project2_anon

March 27, 2021

1 Project 2: Moneyball

For this project, we will be working with SQL databases and be performing exploratory analysis on the data on the moneyball time in baseball history. … Version: 3/26/21

1.0.1 Setting up Libraries and Connections

[1]:
```
# Importing libraries
import sqlite3 # The database is a sqlite db
import pandas as pd # Work with data in dataframes
import numpy as np # Need for types
import requests # Check webpages
import matplotlib.pyplot as plt # For plotting
import matplotlib.cm as cm # For coloring graphs
from pprint import pprint # Because pretty printing
import os # Get data for extra credit
import re # For parsing things
```

[2]:
```
# Establishing a connection the database
sqlite_file = 'lahman2014.sqlite'
conn = sqlite3.connect(sqlite_file)

cursor = conn.cursor() # Explore the sqlite table
```

2 Part 1: Wrangling

I will be using the salaries and teams tables of the "database"

From the documentation of the lahman database:
- Both teams and salaries are considered supplemental tables to tables like MASTER, Batting, Pitching, and Fielding

Teams: yearly stats and standing, important to note! **Team**-specific
Salaries: player salary data, important to note! **Player**-specific

2.0.1 Problem 1: Using SQL, compute a relation containing the total payroll and winning percentage (# wins/ # games * 100) for each team. In terms of SQL, do it for each team and each year (bc we care about seasons). Include other columns after your join to aid in later analysis. Be mindful of possible missing data in either table and explain yourself.

Short prose explaining my code/output below:

Exploring the database and finding the columns I want to use in the future in my relation.
In this section of my solution to problem 1, I want to explore the tables in question: Teams and Salaries. That way, I know what columns I want to grab later in my SQL relation query. I use the Pragma command in order to see the column names/type, and then decide I want to keep data on year, teamid, salary, games, wins, losses, and the name of the team.

What I plan on doing with missing data
In this section of my solution to problem 1, I explore possible missing data in the table.

First, I check for possible missing data (on data that we care about) in the Salaries table by checking existence and type checking. I find some weird \$0 salary data for two players, but according to Piazza @211, I did not impute on them. The table could be right in that they were paid \$0, but it is weird that other baseball references just have *'s for those players' salaries.

Second, I end up not finding any missing data (on data that we care about) in the Teams table with existence and type checking.

Finally, I do some set operations to see what type of JOIN I should do. Through my set operations, I find that data was input incorrectly for San Francisco Giants in 2014 in the Salaries table. The teamID was accidentally put in as 'SFG' and not 'SFN' like the rest of the Salaries and Teams tables. I showed that the players were part of the San Francisco Giants in 2014 by scraping a website and matching playerIDs to the HTML of the webpage. Similar issues were found for the New York Mets ('NYM' instead of 'NYN' like the rest of the table.)

All of the steps led to

For my JOIN, I end up selecting LEFT JOIN, because Salaries (team,year) combinations are actually a subset of the Teams table. In addition, since the min year we matched on in our intersection of (team,year) between Salaries and Teams is 1985, and none of the teams in the Teams table and not in Salaries table is from 1985 onward, we just drop them because we care about correlation of teams and their salary data, and without a corresponding Salary entry(ies) it is useless in our exploratory data analysis.

All of the steps above led to my solution of a SQL LEFT JOIN + inner Queries on the Salaries and Teams tables, to get a relation which is stored in a Pandas dataframe!

2.0.2 Exploring the database and finding the columns I want to use in the future in my relation:

Here, I will be going through the Salaries and Teams tables to see what is useful for exploratory data analysis.

[3]:
```
# Let's look at the tables we will be working with:
# Salaries and Teams

# PRAGMA allows us to see the
# index of a column, name, dtype, and
# other information. It's awesome!

# Get column data of Salaries
cursor.execute("PRAGMA table_info('Salaries');")
columns_salaries = cursor.fetchall()

print('Here are the columns for Salaries\n')

# Print the columns that we grabbed
for col in columns_salaries:
    print(col)

# Get column data of Teams
cursor.execute("PRAGMA table_info('Teams');")
columns_teams = cursor.fetchall()

print('\nHere are the columns for Teams\n')

# Print the columns that we grabbed
for col in columns_teams:
    print(col)
```

```
Here are the columns for Salaries

(0, 'yearID', 'INTEGER', 0, None, 0)
(1, 'teamID', 'TEXT', 0, None, 0)
(2, 'lgID', 'TEXT', 0, None, 0)
(3, 'playerID', 'TEXT', 0, None, 0)
(4, 'salary', 'REAL', 0, None, 0)

Here are the columns for Teams

(0, 'yearID', 'INTEGER', 0, None, 0)
(1, 'lgID', 'TEXT', 0, None, 0)
(2, 'teamID', 'TEXT', 0, None, 0)
(3, 'franchID', 'TEXT', 0, None, 0)
(4, 'divID', 'TEXT', 0, None, 0)
(5, 'Rank', 'INTEGER', 0, None, 0)
(6, 'G', 'INTEGER', 0, None, 0)
(7, 'Ghome', 'INTEGER', 0, None, 0)
(8, 'W', 'INTEGER', 0, None, 0)
(9, 'L', 'INTEGER', 0, None, 0)
```

```
(10, 'DivWin', 'TEXT', 0, None, 0)
(11, 'WCWin', 'TEXT', 0, None, 0)
(12, 'LgWin', 'TEXT', 0, None, 0)
(13, 'WSWin', 'TEXT', 0, None, 0)
(14, 'R', 'INTEGER', 0, None, 0)
(15, 'AB', 'INTEGER', 0, None, 0)
(16, 'H', 'INTEGER', 0, None, 0)
(17, '2B', 'INTEGER', 0, None, 0)
(18, '3B', 'INTEGER', 0, None, 0)
(19, 'HR', 'INTEGER', 0, None, 0)
(20, 'BB', 'INTEGER', 0, None, 0)
(21, 'SO', 'INTEGER', 0, None, 0)
(22, 'SB', 'INTEGER', 0, None, 0)
(23, 'CS', 'INTEGER', 0, None, 0)
(24, 'HBP', 'INTEGER', 0, None, 0)
(25, 'SF', 'INTEGER', 0, None, 0)
(26, 'RA', 'INTEGER', 0, None, 0)
(27, 'ER', 'INTEGER', 0, None, 0)
(28, 'ERA', 'REAL', 0, None, 0)
(29, 'CG', 'INTEGER', 0, None, 0)
(30, 'SHO', 'INTEGER', 0, None, 0)
(31, 'SV', 'INTEGER', 0, None, 0)
(32, 'IPouts', 'INTEGER', 0, None, 0)
(33, 'HA', 'INTEGER', 0, None, 0)
(34, 'HRA', 'INTEGER', 0, None, 0)
(35, 'BBA', 'INTEGER', 0, None, 0)
(36, 'SOA', 'INTEGER', 0, None, 0)
(37, 'E', 'INTEGER', 0, None, 0)
(38, 'DP', 'INTEGER', 0, None, 0)
(39, 'FP', 'REAL', 0, None, 0)
(40, 'name', 'TEXT', 0, None, 0)
(41, 'park', 'TEXT', 0, None, 0)
(42, 'attendance', 'INTEGER', 0, None, 0)
(43, 'BPF', 'INTEGER', 0, None, 0)
(44, 'PPF', 'INTEGER', 0, None, 0)
(45, 'teamIDBR', 'TEXT', 0, None, 0)
(46, 'teamIDlahman45', 'TEXT', 0, None, 0)
(47, 'teamIDretro', 'TEXT', 0, None, 0)
```

From the printout of the columns above, we can see why the directions say to join on teamID and yearID combinations (both columns are in both tables), and that is the data we are interested in.

Considering other columns for Exploratory Data Analysis (EDA):

Here is a list of columns I want to keep for future analysis / just to look at:

Salaries:
yearID, Year
teamID, Team
salary, Salary

Teams:
yearID, Year
teamID, Team
G, Games played
W, Wins
L, Losses
name, Team's full name

So my resulting relation will contain: year, teamid, salary, games, wins, losses, and the name of the team.

A full list of what the columns mean is available at: http://www.seanlahman.com/files/database/readme2014.txt, I only included a few here showing what I wanted for EDA.

2.0.3 What I plan on doing to check for missing data:

Check if there is missing data (1) within a table (2) team & year id missing in either table. Please see (1) or (2) designations in Sections A through C.

Section A: (1, look for missing data within a table) Looking for missing data within the Salaries table.

Importance of Section: Let's us know if we have missing data in our Salaries table. We want to know this for future SQL commands and if we take averages that involve a per player basis (this project does not have such averages per player, so this missing data is not necessarily a problem). But, we still need to know the missing data to be careful with our SQL.

[4]:
```python
# Unfortunately, the documentation does not list anything about missing data
# Let's look at a pattern for the data:
# First, I will use SQL to grab the data and put it into a pandas table
# just so it is a bit easier to look at.

# I am selecting the columns I care about (see previous section)
# from the Salaries table. Here, I am storing what I select in a
# Pandas dataframe called preview_salaries for an easier time
# while checking the data.
preview_salaries_query = "SELECT yearID, teamID, salary FROM Salaries"
preview_salaries = pd.read_sql(preview_salaries_query, conn)
preview_salaries.head(10)
```

[4]:
```
   yearID teamID    salary
0    1985    ATL  870000.0
1    1985    ATL  550000.0
2    1985    ATL  545000.0
3    1985    ATL  633333.0
4    1985    ATL  625000.0
5    1985    ATL  800000.0
6    1985    ATL  150000.0
7    1985    ATL  483333.0
```

```
8     1985    ATL  772000.0
9     1985    ATL  250000.0
```

[5]:
```python
# The column types should be the following,
# grab the first row to get these types, we
# can see there is nothing missing (that we care about)
# there easily from our print out above.

# Also, cool note, these types are the same
# as our pragma printout! But, just easier
# now with Pandas (for me).

# Printing out what the types should be
# for the cols I care about in Salaries.
print(type(preview_salaries.at[0 , 'yearID']))
print(type(preview_salaries.at[0 , 'teamID']))
print(type(preview_salaries.at[0 , 'salary']))
```

```
<class 'numpy.int64'>
<class 'str'>
<class 'numpy.float64'>
```

[6]:
```python
# So, a condition can be that if any of the columns we are
# interested in are missing, we make note of it
for row in preview_salaries.iterrows():
    year = preview_salaries.at[row[0], 'yearID']
    team = preview_salaries.at[row[0], 'teamID']
    salary = preview_salaries.at[row[0], 'salary']

    # We are checking, if the year exists and is an int,
    # if the team exists and the team is a string
    # if the salary exists/is not 0,
    # and the salary is a float, not nan if it is a float.
    if not(year and type(year) == np.int64 and
           team and type(team) == str and
           salary and type(salary) == np.float64 and np.isnan(salary) == False):
        print("Possible missing data (that we are interested in) in Salaries␣
 ↪table")
        print(year)
        print(team)
        print(salary)
```

```
Possible missing data (that we are interested in) in Salaries table
1993
NYA
0.0
Possible missing data (that we are interested in) in Salaries table
```

```
1999
PIT
0.0
```

[7]:
```python
# Completey weird that we have $0 data salary information!
# Let's take a closer look at the table and figure out the players involved.

# Grab the $0 salary players
cursor.execute("SELECT * FROM Salaries WHERE salary == 0.0")
missing_salary_players = cursor.fetchall()

# Let's see the entire entry of the players we grabbed
# from the Salaries table above.
print("Let's take a look at the weird things happening:")
for player in missing_salary_players:
    print(player)

# Let's see their names by going to the MASTER table
# and getting their full names using their playerID.
print("\nWe can see that there are players with \"missing\" salaries.\n")
print("Here are their names:")

for player in missing_salary_players:
    cursor.execute("SELECT nameFirst, nameLast FROM MASTER WHERE playerID == ?
↪", (player[3],))
    print(cursor.fetchall())
```

```
Let's take a look at the weird things happening:
(1993, 'NYA', 'AL', 'jamesdi01', 0.0)
(1999, 'PIT', 'NL', 'martija02', 0.0)

We can see that there are players with "missing" salaries.

Here are their names:
[('Dion', 'James')]
[('Javier', 'Martinez')]
```

Here, we can see that we have problems. A player *probably* did not make $0 an entire season, so lahman *possibly* treated missing data like a number (in this case 0.0).

We will look at the first \$0 player salary from 1993, James Dion: First, I want to double check my understanding is correct (they 0 means they don't know the salary): https://www.baseball-reference.com/players/j/jamesdi01.shtml#all_br-salaries

1987 24 Atlanta Braves \$120,000 ? 1987 USA Today survey
1988 25 Atlanta Braves \$360,000* ?
1989 26 Atlanta Braves \$400,000 ? 1990 TSN Guide 23 Arbitration loss; \$540,000 request
1990 27 Cleveland Indians \$650,000 ? USA Today
1992 29 New York Yankees \$387,500 ? 1994 Joint Exhibit 1 Includes \$22,500 signing bonus,

\$112,500 incentive
1993 30 New York Yankees * ?
1995 32 New York Yankees \$350,000 ? 1996 Joint Exhibit 1 Includes \$50K earned bonus

We can see the 1993 salary is missing on this website as well.

So, ***possibly*** missing salary data in year 1993 for player James Dion.
#### We will look at the second \$0 player salary from 1999, Javier Martinez:
https://www.baseball-reference.com/players/m/martija02.shtml

1998 21 Pittsburgh Pirates \$170,000 ? 11/23/98 USA Today
1999 22 Pittsburgh Pirates * ?

We can see the 1999 salary is missing on this website as well. Side note: he played only in the 1998 season before not being played in the 1999 season.
So, ***possibly*** missing salary data in year 1998 for player Javier Martinez.

*****
End of Section A: We have missing data in the Salaries table. This missing data is when the salaries are zero for the players James Dion (1994 NYA) and Javier Marinez (1999 PIT), see missing_salary_players for the players in particular.

2.0.4 Important conclusions! per Piazza post @211:

- Per Piazza post @211, we only care about the total payroll, so this missing data is 'okay.' And it would not really affect the total payroll if one player is missing, plus it is possible the salaries are zero, since the database documentation does not mention any form of missing data. If I was not instructed on what to do in this situation by the TA, I may have imputed for James Dion because he played in the season and was most likely paid around \$300,000 (though it is possible the table is right and he got paid \$0 dollars). I would not have imputed for Javier Martinez because per @211, he did not play so he was probably not paid.
- Overall, it is possible in 1993 for James Dion (assuming he was paid and the table is possibly wrong in assuming \$0 given our research from other baseball references), it would affect the payroll sum we calculate for NYA 1993, but not by much because the New York Yankees spends *many* millions of dollars usually on baseball players for a season.

Section B: (1, look for missing data within a table) Looking for missing data within the Teams table.

Importance of section: Let's us know if we need to deal with missing data (that we care about) in this section (like no winning percentages) for our teams.

[8]:
```
# Unfortunately, the documentation does not list anything about missing data
# Let's look at a pattern for the data:
# First, I will use SQL to grab the data and put it into a pandas table
# just so it is a bit easier to look at.

# We are only going to look for missing data
# for columns/variables that we care about in
# the Teams table.
preview_teams_query = "SELECT yearID, teamID, G, W, L, name FROM Teams"
```

```
preview_teams = pd.read_sql(preview_teams_query, conn)
preview_teams.head(10)
```

[8]:

```
   yearID teamID   G   W   L                    name
0    1871    BS1  31  20  10     Boston Red Stockings
1    1871    CH1  28  19   9  Chicago White Stockings
2    1871    CL1  29  10  19   Cleveland Forest Citys
3    1871    FW1  19   7  12     Fort Wayne Kekiongas
4    1871    NY2  33  16  17         New York Mutuals
5    1871    PH1  28  21   7   Philadelphia Athletics
6    1871    RC1  25   4  21    Rockford Forest Citys
7    1871    TRO  29  13  15           Troy Haymakers
8    1871    WS3  32  15  15      Washington Olympics
9    1872    BL1  58  35  19       Baltimore Canaries
```

[9]:

```
# The column types should be the following,
# grab the first row to get these types.
print(type(preview_teams.at[0 , 'yearID']))
print(type(preview_teams.at[0 , 'teamID']))
print(type(preview_teams.at[0 , 'G']))
print(type(preview_teams.at[0 , 'W']))
print(type(preview_teams.at[0 , 'L']))
print(type(preview_teams.at[0 , 'name']))
```

```
<class 'numpy.int64'>
<class 'str'>
<class 'numpy.int64'>
<class 'numpy.int64'>
<class 'numpy.int64'>
<class 'str'>
```

[10]:

```
# So, a condition can be that if any of the columns we are
# interested in are missing, we make note of it.
for row in preview_teams.iterrows():
    year = preview_teams.at[row[0], 'yearID']
    team = preview_teams.at[row[0], 'teamID']
    wins = preview_teams.at[row[0], 'W']
    losses = preview_teams.at[row[0], 'L']
    games = preview_teams.at[row[0], 'G']
    name = preview_teams.at[row[0], 'name']

    # We check if the year, team exist and are the correct types (int/str)
    # , that wins/losses are ints,
    # and that games played are not 0 (would be wrong to say no games played),␣
 ↪and
    # the name of the team exists.
    # Note, we do not check for a specific count of wins and losses, because of␣
↪ties,
    # possible cancellations midway, etc. It is not something we care about -␣
↪only
    # that (pretty much) # wins exists.
    if not(year and type(year) == np.int64 and
           team and type(team) == str and
           type(wins) == np.int64 and
           type(losses) == np.int64 and
           games and type(games) == np.int64
           and name and type(name) == str):
        print("Possible missing data (that we are interested in) in Salaries␣
↪table")
        print(year)
        print(team)
        print(wins)
        print(losses)
        print(games)
        print(name)
print('There is no missing data as there is no print out from loop above.')
```

```
There is no missing data as there is no print out from loop above.
```

*****

End of Section B: No missing data (that we are interested in) in the Teams table.

Section C: (2, look for missing data between the two tables) Looking for a missing year/team id combination in either table. So, I want to make sure that the *sets* of (year,team) are the same between Teams and Salaries.

Importance of section: Affects the JOIN for SQL commmand used to combine the tables for Problem 1.

[11]:
```
# If a (teamid, yearid) combination is only missing in one table, I want to
# do a LEFT/RIGHT JOIN
# If data is missing in both tables I want to do
# an INNER JOIN so I can do correlation analysis on "complete" data
# for our EDA.

# I can do set operations easily,
# because we used SQL to grab the tables that we are interested in.
# Everything is in Pandas so we can drop columns and make tuples
# easily.

# Create sets to operate on of tuples (yearID, teamID)
# from both Teams and Salaries.
set_teams = set((preview_teams.drop(columns = ['G', 'W', 'L', 'name'])).
↪apply(tuple, axis=1))
```

```
set_salaries = set((preview_salaries.drop(columns = ['salary'])).apply(tuple,␣
 ↪axis=1))

# Get the intersection of the two sets.
intersection = set_teams.intersection(set_salaries)

# See if there are year/team combos not in the intersection
# for the teams and salaries tables.
# These are the most important set operations - affects JOIN.
set_teams_no_inter = set_teams.difference(intersection)
set_salaries_no_inter = set_salaries.difference(intersection)
```

[12]:
```
# Here are set operations that are interesting

# Here is the size of the intersection, helps with seeing what
# is included in both tables
print("Here is the set size of team/year combinations \
intersecting between the Teams and Salaries tables:")
print(len(intersection))

# Just seeing the amount of data we are dealing with in Teams
print("Here is the set size of team/year combinations \
in Teams alone:")
print(len(set_teams))

# Just seeing the amount of data we are dealing with in Salaries
print("Here is the set size of team/year combinations \
in Salaries alone:")
print(len(set_salaries))

# Important: What is in Teams but not in Salaries, affects JOIN
print("Here is the set size of team/year combinations \
of Team - intersection with Salaries:")
print(len(set_teams_no_inter))

# Important: What is in Salaries but not in Teams, affects JOIN
print("Here is the set size of team/year combinations \
of Salaries - intersection with Teams:")
print(len(set_salaries_no_inter))
```

```
Here is the set size of team/year combinations intersecting between the Teams
and Salaries tables:
858
Here is the set size of team/year combinations in Teams alone:
2775
Here is the set size of team/year combinations in Salaries alone:
860
```

```
Here is the set size of team/year combinations of Team - intersection with
Salaries:
1917
Here is the set size of team/year combinations of Salaries - intersection with
Teams:
2
```

Let's explore what is missing between the two tables to see if it is something that was a simple error that led to our sets being weird (see the following parts (a) and (b) for exploration of why the sets didn't match up in the two tables):

- (a) Explore the team/yearids not matching with the salaries table.

[13]:
```python
# Print out what is in Salaries but not in Teams
print(set_salaries_no_inter)
```

```
{(2014, 'NYM'), (2014, 'SFG')}
```

Let's take a closer look at the first weird team code: NYM. My hypothesis is that this got mixed up with New York Mets. Their team code is weirdly NYN, different from an acronym of their name, so maybe the database mixed them up in 2014.

[14]:
```python
# Showing that in the Teams table, NYM does not exist but NYN does
cursor.execute("SELECT * FROM Teams WHERE teamID=='NYM'")
print("Teams table data with (incorrect) teamID NYM")
print(cursor.fetchall())

cursor.execute("SELECT * FROM Teams WHERE teamID=='NYN'")
print("Teams table data with (correct) teamID NYN")
# print(cursor.fetchall()) # uncomment this line and comment next to see entire␣
 ↪output
print(cursor.fetchall()[0])
print('... more years\n')

print('From output we can see that teamID NYN corresponds with New York Mets')
```

```
Teams table data with (incorrect) teamID NYM
[]
Teams table data with (correct) teamID NYN
(1962, 'NL', 'NYN', 'NYM', None, 10, 161, 80, 40, 120, None, None, 'N', 'N',
617, 5492, 1318, 166, 40, 139, 616, 991, 59, 48, None, None, 948, 801, 5.04, 43,
4, 10, 4290, 1577, 192, 571, 772, 210, 167, 0.96, 'New York Mets', 'Polo Grounds
IV', 922530, 100, 105, 'NYM', 'NYN', 'NYN')
… more years

From output we can see that teamID NYN corresponds with New York Mets
```

We know that we want to use teamID 'NYN' because it is in our Teams table, if my suspicions about 'NYM' accidentally being used for NY Mets are correct. Below, I explore the Salaries table, and I actually show that 'NYM' was accidentally used by

web scraping the roster for the NY Mets in 2014.

[15]:
```
# Pull Salaries results for teamIDs in the Salaries Table: NYM and NYN in 2014,
# see if they are both from the New York Mets roster.
# We can take advantage of the fact that Salaries is player-specific (as noted␣
 ↪earlier)
cursor.execute("SELECT * FROM Salaries WHERE teamID == 'NYM' AND yearID ==␣
 ↪2014")
nym_2014 = cursor.fetchall()

cursor.execute("SELECT * FROM Salaries WHERE teamID == 'NYN' AND yearID ==␣
 ↪2014")
nyn_2014 = cursor.fetchall()

print("Codes NYM (incorrect)")
print(*nym_2014, sep='\n')
print("\nCodes NYN (correct)")
print(*nyn_2014, sep='\n')
```

```
Codes NYM (incorrect)
(2014, 'NYM', 'NL', 'blackvi01', 502250.0)
(2014, 'NYM', 'NL', 'brownan02', 538045.0)
(2014, 'NYM', 'NL', 'colonba01', 9000000.0)
(2014, 'NYM', 'NL', 'darnatr01', 501560.0)
(2014, 'NYM', 'NL', 'dudalu01', 1637500.0)
(2014, 'NYM', 'NL', 'familje01', 502550.0)
(2014, 'NYM', 'NL', 'geedi01', 3625000.0)
(2014, 'NYM', 'NL', 'germego01', 504875.0)
(2014, 'NYM', 'NL', 'grandcu01', 13000000.0)
(2014, 'NYM', 'NL', 'lagarju01', 506636.0)
(2014, 'NYM', 'NL', 'lannajo01', 1500000.0)
(2014, 'NYM', 'NL', 'matsuda01', 1500000.0)
(2014, 'NYM', 'NL', 'mejiaje01', 509675.0)
(2014, 'NYM', 'NL', 'murphda08', 5700000.0)
(2014, 'NYM', 'NL', 'niesejo01', 5000000.0)
(2014, 'NYM', 'NL', 'parnebo01', 3700000.0)
(2014, 'NYM', 'NL', 'reckean01', 505340.0)
(2014, 'NYM', 'NL', 'ricesc01', 542500.0)
(2014, 'NYM', 'NL', 'satinjo01', 506809.0)
(2014, 'NYM', 'NL', 'tejadru01', 1100000.0)
(2014, 'NYM', 'NL', 'torreca01', 561875.0)
(2014, 'NYM', 'NL', 'valvejo01', 1000000.0)
(2014, 'NYM', 'NL', 'wheelza01', 512375.0)
(2014, 'NYM', 'NL', 'younger03', 1850000.0)

Codes NYN (correct)
(2014, 'NYN', 'NL', 'davisik02', 3500000.0)
(2014, 'NYN', 'NL', 'wrighda03', 20000000.0)
```

```
(2014, 'NYN', 'NL', 'youngch04', 7250000.0)
```

Let's double check to make sure the NYM error is isolated to 2014 (so all other salaries for other years are NYN)

[16]:
```
# Select from Salaries again, but this time with no
# year constraint to see if the error is in other
# years besides 2014.
# Try with codes: NYM (incorrect) and NYN (correct)
cursor.execute("SELECT * FROM Salaries WHERE teamID == 'NYM'")
nym_all = cursor.fetchall()

cursor.execute("SELECT * FROM Salaries WHERE teamID == 'NYN'")
nyn_all = cursor.fetchall()

print("Codes NYM (incorrect) all years")
print(*nym_all, sep='\n')
print("\nCodes NYN (correct) all years")
# Commented out because output is huge, essentially from 1985-2014
# Please uncomment to see full output
# print(*nyn_all, sep='\n')
# Printing out some output
print(str(nyn_all[0]) + '\n...\n' +
      str(nyn_all[50]) + '\n...\n' + str(nyn_all[len(nyn_all) - 1]))
```

```
Codes NYM (incorrect) all years
(2014, 'NYM', 'NL', 'blackvi01', 502250.0)
(2014, 'NYM', 'NL', 'brownan02', 538045.0)
(2014, 'NYM', 'NL', 'colonba01', 9000000.0)
(2014, 'NYM', 'NL', 'darnatr01', 501560.0)
(2014, 'NYM', 'NL', 'dudalu01', 1637500.0)
(2014, 'NYM', 'NL', 'familje01', 502550.0)
(2014, 'NYM', 'NL', 'geedi01', 3625000.0)
(2014, 'NYM', 'NL', 'germego01', 504875.0)
(2014, 'NYM', 'NL', 'grandcu01', 13000000.0)
(2014, 'NYM', 'NL', 'lagarju01', 506636.0)
(2014, 'NYM', 'NL', 'lannajo01', 1500000.0)
(2014, 'NYM', 'NL', 'matsuda01', 1500000.0)
(2014, 'NYM', 'NL', 'mejiaje01', 509675.0)
(2014, 'NYM', 'NL', 'murphda08', 5700000.0)
(2014, 'NYM', 'NL', 'niesejo01', 5000000.0)
(2014, 'NYM', 'NL', 'parnebo01', 3700000.0)
(2014, 'NYM', 'NL', 'reckean01', 505340.0)
(2014, 'NYM', 'NL', 'ricesc01', 542500.0)
(2014, 'NYM', 'NL', 'satinjo01', 506809.0)
(2014, 'NYM', 'NL', 'tejadru01', 1100000.0)
(2014, 'NYM', 'NL', 'torreca01', 561875.0)
(2014, 'NYM', 'NL', 'valvejo01', 1000000.0)
```

```
(2014, 'NYM', 'NL', 'wheelza01', 512375.0)
(2014, 'NYM', 'NL', 'younger03', 1850000.0)

Codes NYN (correct) all years
(1985, 'NYN', 'NL', 'backmwa01', 200000.0)
…
(1987, 'NYN', 'NL', 'backmwa01', 550000.0)
…
(2014, 'NYN', 'NL', 'youngch04', 7250000.0)
```

[17]:
```
# The following loop checks if code NYM is isolated to 2014.
# It is from output above, but just showing it programmatically.
for sal in nym_all:
    if sal[0] != 2014:
        print('Not isolated to 2014')
print('No errors from loop above so NYM error is isolated to 2014')
```

```
No errors from loop above so NYM error is isolated to 2014
```

Let's pull a roster of the NY Mets in 2014 and check that the players (pulled from the Salaries table) with team code NYM or NYN actually are all part of the team in 2014.

[18]:
```
# Get request to roster site for NY Mets 2014
r = requests.get('https://www.baseball-reference.com/teams/NYN/2014.shtml')
```

[19]:
```
# Get string of content to do searches on
content = str(r.content)

# I used the output below to look at the HTML
# to see that playerIDs are in the HTML.
# Commented out because the output is huge.
# print(content) #Please feel free to uncomment
```

[20]:
```
# Showing we can access the playerID in our
# pulled lists above.
print(nym_2014[0][3])
print(nyn_2014[0][3])
```

```
blackvi01
davisik02
```

We can access playerIDs, and those player IDs are in the HTML, so we can just check that they are in the roster if their playerID s in the HTML. If curious about the HTML please use the print commented out above.

[21]:
```
# The following loop checks if the players listed under
# 'NYM' (and even 'NYN' as a sanity check) in 2014 are players of
# the New York Mets roster in 2014.
# We do this by checking if the playerID
# is in the HTML.
for player in nym_2014:
    if not(player[3] in content):
        print("Player not included in roster")
for player in nyn_2014:
    if not(player[3] in content):
        print("Player not included in roster")
print("""All players with teamIDs:
NYM or NYN in 2014 were part of the real roster for the New York Mets that␣
 ↪season.
This is shown by no error print outs in the for loops per playerID.""")
```

```
All players with teamIDs:
NYM or NYN in 2014 were part of the real roster for the New York Mets that
season.
This is shown by no error print outs in the for loops per playerID.
```

As shown in the for loops, we looped through the player ids and made sure they were a part of the 2014 roster for the New York Mets. We actually found they are all a part of the New York Mets regardless of their teamID being NYM or NYN, so there was a simple mistake in this table between the two. The correct acronym (and most of the data is this acronym) is NYN, so I will go ahead and edit the database to make any id with NYM to NYN so that we do not accidentally calculate a lower payroll for the NY Mets in 2014!

[22]:
```
# We found earlier that the teamID errors are in the Salaries table.
# We need to update NYM to NYN to get correct data for 2014 Mets payroll.
cursor.execute("SELECT * FROM Salaries WHERE teamID = 'NYM'")
print(cursor.fetchall())
```

```
[(2014, 'NYM', 'NL', 'blackvi01', 502250.0), (2014, 'NYM', 'NL', 'brownan02',
538045.0), (2014, 'NYM', 'NL', 'colonba01', 9000000.0), (2014, 'NYM', 'NL',
'darnatr01', 501560.0), (2014, 'NYM', 'NL', 'dudalu01', 1637500.0), (2014,
'NYM', 'NL', 'familje01', 502550.0), (2014, 'NYM', 'NL', 'geedi01', 3625000.0),
(2014, 'NYM', 'NL', 'germego01', 504875.0), (2014, 'NYM', 'NL', 'grandcu01',
13000000.0), (2014, 'NYM', 'NL', 'lagarju01', 506636.0), (2014, 'NYM', 'NL',
'lannajo01', 1500000.0), (2014, 'NYM', 'NL', 'matsuda01', 1500000.0), (2014,
'NYM', 'NL', 'mejiaje01', 509675.0), (2014, 'NYM', 'NL', 'murphda08',
5700000.0), (2014, 'NYM', 'NL', 'niesejo01', 5000000.0), (2014, 'NYM', 'NL',
'parnebo01', 3700000.0), (2014, 'NYM', 'NL', 'reckean01', 505340.0), (2014,
'NYM', 'NL', 'ricesc01', 542500.0), (2014, 'NYM', 'NL', 'satinjo01', 506809.0),
(2014, 'NYM', 'NL', 'tejadru01', 1100000.0), (2014, 'NYM', 'NL', 'torreca01',
561875.0), (2014, 'NYM', 'NL', 'valvejo01', 1000000.0), (2014, 'NYM', 'NL',
'wheelza01', 512375.0), (2014, 'NYM', 'NL', 'younger03', 1850000.0)]
```

[23]:
```
# Update NYM teamID in the Salaries table to NYN
cursor.execute("UPDATE Salaries SET teamID = 'NYN' WHERE teamID = 'NYM'")
cursor.fetchall() # Just do this to flush (see above cell)
```

```
# Note, we are not committing bc I want to maintain the state
# of the file. Even without commiting, our update will persist
# throughout the rest of this session with fetchall()! Totally cool.
# Just as a note bc I wanted to just
# have changes persist a session rather than
# committing, and changing the db forever.
```

[23]:
```
[]
```

[24]:
```
# We've updated the database
# Let's run the same command to see if
# NYM teamIDs exist in Salaries
cursor.execute("SELECT * FROM Salaries WHERE teamID = 'NYM'")
print(cursor.fetchall()) # show the update worked
# It should be an empty list now!!!!!
# It is, and we are happy.
```

```
[]
```

[25]:
```
# Grab the Salaries cols again, including teamID
preview_salaries_query = "SELECT yearID, teamID, salary FROM Salaries"
# Update this for future use / set operations
preview_salaries = pd.read_sql(preview_salaries_query, conn)
s = list(preview_salaries['teamID']) # Get Series as a list for teamIDs
preview_salaries.head(10)
```

[25]:
```
   yearID teamID    salary
0    1985    ATL  870000.0
1    1985    ATL  550000.0
2    1985    ATL  545000.0
3    1985    ATL  633333.0
4    1985    ATL  625000.0
5    1985    ATL  800000.0
6    1985    ATL  150000.0
7    1985    ATL  483333.0
8    1985    ATL  772000.0
9    1985    ATL  250000.0
```

[26]:
```
# Confirm our change of NYM again in the pandas grab above.
# Just make sure it works for future operations
if 'NYM' in s:
    print("Bad update")
else:
    print("Everything is updated.")
```

```
Everything is updated.
```

Now, let's look at the next weird teamID/code: 'SFG' to see if it is a similar error like above. We would want to update it like we did for 'NYM' so we do not majorly undercount our payroll sum for the Giants in 2014. My hypothesis is that somehow the person inputting into the database mixed up the teamIDs for the San Francisco Giants. The actual team code is SFN. It could also be another acronym issue like we found before. Let's check!

[27]:
```
# Showing that in the Teams table, SFG does not exist but SFN does
cursor.execute("SELECT * FROM Teams WHERE teamID=='SFG'")
print("Team data with (incorrect) teamID SFG")
print(cursor.fetchall())

cursor.execute("SELECT * FROM Teams WHERE teamID=='SFN'")
print("Team data with (correct) teamID SFN")
# print(cursor.fetchall()) # uncomment this line and comment next to see entire␣
 ↪output
print(cursor.fetchall()[0])
print('... more years')

print('From output we can see that teamID SFN corresponds with San Francisco␣
 ↪Giants')
```

```
Team data with (incorrect) teamID SFG
[]
Team data with (correct) teamID SFN
(1958, 'NL', 'SFN', 'SFG', None, 3, 154, 77, 80, 74, None, None, 'N', 'N', 727,
5318, 1399, 250, 42, 170, 531, 817, 64, 29, None, None, 698, 614, 3.98, 38, 7,
25, 4167, 1400, 166, 512, 775, 152, 156, 0.97, 'San Francisco Giants', 'Seals
Stadium', 1272625, 98, 96, 'SFG', 'SFN', 'SFN')
… more years
From output we can see that teamID SFN corresponds with San Francisco Giants
```

So, we know that we want to use teamID 'SFN' because it is in our Teams table if my suspicions about 'SFG' accidentally being used for SF Giants are correct. Below, I explore the Salaries table, and later actually show that 'SFG' was accidentally used by web scraping an official roster of the SF Giants in 2014.

[28]:
```
# Pull results for teamIDs: SFG and SFN in 2014, (remember we found earlier it␣
 ↪is SFG, 2014)
# to see if they are both from the San Francisco Giants roster.
cursor.execute("SELECT * FROM Salaries WHERE teamID == 'SFG' AND yearID ==␣
 ↪2014")
sfg_2014 = cursor.fetchall()

cursor.execute("SELECT * FROM Salaries WHERE teamID == 'SFN' AND yearID ==␣
 ↪2014")
sfn_2014 = cursor.fetchall()

print("Codes SFG (incorrect)")
print(*sfg_2014, sep='\n')
print("\nCodes SFN (correct)")
print(*sfn_2014, sep='\n')
```

```
Codes SFG (incorrect)
(2014, 'SFG', 'NL', 'abreuto01', 745000.0)
(2014, 'SFG', 'NL', 'adriaeh01', 500500.0)
(2014, 'SFG', 'NL', 'affelje01', 6000000.0)
(2014, 'SFG', 'NL', 'ariasjo01', 1150000.0)
(2014, 'SFG', 'NL', 'beltbr01', 2900000.0)
(2014, 'SFG', 'NL', 'blancgr01', 2525000.0)
(2014, 'SFG', 'NL', 'bumgama01', 3750000.0)
(2014, 'SFG', 'NL', 'colvity01', 1000000.0)
(2014, 'SFG', 'NL', 'crawfbr01', 560000.0)
(2014, 'SFG', 'NL', 'garcija01', 4500000.0)
(2014, 'SFG', 'NL', 'gutieju01', 850000.0)
(2014, 'SFG', 'NL', 'hudsoti01', 11000000.0)
(2014, 'SFG', 'NL', 'linceti01', 17000000.0)
(2014, 'SFG', 'NL', 'lopezja02', 4000000.0)
(2014, 'SFG', 'NL', 'machije01', 505000.0)
(2014, 'SFG', 'NL', 'morsemi01', 6000000.0)
(2014, 'SFG', 'NL', 'paganan01', 10250000.0)
(2014, 'SFG', 'NL', 'peavyja01', 14500000.0)
(2014, 'SFG', 'NL', 'pencehu01', 16000000.0)
(2014, 'SFG', 'NL', 'perezju02', 501000.0)
(2014, 'SFG', 'NL', 'petityu01', 845000.0)
(2014, 'SFG', 'NL', 'poseybu01', 12500000.0)
(2014, 'SFG', 'NL', 'romose01', 5500000.0)
(2014, 'SFG', 'NL', 'sanchhe01', 512000.0)
(2014, 'SFG', 'NL', 'sandopa01', 8250000.0)
(2014, 'SFG', 'NL', 'scutama01', 6666667.0)
(2014, 'SFG', 'NL', 'vogelry01', 5000000.0)

Codes SFN (correct)
(2014, 'SFN', 'NL', 'cainma01', 20000000.0)
```

Let's double check to make sure the SFG error is isolated to 2014 (so all other salaries for other years are SFN) in case we have to check other rosters from previous years.

[29]:
```
# Select players with SFG or SFN as their teamID from
# Salaries without a year limitation to see if the error
# is isolated to 2014.
cursor.execute("SELECT * FROM Salaries WHERE teamID == 'SFG'")
sfg_all = cursor.fetchall()

cursor.execute("SELECT * FROM Salaries WHERE teamID == 'SFN'")
sfn_all = cursor.fetchall()
```

```python
print("Codes SFG (incorrect) all years")
print(*sfg_all, sep='\n')
print("\nCodes SFN (correct) all years")
# Commented out because output is huge, essentially from 1985-2014
# Please uncomment to see full output
# print(*sfn_all, sep='\n')
# Printing out some output
print(str(sfn_all[0]) + '\n...\n' +
      str(sfn_all[50]) + '\n...\n' + str(sfn_all[len(sfn_all) - 1]))
```

```
Codes SFG (incorrect) all years
(2014, 'SFG', 'NL', 'abreuto01', 745000.0)
(2014, 'SFG', 'NL', 'adriaeh01', 500500.0)
(2014, 'SFG', 'NL', 'affelje01', 6000000.0)
(2014, 'SFG', 'NL', 'ariasjo01', 1150000.0)
(2014, 'SFG', 'NL', 'beltbr01', 2900000.0)
(2014, 'SFG', 'NL', 'blancgr01', 2525000.0)
(2014, 'SFG', 'NL', 'bumgama01', 3750000.0)
(2014, 'SFG', 'NL', 'colvity01', 1000000.0)
(2014, 'SFG', 'NL', 'crawfbr01', 560000.0)
(2014, 'SFG', 'NL', 'garcija01', 4500000.0)
(2014, 'SFG', 'NL', 'gutieju01', 850000.0)
(2014, 'SFG', 'NL', 'hudsoti01', 11000000.0)
(2014, 'SFG', 'NL', 'linceti01', 17000000.0)
(2014, 'SFG', 'NL', 'lopezja02', 4000000.0)
(2014, 'SFG', 'NL', 'machije01', 505000.0)
(2014, 'SFG', 'NL', 'morsemi01', 6000000.0)
(2014, 'SFG', 'NL', 'paganan01', 10250000.0)
(2014, 'SFG', 'NL', 'peavyja01', 14500000.0)
(2014, 'SFG', 'NL', 'pencehu01', 16000000.0)
(2014, 'SFG', 'NL', 'perezju02', 501000.0)
(2014, 'SFG', 'NL', 'petityu01', 845000.0)
(2014, 'SFG', 'NL', 'poseybu01', 12500000.0)
(2014, 'SFG', 'NL', 'romose01', 5500000.0)
(2014, 'SFG', 'NL', 'sanchhe01', 512000.0)
(2014, 'SFG', 'NL', 'sandopa01', 8250000.0)
(2014, 'SFG', 'NL', 'scutama01', 6666667.0)
(2014, 'SFG', 'NL', 'vogelry01', 5000000.0)

Codes SFN (correct) all years
(1985, 'SFN', 'NL', 'bluevi01', 250000.0)
…
(1987, 'SFN', 'NL', 'clarkwi02', 120000.0)
…
(2014, 'SFN', 'NL', 'cainma01', 20000000.0)
```

[30]:
```
# The following loop checks if code SFG is isolated to 2014.
# It is, but just showing it programmatically.
for sal in sfg_all:
    if sal[0] != 2014:
        print('Not isolated to 2014')
print('No errors from loop above so SFG error is isolated to 2014')
```

```
No errors from loop above so SFG error is isolated to 2014
```

Let's pull a roster of the SF Giants in 2014 and check that the players (pulled from the Salaries table) with team code SFG or SFN actually are all part of the team in 2014.

[31]:
```
# Get request to roster site for the San Francisco Giants in 2014
r = requests.get('https://www.baseball-reference.com/teams/SFN/2014.shtml')
```

[32]:
```
# Get string of content to do searches on
content = str(r.content)
# I looked at the HTML content
# feel free to uncomment next line
# Commented out because output is huge.
#print(content)
```

[33]:
```
# Showing we can access the playerID in our
# pulled lists above.
print(sfg_2014[0][3])
print(sfn_2014[0][3])
```

```
abreuto01
cainma01
```

We can access playerIDs, and those player IDs are in the HTML, so we can just check that they are in the roster if their playerIDs in the HTML. If curious about the HTML please use the print commented out above.

[34]:
```
# The following loop checks if the players listed under
# 'SFG' (and even 'SFN' as a sanity check) in 2014 are players of
# the San Francisco Giants roster in 2014.
for player in sfg_2014:
    if not(player[3] in content):
        print("Player not included in roster")
for player in sfn_2014:
    if not(player[3] in content):
        print("Player not included in roster")
print("""All players with teamIDs:
SFG or SFN in 2014 were part of the real roster for the SF Giants that season.
This is shown by no error print outs in the for loops per playerID.""")
```

```
All players with teamIDs:
SFG or SFN in 2014 were part of the real roster for the SF Giants that season.
```

```
This is shown by no error print outs in the for loops per playerID.
```

As shown by our loops above, we looped through playerIDs and made sure they were a part of the 2014 roster for the SF Giants. We actually found they are all a part of the San Francisco Giants regardless of their team id being SFG or SFN, so I will go ahead and edit the database to make any id with SFG to SFN so that we do not accidentally calculate a lower payroll!

[35]:
```
# We found earlier that the database is wrong in the Salaries table.
# We need to update SFG to SFN to get correct data for 2014 Giants payroll.
cursor.execute("SELECT * FROM Salaries WHERE teamID = 'SFG'")
print(cursor.fetchall())
```

```
[(2014, 'SFG', 'NL', 'abreuto01', 745000.0), (2014, 'SFG', 'NL', 'adriaeh01',
500500.0), (2014, 'SFG', 'NL', 'affelje01', 6000000.0), (2014, 'SFG', 'NL',
'ariasjo01', 1150000.0), (2014, 'SFG', 'NL', 'beltbr01', 2900000.0), (2014,
'SFG', 'NL', 'blancgr01', 2525000.0), (2014, 'SFG', 'NL', 'bumgama01',
3750000.0), (2014, 'SFG', 'NL', 'colvity01', 1000000.0), (2014, 'SFG', 'NL',
'crawfbr01', 560000.0), (2014, 'SFG', 'NL', 'garcija01', 4500000.0), (2014,
'SFG', 'NL', 'gutieju01', 850000.0), (2014, 'SFG', 'NL', 'hudsoti01',
11000000.0), (2014, 'SFG', 'NL', 'linceti01', 17000000.0), (2014, 'SFG', 'NL',
'lopezja02', 4000000.0), (2014, 'SFG', 'NL', 'machije01', 505000.0), (2014,
'SFG', 'NL', 'morsemi01', 6000000.0), (2014, 'SFG', 'NL', 'paganan01',
10250000.0), (2014, 'SFG', 'NL', 'peavyja01', 14500000.0), (2014, 'SFG', 'NL',
'pencehu01', 16000000.0), (2014, 'SFG', 'NL', 'perezju02', 501000.0), (2014,
'SFG', 'NL', 'petityu01', 845000.0), (2014, 'SFG', 'NL', 'poseybu01',
12500000.0), (2014, 'SFG', 'NL', 'romose01', 5500000.0), (2014, 'SFG', 'NL',
'sanchhe01', 512000.0), (2014, 'SFG', 'NL', 'sandopa01', 8250000.0), (2014,
'SFG', 'NL', 'scutama01', 6666667.0), (2014, 'SFG', 'NL', 'vogelry01',
5000000.0)]
```

[36]:
```
# Update SFG teamID in the Salaries table to SFN
cursor.execute("UPDATE Salaries SET teamID = 'SFN' WHERE teamID = 'SFG'")
cursor.fetchall()
```

[36]:
```
[]
```

[37]:
```
# We updated the database
cursor.execute("SELECT * FROM Salaries WHERE teamID = 'SFG'")
print(cursor.fetchall()) # show the update worked
# It should be an empty list now and it is!!!
# No more SFG errors!
```

```
[]
```

[38]:
```
# Grab the Salaries cols again, including teamID
# now that everything is fixed for 2014 for
# the Mets and the Giants.
preview_salaries_query = "SELECT yearID, teamID, salary FROM Salaries"
```

```
preview_salaries = pd.read_sql(preview_salaries_query, conn)
s = list(preview_salaries['teamID']) # for checking update in next cell
preview_salaries.head(10)
```

[38]:
```
   yearID teamID    salary
0    1985    ATL  870000.0
1    1985    ATL  550000.0
2    1985    ATL  545000.0
3    1985    ATL  633333.0
4    1985    ATL  625000.0
5    1985    ATL  800000.0
6    1985    ATL  150000.0
7    1985    ATL  483333.0
8    1985    ATL  772000.0
9    1985    ATL  250000.0
```

[39]:
```
# Confirm our change of SFG again in the pandas grab
# Just make sure it works for future operations
if 'SFG' in s:
    print("Bad update")
else:
    print("Everything is updated.")
```

```
Everything is updated.
```

- End (a):

Closing thoughts on what we found with the the missing things in the salaries table. The weird team codes for SF Giants and NY Mets were only messed up in 2014. Maybe the person inserting the data was new and made mistakes with using acronyms. While unfortunate this mistake happened, we were able to catch it and fix it so it would not adversely affect our payroll data. However, I do not think it is all their fault.

Some interesting things to note about the state of the naming schemes on other baseball databases:

In fact, different databases on the web refer to the same teams by these different acronyms! https://legacy.baseballprospectus.com/sortable/extras/team_codes.php?this_year=2015 uses NYN/SFN ONLY, so it follows the lahman database precedent (all years but 2014).

While other websites like baseball-reference accept both in their URLs for the teams: https://www.baseball-reference.com/teams/SFN/2014.shtml and https://www.baseball-reference.com/teams/SFG/2014.shtml (see SFN vs SFG in URLs), both point to the San Francisco Giants. Same thing happens with the Mets (NYN vs NYM).

So, there is clearly miscommunication or someone did not stick to precedent of which teamid to use in their database for the data we are using for this project.

In fact, it looks like there is miscommunication among other baseball databases all over the internet! The issue just happened to come to light while we were cleaning this database.

It was super interesting to see how this error could have possibly occured! I am glad we could fix the ones we saw.

TL;DR:
For the purposes of this project and maintaining the precedent already set with the Teams table and prior entries in the Salaries table, we updated our table to make sure it uses teamID = 'SFN' for the San Francisco Giants and teamID = 'NYN' for the New York Mets.

- (b) Explore the team/yearids not matching with the teams table.

[40]:
```
# Feel free to uncomment the following:
#print(set_teams_no_inter)

# This shows the (year,team) combos from the teams table
# that did not intersect with the salaries table.
# Basically, prints a lot from the 1800s - 1900s.
```

[41]:
```
# Printing min date in our intersection
# between Salaries and Teams:
mini = 2015
for tup in intersection:
    if tup[0] < mini:
        mini = tup[0]
print("Minimum year match")
print(mini)
```

```
Minimum year match
1985
```

[42]:
```
# I want to know if I care about this data
# that did not match.
# If our earliest match is 1985, then
# there is no point correcting data
# that has no match for payroll/winning percentage EDA.
for team in set_teams_no_inter:
    if team[0] >= 1985:
        print("We care about this team")
print("""\nNo print out that these teams are from 1985 onwards, \
which is the 'first' match on our data, so it is okay to drop \
these extra years/teams combos in Teams with a JOIN""")
```

```

No print out that these teams are from 1985 onwards, which is the 'first' match
on our data, so it is okay to drop these extra years/teams combos in Teams with
a JOIN
```

- End (b):

What I did: I just did a quick check to make sure the teams are not ones I care about for future analysis (1985 onwards). Because we do not care about these teams, we can just ignore them (plus they do not have a match in the Salaries table!!)

Now that we have double checked the data is correct/we care about it, let's redo our set operations to see what SQL/JOIN operation that we want to do on our table (note that we updated the preview_salaries already so we can go ahead with our set operations).

[43]:
```python
# Create sets to operate on of tuples (yearID, teamID)
set_teams = set((preview_teams.drop(columns = ['G', 'W', 'L', 'name'])).
 ↪apply(tuple, axis=1))
set_salaries = set((preview_salaries.drop(columns = ['salary'])).apply(tuple,␣
 ↪axis=1))

# Get the intersection
intersection = set_teams.intersection(set_salaries)

# See if there are year/team combos not in the intersection
# for the teams and salaries tables
set_teams_no_inter = set_teams.difference(intersection)
set_salaries_no_inter = set_salaries.difference(intersection)
```

[44]:
```python
print("Here is the set size of team/year combinations intersecting between the␣
 ↪Teams and Salaries tables:")
print(len(intersection))
print("Here is the set size of team/year combinations in Teams alone:")
print(len(set_teams))
print("Here is the set size of team/year combinations in Salaries alone:")
print(len(set_salaries))
print("Here is the set size of team/year combinations of Team - intersection␣
 ↪with Salaries:")
print(len(set_teams_no_inter))
print("Here is the set size of team/year combinations of Salaries -␣
 ↪intersection with Teams:")
print(len(set_salaries_no_inter))
```

```
Here is the set size of team/year combinations intersecting between the Teams
and Salaries tables:
858
Here is the set size of team/year combinations in Teams alone:
2775
Here is the set size of team/year combinations in Salaries alone:
858
Here is the set size of team/year combinations of Team - intersection with
Salaries:
1917
Here is the set size of team/year combinations of Salaries - intersection with
Teams:
0
```

2.0.5 Important notes from our set operations above:

Looking at our set operations alone, we have huge amounts of noncorrollating data. However, that is ONLY in the Teams table!!
It is important to know that for this project, we are interested in the **correlation** between payroll and team winning percentage.
A LEFT/RIGHT JOIN makes sense because the Salaries table is a subset of the Teams table. See explanation above, we show that the first team/year match is in 1985, and all teams in the Teams table that did not match with the Salaries table had corresponding years <1985, which is not what we care about / what we matched on.
It is actually pretty awesome that Salaries (team,year) combinations is encompassed by the Teams table. It also makes sense because a lot of the Salaries recorded are from the 1900s and Teams has data even from the 1800s, where they probably did not record salaries as well.

Now that we have decided to use an LEFT/RIGHT join, we can go ahead and create our SQL relation! We have reached the point of doing what Problem 1 asked - using SQL to make a relation/table containing the total payroll and winning percentage for team and year combinations.

[45]:
```
# Here is my SQL query to get a new relation that we want
# I get the stuff I want from Salaries
# then the stuff I want from Teams
# * If curious about my selection of cols/vars,
# please refer to the first part of my solution to
# problem 1 above *
# Then after that, I select everything I want and then
# do a JOIN with Salaries LEFT JOIN Teams essentially

# And we have our relation stored in a Pandas dataframe!

# Here is our query (line by line):
# SELECT the year, team, total payroll, games, wins, losses, name, and winning␣
 ↪percentage
# FROM our Salaries table after calculating total_payroll, and
# FROM our Teams table after calculating the winning percentage
# ON year and team combinations as asked in Problem 1.
query = """SELECT Teams_Select.yearID, Salary_Select.teamID, Salary_Select.
 ↪total_payroll,
Teams_Select.G, Teams_Select.W, Teams_Select.L, Teams_Select.name, Teams_Select.
 ↪winning_percentage

FROM (SELECT yearID, teamID, sum(salary) as total_payroll FROM Salaries GROUP␣
 ↪BY yearID, teamID) Salary_Select

LEFT JOIN (SELECT yearID, teamID, G, W, L, name, (100*(CAST(W AS FLOAT))/
 ↪(CAST(G AS FLOAT))) as winning_percentage FROM Teams) Teams_Select
```

```
ON (Salary_Select.teamID=Teams_Select.teamID) AND (Salary_Select.
 ↪yearID=Teams_Select.yearID)"""

moneyball_df = pd.read_sql(query, conn)

# This dataframe is going to be my "main" dataframe
# I will make spin-off dataframes from this one for plotting
# but this is the one that I really considre holding everything
# I care about.
# ^ Just a note if you see me making other dfs for plotting
moneyball_df
```

[45]:

```
     yearID teamID  total_payroll    G   W   L                   name  \
0      1985    ATL     14807000.0  162  66  96        Atlanta Braves
1      1985    BAL     11560712.0  161  83  78     Baltimore Orioles
2      1985    BOS     10897560.0  163  81  81        Boston Red Sox
3      1985    CAL     14427894.0  162  90  72    California Angels
4      1985    CHA      9846178.0  163  85  77     Chicago White Sox
..      …      …              …    …   ..  ..                   …
853    2014    SLN    120693000.0  162  90  72  St. Louis Cardinals
854    2014    TBA     72689100.0  162  77  85        Tampa Bay Rays
855    2014    TEX    112255059.0  162  67  95         Texas Rangers
856    2014    TOR    109920100.0  162  83  79     Toronto Blue Jays
857    2014    WAS    131983680.0  162  96  66  Washington Nationals

     winning_percentage
0             40.740741
1             51.552795
2             49.693252
3             55.555556
4             52.147239
..                  …
853           55.555556
854           47.530864
855           41.358025
856           51.234568
857           59.259259

[858 rows x 8 columns]
```

2.0.6 * Please note, I noticed some weird things in things in the database. I included a section at the end of this file labeled by ** to indicate my further analysis. For the purposes of the project and matching the readme, I matched on teamIDs (which gave me the same regression as we were supposed to get per the readme), however, I noticed/explored some weird things in case you wanted us to look at that as well for grading.

2.0.7 End of Problem 1

3 End of Part 1: Wrangling

4 Part 2: Exploratory Data Analysis

In this part we will be exploring relationships between payroll and winning percentage and more! This will be a lot of fun.

4.1 Exploring Payroll Distribution

4.1.1 Problem 2: Write code to produce plots that illustrate the distribution of payrolls accross teams conditioned on time (from 1990-2014).

