append(None)
```

```
        try:
            races['round'].append(int(item['round']))
        except:
            races['round'].append(None)

        try:
            races['circuit_id'].append(item['Circuit']['circuitId'])
        except:
            races['circuit_id'].append(None)

        try:
            races['country'].append(item['Circuit']['Location']['country'])
        except:
            races['country'].append(None)

        try:
            races['date'].append(item['date'])
        except:
            races['date'].append(None)

        try:
            races['url'].append(item['url'])
        except:
            races['url'].append(None)

races = pd.DataFrame(races)
```

RACE RESULTS:

```
rounds = []

for year in np.array(races.season.unique()):

rounds.append([year, list(races[races.season==year]["round"])])

#query api

results = {'season': [],
'round': [],
'circuit_id': [],
'driver': [],
'date_of_birth': [],
'nationality': [],
'constructor': [],
'grid': [],
'laps': [],
'time': [],
'points': [],
'status': [],
'podium': []
}

for n in list(range(len(rounds))):
```

```
for i in rounds[n][1]: #iterate to each rounds of year

url = 'https://ergast.com/api/f1/{}/{}/results.json'
r = requests.get(url.format(rounds[n][0], i))
json = r.json()

for item in json['MRData']['RaceTable']['Races'][0]['Results']:
try:
results['season'].append(int(json['MRData']['RaceTable']['Races'][0]['season']))
except:
results['season'].append(None)

try:
results['round'].append(int(json['MRData']['RaceTable']['Races'][0]['round']))
except:
results['round'].append(None)

try:
results['circuit_id'].append(json['MRData']['RaceTable']['Races'][0]['Circuit']['circuitId'])
except:
results['circuit_id'].append(None)

try:
results['driver'].append(item['Driver']['driverId'])
except:
results['driver'].append(None)

try:
results['date_of_birth'].append(item['Driver']['dateOfBirth'])
```

```
except:
results['date_of_birth'].append(None)

try:
results['nationality'].append(item['Driver']['nationality'])
except:
results['nationality'].append(None)

try:
results['constructor'].append(item['Constructor']['constructorId'])
except:
results['constructor'].append(None)

try:
results['grid'].append(int(item['grid']))
except:
results['grid'].append(None)

try:
results['laps'].append(int(item['laps']))
except:
results['laps'].append(None)

try:
results['time'].append(int(item['Time']['millis']))
except:
results['time'].append(None)

try:
results['points'].append(int(item['points']))
```

```
except:
results['points'].append(None)

try:
results['status'].append(item['status'])
except:
results['status'].append(None)

try:
results['podium'].append(int(item['position']))
except:
results['podium'].append(None)


results = pd.DataFrame(results)
```

DRIVER STANDINGS:

```
driver_standings = {
'season': [],
'round': [],
'driver': [],
'driver_points': [],
'driver_wins': [],
'driver_pos': [],
}

for n in list(range(len(rounds))):

for i in rounds[n][1]: #iterate to rounds of each year.
```

```
url = 'https://ergast.com/api/f1/{}/{}/driverStandings.json'
r = requests.get(url.format(rounds[n][0], i))
json = r.json();

for item in json['MRData']['StandingsTable']['StandingsLists'][0]['DriverStandings']:
try:
driver_standings['season'].append(int(json['MRData']['StandingsTable']['StandingsLists'][0]['season']))
except:
driver_standings['season'].append(None)

try:
driver_standings['round'].append(int(json['MRData']['StandingsTable']['StandingsLists'][0]['round']))
except:
driver_standings['round'].append(None)

try:
driver_standings['driver'].append(item['Driver']['driverId'])
except:
driver_standings['driver'].append(None)

try:
driver_standings['driver_points'].append(int(item['points']))
except:
driver_standings['driver_points'].append(None)

try:
driver_standings['driver_wins'].append(int(item['wins']))
except:
```

```
driver_standings['driver_wins'].append(None)


try:
driver_standings['driver_pos'].append(int(item['position']))
except:
driver_standings['driver_pos'].append(None)


driver_standings = pd.DataFrame(driver_standings)


