Import Modules

In [1]:
```
#import necessary modules
import requests
from bs4 import BeautifulSoup
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import datetime
import dataframe_image as dfi
import sqlite3
from pandasql import sqldf
import re
import csv

sqlit = lambda q: sqldf(q, globals())

conn = sqlite3.connect('pdga.sqlite')
global players_events_df
players_events_df = pd.read_sql_query("SELECT * FROM players_events", conn)

my_date = datetime.date.today()
this_year, this_week, today = my_date.isocalendar()
```

Database Functions

In [2]:
```
#database functions

def load_pe():
    global players_events_df
    players_events_df = pd.read_sql_query("SELECT * FROM players_events", conn)

def save_pe():
    players_events_df.to_sql('players_events', con = conn, if_exists = 'replace', index = False)
```

Define Tournament Class and Methods

In [3]:

```
#define Tournament class

class Tournament(object):
    '''
    This class is for tournaments.
    '''
    tourney_list = []
    tourney_list_nick = []

    def __init__(self, new_name = None, new_week = None, new_year = None, new_tier = None, new_nick = None, new
        self.name = new_name
        self.week = new_week
        self.year = new_year
        self.tier = new_tier
        self.nick = new_nick
        self.id = new_id
        self.mbest = new_mbest
        self.wbest = new_wbest
        self.limit = new_limit

    def what_info(self):
        print('.set_info(name, week, year, tier, nick, id, mbest, wbest, limit(if any))')

    def set_info(self, new_name, new_week, new_year, new_tier, new_nick, new_id, new_mbest, new_wbest, new_limi
        self.name = new_name
        self.week = new_week
        self.year = new_year
        self.tier = new_tier
        self.nick = new_nick
        self.id = new_id
        self.mbest = new_mbest
        self.wbest = new_wbest
        self.limit = new_limit
        print('".__dict__" to check or ".research_merge_clean()" to load')

    def df(self):
        return pd.read_sql_query("SELECT * FROM {}".format(self.nick), conn)

    def to_db(self):
        df = players_events_df[(players_events_df['{}_par'.format(self.nick)].notnull()) | (players_events_df['
        df = df.rename(columns = {'id' : 'player_id'})
        df.to_sql('{}'.format(self.nick), con = conn, if_exists = 'replace', index = False)


    def make_html(self):
        url = 'https://www.pdga.com/tour/event/' + str(self.id)
        event = requests.get(url)
        doc = '{}.html'.format(self.nick)
        with open(doc, 'w') as f:
            f.write(event.text)

    def parse_to_df(self):
        doc = '{}.html'.format(self.nick)
        with open(doc) as ti:
            soup = BeautifulSoup(ti, 'html.parser')

        places_t = soup.find_all('td', class_ = 'place')
        places = []
        for entry in places_t:
            place = entry.get_text()
            places.append(int(place))

        ids_t = soup.find_all('td', class_ = 'pdga-number')
        ids = []
        for entry in ids_t:
            id = entry.get_text()
            ids.append(id)

        pars_t = soup.find_all(True, {'class':['par under', 'par over', 'par', 'dnf']})
        pars = []
        for entry in pars_t:
            par = entry.get_text()
            if par == 'E':
                par = 0
            pars.append(par)
        pars.remove('Par')
        pars.remove('Par')

        prizes_t = soup.find_all('td', class_ = 'prize')
        prizes = []
        for entry in prizes_t:
            prize = entry.get_text()
            if not prize:
                prize = '$0'
            prize = prize.replace(',', '')
            prize = prize.strip('$')
