**Project Report – Analysis of Formula 1 Dataset**

Name: Atharva Khaire

BUID: U15258568

Course: Data Science with Python (CS 677 A1)

**Note: To run the code, just run all cells and confirm if the datasets are in the correct location.**

**Libraries Required: Pandas, Numpy, Sklearn, Matplotlib**

**Brief Introduction about Formula 1**

* 10 Constructors/Teams
* 20 Drivers (2 per team)
* 3-Day Race Weekend
* 1st day – FP1, FP2 (practice sessions)
* 2nd day – FP3, Qualifying (Q1, Q2, Q3)
* 3rd day – Race
* Note: For some races FP3 is replaced by sprint race which sets up the grid for the main race on Sunday.

**Objectives**

* Car Performance over the years (Average net gain/loss in time on a track)
* Lap Prediction (based on previous laps, the number of laps changing according to a certain window)
* Lap Prediction (based on previous years data)
* Constructor Comparison for the 2022 season
* Driver Comparison for the 2022 season
* Fastest Track (based on Lap time)
* How error-prone each driver is?
* Overtake Rate per race
* Driver Ratings (based on Delta, Driver Error, Overtakes)

**Car Performance over the years (Average net gain/loss in time on a track)**

Over the years, the cars have undergone massive changes in their build. Be it raw speed or cornering, they have certainly become faster and more efficient. To analyze this, I decided to pick two of the oldest tracks that have held races. Monaco and Monza both have been the face of Formula 1. Not a year goes by where these races fail to entertain the drivers and the viewers. I chose Monaco because it’s a street circuit. Therefore, it has only corners and only one straight making it difficult for drivers to keep the car on the track. This track demands heavy downforce which basically means the cars can take maximum speed through corners without slipping away from the driver’s control. I took the fastest lap times for this track from the year 1996-2022 and put them in a table for comparison.

# Storing all the monaco races lap times by year

monaco\_IDs = races[races['circuitId'] == 6].sort\_values(by='year')['raceId']

monaco\_IDs

Text

Description automatically generated with low confidence

Fetching all the raceIds for Monaco’s CircuitId.

# Sorting the fastest laps according to year and calculating the delta to the 1996 fastest lap

fastestlaps\_monaco = pd.DataFrame(columns = ['raceId','year','time','milliseconds','delta'])

for race in monaco\_IDs:

temp = lap\_times[lap\_times['raceId'] == race]

if(temp.empty):

continue

else:

temp = temp[temp['milliseconds'] == min(temp['milliseconds'])]

year = races[races['raceId'] == int(temp['raceId'])]['year']

delta = temp['milliseconds'] - 85205

fastestlaps\_monaco.loc[len(fastestlaps\_monaco)] = [int(temp['raceId']),int(year),str(temp['time'].values),int(temp['milliseconds']),float(delta/1000)]

fastestlaps\_monaco

Calendar

Description automatically generated with medium confidence

Delta is basically the difference in time with the 1996 lap time. As you can see, over the years, the cars have become approximately 10 seconds faster than the 1996 lap time. This explains how the turnings of the cars have become more faster.

Now we do the same thing for Monza. Monza was chosen because of its characteristic of forcing the cars to reach their maximum top speed in a straight line which for today’s cars is approximately 370+ km/hr.

# Storing all the monza lap times by year

monza\_IDs = races[races['circuitId'] == 14].sort\_values(by='year')['raceId']

monza\_IDs

Text

Description automatically generated with medium confidence

Fetching all the raceIds for Monza’s CircuitId.

# Sorting the fastest laps according to year and calculating the delta to the 1996 fastest lap

fastestlaps\_monza = pd.DataFrame(columns = ['raceId','year','time','milliseconds','delta'])

for race in monza\_IDs:

temp = lap\_times[lap\_times['raceId'] == race]

if(temp.empty):

continue

else:

temp = temp[temp['milliseconds'] == min(temp['milliseconds'])]

year = races[races['raceId'] == int(temp['raceId'])]['year']

delta = temp['milliseconds'] - 86110

fastestlaps\_monza.loc[len(fastestlaps\_monza)] = [int(temp['raceId']),int(year),str(temp['time'].values),int(temp['milliseconds']),float(delta/1000)]

fastestlaps\_monza

Calendar

Description automatically generated

Here, we can clearly observe that the raw speed of cars haven’t improved as much as the cornering speeds. In fact, the speeds of the cars keep on changing. For some years, they are faster than the previous years and for some slower. This is because the FIA keeps on imposing engine limitations each year due to safety reasons.

