```
                Selected Project: F1 Race Prediction for Media
Company
```

Our client a leading media company specializing in motorsports coverage and analysis. With an ever-expanding audience of Formula 1 fanatics, they are determined to provide cutting-edge insights and predictions that will revolutionize the viewing experience for their subscribers.

The goal of this project is to leverage advanced machine learning techniques to develop a predictive model capable of accurately forecasting the top finishers in Formula 1 Grand Prix events. Achieving this will enhance viewer engagement, establish them as a thought leader in data-driven motorsports analysis, attract advertising and sponsorship opportunities, and gain a competitive advantage in the motorsport media landscape.

Project Benefits:

Enhanced Viewer Engagement: Imagine captivating our audience with insightful analysis driven by powerful predictions. This will undoubtedly lead to increased viewer engagement, ultimately translating to higher subscription rates and revenue growth. Thought Leadership Status: The successful implementation of this project will solidify this media company’s position as a pioneer in data-driven motorsports analysis, establishing us as thought leaders in the industry. Advertising and Sponsorship Opportunities: Imagine the allure of attracting sponsors and advertisers who want to be associated with cutting-edge analytics and innovative content. Accurate race predictions will make us a highly sought-after partner. Deliverables:

A well-documented machine learning model: This model should be capable of predicting the top finishers (e.g., top 5 or top 10) in Formula 1 races with a high level of accuracy. (Optional) A user-friendly interface or dashboard. Comprehensive report: This report should detail the model's architecture, performance metrics, and potential areas for future improvement. Presentation: Here, you'll outline the project's methodology, findings, and recommendations for integrating the model into the media company’s content and analysis workflow. Success Criteria:

Model Accuracy: The machine learning model must achieve a high level of accuracy in predicting the top finishers in Formula 1 races, as measured on a separate test dataset. Competitive Advantage: This project's ultimate goal is to provide the media company with a significant edge in the motorsport media landscape. By offering accurate and insightful race predictions, we'll attract new subscribers and sponsors, solidifying our leadership position. Project Guidelines:

Timely completion and submission of project deliverables by the given deadline. Deep dive into the provided dataset to explore relationships between different features. Implementation and comparison of the performance of chosen algorithms on the dataset. Fine-tuning of the best performing model to maximize prediction accuracy. Research and proposal of various machine learning algorithms suitable for the project. Documenting the entire process, including code, results, and key decisions taken. Preparation of a final presentation in PowerPoint format to showcase findings and recommendations. Ensuring actionable recommendations supported by evidence from research findings. Maintaining a high level of professionalism in communication and deliverables, including proper formatting, grammar, and citation practices in the report. The

data shared with you is not to be uploaded publicly on any platform(github,kaggle,etc). Feel free to incorporate any other data 'IN ADDITION' to the dataset provided. Submission:

Submitting the Jupyter notebook containing analysis and model training. Submitting the final report in .doc or PowerPoint Presentation format. (Optional) Submitting the Power BI or Tableau dashboard. NOTE: Create a zip file of the above mentioned for the final submission.

Packages

```
import pandas as pd
import numpy as np
```

Mounting the datasets from gdrive

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly
remount, call drive.mount("/content/drive", force_remount=True).
```

Loading the datasets and converting them into Dataframes

```
driver_details=pd.read_csv("/content/drive/MyDrive/BIA
DATA/driver_details.csv")
race_details=pd.read_csv("/content/drive/MyDrive/BIA
DATA/race_details.csv")
starting_grid=pd.read_csv("/content/drive/MyDrive/BIA
DATA/starting_grids.csv")
practices=pd.read_csv("/content/drive/MyDrive/BIA DATA/practices.csv")
fastestlaps=pd.read_csv("/content/drive/MyDrive/BIA
DATA/fastestlaps_detailed.csv")
race_summaries=pd.read_csv("/content/drive/MyDrive/BIA
DATA/race_summaries.csv")
piststop=pd.read_csv("/content/drive/MyDrive/BIA DATA/pitstops.csv")
qualifying=pd.read_csv("/content/drive/MyDrive/BIA
DATA/qualifyings.csv")
race_summaries=pd.read_csv("/content/drive/MyDrive/BIA
DATA/race_summaries.csv")
```

Using inner join to join the datasets

```
df1 = pd.merge(driver_details, race_details, on='Key', how='inner')

print(driver_details.shape,race_details.shape,df1.shape)

(19814, 8) (23978, 12) (18202, 19)

df2=pd.merge(df1, starting_grid, on='Key', how='inner')

print(df1.shape,starting_grid.shape,df2.shape)

(18202, 19) (22529, 10) (16855, 28)

df2.drop(['Grand Prix_x', 'Car_x', 'Year_x', 'Driver_x'], axis=1,
inplace=True)

df3=pd.merge(df2, practices, on='Key', how='inner')

print(df2.shape,practices.shape,df3.shape)

(16855, 24) (37089, 12) (30131, 35)

df3.drop(['DriverCode_x'], axis=1, inplace=True)

df4=pd.merge(df3, fastestlaps, on='Key', how='inner')

print(df3.shape,fastestlaps.shape,df4.shape)

(30131, 34) (15512, 19) (28967, 52)

df4.drop(['Grand Prix_x', 'Car_x', 'Detail_x', 'Year_x', 'Driver_x'],
axis=1, inplace=True)

df5=pd.merge(df4, piststop, on='Key', how='inner')

print(df4.shape,piststop.shape,df5.shape)

(28967, 47) (20293, 12) (50112, 58)

df5.drop(['DriverCode_x'], axis=1, inplace=True)

df6=pd.merge(df5, qualifying, on='Key', how='inner')

print(df5.shape,qualifying.shape,df6.shape)

(50112, 57) (17236, 14) (53991, 70)

df6.to_csv('bigdata.csv')
```

DROPING COLUMNS THAT ARE NOT NECCESARY

```
df6.columns
```

```
Index(['Date', 'driver PTS', 'driver  Position', 'Key', 'Pos', 'No_x',
       'Driver_y', 'Car_y', 'Race Laps', 'Race Time/Retired ', 'Race
points',
       'Year_y', 'Grand Prix_y', 'Detail_y', 'DriverCode_y',
'startgrid No',
       'startgrid Pos', 'startgrid Time', 'Car_y', 'Practice Detail',
       'Driver_y', 'Gap', 'Grand Prix_y', 'Practice Laps', 'Practice
No',
       'Practice Pos', 'Practice Time', 'Year_y', 'Avg Speed',
'Car_x',
       'Detail_x', 'Driver_x', 'DriverCode_y', 'Grand Prix_x',
       'no of fast Lap ', 'fastlaps no', 'fastlaps Pos ',
       'Time taken in fast laps', 'Time of day', 'Year_x', 'Unnamed:
13',
       'Unnamed: 14', 'Unnamed: 15', 'Unnamed: 16', 'Unnamed: 17',
       'Unnamed: 18', 'Stops', 'No_y', 'Driver_y', 'Car_y', 'pitstop
Lap',
       'Pitstop Time of day ', 'Pitstop Total time ', 'Year_y', 'Grand
Prix_y',
       'Detail_y', 'DriverCode_x', 'Car', 'Detail', 'Driver',
'DriverCode_y',
       'Grand Prix', 'qualifying Laps', 'qualifying No', 'qualifying
Pos',
       'qualifying Q1', 'qualifying Q2', 'qualifying Q3', 'qualifying
Time',
       'Year'],
      dtype='object')
```

```
pd.set_option('display.max_columns', None)  # Show all columns
pd.set_option('display.max_rows', None)  # Show all columns

df6.head(1)
```

{"type":"dataframe","variable_name":"df6"}

EDA

```
import pandas as pd
team_details=pd.read_csv("/content/drive/MyDrive/BIA
DATA/team_details.csv")
team_details=team_details.dropna(axis=1)
```

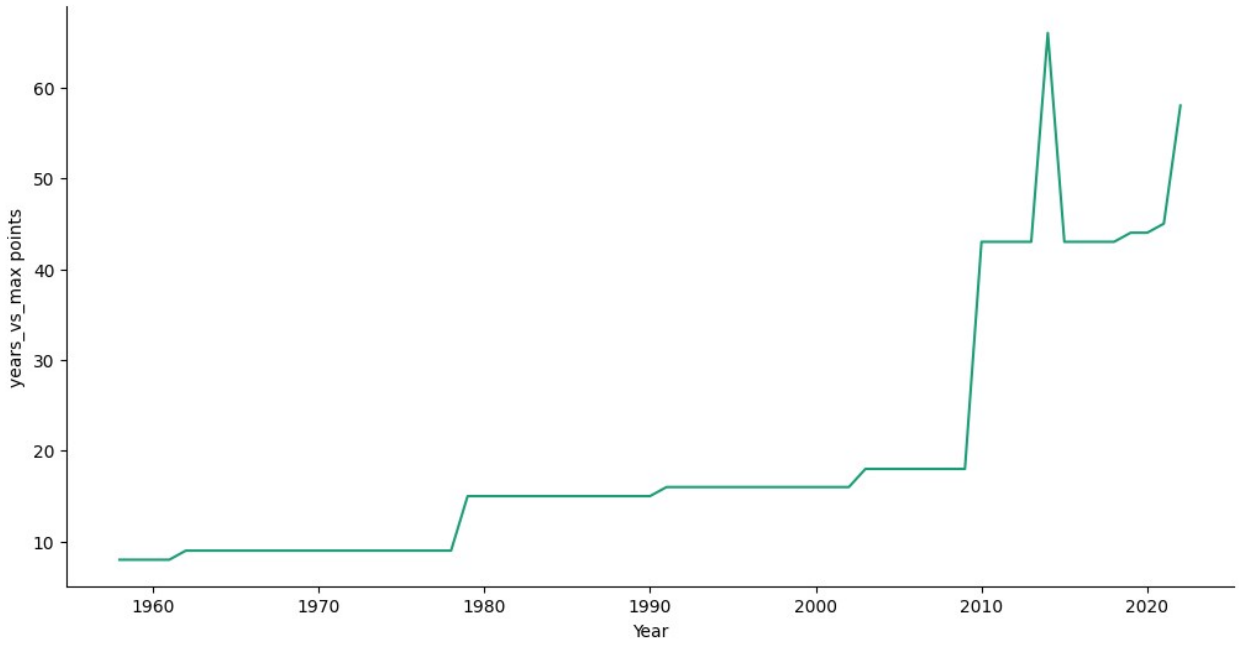
```
max_points_year = team_details.loc[team_details['PTS'].idxmax(),
'Year']
max_points_year
```

```
2014
```

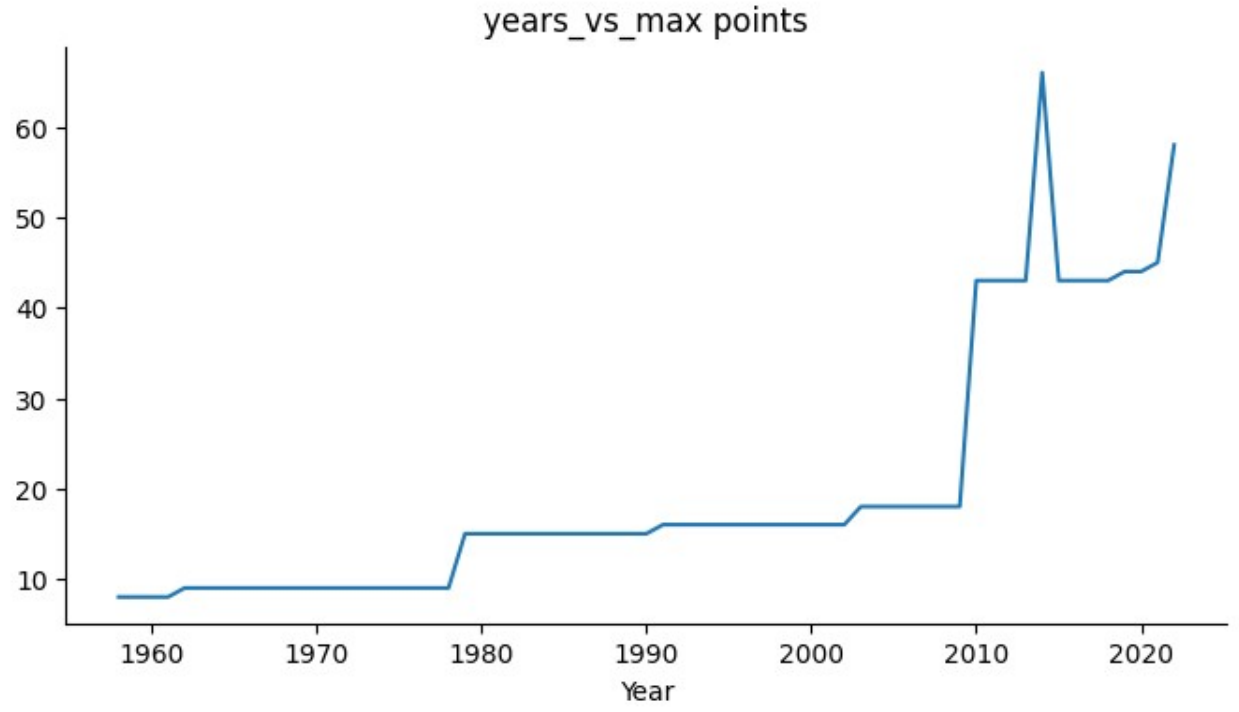
```
# Calculate total points for each year
df = team_details

max_points_per_year = df.groupby('Year')['PTS'].max()
print("Total points per year:")
pd.set_option('display.max_rows', None)  # Show all columns
years_vs_max_points=max_points_per_year.to_frame("years_vs_max_points"
)
years_vs_max_points.dtypes
```

