Python Sample

April 20, 2025

[ ]:
```
#NOTE: These stats have gotten scewed since the 2024-25 NHL playoffs have␣
↪started because the API I used reset some stats to just record
#playoff stats which is why some of the graphs pictured below might look off
```

[ ]:
```
#GOAL: Make a predictor that gives you the best contenders for the Stanley Cup␣
↪based on 4 average stats of past winners
```

[1]:
```
import csv
import pandas as pd
import requests
import numpy as np
from bs4 import BeautifulSoup
```

[2]:
```
pd.options.display.max_columns = 100
```

[3]:
```
Winners = pd.read_csv("StanleyCupWinners.csv")
#cuts off Lg (League),T (Ties), and OL (overtime loses) becuase they are all in␣
↪the NHL and ties used to be a thing before 2005 and overtime
#loses used to not be a thing
#PDO (shoot % + save %)was also taken out because they didn't start calculating␣
↪that before 2005
Winners = Winners.drop(columns = ["Lg"])
Winners = Winners.drop(columns = ["T"])
Winners = Winners.drop(columns = ["OL"])
Winners = Winners.drop(columns = ["PDO"])
Winners = Winners.drop(columns = ["AvAge"])
#create a new dataframe to itterate through the standard deviation so taking␣
↪out Team name and Season
Sdv_Winners = Winners.drop(columns = ["Team"])
Sdv_Winners = Sdv_Winners.drop(columns = ["Season"])
#we also have to get rid of any season before the 1963-64 season because there␣
↪are way less stats for Stanley Cup Winners before that
#(No power play or penatly kill stats)
'''had to look up how to get the column names'''
Winners = Winners[Winners["Season"] > "1963-1964"]
#get the standard deviation for each stat then choose a mark that cuts off any␣
↪higher standard deviations
```

[4]:
```
#create a function to calculate standard deviation that will be called through␣
 ↪for each stat
def StandardDev(stat):
    sdDev = np.std(stat)
    return sdDev
#creates a function to calculate the average for each stat
def Mean(stat):
    avg = np.mean(stat)
    return avg

#creates two new pandas dataframe to store the standard deviaiton amounts and␣
 ↪average amounts
sdvDF = pd.DataFrame()
avgDF = pd.DataFrame()
# loops through each column in the Sdv_Winners dataframe
for column in Sdv_Winners.columns:
    #have to make it a list in order to pass it through the standard deviation␣
 ↪function
    stats_list = Winners[column].tolist()

    #in the new standard deviation and average dataframes make the column names␣
 ↪equal to the name of the column itterated through plus _sdv
    #or _avgthen sets the value equal to the value after called into the␣
 ↪StandardDev and Mean function
    sdvDF[column + "_sdv"] = [(StandardDev(stats_list))]
    avgDF[column + "_avg"] = [(Mean(stats_list))]
```