**Lap Prediction (based on previous laps, the number of laps changing according to a certain window)**

One of the most challenging problems in Formula 1 history has been predicting lap times. I thought of my own approach which was inspired by one of our assignments. I decided to take the training set for each lap having the lap times of previous laps and this number of previous laps range from 5 to 20. The models that I used were Linear Regression, Polyfit with Logarithm and Polyfit with degree 3. Based on this we find the optimal window by calculating the Root Mean Squared error. The one with the least error was used for prediction.

# Predicting lap times based on previous laps with a window of laps from 5 to 20

# Model: Linear Regression

from sklearn.linear\_model import LinearRegression

table\_stat = pd.DataFrame(columns = ['w','error'])

w = 5

while(w<=20):

laps = lap\_times[(lap\_times['raceId'] == 9) & (lap\_times['driverId'] == 1)]

Ypred = []

Y = laps.iloc[w:len(laps)]['milliseconds']

for i in range(w,len(laps)-1):

X = np.array(laps.iloc[i-w:i]['lap']).reshape(-1, 1)

Y = np.array(laps.iloc[i-w:i]['milliseconds']).reshape(-1, 1)

model = LinearRegression()

model.fit(X,Y)

lap\_pred = model.predict(np.array(laps.loc[i+laps.first\_valid\_index(),'lap']).reshape(-1,1))

lap\_pred = lap\_pred[0][0]

Ypred.append(lap\_pred)

RMSE = (np.mean(np.sum(Y - Ypred))\*\*2)\*\*1/2

table\_stat.loc[len(table\_stat)] = [w,RMSE]

w = w + 1

table\_stat

Calendar

Description automatically generated

The driver chosen for this was Lewis Hamilton and the race chosen was the German Grand Prix (Nurburgring). As you can see the least error here was shown by a window of 10 laps which in this case will be W optimal.

# Predicting lap times based on previous laps with a window of laps from 5 to 20

# Model: Polyfit with Log

table\_stat = pd.DataFrame(columns = ['w','error'])

w = 5

while(w<=20):

laps = lap\_times[(lap\_times['raceId'] == 9) & (lap\_times['driverId'] == 1)]

Ypred = []

Y = laps.iloc[w:len(laps)]['milliseconds']

for i in range(w,len(laps)-1):

X = np.array(laps.iloc[i-w:i]['lap'])

Y = np.array(laps.iloc[i-w:i]['milliseconds'])

weights = np.polyfit(np.log(X),np.log(Y),1)

model = np.poly1d(weights)

lap\_pred = model(np.log(np.array(laps.loc[i+laps.first\_valid\_index(),'lap']).reshape(-1,1)))

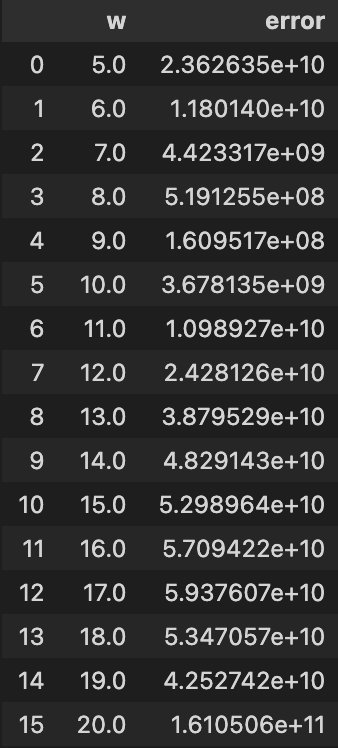
Ypred.append(lap\_pred)

RMSE = (np.mean(np.sum(Y - np.exp(Ypred)))\*\*2)\*\*1/2

table\_stat.loc[len(table\_stat)] = [w,RMSE]

w = w + 1

table\_stat



As you can see, the errors are far too higher than linear regression so I decided to drop this model from any further analysis.

