Web App Development Techniques:

1. To build the web app, the Shiny library for R was employed in combination with HTML and CSS to create complex layouts. Initially, layouts were crafted using raw HTML and CSS, and they were later transitioned to Shiny due to the different syntax used in Shiny. For instance, the HTML code `<div class="container">Hello</div>` is equivalent to `tags$div(class = "container", 'Hello')` in Shiny.

2. User inputs in the web app primarily involve dropdown menus to make selections easier and eliminate the need to match user input word-for-word. Subsequent dropdown menus are populated based on prior user actions, such as the population of circuits, which occurs only after the user selects a year. This functionality is implemented using the *observeEvent* function, which waits for the user to press a button. This was essential to restrict web scraping to only the necessary pages, rather than sending multiple requests for all pages, including unnecessary ones.

3. In Shiny, the web app defines 'ui' and 'server' functions. The 'ui' function deals with the layout of the pages where graphs, tables, and images are displayed, while the logic is included in the 'server' function.

4. Maps - The web app utilizes a well-known JavaScript library to integrate and control Leaflet maps in R. This R package simplifies the creation of map widgets using the leaflet() function, offering various options for customizing the widget.

5. Markers and Popups on Maps - Markers on the map are represented using latitude and longitude coordinates and can appear as icons or circles. Markers are added to the map for each circuit's location based on their latitude and longitude, and popups are integrated using Leaflet. These popups provide information about each circuit, including its name, location, and race date, with each popup being generated from the corresponding row of the data frame.

6. Tables - The GT package in R was utilized to render visually appealing tables, enhancing the overall appearance of the data. It also enabled color-coding of the "Positions Gained and Lost" column in the Starting Grid comparison table under insights, making it more intuitive and user-friendly.

Throughout the project, a variety of techniques were employed:

1. Web Scraping: We conducted web scraping using the rvest library from tidyr. We combined it with 'polite' to limit requests and implement a sleep timer. This ensured that requests had proper headers, preventing overwhelming the websites with excessive requests.

2. During the scraping process, various CSS selectors such as ID, class, and attributes were used. This was especially useful when calculating overlaps for each race track in a race.

3. URL Parsing: We used URL parsing to extract latitude and longitude from links and stored this information in separate columns.

4. Automation: To streamline the scraping process over multiple pages, we utilized functions like sapply and map. These functions allowed us to apply functions to vectors or columns in a dataframe. For example, we scraped year-wise circuits from the gpracingstats website, and then each track was mapped to a website containing latitude and longitude information using 'bind\_rows' to merge the dataframes returned by map or sapply functions, which typically returned a list of tables.

5. Error Handling: We employed 'possibly' from the purrr package to make scraping functions safer. This ensured that if a scrape failed, it would return 'NA' rather than causing the entire script to fail.

6. Data Restructuring: We used 'gather' to convert wide data frames scraped from lap-by-lap pages into long data frames. This conversion allowed us to create line plots for overlapping line charts representing head-to-head driver overlaps.

7. Data Merging: To establish relationships between various dataframes, we used left joins with the driver number as the primary key and full joins to merge data related to countries and circuits for creating a race hotness meter gauge chart.

8. Position Finding: We utilized the 'which' function to determine a driver's position after each lap based on the lap-by-lap table extracted from the statsf1 website.

9. Data Manipulation: Techniques like 'rbind' and 'cbind' were used to add rows and columns to existing dataframes. Functions like groupby, mutate was also used to create meaningful relations and plotting data, like groupby was used to show the total number of overlaps per driver or overlaps per circuit for the year.

10. Data Cleaning: Regular expressions (regex) were applied to clean values in fields, such as extracting the year of the race from full dates, as the year was the main focus of the study.

In terms of data visualization:

1. To display races across the years on a map, the Leaflet JavaScript library was used.

2. Ggplot2 and gganimate were the primary tools for creating visualizations.

3. An animated race chart was generated using a horizontal bar chart in combination with gganimate to transition over the laps.

4. For the overlapping chart showing head-to-head comparisons of two drivers, a simple line chart was employed.

5. The race hotness meter, designed as a gauge chart, utilized a basic rectangular chart, which was then transformed into a semi-circle using the 'coord\_polar' function to resemble a gauge.

Challenges:

1. While extracting total overlaps of each circuit, from 1950 to 1960, the circuit INDIANAPOLIS had no lap by lap data, for example - <https://www.statsf1.com/en/1951/indianapolis/tour-par-tour.aspx>, Hence the script that scraped all the circuits stopped and gave error which encouraged us to use possibly to make a safer version of the function that gave NA for all the Indianapolis circuits from 1950 to 1960.
2. Including the animated race chart was one of the challenge, since shiny doesn’t allow to directly render the animated gif plots to the canvas/app. So to overcome the challenge we first had to render the plot then save it in the location of the project and display it using renderImage function from shiny that reads the gif file from directory then displays it on the app. It drastically slowed the process of plotting animated chart since it already used to take a lot of time to render the plot of a higher resolution in 20fps. It was later dealt with adding the graph rendered last time to show up on the app even if the user didn’t ask for anything to get a glimpse of how things will look for the race he needs to see. For improving the performance of the race chart is still being looked upon.
3. Creating complex layout with shiny using HTML and CSS was not easy to begin with, since the syntax shiny provides for adding tags is different, It sometime made the UI looked cluttered and text boxes, drops downs looked un-aligned, which made the app look ugly than helpful.
4. The website we scraped, statsf1.com, uses French language, so the urls, other non-numerical information is in different language which made it harder to scrape, for e.g.: Pages for ‘Italy’ circuit have a link as ‘https://www.statsf1.com/en/2022/italie/tour-par-tour.aspx’. Where ‘italie’ means Italy and ‘tour-par-tour’ means round by round in English.
5. The lap by lap table scraped had empty cells for the drivers that got eliminated or retire during the race, so while using ‘which’ function for getting position of the driver from the row where the driver doesn’t exists returned ‘integer(0)’ as this the normal behaviour for ‘which’ as it returns integer(0) for non-existent values. So overcome this issue we defined our function named ‘integer0test’ that used ‘identical’ function to check whether which returned integer0 or not, if yes then the function returned 0 or NA based on preference, to avoid miscalculations of total overlaps for the drivers.

Documentation

1. Select Race

Objective:

Select Race page is made to visually depicts the circuits with respect to year on an interactive map.

Input

Takes year as an input using a drop down from 1950 to 2023

Under the hood:

Leaflet R package is used to make the map widget the subsequently plotting the circuits through the use of markers. The year-wise circuits was scraped form gpracing stats website then each race track name was mapped to two custom made functions for scraping latitude and longitude from statsf1 website, where the coordinates were stored in a google maps link, which was then parsed using *url\_parse* function in R and coordinates were extracted and stored in data frame and then saved in a csv file for faster plotting on the map. The program then reads the csv as per the year entered by the user and the latitude and longitude columns get plotted using *addMarkers* leaflet function.

1. Live Racing

Objective:

Live Racing page shows the race for a any selected year and circuit as an animated racing bar chart that it helps to visualise positions, overlapping just like a real race.

Input

Year and circuit name as input using drop down.

Under the hood:

The plot was made using ggplot2 and gganimate library that allows transition across scales like time, or any series like laps in our case using *transition\_states* function. The lap-by-lap data was scraped from statsf1 website that shows position of every driver after each lap, we then swapped every driver name with their position and subtracted position of previous lap with the current lap using *which* function. LOGIC GOES HERE.

1. Overlapping Zone

Objective:

The objective of overlapping zone is to provide insights such as total number of overlaps per driver for a particular race. It also helps the user to compare overlaps of two drivers in a head-to-head fashion.

Input

Year and circuit name as input using drop down for overall overlapping analysis.

Driver 1 and Driver 2 names using drop down for head-to-head overlapping comparison.

Under the hood:

The data used for this feature is same lap by lab table scraped for the purpose of *Live Racing.* But the difference between the position of a driver from previous lap to next lap was then added and stored as row in a data frame that recorded all the drivers from the race with their total number of overlaps. For head-to-head comparison the same data was used but this time two drivers names are passed as parameters to a modded version of the overlapping function that then compares the positions of the two drivers with each other to see if the position of one driver is more than or less than, than the position of the other driver per lap. For example, if driver one has 5th starting position and driver two has 8th starting position so if after 1st lap the position of driver two becomes 4th which is less than the first driver than 1 overlap is added to the pool of driver two overlaps, the process then repeats for all the laps of race. Then the two variables storing overlaps of each drivers get returned in a vector then later gets unpacked to show on the UI.

Additional Functionality

In head-to-head comparison, the overlaps for two drivers also displays photos for the two drivers. The links for the images was scraped from the profile pages of the requested drivers as the lap-by-lap table has driver profile’s linked to the header of the table. The images URL were then scraped and displayed on the page using *renderUI* function of shiny that allows to inject html tags on the app, and since the *img* tag helps to directly render image directly from a URL instead of downloading.

1. Race Hotness Meter

Objective:

The aim of this functionality is to show a gauge chart to show the number of overlaps for a circuit year-wise to determine if the race was exciting according to the history of total overlaps from the all races on the circuit since 1950.

Input

Choose a country from the provided options, and then select a circuit from the dropdown menu, which is automatically populated based on your chosen country. Next, select a year from the year dropdown menu, which is also auto-populated based on your chosen circuit. This will display all the years in which races took place on that circuit.

Under the hood:

The data for this function was scraped before hand as it required all the data for all the circuits available on hand before calculating the excitement meter. The data was scraped from statsf1 circuits page where all the circuits are listed which when opened shows all the races for the circuits with date which then hyperlinks to the lap-by-lap table which is ultimately used to calculate the total laps for that circuit for the year, which then added to the data frame, similarly all the races of the circuits with their overlaps were added to a data frame which was later grouped by the circuit name to summarise the total overlaps by summing them, then new columns were mutated such as min overlaps, max overlaps and average for each circuit. The number of overlaps for a selected race then gets plotted on a scale ranging from minimum to maximum for the specific circuit. The percentage for each of the circuit was calculated using- (Number of Overlaps for the selected race – minimum number of overlaps for that circuit) / (maximum overlaps for that circuit – minimum overlaps for that circuit) x 100.

GitHub Information:

As mentioned, the project and all the files associated with the web app can be found on GitHub. You can access the repository at the following link: https://github.com/pixelatedcode01/feed-for-speed. The repository includes all the Jupyter notebooks that were used to test the code, organized within various directories.

Weekly Journal Entry for Data Wrangling

Week 1 –

Lecture:

In the introductory lecture of wrangling, it was taught about the aim/goal and scopes of the subject and why one should study a subject like wrangling, It also guided about the steps or outline of the steps that should be followed when handling large amounts of data like discovering data, structure and clean data, enriching and additional data sources, validating and checking data consistency and publishing and output for downstream usage.

Lab:

1st introductory lab week provided us a proper documentation to set up all the required software and tools for the course, which included, R kernel, Julia Kernel, Python and other required tools like anaconda, git bash and jupyter lab. Jupyter notebooks was also provided that helped us to test installations of the tools.

What I learnt -

Week 2 –

Lecture:

The second lecture went through verifications of the installations again to make it a breeze to work through the data for wrangling and through the course. For every programming language its building blocks are variables and data types, different data types of R was taught which are –

1. Character
2. Numeric
3. Integer
4. Logical
5. Complex

And composite data types like:

1. Vector 1-D array
2. Matrix 2-D array

Studied about Data Frames – What are data frames and its characteristics, how to create a data frame etc.

What I learnt – I learnt about the characteristics of data frame that are columns should have an unique name, each column should have a similar type of data and should of similar datatype. Every row or record should be unique.