def lookup (df, team, points):
df['lookup1'] = df.season.astype(str) + df[team] + df['round'].astype(str)
df['lookup2'] = df.season.astype(str) + df[team] + (df['round']-1).astype(str)
new_df = df.merge(df[['lookup1', points]], how = 'left',
left_on='lookup2',right_on='lookup1')
new_df.drop(['lookup1_x', 'lookup2', 'lookup1_y'], axis = 1, inplace = True)
new_df.rename(columns = {points+'_x': points+'_after_race', points+'_y': points},
inplace = True)
new_df[points].fillna(0, inplace = True)
return new_df


driver_standings = lookup(driver_standings, 'driver', 'driver_points')
driver_standings = lookup(driver_standings, 'driver', 'driver_wins')
driver_standings = lookup(driver_standings, 'driver', 'driver_pos')


driver_standings.drop(['driver_points_after_race', 'driver_wins_after_race',
'driver_pos_after_race'],
axis = 1, inplace = True)
```

CONSTRUCTOR STANDINGS:

```
constructor_standings = { 'season': [],
'round': [],
'constructor': [],
'constructor_points': [],
'constructor_wins': [],
'constructor_pos': []
}

#query api
for n in list(range(len(rounds))):

for i in rounds[n][1]:

url = 'https://ergast.com/api/f1/{}/{}/constructorStandings.json'
r = requests.get(url.format(rounds[n][0], i))
json = r.json()

for item in
json['MRData']['StandingsTable']['StandingsLists'][0]['ConstructorStandings']:
try:
constructor_standings['season'].append(int(json['MRData']['StandingsTable']['Standin
gsLists'][0]['season']))
except:
constructor_standings['season'].append(None)

try:
```

```
constructor_standings['round'].append(int(json['MRData']['StandingsTable']['StandingsLists'][0]['round']))
except:
constructor_standings['round'].append(None)

try:
constructor_standings['constructor'].append(item['Constructor']['constructorId'])
except:
constructor_standings['constructor'].append(None)

try:
constructor_standings['constructor_points'].append(int(item['points']))
except:
constructor_standings['constructor_points'].append(None)

try:
constructor_standings['constructor_wins'].append(int(item['wins']))
except:
constructor_standings['constructor_wins'].append(None)

try:
constructor_standings['constructor_pos'].append(int(item['position']))
except:
constructor_standings['constructor_pos'].append(None)

constructor_standings = pd.DataFrame(constructor_standings)

constructor_standings = lookup(constructor_standings, 'constructor', 'constructor_points')
constructor_standings = lookup(constructor_standings, 'constructor', 'constructor_wins')
```

```
constructor_standings = lookup(constructor_standings, 'constructor',
'constructor_pos')


constructor_standings.drop(['constructor_points_after_race',
'constructor_wins_after_race', 'constructor_pos_after_race']

, axis=1, inplace=True)
```

QUALIFYING:

```
qualifying_results = pd.DataFrame()


# Qualifying times are only available from 1983


for year in list(range(2021,2022)):

url = 'https://www.formula1.com/en/results.html/{}/races.html'

r = requests.get(url.format(year))

soup = bs4.BeautifulSoup(r.text, 'html.parser')


# find links to all circuits for a certain year


year_links = []

for page in soup.find_all('a', attrs = {'class':"resultsarchive-filter-item-link
FilterTrigger"}):

link = page.get('href')

if f'/en/results.html/{year}/races/' in link:

year_links.append(link)


# for each circuit, switch to the starting grid page and read table


year_df = pd.DataFrame()

new_url = 'https://www.formula1.com{}'

for n, link in list(enumerate(year_links)):
```

```
link = link.replace('race-result.html', 'starting-grid.html')
df = pd.read_html(new_url.format(link))
df = df[0]
df['season'] = year
df['round'] = n+1
for col in df:
if 'Unnamed' in col:
df.drop(col, axis = 1, inplace = True)

year_df = pd.concat([year_df, df])

# concatenate all tables from all years

qualifying_results = pd.concat([qualifying_results, year_df])

# rename columns

qualifying_results.rename(columns = {'Pos': 'grid', 'Driver': 'driver_name', 'Car': 'car',
'Time': 'qualifying_time'}, inplace = True)
# drop driver number column

qualifying_results.drop('No', axis = 1, inplace = True)
```

WEATHER:

```
from selenium import webdriver

weather = races.iloc[:,[0,1,2]]

info = []
```

```
# read wikipedia tables

for link in races.url:
try:
df = pd.read_html(link)[0]
if 'Weather' in list(df.iloc[:,0]):
n = list(df.iloc[:,0]).index('Weather')
info.append(df.iloc[n,1])
else:
df = pd.read_html(link)[1]
if 'Weather' in list(df.iloc[:,0]):
n = list(df.iloc[:,0]).index('Weather')
info.append(df.iloc[n,1])
else:
df = pd.read_html(link)[2]
if 'Weather' in list(df.iloc[:,0]):
n = list(df.iloc[:,0]).index('Weather')
info.append(df.iloc[n,1])
else:
df = pd.read_html(link)[3]
if 'Weather' in list(df.iloc[:,0]):
n = list(df.iloc[:,0]).index('Weather')
info.append(df.iloc[n,1])
else:
driver = webdriver.chrome()
driver.get(link)



except:
```

```
info.append('not found')

# append column with weather information to dataframe

weather['weather'] = info

# set up a dictionary to convert weather information into keywords

weather_dict = {'weather_warm': ['soleggiato', 'clear', 'warm', 'hot', 'sunny', 'fine',
'mild', 'sereno'],
'weather_cold': ['cold', 'fresh', 'chilly', 'cool'],
'weather_dry': ['dry', 'asciutto'],
'weather_wet': ['showers', 'wet', 'rain', 'pioggia', 'damp', 'thunderstorms', 'rainy'],
'weather_cloudy': ['overcast', 'nuvoloso', 'clouds', 'cloudy', 'grey', 'coperto']}

# map new df according to weather dictionary

weather_df = pd.DataFrame(columns = weather_dict.keys())
for col in weather_df:
weather_df[col] = weather['weather'].map(lambda x: 1 if any(i in weather_dict[col] for
i in x.lower().split()) else 0)

weather_info = pd.concat([weather, weather_df], axis = 1)
```

DATA PREPARATION:

```
df1 = pd.merge(races, weather, how='left',
on=['season', 'round', 'circuit_id']).drop(['country','weather'], axis = 1)
df2 = pd.merge(df1, results, how='left',
on=['season', 'round', 'circuit_id']).drop(['points', 'status', 'time'], axis = 1)
df3 = pd.merge(df2, driver_standings, how='left', on=['season', 'round', 'driver'])
```

```
df4 = pd.merge(df3, constructor_standings, how='left', on=['season', 'round',
'constructor'])


final_df = pd.merge(df4, qualifying_results, how='left', on=['season', 'round',
'grid']).drop(['driver_name', 'car'],
axis = 1)



# fill/drop nulls


for col in ['driver_points', 'driver_wins', 'driver_pos', 'constructor_points',
'constructor_wins' , 'constructor_pos']:
final_df[col].fillna(0, inplace = True)
final_df[col] = final_df[col].map(lambda x: int(x))


final_df.dropna(inplace = True )



# convert to boolean to save space


for col in ['weather_warm', 'weather_cold','weather_dry', 'weather_wet',
'weather_cloudy']:
final_df = final_df[col].map(lambda x : bool(x))


# calculate difference in qualifying times


final_df['qualifying_time'] = final_df.qualifying_time.map(lambda x: 0 if str(x) ==
'00.000'
else(float(str(x).split(':')[1]) +
(60 * float(str(x).split(':')[0])) if x != 0 else 0))
final_df = final_df[final_df['qualifying_time'] != 0]
```

```
final_df.sort_values(['season', 'round', 'grid'], inplace = True)

final_df['qualifying_time_diff'] = final_df.groupby(['season',
'round']).qualifying_time.diff()

final_df['qualifying_time'] = final_df.groupby(['season',

'round']).qualifying_time_diff.cumsum().fillna(0)

final_df.drop('qualifying_time_diff', axis = 1, inplace = True)