# Predicting lap times based on previous laps with a window of laps from 5 to 20

# Model: Polyfit with degree 3

table\_stat = pd.DataFrame(columns = ['w','error'])

w = 5

while(w<=20):

laps = lap\_times[(lap\_times['raceId'] == 9) & (lap\_times['driverId'] == 1)]

Ypred = []

Y = laps.iloc[w:len(laps)]['milliseconds']

for i in range(w,len(laps)-1):

X = np.array(laps.iloc[i-w:i]['lap'])

Y = np.array(laps.iloc[i-w:i]['milliseconds'])

weights = np.polyfit(X,Y,3)

model = np.poly1d(weights)

lap\_pred = model(np.array(laps.loc[i+laps.first\_valid\_index(),'lap']).reshape(-1,1))

Ypred.append(lap\_pred)

RMSE = (np.mean(np.sum(Y - Ypred))\*\*2)\*\*1/2

table\_stat.loc[len(table\_stat)] = [w,RMSE]

w = w + 1

table\_stat

Calendar

Description automatically generated

Here the error for W = 11 is comparable to Linear regression and therefore I decided to take this into consideration along with Linear regression.

# Plotting Actual to predicted lap times using optimal window of laps and best model

# W = 11 , Model = Polyfit with degree 3

import matplotlib.pyplot as plt

laps = lap\_times[(lap\_times['raceId'] == 9) & (lap\_times['driverId'] == 1)]

Ypred = []

w = 11

Xmain = laps.iloc[w+1:len(laps)]['lap']

Ymain = laps.iloc[w+1:len(laps)]['milliseconds']

for i in range(w,len(laps)-1):

X = np.array(laps.iloc[i-w:i]['lap'])

Y = np.array(laps.iloc[i-w:i]['milliseconds'])

weights = np.polyfit(X,Y,3)

model = np.poly1d(weights)

lap\_pred = float(model(np.array(laps.loc[i+laps.first\_valid\_index(),'lap']).reshape(-1,1)))

Ypred.append(lap\_pred)

print(Ypred)

plt.plot(Xmain,Ymain,c='blue')

plt.plot(Xmain,Ypred,c='orange')

plt.show()

Chart, line chart

Description automatically generated

The above plot shows the actual lap times (Blue) along with the predicted ones (Orange). Model used here is Polyfit with degree 3 and W optimal = 11.

# Plotting Actual to predicted lap times using optimal window of laps and best model

# W = 10 , Model = Linear Regression

import matplotlib.pyplot as plt

laps = lap\_times[(lap\_times['raceId'] == 9) & (lap\_times['driverId'] == 1)]

Ypred = []

w = 10

Xmain = laps.iloc[w+1:len(laps)]['lap']

Ymain = laps.iloc[w+1:len(laps)]['milliseconds']

for i in range(w,len(laps)-1):

X = np.array(laps.iloc[i-w:i]['lap']).reshape(-1,1)

Y = np.array(laps.iloc[i-w:i]['milliseconds']).reshape(-1,1)

model = LinearRegression()

model.fit(X,Y)

lap\_pred = model.predict(np.array(laps.loc[i+laps.first\_valid\_index(),'lap']).reshape(-1,1))

lap\_pred = lap\_pred[0][0]

Ypred.append(lap\_pred)

print(Ypred)

plt.plot(Xmain,Ymain,c='blue')

plt.plot(Xmain,Ypred,c='orange')

plt.show()

Chart, line chart

Description automatically generated

The above plot shows the actual lap times (Blue) along with the predicted ones (Orange). Model used here is Linear Regression and W optimal = 10.

**Lap Prediction (based on previous years data)**

The purpose of this objective was to observe how previous year lap times would be helpful in predicting current lap times. As proved earlier, the cars keep on changing every year and so does their lap times. Therefore, I decided to choose two years where the cars didn’t undergo many changes. Such two years where 2011 and 2012. Now, I wanted to try something different for prediction. So I chose Lasso Regression whose main property is shrinkage. Shrinkage is the process where the data is made to be centered around the mean. Also, one of the prominent features of this model is it does feature selection on its own.

# Splitting the lap times for driver no. 20 and Indian GP into training and test set

driver\_Id = 20

circuit\_Id = 68

race\_Id\_2011 = races[(races['year'] == 2011) & (races['circuitId'] == circuit\_Id)]['raceId'].values[0]

race\_Id\_2012 = races[(races['year'] == 2012) & (races['circuitId'] == circuit\_Id)]['raceId'].values[0]

Xtrain = np.array(lap\_times[(lap\_times['raceId'] == race\_Id\_2011) & (lap\_times['driverId'] == driver\_Id)]['lap']).reshape(-1,1)

Ytrain = np.array(lap\_times[(lap\_times['raceId'] == race\_Id\_2011) & (lap\_times['driverId'] == driver\_Id)]['milliseconds']).reshape(-1,1)

Xtest = np.array(lap\_times[(lap\_times['raceId'] == race\_Id\_2012) & (lap\_times['driverId'] == driver\_Id)]['lap']).reshape(-1,1)

Ytest = np.array(lap\_times[(lap\_times['raceId'] == race\_Id\_2012) & (lap\_times['driverId'] == driver\_Id)]['milliseconds']).reshape(-1,1)

Xtrain,Ytrain

The above code subsets the lap times for the year 2011 into training set and 2012 into test set. The driver chosen here is Sebastian Vettel and the race chosen is the Indian Grand Prix (Buddh International Circuit).

# Predicting lap times using previous years lap times using Lasso Regression

from sklearn.linear\_model import Lasso

stat = pd.DataFrame(columns = ['Actual','Predicted'])

model = Lasso(alpha=0.1)

model.fit(Xtrain,Ytrain)

Ypred = (model.predict(Xtest)).reshape(len(Ytest),1)

for i in range(0,len(Ytest)):

stat.loc[i,'Actual'] = Ytest[i]

stat.loc[i,'Predicted'] = Ypred[i]

stat

Graphical user interface, calendar

Description automatically generated with medium confidence

# Plotting actual to predicted lap times

plt.plot(Xtest,Ytest,c='blue')

plt.plot(Xtest,Ypred,c='orange')

plt.show()

Chart, line chart

Description automatically generated

As you can see, the model fits perfectly with an average delta error of only -0.5956999999999961. The bump in the actual values is because of the time lost during pitstop. If we disregard that the error will be less than 200 milliseconds.

**Constructor Comparison for the 2022 season**

Now that the prediction part was done, I wanted to statistically analyze the 2022 season. Constructor comparison is a key piece of information which the FIA uses to bring the cars closer in terms of lap time every year in order to ensure fair competition. Yet the cars haven’t been brought to equal capacity.

# Constructor/Team comparison based on fastest lap times of the season

driver\_Ids = [844,1,846,4,842,20,822,825,848]

constructor\_names = ['Ferrari','Mercedes','Mclaren','Alpine','Alpha Tauri','Aston Martin','Alfa Romeo','Haas','Williams']

race\_Id = 1082

table\_stat = pd.DataFrame(columns = ['Constructor','delta'])

benchmark = lap\_times[(lap\_times['driverId'] == 830) & (lap\_times['raceId'] == race\_Id)]

benchmark\_lap = benchmark[benchmark['milliseconds'] == min(benchmark['milliseconds'])]['milliseconds'].values[0]

delta = (benchmark\_lap - benchmark\_lap)/1000

table\_stat.loc[len(table\_stat)] = ['Red Bull',delta]

for i in range(0,len(driver\_Ids)):

temp = lap\_times[(lap\_times['driverId'] == driver\_Ids[i]) & (lap\_times['raceId'] == race\_Id)]

temp\_lap = temp[temp['milliseconds'] == min(temp['milliseconds'])]['milliseconds'].values[0]

delta = (temp\_lap - benchmark\_lap)/1000

table\_stat.loc[len(table\_stat)] = [constructor\_names[i],delta]

table\_stat

A picture containing text, scoreboard, screenshot

Description automatically generated

Here we can see that the fastest car was Red bull and the slowest one was Haas with Haas being 2.207 seconds slower per lap which over a 50 lap race would be a minute and 40 seconds slower. Basically, every race Red bull cars were able to lap them at least once.

# Plotting Constructor/Team performance relative to the fastest team (Red Bull)

import matplotlib.pyplot as plt

plt.style.use('ggplot')

plt.barh(table\_stat['Constructor'],table\_stat['delta'],color = ['#0600EF','#DC0000','#00D2BE','#FF8700','#0090FF','#2B4562','#006F62','#900000','#FFFFFF','#005AFF'])

plt.show()

Chart, bar chart

Description automatically generated

The more the value, the slower was the team.

