Weighted Average Combine Approach

Using 2020 NFL Combine data to better understand positional strengths to develop a data-based athleticism score

In [68]:
```
import math as math
import numpy as np
import pandas as pd
import zipfile
import os
```

Our first task is reading the csv from our Combine data export found on Pro Football Reference and converting it to a usable dataframe

In [98]:
```
combine = pd.read_csv("combine.csv")
combine
```

Out[98]:

	Player	Pos	School	College	Ht	Wt	40yd	Vertical	Bench	Broad Jump	3Cone	Shuttle	(tm
0	Trey Adams\AdamTr00	OL	Washington	College Stats	6-8	318	5.60	24.5	NaN	92.0	NaN	NaN	
1	Hakeem Adeniji\AdenHa00	OL	Kansas	College Stats	6-4	302	5.17	34.0	26.0	115.0	NaN	NaN	C B 6th pic
2	McTelvin Agim\AgimMc00	DL	Arkansas	College Stats	6-3	309	4.98	NaN	27.0	NaN	NaN	NaN	B 3 pic
3	Salvon Ahmed\AhmeSa00	RB	Washington	College Stats	5-11	197	4.62	34.5	NaN	120.0	NaN	NaN	
4	Brandon Aiyuk\AiyuBr00	WR	Arizona State	College Stats	6-0	205	4.50	40.0	11.0	128.0	NaN	NaN	F 49e 25
...	...	...	...	...	...	...	...	...	...	...	...	...	
332	D.J. Wonnum\WonnDJ00	DL	South Carolina	College Stats	6-5	258	4.73	34.5	20.0	123.0	7.25	4.44	Mi 4th pic
333	Dom Wood-Anderson\WoodDo01	TE	Tennessee	College Stats	6-4	261	4.92	35.0	NaN	119.0	NaN	NaN	
334	David Woodward\WoodDa04	LB	Utah State	College Stats	6-2	230	4.79	33.5	16.0	114.0	7.34	4.37	
335	Chase Young\YounCh04	DL	Ohio State	College Stats	6-5	264	NaN	NaN	NaN	NaN	NaN	NaN	Was Tea 2
336	Jabari Zuniga\ZuniJa00	DL	Florida	College Stats	6-3	264	4.64	33.0	29.0	127.0	NaN	NaN	N Je 79

337 rows × 13 columns

Since our height is not in a functional format, we define a function `parse_ht` that is able to convert our entries as floats that we can use for our analysis

In [99]:
```
def parse_ht(ht):
    ht_ = ht.split("-")
    ft_ = float(ht_[0])
    in_ = float(ht_[1])
    return (12*ft_) + in_
```

In [101…
```
combine['HtNum'] = combine['Ht'].apply(lambda x: parse_ht(x))
```

To simply the process of hard-coding every position for all the events, we create an array that has all the unique positions except Kickers (stored as `unique_positions`) and an array of the events (stored as `events`)

In [102…
```
events = ["HtNum", "Wt", "40yd", "Vertical", "Broad Jump", "3Cone", "Shuttle"]
unique_positions = combine.Pos.unique()
unique_positions = np.delete(unique_positions, 7)
```

We define a function `create_position_average` that takes in a string arguement (position) and calculates the average for all the combine events for the given position

In [113…
```
def create_position_averages(pos):
    position = combine[combine["Pos"] == pos]
    avg = []
    for i in events:
        avg.append(position[i].mean())
    return avg
```

An example of the function for Quarterbacks is shown below

In [114…
```
create_position_averages("QB")
```

Out[114…
```
[75.23529411764706,
 222.76470588235293,
 4.786153846153846,
 31.923076923076923,
 116.0,
 7.2441666666666675,
 4.5075]
```

Our next task involves us looking at how to convert the raw averages to functional values that we can compare against each other. To do this, we calculate the mean and standard deviation of the population (all Combine athletes) and compute a z-score for each event by position. The z-score will then be converted to a percentile that can be used to make a radar graph.

We create a list similar to the one created by our `create_position_average` function for the mean and standard deviation of the Combine athletes stored in `combine_mean` and `combine_std` respectively.

In [115…
```
combine_mean = []
combine_std = []
for i in events:
    combine_mean.append(combine[i].mean())
    combine_std.append(combine[i].std())
```

The `compute_z` takes in the position as a string argument and uses the z-score formula below to calculate the z-score for each event by position:

$$z = \frac{x_i - \mu}{\sigma}$$

In [116…

```
def compute_z(pos):
    avg = create_position_averages(pos)
    z = []
    for i in np.arange(7):
        z.append((avg[i] - combine_mean[i])/(combine_std[i]))
    return z
```

An example of the function for Quarterbacks is shown below

In [117…

```
compute_z("QB")
```

Out[117…

```
[0.5442463193902194,
 -0.3997427653411131,
 0.1782657514288664,
 -0.4033742320536672,
 -0.3460456898575482,
 -0.2158857124908091,
 0.18694906051444318]
```

The `z_to_tile` converts the z-score to appropriate the percentile

In [118…

```
def z_to_tile(z_score):
    return (.5 * (math.erf(z_score / 2 ** .5) + 1)) * 100
```

The last thing we do is apply these functions appropriately to get our final percentiles by position. We will print these out and manually add them to our Google Sheets to do further analysis

In [120…

```
for i in unique_positions:
    tile = []
    z = compute_z(i)
    for j in z:
        tile.append(z_to_tile(j))
    print(tile)
```

```
[84.53840154231361, 95.21086716980048, 95.42932163760088, 12.499345679627, 7.8958677124236
71, 93.26492178174422, 92.00210455758338]
[71.55470102482074, 84.05014719133568, 75.93785463008938, 21.13585497765763, 28.8040522368
35183, 75.39110953069982, 73.00109994595522]
[6.76960009573771, 26.45638433348838, 26.090790055527872, 67.21902318649876, 62.9949346332
1713, 28.098505769852267, 23.980193691192753]
[41.62124358233166, 21.44884585191573, 20.940063177124337, 73.88860221481143, 71.707952979
99537, 30.640950918809907, 37.457151281883014]
[17.604593452482366, 14.569719154652333, 18.991969942953666, 74.09210935541991, 78.9901081
98, 20.019545891899206, 15.771655812835416]
[80.58007609022046, 58.609163512563015, 52.3681069776022, 54.391769755096774, 41.321018482
06165, 37.69022933049453, 37.17100901092634]
[54.23249662797572, 49.30158083017649, 39.8081796249434, 55.65476855252203, 62.15027351399
2325, 27.229180101764783, 27.02044935643747]
[25.43861639165417, 21.834334425291367, 25.686481357211775, 71.91213642115444, 77.85167770
578636, 12.82185064616071, 17.052862954414916]
[70.6864009316741, 34.46729951063801, 57.074286281480326, 34.33364700399576, 36.4654188944
45136, 41.4538417142213, 57.4149713747986]
[35.06074693499986, 21.03051238331949, 63.23277129893856, 46.18402309421133, 13.1690927271
773, 21.830266889237954, 44.43686010702655]
[73.94590831902721, 42.31500636908914, 74.14211194468928, 10.564982712639653, 36.465418894
445136, 44.63187364069907, 40.557289929059294]
```