In [ ]:
```
# install basic libraries
import numpy as np
import pandas as pd
import json
import difflib
import time
import requests


# install correct libraries
from nba_api.stats.static import teams, players
from nba_api.stats.endpoints import leaguegamefinder, boxscoretraditionalv2,
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

In [ ]:
```
# Retry Wrapper
def retry(func, retries=3):
    def retry_wrapper(*args, **kwargs):
        attempts = 0
        while attempts < retries:
            try:
                return func(*args, **kwargs)
            except requests.exceptions.RequestException as e:
                print(e)
                time.sleep(30)
                attempts += 1

    return retry_wrapper
```

In [ ]:
```
# Get Season Schedule Function

def getSeasonScheduleFrame(seasons,seasonType):

    # Get date from string
    def getGameDate(matchup):
        return matchup.partition(' at')[0][:10]

    # Get Home team from string
    def getHomeTeam(matchup):
        return matchup.partition(' at')[2]

    # Get Away team from string
    def getAwayTeam(matchup):
        return matchup.partition(' at')[0][10:]

    # Match nickname from schedule to team table to find ID
    def getTeamIDFromNickname(nickname):
        return teamLookup.loc[teamLookup['nickname'] == difflib.get_close_ma

    @retry
    def getRegularSeasonSchedule(season,teamID,seasonType):
        season = str(season) + "-" + str(season+1)[-2:] # Convert year to se
        teamGames = cumestatsteamgames.CumeStatsTeamGames(league_id = '00',s
```

```
                                                                    seasor
                                                                    team_i

        teamGames = pd.DataFrame(json.loads(teamGames)['CumeStatsTeamGames']
        teamGames['SEASON'] = season
        return teamGames

    # Get team lookup table
    teamLookup = pd.DataFrame(teams.get_teams())

    # Get teams schedule for each team for each season
    scheduleFrame = pd.DataFrame()

    for season in seasons:
        for id in teamLookup['id']:
            time.sleep(1)
            # scheduleFrame = scheduleFrame.append(getRegularSeasonSchedule(
            scheduleFrame = pd.concat([scheduleFrame,getRegularSeasonSchedul

    scheduleFrame['GAME_DATE'] = pd.to_datetime(scheduleFrame['MATCHUP'].map
    scheduleFrame['HOME_TEAM_NICKNAME'] = scheduleFrame['MATCHUP'].map(getHo
    scheduleFrame['HOME_TEAM_ID'] = scheduleFrame['HOME_TEAM_NICKNAME'].map(
    scheduleFrame['AWAY_TEAM_NICKNAME'] = scheduleFrame['MATCHUP'].map(getAw
    scheduleFrame['AWAY_TEAM_ID'] = scheduleFrame['AWAY_TEAM_NICKNAME'].map(
    scheduleFrame = scheduleFrame.drop_duplicates() # There's a row for both
    scheduleFrame = scheduleFrame.reset_index(drop=True)

    return scheduleFrame
```

In [ ]:

```
# Get Single Game aggregation columns

def getSingleGameMetrics(gameID,homeTeamID,awayTeamID,awayTeamNickname,seasc

    @retry
    def getGameStats(teamID,gameID,seasonYear):
        gameStats = cumestatsteam.CumeStatsTeam(game_ids=gameID,league_id ='
                                                season=seasonYear,season_type
                                                team_id = teamID).get_normali

        gameStats = pd.DataFrame(json.loads(gameStats)['TotalTeamStats'])

        return gameStats

    data = getGameStats(homeTeamID,gameID,seasonYear)
    data.at[1,'NICKNAME'] = awayTeamNickname
    data.at[1,'TEAM_ID'] = awayTeamID
    data.at[1,'OFFENSIVE_EFFICIENCY'] = (data.at[1,'FG'] + data.at[1,'AST'])
    data.at[1,'SCORING_MARGIN'] = data.at[1,'PTS'] - data.at[0,'PTS']

    data.at[0,'OFFENSIVE_EFFICIENCY'] = (data.at[0,'FG'] + data.at[0,'AST'])
    data.at[0,'SCORING_MARGIN'] = data.at[0,'PTS'] - data.at[1,'PTS']

    data['SEASON'] = seasonYear
    data['GAME_DATE'] = gameDate
    data['GAME_ID'] = gameID
```

```
    return data
```

In [ ]:

```
def getGameLogs(gameLogs,scheduleFrame):

    # Functions to prepare additional columns after gameLogs table loads
    def getHomeAwayFlag(gameDF):
        gameDF['HOME_FLAG'] = np.where((gameDF['W_HOME']==1) | (gameDF['L_HO
        gameDF['AWAY_FLAG'] = np.where((gameDF['W_ROAD']==1) | (gameDF['L_RO

    def getTotalWinPctg(gameDF):
        gameDF['TOTAL_GAMES_PLAYED'] = gameDF.groupby(['TEAM_ID','SEASON'])[
        gameDF['TOTAL_WINS'] = gameDF.sort_values(by='GAME_DATE').groupby(['
        gameDF['TOTAL_WIN_PCTG'] = gameDF['TOTAL_WINS']/gameDF['TOTAL_GAMES_
        return gameDF.drop(['TOTAL_GAMES_PLAYED','TOTAL_WINS'],axis=1)

    def getHomeWinPctg(gameDF):
        gameDF['HOME_GAMES_PLAYED'] = gameDF.sort_values(by='GAME_DATE').gro
        gameDF['HOME_WINS'] = gameDF.sort_values(by='GAME_DATE').groupby(['T
        gameDF['HOME_WIN_PCTG'] = gameDF['HOME_WINS']/gameDF['HOME_GAMES_PLA
        return gameDF.drop(['HOME_GAMES_PLAYED','HOME_WINS'],axis=1)

    def getAwayWinPctg(gameDF):
        gameDF['AWAY_GAMES_PLAYED'] = gameDF.sort_values(by='GAME_DATE').gro
        gameDF['AWAY_WINS'] = gameDF.sort_values(by='GAME_DATE').groupby(['T
        gameDF['AWAY_WIN_PCTG'] = gameDF['AWAY_WINS']/gameDF['AWAY_GAMES_PLA
        return gameDF.drop(['AWAY_GAMES_PLAYED','AWAY_WINS'],axis=1)

    def getRollingOE(gameDF):
        gameDF['ROLLING_OE'] = gameDF.sort_values(by='GAME_DATE').groupby(['

    def getRollingScoringMargin(gameDF):
        gameDF['ROLLING_SCORING_MARGIN'] = gameDF.sort_values(by='GAME_DATE'

    def getRestDays(gameDF):
        gameDF['LAST_GAME_DATE'] = gameDF.sort_values(by='GAME_DATE').groupb
        gameDF['NUM_REST_DAYS'] = (gameDF['GAME_DATE'] - gameDF['LAST_GAME_D
        return gameDF.drop('LAST_GAME_DATE',axis=1)

    start = time.perf_counter_ns()

    i = int(len(gameLogs)/2) #Can use a previously completed gameLog dataset

    while i<len(scheduleFrame):


        time.sleep(1)
        # gameLogs =  gameLogs.append(getSingleGameMetrics(scheduleFrame.at[
        #                             scheduleFrame.at[i,'AWAY_TEAM_ID'],sch
        #                             scheduleFrame.at[i,'SEASON'],scheduleF
        gameLogs = pd.concat([
            gameLogs,
            getSingleGameMetrics(scheduleFrame.at[i,'GAME_ID'],scheduleFrame
                                 scheduleFrame.at[i,'AWAY_TEAM_ID'],scheduleF
                                 scheduleFrame.at[i,'SEASON'],scheduleFrame.a
        ])
```

```
        gameLogs = gameLogs.reset_index(drop=True)

        end = time.perf_counter_ns()

        #Output time it took to load x amount of records
        if i%100 == 0:
            mins = ((end-start)/1e9)/60
            print(i,str(mins) + ' minutes')

        i+=1

    # Get Table Level Aggregation Columns
    getHomeAwayFlag(gameLogs)
    gameLogs = getHomeWinPctg(gameLogs)
    gameLogs = getAwayWinPctg(gameLogs)
    gameLogs = getTotalWinPctg(gameLogs)
    getRollingScoringMargin(gameLogs)
    getRollingOE(gameLogs)
    gameLogs = getRestDays(gameLogs)

    return gameLogs.reset_index(drop=True)
```

In [ ]:
```
#Get ScheduleFrame

seasons = [2020,2021,2022]
seasonType = 'Regular Season'

start = time.perf_counter_ns() # Track cell's runtime
scheduleFrame = getSeasonScheduleFrame(seasons,seasonType)
end = time.perf_counter_ns()

secs = (end-start)/1e9
mins = secs/60
print(mins)
```

```
2.094102697233333
```

In [ ]:
```
# schedule frame includes all games from respective seasons and the game tea
scheduleFrame.head()
```

Out[ ]:

	MATCHUP	GAME_ID	SEASON	GAME_DATE	HOME_TEAM_NICKNAME	HOME_T
0	05/16/2021 Rockets at Hawks	0022001066	2020-21	2021-05-16	Hawks	161
1	05/13/2021 Magic at Hawks	0022001049	2020-21	2021-05-13	Hawks	161
2	05/12/2021 Wizards at Hawks	0022001042	2020-21	2021-05-12	Hawks	161
3	05/10/2021 Wizards at Hawks	0022001026	2020-21	2021-05-10	Hawks	161
4	05/06/2021 Hawks at Pacers	0022001000	2020-21	2021-05-06	Pacers	161

In [ ]:
```
#Create the gameLogs DataFrame
gameLogs = pd.DataFrame()
gameLogs = getGameLogs(gameLogs,scheduleFrame)
gameLogs.to_csv('gameLogs.csv')
```

```
0 0.02421859845 minutes
100 2.588284267833333 minutes
200 5.137943998916667 minutes
300 7.6554500024 minutes
400 10.3696445492 minutes
500 13.0643912914 minutes
600 15.718385262916666 minutes
700 18.243165964833334 minutes
800 20.753762765450002 minutes
900 23.266447112616667 minutes
1000 25.736427251233334 minutes
1100 28.336117988566667 minutes
1200 30.922886253 minutes
1300 33.4363804461 minutes
1400 35.98393400766667 minutes
1500 38.59013183943333 minutes
1600 41.04990109173333 minutes
1700 43.549907097416664 minutes
1800 46.038305438749994 minutes
1900 48.5225339586 minutes
2000 51.04542726406667 minutes
2100 53.60723333558334 minutes
2200 56.09513774451667 minutes
2300 58.67886037316667 minutes
2400 61.143107950516665 minutes
2500 63.84030319756666 minutes
2600 66.3490747249 minutes
2700 68.90855629188333 minutes
2800 71.36238123036667 minutes
2900 73.97829276236666 minutes
3000 76.42635629944999 minutes
3100 78.90800495698333 minutes
3200 81.41434198643333 minutes
3300 83.95515916626667 minutes
3400 86.50015521056666 minutes
3500 89.06317405906665 minutes
```

In [ ]:
```
def getGameLogFeatureSet(gameDF):

    def shiftGameLogRecords(gameDF):
        gameDF['LAST_GAME_OE'] = gameLogs.sort_values('GAME_DATE').groupby([
        gameDF['LAST_GAME_HOME_WIN_PCTG'] = gameDF.sort_values('GAME_DATE').
        gameDF['LAST_GAME_AWAY_WIN_PCTG'] = gameDF.sort_values('GAME_DATE').
        gameDF['LAST_GAME_TOTAL_WIN_PCTG'] = gameDF.sort_values('GAME_DATE')
        gameDF['LAST_GAME_ROLLING_SCORING_MARGIN'] = gameDF.sort_values('GAM
        gameDF['LAST_GAME_ROLLING_OE'] = gameDF.sort_values('GAME_DATE').gro


    def getHomeTeamFrame(gameDF):
        homeTeamFrame = gameDF[gameDF['CITY'] != 'OPPONENTS']
        homeTeamFrame = homeTeamFrame[['LAST_GAME_OE','LAST_GAME_HOME_WIN_PC

        colRenameDict = {}
        for col in homeTeamFrame.columns:
            if (col != 'GAME_ID') & (col != 'SEASON') :
                colRenameDict[col] = 'HOME_' + col
```

```
        homeTeamFrame.rename(columns=colRenameDict,inplace=True)

        return homeTeamFrame

    def getAwayTeamFrame(gameDF):
        awayTeamFrame = gameDF[gameDF['CITY'] == 'OPPONENTS']
        awayTeamFrame = awayTeamFrame[['LAST_GAME_OE','LAST_GAME_HOME_WIN_PC

        colRenameDict = {}
        for col in awayTeamFrame.columns:
            if (col != 'GAME_ID') & (col != 'SEASON'):
                colRenameDict[col] = 'AWAY_' + col

        awayTeamFrame.rename(columns=colRenameDict,inplace=True)

        return awayTeamFrame

    shiftGameLogRecords(gameLogs)
    awayTeamFrame = getAwayTeamFrame(gameLogs)
    homeTeamFrame = getHomeTeamFrame(gameLogs)

    return pd.merge(homeTeamFrame, awayTeamFrame, how="inner", on=[ "GAME_ID
```

In [ ]:
```
gameLogs = pd.read_csv('gameLogs.csv')
```

In [ ]:
```
modelData = getGameLogFeatureSet(gameLogs)
```

In [ ]:
```
modelData.to_csv('nbaHomeWinLossModelDataset.csv')
```

Field Name	Annotation
HOME_LAST_GAME_OE	Offensive efficiency in the home team's last game
HOME_LAST_GAME_HOME_WIN_PCTG	Home team's winning percentage in their last home game
HOME_NUM_REST_DAYS	Number of rest days for the home team before the current game
HOME_LAST_GAME_AWAY_WIN_PCTG	Home team's winning percentage in their last away game
HOME_LAST_GAME_TOTAL_WIN_PCTG	Home team's total winning percentage across all games before the current game
HOME_LAST_GAME_ROLLING_SCORING_MARGIN	Home team's average scoring margin over last 3 games
HOME_LAST_GAME_ROLLING_OE	Home team's offensive efficiency averaged over last 3 games
HOME_W	Indicates if the home team won the last game (uses binary values)

Field Name	Annotation
SEASON	The season during which the game is played
AWAY_LAST_GAME_OE	Offensive efficiency in the away team's last game
AWAY_LAST_GAME_HOME_WIN_PCTG	Away team's winning percentage in their last game played at home
AWAY_NUM_REST_DAYS	Number of rest days for the away team before the current game
AWAY_LAST_GAME_AWAY_WIN_PCTG	Away team's winning percentage in their last game played away from home
AWAY_LAST_GAME_TOTAL_WIN_PCTG	Away team's total winning percentage across all games before the current game
AWAY_LAST_GAME_ROLLING_SCORING_MARGIN	Away team's average scoring margin over last 3 games
AWAY_LAST_GAME_ROLLING_OE	Away team's offensive efficiency averaged over last 3 games

In [ ]: