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'
```

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_ti
        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(i

    def set_info(self, new_name, new_week, new_year, new_tier, new_nick, new_id,
        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)].no
        df.rename(columns = {'id' : 'player_id'})
        df.to_sql('{}'.format(self.nick), con = conn, if_exists = 'replace', ind


    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', '
        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

        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


    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

    def add_to_txt(self):
        entry = ["","{},{},{},{},{},{},{},{},{},{}".format(self.nick, self.name,
        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 W
        top_money_FPO_df = sqlit("SELECT id, total_cash from players_events_df W
        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)
```

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_nul
        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', 'pane
        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_n

        #adding their id
        players_events_df.loc[players_events_df.player_id == players_events_df_n
```

```
        #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_n

        #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_n

        #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_n

    #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'
    players_events_df['division'] = sqlit("SELECT REPLACE(division, 'Advanced Wo
    players_events_df['division'] = sqlit("SELECT REPLACE(division, 'Open', 'MPO
    players_events_df['division'] = sqlit("SELECT REPLACE(division, 'Advanced',
    players_events_df['division'] = sqlit("SELECT REPLACE(division, 'Pro Masters
    players_events_df['division'] = sqlit("SELECT REPLACE(division, 'Intermediat

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_eve
        return print('Divisions OK!')
    else:
        return players_events_df[(players_events_df['division'] != 'MPO') & (pla

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}_pl

    #null prize = no prize

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

    #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_p
        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'].asty

#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())
        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_s

#compute std away

for tn in tourney_list:
    players_events_df.loc[players_events_df.division == 'MPO', '{}_par_best'
    players_events_df.loc[players_events_df.division == 'FPO', '{}_par_best'

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

 #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', '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,
```

```
    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
        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].nam
        else:
            return int(players_events_df.loc[players_events_df.id == self.id, f'

    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_{}'.f



#POWER RANKING!!!

    def power_ranking(self):

        power_ranking_list = []

        place_points_dict = {0 : 1 , 1 : 1000, 2 : 750, 3 : 600, 4 : 500, 5 : 40
        tier_penalty_dict = {'Major' : 1.2, 'NT' : 1, 'A' : 0.8, 'A/B' : 0.8, 'B

        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(
                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
    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['
    return dct

def check_name(pname):
    '''
    Given a possible instance name or part of a name, returns count of how many
    '''
    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
```

```
    else:
        print("Here are the players with that name:")
        for dup in dups:
            print(dup +": " + players()[dup].name+ ' id: ' + players()[dup].id)

def head_to_head(player1, player2, inc = None):
    print('                    ' + players()[player1].name + '   vs.    ' + players
    for tn in tourney_list:
        if players()[player1].id in tn.participants_list() and players()[player2
            print(tn.name.ljust(20) + '      ' + str(players()[player1].place(tn
        elif inc == 'all':
            if players()[player1].id in tn.participants_list() and players()[pla
                print(tn.name.ljust(20) + '      ' + str(players()[player1].plac
            if players()[player1].id not in tn.participants_list() and players()
                print(tn.name.ljust(20) + '      ' + 'DNP' + '           ' + str
```

Other Functions

In [9]:

```
def pr_df():
    '''Creates power ranking dataframe as power_rankings_df, doesnt return anyth
    #powerrating
#   1) Tournament place weighted (top 20): 100, 75, 60, 50, 40, 35, 30, 25, 20, 1
#   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
    tier_penalty_dict = {'Major' : 1.2, 'NT' : 1, 'A' : 0.8, 'A/B' : 0.8, 'B/A'


#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
    top_money_FPO_df = sqlit("SELECT id, total_cash from players_events_df WHERE
```

```
    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_y
        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].stre

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

#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]*powe
            else:
                power_points = power_rankings_df[f'{tn}_place_points'][ind]*powe
            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('
    df.index = df.index + 1
    return df
```

```
def pr_30_f():
    df = power_rankings_df[power_rankings_df['division'] == 'FPO'].sort_values('
    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]:

```
{'SSM': <__main__.Tournament at 0x7fea39b84220>,
 'VO': <__main__.Tournament at 0x7fea39b842e0>,
 'LVC': <__main__.Tournament at 0x7fea39b84310>,
 'Memorial': <__main__.Tournament at 0x7fea39b84340>,
 'Waco': <__main__.Tournament at 0x7fea39b84370>,
 'SP': <__main__.Tournament at 0x7fea39b843a0>,
 'SK': <__main__.Tournament at 0x7fea39b843d0>,
 'Belton': <__main__.Tournament at 0x7fea39b84400>,
 'Paradise': <__main__.Tournament at 0x7fea39b84430>,
 'Texas': <__main__.Tournament at 0x7fea39b84460>,
 'TVC': <__main__.Tournament at 0x7fea39b84490>,
 'Dogwood': <__main__.Tournament at 0x7fea39b844c0>,
 'Vintage': <__main__.Tournament at 0x7fea39b844f0>,
 'JB': <__main__.Tournament at 0x7fea39b84520>,
 'MAO': <__main__.Tournament at 0x7fea39b84550>,
 'BGO': <__main__.Tournament at 0x7fea39b84580>,
 'DDO': <__main__.Tournament at 0x7fea39b845b0>,
 'COM': <__main__.Tournament at 0x7fea39b845e0>,
 'GHP': <__main__.Tournament at 0x7fea39b84610>,
 'Dust': <__main__.Tournament at 0x7fea39b84640>,
 'LPO': <__main__.Tournament at 0x7fea39b84670>,
 'Rumble': <__main__.Tournament at 0x7fea39b846a0>,
 'Three': <__main__.Tournament at 0x7fea39b846d0>,
 'HUK': <__main__.Tournament at 0x7fea39b84700>,
 'PH': <__main__.Tournament at 0x7fea39b84730>,
 'Kitsap': <__main__.Tournament at 0x7fea39b84760>,
```

```
'OTB': <__main__.Tournament at 0x7fea39b84790>,
'Norm': <__main__.Tournament at 0x7fea39b847c0>,
'Mich': <__main__.Tournament at 0x7fea39b847f0>,
'USW': <__main__.Tournament at 0x7fea39b84820>,
'Tam': <__main__.Tournament at 0x7fea39b84850>,
'SCM': <__main__.Tournament at 0x7fea39b84880>,
'FSO': <__main__.Tournament at 0x7fea39b848b0>,
'Port': <__main__.Tournament at 0x7fea39b848e0>,
'Tenn': <__main__.Tournament at 0x7fea39b84910>,
'NT3': <__main__.Tournament at 0x7fea39b84940>,
'Turk': <__main__.Tournament at 0x7fea39b84970>,
'Utah': <__main__.Tournament at 0x7fea39b849a0>,
'HPC': <__main__.Tournament at 0x7fea39b849d0>,
'GCC': <__main__.Tournament at 0x7fea39b84a00>,
'Mega': <__main__.Tournament at 0x7fea39b84a30>,
'Titan': <__main__.Tournament at 0x7fea39b84a60>,
'Chain2020': <__main__.Tournament at 0x7fea39b84a90>,
'Holiday2020': <__main__.Tournament at 0x7fea39b84ac0>,
'Belton2020': <__main__.Tournament at 0x7fea39b84af0>}
```

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]:
```
run_stats()
```

In [14]:
```
pr_df()
```

Out[14]:

	name	id	division	sponsor_2021	SSM_place	VO_place	LVC_place	Memorial_pla
0	C. Allen	44184	FPO	Prodigy	0	0	2	
1	M. Gannon	85942	FPO	Discraft	0	0	9	
2	S. Hokom	34563	FPO	MVP	0	0	6	
3	H. King	81351	FPO	Discraft	0	0	5	
4	V. Mandujano	62879	FPO	Innova	0	3	10	
...	...	...	...	...	...	...	...	
2138	D. Hungerford	148947	MPO	None	0	0	0	
2139	J. Sanders	19858	MPO	None	0	0	0	
2140	A. Sloan	67404	FPO	None	0	0	0	
2141	A. Dre	89793	FPO	None	0	0	0	
2142	L. Carlsen	134010	FPO	None	0	0	0	

2143 rows × 275 columns

In [18]:
```
pr_30_m()
```

Out[18]:

	name	total_points
1	R. Wysocki	4117.638740
2	P. McBeth	3618.776027
3	E. McMahon	3386.892557
4	C. Heimburg	3269.888287
5	A. Hammes	1888.349073
6	C. Dickerson	1854.069130
7	K. Jones	1830.300747
8	J. Conrad	1526.702404
9	K. Klein	1519.984876
10	N. Locastro	1428.831405
11	D. Gibson	1271.176801
12	A. Barela	988.248426
13	E. Keith	953.159993
14	G. Rathbun	931.933312
15	J. Proctor	876.577743
16	P. Ulibarri	873.735759

	name	total_points
17	M. Orum	843.301516
18	E. Aderhold	833.862480
19	G. Gurthie	817.679784
20	N. Queen	793.490583
21	J. Freeman	791.228210
22	C. White	720.039653
23	B. Callaway	608.694746
24	B. Williams	606.394040
25	T. Rothlisberger	554.390706
26	G. Barsby	539.756532
27	R. Newsom	490.844952
28	M. Ford	489.431729
29	C. Leiviska	483.164300
30	M. Bell	462.727799

In [33]:
```
#drop all no event people
```

Out[33]:
```
[0]
```

In [34]:
```
dfi.export(pr_30_m(), 'prm_top_30_062821_post.png')
```

In [35]:
```
dfi.export(pr_30_f(), 'prf_top_30_062821_post.png')
```

In [ ]:
```

```