Short prose explaining my code for this problem:
For this problem, I use my main dataframe called moneyball_df and then grab a dataframe of just want I am interested in (years 1990-2014 as specified in the problem) for plotting purposes. Then I use this sub-dataframe for plotting. In addition, I noticed that the matplotlib color cycle is only about 10 colors on rotation. Since we have ~30 teams, we want more colors! So I went to the matplotlib website and made my own color cycle of colors I thought would be ncie for the graphs. Then, I plotted a line graph of payroll over time for the teams (can see mean and spread on this graph) and a stacked bar chart of teams over time (see totals and how the total pool of money gets larger and larger).

[46]:
```python
# Let's make a dataframe form our moneyball_df dataframe that
# only includes years 1990 to 2014. We can do that easily!
payroll_plotting_df = moneyball_df[moneyball_df['yearID'].between(1990, 2014)]
# For the purposes of plotting, drop other columns
payroll_plotting_df = payroll_plotting_df.drop(['G', 'W', 'L', 'name',␣
↪'winning_percentage'], axis=1)
payroll_plotting_df
# Use the yearID on our x-axis, teamID as our legend and winning percentage
```

[46]:
```
     yearID teamID  total_payroll
130    1990    ATL     14555501.0
131    1990    BAL      9680084.0
132    1990    BOS     20558333.0
133    1990    CAL     21720000.0
134    1990    CHA      9491500.0
..      ...    ...            ...
853    2014    SLN    120693000.0
854    2014    TBA     72689100.0
```

```
855     2014    TEX    112255059.0
856     2014    TOR    109920100.0
857     2014    WAS    131983680.0

[728 rows x 3 columns]
```

Because we have a large number of teams about 30, I want to increase the default color set from 10 in matplotlib to something more using https://matplotlib.org/stable/gallery/color/named_colors.html. I am just going to choose colors easy to see for myself, and based on what I think looks 'nice.'

[47]:
```
# Here is how many teams there are in the moneyball dataframe
print(len(set(moneyball_df['teamID'])))
```

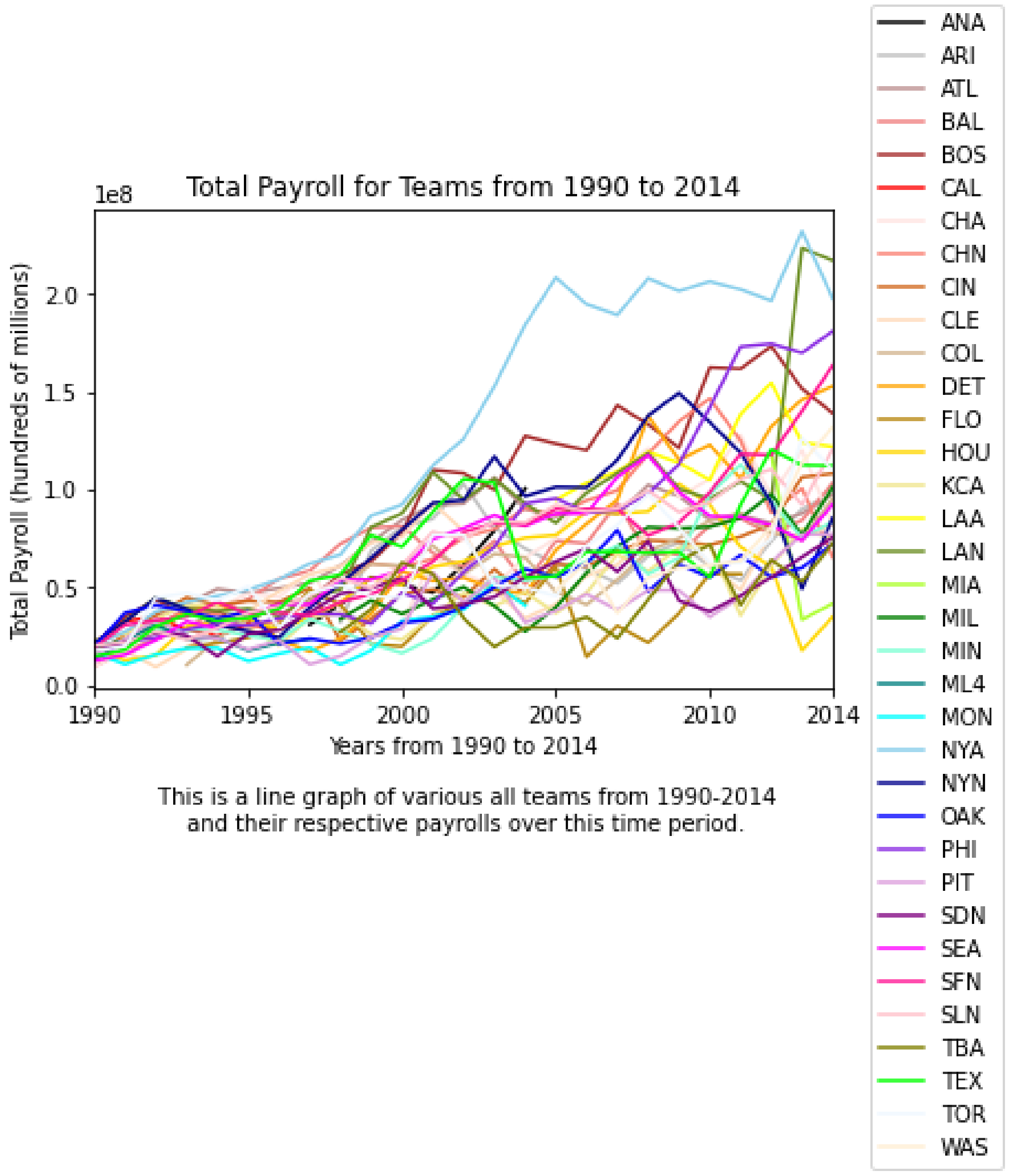
```
35
```

[48]:
```
# Defining about 40 colors that I like
# Want unique colors!
# Also, these colors are beautiful and it is nice
# to check out the colors from matplotlib
colors = ['black', 'silver', 'rosybrown', 'lightcoral',
          'brown', 'red', 'mistyrose', 'salmon', 'chocolate',
          'peachpuff', 'tan', 'orange', 'darkgoldenrod',
          'gold', 'khaki', 'yellow', 'olivedrab', 'greenyellow',
          'green', 'aquamarine', 'teal',
          'aqua', 'skyblue', 'darkblue', 'blue', 'blueviolet', 'plum',
          'purple', 'magenta', 'deeppink', 'pink', 'olive', 'lime',
          'aliceblue', 'papayawhip', 'honeydew', 'seashell']
```

Side note (mainly for me because I am new to matplotlib) but 1e8 means 10 ^ 8, matplotlib uses scientific notation for the y-axis sometimes (it does not mean applying a function like log on the y data). https://stackoverflow.com/questions/25715333/in-python-plots-what-does-1e8-mean-in-reference-to-the-y-axis

[49]:
```
# Want to plot per team over the years,
# and we already know the total payroll
# so grab first and unstack it, from our plotting df

# Plotting a line graph ( a plot for distribution of payrolls conditioned on␣
↪time )
payroll_plot = payroll_plotting_df.groupby(['yearID', 'teamID']).
↪first()['total_payroll'].unstack().plot(color=colors)

# Plotting, extra fields (eg legend, ticks, title, etc)
payroll_plot.legend(loc=(1.05,-1))

payroll_plot.set_xlabel("Years from 1990 to 2014\n\n \
This is a line graph of various all teams from 1990-2014\n \
and their respective payrolls over this time period.")

payroll_plot.set_xlim(1990,2014)
payroll_plot.set_xticks([1990, 1995, 2000, 2005, 2010, 2014])
payroll_plot.set_xticklabels([1990, 1995, 2000, 2005, 2010, 2014])

payroll_plot.set_ylabel("Total Payroll (hundreds of millions)")

payroll_plot.set_title('Total Payroll for Teams from 1990 to 2014')
```

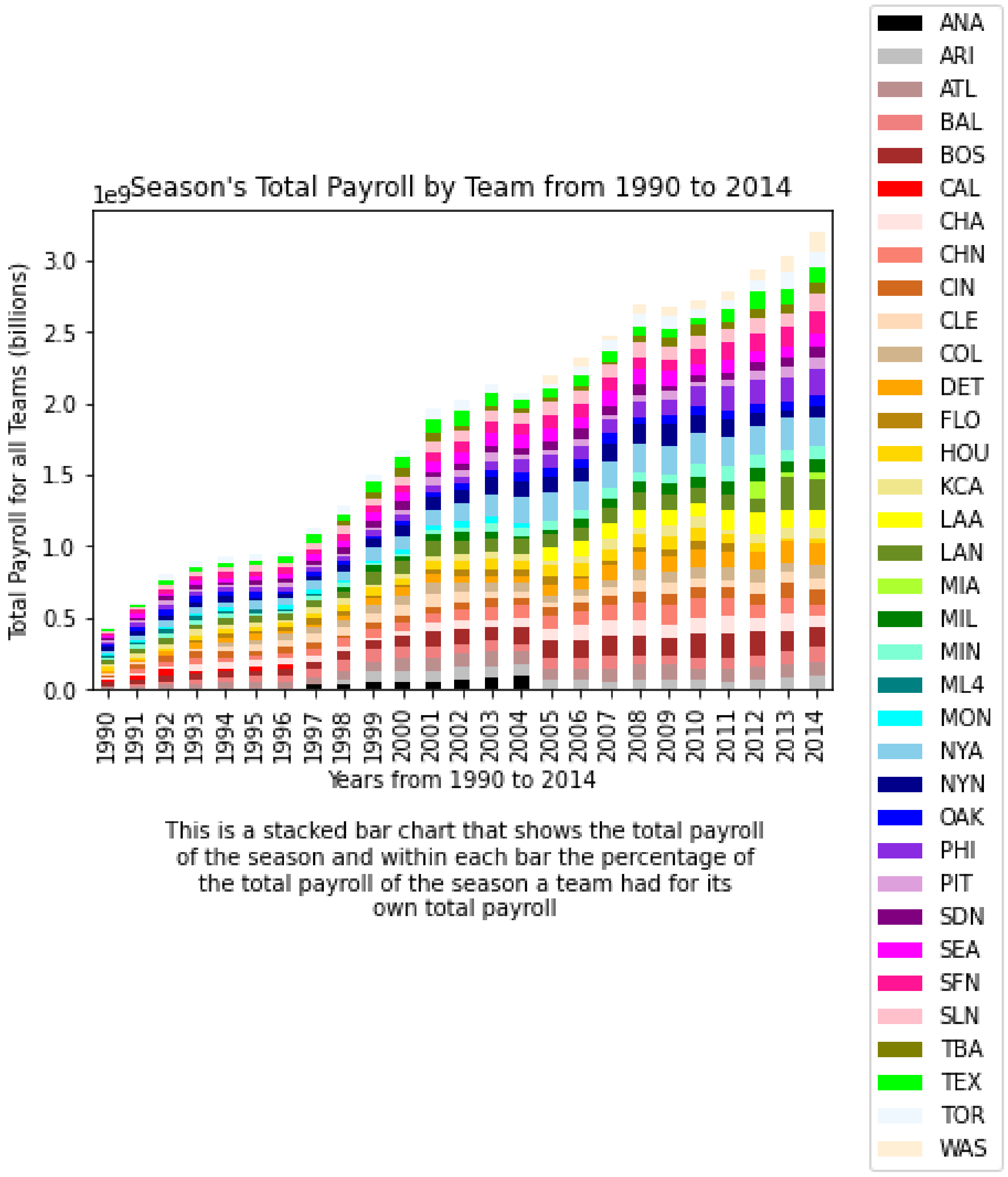
```
[49]: Text(0.5, 1.0, 'Total Payroll for Teams from 1990 to 2014')
```



This is a line graph of various all teams from 1990-2014
and their respective payrolls over this time period.

[50]:
```
# Another plot showing distribution: stacked bar plot!
# Each year will be the total payroll for that year
# and then the bar graph will have different colors
# representing teams.
# this will help with getting an idea of the total payroll as well.

# Also, we can see the distribution not just over the years,
# of how the money was distrbuted per season with this graph for each team!
# So, we can see NYA as larger chunks per year than teams like BAL.

season_plot = payroll_plotting_df.groupby(['yearID', 'teamID']).
 ↪first()['total_payroll'].unstack().plot.bar(stacked=True, color=colors)

# Plotting, extra fields (eg legend, ticks, title, etc)
season_plot.legend(loc=(1.05,-1))

season_plot.set_xlabel("Years from 1990 to 2014\n\n \
This is a stacked bar chart that shows the total payroll\n \
of the season and within each bar the percentage of\n \
the total payroll of the season a team had for its\n \
own total payroll")

season_plot.set_ylabel("Total Payroll for all Teams (billions)")
season_plot.set_title("Season's Total Payroll by Team from 1990 to 2014")
```

```
[50]: Text(0.5, 1.0, "Season's Total Payroll by Team from 1990 to 2014")
```



This is a stacked bar chart that shows the total payroll of the season and within each bar the percentage of the total payroll of the season a team had for its own total payroll

4.1.2 End of Problem 2

4.1.3 Question 1: What statements can you make about the distributions of payrolls conditioned on time based on these plots? Remember you can make statements in terms of central tendency, spread, etc.

From the graphs of payrolls conditioned on time, the graph, 'Total Payroll for Teams from 1990 to 2014', is a line graph that shows the total payrolls of teams per year from 1990-2014. In relation

to the question, it appears to show that the payroll of teams is more spread out as time increases because the lines become less and less bunched up together. In addition, this graph shows that there is a trend that the payrolls tend to increase over time.
The graph, 'Season's Total Payroll by Team from 1990 to 2014', is a stacked bar char that shows the total payroll for a season and what percentage each team had of the season's total payroll. In relation to the question, it appears to show that the total payroll for a season increased over time because the bars get taller as the years progress.

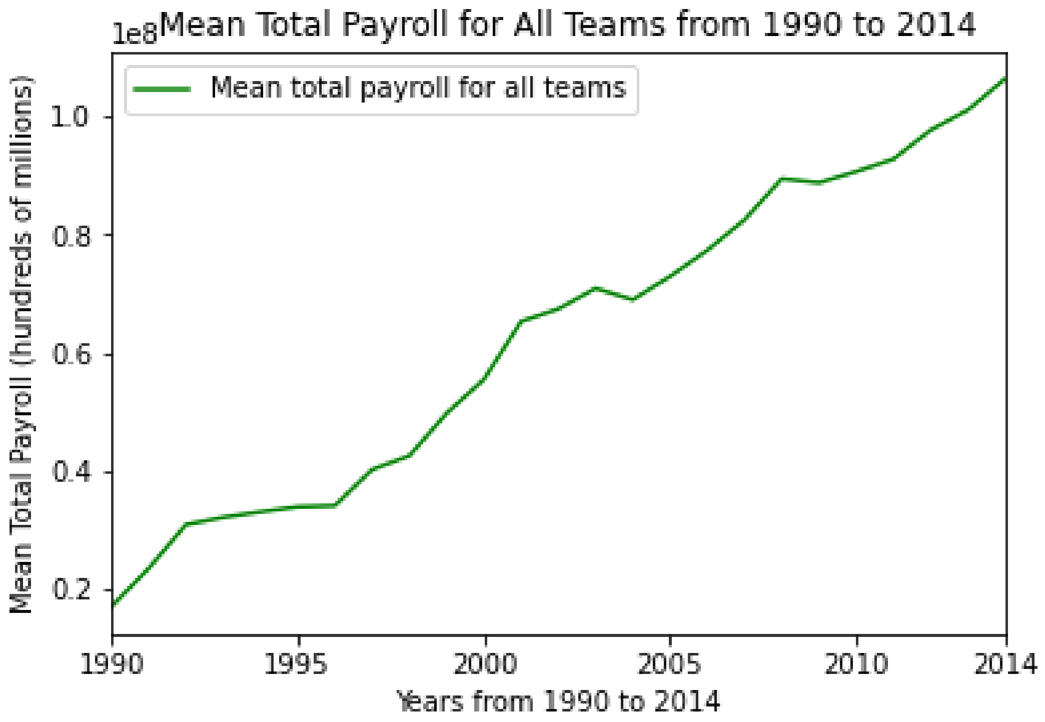
4.1.4 Problem 3: Write code to produce plots that specifically show at least one of the statements you made in Question 1. For example, if you make a statement that there is a trend for payrolls to decrease over time, make a plot of a statistic for central tendency (e.g. mean payroll) vs. time to show that specifically.

Short prose explaining my code:
For this problem, I am going to show my statement that payrolls tend to increase over time. I do this by plotting a graph of the mean of total payrolls over 1990-2014, as well as a graph of the median of total payrolls over 1990-2014. I do this by grouping the df by year, and then calulate mean/median on total payroll column for my teamsand plot! Pandas groupby was heavily used to make plotting easier. In addition, I plotted another statement I made. I said that it seems like payrolls among teams get more spread out over time, so I plotted the variance of all teams from 1990-2014 as well using the same groupby method and then calling var() from the Pandas groupby functions on the total payroll.

I made a statement that there is a trend that the payrolls tend to increase over time. I will show plots of both the mean and median over time to "show" my statement that the payrolls tend to increase over time is correct.

[51]:
```
# Mean plot, grab our plotting df and
# calculate the mean of the total payroll for all teams
mean_plot = payroll_plotting_df.groupby(['yearID']).mean()['total_payroll'].
 ↪plot(style='g')

# Plotting, (legend, labels, etc)
mean_plot.legend(['Mean total payroll for all teams'])

mean_plot.set_xlabel("Years from 1990 to 2014\n\n \
This is a line plot of the mean payroll for all teams from 1990 to 2014")

mean_plot.set_xlim(1990,2014)
mean_plot.set_xticks([1990, 1995, 2000, 2005, 2010, 2014])
mean_plot.set_xticklabels([1990, 1995, 2000, 2005, 2010, 2014])

mean_plot.set_ylabel("Mean Total Payroll (hundreds of millions)")
mean_plot.set_title('  Mean Total Payroll for All Teams from 1990 to 2014')
```

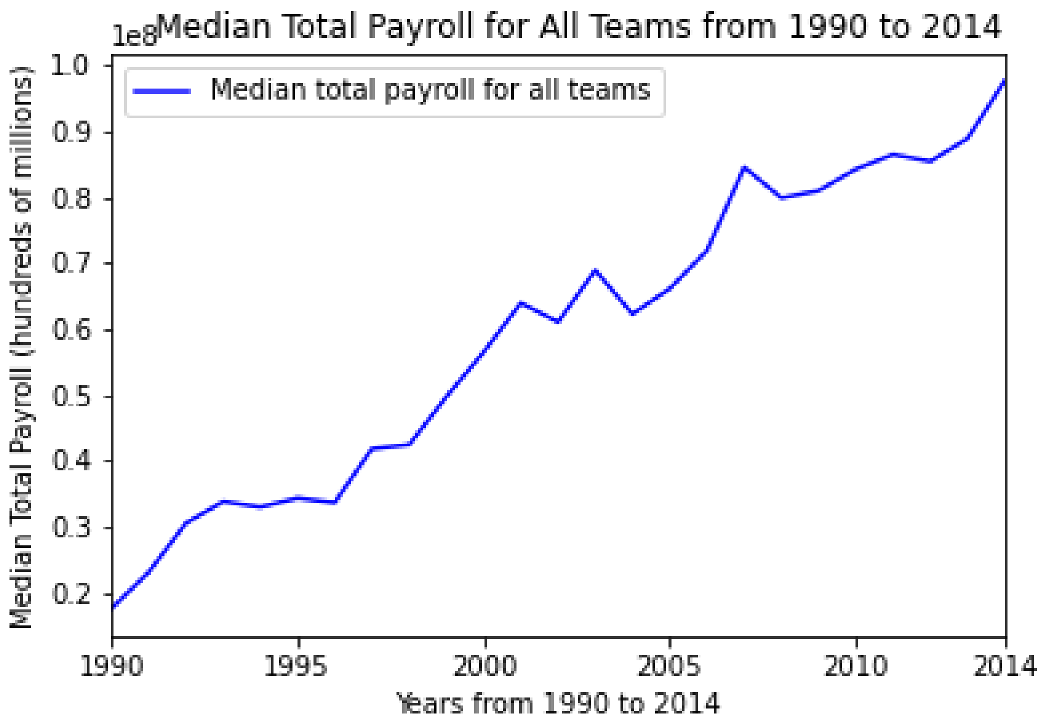
[51]:
```
Text(0.5, 1.0, '  Mean Total Payroll for All Teams from 1990 to 2014')
```



This is a line plot of the mean payroll for all teams from 1990 to 2014

[52]:
```
# Median plot, grab our plotting df and
# calculate the median of the total payroll for all teams
median_plot = payroll_plotting_df.groupby(['yearID']).median()['total_payroll'].
 ↪plot(style='b')

# Plotting (legend, labels, etc)
median_plot.legend(['Median total payroll for all teams'])

median_plot.set_xlabel("Years from 1990 to 2014\n\n \
This is a line plot of the median payroll for all teams from 1990 to 2014")

median_plot.set_xlim(1990,2014)
median_plot.set_xticks([1990, 1995, 2000, 2005, 2010, 2014])
median_plot.set_xticklabels([1990, 1995, 2000, 2005, 2010, 2014])

median_plot.set_ylabel("Median Total Payroll (hundreds of millions)")
median_plot.set_title('    Median Total Payroll for All Teams from 1990 to␣
 ↪2014')
```

[52]:
```
Text(0.5, 1.0, '    Median Total Payroll for All Teams from 1990 to 2014')
```

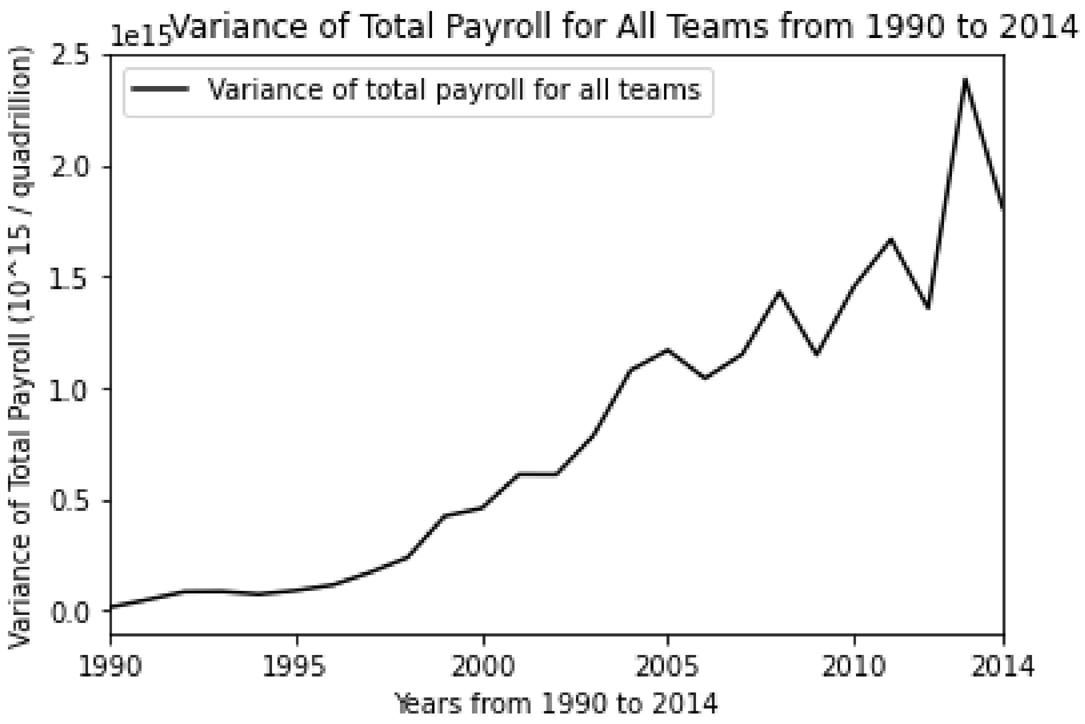


This is a line plot of the median payroll for all teams from 1990 to 2014

I also made a statement that the data seems to spread out more over time. I will show a variance plot just as an extra to ones shown above.

[53]:
```python
# Variance plot, grab our plotting df and
# calculate the variance of the total payroll for all teams
var_plot = payroll_plotting_df.groupby(['yearID']).var()['total_payroll'].
 ↪plot(style='k')

# Plotting (legend, labels, ticks, etc)
var_plot.legend(['Variance of total payroll for all teams'])

var_plot.set_xlabel("Years from 1990 to 2014\n\n \
This is a line plot of the variance of the total payroll\n \
for all teams from 1990 to 2014")

var_plot.set_xlim(1990,2014)
var_plot.set_xticks([1990, 1995, 2000, 2005, 2010, 2014])
var_plot.set_xticklabels([1990, 1995, 2000, 2005, 2010, 2014])

var_plot.set_ylabel("Variance of Total Payroll (10^15 / quadrillion)")
var_plot.set_title('             Variance of Total Payroll for All Teams from␣
 ↪1990 to 2014')
```

```
[53]: Text(0.5, 1.0, '            Variance of Total Payroll for All Teams from 1990
      to 2014')
```



This is a line plot of the variance of the total payroll for all teams from 1990 to 2014

4.1.5 End of Problem 3

4.2 Correlation between payroll and winning percentage

4.2.1 Problem 4: Write code to discretize year into five time periods (can use pandas.cut), and then make a scatterplot showing mean winning percentage (y-axis) vs. mean payroll (x-axis) for each of the five time periods. You could add a regression line (Numpy has a polyfit method) in each scatter plot to ease interpretation.

Short prose explaining my code:
In this problem, I get a sub-dataframe of my main dataframe containing all of my information (moneyball_df) and store it in another dataframe corr_df.

With this sub-dataframe called corr_df, I calculate the year into five time periods by using the pandas.cut function on the column yearID. I drop irrelevant rows in terms of plotting and leave the teamID, total_payroll, winning_percentage, and time_period.
With this sub_dataframe, I cut it into 5 separate dataframes corr_1 through corr_5, each grouped by the time_period, which I group by grabbing the bin numbers and creating an Interval object to compare to.

Now, with these five dataframes for each time period, I calculate a dataframe for plotting for each, where I get the mean total payroll and mean winning percentage per team. So each corr_1 to corr_5, has its own plotting thing.

Then, I plot for each time period. I plot the regression I got from numpy polyfit after passing the series of the mean payroll and mean winning percentage. In addition, I create a scatter plot for each of the teams' mean winning percentage and mean payroll, which is placed on the same plot as the regression line. Also, I calcualte the difference in the acutal avg winning percentage - the value of the of regression line, in order to make analysis of teams that stand out easier. I print out the result for easier analysis later!

I repeat this until all 5 time periods are plotted.

Note: I did this on all years I matched on. If we just did 1990-2014, we would be cutting out data we actually have correlation on. I think it is important to see all the data we have (not just 1990-2014), and use it! It will affect our regressions too, so I decided to do all years I matched on (1985-2014).

[54]:
```
# I am going to use pandas.cut to cut our table into 5 sections
# Goal: Make a df for plotting the correlation between
# payroll and winning percentage.

# From out main dataframe moneyball_df, drop not needed
# columns for plotting payroll/winning percentage
corr_df = moneyball_df.drop(['G', 'W', 'L', 'name'], axis=1)

# Use Pandas cut to discretize into five time periods
corr_df['time_period'] = pd.cut(corr_df['yearID'], bins=5)

# Drop the yearID because we have bins now
corr_df.drop(['yearID'], axis=1)
```

[54]:
```
    teamID  total_payroll  winning_percentage         time_period
0      ATL     14807000.0           40.740741  (1984.971, 1990.8]
1      BAL     11560712.0           51.552795  (1984.971, 1990.8]
2      BOS     10897560.0           49.693252  (1984.971, 1990.8]
3      CAL     14427894.0           55.555556  (1984.971, 1990.8]
4      CHA      9846178.0           52.147239  (1984.971, 1990.8]
..     …              …                   …                   …
853    SLN    120693000.0           55.555556    (2008.2, 2014.0]
854    TBA     72689100.0           47.530864    (2008.2, 2014.0]
855    TEX    112255059.0           41.358025    (2008.2, 2014.0]
856    TOR    109920100.0           51.234568    (2008.2, 2014.0]
857    WAS    131983680.0           59.259259    (2008.2, 2014.0]

[858 rows x 4 columns]
```

[55]:
```
# Grab the cuts, that way we can grab the intervals!
cuts = pd.cut(corr_df['yearID'], bins=5, retbins=True)
```

```
cuts[1][3] # just showing we can grab the intervals bc given bins
```

```
[55]: 2002.4
```

```
[56]: # Now that we have cut everything into time periods,
# we can get separate dataframes for each of the time periods.

# Will print heads to show it worked
# Drop yeaID and time_period because
# we won't need them for these plots

# Grab the first period
corr_1 = corr_df[corr_df['time_period'] == pd.Interval(cuts[1][0], cuts[1][1],␣
 ↪closed='right')]
# I print out here just to show the cut works by printing year and time_period
# I end up dropping those two because they become irrelevant, but
# I just wanted to show it for purposes of the project.
print(corr_1.head())
corr_1 = corr_1.drop(['yearID', 'time_period'], axis=1)

# Grab the second period
corr_2 = corr_df[corr_df['time_period'] == pd.Interval(cuts[1][1], cuts[1][2],␣
 ↪closed='right')]
print(corr_2.head())
corr_2 = corr_2.drop(['yearID', 'time_period'], axis=1)

# Grab the third period
corr_3 = corr_df[corr_df['time_period'] == pd.Interval(cuts[1][2], cuts[1][3],␣
 ↪closed='right')]
print(corr_3.head())
corr_3 = corr_3.drop(['yearID', 'time_period'], axis=1)

# Grab the fourth period
corr_4 = corr_df[corr_df['time_period'] == pd.Interval(cuts[1][3], cuts[1][4],␣
 ↪closed='right')]
print(corr_4.head())
corr_4 = corr_4.drop(['yearID', 'time_period'], axis=1)

# Grab the fifth period
corr_5 = corr_df[corr_df['time_period'] == pd.Interval(cuts[1][4], cuts[1][5],␣
 ↪closed='right')]
print(corr_5.head())
corr_5 = corr_5.drop(['yearID', 'time_period'], axis=1)
```

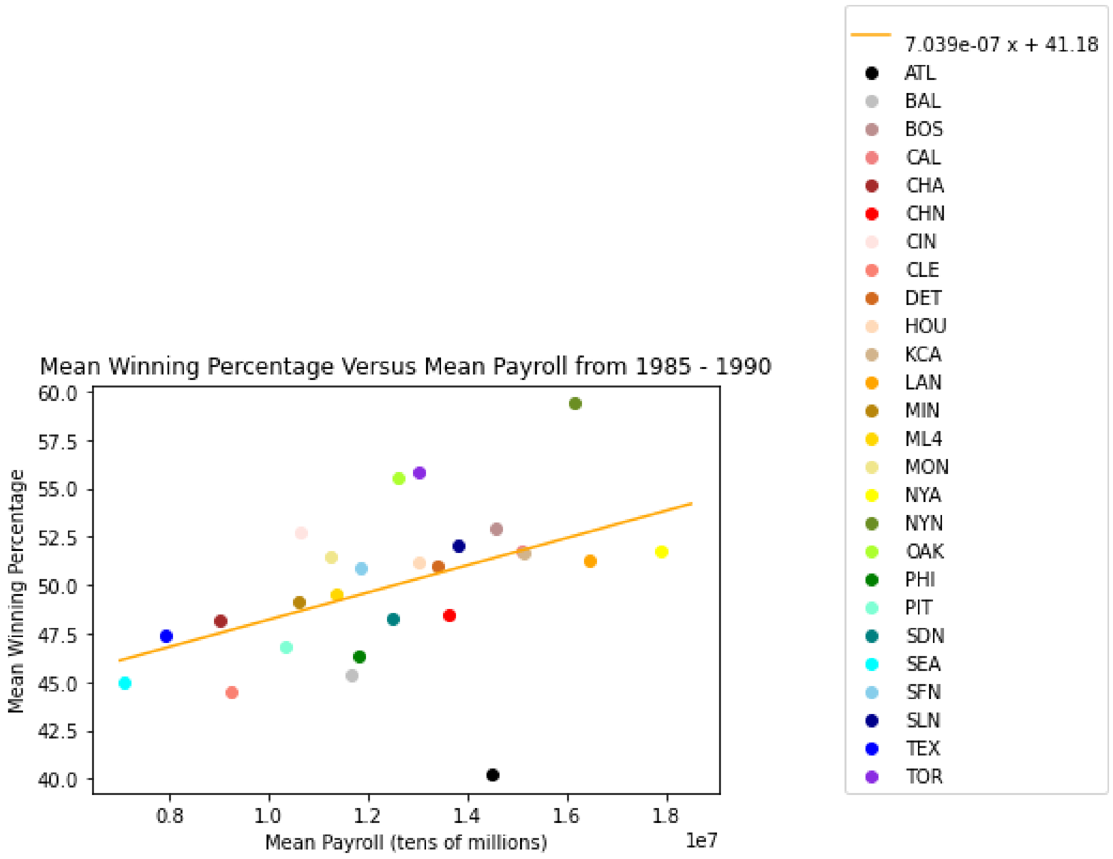
```
   yearID teamID  total_payroll  winning_percentage         time_period
0    1985    ATL     14807000.0           40.740741  (1984.971, 1990.8]
1    1985    BAL     11560712.0           51.552795  (1984.971, 1990.8]
2    1985    BOS    10897560.0           49.693252  (1984.971, 1990.8]
3    1985    CAL    14427894.0           55.555556  (1984.971, 1990.8]
4    1985    CHA     9846178.0           52.147239  (1984.971, 1990.8]
     yearID teamID  total_payroll  winning_percentage        time_period
156    1991    ATL     18403500.0           58.024691  (1990.8, 1996.6]
157    1991    BAL     17519000.0           41.358025  (1990.8, 1996.6]
158    1991    BOS     35167500.0           51.851852  (1990.8, 1996.6]
159    1991    CAL     33060001.0           50.000000  (1990.8, 1996.6]
160    1991    CHA     16919667.0           53.703704  (1990.8, 1996.6]
     yearID teamID  total_payroll  winning_percentage        time_period
320    1997    ANA     31135472.0           51.851852  (1996.6, 2002.4]
321    1997    ATL     52278500.0           62.345679  (1996.6, 2002.4]
322    1997    BAL     58516400.0           60.493827  (1996.6, 2002.4]
323    1997    BOS     43558750.0           48.148148  (1996.6, 2002.4]
324    1997    CHA     57740000.0           49.689441  (1996.6, 2002.4]
     yearID teamID  total_payroll  winning_percentage        time_period
498    2003    ANA     79031667.0           47.530864  (2002.4, 2008.2]
499    2003    ARI     80657000.0           51.851852  (2002.4, 2008.2]
500    2003    ATL    106243667.0           62.345679  (2002.4, 2008.2]
501    2003    BAL     73877500.0           43.558282  (2002.4, 2008.2]
502    2003    BOS     99946500.0           58.641975  (2002.4, 2008.2]
     yearID teamID  total_payroll  winning_percentage        time_period
678    2009    ARI     73115666.0           43.209877  (2008.2, 2014.0]
679    2009    ATL     96726166.0           53.086420  (2008.2, 2014.0]
680    2009    BAL     67101666.0           39.506173  (2008.2, 2014.0]
681    2009    BOS    121345999.0           58.641975  (2008.2, 2014.0]
682    2009    CHA     96068500.0           48.765432  (2008.2, 2014.0]
```

Now that I have the time period dataframes, I need to calculate the mean payroll and the mean winning_percentage for each team during that time period. Then, plot it! Luckily, Pandas and matplotlib can do it for me!

[57]:
```
# Plotting with matplotlib!
# I learned a lot!
# Figure is the "window" you are working in
# In your window, you can have multiple plots within it
# Think similar to MATLAB plotting
# So, we can add a plot to a figure, which is our ax
# This plot that we add is the "graph" that we think of
# A graph has a title, legend, and most importantly, the data.

# Because our data is so different per time period, I need to recreate
# each figure and subplot within it because of axes.
```

[58]:
```
# First plot: (1984.971, 1990.8]
# This is essentially 1985 - 1990 (no partial years)
# We know this because in part 1 we saw years
# are represented by integers.
# Grabbing a dataframe from our time period, that contains the data we want to␣
 ↪plot
plot_df1 = pd.DataFrame(data=[corr_1.groupby(['teamID']).
 ↪mean()['total_payroll'],
                           corr_1.groupby(['teamID']).
 ↪mean()['winning_percentage']]).transpose()
# Period 1 Plot
fig1 = plt.figure()
win_pay1 = fig1.add_subplot(111)
win_pay1.set_prop_cycle('color', colors)

# Linear Regression
p1 = np.poly1d(np.polyfit(plot_df1['total_payroll'],␣
 ↪plot_df1['winning_percentage'], 1))
xp1 = np.linspace(.7*(10 ** 7), 1.85*(10 ** 7), 100)
win_pay1.plot(xp1, p1(xp1), label=p1, color='orange')
dis1 = list() # Get distances

# Scatter Plot
for key,data in plot_df1.groupby('teamID'):
    win_pay1.scatter(data['total_payroll'], data['winning_percentage'],␣
 ↪label=key)
    dis1.append((key, float(data['winning_percentage']) -␣
 ↪p1(data['total_payroll'])[0]))

# Plotting (legend, labels, etc)
win_pay1.legend(loc=(1.2,0))
win_pay1.set_xlabel("Mean Payroll (tens of millions)\n\n \
This is a scatter plot of the avg mean payroll and avg winning percentage\n \
for each team, along with a regression of the plot for time period 1.")
win_pay1.set_ylabel("Mean Winning Percentage")
win_pay1.set_title('Mean Winning Percentage Versus Mean Payroll from 1985 -␣
 ↪1990')

fig1.show()
dis1.sort(key=lambda x:-x[1]) # reverse sort
print('Distances from regression')
pprint(dis1)
```

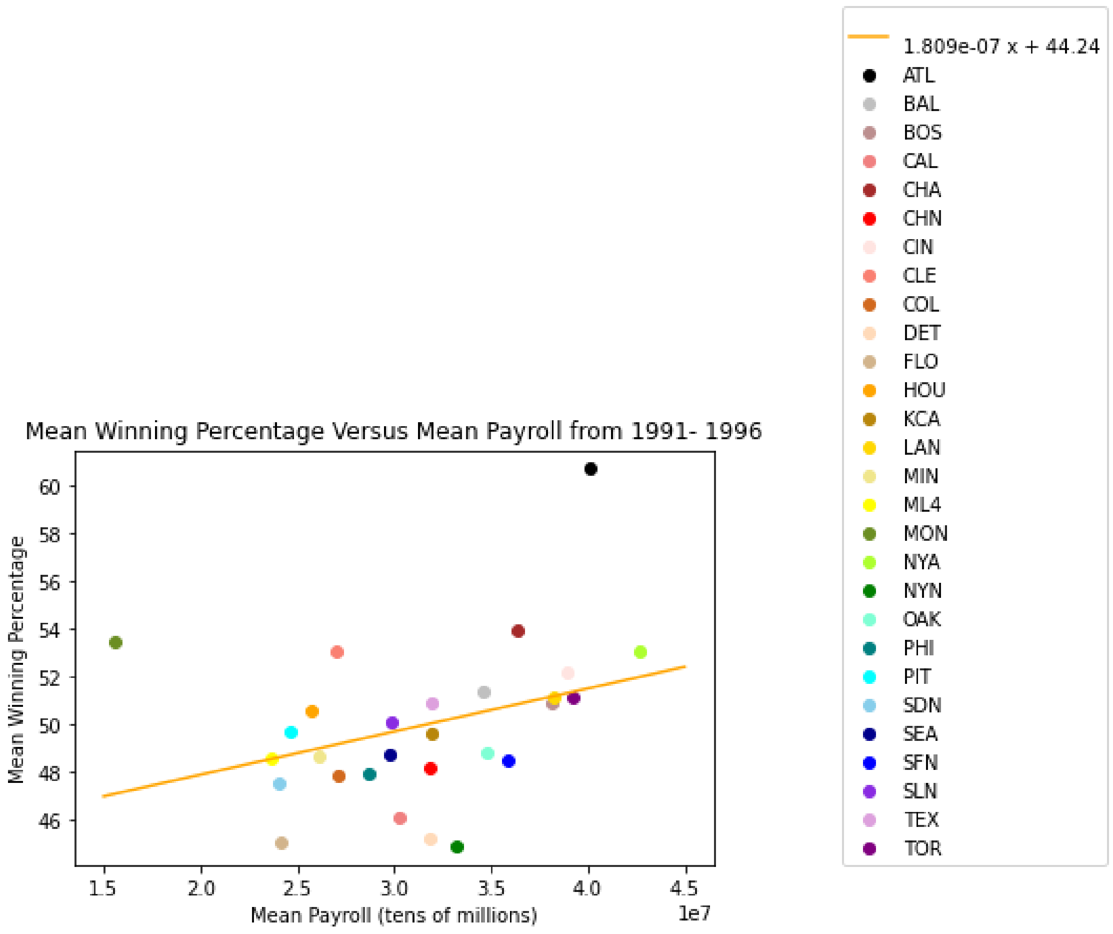
```
Distances from regression
[('NYN', 6.832389235043038),
 ('TOR', 5.521909515970691),
 ('OAK', 5.492092181839382),
 ('CIN', 4.054945308742269),
 ('MON', 2.3907910896963998),
 ('BOS', 1.4576893061659248),
 ('SFN', 1.3988678972435977),
 ('SLN', 1.1923055683998598),
 ('HOU', 0.8882791961956471),
 ('TEX', 0.6657997775683668),
 ('CHA', 0.6609197553846613),
 ('MIN', 0.5449771584652225),
 ('ML4', 0.4005532635429745),
 ('DET', 0.36401949460252325),
 ('CAL', -0.04533108024064347),
 ('KCA', -0.18823423290141506),
 ('SEA', -1.1665956639222728),
 ('LAN', -1.438371679318145),
 ('PIT', -1.5933822069451438),
 ('SDN', -1.6559261215171404),
 ('NYA', -1.9596687383832858),
 ('CHN', -2.3141410373960056),
 ('PHI', -3.183858485676801),
 ('CLE', -3.185830774530558),
 ('BAL', -3.98417586336555),
 ('ATL', -11.150022864663825)]
```



This is a scatter plot of the avg mean payroll and avg winning percentage for each team, along with a regression of the plot for time period 1.

[59]:
```
# Second plot: (1990.8, 1996.6]
# This is essentially 1991 - 1996 (no partial years)
# We know this because in part 1 we saw years
# are represented by integers.

# Grabbing a dataframe from our time period, that contains the data we want to␣
↪plot
plot_df2 = pd.DataFrame(data=[corr_2.groupby(['teamID']).
↪mean()['total_payroll'],
                        corr_2.groupby(['teamID']).
↪mean()['winning_percentage']]).transpose()
# Period 2 Plot
fig2 = plt.figure()
win_pay2 = fig2.add_subplot(111)
win_pay2.set_prop_cycle('color', colors)

# Linear Regression
p2 = np.poly1d(np.polyfit(plot_df2['total_payroll'],␣
↪plot_df2['winning_percentage'], 1))
xp2 = np.linspace(1.5*(10 ** 7), 4.5*(10 ** 7), 100)
win_pay2.plot(xp2, p2(xp2), label=p2, color='orange')
dis2 = list() # Get distances

# Scatter Plot
for key,data in plot_df2.groupby('teamID'):
    win_pay2.scatter(data['total_payroll'], data['winning_percentage'],␣
↪label=key)
    dis2.append((key, float(data['winning_percentage']) -␣
↪p2(data['total_payroll'])[0]))

# Plotting (legend, labels, etc)
win_pay2.legend(loc=(1.2,0))
win_pay2.set_xlabel("Mean Payroll (tens of millions)\n\n \
This is a scatter plot of the avg mean payroll and avg winning percentage\n \
for each team, along with a regression of the plot for time period 2.")
win_pay2.set_ylabel("Mean Winning Percentage")
win_pay2.set_title('Mean Winning Percentage Versus Mean Payroll from 1991-␣
↪1996')

fig2.show()
dis2.sort(key=lambda x:-x[1]) # reverse sort
print('Distances from regression')
pprint(dis2)
```

```
Distances from regression
[('ATL', 9.18403697753287),
 ('MON', 6.37576560777778),
 ('CLE', 3.9338749376025746),
 ('CHA', 3.0999034947703237),
 ('HOU', 1.6746454416372742),
 ('NYA', 1.0816051750090168),
 ('PIT', 0.9469557208677912),
 ('BAL', 0.8856104557582825),
 ('TEX', 0.8778300904712566),
 ('CIN', 0.834082775537297),
 ('SLN', 0.44408819995261695),
 ('ML4', 0.024105625693408683),
 ('LAN', -0.008930144910266336),
 ('BOS', -0.22274703387276418),
 ('TOR', -0.26139259426799555),
 ('MIN', -0.2977347981651377),
 ('KCA', -0.39715376635487587),
 ('SEA', -0.8917349005461261),
 ('SDN', -1.071234746132589),
 ('COL', -1.3070889246629918),
 ('PHI', -1.508924916291022),
 ('OAK', -1.789180299364702),
 ('CHN', -1.874747327979712),
 ('SFN', -2.2910747142644254),
 ('FLO', -3.579626472500344),
 ('CAL', -3.6416406520097198),
 ('DET', -4.822422991397296),
 ('NYN', -5.396870219890403)]
```

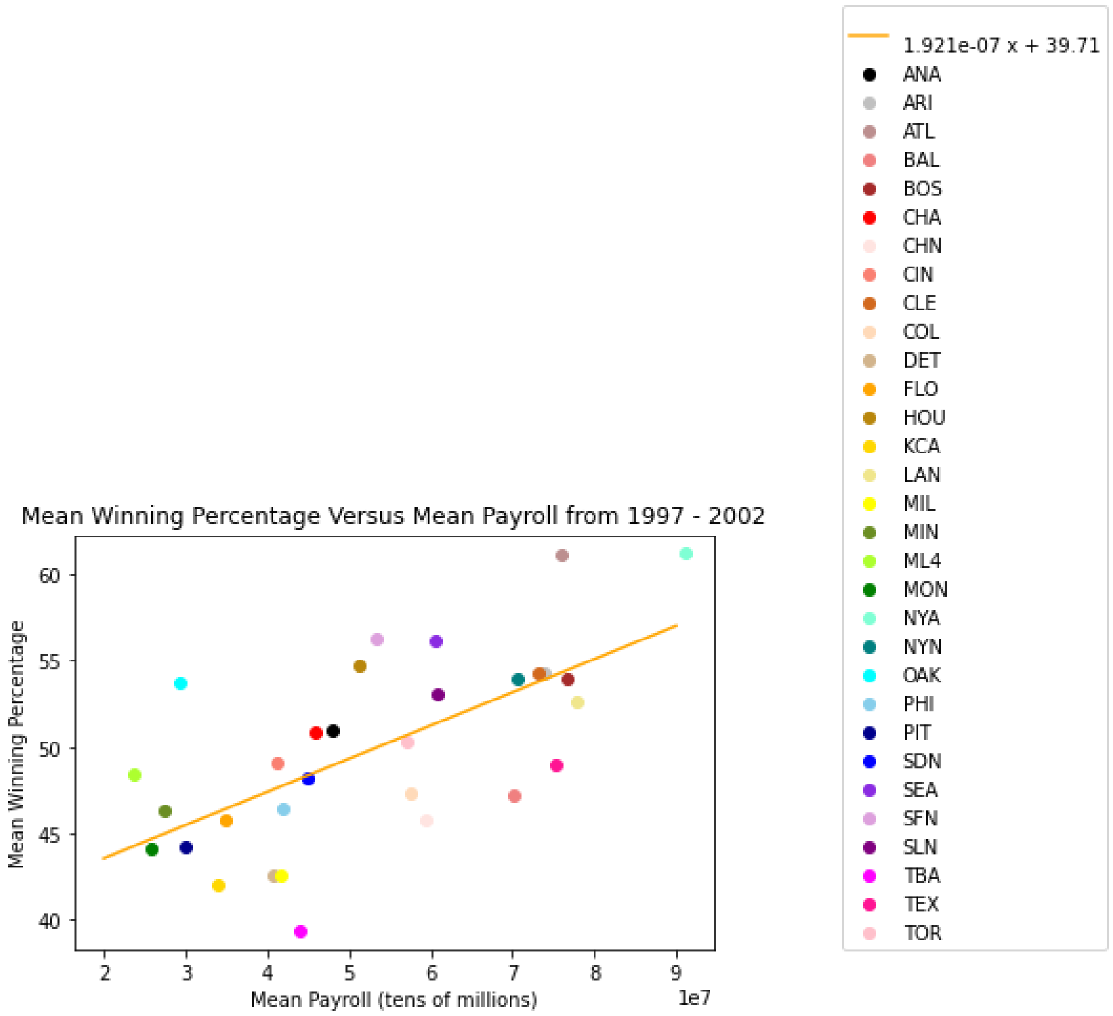


This is a scatter plot of the avg mean payroll and avg winning percentage
for each team, along with a regression of the plot for time period 2.

[60]:
```
# Third plot: (1996.6, 2002.4]
# This is essentially 1997 - 2002 (no partial years)
# We know this because in part 1 we saw years
# are represented by integers.

# Grabbing a dataframe from our time period, that contains the data we want to␣
↪plot
plot_df3 = pd.DataFrame(data=[corr_3.groupby(['teamID']).
↪mean()['total_payroll'],
                              corr_3.groupby(['teamID']).
↪mean()['winning_percentage']]).transpose()
# Period 3 Plot
fig3 = plt.figure()
win_pay3 = fig3.add_subplot(111)
win_pay3.set_prop_cycle('color', colors)
```

```
# Linear Regression
p3 = np.poly1d(np.polyfit(plot_df3['total_payroll'],␣
↪plot_df3['winning_percentage'], 1))
xp3 = np.linspace(2*(10 ** 7), 9*(10 ** 7), 100)
win_pay3.plot(xp3, p3(xp3), label=p3, color='orange')
dis3 = list() # Get distances

# Scatter Plot
for key,data in plot_df3.groupby('teamID'):
    win_pay3.scatter(data['total_payroll'], data['winning_percentage'],␣
↪label=key)
    dis3.append((key, float(data['winning_percentage']) -␣
↪p3(data['total_payroll'])[0]))

# Plotting (legend, labels, etc)
win_pay3.legend(loc=(1.2,0))
win_pay3.set_xlabel("Mean Payroll (tens of millions)\n\n \
This is a scatter plot of the avg mean payroll and avg winning percentage\n \
for each team, along with a regression of the plot for time period 3.")
win_pay3.set_ylabel("Mean Winning Percentage")
win_pay3.set_title('Mean Winning Percentage Versus Mean Payroll from 1997 -␣
↪2002')

fig3.show()
dis3.sort(key=lambda x:-x[1]) # reverse sort
print('Distances from regression')
pprint(dis3)
```

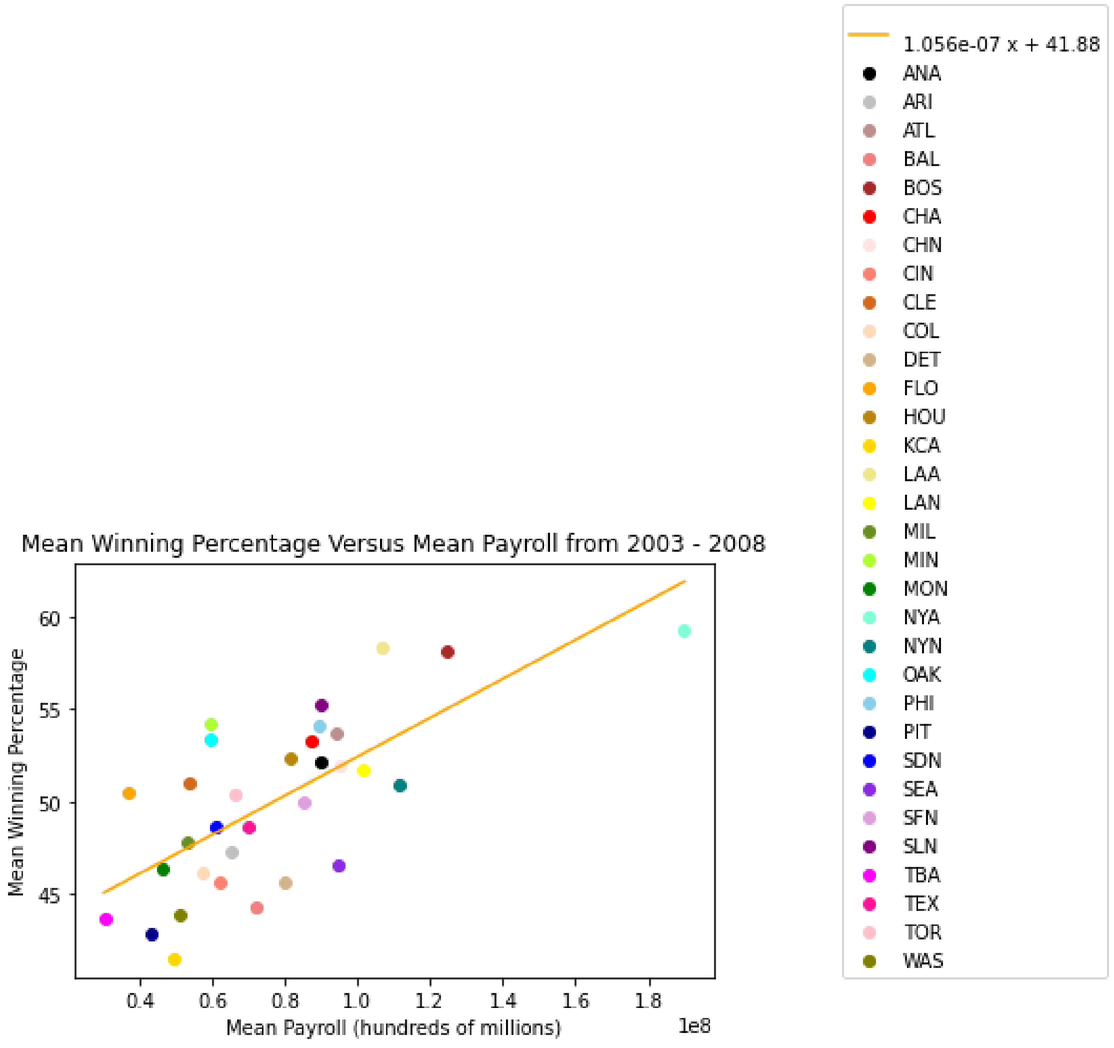
```
Distances from regression
[('OAK', 8.431416542900166),
 ('ATL', 6.865201427603132),
 ('SFN', 6.2605360701268395),
 ('HOU', 5.178785267851481),
 ('SEA', 4.76323500908029),
 ('ML4', 4.193038462670216),
 ('NYA', 3.985257096013598),
 ('CHA', 2.2909159806131854),
 ('ANA', 1.9771485656133265),
 ('SLN', 1.69819182558971),
 ('CIN', 1.42330521599213),
 ('MIN', 1.3222222151197585),
 ('NYN', 0.5996370174270282),
 ('CLE', 0.5206598282615715),
 ('ARI', 0.39389468801468297),
 ('SDN', -0.08596686296518641),
 ('TOR', -0.4241515198329395),
 ('BOS', -0.4833010262883235),
 ('MON', -0.5261118021476392),
 ('FLO', -0.6633611349638713),
 ('PIT', -1.2706731377342138),
 ('PHI', -1.3150944370945368),
 ('LAN', -1.9876663259951783),
 ('COL', -3.4477377438656163),
 ('KCA', -4.235081684393386),
 ('DET', -4.971715318493629),
 ('MIL', -5.107086105795169),
 ('TEX', -5.202752403115213),
 ('CHN', -5.393574824098259),
 ('BAL', -5.969589722954453),
 ('TBA', -8.819581163138864)]
```



This is a scatter plot of the avg mean payroll and avg winning percentage for each team, along with a regression of the plot for time period 3.

```
[61]: # Fourth plot: (2002.4, 2008.2]
      # This is essentially 2003 - 2008 (no partial years)
      # We know this because in part 1 we saw years
      # are represented by integers.

      # Grabbing a dataframe from our time period, that contains the data we want to␣
       ↪plot
      plot_df4 = pd.DataFrame(data=[corr_4.groupby(['teamID']).
       ↪mean()['total_payroll'],
                                 corr_4.groupby(['teamID']).
       ↪mean()['winning_percentage']]).transpose()
      # Period 4 Plot
      fig4 = plt.figure()
      win_pay4 = fig4.add_subplot(111)
      win_pay4.set_prop_cycle('color', colors)

      # Linear Regression
      p4 = np.poly1d(np.polyfit(plot_df4['total_payroll'],␣
       ↪plot_df4['winning_percentage'], 1))
      xp4 = np.linspace(0.3*(10 ** 8), 1.9*(10 ** 8), 100)
      win_pay4.plot(xp4, p4(xp4), label=p4, color='orange')
      dis4 = list() # Get distances

      # Scatter Plot
      for key,data in plot_df4.groupby('teamID'):
          win_pay4.scatter(data['total_payroll'], data['winning_percentage'],␣
       ↪label=key)
          dis4.append((key, float(data['winning_percentage']) -␣
       ↪p4(data['total_payroll'])[0]))

      # Plotting (legend, labels, etc)
      win_pay4.legend(loc=(1.2,0))
      win_pay4.set_xlabel("Mean Payroll (hundreds of millions)\n\n \
      This is a scatter plot of the avg mean payroll and avg winning percentage\n \
      for each team, along with a regression of the plot for time period 4.")
      win_pay4.set_ylabel("Mean Winning Percentage")
      win_pay4.set_title('Mean Winning Percentage Versus Mean Payroll from 2003 -␣
       ↪2008')

      fig4.show()
      dis4.sort(key=lambda x:-x[1]) # reverse sort
      print('Distances from regression')
      pprint(dis4)
```

```
Distances from regression
[('MIN', 6.101441264455353),
 ('OAK', 5.3204459118896),
('LAA', 5.183635883429311),
('FLO', 4.731871574029867),
('SLN', 3.9512072810526604),
('CLE', 3.4979968468267515),
('BOS', 3.0929678640891893),
('PHI', 2.8214488655967287),
('CHA', 2.235093263751004),
('ATL', 1.9187944826378285),
('HOU', 1.8847458178128136),
('TOR', 1.4644611023312137),
('ANA', 0.7981126490625172),
('MIL', 0.3102325442737879),
('SDN', 0.2922623670221185),
('CHN', 0.002067658756544688),
('MON', -0.48545462509305537),
('TEX', -0.5861514197629276),
('LAN', -0.816047326462062),
('SFN', -0.8816670222783287),
('TBA', -1.4110929156643408),
('ARI', -1.431850597570886),
('COL', -1.8046492176444744),
('NYA', -2.599442288415908),
('CIN', -2.7841027723895735),
('NYN', -2.790535013176523),
('WAS', -3.37186626154584),
('PIT', -3.5954749069032275),
('DET', -4.75609459239201),
('BAL', -5.252965402551517),
('SEA', -5.381795763118639),
('KCA', -5.657595252048019)]
```

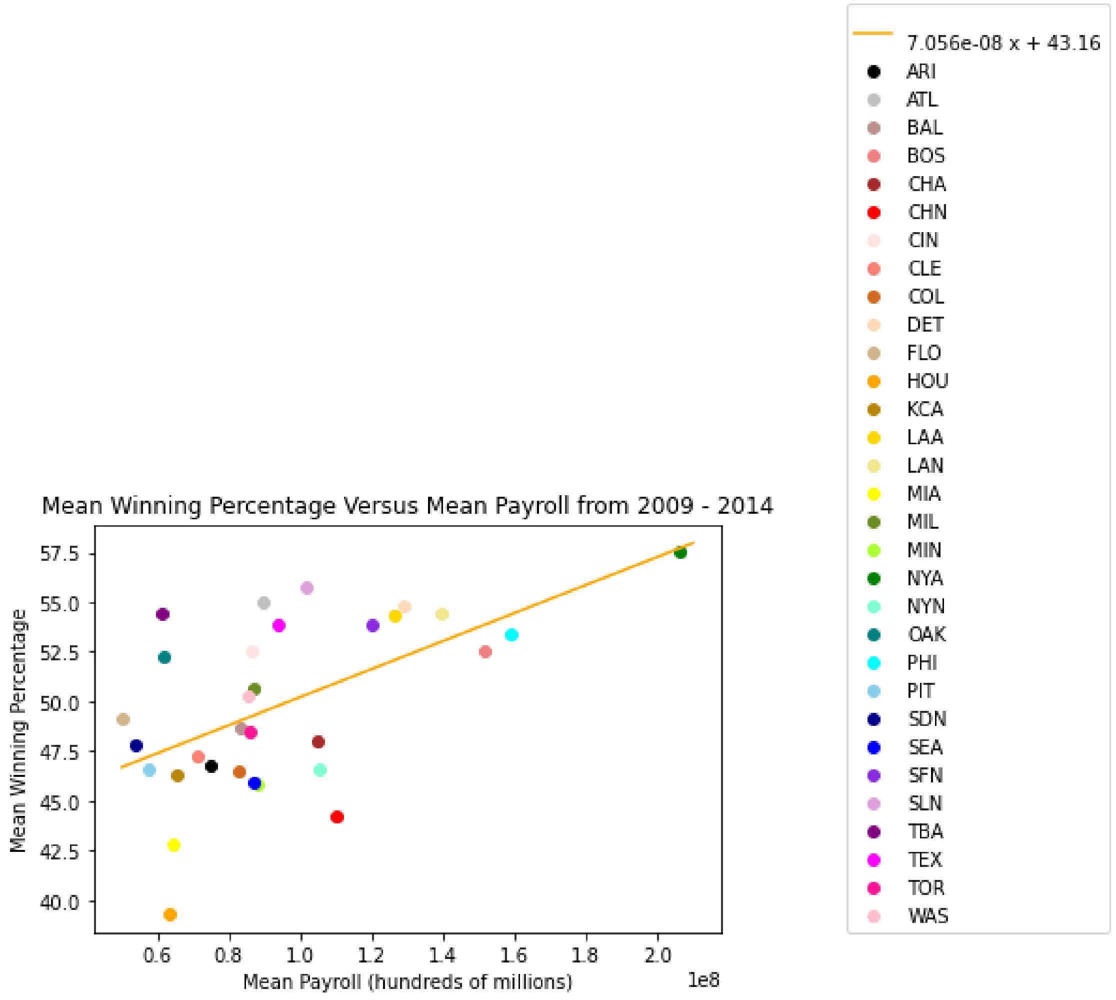


This is a scatter plot of the avg mean payroll and avg winning percentage for each team, along with a regression of the plot for time period 4.

[62]:
```
# Fifth plot: (2008.2, 2014.0]
# This is essentially 2009 - 2014 (no partial years)
# We know this because in part 1 we saw years
# are represented by integers.

# Grabbing a dataframe from our time period, that contains the data we want to␣
↪plot
plot_df5 = pd.DataFrame(data=[corr_5.groupby(['teamID']).
↪mean()['total_payroll'],
                              corr_5.groupby(['teamID']).
↪mean()['winning_percentage']]).transpose()
# Period 5 Plot
fig5 = plt.figure()
```

```
win_pay5 = fig5.add_subplot(111)
win_pay5.set_prop_cycle('color', colors)

# Linear Regression
p5 = np.poly1d(np.polyfit(plot_df5['total_payroll'],␣
 ↪plot_df5['winning_percentage'], 1))
xp5 = np.linspace(0.5*(10 ** 8), 2.1*(10 ** 8), 100)
win_pay5.plot(xp5, p5(xp5), label=p5, color='orange')
dis5 = list() # Get distances

for key,data in plot_df5.groupby('teamID'):
    win_pay5.scatter(data['total_payroll'], data['winning_percentage'],␣
 ↪label=key)
    dis5.append((key, float(data['winning_percentage']) -␣
 ↪p5(data['total_payroll'])[0]))

# Plotting (legend, labels, etc)
win_pay5.legend(loc=(1.2,0))
win_pay5.set_xlabel("Mean Payroll (hundreds of millions)\n\n \
This is a scatter plot of the avg mean payroll and avg winning percentage\n \
for each team, along with a regression of the plot for time period 5.")
win_pay5.set_ylabel("Mean Winning Percentage")
win_pay5.set_title('Mean Winning Percentage Versus Mean Payroll from 2009 -␣
 ↪2014')

fig5.show()
dis5.sort(key=lambda x:-x[1]) # reverse sort
print('Distances from regression')
pprint(dis5)
```

```
Distances from regression
[('TBA', 6.998558082412082),
 ('ATL', 5.567613477754648),
 ('SLN', 5.414066489368601),
 ('OAK', 4.7284893286309),
 ('TEX', 4.089726042711007),
 ('CIN', 3.3145159975943486),
 ('DET', 2.504727323209302),
 ('FLO', 2.4652785918009457),
 ('SFN', 2.2636423704207758),
 ('LAA', 2.2437184259513714),
 ('LAN', 1.4861945192514554),
 ('MIL', 1.306705590844949),
 ('WAS', 1.07424419995725 63),
 ('SDN', 0.8702502638191802),
 ('NYA', -0.19117866433857245),
 ('BAL', -0.37254863644695746),
('PIT', -0.6905027710537652),
('TOR', -0.7826553905099161),
('CLE', -0.9747512494298434),
('PHI', -0.9800600024256028),
('BOS', -1.2860836057664855),
('KCA', -1.492889475338508),
('ARI', -1.63745254327619),
('COL', -2.4892553064611462),
('CHA', -2.507152828922983),
('SEA', -3.395615218855916),
('MIN', -3.5533281787009443),
('NYN', -3.980419880785533),
('MIA', -4.917859974180537),
('CHN', -6.746373453312756),
('HOU', -8.329603523920575)]
```



This is a scatter plot of the avg mean payroll and avg winning percentage for each team, along with a regression of the plot for time period 5.

4.2.2 End of Problem 4

4.2.3 Question 2: What can you say about team payrolls across these periods? Are there any teams that standout as being particularly good at paying for wins across these time periods? What can you say about the Oakland A's spending efficiency across these time periods.

Across the periods:
One thing to note about the payrolls is that the payrolls for all teams grow across these time periods as shown by the shift from having to display the mean payrolls from tens of millions to hundreds of millions. Across all time periods, there is a trend that as mean payroll increases, winning percentage increases as well.

Teams that stand out in terms of paying for wins:
I will be doing which has the greatest positive distance from our regression (so doing *better* than predicted):
- Time Period 1 1984-1990 (This is a scatter plot with linear regression of the mean winning percentage vs mean payroll for (1984.971, 1990.8]): >Teams that stand out in time period 1 are NYN (New York Mets), TOR (Toronto Blue Jays), OAK (Oakland Atheletics). They seem to stand out on the graph and the distances that I calculated. They are well above the winning percentage regression line and seem to just be doing better than others, their distance from regression is about 5-6 positive percentage points, and seem to form their own "group." - Time Period 2 1991-1996 (This is a scatter plot with linear regression of the mean winning percentage vs mean payroll for (1990.8, 1996.6]): >Teams that stand out in time period 2 are ATL (Atlanta Braves) and MON (Montreal Expos). The Atlanta Braves really stand out at 9 percentage points above the regression. Montral Expos also stand out at 6 percentage points above the winning percentage regression. However, other teams do not seem to come close to ATL or MON, the closest is 3 positive percentage points.
- Time Period 3 1997-2002 (This is a scatter plot with linear regression of the mean winning percentage vs mean payroll for (1996.6, 2002.4]): >The team that stands out in time period 3 is OAK (Oakland Athletics). They are about 8.4 percentage points above the winning percentage regression. Other teams come close like the Atlanta Braves, but only about 6 percentage points, it is just clear that the Oakland A's are really good at spending efficiently relative to their peers during this time period from the graph and distances calculated. - Time Period 4 2003-2008 (This is a scatter plot with linear regression of the mean winning percentage vs mean payroll for (2002.4, 2008.2]): >Teams that stand out in time period 4 are MIN (Minnesota Twins), OAK (Oakland Athletics), LAA (Los Angeles of Anaheim), and FLO (Florida Marlins). In this time period, it looks like from the graph that these teams are a bit higher above the regression than everyone else. Maybe we are seeing that other teams are "catching" up in terms of efficiency spending as suggested in the readme. - Time Period 5 2009-2014 (This is a scatter plot with linear regression of the mean winning percentage vs mean payroll for (2008.2, 2014.0]): >Teams that stand out in time period 5 are TBA (Tampa Bay), ATL (Atlanta Braves), SLN (St. Louis Cardinals), and OAK (Oakland Athletics). Like the time period 4, this upper group is not that much higher and distinct like other time periods (ie. time period 3). Maybe we are seeing that other teams are "catching" up in terms of efficiency spending as suggested in the readme.

Oakland Athletics' spending efficiency is pretty good over all the time periods (except they didn't seem to standout in time period 2). They were really the best in Time Period 3 and seemed great in

Time Period 4 in terms of spending efficiency. Particularly in time Period 3, looking at the graph, they were reaching a winning percentage that matched teams that spent ~80 million (Oakland Atheltics spent ~30 million during on avg during this time period)!

5 Part 3: Data Transformations

In this part we will learn about standardizing. We can tell there is a problem comparing payroll year to payroll year already because of the sheer total payroll difference highlighted many times earlier. With our five period graphs earlier, we tried to mitigate this, but we can do better by standardizing!

5.1 Standardizing across years

Let's do transformations!

5.1.1 Problem 5: Create a new variable in your dataset that standardizes payroll on year. So, this column for team i in year j should equal: stn_pay (ij) = payroll (ij) - avgpayroll(j) / s(j)

Short prose explaining my code:
For this problem, I have to create a new variable that is a standardized payroll. However, if we look closely at the equation it requires the average payroll of the year and the standard deviation of the payroll for each year. I decided that I wanted to calculate these things beforehand to make my loops better. So, I created a dataframe called std_calcs, from my moneyball_df dataframe, by grouping by year and then calculating the mean and standard deviation for the year. Then I just went through the rows in moneyball, and calculated the std_payroll for that team, year using values from the std_calcs dataframe for mean and standard deviation of payrolls for a particular year.

[63]:
```
# Here is what I plan to do,
# calculate the average payroll and standard deviation per year
# and store it in a separate table.
# Then for each grab the year and
# then do the calculation with the team/year payroll.

# Getting the data wanted to calculate mean and stdev
d = {'mean': moneyball_df.groupby(['yearID']).mean()['total_payroll'],
    'stdev':moneyball_df.groupby(['yearID']).std()['total_payroll']}

# Create a dataframe of all the years along with
# their mean and stdevs.
std_calcs = pd.DataFrame(data=d)

# Here is a printout of the data we just calulated
std_calcs.head()
```

[63]:
```
                mean         stdev
yearID
```

```
1985    1.007557e+07  2.470845e+06
1986    1.184056e+07  3.186956e+06
1987    1.048367e+07  3.848337e+06
1988    1.155586e+07  3.386331e+06
1989    1.384599e+07  3.568844e+06
```

[64]:
```
# Now, let's go through and calculate the standardized payroll in
# our moneyball_df!
std_vals = list()

for row in moneyball_df.iterrows():
    pay = moneyball_df.at[row[0], 'total_payroll']
    year = moneyball_df.at[row[0], 'yearID']
    std = (pay - std_calcs.at[year, 'mean']) / std_calcs.at[year, 'stdev']
    std_vals.append(std)
```

[65]:
```
# Create the new dataset! Problem 5 now complete!
moneyball_df = moneyball_df.assign(std_payroll = std_vals)
moneyball_df
```

[65]:
```
     yearID teamID  total_payroll    G   W   L                   name  \
0      1985    ATL     14807000.0  162  66  96         Atlanta Braves
1      1985    BAL     11560712.0  161  83  78      Baltimore Orioles
2      1985    BOS     10897560.0  163  81  81         Boston Red Sox
3      1985    CAL     14427894.0  162  90  72      California Angels
4      1985    CHA      9846178.0  163  85  77      Chicago White Sox
..      ...    ...            ...  ...  ..  ..                    ...
853    2014    SLN    120693000.0  162  90  72    St. Louis Cardinals
854    2014    TBA     72689100.0  162  77  85         Tampa Bay Rays
855    2014    TEX    112255059.0  162  67  95         Texas Rangers
856    2014    TOR    109920100.0  162  83  79      Toronto Blue Jays
857    2014    WAS    131983680.0  162  96  66   Washington Nationals

     winning_percentage  std_payroll
0             40.740741     1.914905
1             51.552795     0.601068
2             49.693252     0.332678
3             55.555556     1.761474
4             52.147239    -0.092838
..                  ...          ...
853           55.555556     0.336014
854           47.530864    -0.793346
855           41.358025     0.137500
856           51.234568     0.082566
857           59.259259     0.601644

[858 rows x 9 columns]
```

5.1.2 End of Problem 5

5.1.3 Problem 6: Repeat the same plots as Problem 4, but use this new standardized payroll variable.

Short prose explaining my code:
Like problem 4, I grab a dataframe from moneyball_df, which is then separate by the time period bins. I also drop what I don't need, and end up keeping the winning_percentage and the std_payroll, as well as the teamID only for purposes of plotting. After sorting by the bins and getting 5 dataframes for each time period, I drop the time_period column know that I know everything in that sub dataframe is in the same time period. Then, I calculate a regression and plot it, as well as get the avg winning percentage distance from the regression. I also plot the scatter plot of the mean winning percentage vs the mean standard payroll value for each team as asked for in the problem.

[66]:
```
# To repeat the same plots, let's get our sub_dfs again for each time period

# Grab correlation df for plotting purposes from main df 'moneyball_df'

# Drop things I don't need (making sure to keep std_payroll and␣
↪winning_percentage)
corr_df = moneyball_df.drop(['G', 'W', 'L', 'name', 'total_payroll'], axis=1)
# Get time_period bins again
corr_df['time_period'] = pd.cut(corr_df['yearID'], bins=5)
```

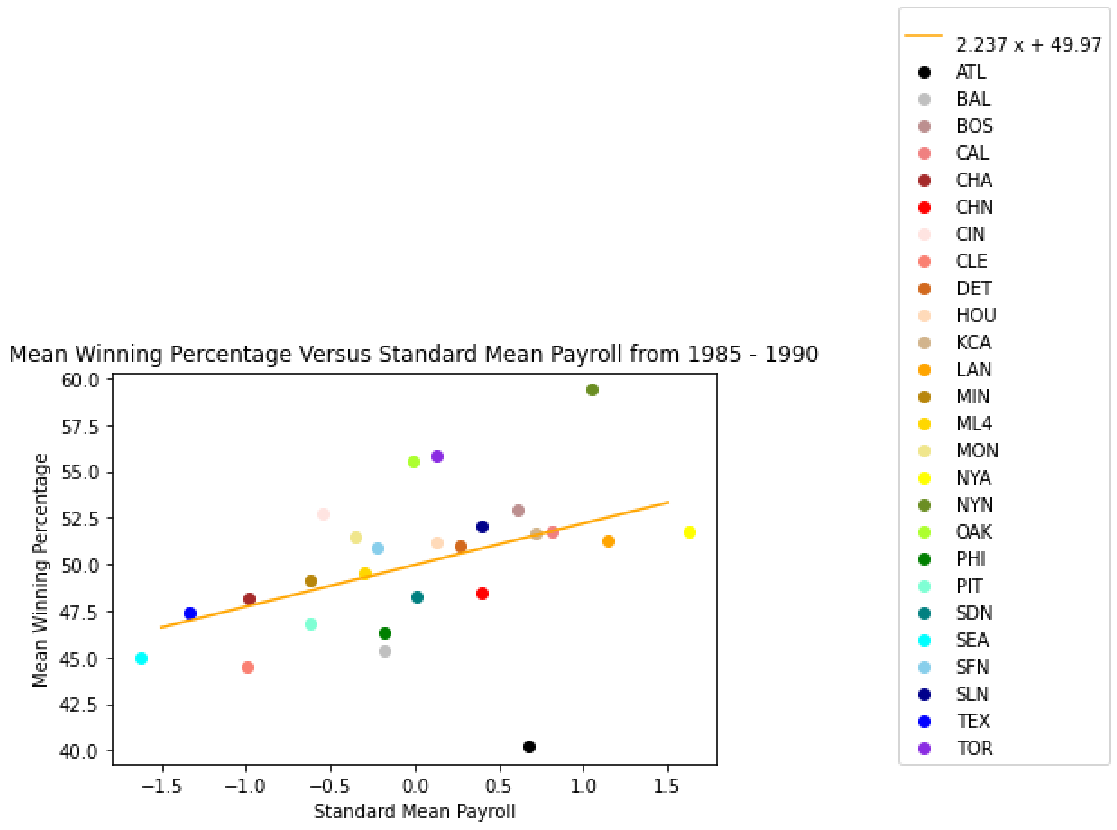
[67]:
```
# Print what I got within the time period
# After printing, I drop some extra rows
# not needed for producing the graphs (year, time_period)
# Grab the first period
corr_1 = corr_df[corr_df['time_period'] == pd.Interval(cuts[1][0], cuts[1][1],␣
↪closed='right')]
print(corr_1.head()) # print here just to see year and time_period, drop later␣
↪bc not needed
corr_1 = corr_1.drop(['yearID', 'time_period'], axis=1)

# Grab the second period
corr_2 = corr_df[corr_df['time_period'] == pd.Interval(cuts[1][1], cuts[1][2],␣
↪closed='right')]
print(corr_2.head())
corr_2 = corr_2.drop(['yearID', 'time_period'], axis=1)

# Grab the third period
corr_3 = corr_df[corr_df['time_period'] == pd.Interval(cuts[1][2], cuts[1][3],␣
↪closed='right')]
print(corr_3.head())
corr_3 = corr_3.drop(['yearID', 'time_period'], axis=1)

# Grab the fourth period
```

```
corr_4 = corr_df[corr_df['time_period'] == pd.Interval(cuts[1][3], cuts[1][4],␣
↪closed='right')]
print(corr_4.head())
corr_4 = corr_4.drop(['yearID', 'time_period'], axis=1)

# Grab the fifth period
corr_5 = corr_df[corr_df['time_period'] == pd.Interval(cuts[1][4], cuts[1][5],␣
↪closed='right')]
print(corr_5.head())
corr_5 = corr_5.drop(['yearID', 'time_period'], axis=1)
```

```
   yearID teamID  winning_percentage  std_payroll         time_period
0    1985    ATL           40.740741     1.914905  (1984.971, 1990.8]
1    1985    BAL           51.552795     0.601068  (1984.971, 1990.8]
2    1985    BOS           49.693252     0.332678  (1984.971, 1990.8]
3    1985    CAL           55.555556     1.761474  (1984.971, 1990.8]
4    1985    CHA           52.147239    -0.092838  (1984.971, 1990.8]
     yearID teamID  winning_percentage  std_payroll     time_period
156    1991    ATL           58.024691    -0.750621  (1990.8, 1996.6]
157    1991    BAL           41.358025    -0.878909  (1990.8, 1996.6]
158    1991    BOS           51.851852     1.680823  (1990.8, 1996.6]
159    1991    CAL           50.000000     1.375152  (1990.8, 1996.6]
160    1991    CHA           53.703704    -0.965836  (1990.8, 1996.6]
     yearID teamID  winning_percentage  std_payroll     time_period
320    1997    ANA           51.851852    -0.698639  (1996.6, 2002.4]
321    1997    ATL           62.345679     0.920185  (1996.6, 2002.4]
322    1997    BAL           60.493827     1.397793  (1996.6, 2002.4]
323    1997    BOS           48.148148     0.252554  (1996.6, 2002.4]
324    1997    CHA           49.689441     1.338347  (1996.6, 2002.4]
     yearID teamID  winning_percentage  std_payroll     time_period
498    2003    ANA           47.530864     0.288791  (2002.4, 2008.2]
499    2003    ARI           51.851852     0.346814  (2002.4, 2008.2]
500    2003    ATL           62.345679     1.260233  (2002.4, 2008.2]
501    2003    BAL           43.558282     0.104792  (2002.4, 2008.2]
502    2003    BOS           58.641975     1.035430  (2002.4, 2008.2]
     yearID teamID  winning_percentage  std_payroll     time_period
678    2009    ARI           43.209877    -0.463967  (2008.2, 2014.0]
679    2009    ATL           53.086420     0.233391  (2008.2, 2014.0]
680    2009    BAL           39.506173    -0.641596  (2008.2, 2014.0]
681    2009    BOS           58.641975     0.960560  (2008.2, 2014.0]
682    2009    CHA           48.765432     0.213966  (2008.2, 2014.0]
```

[68]:
```
# First plot: (1984.971, 1990.8]
# This is essentially 1985 - 1990 (no partial years)
# We know this because in part 1 we saw years
# are represented by integers.
# Grabbing a dataframe from our time period, that contains the data we want to␣
↪plot
plot_df1 = pd.DataFrame(data=[corr_1.groupby(['teamID']).mean()['std_payroll'],
                        corr_1.groupby(['teamID']).
↪mean()['winning_percentage']]).transpose()
# Period 1 Plot
fig1 = plt.figure()
win_pay1 = fig1.add_subplot(111)
win_pay1.set_prop_cycle('color', colors)

# Linear Regression
p1 = np.poly1d(np.polyfit(plot_df1['std_payroll'],␣
↪plot_df1['winning_percentage'], 1))
xp1 = np.linspace(-1.5, 1.5, 100)
win_pay1.plot(xp1, p1(xp1), label=p1, color='orange')
dis1 = list() # Get distances

# Scatter Plot
for key,data in plot_df1.groupby('teamID'):
    win_pay1.scatter(data['std_payroll'], data['winning_percentage'], label=key)
    dis1.append((key, float(data['winning_percentage']) -␣
↪p1(data['std_payroll'])[0]))

# Plotting (labels, etc)
win_pay1.legend(loc=(1.3,0))
win_pay1.set_xlabel("Standard Mean Payroll\n\n \
This is a scatter plot of the standard mean payroll and avg winning␣
↪percentage\n \
for each team, along with a regression of the plot for time period 1.")
win_pay1.set_ylabel("Mean Winning Percentage")
win_pay1.set_title('Mean Winning Percentage Versus Standard Mean Payroll from␣
↪1985 - 1990')

fig1.show()
dis1.sort(key=lambda x:-x[1]) # reverse sort
print('Distances from regression')
pprint(dis1)
```

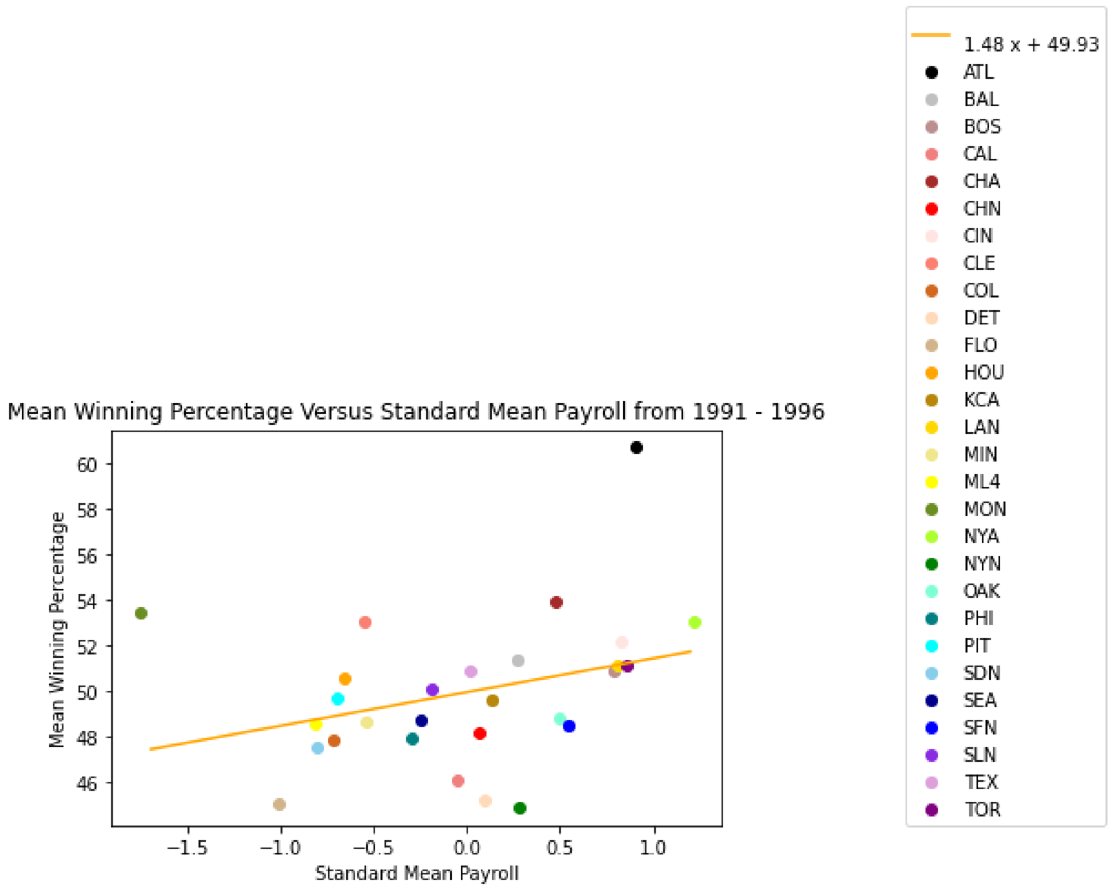
```
Distances from regression
[('NYN', 7.067242032756283),
 ('TOR', 5.613127441102137),
 ('OAK', 5.611247012889379),
 ('CIN', 3.9831136498657074),
 ('MON', 2.314883391157373),
 ('BOS', 1.5520823130320096),
 ('SFN', 1.456950408075933),
 ('SLN', 1.2612712559463048),
 ('HOU', 0.9775418643917106),
 ('MIN', 0.5876633348387941),
 ('TEX', 0.4376826093957433),
 ('DET', 0.42437862338597654),
 ('CHA', 0.40938638149054896),
 ('ML4', 0.2905161133141192),
 ('KCA', 0.06503387268575267),
 ('CAL', -0.0375014188281142),
 ('LAN', -1.192442276972912),
 ('SEA', -1.3194830338610615),
 ('SDN', -1.676829962555324),
 ('PIT', -1.7055579409150994),
 ('NYA', -1.7919514067474225),
 ('CHN', -2.415603471273606),
 ('CLE', -3.2369637083907037),
 ('PHI', -3.2617560105492416),
 ('BAL', -4.15770662275785),
 ('ATL', -11.256324451476878)]
```



This is a scatter plot of the standard mean payroll and avg winning percentage for each team, along with a regression of the plot for time period 1.

[69]:
```
# Second plot: (1990.8, 1996.6]
# This is essentially 1991 - 1996 (no partial years)
# We know this because in part 1 we saw years
# are represented by integers.

# Grabbing a dataframe from our time period, that contains the data we want to␣
↪plot
plot_df2 = pd.DataFrame(data=[corr_2.groupby(['teamID']).mean()['std_payroll'],
                        corr_2.groupby(['teamID']).
↪mean()['winning_percentage']]).transpose()
# Period 2 Plot
fig2 = plt.figure()
win_pay2 = fig2.add_subplot(111)
win_pay2.set_prop_cycle('color', colors)

# Linear Regression
p2 = np.poly1d(np.polyfit(plot_df2['std_payroll'],␣
↪plot_df2['winning_percentage'], 1))
xp2 = np.linspace(-1.7, 1.2, 100)
win_pay2.plot(xp2, p2(xp2), label=p2, color='orange')
dis2 = list() # Get distances

# Scatter Plot
for key,data in plot_df2.groupby('teamID'):
    win_pay2.scatter(data['std_payroll'], data['winning_percentage'], label=key)
    dis2.append((key, float(data['winning_percentage']) -␣
↪p2(data['std_payroll'])[0]))

# Plotting (labels, etc)
win_pay2.legend(loc=(1.3,0))
win_pay2.set_xlabel("Standard Mean Payroll\n\n \
This is a scatter plot of the standard mean payroll and avg winning␣
↪percentage\n \
for each team, along with a regression of the plot for time period 2.")
win_pay2.set_ylabel("Mean Winning Percentage")
win_pay2.set_title('Mean Winning Percentage Versus Standard Mean Payroll from␣
↪1991 - 1996')

fig2.show()
dis2.sort(key=lambda x:-x[1]) # reverse sort
print('Distances from regression')
pprint(dis2)
```

```
Distances from regression
[('ATL', 9.41720075642295),
 ('MON', 6.0991290918213465),
 ('CLE', 3.939314435500158),
('CHA', 3.2745628098455555),
('HOU', 1.6154928058068165),
('NYA', 1.3111113158508942),
('BAL', 1.0536342052476186),
('CIN', 0.9604719323298667),
('TEX', 0.9328153796482113),
('PIT', 0.7542993976437273),
('SLN', 0.44014616031908105),
('LAN', 0.020211493237383138),
('TOR', -0.12105769097895802),
('ML4', -0.18967642059972434),
('BOS', -0.1962849919165066),
('MIN', -0.46167499655542343),
('KCA', -0.5023994543514121),
('SEA', -0.8380972451664306),
('COL', -1.027861063867583),
('SDN', -1.2171074964793718),
('PHI', -1.5745921067319415),
('CHN', -1.896831760318058),
('OAK', -1.9116477912436665),
('SFN', -2.2965314165426634),
('FLO', -3.413938057307135),
('CAL', -3.778143501073579),
('DET', -4.891870434170528),
('NYN', -5.50067535637011)]
```

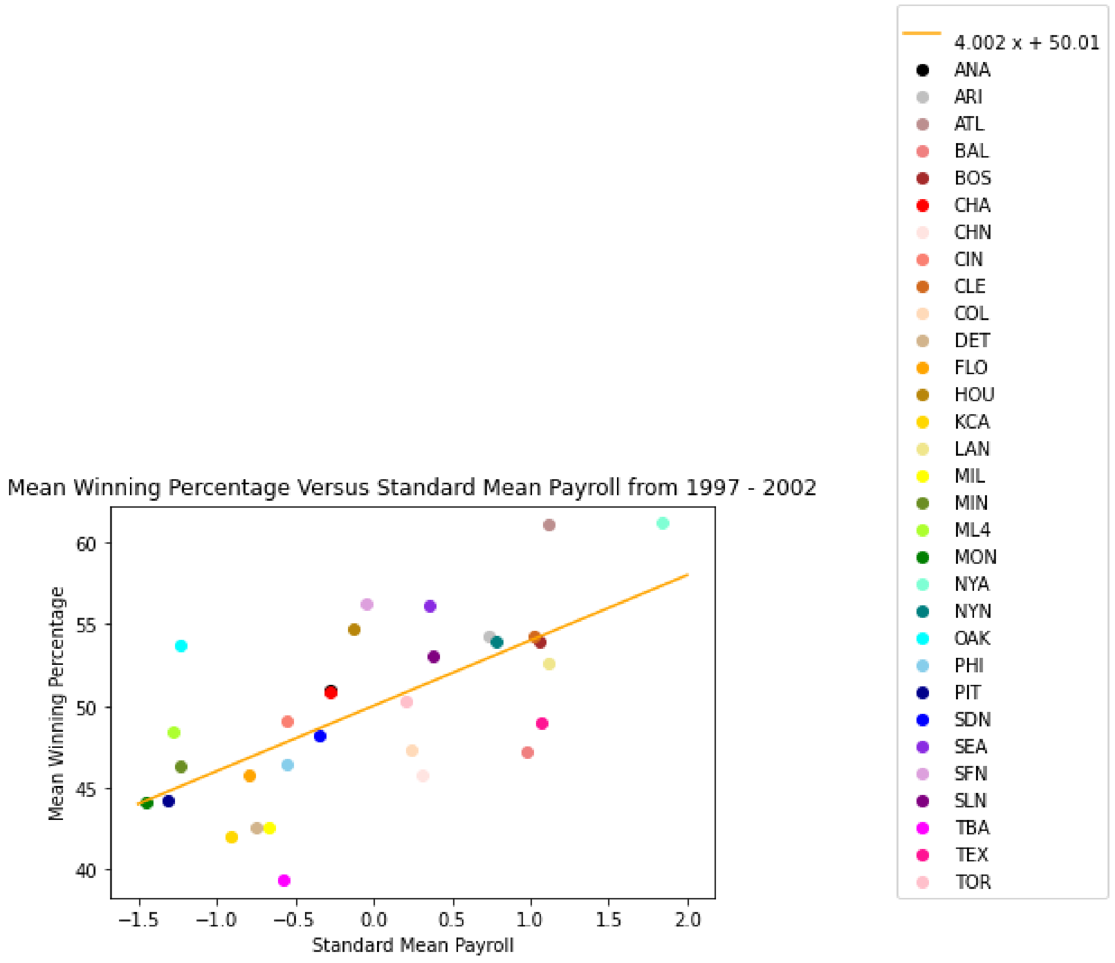


This is a scatter plot of the standard mean payroll and avg winning percentage for each team, along with a regression of the plot for time period 2.

[70]:
```
# Third plot: (1996.6, 2002.4]
# This is essentially 1997 - 2002 (no partial years)
# We know this because in part 1 we saw years
# are represented by integers.

# Grabbing a dataframe from our time period, that contains the data we want to␣
↪plot
plot_df3 = pd.DataFrame(data=[corr_3.groupby(['teamID']).mean()['std_payroll'],
                              corr_3.groupby(['teamID']).
↪mean()['winning_percentage']]).transpose()
# Period 3 Plot
fig3 = plt.figure()
win_pay3 = fig3.add_subplot(111)
win_pay3.set_prop_cycle('color', colors)

# Linear Regression
```

```
p3 = np.poly1d(np.polyfit(plot_df3['std_payroll'],␣
 ↪plot_df3['winning_percentage'], 1))
xp3 = np.linspace(-1.5, 2, 100)
win_pay3.plot(xp3, p3(xp3), label=p3, color='orange')
dis3 = list() # Get distances

# Scatter Plot
for key,data in plot_df3.groupby('teamID'):
    win_pay3.scatter(data['std_payroll'], data['winning_percentage'], label=key)
    dis3.append((key, float(data['winning_percentage']) -␣
 ↪p3(data['std_payroll'])[0]))

# Plotting (labels, etc)
win_pay3.legend(loc=(1.3,0))
win_pay3.set_xlabel("Standard Mean Payroll\n\n \
This is a scatter plot of the standard mean payroll and avg winning␣
 ↪percentage\n \
for each team, along with a regression of the plot for time period 3.")
win_pay3.set_ylabel("Mean Winning Percentage")
win_pay3.set_title('Mean Winning Percentage Versus Standard Mean Payroll from␣
 ↪1997 - 2002')

fig3.show()
dis3.sort(key=lambda x:-x[1]) # reverse sort
print('Distances from regression')
pprint(dis3)
```

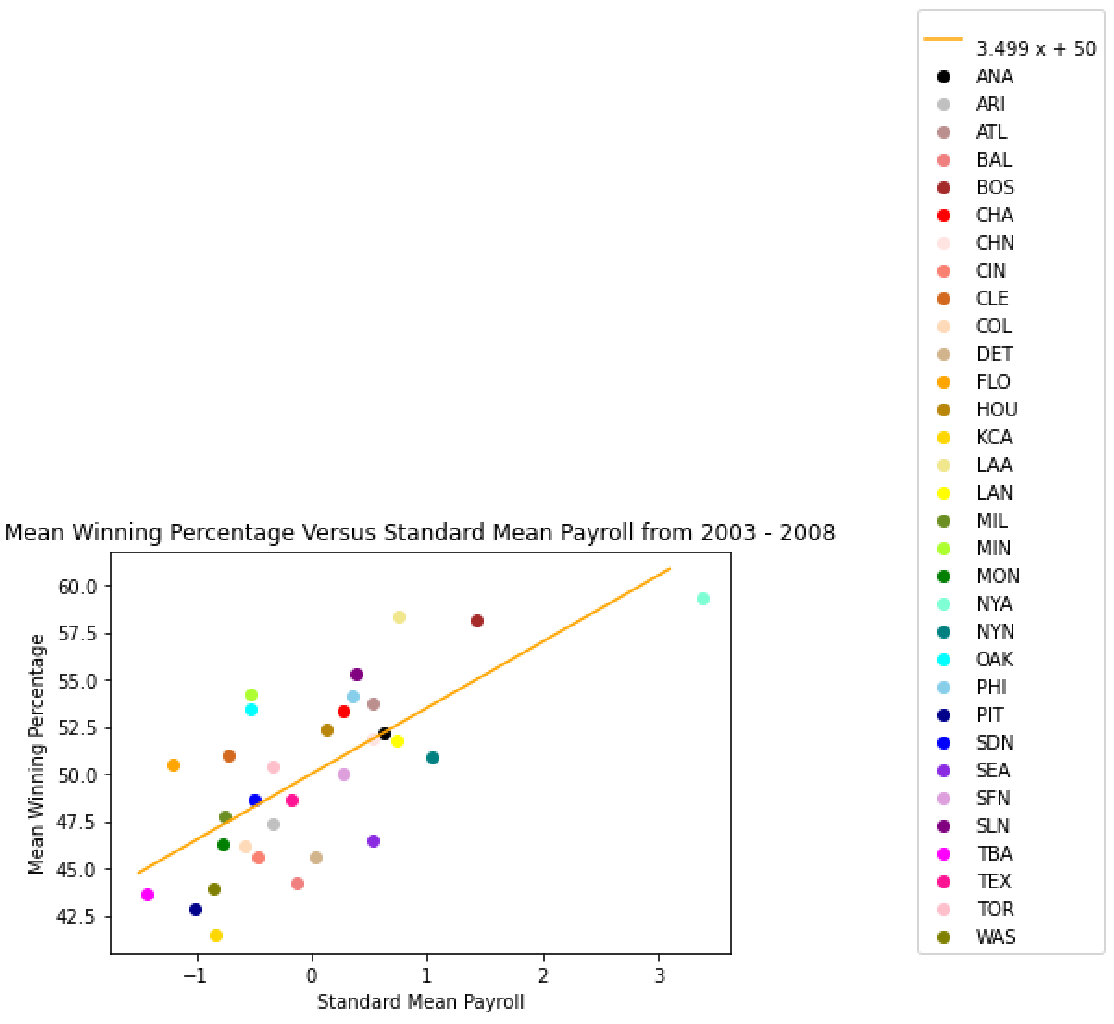
```
Distances from regression
[('OAK', 8.655927853996417),
 ('ATL', 6.682534472266589),
 ('SFN', 6.373588303863059),
 ('HOU', 5.216074445160416),
 ('SEA', 4.701247938259058),
 ('NYA', 3.8183624807476804),
 ('ML4', 3.525218101934108),
 ('ANA', 2.021729728470717),
 ('CHA', 1.9244151661107054),
 ('SLN', 1.5500456418470492),
 ('ARI', 1.3718587733494516),
 ('CIN', 1.2488914351696252),
 ('MIN', 1.2212746790827254),
 ('NYN', 0.7718823452640677),
 ('CLE', 0.18480210603508596),
 ('MON', -0.08703302085081077),
 ('BOS', -0.2827915437703652),
 ('SDN', -0.361733990126055),
 ('TOR', -0.575863582280121),
('PIT', -0.5938545749009236),
('FLO', -1.1253392612274595),
('PHI', -1.361372553616512),
('LAN', -1.802666552752484),
('COL', -3.6420593147178764),
('KCA', -4.353399283625798),
('DET', -4.457910801246008),
('MIL', -4.749433961388981),
('TEX', -5.322133366800259),
('CHN', -5.505730510157619),
('BAL', -6.688211694498612),
('TBA', -8.358319459596217)]
```



This is a scatter plot of the standard mean payroll and avg winning percentage
for each team, along with a regression of the plot for time period 3.

[71]:
```
# Fourth plot: (2002.4, 2008.2]
# This is essentially 2003 - 2008 (no partial years)
# We know this because in part 1 we saw years
# are represented by integers.

# Grabbing a dataframe from our time period, that contains the data we want to␣
 ↪plot
plot_df4 = pd.DataFrame(data=[corr_4.groupby(['teamID']).mean()['std_payroll'],
                              corr_4.groupby(['teamID']).
 ↪mean()['winning_percentage']]).transpose()
# Period 4 Plot
fig4 = plt.figure()
win_pay4 = fig4.add_subplot(111)
win_pay4.set_prop_cycle('color', colors)

# Linear Regression
p4 = np.poly1d(np.polyfit(plot_df4['std_payroll'],␣
 ↪plot_df4['winning_percentage'], 1))
xp4 = np.linspace(-1.5, 3.1, 100)
win_pay4.plot(xp4, p4(xp4), label=p4, color='orange')
dis4 = list() # Get distances

# Scatter Plot
for key,data in plot_df4.groupby('teamID'):
    win_pay4.scatter(data['std_payroll'], data['winning_percentage'], label=key)
    dis4.append((key, float(data['winning_percentage']) -␣
 ↪p4(data['std_payroll'])[0]))

# Plotting (labels, etc)
win_pay4.legend(loc=(1.3,0))
win_pay4.set_xlabel("Standard Mean Payroll\n\n \
This is a scatter plot of the standard mean payroll and avg winning␣
 ↪percentage\n \
for each team, along with a regression of the plot for time period 4.")
win_pay4.set_ylabel("Mean Winning Percentage")
win_pay4.set_title('Mean Winning Percentage Versus Standard Mean Payroll from␣
 ↪2003 - 2008')

fig4.show()
dis4.sort(key=lambda x:-x[1]) # reverse sort
print('Distances from regression')
pprint(dis4)
```

```
Distances from regression
[('MIN', 6.096823046206005),
 ('LAA', 5.692374138025997),
 ('OAK', 5.315548989734346),
 ('FLO', 4.6801004400904915),
 ('SLN', 3.9596675928719094),
 ('CLE', 3.558141964987243),
 ('BOS', 3.140422783366283),
 ('PHI', 2.8681152381600867),
 ('CHA', 2.390502119382006),
 ('HOU', 1.9027383489060696),
 ('ATL', 1.8551679757692554),
 ('TOR', 1.5580343155141563),
 ('MIL', 0.4035281887606317),
 ('SDN', 0.35231284719133527),
 ('CHN', 0.06129268361774365),
 ('ANA', -0.020732626266372733),
 ('TEX', -0.7252154252899103),
 ('LAN', -0.8396124144353365),
 ('SFN', -0.9218647831574245),
 ('MON', -1.015286600243897),
 ('TBA', -1.3459233898346383),
 ('ARI', -1.5023920996014937),
 ('COL', -1.82438499616471),
 ('NYA', -2.537538572038393),
 ('CIN', -2.7594209821941647),
 ('NYN', -2.7975889715736173),
 ('WAS', -3.127425702192582),
 ('PIT', -3.6280163876413667),
 ('DET', -4.555503121141719),
 ('BAL', -5.279580790898123),
 ('SEA', -5.33919989399724),
 ('KCA', -5.615083915912344)]
```

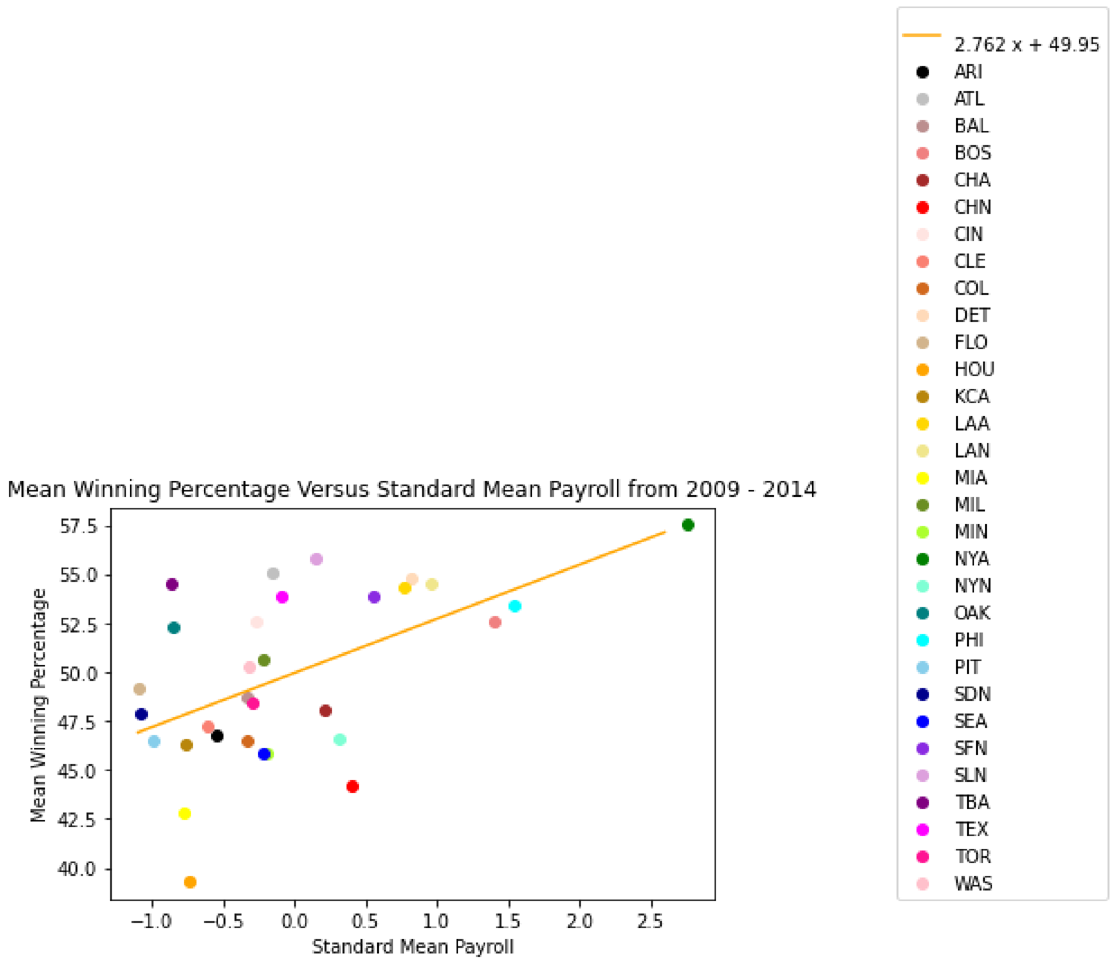


This is a scatter plot of the standard mean payroll and avg winning percentage
for each team, along with a regression of the plot for time period 4.

```
[72]: # Fifth plot: (2008.2, 2014.0]
# This is essentially 2009 - 2014 (no partial years)
# We know this because in part 1 we saw years
# are represented by integers.

# Grabbing a dataframe from our time period, that contains the data we want to␣
 ↪plot
plot_df5 = pd.DataFrame(data=[corr_5.groupby(['teamID']).mean()['std_payroll'],
                       corr_5.groupby(['teamID']).
 ↪mean()['winning_percentage']]).transpose()
# Period 5 Plot
fig5 = plt.figure()
win_pay5 = fig5.add_subplot(111)
win_pay5.set_prop_cycle('color', colors)
```

```
# Linear Regression
p5 = np.poly1d(np.polyfit(plot_df5['std_payroll'],␣
 ↪plot_df5['winning_percentage'], 1))
xp5 = np.linspace(-1.1, 2.6, 100)
win_pay5.plot(xp5, p5(xp5), label=p5, color='orange')
dis5 = list() # Get distances

for key,data in plot_df5.groupby('teamID'):
    win_pay5.scatter(data['std_payroll'], data['winning_percentage'], label=key)
    dis5.append((key, float(data['winning_percentage']) -␣
 ↪p5(data['std_payroll'])[0]))

# Plotting (labels, etc)
win_pay5.legend(loc=(1.3,0))
win_pay5.set_xlabel("Standard Mean Payroll\n\n \
This is a scatter plot of the standard mean payroll and avg winning␣
 ↪percentage\n \
for each team, along with a regression of the plot for time period 5.")
win_pay5.set_ylabel("Mean Winning Percentage")
win_pay5.set_title('Mean Winning Percentage Versus Standard Mean Payroll from␣
 ↪2009 - 2014')

fig5.show()
dis5.sort(key=lambda x:-x[1]) # reverse sort
print('Distances from regression')
pprint(dis5)
```

```
Distances from regression
[('TBA', 6.912493532894594),
 ('ATL', 5.528903164489982),
 ('SLN', 5.407594595796873),
 ('OAK', 4.685561931383681),
 ('TEX', 4.158358914605429),
 ('CIN', 3.3706496384184987),
 ('DET', 2.5790516245335198),
 ('SFN', 2.4178636254767483),
 ('FLO', 2.26033181181117),
 ('LAA', 2.245966243645725),
 ('LAN', 1.8632095099032),
 ('MIL', 1.2655734294893008),
 ('WAS', 1.1930312470294027),
 ('SDN', 0.8729611294915216),
 ('NYA', -0.06156155561694021),
 ('BAL', -0.35538211860003344),
 ('PIT', -0.6747102832884764),
 ('TOR', -0.6894255219696745),
 ('PHI', -0.8253755386972017),
 ('CLE', -1.0358715108593515),
 ('BOS', -1.2393417804849207),
 ('KCA', -1.5271163799438412),
 ('ARI', -1.6290068785148861),
 ('CHA', -2.4818816954811282),
 ('COL', -2.517700840392159),
 ('SEA', -3.475841080998798),
 ('MIN', -3.5765994761844198),
 ('NYN', -4.225158277958705),
 ('MIA', -4.990820066184234),
 ('CHN', -6.860819473655994),
 ('HOU', -8.594937920139238)]
```



This is a scatter plot of the standard mean payroll and avg winning percentage for each team, along with a regression of the plot for time period 5.

5.1.4 End of Problem 6

5.1.5 Question 3: Discuss how the plots from Problem 4 and Problem 6 reflect the transformation you did on the payroll variable.

Before we discss, an explanation of the plots in 4 and 6:
In general the plots in problem 4 show mean winning percentage vs mean payroll for a certain time period for teams. The plots in problem 6 show mean winning percentage vs standardized mean payroll for a certain time period for teams.

With the plots in problem 6, everything is in terms of the 'mean' of that year, which is at x = 0. Whereas in problem 4, you can see that the payrolls were everywhere (from tens of millions to hundreds of millions), and not standardized, so it was hard to tell where the mean really was and how teams relate to that mean payroll. In terms of problem 6, you can see that relationship because x=1 means you are one stdv from the mean payroll for that year. In addition, the graphs actually look super similar between 4 and 6, it is just the x-axis that really changed, which is to be expected, because the distribution of the data did not change but all we did was standardize how we look at the graphs/data. This will actually allow us to see relationships between different time periods (the sheer difference in total payroll of a season no longer matters), as show in the problem 7.

5.2 Expected wins

5.2.1 Problem 7: Make a single scatter plot of winning percentage (y-axis) vs. standardized payroll (x-axis). Add a regression line to highlight the relationship.

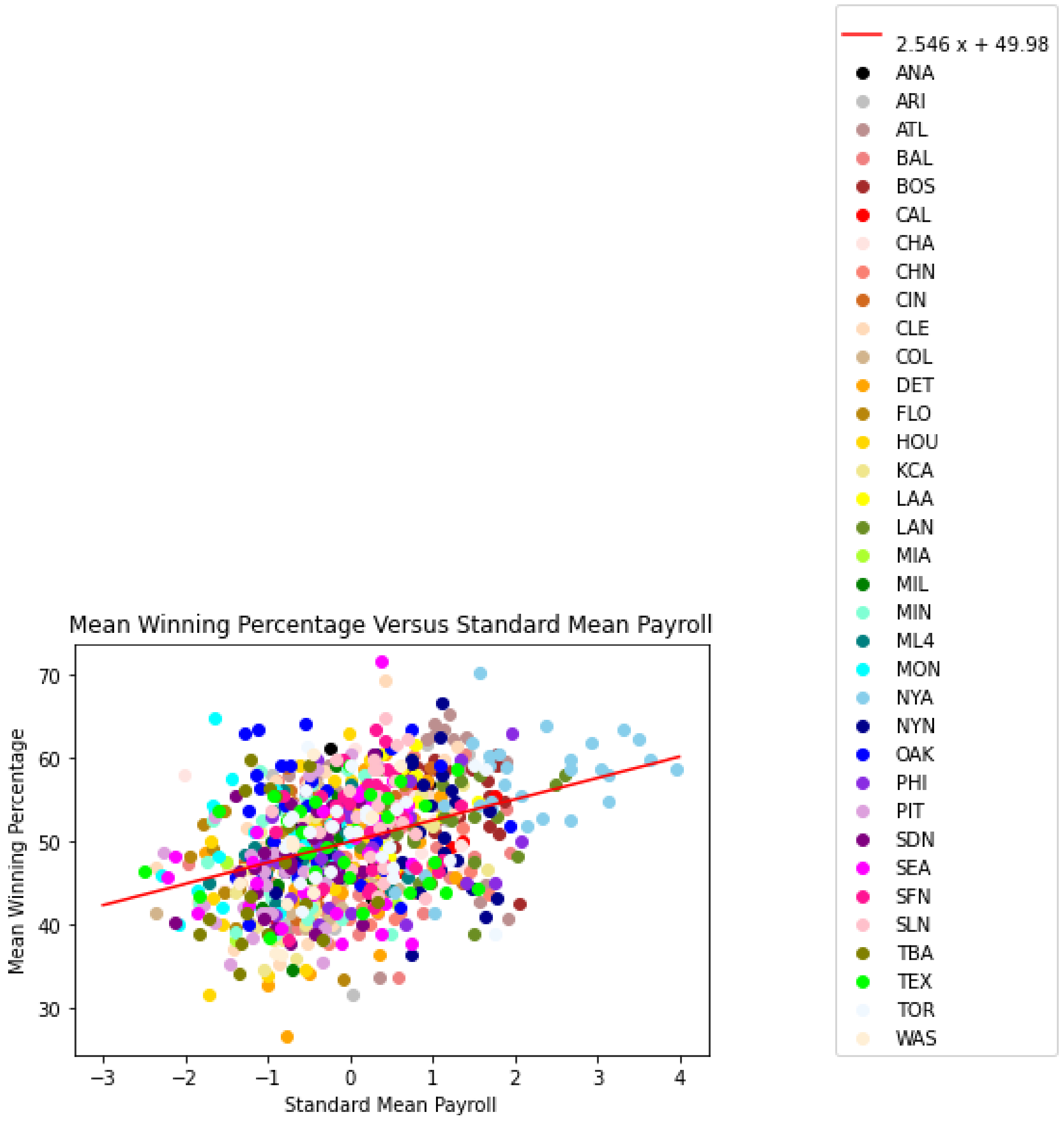
Short prose explaining my code:
In this problem, I grab a subset of the moneyball_df just for correlation analysis between winning percentage and standardized payroll. I already took a subset of this in the preivous problem, so I will take that (corr_df) and drop extra columns I don't need, and then save that to a new dataframe called std_df (standard values). This dataframe is now prepped for plotting, so I plot it and use a linear regression (from numpy polyfit) as well on the entire scatter plot for all teams across all years using matplotlib.

[73]:
```
# Again, we are going to grab a subset dataframe just for plotting.

# We already took a subset dataframe: corr_df
# Just drop the cols we no longer need
std_df = corr_df.drop(['time_period', 'yearID'], axis=1)
std_df
```

[73]:
```
    teamID  winning_percentage  std_payroll
0      ATL           40.740741     1.914905
1      BAL           51.552795     0.601068
2      BOS           49.693252     0.332678
3      CAL           55.555556     1.761474
4      CHA           52.147239    -0.092838
..     …                   …         …
853    SLN           55.555556     0.336014
854    TBA           47.530864    -0.793346
```

```
855     TEX            41.358025     0.137500
856     TOR            51.234568     0.082566
857     WAS            59.259259     0.601644

[858 rows x 3 columns]
```

[74]:
```
# Set our "plot" to our std_df we found earlier
plot_df = std_df

# Plotting our standardized values for all years
fig = plt.figure()
win_pay = fig.add_subplot(111)
win_pay.set_prop_cycle('color', colors)

# Linear Regression
p = np.poly1d(np.polyfit(plot_df['std_payroll'], plot_df['winning_percentage'],␣
 ↪1))
xp = np.linspace(-3, 4, 100)
win_pay.plot(xp, p(xp), label=p, color='red')

# Scatter Plot
for key,data in plot_df.groupby('teamID'):
    win_pay.scatter(data['std_payroll'], data['winning_percentage'], label=key)
win_pay.legend(loc=(1.2,0))
win_pay.set_xlabel("Standard Mean Payroll\n\n \
This is a scatter plot with a linear regression showing\n \
the mean winning percentage vs standard mean payroll for teams.")
win_pay.set_ylabel("Mean Winning Percentage")
win_pay.set_title('Mean Winning Percentage Versus Standard Mean Payroll')

fig.show()
```



This is a scatter plot with a linear regression showing
the mean winning percentage vs standard mean payroll for teams.

5.2.2 End of Problem 7

#### From the readme:
#### expected_win_percentage (team,year) = 50 + 2.5 X standardized_payroll(team,year)

This matches what we got (see legend from above graph).

5.3 Spending efficiency

We can make a singple plot that makes it easier to compare teams efficiency. Plus more calculations!

5.3.1 Problem 8: Create a new field to compute each team's spending efficiency, given by efficiency (year,team) = winning percentage (year, team) - expected winning percentage (year,team). Then, make a line plot with the year (x axis) and efficiency (y axis). Plot with teams like Oakland, New York Yankees, Boston, Atlanta, and Tampa Bay.

Short prose explaining my code:
In this problem, we want to compute a new field called efficiency. I just go through the moneyball_df (which is my main dataframe house of data), and then go though the values, plugging into the efficiency equation above. Once I calculate the value for that row, I append it to my efc_vals list. Then, I create a column with that list to add the efficiency field to each row (each team/year) in my main dataframe moneyball_df.
After adding that new field, the problem wanted a line plot of the efficiency of teams like Oakland, NY Yankees, Boston, Atlanta and Tampa Bay. So, I grab a sub dataframe called efc_df with the columns of interest (efficiency, year, etc) from the moneyball_df for the purposes of plotting. Then, I select the rows which match the list of teams we want (Oakladn, NYA, etc). Finally, I plot the efficiency of these teams over time.

[75]:
```python
# In our moneyball_df we have to add yet another column,
# efficiency.
# Using our formulas above, we will go through
# and calculate it, then add it as a column.
# Now, let's go through and calculate efficiency.
efc_vals = list()

for row in moneyball_df.iterrows():
    # Grab values from the table, std payroll and winning %
    # to plug into our formula
    std_pr = moneyball_df.at[row[0], 'std_payroll']
    win = moneyball_df.at[row[0], 'winning_percentage']

    # Calculate efficiency
    exp_win_pct = 50 + 2.5*std_pr
    efc = win - exp_win_pct

    # Append to our list
    efc_vals.append(efc)
```

[76]:
```python
# Create the new dataset! Problem 8 part 1 is done.
moneyball_df = moneyball_df.assign(efficiency = efc_vals)
moneyball_df
```

[76]:
```
   yearID teamID  total_payroll    G   W   L                 name  \
0    1985    ATL     14807000.0  162  66  96       Atlanta Braves
1    1985    BAL     11560712.0  161  83  78    Baltimore Orioles
2    1985    BOS     10897560.0  163  81  81       Boston Red Sox
3    1985    CAL     14427894.0  162  90  72    California Angels
4    1985    CHA      9846178.0  163  85  77    Chicago White Sox
```

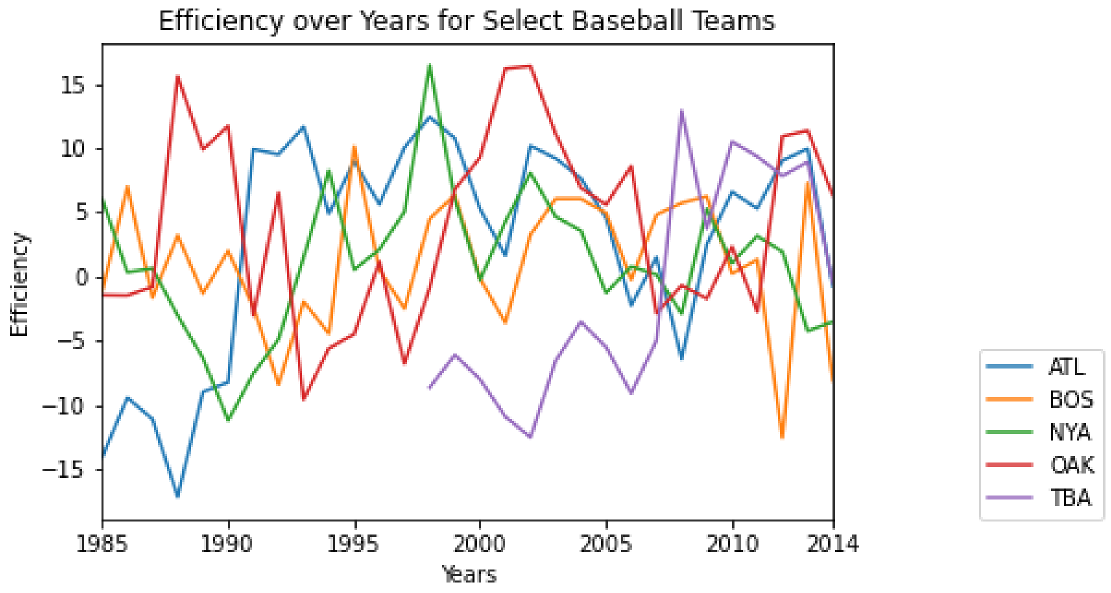
```
..      …     …           …   …  ..  ..                   …
853    2014    SLN    120693000.0  162  90  72   St. Louis Cardinals
854    2014    TBA     72689100.0  162  77  85        Tampa Bay Rays
855    2014    TEX    112255059.0  162  67  95         Texas Rangers
856    2014    TOR    109920100.0  162  83  79      Toronto Blue Jays
857    2014    WAS    131983680.0  162  96  66  Washington Nationals

     winning_percentage  std_payroll  efficiency
0             40.740741     1.914905  -14.046522
1             51.552795     0.601068    0.050124
2             49.693252     0.332678   -1.138442
3             55.555556     1.761474    1.151872
4             52.147239    -0.092838    2.379333
..                  …            …          …
853           55.555556     0.336014    4.715520
854           47.530864    -0.793346   -0.485770
855           41.358025     0.137500   -8.985724
856           51.234568     0.082566    1.028152
857           59.259259     0.601644    7.755150

[858 rows x 10 columns]
```

[77]:
```
# Now I have to make a line plot with year on the x-axis and efficiency on the␣
↪y axis
# Now, I want to grab a sub-dataframe for plotting from our main moneyball_df
efc_df = moneyball_df.drop(['total_payroll', 'std_payroll',
                            'winning_percentage', 'name', 'G', 'W', 'L'],␣
↪axis=1)

# Go thorugh and choose the teams stated above: OAK, BOS, NYA, ATL, TBA

# Create a list of teams we are interested in
efc_teams = ['OAK', 'BOS', 'NYA', 'ATL', 'TBA']
# Choose rows that are in our list of chosen teams
efc_df = efc_df.loc[efc_df['teamID'].isin(efc_teams)]
# Here is how it looks
efc_df
```

[77]:
```
     yearID teamID  efficiency
0      1985    ATL  -14.046522
2      1985    BOS   -1.138442
15     1985    NYA    6.036692
17     1985    OAK   -1.440177
26     1986    ATL   -9.407444
..      …     …          …
829    2014    ATL   -0.716893
831    2014    BOS   -8.090793
```

```
845    2014    NYA   -3.508253
847    2014    OAK    6.320863
854    2014    TBA   -0.485770

[137 rows x 3 columns]
```

[78]:
```python
# Set our "plot" to our efficiency_df we found earlier
plot_df = efc_df

# Plotting our standardized values for all years
fig = plt.figure()
efc_yr = fig.add_subplot(111)

# Scatter Plot
for key,data in plot_df.groupby('teamID'):
    efc_yr.plot(data['yearID'], data['efficiency'], label=key)

# Plotting (legend, labels, etc)
efc_yr.legend(loc=(1.2,0))
efc_yr.set_xlabel("Years\n\n \
This is a line plot showing the efficiency of select teams over the years.")
efc_yr.set_ylabel("Efficiency")
efc_yr.set_title('Efficiency over Years for Select Baseball Teams')
efc_yr.set_xlim(1985,2014)
efc_yr.set_xticks([1985, 1990, 1995, 2000, 2005, 2010, 2014])
efc_yr.set_xticklabels([1985, 1990, 1995, 2000, 2005, 2010, 2014])

fig.show()
```



This is a line plot showing the efficiency of select teams over the years.

5.3.2 End of Problem 8

5.3.3 Question 4: What can you learn from this plot compared to the set of plots you looked at in Question 2 and 3? How good was Oakland's efficiency during the Moneyball period?

The plot in discussion is the one above, which is a line plot showing the efficiency of teams (ATL, BOS, NYA, OAK, TBA) over the years. Each team is dipicted with its own line/color.

From this plot, we can see that Oakland is actually more efficient relative to the other teams in the graph around the years 2001-2002, it looks like they were around 15% efficient while other strong teams like the New York Yankees were around 5-10% efficient. The moneyball period was between 2000 and 2005, and here we can see that Oakland Athletics had a higher efficiency than other strong teams for most of this time (except fell off towards the end of this period). Also, we can see prior to 2000, Oakland was not very efficient in their spending and was mostly negative from 1990 to 2000. However, they were efficient in the couple of years before 1990 before dipping. We can also see that all the teams including Oakland Athletics, have closer efficiency spending around 2006, right after the moneyball period. Compared to the plots we looked at in questions 2 and 3, it was not apparent how well Oakland was managing their money compared to teams with different payroll budgets (even when standardized). With this efficiency, we can tell how a efficient a team is over years (not over payroll on the x-axis), which allows us to really see how well Oakland did during the moneyball period. It also did not help that questions 2 and 3 had us look at graphs split into different time periods.

5.4 Extra Credit

Make a new SQL database with the newest data. I am using the baseball data linked inside the readme! It has 2020 csv data, but it is not in sqlite form.

Short prose explaining my code:
I use the os library to run some commands, like download the csv data from the lahman database website and unzipping. Then, I use Pandas to read the csv, convert the values to the best data types, and create a sql table in a new sqlite database called extra_credit.db

[79]:
```
# Create a new database!
conn_ec = sqlite3.connect('extra_credit.db')
cursor_ec = conn_ec.cursor()
```

Using os to execute commands to get the comma-delimited data from: http://www.seanlahman.com/baseball-archive/statistics/ If not on Ubuntu 18, these commands may have to change.

[80]:
```
# Just run through this in case there are any files
# on the system with the same name.
# Want code to work later.
os.system('rm -r master.zip')
os.system('rm -r baseballdatabank-master')
```

[80]: 0

[81]:
```
# Let's execute some shell commands to grab the data.
# I am on Ubuntu / Linux Version 18.
# The file is is a Windows zip file, so make sure unzip is installed
os.system('wget https://github.com/chadwickbureau/baseballdatabank/archive/
↪master.zip')
os.system('unzip master.zip')
```

[81]: 0

[82]:
```
# The documentation for os has very interesting functions
# one of them lists everything in a path.
# Perfect!
directory = os.listdir(path='./baseballdatabank-master/core')
csv_files = list()

# Go through and make sure everything is a
# csv file, if not, do not count it as such.
for file in directory:
    is_csv = re.match("^(.*)\.csv$", file)
    if is_csv:
        # Just save the name, if we keep the .csv
        # it won't we be as easy when we create
        # the table name.
        csv_files.append(is_csv.group(1))
csv_files
```

[82]:
```
['ManagersHalf',
 'AwardsPlayers',
 'CollegePlaying',
 'FieldingOFsplit',
 'AwardsManagers',
 'FieldingPost',
 'AwardsShareManagers',
 'AwardsSharePlayers',
 'People',
 'Pitching',
 'Salaries',
 'BattingPost',
 'AllstarFull',
 'Parks',
 'PitchingPost',
 'TeamsFranchises',
 'FieldingOF',
 'SeriesPost',
 'Batting',
 'Fielding',
 'TeamsHalf',
 'Schools',
 'HallOfFame',
 'Managers',
 'Appearances',
 'HomeGames',
 'Teams']
```

Pandas can do everything that I want it to. I spent time reading the documentation, and this is actually super nice. For full information on the functions I use (though I do give an explanation below), please check out: https://pandas.pydata.org/docs/index.html. - Pandas can read_csv() - Pandas can then convert to the best data types (reads ints in as floats by default with csv, but can convert 0.0 to 0) with convert_dtypes() - Pandas can convert a dataframe to_sql() and add it to a sqlite database

Miniature example: In the next three cells, I will give a quick example of what functions I am using.

```
[83]: # Get the dataframe with read csv, just giving path
df = pd.read_csv('./baseballdatabank-master/core/AllstarFull.csv')
# Print out what we read
df
# See that the integer types are converted to float by pandas
```

```
[83]:      playerID  yearID  gameNum        gameID teamID lgID  GP  startingPos
      0     gomezle01  1933.0      0.0  ALS193307060    NYA   AL   1          1.0
      1     ferreri01  1933.0      0.0  ALS193307060    BOS   AL   1          2.0
      2     gehrilo01  1933.0      0.0  ALS193307060    NYA   AL   1          3.0
      3     gehrich01  1933.0      0.0  ALS193307060    DET   AL   1          4.0
      4     dykesji01  1933.0      0.0  ALS193307060    CHA   AL   1          5.0
      …           …       …        …             …      …    …  ..          …
      5370  sorokmi01  2019.0      0.0  ALS201907090    ATL   NL   1          NaN
      5371  storytr01  2019.0      0.0  ALS201907090    COL   NL   1          NaN
      5372  woodrbr01  2019.0      0.0  ALS201907090    MIL   NL   1          NaN
      5373  yateski01  2019.0      0.0  ALS201907090    SDN   NL   0          NaN
      5374  bailean01     NaN      NaN           NaN    OAK   AL   0          NaN

      [5375 rows x 8 columns]
```

```
[84]: # Have it automatically optimize data types (so int if it is like 2019.0)
# and use pandas NA
df = df.convert_dtypes()
df
# See that everything looks great
```

```
[84]:      playerID  yearID  gameNum        gameID teamID lgID  GP  startingPos
      0     gomezle01    1933        0  ALS193307060    NYA   AL   1            1
```

```
1     ferreri01    1933        0  ALS193307060    BOS   AL   1            2
2     gehrilo01    1933        0  ALS193307060    NYA   AL   1            3
3     gehrich01    1933        0  ALS193307060    DET   AL   1            4
4     dykesji01    1933        0  ALS193307060    CHA   AL   1            5
…         …     …       …          …     …   …   ..         …
5370  sorokmi01    2019        0  ALS201907090    ATL   NL   1         <NA>
5371  storytr01    2019        0  ALS201907090    COL   NL   1         <NA>
5372  woodrbr01    2019        0  ALS201907090    MIL   NL   1         <NA>
5373  yateski01    2019        0  ALS201907090    SDN   NL   0         <NA>
5374  baileand01   <NA>     <NA>          <NA>    OAK   AL   0         <NA>

[5375 rows x 8 columns]
```

[85]:
```python
# Here is how we store it in the sqlite database
df.to_sql('AllstarFull', conn_ec, if_exists='replace', index=False)
# only want to store vars so index=False

# Print out to show what we did
print("Columns of table AllstarFull (col position, col name)")
cursor_ec.execute("PRAGMA table_info('AllstarFull')")
pprint(cursor_ec.fetchall())

cursor_ec.execute("SELECT * FROM AllstarFull LIMIT 15")
print("Here are the first 15 rows from our table AllstarFull from our new SQL␣
 ↪database")
pprint(cursor_ec.fetchall())
```

```
Columns of table AllstarFull (col position, col name)
[(0, 'playerID', 'TEXT', 0, None, 0),
 (1, 'yearID', 'INTEGER', 0, None, 0),
 (2, 'gameNum', 'INTEGER', 0, None, 0),
 (3, 'gameID', 'TEXT', 0, None, 0),
 (4, 'teamID', 'TEXT', 0, None, 0),
 (5, 'lgID', 'TEXT', 0, None, 0),
 (6, 'GP', 'INTEGER', 0, None, 0),
 (7, 'startingPos', 'INTEGER', 0, None, 0)]
Here are the first 15 rows from our table AllstarFull from our new SQL database
[('gomezle01', 1933, 0, 'ALS193307060', 'NYA', 'AL', 1, 1),
 ('ferreri01', 1933, 0, 'ALS193307060', 'BOS', 'AL', 1, 2),
 ('gehrilo01', 1933, 0, 'ALS193307060', 'NYA', 'AL', 1, 3),
 ('gehrich01', 1933, 0, 'ALS193307060', 'DET', 'AL', 1, 4),
 ('dykesji01', 1933, 0, 'ALS193307060', 'CHA', 'AL', 1, 5),
 ('cronijo01', 1933, 0, 'ALS193307060', 'WS1', 'AL', 1, 6),
 ('chapmbe01', 1933, 0, 'ALS193307060', 'WS1', 'AL', 1, 7),
 ('simmoal01', 1933, 0, 'ALS193307060', 'CHA', 'AL', 1, 8),
 ('ruthba01', 1933, 0, 'ALS193307060', 'NYA', 'AL', 1, 9),
 ('averiea01', 1933, 0, 'ALS193307060', 'CLE', 'AL', 1, None),
 ('crowdal01', 1933, 0, 'ALS193307060', 'WS1', 'AL', 1, None),
 ('dickebi01', 1933, 0, 'ALS193307060', 'NYA', 'AL', 0, None),
 ('ferrewe01', 1933, 0, 'ALS193307060', 'CLE', 'AL', 0, None),
 ('foxxji01', 1933, 0, 'ALS193307060', 'PHA', 'AL', 0, None),
 ('grovele01', 1933, 0, 'ALS193307060', 'PHA', 'AL', 1, None)]
```

End Miniature Example. I will be using the code and ideas from this example to add all the tables to teh extra_credit sqlite database!

```
[86]: # I will use our csv_files list we found earlier
      # Go through and add to our sqlite database
      for csv in csv_files:
          path_csv = './baseballdatabank-master/core/' + csv + '.csv'
          pd.read_csv(path_csv).convert_dtypes().to_sql(csv, conn_ec,␣
       ↪if_exists='replace', index=False)

          print("Columns (col pos, col name) of table " + csv)
          # Commented out because the output was getting too large
          # Please uncomment to check out column names and types
          #cursor_ec.execute("PRAGMA table_info('" + csv + "')")
          #pprint(cursor_ec.fetchall())

          print("First 2 rows from table " + csv +  " from extra_credit sqlite␣
       ↪database")
          cursor_ec.execute("SELECT * FROM " + csv + " LIMIT 2")
          print(cursor_ec.fetchall())

          print('\n')
```

```
Columns (col pos, col name) of table ManagersHalf
First 2 rows from table ManagersHalf from extra_credit sqlite database
[('hanlone01', 1892, 'BLN', 'NL', 3, 1, 56, 17, 39, 12), ('hanlone01', 1892,
'BLN', 'NL', 3, 2, 77, 26, 46, 10)]


Columns (col pos, col name) of table AwardsPlayers
First 2 rows from table AwardsPlayers from extra_credit sqlite database
[('bondto01', 'Pitching Triple Crown', 1877, 'NL', None, None), ('hinespa01',
'Triple Crown', 1878, 'NL', None, None)]


Columns (col pos, col name) of table CollegePlaying
First 2 rows from table CollegePlaying from extra_credit sqlite database
[('aardsda01', 'pennst', 2001), ('aardsda01', 'rice', 2002)]


Columns (col pos, col name) of table FieldingOFsplit
First 2 rows from table FieldingOFsplit from extra_credit sqlite database
```

```
[('aaronha01', 1954, 1, 'ML1', 'NL', 'LF', 105, 102, 2773, 205, 4, 6, 0, None,
None, None, None, None), ('aaronha01', 1954, 1, 'ML1', 'NL', 'RF', 11, 11, 320,
12, 1, 1, 1, None, None, None, None, None)]


Columns (col pos, col name) of table AwardsManagers
First 2 rows from table AwardsManagers from extra_credit sqlite database
[('larusto01', 'BBWAA Manager of the Year', 1983, 'AL', None, None),
('lasorto01', 'BBWAA Manager of the Year', 1983, 'NL', None, None)]


Columns (col pos, col name) of table FieldingPost
First 2 rows from table FieldingPost from extra_credit sqlite database
[('beaumgi01', 1903, 'PIT', 'NL', 'WS', 'CF', 8, 8, 210, 21, 0, 0, 0, 0, None,
None, None), ('branski01', 1903, 'PIT', 'NL', 'WS', '1B', 8, 8, 210, 81, 6, 3,
5, 0, None, None, None)]


Columns (col pos, col name) of table AwardsShareManagers
First 2 rows from table AwardsShareManagers from extra_credit sqlite database
[('BBWAA Manager of the Year', 1983, 'AL', 'altobjo01', 7, 28, 7), ('BBWAA
Manager of the Year', 1983, 'AL', 'coxbo01', 4, 28, 4)]


Columns (col pos, col name) of table AwardsSharePlayers
First 2 rows from table AwardsSharePlayers from extra_credit sqlite database
[('Cy Young', 1956, 'ML', 'fordwh01', 1.0, 16, 1.0), ('Cy Young', 1956, 'ML',
'maglisa01', 4.0, 16, 4.0)]


Columns (col pos, col name) of table People
First 2 rows from table People from extra_credit sqlite database
[('aardsda01', 1981, 12, 27, 'USA', 'CO', 'Denver', None, None, None, None,
None, None, 'David', 'Aardsma', 'David Allan', 215, 75, 'R', 'R', '2004-04-06',
'2015-08-23', 'aardd001', 'aardsda01'), ('aaronha01', 1934, 2, 5, 'USA', 'AL',
'Mobile', 2021, 1, 22, 'USA', 'GA', 'Atlanta', 'Hank', 'Aaron', 'Henry Louis',
180, 72, 'R', 'R', '1954-04-13', '1976-10-03', 'aaroh101', 'aaronha01')]


Columns (col pos, col name) of table Pitching
First 2 rows from table Pitching from extra_credit sqlite database
[('bechtge01', 1871, 1, 'PH1', None, 1, 2, 3, 3, 2, 0, 0, 78, 43, 23, 0, 11, 1,
None, 7.96, None, 7, None, 0, 146, 0, 42, None, None, None), ('brainas01', 1871,
1, 'WS3', None, 12, 15, 30, 30, 30, 0, 0, 792, 361, 132, 4, 37, 13, None, 4.5,
None, 7, None, 0, 1291, 0, 292, None, None, None)]


Columns (col pos, col name) of table Salaries
```

```
First 2 rows from table Salaries from extra_credit sqlite database
[(1985, 'ATL', 'NL', 'barkele01', 870000), (1985, 'ATL', 'NL', 'bedrost01',
550000)]


Columns (col pos, col name) of table BattingPost
First 2 rows from table BattingPost from extra_credit sqlite database
[(1884, 'WS', 'becanbu01', 'NY4', 'AA', 1, 2, 0, 1, 0, 0, 0, 0, 0, None, 0, 0,
0, None, None, None, None), (1884, 'WS', 'bradyst01', 'NY4', 'AA', 3, 10, 1, 0,
0, 0, 0, 0, 0, None, 0, 1, 0, None, None, None, None)]


Columns (col pos, col name) of table AllstarFull
First 2 rows from table AllstarFull from extra_credit sqlite database
[('gomezle01', 1933, 0, 'ALS193307060', 'NYA', 'AL', 1, 1), ('ferreri01', 1933,
0, 'ALS193307060', 'BOS', 'AL', 1, 2)]


Columns (col pos, col name) of table Parks
First 2 rows from table Parks from extra_credit sqlite database
[('ALB01', 'Riverside Park', None, 'Albany', 'NY', 'US'), ('ALT01', 'Columbia
Park', None, 'Altoona', 'PA', 'US')]


Columns (col pos, col name) of table PitchingPost
First 2 rows from table PitchingPost from extra_credit sqlite database
[('becanbu01', 1884, 'WS', 'NY4', 'AA', 0, 1, 1, 1, 1, 0, 0, 18, 9, 7, 0, 2, 1,
None, 10.5, None, None, None, None, None, 0, 12, None, None, None),
('keefeti01', 1884, 'WS', 'NY4', 'AA', 0, 2, 2, 2, 2, 0, 0, 45, 10, 6, 1, 3, 12,
None, 3.6, None, None, None, None, None, 0, 9, None, None, None)]


Columns (col pos, col name) of table TeamsFranchises
First 2 rows from table TeamsFranchises from extra_credit sqlite database
[('ALT', 'Altoona Mountain City', 'N', None), ('ANA', 'Los Angeles Angels of
Anaheim', 'Y', None)]


Columns (col pos, col name) of table FieldingOF
First 2 rows from table FieldingOF from extra_credit sqlite database
[('allisar01', 1871, 1, 0, 29, 0), ('ansonca01', 1871, 1, 1, 0, 0)]


Columns (col pos, col name) of table SeriesPost
First 2 rows from table SeriesPost from extra_credit sqlite database
[(1884, 'WS', 'PRO', 'NL', 'NY4', 'AA', 3, 0, 0), (1885, 'WS', 'CHN', 'NL',
'SL4', 'AA', 3, 3, 1)]
```

```
Columns (col pos, col name) of table Batting
First 2 rows from table Batting from extra_credit sqlite database
[('abercda01', 1871, 1, 'TRO', None, 1, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, None,
None, None, None, 0), ('addybo01', 1871, 1, 'RC1', None, 25, 118, 30, 32, 6, 0,
0, 13, 8, 1, 4, 0, None, None, None, None, 0)]


Columns (col pos, col name) of table Fielding
First 2 rows from table Fielding from extra_credit sqlite database
[('abercda01', 1871, 1, 'TRO', None, 'SS', 1, 1, 24, 1, 3, 2, 0, None, None,
None, None, None), ('addybo01', 1871, 1, 'RC1', None, '2B', 22, 22, 606, 67, 72,
42, 5, None, None, None, None, None)]


Columns (col pos, col name) of table TeamsHalf
First 2 rows from table TeamsHalf from extra_credit sqlite database
[(1981, 'NL', 'ATL', 1, 'W', 'N', 4, 54, 25, 29), (1981, 'NL', 'ATL', 2, 'W',
'N', 5, 52, 25, 27)]


Columns (col pos, col name) of table Schools
First 2 rows from table Schools from extra_credit sqlite database
[('abilchrist', 'Abilene Christian University', 'Abilene', 'TX', 'USA'),
('adelphi', 'Adelphi University', 'Garden City', 'NY', 'USA')]


Columns (col pos, col name) of table HallOfFame
First 2 rows from table HallOfFame from extra_credit sqlite database
[('cobbty01', 1936, 'BBWAA', 226, 170, 222, 'Y', 'Player', None), ('ruthba01',
1936, 'BBWAA', 226, 170, 215, 'Y', 'Player', None)]


Columns (col pos, col name) of table Managers
First 2 rows from table Managers from extra_credit sqlite database
[('wrighha01', 1871, 'BS1', None, 1, 31, 20, 10, 3, 'Y'), ('woodji01', 1871,
'CH1', None, 1, 28, 19, 9, 2, 'Y')]


Columns (col pos, col name) of table Appearances
First 2 rows from table Appearances from extra_credit sqlite database
[(1871, 'TRO', None, 'abercda01', 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
0, 0), (1871, 'RC1', None, 'addybo01', 25, 25, 25, 25, 0, 0, 0, 22, 0, 3, 0, 0,
0, 0, 0, 0, 0)]


Columns (col pos, col name) of table HomeGames
First 2 rows from table HomeGames from extra_credit sqlite database
```

```
[(1871, None, 'BS1', 'BOS01', '1871-05-16', '1871-10-07', 16, 16, 32600), (1871,
None, 'BS1', 'NYC01', '1871-05-27', '1871-05-27', 1, 1, 3000)]


Columns (col pos, col name) of table Teams
First 2 rows from table Teams from extra_credit sqlite database
[(1871, None, 'BS1', 'BNA', None, 3, 31, None, 20, 10, None, None, 'N', None,
401, 1372, 426, 70, 37, 3, 60, 19, 73, 16, None, None, 303, 109, 3.55, 22, 1, 3,
828, 367, 2, 42, 23, 243, 24, 0.8340000000000001, 'Boston Red Stockings', 'South
End Grounds I', None, 103, 98, 'BOS', 'BS1', 'BS1'), (1871, None, 'CH1', 'CNA',
None, 2, 28, None, 19, 9, None, None, 'N', None, 302, 1196, 323, 52, 21, 10, 60,
22, 69, 21, None, None, 241, 77, 2.76, 25, 0, 1, 753, 308, 6, 28, 22, 229, 16,
0.8290000000000001, 'Chicago White Stockings', 'Union Base-Ball Grounds', None,
104, 102, 'CHI', 'CH1', 'CH1')]
```

5.4.1 The extra credit is completed. Feel free to check out the database in more detail.

5.5 **Here is just extra exploration of the data because I noticed weird things happening in the lahman2014 database.

5.5.1 Not in scope of Part 1. Please check out if interested.

I noticed from researching online that some teamIDs, franchiseIDs and names change. From https://en.wikipedia.org/wiki/Major_League_Baseball, we can see that franchises each "own" one team. Let's take a look at some weird ones to see if there are more updates that we have to do to the database.

[87]:
```
cursor.execute("SELECT teamID, name FROM Teams WHERE franchID == 'ANA';")
pprint(cursor.fetchall())
cursor.execute("SELECT teamID, name FROM Teams WHERE franchID == 'WNA';")
pprint(cursor.fetchall())
cursor.execute("SELECT teamID, name FROM Teams WHERE franchID ==  'WNL';")
pprint(cursor.fetchall())
```

```
[('LAA', 'Los Angeles Angels'),
 ('LAA', 'Los Angeles Angels'),
 ('LAA', 'Los Angeles Angels'),
 ('LAA', 'Los Angeles Angels'),
 ('CAL', 'California Angels'),
 ('CAL', 'California Angels'),
 ('CAL', 'California Angels'),
 ('CAL', 'California Angels'),
 ('CAL', 'California Angels'),
 ('CAL', 'California Angels'),
 ('CAL', 'California Angels'),
 ('CAL', 'California Angels'),
 ('CAL', 'California Angels'),
 ('CAL', 'California Angels'),
 ('CAL', 'California Angels'),
 ('CAL', 'California Angels'),
 ('CAL', 'California Angels'),
 ('CAL', 'California Angels'),
 ('CAL', 'California Angels'),
 ('CAL', 'California Angels'),
 ('CAL', 'California Angels'),
 ('CAL', 'California Angels'),
 ('CAL', 'California Angels'),
 ('CAL', 'California Angels'),
 ('CAL', 'California Angels'),
 ('CAL', 'California Angels'),
 ('CAL', 'California Angels'),
 ('CAL', 'California Angels'),
 ('CAL', 'California Angels'),
 ('CAL', 'California Angels'),
 ('CAL', 'California Angels'),
 ('CAL', 'California Angels'),
 ('CAL', 'California Angels'),
 ('CAL', 'California Angels'),
 ('CAL', 'California Angels'),
 ('CAL', 'California Angels'),
 ('ANA', 'Anaheim Angels'),
 ('ANA', 'Anaheim Angels'),
 ('ANA', 'Anaheim Angels'),
 ('ANA', 'Anaheim Angels'),
 ('ANA', 'Anaheim Angels'),
 ('ANA', 'Anaheim Angels'),
 ('ANA', 'Anaheim Angels'),
 ('ANA', 'Anaheim Angels'),
 ('LAA', 'Los Angeles Angels of Anaheim'),
 ('LAA', 'Los Angeles Angels of Anaheim'),
 ('LAA', 'Los Angeles Angels of Anaheim'),
 ('LAA', 'Los Angeles Angels of Anaheim'),
 ('LAA', 'Los Angeles Angels of Anaheim'),
 ('LAA', 'Los Angeles Angels of Anaheim'),
 ('LAA', 'Los Angeles Angels of Anaheim'),
 ('LAA', 'Los Angeles Angels of Anaheim'),
 ('LAA', 'Los Angeles Angels of Anaheim'),
 ('LAA', 'Los Angeles Angels of Anaheim')]
[('WSU', 'Washington Nationals')]
[('WS8', 'Washington Nationals'),
 ('WS8', 'Washington Nationals'),
 ('WS8', 'Washington Nationals'),
 ('WS8', 'Washington Nationals')]
```

From the output for just a couple of teams above, we can see that franchiseIDs, team names, and teamIDs change over time. The Nationals have different franchiseIDs and teamIDs but same name, but the Angels have different teamIDs, names, and pretty much the same franchise. A few more SQL operations can show how messy this data is.
So, what do we do in this situation? The best solution that I can think of is try to match on all of them. How so? I would think that since we care about matching on teamIDs between the Salaries and Teams tables, that we go through each and every teamID of our Salaries/Teams. For each teamID: Get the teamID. Get the names associated with the teamID and teamIDs associated with those names. Get the franchises of the teamID, and then get other teamIDs associated with the franchises of the current teamID (franchise names change too). Concat both lists. Any row in the db that matches within that "branch" I found should be updated to the the "latest" teamID.

[88]:
```
# Grab a set of all teamIDs in the Salaries and Teams tables
cursor.execute("SELECT teamID FROM Teams;")
lst = cursor.fetchall()
ids_1 = list(set([item for sub in lst for item in sub])) # flatten our results
cursor.execute("SELECT teamID FROM Salaries;")
lst = cursor.fetchall()
ids_2 = list(set([item for sub in lst for item in sub])) # flatten our results
# Our teamIDs set as list
ids = list(set(ids_1 + ids_2))
print(len(ids))
```

```
149
```

The helper function below essentially does the following:
Given a list of teamIDs (we feed in just one initially), find teamIDs under the same franchise and the same names associated with the teamID. See if those ones that we find are the same as the list we already know of. If not, then go ahead and continue searching until you get no more new teamIDs related found.
I am essentially branching out until I cannot anymore.

[89]:
```
# Let's make a recursive helper function
def rec_fix(iter_set_teams):
    # This is going to be a set of things I found
    # and will find in this iteration / recursive call
    relation = iter_set_teams

    # Go through the teamIDs of things we found
    # in previous recursive call.
    for tid in iter_set_teams:
        # Find the franchises that are associated with
        # the teamID. A teamID can map to multiple franchises.
        cursor.execute("SELECT franchID from Teams WHERE teamID==?", (tid,))
        lst = cursor.fetchall()
        franch_match = list(set([item for sub in lst for item in sub]))
        #print(franch_match) # uncomment to see output
        teams_same_franch = list() # To grab teams with same franchID but
        # possibly diff teamIDs

        # Grab teamIDs from that franchise list associated with current teamID
        for f in franch_match:
            cursor.execute("SELECT teamID from Teams WHERE franchID==?", (f,))
            lst = cursor.fetchall()
            teams_same_franch = list(set([item for sub in lst for item in sub]))
            #print(teams_same_franch) # uncomment to see output

        # Now, go through and grab teams with the same name(s) as our current␣
↪teamid.
        # A teamID can map to multiple names.
        cursor.execute("SELECT name from Teams WHERE teamID==?", (tid,))
        lst = cursor.fetchall()
        curr_name = list(set([item for sub in lst for item in sub]))
        #pprint(curr_name) # uncomment to see output

        teams_same_name = list()
        # Now, grab any teamID with the same name as the ones we found
        # that associate with the current teamID.
        for name in curr_name:
            cursor.execute("SELECT teamID from Teams WHERE name==?", (name,))
            lst = cursor.fetchall()
            teams_same_name = list(set(teams_same_name + list(set([item for sub␣
↪in lst for item in sub]))))
            #pprint(teams_same_name) # uncomment to see output

        # Now, I found all teams under the same franchIDs and that have the␣
↪same name(s)
        # associated with the teamID we are looking at in this iteration of our␣
↪loop.
        teams = teams_same_franch + teams_same_name
        teams.append(tid)
        teams = list(set(((teams))))
        relation = relation + teams
        #print(relation) # uncomment to see output

    # Check if we found every possibility,
    # if we found no new relations, we are done,
    # else, continue the search in another recursive call.
    if set(iter_set_teams) == set(relation):
        return list(set(relation))
    else:
        return list(set(rec_fix(list(set(relation)))))
        #print('failed')
# Side note, I do list concats and switch to set to list
# This is because Python gets upset if you concat sets
# and then make a list of that addition.
# I looked into possible set functions to avoid this
# but at the momement found the workaround with list/set
# to be the best solution at the time.
```

[90]:
```
# Here is a small example for CAL
li = rec_fix(['CAL'])
```

[91]:
```
# Can see other teamIDs it is registered as
print(li)
```

```
['LAA', 'ANA', 'CAL']
```

[92]:
```
# Because of Uniqueness constraints
# by the person who made this sqlite database, we must
# make a new column called latest_teamID (can't edit all teamIDs).
cursor.execute("ALTER TABLE Teams ADD COLUMN latest_teamID char(10);")
cursor.fetchall()
cursor.execute("ALTER TABLE Salaries ADD COLUMN latest_teamID char(10);")
cursor.fetchall()
# Unlike other operations, this really edits the database even without␣
 ↪committing.
# You can't avoid it, so make extra copies if you want to run functionality
# again. Else, if you would like to see output while restarting kernel,
# comment out these alters as they have been done.
```

[92]:
```
[]
```

[93]:
```
# Earlier, we had found the set of teamIDs
# in both Salaries and Teams.
# Go through them and then call our
# recrusive branching function.
# Update the branch we find to the latest teamID, and also
# print out results to see if anything else is weird.
for tid in ids:
    # Get a list of related teams from our helper above
    related_teams = rec_fix([tid])
    # Now that we have a list of teamIds related by name or franchise id
    # we can continue!

    # Can see relationships!
    # I decided to print it out to show how messy this data is.
    # All the printing is at the end of submission and should not affect project
    # results.
    print('Team ID in question')
    print(tid)
    print(related_teams)
    print()

    # Switch the teamIDs of any related team to the "lastest" teamID, as in␣
↪year closest
    # to present time in our latest_teamID column.

    # SQL let's us do that easily, let's go! We can just select by yearID␣
↪descending.
    tup_teams = str(tuple([key for key in related_teams])).replace(',)', ')')
    query = "SELECT teamID FROM Teams WHERE teamID IN {} ORDER BY yearID DESC␣
↪LIMIT 1".format(tup_teams)
    cursor.execute(query)
    lst = cursor.fetchall()

    # Get the latest team name
    latest_teamID = list(set([item for sub in lst for item in sub]))[0]
    #print(latest_teamID) # uncomment to see latest teamID of branch

    # Replace any teamID matching in related_teams with the latest_teamID
    query = "UPDATE or IGNORE Teams SET latest_teamID = ? WHERE teamID IN {}".
↪format(tup_teams)
    cursor.execute(query, (latest_teamID,))
    cursor.fetchall() # commit changes for session
    query = "UPDATE or IGNORE Salaries SET latest_teamID = ? WHERE teamID IN␣
↪{}".format(tup_teams)
    cursor.execute(query, (latest_teamID,))
    cursor.fetchall() # commit changes for session
```

```
Team ID in question
SL3
['BLA', 'MLU', 'SLN', 'SL3', 'MIL', 'NYA', 'SL4', 'MLA', 'BL2', 'BAL', 'ML3',
'SLA', 'SL2', 'SE1', 'ML4', 'BL3', 'BLN']

Team ID in question
CHN
['CHA', 'CHN', 'CH2', 'CH1']

Team ID in question
BS2
['BSP', 'BS2', 'BSU']

Team ID in question
WS1
['MON', 'WS7', 'WS2', 'WS4', 'WS1', 'MIN', 'WS8', 'WSU', 'WAS', 'TEX', 'WS6']

Team ID in question
```

```
PRO
['PRO']

Team ID in question
CL5
['CL5']

Team ID in question
BFN
['BFN', 'BFP']

Team ID in question
CL1
['CL1']

Team ID in question
WSU
['WS7', 'MON', 'WS2', 'WS4', 'WS1', 'MIN', 'WS8', 'WSU', 'WAS', 'TEX', 'WS6']

Team ID in question
CHF
['CHF']

Team ID in question
BSN
['BSN', 'ML1', 'ATL']

Team ID in question
MLU
['BLA', 'MLU', 'NYA', 'SLN', 'MIL', 'SL3', 'BL2', 'MLA', 'SL4', 'BAL', 'ML3',
'SLA', 'SL2', 'SE1', 'ML4', 'BL3', 'BLN']

Team ID in question
SLN
['BLA', 'MLU', 'SL3', 'SLN', 'MIL', 'NYA', 'SL4', 'MLA', 'BL2', 'BAL', 'ML3',
'SL2', 'SLA', 'SE1', 'ML4', 'BL3', 'BLN']

Team ID in question
PHI
['PHI']

Team ID in question
BL2
['BLA', 'MLU', 'NYA', 'SLN', 'MIL', 'SL3', 'BL2', 'MLA', 'SL4', 'BAL', 'ML3',
'SLA', 'SL2', 'SE1', 'ML4', 'BL3', 'BLN']

Team ID in question
PTF
```

```
['PTF']

Team ID in question
BLF
['BLF']

Team ID in question
PH3
['PH3']

Team ID in question
NYA
['BLA', 'MLU', 'NYA', 'SLN', 'MIL', 'SL3', 'BL2', 'MLA', 'SL4', 'BAL', 'ML3',
'SLA', 'SL2', 'SE1', 'ML4', 'BL3', 'BLN']

Team ID in question
MIL
['BLA', 'MLU', 'NYA', 'SLN', 'MIL', 'SL3', 'BL2', 'MLA', 'SL4', 'BAL', 'ML3',
'SLA', 'SL2', 'SE1', 'ML4', 'BL3', 'BLN']

Team ID in question
WOR
['WOR']

Team ID in question
NY1
['NYP', 'SFN', 'NY1']

Team ID in question
PH1
['PH1', 'PHA', 'PH4', 'KC1', 'OAK', 'PHN', 'PHP']

Team ID in question
KCN
['KCU', 'KC2', 'KCN']

Team ID in question
CLP
['CLP']

Team ID in question
BLU
['BLU']

Team ID in question
WAS
['WS7', 'MON', 'WS2', 'WS4', 'WS1', 'MIN', 'WS8', 'WSU', 'WAS', 'TEX', 'WS6']
```

```
Team ID in question
CN3
['CN3']

Team ID in question
PH2
['PH2']

Team ID in question
IN2
['IN2', 'IND', 'IN3', 'NEW']

Team ID in question
NY2
['NY3', 'NY2']

Team ID in question
KC2
['KCU', 'KC2', 'KCN']

Team ID in question
CN1
['CN2', 'CN1', 'CIN']

Team ID in question
PTP
['PTP']

Team ID in question
PHN
['PH1', 'PHA', 'PH4', 'KC1', 'OAK', 'PHN', 'PHP']

Team ID in question
ALT
['ALT']

Team ID in question
CHU
['CHU']

Team ID in question
SLF
['SLF']

Team ID in question
MON
['MON', 'WS7', 'WS2', 'WS4', 'WS1', 'MIN', 'WS8', 'WSU', 'WAS', 'TEX', 'WS6']
```

```
Team ID in question
HOU
['HOU']

Team ID in question
PIT
['PT1', 'PIT']

Team ID in question
HR1
['HR1', 'HAR']

Team ID in question
KEO
['KEO']

Team ID in question
HAR
['HR1', 'HAR']

Team ID in question
KC1
['PH1', 'PHA', 'PH4', 'KC1', 'OAK', 'PHN', 'PHP']

Team ID in question
NY4
['NY4']

Team ID in question
WS8
['WS7', 'MON', 'WS2', 'WS4', 'WS1', 'MIN', 'WS8', 'WSU', 'WAS', 'TEX', 'WS6']

Team ID in question
LS1
['LS1']

Team ID in question
ML4
['BLA', 'MLU', 'NYA', 'SLN', 'MIL', 'SL3', 'BL2', 'MLA', 'SL4', 'BAL', 'ML3',
'SLA', 'SL2', 'SE1', 'ML4', 'BL3', 'BLN']

Team ID in question
PHP
['PH1', 'PHA', 'PH4', 'KC1', 'OAK', 'PHN', 'PHP']

Team ID in question
BL3
['BLA', 'MLU', 'NYA', 'SLN', 'MIL', 'SL3', 'BL2', 'MLA', 'SL4', 'BAL', 'ML3',
'SLA', 'SL2', 'SE1', 'ML4', 'BL3', 'BLN']

Team ID in question
BLA
['BLA', 'MLU', 'NYA', 'SLN', 'MIL', 'SL3', 'BL2', 'MLA', 'SL4', 'BAL', 'ML3',
'SLA', 'SL2', 'SE1', 'ML4', 'BL3', 'BLN']

Team ID in question
MLA
['BLA', 'MLU', 'NYA', 'SLN', 'MIL', 'SL3', 'BL2', 'MLA', 'SL4', 'BAL', 'ML3',
'SLA', 'SL2', 'SE1', 'ML4', 'BL3', 'BLN']

Team ID in question
ARI
['ARI']

Team ID in question
LS2
['LS3', 'LS2']

Team ID in question
SLU
['SL5', 'SLU']

Team ID in question
BRO
['BR2', 'BR3', 'BRO', 'LAN']

Team ID in question
DET
['DET']

Team ID in question
BR3
['BR2', 'BR3', 'BRO', 'LAN']

Team ID in question
NEW
['IN2', 'IND', 'IN3', 'NEW']

Team ID in question
SFN
['NY1', 'SFN', 'NYP']

Team ID in question
BFP
['BFN', 'BFP']
```

```
Team ID in question
WS2
['MON', 'WS7', 'WS2', 'WS4', 'WS1', 'MIN', 'WS8', 'WSU', 'WAS', 'TEX', 'WS6']

Team ID in question
PHA
['PH1', 'PHA', 'PH4', 'KC1', 'OAK', 'PHN', 'PHP']

Team ID in question
SEA
['SEA']

Team ID in question
NYP
['NY1', 'SFN', 'NYP']

Team ID in question
LAN
['BR2', 'BR3', 'BRO', 'LAN']

Team ID in question
OAK
['PH1', 'PHA', 'PH4', 'KC1', 'OAK', 'PHN', 'PHP']

Team ID in question
NYN
['NYN']

Team ID in question
RC1
['RC1']

Team ID in question
ML2
['ML2']

Team ID in question
WIL
['WIL']

Team ID in question
TRO
['TRO']

Team ID in question
BS1
['BS1']
```

```
Team ID in question
RC2
['RC2']

Team ID in question
ELI
['ELI']

Team ID in question
TRN
['TRN']

Team ID in question
CL4
['CLE', 'CL2', 'CL4', 'CL3']

Team ID in question
CIN
['CN2', 'CN1', 'CIN']

Team ID in question
BR2
['BR2', 'BR3', 'BRO', 'LAN']

Team ID in question
NH1
['NH1']

Team ID in question
IN3
['IN2', 'IND', 'IN3', 'NEW']

Team ID in question
WS5
['WS5']

Team ID in question
BSU
['BSP', 'BS2', 'BSU']

Team ID in question
RIC
['RIC']

Team ID in question
TL1
['TL1']
```

```
Team ID in question
WS3
['WS3']

Team ID in question
WS7
['WS7', 'MON', 'WS2', 'WS4', 'WS1', 'MIN', 'WS8', 'WSU', 'WAS', 'TEX', 'WS6']

Team ID in question
SL5
['SL5', 'SLU']

Team ID in question
FLO
['MIA', 'FLO']

Team ID in question
TL2
['TL2']

Team ID in question
WS4
['WS7', 'MON', 'WS2', 'WS4', 'WS1', 'MIN', 'WS8', 'WSU', 'WAS', 'TEX', 'WS6']

Team ID in question
COL
['COL']

Team ID in question
BRF
['BRF']

Team ID in question
BOS
['BOS']

Team ID in question
SPU
['SPU']

Team ID in question
CHA
['CHA', 'CHN', 'CH2', 'CH1']

Team ID in question
CLE
['CLE', 'CL2', 'CL4', 'CL3']
```

```
Team ID in question
CNU
['CNU']

Team ID in question
CL2
['CLE', 'CL2', 'CL4', 'CL3']

Team ID in question
SL4
['BLA', 'MLU', 'SLN', 'SL3', 'MIL', 'NYA', 'SL4', 'MLA', 'BL2', 'BAL', 'ML3',
'SLA', 'SL2', 'SE1', 'ML4', 'BL3', 'BLN']

Team ID in question
SR2
['SR2', 'SR1']

Team ID in question
DTN
['DTN']

Team ID in question
KCF
['KCF']

Team ID in question
CH1
['CHA', 'CHN', 'CH2', 'CH1']

Team ID in question
KCU
['KCU', 'KC2', 'KCN']

Team ID in question
SDN
['SDN']

Team ID in question
PH4
['PH1', 'PHA', 'PH4', 'KC1', 'OAK', 'PHN', 'PHP']

Team ID in question
SLA
['BLA', 'MLU', 'SLN', 'SL3', 'MIL', 'NYA', 'SL4', 'MLA', 'BL2', 'BAL', 'ML3',
'SLA', 'SL2', 'SE1', 'ML4', 'BL3', 'BLN']

Team ID in question
IND
```

```
['IN2', 'IND', 'IN3', 'NEW']

Team ID in question
TEX
['MON', 'WS7', 'WS2', 'WS4', 'WS1', 'MIN', 'WS8', 'WSU', 'WAS', 'TEX', 'WS6']

Team ID in question
TBA
['TBA']

Team ID in question
CH2
['CHA', 'CHN', 'CH2', 'CH1']

Team ID in question
LS3
['LS3', 'LS2']

Team ID in question
TOR
['TOR']

Team ID in question
SL1
['SL1']

Team ID in question
PHU
['PHU']

Team ID in question
BR4
['BR4']

Team ID in question
BLN
['BLA', 'MLU', 'NYA', 'SLN', 'MIL', 'SL3', 'BL2', 'MLA', 'SL4', 'BAL', 'ML3',
'SLA', 'SL2', 'SE1', 'ML4', 'BL3', 'BLN']

Team ID in question
FW1
['FW1']

Team ID in question
WS9
['WS7', 'MON', 'WS2', 'WS4', 'WS1', 'MIN', 'WS9', 'WS8', 'WSU', 'WAS', 'TEX',
'WS6']
```

```
Team ID in question
CHP
['CHP']

Team ID in question
WS6
['WS7', 'MON', 'WS2', 'WS4', 'WS1', 'MIN', 'WS8', 'WSU', 'WAS', 'TEX', 'WS6']

Team ID in question
MIN
['MON', 'WS7', 'WS2', 'WS4', 'WS1', 'MIN', 'WS8', 'WSU', 'WAS', 'TEX', 'WS6']

Team ID in question
BSP
['BSP', 'BS2', 'BSU']

Team ID in question
KCA
['KCA']

Team ID in question
ML1
['ATL', 'ML1', 'BSN']

Team ID in question
BL1
['BL1']

Team ID in question
BL4
['BL4']

Team ID in question
ATL
['ATL', 'ML1', 'BSN']

Team ID in question
PT1
['PT1', 'PIT']

Team ID in question
BUF
['BUF']

Team ID in question
SL2
['BLA', 'MLU', 'SLN', 'SL3', 'MIL', 'NYA', 'SL4', 'MLA', 'BL2', 'BAL', 'ML3',
'SLA', 'SL2', 'SE1', 'ML4', 'BL3', 'BLN']
```

```
Team ID in question
BAL
['BLA', 'MLU', 'NYA', 'SLN', 'MIL', 'SL3', 'BL2', 'MLA', 'SL4', 'BAL', 'ML3',
'SLA', 'SL2', 'SE1', 'ML4', 'BL3', 'BLN']

Team ID in question
LAA
['LAA', 'ANA', 'CAL']

Team ID in question
SR1
['SR2', 'SR1']

Team ID in question
MIA
['MIA', 'FLO']

Team ID in question
MID
['MID']

Team ID in question
CAL
['LAA', 'ANA', 'CAL']

Team ID in question
ANA
['LAA', 'ANA', 'CAL']

Team ID in question
BRP
['BRP']

Team ID in question
BR1
['BR1']

Team ID in question
CL3
['CLE', 'CL2', 'CL4', 'CL3']

Team ID in question
CN2
['CN2', 'CN1', 'CIN']

Team ID in question
NY3
```

```
['NY3', 'NY2']

Team ID in question
CL6
['CL6']

Team ID in question
ML3
['BLA', 'MLU', 'NYA', 'SLN', 'MIL', 'SL3', 'BL2', 'MLA', 'SL4', 'BAL', 'ML3',
'SLA', 'SL2', 'SE1', 'ML4', 'BL3', 'BLN']

Team ID in question
IN1
['IN1']

Team ID in question
SE1
['BLA', 'MLU', 'NYA', 'SLN', 'MIL', 'SL3', 'BL2', 'MLA', 'SL4', 'BAL', 'ML3',
'SLA', 'SL2', 'SE1', 'ML4', 'BL3', 'BLN']
```

This database is actually very messy. Through links between same team names, to same franchise ids, it connects New York Yankees to Baltimore Orioles. However, we do get better matches like IN2 nad IN3, for Indianapolis Hoosiers that we might have missed before. We can conclude that the data is in fact extremely messy and almost to a point of beyond fixing unless this fixing is done by hand (reading baseball history), which can take hours for just a few teams. If a database like this were to be built again, then adding a column called uniqID will help, so that baseball teams are linked properly even when franchID, teamID, and team name changes. If we try to repair links here using our recursive function, we end up with better connections, but also messed up data with too many bad connections.

I could have easily matched on my new column for team_ID, but for the purposes of the project and matching on the matching columns between Salaries and Teams (teamID, yearID), I matched simply on teamID. Also, even if I did do that it would be wrong to say NYA is the same as BAL (very different teams).

5.6 End of extra stuff out of scope of the project