Team Information

SI 206 Final Project Report
Team Name: Motor City Dan Campbell Fan Club
Team Members: Zack Eisman, Liam Kendall
Git Repository: https://github.com/liamkend/SI206FinalProjectMCDCFC

Goals

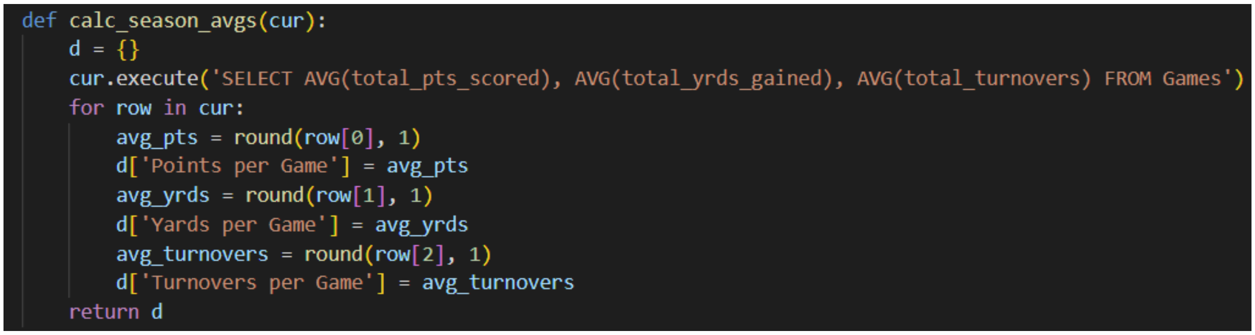
Initial Goals: Our primary goal was to see what impact weather had on certain pro football statistics. To get the football data, we wanted to use Pro Football Reference and scrape an abundance of data including team names, date, city, total points scored, total rushing yards, total passing yards, total yards gained, total turnovers, roof type, and the over/under result. As for the weather data, we planned on using the Weather Stack API and would retrieve data such as city, time, weather type, temperature, wind, precipitation, and visibility.
Once we had this data, we planned to calculate various football averages including average points, rushing yards, passing yards, and turnovers per game by weather conditions. As for visualization, we listed an abundance of possible graphs in our proposal such as total points vs temperature, rushing yards vs precipitation, passing yards vs wind speed and many more.

Successes: Storing the NFL data was mostly successful. We were able to get the team names, date, city, total points scored, total rushing yards, total passing yards, total yards gained, total turnovers, and over/under result for each of the 284 NFL games in the 2022 season. This data was stored into five tables in sqlite. The biggest was titled Games which had the statistics or ids for each game. The four additional tables, Cities, Dates, Teams, and OverUnder were created to avoid duplicate string data.
Additionally, we were mainly successful working with the Weather Stack API. We were able to get the temperature, weather conditions, wind speed, precipitation, and visibility for each date and city of every NFL game. For this, we had to create two tables in sqlite, one called Weather that contained all of this data and one called Types for all of the weather types. Date and city shared the same Dates and Cities tables that were created when storing NFL data.

Setbacks: Our biggest setback was the Pro Football Reference website. Some of the data became extremely hard to scrape as it was hidden by comments within the HTML. So, we pivoted and used ESPN instead which provided us all the same data except for the roof type. This category is very important to NFL games and our data would be more accurate with the information but it would have required scraping an additional website or two to get this information. We also wound up having to pay $10 to access historical weather data from our API which was unexpected.

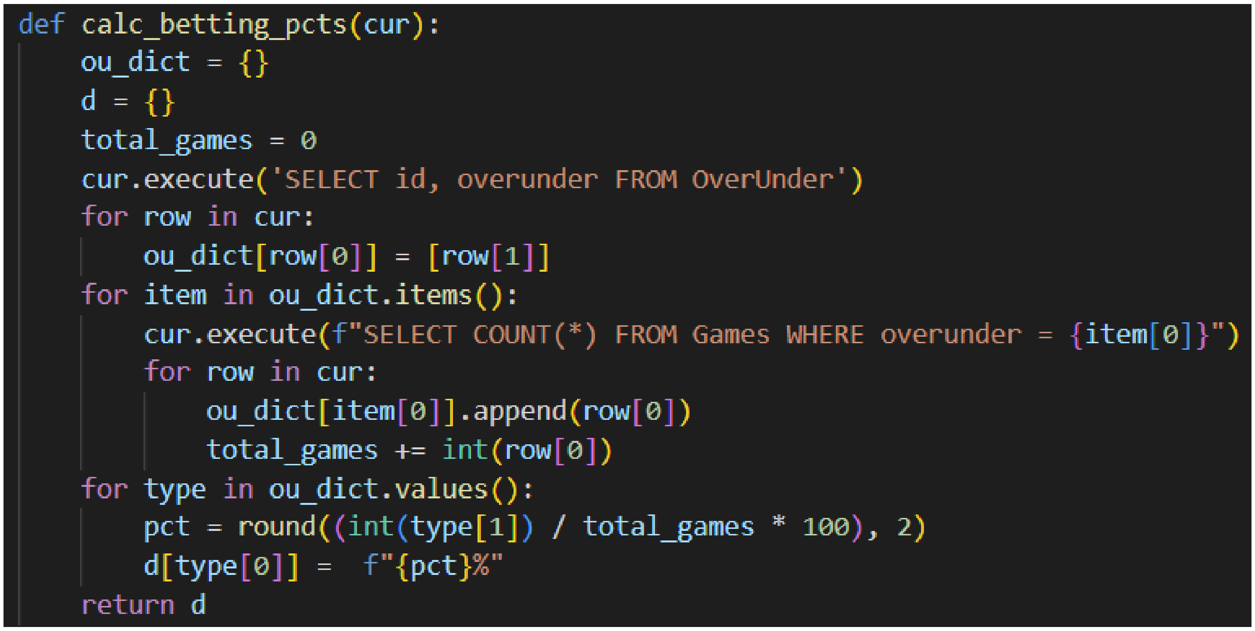
Calculations and Visualizations

We were able to make two significant calculations from our data. The first was the average total points, total yards, and total turnovers for any given game during the 2022 NFL season.



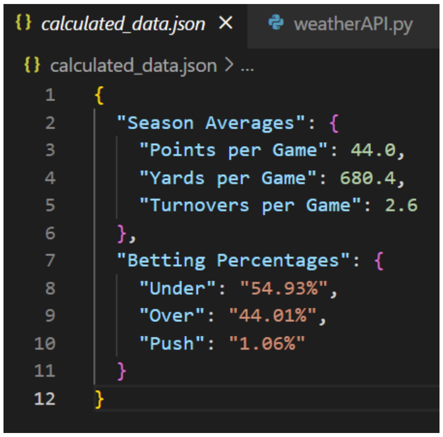
```python
def calc_season_avgs(cur):
    d = {}
    cur.execute('SELECT AVG(total_pts_scored), AVG(total_yrds_gained), AVG(total_turnovers) FROM Games')
    for row in cur:
        avg_pts = round(row[0], 1)
        d['Points per Game'] = avg_pts
        avg_yrds = round(row[1], 1)
        d['Yards per Game'] = avg_yrds
        avg_turnovers = round(row[2], 1)
        d['Turnovers per Game'] = avg_turnovers
    return d
```

Additionally, we calculated the percentage of NFL games that resulted in the Over or Under hitting as well as neither (Push) according to the lines created by Caesars sportsbook.



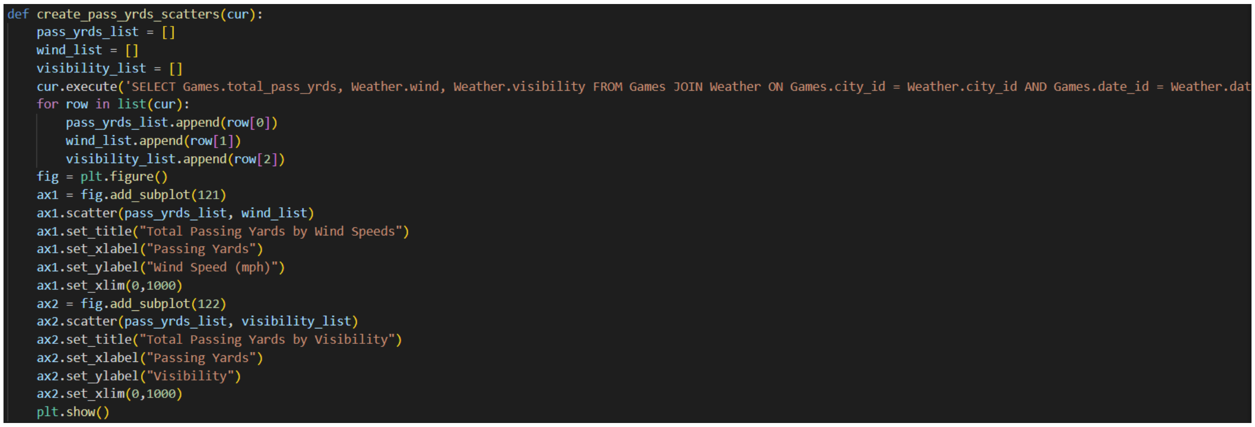
```python
def calc_betting_pcts(cur):
    ou_dict = {}
    d = {}
    total_games = 0
    cur.execute('SELECT id, overunder FROM OverUnder')
    for row in cur:
        ou_dict[row[0]] = [row[1]]
    for item in ou_dict.items():
        cur.execute(f"SELECT COUNT(*) FROM Games WHERE overunder = {item[0]}")
        for row in cur:
            ou_dict[item[0]].append(row[0])
            total_games += int(row[0])
    for type in ou_dict.values():
        pct = round((int(type[1]) / total_games * 100), 2)
        d[type[0]] =  f"{pct}%"
    return d
```

We then combined our calculations into a singular dictionary and wrote the data to a separate file called ‘calculated_data.json’

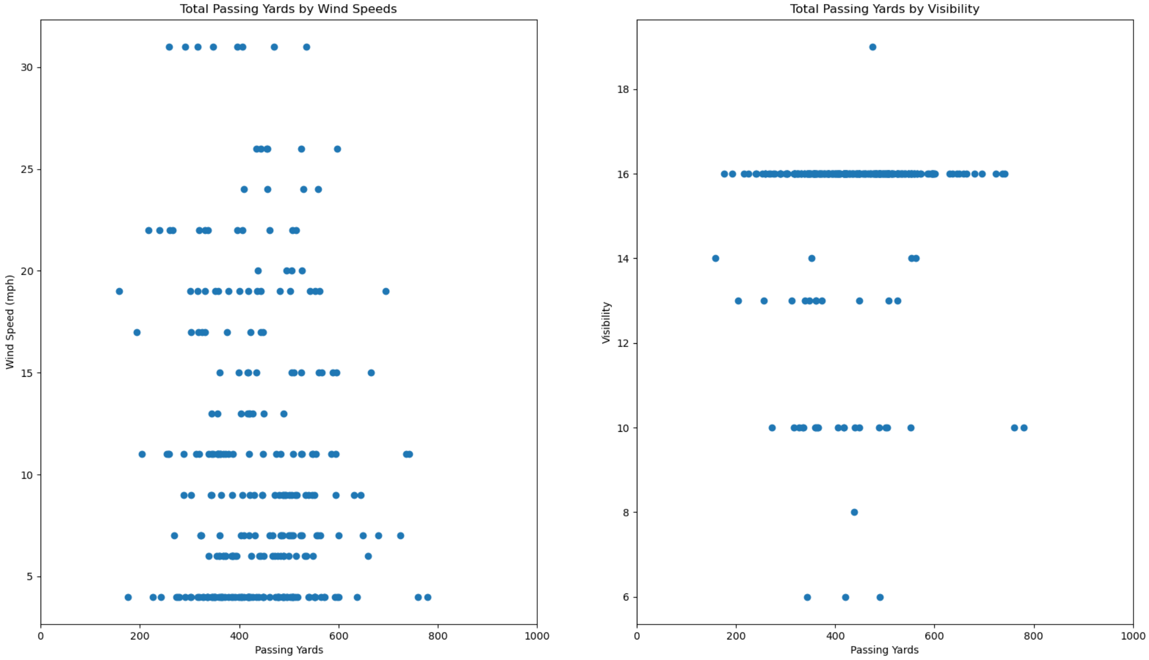


```json
{
  "Season Averages": {
    "Points per Game": 44.0,
    "Yards per Game": 680.4,
    "Turnovers per Game": 2.6
  },
  "Betting Percentages": {
    "Under": "54.93%",
    "Over": "44.01%",
    "Push": "1.06%"
  }
}
```

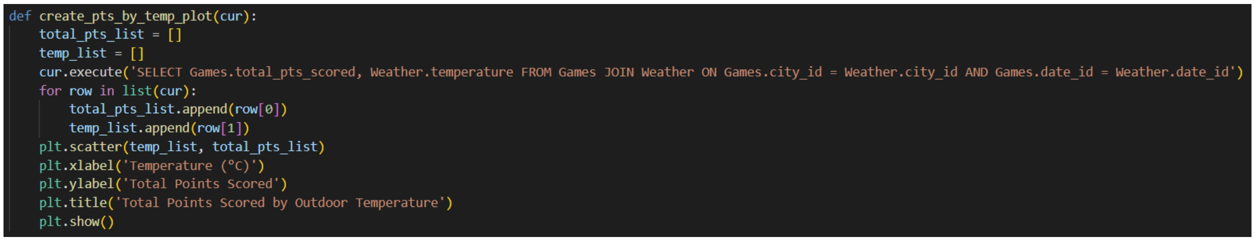
As for our graphs, we created five interesting figures that show many important NFL trends relating to weather. First, we created two side-by-side scatter plots showing how total passing yards are affected by wind speed and visibility.



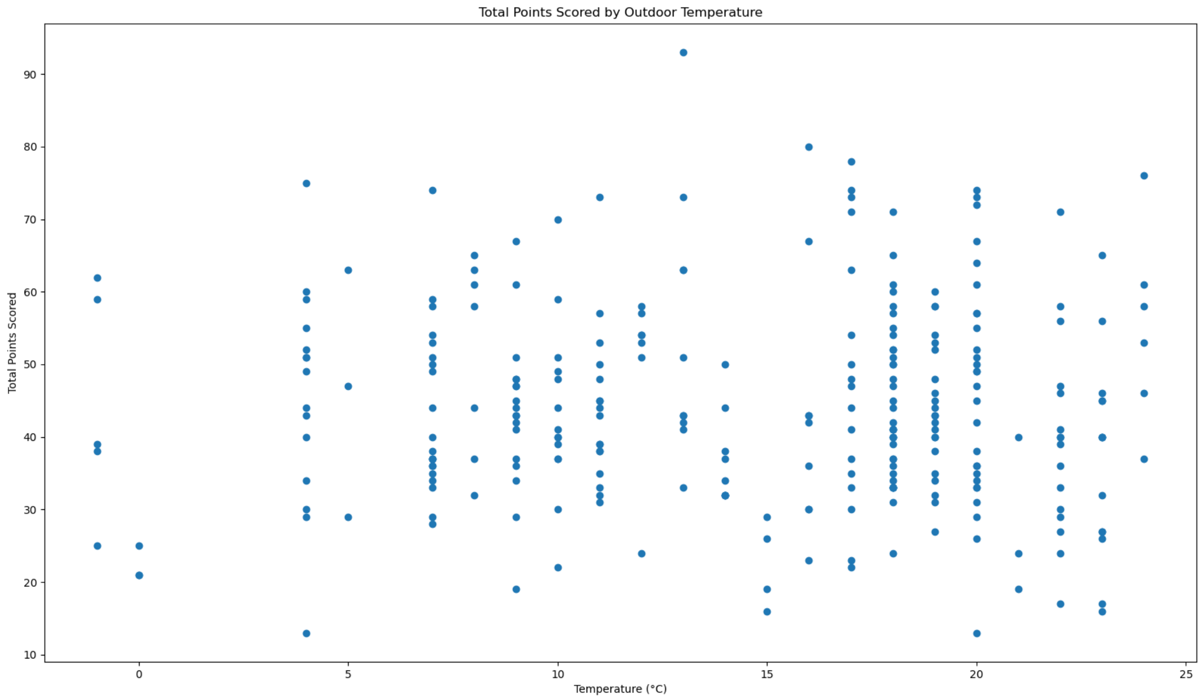
```
def create_pass_yrds_scatters(cur):
    pass_yrds_list = []
    wind_list = []
    visibility_list = []
    cur.execute('SELECT Games.total_pass_yrds, Weather.wind, Weather.visibility FROM Games JOIN Weather ON Games.city_id = Weather.city_id AND Games.date_id = Weather.dat
    for row in list(cur):
        pass_yrds_list.append(row[0])
        wind_list.append(row[1])
        visibility_list.append(row[2])
    fig = plt.figure()
    ax1 = fig.add_subplot(121)
    ax1.scatter(pass_yrds_list, wind_list)
    ax1.set_title("Total Passing Yards by Wind Speeds")
    ax1.set_xlabel("Passing Yards")
    ax1.set_ylabel("Wind Speed (mph)")
    ax1.set_xlim(0,1000)
    ax2 = fig.add_subplot(122)
    ax2.scatter(pass_yrds_list, visibility_list)
    ax2.set_title("Total Passing Yards by Visibility")
    ax2.set_xlabel("Passing Yards")
    ax2.set_ylabel("Visibility")
    ax2.set_xlim(0,1000)
    plt.show()
```



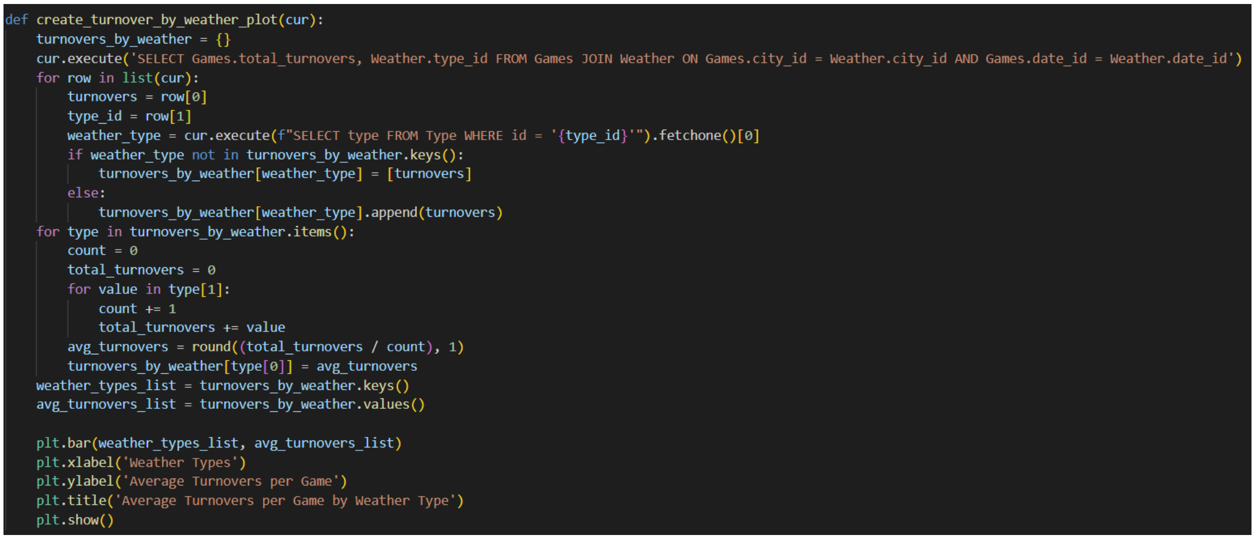
Additionally, we displayed a scatter plot comparing total points scored and outdoor temperature to see if colder temps naturally slow down the game and prevent points from being scored.



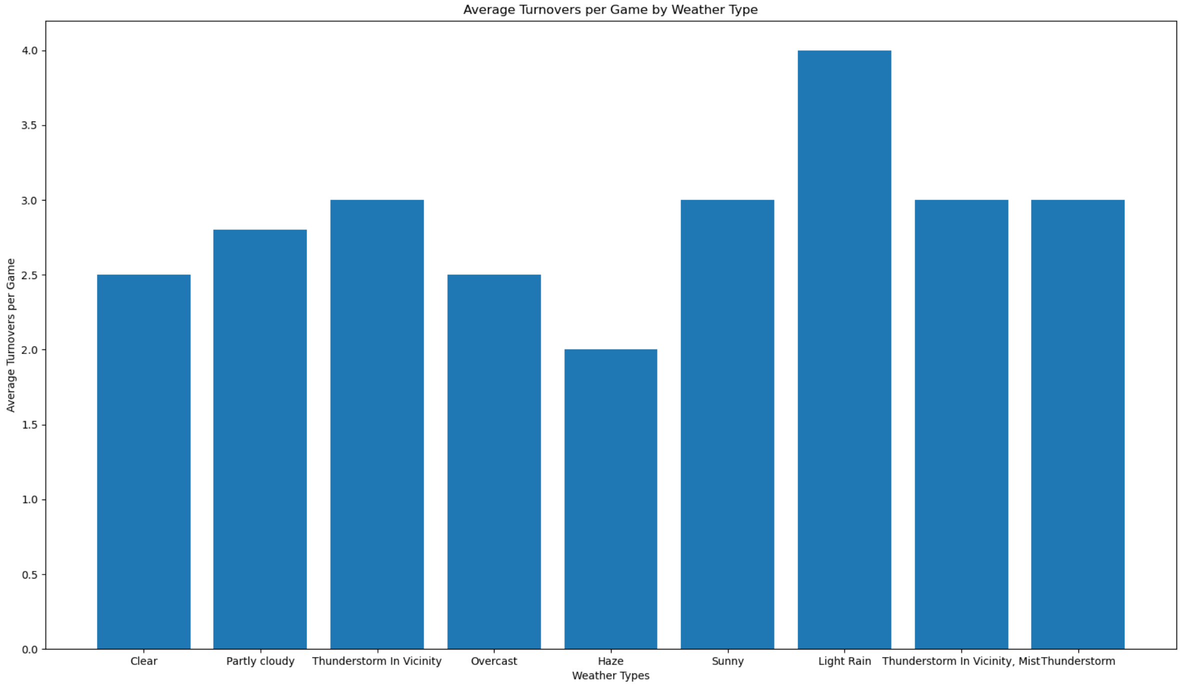
```
def create_pts_by_temp_plot(cur):
    total_pts_list = []
    temp_list = []
    cur.execute('SELECT Games.total_pts_scored, Weather.temperature FROM Games JOIN Weather ON Games.city_id = Weather.city_id AND Games.date_id = Weather.date_id')
    for row in list(cur):
        total_pts_list.append(row[0])
        temp_list.append(row[1])
    plt.scatter(temp_list, total_pts_list)
    plt.xlabel('Temperature (°C)')
    plt.ylabel('Total Points Scored')
    plt.title('Total Points Scored by Outdoor Temperature')
    plt.show()
```



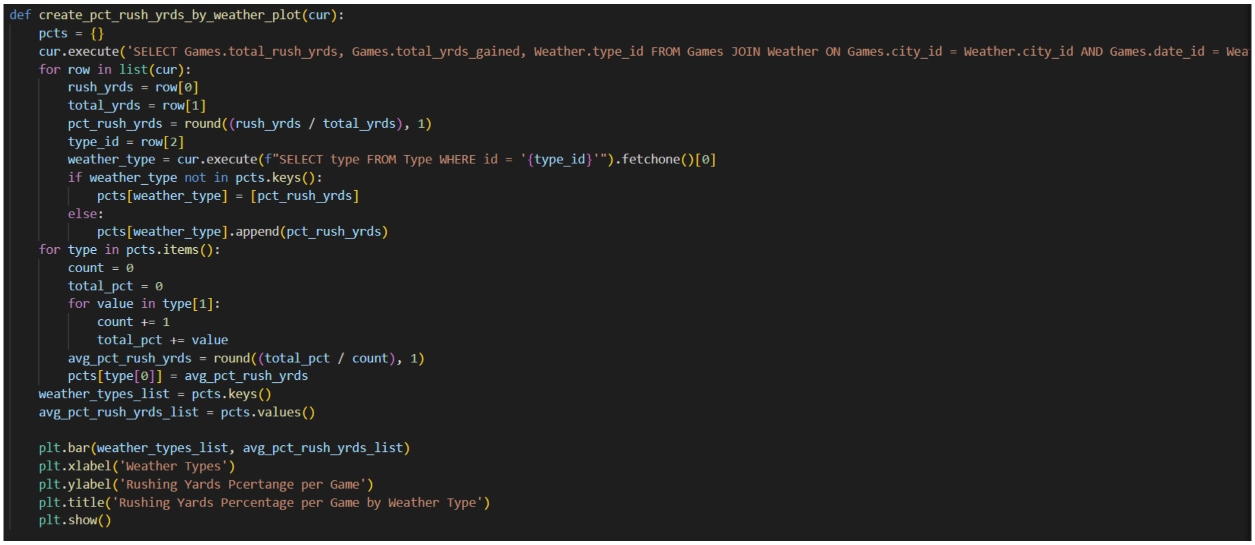
We created a similar bar chart that compares the average turnovers per game by weather type to see if certain conditions (rainy, windy) cause more turnovers than a clear day.



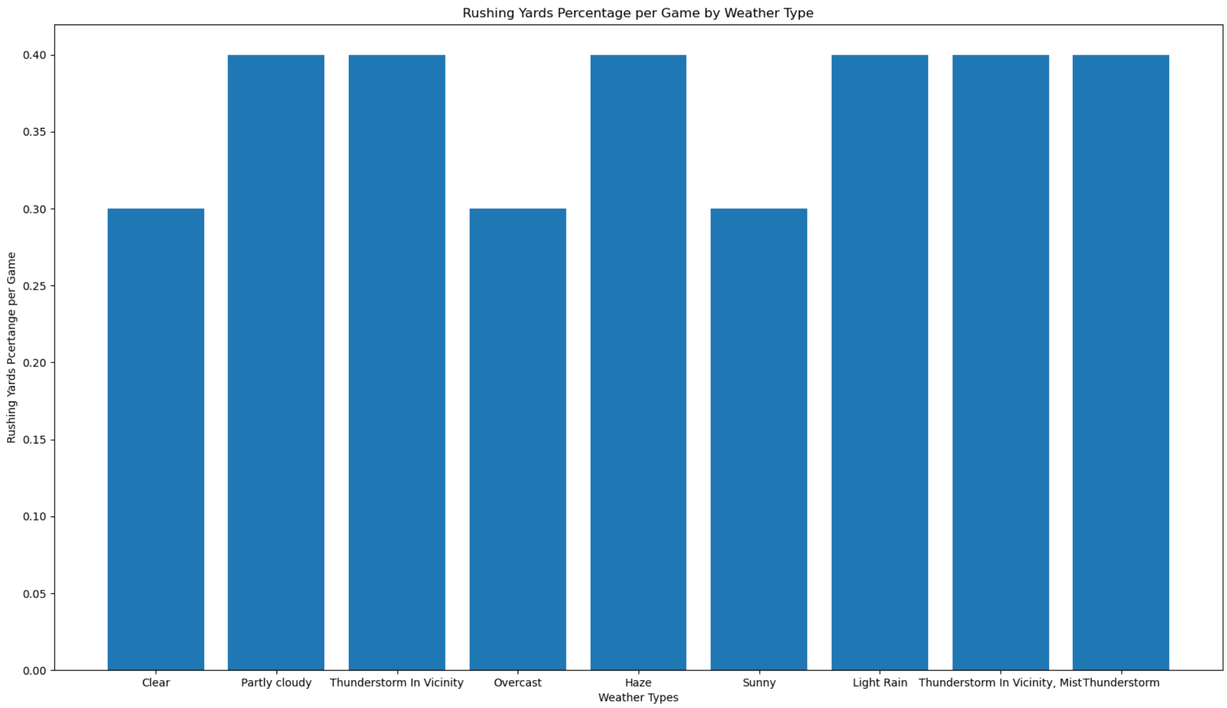
```python
def create_turnover_by_weather_plot(cur):
    turnovers_by_weather = {}
    cur.execute('SELECT Games.total_turnovers, Weather.type_id FROM Games JOIN Weather ON Games.city_id = Weather.city_id AND Games.date_id = Weather.date_id')
    for row in list(cur):
        turnovers = row[0]
        type_id = row[1]
        weather_type = cur.execute(f"SELECT type FROM Type WHERE id = '{type_id}'").fetchone()[0]
        if weather_type not in turnovers_by_weather.keys():
            turnovers_by_weather[weather_type] = [turnovers]
        else:
            turnovers_by_weather[weather_type].append(turnovers)
    for type in turnovers_by_weather.items():
        count = 0
        total_turnovers = 0
        for value in type[1]:
            count += 1
            total_turnovers += value
        avg_turnovers = round((total_turnovers / count), 1)
        turnovers_by_weather[type[0]] = avg_turnovers
    weather_types_list = turnovers_by_weather.keys()
    avg_turnovers_list = turnovers_by_weather.values()

    plt.bar(weather_types_list, avg_turnovers_list)
    plt.xlabel('Weather Types')
    plt.ylabel('Average Turnovers per Game')
    plt.title('Average Turnovers per Game by Weather Type')
    plt.show()
```



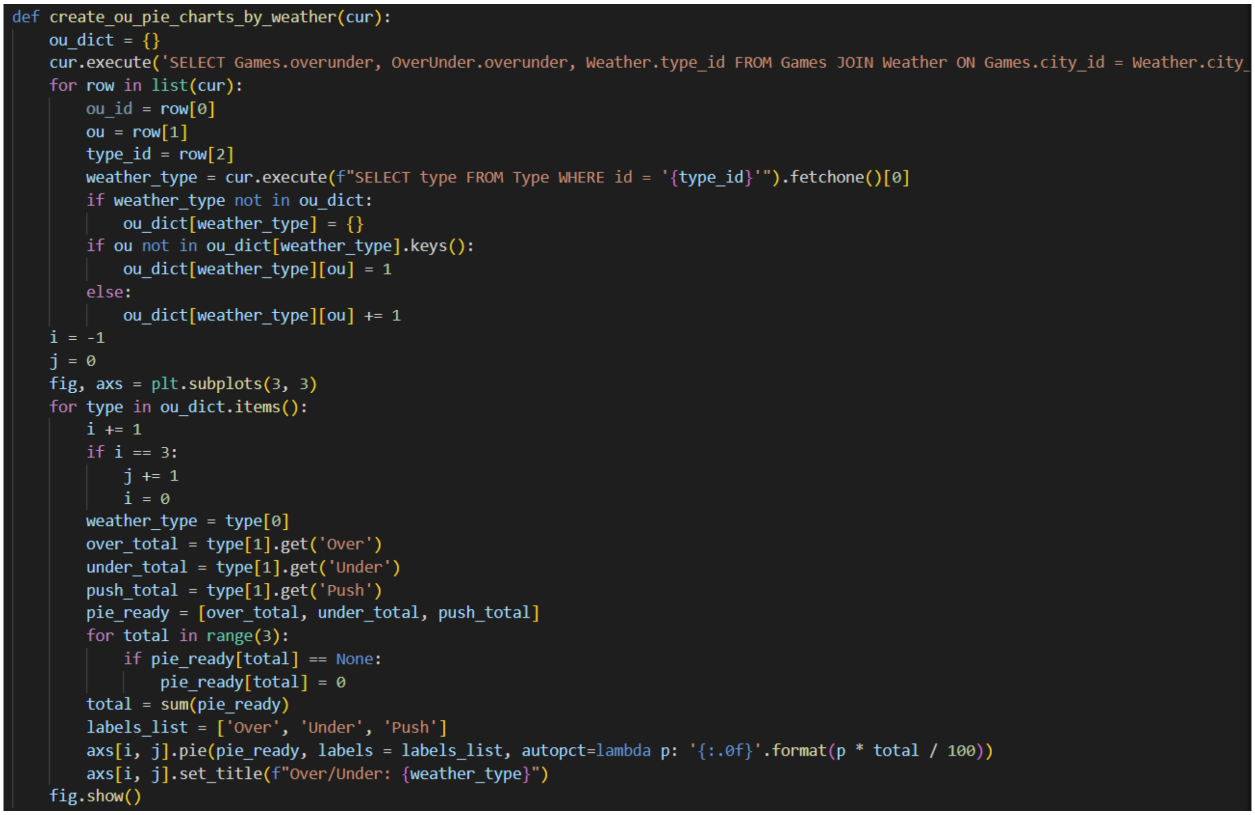
We created a bar chart that displays how the percentage of total yards that are achieved through rushing the football compares to each individual weather condition.



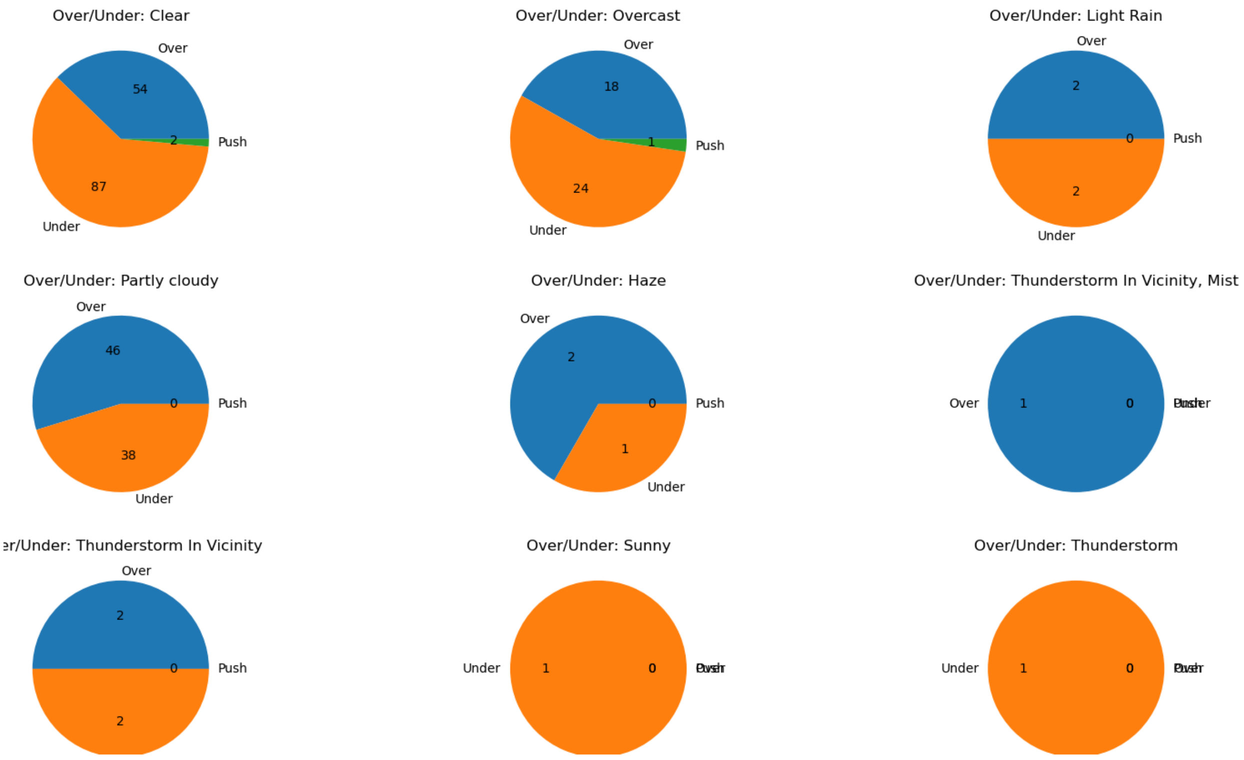
```
def create_pct_rush_yrds_by_weather_plot(cur):
    pcts = {}
    cur.execute('SELECT Games.total_rush_yrds, Games.total_yrds_gained, Weather.type_id FROM Games JOIN Weather ON Games.city_id = Weather.city_id AND Games.date_id = Wea
    for row in list(cur):
        rush_yrds = row[0]
        total_yrds = row[1]
        pct_rush_yrds = round((rush_yrds / total_yrds), 1)
        type_id = row[2]
        weather_type = cur.execute(f"SELECT type FROM Type WHERE id = '{type_id}'").fetchone()[0]
        if weather_type not in pcts.keys():
            pcts[weather_type] = [pct_rush_yrds]
        else:
            pcts[weather_type].append(pct_rush_yrds)
    for type in pcts.items():
        count = 0
        total_pct = 0
        for value in type[1]:
            count += 1
            total_pct += value
        avg_pct_rush_yrds = round((total_pct / count), 1)
        pcts[type[0]] = avg_pct_rush_yrds
    weather_types_list = pcts.keys()
    avg_pct_rush_yrds_list = pcts.values()

    plt.bar(weather_types_list, avg_pct_rush_yrds_list)
    plt.xlabel('Weather Types')
    plt.ylabel('Rushing Yards Pcertange per Game')
    plt.title('Rushing Yards Percentage per Game by Weather Type')
    plt.show()
```



Finally, we created a series of pie charts that show the number of games ending in Over, Under, or Push for each weather condition.



```
def create_ou_pie_charts_by_weather(cur):
    ou_dict = {}
    cur.execute('SELECT Games.overunder, OverUnder.overunder, Weather.type_id FROM Games JOIN Weather ON Games.city_id = Weather.city_
    for row in list(cur):
        ou_id = row[0]
        ou = row[1]
        type_id = row[2]
        weather_type = cur.execute(f"SELECT type FROM Type WHERE id = '{type_id}'").fetchone()[0]
        if weather_type not in ou_dict:
            ou_dict[weather_type] = {}
        if ou not in ou_dict[weather_type].keys():
            ou_dict[weather_type][ou] = 1
        else:
            ou_dict[weather_type][ou] += 1
    i = -1
    j = 0
    fig, axs = plt.subplots(3, 3)
    for type in ou_dict.items():
        i += 1
        if i == 3:
            j += 1
            i = 0
        weather_type = type[0]
        over_total = type[1].get('Over')
        under_total = type[1].get('Under')
        push_total = type[1].get('Push')
        pie_ready = [over_total, under_total, push_total]
        for total in range(3):
            if pie_ready[total] == None:
                pie_ready[total] = 0
        total = sum(pie_ready)
        labels_list = ['Over', 'Under', 'Push']
        axs[i, j].pie(pie_ready, labels = labels_list, autopct=lambda p: '{:.0f}'.format(p * total / 100))
        axs[i, j].set_title(f"Over/Under: {weather_type}")
    fig.show()
```



Instructions to Run Code

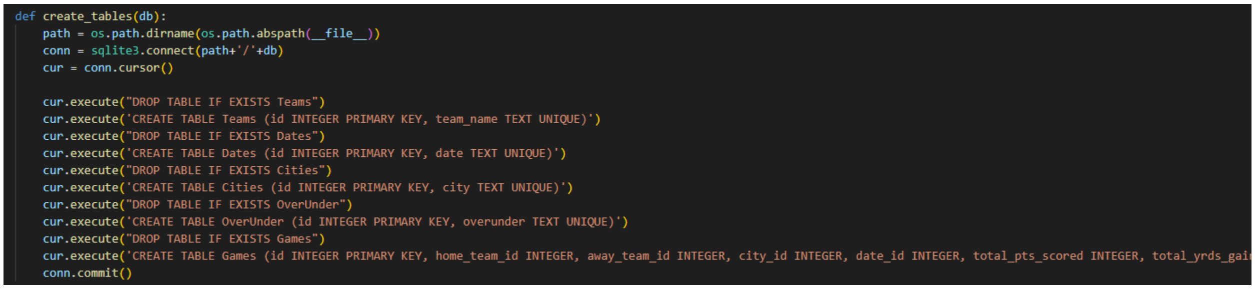
Step 1: In the main.py file, uncomment the call to emptyDatabase and run the file to set up the database with empty tables.
Step 2: Either uncomment the for loop containing insertIntoDatabase or just insertIntoDatabase itself to add games to the database. This will access the ESPN.py file and the weatherAPI.py and add 25 games to the database with each iteration of the loop (it will take a while to run the whole thing). Alternatively, you can add 25 games only by calling insertIntoDatabase outside the for loop and keeping the loop commented out.
Step 3: Comment out all calls to emptyDatabase and insertIntoDatabase. Then, uncomment the series of lines that make calls to calc_season_avgs and calc_betting_pcts (lines 205-209). This will do both of our calculations and write the outputs into a json file.
Step 4: Re-comment the five lines relating to calculation calls and uncomment the create_pass_yrds_scatters call. This will create our first graph
Step 5: Re-comment the previous calls and uncomment the call for clear_plot and create_pts_by_temp_plot to get our second graph
Step 6: Re-comment the previous calls and uncomment the call for clear_plot and create_turnover_by_weather_plot to get our third graph
Step 7: Re-comment the previous calls and uncomment the call for clear_plot and create_pct_rush_yrds_by_weather_plot to get our fourth graph
Step 8: Re-comment the previous calls and uncomment the call for clear_plot and create_ou_pie_charts_by_weather to get our fifth and final graph

Code Documentation

ESPN.py File:

create_tables(db) function:

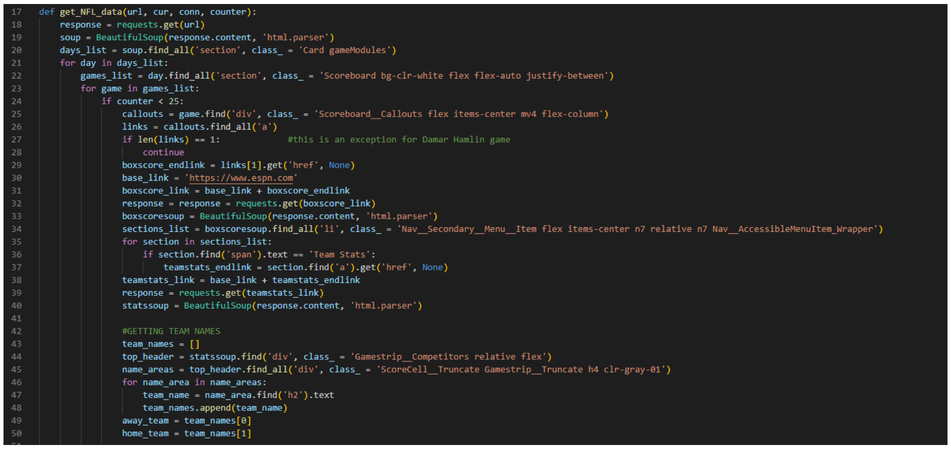
- Inputs the name of the database
- Creates five tables that store all of the ESPN data
- Does not output anything



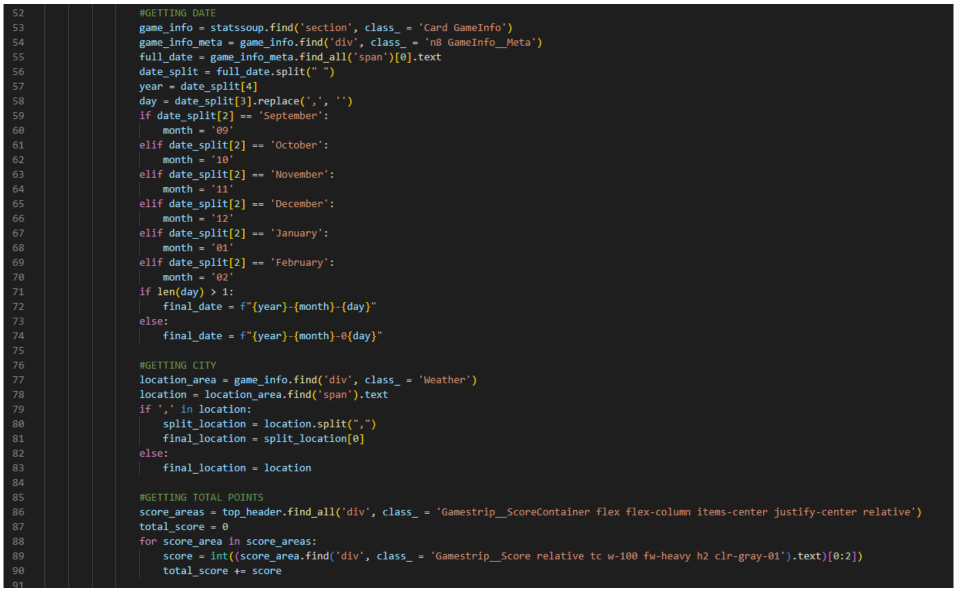
```python
def create_tables(db):
    path = os.path.dirname(os.path.abspath(__file__))
    conn = sqlite3.connect(path+'/'+db)
    cur = conn.cursor()

    cur.execute("DROP TABLE IF EXISTS Teams")
    cur.execute('CREATE TABLE Teams (id INTEGER PRIMARY KEY, team_name TEXT UNIQUE)')
    cur.execute("DROP TABLE IF EXISTS Dates")
    cur.execute('CREATE TABLE Dates (id INTEGER PRIMARY KEY, date TEXT UNIQUE)')
    cur.execute("DROP TABLE IF EXISTS Cities")
    cur.execute('CREATE TABLE Cities (id INTEGER PRIMARY KEY, city TEXT UNIQUE)')
    cur.execute("DROP TABLE IF EXISTS OverUnder")
    cur.execute('CREATE TABLE OverUnder (id INTEGER PRIMARY KEY, overunder TEXT UNIQUE)')
    cur.execute("DROP TABLE IF EXISTS Games")
    cur.execute('CREATE TABLE Games (id INTEGER PRIMARY KEY, home_team_id INTEGER, away_team_id INTEGER, city_id INTEGER, date_id INTEGER, total_pts_scored INTEGER, total_yrds_gai
    conn.commit()
```

get_NFL_data(url, cur, conn, counter) function:

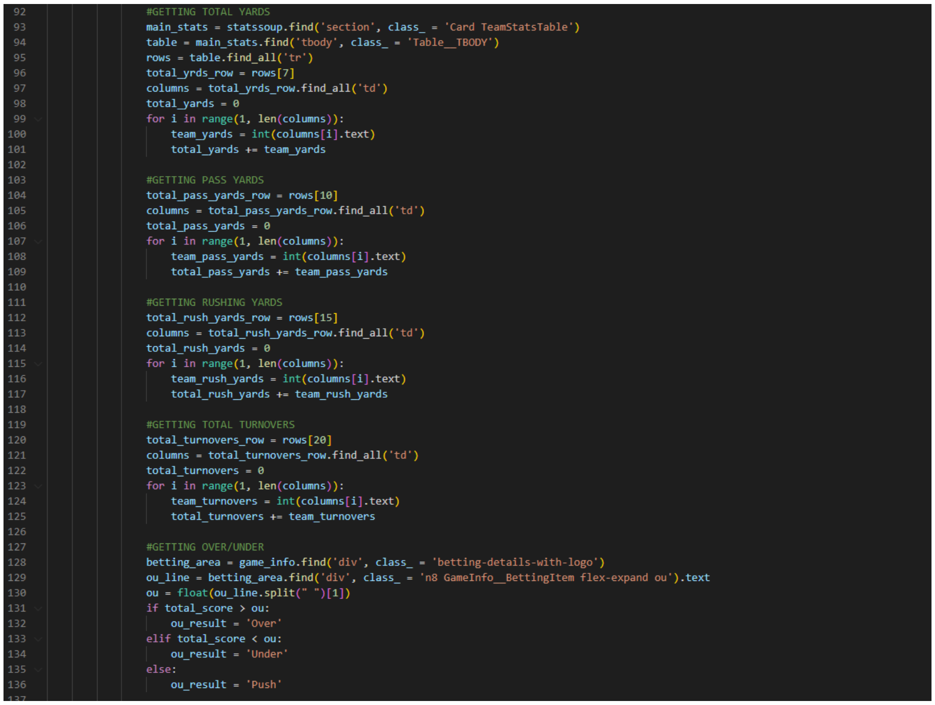
- Inputs the url to the ESPN weekly NFL page, the cursor and connection to access the database, and the current counter from how many games have been added to the database so far during this run
- For each game in the week, it will find and scrape all of the necessary information (team names, date, city, total points, total yards, pass yards, rush yards, turnovers, over/under) and add them to the databases as long as there have not been 25 games found. The names, dates, cities, and over/under results each go to their respective tables and those ids go into the Games table with the rest of the data values
- Returns the counter of games found so far



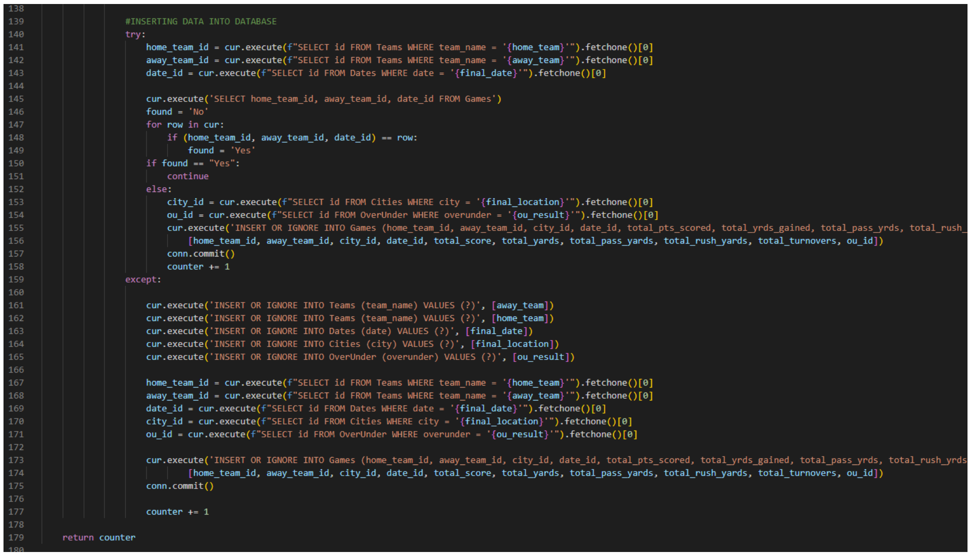
```python
def get_NFL_data(url, cur, conn, counter):
    response = requests.get(url)
    soup = BeautifulSoup(response.content, 'html.parser')
    days_list = soup.find_all('section', class_ = 'Card gameModules')
    for day in days_list:
        games_list = day.find_all('section', class_ = 'Scoreboard bg-clr-white flex flex-auto justify-between')
        for game in games_list:
            if counter < 25:
                callouts = game.find('div', class_ = 'Scoreboard__Callouts flex items-center mv4 flex-column')
                links = callouts.find_all('a')
                if len(links) == 1:             #this is an exception for Damar Hamlin game
                    continue
                boxscore_endlink = links[1].get('href', None)
                base_link = 'https://www.espn.com'
                boxscore_link = base_link + boxscore_endlink
                response = response = requests.get(boxscore_link)
                boxscoresoup = BeautifulSoup(response.content, 'html.parser')
                sections_list = boxscoresoup.find_all('li', class_ = 'Nav__Secondary__Menu__Item flex items-center n7 relative n7 Nav__AccessibleMenuItem_Wrapper')
                for section in sections_list:
                    if section.find('span').text == 'Team Stats':
                        teamstats_endlink = section.find('a').get('href', None)
                teamstats_link = base_link + teamstats_endlink
                response = requests.get(teamstats_link)
                statssoup = BeautifulSoup(response.content, 'html.parser')

                #GETTING TEAM NAMES
                team_names = []
                top_header = statssoup.find('div', class_ = 'Gamestrip__Competitors relative flex')
                name_areas = top_header.find_all('div', class_ = 'ScoreCell__Truncate Gamestrip__Truncate h4 clr-gray-01')
                for name_area in name_areas:
                    team_name = name_area.find('h2').text
                    team_names.append(team_name)
                away_team = team_names[0]
                home_team = team_names[1]
```



```python
                #GETTING DATE
                game_info = statssoup.find('section', class_ = 'Card GameInfo')
                game_info_meta = game_info.find('div', class_ = 'n8 GameInfo__Meta')
                full_date = game_info_meta.find_all('span')[0].text
                date_split = full_date.split(" ")
                year = date_split[4]
                day = date_split[3].replace(',', '')
                if date_split[2] == 'September':
                    month = '09'
                elif date_split[2] == 'October':
                    month = '10'
                elif date_split[2] == 'November':
                    month = '11'
                elif date_split[2] == 'December':
                    month = '12'
                elif date_split[2] == 'January':
                    month = '01'
                elif date_split[2] == 'February':
                    month = '02'
                if len(day) > 1:
                    final_date = f"{year}-{month}-{day}"
                else:
                    final_date = f"{year}-{month}-0{day}"

                #GETTING CITY
                location_area = game_info.find('div', class_ = 'Weather')
                location = location_area.find('span').text
                if ',' in location:
                    split_location = location.split(",")
                    final_location = split_location[0]
                else:
                    final_location = location

                #GETTING TOTAL POINTS
                score_areas = top_header.find_all('div', class_ = 'Gamestrip__ScoreContainer flex flex-column items-center justify-center relative')
                total_score = 0
                for score_area in score_areas:
                    score = int((score_area.find('div', class_ = 'Gamestrip__Score relative tc w-100 fw-heavy h2 clr-gray-01').text)[0:2])
                    total_score += score

```



```python
                #GETTING TOTAL YARDS
                main_stats = statssoup.find('section', class_ = 'Card TeamStatsTable')
                table = main_stats.find('tbody', class_ = 'Table__TBODY')
                rows = table.find_all('tr')
                total_yrds_row = rows[7]
                columns = total_yrds_row.find_all('td')
                total_yards = 0
                for i in range(1, len(columns)):
                    team_yards = int(columns[i].text)
                    total_yards += team_yards

                #GETTING PASS YARDS
                total_pass_yards_row = rows[10]
                columns = total_pass_yards_row.find_all('td')
                total_pass_yards = 0
                for i in range(1, len(columns)):
                    team_pass_yards = int(columns[i].text)
                    total_pass_yards += team_pass_yards

                #GETTING RUSHING YARDS
                total_rush_yards_row = rows[15]
                columns = total_rush_yards_row.find_all('td')
                total_rush_yards = 0
                for i in range(1, len(columns)):
                    team_rush_yards = int(columns[i].text)
                    total_rush_yards += team_rush_yards

                #GETTING TOTAL TURNOVERS
                total_turnovers_row = rows[20]
                columns = total_turnovers_row.find_all('td')
                total_turnovers = 0
                for i in range(1, len(columns)):
                    team_turnovers = int(columns[i].text)
                    total_turnovers += team_turnovers

                #GETTING OVER/UNDER
                betting_area = game_info.find('div', class_ = 'betting-details-with-logo')
                ou_line = betting_area.find('div', class_ = 'n8 GameInfo__BettingItem flex-expand ou').text
                ou = float(ou_line.split(" ")[1])
                if total_score > ou:
                    ou_result = 'Over'
                elif total_score < ou:
                    ou_result = 'Under'
                else:
                    ou_result = 'Push'
```

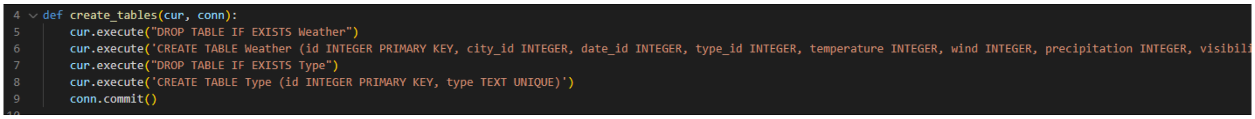


```
                #INSERTING DATA INTO DATABASE
                try:
                    home_team_id = cur.execute(f"SELECT id FROM Teams WHERE team_name = '{home_team}'").fetchone()[0]
                    away_team_id = cur.execute(f"SELECT id FROM Teams WHERE team_name = '{away_team}'").fetchone()[0]
                    date_id = cur.execute(f"SELECT id FROM Dates WHERE date = '{final_date}'").fetchone()[0]

                    cur.execute('SELECT home_team_id, away_team_id, date_id FROM Games')
                    found = 'No'
                    for row in cur:
                        if (home_team_id, away_team_id, date_id) == row:
                            found = 'Yes'
                    if found == "Yes":
                        continue
                    else:
                        city_id = cur.execute(f"SELECT id FROM Cities WHERE city = '{final_location}'").fetchone()[0]
                        ou_id = cur.execute(f"SELECT id FROM OverUnder WHERE overunder = '{ou_result}'").fetchone()[0]
                        cur.execute('INSERT OR IGNORE INTO Games (home_team_id, away_team_id, city_id, date_id, total_pts_scored, total_yrds_gained, total_pass_yrds, total_rush_
                            [home_team_id, away_team_id, city_id, date_id, total_score, total_yards, total_pass_yards, total_rush_yards, total_turnovers, ou_id])
                        conn.commit()
                        counter += 1
                except:

                    cur.execute('INSERT OR IGNORE INTO Teams (team_name) VALUES (?)', [away_team])
                    cur.execute('INSERT OR IGNORE INTO Teams (team_name) VALUES (?)', [home_team])
                    cur.execute('INSERT OR IGNORE INTO Dates (date) VALUES (?)', [final_date])
                    cur.execute('INSERT OR IGNORE INTO Cities (city) VALUES (?)', [final_location])
                    cur.execute('INSERT OR IGNORE INTO OverUnder (overunder) VALUES (?)', [ou_result])

                    home_team_id = cur.execute(f"SELECT id FROM Teams WHERE team_name = '{home_team}'").fetchone()[0]
                    away_team_id = cur.execute(f"SELECT id FROM Teams WHERE team_name = '{away_team}'").fetchone()[0]
                    date_id = cur.execute(f"SELECT id FROM Dates WHERE date = '{final_date}'").fetchone()[0]
                    city_id = cur.execute(f"SELECT id FROM Cities WHERE city = '{final_location}'").fetchone()[0]
                    ou_id = cur.execute(f"SELECT id FROM OverUnder WHERE overunder = '{ou_result}'").fetchone()[0]

                    cur.execute('INSERT OR IGNORE INTO Games (home_team_id, away_team_id, city_id, date_id, total_pts_scored, total_yrds_gained, total_pass_yrds, total_rush_yrds
                            [home_team_id, away_team_id, city_id, date_id, total_score, total_yards, total_pass_yards, total_rush_yards, total_turnovers, ou_id])
                    conn.commit()

                    counter += 1

    return counter
```

weatherAPI.py File

create_tables(cur, conn) function:

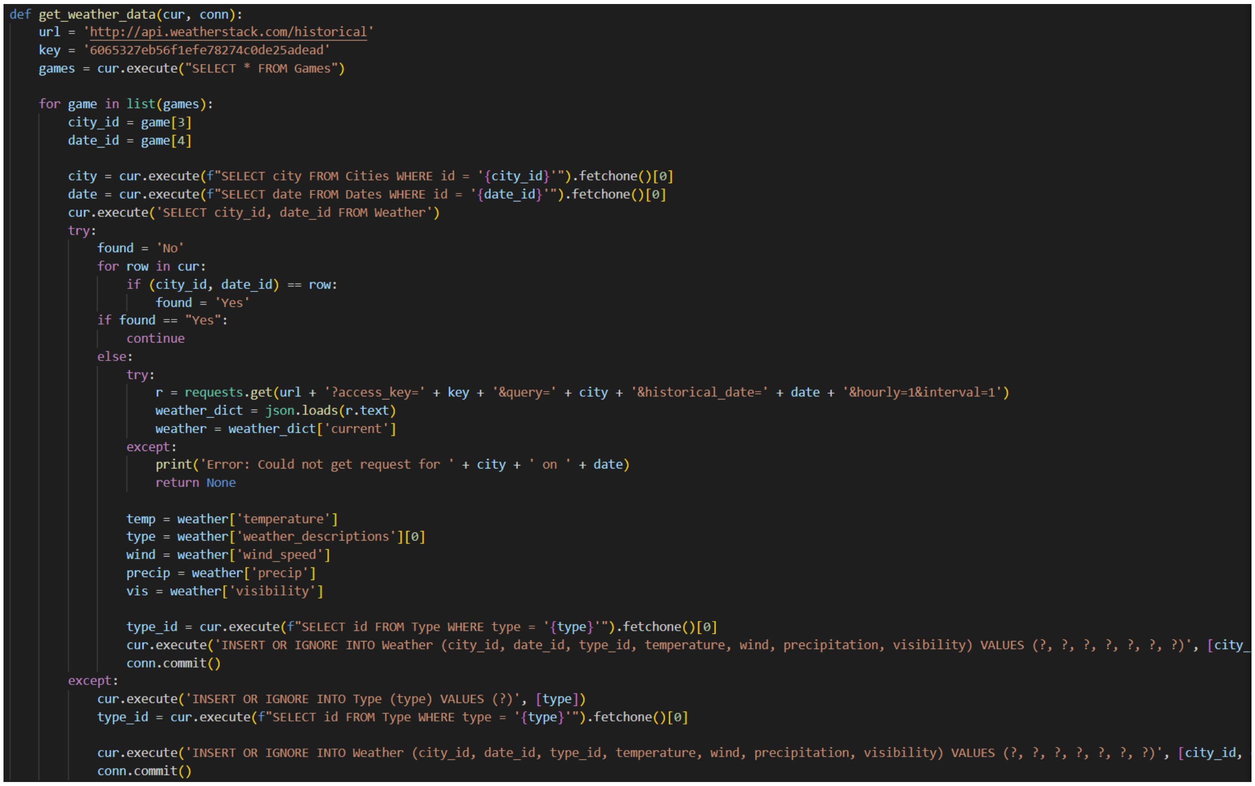
- Inputs the cursor and connection to the database
- Creates 2 tables that store all of the weather data
- Returns nothing



```
def create_tables(cur, conn):
    cur.execute("DROP TABLE IF EXISTS Weather")
    cur.execute('CREATE TABLE Weather (id INTEGER PRIMARY KEY, city_id INTEGER, date_id INTEGER, type_id INTEGER, temperature INTEGER, wind INTEGER, precipitation INTEGER, visibili
    cur.execute("DROP TABLE IF EXISTS Type")
    cur.execute('CREATE TABLE Type (id INTEGER PRIMARY KEY, type TEXT UNIQUE)')
    conn.commit()
```

get_weather_data(cur, conn) function:

- Inputs the cursor and connection to the database
- Gets all of the current games information from the Games table and for each game, it gets the city and date information. Using that information, it requests weather data including temperature, type, wind speeds, precipitation, and visibility from the weather API. This information is then stored into the Weather table in the database. Type is stored in the Type table with type_id in the Weather table to avoid duplicate string data
- Returns nothing

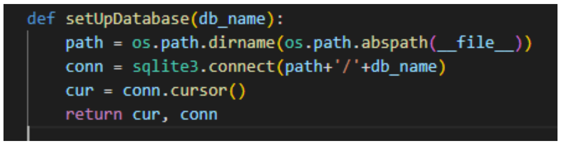


```python
def get_weather_data(cur, conn):
    url = 'http://api.weatherstack.com/historical'
    key = '6065327eb56f1efe78274c0de25adead'
    games = cur.execute("SELECT * FROM Games")

    for game in list(games):
        city_id = game[3]
        date_id = game[4]

        city = cur.execute(f"SELECT city FROM Cities WHERE id = '{city_id}'").fetchone()[0]
        date = cur.execute(f"SELECT date FROM Dates WHERE id = '{date_id}'").fetchone()[0]
        cur.execute('SELECT city_id, date_id FROM Weather')
        try:
            found = 'No'
            for row in cur:
                if (city_id, date_id) == row:
                    found = 'Yes'
            if found == "Yes":
                continue
            else:
                try:
                    r = requests.get(url + '?access_key=' + key + '&query=' + city + '&historical_date=' + date + '&hourly=1&interval=1')
                    weather_dict = json.loads(r.text)
                    weather = weather_dict['current']
                except:
                    print('Error: Could not get request for ' + city + ' on ' + date)
                    return None

                temp = weather['temperature']
                type = weather['weather_descriptions'][0]
                wind = weather['wind_speed']
                precip = weather['precip']
                vis = weather['visibility']

                type_id = cur.execute(f"SELECT id FROM Type WHERE type = '{type}'").fetchone()[0]
                cur.execute('INSERT OR IGNORE INTO Weather (city_id, date_id, type_id, temperature, wind, precipitation, visibility) VALUES (?, ?, ?, ?, ?, ?, ?)', [city_
                conn.commit()
        except:
            cur.execute('INSERT OR IGNORE INTO Type (type) VALUES (?)', [type])
            type_id = cur.execute(f"SELECT id FROM Type WHERE type = '{type}'").fetchone()[0]

            cur.execute('INSERT OR IGNORE INTO Weather (city_id, date_id, type_id, temperature, wind, precipitation, visibility) VALUES (?, ?, ?, ?, ?, ?, ?)', [city_id,
            conn.commit()
```

Main.py File

setUpDatabase(db_name) function:

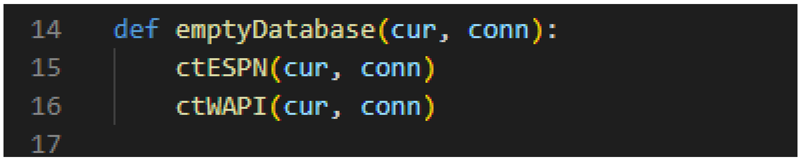
- Inputs the name of the database
- Creates a connection to the database
- Returns the cursor and connection to the database



```python
def setUpDatabase(db_name):
    path = os.path.dirname(os.path.abspath(__file__))
    conn = sqlite3.connect(path+'/'+db_name)
    cur = conn.cursor()
    return cur, conn
```

emptyDatabase(cur, conn) function:

- Inputs the cursor and connection to the database
- Creates and/or empties all of the tables within our database
- Returns nothing



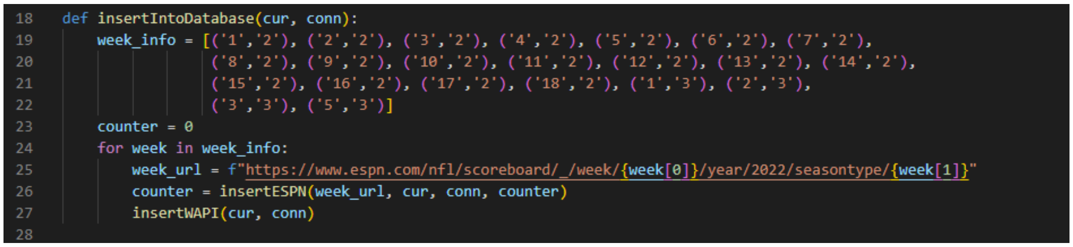
```python
def emptyDatabase(cur, conn):
    ctESPN(cur, conn)
    ctWAPI(cur, conn)

```

insertIntoDatabase(cur, conn) function:

- Inputs the cursor and connection to the database
- Goes through each weekly ESPN page and finds 25 uniquely new games by calling the get_NFL_data function in the ESPN.py file. The data for these games will be inputted into the Games table and new information will be added to the other football related databases as well if needed. This function will also call into get_weather_data function in

weatherAPI.py and get the necessary information given the date and city that was found on ESPN. This additional information is added to the Weather table and if applicable, new weather conditions will be inserted into the Type table. One 25 new games are found, this function will stop

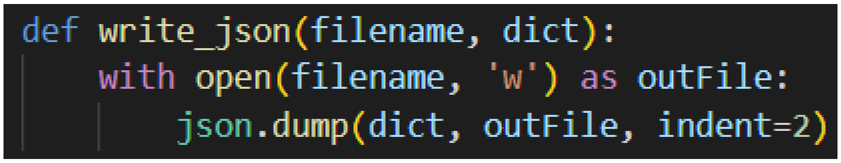
- Returns nothing



```python
def insertIntoDatabase(cur, conn):
    week_info = [('1','2'), ('2','2'), ('3','2'), ('4','2'), ('5','2'), ('6','2'), ('7','2'),
                 ('8','2'), ('9','2'), ('10','2'), ('11','2'), ('12','2'), ('13','2'), ('14','2'),
                 ('15','2'), ('16','2'), ('17','2'), ('18','2'), ('1','3'), ('2','3'),
                 ('3','3'), ('5','3')]
    counter = 0
    for week in week_info:
        week_url = f"https://www.espn.com/nfl/scoreboard/_/week/{week[0]}/year/2022/seasontype/{week[1]}"
        counter = insertESPN(week_url, cur, conn, counter)
        insertWAPI(cur, conn)
```

write_json(filename, dict):

- Inputs name of file and a dictionary
- Writes the dictionary to a file of the name filename
- Returns nothing



```python
def write_json(filename, dict):
    with open(filename, 'w') as outFile:
        json.dump(dict, outFile, indent=2)
```

calc_season_avgs(cur) function:

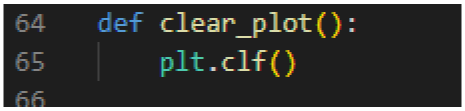
- Inputs the cursor
- Calculates and stores the average total points, yards, and turnovers per game in the 2022 NFL season into a dictionary
- Returns said dictionary
- Documentation of code in calculations and visualizations section

calc_betting_pcts(db) function:

- Inputs the cursor
- Calculates and stores the percentage of games that resulted in Over, Under, or Push being hit into a dictionary
- Returns said dictionary
- Documentation of code in calculations and visualizations section

clear_plot():

- Inputs nothing
- Clears out the matplotlib plotting space for the next graph
- Returns nothing



```python
def clear_plot():
    plt.clf()
```

create_pass_yrds_scatters(db):

- Inputs the cursor
- Plots two side-by-side scatter plots graphing the relationship between total passing yards and wind speed and visibility
- Returns nothing

- Documentation of code in calculations and visualizations section

create_pts_by_temp_plot(db):
- Inputs the cursor
- Plots a scatter plot graphing the relationship between total points scored and outdoor temperature
- Returns nothing
- Documentation of code in calculations and visualizations section

create_turnover_by_weather_plot(db):
- Inputs the cursor
- Plots a bar chart showing the average number of turnovers per weather condition
- Returns nothing
- Documentation of code in calculations and visualizations section

create_pct_rush_yrds_by_weather_plot(db):
- Inputs the cursor
- Plots a bar chart showing the percentage of rushing yards achieved (from total yards) by weather condition
- Returns nothing
- Documentation of code in calculations and visualizations section

create_ou_pie_charts_by_weather(db):
- Inputs the cursor
- Plots a series of pie charts showing the percentage of games that ended in Over, Under, or Push by each individual weather condition
- Returns nothing
- Documentation of code in calculations and visualizations section

Resources

Date	**Issue Description**	**Location of Resource**	**Result** **(did it solve the issue?)**
4/16	ProFootballReference data is unreachable due to comments in HTML	Stack Overflow: BeautifulSoup 4: Remove comment tag and its content	Helpful but did not solve the issue
4/18	Incorrect number of bindings supplied error	Stack Overflow	Yes, example of exact same problem w/ solution
4/19	Struggling to count instances from sqlite	Sqlitetutorial.net	Yes, very successful
4/20	Struggling with syntax and formatting of code	w3schools.com	Yes, being able to see examples of similar code allows me to better

			understand how to write certain parts of my functions
4/21	Struggling to call the right functions when trying to plot and chart	Python Functions Library	Yes, allowed me to figure out which functions to use where