```python
import os

from calendar import monthrange
from nba_py import game, Scoreboard
import pandas as p
import argparse


MONTHS_BEFORE_NEWYEAR = range(10,13)
MONTHS_AFTER_NEWYEAR = range(1,5)
UNNEEDED_FEATURES = ['GAME_ID','TEAM_ID', 'TEAM_ABBREVIATION', 'TEAM_CITY', 'MIN', 'FGA', 'FGM',
NO_DEFICIT_FEATURES = ['PCT', 'TEAM', 'ID', 'MIN', 'OUTCOME', 'PTS', 'PLUS_MINUS']
AWAY_WIN = [0,1]
HOME_WIN = [1,0]
CSV = '.csv'



class Create_csv_files(object):
    '''''
    Creating csv file containing the data of an NBA season
    '''


    def __init__(self, root_dir, season):
        '''''
        Initialize the object
        @param root_dir: a path to the root directory which all the files will be stored
        @type root_dir: string
        @param season: the years the season is being played
        @type season: list of strings, containing 2 strings
        '''
        self._root_dir = root_dir +'/'
        # Season must be a list with two strings in the form of [2012,2013]
        if not (len(season) == 2 and (season[1] - season[0] == 1)):
            print('season in not correct')
        self._season = season
        self.df = p.DataFrame()
        self._current_teams = []

    def _create_directory(self, curr_dir):
        '''''
        Create a directory to contain the new csv files
        '''
        if not os.path.exists(curr_dir):
            os.makedirs(curr_dir)

    def season_name(self):
        '''''
        @return: the season name as a form of string
        @rtype: str
        '''
        return str(self._season[0]) + '-' + str(self._season[1])

    def scrape_season(self, test=False):
        '''''
        Scrape a full season from the nba stats website
        @param test: for debugging
        @type test: bool
        '''
        season_dir = self._root_dir + '/' + self.season_name() + '/'
        month_years = [(self._season[0], month) for month in MONTHS_BEFORE_NEWYEAR] + \
                      [(self._season[1], month) for month in MONTHS_AFTER_NEWYEAR]

        self._create_directory(season_dir)

        if test:
            # For debugging
            self.scrape_month(season_dir, 2014, 11, first_day=22,last_day=25)
        else:
            # Scrape the all season
            for year, month in month_years:
                print('currently scraping: {}/{}'.format(year,month))
```

```
                self.scrape_month(season_dir, year, month)

        all_boxscores_file = season_dir + 'all.csv'
        print("create a csv file with all the games named ", all_boxscores_file)
        self.df.to_csv(all_boxscores_file)

    def scrape_month(self, root_dir, year, month, first_day=1, last_day=31):
        '''''
        Create a directory containing all the boxscore of the entered month
        @param root_dir: the directory to hold the current month data
        @type root_dir: str
        @param year: the current year to gather data from
        @type year: int
        @param month: the current month to gather data from
        @type month: int
        @param first_day: the first day of games in the month
        @type first_day: int
        @param last_day: the last day of games in the month
        @type last_day: int
        '''
        last_day = min([monthrange(year, month)[1], last_day])
        curr_dir = root_dir + str(month) + '/'

        self._create_directory(curr_dir)

        for day in range(first_day, last_day+1):
            print("currently scraping {day}/{month}/{year}".format(day=day, month=month, year=yea
            self.process_scoreboard(year, month, day, curr_dir)

    def _fill_teams(self, sb):
        '''''
        Gather all the teams of the current season
        @param sb: data about the current day
        @type sb: ScoreBoard Object
        '''
        es = sb.east_conf_standings_by_day()
        ws = sb.west_conf_standings_by_day()

        self._current_teams += list(ws['TEAM']) + list(es['TEAM'])

    def process_scoreboard(self, root_dir, year, month, day ):
        '''''
        Create csv files with game id inside a directory by day
        @param root_dir: the directory to hold the current month data
        @type root_dir: str
        @param year: the current year to gather data from
        @type year: int
        @param month: the current month to gather data from
        @type month: int
        @param day: the current day of games
        @type day: int
        '''
        sb = Scoreboard(month=month, day=day, year=year)
        curr_dir = root_dir + str(day) + '/'

        if not self._current_teams:
            self._fill_teams(sb)
        self._create_directory(curr_dir)

        for gid in sb.available().GAME_ID:
            self.boxscore_to_csv(gid, curr_dir)

    def remove_columns(self, ts):
        '''''
        Remove unnecessary features we don't need to be in the final data
        '''
        for feature in UNNEEDED_FEATURES:
            del ts[feature]
        return ts

    def add_deficit_column(self, ts, feature):
        '''''
        Add data containing the deficit between the teams
        '''
```

```
148.          ts['DEFICIT_'+feature] = ts[feature] / sum(ts[feature])
149.          return ts
150.
151.      def add_columns(self, ts):
152.          '''''
153.          Adding columns to a given dataframe
154.          @param ts: box score
155.          @type ts: dataframe
156.          '''
157.          # Adding 2 pt field goals data
158.          ts['FG2M'] = ts['FGM'] - ts['FG3M']
159.          ts['FG2A'] = ts['FGA'] - ts['FG3A']
160.          ts['FG2_PCT'] = ts['FG2M'] / ts['FG2A']
161.
162.          # 1 represents win 0 represents loss
163.
164.          ts['OUTCOME'] = HOME_WIN if ts['PLUS_MINUS'][0] > 0 else AWAY_WIN
165.
166.          # Add deficit
167.          for c in ts.columns:
168.              if any(s in c for s in NO_DEFICIT_FEATURES):
169.                  continue
170.              ts = self.add_deficit_column(ts, c)
171.
172.          return ts
173.
174.      def irrelevant_data(self, ts):
175.          '''''
176.          Test if the current receive data needs to be deleted
177.          '''
178.          if len(ts.index) == 0:
179.              return True
180.
181.          if len(ts.dropna()) < 2:
182.              return True
183.
184.          if not all(team in self._current_teams for team in ts['TEAM_CITY']):
185.              return True
186.
187.          return False
188.
189.
190.      # TODO: add manipulation over the data
191.      def manipulate_df(self, ts):
192.          '''''
193.          Manipulate data to fit for the research
194.          '''
195.          if self.irrelevant_data(ts):
196.              return None
197.
198.          ts = self.add_columns(ts)
199.
200.          ts = self.remove_columns(ts)
201.
202.          return ts
203.
204.      def boxscore_to_csv(self, gid, root_dir):
205.          '''''
206.          Create a csv file from a boxscore according to the game id
207.          '''
208.          bs = game.Boxscore(gid)
209.          team_stats = bs.team_stats()
210.          team_stats = self.manipulate_df(team_stats)
211.          if team_stats is None :
212.              return
213.
214.          csv_name = root_dir + gid + CSV
215.          team_stats.to_csv(csv_name)
216.
217.          self.df = self.df.append(team_stats, ignore_index=True)
218.
219.
220.  def main():
221.      parser = argparse.ArgumentParser(description="")
```

```
    parser.add_argument('--directory', dest='dir')
    parser.add_argument('--season', dest='se', nargs='+', type=int)
    args = parser.parse_args()

    my_tool = Create_csv_files(args.dir, args.se)
    my_tool.scrape_season()


if __name__ == "__main__":
    main()
```