Final Project

Tyler Ursuy

May 16, 2019

Intro:

My dataset is made up of ten years of English Premier League (PL) soccer seasons hosted in Great Britain. The data includes the date, home team, away team, home team goals, away team goals, the referee for the match, home and away team red and yellow cards, and more. Each row is comparable to a less detailed box score report. I was lucky enough to find a website that had all of this conveniently in one place and downloaded all ten CSVs, one for each season, from datahub.

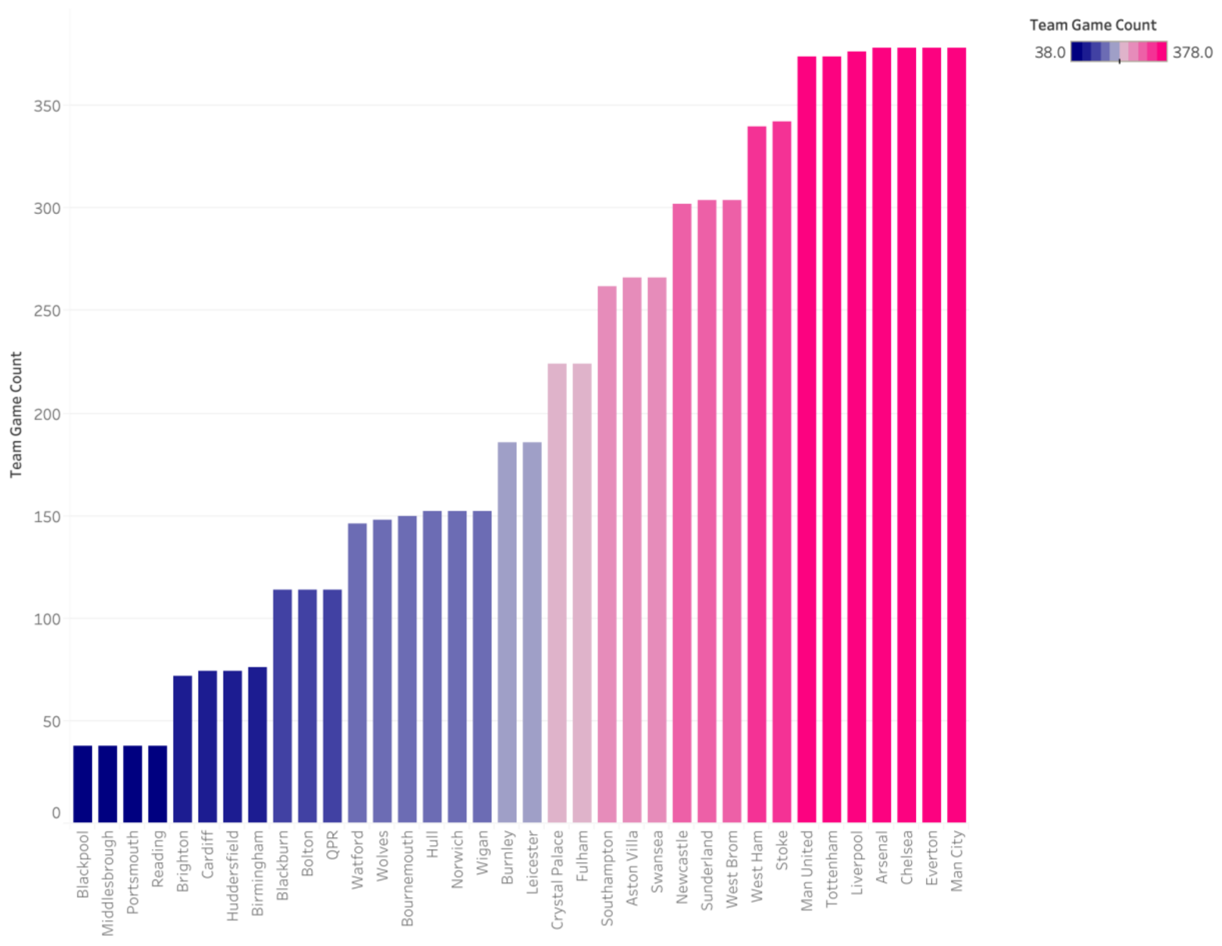
As someone who has followed the PL for many years, I was very excited to find a dataset that would allow me to learn more about my favorite league. Additionally, I knew that I would be able to tease out plenty of insights by asking the right questions since I have some level domain expertise. The PL also happens to be the home of my favorite team, Manchester United, and the opportunity to be a data driven fan was hard to pass up on.

The dataset provided a great foundation to start but was lacking in some areas. Firstly, the locations of each team were not listed; however, I was able to manually add this by matching each to team to its corresponding subnational territory listed in the Nomenclature of Territorial Units for Statistics, commonly referred to as NUTS Europe. For the purposes of one particular visualization below, I gathered the coordinates of London team stadiums. Next, as I will expand on later, I added the number of times each team has won the PL. The last piece of additional information I had to find was the rival of each team. Each of these was done by internet searches and manually recorded. Any additional information used other than the ones just listed was done by feature engineering using the data available.

While beginning the project, I had a few initial questions that I wanted answers for, but my curiosity grew with each type of visualization. One of my first questions aimed to find out what teams would be leading if I treated all ten years as one season, that is, keeping track of a cumulative point count for ten years rather than resetting each year. I then wanted to investigate the impact of referees on games by observing how many yellow and red cards each referee issued. Next, I was curious to see if teams played differently when they were home against being away. These are only a few initial questions, but I will elaborate on these and many more with this report.

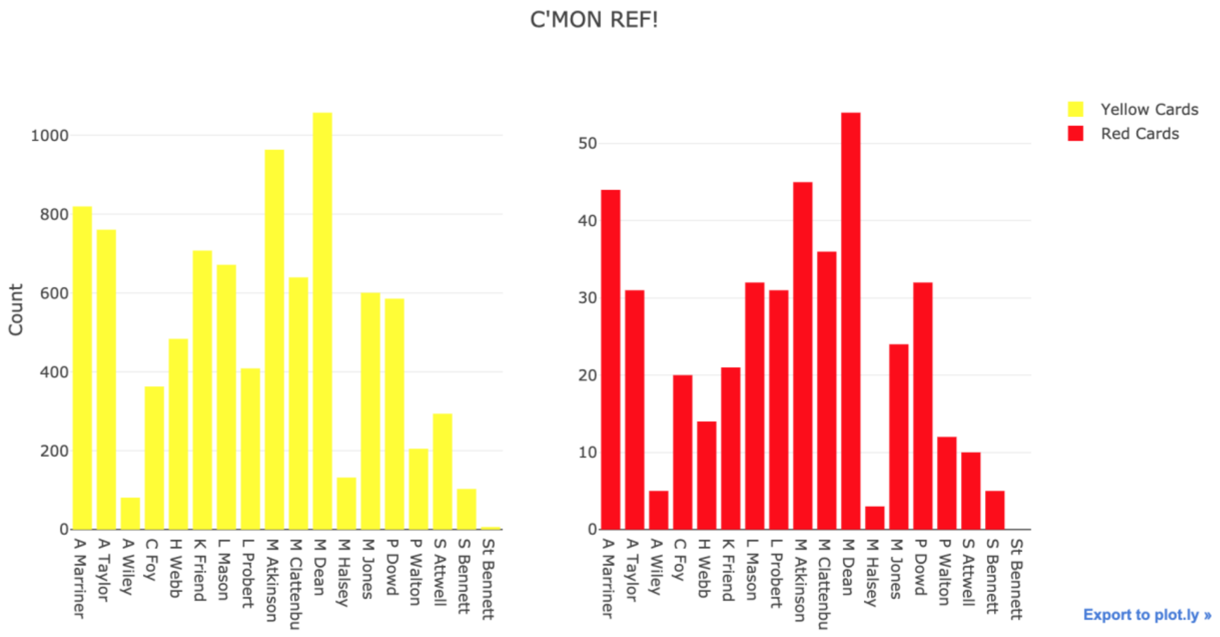
Please see the next pages for visualizations and a short explanation for each.

Histogram:



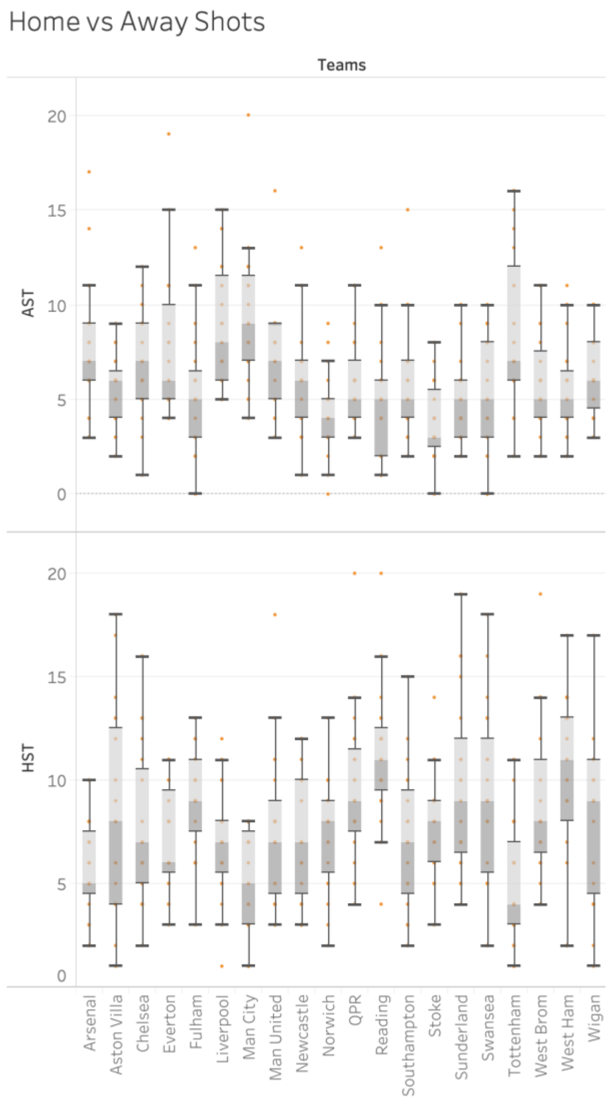
Each, year English soccer clubs fight to either avoid relegation, seek promotion, or be crowned champions. The bottom three teams at the end the year in each league are relegated to the league below. Then the top three teams of each league, except the Premier League, are promoted to next league above. Finally, the top team in the Premier League is crowned champion. With each year consisting of 38 games played per team, we are able to see through a game count the number of years a team has played in the Premier League in the last ten years. For example, we can see teams like Middlesbrough have only been in the league one out of ten years. On the other end of the spectrum, teams like Manchester United have played in the Premier League consistently for the past ten years.

Bar plot:



One of many influential factors in any sport is the referee. In soccer, the referee penalizes players for particularly bad fouls with either a yellow or red card. Yellow cards serve as warnings, while red cards force the player to leave the game, is banned from the next game, and the team must play the rest of the game one person short. In addition, if a red card is issued, the league officials can review the incident after the game to determine if further punishment is required, such as being banned from play for an additional number of games and/or fines. In addition, if a player receives two yellow cards in the same game, the player is then showed a red with the same implications (usually without the post-game review, as this was most likely a less severe foul). Therefore, the referee has a great influence of the course of the game. The bar plot above shows how many yellow and red cards each referee has shown to players over the past 10 years. As we can see, some referees are very particular about showing cards, while some hand out cards like candy on Halloween. For example, C Foy has given 363 yellow cards and 20 red cards. On the other hand, M Dean has shown 1058 yellow cards and 54 red cards. However, one problem with this bar plot is that it does not account for a referee retiring, or another beginning in the course of these 10 years. Thus, some of the referees might look like they never give cards such as A Wiley, who has only recorded 81 yellow cards and 5 red cards. I would suspect that A Wiley either retired soon after 2009 or has started his PL referee career within the past year. Furthermore, it is interesting that we see almost the same pattern in both plots where referees with high yellow card counts relative to other referees also have high red card counts relative to other referees. Another takeaway from these plots is that it is clear that referees do officiate uniformly and can be very strict or lenient. So, as a player, especially one who tends to foul often, it would be useful to know who is officiating the game.

