Formula 1 Analysis

Thomas Ngo
Nick Rodio
Jeesu Kwon
Arman Bains
Zoe Bennett
Yvette Song

Project Objective

Create an interactive web application describing formula 1 racing data over the past 75 years

Original Dataset

13 csvs, 860 Drivers, 120 columns, 75 years, 26k results

- constructors.csv
- driver_standings.csv
- drivers.csv
- results.csv
- races.csv

Source: Kaggle (Formula 1 Race Data)

- Nationality.csv
- countries.csv

Source: Google

[Google]
[Kaggle]

constructors

constructorId	constructorRef	name	nationality	url
1	mclaren	McLaren	British	http://en.wikipedia.org/wiki/McLaren
2	bmw_sauber	BMW Sauber	German	http://en.wikipedia.org/wiki/BMW_Sauber
3	williams	Williams	British	http://en.wikipedia.org/wiki/Williams_Grand_Prix_Engineering

driver_standings

driverStandingsId	raceId	driverId	points	position	positionText	wins
1	18	1	10	1	1	1
2	18	2	8	2	2	0
3	18	3	6	3	3	0

drivers

driverId	driverRef	number	code	forename	surname	dob	nationality	url
1	hamilton	44	HAM	Lewis	Hamilton	1985-01-07	British	http://en.wikipedia.org/wiki/Lewis_Hamilton
2	heidfeld	\N	HEI	Nick	Heidfeld	1977-05-10	German	http://en.wikipedia.org/wiki/Nick_Heidfeld
3	rosberg	6	ROS	Nico	Rosberg	1985-06-27	German	http://en.wikipedia.org/wiki/Nico_Rosberg

results

resultId	raceId	driverId	constructorId	number	grid	position	positionText	positionOrder	points	laps	time	milliseconds	fastestLap	rank	fastestLapTime	fastestLapSpeed	statusId
1	18	1	1	22	1	1	1	1	10	58	1:34:50.616	5690616	39	2	1:27.452	218.300	1
2	18	2	2	3	5	2	2	2	8	58	5.478	5696094	41	3	1:27.739	217.586	1
3	18	3	3	7	7	3	3	3	6	58	8.163	5698779	41	5	1:28.090	216.719	1

nationality

num_code	alpha_2_code	alpha_3_code	en_short_name	nationality
4	AF	AFG	Afghanistan	Afghan
248	AX	ALA	Åland Islands	Åland Island
8	AL	ALB	Albania	Albanian

countries

country	latitude	longitude	name
AD	42.546245	1.601554	Andorra
AE	23.424076	53.847818	United Arab Emirates

Data Processing

Update Nationality Names

Ex: ‘East German’ => ‘German’, ‘British, UK’ => ‘British’

Pull in Additional Datasets

Used public google csvs to match nationalities, country names and lat-long data

Drop Unused Columns

URLs, codes, dobs, positions, etc and combine first and last name cols

Find Career Wins

Trim results.cvs to winners only and count driver wins to create career_wins.csv

Merge Tables

merge nationality/location data to drivers.csv
merge constructor/lap time data to car_dropdown.csv

Export

Export final four csvs

Final csvs

constructors.csv

constructor_id	make
1	McLaren
2	BMW Sauber
3	Williams

career_wins.csv

driver_id	first_name	last_name	full_name	career_wins
1	Lewis	Hamilton	Lewis Hamilton	103
3	Nico	Rosberg	Nico Rosberg	23
4	Fernando	Alonso	Fernando Alonso	32

car_dropdown.csv

year	make	fastest_lap_speed
2014	Ferrari	196.038
2020	Ferrari	215.019
2024	Ferrari	219.785

drivers.csv

driver_id	first_name	last_name	dob	nationality	country_name	latitude	longitude
1	Lewis	Hamilton	1985-01-07	British	United Kingdom	55.378051	-3.435973
2	Nick	Heidfeld	1977-05-10	German	Germany	51.165691	10.451526
3	Nico	Rosberg	1985-06-27	German	Germany	51.165691	10.451526

Next Steps

01 Populate Database

02 Create the Flask routes for the dashboard

03 Create HTML files

04 Create .js files for visualizations

Population of Database

- Populated our formula1 database in PostgreSQL with updated CSV files

```python
from sqlalchemy import create_engine
import pandas as pd

engine=create_engine('postgresql://postgres:postgres@localhost:5432/formula1')
conn=engine.connect()

# Reading first csv file for drivers
drivers_df=pd.read_csv('Resources/output_data_for_tables/drivers.csv')
# creates schema and inserts records
drivers_df.to_sql('drivers', if_exists='replace', con=conn)

# Reading second csv file for cars
constructors_df=pd.read_csv('Resources/output_data_for_tables/constructors.csv')
# creates schema and inserts records
constructors_df.to_sql('cars', if_exists='replace', con=conn)

# Reading second csv file for career_wins
wins_df=pd.read_csv('Resources/output_data_for_tables/career_wins.csv')
# creates schema and inserts records
wins_df.to_sql('career_wins', if_exists='replace', con=conn)

# Reading second csv file for car_speeds
car_dropdown_df=pd.read_csv('Resources/output_data_for_tables/car_dropdown.csv')
# creates schema and inserts records
car_dropdown_df.to_sql('car_speeds', if_exists='replace', con=conn)

conn.close()
```

SQLAlchemy/ Flask

- Defined our data/visualization routes
- Used select * queries to retrieve information
- Converted results into JSON format

```python
# Setting up the F1 driver data API route
@app.route('/driver_data')
def get_driver_data():
    query=text('''
               SELECT *
               FROM drivers
               ''')
    conn=engine.connect()
    results=conn.execute(query)
    conn.close()
    results=[tuple(row[1:]) for row in results]
    return jsonify(results)

# Setting up the car API route
@app.route('/car_data')
def get_car_data():
    query=text('''
               SELECT *
               FROM cars
               ''')
    conn=engine.connect()
    results=conn.execute(query)
    conn.close()
    results=[tuple(row[1:]) for row in results]
    return jsonify(results)

# Setting up the F1 drivers career wins API route
@app.route('/career_wins_data')
def get_career_wins_data():
    query=text('''
               SELECT *
               FROM career_wins
               ''')
    conn=engine.connect()
    results=conn.execute(query)
    conn.close()
    results=[tuple(row[1:]) for row in results]
    return jsonify(results)

# Setting up the cars fastest lap time over the years 2014-2024 data API route
@app.route('/car_speeds_data')
def get_car_speeds_data():
    query=text('''
               SELECT *
               FROM car_speeds
               ORDER BY make ASC, year ASC
               ''')
    conn=engine.connect()
    results=conn.execute(query)
    conn.close()
    results=[tuple(row[1:]) for row in results]
    return jsonify(results)
```

```python
@app.route('/barchart')
def show_barchart():
    return render_template('barchart.html')

@app.route('/linechart')
def show_linechart():
    return render_template('linechart.html')

@app.route('/maps')
def show_maps():
    return render_template('maps.html')

@app.route('/animation')
def show_animation():
    return render_template('animation.html')

if __name__ == '__main__':
    app.run(debug=True)
```

HTML

- Multiple HTML files for each respective visualization
- Each HTML holds the code for visualizations (bar, line, map, & animation)
- Each HTML is rendered for their own route

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Driver by Country</title>
    <!-- Leaflet CSS -->
    <link rel="stylesheet" href="https://unpkg.com/leaflet/dist/leaflet.css" />
    <!-- Leaflet JavaScript -->
    <script src="https://unpkg.com/leaflet/dist/leaflet.js"></script>
    <!-- D3 library -->
    <script src="https://d3js.org/d3.v7.min.js"></script>

    <style>
        #map {
            height: 600px;
            width: 100%;
        }
    </style>
</head>
<body>
    <h1>Driver by Country</h1>
    <div id="map"></div>
        <!-- Your JavaScript file -->
        <script src="../static/js/maps.js"></script>
</body>
</html>
```

Animation

- Created a div for an animated element
- Initialize a variable position and increment it by 1 pixel each frame
- Call the animate() function on the homepage

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Simple Animation</title>
    <style>
        #animatedElement {
            width: 100px;
            height: 100px;
            background-color: red;
            position: absolute;
        }
    </style>
</head>
<body>
    <div id="animatedElement"></div>

    <script>
        // Get the animated element
        var animatedElement = document.getElementById('animatedElement');

        // Initial position
        var position = 0;

        // Animation loop
        function animate() {
            // Increment position
            position += 1;

            // Update element's position
            animatedElement.style.left = position + 'px';

            // Request next frame
            requestAnimationFrame(animate);
        }

        // Start animation
        animate();
    </script>
</body>
</html>
```