```
Total points per year:

years_vs_max_points    float64
dtype: object
```

```
from matplotlib import pyplot as plt
import seaborn as sns
def _plot_series(series, series_name, series_index=0):
  palette = list(sns.palettes.mpl_palette('Dark2'))
  xs = series.index
  ys = series['years_vs_max_points']
  plt.plot(xs, ys, label=series_name, color=palette[series_index %
len(palette)])

fig, ax = plt.subplots(figsize=(10, 5.2), layout='constrained')
df_sorted = years_vs_max_points.sort_values('Year', ascending=True)
_plot_series(df_sorted, '')
sns.despine(fig=fig, ax=ax)
plt.xlabel('Year')
_ = plt.ylabel('years_vs_max points')
```



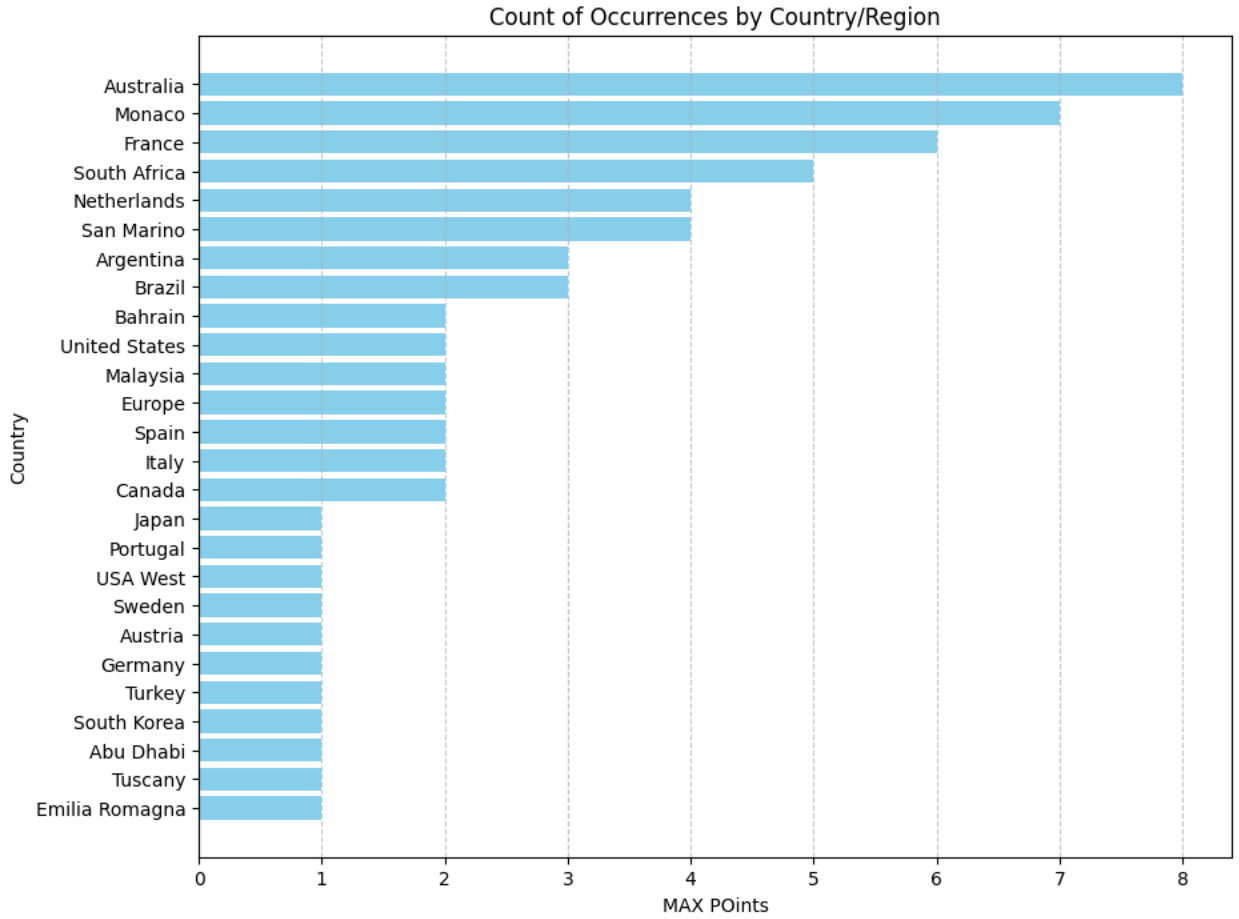

```
from matplotlib import pyplot as plt
years_vs_max_points['years_vs_max_points'].plot(kind='line',
figsize=(8, 4), title='years_vs_max points')
plt.gca().spines[['top', 'right']].set_visible(False)
```




```
import matplotlib.pyplot as plt
```

```
# Find the year with the highest total points
year_with_max_points = max_points_per_year.idxmax()
max_points = max_points_per_year.max()
print("year_with_max_points : ",year_with_max_points,"\nmax_points :
",year_with_max_points)
Year_team_pts=df.loc[df.groupby('Year')['PTS'].idxmax()]
grandpix_wins=Year_team_pts['Grand
Prix'].value_counts().to_frame("grandpix_wins")
# Plotting
plt.figure(figsize=(10, 8))
plt.barh(grandpix_wins.index,grandpix_wins['grandpix_wins'],
color='skyblue')
plt.xlabel('MAX POints')
plt.ylabel('Country')
plt.title('Count of Occurrences by Country/Region')
plt.gca().invert_yaxis()  # Invert y-axis to have the highest count at
the top
plt.grid(axis='x', linestyle='--', alpha=0.7)
plt.show()
```

```
year_with_max_points :  2014
max_points :  2014
```

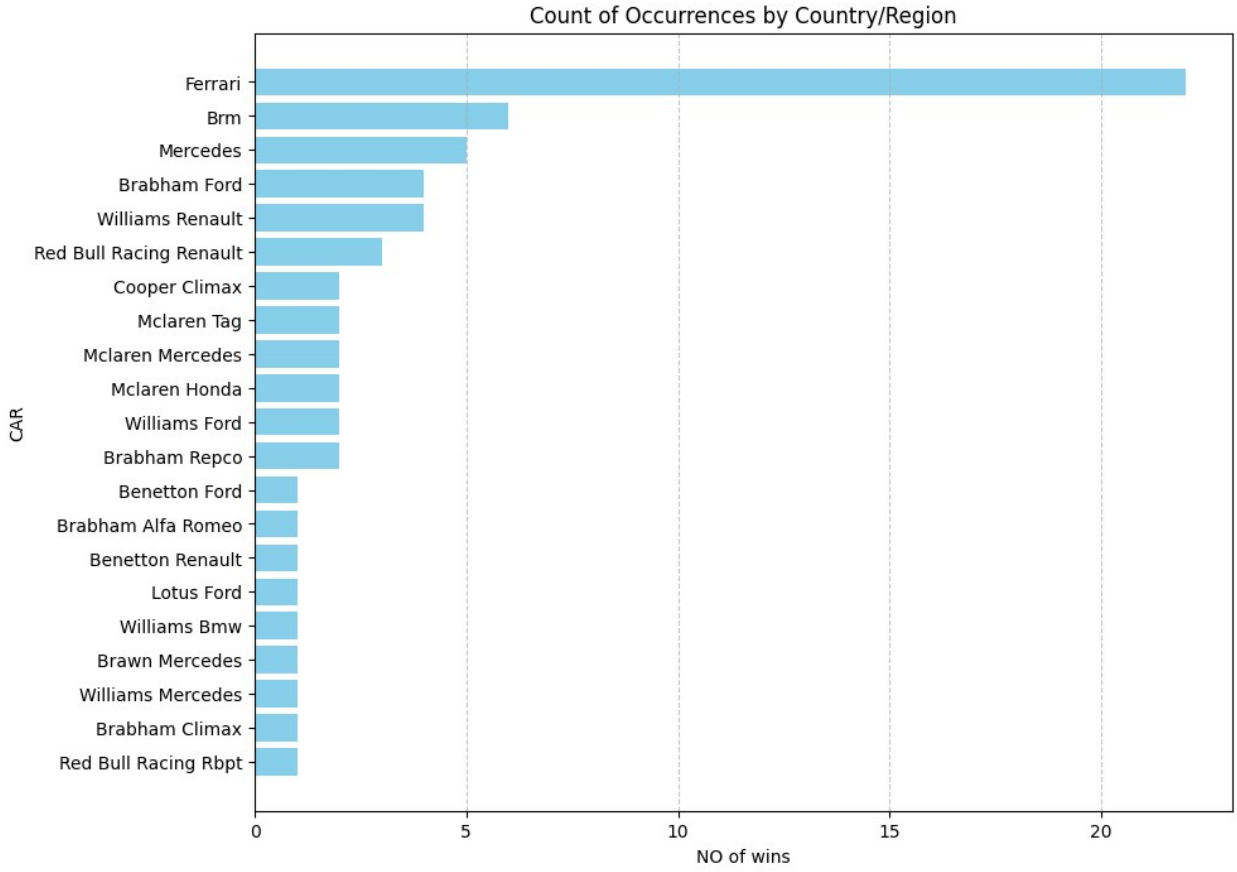



```
# Find the year with the highest total points
year_with_max_points = max_points_per_year.idxmax()
max_points = max_points_per_year.max()
print("year_with_max_points : ",year_with_max_points,"\nmax_points :
",year_with_max_points)
```

```
year_with_max_points :  2014
max_points :  2014
```

```
grandpix_wins=Year_team_pts['Grand
Prix'].value_counts().to_frame("grandpix_wins")

team_wins=Year_team_pts['Team'].value_counts().to_frame("team_wins")
# Plotting
plt.figure(figsize=(10, 8))
plt.barh(team_wins.index,team_wins['team_wins'], color='skyblue')
plt.xlabel('NO of wins')
plt.ylabel('CAR')
plt.title('Count of Occurrences by Country/Region')
plt.gca().invert_yaxis()  # Invert y-axis to have the highest count at
the top
```

```
plt.grid(axis='x', linestyle='--', alpha=0.7)
plt.show()
```



Race Summeries from 1950-2022

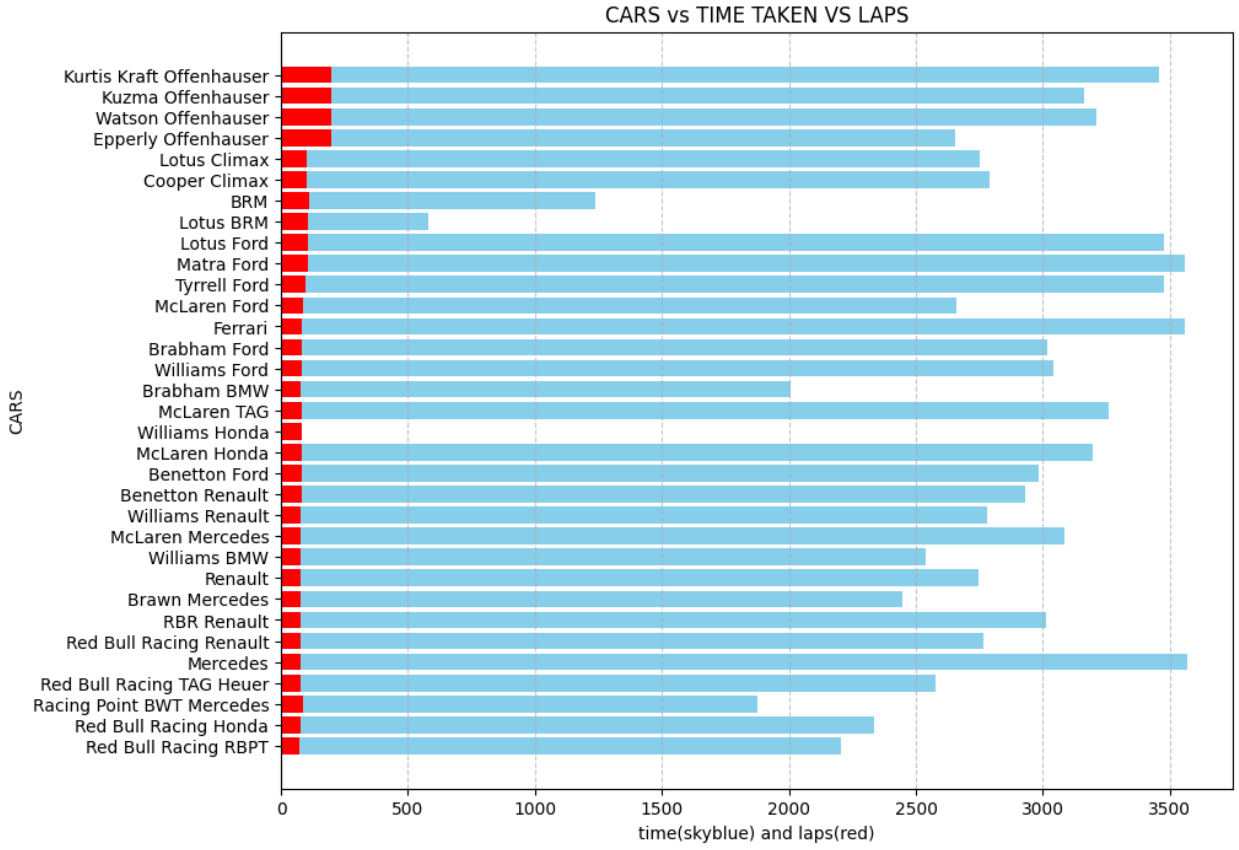
```
race_sum=pd.read_csv("/content/drive/MyDrive/BIA
DATA/race_summaries.csv")

converted_race_sum=race_sum.copy(deep=True)
# Function to convert time format
def convert_to_timedelta(time_str):
    if pd.isnull(time_str):
        return pd.NaT  # Return NaT (Not a Time) for NaN values

    # Handle cases where time_str is already a Timedelta (for direct
conversion)
    if isinstance(time_str, pd.Timedelta):
        return time_str

    # Convert to string if not already
    time_str = str(time_str)
```

```
    # Split the string to get minutes and seconds.tenths
    minutes, seconds_and_tenths = time_str.split(':')
    seconds, tenths = seconds_and_tenths.split('.')

