CS Semester Project

Drew Haynes (rbc6wr), Sarah Rodgers (pjk2wq) and Thomas McIntyre (gem5cm)

Introduction

The excitement of college football immerses and intrigues the American population every fall. In order to better understand the predictive odds of a team's success in a season and to identify trends over time, exploratory data analysis (EDA) is key to make these inferences. To do so, our project focuses on building an interactive tool that allows a user to perform exploratory analyses on college football teams from 2013 to 2020. This tool will then allow the user to make data-driven inferences and predictions using the statistics and trends identified.

There is a massive amount of data that is tracked in college football games, thus building a tool to allow for quick user interactivity to learn more about potential variables of interest quickly would be beneficial. Using the python fundamentals we developed in CS5010 we will be able to find the correct packages to download and use to create our first interactive visualization tool.

As alluded to above, one of the main goals is to better understand the predictive odds of a team's success resulting from many variables within the data set. This tool will be useful in accelerating the process of exploratory data analysis (EDA). Our tool focuses on college football over the past couple years, however we will be able to determine the effectiveness of building a tool similar to ours for other domains. Another goal of the project is to determine whether we believe that our application will be easily repeatable for data sets in other areas of interest.

The Data & Experimental Design

Our data contains college football statistics since 2013 with a different data set for each year with more than 100 variables to be utilized. The data sets were obtained from Kaggle and can be found using the link here**.** The use of pandas will be crucial in cleaning our data since some of the years have some extra variables. The variables in the data sets range from simple everyday football statistics such as wins and losses to more complex statistics such as red zone touchdowns and red zone rushing touchdowns. The levels of specificity ranges greatly throughout the dataset. Thus, the idea of building a visualization tool came to fruition.

Since the data sets are broken out by each year of college football statistics, we had to perform data pre-processing and cleaning techniques in order to successfully utilize the variables and records across each data set within the tool. The basis of the tool is built around one core dataframe that includes the data across each year of statistics. In order to concatenate the separate data sets, college team names were standardized to the following format 'School (Conference)'. This included ensuring all abbreviated text forms such 'St.' were spelled out as 'State'. After doing so, additional columns were created that contain the following information: year of the statistic, win percentage, school, and conference.

In order to enhance our tool and the generated visualizations, we decided to integrate each college football team's logos as the main data point icon within the plots. Using Beautiful Soup, we scraped ESPN's College Football Power Index (FPI) 2020 to gather the logo for each college football team. More specifically, we used Beautiful Soup methods to extract the text from the ESPN site and store this as an element. In order to access the contents of each element, we looped through the element and extracted the texts from the tags using regular expressions and stored this into a dataframe. This logo is then used to represent each team in the data visualizations.

After parsing through this information, we identified that the college football team names from Kaggle and ESPN were in different formats. The data from Kaggle contained the school name and conference whereas the ESPN data contained the school name and mascot. In order to cohesively associate the logo with each school, we scraped Wikipedia's List of NCAA Division I FBS Football Programs to retrieve the school name and mascot and stored this information into respective lists. Using this information, we then mapped the logos and mascot of each school in the data frame. Lastly, using the melt function in python we re-shaped the data frame so that each college football team statistic and year runs vertically rather than horizontally.

The interactive tool was then developed using the Dash package as the web framework for the application. The Dash library allows for flexible and reactive web applications to be created for data analysis, exploration, and visualization. The tool is dependent on user activity to produce the desired visualizations. The user begins by selecting the statistics of interest in the X-variable Selection and Y-variable Selection fields. These values are populated in a drop-down list from the variables that exist across each year of the college football statistics. If a selected value was not reported for the selected year, the value will default to a valid selection. This prevents the user from selecting a statistic that cannot successfully generate a visualization. For example, if a variable existed in only one year's college football statistics dataset but not another, this variable will not be an option for the user to select. This allows for error prevention as each of the plots are dependent on the cleaned data frame mentioned earlier. Additionally, the user may select the year of interest to explore with options from 2013 to 2020 by utilizing the Select Year filter.

Once the variable is selected three plots are generated within the Conference Filter pane. This includes a scatter plot displaying the distribution of the selected statistics with each data point representing a college football team. The user default Conference Filter value is 'FBS' which stands for Division I Football Bowl Subdivision (FBS) of the National Collegiate Athletic Association (NCAA) in the United States. A specific conference of interest can be filtered on by selecting a value of interest from a predefined drop-down list and the subsequent plots will update based on the selected value. Additionally, each college football team is represented by their respective logo. By hovering over the data point, the user can see the team's college name and the associated statistical values (i.e., the x and y values). The scatter plot is dynamic and reactive to zoom in and lasso quadrants of interest to further analyze.

The user then can drill down on a team of interest to better understand their trends of the statistics over time. A line plot will be generated representing each of the statistics selected over

the years available. This allows for the user to analyze statistics trends for each college football team and identify their successes and failures. As a data analyst or coach, this will be a helpful capability to identify areas of improvement.

Results

The main result of our project is the data visualization tool created. The application can be found following this link and cloning the GitHub repository to a local machine (tjmcintyre/CS5010_Semester_Project: CS5010 Semester Project (github.com)). Once the data sets and python scripts are stored on the local machine they can be run to examine how the different pieces of the project were constructed. The cfb_dictionaries.py script contains the web-scraper for additional data and logos, the cfb_dataframe.py contains the data cleansing portion of the project, and finally the cfb_app.py contains the code using the Dash package in order to make an applet.

After running the first two scripts, the results section will focus on the last script which produces our data visualization tool. When the cfb_app.py script is executed, then python will display a url to enter into the web browser of choice (developmental testing was primarily conducted on Google Chrome). The landing page of the application will appear similar to Figure 1 below. The tool is styled using CSS for the formatting, color scheme, and fonts.

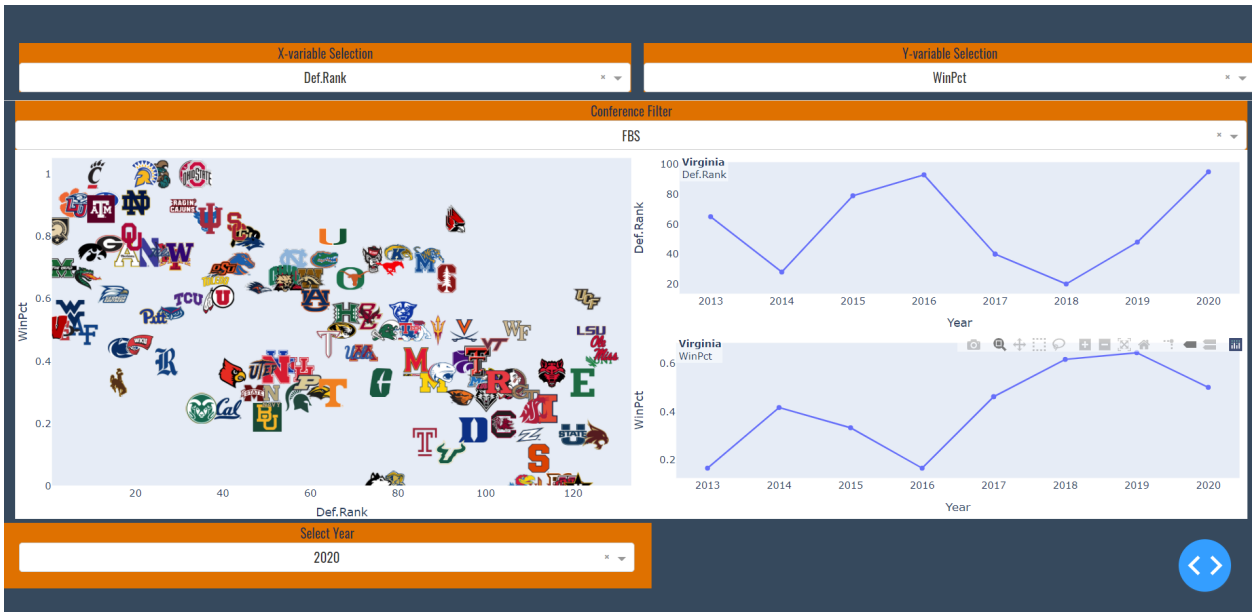

Figure 1: Tool Default Homepage

In Figure 1, there are 4 visible drop down menus to query with to produce unique visualizations on the tool. The first two drops down menus are the X-variable Selection and the Y-variable selection. Within these drop down menus the user is able to select between the 100+ variables in our data sets. This allows the user to have hundreds of different options to compare on the scatterplot while also producing time series for each of the X and Y variables selected. In the

screenshot above, the time series is showing the data for Virginia at the moment. However, by hovering over a different team's logo the tool will update the time series to display that particular school. Also, when hovering over a team's logo the tool will display the X and Y value of that data point as a tooltip.

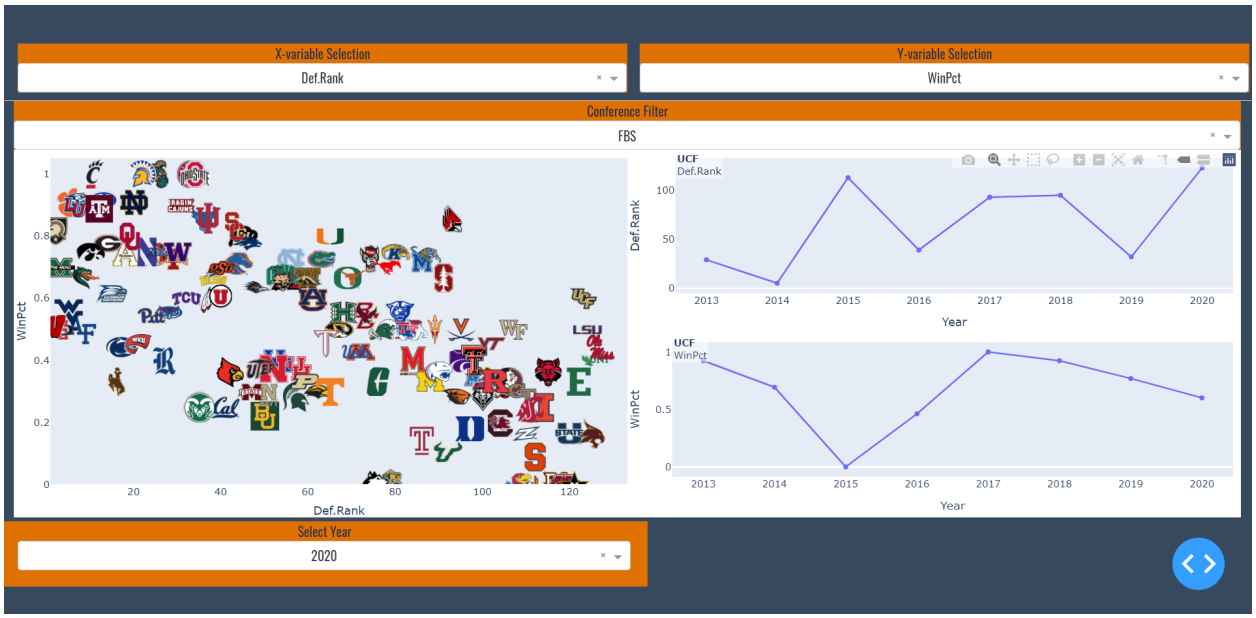

Figure 2: College Team Drilldown

Figure 2 displays the same X and Y variable as the previous image, however UCF was selected instead of Virginia. Now the Defense Rank and WinPct time series display the time series for UCF instead. The other two drop down features primarily affect the scatterplots data. The first one is the Conference Filter which would allow the user to select a particular conference of interest to examine teams they personally care about. The second drop down filter at the bottom of the tool is to select the year. This will adjust the X and Y variable to the football team's statistics from that particular year in the scatterplot. For example, I will adjust the conference filter to the ACC to see how Virginia falls in the scatterplot comparing WinPct vs Def.Rank. This selection does not impact the time series displays.

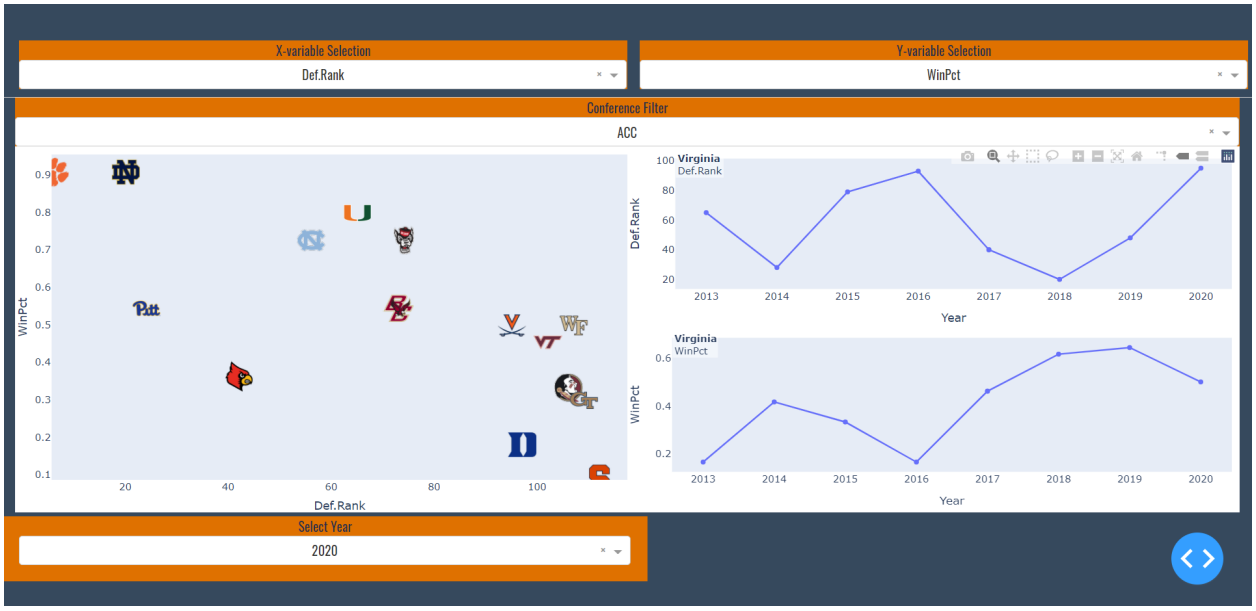

Figure 3: Conference Filtering

In Figure 3, our query is now narrowed down to only the ACC teams. This helps to remove a lot of the noise in the scatterplot if a user is only interested in comparing these particular statistics within a given conference. The Virginia logo was clicked on once again in the new scatterplot to show how the time series plots appear the same for Figure 1 and Figure 3. Filtering by conference will only have a change on the scatterplot. The last filter available in the tool is the "Select Year" drop down button. For demonstration purposes, the year will now be moved from 2020 to 2019 to compare and see the changes in the visuals.

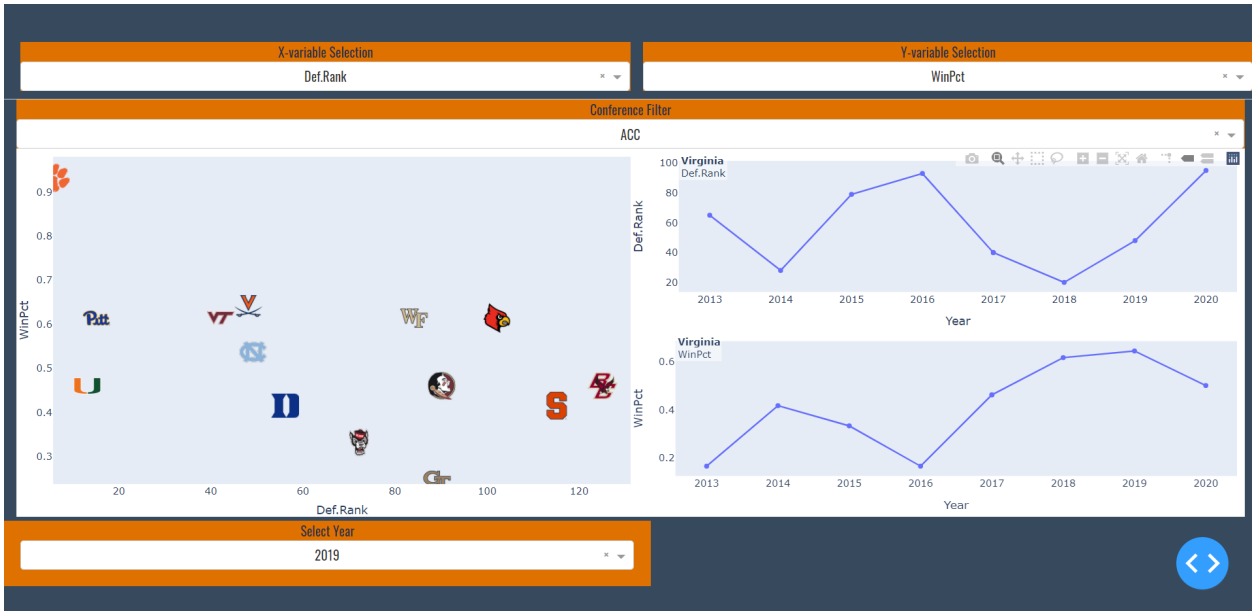

Figure 4: Year Filtering

Like the previous filter, the only changes to the tool from adjusting the year filter to 2019 comes in the scatterplot. The time series plots are still selected for Virginia, however the scatterplot has changed. In 2019 Virginia had a better (lower) Def.Rank and a higher WinPct. The results in the 2019 scatter plot could be expected by following the timeseries in the previous screenshots. There are also several built in features to each of the visualizations within the tool, which will be discussed below.

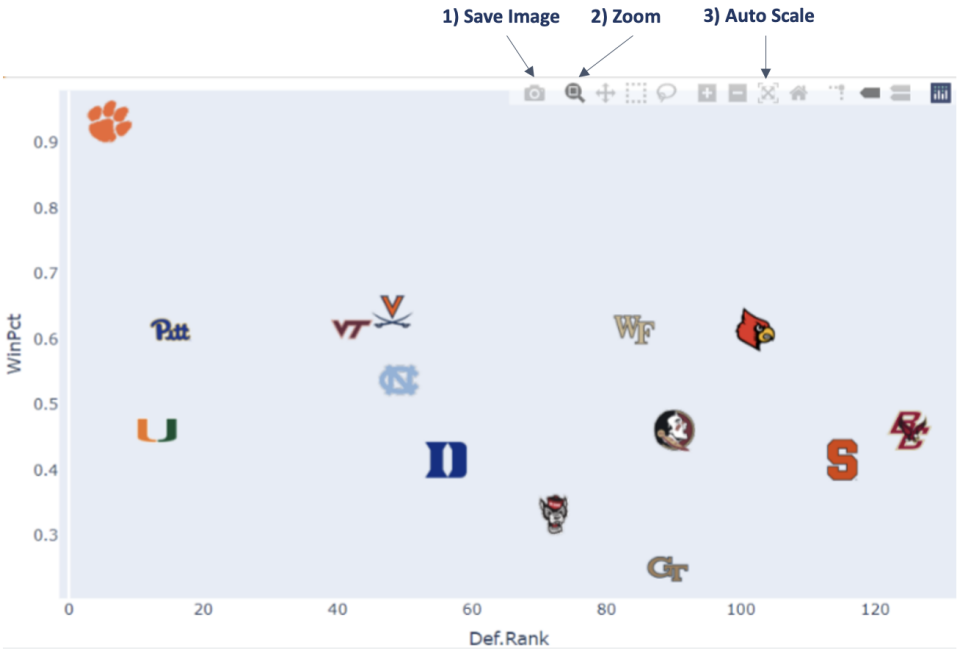

Figure 5: Additional Built-in Features

The features that are built into the visualizations allow for easy ways to explore the data and potentially save plots for potential EDA sections of reports. As annotated in Figure 5, the user can click the camera button at the top of the plot to save the image as a png file. The user can zoom in and out, or even autoscale the figure to produce a perfect image for a report. These additional features help make the visualization tool perfect for EDA.

Between the functionality of all of the filters, the built-in features, and the interactivity of the application, we believe that using python to create an interactive application was productive. Learning how to leverage the Dash package to create an interactive data visualization experience proved to have many potential benefits. Following the structure of our project, we believe that we would be able to create applications like this for other domains that could reap massive benefits for particular populations. Our application in particular could be very beneficial for a variety of users. An ordinary casual college football fan would be able to explore the potential relationships between a multitude of statistics, while a college football coach could examine how his team is stacking up against the rest of their conference across different seasons.

Testing

Due to the structure of the interactive application being dependent on user activity of pre-populated values, typical unit testing is difficult to perform. We performed testing on the information scraped from the ESPN and Wikipedia sites to ensure the content was parsed correctly. Examples of the test cases are provided in the Figures below:

```python
# testing that the web scraped information was pulled correctly from ESPN
class CFBESPNTestCase(unittest.TestCase): #must inherit from unittest.TestCase

    # Are all school names stored correctly as keys
    def test_9(self):
        self.assertEqual(len(logos.keys()),  131)

    # Are all logos stored correctly as values
    def test_10(self):
        self.assertEqual(len(logos.values()),  131)

    # Is the expected logo id associated with the correct school
    def test_11(self):
        self.assertTrue('50' in logos['Penn State'])

    def test_12(self):
        self.assertTrue('333' in logos['Alabama'])

    # Check that there are no missing logo values for schools with known logos
    def test_13(self):
        self.assertFalse(logos['Auburn'] == '', msg= "The Auburn logo is not missing")

    def test_14(self):
        self.assertFalse(logos['Navy'] == '', msg= "The Navy logo is not missing")
```

Figure 6: Test Cases for ESPN

```python
# testing that the web scraped information was pulled correctly from Wikipedia
class CFBWikipediaTestCase(unittest.TestCase): #must inherit from unittest.TestCase

    # Are all school names stored correctly as keys
    def test_1(self):
        self.assertEqual(len(nicknames.keys()), 131)

    # Are all mascot names stored correctly as values
    def test_2(self):
        self.assertEqual(len(nicknames.values()), 131)

    # Is the expected mascot associated with the correct school
    def test_3(self):
        self.assertEqual(nicknames["Penn State"], 'Nittany Lions')

    def test_4(self):
        self.assertEqual(nicknames["Virginia"], 'Cavaliers')

    def test_5(self):
        self.assertEqual(nicknames.get('Air Force'), 'Falcons')

    def test_6(self):
        self.assertEqual(nicknames.get('Syracuse'), 'Orange')

    # Check that there are no missing mascot values for schools with known mascots
    def test_7(self):
        self.assertFalse(nicknames['Ohio State'] == '', msg= "The Ohio State mascot is not missing")

    def test_8(self):
        self.assertFalse(nicknames['Michigan'] == '', msg= "The Michigan mascot is not missing")
```

Figure 7: Test Cases for Wikipedia

In order to test the statistics and filters used in the visualizations, test cases were used on the data frame that the application is dependent on. Below are example test cases assessing the expected outputs of the filters and statistics.

```
# testing that the dataframe used for the visualizations outputs the expected statsitics
class CFBVisualizationTestCase(unittest.TestCase): #must inherit from unittest.TestCase

    # Is the highest offensive ranked team what is expected for various years?
    def test_15(self):
        # For 2020
        temp = df_cfb[df_cfb['Indicator Name'] == 'Off.Rank']
        temp = temp[temp['Year'] == 2020]
        temp['Value'] = temp['Value'].astype(str).astype(int)
        max_off_rank = temp.loc[temp['Value'].idxmin()]
        self.assertEqual(max_off_rank['School'],  'Kent State')

    # Is the highest offensive ranked team what is expected for various years?
    def test_16(self):
        # For 2013
        temp = df_cfb[df_cfb['Indicator Name'] == 'Off.Rank']
        temp = temp[temp['Year'] == 2013]
        temp['Value'] = temp['Value'].astype(str).astype(int)
        max_off_rank = temp.loc[temp['Value'].idxmin()]
        self.assertEqual(max_off_rank['School'],  'Baylor')

    # Is the conference filter working properly?
    def test_17(self):
        # For AAC
        aac_conf = df_cfb[df_cfb['Conference'] == 'AAC']
        aac_teams = np.array(['Cincinnati', 'Houston', 'Louisville', 'Memphis', 'Rutgers', 'SMU',
       'South Florida', 'Temple', 'UCF', 'UConn', 'East Carolina',
       'Tulane', 'Tulsa', 'Navy'])
        self.assertEqual(print(aac_conf['School'].unique()), print(aac_teams))

    # Is the conference filter working properly?
    def test_18(self):
        # For Big Ten
        big_ten_conf = df_cfb[df_cfb['Conference'] == 'Big Ten']
        big_ten_teams = np.array(['Illinois', 'Indiana', 'Iowa', 'Michigan', 'Michigan State',
       'Minnesota', 'Nebraska', 'Northwestern', 'Ohio State',
       'Penn State', 'Purdue', 'Wisconsin', 'Maryland', 'Rutgers'])
        self.assertEqual(print(big_ten_conf['School'].unique()), print(big_ten_teams))
```

Figure 8: Test Cases for Visualization Parameters

Each test case ran successfully with no errors. The output of the test cases is stored in the txt file provided in the zipped submission titled ‘CFB Testing.txt’. We used various Assert Methods such as assertEqual and assertTrue to identify if the contents of the resulting dictionaries contained the expected information. After performing these tests, we were confident that the information scraped from the web was accurate and can be used as a source for the tool.

Conclusion

This tool can further be expanded or leveraged for a different purpose or domain. This may include a different college sport, outside of the college realm, or simply for exploratory data analysis. The versatility of python and the packages available allow for a variety of different approaches to create visualizations. The use of the “Dash” package allowed to create a successful application to help the process of EDA in datasets. The framework of our application could be tweaked and modified to fit the needs of other data sets.

Additionally, we would like to further expand the tool to incorporate feedback from our peers. This includes including trend and/or regression lines in the plots to provide further insight on the relationship between the statistics of interest. Furthermore, we will explore incorporating preset statistics that will serve as recommended dashboards so that frequent users have the ability to

reference commonly used statistics and new users that may be unfamiliar with college football have suggestions for the visualizations. While doing so, we will provide more detail about what each statistic represents to limit the confusion while selecting the respective statistic.

The culmination of the lessons in CS5010 provided the foundational knowledge to complete this project. The incorporation of web scraping allowed us to have dynamic images in the scatterplot. The modules on unit testing allowed us to be comfortable that the data was scraped correctly. The understanding of pandas paved the way for cleaning, merging and appending data sets to form one master set. Learning to create visualizations within python aided in creating the plots within the dash application. Finally, the basic understanding of python allowed us to figure out how to work with a brand new package like “dash”.

References

College Football Team Stats Seasons 2013 to 2020. (2020, December 28). Kaggle. https://www.kaggle.com/jeffgallini/college-football-team-stats-2019

ESPN: Serving sports fans. Anytime. Anywhere. (n.d.). ESPN.Com. https://www.espn.com/college-football/fpi/_/season/2020

Wikipedia contributors. (2021, April 21). *List of NCAA Division I FBS football programs*. Wikipedia. https://en.wikipedia.org/wiki/List_of_NCAA_Division_I_FBS_football_programs