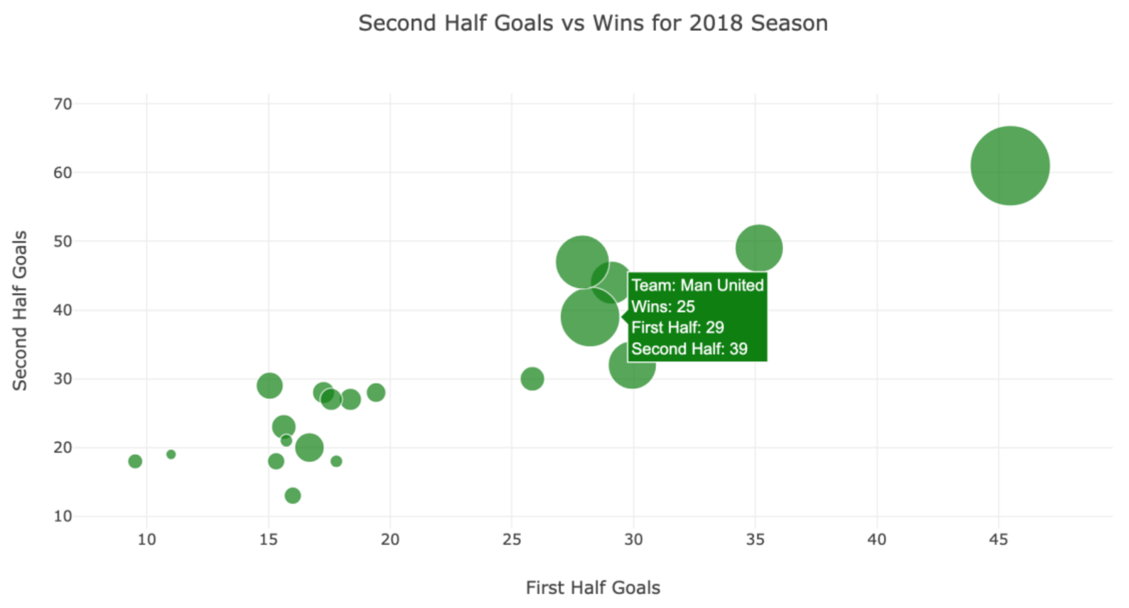
Boxplot:



AST = Away shots
HST = Home

As I mentioned above, I was curious to see if teams performed differently depending if they were at home or away. I wanted to test this by seeing how many shots towards goal each team takes in each situation. It seems that there actually is a difference by observing the boxplot. One can see that teams typically have a higher mean shot count at home, as well as a higher variance. This could be because teams are often tempted to play more conservatively on the road, compared to playing in front of their own fans cheering them on.

Scatterplot:

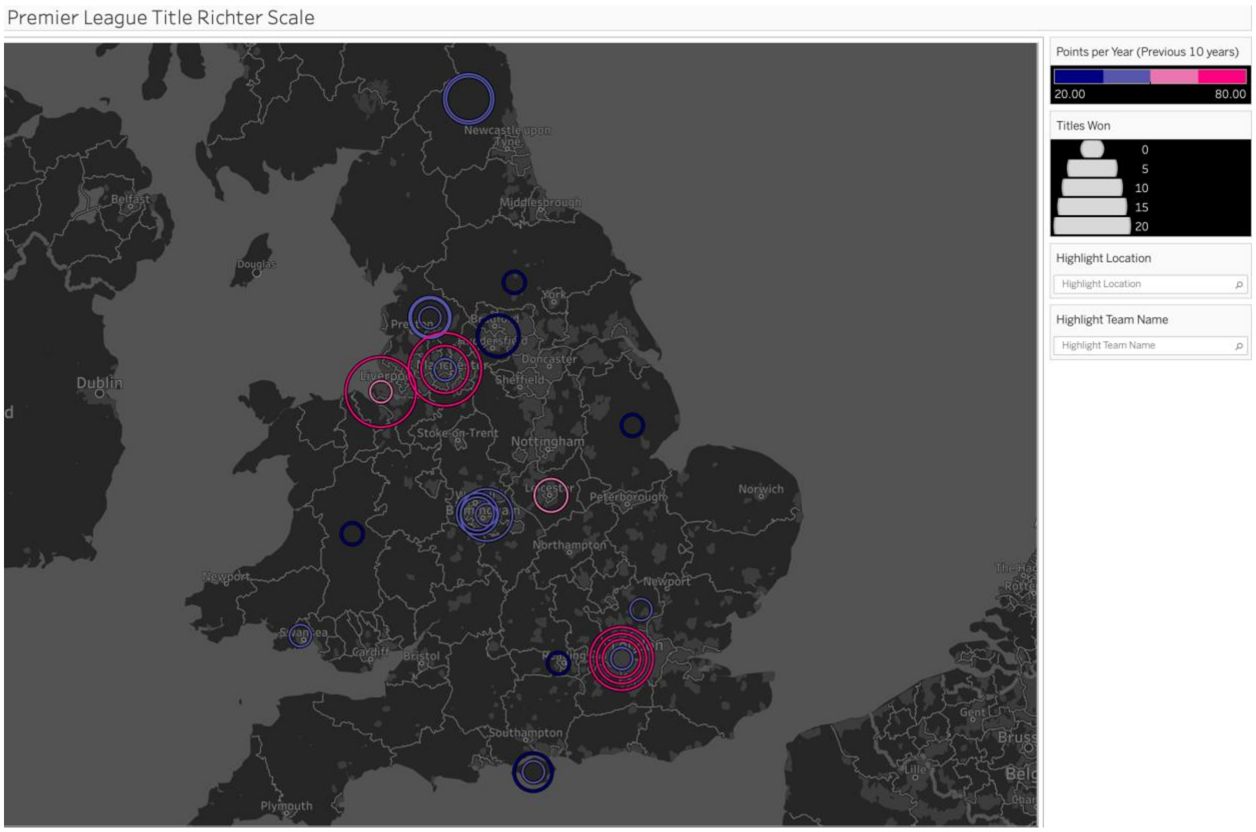


The scatter plot above shows the relationship between total first half goals, second half goals, and wins in the 2017/18 Premier League season. The axes correspond to each half and the size of the point reflects the number of wins the team earned in the year. As we would expect, the teams with more goals in both halves won more games; however, we do see in some cases that teams won more games but did not score as many goals as other teams. For example, Man United had the second most wins this year (25) but were outscored in either dimension by four teams that finished with less wins. We can conclude that Man United played more defensive this year, scoring less goals but also conceding less goals, and winning more as a result. A similar situation can be seen in Burnley's performance. On the other hand, Liverpool scored more goals in the first and second half but only had 21 wins, so we can assume that Liverpool probably conceded more goals and either lost or tied games because of it. Thus, we can tease out the attitudes some teams approached games with using this logic.

Here is the link to the interactive Plotly visualization: https://plot.ly/~tylerursuy/21.

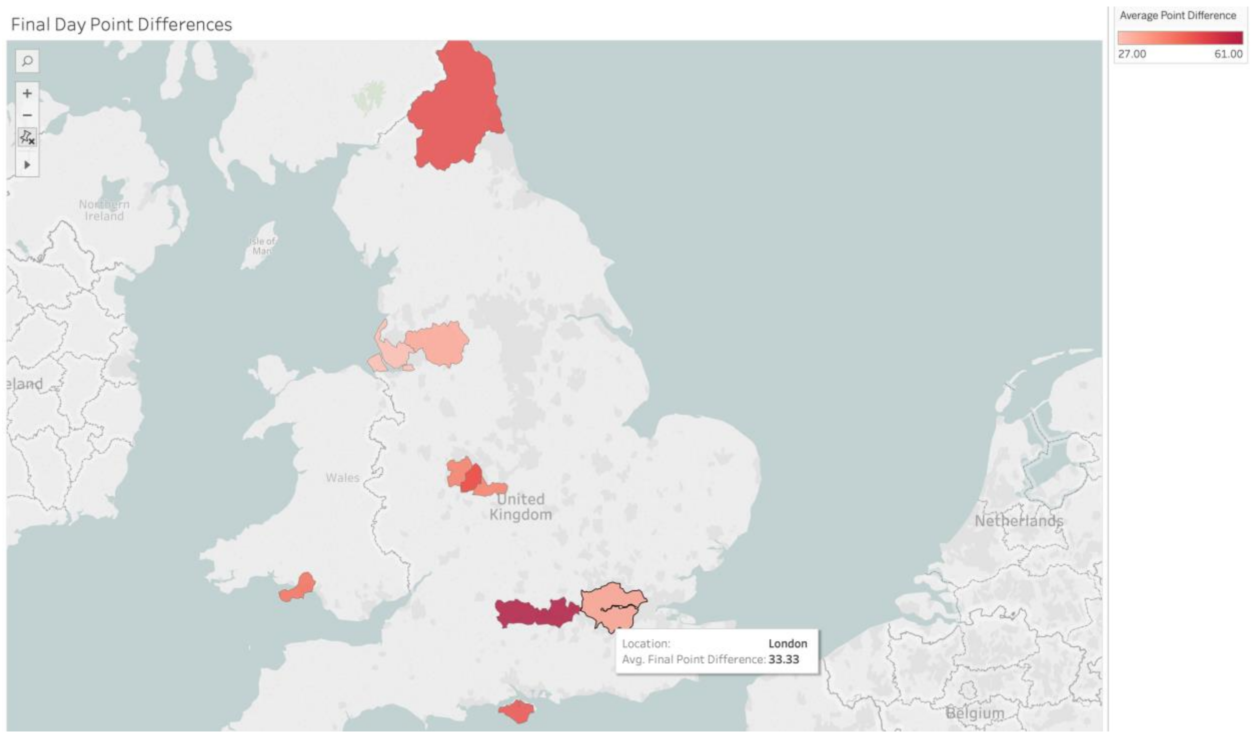
Note: In order to make it easier to navigate and see the information while hovering over specific teams, I added a random number between (-1, 1) to each first half goal on the plot. This is not reflected in the hover text, however.

Bubble map:



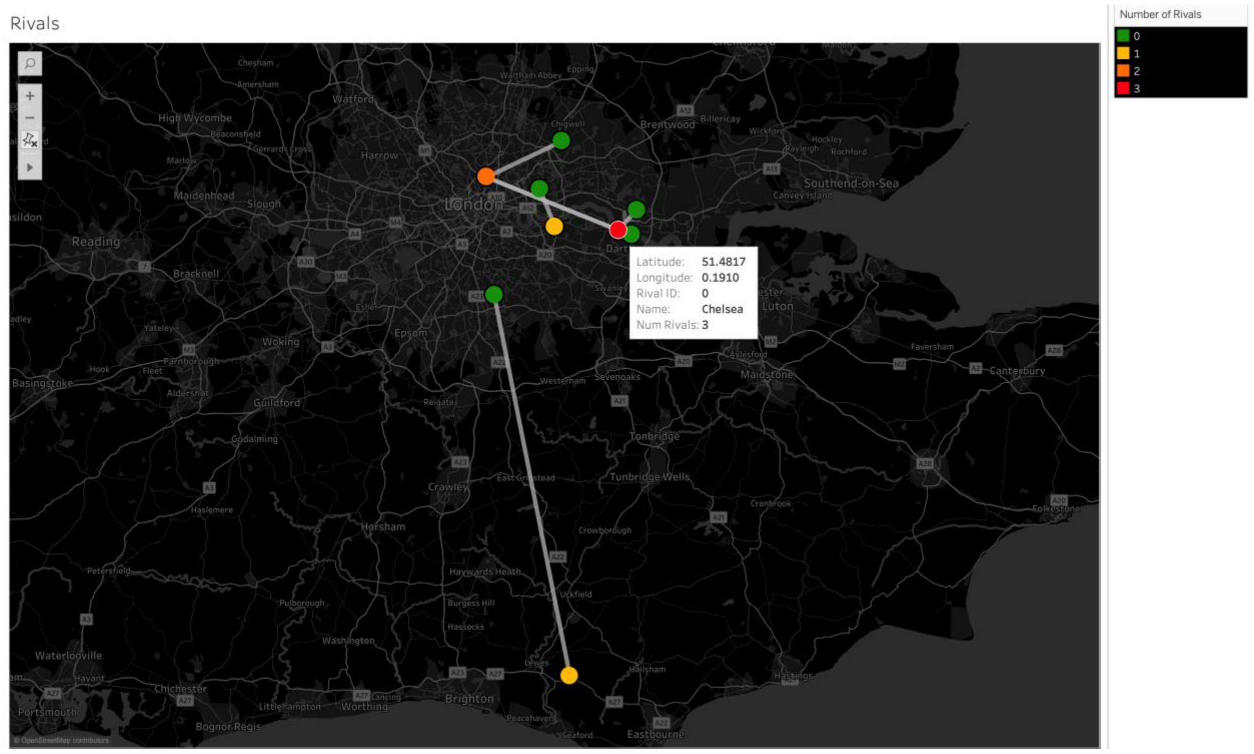
Every year, the English Premier League (PL) crowns one club as champion. The team that accumulates the most points from 38 games (Win = 3 points, Draw = 1 point, Loss = 0 points) gets the only opportunity of the year to raise the coveted PL trophy. In the situation that two teams are level on points, the team with a higher season-long goal differential (goals scored - goals conceded) comes out on top and the list of subsequent tiebreakers follows. The choropleth plot in the file above displays rings sized to the total number of PL titles won in the club's history and the color corresponds to the average points earned per year for the past 10 years, dark purple being low and dark pink being high. The colors are divided into segments from (20-35, 35-50, 50-65, 65-80) to make the distinction easier among teams close to each other. Additionally, the colors I chose represent the official PL colors, which can be seen on their webpage premier league. For additional exploration, I've made it possible for the viewer to isolate rings according to location or by team name. I gave this visualization the name "Premier League Title Richter Scale" because it reminds me of common earthquake severity visualizations. As a Manchester United (listed as Man United) fan, I am very happy to see that our ring is the biggest (20 titles) and dark pink (76 points)!