    # Calculate total seconds
    total_seconds = int(minutes) * 60 + int(seconds) + int(tenths) /
10

    # Convert to Timedelta
    return pd.to_timedelta(f'{total_seconds} seconds')

# Apply the conversion function to the column
converted_race_sum['race summary Time'] = converted_race_sum['race
summary Time'].apply(convert_to_timedelta)

race_sum.dtypes
```

```
Grand Prix             object
race Date              object
Winner                 object
Car                    object
Laps                  float64
race summary Time      object
WinnerCode             object
Year                    int64
Key                    object
dtype: object
```

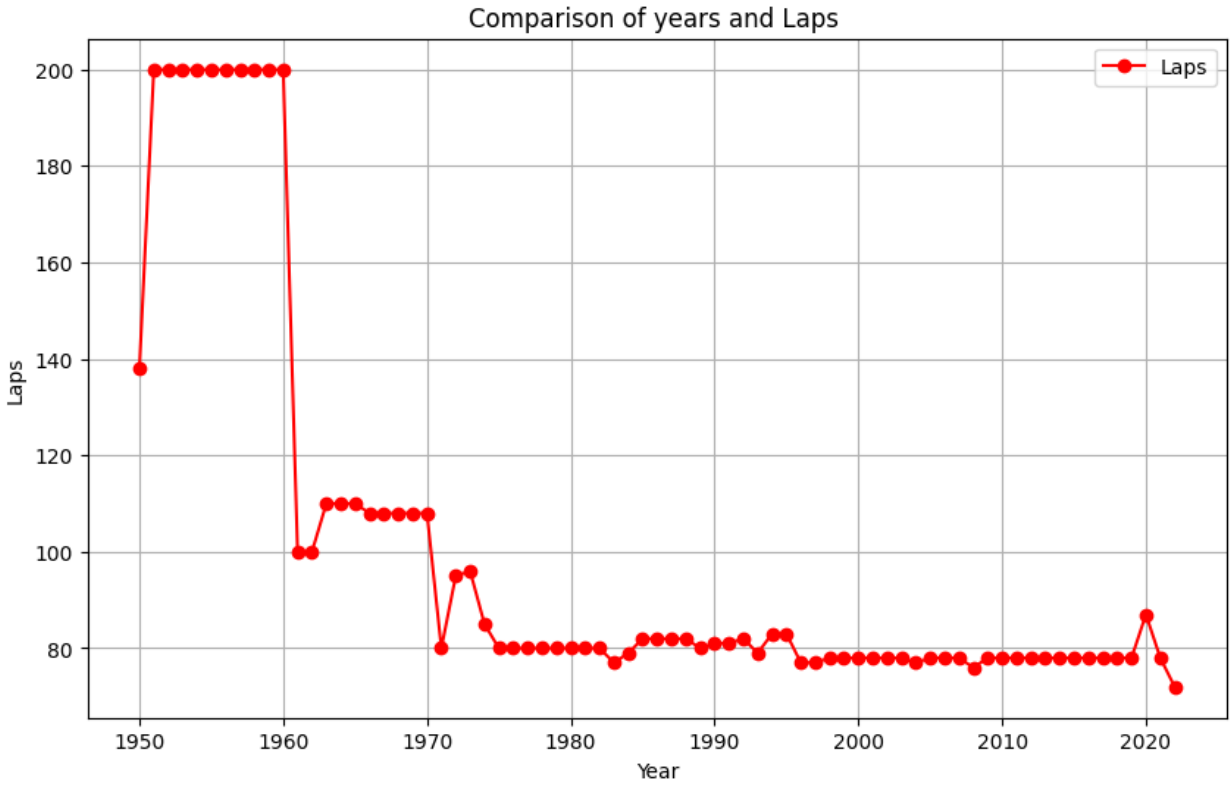
```
# Function to convert time format to seconds for easier plotting
def convert_to_seconds(time_str):
    if pd.isnull(time_str):
      return None
    minutes, seconds_and_tenths = time_str.split(':')
    seconds, tenths = seconds_and_tenths.split('.')
    total_seconds = int(minutes) * 60 + int(seconds) + int(tenths) /
10
    return total_seconds
race_sum['time_seconds'] = race_sum['race summary
Time'].apply(convert_to_seconds)


# Group by 'Year' and find the row with maximum 'laps'
max_laps_idx = race_sum.groupby('Year')['Laps'].idxmax()

# Use .loc to get the corresponding 'time_seconds' for the max 'laps'
time_seconds_max_laps = race_sum.loc[max_laps_idx, ['Year',
'time_seconds', 'Laps','Car']].reset_index(drop=True)

time_seconds_max_laps['Laps'].unique()
```

```
# Plotting
plt.figure(figsize=(10, 8))
plt.barh(time_seconds_max_laps['Car'],time_seconds_max_laps['time_seco
nds'], color='skyblue')
plt.barh(time_seconds_max_laps['Car'],time_seconds_max_laps['Laps'],
color='red')

plt.xlabel('time(skyblue) and laps(red)')
plt.ylabel('CARS')
plt.title('CARS vs TIME TAKEN VS LAPS')
plt.gca().invert_yaxis()  # Invert y-axis to have the highest count at
the top
plt.grid(axis='x', linestyle='--', alpha=0.7)
plt.show()
```




```
# Plotting
plt.figure(figsize=(10, 6))

# Plot time_seconds
plt.plot(time_seconds_max_laps['Year'], time_seconds_max_laps['Laps'],
marker='o', linestyle='-', color='r', label='Laps')
```

```
# Add labels and title
plt.xlabel('Year')
plt.ylabel('Laps')
plt.title('Comparison of years and Laps ')
plt.legend()

# Display the plot
plt.grid(True)
plt.show()
```

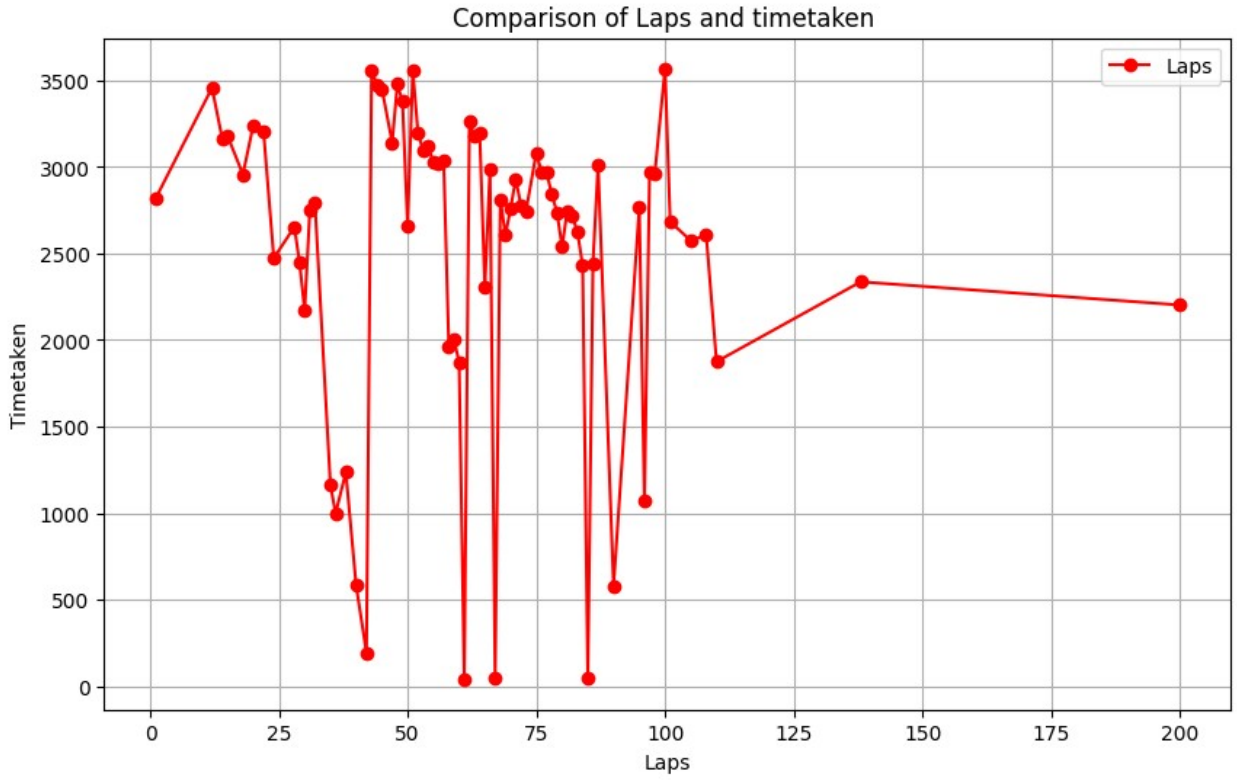



```
# Group by 'laps' and find the fastest time for each group
result = race_sum.groupby('Laps').apply(lambda x:
x.loc[x['time_seconds'].idxmin()])

# Display the result
myquery=result[['Laps', 'time_seconds', 'Car']]
myquery
```

```
{"repr_error":"cannot insert Laps, already
exists","type":"dataframe","variable_name":"myquery"}
```

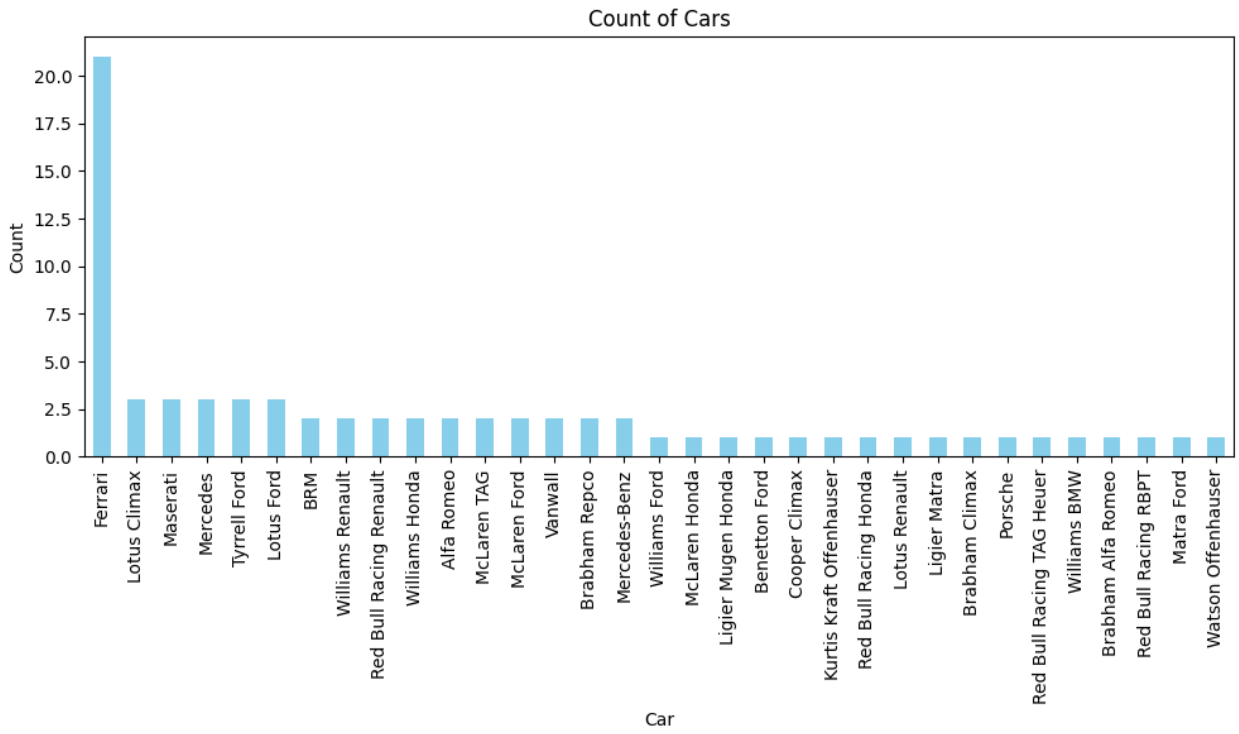
```
# Plotting
plt.figure(figsize=(10, 6))
```

```
# Plot time_seconds
plt.plot(myquery['Laps'], time_seconds_max_laps['time_seconds'],
marker='o', linestyle='-', color='r', label='Laps')