# get dummies


df_dum = pd.get_dummies(final_df, columns = ['circuit_id', 'nationality', 'constructor']
)


for col in df_dum.columns:

if 'nationality' in col and df_dum[col].sum() < 140:

df_dum.drop(col, axis = 1, inplace = True)


elif 'constructor' in col and df_dum[col].sum() < 140:

df_dum.drop(col, axis = 1, inplace = True)


elif 'circuit_id' in col and df_dum[col].sum() < 70:

df_dum.drop(col, axis = 1, inplace = True)


else:

pass
```

REGRESSION OR CLASSIFICATION:

1.REGRESSION:

```
def score_regression(model):

score = 0
```

```
for circuit in df[df.season == 2021]['round'].unique():

test = df[(df.season == 2021) & (df['round'] == circuit)]
X_test = test.drop(['driver', 'podium'], axis = 1)
y_test = test.podium

#scaling
X_test = pd.DataFrame(scaler.transform(X_test), columns = X_test.columns)

# make predictions
prediction_df = pd.DataFrame(model.predict(X_test), columns = ['results'])
prediction_df['podium'] = y_test.reset_index(drop = True)
prediction_df['actual'] = prediction_df.podium.map(lambda x: 1 if x == 1 else 0)
prediction_df.sort_values('results', ascending = True, inplace = True)
prediction_df.reset_index(inplace = True, drop = True)
prediction_df['predicted'] = prediction_df.index
prediction_df['predicted'] = prediction_df.predicted.map(lambda x: 1 if x == 0 else 0)

score += precision_score(prediction_df.actual, prediction_df.predicted)
model_score = score / df[df.season == 2021]['round'].unique().max()
return model_score
```

CLASSIFICATION:

```
def score_classification(model):
score = 0
for circuit in df[df.season == 2021]['round'].unique():

test = df[(df.season == 2021) & (df['round'] == circuit)]
X_test = test.drop(['driver', 'podium'], axis = 1)
y_test = test.podium
```

```
#scaling
X_test = pd.DataFrame(scaler.transform(X_test), columns = X_test.columns)

# make predictions
prediction_df = pd.DataFrame(model.predict_proba(X_test), columns = ['proba_0', 'proba_1'])
prediction_df['actual'] = y_test.reset_index(drop = True)
prediction_df.sort_values('proba_1', ascending = False, inplace = True)
prediction_df.reset_index(inplace = True, drop = True)
prediction_df['predicted'] = prediction_df.index
prediction_df['predicted'] = prediction_df.predicted.map(lambda x: 1 if x == 0 else 0)

score += precision_score(prediction_df.actual, prediction_df.predicted)

model_score = score / df[df.season == 2021]['round'].unique().max()
return model_score
```

MACHINE LEARNING:

```
df = data.copy()

#train split

train = df[df.season == 2021]
X_train = train.drop(['driver', 'podium'], axis = 1)
y_train = train.podium

scaler = StandardScaler()
X_train = pd.DataFrame(scaler.fit_transform(X_train), columns = X_train.columns)
```

```
# Linear Regression

params={'fit_intercept': ['True', 'False']}

for fit_intercept in params['fit_intercept']:
model_params = (fit_intercept)
model = LinearRegression(fit_intercept = fit_intercept)
model.fit(X_train, y_train)

model_score = score_regression(model)

comparison_dict['model'].append('linear_regression')
comparison_dict['params'].append(model_params)
comparison_dict['score'].append(model_score)



# Random Forest Regressor

params={'criterion': ['mse'],
'max_features': [0.8, 'auto', None],
'max_depth': list(np.linspace(5, 55, 26)) + [None]}

for criterion in params['criterion']:
for max_features in params['max_features']:
for max_depth in params['max_depth']:
model_params = (criterion, max_features, max_depth)
model = RandomForestRegressor(criterion = criterion,
max_features = max_features, max_depth = max_depth, random_state = 1)
model.fit(X_train, y_train)
```

```
model_score = score_regression(model)

comparison_dict['model'].append('random_forest_regressor')
comparison_dict['params'].append(model_params)
comparison_dict['score'].append(model_score)



# Support Vector Machines

params={'gamma': np.logspace(-4, -1, 10),
'C': np.logspace(-2, 1, 10),
'kernel': ['linear', 'poly', 'rbf', 'sigmoid']}

for gamma in params['gamma']:
for c in params['C']:
for kernel in params['kernel']:
model_params = (gamma, c, kernel)
model = svm.SVR(gamma = gamma, C = c, kernel = kernel)
model.fit(X_train, y_train)

model_score = score_regression(model)

comparison_dict['model'].append('svm_regressor')
comparison_dict['params'].append(model_params)
comparison_dict['score'].append(model_score)



# Neural network

params={'hidden_layer_sizes': [(80,20,40,5), (75,30,50,10,3)],
```

```
'activation': ['identity', 'relu','logistic', 'tanh',],
'solver': ['lbfgs','sgd', 'adam'],
'alpha': np.logspace(-4,1,20)}

for hidden_layer_sizes in params['hidden_layer_sizes']:
for activation in params['activation']:
for solver in params['solver']:
for alpha in params['alpha']:
model_params = (hidden_layer_sizes, activation, solver, alpha )
model = MLPRegressor(hidden_layer_sizes = hidden_layer_sizes,
activation = activation, solver = solver, alpha = alpha, random_state = 1)
model.fit(X_train, y_train)

model_score = score_regression(model)

comparison_dict['model'].append('nn_regressor')
comparison_dict['params'].append(model_params)
comparison_dict['score'].append(model_score)

#print best models
pd.DataFrame(comparison_dict).groupby('model')['score'].max()
```

CLASSIFICATION:

```
df = data.copy()
df.podium = df.podium.map(lambda x: 1 if x == 1 else 0)

#split train

train = df[df.season <2019]
X_train = train.drop(['driver', 'podium'], axis = 1)
```

```
y_train = train.podium

scaler = StandardScaler()
X_train = pd.DataFrame(scaler.fit_transform(X_train), columns = X_train.columns)



# gridsearch dictionary

comparison_dict ={'model':[],
'params': [],
'score': []}

# Logistic Regression

params={'penalty': ['l1', 'l2'],
'solver': ['saga', 'liblinear'],
'C': np.logspace(-3,1,20)}

for penalty in params['penalty']:
for solver in params['solver']:
for c in params['C']:
model_params = (penalty, solver, c)
model = LogisticRegression(penalty = penalty, solver = solver, C = c, max_iter =
10000)
model.fit(X_train, y_train)

model_score = score_classification(model)

comparison_dict['model'].append('logistic_regression')
comparison_dict['params'].append(model_params)
comparison_dict['score'].append(model_score)
```

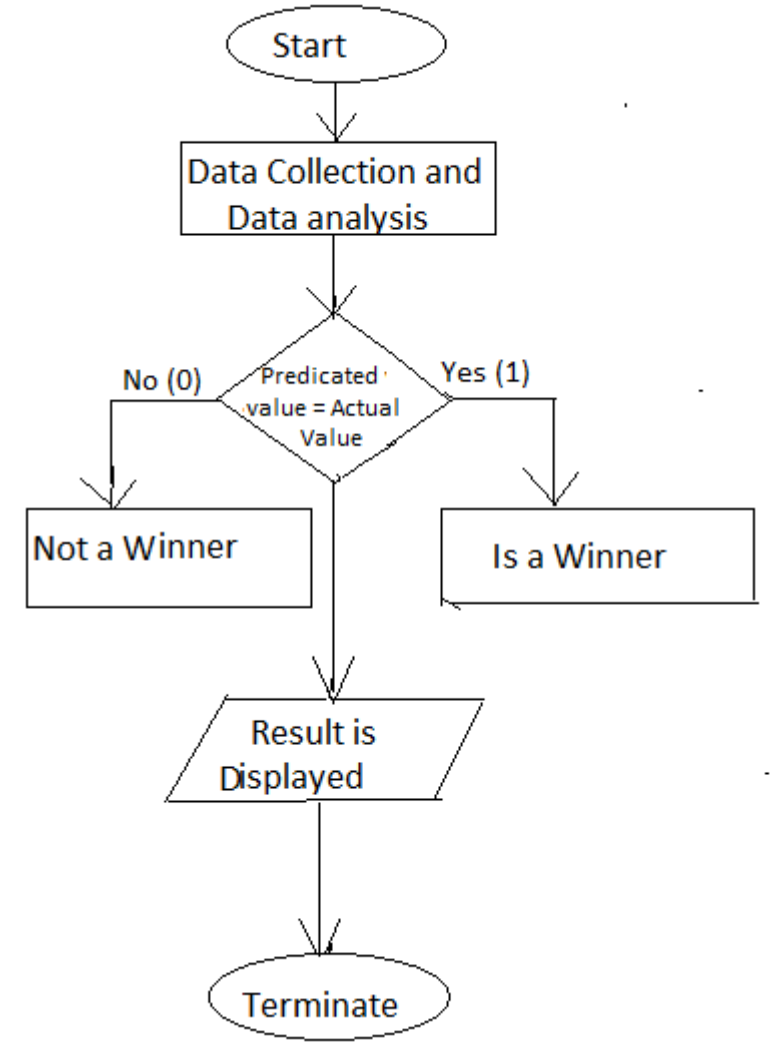
```
# Random Forest Classifier

params={'criterion': ['gini', 'entropy'],
'max_features': [0.8, 'auto', None],
'max_depth': list(np.linspace(5, 55, 26)) + [None]}

for criterion in params['criterion']:
for max_features in params['max_features']:
for max_depth in params['max_depth']:
model_params = (criterion, max_features, max_depth)
model = RandomForestClassifier(criterion = criterion, max_features = max_features,
max_depth = max_depth)
model.fit(X_train, y_train)

model_score = score_classification(model)

comparison_dict['model'].append('random_forest_classifier')
comparison_dict['params'].append(model_params)
comparison_dict['score'].append(model_score)

# Support Vector Machines

params={'gamma': np.logspace(-4, -1, 20),
'C': np.logspace(-2, 1, 20),
'kernel': ['linear', 'poly', 'rbf', 'sigmoid']}

for gamma in params['gamma']:
for c in params['C']:
for kernel in params['kernel']:
model_params = (gamma, c, kernel)
```

```
model = svm.SVC(probability = True, gamma = gamma, C = c, kernel = kernel )
model.fit(X_train, y_train)

model_score = score_classification(model)

comparison_dict['model'].append('svm_classifier')
comparison_dict['params'].append(model_params)
comparison_dict['score'].append(model_score)

# Neural network

params={'hidden_layer_sizes': [(80,20,40,5), (75,25,50,10)],
'activation': ['identity', 'logistic', 'tanh', 'relu'],
'solver': ['lbfgs', 'sgd', 'adam', 'logistic'],
'alpha': np.logspace(-4,2,20)}

for hidden_layer_sizes in params['hidden_layer_sizes']:
for activation in params['activation']:
for solver in params['solver']:
for alpha in params['alpha']:
model_params = (hidden_layer_sizes, activation, solver, alpha )
model = MLPClassifier(hidden_layer_sizes = hidden_layer_sizes,
activation = activation, solver = solver, alpha = alpha, random_state = 1)
model.fit(X_train, y_train)

model_score = score_classification(model)

comparison_dict['model'].append('neural_network_classifier')
comparison_dict['params'].append(model_params)
comparison_dict['score'].append(model_score)
```

CHAPTER-4:

MODULE DESCRIPTION

DATA FLOW DIAGRAM:



TASK:

Identifying the analytical problems to the data set which have collected in order to gives out the best prediction.

LIMITATIONS:

1. Incompletion of Data.
2. Don’t always provide the accurate information.
3. Data collected from different sources can vary in quality and format.

4. Computers and algorithms fail to consider variables.
5. Mastering is near to impossible.

CHAPTER - 5

RESULTS AND DISCUSSION

APPLICATIONS:

1. Fraud detection
2. Health Care
3. Advanced Image Recognition.
4. Speech Recognition.
5. Internet Search.

ADVANTAGES:

1.Makes data's as better to understand.

2.They not only analyse the data but also improves its quality.

3.Understandable for teams for their strategic decision making during race.

CONCLUSION AND FUTURE ENHANCEMENT:

CONCLUSION:

1. Success is the better decision making. As the data size has grown to incredible proportions, these data driven decision are based often on quantitative models created using a typical closed loop process.
2. Data Science is using data to make better decisions with analysis for casuality, and machine leaning for prediction.

FUTURE ENHANCEMENT:

1.Data Science's future getting bright is its hight end demand because of digitalization.