Choropleth map:



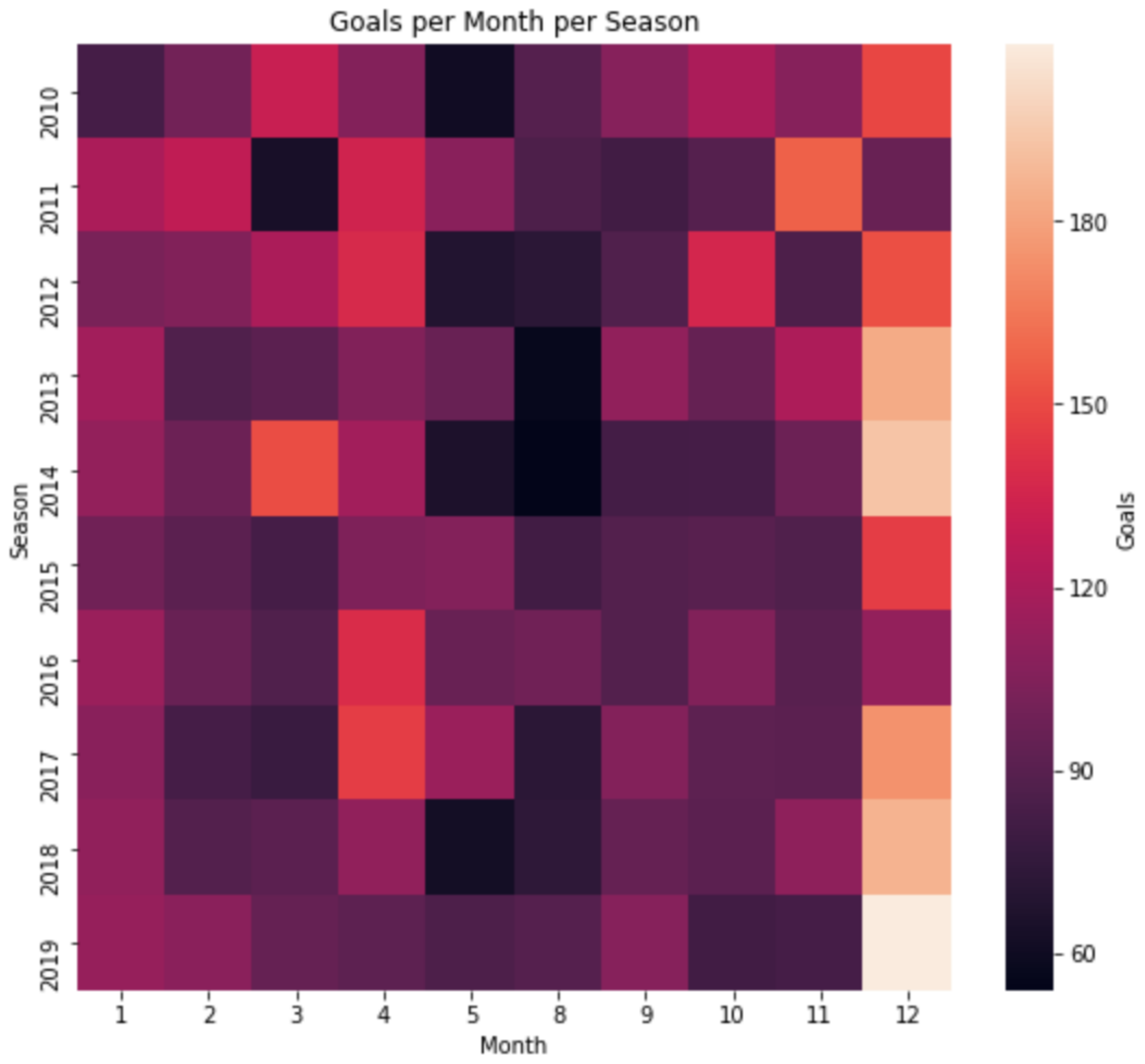
In this map, we are able to see the average difference in points per sub-region compared to Manchester United, who won the league with 89 points at the end of the 2012/13 season. For example, the average difference for the five teams in London was 33 points at the end of the season. This is mainly because Queens Park Rangers (QPR) was only able to earn 25 points. The darkest red region next to London is Berkshire, home of Reading, was 61 points behind Manchester United at the end of season. This is a testament to the incredible year Manchester United had in such a competitive league. The closest team was Chelsea, in London, at 75 points.

Connection map:



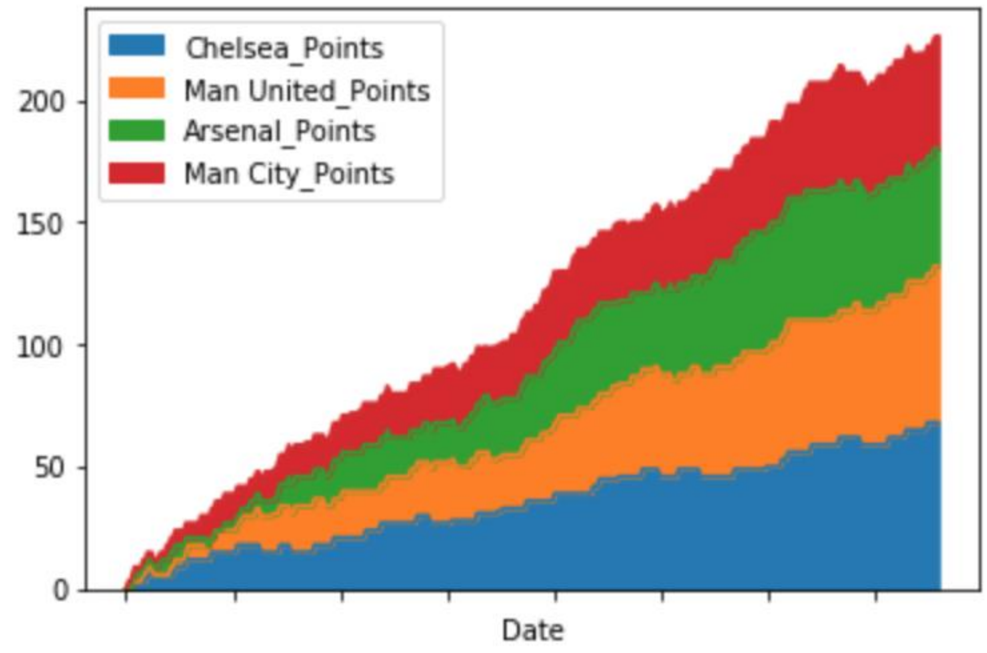
Here we see a zoomed in map of London, each dot representing the coordinates of the stadium of each team London based team. The connections represent where at least one of the two teams consider the other a rival. What is interesting about this map is that the colors correspond to how many other teams consider that team to be their main rival. So, for example, Chelsea is considered to be the main rival of three other teams. But, if we look at the teams Chelsea is connected to, we see two green dots (no teams consider as rivals) and one orange dot (two teams consider as rivals). So, we can deduce that the two green dots feel that Chelsea is their main rival, but Chelsea does not feel the same. In fact, we find that there is an interesting dynamic in London. The only other connection Chelsea has is Arsenal, which has two teams that consider them rivals, one of which must be Chelsea now. But Arsenal is also connected to a green dot that happens to be Tottenham. Thus, Tottenham must consider Arsenal to be their rival, but Arsenal claims their main rival is Chelsea since that is the only other connection. By applying this logic to the rest of the connections, we see that the Chelsea-Arsenal rivalry is the only mutual rivalry in London. However, this should not take away that there is always fierce competition among the London teams and is always an entertaining game.

Heat map:



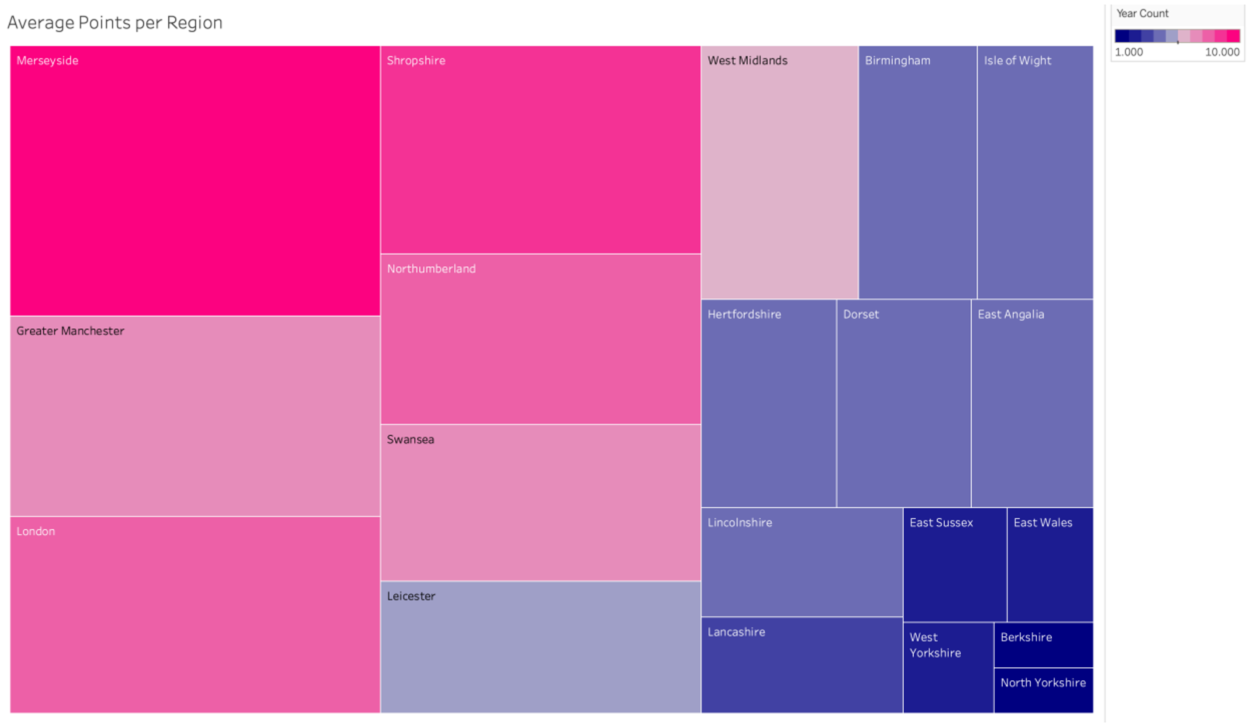
The heatmap above shows the relationship of when most goals are scored during each season. Clearly, we see that most goals are scored in December across all seasons. This could be because there is also a spike in the number of games around during this time of year as a part of the “40 Festive Fixtures” in 14 days. During these two weeks, NBC Sports covers nearly daily matches between PL teams and talk show discussions to celebrate the holidays. On the other hand, it seems that August has the least amount of goals across seasons. The new season begins in August, so this is the result of teams trying to settle into the new season. There also seems to be a slight decline in goals in May, which I find interesting as this is the end of the season, and teams are usually fighting to either avoid relegation and stay in the PL or to become PL champions.

Stacked Area:



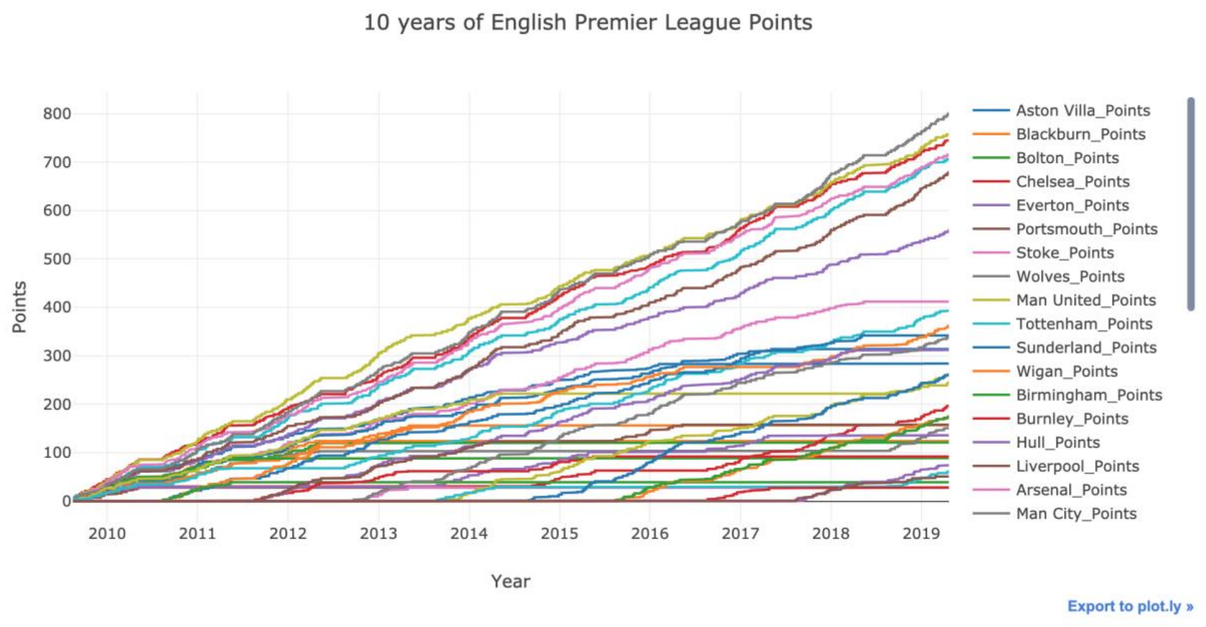
This stacked area map shows the season long performance of the teams that finished in the top 4 spots in the 2009/10 season. The x-axis refers to the dates of each game beginning in August 2009 and ending in May 2010. One adjustment I made while making this graph that would not be normally done is to assign -3 points for losing a game, which would normally be 0 points. I did this to better illustrate team form during certain stretches of time. Positive slopes indicate either a win, +3 points, or a draw, +1 point, and negative slopes indicate a loss, -3 points. Flat spans indicate a break in games and no points earned. As we can see, Chelsea has the most area throughout the season and at the end, thus we know that Chelsea won the PL in 2010.

Treemapping:



Here we see the relationship between the average points earned by teams in their respective sub-regions as well as average number of years the teams in each region have competed in the PL over the past ten years. We can see that Merceyside not only has the highest average points per season, but also has a perfect ten years of PL participation. This is due to the two teams in Merceyside, Liverpool and Everton. Both of these teams are top quality teams and are generally among the top six teams. Right behind Merceyside, we see Greater Manchester that is home to Manchester United, Manchester City, Bolton Wanderers, and Wigan Athletic. As we know, Manchester United is the best team in the PL and is often challenged by Manchester City in recent years. However, Bolton and Wigan are among the less talented teams and are in and out of the PL, explaining the duller shade of pink and smaller area.

Storyline:



Over the past ten years of English Premier League (PL) football/soccer, we have seen teams rise to the peak of performance or tumble down the face of the mountain. The plot above is a time series of cumulative points earned over the past ten years. Points are earned for each game played, 3 points for a win, 1 point for a draw, and 0 points for a loss over the course of 38 games per season.

One example of teams rising and falling is the rivalry between both Manchester clubs, Manchester United and Manchester City (listed as Man United and Man City. From 2011-2014, Man United was dominating not only Man City, but the entire league having over 40 more cumulative points earned at sometimes. However, we see a shift in power in Manchester after 2014 as Man City begins to close the point gap. Then at the end of the 2017 season, we see Man City definitively take the point lead by earning a total of 100 points in one season, with Man United coming in at second that season with 81 points. We can draw from this that the power in Manchester has moved from the Man United Red Devils to the Man City Moonbeams.

Another noteworthy feature of the plot is the visualization of teams being promoted or relegated to and from the PL. Each year, the top three teams in the English Football Championship League are promoted the PL and the bottom three teams of the PL are relegated to the English Football Championship League. We can see new lines rise up along the x-axis. These represent teams being promoted into the PL for their chance at competing at the highest level in Great Britain. On the other hand, we can see teams that have been relegated from the PL indicated by long flat lines. These flat lines occur because the relegated teams are no longer accumulating points in the PL until they are promoted again.

The last interesting visible feature is the clear gap between the top six teams in the PL and the rest of the teams that have competed in the PL over the past ten years. These teams are Man United, Man City, Liverpool, Chelsea, Arsenal, and Tottenham. Everton is caught in the middle of being considered one of the elite teams but have been held back from achieving greatness. The rest of the teams are in a battle near the bottom to survive and keep their spots in the PL.

Conclusion:

In summary, there are many interesting details about the English Premier League that I learned by using the right visualization to portray stories in the data. Some of these took me by surprise at first, however I was able to make sense of the unexpected details by applying my knowledge of soccer and the PL. For example, the heatmap showing that December was a hotspot for goals surprised me at first, but then I remembered that 40 Festive Fixtures happens at this time which explained the spike in goals. One take away that I got from creating these visualizations is that the PL can often be unpredictable. Although there are a handful of teams that are consistently at the top of the league, there are a few teams that pop up occasionally to surprise everyone. I also found it interesting that there are regions in Great Britain that seem to produce better teams, like Great Manchester, Merceyside, and London. I am very passionate about the PL and I hope I was able to convey some of its appeal through this report.

<u>**Appendix (Code):**</u>

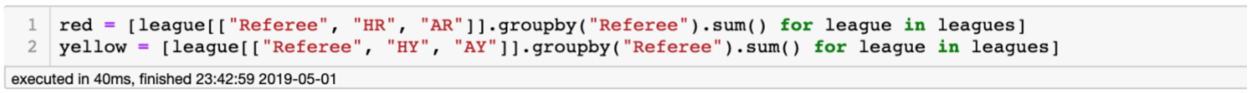
Histogram:

Tableau calculated field: COUNT([Home Team]) + COUNT([Away Team])

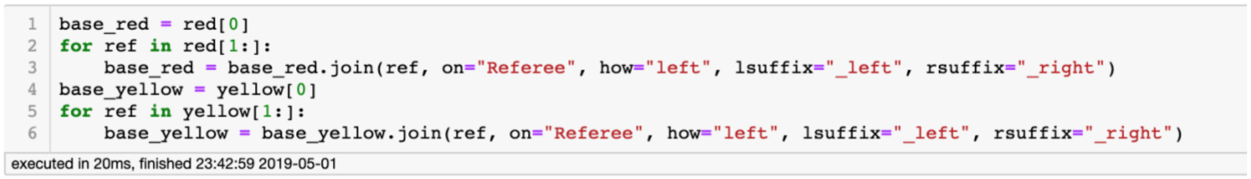
Plotted in Tableau.

Bar plot:

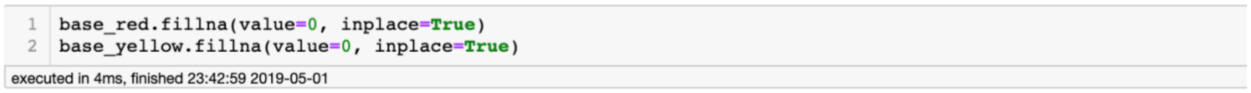
Aggregate and prepare data



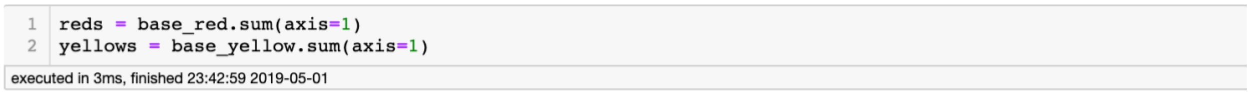
```python
red = [league[["Referee", "HR", "AR"]].groupby("Referee").sum() for league in leagues]
yellow = [league[["Referee", "HY", "AY"]].groupby("Referee").sum() for league in leagues]
```
executed in 40ms, finished 23:42:59 2019-05-01



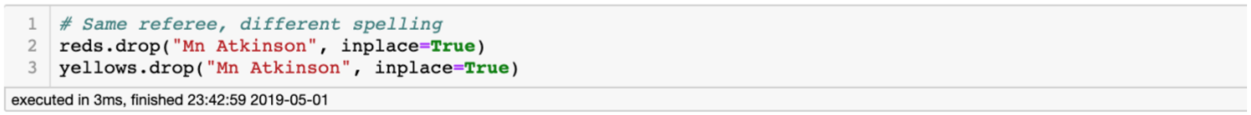
```python
base_red = red[0]
for ref in red[1:]:
    base_red = base_red.join(ref, on="Referee", how="left", lsuffix="_left", rsuffix="_right")
base_yellow = yellow[0]
for ref in yellow[1:]:
    base_yellow = base_yellow.join(ref, on="Referee", how="left", lsuffix="_left", rsuffix="_right")
```
executed in 20ms, finished 23:42:59 2019-05-01



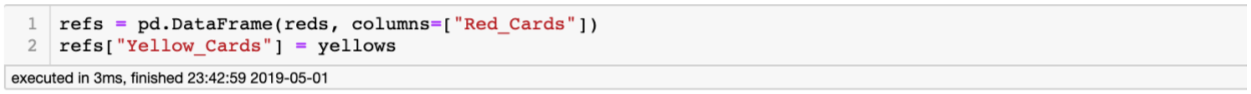
```python
base_red.fillna(value=0, inplace=True)
base_yellow.fillna(value=0, inplace=True)
```
executed in 4ms, finished 23:42:59 2019-05-01



```python
reds = base_red.sum(axis=1)
yellows = base_yellow.sum(axis=1)
```
executed in 3ms, finished 23:42:59 2019-05-01

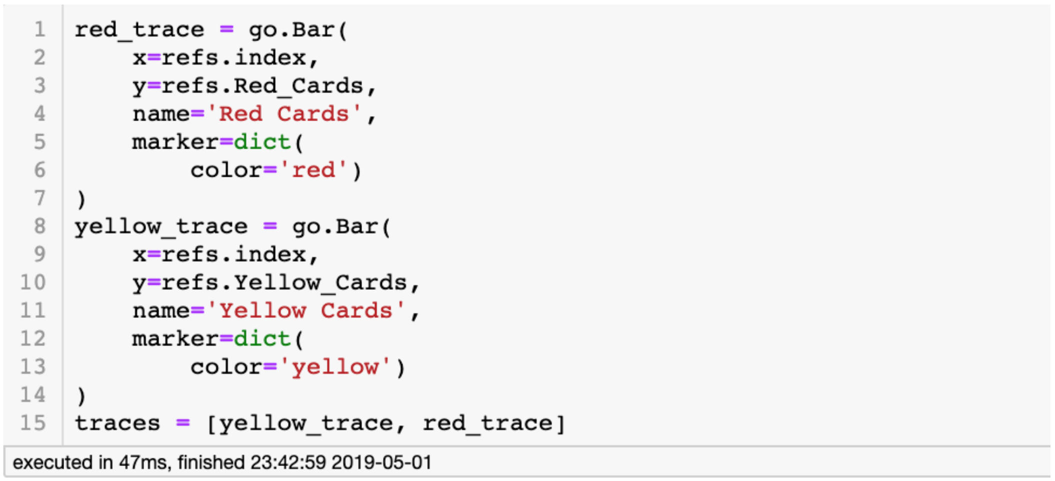


```python
# Same referee, different spelling
reds.drop("Mn Atkinson", inplace=True)
yellows.drop("Mn Atkinson", inplace=True)
```
executed in 3ms, finished 23:42:59 2019-05-01

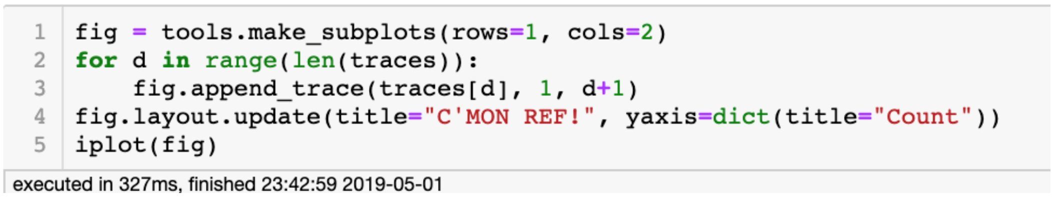


```python
refs = pd.DataFrame(reds, columns=["Red_Cards"])
refs["Yellow_Cards"] = yellows
```
executed in 3ms, finished 23:42:59 2019-05-01

Plot data



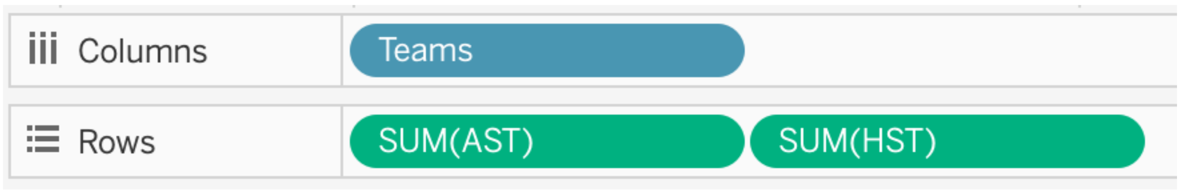
```python
red_trace = go.Bar(
    x=refs.index,
    y=refs.Red_Cards,
    name='Red Cards',
    marker=dict(
        color='red')
)
yellow_trace = go.Bar(
    x=refs.index,
    y=refs.Yellow_Cards,
    name='Yellow Cards',
    marker=dict(
        color='yellow')
)
traces = [yellow_trace, red_trace]
```
executed in 47ms, finished 23:42:59 2019-05-01



```python
fig = tools.make_subplots(rows=1, cols=2)
for d in range(len(traces)):
    fig.append_trace(traces[d], 1, d+1)
fig.layout.update(title="C'MON REF!", yaxis=dict(title="Count"))
iplot(fig)
```
executed in 327ms, finished 23:42:59 2019-05-01

Boxplot:

Tableau rows and columns:



AST = Away shots | HST = Home shots

Plotted in Tableau.

Scatterplot:

Aggregate and prepare data

```python
league_18 = pd.read_csv("data/season-1718_csv.csv")
```
executed in 148ms, finished 00:31:23 2019-05-16

```python
home_sums = league_18[["HomeTeam", "HTHG", "FTHG"]].groupby("HomeTeam").sum()
away_sums = league_18[["AwayTeam", "HTAG", "FTAG"]].groupby("AwayTeam").sum()
```
executed in 11ms, finished 00:31:23 2019-05-16

```python
home_sums.index.name = "Team"
away_sums.index.name = "Team"
home_sums.columns = ["HalfTime", "FullTime"]
away_sums.columns = ["HalfTime", "FullTime"]
```
executed in 4ms, finished 00:31:23 2019-05-16

```python
full_sums = home_sums + away_sums
full_sums["SecondHalf"] = full_sums.loc[:, "FullTime"] - full_sums.loc[:, "HalfTime"]
```
executed in 7ms, finished 00:31:24 2019-05-16

```python
full_sums.drop(columns="FullTime", inplace=True)
```
executed in 4ms, finished 00:31:24 2019-05-16

```python
full_sums["Ratio"] = full_sums["SecondHalf"] / (full_sums["HalfTime"] + full_sums["SecondHalf"])
```
executed in 5ms, finished 00:31:26 2019-05-16

```python
full_sums.reset_index(inplace=True)
```
executed in 4ms, finished 00:31:26 2019-05-16

```python
def wins(df):
    teams = df.HomeTeam.unique()
    team_wins = dict()
    for team in teams:
        wins = 0
        for row in range(df.shape[0]):
            if df.HomeTeam.iloc[row] == team and df.FTHG.iloc[row] > df.FTAG.iloc[row]:
                wins += 1
            elif df.AwayTeam.iloc[row] == team and df.FTAG.iloc[row] > df.FTHG.iloc[row]:
                wins += 1
        team_wins[team] = wins
    return team_wins
```
executed in 5ms, finished 00:31:26 2019-05-16

```python
team_wins = wins(league_18)
```
executed in 210ms, finished 00:31:28 2019-05-16

```python
full_sums["Wins"] = full_sums.Team.replace(team_wins)
```

Plot data

```python
hover_text = [f"Team: {row[0]}<br>Wins: {row[1]}<br>First Half: {row[2]}<br>Second Half: {row[3]}"
              for row in full_sums[["Team", "Wins", "HalfTime", "SecondHalf"]].values]
```
executed in 6ms, finished 00:31:28 2019-05-16

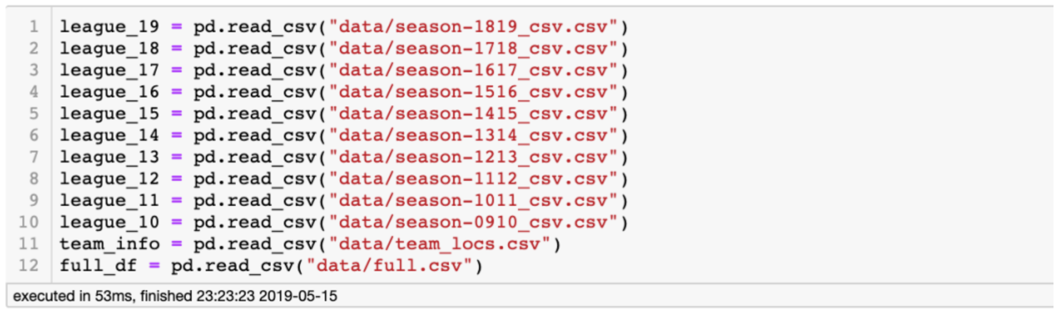
```python
trace = [go.Scatter(
    x=full_sums["HalfTime"]+np.random.choice(a=np.linspace(-1, 1, 20), replace=False, size=20),
    y=full_sums["SecondHalf"].values,
    text=hover_text,
    hoverinfo="text",
    mode="markers",
    marker=dict(
        size=full_sums["Wins"]**1.2,
        color="Green"))]
```
executed in 103ms, finished 00:31:28 2019-05-16

```python
layout = go.Layout(
    title="Second Half Goals vs Wins for 2018 Season",
    xaxis=dict(title="First Half Goals"),
    yaxis=dict(title="Second Half Goals"))
```
executed in 491ms, finished 00:31:31 2019-05-16

```python
iplot(dict(data=trace, layout=layout))
```

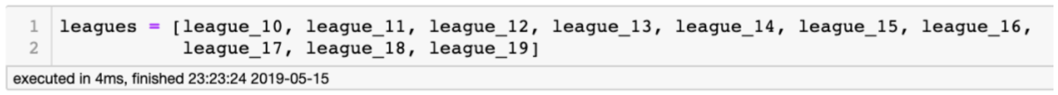
Bubble map:

Aggregate and prepare data



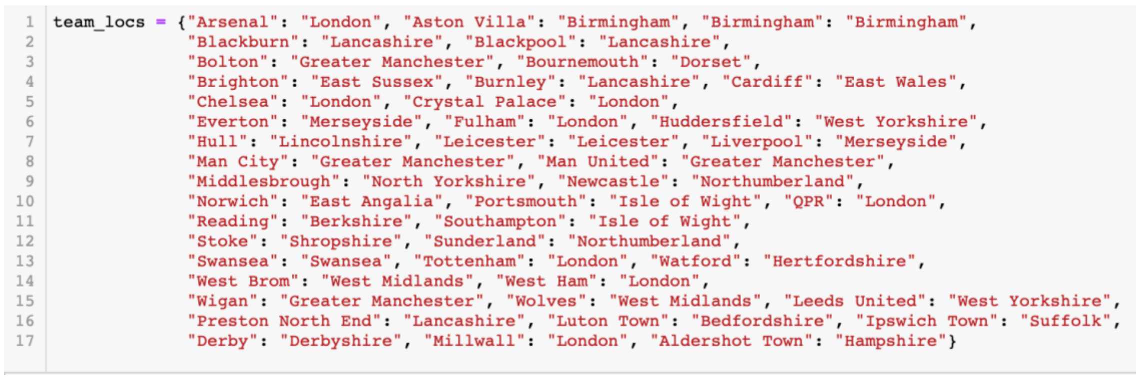
```python
league_19 = pd.read_csv("data/season-1819_csv.csv")
league_18 = pd.read_csv("data/season-1718_csv.csv")
league_17 = pd.read_csv("data/season-1617_csv.csv")
league_16 = pd.read_csv("data/season-1516_csv.csv")
league_15 = pd.read_csv("data/season-1415_csv.csv")
league_14 = pd.read_csv("data/season-1314_csv.csv")
league_13 = pd.read_csv("data/season-1213_csv.csv")
league_12 = pd.read_csv("data/season-1112_csv.csv")
league_11 = pd.read_csv("data/season-1011_csv.csv")
league_10 = pd.read_csv("data/season-0910_csv.csv")
team_info = pd.read_csv("data/team_locs.csv")
full_df = pd.read_csv("data/full.csv")
```

executed in 53ms, finished 23:23:23 2019-05-15

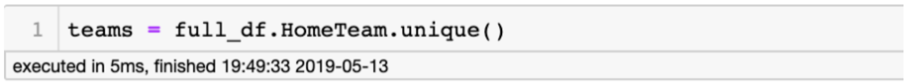


```python
leagues = [league_10, league_11, league_12, league_13, league_14, league_15, league_16,
           league_17, league_18, league_19]
```

executed in 4ms, finished 23:23:24 2019-05-15

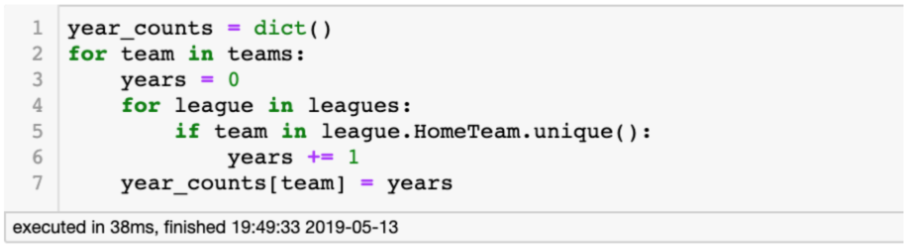


```python
team_locs = {"Arsenal": "London", "Aston Villa": "Birmingham", "Birmingham": "Birmingham",
             "Blackburn": "Lancashire", "Blackpool": "Lancashire",
             "Bolton": "Greater Manchester", "Bournemouth": "Dorset",
             "Brighton": "East Sussex", "Burnley": "Lancashire", "Cardiff": "East Wales",
             "Chelsea": "London", "Crystal Palace": "London",
             "Everton": "Merseyside", "Fulham": "London", "Huddersfield": "West Yorkshire",
             "Hull": "Lincolnshire", "Leicester": "Leicester", "Liverpool": "Merseyside",
             "Man City": "Greater Manchester", "Man United": "Greater Manchester",
             "Middlesbrough": "North Yorkshire", "Newcastle": "Northumberland",
             "Norwich": "East Angalia", "Portsmouth": "Isle of Wight", "QPR": "London",
             "Reading": "Berkshire", "Southampton": "Isle of Wight",
             "Stoke": "Shropshire", "Sunderland": "Northumberland",
             "Swansea": "Swansea", "Tottenham": "London", "Watford": "Hertfordshire",
             "West Brom": "West Midlands", "West Ham": "London",
             "Wigan": "Greater Manchester", "Wolves": "West Midlands", "Leeds United": "West Yorkshire",
             "Preston North End": "Lancashire", "Luton Town": "Bedfordshire", "Ipswich Town": "Suffolk",
             "Derby": "Derbyshire", "Millwall": "London", "Aldershot Town": "Hampshire"}
```



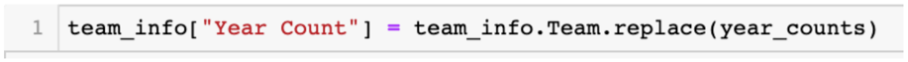
```python
teams = full_df.HomeTeam.unique()
```

executed in 5ms, finished 19:49:33 2019-05-13

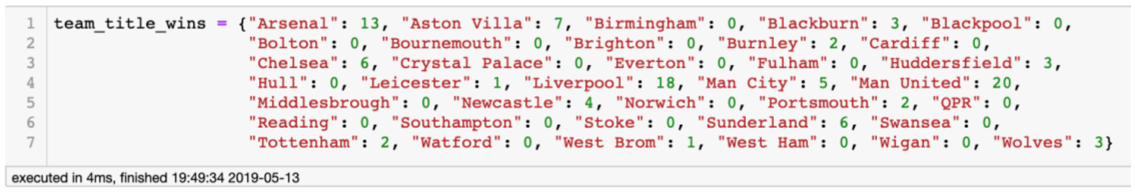


```python
year_counts = dict()
for team in teams:
    years = 0
    for league in leagues:
        if team in league.HomeTeam.unique():
            years += 1
    year_counts[team] = years
```

executed in 38ms, finished 19:49:33 2019-05-13

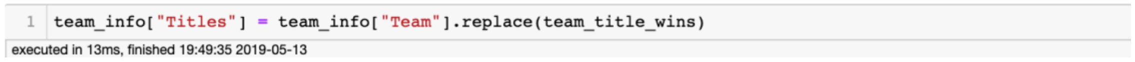


```python
team_info["Year Count"] = team_info.Team.replace(year_counts)
```



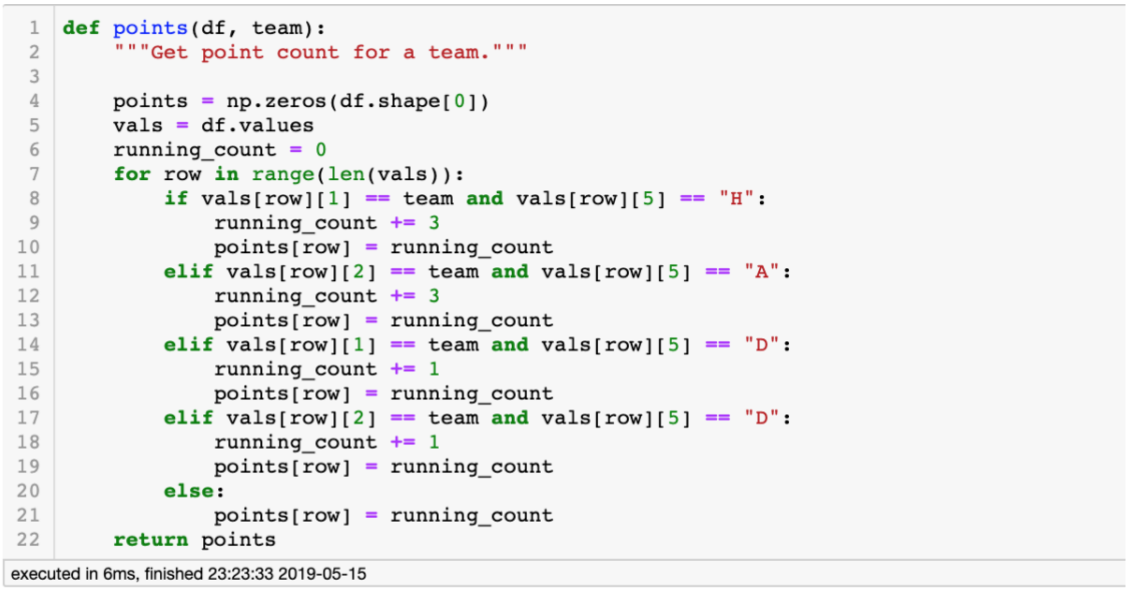
```python
team_title_wins = {"Arsenal": 13, "Aston Villa": 7, "Birmingham": 0, "Blackburn": 3, "Blackpool": 0,
                   "Bolton": 0, "Bournemouth": 0, "Brighton": 0, "Burnley": 2, "Cardiff": 0,
                   "Chelsea": 6, "Crystal Palace": 0, "Everton": 0, "Fulham": 0, "Huddersfield": 3,
                   "Hull": 0, "Leicester": 1, "Liverpool": 18, "Man City": 5, "Man United": 20,
                   "Middlesbrough": 0, "Newcastle": 4, "Norwich": 0, "Portsmouth": 2, "QPR": 0,
                   "Reading": 0, "Southampton": 0, "Stoke": 0, "Sunderland": 6, "Swansea": 0,
                   "Tottenham": 2, "Watford": 0, "West Brom": 1, "West Ham": 0, "Wigan": 0, "Wolves": 3}
```

executed in 4ms, finished 19:49:34 2019-05-13



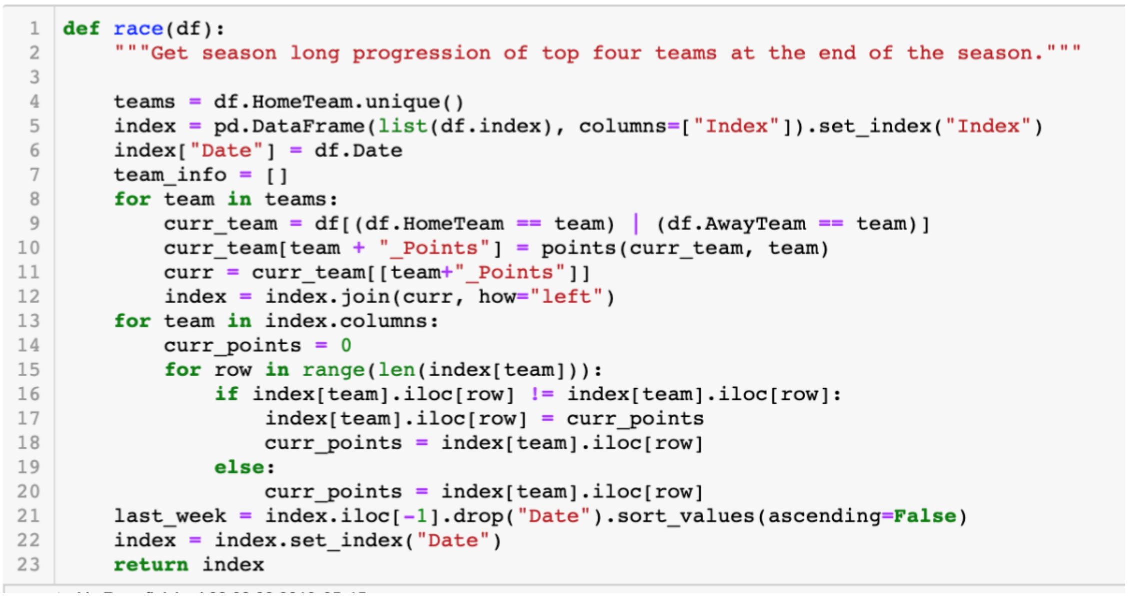
```python
team_info["Titles"] = team_info["Team"].replace(team_title_wins)
```

executed in 13ms, finished 19:49:35 2019-05-13

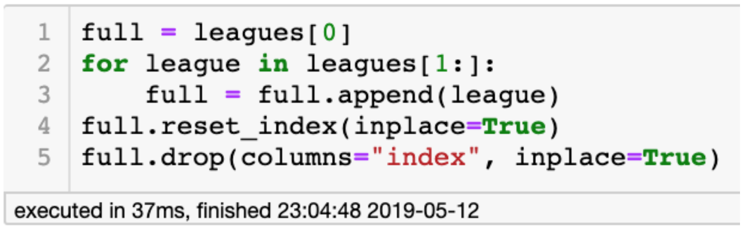


```python
def points(df, team):
    """Get point count for a team."""

    points = np.zeros(df.shape[0])
    vals = df.values
    running_count = 0
    for row in range(len(vals)):
        if vals[row][1] == team and vals[row][5] == "H":
            running_count += 3
            points[row] = running_count
        elif vals[row][2] == team and vals[row][5] == "A":
            running_count += 3
            points[row] = running_count
        elif vals[row][1] == team and vals[row][5] == "D":
            running_count += 1
            points[row] = running_count
        elif vals[row][2] == team and vals[row][5] == "D":
            running_count += 1
            points[row] = running_count
        else:
            points[row] = running_count
    return points
```

executed in 6ms, finished 23:23:33 2019-05-15

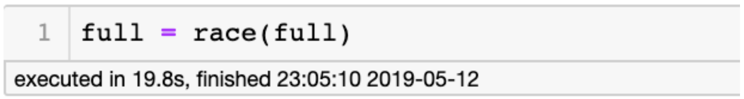


```python
def race(df):
    """Get season long progression of top four teams at the end of the season."""

    teams = df.HomeTeam.unique()
    index = pd.DataFrame(list(df.index), columns=["Index"]).set_index("Index")
    index["Date"] = df.Date
    team_info = []
    for team in teams:
        curr_team = df[(df.HomeTeam == team) | (df.AwayTeam == team)]
        curr_team[team + "_Points"] = points(curr_team, team)
        curr = curr_team[[team+"_Points"]]
        index = index.join(curr, how="left")
    for team in index.columns:
        curr_points = 0
        for row in range(len(index[team])):
            if index[team].iloc[row] != index[team].iloc[row]:
                index[team].iloc[row] = curr_points
                curr_points = index[team].iloc[row]
            else:
                curr_points = index[team].iloc[row]
    last_week = index.iloc[-1].drop("Date").sort_values(ascending=False)
    index = index.set_index("Date")
    return index
```



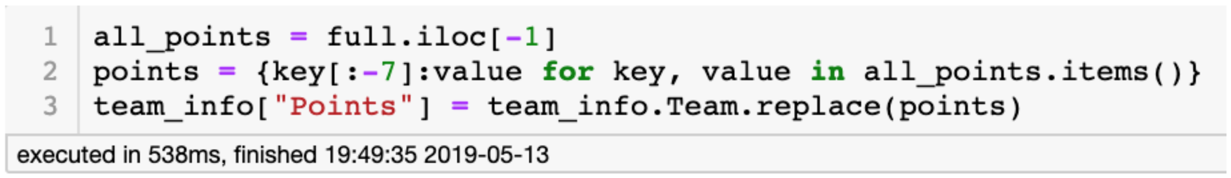
```python
full = leagues[0]
for league in leagues[1:]:
    full = full.append(league)
full.reset_index(inplace=True)
full.drop(columns="index", inplace=True)
```

executed in 37ms, finished 23:04:48 2019-05-12



```python
full = race(full)
```

executed in 19.8s, finished 23:05:10 2019-05-12



```python
all_points = full.iloc[-1]
points = {key[:-7]:value for key, value in all_points.items()}
team_info["Points"] = team_info.Team.replace(points)
```

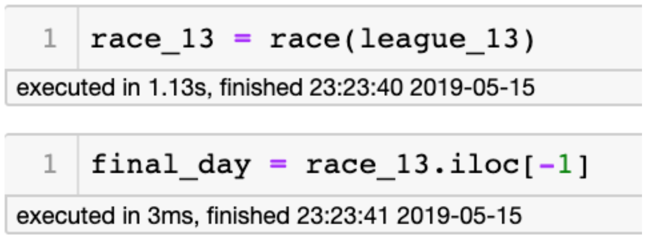
```
executed in 538ms, finished 19:49:35 2019-05-13
```

Tableau Calculated Field: [Points] / [Year Count]

Plotted in Tableau.

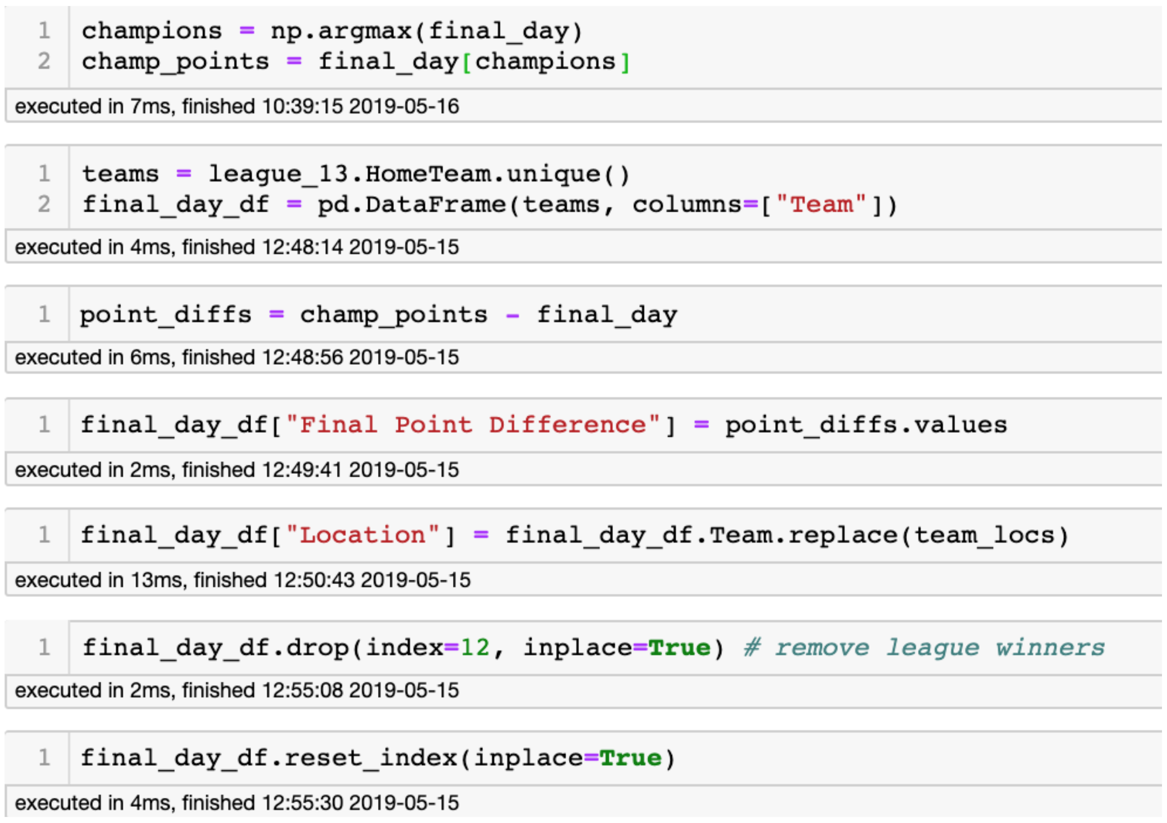
Choropleth map:

Aggregate and prepare data



```
race_13 = race(league_13)
```
```
executed in 1.13s, finished 23:23:40 2019-05-15
```

```
final_day = race_13.iloc[-1]
```
```
executed in 3ms, finished 23:23:41 2019-05-15
```



```
champions = np.argmax(final_day)
champ_points = final_day[champions]
```
```
executed in 7ms, finished 10:39:15 2019-05-16
```

```
teams = league_13.HomeTeam.unique()
final_day_df = pd.DataFrame(teams, columns=["Team"])
```
```
executed in 4ms, finished 12:48:14 2019-05-15
```

```
point_diffs = champ_points - final_day
```
```
executed in 6ms, finished 12:48:56 2019-05-15
```

```
final_day_df["Final Point Difference"] = point_diffs.values
```
```
executed in 2ms, finished 12:49:41 2019-05-15
```

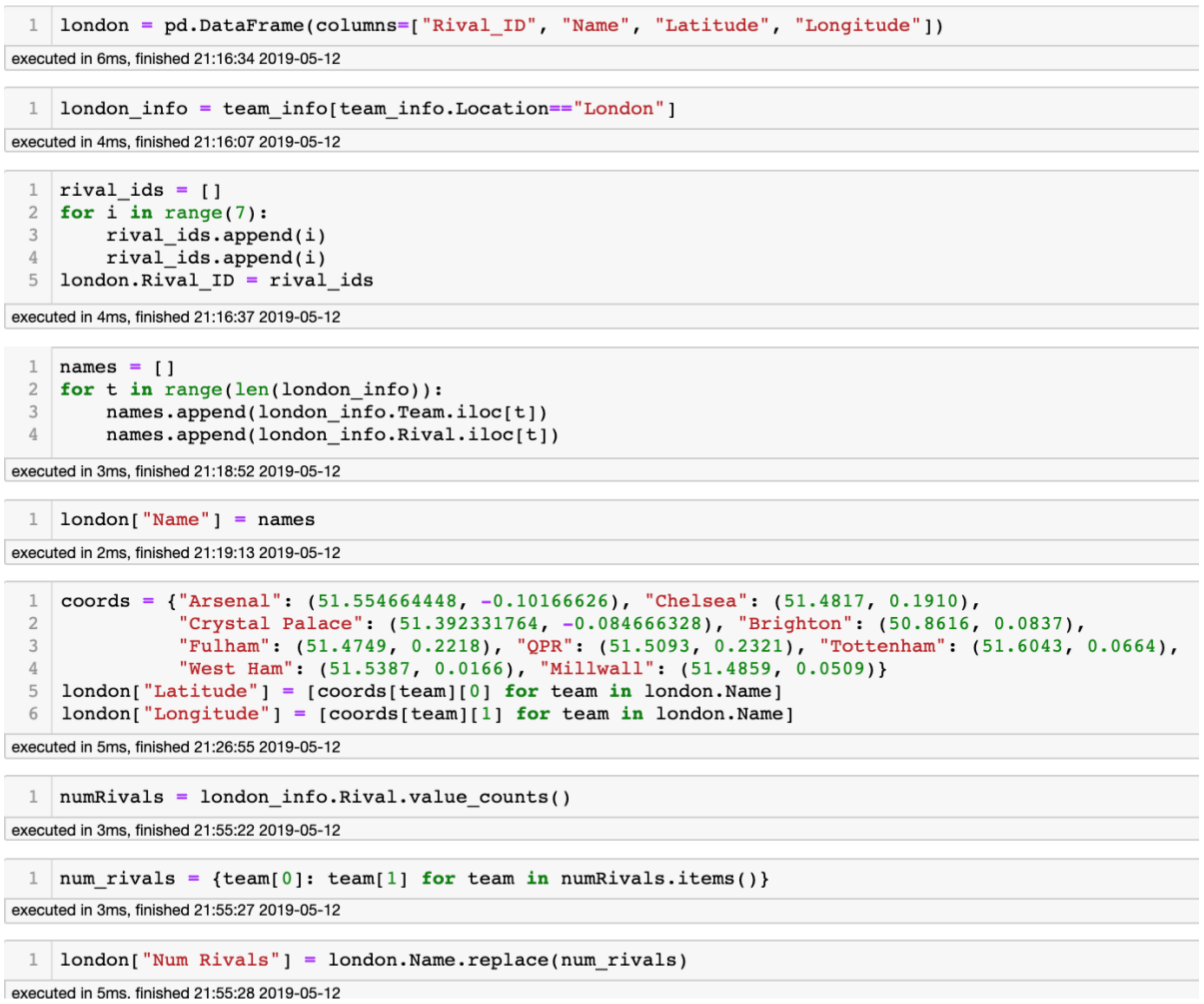
```
final_day_df["Location"] = final_day_df.Team.replace(team_locs)
```
```
executed in 13ms, finished 12:50:43 2019-05-15
```

```
final_day_df.drop(index=12, inplace=True) # remove league winners
```
```
executed in 2ms, finished 12:55:08 2019-05-15
```

```
final_day_df.reset_index(inplace=True)
```
```
executed in 4ms, finished 12:55:30 2019-05-15
```

Plotted in Tableau.

Connection map:



```python
london = pd.DataFrame(columns=["Rival_ID", "Name", "Latitude", "Longitude"])
```

executed in 6ms, finished 21:16:34 2019-05-12

```python
london_info = team_info[team_info.Location=="London"]
```

executed in 4ms, finished 21:16:07 2019-05-12

```python
rival_ids = []
for i in range(7):
    rival_ids.append(i)
    rival_ids.append(i)
london.Rival_ID = rival_ids
```

executed in 4ms, finished 21:16:37 2019-05-12

```python
names = []
for t in range(len(london_info)):
    names.append(london_info.Team.iloc[t])
    names.append(london_info.Rival.iloc[t])
```

executed in 3ms, finished 21:18:52 2019-05-12

```python
london["Name"] = names
```

executed in 2ms, finished 21:19:13 2019-05-12

```python
coords = {"Arsenal": (51.554664448, -0.10166626), "Chelsea": (51.4817, 0.1910),
          "Crystal Palace": (51.392331764, -0.084666328), "Brighton": (50.8616, 0.0837),
          "Fulham": (51.4749, 0.2218), "QPR": (51.5093, 0.2321), "Tottenham": (51.6043, 0.0664),
          "West Ham": (51.5387, 0.0166), "Millwall": (51.4859, 0.0509)}
london["Latitude"] = [coords[team][0] for team in london.Name]
london["Longitude"] = [coords[team][1] for team in london.Name]
```

executed in 5ms, finished 21:26:55 2019-05-12

```python
numRivals = london_info.Rival.value_counts()
```

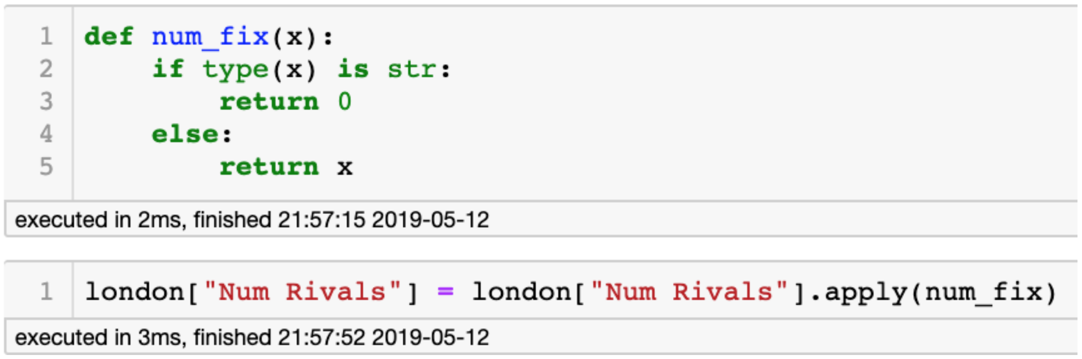
executed in 3ms, finished 21:55:22 2019-05-12

```python
num_rivals = {team[0]: team[1] for team in numRivals.items()}
```

executed in 3ms, finished 21:55:27 2019-05-12

```python
london["Num Rivals"] = london.Name.replace(num_rivals)
```

executed in 5ms, finished 21:55:28 2019-05-12



```python
def num_fix(x):
    if type(x) is str:
        return 0
    else:
        return x
```

executed in 2ms, finished 21:57:15 2019-05-12

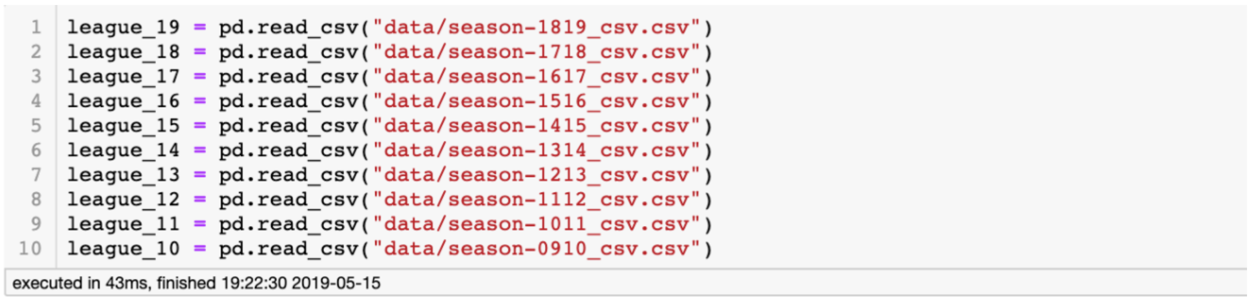
```python
london["Num Rivals"] = london["Num Rivals"].apply(num_fix)
```

executed in 3ms, finished 21:57:52 2019-05-12

Plotted in Tableau.