# Add labels and title
plt.xlabel('Laps')
plt.ylabel('Timetaken')
plt.title('Comparison of Laps and timetaken ')
plt.legend()

# Display the plot
plt.grid(True)
plt.show()
```




```
# Count occurrences of each car
car_counts = myquery['Car'].value_counts()

# Plotting
plt.figure(figsize=(10, 6))
car_counts.plot(kind='bar', color='skyblue')
plt.title('Count of Cars')
plt.xlabel('Car')
plt.ylabel('Count')
plt.xticks(rotation=90)
```

```
plt.tight_layout()
plt.show()
```

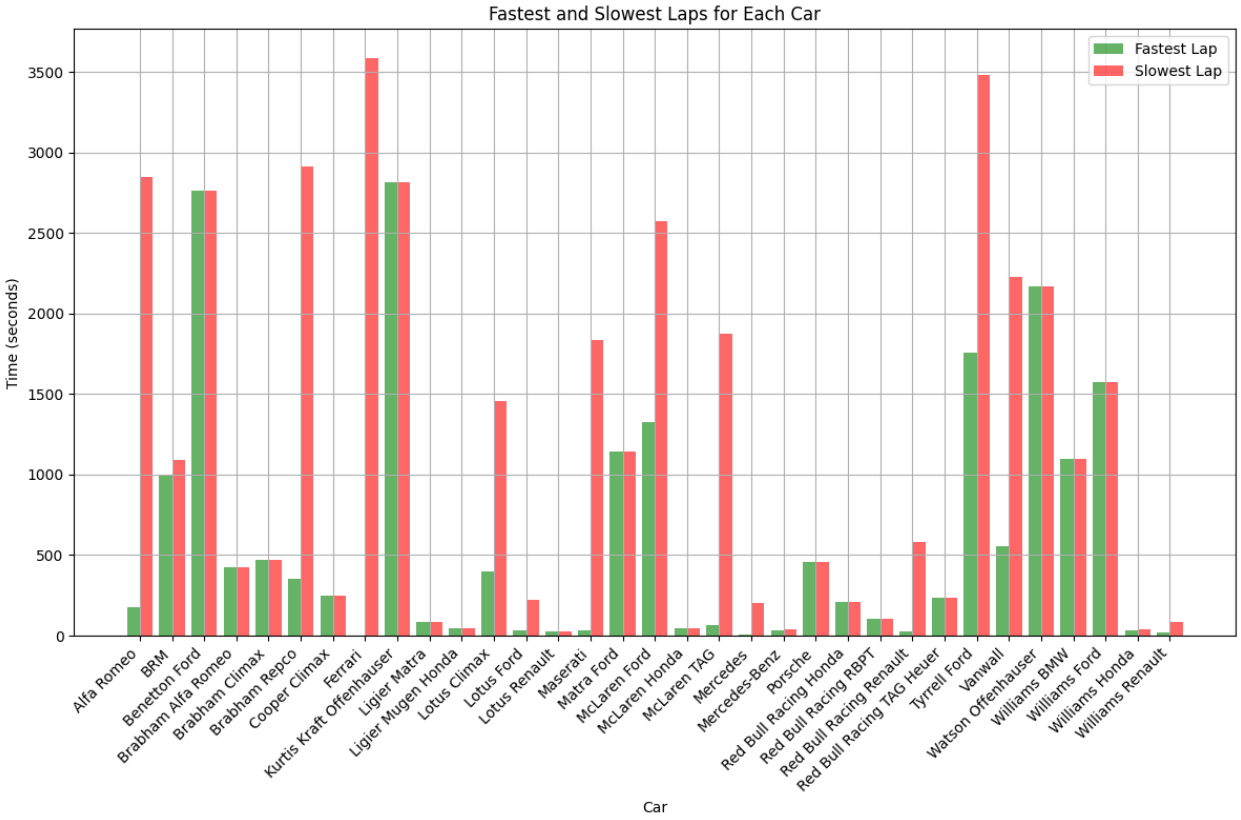



```
!pip install mplcursors

Collecting mplcursors
  Downloading mplcursors-0.5.3.tar.gz (88 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0.0/88.8 kB ? eta -:--:--
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ ━━ 81.9/88.8 kB 2.9 MB/s eta
0:00:01 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 88.8/88.8 kB 2.2 MB/s
eta 0:00:00
ents to build wheel ... etadata (pyproject.toml) ... atplotlib!
=3.7.1,>=3.1 (from mplcursors)
  Downloading matplotlib-3.9.1-cp310-cp310-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl (8.3 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 8.3/8.3 MB 19.5 MB/s eta
0:00:00
ent already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib!=3.7.1,>=3.1-
>mplcursors) (1.2.1)
Requirement already satisfied: cycler>=0.10 in
/usr/local/lib/python3.10/dist-packages (from matplotlib!=3.7.1,>=3.1-
>mplcursors) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib!=3.7.1,>=3.1-
>mplcursors) (4.53.0)
```

```
Requirement already satisfied: kiwisolver>=1.3.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib!=3.7.1,>=3.1-
>mplcursors) (1.4.5)
Requirement already satisfied: numpy>=1.23 in
/usr/local/lib/python3.10/dist-packages (from matplotlib!=3.7.1,>=3.1-
>mplcursors) (1.25.2)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib!=3.7.1,>=3.1-
>mplcursors) (24.1)
Requirement already satisfied: pillow>=8 in
/usr/local/lib/python3.10/dist-packages (from matplotlib!=3.7.1,>=3.1-
>mplcursors) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib!=3.7.1,>=3.1-
>mplcursors) (3.1.2)
Requirement already satisfied: python-dateutil>=2.7 in
/usr/local/lib/python3.10/dist-packages (from matplotlib!=3.7.1,>=3.1-
>mplcursors) (2.8.2)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7-
>matplotlib!=3.7.1,>=3.1->mplcursors) (1.16.0)
Building wheels for collected packages: mplcursors
  Building wheel for mplcursors (pyproject.toml) ... plcursors:
filename=mplcursors-0.5.3-py3-none-any.whl size=20728
sha256=f19e08ff4bbec1078cd85e1cc80ebb33792b1babb7da6d931aaa8154faf88be
b
  Stored in directory:
/root/.cache/pip/wheels/83/43/92/44f9515471f56877c774a515a2902d3e5484e
a1bc7fd412d03
Successfully built mplcursors
Installing collected packages: matplotlib, mplcursors
  Attempting uninstall: matplotlib
    Found existing installation: matplotlib 3.7.1
    Uninstalling matplotlib-3.7.1:
      Successfully uninstalled matplotlib-3.7.1
Successfully installed matplotlib-3.9.1 mplcursors-0.5.3
```

```python
import pandas as pd
import matplotlib.pyplot as plt
import mplcursors
# Group by 'Car' to find fastest and slowest laps
fastest_laps = myquery.loc[myquery.groupby('Car')
['time_seconds'].idxmin()]
slowest_laps = myquery.loc[myquery.groupby('Car')
['time_seconds'].idxmax()]

# Set up plotting
fig, ax = plt.subplots(figsize=(12, 8))
```

```
# Width of each bar
bar_width = 0.4

# X locations for the bars
index = range(len(fastest_laps))

# Plot fastest laps with hover information
bars_fastest = ax.bar(index, fastest_laps['time_seconds'], bar_width,
label='Fastest Lap', color='g', alpha=0.6)
mplcursors.cursor(bars_fastest, hover=True).connect(
    "add", lambda sel: sel.annotation.set_text(f"Laps:
{fastest_laps.iloc[sel.target.index]['Laps']}"))

# Plot slowest laps with hover information
bars_slowest = ax.bar([i + bar_width for i in index],
slowest_laps['time_seconds'], bar_width, label='Slowest Lap',
color='r', alpha=0.6)
mplcursors.cursor(bars_slowest, hover=True).connect(
    "add", lambda sel: sel.annotation.set_text(f"Laps:
{slowest_laps.iloc[sel.target.index]['Laps']}"))

# X-axis labels
ax.set_xticks([i + bar_width / 2 for i in index])
ax.set_xticklabels(fastest_laps['Car'], rotation=45, ha='right')

# Labels and title
ax.set_xlabel('Car')
ax.set_ylabel('Time (seconds)')
ax.set_title('Fastest and Slowest Laps for Each Car')
ax.legend()

# Grid lines
ax.grid(True)

# Show plot
plt.tight_layout()
plt.show()
```




```
sprint_result=pd.read_csv("/content/drive/MyDrive/BIA DATA/sprint_results.csv")
sprint_result
```


```
{"summary":"{\n  \"name\": \"sprint_result\",\n  \"rows\": 120,\n  \"fields\": [\n    {\n      \"column\": \"Pos\",\n      \"properties\": {\n        \"dtype\": \"category\",\n        \"num_unique_values\": 21,\n        \"samples\": [\n          \"1\",\n          \"18\",\n          \"16\"\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"No\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 24,\n        \"min\": 1,\n        \"max\": 99,\n        \"num_unique_values\": 25,\n        \"samples\": [\n          63,\n          6,\n          33\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"Driver\",\n      \"properties\": {\n        \"dtype\": \"category\",\n        \"num_unique_values\": 24,\n        \"samples\": [\n          \"George Russell\",\n          \"Nicholas Latifi\",\n          \"Max Verstappen\"\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"Car\",\n      \"properties\": {\n        \"dtype\": \"category\",\n        \"num_unique_values\": 14,\n        \"samples\": [\n          \"Haas Ferrari\",\n          \"Alfa Romeo Ferrari\",\n          \"Red Bull Racing Honda\"\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
```

```
n    },\n    {\n      \"column\": \"Laps\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 4,\n        \"min\": 0,\n
\"max\": 24,\n        \"num_unique_values\": 8,\n        \"samples\":
[\n          16,\n          21,\n          17\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Time/Retired\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 116,\n        \"samples\": [\n
\"+31.032s\",\n          \"+24.111s\",\n          \"+19.787s\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"PTS\",\n      \"properties\": {\
n        \"dtype\": \"number\",\n        \"std\": 2,\n        \"min\":
0,\n        \"max\": 8,\n        \"num_unique_values\": 9,\n
\"samples\": [\n          5,\n          2,\n          7\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Year\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0,\n        \"min\": 2021,\n
\"max\": 2022,\n        \"num_unique_values\": 2,\n
\"samples\": [\n          2022,\n          2021\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Grand Prix\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 5,\n        \"samples\": [\n
\"Italy\",\n          \"Austria\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Detail\",\n      \"properties\":
{\n        \"dtype\": \"category\",\n        \"num_unique_values\":
1,\n        \"samples\": [\n          \"Sprint-Results\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"DriverCode\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 24,\n        \"samples\": [\n          \"RUS\"\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Key\",\n      \"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 120,\n        \"samples\": [\n
\"Mercedes-Lewis Hamilton-Brazil-2021\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe","variable_name":"sprint_result"}
```


```
pos_pts1=sprint_result.groupby('Pos')['PTS'].agg(list).to_frame("PTS")
pos_pts1
pos_pts2=sprint_result.groupby('Pos')
['Laps'].agg(list).to_frame("LAPS")
pos_pts2
mymerged=pos_pts2.merge(pos_pts1,left_index=True,right_index=True)
mymerged
```


```
{"summary":"{\n  \"name\": \"mymerged\",\n  \"rows\": 21,\n
\"fields\": [\n    {\n      \"column\": \"Pos\",\n
```

\"properties\": {\n   \"dtype\": \"string\",\n
\"num_unique_values\": 21,\n   \"samples\": [\n   \"1\",\n
\"7\",\n   \"5\"\n   ],\n   \"semantic_type\": \"\",\
n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"LAPS\",\n   \"properties\": {\n   \"dtype\":
\"object\",\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"PTS\",\n   \"properties\": {\n   \"dtype\": \"object\",\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   }\n]\n}","type":"dataframe","variable_name":"mymerged"}

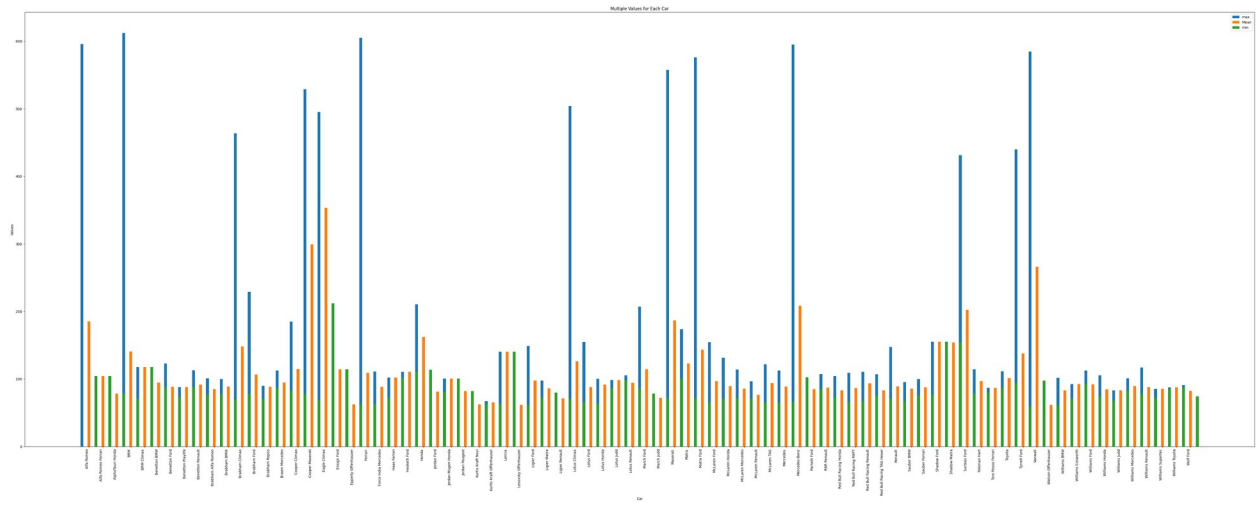
```
fastest_laps=pd.read_csv("/content/drive/MyDrive/BIA
DATA/fastest_laps.csv")
fastest_laps
```

{"summary":"{\n \"name\": \"fastest_laps\",\n \"rows\": 1078,\n
\"fields\": [\n   {\n   \"column\": \"Grand Prix\",\n
\"properties\": {\n   \"dtype\": \"category\",\n
\"num_unique_values\": 52,\n   \"samples\": [\n
\"Brazil\",\n   \"Russia\",\n   \"Emilia Romagna\"\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"Driver\",\n   \"properties\":
{\n   \"dtype\": \"category\",\n   \"num_unique_values\":
133,\n   \"samples\": [\n   \"Carlos Pace\",\n
\"Riccardo Patrese\",\n   \"Lorenzo Bandini\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"Car\",\n   \"properties\": {\n
\"dtype\": \"category\",\n   \"num_unique_values\": 80,\n
\"samples\": [\n   \"Parnelli Ford\",\n   \"Alfa
Romeo\",\n   \"Matra Ford\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"Time\",\n   \"properties\": {\n
\"dtype\": \"category\",\n   \"num_unique_values\": 524,\n
\"samples\": [\n   \"01:14.2\",\n   \"01:13.6\",\n
\"02:51.1\"\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"DriverCode\",\n   \"properties\": {\n   \"dtype\":
\"category\",\n   \"num_unique_values\": 121,\n
\"samples\": [\n   \"CEV\",\n   \"FIT\",\n
\"TAR\"\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"Year\",\n   \"properties\": {\n   \"dtype\": \"number\",\n
\"std\": 19,\n   \"min\": 1950,\n   \"max\": 2022,\n
\"num_unique_values\": 73,\n   \"samples\": [\n   1954,\n
2013,\n   1968\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"Key\",\n   \"properties\": {\n   \"dtype\": \"string\",\n
\"num_unique_values\": 1078,\n   \"samples\": [\n
\"McLaren TAG-Alain Prost-Monaco-1986\",\n   \"Benetton Ford-
Michael Schumacher-Brazil-1994\",\n   \"McLaren Mercedes-Lewis

```
Hamilton-Brazil-2010\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"fastest_laps"}

import numpy as np
# Function to convert time format to seconds for easier plotting
# Function to convert time string to seconds
def convert_to_seconds(time_str):
    if isinstance(time_str, str):
        try:
            # Split the string to get minutes and seconds.tenths
            minutes, seconds_and_tenths = time_str.split(':')
            seconds, tenths = seconds_and_tenths.split('.')

            # Calculate total seconds including tenths
            total_seconds = int(minutes) * 60 + int(seconds) +
int(tenths) / 10

            return total_seconds

        except ValueError:
            return np.nan  # Return NaN for invalid or missing data

    return np.nan  # Return NaN if time_str is not a string

fastest_laps['time_seconds'] =
fastest_laps['Time'].apply(convert_to_seconds)

mean=fastest_laps.groupby('Car')['time_seconds'].mean()
max=fastest_laps.groupby('Car')['time_seconds'].max()
min=fastest_laps.groupby('Car')['time_seconds'].min()
mean=mean.to_frame("mean")
max=max.to_frame("max")
min=min.to_frame("min")
mergedvalues=mean.merge(max,left_index=True,right_index=True).merge(mi
n,left_index=True,right_index=True)
mergedvalues

{"summary":"{\n  \"name\": \"mergedvalues\",\n  \"rows\": 80,\n
\"fields\": [\n    {\n      \"column\": \"Car\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 80,\n        \"samples\": [\n
\"Lancia\",\n          \"Alfa Romeo\",\n          \"Haas Ferrari\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"mean\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
50.10705577165448,\n        \"min\": 61.6,\n        \"max\": 353.5,\n
\"num_unique_values\": 80,\n        \"samples\": [\n          140.4,\n
185.32857142857142,\n          102.1\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
```

n   },\n   {\n   \"column\": \"max\",\n   \"properties\": {\n\"dtype\": \"number\",\n   \"std\": 163.40352624965834,\n\"min\": 61.6,\n   \"max\": 612.2,\n\"num_unique_values\": 78,\n   \"samples\": [\n   97.6,\n595.8,\n   71.5\n   ],\n   \"semantic_type\": \"\",\n\"description\": \"\"\n   }\n   },\n   {\n   \"column\":\"min\",\n   \"properties\": {\n   \"dtype\": \"number\",\n\"std\": 24.05950662788444,\n   \"min\": 60.0,\n   \"max\":211.9,\n   \"num_unique_values\": 77,\n   \"samples\": [\n117.8,\n   66.0,\n   69.9\n   ],\n\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe","variable_name":"mergedvalues"}

```
fig, ax = plt.subplots(figsize=(50, 20))  # Adjust figsize as needed

# Define positions and width for bars
positions = np.arange(len(mergedvalues.index))
width = 0.2  # Adjust width as needed
space = 0.3  # Adjust space between groups
# Plotting bars for each value
ax.bar(positions - width - space, mergedvalues['max'], width,
label='max')
ax.bar(positions, mergedvalues['mean'], width, label='Mean')
ax.bar(positions + width+space, mergedvalues['min'], width,
label='min')

# Adding labels and title
ax.set_xlabel('Car')
ax.set_ylabel('Values')
ax.set_title('Multiple Values for Each Car')
ax.set_xticks(positions)
ax.set_xticklabels(mergedvalues.index, rotation=90, ha='right')  #
Rotating x-axis labels for better readability
ax.legend()
mplcursors.cursor(hover=True)

# Show plot
plt.tight_layout()
plt.show()
```



DATA TANSFORMATION

```
# Function to convert time format to seconds for easier plotting
# Function to convert time string to seconds
def convert_to_seconds(time_str):
    if isinstance(time_str, str):
        try:
            # Split the string to get minutes and seconds.tenths
            minutes, seconds_and_tenths = time_str.split(':')
            seconds, tenths = seconds_and_tenths.split('.')

            # Calculate total seconds including tenths
            total_seconds = int(minutes) * 60 + int(seconds) +
int(tenths) / 10

            return total_seconds

        except ValueError:
            return np.nan  # Return NaN for invalid or missing data

    return np.nan  # Return NaN if time_str is not a string

df6['Practice_Time_seconds'] = df6['Practice
Time'].apply(convert_to_seconds)
df6['startgrid_Time_seconds']=df6['startgrid
Time'].apply(convert_to_seconds)
df6['Time_taken_in_fast_laps_seconds']=df6['Time taken in fast
laps'].apply(convert_to_seconds)
df6['qualifying_Time  _seconds']=df6['qualifying
Time'].apply(convert_to_seconds)
```

```
# Convert to datetime format
df6['Pitstop Time of day '] = pd.to_datetime(df6['Pitstop Time of day
'], format='%H:%M:%S')

# Extract hour, minute, second if needed
df6['Pitstop Hour '] = df6['Pitstop Time of day '].dt.hour
df6['Pitstop Minute'] = df6['Pitstop Time of day '].dt.minute
df6['Pitstop Second'] = df6['Pitstop Time of day '].dt.second

# Example: Calculate time difference between consecutive pitstops
df6['Time Difference'] = df6['Pitstop Time of day
'].diff().dt.total_seconds()

def convert_to_seconds(time_str):
    if 'lap' in time_str:
        return None  # Or handle lap counts differently if needed
    elif 'DNF' in time_str:
        return None  # Or handle DNF (Did Not Finish) differently
    else:
        try:
            return float(time_str.rstrip('s').lstrip('+'))
        except ValueError:
            return None  # Handle any other unexpected formats

df6['Race Time/Retired'] = df6['Race Time/Retired
'].apply(convert_to_seconds)

def is_lap_count(time_str):
    return 'lap' in time_str

df6['Lap Count'] = df6['Race Time/Retired '].apply(is_lap_count)

def is_dnf(time_str):
    return 'DNF' in time_str

df6['DNF'] = df6['Race Time/Retired '].apply(is_dnf).astype(int)


df6.columns
```

```
Index(['Date', 'driver PTS', 'driver  Position', 'Key', 'Pos', 'No_x',
       'Driver_y', 'Car_y', 'Race Laps', 'Race Time/Retired ', 'Race
points',
       'Year_y', 'Grand Prix_y', 'Detail_y', 'DriverCode_y',
'startgrid No',
       'startgrid Pos', 'startgrid Time', 'Car_y', 'Practice Detail',
       'Driver_y', 'Gap', 'Grand Prix_y', 'Practice Laps', 'Practice
No',
       'Practice Pos', 'Practice Time', 'Year_y', 'Avg Speed',
'Car_x',
       'Detail_x', 'Driver_x', 'DriverCode_y', 'Grand Prix_x',
```

```
       'no of fast Lap ', 'fastlaps no', 'fastlaps Pos ',
       'Time taken in fast laps', 'Time of day', 'Year_x', 'Unnamed:
13',
       'Unnamed: 14', 'Unnamed: 15', 'Unnamed: 16', 'Unnamed: 17',
       'Unnamed: 18', 'Stops', 'No_y', 'Driver_y', 'Car_y', 'pitstop
Lap',
       'Pitstop Time of day ', 'Pitstop Total time ', 'Year_y', 'Grand
Prix_y',
       'Detail_y', 'DriverCode_x', 'Car', 'Detail', 'Driver',
'DriverCode_y',
       'Grand Prix', 'qualifying Laps', 'qualifying No', 'qualifying
Pos',
       'qualifying Q1', 'qualifying Q2', 'qualifying Q3', 'qualifying
Time',
       'Year', 'Practice_Time_seconds', 'startgrid_Time_seconds',
       'Time_taken_in_fast_laps_seconds', 'qualifying_Time\t_seconds',
       'Pitstop Hour ', 'Pitstop Minute', 'Pitstop Second', 'Time
Difference',
       'Race Time/Retired', 'Lap Count', 'DNF'],
      dtype='object')
```

```
df6.drop(['Grand Prix_x', 'Car_x', 'Detail_x', 'Year_x',
'Driver_x','qualifying Q1',
       'qualifying Q2', 'qualifying Q3','Time of day', 'Unnamed: 13',
'Unnamed: 14',
       'Unnamed: 15', 'Unnamed: 16', 'Unnamed: 17', 'Unnamed:
18','Car_y','Grand Prix_y','Year_y','No_y','Detail_y','startgrid
Time','qualifying Time','Time taken in fast laps','Practice
Time','Race Time/Retired '], axis=1, inplace=True)
```

```
columns_to_keep = [
    'Date', 'Car','driver PTS','driver  Position','Grand Prix',
'Year','Pos', 'Race Laps', 'Race points',
    'Driver', 'DriverCode_x', 'driver PTS', 'driver  Position', 'Avg
Speed', 'Practice Laps','Practice Pos','no of fast Lap ',
    'startgrid No', 'startgrid Pos', 'qualifying Pos', 'qualifying
Laps',
    'Stops', 'Practice_Time_seconds', 'startgrid_Time_seconds',
       'Time_taken_in_fast_laps_seconds',  'Lap Count', 'DNF',
'Pitstop Hour ',
       'Pitstop Minute', 'Pitstop Second'
]

# Select only the columns you need
df6_selected = df6[columns_to_keep]

print(df6_selected.head())  # Display the first few rows to verify
```

```
        Date           Car  driver PTS driver  Position Grand Prix
Year Pos  \
```

```
0  17-Apr-94  Minardi Ford         0.0               DNF    Pacific
1994  NC
1  17-Apr-94  Minardi Ford         0.0               DNF    Pacific
1994  NC
2  17-Apr-94  Minardi Ford         0.0               DNF    Pacific
1994  NC
3  17-Apr-94  Minardi Ford         0.0               DNF    Pacific
1994  NC
4  17-Apr-94  Minardi Ford         0.0               DNF    Pacific
1994  NC

   Race Laps  Race points            Driver DriverCode_x  driver
PTS  \
0       69.0          0.0  Michele Alboreto          ALB         0.0

1       69.0          0.0  Michele Alboreto          ALB         0.0

2       69.0          0.0  Michele Alboreto          ALB         0.0

3       69.0          0.0  Michele Alboreto          ALB         0.0

4       69.0          0.0  Michele Alboreto          ALB         0.0


  driver  Position  Avg Speed  Practice Laps  Practice Pos  no of fast
Lap   \
0              DNF        NaN           21.0            11
44.0
1              DNF        NaN           21.0            11
44.0
2              DNF        NaN           21.0            11
44.0
3              DNF        NaN           22.0            21
44.0
4              DNF        NaN           22.0            21
44.0

   startgrid No  startgrid Pos qualifying Pos  qualifying Laps  Stops
\
0            24             15             15             20.0      1

1            24             15             15             20.0      2

2            24             15             15             20.0      3

3            24             15             15             20.0      1

4            24             15             15             20.0      2


   Practice_Time_seconds  startgrid_Time_seconds  \
```

```
0                    74.0                     73.0
1                    74.0                     73.0
2                    74.0                     73.0
3                    74.7                     73.0
4                    74.7                     73.0

   Time_taken_in_fast_laps_seconds  Lap Count  DNF  Pitstop Hour   \
0                             76.0      False    1             14
1                             76.0      False    1             14
2                             76.0      False    1             15
3                             76.0      False    1             14
4                             76.0      False    1             14

   Pitstop Minute  Pitstop Second
0              24              39
1              54              52
2              22              17
3              24              39
4              54              52
```

```
# Calculate average 'Avg Speed' for each driver
driver_avg_speed = df6.groupby('Car')['Avg Speed'].mean()

# Fill NaN values in 'Avg Speed' based on driver average
df6['Avg Speed'] = df6.apply(lambda row: driver_avg_speed[row['Car']]
if pd.isna(row['Avg Speed']) else row['Avg Speed'], axis=1)

nan_count = df6_selected['Avg Speed'].isna().sum()
nan_count
```

```
2916
```

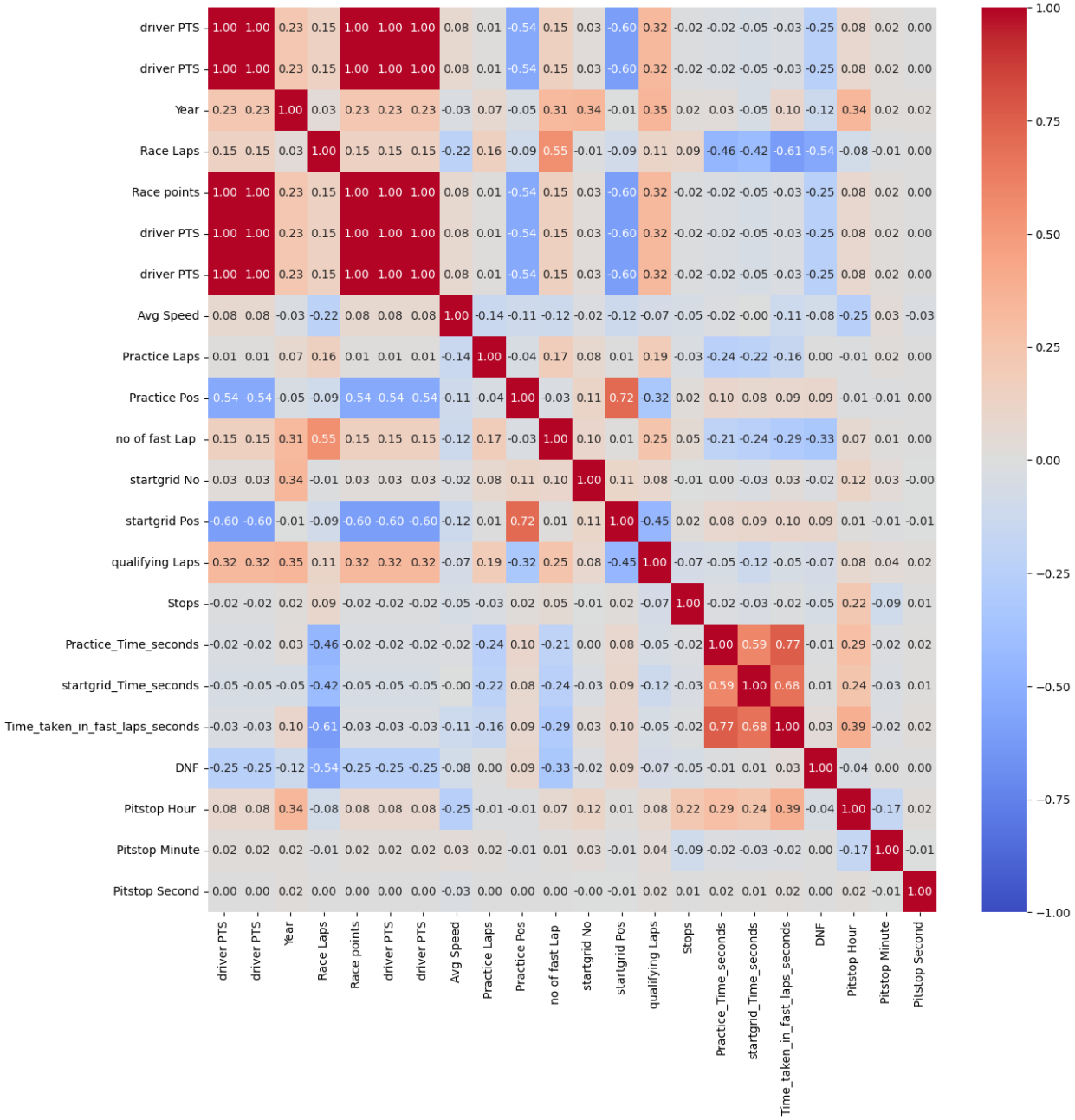
```
df6_selected.dropna(inplace=True)
```

```
<ipython-input-61-9f600063bb99>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  df6_selected.dropna(inplace=True)
```

```
#feature selection
# Identify numeric columns (excluding string columns)
numeric_columns =
df6_selected.select_dtypes(include=['number']).columns

# Calculate the correlation matrix
correlation_matrix = df6_selected[numeric_columns].corr()
import seaborn as sns
import matplotlib.pyplot as plt
```

```
# Set up the matplotlib figure
plt.figure(figsize=(14, 14))

# Generate a heatmap with the numeric correlation matrix
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm',
fmt=".2f", vmin=-1, vmax=1)
```

```
<Axes: >
```



```
df6_selected.describe()
```


```
{"summary":"{\n  \"name\": \"df6_selected\",\n  \"rows\": 8,\n
\"fields\": [\n    {\n      \"column\": \"driver PTS\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
16764.97529386007,\n        \"min\": 0.0,\n        \"max\": 47428.0,\n
\"num_unique_values\": 6,\n        \"samples\": [\n          47428.0,\
n          4.155699165050181,\n          50.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Year\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 16173.501033325976,\n
\"min\": 7.043956628802689,\n        \"max\": 47428.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
2012.3995951758454,\n          2013.0,\n          47428.0\n        ],\
n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Race Laps\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
16751.86057599224,\n        \"min\": 2.0,\n        \"max\": 47428.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
57.834591380619045,\n          57.0,\n          47428.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Race points\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
16764.97529386007,\n        \"min\": 0.0,\n        \"max\": 47428.0,\n
\"num_unique_values\": 6,\n        \"samples\": [\n          47428.0,\
n          4.155699165050181,\n          50.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"driver PTS\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
16764.97529386007,\n        \"min\": 0.0,\n        \"max\": 47428.0,\n
\"num_unique_values\": 6,\n        \"samples\": [\n          47428.0,\
n          4.155699165050181,\n          50.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Avg Speed\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
16708.861376359968,\n        \"min\": 19.596900072666937,\n
\"max\": 47428.0,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          202.64333488656487,\n          202.871,\n
47428.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Practice Laps\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 16760.560349729465,\n        \"min\":
2.0,\n        \"max\": 47428.0,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          21.993737876359955,\n          21.0,\n
47428.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Practice Pos\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 16764.346055959777,\n        \"min\":
1.0,\n        \"max\": 47428.0,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          10.742599308425403,\n          11.0,\n
```

```
47428.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"no
of fast Lap \",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 16754.54818721142,\n        \"min\":
2.0,\n        \"max\": 47428.0,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          43.29733490764949,\n          46.0,\n
47428.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"startgrid No\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 16759.390785730702,\n        \"min\":
1.0,\n        \"max\": 47428.0,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          18.163595344522225,\n          14.0,\n
47428.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"startgrid Pos\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 16764.66451092639,\n        \"min\":
1.0,\n        \"max\": 47428.0,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          10.523551488572151,\n          10.0,\n
47428.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"qualifying Laps\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 16763.458629836907,\n        \"min\":
3.0,\n        \"max\": 47428.0,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          13.326262967023698,\n          12.0,\n
47428.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Stops\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 16767.595278327663,\n        \"min\": 0.8498334153781559,\n
\"max\": 47428.0,\n        \"num_unique_values\": 6,\n
\"samples\": [\n          47428.0,\n          1.7026861769418908,\n
6.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Practice_Time_seconds\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 16694.791694194944,\n
\"min\": 14.297975710497651,\n        \"max\": 47428.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
89.40169730960615,\n          88.2,\n          47428.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"startgrid_Time_seconds\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
16736.166904762762,\n        \"min\": 15.692645447494138,\n
\"max\": 47428.0,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          89.23635405245847,\n          87.1,\n
47428.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Time_taken_in_fast_laps_seconds\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 16736.123232377875,\n
\"min\": 12.025732532207147,\n        \"max\": 47428.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
90.48330311208568,\n          88.9,\n          47428.0\n        ],\n
```

```
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"DNF\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 16768.256306744075,\n
\"min\": 0.0,\n        \"max\": 47428.0,\n
\"num_unique_values\": 5,\n        \"samples\": [\n
0.1285527536476343,\n          1.0,\n          0.33470779087043856\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Pitstop Hour \",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
16763.491335201386,\n        \"min\": 1.6745224110075767,\n
\"max\": 47428.0,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          15.149321076157545,\n          15.0,\n
47428.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Pitstop Minute\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 16759.02578325418,\n        \"min\":
0.0,\n        \"max\": 47428.0,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          27.661381462427258,\n          27.0,\n
47428.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Pitstop Second\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 16758.498356926124,\n        \"min\":
0.0,\n        \"max\": 47428.0,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          29.616639959517585,\n          30.0,\n
47428.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```

```
df6.head(1)
```

```
{"type":"dataframe","variable_name":"df6"}
```

```
df6_selected.drop(['Date','Car','Driver','driver  Position','Grand
Prix','DriverCode_x','qualifying Pos','driver PTS'], axis=1,
inplace=True)
```

```
<ipython-input-65-9fe9e0c1f684>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  df6_selected.drop(['Date','Car','Driver','driver  Position','Grand
Prix','DriverCode_x','qualifying Pos','driver PTS'], axis=1,
inplace=True)
```

```
df6_selected.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 47428 entries, 2916 to 53990
Data columns (total 20 columns):
 #   Column                          Non-Null Count  Dtype
```

```
---  ------                                 --------------  -----
 0   Year                                   47428 non-null  int64
 1   Pos                                    47428 non-null  object
 2   Race Laps                              47428 non-null  float64
 3   Race points                            47428 non-null  float64
 4   Avg Speed                              47428 non-null  float64
 5   Practice Laps                          47428 non-null  float64
 6   Practice Pos                           47428 non-null  int64
 7   no of fast Lap                         47428 non-null  float64
 8   startgrid No                           47428 non-null  int64
 9   startgrid Pos                          47428 non-null  int64
 10  qualifying Laps                        47428 non-null  float64
 11  Stops                                  47428 non-null  int64
 12  Practice_Time_seconds                  47428 non-null  float64
 13  startgrid_Time_seconds                 47428 non-null  float64
 14  Time_taken_in_fast_laps_seconds  47428 non-null  float64
 15  Lap Count                              47428 non-null  bool
 16  DNF                                    47428 non-null  int64
 17  Pitstop Hour                           47428 non-null  int32
 18  Pitstop Minute                         47428 non-null  int32
 19  Pitstop Second                         47428 non-null  int32
dtypes: bool(1), float64(9), int32(3), int64(6), object(1)
memory usage: 6.7+ MB
```

```
df6_selected.columns
```

```
Index(['Year', 'Pos', 'Race Laps', 'Race points', 'Avg Speed',
'Practice Laps',
       'Practice Pos', 'no of fast Lap ', 'startgrid No', 'startgrid
Pos',
       'qualifying Laps', 'Stops', 'Practice_Time_seconds',
       'startgrid_Time_seconds', 'Time_taken_in_fast_laps_seconds',
       'Lap Count', 'DNF', 'Pitstop Hour ', 'Pitstop Minute',
       'Pitstop Second'],
      dtype='object')
```

```
df6_selected.head(5)
```


```
{"summary":"{\n  \"name\": \"df6_selected\",\n  \"rows\": 47428,\n
\"fields\": [\n    {\n      \"column\": \"Year\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
7,\n        \"min\": 1998,\n        \"max\": 2022,\n
\"num_unique_values\": 25,\n        \"samples\": [\n          2006,\n
2014,\n          1998\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Pos\",\n      \"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 27,\n        \"samples\": [\n          \"11\",\
n          \"4\",\n          \"2\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Race Laps\",\n
```

```
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
11.660107710565033,\n        \"min\": 2.0,\n        \"max\": 78.0,\n
\"num_unique_values\": 77,\n        \"samples\": [\n          63.0,\n
57.0,\n          66.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Race points\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 6.433671481744911,\n        \"min\":
0.0,\n        \"max\": 50.0,\n        \"num_unique_values\": 27,\n
\"samples\": [\n          5.0,\n          12.0,\n          0.5\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Avg Speed\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
19.596900072666937,\n        \"min\": 91.61,\n        \"max\":
257.32,\n        \"num_unique_values\": 7075,\n        \"samples\": [\
n          202.808,\n          215.454,\n          195.447\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Practice Laps\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 8.993353580903191,\n        \"min\":
2.0,\n        \"max\": 56.0,\n        \"num_unique_values\": 54,\n
\"samples\": [\n          18.0,\n          54.0,\n          48.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Practice Pos\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
6,\n        \"min\": 1,\n        \"max\": 29,\n
\"num_unique_values\": 29,\n        \"samples\": [\n          26,\n
1,\n          15\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"no
of fast Lap \",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 15.888596141960154,\n        \"min\":
2.0,\n        \"max\": 78.0,\n        \"num_unique_values\": 77,\n
\"samples\": [\n          31.0,\n          9.0,\n          7.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"startgrid No\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
18,\n        \"min\": 1,\n        \"max\": 99,\n
\"num_unique_values\": 47,\n        \"samples\": [\n          26,\n
47,\n          27\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"startgrid Pos\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 6,\n        \"min\": 1,\n
\"max\": 24,\n        \"num_unique_values\": 24,\n        \"samples\":
[\n          16,\n          7,\n          12\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"qualifying Laps\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
6.18246020232189,\n        \"min\": 3.0,\n        \"max\": 35.0,\n
\"num_unique_values\": 33,\n        \"samples\": [\n          35.0,\n
23.0,\n          13.0\n        ],\n        \"semantic_type\": \"\",\n
```

\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"Stops\",\n   \"properties\": {\n   \"dtype\": \"number\",\n
\"std\": 0,\n   \"min\": 1,\n   \"max\": 6,\n
\"num_unique_values\": 6,\n   \"samples\": [\n   1,\n
2,\n   6\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"Practice_Time_seconds\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 14.297975710497651,\n
\"min\": 63.7,\n   \"max\": 1100.6,\n
\"num_unique_values\": 668,\n   \"samples\": [\n   76.1,\n
99.0,\n   106.8\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"startgrid_Time_seconds\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 15.692645447494138,\n
\"min\": 62.9,\n   \"max\": 208.4,\n
\"num_unique_values\": 697,\n   \"samples\": [\n   93.4,\n
95.7,\n   118.2\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"Time_taken_in_fast_laps_seconds\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 12.025732532207147,\n
\"min\": 65.6,\n   \"max\": 202.3,\n
\"num_unique_values\": 561,\n   \"samples\": [\n   118.1,\
n   78.7,\n   83.1\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"Lap Count\",\n
\"properties\": {\n   \"dtype\": \"boolean\",\n
\"num_unique_values\": 2,\n   \"samples\": [\n   true,\n
false\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"DNF\",\n   \"properties\": {\n   \"dtype\": \"number\",\n
\"std\": 0,\n   \"min\": 0,\n   \"max\": 1,\n
\"num_unique_values\": 2,\n   \"samples\": [\n   0,\n
1\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"Pitstop Hour \",\n   \"properties\": {\n   \"dtype\":
\"int32\",\n   \"num_unique_values\": 11,\n   \"samples\":
[\n   20,\n   14\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"Pitstop Minute\",\n   \"properties\": {\n
\"dtype\": \"int32\",\n   \"num_unique_values\": 60,\n
\"samples\": [\n   50,\n   4\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"Pitstop Second\",\n
\"properties\": {\n   \"dtype\": \"int32\",\n
\"num_unique_values\": 60,\n   \"samples\": [\n   39,\n
41\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   }\n]\
n}","type":"dataframe","variable_name":"df6_selected"}

```
df6_selected['Pos'].unique()
```

```
array(['NC', '9', '5', '6', '10', '12', '7', '3', '11', '2', '1',
'14',
       '8', '4', '13', '17', '15', '16', '18', '19', '20', '21', '22',
       '23', '24', 'DQ', 'EX'], dtype=object)
```

```
def position_tranformation(time_str):
    if 'NC' in time_str:
      return 0
    elif 'DQ' in time_str:
      return 0
    elif 'EX' in time_str:
      return 0
    else:
      return int(time_str)

df6_selected['Pos']=df6_selected['Pos'].apply(position_tranformation)
```

```
<ipython-input-70-395b839909df>:11: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy

df6_selected['Pos']=df6_selected['Pos'].apply(position_tranformation)

df6_selected.dtypes
```

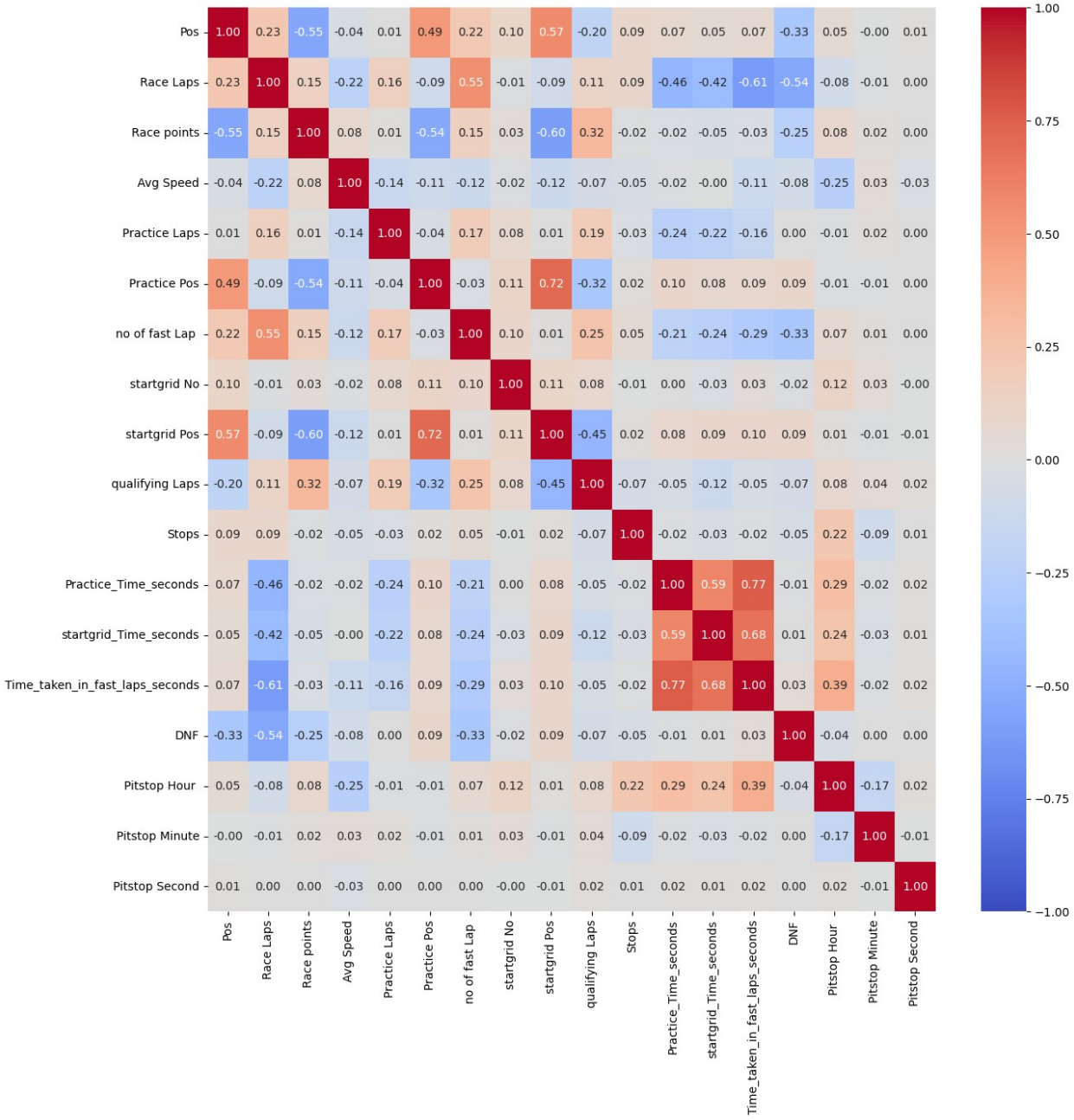
```
Year                                  int64
Pos                                   int64
Race Laps                           float64
Race points                         float64
Avg Speed                           float64
Practice Laps                       float64
Practice Pos                          int64
no of fast Lap                      float64
startgrid No                          int64
startgrid Pos                         int64
qualifying Laps                     float64
Stops                                 int64
Practice_Time_seconds               float64
startgrid_Time_seconds              float64
Time_taken_in_fast_laps_seconds     float64
Lap Count                            bool
DNF                                   int64
Pitstop Hour                          int32
Pitstop Minute                        int32
Pitstop Second                        int32
dtype: object
```

```
df6_selected.drop(['Year'], axis=1, inplace=True)
```

```
<ipython-input-72-c8b1ee70a26f>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  df6_selected.drop(['Year'], axis=1, inplace=True)
```

```
#feature selection
# Identify numeric columns (excluding string columns)
numeric_columns =
df6_selected.select_dtypes(include=['number']).columns

# Calculate the correlation matrix
correlation_matrix = df6_selected[numeric_columns].corr()
import seaborn as sns
import matplotlib.pyplot as plt
# Set up the matplotlib figure
plt.figure(figsize=(14, 14))

# Generate a heatmap with the numeric correlation matrix
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm',
fmt=".2f", vmin=-1, vmax=1)
```

```
<Axes: >
```



MODEL BUILDING

```
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Standardize the features for Logistic Regression
scaler = StandardScaler()
X = df6_selected.drop('Pos', axis=1)
y=df6_selected['Pos']
```

```
X_scaled = scaler.fit_transform(X)

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y,
test_size=0.2, random_state=42)

# Create a Logistic Regression model
logreg_model = LogisticRegression(max_iter=1000)

# Train the model
logreg_model.fit(X_train, y_train)

# Predictions
logreg_predictions = logreg_model.predict(X_test)

# Accuracy
logreg_accuracy = accuracy_score(y_test, logreg_predictions)

logreg_accuracy
```

```
0.36348302761965
```

```
from sklearn.metrics import accuracy_score
from sklearn.ensemble import RandomForestClassifier


# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Create a Random Forest classifier
rf_model = RandomForestClassifier(n_estimators=150, random_state=42)

# Train the model
rf_model.fit(X_train, y_train)

# Predictions
rf_predictions = rf_model.predict(X_test)

# Accuracy
rf_accuracy = accuracy_score(y_test, rf_predictions)
print(f"Random Forest Accuracy: {rf_accuracy}")
```

```
Random Forest Accuracy: 0.9918827746152225
```

```
from sklearn.metrics import classification_report

# Get classification report
report = classification_report(y_test, rf_predictions)

# Print the report which includes F1-score
print(report)
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       971
           1       1.00      1.00      1.00       552
           2       1.00      1.00      1.00       487
           3       1.00      1.00      1.00       504
           4       1.00      1.00      1.00       536
           5       1.00      0.98      0.99       494
           6       1.00      1.00      1.00       502
           7       1.00      0.99      0.99       514
           8       0.99      1.00      0.99       475
           9       0.98      1.00      0.99       448
          10       0.99      0.99      0.99       523
          11       0.98      0.99      0.99       462
          12       0.99      0.97      0.98       477
          13       0.97      0.98      0.98       503
          14       0.97      0.99      0.98       448
          15       0.99      0.98      0.99       424
          16       0.99      0.98      0.98       340
          17       0.99      0.99      0.99       327
          18       1.00      1.00      1.00       208
          19       1.00      0.98      0.99       177
          20       1.00      1.00      1.00        62
          21       1.00      1.00      1.00        26
          22       1.00      1.00      1.00        23
          23       1.00      1.00      1.00         3

    accuracy                           0.99      9486
   macro avg       0.99      0.99      0.99      9486
weighted avg       0.99      0.99      0.99      9486
```

```
from sklearn.metrics import confusion_matrix

# Assuming you have y_test (ground truth labels) and y_pred (model
predictions)

# Generate confusion matrix
cm = confusion_matrix(y_test, rf_predictions)

# Print the confusion matrix
print(cm)
```

```
[[971   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0
    0   0   0   0   0   0]
 [  0 551   1   0   0   0   0   0   0   0   0   0   0   0   0   0
0
    0   0   0   0   0   0]
 [  0   0 487   0   0   0   0   0   0   0   0   0   0   0   0   0
```

```
0
    0   0   0   0   0   0]
 [  0   0   0 502   0   1   0   0   1   0   0   0   0   0   0   0   0
0
    0   0   0   0   0   0]
 [  0   0   0   0 535   0   1   0   0   0   0   0   0   0   0   0   0
0
    0   0   0   0   0   0]
 [  0   0   0   0   0 486   1   0   0   7   0   0   0   0   0   0   0
0
    0   0   0   0   0   0]
 [  0   0   0   0   0   0 501   0   0   0   1   0   0   0   0   0   0
0
    0   0   0   0   0   0]
 [  0   0   0   0   0   0   0 510   2   1   1   0   0   0   0   0   0
0
    0   0   0   0   0   0]
 [  0   0   0   0   0   1   0   0 474   0   0   0   0   0   0   0   0
0
    0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0 447   0   0   0   0   0   0   1
0
    0   0   0   0   0   0]
 [  1   0   0   0   0   0   0   0   1   0 518   0   0   2   0   0   1
0
    0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   1   1 457   0   0   3   0   0
0
    0   0   0   0   0   0]
 [  1   0   0   0   0   0   0   0   0   0   0   3 463   5   3   1   0
1
    0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   2   0   0   0   2   2 494   3   0   0
0
    0   0   0   0   0   0]
 [  2   0   0   0   0   0   0   0   0   0   0   0   0   2 442   0   0
2
    0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   1   0   1   2 417   2
0
    1   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   2   0   3   1   0 334
0
    0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   2   0
325
    0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0
```

```
  208   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   1   0   0   2   1
0
    0 173   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0
    0   0  62   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0
    0   0   0  26   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0
    0   0   0   0  23   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0
    0   0   0   0   0   3]]

df6_selected.to_csv("download.csv")

predictions = rf_model.predict(np.array([[41, 2, 200,
26,9,28,14,12,11,1,92.5,93.5,98.6,0,1,12,50,59]]))
predictions

/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439:
UserWarning: X does not have valid feature names, but
RandomForestClassifier was fitted with feature names
  warnings.warn(

array([0])

predictions[0]

0

# save
import joblib
joblib.dump(rf_model, "my_random_forest.joblib")
# load
loaded_rf = joblib.load("/content/my_random_forest.joblib")
```