            prizes.append(int(prize))

        totals_t = soup.find_all('td', class_ = 'total')
        totals = []
        for entry in totals_t:
            total = entry.get_text()
```

```
                totals.append(total)

            df = pd.DataFrame(list(zip(places, ids, pars, prizes, totals)), columns = ['{}_place'.format(self.nick)

            df.loc[df.total == 'DNF', '{}_place'.format(self.nick)] = np.nan
            df.loc[df.total == 'DNF', '{}_par'.format(self.nick)] = np.nan
            df.loc[df.total == 'DNF', '{}_prize'.format(self.nick)] = np.nan
            df.loc[df.total == 'DNF', '{}_DNF'.format(self.nick)] = 'yes'
            df.loc[df.total != 'DNF', '{}_DNF'.format(self.nick)] = 'no'

            df = df.drop(columns = 'total')

            df = df.head(self.limit)

            return df

        def merge_pe(self, df):
            global players_events_df
            players_events_df = players_events_df.merge(df, left_on = 'id', right_on = 'player_id', how = 'outer')


        def research_merge_clean(self):
            self.make_html()
            df = self.parse_to_df()
            self.merge_pe(df)
            clean_pe()
            print('Any more null names in p_e (check_nulls())? Any strange divisions in p_e(check_divisions())?')

        def add_to_txt(self):
            entry = ["","{},{},{},{},{},{},{},{},{},{}".format(self.nick, self.name, self.week, self.year, self.tie
            with open('tournaments.txt', 'a') as file:
                file.writelines("\n".join(entry))

        def participants_list(self):
            return self.df()['player_id'].to_list()

        def strength(self):
            top_money_MPO_df = sqlit("SELECT id, total_cash from players_events_df WHERE division = 'MPO' ORDER BY
            top_money_FPO_df = sqlit("SELECT id, total_cash from players_events_df WHERE division = 'FPO' ORDER BY
            top_money_MPO_df['spoints'] = 0
            top_money_FPO_df['spoints'] = 0
            top_money_MPO_list = top_money_MPO_df['id'].tolist()
            top_money_FPO_list = top_money_FPO_df['id'].tolist()

            for ind, row in top_money_MPO_df.iterrows():
                top_money_MPO_df.loc[ind, 'spoints'] = 20 - ind
            for ind, row in top_money_FPO_df.iterrows():
                top_money_FPO_df.loc[ind, 'spoints'] = 20 - ind

            mst = 0
            for p in top_money_MPO_list:
                if p in self.participants_list():
                    for ind, row in top_money_MPO_df.iterrows():
                        if row['id'] == p:
                            mst += row['spoints']
            if mst >= 150:
                ms = 1
            else:
                ms = .9**((150-mst)/10)

            fst = 0
            for p in top_money_FPO_list:
                if p in self.participants_list():
                    for ind, row in top_money_FPO_df.iterrows():
                        if row['id'] == p:
                            fst += row['spoints']
            if fst >= 150:
                fs = 1
            else:
                fs = .9**((150-fst)/10)

            return (ms, fs)

        def pr_value(self):

            tier_penalty_dict = {'Major' : 1.2, 'NT' : 1, 'A' : 0.8, 'A/B' : 0.8, 'B/A': 0.8}

            my_date = datetime.date.today()
            this_year, this_week, day_of_week = my_date.isocalendar()

            tier_pen = tier_penalty_dict[self.tier]
            age_pen = .96**(this_week - int(self.week) + (52*(this_year - int(self.year))))
            mstr = self.strength()[0]
            fstr = self.strength()[1]

            return (round(tier_pen*age_pen*mstr, 4), round(tier_pen*age_pen*fstr, 4))
```

