Final Project Report: Weather & College Football

SI206

Team: Big Steppas 2024

Members: Sriram Kumaran, Emily Jennett, Daniel Vega

Original Goal

Our project's objective is to identify relationships between weather conditions and college football game outcomes by integrating and analyzing data from three APIs. By collecting game statistics such as team performance, game locations, and scores from the College Football API and comparing this with weather data including temperature, wind speed, precipitation, and air quality from weather APIs, we aim to identify potential correlations. More specifically, we seek to determine if various weather conditions impact scoring, win-loss records, and overall team performance. Through statistical analysis and visualizations created with Matplotlib and Seaborn, we will uncover trends and insights that will develop our understanding of how environmental factors impact college football games.

Achieved Goal

We were able to successfully identify relationships between weather conditions and college football game outcomes. The team collected game statistics such as team performance, game locations, and scores from the College Football Data API and combined this with weather data including temperature, wind speed, and precipitation from historical weather APIs. Despite initial challenges with data inconsistencies due to mismatched longitude and latitude values, we resolved these issues and created a database linking football games and corresponding weather conditions. In the end, we uncovered trends such as the influence of extreme wind speeds on scoring and correlations between temperature ranges and total points scored. These insights supported our original hypothesis that environmental factors impact college football games.

Problems Faced

In our project, we encountered several challenges starting with the data collection step, especially when working with the weather API and creating the weather_data table. While we were able to successfully store 100 unique items in our database without duplication, many of the entries were either incomplete or incorrect. For example, only 63 out of 100 weather_id values were stored correctly, the rest showing up as "NULL." Additionally, the precipitation data (rain and snow conditions) was consistently recorded as 0, all temperature values were negative, and the wind speeds were unrealistically high. After a closer analysis of the data, we realized the longitude and latitude data were swapped, affecting the weather precipitation, temperature, and wind speeds.

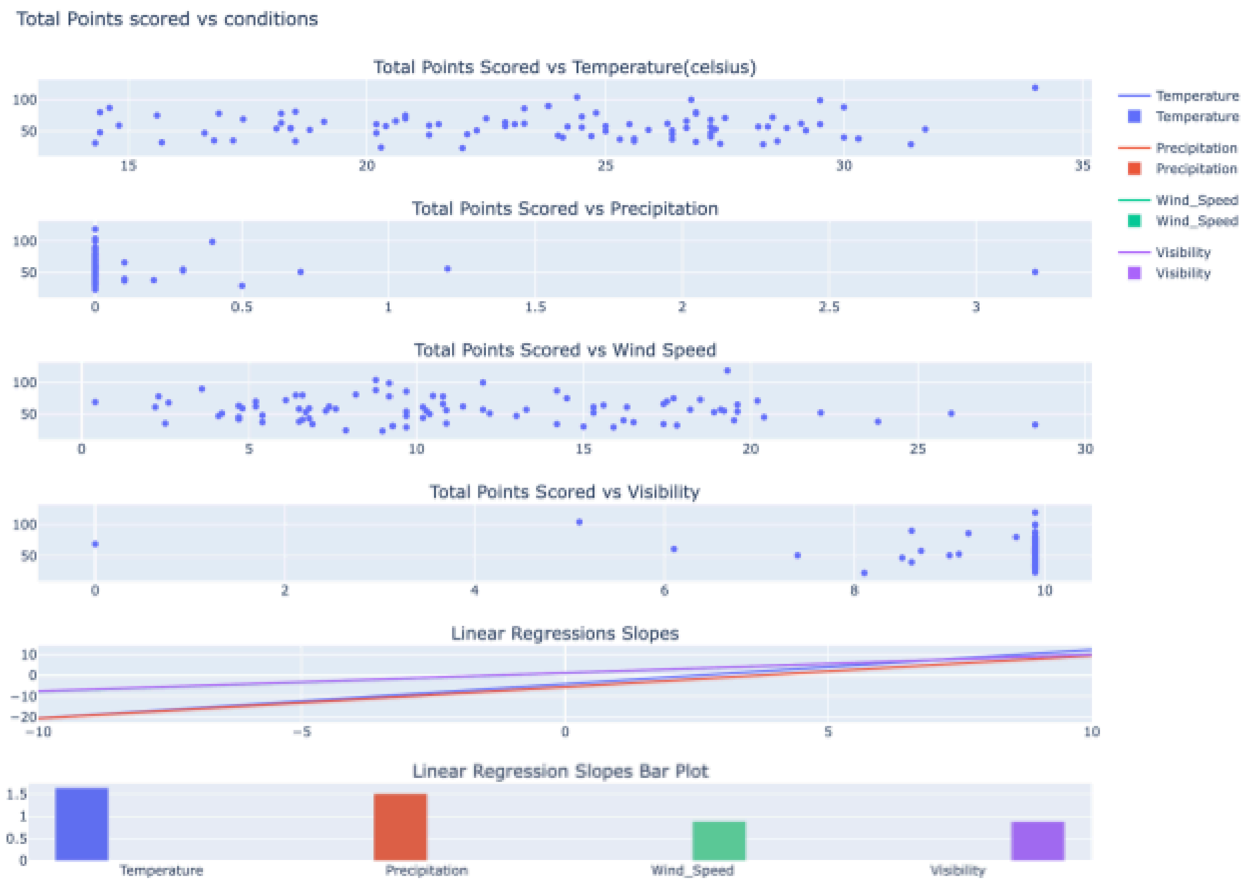
Calculation File (Screenshot)

```python
def fetch_football_weather_data():
    """
    Preform an INNER JOIN to retreive each football game's total points, temp, precipitation, and wind speed.
    """
    join_command = f"""SELECT football_games.total_points,
                        weather_data.temperature,
                        weather_data.precipitation,
                        weather_data.wind_speed,
                        weather_data.visibility
                    FROM football_games
                    INNER JOIN weather_data
                    ON football_games.weather_id = weather_data.weather_id"""
    conn = sqlite3.connect(FOOTBALL_DB_FILENAME)
    cursor = conn.cursor()
    cursor.execute(join_command)
    rows = cursor.fetchall()
    conn.close()
    return list(filter(lambda row: is_valid_row(row), rows))
```

```python
#calculate linear regression
def calc_linear_regression(rows, x_index, y_index):
    """
    Calculate the linear regression (slope and intercept) between two variables.

    Args:
        rows (list of tuples): the data retrieved from the database.
        x_index (int): the index of the independent variable (temp, precipitation, or wind_speed).
        y_index (int): the index of the dependent variable (total_points).

    Returns:
        dict: a dictionary with the slope and intercept.
    """
    # extracting x and y values
    x_values = [row[x_index] for row in rows]
    y_values = [row[y_index] for row in rows]
    # print(x_values)
    # print(y_values)

    # number of data points
    n = len(x_values)

    # calculate sum
    x_sum = sum(x_values)
    # print(x_sum)
    y_sum = sum(y_values)
    # print(y_sum)
    x_sum_squared = sum(x ** 2 for x in x_values)
    # print(x_sum_squared)
    xy_sum = sum(x * y for x, y in zip(x_values, y_values))
    print(xy_sum)

    # calculate slope (m) and intercept (b):
    denominator = (n * x_sum_squared - x_sum ** 2)
    if not denominator == 0:
        m = (n * xy_sum - x_sum * y_sum) / denominator
    else:
        m = 99999999
    # print(m)
    b = (y_sum - m * x_sum) / n
    # print(b)

    return {"slope": m, "intercept": b}
```

```
{
    "linear_regressions": {
        "temperature": {
            "slope": 0.18546456907775505,
            "intercept": 53.26037675200547
        },
        "precipitation": {
            "slope": -2.826855771168894,
            "intercept": 57.854726696351676
        },
        "wind_speed": {
            "slope": -0.316444484752332,
            "intercept": 61.227080608099826
        }
    },
    "data": [
        {
            "total_points": 88,
            "temperature": 30.0,
            "precipitation": 0.0,
            "wind_speed": 8.8,
            "visibility": 9.9
        },
        {
            "total_points": 53,
            "temperature": 27.3,
            "precipitation": 0.0,
            "wind_speed": 6.7,
            "visibility": 9.1
        },
        {
            "total_points": 61,
            "temperature": 23.1,
            "precipitation": 0.0,
            "wind_speed": 15.3,
            "visibility": 9.9
```


Data Visualizations (5 total)



Instructions for running the code:

1. Run the bash script run.sh (it runs all the files in the order)
2. Terminal run command chmod +x run.sh
3. Terminal run command ./run.sh

Code Documentation (includes describing the input and output for each function)

GET DATA

<u>“sqlfunctions.py“ Functions</u>

fetch_games() - The fetch_games() function retrieves data on college football games from the College Football Data API. It takes in two inputs: year (int) and season_type (str defaulting to "regular", specifying season type). The function creates an API request with the provided parameters, sends the request, and returns the game data in JSON format. If errors occur during the request, it will print an error message and return an empty list. The purpose of this function is to collect structured data about football games, including information such as teams, scores, and dates, to be used in further analysis.

fetch_venues() -The fetch_venues() function gathers data on football venues from the College Football Data API. It takes in no inputs, just sending a GET request to the API. This function returns a list of venues in JSON format. If the request does not work, it will print an error message and return an empty list. This function is made to collect details about game venues, including location, longitude, and latitude, which is crucial for linking games to weather data.

combine_data() - combine_data() merges the game and venue data into a single structure. It takes in two inputs, games (list of game dicts) and venues (list of venue dicts). The function creates a lookup dictionary for venues by their id and matches each game to its corresponding venue based on venue_id. The combined data includes game details like teams playing, the scores, and overall results as well as venue information like city, latitude, and longitude. The output is a list of dictionaries, each representing a merged game and venue entry.

create_table() - This function initializes the football_games table in a SQLite database. It creates a table schema to store football game details such as game scores, teams, location, and a foreign key (weather_id) to link it to weather data. Create_table() does not take in inputs and return nothing. This is because it serves to ensure the database structure is in place to store football game data.

addFootballDataToTable() - This function inserts game and venue data into the football_games table. It takes one input: combined_data, which is a list of dictionaries with game and venue information. The function iterates over the data and inserts 100 entries into the database, making sure each game has unique identification by game_id. The main purpose of this function is to ensure the data is correctly stored and populate the database with game data for further analysis in proceeding steps.

fetch_weather() - This function retrieves hourly historical weather data for a given location and date range from the Open-Meteo API. It takes in four string inputs: latitude, longitude, start_date, and end_date. Fetch_weather() sends a request with these parameters, returning the weather data in JSON format. This function is needed for linking weather conditions to football games based on shared locations and times.

create_weather_table() - Create_weather_table() sets up the weather_data table in our SQLite database. It defines a schema to store temperature, precipitation, wind speed, and cloud cover data as well as location and date of the weather conditions. This function takes no inputs and does not return anything, it simply establishes the structure needed to store weather data linked to football games.

find_closest_time_index() - This function determines the time index in a list that is closest to a target time. It takes two string inputs: time_list and target_time. Next, it parses the times and calculates the absolute time difference to find the closest match, returning the index of the closest game time. The purpose of this function is to match game times with the corresponding weather conditions, aligning the weather conditions during a given game.

fetch_football_data_from_db() - Fetch_football_data_from_db() retrieves all rows from the football_games table we made previously, where weather_id is NULL. It returns a list of tuples containing game data such as game_id, game_date, and location information. This function's purpose is to identify games that need weather data added.

addWeatherDataFromDb() - This function fetches weather data for given games stored in the database and updates the rows with weather details. It also retrieves games that are missing weather data, calls fetch_weather() to collect weather information, and then inserts the data into the weather_data table. It updates the football_games table with each corresponding weather_id. This function links football games with weather data to allow us to analyze for potential correlations.

"sqlschemas.py" Functions

create_football_games_table() - This function creates a SQLite database table named football_games, storing information about football game details like the game date, time, location, teams, scores, results, and performance statistics like yards gained, turnovers, and penalties. If the table already exists, the function ensures it is not recreated. No value is returned from this function.

PROCESS THE DATA

processData.py Functions

fetch_football_weather_data() - The fetch_football_weather_data() function retrieves data by doing an SQL INNER JOIN between the football_games and weather_data tables in the SQLite database made in the gathering data step. It combines columns from both tables based on the weather_id key, matching records. The function uses SQL queries to select total_points, temperature, precipitation, and wind_speed values from the joined tables. It then filters out rows with any None values using the helper function is_valid_row, making sure it completes and valid data is returned as a list of tuples.

is_valid_row() - This function checks if all elements in each row are valid, returning False if any value is None and True otherwise. It is a utility function used within fetch_football_weather_data() to ensure that rows containing incomplete data are excluded from the analysis for the visualization step.

calc_linear_regression() - This function computes the linear regression parameters, slope and intercept. It takes as input the list of data rows, the index of the independent variable (x), and the dependent variable (y). It takes x and y values, calculates the sums, and applies it to the linear regression formula. The function handles edge cases such as division by zero, by assigning a placeholder value for the slope if the denominator becomes zero. Lastly, results are returned as a dictionary containing the slope and intercept.

save_to_json_file() - Save_to_json_file() saves the joined data and calculated regression done previously and returns them into a structured JSON file. It formats the data as a list of dictionaries where each dictionary corresponds to a row containing total_points, temperature, precipitation, and wind_speed. The regression results are included in the output JSON under the linear_regressions key. The formatted data is then written to the process_data.json file with indentation for readability, making sure the processed data is easily accessible for the visualization step.

VISUALIZE THE DATA

dataVisualization.py Functions

load_data() - This function loads the JSON data file process_data.json with football and weather information made in the previous step. It reads the file and converts the content into a python dictionary using the json.load() method, making it accessible for further steps in the visualization.

initialize_dict() - initialize_dict() processes the loaded data to extract key values into separate lists. It iterates through the JSON dictionary, populating lists for total points scored, temperature, precipitation, and wind speed. These lists are then returned as a tuple.

set_plot_data() - This function organizes the data into a dictionary (plot_data) with keys (total_points, temperature, precipitation, and wind speed). It uses the tuple from the function initialize_dict() and returns both the dictionary and the tuple for flexibility in subsequent data handling.

linear_regression() - This part computes the linear regression (slope and intercept) for the relationships between total points scored and the three weather variables (temperature, precipitation, and wind speed). It uses the calc_linear_regression from processData.py, and adds a label to each regression result to identify them during plotting.

make_plot() - This function generates 6 visualizations to analyze the relationships between football scores and weather conditions. It creates scatter plots, a linear regression line graph, and a bar plot summarizing regression slopes. The visualizations are combined into a single figure with 6 subplots using Plotly's make_subplots. The visualizations are displayed using the fig.show() method.

Resource Documentation

Date	Issue Description	Location of Resource	Result (did it solve the issue?)
12/06/2024	Our team needed historical weather data (location, temperature, humidity, precipitation, and wind speed) for analysis of correlation between weather and football game scores.	Historical Weather API	Yes, it provided detailed historical weather data within the year (2020) which aligned with our football game data.
12/07/2024	We needed access to this college football game data to identify games' total scores and location, so we can align location of games to location of weather condition.	ESPN API List	Yes, it offered a list of potential endpoints for game data retrieval.
12/09/2024	We needed access to the visibility data from certain locations but were not provided by the initial weather API we used	Visibiltiy API	Yes, if offers a detailed response on the visibility of the location on a specific date