[5]:
```
#use requests to get the api, in json form, and store it as league
league = requests.get("https://api-web.nhle.com/v1/standings/now").json()
#gets just the standings of the NHL currently
Standings = league["standings"]

#a function that calculates the save percent for a whole team using a for loop␣
 ↪for each goalie on every team
def savePer(tender):
    shotsA = 0
    saves = 0
    for goalie in tender:
        saves += goalie["saves"]
        shotsA += goalie["shotsAgainst"]
        svPer = saves/shotsA
    return svPer
#create a dataframe for all the teams current stats set the header names to␣
 ↪Team Names and then all the stats that have a standard deviation
#of lower than 1 (the closer the number is to 0 the less variance there is in␣
 ↪Stanley Cup winners stats which shows correlation in the stat
#for all the winners).
#Had to leave out SRS (Simple Rating System) and SOS (Strength of Schedule)␣
 ↪because the API call doesn't have those stats
currStatsDF = pd.DataFrame(columns = ["Team Name","Abrev","Points %","Goals For␣
 ↪per Game","Goals Against per Game","Save %"])

#create lists to store all the teams and the stats we will use to add into the␣
 ↪dataframe above
#also creates a list of team abreviation in order to get save percentage
teamList = []
pointper = []
GFpG = []
GApG = []
saveper = []
teamabr = []

#make a for loop to itterate through every team in standings and append the␣
 ↪team name, point %, goals for per game, goals against per game,
#and save %
#have to calculate goals for and against per game by taking each teams goals␣
 ↪for and against then dividing it by the games played
#for save percent I had to get the team abbreviation then feed every team into␣
 ↪another api call to get
#each teams save percentage
for team in Standings:
    teamList.append(team["teamName"]["default"])
    pointper.append(team["pointPctg"])
    GFpG.append((team["goalFor"]/team["gamesPlayed"]))
    GApG.append((team["goalAgainst"]/team["gamesPlayed"]))
    teamabr.append(team["teamAbbrev"]["default"])

#itterates through the list of abbreviations and puts them into an api request␣
 ↪that gets the stats of players
#from each team. Then specifies it to goalie stats only and passes the goalies␣
 ↪stats into a function that
#calculates the save percent total for each team

for abv in teamabr:
    team_stats = requests.get("https://api-web.nhle.com/v1/club-stats/
 ↪"+str(abv)+"/now").json()
    goalies = team_stats["goalies"]
    saveper.append(savePer(goalies))

#adds all the list values for each stat into the dataframe for current stats
currStatsDF["Team Name"] = teamList
currStatsDF["Abrev"] = teamabr
```

```
currStatsDF["Points %"] = pointper
currStatsDF["Goals For per Game"] = GFpG
currStatsDF["Goals Against per Game"] = GApG
currStatsDF["Save %"] = saveper
```

[ ]:

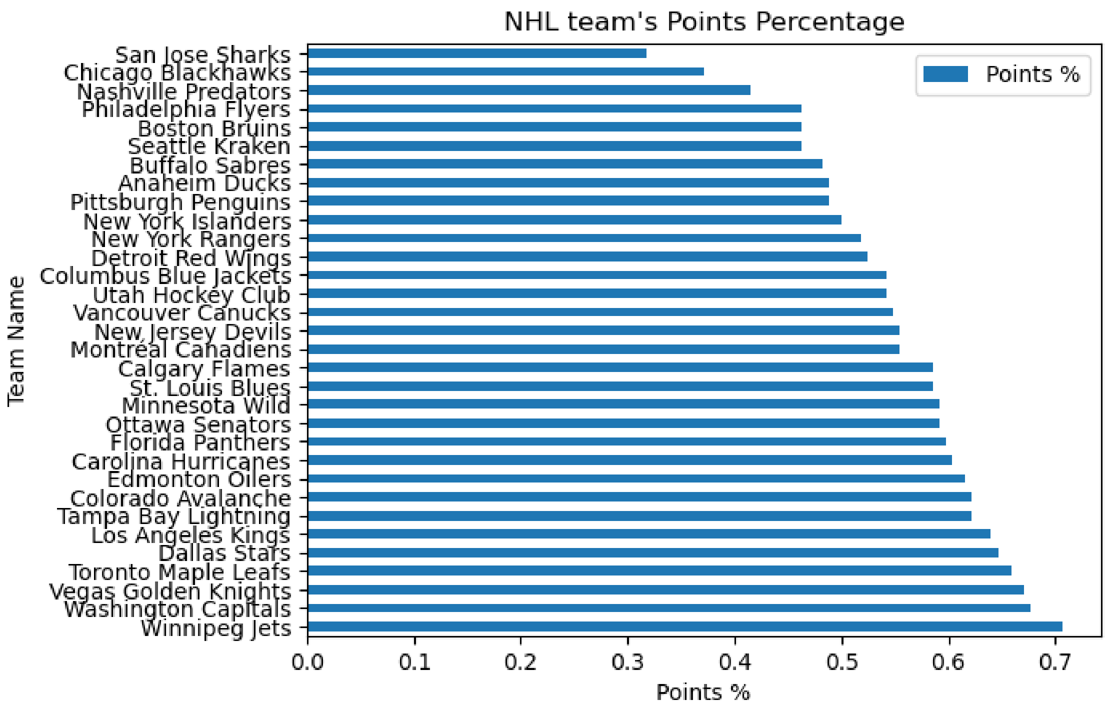
[ ]:

[ ]:

[21]:
```
#sorts all the stats I'm looking at for the current NHL teams and sort them for␣
↪top to bottom or bottom to top
points_percent_leader = currStatsDF.sort_values("Points %",ascending = False)
GFpG = currStatsDF.sort_values("Goals For per Game", ascending = False)
GAfG = currStatsDF.sort_values("Goals Against per Game", ascending = True)
Save_per = currStatsDF.sort_values("Save %", ascending = True)
```

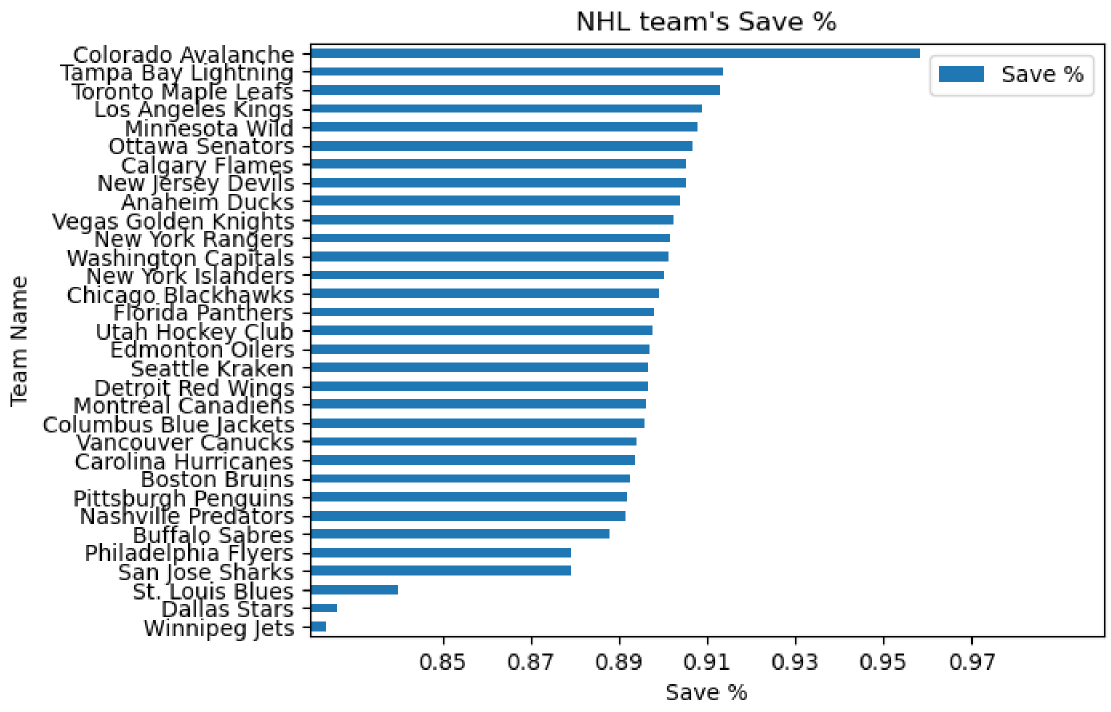
[7]:
```
#horizontal bar graph for Points %
Points_graph = points_percent_leader.plot(kind ="barh", x ="Team Name", y=␣
↪"Points %")
Points_graph.set_title("NHL team's Points Percentage")
Points_graph.set_xlabel("Points %")
```

[7]:
```
Text(0.5, 0, 'Points %')
```



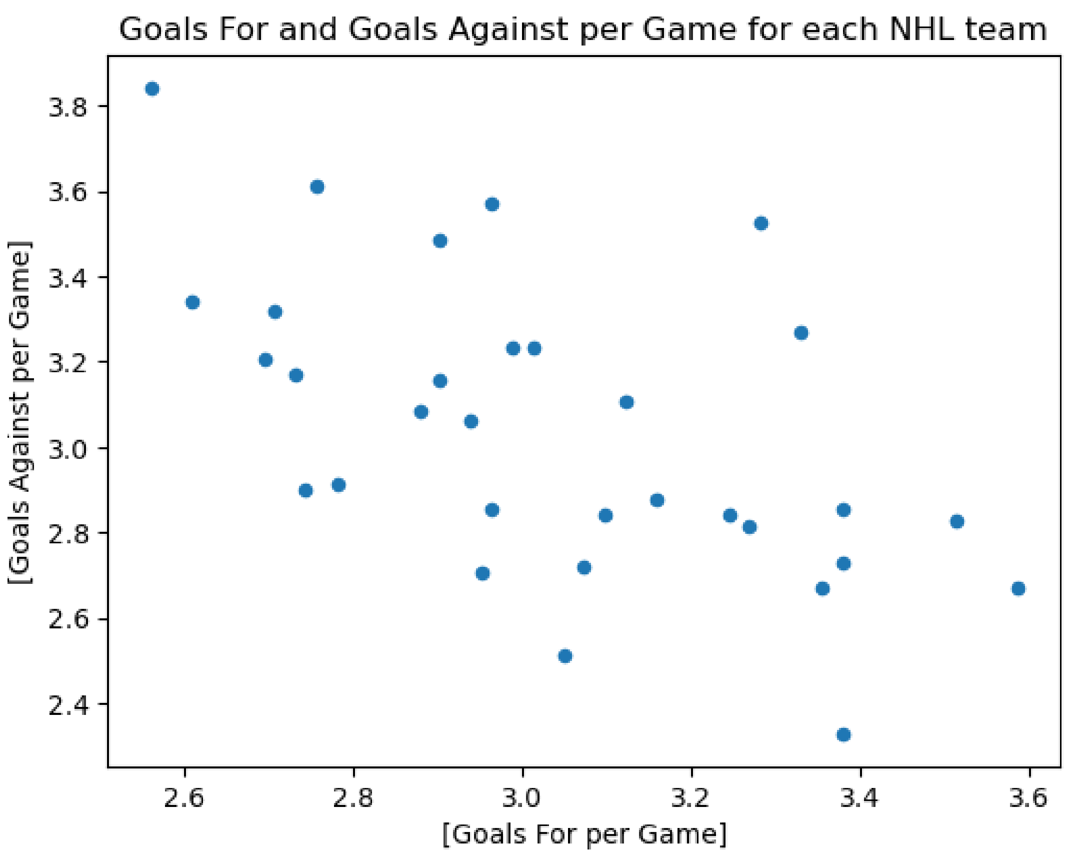
```
[8]: Sv_graph = Save_per.plot(kind = "barh", x="Team Name", y="Save %")
     Sv_graph.set_xticks([0.85, 0.87, 0.89, 0.91, 0.93, 0.95, 0.97])
     Sv_graph.set_xlim(0.82, 1.00)
     Sv_graph.set_title("NHL team's Save %")
     Sv_graph.set_xlabel("Save %")
```

```
[8]: Text(0.5, 0, 'Save %')
```



```
[45]: GF_GA_NHL = currStatsDF.plot.scatter(x=["Goals For per Game"], y= ["Goals␣
       ↪Against per Game"])
      GF_GA_NHL.set_title("Goals For and Goals Against per Game for each NHL team")
```

```
[45]: Text(0.5, 1.0, 'Goals For and Goals Against per Game for each NHL team')
```



[38]:
```
#based on the standard deviations and average stats for Stanley Cup winners␣
↪narrow down the current teams to show what teams have
#above the average stat that past Stanley Cup winners team had
#the teams that are put into Current_Cup_Prospects will update every game as␣
↪the api updates
Current_Cup_Prospects = currStatsDF.query("`Points %` >= 0.658233 and `Save %`␣
↪>= 0.905 and `Goals For per Game` >= 3.568 and `Goals Against per Game` <= 2.
↪699")

#return are the teams that are above the average stat that each Stanley Cup␣
↪winning team had
Current_Cup_Prospects
```

[38]:
```
Empty DataFrame
Columns: [Team Name, Abrev, Points %, Goals For per Game, Goals Against per
Game, Save %]
Index: []
```

[11]:
```
#read in last season stats in html form using an api call
LastSeason = requests.get("https://www.hockey-reference.com/leagues/NHL_2024.
 ↪html").text
#removing comments
OneComments = LastSeason.replace("<!--", "")
NoComments = OneComments.replace("-->","")
#use BeautifulSoup
LastSeasonSoup = BeautifulSoup(NoComments)
#use .find to get to the table with the id=div_stats
LSstats = LastSeasonSoup.find("div", {"id":"div_stats"})
```

[32]:
```
#make a list for the headers
#headers = []
#use .find to get the headers
#thead_headers = LSstats.find("thead")
#headers = thead_headers.find_all("th")

#print(body)
#print(thead_headers)
#print(type(rows))
#for row in headers:
    #print(row)
    #if row != None:
        #row_text = row.strip()
    #if row_text in needed_headers:
        #headers.append(row_text)

#everything above is what I did to try to get the headers neaturally

#create a list of all the stats I need
needed_headers = ["Team Name","PTS%","GF/G","GA/G","SV%"]
teamName = []
pper = []
gfg = []
gag = []
svper = []
#create a dataframe for last years teams with the needed stats
LastSznDF = pd.DataFrame(columns = needed_headers)
LastSznDF
#uses .find to get the body where the stats are
body = LSstats.find("tbody")
rows = body.find_all("tr")
for row in rows:
    teamName.append(row.find("td",{"data-stat":"team_name"}).text.strip())
    pper.append(row.find("td",{"data-stat":"points_pct"}).text.strip())
    gfg.append(row.find("td",{"data-stat":"goals_for_per_game"}).text.strip())
    gag.append((row.find("td",{"data-stat":"goals_against_per_game"}).text.
↪strip()))
    svper.append(row.find("td",{"data-stat":"save_pct"}).text.strip())

#function that makes all the stats added into INT
def makeINT(listx):
    new_list = []
    for item in listx:
        float(item)
        new_list.append(item)
    return new_list

#puts all the numbers through the makeINT function
INTpper = makeINT(pper)
INTgfg = makeINT(gfg)
INTgag = makeINT(gag)
INTsvper = makeINT(svper)

#adds all the stats from last season into LastSznDF
LastSznDF["Team Name"] = teamName
LastSznDF["PTS%"] = INTpper
LastSznDF["GF/G"] = INTgfg
LastSznDF["GA/G"] = INTgag
LastSznDF["SV%"] = INTsvper
```

[37]:
```
#convert all the data from object type into floats so that I can run a DF.query
LastSznDF["PTS%"] = LastSznDF["PTS%"].astype(float)
LastSznDF["SV%"] = LastSznDF["SV%"].astype(float)
LastSznDF["GF/G"] = LastSznDF["GF/G"].astype(float)
LastSznDF["GA/G"] = LastSznDF["GA/G"].astype(float)

#gives the teams from last season above the average stat that past Stanley Cup␣
↪winners team had
LastSznContenders = LastSznDF.query("`PTS%` >= 0.658233 and `SV%` >= 0.905 and␣
↪`GF/G` >= 3.568 and `GA/G` <= 2.699")

LastSznContenders
```

[37]:
```
Empty DataFrame
Columns: [Team Name, PTS%, GF/G, GA/G, SV%]
Index: []
```

[43]:
```
#FINDINGS: I wanted to make a predictor of Stanley Cup contenders based on the␣
↪statistics of past Stanley Cup Winners.
#After running my predictor with this year and last years NHL stats it became␣
↪apparent that my predictor didn't work well because
```

```
#no teams were returned after my query on average of Save Percentage, Points␣
 ↪Percentage, Goals For per Game, and Goals Against per Game
#for past Stanley Cup winners. This was due to two main issues:
#1. Setting the average of past winners: Using 4 average stats of the past␣
 ↪winners assumes that all past winners had the average stat
#or better for all 4 statistics when in relaity it's is more likely that some␣
 ↪teams were better defensivly or offensivly and had a better
#GA/G and Save % but not a better GF/G and my query is looking for teams that␣
 ↪are better or equal in every stat category
#2. How hockey is played has evolved over the years and the amount of goals␣
 ↪scored has shifted making some of the older data scewed compared
#current times with GF/G, GA/G, and SV%
```

[44]:
```
#Possible Fix: Either adding percentiles to try and expand the qualifications␣
 ↪or switching the ands to ors so that only one stat has to be
#above the average of past Stanley Cup winners but that way gets you more teams␣
 ↪than you want for or make it so NHL teams only have to
#have equal or better than two stats out of the 4 stats
```

[ ]:
```

```