Add Tournament-Relevant functions

In [4]:
```
#other functions

def clean_pe():
    global players_events_df
    players_events_df_nulls = players_events_df[players_events_df['name'].isnull()]

    for ind in players_events_df_nulls.index:

        try:
            player_page = "https://www.pdga.com/player/" + players_events_df_nulls['player_id'][ind]
        except:
            continue
        player_get = requests.get(player_page)
        soup = BeautifulSoup(player_get.content, 'html.parser')
        name_t = soup.find(True, {'class' : ['panel-pane pane-page-title', 'panel-pane pane-page-title E']})
        soup2 = BeautifulSoup(str(name_t), 'html.parser')
        name_u = soup2.find('h1')
        try:
            name_z = name_u.text
        except:
            continue
        name_parts = name_z.split()
        name = name_parts[0][0] + '.' + ' ' + name_parts[1]
        players_events_df.loc[players_events_df.player_id == players_events_df_nulls['player_id'][ind], 'name']

        #adding their id
        players_events_df.loc[players_events_df.player_id == players_events_df_nulls['player_id'][ind], 'id'] =

        #adding their division
        division_t = soup.find('td', class_ = 'division')
        try:
            division = division_t.get_text()
        except:
            continue
        players_events_df.loc[players_events_df.player_id == players_events_df_nulls['player_id'][ind], 'divisi

        #adding their country

        location_t = soup.find('li', class_ = 'location')
        try:
            location_u = location_t.text
        except:
            continue
        location_u_split = location_u.split(':')
        location = location_u_split[1]
        country_t = location.split(',')[-1]
        country_u = country_t.replace('Classification', '')
        country = country_u.strip()
        players_events_df.loc[players_events_df.player_id == players_events_df_nulls['player_id'][ind], 'countr

        #adding their state_province

        state_t = location.split(',')[-2]
        state = state_t.strip()
        players_events_df.loc[players_events_df.player_id == players_events_df_nulls['player_id'][ind], 'state_

    #and drop the player_id column

    players_events_df = players_events_df.drop(columns = 'player_id')

    players_events_df['division'] = sqlit("SELECT REPLACE(division, 'Open Women', 'FPO') as division FROM playe
    players_events_df['division'] = sqlit("SELECT REPLACE(division, 'Advanced Women', 'FPO') as division FROM p
    players_events_df['division'] = sqlit("SELECT REPLACE(division, 'Open', 'MPO') as division FROM players_eve
    players_events_df['division'] = sqlit("SELECT REPLACE(division, 'Advanced', 'MPO') as division FROM players
    players_events_df['division'] = sqlit("SELECT REPLACE(division, 'Pro Masters 50+', 'MPO') as division FROM 
    players_events_df['division'] = sqlit("SELECT REPLACE(division, 'Intermediate', 'MPO') as division FROM pla

def check_nulls():
    if not players_events_df[players_events_df['name'].isnull()].any().any():
        return print ('No name nulls!')
    else:
        return players_events_df[players_events_df['name'].isnull()]

def check_divisions():
    if players_events_df[(players_events_df['division'] != 'MPO') & (players_events_df['division'] != 'FPO')].e
        return print('Divisions OK!')
    else:
        return players_events_df[(players_events_df['division'] != 'MPO') & (players_events_df['division'] != '

def tourneys_list():
    return Tournament.tourneys_list

def participants_list_dictionary():
    participants_list_dictionary = {}
    for t in tourneys_list():
        participants_list_dictionary[t] = t.participants_list()
    return participants_list_dictionary

def search_tournaments(text):
    for tn in tourney_list:
        if re.match(r'[A-Za-z \.]*{}[A-Za-z \.]*'.format(text), tn.name):
            print(tn.__dict__)
```

Run Stats!