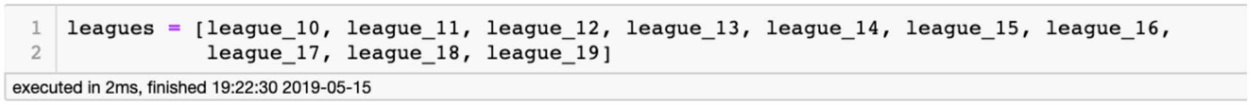
Heat map:

Aggregate and prepare data



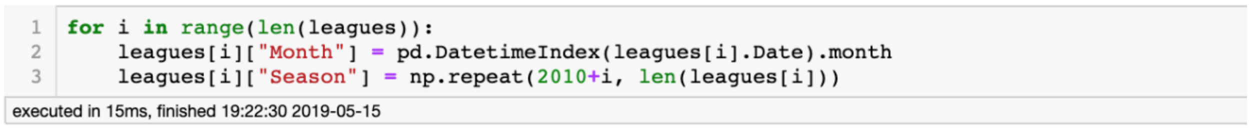
```
league_19 = pd.read_csv("data/season-1819_csv.csv")
league_18 = pd.read_csv("data/season-1718_csv.csv")
league_17 = pd.read_csv("data/season-1617_csv.csv")
league_16 = pd.read_csv("data/season-1516_csv.csv")
league_15 = pd.read_csv("data/season-1415_csv.csv")
league_14 = pd.read_csv("data/season-1314_csv.csv")
league_13 = pd.read_csv("data/season-1213_csv.csv")
league_12 = pd.read_csv("data/season-1112_csv.csv")
league_11 = pd.read_csv("data/season-1011_csv.csv")
league_10 = pd.read_csv("data/season-0910_csv.csv")
```

executed in 43ms, finished 19:22:30 2019-05-15



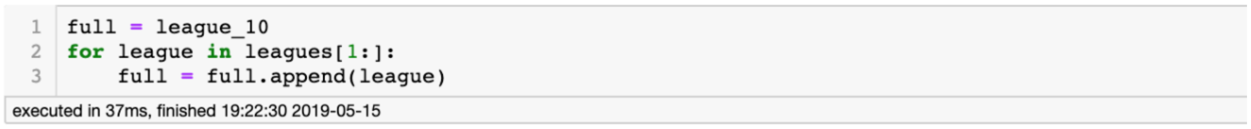
```
leagues = [league_10, league_11, league_12, league_13, league_14, league_15, league_16,
           league_17, league_18, league_19]
```

executed in 2ms, finished 19:22:30 2019-05-15



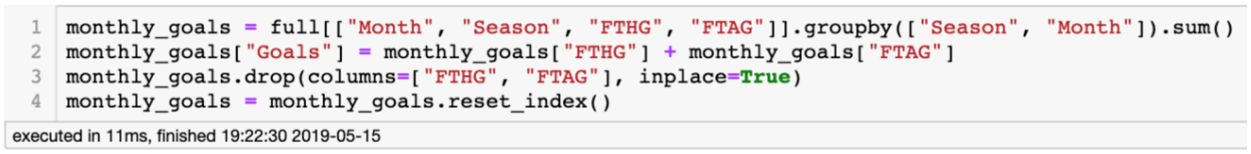
```
for i in range(len(leagues)):
    leagues[i]["Month"] = pd.DatetimeIndex(leagues[i].Date).month
    leagues[i]["Season"] = np.repeat(2010+i, len(leagues[i]))
```

executed in 15ms, finished 19:22:30 2019-05-15



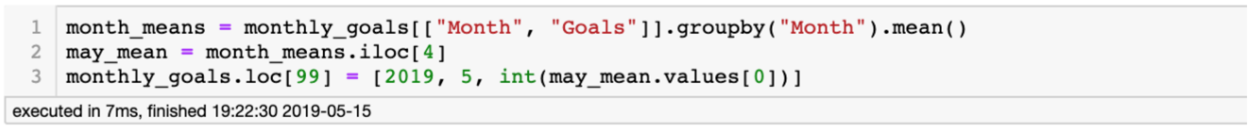
```
full = league_10
for league in leagues[1:]:
    full = full.append(league)
```

executed in 37ms, finished 19:22:30 2019-05-15



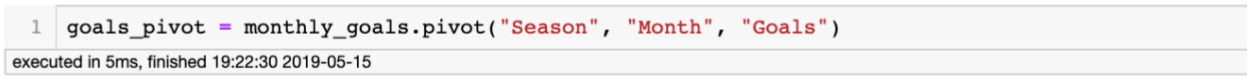
```
monthly_goals = full[["Month", "Season", "FTHG", "FTAG"]].groupby(["Season", "Month"]).sum()
monthly_goals["Goals"] = monthly_goals["FTHG"] + monthly_goals["FTAG"]
monthly_goals.drop(columns=["FTHG", "FTAG"], inplace=True)
monthly_goals = monthly_goals.reset_index()
```

executed in 11ms, finished 19:22:30 2019-05-15



```
month_means = monthly_goals[["Month", "Goals"]].groupby("Month").mean()
may_mean = month_means.iloc[4]
monthly_goals.loc[99] = [2019, 5, int(may_mean.values[0])]
```

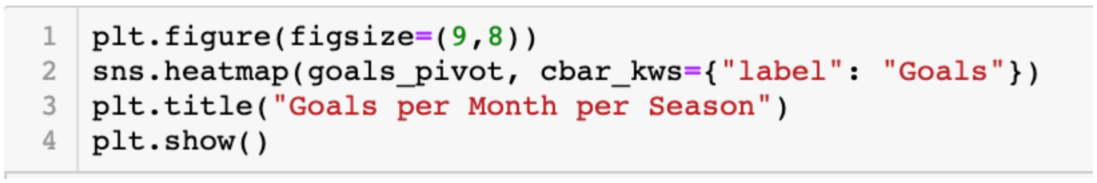
executed in 7ms, finished 19:22:30 2019-05-15



```
goals_pivot = monthly_goals.pivot("Season", "Month", "Goals")
```

executed in 5ms, finished 19:22:30 2019-05-15

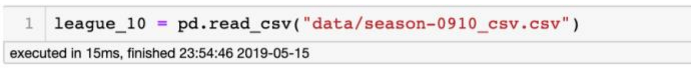
Plot data



```
plt.figure(figsize=(9,8))
sns.heatmap(goals_pivot, cbar_kws={"label": "Goals"})
plt.title("Goals per Month per Season")
plt.show()
```

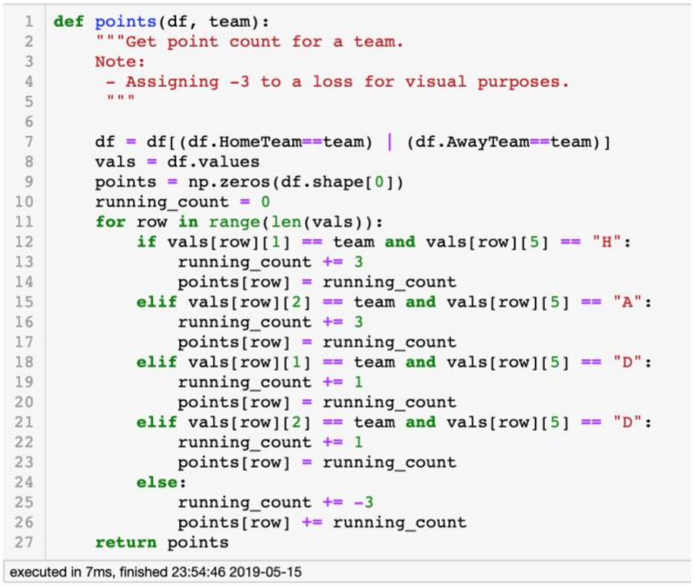
Stacked Area:

Aggregate and prepare data



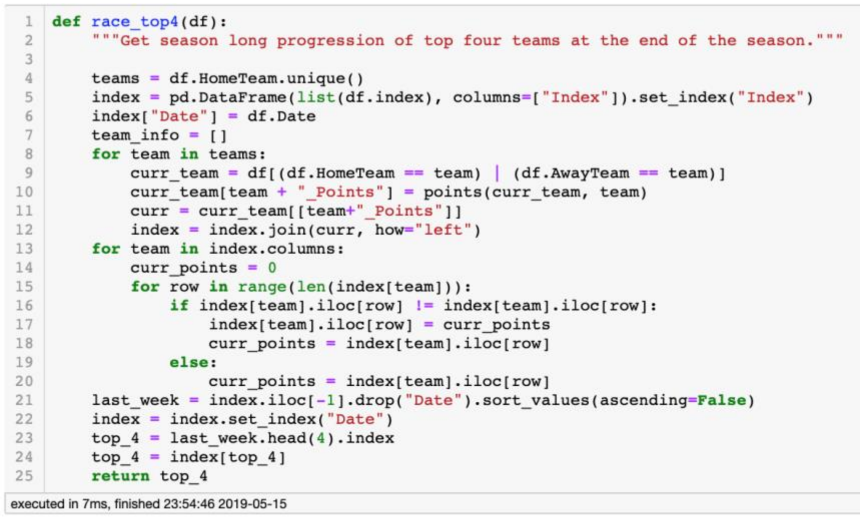
```python
league_10 = pd.read_csv("data/season-0910_csv.csv")
```

```
executed in 15ms, finished 23:54:46 2019-05-15
```



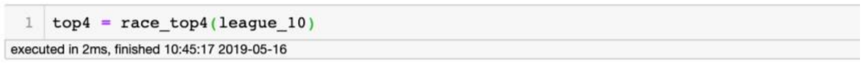
```python
def points(df, team):
    """Get point count for a team.
    Note:
     - Assigning -3 to a loss for visual purposes.
     """

    df = df[(df.HomeTeam==team) | (df.AwayTeam==team)]
    vals = df.values
    points = np.zeros(df.shape[0])
    running_count = 0
    for row in range(len(vals)):
        if vals[row][1] == team and vals[row][5] == "H":
            running_count += 3
            points[row] = running_count
        elif vals[row][2] == team and vals[row][5] == "A":
            running_count += 3
            points[row] = running_count
        elif vals[row][1] == team and vals[row][5] == "D":
            running_count += 1
            points[row] = running_count
        elif vals[row][2] == team and vals[row][5] == "D":
            running_count += 1
            points[row] = running_count
        else:
            running_count += -3
            points[row] += running_count
    return points
```

```
executed in 7ms, finished 23:54:46 2019-05-15
```



```python
def race_top4(df):
    """Get season long progression of top four teams at the end of the season."""

    teams = df.HomeTeam.unique()
    index = pd.DataFrame(list(df.index), columns=["Index"]).set_index("Index")
    index["Date"] = df.Date
    team_info = []
    for team in teams:
        curr_team = df[(df.HomeTeam == team) | (df.AwayTeam == team)]
        curr_team[team + "_Points"] = points(curr_team, team)
        curr = curr_team[[team+"_Points"]]
        index = index.join(curr, how="left")
    for team in index.columns:
        curr_points = 0
        for row in range(len(index[team])):
            if index[team].iloc[row] != index[team].iloc[row]:
                index[team].iloc[row] = curr_points
                curr_points = index[team].iloc[row]
            else:
                curr_points = index[team].iloc[row]
    last_week = index.iloc[-1].drop("Date").sort_values(ascending=False)
    index = index.set_index("Date")
    top_4 = last_week.head(4).index
    top_4 = index[top_4]
    return top_4
```

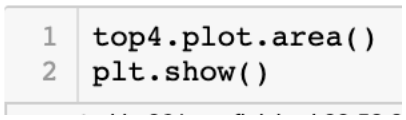
```
executed in 7ms, finished 23:54:46 2019-05-15
```



```python
top4 = race_top4(league_10)
```

```
executed in 2ms, finished 10:45:17 2019-05-16
```

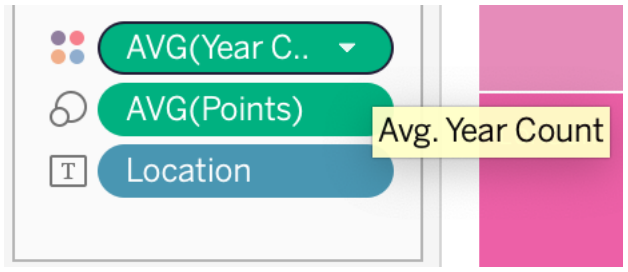
Plot data



```python
top4.plot.area()
plt.show()
```

Treemapping:

Used previously calculated data:



Plotted in Tableau

Interactive:
Same as scatter plot.

Link to Github repository: https://github.com/tylerursuy/Visualizations

Citations:

Data source: https://datahub.io/sports-data/english-premier-league