Dylan Webb Tennis Data Project

December 10, 2020

1 Appendices

1.1 Appendix A - code used to generate the tennis.csv and stats.csv files

[ ]:
```
#TENNIS STATS GENERATOR
#WRITTEN BY DYLAN WEBB 11.28.20

import numpy as np
import pandas as pd

#Generate statistics summarizing past player performance
def generateStats(player, opponent, year, stat):
  df = pd.read_csv("tennis.csv")

  yearMax = df["year"] < year
  yearMin = df["year"] >= year - 5
  past_mask = yearMin & yearMax

  df = df[past_mask]

  playerW = df["winner_name"] == player
  playerL = df["loser_name"] == player

  opponentW = df["winner_name"] == opponent
  opponentL = df["loser_name"] == opponent

  pCommonOpponentWStat = 0
  pCommonOpponentLStat = 0
  pCommonOpponentWVar = 0
  pCommonOpponentLVar = 0

  oCommonOpponentWStat = 0
  oCommonOpponentLStat = 0
  oCommonOpponentWVar = 0
  oCommonOpponentLVar = 0

  #Calculate Player Statistics
  playerWStat = np.mean(df[playerW]["w_" + stat])
```

```python
playerLStat = np.mean(df[playerL]["l_" + stat])

playerWVar = np.var(df[playerW]["w_" + stat])
playerLVar = np.var(df[playerL]["l_" + stat])

#Calculate Opponent Statistics
opponentWStat = np.mean(df[opponentW]["w_" + stat])
opponentLStat = np.mean(df[opponentL]["l_" + stat])

opponentWVar = np.var(df[opponentW]["w_" + stat])
opponentLVar = np.var(df[opponentL]["l_" + stat])

#find common opponents
opponents1 = pd.DataFrame(pd.concat([df[playerW]["loser_name"],
                                     df[playerL]["winner_name"]]))
opponents2 = pd.DataFrame(pd.concat([df[opponentW]["loser_name"],
                                     df[opponentL]["winner_name"]]))
opponents1.drop_duplicates(keep = "first", inplace = True)
opponents2.drop_duplicates(keep = "first", inplace = True)

commonOpponents = pd.merge(opponents1, opponents2)

#calculate Player Common Opponent Statistics
winTotal = np.empty(0)
loseTotal = np.empty(0)

for i in range(len(commonOpponents)):
  commonOpponentW = df["winner_name"] == commonOpponents.iloc[i][0]
  commonOpponentL = df["loser_name"] == commonOpponents.iloc[i][0]

  playerW_mask = playerW & commonOpponentL
  playerL_mask = playerL & commonOpponentW

  winTotal = np.append(winTotal, df[playerW_mask]["w_" + stat].values)
  loseTotal = np.append(loseTotal, df[playerL_mask]["l_" + stat].values)

if winTotal.size > 0 and loseTotal.size > 0:
  pCommonOpponentWStat = np.mean(winTotal)
  pCommonOpponentLStat = np.mean(loseTotal)

  pCommonOpponentWVar = np.var(winTotal)
  pCommonOpponentLVar = np.var(loseTotal)
else:
  playerWStat = float("NaN")

#calculate Opponent Common Opponent Statistics
winTotal = np.empty(0)
```

```
  loseTotal = np.empty(0)

  for i in range(len(commonOpponents)):
    commonOpponentW = df["winner_name"] == commonOpponents.iloc[i][0]
    commonOpponentL = df["loser_name"] == commonOpponents.iloc[i][0]

    opponentW_mask = opponentW & commonOpponentL
    opponentL_mask = opponentL & commonOpponentW

    winTotal = np.append(winTotal, df[opponentW_mask]["w_" + stat].values)
    loseTotal = np.append(loseTotal, df[opponentL_mask]["l_" + stat].values)

  if winTotal.size > 0 and loseTotal.size > 0:
    oCommonOpponentWStat = np.mean(winTotal)
    oCommonOpponentLStat = np.mean(loseTotal)

    oCommonOpponentWVar = np.var(winTotal)
    oCommonOpponentLVar = np.var(loseTotal)
  else:
    playerWStat = float("NaN")

  results = pd.DataFrame([[playerWStat, playerLStat, opponentWStat,
                           opponentLStat, pCommonOpponentWStat,
                           pCommonOpponentLStat, oCommonOpponentWStat,
                           oCommonOpponentLStat, playerWVar, playerLVar,
                           opponentWVar, opponentLVar, pCommonOpponentWVar,
                           pCommonOpponentLVar, oCommonOpponentWVar,
                           oCommonOpponentLVar]])

  results.columns = [stat + "player_w", stat + "player_l",
                     stat + "opponent_w", stat + "opponent_l",
                     stat + "playerCommon_w", stat + "playerCommon_l",
                     stat + "opponentCommon_w", stat + "opponentCommon_l",
                     stat + "pVariance_w", stat + "pVariance_l",
                     stat + "oVariance_w", stat + "oVariance_l",
                     stat + "pCommonVar_w", stat + "pCommonVar_l",
                     stat + "oCommonVar_w", stat + "oCommonVar_l"]

  return results

#CREATE TENNIS.CSV FILE
#Cleans up data for calculations
df = pd.read_csv("tennis_atp-master/atp_matches_2003.csv")
df["year"] = 2003
for year in range(2004,2020):
  file = "tennis_atp-master/atp_matches_" + str(year) + ".csv"
```

```python
  newdf = pd.read_csv(file)
  newdf["year"] = year
  df = df.append(newdf, ignore_index=True)

mask1 = df["tourney_name"] == "Australian Open"
mask2 = df["tourney_name"] == "Roland Garros"
mask3 = df["tourney_name"] == "US Open"
mask4 = df["tourney_name"] == "Wimbledon"

mask = mask1 | mask2 | mask3 | mask4

df = df[mask]

#Feature engineering
df["w_2ndIn"] = df["w_svpt"] - df["w_1stIn"]
df["l_2ndIn"] = df["l_svpt"] - df["l_1stIn"]

df["w_svWon%"] = (df["w_1stWon"] + df["w_2ndWon"]) / df["w_svpt"]
df["l_svWon%"] = (df["l_1stWon"] + df["l_2ndWon"]) / df["l_svpt"]

df["w_1stRnWon%"] = (df["l_1stIn"] - df["l_1stWon"]) / df["l_1stIn"]
df["l_1stRnWon%"] = (df["w_1stIn"] - df["w_1stWon"]) / df["w_1stIn"]

df["w_2ndRnWon%"] = (df["l_2ndIn"] - df["l_2ndWon"]) / df["l_2ndIn"]
df["l_2ndRnWon%"] = (df["w_2ndIn"] - df["w_2ndWon"]) / df["w_1stIn"]

#label encode surface
df["surface"] = df["surface"].astype("category")
df["surface"] = df["surface"].cat.codes
df["surface"] /= 2

#remove unecessary columns and output tennis.csv
df = df[["year", "tourney_name", "surface", "winner_name", "loser_name",
         "w_svWon%", "w_1stRnWon%", "w_2ndRnWon%",
         "l_svWon%", "l_1stRnWon%", "l_2ndRnWon%"]]

df.dropna(how = 'any', inplace = True)
df.to_csv('tennis.csv', index = False)

#CREATE STATS.CSV FILE
#remove first five years for generateStats function
year_mask = df["year"] >= 2008
df = df[year_mask]

outcome = pd.DataFrame(np.zeros((len(df),8)))
outcome.columns = ["player1", "player2", "year", "surface",
                   "2ndRnWon%", "svWon%", "1stRnWon%", "outcome"]
```

```python
#randomly rearrange data frame and assign statistics
for i in range(len(df)):
  outcome.loc[i, "year"] = df["year"].iloc[i]
  outcome.loc[i, "surface"] = df["surface"].iloc[i]
  p = np.random.random()
  if p < .5:
    outcome.loc[i, "outcome"] = 1
    outcome.loc[i, "player1"] = df["winner_name"].iloc[i]
    outcome.loc[i, "player2"] = df["loser_name"].iloc[i]
    outcome.loc[i, "2ndRnWon%"] = df["w_2ndRnWon%"].iloc[i]
    outcome.loc[i, "svWon%"] = df["w_svWon%"].iloc[i]
    outcome.loc[i, "1stRnWon%"] = df["w_1stRnWon%"].iloc[i]
  else:
    outcome.loc[i, "player1"] = df["loser_name"].iloc[i]
    outcome.loc[i, "player2"] = df["winner_name"].iloc[i]
    outcome.loc[i, "2ndRnWon%"] = df["l_2ndRnWon%"].iloc[i]
    outcome.loc[i, "svWon%"] = df["l_svWon%"].iloc[i]
    outcome.loc[i, "1stRnWon%"] = df["l_1stRnWon%"].iloc[i]

#run generate stats function on every player matchup
stat1 = generateStats(outcome["player1"].iloc[0], outcome["player2"].iloc[0],
                      outcome["year"].iloc[0], "2ndRnWon%")
stat2 = generateStats(outcome["player1"].iloc[0], outcome["player2"].iloc[0],
                      outcome["year"].iloc[0], "1stRnWon%")
stat3 = generateStats(outcome["player1"].iloc[0], outcome["player2"].iloc[0],
                      outcome["year"].iloc[0], "svWon%")

for i in range(1,len(df)):
  stat1 = stat1.append(generateStats(outcome["player1"].iloc[i],
                                     outcome["player2"].iloc[i],
                                    outcome["year"].iloc[i], "2ndRnWon%"),
                       ignore_index = True)

  stat2 = stat2.append(generateStats(outcome["player1"].iloc[i],
                                     outcome["player2"].iloc[i],
                                    outcome["year"].iloc[i], "1stRnWon%"),
                       ignore_index = True)

  stat3 = stat3.append(generateStats(outcome["player1"].iloc[i],
                                     outcome["player2"].iloc[i],
                                    outcome["year"].iloc[i], "svWon%"),
                       ignore_index = True)

#combine data frames and output stats.csv
outcome = outcome.drop(["player1", "player2"], axis = 1)
stats = outcome.join(stat1)
```

```
stats = stats.join(stat2)
stats = stats.join(stat3)

stats.dropna(how = 'any', inplace = True)
stats.to_csv('stats.csv', index = False)
```

1.2 Appendix B - code used to predict the winners of tennis matches

[ ]:
```
#TENNIS PREDICTION MODEL
#WRITTEN BY DYLAN WEBB 11.28.20

import numpy as np
import pandas as pd

from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split

#ROUND ROBIN PREDICTION

def roundRobin(nameList, year, tourney):
  print("Round Robin")
  df = nameList
  df["score"] = 0
  matches = pd.DataFrame()

  for i in range(len(df) - 1):

    #match player against players they haven't yet played
    names = pd.DataFrame(df["name"])
    for j in range(i + 1):
      names = names.drop(j)
    names.columns = ["player1"]
    names["player2"] = df["name"].iloc[i]

    if i == 0:
      matches = names
    else:
      matches = matches.append(names)

  #predicts outcomes of all matches
  outcome = predictionModel(matches, year, tourney)

  #sort results into the score column
  for i in range(len(outcome)):
    for j in range(len(df)):
      if df["name"].iloc[j] == outcome["predicted_winner"].iloc[i]:
```

```
      df.loc[j, "score"] += 1

  pd.set_option('display.max_rows', None)
  df.index += 1
  df = df.sort_values(by = "score", ascending = False)
  print(df)

#TOURNAMENT WINNER PREDICTION

def predictTournament(round1, year, tourney, winner = False):
  pd.set_option('display.max_rows', None)

  prediction = predictionModel(round1, year, tourney)

  r = 0
  while len(prediction) > 1:
    r += 1
    if winner == False:
      print("\nRound", r)
      print(prediction)

    nextRound = pd.DataFrame(np.zeros((int(len(prediction)/2),2))).astype(str)
    nextRound.columns = ["player1", "player2"]
    for i in range(len(prediction)):
      if i % 2 == 0:
        nextRound.loc[int(i/2), "player1"] = prediction["predicted_winner"
        ].iloc[i]
      else:
        nextRound.loc[int((i-1)/2), "player2"] = prediction["predicted_winner"
        ].iloc[i]

    prediction = predictionModel(nextRound, 2019, "Wimbledon")

  print("\nFinal Round")
  print(prediction)

#DOUBLE FOREST TENNIS PREDICTION MODEL

def predictionModel(names, year, tourney):

  #Generate features for regressor forest
  features1 = approximateFeatures(names["player1"].iloc[0],
                                  names["player2"].iloc[0],
                                  year, tourney, "2ndRnWon%")
  features2 = approximateFeatures(names["player1"].iloc[0],
                                  names["player2"].iloc[0],
                                  year, tourney, "svWon%")
```

```
  features3 = approximateFeatures(names["player1"].iloc[0],
                                  names["player2"].iloc[0],
                                  year, tourney, "1stRnWon%")

  for i in range(1, len(names)):
    features1 = features1.append(approximateFeatures(names["player1"].iloc[i],
                                                     names["player2"].iloc[i],
                                                     year, tourney, "2ndRnWon%"),
                                 ignore_index = True)

    features2 = features2.append(approximateFeatures(names["player1"].iloc[i],
                                                     names["player2"].iloc[i],
                                                     year, tourney, "svWon%"),
                                 ignore_index = True)

    features3 = features3.append(approximateFeatures(names["player1"].iloc[i],
                                                     names["player2"].iloc[i],
                                                     year, tourney, "1stRnWon%"),
                                 ignore_index = True)

  #classifier forest takes input from regressor forest to predict the winner
  #run 15 times total to capture average forest vote
  p = np.zeros((len(names)))
  for i in range(15):
    p = np.add(p, forestClassify(pd.DataFrame(forestRegress(features1,
                                                            features2,
                                                            features3, year))))
  p /= 15

  #stores predictions in a dataframe of names to return
  prediction = pd.DataFrame(np.zeros((len(names),2)))
  prediction.columns = ["predicted_winner","likelihood"]
  for i in range(len(names)):
    if p[i] > .5:
      prediction.loc[i, "predicted_winner"] = names["player1"].iloc[i]
      prediction.loc[i, "likelihood"] = p[i]
    else:
      prediction.loc[i, "predicted_winner"] = names["player2"].iloc[i]
      prediction.loc[i, "likelihood"] = 1 - p[i]

  return prediction

#APPROXIMATE FEATURES ALGORITHM

def approximateFeatures(player, opponent, year, tourney, stat):
  df = pd.read_csv("tennis.csv")
  stats = pd.read_csv("stats.csv")
```

```
yearMax = df["year"] < year
yearMin = df["year"] >= year - 5
past_mask = yearMin & yearMax
df = df[past_mask]

yearMax = stats["year"] < year
yearMin = stats["year"] >= year - 5
past_mask = yearMin & yearMax
stats = stats[past_mask]

playerW = df["winner_name"] == player
playerL = df["loser_name"] == player

opponentW = df["winner_name"] == opponent
opponentL = df["loser_name"] == opponent

#average values to fill in case of missing data
#this is caused by players who don't usually compete in Grand Slams
meanOffset = .9
varOffset = .7

playerWStat = np.mean(stats[stat + "player_w"]) * meanOffset
playerLStat = np.mean(stats[stat + "player_l"]) * meanOffset
playerWVar = np.mean(stats[stat + "pVariance_w"]) * varOffset
playerLVar = np.mean(stats[stat + "pVariance_l"]) * varOffset

opponentWStat = np.mean(stats[stat + "opponent_w"]) * meanOffset
opponentLStat = np.mean(stats[stat + "opponent_l"]) * meanOffset
opponentWVar = np.mean(stats[stat + "oVariance_w"]) * varOffset
opponentLVar = np.mean(stats[stat + "oVariance_l"]) * varOffset

pCommonOpponentWStat = np.mean(stats[stat + "playerCommon_w"]) * meanOffset
pCommonOpponentLStat = np.mean(stats[stat + "playerCommon_l"]) * meanOffset
pCommonOpponentWVar = np.mean(stats[stat + "pCommonVar_w"]) * varOffset
pCommonOpponentLVar = np.mean(stats[stat + "pCommonVar_l"]) * varOffset

oCommonOpponentWStat = np.mean(stats[stat + "opponentCommon_w"]) * meanOffset
oCommonOpponentLStat = np.mean(stats[stat + "opponentCommon_l"]) * meanOffset
oCommonOpponentWVar = np.mean(stats[stat + "oCommonVar_w"]) * varOffset
oCommonOpponentLVar = np.mean(stats[stat + "oCommonVar_l"]) * varOffset

#Calculate Player Statistics
win = df[playerW]["w_" + stat]
lose = df[playerL]["l_" + stat]

if len(win) > 0:
```

```
  playerWStat = np.mean(win)
  playerWVar = np.var(win)
if len(lose) > 0:
  playerLStat = np.mean(lose)
  playerLVar = np.var(lose)

#Calculate Opponent Statistics
win = df[opponentW]["w_" + stat]
lose = df[opponentL]["l_" + stat]

if len(win) > 0:
  opponentWStat = np.mean(win)
  opponentWVar = np.var(win)
if len(lose) > 0:
  opponentLStat = np.mean(lose)
  opponentLVar = np.var(lose)

#find common opponents
opponents1 = pd.DataFrame(pd.concat([df[playerW]["loser_name"],
                                     df[playerL]["winner_name"]]))
opponents2 = pd.DataFrame(pd.concat([df[opponentW]["loser_name"],
                                     df[opponentL]["winner_name"]]))
opponents1.drop_duplicates(keep="first", inplace = True)
opponents2.drop_duplicates(keep="first", inplace = True)

commonOpponents = pd.merge(opponents1, opponents2)

#calculate Player Common Opponent Statistics
winTotal = np.empty(0)
loseTotal = np.empty(0)

for i in range(len(commonOpponents)):
  commonOpponentW = df["winner_name"] == commonOpponents.iloc[i][0]
  commonOpponentL = df["loser_name"] == commonOpponents.iloc[i][0]

  playerW_mask = playerW & commonOpponentL
  playerL_mask = playerL & commonOpponentW

  winTotal = np.append(winTotal, df[playerW_mask]["w_" + stat].values)
  loseTotal = np.append(loseTotal, df[playerL_mask]["l_" + stat].values)

if winTotal.size > 0:
  pCommonOpponentWStat = np.mean(winTotal)
  pCommonOpponentWVar = np.var(winTotal)
if loseTotal.size > 0:
  pCommonOpponentLStat = np.mean(loseTotal)
  pCommonOpponentLVar = np.var(loseTotal)
```

```python
#calculate Opponent Common Opponent Statistics
winTotal = np.empty(0)
loseTotal = np.empty(0)

for i in range(len(commonOpponents)):
  commonOpponentW = df["winner_name"] == commonOpponents.iloc[i][0]
  commonOpponentL = df["loser_name"] == commonOpponents.iloc[i][0]

  opponentW_mask = opponentW & commonOpponentL
  opponentL_mask = opponentL & commonOpponentW

  winTotal = np.append(winTotal, df[opponentW_mask]["w_" + stat].values)
  loseTotal = np.append(loseTotal, df[opponentL_mask]["l_" + stat].values)

if winTotal.size > 0:
  oCommonOpponentWStat = np.mean(winTotal)
  oCommonOpponentWVar = np.var(winTotal)
if loseTotal.size > 0:
  oCommonOpponentLVar = np.var(loseTotal)
  oCommonOpponentLStat = np.mean(loseTotal)

#label encode surface depending on tournament
surface = 1
if tourney == "Roland Garros":
  surface = 0
elif tourney == "Wimbledon":
  surface = .5

#store results and return
results = pd.DataFrame([[surface, playerWStat, playerLStat, opponentWStat,
                         opponentLStat, pCommonOpponentWStat,
                         pCommonOpponentLStat, oCommonOpponentWStat,
                         oCommonOpponentLStat, playerWVar, playerLVar,
                         opponentWVar, opponentLVar, pCommonOpponentWVar,
                         pCommonOpponentLVar, oCommonOpponentWVar,
                         oCommonOpponentLVar]])

results.columns = ["surface", stat + "player_w", stat + "player_l",
                   stat + "opponent_w", stat + "opponent_l",
                   stat + "playerCommon_w", stat + "playerCommon_l",
                   stat + "opponentCommon_w", stat + "opponentCommon_l",
                   stat + "pVariance_w", stat + "pVariance_l",
                   stat + "oVariance_w", stat + "oVariance_l",
                   stat + "pCommonVar_w", stat + "pCommonVar_l",
                   stat + "oCommonVar_w", stat + "oCommonVar_l"]
```

```
  return results

#RANDOM FOREST 1 - REGRESSOR

def forestRegress(in1, in2, in3, year):
  df = pd.read_csv("stats.csv")
  predicted = pd.DataFrame()

  yearMax = df["year"] < year
  yearMin = df["year"] >= year - 5
  past_mask = yearMin & yearMax

  df = df[past_mask]

  stats = ["2ndRnWon%", "svWon%", "1stRnWon%"]

  X = df.drop(["year", "outcome", "2ndRnWon%", "svWon%", "1stRnWon%"], axis = 1)
  y = df[stats]

  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = .3)

  for i in range(len(stats)):
    stat = stats[i]

    statList = ["surface", stat + "player_w", stat + "player_l",
              stat + "opponent_w", stat + "opponent_l",
              stat + "playerCommon_w", stat + "playerCommon_l",
              stat + "opponentCommon_w", stat + "opponentCommon_l",
              stat + "pVariance_w", stat + "pVariance_l",
              stat + "oVariance_w", stat + "oVariance_l",
              stat + "pCommonVar_w", stat + "pCommonVar_l",
              stat + "oCommonVar_w", stat + "oCommonVar_l"]

    sX_train = X_train[statList]
    sy_train = y_train[stat]

    forest = RandomForestRegressor(warm_start = True, oob_score = True,
                                  min_samples_leaf = 6, n_estimators = 200,
                                   max_depth = 150)
    forest.fit(sX_train, sy_train.values.ravel())

    if i == 0:
      tempPrediction = pd.DataFrame(forest.predict(in1))
    elif i == 1:
      tempPrediction = pd.DataFrame(forest.predict(in2))
    else:
      tempPrediction = pd.DataFrame(forest.predict(in3))
```

```
    tempPrediction.columns = [stat]

    if stat == "2ndRnWon%":
      predicted = tempPrediction
    else:
      predicted = predicted.join(tempPrediction)

  return predicted

#RANDOM FOREST 2 - CLASSIFIER

def forestClassify(in_test):
  df = pd.read_csv("tennis.csv")

  win = df[["w_2ndRnWon%","w_svWon%","w_1stRnWon%"]]
  win.columns = ["2ndRnWon%","svWon%","1stRnWon%"]

  lose = df[["l_2ndRnWon%","l_svWon%","l_1stRnWon%"]]
  lose.columns = ["2ndRnWon%","svWon%","1stRnWon%"]

  #data
  X = pd.concat([win,lose])

  #target
  y = pd.concat([pd.DataFrame(np.ones((len(win),1))),
                 pd.DataFrame(np.zeros((len(lose),1)))])

  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = .3)

  forest = RandomForestClassifier(warm_start = True, oob_score = True)

  forest.fit(X_train, y_train.values.ravel())
  predicted = forest.predict(in_test)

  return predicted
```

1.3 Appendix C - code used to rank features by importance in random forest classifier

[ ]:
```
from sklearn.metrics import accuracy_score
from sklearn.inspection import permutation_importance

#Random forest classifier used to compare features
def forestFeatureRank(win, lose):
  #data
  X = pd.concat([win,lose])
```

```python
  #target
  y = pd.concat([pd.DataFrame(np.ones((len(win),1))),
                 pd.DataFrame(np.zeros((len(lose),1)))])

  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = .3)
  forest = RandomForestClassifier(oob_score = True)

  #fit model
  forest.fit(X_train, y_train.values.ravel())
  predicted = forest.predict(X_test)

  #feature importance
  importance = permutation_importance(forest,
                                     X_train, y_train)["importances_mean"]
  features = sorted(zip(importance, win.columns), reverse=True)
  for i in range(len(features[0:])):
    print(features[0:][i][1])
  print("Accuracy =",accuracy_score(y_test, predicted),"\n")

df = pd.read_csv("fulltennis.csv")

win = df[["surface","w_ace","w_df","w_svpt","w_1stIn","w_2ndIn","w_1stWon",
          "w_2ndWon","w_SvGms","w_bpSaved","w_bpFaced",
          "w_1stIn%","w_2ndIn%","w_1stWon%","w_2ndWon%","w_svWon%",
          "w_1stRnWon%","w_2ndRnWon%", "w_rnWon%"]]
win.columns = ["surface","ace","df","svpt","1stIn","2ndIn","1stWon","2ndWon",
               "SvGms","bpSaved","bpFaced","1stIn%","2ndIn%","1stWon%",
               "2ndWon%","svWon%","1stRnWon%","2ndRnWon%", "rnWon%"]

lose = df[["surface","l_ace","l_df","l_svpt","l_1stIn","l_2ndIn","l_1stWon",
           "l_2ndWon","l_SvGms","l_bpSaved","l_bpFaced",
           "l_1stIn%","l_2ndIn%","l_1stWon%","l_2ndWon%","l_svWon%",
           "l_1stRnWon%","l_2ndRnWon%", "w_rnWon%"]]
lose.columns = win.columns

print("Ranking All Features:")
forestFeatureRank(win, lose)

win = df[["w_svWon%","w_1stRnWon%","w_2ndRnWon%"]]
win.columns = ["svWon%","1stRnWon%","2ndRnWon%"]

lose = df[["l_svWon%","l_1stRnWon%","l_2ndRnWon%"]]
lose.columns = ["svWon%","1stRnWon%","2ndRnWon%"]

print("Ranking Top Three Features:")
forestFeatureRank(win, lose)
```

1.4 Appendix D - code used to create data visualizations

[ ]:
```
import matplotlib.pyplot as plt
df = pd.read_csv("tennis.csv")

#Graphing 2ndRnWon%
sum = 0
for i in range(len(df["w_2ndRnWon%"])):
  if df["w_2ndRnWon%"][i] - df["l_2ndRnWon%"][i] >= 0:
    sum += 100
lineStat = "(" + str(sum/len(df["w_1stRnWon%"
])) + "% of the data is to the left of the red line)"

rn2Plot = df.plot(kind = "scatter", x = "l_2ndRnWon%", y = "w_2ndRnWon%",
                  alpha = .1)
rn2Plot.plot((0,1), 'r--')
plt.title("Comparing Ratio of 2nd Returns Won")
plt.xlabel("Ratio of 2nd Returns Won by Loser")
plt.ylabel("Ratio of 2nd Returns Won by Winner")
plt.text(-.02,-.3,lineStat)
plt.show()

(df["w_2ndRnWon%"] - df["l_2ndRnWon%"]).plot(kind = "box", vert = False,
                                             ylabel = "difference")
plt.yticks(color = 'w', rotation = 90)
plt.gca().invert_xaxis()
plt.vlines(0, .8, 1.2, colors = "red", linestyles="dashed")
plt.title(
    "Distribution of Difference Between Winner and Loser\nin 2nd Returns Won")
plt.text(1.02,.3,lineStat)
plt.show()


#Graphing svWon%
sum = 0
for i in range(len(df["w_svWon%"])):
  if df["w_svWon%"][i] - df["l_svWon%"][i] >= 0:
    sum += 100
lineStat = "(" + str(sum/len(df["w_1stRnWon%"
])) + "% of the data is to the left of the red line)"

svPlot = df.plot(kind = "scatter", x = "l_svWon%", y = "w_svWon%", alpha = .1)
svPlot.plot((0,1), 'r--')
plt.title("Comparing Ratio of Serves Won")
plt.xlabel("Ratio of Serves Won by Loser")
plt.ylabel("Ratio of Serves Won by Winner")
plt.text(-.02,-.3,lineStat)
```

```
plt.show()

(df["w_svWon%"] - df["l_svWon%"]).plot(kind = "box", vert = False,
                                       ylabel = "difference")
plt.yticks(color = 'w', rotation = 90)
plt.gca().invert_xaxis()
plt.vlines(0, .8, 1.2, colors = "red", linestyles="dashed")
plt.title("Distribution of Difference Between Winner and Loser\nin Serves Won")
plt.text(.82,.3,lineStat)
plt.show()


#Graphing 1stRnWon%
sum = 0
for i in range(len(df["w_1stRnWon%"])):
  if df["w_1stRnWon%"][i] - df["l_1stRnWon%"][i] >= 0:
    sum += 100
lineStat = "(" + str(sum/len(df["w_1stRnWon%"
])) + "% of the data is to the left of the red line)"

rn1Plot = df.plot(kind = "scatter", x = "l_1stRnWon%", y = "w_1stRnWon%",
                  alpha = .1)
rn1Plot.plot((0,1), 'r--')
plt.title("Comparing Ratio of 1st Returns Won")
plt.xlabel("Ratio of 1st Returns Won by Loser")
plt.ylabel("Ratio of 1st Returns Won by Winner")
plt.text(0,-.3,lineStat)
plt.show()

(df["w_1stRnWon%"] - df["l_1stRnWon%"]).plot(kind = "box", vert = False,
                                             ylabel = "difference")
plt.yticks(color = 'w', rotation = 90)
plt.gca().invert_xaxis()
plt.vlines(0, .8, 1.2, colors = "red", linestyles="dashed")
plt.title(
    "Distribution of Difference Between Winner and Loser\nin 1st Returns Won")
plt.text(.8,.3,lineStat)
plt.show()

#Graphing clustering of 2ndRnWon% vs 2ndRnWon%player_w
from sklearn.cluster import KMeans

#apply k-means clustering to data
df = pd.read_csv("stats.csv")
kmeans = KMeans(n_clusters = 2, algorithm = "full").fit(df[["2ndRnWon%player_w",
                                                            "2ndRnWon%"]])
y_kmeans = kmeans.predict(df[["2ndRnWon%player_w", "2ndRnWon%"]])
```

```
centers = kmeans.cluster_centers_

#plot data and separate visually into clusters defined by k-means
plt.scatter(centers[:, 0], centers[:, 1], c='black', s=200)
plt.scatter(x = df["2ndRnWon%player_w"], y = df["2ndRnWon%"], c=y_kmeans,
            cmap='coolwarm', alpha = .05)
plt.xlabel("Average Ratio of 2nd Returns Won Over 5 Years Prior")
plt.ylabel("Ratio of 2nd Returns Won")
plt.title("Clustering Average Ratio of 2nd Returns Won over 5 Years␣
 ↪Prior\nCompared to Present Ratio of 2nd Returns Won")
plt.show()
```

1.5 Appendix E - code used to test final prediction model accuracy

[ ]:
```
#Testing prediction model on all match data from 2019
df = pd.read_csv("tennis.csv")
mask = df["year"] == 2019
df = df[mask]

#create dataframe of player names
names = pd.DataFrame()
names["player1"] = df["winner_name"]
names["player2"] = df["loser_name"]
names["tourney"] = df["tourney_name"]

#randomly rearrange dataframe
for i in range(len(names)):
  p = np.random.random()
  if p < .5:
    temp = names['player1'].iloc[i]
    names["player1"].iloc[i] = names["player2"].iloc[i]
    names["player2"].iloc[i] = temp

#predict results using prediction model accross the various tournaments
mask = names["tourney"] == "Australian Open"
results = predictionModel(names[mask], 2019, "Australian Open")
mask = names["tourney"] == "Roland Garros"
results = results.append(predictionModel(names[mask], 2019, "Roland Garros"),
                         ignore_index = True)
mask = names["tourney"] == "Wimbledon"
results = results.append(predictionModel(names[mask], 2019, "Wimbledon"),
                         ignore_index = True)
mask = names["tourney"] == "US Open"
results = results.append(predictionModel(names[mask], 2019, "US Open"),
                         ignore_index = True)

#compare predictions to actual match results and report accuracy
```

```
accuracy = 0
for i in range(len(results)):
  if results["predicted_winner"].iloc[i] == df["winner_name"].iloc[i]:
    accuracy += 1
accuracy /= len(results)

#accuracy usually between 60% and 63%
print("Proportion of Accurate Predictions:", accuracy)
```

1.6 Appendix F - code for PCA run on indidivual match features

[ ]:
```
from sklearn.decomposition import PCA
df = pd.read_csv("fulltennis.csv")
pca = PCA(n_components = 3)

win = df[["surface","w_ace","w_df","w_svpt","w_1stIn","w_2ndIn","w_1stWon",
          "w_2ndWon","w_SvGms","w_bpSaved","w_bpFaced",
          "w_1stIn%","w_2ndIn%","w_1stWon%","w_2ndWon%","w_svWon%",
          "w_1stRnWon%","w_2ndRnWon%", "w_rnWon%"]]
win.columns = ["surface","ace","df","svpt","1stIn","2ndIn","1stWon","2ndWon",
               "SvGms","bpSaved","bpFaced","1stIn%","2ndIn%","1stWon%",
               "2ndWon%","svWon%","1stRnWon%","2ndRnWon%", "rnWon%"]
lose = df[["surface","l_ace","l_df","l_svpt","l_1stIn","l_2ndIn","l_1stWon",
           "l_2ndWon","l_SvGms","l_bpSaved","l_bpFaced",
           "l_1stIn%","l_2ndIn%","l_1stWon%","l_2ndWon%","l_svWon%",
           "l_1stRnWon%","l_2ndRnWon%", "w_rnWon%"]]
lose.columns = win.columns

pca.fit(pd.concat([win,lose]))
print("Variance Explained by Top Three Features:",
      sum(pca.explained_variance_ratio_))
```

1.7 Appendix G - code used for initial regressor test on stats.csv

[ ]:
```
#Testing random forest regressor on stats.csv
df = pd.read_csv("stats.csv")
predicted = pd.DataFrame()

stats = ["2ndRnWon%", "svWon%", "1stRnWon%"]

X = df.drop(["year", "outcome", "2ndRnWon%", "svWon%", "1stRnWon%"], axis = 1)
y = df[stats]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = .3)

for i in range(len(stats)):
```

```
  stat = stats[i]

  statList = ["surface", stat + "player_w", stat + "player_l",
              stat + "opponent_w", stat + "opponent_l",
              stat + "playerCommon_w", stat + "playerCommon_l",
              stat + "opponentCommon_w", stat + "opponentCommon_l",
              stat + "pVariance_w", stat + "pVariance_l",
              stat + "oVariance_w", stat + "oVariance_l",
              stat + "pCommonVar_w", stat + "pCommonVar_l",
              stat + "oCommonVar_w", stat + "oCommonVar_l"]

  sX_train = X_train[statList]
  sX_test = X_test[statList]
  sy_train = y_train[stat]

  forest = RandomForestRegressor(warm_start = True, oob_score = True,
                              min_samples_leaf = 6, n_estimators = 200,
                               max_depth = 150)
  forest.fit(sX_train, sy_train.values.ravel())
  tempPrediction = pd.DataFrame(forest.predict(sX_test))
  tempPrediction.columns = [stat]

  if stat == "2ndRnWon%":
    predicted = tempPrediction
  else:
    predicted = predicted.join(tempPrediction)

print("Accuracy =",accuracy_score(df["outcome"].iloc[X_test.index],
                                  forestClassify(predicted)))
```

1.8 Appendix H - code used to test kmeans and spectral accuracy

[ ]:
```
from sklearn.metrics import accuracy_score
from sklearn.cluster import KMeans
from sklearn.cluster import SpectralClustering
df = pd.read_csv("stats.csv")

X = df.drop(["year", "surface", "2ndRnWon%", "svWon%",
             "1stRnWon%", "outcome"], axis = 1)

y = df["outcome"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = .3)

#fit kmeans clustering and output accuracy
kmeans = KMeans(n_clusters = 2).fit(X_train)
kPredicted = kmeans.predict(X_test)
```

```
kAccuracy = accuracy_score(y_test, kPredicted)
if kAccuracy < .5:
  kAccuracy = 1 - kAccuracy
print("Proportion of accurate predictions with kmeans:", kAccuracy)

#fit spectral clustering and output accuracy
spectral = SpectralClustering(n_clusters = 2).fit(X_train)
sPredicted = spectral.fit_predict(X_test)
sAccuracy = accuracy_score(y_test, sPredicted)
if sAccuracy < .5:
  sAccuracy = 1 - sAccuracy
print("Proportion of accurate predictions with spectral clustering:", sAccuracy)
```

1.9 Appendix I - first five rows of the stats.csv file

[ ]:
```
import pandas as pd
pd.set_option('display.max_columns', None)
pd.read_csv("stats.csv").head()
```

[ ]:
```
     year  surface  2ndRnWon%    svWon%  1stRnWon%  outcome  \
0  2008.0      0.0   0.489362  0.723214   0.283582      1.0
1  2008.0      0.0   0.178571  0.504854   0.267857      0.0
2  2008.0      0.0   0.538462  0.611111   0.346535      1.0
3  2008.0      0.0   0.268293  0.511111   0.195122      0.0
4  2008.0      0.0   0.192308  0.522727   0.307692      0.0

   2ndRnWon%player_w  2ndRnWon%player_l  2ndRnWon%opponent_w  \
0           0.552353           0.263233             0.526618
1           0.587202           0.310956             0.527070
2           0.570517           0.269485             0.602686
3           0.580924           0.406957             0.528473
4           0.575643           0.256561             0.527659

   2ndRnWon%opponent_l  2ndRnWon%playerCommon_w  2ndRnWon%playerCommon_l  \
0             0.289022                 0.571429                 0.253140
1             0.282955                 0.551724                 0.332156
2             0.299596                 0.587520                 0.265466
3             0.245692                 0.600000                 0.429662
4             0.266432                 0.916667                 0.304328

   2ndRnWon%opponentCommon_w  2ndRnWon%opponentCommon_l  2ndRnWon%pVariance_w  \
0                   0.500000                   0.364224              0.011655
1                   0.576606                   0.397590              0.001628
2                   0.628571                   0.241978              0.010450
3                   0.474722                   0.147368              0.005353
4                   0.600000                   0.276648              0.013896
```

	2ndRnWon%pVariance_l	2ndRnWon%oVariance_w	2ndRnWon%oVariance_l	\
0	0.011504	0.009158	0.004851	
1	0.010242	0.006618	0.009944	
2	0.009866	0.002366	0.012371	
3	0.021901	0.003301	0.006540	
4	0.008973	0.007275	0.009295	

	2ndRnWon%pCommonVar_w	2ndRnWon%pCommonVar_l	2ndRnWon%oCommonVar_w	\
0	0.000000	0.001278	0.000000	
1	0.000000	0.005037	0.002748	
2	0.000682	0.011651	0.000000	
3	0.000000	0.024515	0.006958	
4	0.000000	0.004729	0.000000	

	2ndRnWon%oCommonVar_l	1stRnWon%player_w	1stRnWon%player_l	\
0	0.004234	0.303115	0.187896	
1	0.000000	0.308748	0.266854	
2	0.002986	0.342633	0.284496	
3	0.000000	0.371813	0.246415	
4	0.006333	0.381543	0.259840	

	1stRnWon%opponent_w	1stRnWon%opponent_l	1stRnWon%playerCommon_w	\
0	0.400668	0.221569	0.316667	
1	0.357139	0.271407	0.340000	
2	0.366892	0.242924	0.232965	
3	0.283163	0.259052	0.423077	
4	0.323779	0.267443	0.666667	

	1stRnWon%playerCommon_l	1stRnWon%opponentCommon_w	\
0	0.197826	0.459459	
1	0.238066	0.383273	
2	0.285918	0.431034	
3	0.281900	0.323077	
4	0.296467	0.283951	

	1stRnWon%opponentCommon_l	1stRnWon%pVariance_w	1stRnWon%pVariance_l	\
0	0.213740	0.009360	0.002843	
1	0.361446	0.003881	0.002273	
2	0.206424	0.007505	0.006023	
3	0.284211	0.005616	0.004499	
4	0.279129	0.011367	0.004012	

	1stRnWon%oVariance_w	1stRnWon%oVariance_l	1stRnWon%pCommonVar_w	\
0	0.001841	0.005163	0.000000	
1	0.006069	0.004980	0.000000	
2	0.004104	0.003504	0.000408	
3	0.004983	0.001208	0.000000	

4	0.002525	0.004968	0.000000

	1stRnWon%pCommonVar_l	1stRnWon%oCommonVar_w	1stRnWon%oCommonVar_l \
0	0.000005	0.000000	0.000752
1	0.000623	0.010442	0.000000
2	0.002587	0.000000	0.003510
3	0.011416	0.005917	0.000000
4	0.000008	0.000000	0.005096

	svWon%player_w	svWon%player_l	svWon%opponent_w	svWon%opponent_l \
0	0.716897	0.530297	0.691051	0.581457
1	0.655625	0.563563	0.701330	0.577701
2	0.673278	0.510357	0.662208	0.561191
3	0.658385	0.551997	0.709836	0.569475
4	0.707384	0.584417	0.672128	0.550544

	svWon%playerCommon_w	svWon%playerCommon_l	svWon%opponentCommon_w \
0	0.750000	0.535714	0.681818
1	0.704545	0.563026	0.680642
2	0.664888	0.510399	0.662791
3	0.584746	0.500536	0.661302
4	0.724138	0.542244	0.685950

	svWon%opponentCommon_l	svWon%pVariance_w	svWon%pVariance_l \
0	0.573745	0.003324	0.005815
1	0.560000	0.003032	0.001090
2	0.570244	0.005353	0.004864
3	0.568493	0.004332	0.003984
4	0.540140	0.000865	0.003363

	svWon%oVariance_w	svWon%oVariance_l	svWon%pCommonVar_w \
0	0.004006	0.002244	0.000000
1	0.004114	0.004710	0.000000
2	0.002521	0.002713	0.000511
3	0.003295	0.003535	0.000000
4	0.002965	0.002965	0.000000

	svWon%pCommonVar_l	svWon%oCommonVar_w	svWon%oCommonVar_l
0	0.001276	0.000000	0.000909
1	0.001779	0.002537	0.000000
2	0.006685	0.000000	0.003061
3	0.000941	0.003582	0.000000
4	0.000704	0.000000	0.004586

1.10 Appendix J - results of coefficient gridsearch

[ ]:
```python
from IPython.display import Image
Image("Gridsearch.png")
```

[ ]:

mean ->	0.6	0.65	0.7	0.75	0.8	0.85	0.9	0.95	1	
0.6	0.575099	0.594862	0.590909	0.592885	0.588933	0.600791	0.596838	0.588933	0.594862	0.591568
0.65	0.590909	0.586957	0.588933	0.592885	0.588933	0.586957	0.612648	0.602767	0.602767	0.594862
0.7	0.58498	0.590909	0.581028	0.596838	0.600791	0.596838	0.610672	0.602767	0.604743	0.596618
0.75	0.586957	0.594862	0.590909	0.590909	0.596838	0.592885	0.594862	0.602767	0.588933	0.593325
0.8	0.586957	0.58498	0.581028	0.586957	0.583004	0.58498	0.588933	0.586957	0.581028	0.58498
0.85	0.600791	0.583004	0.579051	0.581028	0.588933	0.58498	0.594862	0.58498	0.588933	0.587396
0.9	0.583004	0.588933	0.577075	0.586957	0.581028	0.583004	0.581028	0.58498	0.586957	0.583663
0.95	0.590909	0.590909	0.586957	0.586957	0.58498	0.581028	0.594862	0.592885	0.579051	0.587615
1	0.583004	0.590909	0.592885	0.586957	0.583004	0.594862	0.58498	0.586957	0.581028	0.587176
var ^^	0.586957	0.589592	0.585419	0.589152	0.588494	0.589592	0.59552	0.592666	0.589811	