In [5]:

```
################## Run Stats!

#events_played column

def run_stats():

    #events_played column

    events_played_list = []
    for ind in players_events_df.index:
        n = 0
        for tn in tourney_list_nick:
            if pd.notnull(players_events_df[f'{tn}_par'][ind]):
                n += 1
        events_played_list.append(n)
    players_events_df['events_played'] = events_played_list

    #DNFs to 'No'

    for index, row in players_events_df.iterrows():
        for tn in tourney_list_nick:
            if pd.isnull(players_events_df[f'{tn}_DNF'][index]):
                players_events_df.loc[index, f'{tn}_DNF'] = 'no'

    #no par = no place


    for tn in tourney_list_nick:
        players_events_df.loc[players_events_df[f'{tn}_par'].isnull(), f'{tn}_place'] = 0

    #null prize = no prize

    for tn in tourney_list_nick:
        players_events_df.loc[players_events_df[f'{tn}_prize'].isnull(), f'{tn}_prize'] = 0

    #average place

    for index, row in players_events_df.iterrows():
        total_place = 0
        for tn in tourney_list_nick:
            total_place += int(row[f'{tn}_place'])
        events_played = row['events_played']
        if events_played != 0:
            players_events_df.loc[index, 'average_place'] = total_place/events_played
        else:
            players_events_df.loc[index, 'average_place'] = 999

    #total cash

    players_events_df['total_cash'] = 0
    for tn in tourney_list_nick:
        players_events_df['total_cash'] += players_events_df[f'{tn}_prize'].astype(int)

    #total cash this year

    players_events_df[f'total_cash_{str(this_year)}'] = 0
    for tn in tourney_list_nick:
        if tournaments()[tn].year == str(this_year):
            players_events_df[f'total_cash_{str(this_year)}'] += players_events_df[f'{tn}_prize'].astype(int)


    #calculate cash_std

    for g in ['MPO', 'FPO']:
        for tn in tourney_list_nick:
            df = players_events_df[(players_events_df[f'{tn}_prize'].notnull()) & (players_events_df[f'{tn}_pri
            if df.empty:
                std = np.nan
            else:
                array = df[f'{tn}_par'].to_numpy()
                array = array.astype(int)
                std = array.std()
            players_events_df.loc[players_events_df.division == g, f'{tn}_cash_std'] = std

    #compute std away

    for tn in tourney_list:
        players_events_df.loc[players_events_df.division == 'MPO', '{}_par_best'.format(tn.nick)] = tn.mbest
        players_events_df.loc[players_events_df.division == 'FPO', '{}_par_best'.format(tn.nick)] = tn.wbest

    for tn in tourney_list_nick:
        players_events_df[f'{tn}_par'] = players_events_df[f'{tn}_par'].astype(float)
        players_events_df[f'{tn}_par_best'] = players_events_df[f'{tn}_par_best'].astype(float)
        players_events_df[f'{tn}_std_away'] = (players_events_df[f'{tn}_par'] - players_events_df[f'{tn}_par_be

     #compute avg std away
    avg_std_away = []
    for index, row in players_events_df.iterrows():
        total_std = 0
        events = 0
        for tn in tourney_list_nick:
            if pd.isnull(players_events_df[f'{tn}_std_away'][index]):
                continue
            else:
                total_std += players_events_df[f'{tn}_std_away'][index]
                events += 1
        if events == 0:
            asa = np.nan
        else:
            asa = total_std/events
        avg_std_away.append(asa)
    players_events_df['avg_std_away'] = avg_std_away

    #build player_prize_df


    col_list = ['name', 'id', 'division', 'total_cash', f'total_cash_{str(this_year)}', 'sponsor_2021', 'events
    for tn in tourney_list_nick:
        if tournaments()[tn].year == str(this_year):
            col = '{}_prize'.format(tn)
            col_list.append(col)

    global player_prize_df
    player_prize_df = players_events_df[col_list].copy()
    for index in player_prize_df.index:
        if player_prize_df.loc[index, 'events_played'] == 0:
            player_prize_df.loc[index, 'avg_prize'] = np.nan
        else:
            player_prize_df.loc[index, 'avg_prize'] = player_prize_df.loc[index, 'total_cash']/player_prize_df.

    participants_list_dictionary = {}
    for t in tourney_list:
        participants_list_dictionary[t] = t.participants_list()
```