**Driver Comparison for the 2022 season**

Another key piece of information is the driver comparison. This information is used by teams to figure which driver would be a good fit for the team. Now unfortunately this metric also take into account the car performance therefore it is pretty difficult to actually measure a driver’s skill unless they are given an equal performance car but that hasn’t happened yet.

# Driver comparison based on fastest lap times of the season

race\_Id = [1081,1082,1084,1086]

year = 2022

driver\_Ids = [830,815,844,832,1,847,846,817,4,839,842,852,20,840,822,855,825,854,848,849]

driver\_names = []

lap\_stat = pd.DataFrame(columns = ['raceId','driver\_Id','drivername','milliseconds'])

pace\_stat = pd.DataFrame(columns = ['drivername','delta'])

for i in driver\_Ids:

driver\_names.append(drivers[drivers['driverId'] == i]['driverRef'].values[0])

for race in range(0,len(race\_Id)):

for driver in range(0,len(driver\_Ids)):

temp = lap\_times[(lap\_times['driverId'] == driver\_Ids[driver]) & (lap\_times['raceId'] == race\_Id[race])]

temp\_lap = temp[temp['milliseconds'] == min(temp['milliseconds'])]['milliseconds'].values[0]

lap\_stat.loc[len(lap\_stat)] = [race\_Id[race],driver\_Ids[driver],driver\_names[driver],temp\_lap]

for driver in range(0,len(driver\_Ids)):

avg\_lap = np.mean(lap\_stat[lap\_stat['driver\_Id'] == driver\_Ids[driver]]['milliseconds'].values[0])

pace\_stat.loc[len(pace\_stat)] = [driver\_names[driver],avg\_lap]

pace\_stat['delta'] = (pace\_stat['delta'] - min(pace\_stat['delta']))/1000

pace\_stat



As you can see, the fastest driver was Sergio Perez from Red Bull while the slowest was Nicholas Latifi with him being 3.5 seconds slower than Perez.

# Plotting Driver performance relative to the fastest Driver (Sergio Perez)

import matplotlib.pyplot as plt

plt.style.use('ggplot')

plt.barh(pace\_stat['drivername'],pace\_stat['delta'],color = ['#0600EF','#0600EF','#DC0000','#DC0000','#00D2BE','#00D2BE','#FF8700','#FF8700','#0090FF','#0090FF','#2B4562','#2B4562','#006F62','#006F62','#900000','#900000','#FFFFFF','#FFFFFF','#005AFF','#005AFF'])

plt.show()

Chart, bar chart

Description automatically generated

**Fastest Track (based on Lap time)**

Now this metric isn’t really the best method to find the fastest tracks but certainly can be used by teams to correctly predict how much fuel they need to put in the car.

# Fastest Tracks based on lap time for the 2018 season

races\_2018 = races[races['year'] == 2018]['raceId']

track = pd.DataFrame(columns = ['Track','Lap-Time','milliseconds'])

for race in races\_2018:

laps = lap\_times[(lap\_times['driverId'] == 830) & (lap\_times['raceId'] == race)]['milliseconds']

fastest\_lap = min(laps)

lap\_time = lap\_times[(lap\_times['driverId'] == 830) & (lap\_times['raceId'] == race) & (lap\_times['milliseconds'] == fastest\_lap)]['time']

track.loc[len(track)] = [races[races['raceId'] == race]['name'].values[0],lap\_time.values[0],fastest\_lap]

track = track.sort\_values(by = 'milliseconds')

track

Graphical user interface, calendar

Description automatically generated

**How error-prone each driver is?**

To better understand a driver’s skill, it is important to know how careful a driver is while racing. Therefore I decided to analyze the number of self-inflicted accidents and collisions with other drivers.

# Calculating Driver error percentage based on accidents and collisions for the 2022 season

error\_stat = pd.DataFrame(columns=['drivername','Error-Prone Percentage'])

for driver in range(0,len(driver\_Ids)):

temp = results[results['driverId'] == driver\_Ids[driver]]

race\_count = len(temp)

temp = temp[(temp['statusId'] == 3) | (temp['statusId'] == 4)]

incident\_count = len(temp)

error\_stat.loc[len(error\_stat)] = [driver\_names[driver],(incident\_count/race\_count)\*100]

error\_stat

Table

Description automatically generated

The lowest error percentage was shown by Alexander Albon (Williams Racing) and the highest by, quite ironically, his teammate Nicholas Latifi.

**Overtake Rate per race**

A driver’s true skill can be measured by how aggressive he/she can be on the track when competing with other drivers while also maintaining race etiquette. Overtakes per race measures how many overtakes a driver makes each race.

# Calculating overtakes per race for the 2022 season

race\_Id = [1081,1082,1084,1086]

overtake\_stat = pd.DataFrame(columns = ['drivername','overtakes/race'])

for driver in range(0,len(driver\_Ids)):

total\_overtakes = 0

for race in race\_Id:

temp = lap\_times[(lap\_times['driverId'] == driver\_Ids[driver]) & (lap\_times['raceId'] == race)]

grid = temp['position'].head(1).values[0]

position = temp['position'].tail(1).values[0]

overtakes = position - grid

if(overtakes<0):

overtakes = 0

total\_overtakes = total\_overtakes + overtakes

overtake\_stat.loc[len(overtake\_stat)] = [driver\_names[driver],total\_overtakes/len(race\_Id)]

overtake\_stat

A black screen with white text

Description automatically generated with low confidence

Here we see Kevin Magnussen had the most number of overtakes while quite a few drivers have none. Now here some drivers are at better positions than Magnussen which explains that the cars in front don’t really need overtakes for points, all they need is better qualifying positions.

**Driver Ratings (based on Delta, Driver Error, Overtakes)**

I chose to create my own driver rating by ranking each driver based on their pace, errors and overtakes. After ranking them, I took an average of the rankings to produce the final ratings.

# Ranking each driver based on Error percentage, Delta and overtakes

driver\_stat = driver\_stat.sort\_values(by='delta')

delta\_rank = []

error\_rank = []

overtake\_rank = []

for i in range(1,21):

delta\_rank.append(i)

error\_rank.append(i)

overtake\_rank.append(i)

driver\_stat['delta\_rank'] = delta\_rank

driver\_stat = driver\_stat.sort\_values(by='Error-Prone Percentage')

driver\_stat['error\_rank'] = error\_rank

driver\_stat = driver\_stat.sort\_values(by='overtakes/race',ascending=False)

driver\_stat['overtake\_rank'] = overtake\_rank

driver\_stat = driver\_stat.sort\_values(by='delta')

driver\_stat

Graphical user interface

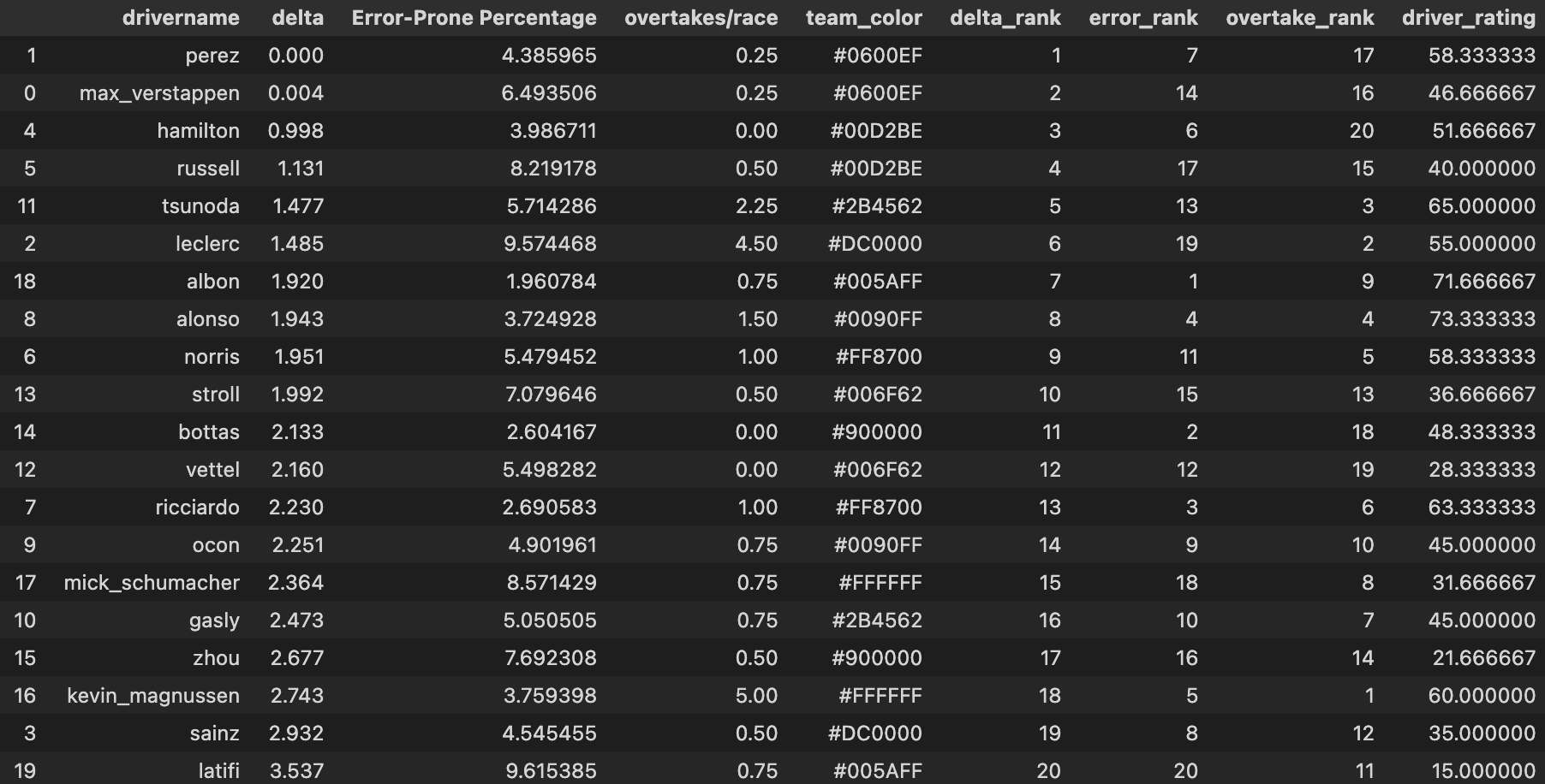
Description automatically generated with medium confidence

# Calculating the driver ratings based on the ranks calculated earlier

# Driver rating = ((((20-delta\_rank)/20) + ((20-error\_rank)/20) + ((20-overtake\_rank)/20))/3

driver\_stat['driver\_rating'] = ((((20-driver\_stat['delta\_rank'])/20) + ((20-driver\_stat['error\_rank'])/20) + ((20-driver\_stat['overtake\_rank'])/20))/3)\*100

driver\_stat



# Plotting driver ranks

x = driver\_stat['delta\_rank']

y = driver\_stat['driver\_rating']

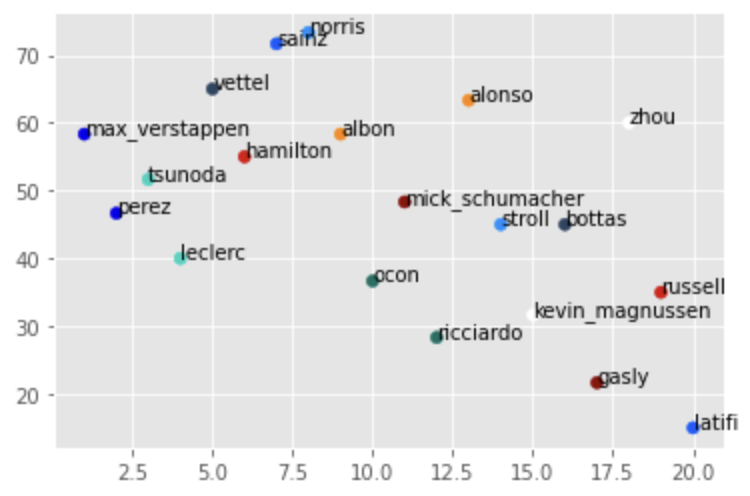
z = driver\_stat['team\_color']

for i, driver in enumerate(driver\_stat['drivername']):

plt.annotate(driver, (x[i], y[i]))

plt.scatter(x,y,c=z)

plt.show()



As you can see, the driver with the most ratings was Fernando Alonso (Alpine) and the with the least was Nicholas Latifi (Williams).

**Conclusion**

This project has been important in developing my skills in the field of Data Science and statistics. Lap prediction has always been something I looked forward to doing from such a long time and this course has finally provided me with skills to work on this. Future works would include live lap time prediction, better driver ratings along with taking into account weather conditions and tyre life for prediction.