JavaScript

- Created four .js files for each of the three visualizations & car animation
- Each respective HTML file rendered the four .js files

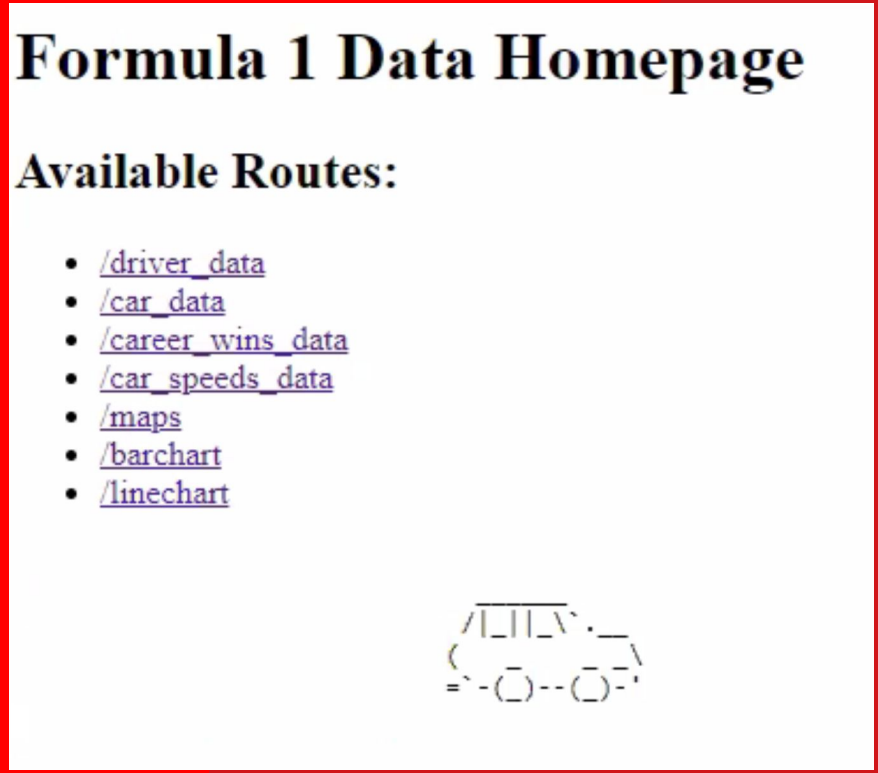
```javascript
// Create a map centered around a specific location
var myMap = L.map('map').setView([0, 0], 2);

// Add a tile layer for the map
L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
    attribution: '&copy; <a href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
}).addTo(myMap);

d3.json('http://127.0.0.1:5000/driver_data').then(data => {
    var driverCountByCountry = {};

    // Loop through the driver data and create markers
    data.forEach(driver => {
        var latitude = driver[6];  // Replace with the actual field name for latitude
        var longitude = driver[7];  // Replace with the actual field name for longitude
        var country = driver[5];  // Replace with the actual field name for country

        if (driverCountByCountry.hasOwnProperty(country)) {
            driverCountByCountry[country]++;
        } else {
            driverCountByCountry[country] = 1;
        }
    });

    // Define a color scale based on the number of drivers (domain: 1 to 166)
    var colorScale = d3.scaleLinear()
        .domain([1, 166])
        .range(["green", "yellow"]);  // Change the range of colors from black to white

    // Add markers to the map with dynamically determined fill color
    data.forEach(driver => {
        var latitude = driver[6];  // Replace with the actual field name for latitude
        var longitude = driver[7];  // Replace with the actual field name for longitude
        var country = driver[5];  // Replace with the actual field name for country

        var fillColor = colorScale(driverCountByCountry[country]);

        L.circle([latitude, longitude], {
            fillOpacity: .25,
            color: 'black',
            weight: 0.5,
            opacity: 0.25,
            fillColor: fillColor,
            radius: driverCountByCountry[country] * 4000
        }).bindPopup(`<b>${driverCountByCountry[country]} Drivers<br><b>Country:</b> ${country}`)
          .addTo(myMap);
    });

        // Create a legend
        var legend = L.control({ position: 'bottomright' });
        legend.onAdd = function () {
            var div = L.DomUtil.create('div', 'info legend');
            div.style.backgroundColor = 'white';
            div.style.padding = '15px';
            div.innerHTML = '<h3>Number of Drivers</h3>' +
                '<i class="square" style="background: green;"></i>Lowest<br>' +
                '<i class="square" style="background: yellow;"></i>Highest<br>';
```

Web App Home

- Use Flask to create homepage displaying API Routes and visualizations

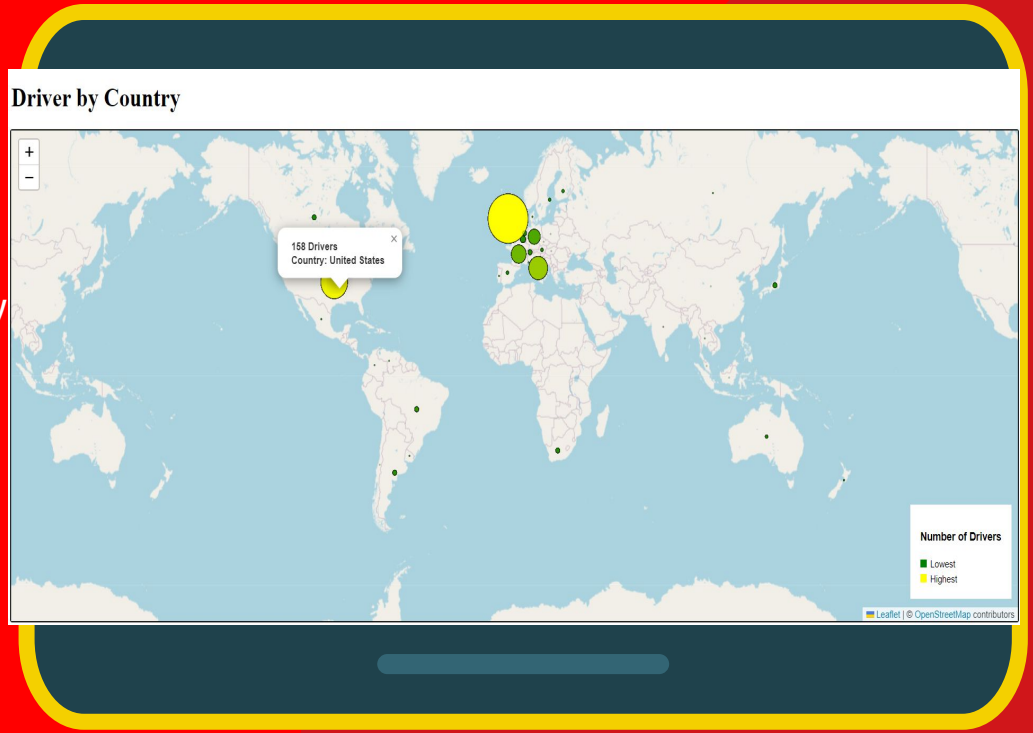
- ASCII Art car drives across the screen

Available Routes:
- JSON Table Data (Routes 1-4)
- Three visualizations (Routes 5-7)



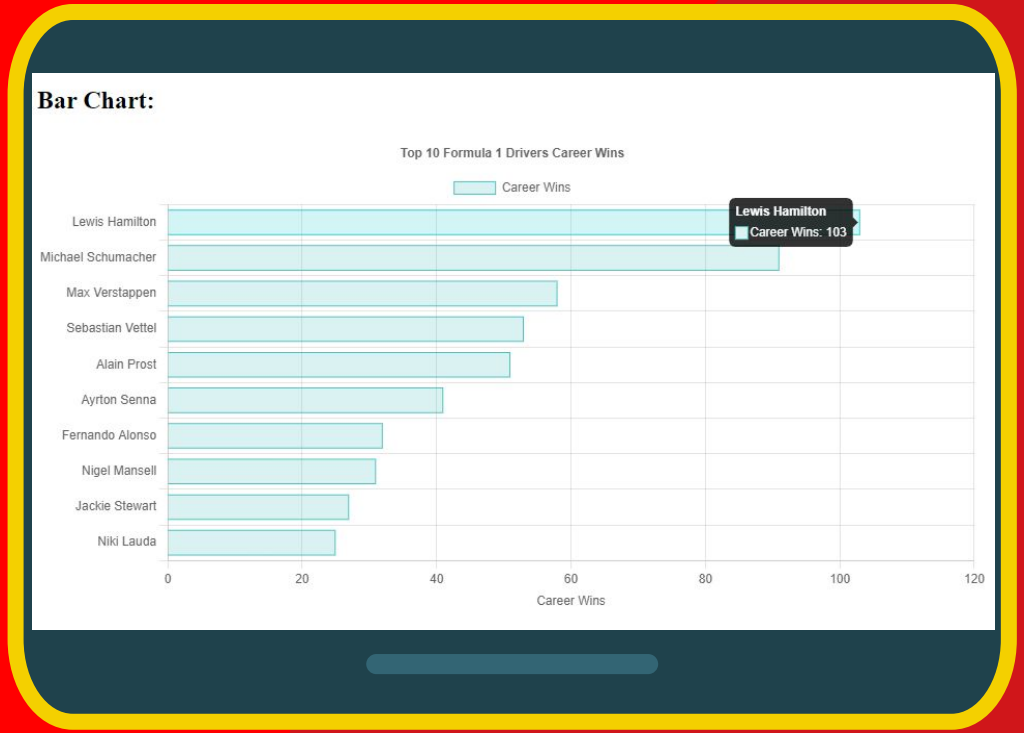
Visualization 1: Map

- Bubble Chart describing driver origin, how many drivers by country
- Clickable pop up for driver number



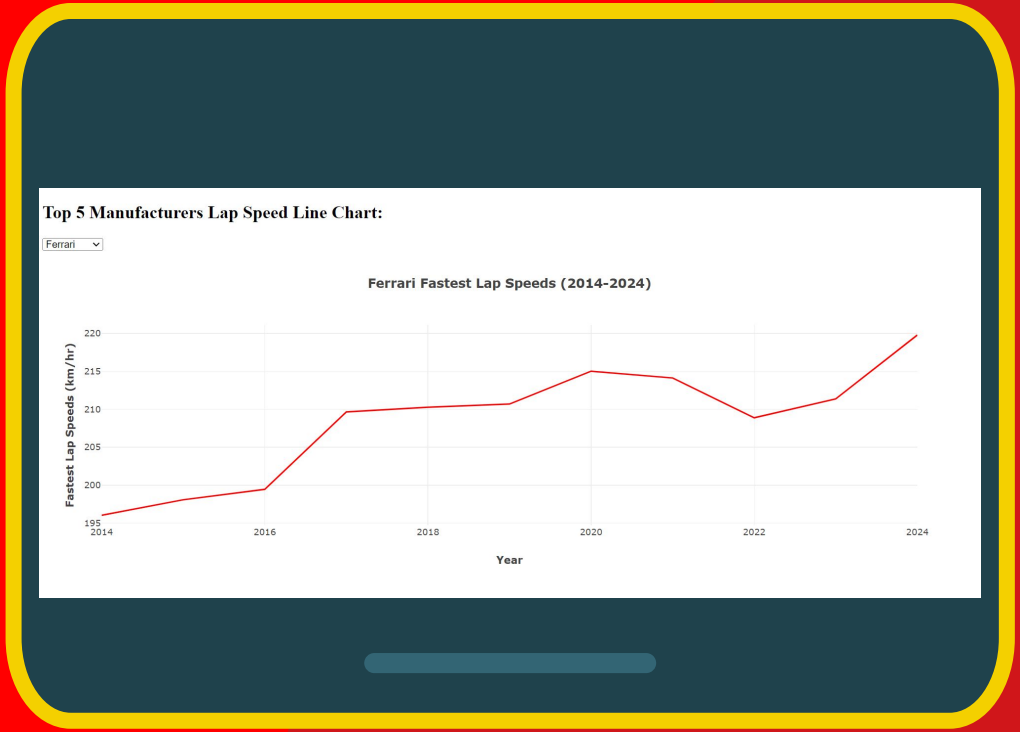
Visualization 2: Bar Chart

- Bar chart showing the top drivers by their career wins
- Hover over for career wins



Visualization 3: Line Chart

- Line chart showing the last 10 years of data for the top five manufacturers' fastest average lap speeds
- Dropdown menu allows user to toggle between each of the top five automakers
- Hover over for average lap speeds



Website Demo

Conclusion

Most successful drivers:

***** Mercedes drivers from the UK *****

Fastest Average Lap Speed % Increase (2014-2024)
Red Bull - 12.75%
Ferrari - 12%
Mclaren - 11.8%
Mercedes - 11%
Williams - 10.2%

Next steps:

- Using a different visualization style for the driver origin map
- Further standardizing the data for the manufacturer over time line chart

THANK YOU!