Define Player Class and Methods

In [6]:

```
class Player:
    player_list = []
    player_list_name = []
    def __init__(self, new_instance_name, new_name, new_id, new_div, new_sponsor = None):
        self.instance_name = new_instance_name
        self.name = new_name
        self.id = new_id
        self.sponsor = new_sponsor
        self.division = new_div
        self.player_list.append(self)
        self.player_list_name.append(new_name)

    @classmethod
    def get_by_id(cls, value):
        return [inst for inst in player_list if inst.id == value][0]

    def place(self, tournament):
        if tournament not in tourney_list_nick:
            print('Please use a correct tourney nick as an argument')
        elif self.id not in tournaments()[tournament].participants_list():
            print(self.name + " didn't play at " + tournaments()[tournament].name + "!")
        else:
            return int(players_events_df.loc[players_events_df.id == self.id, f'{tournament}_place'].item())

    def tourney_results(self, year):
        print('Name: ' + self.name + ' ID: ' + self.id)
        for tn in tourney_list:
            if tn.year == year:
                if self.id in tn.participants_list():
                    print(tn.name + ': ' + str(self.place(tn.nick)))

    def change_sponsor(self, new_sponsor, year):
        self.sponsor = new_sponsor
        players_events_df.loc[players_events_df['id'] == self.id, 'sponsor_{}'.format(year)] = new_sponsor



#POWER RANKING!!!

    def power_ranking(self):

        power_ranking_list = []

        place_points_dict = {0 : 1 , 1 : 1000, 2 : 750, 3 : 600, 4 : 500, 5 : 400, 6 : 350, 7 : 300, 8 : 250, 9
        tier_penalty_dict = {'Major' : 1.2, 'NT' : 1, 'A' : 0.8, 'A/B' : 0.8, 'B/A': 0.8}

        my_date = datetime.date.today()
        this_year, this_week, day_of_week = my_date.isocalendar()
        for tn in tourney_list:
            if self.id in tn.participants_list():
                tpoints = place_points_dict[self.place(tn.nick)]
                tier_pen = tier_penalty_dict[tn.tier]
                age_pen = .96**(this_week - int(tn.week) + (52*(this_year - int(tn.year))))
                if self.division == 'MPO':
                    tscore = tpoints*tn.strength()[0]*tier_pen*age_pen
                else:
                    tscore = tpoints*tn.strength()[1]*tier_pen*age_pen
                power_ranking_list.append(tscore)

        while len(power_ranking_list) >10:
            power_ranking_list.remove(min(power_ranking_list))

        power_ranking = sum(power_ranking_list)
        return power_ranking
```

Define Player-Related Lists and Dictionaries

In [7]:

```
player_list = Player.player_list
player_list_name = Player.player_list_name
```

Define Player-Related Functions

In [8]:

```
def players():
    '''
    Creates an Player instance for every player in players_events_df. Returns a dictionary
    in which the keys are the instance names and the values are the instances
    '''
    global instancelist
    instancelist = []
    dct = {}
    for ind, row in players_events_df.iterrows():
        instance_name = row['name'].replace('. ', '')
        if instance_name in instancelist:
            instance_name = instance_name + '1'
        n=2
        while instance_name in instancelist:
            instance_name = instance_name[:-1]
            instance_name = instance_name + str(n)
            n+=1
        instancelist.append(instance_name)
        dct[instance_name] = Player(instance_name, row['name'], row['id'], row['division'], row['sponsor_2021']
    return dct

def check_name(pname):
    '''
    Given a possible instance name or part of a name, returns count of how many instances contain that pattern
    '''
    dups = []
    for n in instancelist:
        if re.match('[a-zA-Z \-\']*{}[a-zA-Z \-\']*'.format(pname), n):
            dups.append(n)
    if len(dups) == 0:
        print("There are no players with that name!")
    elif len(dups) == 1:
        print("There is only one " + pname + ": " + players()[pname].name + ' id: ' + players()[pname].id)
    else:
        print("Here are the players with that name:")
        for dup in dups:
            print(dup +": " + players()[dup].name+ ' id: ' + players()[dup].id)

def pr_pts_earned(playernick, tournamentinst):

    if players()[playernick].division == 'MPO':
        x = 0
    else:
        x = 1

    place_points_dict = {0 : 1 , 1 : 1000, 2 : 750, 3 : 600, 4 : 500, 5 : 400, 6 : 350, 7 : 300, 8 : 250, 9 : 2
    ppe = tournamentinst.pr_value()[x]*place_points_dict[players()[playernick].place(tournamentinst.nick)]
    rppe = round(ppe, 2)
    return "({})".format(rppe).ljust(10)

def head_to_head(player1, player2, inc = None):

    print('                                                     ' + players()[player1].name + '         vs.

    if players()[player1].division == players()[player1].division:
        x = None
        if players()[player1].division == 'MPO':
            x = 0
        else:
            x = 1
        for tn in tourney_list:
            if players()[player1].id in tn.participants_list() and players()[player2].id in tn.participants_lis
```

```
                print(tn.name.ljust(35) + str(tn.pr_value()[x]).ljust(15) + '       ' + str(players()[player1].p
            elif inc == 'all':
                if players()[player1].id in tn.participants_list() and players()[player2].id not in tn.particip
                    print(tn.name.ljust(35) + str(tn.pr_value()[x]).ljust(15) + '       ' + str(players()[player
                if players()[player1].id not in tn.participants_list() and players()[player2].id in tn.particip
                    print(tn.name.ljust(35) + str(tn.pr_value()[x]).ljust(15) + '       ' + 'DNP'.ljust(18) + '
        print('')
        print('Total PR points (Top 10 results)'.ljust(57) + str(round(players()[player1].power_ranking(),2)).l

    else:
        for tn in tourney_list:
            if players()[player1].id in tn.participants_list() and players()[player2].id in tn.participants_lis
                print(tn.name.ljust(35) + str(tn.pr_value()) + '       ' + str(players()[player1].place(tn.nick)
            elif inc == 'all':
                if players()[player1].id in tn.participants_list() and players()[player2].id not in tn.particip
                    print(tn.name.ljust(35) + str(tn.pr_value()) + '       ' + str(players()[player1].place(tn.n
                if players()[player1].id not in tn.participants_list() and players()[player2].id in tn.particip
                    print(tn.name.ljust(35) + str(tn.pr_value()) + '       ' + 'DNP' + '            ' + str(playe
```

