Team: SheBotChoi
Members: Anthony Shepard, Jonah Botkin, Irene Choi
Github Repository: https://github.com/antshep-umich/SI206FinalSportsStats.git

SI 206 Final Project Report

Planned APIs/Websites and Data

Going into our project, we intended to use two APIs. We wanted to use an NHL API wrapper from pypi, and an NCAA API from sportradar. Additionally, we planned to scrape salary information from puckpedia.com. From these sources, we wanted to gather team and player information, including goals, points, penalty minutes, games, and salary.

Actual APIs/Websites and Data

In our actual project, we ended up gathering the same data as we'd planned, but in a slightly different way. We still used the NHL API to gather player game information and link players to teams. Next, we used Puckpedia's API to gather NHL player salary information. Finally, for our web scraping component, we scraped NCAA player game information and linked players to teams using hockeydb.com.

Problems we encountered

The biggest problem we encountered was with access to salary data. We wanted to scrape salary information, and had ideas for a couple of sources. However, all of them were protected from scraping. This meant we had to look for another way to find all of this. We came up with a couple of options, but all of them required private API keys. From there, we emailed a couple of them and ended up getting access to the Puckpedia API for free (eliteprospects.com wanted $3,750).

Additionally, while scraping data from hockeydb.com, we faced website blocking and IP address restrictions due to limitations on web scraping. This experience highlighted a key challenge in web scraping which is the uncertainty of how much data can be retrieved before encountering restrictions.

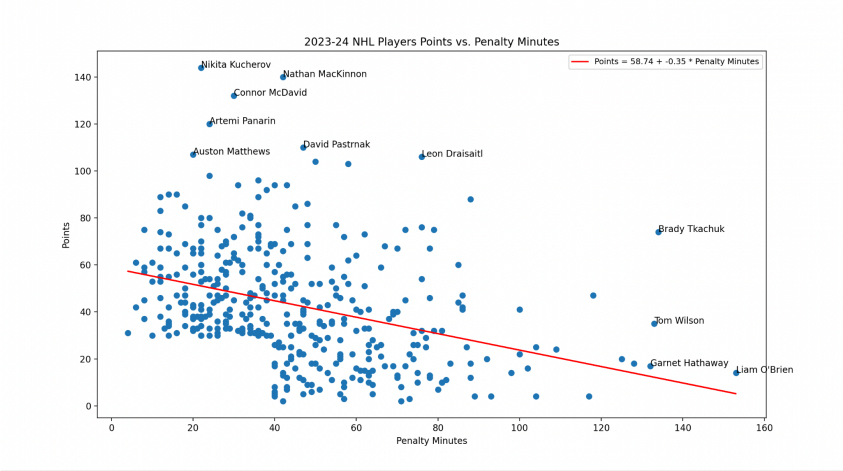
Calculations

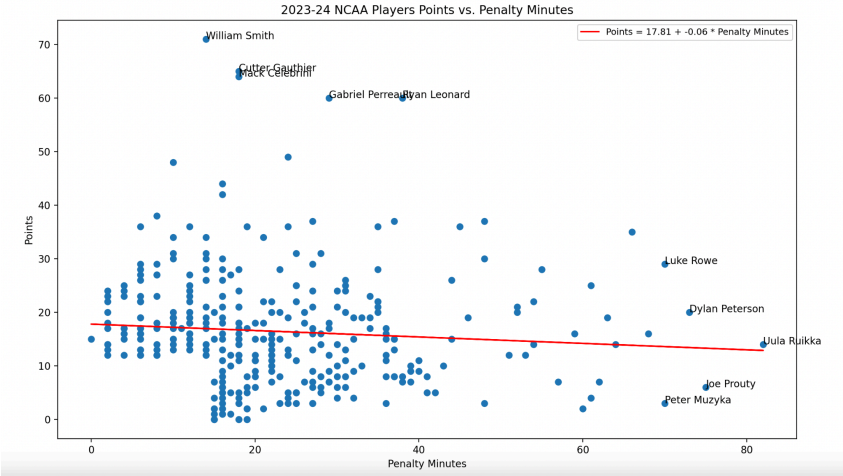
```
def write_team_csv(data, filename):
    header = ['Team', 'Goals', 'Penalties', 'Salary', 'Goals/million', 'Penalties/million']

    with open(filename, 'w', newline='') as csvfile:
        csvwriter = csv.writer(csvfile)

        csvwriter.writerow(header)


        team_dict = {}
        processed_data = []

        for row in data:
            if row[3]:
                teams = row[0].split(',')
                goals = row[1]
                penalties = row[2]
                salary = row[3]

                for team in teams:
                    if team not in team_dict:
                        team_dict[team] = {'Goals': 0, 'Penalties': 0, 'Salary': 0}

                    team_dict[team]['Goals'] += goals
                    team_dict[team]['Penalties'] += penalties
                    team_dict[team]['Salary'] += salary

        for team, stats in team_dict.items():
            if stats['Salary'] > 0:
                team_dict[team]['goals_per_million'] = stats['Goals'] / (stats['Salary'] / 1000000)
                team_dict[team]['penalty_minutes_per_million'] = stats['Penalties'] / (stats['Salary'] / 1000000)
            else:
                team_dict[team]['goals_per_million'] = 0
                team_dict[team]['penalty_minutes_per_million'] = 0



        for team in team_dict:
            row = [team, team_dict[team]['Goals'], team_dict[team]['Penalties'], team_dict[team]['Salary'],
                   round(team_dict[team]['goals_per_million'], 2), round(team_dict[team]['penalty_minutes_per_million'], 2)]
            csvwriter.writerow(row)
```

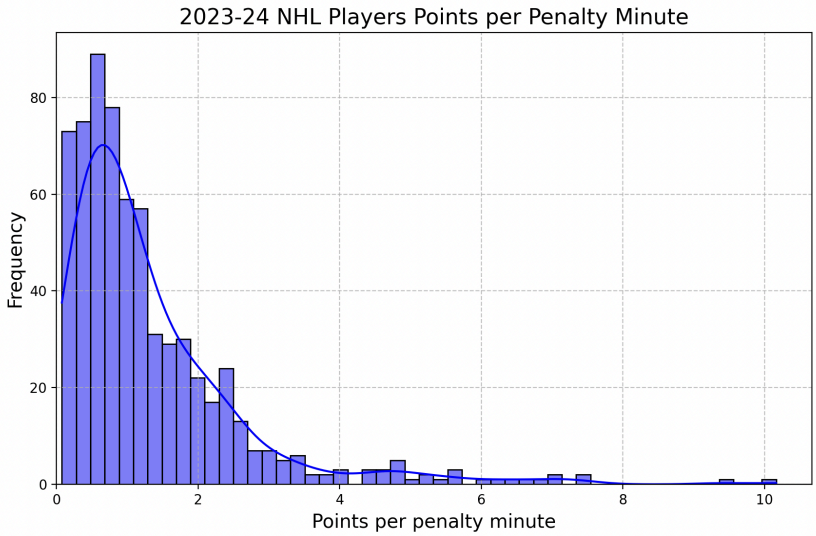
```
    query = f'''
    SELECT NHL_Teams.name, goals, penalty_min, salary
    FROM Players
    JOIN NHL_Teams
    ON Players.team_id = NHL_Teams.team_id
    '''

    cur.execute(query)
    result = cur.fetchall()

    return result
```

```
query = f'''
SELECT points, penalty_min
FROM {table}
WHERE games >= ? AND (points >= ? AND penalty_min >= ?)
'''

cur.execute(query, (min_gp, min_pts, min_pen))
result = cur.fetchall()

Points_per_pen = []
for player in result:
    print(player)
    Points_per_pen.append(player[0]/player[1])


return Points_per_pen
```

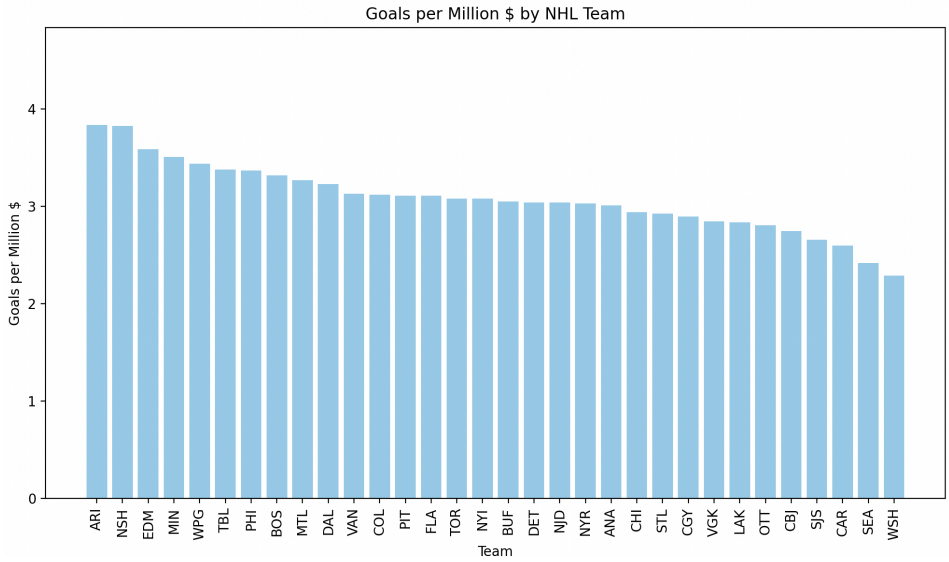
Visualizations



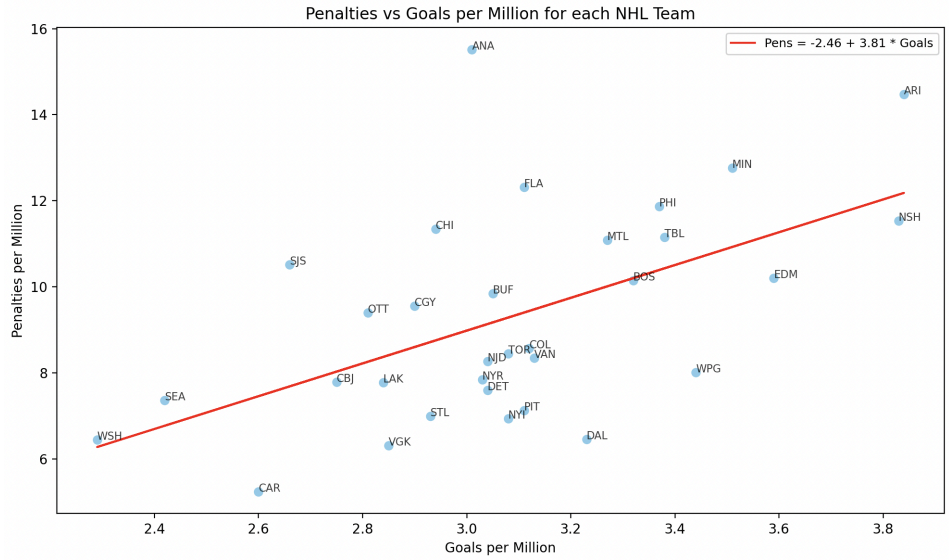




Extra 1:



Extra 2:



Instructions for running our code:

1. Run Penalty_vs_Points_Graph.py

Please note:
Salary pull from Puckpedia will not work without a private API key that is not stored on Github, as this would publicly expose the API key. It has been included in the .zip file and is read in using file io.

Documentation

```
def set_up_database(db_name):
    """
    Sets up a SQLite database connection and cursor.

    Parameters
    ----------
    db_name: str
        The name of the SQLite database.

    Returns
    -------
    Tuple (Cursor, Connection):
        A tuple containing the database cursor and connection objects.
    """
```

```
def graph_points_per_pen(Points_per_pen, league):
    """
    Creates graphs of points per penalty minute for the NHL and NCAA players

    Parameters
    ----------
    league: string
        NHL or NCAA

    pts_per_pen: List
        list of values for points per penalty minute

    Returns
    -------
    Nothing

    """
    plt.figure(figsize=(10, 6))
```

```
def graph_points_pens(points, pens, names, min_pts, min_pens, league):
    """
    Creates graphs of points against penalty minutes for NHL and NCAA players

    Parameters
    ----------
    points: list
        list of points by each player

    pens: list
        list of penalty minutes by each player

    names: list
        list of player names

    min_pts: int
        minimum number of points to display names

    min_pens: int
        minimum number of penalty minutes to display names

    Returns
    -------
    Nothing

    """
```

```
def get_pts_per_penalty_minute(table, min_gp, min_pts, min_pen, cur, conn):
    """
    Creates a list with all player points for players with at least a minimum number of games played

    Parameters
    ----------
    table: str
        Name of the table in the database.

    min_gp: int
        Minimum number of games played.

    min_pts: int
        Minimum number of points scored.

    min_pen: int
        Minimum number of penalty minutes.

    cur: Cursor
        The database cursor object.

    conn: Connection
        The database connection object.

    Returns
    -------
    pts_per_pen: List
        list of values for points per penalty minute
    """
```

```
def get_player_points_pens(table, min_gp, min_pts, min_pen, cur, conn):
    """
    Creates a list with all player points for players with at least a minimum number of games played

    Parameters
    ----------
    table: str
        Name of the table in the database.

    min_gp: int
        Minimum number of games played.

    min_pts: int
        Minimum number of points scored.

    min_pen: int
        Minimum number of penalty minutes.

    cur: Cursor
        The database cursor object.

    conn: Connection
        The database connection object.

    Returns
    -------
    Points (list):
        A list containing entries for each player's point total.

    Penalty_min (list):
        A list containing entries for each player's penalty minute total.

    Names (list):
        A list containing entries for each player's name.
    """
```

```
def set_up_database(db_name):
    """
    Sets up a SQLite database connection and cursor.

    Parameters
    ------------------------
    db_name: str
        The name of the SQLite database.

    Returns
    ------------------------
    Tuple (Cursor, Connection):
        A tuple containing the database cursor and connection objects.
    """
```

```
def write_team_csv(data, filename):
    """
    Creates a list with all player points for players with at least a minimum number of games played

    Parameters
    ------------------------
    data: List of tuples
        Each tuple contains player details including team name, goals, penalty minutes, and salary.

    filename: str
        The name of the CSV file to write data to.



    Returns
    ------------------------
    None

    """
```

```
def get_info(cur, conn):
    """
    Creates a list with all player points for players with at least a minimum number of games played

    Parameters
    ------------------------
    None

    Returns
    ------------------------
    Result: List of tuples
        A list of tuples representing players team, goals, penalty minutes, and salary
    """
```

```
def set_up_database(db_name):
    """
    Sets up a SQLite database connection and cursor.

    Parameters
    ------------------------
    db_name: str
        The name of the SQLite database.

    Returns
    ------------------------
    Tuple (Cursor, Connection):
        A tuple containing the database cursor and connection objects.
    """
```

```
def penalties_per_mil_graph(data):
    """
    Reads team-related player information from a CSV file.

    Parameters
    ------------------------
    data: list of lists
        A list of lists, each representing a row of the CSV file.

    Returns
    ------------------------
    None
    """
```

```
def goals_per_mil_graph(data):
    """
    Reads team-related player information from a CSV file.

    Parameters
    ------------------------
    data: list of lists
        A list of lists, each representing a row of the CSV file.

    Returns
    ------------------------
    None
    """
```

```
def get_player_data():
    """
    Gets a dictionary containing all player stats from the 23/24 season from the NHL API.
    Goes through a wrapper from https://github.com/coreyjs/nhl-api-py

    Parameters
    ----------
    none

    Returns
    ----------
    Dictionary {'data':[{player_id, name, games, points, penalty_min, avg_icetime, goals, assists, p
        A dictionary containing a list of players and their stats for the season.
    """
```

```
def set_up_player_table(data, cur, conn):
    """
    Sets up the Players table in the database using the provided NHL Player data.
    Calls the add_salary function to get salary data and add it to the DB from the Puckpedia api.

    Parameters
    ----------
    data: dictionary
        dictionary of Player data in JSON format.

    cur: Cursor
        The database cursor object.

    conn: Connection
        The database connection object.

    Returns
    ----------
    None
    """
```

```
def add_salary(cur, conn):
    """
    Adds player salary data from Puckpedia API.

    Parameters
    ----------
    cur: Cursor
        The database cursor object.

    conn: Connection
        The database connection object.

    Returns
    ----------
    Nothing
    """
```

```
def testpd():
    """Just a test function

    Parameters
    ----------
    Nothing

    Returns
    ----------
    Nothing
    """
```

```
def get_data():
    """
    Calls the set_up_player_table function and the PIM.get_college_players function.

    Parameters
    ----------
    Nothing

    Returns
    -------
    Nothing
    """
```

```
def get_page_content(url):
    """
    Takes a URL and returns a BeautifulSoup object from the response.

    Parameters
    ----------
    URL:
        A website URL

    Returns
    -------
    A BeautifulSoup object or nothing.
    """
```

```
def get_team_links(base_url):
    """
    Gets all of the team URLs from the NCAA stats page with the teams listed.

    Parameters
    ----------
    base_url:
        The page URL for the NCAA team stats page.

    Returns
    -------
    team_links:
        A list of URLs for the teams.
    """
```

```
def get_season_link(team_url):
    """
    Checks a team's URL to see if they have a team listed for the 2023-2024 season
    and returns the link to that season's stats for the team.

    Parameters
    ----------
    team_url:
        A NCAA team's URL.

    Returns
    -------
    Nothing or a string with the URL for the 2023-2024 season for that team.
    """
```

```
def penalties_vs_goals_per_mil_graph(data):
    """
    Graphs penalties per million in salary against goals per million in salary.

    Parameters
    ----------------------
    data: list of lists
        A list of lists, each representing a row of the CSV file.

    Returns
    ----------------------
    None
    """
```

```
def team_graphs():
    """
    Calls all the functions to do what this file does.

    Parameters
    -------------------------
    None

    Returns
    -------------------------
    None
    """
```

```
def scrape_players(season_url, team_name):
    """
    Scrapes the player data from the 2023-2024 season stats for a team.

    Parameters
    ----------------------
    season_url:
        The URL from get_season_link for this team.

    team_name:
        The team this data is for

    Returns
    ----------------------
    players_data:
        A list of player dictionaries with each player's stats.
    """
```

```
def set_up_ncaa_table(cur, conn):
    """
    Creates the NCAA_Teams and NCAA_Players tables in the DB if they don't exist.

    Parameters
    ----------------------
    cur:
        The database cursor

    conn:
        The database connection

    Returns
    ----------------------
    Nothing
    """
```

```
def write_csv():
    """
    Writes CSV with team calculations

    Parameters
    -------------------------
    None

    Returns
    -------------------------
    None
    """
```

```
def insert_player_data(players, cur, conn):
    """
    Iterates through the player data and adds any new data to the NCAA_Players table.

    Parameters
    ----------------------
    players:
        The list of player dictionaries from scrape_players

    cur:
        The database cursor

    conn:
        The database connection

    Returns
    ----------------------
    Nothing
    """
```

```
def get_college_players(cur, conn):
    """
    Utilizes the prior defined functions in PIM.py to scrape and add player data.

    Parameters
    ----------------------
    cur:
        database cursor

    conn:
        database connection

    Returns
    ----------------------
    Nothing
    """
```

Resources

Date	Issue Description	Location of Resource	Result
11/22	Nhl-api-py returning same 70 players	https://github.com/coreyjs/nhl-api-py	Emailed maintainer and was provided link to ticket addressing issue with query parameters: https://github.com/coreyjs/nhl-api-py/issues/83
11/22	Needed 2nd API resource. Emailed Puckpedia support	puckpedia.com	Was gifted a private API key to their service