PREDICTING THE POSITION BASED VARIOUS TESTDATA

```
print(rf_model.predict(np.array([[41, 0, 201.214, 26, 9, 28, 14, 12,
11, 1, 92, 93.2, 94.9, 1, 1, 14, 50, 39]])))
print(rf_model.predict(np.array([[41, 0, 201.214, 9, 8, 28, 14, 12,
11, 1, 96.1, 93.2, 94.9, 1, 1, 14, 50, 39 ]])))
print(rf_model.predict(np.array([[71, 0, 191.647, 17, 13, 63, 14, 15,
```

```
12, 1, 79.7, 79.4, 80.6, 1, 0, 14, 0, 50]]))
print(rf_model.predict(np.array([[71, 0, 191.647, 42, 9, 63, 14, 15,
12, 1, 80, 79.4, 80.6, 1, 0, 14, 0, 50] ]))
print(rf_model.predict(np.array([[72, 2, 172.274, 30, 7, 44, 14, 11,
11, 1, 89.2, 87.8, 89, 0, 0, 13, 43, 52 ]])))
print(rf_model.predict(np.array([[72, 2, 172.274, 30, 7, 44, 14, 11,
11, 2, 89.2, 87.8, 89, 0, 0, 14, 18, 48 ]])))
print(rf_model.predict(np.array([[72, 2, 172.274, 14, 10, 44, 14, 11,
11, 1, 90.9, 87.8, 89, 0, 0, 13, 43, 52 ]])))
print(rf_model.predict(np.array([[72, 2, 172.274, 14, 10, 44, 14, 11,
11, 2, 90.9, 87.8, 89, 0, 0, 14, 18, 48 ]])))
print(rf_model.predict(np.array([[61, 1, 196.346, 26, 11, 23, 14, 12,
11, 1, 87.9, 88.2, 90.4, 1, 0, 14, 36, 55 ]])))
print(rf_model.predict(np.array([[61, 1, 196.346, 26, 11, 23, 14, 12,
11, 2, 87.9, 88.2, 90.4, 1, 0, 15, 4, 41 ]])))
print(rf_model.predict(np.array([[61, 1, 196.346, 30, 11, 23, 14, 12,
11, 1, 89.5, 88.2, 90.4, 1, 0, 14, 36, 55 ]])))
print(rf_model.predict(np.array([[61, 1, 196.346, 30, 11, 23, 14, 12,
11, 2, 89.5, 88.2, 90.4, 1, 0, 15, 4, 41 ]])))
print(rf_model.predict(np.array([[63, 0, 198.683, 27, 13, 31, 14, 14,
11, 1, 83.4, 83.3, 85.7, 1, 0, 14, 38, 32 ]])))
print(rf_model.predict(np.array([[63, 0, 198.683, 27, 13, 31, 14, 14,
11, 2, 83.4, 83.3, 85.7, 1, 0, 14, 53, 18 ]])))
print(rf_model.predict(np.array([[63, 0, 198.683, 4, 9, 31, 14, 14,
11, 1, 84.3, 83.3, 85.7, 1, 0, 14, 38, 32 ]])))
print(rf_model.predict(np.array([[63, 0, 198.683, 4, 9, 31, 14, 14,
11, 2, 84.3, 83.3, 85.7, 1, 0, 14, 53, 18 ]])))
print(rf_model.predict(np.array([[72, 0, 143.379, 20, 11, 51, 14, 11,
12, 1, 83.8, 82.3, 84.5, 0, 1, 15, 16, 2] ]))
print(rf_model.predict(np.array([[72, 0, 143.379, 20, 11, 51, 14, 11,
12, 2, 83.8, 82.3, 84.5, 0, 1, 15, 52, 3 ]])))
print(rf_model.predict(np.array([[72, 0, 143.379, 35, 12, 51, 14, 11,
12, 1, 84.9, 82.3, 84.5, 0, 1, 15, 16, 2 ]])))
print(rf_model.predict(np.array([[72, 0, 143.379, 35, 12, 51, 14, 11,
12, 2, 84.9, 82.3, 84.5, 0, 1, 15, 52, 3 ]])))
print(rf_model.predict(np.array([[70, 0, 192.066, 4, 20, 46, 14, 11,
12, 1, 78.9, 76.6, 79.7, 1, 0, 14, 41, 20 ]])))
print(rf_model.predict(np.array([[70, 0, 192.066, 4, 20, 46, 14, 11,
12, 2, 78.9, 76.6, 79.7, 1, 0, 15, 12, 36 ]])))
print(rf_model.predict(np.array([[70, 0, 192.066, 22, 12, 46, 14, 11,
12, 1, 78.2, 76.6, 79.7, 1, 0, 14, 41, 20 ]])))
print(rf_model.predict(np.array([[70, 0, 192.066, 22, 12, 46, 14, 11,
12, 2, 78.2, 76.6, 79.7, 1, 0, 15, 12, 36 ]])))
print(rf_model.predict(np.array([[53, 0, 190.366, 27, 9, 11, 14, 8,
12, 1, 85.8, 85.1, 97.2, 0, 1, 14, 39, 31 ]])))
print(rf_model.predict(np.array([[53, 0, 190.366, 27, 9, 11, 14, 8,
12, 2, 85.8, 85.1, 97.2, 0, 1, 15, 10, 47 ]])))
print(rf_model.predict(np.array([[53, 0, 190.366, 27, 12, 11, 14, 8,
12, 1, 88.1, 85.1, 97.2, 0, 1, 14, 39, 31 ]])))
```

```python
print(rf_model.predict(np.array([[53, 0, 190.366, 27, 12, 11, 14, 8,
12, 2, 88.1, 85.1, 97.2, 0, 1, 15, 10, 47 ]])))
print(rf_model.predict(np.array([[45, 0, 229.636, 23, 8, 43, 14, 11,
12, 1, 104.4, 103.7, 107, 0, 0, 14, 41, 7 ]])))
print(rf_model.predict(np.array([[45, 0, 229.636, 34, 4, 43, 14, 11,
12, 1, 104.5, 103.7, 107, 0, 0, 14, 41, 7 ]])))
print(rf_model.predict(np.array([[76, 0, 175.581, 26, 11, 35, 14, 11,
12, 1, 79.4, 79.2, 81.4, 1, 0, 14, 44, 31]])))
print(rf_model.predict(np.array([[76, 0, 175.581, 26, 11, 35, 14, 11,
12, 2, 79.4, 79.2, 81.4, 1, 0, 15, 13, 48 ]])))
print(rf_model.predict(np.array([[76, 0, 175.581, 22, 13, 35, 14, 11,
12, 1, 82, 79.2, 81.4, 1, 0, 14, 44, 31]])))
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439:
UserWarning: X does not have valid feature names, but
RandomForestClassifier was fitted with feature names
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439:
UserWarning: X does not have valid feature names, but
RandomForestClassifier was fitted with feature names
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439:
UserWarning: X does not have valid feature names, but
RandomForestClassifier was fitted with feature names
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439:
UserWarning: X does not have valid feature names, but
RandomForestClassifier was fitted with feature names
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439:
UserWarning: X does not have valid feature names, but
RandomForestClassifier was fitted with feature names
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439:
UserWarning: X does not have valid feature names, but
RandomForestClassifier was fitted with feature names
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439:
UserWarning: X does not have valid feature names, but
RandomForestClassifier was fitted with feature names
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439:
UserWarning: X does not have valid feature names, but
RandomForestClassifier was fitted with feature names
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439:
UserWarning: X does not have valid feature names, but
RandomForestClassifier was fitted with feature names
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439:
```

```
UserWarning: X does not have valid feature names, but
RandomForestClassifier was fitted with feature names
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439:
UserWarning: X does not have valid feature names, but
RandomForestClassifier was fitted with feature names
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439:
UserWarning: X does not have valid feature names, but
RandomForestClassifier was fitted with feature names
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439:
UserWarning: X does not have valid feature names, but
RandomForestClassifier was fitted with feature names
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439:
UserWarning: X does not have valid feature names, but
RandomForestClassifier was fitted with feature names
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439:
UserWarning: X does not have valid feature names, but
RandomForestClassifier was fitted with feature names
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439:
UserWarning: X does not have valid feature names, but
RandomForestClassifier was fitted with feature names
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439:
UserWarning: X does not have valid feature names, but
RandomForestClassifier was fitted with feature names
  warnings.warn(

[0]
[0]
[9]
[9]
[5]
[5]
[5]
[5]
[6]
[6]
[6]
[6]
[10]
[10]
[10]
[10]
[12]
[12]
```

```
[12]
[12]
[7]
[7]
[7]
[7]
[0]
[0]
[0]
[0]
[10]
[10]
[7]
[7]
[7]

/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439:
UserWarning: X does not have valid feature names, but
RandomForestClassifier was fitted with feature names
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439:
UserWarning: X does not have valid feature names, but
RandomForestClassifier was fitted with feature names
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439:
UserWarning: X does not have valid feature names, but
RandomForestClassifier was fitted with feature names
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439:
UserWarning: X does not have valid feature names, but
RandomForestClassifier was fitted with feature names
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439:
UserWarning: X does not have valid feature names, but
RandomForestClassifier was fitted with feature names
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439:
UserWarning: X does not have valid feature names, but
RandomForestClassifier was fitted with feature names
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439:
UserWarning: X does not have valid feature names, but
RandomForestClassifier was fitted with feature names
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439:
UserWarning: X does not have valid feature names, but
RandomForestClassifier was fitted with feature names
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439:
UserWarning: X does not have valid feature names, but
```

```
RandomForestClassifier was fitted with feature names
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439:
UserWarning: X does not have valid feature names, but
RandomForestClassifier was fitted with feature names
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439:
UserWarning: X does not have valid feature names, but
RandomForestClassifier was fitted with feature names
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439:
UserWarning: X does not have valid feature names, but
RandomForestClassifier was fitted with feature names
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439:
UserWarning: X does not have valid feature names, but
RandomForestClassifier was fitted with feature names
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439:
UserWarning: X does not have valid feature names, but
RandomForestClassifier was fitted with feature names
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439:
UserWarning: X does not have valid feature names, but
RandomForestClassifier was fitted with feature names
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439:
UserWarning: X does not have valid feature names, but
RandomForestClassifier was fitted with feature names
  warnings.warn(
```

SAVING THE MODEL STATE

```
loaded_rf = joblib.load("/content/my_random_forest.joblib")

print(loaded_rf.predict(np.array([[41, 0, 201.214, 26, 9, 28, 14, 12,
11, 1, 92, 93.2, 94.9, 1, 1, 14, 50, 39]])))

[0]

/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439:
UserWarning: X does not have valid feature names, but
RandomForestClassifier was fitted with feature names
  warnings.warn(
```

Inference and Analytical result

- From the above data and analysis , it is found that ferari has maximum number of wins and also higher participation
- There are maximum Pitstops at 15th min of the race more than normal pitstops
- There is a high correation between position of the race and starting position of the race for example if a person leads the race then there is higher chance that he might win the race
- Also there is higher correlation between pratice points and race points , if a racer performs well in practise session then there is higher propabiblity that he performs good in the actual race .

Models used

Random forest classifier - 99% behaved well even with the testdata Regression model - 33%

Features considered

Pos int64 Position achieved in the race (integer)

Race Laps float64 Number of laps completed in the race (floating-point number)

Race points float64 Points awarded for the race finish position (floating-point number)

Avg Speed float64 Average speed during the race (floating-point number)

Practice Laps float64 Number of laps completed during practice sessions (floating-point number)

Practice Pos int64 Position achieved during practice sessions (integer)

no of fast Lap float64 Number of fastest laps achieved during the race (floating-point number)

startgrid No int64 Starting grid position for the race (integer)

startgrid Pos int64 Position on the starting grid relative to other drivers (integer)

qualifying Laps float64 Number of laps completed during qualifying sessions (floating-point number)

Stops int64 Number of pit stops made during the race (integer)

Practice_Time_seconds float64 Total time spent practicing in seconds (floating-point number)

startgrid_Time_seconds float64 Time taken to reach the starting grid position in seconds (floating-point number)

Time_taken_in_fast_laps_seconds float64 Total time spent in the fastest laps in seconds (floating-point number)

Lap Count bool Boolean value indicating whether data represents a complete lap (True) or not (False)

DNF int64 Indicates whether the driver did not finish (DNF) the race with 1 or 0 (integer)

Pitstop Hour int32 Hour of the pit stop (integer)

Pitstop Minute int32 Minute of the pit stop (integer)

Pitstop Second int32 Second of the pit stop (integer)

AIM of the project

- To find out the race position of the driver based on few inputs