Other Functions

In [9]:

```
def pr_df():
    '''Creates power ranking dataframe as power_rankings_df, doesnt return anything'''
    #powerrating
#   1) Tournament place weighted (top 20): 100, 75, 60, 50, 40, 35, 30, 25, 20, 15, 10, 9, 8, 7, 6, 5, 4, 3, 2,
#   2) Decreased by tier
#   3) Decreased by how many weeks ago
#   4) Decreased by weak field

    place_points_dict = {0 : 1 , 1 : 1000, 2 : 750, 3 : 600, 4 : 500, 5 : 400, 6 : 350, 7 : 300, 8 : 250, 9 : 2
    tier_penalty_dict = {'Major' : 1.2, 'NT' : 1, 'A' : 0.8, 'A/B' : 0.8, 'B/A' : 0.8}


#build the df

    global power_rankings_df
    power_rankings_df = sqlit(

    '''
    SELECT name
        ,id
        ,division
        ,sponsor_2021
    FROM players_events_df
    '''

    )

    for tn in tourney_list_nick:
        power_rankings_df[f'{tn}_place'] = players_events_df[f'{tn}_place']


#Find today's week for later use...

    my_date = datetime.date.today()
    year, this_week, day_of_week = my_date.isocalendar()

#create lists of top money earners

    top_money_MPO_df = sqlit("SELECT id, total_cash from players_events_df WHERE division = 'MPO' ORDER BY tota
    top_money_FPO_df = sqlit("SELECT id, total_cash from players_events_df WHERE division = 'FPO' ORDER BY tota
    top_money_MPO_list = top_money_MPO_df['id'].tolist()
    top_money_FPO_list = top_money_FPO_df['id'].tolist()

#Create place_points based on place for each tournament and add to df


    for tn in tourney_list_nick:

        place_points_list = []
        for ind in power_rankings_df.index:


            place = power_rankings_df[f'{tn}_place'][ind]
            if int(place) > 30 or int(place) == 0:
                place_points = 0
            else:
                place_points = place_points_dict[int(place)]

            place_points_list.append(place_points)

        power_rankings_df[f'{tn}_place_points'] = place_points_list

#create and fill time penalty columns (4 weeks per penalty)

        age_penalty_zone = this_week - int(tournaments()[tn].week) + (52*(this_year - int(tournaments()[tn].yea
        power_rankings_df[f'{tn}_age_penalty'] =.96**age_penalty_zone

#create and fill tier penalty columns
```

```
        tier_penalty_zone = tier_penalty_dict[tournaments()[tn].tier]
        power_rankings_df[f'{tn}_tier_penalty'] = tier_penalty_zone

#create strength penalty columns

        power_rankings_df[f'{tn}_MPO_strength_penalty'] = tournaments()[tn].strength()[0]

        power_rankings_df[f'{tn}_FPO_strength_penalty'] = tournaments()[tn].strength()[1]

#compile the final scores

    power_ranking_list = []
    for ind in power_rankings_df.index:
        scores = []
        for tn in tourney_list_nick:
            if power_rankings_df.division[ind] == 'MPO':
                power_points = power_rankings_df[f'{tn}_place_points'][ind]*power_rankings_df[f'{tn}_tier_penal
            else:
                power_points = power_rankings_df[f'{tn}_place_points'][ind]*power_rankings_df[f'{tn}_tier_penal
            scores.append(power_points)
            while len(scores) > 10:
                scores.remove(min(scores))
            pp = sum(scores)
        power_ranking_list.append(pp)
    power_rankings_df['total_points'] = power_ranking_list
    return power_rankings_df

def pr_30_m():
    df = power_rankings_df[power_rankings_df['division'] == 'MPO'].sort_values('total_points', ascending = Fals
    df.index = df.index + 1
    return df

def pr_30_f():
    df = power_rankings_df[power_rankings_df['division'] == 'FPO'].sort_values('total_points', ascending = Fals
    df.index = df.index + 1
    return df
```

Load all Basic Tournament Data

In [10]:

```
def tournaments():
    global tourney_list
    tourney_list = []
    global tourney_list_nick
    tourney_list_nick = []
    with open('tournaments.txt', 'r') as csvfile:
        tdict = {}
        for row in csv.reader(csvfile):
            tdict[row[0]] = Tournament(*row[1:])
    for n in tdict.keys():
        tourney_list.append(tdict[n])
    for key in tdict.keys():
        tourney_list_nick.append(key)
    return tdict
```

Do you need these loaded?

In [11]:

```
players()
tournaments()
#pr_df() #creates df for power rankings called power_rankings_df
#run_stats() #does some math. Creates some stats. Creates player_prize_df
```

Out[11]:

```
{'NorCal2020': <__main__.Tournament at 0x7fd0ff3610d0>,
 'MBO2020': <__main__.Tournament at 0x7fd0ff361610>,
 'WOM2020': <__main__.Tournament at 0x7fd0ff361cd0>,
 'Oklahoma2020': <__main__.Tournament at 0x7fd0ff361f10>,
 'Hub2020': <__main__.Tournament at 0x7fd0ff361d60>,
 'VPO2020': <__main__.Tournament at 0x7fd0ff361a30>,
 'NWA2020': <__main__.Tournament at 0x7fd0ff361bb0>,
 'LCT2020': <__main__.Tournament at 0x7fd0ff361310>,
 'Belton2020': <__main__.Tournament at 0x7fd0ff361fa0>,
 'Holiday2020': <__main__.Tournament at 0x7fd0ff361190>,
 'Chain2020': <__main__.Tournament at 0x7fd0ff3617c0>,
 'SSM': <__main__.Tournament at 0x7fd0fbde0850>,
 'VO': <__main__.Tournament at 0x7fd1030df100>,
 'LVC': <__main__.Tournament at 0x7fd1030df1c0>,
 'Memorial': <__main__.Tournament at 0x7fd1030df1f0>,
 'Waco': <__main__.Tournament at 0x7fd1030df220>,
 'SP': <__main__.Tournament at 0x7fd1030df250>,
 'SK': <__main__.Tournament at 0x7fd1030df280>,
 'Belton': <__main__.Tournament at 0x7fd1030df2b0>,
 'Paradise': <__main__.Tournament at 0x7fd1030df2e0>,
 'Texas': <__main__.Tournament at 0x7fd1030df310>,
 'TVC': <__main__.Tournament at 0x7fd1030df340>,
 'Dogwood': <__main__.Tournament at 0x7fd1030df370>,
 'Vintage': <__main__.Tournament at 0x7fd1030df3a0>,
 'JB': <__main__.Tournament at 0x7fd1030df3d0>,
 'MAO': <__main__.Tournament at 0x7fd1030df400>,
 'BGO': <__main__.Tournament at 0x7fd1030df430>,
 'DDO': <__main__.Tournament at 0x7fd1030df460>,
 'COM': <__main__.Tournament at 0x7fd1030df490>,
 'GHP': <__main__.Tournament at 0x7fd1030df4c0>,
 'Dust': <__main__.Tournament at 0x7fd1030df4f0>,
 'LPO': <__main__.Tournament at 0x7fd1030df520>,
 'Rumble': <__main__.Tournament at 0x7fd1030df550>,
 'Three': <__main__.Tournament at 0x7fd1030df580>,
 'HUK': <__main__.Tournament at 0x7fd1030df5b0>,
 'PH': <__main__.Tournament at 0x7fd1030df5e0>,
 'Kitsap': <__main__.Tournament at 0x7fd1030df610>,
 'OTB': <__main__.Tournament at 0x7fd1030df640>,
 'Norm': <__main__.Tournament at 0x7fd1030df670>,
 'Mich': <__main__.Tournament at 0x7fd1030df6a0>,
 'USW': <__main__.Tournament at 0x7fd1030df6d0>,
 'Tam': <__main__.Tournament at 0x7fd1030df700>,
 'SCM': <__main__.Tournament at 0x7fd1030df730>,
 'FSO': <__main__.Tournament at 0x7fd1030df760>,
 'Port': <__main__.Tournament at 0x7fd1030df790>,
 'Tenn': <__main__.Tournament at 0x7fd1030df7c0>,
 'NT3': <__main__.Tournament at 0x7fd1030df7f0>,
 'Turk': <__main__.Tournament at 0x7fd1030df820>,
 'Utah': <__main__.Tournament at 0x7fd1030df850>,
 'HPC': <__main__.Tournament at 0x7fd1030df880>,
 'GCC': <__main__.Tournament at 0x7fd1030df8b0>,
 'Mega': <__main__.Tournament at 0x7fd1030df8e0>,
 'Titan': <__main__.Tournament at 0x7fd1030df910>,
 'ProWorlds': <__main__.Tournament at 0x7fd1030df940>}
```

In [12]:
```
#you can run them here
```

Tournament to do list:

1) T = Tournament()

2) T.set_info...

3) T.research_merge_clean()

4) Check data. check_nulls(). check_divisions(). Fix.

5) All good?

6) T.to_db()

7) T.add_to_txt()

8)tournaments()

9) run_stats()

10) save_pe()

In [13]:
```
head_to_head('PMcBeth', 'RWysocki', 'all')
```

```
                                                    P. McBeth        vs.   R. Wysocki:

Hub City Halloween Open 2020       0.0742             DNP                      1  (74.2)
Las Vegas Classic                  0.4996             9  (99.92)                5  (199.84)
Memorial Open                      0.2144             1  (214.4)               DNP
Waco Charity Open                  0.5421             5  (216.84)               11 (54.21)
The Open at Belton                 0.3981             2  (298.57)               1  (398.1)
Texas States                       0.5882             3  (352.92)               1  (588.2)
Vintage Open                       0.436              DNP                      5  (174.4)
Jonesboro                          0.6648             5  (265.92)               1  (664.8)
Dynamic Discs Open                 0.7214             1  (721.4)                2  (541.05)
Goat Hill Park                     0.1828             DNP                      1  (182.8)
Huk Central                        0.1809             2  (135.68)              DNP
OTB Open                           0.7828             6  (273.98)               4  (391.4)
Santa Cruz Masters                 0.8493             4  (424.65)               14 (72.19)
Portland Open                      0.8847             2  (663.52)               2  (663.52)
Utah Open                          0.3276             DNP                      2  (245.7)
Pro Worlds                         1.2                2  (900.0)                7  (360.0)
/nTotal PR points                                      4332.2007444849305        4235.384217687053
```

In [ ]: