In [88]:
```
torch.save(eccentricity_dfz, 'eccentricity_dfz.pth')
eccentricity_dfz = torch.load("eccentricity_dfz.pth", weights_only=Fals
e)
eccentricity_dfz.shape
```

Out[88]:
```
(173150, 40)
```

In [ ]:
```

```

In [ ]:
```

```

In [89]:
```
eccentricity_dfz.shape
```

Out[89]:
```
(173150, 40)
```

In [90]:
```
eccentricity_dfz.head()
```

```
/usr/local/lib/python3.11/dist-packages/pandas/io/formats/format.p
y:1458: RuntimeWarning: invalid value encountered in greater
  has_large_values = (abs_vals > 1e6).any()
/usr/local/lib/python3.11/dist-packages/pandas/io/formats/format.p
y:1459: RuntimeWarning: invalid value encountered in less
  has_small_values = ((abs_vals < 10 ** (-self.digits)) & (abs_vals
> 0)).any()
/usr/local/lib/python3.11/dist-packages/pandas/io/formats/format.p
y:1459: RuntimeWarning: invalid value encountered in greater
  has_small_values = ((abs_vals < 10 ** (-self.digits)) & (abs_vals
> 0)).any()
```

Out[90]:

	game_id	play_id	nfl_id	week	quarter	down	arxy	ars	ara	
0	2023090700	101	43290	1	1	3	4.285046	17.689530	13.986375	
1	2023090700	101	44930	1	1	3	1272.317907	29.623264	10.836651	
2	2023090700	101	53541	1	1	3	7.995704	16.799922	5.577154	
3	2023090700	101	53959	1	1	3	15.961717	21.050669	1.979233	
4	2023090700	101	46137	1	1	3	8.025947	24.178478	77.014893	

In [91]:
```
# ((0.001110)^2)/((0.035468)^2)
((0.001110)**2.0)/((0.030741)**2.0)
```

Out[91]:
```
0.0013037969949412584
```

In [92]:

```
# def calculate_eccentricity_fx(A, B, C, s, acc, dis, o, dir_, theta_) :
def calculate_eccentricity_fx(A, B, C) :
    eig_val1 = 0.5 * (A + C + np.sqrt((A - C)**2 + B**2))
    eig_val2 = 0.5 * (A + C - np.sqrt((A - C)**2 + B**2))
    a = np.sqrt(1 / np.abs(eig_val1))
    b = np.sqrt(1 / np.abs(eig_val2))
    a, b = sorted([a, b], reverse=True)

    return (a**2/b**2)


# p0 = torch.tensor([1, 1, 1, 1, 1, 1], dtype=torch.float32, requires_gr
ad=True, device=device)

a, b, c =0.001110, -0.035468, -0.030741 #0.965663        0.965806
0.998866

# arxy =
calculate_eccentricity_fx(a,b,c)
```

Out[92]:

```
4.285145980848879
```

Show hidden code

In [ ]:

```
import pandas as pd
import numpy as np
from tqdm import tqdm
from scipy.optimize import minimize


# Objective function to minimize
def objective_fn(params, x, y):
    return torch.sum(conics(params, x, y)**2)


import torch
from torch.optim import LBFGS

# # Define the closure function for optimization


# def closure():
def closure(optimiser, sx ,sy, params ):
    optimizer.zero_grad()
    # Split params into sx and sy (assuming sx and sy are of the same si
ze)


    # Now call the objective function with sx and sy
    loss = objective_fn(params, sx, sy)
    loss.backward()
    return loss



# Define conic equation
def conics(params, x, y):
    a, b, c, d, e, f = params
    return a * x**2 + b * x * y + c * y**2 + d * x + e * y + f


def curvature_conic(A, B, C, D, E, x, y):
    # First derivative (dy/dx)
    numerator_first = -2*A*x - D
    denominator_first = B*x + 2*C*y + E
    dy_dx = numerator_first / denominator_first
```

```
    # Second derivative (d^2y/dx^2)
    # Derivatives of the numerator and denominator
    d_numerator_first_dx = -2*A
    d_denominator_first_dx = B

    # Using the quotient rule for the second derivative
    numerator_second = (d_numerator_first_dx * denominator_first) - (nu
merator_first * d_denominator_first_dx)
    denominator_second = denominator_first**2

    d2y_dx2 = numerator_second / denominator_second

    # Curvature calculation
    curvature = d2y_dx2 / (1 + dy_dx**2)**(3/2)

    return curvature



def average_curvature(A, B, C, D, E, x_start, x_end, num_points=100):
    # Create a tensor of x values, using torch.linspace
    x_values = torch.linspace(x_start, x_end, num_points, device='cuda'
if torch.cuda.is_available() else 'cpu')

    # Initialize the curvature_values tensor
    curvature_values = torch.zeros_like(x_values)

    # Compute the curvature for each x value
    for i, x in enumerate(x_values):
        y = 0  # Fixed y value for the calculation (this can be modified
based on specific requirements)
        curvature_values[i] = curvature_conic(A, B, C, D, E, x, y)

    # Calculate the average curvature using torch.mean
    average_curv = torch.mean(curvature_values)

    return average_curv







# #hyperbolic only
```

```
# def calculate_eccentricity(A, B, C, s, acc, dis, o, dir_) :
# def calculate_eccentricity_fx(A, B, C, s, acc, dis, o, dir_, theta_) :
def calculate_eccentricity_fx(A, B, C) :
    eig_val1 = 0.5 * (A + C + np.sqrt((A - C)**2 + B**2))
    eig_val2 = 0.5 * (A + C - np.sqrt((A - C)**2 + B**2))
    a = np.sqrt(1 / np.abs(eig_val1))
    b = np.sqrt(1 / np.abs(eig_val2))

    a, b = sorted([a, b], reverse=True)


    return (a**2/b**2)



# Initial guess for the conic parameters
# p0 = [1, 1, 1, 1, 1, 1]
p0 = torch.tensor([1, 1, 1, 1, 1, 1], dtype=torch.float32, requires_gra
d=True, device=device)


# Initialize the main dictionary to store no transforms
no_transforms = {}

# Summary stats for tracking processing
play_count_with_snap_event = 0
play_count_without_snap_event = 0
events_without_snap = {}

n = 50#200 #50 #500  # Subset for testing
# tracking_data_dict_play_subset = dict(list(tracking_data_dict_play.ite
ms())[:n])
tracking_data_dict_play_subset = dict(list(tracking_data_dict_play_op.i
tems()))

###NEWLY
# tracking_data_dict_play_subset = dict_gp


# Device setup: Use GPU if available
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")


for (gameId, playId), group_data in tqdm(tracking_data_dict_play_subse
```

```
t.items()) :

    # print(group_data.columns)
    # print(group_data['output'].columns)
    # print(group_data['output']['nfl_id'].value_counts())
    # print(group_data['output']['nfl_id'])

    # Initialize gameId in no_transforms if not already present
    if gameId not in no_transforms:
        no_transforms[gameId] = {}

    # Initialize playId within the gameId if not already present
    if playId not in no_transforms[gameId]:
        no_transforms[gameId][playId] = {
            'offense': {},
            'defense': {},
        }


    # offensiveTeamAbbr = plays[(plays['gameId'] == gameId) & (plays['pl
ayId'] == playId)].iloc[0]['possessionTeam']
    # defensiveTeamAbbr = plays[(plays['gameId'] == gameId) & (plays['pl
ayId'] == playId)].iloc[0]['defensiveTeam']
    offensiveTeamAbbr = sup_data[(sup_data['game_id'] == gameId) & (sup
_data['play_id'] == playId)].iloc[0]['possession_team']
    defensiveTeamAbbr = sup_data[(sup_data['game_id'] == gameId) & (sup
_data['play_id'] == playId)].iloc[0]['defensive_team']

    # Retrieve tracking data for the current game/play
    # tracking_data = group_data['tracking_data']
    # tracking_data = group_data['input']
    tracking_data_op = group_data['output']
    # print(tracking_data_op.shape)
    # print(group_data['output'])
    # print(tracking_data_op)
    # print(tracking_data_op.groupby('nfl_id'))



    #     # Skip plays with no output
    # if tracking_data_op is None or not isinstance(tracking_data_op, p
d.DataFrame) or tracking_data_op.empty:
    #     # print(f"Skipping play (no output): {key}")
    #     continue
```

```
    # for nflId, player_data in group_data['output'].groupby('nfl_id'):
    for nflId, player_data in tracking_data_op.groupby('nfl_id'):

        # print('not skipping')
        # print(tracking_data_op.columns)
        if pd.isna(nflId):  # Skip if it's NaN (ball)
        # if pd.isna(nfl_id):
            continue



        snap_event_idx = 30
        # player_data_truncated = player_data

        if pd.notna(snap_event_idx):
            # player_data_truncated = player_data.loc[:snap_event_idx]
            player_data_truncated = player_data



            x_ = torch.tensor(player_data_truncated['x'].to_list(), dev
ice=device)
            y_ = torch.tensor(player_data_truncated['y'].to_list(), dev
ice=device)




##################################

#             params = torch.tensor(p0, requires_grad=True, device=devic
e)
            params = p0.clone().detach().requires_grad_(True).to(devic
e)
            optimizer = LBFGS([params])
            optimizer.step(lambda: closure(optimizer, x_, y_, params))
# xx or xa
            optimized_params = params.detach()
            a,b,c,d,e,f = optimized_params.tolist() # aa bb cc ..
#             theta_ = torch.arctan2(ab, aa - ac) / 2
            theta_ = torch.arctan2(torch.tensor(b, device=device), torc
h.tensor(a - c, device=device)) / 2
            cos_theta_, sin_theta_ = torch.cos(theta_), torch.sin(theta
```

```
_)


            # ecc
            # eccentricity = calculate_eccentricity(player_data['a'], pl
ayer_data['b'], player_data['c'], torch.mean(s_), torch.mean(a_), torch.
mean(dis_), torch.mean(o_), torch.mean(dir_), theta_)

            arxy = calculate_eccentricity_fx(a, b, c)


#             arxy = calculate_eccentricity_fx(a, b, c, np.mean(s_), np.
mean(a_), np.mean(dis_), np.mean(o_), np.mean(dir_), theta_) #aspect rat
io

            # if player_data['club'].iloc[0] == offensiveTeamAbbr:
            if player_data['player_side'].iloc[0] == "Offense":
#                 no_transforms[gameId][playId]['offense'][nflId] = {'ec
centricity': eccentricity, 'disp':disp, 'theta': theta_, 'cuv':curvatur
e, 'cuvx': avg_cuvx, 'cuvy': avg_cuvy, 'disc':discriminant, 'major':al,
'minor':bl, 'arxy':arxy, 'ars':ars, 'ara': ara, 'theta_s': theta_s, 'the
ta_a':theta_a, 'xl':xl, 'yl':yl, 'x_mean':x_mean, 'y_mean':y_mean, 'game
id': group_data['meta_data']['gameId'], 'playid': group_data['meta_dat
a']['playId'], 'a'=a,'b'= b,'c'=c,'d'=d,'e'=e,'f'=f, 'sa'=sa, 'sb'=sb,
'sc'=sc, s'sd'=d, 'se'=se, 'sf'sf, 'aa'=aa, 'ab'=ab, 'ac'= ac, 'ad'=ad,
'ae'=ae, 'af'=af}
#                 no_transforms[gameId][playId]['offense'][nflId] =
{'a'=a,'b'= b,'c'=c,'d'=d,'e'=e,'f'=f, 'sa'=sa, 'sb'=sb, 'sc'=sc, s'sd'=
d, 'se'=se, 'sf'sf, 'aa'=aa, 'ab'=ab, 'ac'= ac, 'ad'=ad, 'ae'=ae, 'af'=a
f}
                # no_transforms[gameId][playId]['offense'][nflId] =
{'a':a,'b': b,'c': c,'d': d,'e': e,'f': f, 'sa': sa, 'sb': sb, 'sc':sc,
'sd': sd, 'se':se, 'sf': sf, 'aa': aa, 'ab': ab, 'ac': ac, 'ad': ad, 'a
e': ae, 'af': af, 'xl':xl, 'yl':yl, 'x_mean':x_mean, 'y_mean':y_mean, 't
heta_':theta_, 'theta_s':theta_s, 'theta_a':theta_a, 'gameid': group_dat
a['meta_data']['gameId'], 'playid': group_data['meta_data']['playId']}
#, 'last_event'=last_event}
                no_transforms[gameId][playId]['offense'][nflId] = {'a':
a,'b': b,'c': c,'d': d,'e': e,'f': f, 'x' : x_, 'y' : y_, 'arxy' : arx
y} #, 'last_event'=last_event}

                # 'disp':disp,

            # elif player_data['club'].iloc[0] == defensiveTeamAbbr:
```

```
            elif player_data['player_side'].iloc[0] == "Defense":
#                 no_transforms[gameId][playId]['defense'][nflId] = {'ec
centricity': eccentricity, 'disp':disp, 'theta': theta_, 'cuv':curvatur
e,  'cuvx': avg_cuvx, 'cuvy': avg_cuvy, 'disc':discriminant, 'major':al,
'minor':bl, 'arxy':arxy, 'ars':ars, 'ara': ara, 'theta_s': theta_s, 'the
ta_a':theta_a, 'xl':xl, 'yl':yl, 'x_mean':x_mean, 'y_mean':y_mean, 'game
id': group_data['meta_data']['gameId'], 'playid': group_data['meta_dat
a']['playId']}
                # no_transforms[gameId][playId]['defense'][nflId] =
{'a':a,'b': b,'c': c,'d': d,'e': e,'f': f, 'sa': sa, 'sb': sb, 'sc':sc,
'sd': sd, 'se':se, 'sf': sf, 'aa': aa, 'ab': ab, 'ac': ac, 'ad': ad, 'a
e': ae, 'af': af, 'xl':xl, 'yl':yl, 'x_mean':x_mean, 'y_mean':y_mean, 't
heta_':theta_, 'theta_s':theta_s, 'theta_a':theta_a, 'gameid': group_dat
a['meta_data']['gameId'], 'playid': group_data['meta_data']['playId']}
#, 'last_event'=last_event}
                # 'disp':disp,
                no_transforms[gameId][playId]['defense'][nflId] = {'a':
a,'b': b,'c': c,'d': d,'e': e,'f': f, 'x' : x_, 'y' : y_, 'arxy' : arx
y} #, 'last_event'=last_event}



        else:
            play_count_without_snap_event += 1
            # events_without_snap[(gameId, playId)] = player_data['even
t'].unique()  # Store unique events

# Final summary
print(f"Total plays processed: {len(tracking_data_dict_play_subset)}")
print(f"Plays with 'ball_snap' or 'snap_direct' event: {play_count_with
_snap_event}")
print(f"Plays without 'ball_snap' or 'snap_direct' event: {play_count_w
ithout_snap_event}")



# torch.save(no_transforms, 'no_transforms[all].pth')
```

In [ ]:

```
# print(tracking_data_op)
```

In [95]:
```
torch.save(no_transforms, 'no_transforms4[all].pth')


no_transforms4 = torch.load('/kaggle/working/no_transforms4[all].pth',
weights_only=False)
```

In [ ]:
```
game_id = 2023091709 #2023102908      #2023122406
play_id = 1228 #596 # 1653
nfl_id = 44959 #52433 #47834    #54475

len(no_transforms4)
no_transforms4[game_id][play_id]['offense'][nfl_id]#[54481]
```

In [ ]:

```
data_list4 = []

for gameId in tqdm(no_transforms4, desc="Games", unit="game"):
#     print('YG')
# for gameId in no_transforms:
    for playId in no_transforms4[gameId]: #2022101607 1127

        try :

            w = sup_data.loc[sup_data['game_id'] == gameId, 'week'].val
ues[0]#,

            # q = sup_data.loc[(sup_data['game_id'] == gameId) &
            #                              (sup_data['play_id'] == playI
d), 'quarter'].values[0]#,


        except IndexError:
            print('x', gameId, playId)

            # w = None
            # q = None

        for team_type in ['offense', 'defense']:

            for nflId, player_data in no_transforms4[gameId][playId][te
am_type].items():

                try:

                    # pos = tracking_data_ip.loc[(tracking_data_ip['nfl_
id'] == nflId) & (tracking_data_ip['game_id'] == gameId) & (tracking_dat
a_ip['play_id'] == playId), 'player_position'].values[0] #, # cant speak
why becomes lal on coment

                    # eccentricity = calculate_eccentricity(player_data
['a'], player_data['b'], player_data['c'], torch.mean(s_), torch.mean(a
_), torch.mean(dis_), torch.mean(o_), torch.mean(dir_), theta_)


                    # a3 = player_data['a']
                    # b3 = player_data['b']
                    # c3 = player_data['c']
```

```
            # d3 = player_data['d']
            # e3 = player_data['e']
            # f3 = player_data['f']
            a = player_data['a']
            b = player_data['b']
            c = player_data['c']
            d = player_data['d']
            e = player_data['e']
            f = player_data['f']

            x = player_data['x']
            y = player_data['y']
            arxy = player_data['arxy']

            # player_side =


        except IndexError:


            a = None
            b = None
            c = None
            d = None
            e = None
            f = None

        data_list4.append({
            'game_id': gameId,
            'play_id': playId,
            'nfl_id': nflId,


            'a': a,
            'b': b,
            'c' : c,
            'd' : d,
            'e' : e,
            'f' : f,

            'arxy' : arxy,
            'x' : x,
            'y' : y
```

```
                })

# Create a DataFrame from the list
eccentricity_df4 = pd.DataFrame(data_list4)
eccentricity_df4.head()



torch.save(eccentricity_df4, 'eccentricity_df4.pth')
# eccdf.to_parquet('eccentricity_df.parquet', index=False)   # when you
next save
```

In [100]:

```
# torch.save(eccentricity_df4, 'eccentricity_df36.pth')
# eccentricity_df4.head()

eccentricity_df4 = torch.load("eccentricity_df36.pth", weights_only=Fal
se)
```

In [101]:

```
calculate_eccentricity_fx(
             -0.001655521453358233, -0.06709536910057068,  0.0157394390
553236#,
            # 0.0, 0.0, 0.0, 0.0, 0.0,
            # 0.0
        )

# {'a': -0.001655521453358233,
#  'b': -0.06709536910057068,
#  'c': 0.0157394390553236,
#  'd': 0.9706045389175415,
#  'e': 0.970708429813385,
#  'f': 0.9991639852523804,
#  'x': tensor([39.6000, 39.8900, 40.1800, 40.4900, 40.8200, 41.1700, 4
1.5400]),
#  'y': tensor([30.5600, 30.1100, 29.6700, 29.2200, 28.7900, 28.3600, 2
7.9400]),
#  'arxy': 1.5100127524724039}
```

Out[101]:

```
1.5100127524724039
```

In [ ]:
```
```

In [102]:
```
eccentricity_df4.shape
# eccentricity_df4.columns
```

Out[102]:
```
(46045, 12)
```

In [103]:
```
print("input files")
tracking_data_ip = pd.concat([
#     pd.read_csv(f'/kaggle/input/nfl-big-data-bowl-2025/tracking_week_{week}.csv') for week in tqdm(range(6, 8))
    # pd.read_csv(f'/kaggle/input/nfl-big-data-bowl-2025/tracking_week_{week}.csv') for week in tqdm(range(1, 10))
    pd.read_csv(f'/kaggle/input/nfl-big-data-bowl-2026-analytics/114239_nfl_competition_files_published_analytics_final/train/input_2023_w{week:02d}.csv') for week in tqdm(range(1, 19))
])

ip_side = tracking_data_ip[['game_id','play_id','nfl_id','player_side']].drop_duplicates()




eccentricity_df4 = eccentricity_df4.merge(
    ip_side,
    on=['game_id','play_id','nfl_id'],
    how='left'
)
eccentricity_df4.shape
```

```
input files
```

```
100%|██████████| 18/18 [00:15<00:00,  1.19it/s]
```

Out[103]:
```
(46045, 13)
```

In [104]:
```
ip_side.columns
```

Out[104]:
```
Index(['game_id', 'play_id', 'nfl_id', 'player_side'], dtype='objec
t')
```

In [105]:
```
eccentricity_df4.columns
```

Out[105]:
```
Index(['game_id', 'play_id', 'nfl_id', 'a', 'b', 'c', 'd', 'e',
'f', 'arxy',
       'x', 'y', 'player_side'],
      dtype='object')
```

In [106]:
```
torch.save(eccentricity_df4, 'eccentricity_df4.pth')
```

In [ ]:

```
import pandas as pd
import numpy as np
from tqdm import tqdm
from scipy.optimize import minimize


# Objective function to minimize
def objective_fn(params, x, y):
    return torch.sum(conics(params, x, y)**2)


import torch
from torch.optim import LBFGS

# # Define the closure function for optimization


# def closure():
def closure(optimiser, sx ,sy, params ):
    optimizer.zero_grad()
    # Split params into sx and sy (assuming sx and sy are of the same si
ze)


    # Now call the objective function with sx and sy
    loss = objective_fn(params, sx, sy)
    loss.backward()
    return loss



# Define conic equation
def conics(params, x, y):
    a, b, c, d, e, f = params
    return a * x**2 + b * x * y + c * y**2 + d * x + e * y + f


def curvature_conic(A, B, C, D, E, x, y):
    # First derivative (dy/dx)
    numerator_first = -2*A*x - D
    denominator_first = B*x + 2*C*y + E
    dy_dx = numerator_first / denominator_first
```

```
    # Second derivative (d^2y/dx^2)
    # Derivatives of the numerator and denominator
    d_numerator_first_dx = -2*A
    d_denominator_first_dx = B

    # Using the quotient rule for the second derivative
    numerator_second = (d_numerator_first_dx * denominator_first) - (nu
merator_first * d_denominator_first_dx)
    denominator_second = denominator_first**2

    d2y_dx2 = numerator_second / denominator_second

    # Curvature calculation
    curvature = d2y_dx2 / (1 + dy_dx**2)**(3/2)

    return curvature



def average_curvature(A, B, C, D, E, x_start, x_end, num_points=100):
    # Create a tensor of x values, using torch.linspace
    x_values = torch.linspace(x_start, x_end, num_points, device='cuda'
if torch.cuda.is_available() else 'cpu')

    # Initialize the curvature_values tensor
    curvature_values = torch.zeros_like(x_values)

    # Compute the curvature for each x value
    for i, x in enumerate(x_values):
        y = 0  # Fixed y value for the calculation (this can be modified
based on specific requirements)
        curvature_values[i] = curvature_conic(A, B, C, D, E, x, y)

    # Calculate the average curvature using torch.mean
    average_curv = torch.mean(curvature_values)

    return average_curv







# #hyperbolic only
```

```
# def calculate_eccentricity(A, B, C, s, acc, dis, o, dir_) :
# def calculate_eccentricity_fx(A, B, C, s, acc, dis, o, dir_, theta_) :
def calculate_eccentricity_fx(A, B, C) :
    eig_val1 = 0.5 * (A + C + np.sqrt((A - C)**2 + B**2))
    eig_val2 = 0.5 * (A + C - np.sqrt((A - C)**2 + B**2))
    a = np.sqrt(1 / np.abs(eig_val1))
    b = np.sqrt(1 / np.abs(eig_val2))

    a, b = sorted([a, b], reverse=True)


    return (a**2/b**2)



# Initial guess for the conic parameters
# p0 = [1, 1, 1, 1, 1, 1]
p0 = torch.tensor([1, 1, 1, 1, 1, 1], dtype=torch.float32, requires_gra
d=True, device=device)


# Initialize the main dictionary to store no transforms
no_transforms = {}

# Summary stats for tracking processing
play_count_with_snap_event = 0
play_count_without_snap_event = 0
events_without_snap = {}

n = 50#200 #50 #500  # Subset for testing
tracking_data_dict_play_subset = dict(list(tracking_data_dict_play.item
s())[:n])
# tracking_data_dict_play_subset = dict(list(tracking_data_dict_play.ite
ms()))

###NEWLY
# tracking_data_dict_play_subset = dict_gp


# Device setup: Use GPU if available
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")


for (gameId, playId), group_data in tqdm(tracking_data_dict_play_subse
```

```
t.items()) :

    # Initialize gameId in no_transforms if not already present
    if gameId not in no_transforms:
        no_transforms[gameId] = {}

    # Initialize playId within the gameId if not already present
    if playId not in no_transforms[gameId]:
        no_transforms[gameId][playId] = {
            'offense': {},
            'defense': {},
        }


    # offensiveTeamAbbr = plays[(plays['gameId'] == gameId) & (plays['pl
ayId'] == playId)].iloc[0]['possessionTeam']
    # defensiveTeamAbbr = plays[(plays['gameId'] == gameId) & (plays['pl
ayId'] == playId)].iloc[0]['defensiveTeam']
    offensiveTeamAbbr = sup_data[(sup_data['game_id'] == gameId) & (sup
_data['play_id'] == playId)].iloc[0]['possession_team']
    defensiveTeamAbbr = sup_data[(sup_data['game_id'] == gameId) & (sup
_data['play_id'] == playId)].iloc[0]['defensive_team']

    # Retrieve tracking data for the current game/play
    # tracking_data = group_data['tracking_data']
    tracking_data = group_data['input']
    tracking_data_op = group_data['output']
    # print(tracking_data_op.shape)
    # print(group_data['output'])
    # print(tracking_data_op)
    # print(tracking_data_op.groupby('nfl_id'))




    # Process tracking data for each player (nflId)
    # for nflId, player_data in tracking_data.groupby('nfl_id'):
    # for nflId, player_data in tracking_data_op.groupby('nfl_id'):
    for nflId, player_data in group_data['output'].groupby('nfl_id'):

        # print(tracking_data_op.columns)
        if pd.isna(nflId):  # Skip if it's NaN (ball)
        # if pd.isna(nfl_id):
            continue
```

```
        snap_event_idx = 30
        # player_data_truncated = player_data

        if pd.notna(snap_event_idx):
            # player_data_truncated = player_data.loc[:snap_event_idx]
            player_data_truncated = player_data




#             x_ = player_data_truncated['x'].values
            # x_ = torch.tensor(player_data_truncated['x'].values, devic
e=device)
            # y_ = torch.tensor(player_data_truncated['y'].values, devic
e=device)
            x_ = torch.tensor(player_data_truncated['x'].to_list(), dev
ice=device)
            y_ = torch.tensor(player_data_truncated['y'].to_list(), dev
ice=device)



            ################

            # s_ = torch.tensor(player_data_truncated['s'].values, devic
e=device)
            # a_ = torch.tensor(player_data_truncated['a'].values, devic
e=device)
            # # dis_ = torch.tensor(player_data_truncated['dis'].values,
device=device)
            # dir_ = torch.tensor(player_data_truncated['dir'].values, d
evice=device)

            # o_ = torch.tensor(player_data_truncated['o'].values, devic
e=device)

            ################
```

```
            #
# #             disp = np.sum(dis_) #$disall = np.sum(dis_)
#             disp = player_data_truncated['dis'].sum()  # Instead of n
p.sum
#             xl = x_[-1]
#             yl = y_[-1]
# #             x_mean = np.mean(x_)
# #             y_mean = np.mean(y_)
#             x_mean = player_data_truncated['x'].mean()
#             y_mean = player_data_truncated['y'].mean()

#             # Sum of 'dis' (already in CUDA if player_data_truncated
['dis'] is transferred as tensor)
#             disp = dis_.sum()  # PyTorch automatically performs the su
m on GPU


            #
#             # Last elements of x_ and y_ (directly available in GPU te
nsors)
#             xl = x_[-1]
#             yl = y_[-1]

#             # Mean calculations using PyTorch
#             x_mean = x_.mean()  # Mean calculated on GPU
#             y_mean = y_.mean()

            #
            #optimized version
            # disp = dis_.sum()  # Sum of 'dis' directly on GPU

            ###
            # xl = x_[-1]        # Last x-coordinate
            # yl = y_[-1]        # Last y-coordinate
            # x_mean = x_.mean() # Mean of x-coordinates on GPU
            # y_mean = y_.mean() # Mean of y-coordinates on GPU




            #
#             dir_rad = np.radians(dir_)
#             sx = s_*np.sin(dir_rad) #np.mean(s_*np.sin()
#             sy = s_*np.cos(dir_rad)
#             ax = a_*np.sin(dir_rad)
#             ay = a_*np.cos(dir_rad)
```

```
#             param_s = minimize(objective_fn, p0, args=(sx, sy), method
='L-BFGS-B') #xs
#             param_a = minimize(objective_fn, p0, args=(sy, ay), method
='L-BFGS-B')
#             sa,sb,sc,sd,se,sf = param_s.x #$params_s.
#             aa,ab,ac,ad,ae,af = param_a.x
#             theta_s = np.arctan2(sb,sa-sc)/2
#             theta_a = np.arctan2(ab,aa-ac)/2

            #
#             # Convert dir_ to radians
#             dir_rad = torch.radians(dir_)

#             # Trigonometric operations (PyTorch versions)
#             sx = s_ * torch.sin(dir_rad)  # PyTorch handles this on GP
U
#             sy = s_ * torch.cos(dir_rad)
#             ax = a_ * torch.sin(dir_rad)
#             ay = a_ * torch.cos(dir_rad)
#             param_s = minimize(objective_fn, p0, args=(sx.cpu().numpy
(), sy.cpu().numpy()), method='L-BFGS-B')
#             param_a = minimize(objective_fn, p0, args=(sy.cpu().numpy
(), ay.cpu().numpy()), method='L-BFGS-B')

#             # Extract parameters
#             sa, sb, sc, sd, se, sf = param_s.x
#             aa, ab, ac, ad, ae, af = param_a.x

#             # Trigonometric calculations
#             theta_s = torch.arctan2(torch.tensor(sb), torch.tensor(sa
- sc)) / 2
#             theta_a = torch.arctan2(torch.tensor(ab), torch.tensor(aa
- ac)) / 2


#####################################

#             dir_rad = torch.deg2rad(dir_) #not radians
#             sx = s_ * torch.sin(dir_rad)  # PyTorch handles this on GP
U
#             sy = s_ * torch.cos(dir_rad)
#             ax = a_ * torch.sin(dir_rad)
#             ay = a_ * torch.cos(dir_rad)
```

```
#             # Define initial parameters for optimization
# #               params = torch.tensor(p0, requires_grad=True, device=dev
ice)
#             params = p0.clone().detach().requires_grad_(True).to(devic
e)



#             # Create LBFGS optimizer
#             optimizer = LBFGS([params])

# #               # Step through the optimizer
# #               optimizer.step(closure)

#             # Use closure for (sx, sy)
#             optimizer.step(lambda: closure(optimizer, sx, sy, params))
# #               optimizer.step(lambda: closure(optimizer, sx, sy))

#             # After optimization, the parameters are updated. You can
now access the optimized values:
#             optimized_params = params.detach()  # Detach the tensor fr
om the computation graph

#             # Extract the optimized parameters (sa, sb, sc, sd, se, s
f) for further use
#             sa, sb, sc, sd, se, sf = optimized_params.tolist()  # Conv
ert tensor to list for easy usage

#             # Perform any further calculations or use these parameters
in your code
# #               theta_s = torch.arctan2(sb, sa - sc) / 2
#             theta_s = torch.arctan2(torch.tensor(sb, device=device), t
orch.tensor(sa - sc, device=device)) / 2




# #               params = torch.tensor(p0, requires_grad=True, device=dev
ice)
#             params = p0.clone().detach().requires_grad_(True).to(devic
e)
#             optimizer = LBFGS([params])
#             optimizer.step(lambda: closure(optimizer, ax, ay, params))
#             optimized_params = params.detach()
#             aa, ab, ac, ad, ae, af = optimized_params.tolist()
```

```
# #             theta_a = torch.arctan2(ab, aa - ac) / 2
#             theta_a = torch.arctan2(torch.tensor(ab, device=device), t
orch.tensor(aa - ac, device=device)) / 2



##################################

#             params = torch.tensor(p0, requires_grad=True, device=devic
e)
            params = p0.clone().detach().requires_grad_(True).to(devic
e)
            optimizer = LBFGS([params])
            optimizer.step(lambda: closure(optimizer, x_, y_, params))
# xx or xa
            optimized_params = params.detach()
            a,b,c,d,e,f = optimized_params.tolist() # aa bb cc ..
#             theta_ = torch.arctan2(ab, aa - ac) / 2
            theta_ = torch.arctan2(torch.tensor(b, device=device), torc
h.tensor(a - c, device=device)) / 2
            cos_theta_, sin_theta_ = torch.cos(theta_), torch.sin(theta
_)



            # ecc
            # eccentricity = calculate_eccentricity(player_data['a'], pl
ayer_data['b'], player_data['c'], torch.mean(s_), torch.mean(a_), torch.
mean(dis_), torch.mean(o_), torch.mean(dir_), theta_)

            arxy = calculate_eccentricity_fx(a, b, c)


#             arxy = calculate_eccentricity_fx(a, b, c, np.mean(s_), np.
mean(a_), np.mean(dis_), np.mean(o_), np.mean(dir_), theta_) #aspect rat
io

            # if player_data['club'].iloc[0] == offensiveTeamAbbr:
            if player_data['player_side'].iloc[0] == "Offense":
#                 no_transforms[gameId][playId]['offense'][nflId] = {'ec
centricity': eccentricity, 'disp':disp, 'theta': theta_, 'cuv':curvatur
e, 'cuvx': avg_cuvx, 'cuvy': avg_cuvy, 'disc':discriminant, 'major':al,
'minor':bl, 'arxy':arxy, 'ars':ars, 'ara': ara, 'theta_s': theta_s, 'the
ta_a':theta_a, 'xl':xl, 'yl':yl, 'x_mean':x_mean, 'y_mean':y_mean, 'game
id': group_data['meta_data']['gameId'], 'playid': group_data['meta_dat
```

```
a']['playId'], 'a'=a,'b'= b,'c'=c,'d'=d,'e'=e,'f'=f, 'sa'=sa, 'sb'=sb,
'sc'=sc, s'sd'=d, 'se'=se, 'sf'sf, 'aa'=aa, 'ab'=ab, 'ac'= ac, 'ad'=ad,
'ae'=ae, 'af'=af}
#                 no_transforms[gameId][playId]['offense'][nflId] =
{'a'=a,'b'= b,'c'=c,'d'=d,'e'=e,'f'=f, 'sa'=sa, 'sb'=sb, 'sc'=sc, s'sd'=
d, 'se'=se, 'sf'sf, 'aa'=aa, 'ab'=ab, 'ac'= ac, 'ad'=ad, 'ae'=ae, 'af'=a
f}
                # no_transforms[gameId][playId]['offense'][nflId] =
{'a':a,'b': b,'c': c,'d': d,'e': e,'f': f, 'sa': sa, 'sb': sb, 'sc':sc,
'sd': sd, 'se':se, 'sf': sf, 'aa': aa, 'ab': ab, 'ac': ac, 'ad': ad, 'a
e': ae, 'af': af, 'xl':xl, 'yl':yl, 'x_mean':x_mean, 'y_mean':y_mean, 't
heta_':theta_, 'theta_s':theta_s, 'theta_a':theta_a, 'gameid': group_dat
a['meta_data']['gameId'], 'playid': group_data['meta_data']['playId']}
#, 'last_event'=last_event}
                no_transforms[gameId][playId]['offense'][nflId] = {'a':
a,'b': b,'c': c,'d': d,'e': e,'f': f, 'x' : x_, 'y' : y_, 'arxy' : arx
y} #, 'last_event'=last_event}

                # 'disp':disp,

            # elif player_data['club'].iloc[0] == defensiveTeamAbbr:
            elif player_data['player_side'].iloc[0] == "Defense":
#                 no_transforms[gameId][playId]['defense'][nflId] = {'ec
centricity': eccentricity, 'disp':disp, 'theta': theta_, 'cuv':curvatur
e,  'cuvx': avg_cuvx, 'cuvy': avg_cuvy, 'disc':discriminant, 'major':al,
'minor':bl, 'arxy':arxy, 'ars':ars, 'ara': ara, 'theta_s': theta_s, 'the
ta_a':theta_a, 'xl':xl, 'yl':yl, 'x_mean':x_mean, 'y_mean':y_mean, 'game
id': group_data['meta_data']['gameId'], 'playid': group_data['meta_dat
a']['playId']}
                # no_transforms[gameId][playId]['defense'][nflId] =
{'a':a,'b': b,'c': c,'d': d,'e': e,'f': f, 'sa': sa, 'sb': sb, 'sc':sc,
'sd': sd, 'se':se, 'sf': sf, 'aa': aa, 'ab': ab, 'ac': ac, 'ad': ad, 'a
e': ae, 'af': af, 'xl':xl, 'yl':yl, 'x_mean':x_mean, 'y_mean':y_mean, 't
heta_':theta_, 'theta_s':theta_s, 'theta_a':theta_a, 'gameid': group_dat
a['meta_data']['gameId'], 'playid': group_data['meta_data']['playId']}
#, 'last_event'=last_event}
                # 'disp':disp,
                no_transforms[gameId][playId]['defense'][nflId] = {'a':
a,'b': b,'c': c,'d': d,'e': e,'f': f, 'x' : x_, 'y' : y_, 'arxy' : arx
y} #, 'last_event'=last_event}



        else:
            play_count_without_snap_event += 1
```

```
            # events_without_snap[(gameId, playId)] = player_data['even
t'].unique()  # Store unique events

# Final summary
print(f"Total plays processed: {len(tracking_data_dict_play_subset)}")
print(f"Plays with 'ball_snap' or 'snap_direct' event: {play_count_with
_snap_event}")
print(f"Plays without 'ball_snap' or 'snap_direct' event: {play_count_w
ithout_snap_event}")


# torch.save(no_transforms, 'no_transforms[all].pth')
```

In [ ]:

```
# torch.save(no_transforms, 'no_transforms3[all].pth')
```

In [109]:

```
no_transforms3 = torch.load('/kaggle/working/no_transforms3[all].pth',
weights_only=False)
```

In [110]:
```
len(no_transforms3)
no_transforms3[2023120306][1620]['offense'][54481]#[54481]
```

Out[110]:
```
{'a': 0.017734315246343613,
 'b': -0.16258327662944794,
 'c': 0.1559484302997589,
 'd': 0.7624068260192871,
 'e': 0.8475053906440735,
 'f': 0.9914431571960449,
 'x': tensor([54.9300, 54.9100, 54.8600, 54.7600, 54.6100, 54.4100,
54.1600, 53.8700,
         53.5400, 53.1900, 52.8300, 52.4700, 52.1100, 51.7600, 51.4
200, 51.1100,
         50.8200, 50.5500, 50.3000, 50.0700, 49.8700, 49.6900, 49.5
200, 49.3700,
         49.2200, 49.0800, 48.9300, 48.7800]),
 'y': tensor([32.2600, 32.2600, 32.2500, 32.2400, 32.2100, 32.1500,
32.0600, 31.9500,
         31.7900, 31.5900, 31.3400, 31.0300, 30.6700, 30.2600, 29.8
000, 29.2900,
         28.7500, 28.1700, 27.5500, 26.9000, 26.2200, 25.5200, 24.7
900, 24.0600,
         23.3100, 22.5500, 21.8000, 21.0300]),
 'arxy': 9.747557278759214}
```

In [ ]:
```

```

In [111]:
```
len(no_transforms3)
no_transforms3[2023111910][3746]['offense'][54054]#[54481]
```

Out[111]:
```
{'a': -0.027767855674028397,
 'b': 0.033033911138772964,
 'c': -0.053708042949438095,
 'd': 0.968994677066803,
 'e': 0.9769294261932373,
 'f': 0.9992486238479614,
 'x': tensor([59.5500, 59.5500, 59.5500, 59.5600, 59.5700, 59.6000,
59.6300, 59.6800,
         59.7400, 59.8000, 59.8600, 59.9200, 59.9700, 59.9900, 59.9
800, 59.9500,
         59.8800, 59.7800, 59.6400, 59.4900, 59.3200, 59.1300, 58.9
400]),
 'y': tensor([24.2400, 24.2400, 24.2600, 24.2800, 24.3100, 24.3800,
24.5000, 24.7100,
         24.9700, 25.2900, 25.6800, 26.1100, 26.5900, 27.1000, 27.6
400, 28.2000,
         28.7800, 29.3900, 30.0200, 30.6500, 31.3000, 31.9600, 32.6
400]),
 'arxy': 3.128045700576517}
```

In [ ]:
```
calculate_eccentricity_fx(
            0.091603, -0.410195, 0.403104,
            0.0, 0.0, 0.0, 0.0, 0.0,
            0.0
        )
```

In [ ]:
```
calculate_eccentricity_fx(
            -0.027767855674028397, 0.033033911138772964,  -0.0537080429
49438095,
            0.0, 0.0, 0.0, 0.0, 0.0,
            0.0
        )
```

In [ ]:
```
calculate_eccentricity_fx(
            -0.02, 0.03,  -0.05,
            0.0, 0.0, 0.0, 0.0, 0.0,
            0.0
        )
```

In [ ]:
```

```

In [ ]:
```

```

In [113]:

```
data_list3 = []

for gameId in tqdm(no_transforms3, desc="Games", unit="game"):
#     print('YG')
# for gameId in no_transforms:
    for playId in no_transforms3[gameId]: #2022101607 1127

        try :

            w = sup_data.loc[sup_data['game_id'] == gameId, 'week'].val
ues[0]#,

            # q = sup_data.loc[(sup_data['game_id'] == gameId) &
            #                              (sup_data['play_id'] == playI
d), 'quarter'].values[0]#,


        except IndexError:
            print('x', gameId, playId)

            # w = None
            # q = None

        for team_type in ['offense', 'defense']:

            for nflId, player_data in no_transforms3[gameId][playId][te
am_type].items():

                try:

                    # pos = tracking_data_ip.loc[(tracking_data_ip['nfl_
id'] == nflId) & (tracking_data_ip['game_id'] == gameId) & (tracking_dat
a_ip['play_id'] == playId), 'player_position'].values[0] #, # cant speak
why becomes lal on coment

                    # eccentricity = calculate_eccentricity(player_data
['a'], player_data['b'], player_data['c'], torch.mean(s_), torch.mean(a
_), torch.mean(dis_), torch.mean(o_), torch.mean(dir_), theta_)


                    # a3 = player_data['a']
                    # b3 = player_data['b']
                    # c3 = player_data['c']
```

```
            # d3 = player_data['d']
            # e3 = player_data['e']
            # f3 = player_data['f']
            a = player_data['a']
            b = player_data['b']
            c = player_data['c']
            d = player_data['d']
            e = player_data['e']
            f = player_data['f']

            x = player_data['x']
            y = player_data['y']
            arxy = player_data['arxy']


        except IndexError:


            a = None
            b = None
            c = None
            d = None
            e = None
            f = None

        data_list3.append({
            'game_id': gameId,
            'play_id': playId,
            'nfl_id': nflId,


            'a': a,
            'b': b,
            'c' : c,
            'd' : d,
            'e' : e,
            'f' : f,

            'arxy' : arxy,
            'x' : x,
            'y' : y

        })
```

```
# Create a DataFrame from the list
eccentricity_df3 = pd.DataFrame(data_list3)
eccentricity_df3.head()


torch.save(eccentricity_df3, 'eccentricity_df3.pth')
# eccdf.to_parquet('eccentricity_df.parquet', index=False)   # when you
next save
```

```
Games: 100%|██████████| 273/273 [00:05<00:00, 49.98game/s]
```

In [114]:

```
# torch.save(eccentricity_df3, 'eccentricity_df34.pth')
```

In [115]:
```
eccentricity_df3.head()
```

Out[115]:

	game_id	play_id	nfl_id	a	b	c	d	e	f
0	2023090700	101	43290	-0.116455	0.260432	-0.192479	0.620058	0.660246	0.978
1	2023090700	101	44930	0.035372	-0.229709	0.376706	-0.223182	0.618122	0.945

	game_id	play_id	nfl_id	a	b	c	d	e	f
2	2023090700	101	53541	-0.036644	0.089603	-0.174998	0.890057	0.934053	0.995
3	2023090700	101	53959	-0.115783	0.279334	-0.223705	0.888027	0.887744	0.994

	game_id	play_id	nfl_id	a	b	c	d	e	f
4	2023090700	101	46137	-0.030459	0.079638	-0.165843	0.901151	0.941819	0.996

In [116]:

```
# Reload your dataframe
# eccdf = torch.load("eccentricity_df.pth")
eccentricity_dfx = eccentricity_df2.copy()
eccentricity_dfy = eccentricity_df3.copy()

# eccdf = eccdf.merge(sup_add, on=['game_id','play_id'], how='left')
eccentricity_dfz = eccentricity_dfx.merge(eccentricity_dfy, on=['game_i
d','play_id', 'nfl_id'], how='inner')
```

In [ ]:

```
data_list = []

for gameId in tqdm(no_transforms, desc="Games", unit="game"):
#     print('YG')
# for gameId in no_transforms:
    for playId in no_transforms[gameId]: #2022101607 1127

        try :

            w = sup_data.loc[sup_data['game_id'] == gameId, 'week'].val
ues[0]#,

            q = sup_data.loc[(sup_data['game_id'] == gameId) &
                                        (sup_data['play_id'] == playI
d), 'quarter'].values[0]#,


        except IndexError:
            print('x', gameId, playId)

            w = None
            q = None

        for team_type in ['offense', 'defense']:

            for nflId, player_data in no_transforms[gameId][playId][tea
m_type].items():

                try:

                    pos = tracking_data_ip.loc[(tracking_data_ip['nfl_i
d'] == nflId) & (tracking_data_ip['game_id'] == gameId) & (tracking_dat
a_ip['play_id'] == playId), 'player_position'].values[0] #, # cant spea
k why becomes lal on coment

                    eccentricity = calculate_eccentricity(player_data
['a'], player_data['b'], player_data['c'], torch.mean(s_), torch.mean(a
_), torch.mean(dis_), torch.mean(o_), torch.mean(dir_), theta_)


                except IndexError:
```

```
                    w = None
                    q = None
                    d = None

                data_list.append({
                    'game_id': gameId,
                    'play_id': playId,
                    'nfl_id': nflId,


                    'week': w,
                    'quarter': q,
                    'down' : d

                })

# Create a DataFrame from the list
eccentricity_df = pd.DataFrame(data_list)
eccentricity_df.head()


# torch.save(eccentricity_df, 'eccentricity_df.pth')
# eccdf.to_parquet('eccentricity_df.parquet', index=False)   # when you
next save
```

In [ ]:

```
# each row becomes a dict
# data_list2 = eccentricity_df.to_dict(orient="records")
```

In [ ]:

```
# data_list == data_list2
```

In [118]:
```
# Reload your dataframe
# eccdf = torch.load("eccentricity_df.pth")
eccentricity_dfx = eccentricity_df.copy()
eccentricity_dfy = eccentricity_df2.copy()

# Add week and quarter from sup_data
# sup_add = sup_data[['game_id','play_id','week','quarter']].drop_duplic
ates(['game_id','play_id'])

# eccdf = eccdf.merge(sup_add, on=['game_id','play_id'], how='left')
eccentricity_dfz = eccentricity_dfx.merge(eccentricity_dfy, on=['game_i
d','play_id', 'nfl_id'], how='inner')
```

In [119]:
```
len(eccentricity_dfx)
```

Out[119]:
```
173150
```

In [ ]:

In [ ]:

In [120]:
```
eccentricity_df[(eccentricity_df['game_id']==2023090700) & (eccentricit
y_df['play_id']== 101)][['pos', 'player_role']]
```

Out[120]:

	pos	player_role
0	QB	Passer
1	WR	Targeted Receiver
2	WR	Other Route Runner
3	TE	Other Route Runner
4	SS	Defensive Coverage
5	CB	Defensive Coverage
6	MLB	Defensive Coverage
7	CB	Defensive Coverage
8	FS	Defensive Coverage

In [121]:
```
eccentricity_df[(eccentricity_df['player_role']=='Targeted Receiver')]
['pos'].value_counts(dropna=False)
```

Out[121]:
```
pos
WR     8611
TE     3249
RB     2111
FB       77
QB       35
CB       21
T         2
DT        1
ILB       1
Name: count, dtype: int64
```

In [122]:
```
eccentricity_df['pos'].value_counts(dropna=False)
```

Out[122]:
```
pos
WR     38002
CB     37561
FS     17018
TE     14856
QB     14266
SS     13764
RB     11014
ILB    10405
OLB     7297
MLB     7065
FB       658
DE       596
S        481
DT       117
NT        45
T          2
LB         1
K          1
P          1
Name: count, dtype: int64
```

In [123]:
```
eccentricity_df['player_role'].value_counts(dropna=False)
```

Out[123]:
```
player_role
Defensive Coverage    94293
Other Route Runner    50644
Targeted Receiver     14108
Passer                14105
Name: count, dtype: int64
```

In [ ]:
```

```

In [124]:
```
eccentricity_df['week'].value_counts(dropna=False, ascending=True)
eccentricity_df['week'].unique()
```

Out[124]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15,
16, 17,
       18])
```

In [ ]:
```

```

In [125]:
```
eccentricity_df_ = eccentricity_df[eccentricity_df['player_role'] == 'T
argeted Receiver']
```

In [126]:
```
eccentricity_df_.shape
```

Out[126]:
```
(14108, 34)
```

In [127]:

```
route_stats = eccentricity_df_.groupby('pass_result', dropna=False)['ar
xy'].agg(
    mean='mean',
    median='median',
    std='std',
    count='count',
    min='min',
    max='max',
    percentile_25=lambda x: np.percentile(x, 25),
    percentile_75=lambda x: np.percentile(x, 75),
    percentile_90=lambda x: np.percentile(x, 90)
)

# print('Descriptive Statistics')
print(route_stats)
```

```
                   mean     median           std  count       min
\
pass_result
C             524.297224   9.233270  12970.079342   9738  1.002880
I             379.279178  10.501163   3436.639208   4032  1.005501
IN            198.407917  10.115586    684.364196    338  1.025826

                     max  percentile_25  percentile_75  percentile
_90
pass_result
C           1.042796e+06       3.233780      34.812666     282.449
063
I           1.412897e+05       3.694305      43.578995     444.426
050
IN          7.608613e+03       3.647884      35.184993     409.094
813
```

In [128]:

```
route_stats = eccentricity_df.groupby(['pass_result', 'player_role' ],
dropna=False)['arxy'].agg(
    mean='mean',
    median='median',
    std='std',
    count='count',
    min='min',
    max='max',
    percentile_25=lambda x: np.percentile(x, 25),
    percentile_75=lambda x: np.percentile(x, 75),
    percentile_90=lambda x: np.percentile(x, 90)
)

# print('Descriptive Statistics')
print(route_stats)
```

```
                                          mean     median           st
d  count  \
pass_result player_role
C           Defensive Coverage   308.685990  12.360063   7100.15986
0  65285
            Other Route Runner   349.892126   9.091112   6204.15110
5  35112
            Passer               150.571190  15.259530   1807.02680
7   9737
            Targeted Receiver    524.297224   9.233270  12970.07934
2   9738
I           Defensive Coverage   339.417342  12.618185   8980.92273
5  26745
            Other Route Runner   820.217000   8.951329  46195.33707
7  14337
            Passer               885.812926  14.926947  47698.14251
2   4031
            Targeted Receiver    379.279178  10.501163   3436.63920
8   4032
IN          Defensive Coverage   384.988412  12.534791   6775.34210
0   2263
            Other Route Runner  1115.423514   9.004164  31567.08693
6   1195
            Passer               100.154857  15.109401    472.64554
1    337
            Targeted Receiver    198.407917  10.115586    684.36419
6    338

                                       min           max  percentile_
25  \
pass_result player_role
C           Defensive Coverage  1.000364  1.045510e+06       4.3508
34
            Other Route Runner  1.000079  6.739609e+05       3.3932
66
            Passer              1.001481  1.472320e+05       5.7767
30
            Targeted Receiver   1.002880  1.042796e+06       3.2337
80
I           Defensive Coverage  1.000464  1.262552e+06       4.2377
51
            Other Route Runner  1.001064  3.977766e+06       3.3051
28
```

```
                Passer              1.020266  3.027449e+06       5.6769
60
                Targeted Receiver   1.005501  1.412897e+05       3.6943
05
IN              Defensive Coverage  1.001384  2.813866e+05       4.3454
57
                Other Route Runner  1.001627  1.090478e+06       3.0924
96
                Passer              1.001703  5.541536e+03       6.2870
84
                Targeted Receiver   1.025826  7.608613e+03       3.6478
84
```

pass_result	player_role	percentile_75	percentile_90
C	Defensive Coverage	35.755746	183.264044
	Other Route Runner	32.171176	232.476085
	Passer	30.630743	97.185316
	Targeted Receiver	34.812666	282.449063
I	Defensive Coverage	38.915138	230.748450
	Other Route Runner	33.657051	237.798841
	Passer	30.967730	101.572106
	Targeted Receiver	43.578995	444.426050
IN	Defensive Coverage	42.420357	274.214801
	Other Route Runner	33.575127	216.838040
	Passer	32.479538	70.141470
	Targeted Receiver	35.184993	409.094813

In [129]:

```
route_stats = eccentricity_df.groupby(['pass_result', 'player_side' ],
dropna=False)['arxy'].agg(
    mean='mean',
    median='median',
    std='std',
    count='count',
    min='min',
    max='max',
    percentile_25=lambda x: np.percentile(x, 25),
    percentile_75=lambda x: np.percentile(x, 75),
    percentile_90=lambda x: np.percentile(x, 90)
)

# print('Descriptive Statistics')
print(route_stats)
```

```
                                    mean     median           std  count
min  \
pass_result player_side
C           Defense      308.685990  12.360063   7100.159860  65285
1.000364
            Offense      345.450938  10.109979   7440.496690  54587
1.000079
I           Defense      339.417342  12.618185   8980.922735  26745
1.000464
            Offense      752.652530  10.273067  42158.260727  22400
1.001064
IN          Defense      384.988412  12.534791   6775.342100   2263
1.001384
            Offense      766.708643  10.339399  25237.612242   1870
1.001627

                                        max  percentile_25  percentile_75
\
pass_result player_side
C           Defense      1.045510e+06       4.350834      35.755746
            Offense      1.042796e+06       3.630501      31.949726
I           Defense      1.262552e+06       4.237751      38.915138
            Offense      3.977766e+06       3.677353      33.999626
IN          Defense      2.813866e+05       4.345457      42.420357
            Offense      1.090478e+06       3.637398      33.402323

                         percentile_90
pass_result player_side
C           Defense         183.264044
            Offense         209.994737
I           Defense         230.748450
            Offense         231.269968
IN          Defense         274.214801
            Offense         202.466497
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [130]:

```
route_stats = eccentricity_df.groupby('route_ran', dropna=False)['arx
y'].agg(
    mean='mean',
    median='median',
    std='std',
    count='count',
    min='min',
    max='max',
    percentile_25=lambda x: np.percentile(x, 25),
    percentile_75=lambda x: np.percentile(x, 75),
    percentile_90=lambda x: np.percentile(x, 90)
)

# print('Descriptive Statistics')
print(route_stats)
```

	mean	median	std	count	min	max
route_ran						
ANGLE	416.560671	13.187655	6450.094877	6978	1.000464	4.698867e+05
CORNER	324.260843	12.826943	3089.716670	6105	1.003619	1.289601e+05
CROSS	292.508186	9.483081	3829.427537	18012	1.000984	2.208322e+05
FLAT	373.678871	10.941297	19586.056560	24612	1.000323	3.027449e+06
GO	599.090695	11.810070	32132.316888	16884	1.000379	3.977766e+06
HITCH	422.501172	10.855021	22350.240365	32986	1.000079	3.833429e+06
IN	382.643952	11.815512	9533.432518	13705	1.001627	1.042796e+06
OUT	397.861314	12.203457	10359.763553	27036	1.000456	1.262552e+06
POST	341.155066	12.489801	6791.695833	9414	1.000489	5.868392e+05
SCREEN	240.577873	11.729164	2626.312675	3640	1.001871	1.114964e+05
SLANT	229.455474	11.409960	7249.944007	12801	1.001384	7.670438e+05
WHEEL	863.455571	9.063257	18777.343792	952	1.007510	5.745764e+05
NaN	31.870335	4.514713	120.915322	25	1.333532	6.097590e+02

	percentile_25	percentile_75	percentile_90
route_ran			
ANGLE	4.437985	42.997643	362.532073
CORNER	4.264541	45.153823	364.007087
CROSS	3.409126	33.810539	238.893879
FLAT	3.743000	33.185724	175.728069
GO	4.024057	36.415065	215.545956
HITCH	3.898697	32.189373	157.522183
IN	4.146611	38.694729	298.163357
OUT	4.320721	35.699794	216.591160
POST	4.303980	38.383822	265.785450
SCREEN	3.971142	32.278334	134.058048
SLANT	4.167074	30.964244	87.838791

```
WHEEL          3.127752      30.876947     161.757770
NaN            2.978501       8.539467      19.314777
```

In [131]:

```
route_stats = eccentricity_df.groupby('pass_result', dropna=False)['arx
y'].agg(
    mean='mean',
    median='median',
    std='std',
    count='count',
    min='min',
    max='max',
    percentile_25=lambda x: np.percentile(x, 25),
    percentile_75=lambda x: np.percentile(x, 75),
    percentile_90=lambda x: np.percentile(x, 90)
)

# print('Descriptive Statistics')
print(route_stats)
```

```
                   mean     median           std   count       min
\
pass_result
C             325.427917  11.304698   7257.113608  119872  1.000079
I             527.767494  11.516510  29223.404715   49145  1.000464
IN            557.699961  11.228820  17699.371029    4133  1.001384

                     max  percentile_25  percentile_75  percentile
_90
pass_result
C           1.045510e+06       3.993505      33.965205     193.816
853
I           3.977766e+06       3.939089      36.544898     230.841
956
IN          1.090478e+06       3.990309      37.685688     252.896
799
```

In [ ]:

In [132]:

```
route_stats = eccentricity_df.groupby(['pass_result', 'route_ran'], dro
pna=False)['arxy'].agg(
    mean='mean',
    median='median',
    std='std',
    count='count',
    min='min',
    max='max',
    percentile_25=lambda x: np.percentile(x, 25),
    percentile_75=lambda x: np.percentile(x, 75),
    percentile_90=lambda x: np.percentile(x, 90)
)

# print('Descriptive Statistics')
print(route_stats)
```

```
                                  mean     median           std  count
min  \
pass_result route_ran
C           ANGLE       440.001102  13.059346   6988.833307   5811
1.001473
            CORNER      360.350345  13.371606   3183.470810   3107
1.005993
            CROSS       305.006491   9.690375   4334.389347  12838
1.001006
            FLAT        262.545633  11.044850   3571.519245  20943
1.000323
            GO          380.263340  12.086214   6778.190472   7041
1.000379
            HITCH       286.771196  10.903105   7745.791119  25133
1.000079
            IN          360.201968  11.546030  11577.276985   8242
1.001734
            OUT         370.126737  12.138326   7680.706938  18885
1.000456
            POST        403.084869  12.020013   8923.004526   5158
1.000489
            SCREEN      258.407954  11.766005   2813.451162   3145
1.001871
            SLANT       267.786595  11.275478   8572.034006   9100
1.002170
            WHEEL      1676.718932   8.558595  27443.340756    445
1.066619
            NaN          32.849909   4.071708    123.414576     24
1.333532
I           ANGLE       293.625796  13.521371   2445.667563   1041
1.000464
            CORNER      294.028994  12.271006   3124.273207   2726
1.003619
            CROSS       272.494359   8.745671   2201.474669   4717
1.000984
            FLAT       1032.254079  10.397059  50694.661189   3570
1.001064
            GO          677.106944  11.707989  41844.786257   9092
1.001412
            HITCH       893.475152  10.500441  45439.070628   7239
1.003650
            IN          384.262561  12.131360   3535.567100   4921
1.002502
```

```
             OUT         474.519478  12.501804  15159.775750    7772
1.001723
             POST        283.545159  13.497079   2527.920303    3632
1.002403
             SCREEN      127.293820  11.453134    646.324509     495
1.018756
             SLANT       141.125117  12.055179   1094.656671    3458
1.006189
             WHEEL       156.725179   9.729712    872.110171     482
1.007510
IN           ANGLE       351.186532  17.310253   1404.993168     126
1.193449
             CORNER      215.003258  10.321477    825.032022     272
1.109196
             CROSS       147.982980  12.342628    432.405497     457
1.004201
             FLAT        134.789202  11.136406    496.686942      99
1.052382
             GO         1706.200776  10.790014  39807.617131     751
1.010233
             HITCH       425.629610  12.622688   6101.228689     614
1.020061
             IN          709.215273  12.596846  12112.721170     542
1.001627
             OUT         207.835556  10.568662   1991.249859     379
1.044047
             POST        164.560932  10.577303    738.241922     624
1.014499
             SLANT        50.991168   9.585638    166.233008     243
1.001384
             WHEEL        13.129726   5.660145     21.552772      25
1.013313
             NaN           8.360550   8.360550           NaN       1
8.360550

                                  max  percentile_25  percentile_75
\
pass_result route_ran
C            ANGLE      4.698867e+05       4.401874      42.219166
             CORNER     1.019827e+05       4.514986      46.991498
             CROSS      2.208322e+05       3.434231      33.706475
             FLAT       2.522119e+05       3.743839      33.155822
             GO         3.872550e+05       4.114433      36.683918
             HITCH      1.045510e+06       3.954302      31.198002
```

	IN	1.042796e+06	4.109406	36.973850
	OUT	6.739609e+05	4.331342	34.778711
	POST	5.868392e+05	4.306846	37.481099
	SCREEN	1.114964e+05	3.960450	32.261013
	SLANT	7.670438e+05	4.213822	30.814482
	WHEEL	5.745764e+05	2.985975	30.858237
	NaN	6.097590e+02	2.916371	8.692312
I	ANGLE	6.760590e+04	4.618975	46.320823
	CORNER	1.289601e+05	4.091760	44.005397
	CROSS	6.384269e+04	3.270310	32.470880
	FLAT	3.027449e+06	3.711205	33.438046
	GO	3.977766e+06	3.980706	35.964182
	HITCH	3.833429e+06	3.735544	34.990282
	IN	1.219581e+05	4.135282	41.395334
	OUT	1.262552e+06	4.309196	38.271273
	POST	1.157489e+05	4.456848	41.810743
	SCREEN	9.664751e+03	4.160660	32.322604
	SLANT	3.311624e+04	4.080569	31.573972
	WHEEL	1.407585e+04	3.508430	31.685836
IN	ANGLE	1.128622e+04	5.510468	50.047026
	CORNER	8.203397e+03	3.296339	36.916286
	CROSS	5.063095e+03	3.922528	51.788684
	FLAT	3.933641e+03	4.658837	32.200460
	GO	1.090478e+06	3.830236	39.068949
	HITCH	1.461036e+05	4.180836	42.185118
	IN	2.813866e+05	4.682401	41.470203
	OUT	3.828370e+04	4.283506	34.659520
	POST	1.104458e+04	3.610817	33.285933
	SLANT	1.390861e+03	3.671614	23.204235
	WHEEL	1.080684e+02	2.714313	16.664605
	NaN	8.360550e+00	8.360550	8.360550

		percentile_90
pass_result	route_ran	
C	ANGLE	357.804377
	CORNER	419.058584
	CROSS	232.975415
	FLAT	177.095048
	GO	229.981560
	HITCH	139.876196
	IN	270.392474
	OUT	212.385984
	POST	224.658597
	SCREEN	127.907360

```
             SLANT          88.966871
             WHEEL         173.981691
             NaN            20.141824
I            ANGLE         386.057032
             CORNER        308.454090
             CROSS         239.411711
             FLAT          169.442679
             GO            193.562353
             HITCH         219.919327
             IN            337.423073
             OUT           228.537563
             POST          330.465072
             SCREEN        155.211127
             SLANT          84.677824
             WHEEL         161.757770
IN           ANGLE         523.995899
             CORNER        326.391140
             CROSS         461.009082
             FLAT           77.002554
             GO            276.710917
             HITCH         195.489917
             IN            324.376582
             OUT           204.982054
             POST          186.817528
             SLANT          83.215711
             WHEEL          25.566173
             NaN             8.360550
```

```
/usr/local/lib/python3.11/dist-packages/pandas/io/formats/format.p
y:1458: RuntimeWarning: invalid value encountered in greater
  has_large_values = (abs_vals > 1e6).any()
/usr/local/lib/python3.11/dist-packages/pandas/io/formats/format.p
y:1459: RuntimeWarning: invalid value encountered in less
  has_small_values = ((abs_vals < 10 ** (-self.digits)) & (abs_vals
> 0)).any()
/usr/local/lib/python3.11/dist-packages/pandas/io/formats/format.p
y:1459: RuntimeWarning: invalid value encountered in greater
  has_small_values = ((abs_vals < 10 ** (-self.digits)) & (abs_vals
> 0)).any()
```

In [133]:

```
route_stats = eccentricity_df.groupby(['pass_result', 'route_ran'], dro
pna=False)['arxy'].agg(
    mean='mean',
    median='median',
    std='std',
    count='count',
    min='min',
    max='max',
    percentile_25=lambda x: np.percentile(x, 25),
    percentile_75=lambda x: np.percentile(x, 75),
    percentile_90=lambda x: np.percentile(x, 90)
)

# print('Descriptive Statistics')
# print(route_stats)


pivot = route_stats.reset_index().pivot_table(
    index='route_ran',
    columns='pass_result',
    values='mean'   # or 'median', 'std', etc.
)

print(pivot)
```

pass_result	C	I	IN
route_ran			
ANGLE	440.001102	293.625796	351.186532
CORNER	360.350345	294.028994	215.003258
CROSS	305.006491	272.494359	147.982980
FLAT	262.545633	1032.254079	134.789202
GO	380.263340	677.106944	1706.200776
HITCH	286.771196	893.475152	425.629610
IN	360.201968	384.262561	709.215273
OUT	370.126737	474.519478	207.835556
POST	403.084869	283.545159	164.560932
SCREEN	258.407954	127.293820	NaN
SLANT	267.786595	141.125117	50.991168
WHEEL	1676.718932	156.725179	13.129726

```
/usr/local/lib/python3.11/dist-packages/pandas/io/formats/format.p
y:1458: RuntimeWarning: invalid value encountered in greater
  has_large_values = (abs_vals > 1e6).any()
/usr/local/lib/python3.11/dist-packages/pandas/io/formats/format.p
y:1459: RuntimeWarning: invalid value encountered in less
  has_small_values = ((abs_vals < 10 ** (-self.digits)) & (abs_vals
> 0)).any()
/usr/local/lib/python3.11/dist-packages/pandas/io/formats/format.p
y:1459: RuntimeWarning: invalid value encountered in greater
  has_small_values = ((abs_vals < 10 ** (-self.digits)) & (abs_vals
> 0)).any()
```

In [134]:
```
pivot = route_stats.reset_index().pivot_table(
    index='route_ran',
    columns='pass_result',
    values='median'   # or 'median', 'std', etc.
)

print(pivot)
```

```
pass_result          C          I         IN
route_ran
ANGLE        13.059346  13.521371  17.310253
CORNER       13.371606  12.271006  10.321477
CROSS         9.690375   8.745671  12.342628
FLAT         11.044850  10.397059  11.136406
GO           12.086214  11.707989  10.790014
HITCH        10.903105  10.500441  12.622688
IN           11.546030  12.131360  12.596846
OUT          12.138326  12.501804  10.568662
POST         12.020013  13.497079  10.577303
SCREEN       11.766005  11.453134        NaN
SLANT        11.275478  12.055179   9.585638
WHEEL         8.558595   9.729712   5.660145
```

```
/usr/local/lib/python3.11/dist-packages/pandas/io/formats/format.p
y:1458: RuntimeWarning: invalid value encountered in greater
  has_large_values = (abs_vals > 1e6).any()
/usr/local/lib/python3.11/dist-packages/pandas/io/formats/format.p
y:1459: RuntimeWarning: invalid value encountered in less
  has_small_values = ((abs_vals < 10 ** (-self.digits)) & (abs_vals
> 0)).any()
/usr/local/lib/python3.11/dist-packages/pandas/io/formats/format.p
y:1459: RuntimeWarning: invalid value encountered in greater
  has_small_values = ((abs_vals < 10 ** (-self.digits)) & (abs_vals
> 0)).any()
```

In [135]:

```
counts = eccentricity_df.groupby(['pass_result', 'route_ran'], dropna=F
alse).size().reset_index(name='count')
print(counts)
```

	pass_result	route_ran	count
0	C	ANGLE	5811
1	C	CORNER	3107
2	C	CROSS	12838
3	C	FLAT	20943
4	C	GO	7041
5	C	HITCH	25133
6	C	IN	8242
7	C	OUT	18885
8	C	POST	5158
9	C	SCREEN	3145
10	C	SLANT	9100
11	C	WHEEL	445
12	C	NaN	24
13	I	ANGLE	1041
14	I	CORNER	2726
15	I	CROSS	4717
16	I	FLAT	3570
17	I	GO	9092
18	I	HITCH	7239
19	I	IN	4921
20	I	OUT	7772
21	I	POST	3632
22	I	SCREEN	495
23	I	SLANT	3458
24	I	WHEEL	482
25	IN	ANGLE	126
26	IN	CORNER	272
27	IN	CROSS	457
28	IN	FLAT	99
29	IN	GO	751
30	IN	HITCH	614
31	IN	IN	542
32	IN	OUT	379
33	IN	POST	624
34	IN	SLANT	243
35	IN	WHEEL	25
36	IN	NaN	1

In [136]:
```
# Matrix (routes vs pass results) with counts
matrix_counts = eccentricity_df.pivot_table(
    index='route_ran',       # rows = routes
    columns='pass_result',   # cols = outcomes
    values='arxy',           # any column works, we just need something
non-null
    aggfunc='count',
    fill_value=0
)

print(matrix_counts)
```

pass_result	C	I	IN
route_ran			
ANGLE	5811	1041	126
CORNER	3107	2726	272
CROSS	12838	4717	457
FLAT	20943	3570	99
GO	7041	9092	751
HITCH	25133	7239	614
IN	8242	4921	542
OUT	18885	7772	379
POST	5158	3632	624
SCREEN	3145	495	0
SLANT	9100	3458	243
WHEEL	445	482	25

In [137]:
```
# Matrix (routes vs pass results) with counts
matrix_counts = eccentricity_df.pivot_table(
    index='pass_result',        # rows = routes
    columns='route_ran',    # cols = outcomes
    values='arxy',            # any column works, we just need something
non-null
    aggfunc='count',
    fill_value=0
)

print(matrix_counts)
```

```
route_ran    ANGLE  CORNER  CROSS   FLAT    GO  HITCH    IN    OUT
POST  \
pass_result
C             5811    3107  12838  20943  7041  25133  8242  18885
5158
I             1041    2726   4717   3570  9092   7239  4921   7772
3632
IN             126     272    457     99   751    614   542    379
624

route_ran    SCREEN  SLANT  WHEEL
pass_result
C              3145   9100    445
I               495   3458    482
IN                0    243     25
```

In [138]:

```
# Matrix (routes vs pass results) with counts
matrix_counts = eccentricity_df.pivot_table(
    index='pass_result',        # rows = routes
    columns='route_ran',    # cols = outcomes
    values='arxy',            # any column works, we just need something
non-null
    aggfunc='count',
    fill_value=00
)

normalized = matrix_counts.div(matrix_counts.sum(axis=1), axis=0)
normalized = matrix_counts.div(matrix_counts.sum(axis=1), axis=0) *100
print(normalized.round(2))  # keep 2 decimals
```

```
route_ran    ANGLE  CORNER  CROSS   FLAT     GO  HITCH     IN    OU
T   POST  \
pass_result
C             4.85    2.59  10.71  17.47   5.87  20.97   6.88  15.7
6   4.30
I             2.12    5.55   9.60   7.26  18.50  14.73  10.01  15.8
1   7.39
IN            3.05    6.58  11.06   2.40  18.18  14.86  13.12   9.1
7  15.10

route_ran    SCREEN  SLANT  WHEEL
pass_result
C              2.62   7.59   0.37
I              1.01   7.04   0.98
IN             0.00   5.88   0.61
```

In [139]:

```
eccentricity_df_new = eccentricity_df[eccentricity_df['player_role']=
='Targeted Receiver']
eccentricity_df_new.head()



import pandas as pd
import numpy as np

# Define custom bin edges for displacement
# bin_edges = [0, 9, 20, np.inf]  # Define your custom ranges
bin_edges = [0, 7, 15, np.inf]
bin_labels = ['Short', 'Medium', 'Long']  # Labels for the ranges

# bin_edges = [0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, np.inf]  # Define
your custom ranges
# bin_edges = [0, 7, 15, np.inf]
# bin_labels = ['0-2', '-4', '6', '8', '10', '12', '14', '16', '18', '2
0', '20+']  # Labels for the ranges

# Apply the custom bins to the 'disp' column
eccentricity_df_new['disp_binned'] = pd.cut(eccentricity_df_new['pass_l
ength'], bins=bin_edges, labels=bin_labels, right=False)

# Calculate descriptive statistics for eccentricity by the custom binned
displacement
# dis_stats = eccentricity_df.groupby('disp_binned')['eccentricity'].agg
(['mean', 'median', 'std', 'count'])

# dis_stats = eccentricity_df.groupby('disp_binned')['eccentricity'].agg
(['mean', 'median', 'std', 'count','min','max'])
dis_stats = eccentricity_df_new.groupby('disp_binned', dropna=False)['a
rxy'].agg(
    mean='mean',
    median='median',
    std='std',
    count='count',
    min='min',
    max='max',
    percentile_25=lambda x: np.percentile(x, 25),
    percentile_75=lambda x: np.percentile(x, 75),
    percentile_90=lambda x: np.percentile(x, 90)
)
```

```
print("\n Displacement stats\n")
#: Descriptive Statistics by Custom Displacement Categories:")
print(dis_stats)
```

```
 Displacement stats

                  mean     median           std  count       min
\
disp_binned
Short        340.734900   9.452515   7739.410064   6326  1.002880
Medium       498.364029  12.510411   5463.002840   3515  1.003236
Long         861.936695  10.342953  19440.524659   3179  1.003527
NaN           50.171503   3.365773    475.495017   1088  1.008000

                     max  percentile_25  percentile_75  percentile
_90
disp_binned
Short        5.745764e+05       3.451726      32.662988     197.423
372
Medium       2.247250e+05       4.277657      53.166562     530.967
692
Long         1.042796e+06       3.679735      51.442123     656.147
895
NaN          1.040710e+04       1.879340       9.956245      31.883
343


/tmp/ipykernel_39/1349767399.py:18: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pan
das-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-
copy
  eccentricity_df_new['disp_binned'] = pd.cut(eccentricity_df_new
['pass_length'], bins=bin_edges, labels=bin_labels, right=False)
/tmp/ipykernel_39/1349767399.py:24: FutureWarning: The default of o
bserved=False is deprecated and will be changed to True in a future
version of pandas. Pass observed=False to retain current behavior o
r observed=True to adopt the future default and silence this warnin
g.
  dis_stats = eccentricity_df_new.groupby('disp_binned', dropna=Fal
se)['arxy'].agg(
```

In [140]:

```
matrix_counts = eccentricity_df_new.pivot_table(
    index='disp_binned',        # rows = routes
    columns='route_ran',    # cols = outcomes
    values='arxy',            # any column works, we just need something
non-null
    aggfunc='count',
    fill_value=00
)

print(matrix_counts)
normalized = matrix_counts.div(matrix_counts.sum(axis=1), axis=0)
# normalized = matrix_counts.div(matrix_counts.sum(axis=1), axis=0) *100
print(normalized.round(2))  # keep 2 decimals
```

```
route_ran    ANGLE  CORNER  CROSS  FLAT    GO  HITCH    IN   OUT  PO
ST  SCREEN  \
disp_binned
Short          493      32    738  1224    57   1531  327  1245
24      89
Medium          28      60    322    38   212    921  500   698   2
36       3
Long             2     417    311    10  1128    204  280   251   5
09       4

route_ran    SLANT  WHEEL
disp_binned
Short          557      8
Medium         480     17
Long            11     51
route_ran    ANGLE  CORNER  CROSS  FLAT    GO  HITCH    IN   OUT  P
OST  \
disp_binned
Short         0.08    0.01   0.12  0.19  0.01   0.24  0.05  0.20
0.00
Medium        0.01    0.02   0.09  0.01  0.06   0.26  0.14  0.20
0.07
Long          0.00    0.13   0.10  0.00  0.35   0.06  0.09  0.08
0.16

route_ran    SCREEN  SLANT  WHEEL
disp_binned
Short          0.01   0.09   0.00
Medium         0.00   0.14   0.00
Long           0.00   0.00   0.02


/tmp/ipykernel_39/828960298.py:1: FutureWarning: The default value
of observed=False is deprecated and will change to observed=True in
a future version of pandas. Specify observed=False to silence this
warning and retain the current behavior
  matrix_counts = eccentricity_df_new.pivot_table(
```

In [141]:

```
# dashoard cl

matrix_counts = eccentricity_df_new.pivot_table(
    index='route_ran',       # rows = routes
    columns='disp_binned',   # cols = outcomes
    values='arxy',           # any column works, we just need something
non-null
    aggfunc='count',
    fill_value=00
)

print(matrix_counts)
normalized = matrix_counts.div(matrix_counts.sum(axis=1), axis=0)
normalized = matrix_counts.div(matrix_counts.sum(axis=1), axis=0) *100
print(normalized.round(2))  # keep 2 decimals
```

disp_binned	Short	Medium	Long
route_ran			
ANGLE	493	28	2
CORNER	32	60	417
CROSS	738	322	311
FLAT	1224	38	10
GO	57	212	1128
HITCH	1531	921	204
IN	327	500	280
OUT	1245	698	251
POST	24	236	509
SCREEN	89	3	4
SLANT	557	480	11
WHEEL	8	17	51

disp_binned	Short	Medium	Long
route_ran			
ANGLE	94.26	5.35	0.38
CORNER	6.29	11.79	81.93
CROSS	53.83	23.49	22.68
FLAT	96.23	2.99	0.79
GO	4.08	15.18	80.74
HITCH	57.64	34.68	7.68
IN	29.54	45.17	25.29
OUT	56.75	31.81	11.44
POST	3.12	30.69	66.19
SCREEN	92.71	3.12	4.17
SLANT	53.15	45.80	1.05
WHEEL	10.53	22.37	67.11

```
/tmp/ipykernel_39/1226184317.py:3: FutureWarning: The default value
of observed=False is deprecated and will change to observed=True in
a future version of pandas. Specify observed=False to silence this
warning and retain the current behavior
  matrix_counts = eccentricity_df_new.pivot_table(
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [142]:
```
eccentricity_df_ = eccentricity_df.copy()

role_map = {
    "Targeted Receiver": "TR",
    "Passer": "Passer",
    "Other Route Runner": "Other",
    "Defensive Coverage": "Other"
}



# create a new column
eccentricity_df_["role_group"] = eccentricity_df_["player_role"].map(ro
le_map)


# # map detailed roles to simplified groups
# role_map = {
#     "Targeted Receiver": "TR",
#     "Passer": "Passer",
#     "Other Route Runner": "Other",
#     "Defensive Coverage": "Defense"
# }

# # create a new column
# df["role_group"] = df["player_role"].map(role_map)

# # if you want to lump Passer + TR vs Others
# df["role_group2"] = df["player_role"].replace({
#     "Targeted Receiver": "TR",
#     "Passer": "Passer"
# }).fillna("Other")
```

In [143]:

```
route_stats = eccentricity_df_.groupby(['role_group', 'pass_result'], d
ropna=False)['arxy'].agg(
    mean='mean',
    median='median',
    std='std',
    count='count',
    min='min',
    max='max',
    percentile_25=lambda x: np.percentile(x, 25),
    percentile_75=lambda x: np.percentile(x, 75),
    percentile_90=lambda x: np.percentile(x, 90)
)

# print('Descriptive Statistics')
print(route_stats)
```

```
                                  mean     median           std   count
min  \
role_group pass_result
Other      C            323.097077  11.134649   6800.232241  100397
1.000079
           I            507.209190  11.263627  28235.864050   41082
1.000464
           IN           637.408871  10.942921  19347.567968    3458
1.001384
Passer     C            150.571190  15.259530   1807.026807    9737
1.001481
           I            885.812926  14.926947  47698.142512    4031
1.020266
           IN           100.154857  15.109401    472.645541     337
1.001703
TR         C            524.297224   9.233270  12970.079342    9738
1.002880
           I            379.279178  10.501163   3436.639208    4032
1.005501
           IN           198.407917  10.115586    684.364196     338
1.025826

                                   max  percentile_25  percentile_75
\
role_group pass_result
Other      C            1.045510e+06       3.950342      34.593752
           I            3.977766e+06       3.840303      37.220553
           IN           1.090478e+06       3.861300      38.888560
Passer     C            1.472320e+05       5.776730      30.630743
           I            3.027449e+06       5.676960      30.967730
           IN           5.541536e+03       6.287084      32.479538
TR         C            1.042796e+06       3.233780      34.812666
           I            1.412897e+05       3.694305      43.578995
           IN           7.608613e+03       3.647884      35.184993

                        percentile_90
role_group pass_result
Other      C               198.157916
           I               232.909254
           IN              263.378592
Passer     C                97.185316
           I               101.572106
           IN               70.141470
```

TR	C	282.449063
	I	444.426050
	IN	409.094813

In [ ]:

In [ ]:

In [144]:

```
route_stats = eccentricity_df.groupby(['player_role', 'pass_result'], d
ropna=False)['arxy'].agg(
    mean='mean',
    median='median',
    std='std',
    count='count',
    min='min',
    max='max',
    percentile_25=lambda x: np.percentile(x, 25),
    percentile_75=lambda x: np.percentile(x, 75),
    percentile_90=lambda x: np.percentile(x, 90)
)

# print('Descriptive Statistics')
print(route_stats)
```

```
                                          mean     median           st
d  count  \
player_role        pass_result
Defensive Coverage C            308.685990  12.360063   7100.15986
0  65285
                   I            339.417342  12.618185   8980.92273
5  26745
                   IN           384.988412  12.534791   6775.34210
0   2263
Other Route Runner C            349.892126   9.091112   6204.15110
5  35112
                   I            820.217000   8.951329  46195.33707
7  14337
                   IN          1115.423514   9.004164  31567.08693
6   1195
Passer             C            150.571190  15.259530   1807.02680
7   9737
                   I            885.812926  14.926947  47698.14251
2   4031
                   IN           100.154857  15.109401    472.64554
1    337
Targeted Receiver  C            524.297224   9.233270  12970.07934
2   9738
                   I            379.279178  10.501163   3436.63920
8   4032
                   IN           198.407917  10.115586    684.36419
6    338

                                      min           max  percentile_
25  \
player_role        pass_result
Defensive Coverage C           1.000364  1.045510e+06       4.3508
34
                   I           1.000464  1.262552e+06       4.2377
51
                   IN          1.001384  2.813866e+05       4.3454
57
Other Route Runner C           1.000079  6.739609e+05       3.3932
66
                   I           1.001064  3.977766e+06       3.3051
28
                   IN          1.001627  1.090478e+06       3.0924
96
```

```
Passer             C            1.001481  1.472320e+05       5.7767
30
                   I            1.020266  3.027449e+06       5.6769
60
                   IN           1.001703  5.541536e+03       6.2870
84
Targeted Receiver  C            1.002880  1.042796e+06       3.2337
80
                   I            1.005501  1.412897e+05       3.6943
05
                   IN           1.025826  7.608613e+03       3.6478
84
```

player_role	pass_result	percentile_75	percentile_90
Defensive Coverage	C	35.755746	183.264044
	I	38.915138	230.748450
	IN	42.420357	274.214801
Other Route Runner	C	32.171176	232.476085
	I	33.657051	237.798841
	IN	33.575127	216.838040
Passer	C	30.630743	97.185316
	I	30.967730	101.572106
	IN	32.479538	70.141470
Targeted Receiver	C	34.812666	282.449063
	I	43.578995	444.426050
	IN	35.184993	409.094813

In [145]:

```
route_stats = eccentricity_df.groupby(['player_role', 'pass_result'], d
ropna=False)['arxy'].agg(
    mean='mean',
    median='median',
    std='std',
    count='count',
    min='min',
    max='max',
    percentile_25=lambda x: np.percentile(x, 25),
    percentile_75=lambda x: np.percentile(x, 75),
    percentile_90=lambda x: np.percentile(x, 90)
)

# print('Descriptive Statistics')
print(route_stats)
```

```
                                          mean     median           st
d  count  \
player_role        pass_result
Defensive Coverage C            308.685990  12.360063   7100.15986
0  65285
                   I            339.417342  12.618185   8980.92273
5  26745
                   IN           384.988412  12.534791   6775.34210
0   2263
Other Route Runner C            349.892126   9.091112   6204.15110
5  35112
                   I            820.217000   8.951329  46195.33707
7  14337
                   IN          1115.423514   9.004164  31567.08693
6   1195
Passer             C            150.571190  15.259530   1807.02680
7   9737
                   I            885.812926  14.926947  47698.14251
2   4031
                   IN           100.154857  15.109401    472.64554
1    337
Targeted Receiver  C            524.297224   9.233270  12970.07934
2   9738
                   I            379.279178  10.501163   3436.63920
8   4032
                   IN           198.407917  10.115586    684.36419
6    338

                                      min           max  percentile_
25  \
player_role        pass_result
Defensive Coverage C           1.000364  1.045510e+06       4.3508
34
                   I           1.000464  1.262552e+06       4.2377
51
                   IN          1.001384  2.813866e+05       4.3454
57
Other Route Runner C           1.000079  6.739609e+05       3.3932
66
                   I           1.001064  3.977766e+06       3.3051
28
                   IN          1.001627  1.090478e+06       3.0924
96
```

```
Passer             C            1.001481  1.472320e+05       5.7767
30
                   I            1.020266  3.027449e+06       5.6769
60
                   IN           1.001703  5.541536e+03       6.2870
84
Targeted Receiver  C            1.002880  1.042796e+06       3.2337
80
                   I            1.005501  1.412897e+05       3.6943
05
                   IN           1.025826  7.608613e+03       3.6478
84
```

player_role	pass_result	percentile_75	percentile_90
Defensive Coverage	C	35.755746	183.264044
	I	38.915138	230.748450
	IN	42.420357	274.214801
Other Route Runner	C	32.171176	232.476085
	I	33.657051	237.798841
	IN	33.575127	216.838040
Passer	C	30.630743	97.185316
	I	30.967730	101.572106
	IN	32.479538	70.141470
Targeted Receiver	C	34.812666	282.449063
	I	43.578995	444.426050
	IN	35.184993	409.094813

In [146]:

```
route_stats = eccentricity_df.groupby(['player_role'], dropna=False)['a
rxy'].agg(
    mean='mean',
    median='median',
    std='std',
    count='count',
    min='min',
    max='max',
    percentile_25=lambda x: np.percentile(x, 25),
    percentile_75=lambda x: np.percentile(x, 75),
    percentile_90=lambda x: np.percentile(x, 90)
)

# print('Descriptive Statistics')
print(route_stats)
```

```
                          mean     median           std  count
min  \
player_role
Defensive Coverage  319.233776  12.437977   7673.439215  94293  1.0
00364
Other Route Runner  501.101701   9.058974  25579.839416  50644  1.0
00079
Passer              359.487824  15.165908  25543.073843  14105  1.0
01481
Targeted Receiver   475.044080   9.549945  10931.776833  14108  1.0
02880

                             max  percentile_25  percentile_75  per
centile_90
player_role
Defensive Coverage  1.262552e+06       4.319003      36.679162
197.802749
Other Route Runner  3.977766e+06       3.361503      32.598519
233.212088
Passer              3.027449e+06       5.761413      30.769331
97.805984
Targeted Receiver   1.042796e+06       3.370616      36.958388
317.775718
```

In [147]:

```
route_stats = eccentricity_df.groupby('pass_result', dropna=False)['ar
s'].agg(
    mean='mean',
    median='median',
    std='std',
    count='count',
    min='min',
    max='max',
    percentile_25=lambda x: np.percentile(x, 25),
    percentile_75=lambda x: np.percentile(x, 75),
    percentile_90=lambda x: np.percentile(x, 90)
)

# print('Descriptive Statistics')
print(route_stats)
```

```
                  mean    median            std   count       min
\
pass_result
C            350.181502  8.880311   16085.415216  119872  1.000070
I            843.760287  8.270095  112375.579299   49145  1.000015
IN           179.110216  7.639401    3795.743279    4133  1.003375

                     max  percentile_25  percentile_75  percentile
_90
pass_result
C            3.269475e+06       3.171188      34.467418     145.839
487
I            2.452732e+07       3.006301      31.670342     131.160
998
IN           2.319293e+05       2.835453      28.722909     117.355
068
```

In [148]:

```
route_stats = eccentricity_df.groupby(['player_role', 'pass_result'], d
ropna=False)['eccentricity'].agg(
    mean='mean',
    median='median',
    std='std',
    count='count',
    min='min',
    max='max',
    percentile_25=lambda x: np.percentile(x, 25),
    percentile_75=lambda x: np.percentile(x, 75),
    percentile_90=lambda x: np.percentile(x, 90)
)

# print('Descriptive Statistics')
print(route_stats)
```

```
                                          mean     median           st
d  count  \
player_role        pass_result
Defensive Coverage C            275.971812   5.708996   6633.52050
7  65285
                   I            299.059674   5.081979   8799.14419
4  26745
                   IN           366.789595   4.236220   6775.38701
4   2263
Other Route Runner C            328.541239   2.141941   6197.01874
0  35112
                   I            788.215428   1.897597  46183.27631
0  14337
                   IN          1094.965311   1.216551  31567.29841
8   1195
Passer             C            129.946718  10.130307   1753.33772
6   9737
                   I            873.494207   9.683661  47698.24439
6   4031
                   IN            83.999969   8.514266    460.99616
3    337
Targeted Receiver  C            508.615406   1.967509  12969.81314
9   9738
                   I            362.268537   1.783533   3436.51425
1   4032
                   IN           182.563879   1.183806    684.36015
5    338

                                      min           max  percentile_
25  \
player_role        pass_result
Defensive Coverage C           0.000002  1.045510e+06       0.3172
00
                   I           0.000005  1.262552e+06       0.2796
70
                   IN          0.000473  2.813866e+05       0.2274
72
Other Route Runner C           0.000045  6.739609e+05       0.2059
04
                   I           0.000008  3.977766e+06       0.2074
29
                   IN          0.000198  1.090478e+06       0.1733
39
```

```
Passer             C            0.000027  1.472320e+05       0.8169
57
                   I            0.000204  3.027449e+06       0.6549
23
                   IN           0.000494  5.541536e+03       0.2715
54
Targeted Receiver  C            0.000108  1.042796e+06       0.2195
68
                   I            0.000247  1.412897e+05       0.1723
12
                   IN           0.000958  7.608613e+03       0.1557
66
```

player_role	pass_result	percentile_75	percentile_90
Defensive Coverage	C	26.296520	123.319349
	I	27.979227	166.800707
	IN	25.645269	194.693076
Other Route Runner	C	18.490795	159.176113
	I	18.042937	164.584136
	IN	16.144412	144.632611
Passer	C	26.241125	50.161643
	I	26.609764	48.805225
	IN	25.739791	40.751918
Targeted Receiver	C	19.887541	216.677812
	I	23.127351	357.635559
	IN	16.959438	314.303088

In [ ]:

In [149]:

```
route_stats = eccentricity_df.groupby('pass_result', dropna=False)['ar
a'].agg(
    mean='mean',
    median='median',
    std='std',
    count='count',
    min='min',
    max='max',
    percentile_25=lambda x: np.percentile(x, 25),
    percentile_75=lambda x: np.percentile(x, 75),
    percentile_90=lambda x: np.percentile(x, 90)
)

# print('Descriptive Statistics')
print(route_stats)
```

```
                  mean    median           std   count       min
\
pass_result
C            166.056008  7.044332   4671.399174  119872  1.000026
I            164.315081  6.465835  11123.642304   49145  1.000004
IN           111.067537  6.165059   2440.629626    4133  1.002419

                    max  percentile_25  percentile_75  percentile
_90
pass_result
C           8.434914e+05       2.727122      27.273577     109.882
536
I           2.441499e+06       2.633944      24.305105      95.783
252
IN          1.541803e+05       2.507484      23.742404      89.054
074
```

In [150]:
```
route_stats = eccentricity_df.groupby('pass_result', dropna=False)['ecc
entricity'].agg(
    mean='mean',
    median='median',
    std='std',
    count='count',
    min='min',
    max='max',
    percentile_25=lambda x: np.percentile(x, 25),
    percentile_75=lambda x: np.percentile(x, 75),
    percentile_90=lambda x: np.percentile(x, 90)
)

# print('Descriptive Statistics')
print(route_stats)
```

```
                  mean    median           std   count       min
\
pass_result
C            298.407867  4.336970   7009.552720  119872  0.000002
I            494.062823  3.837596  29187.785479   49145  0.000005
IN           539.207109  3.002471  17699.435688    4133  0.000198

                     max  percentile_25  percentile_75  percentile
_90
pass_result
C           1.045510e+06       0.271745      24.217287     129.094
661
I           3.977766e+06       0.247937      25.233102     161.741
192
IN          1.090478e+06       0.199498      22.863065     171.015
043
```

player : targeted receiver

In [151]:
```
eccentricity_df_new = eccentricity_df[eccentricity_df['player_role']=
='Targeted Receiver']
eccentricity_df_new.head()
```

```
/usr/local/lib/python3.11/dist-packages/pandas/io/formats/format.p
y:1458: RuntimeWarning: invalid value encountered in greater
  has_large_values = (abs_vals > 1e6).any()
/usr/local/lib/python3.11/dist-packages/pandas/io/formats/format.p
y:1459: RuntimeWarning: invalid value encountered in less
  has_small_values = ((abs_vals < 10 ** (-self.digits)) & (abs_vals
> 0)).any()
/usr/local/lib/python3.11/dist-packages/pandas/io/formats/format.p
y:1459: RuntimeWarning: invalid value encountered in greater
  has_small_values = ((abs_vals < 10 ** (-self.digits)) & (abs_vals
> 0)).any()
```

Out[151]:

	game_id	play_id	nfl_id	week	quarter	down	arxy	ars	ara
1	2023090700	101	44930	1	1	3	1272.317907	29.623264	10.836651
9	2023090700	194	41325	1	1	3	34.658407	5.417797	2.719095
26	2023090700	219	53591	1	1	1	29.361937	52.874383	56.887368
35	2023090700	361	38696	1	1	3	27.059208	49.277923	9.176576
49	2023090700	436	53541	1	1	2	92.242089	9.714801	1.729244

In [152]:
```
eccentricity_df_new.shape
```

Out[152]:
```
(14108, 34)
```

In [153]:

```
route_stats = eccentricity_df_new.groupby(['pass_result', 'player_rol
e'], dropna=False)['arxy'].agg(
    mean='mean',
    median='median',
    std='std',
    count='count',
    min='min',
    max='max',
    percentile_25=lambda x: np.percentile(x, 25),
    percentile_75=lambda x: np.percentile(x, 75),
    percentile_90=lambda x: np.percentile(x, 90)
)

# print('Descriptive Statistics')
print(route_stats)
```

```
                                      mean     median           std
count  \
pass_result player_role
C           Targeted Receiver  524.297224   9.233270  12970.079342
9738
I           Targeted Receiver  379.279178  10.501163   3436.639208
4032
IN          Targeted Receiver  198.407917  10.115586    684.364196
338

                                      min           max  percentile_2
5  \
pass_result player_role
C           Targeted Receiver  1.002880  1.042796e+06       3.23378
0
I           Targeted Receiver  1.005501  1.412897e+05       3.69430
5
IN          Targeted Receiver  1.025826  7.608613e+03       3.64788
4

                                  percentile_75  percentile_90
pass_result player_role
C           Targeted Receiver      34.812666     282.449063
I           Targeted Receiver      43.578995     444.426050
IN          Targeted Receiver      35.184993     409.094813
```

In [154]:

```
route_stats = eccentricity_df_new.groupby(['pass_result', 'player_rol
e'], dropna=False)['eccentricity'].agg(
    mean='mean',
    median='median',
    std='std',
    count='count',
    min='min',
    max='max',
    percentile_25=lambda x: np.percentile(x, 25),
    percentile_75=lambda x: np.percentile(x, 75),
    percentile_90=lambda x: np.percentile(x, 90)
)

# print('Descriptive Statistics')
print(route_stats)
```

```
                                      mean    median           std
count  \
pass_result player_role
C           Targeted Receiver  508.615406  1.967509  12969.813149
9738
I           Targeted Receiver  362.268537  1.783533   3436.514251
4032
IN          Targeted Receiver  182.563879  1.183806    684.360155
338

                                       min           max  percentile_2
5  \
pass_result player_role
C           Targeted Receiver  0.000108  1.042796e+06       0.21956
8
I           Targeted Receiver  0.000247  1.412897e+05       0.17231
2
IN          Targeted Receiver  0.000958  7.608613e+03       0.15576
6

                               percentile_75  percentile_90
pass_result player_role
C           Targeted Receiver      19.887541     216.677812
I           Targeted Receiver      23.127351     357.635559
IN          Targeted Receiver      16.959438     314.303088
```

In [155]:

```
route_stats = eccentricity_df_new.groupby(['pass_result', 'player_rol
e'], dropna=False)['ars'].agg(
    mean='mean',
    median='median',
    std='std',
    count='count',
    min='min',
    max='max',
    percentile_25=lambda x: np.percentile(x, 25),
    percentile_75=lambda x: np.percentile(x, 75),
    percentile_90=lambda x: np.percentile(x, 90)
)

# print('Descriptive Statistics')
print(route_stats)
```

```
                                      mean    median            std
count  \
pass_result player_role
C           Targeted Receiver   294.665285  6.559281    9995.403305
9738
I           Targeted Receiver  6420.142833  5.641033  386394.070618
4032
IN          Targeted Receiver   119.017653  5.054420    1000.777017
338

                                       min           max  percentile_2
5  \
pass_result player_role
C           Targeted Receiver  1.000534  9.340066e+05       2.69699
1
I           Targeted Receiver  1.002273  2.452732e+07       2.38137
3
IN          Targeted Receiver  1.004300  1.714561e+04       2.19120
5

                              percentile_75  percentile_90
pass_result player_role
C           Targeted Receiver      24.960041     102.193583
I           Targeted Receiver      18.863677      80.470226
IN          Targeted Receiver      17.909342      78.319289
```

In [156]:

```
route_stats = eccentricity_df_new.groupby(['pass_result', 'player_rol
e'], dropna=False)['ara'].agg(
    mean='mean',
    median='median',
    std='std',
    count='count',
    min='min',
    max='max',
    percentile_25=lambda x: np.percentile(x, 25),
    percentile_75=lambda x: np.percentile(x, 75),
    percentile_90=lambda x: np.percentile(x, 90)
)

# print('Descriptive Statistics')
print(route_stats)
```

```
                                      mean    median           std
count  \
pass_result player_role
C           Targeted Receiver  142.095586  5.407767   4140.368017
9738
I           Targeted Receiver  692.179105  5.375015  38456.187983
4032
IN          Targeted Receiver   89.873641  4.723798    362.237126
338

                                       min           max  percentile_2
5  \
pass_result player_role
C           Targeted Receiver  1.000026  3.930659e+05       2.24411
4
I           Targeted Receiver  1.000585  2.441499e+06       2.29945
3
IN          Targeted Receiver  1.016114  3.434373e+03       1.97231
0

                               percentile_75  percentile_90
pass_result player_role
C           Targeted Receiver      22.398871     100.029924
I           Targeted Receiver      23.034937      90.215414
IN          Targeted Receiver      25.643214     120.392038
```

In [ ]:

In [ ]:

In [157]:

```
import pandas as pd
import numpy as np

# Define custom bin edges for displacement
# bin_edges = [0, 9, 20, np.inf]  # Define your custom ranges
bin_edges = [0, 7, 15, np.inf]
bin_labels = ['Short', 'Medium', 'Long']  # Labels for the ranges

# bin_edges = [0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, np.inf]  # Define
your custom ranges
# bin_edges = [0, 7, 15, np.inf]
# bin_labels = ['0-2', '-4', '6', '8', '10', '12', '14', '16', '18', '2
0', '20+']  # Labels for the ranges

# Apply the custom bins to the 'disp' column
eccentricity_df_new['disp_binned'] = pd.cut(eccentricity_df_new['pass_l
ength'], bins=bin_edges, labels=bin_labels, right=False)

# Calculate descriptive statistics for eccentricity by the custom binned
displacement
# dis_stats = eccentricity_df.groupby('disp_binned')['eccentricity'].agg
(['mean', 'median', 'std', 'count'])

# dis_stats = eccentricity_df.groupby('disp_binned')['eccentricity'].agg
(['mean', 'median', 'std', 'count','min','max'])
dis_stats = eccentricity_df_new.groupby('disp_binned', dropna=False)['a
rxy'].agg(
    mean='mean',
    median='median',
    std='std',
    count='count',
    min='min',
    max='max',
    percentile_25=lambda x: np.percentile(x, 25),
    percentile_75=lambda x: np.percentile(x, 75),
    percentile_90=lambda x: np.percentile(x, 90)
)

print("\n Displacement stats\n")
#: Descriptive Statistics by Custom Displacement Categories:")
print(dis_stats)
```

```
 Displacement stats

                   mean     median           std  count       min
\
disp_binned
Short        340.734900   9.452515   7739.410064   6326  1.002880
Medium       498.364029  12.510411   5463.002840   3515  1.003236
Long         861.936695  10.342953  19440.524659   3179  1.003527
NaN           50.171503   3.365773    475.495017   1088  1.008000

                      max  percentile_25  percentile_75  percentile
_90
disp_binned
Short        5.745764e+05       3.451726      32.662988     197.423
372
Medium       2.247250e+05       4.277657      53.166562     530.967
692
Long         1.042796e+06       3.679735      51.442123     656.147
895
NaN          1.040710e+04       1.879340       9.956245      31.883
343


/tmp/ipykernel_39/1433549114.py:14: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pan
das-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-
copy
  eccentricity_df_new['disp_binned'] = pd.cut(eccentricity_df_new
['pass_length'], bins=bin_edges, labels=bin_labels, right=False)
/tmp/ipykernel_39/1433549114.py:20: FutureWarning: The default of o
bserved=False is deprecated and will be changed to True in a future
version of pandas. Pass observed=False to retain current behavior o
r observed=True to adopt the future default and silence this warnin
g.
  dis_stats = eccentricity_df_new.groupby('disp_binned', dropna=Fal
se)['arxy'].agg(
```

In [ ]:

In [158]:

```
route_stats = eccentricity_df_new.groupby(['play_action'], dropna=Fals
e)['arxy'].agg(
    mean='mean',
    median='median',
    std='std',
    count='count',
    min='min',
    max='max',
    percentile_25=lambda x: np.percentile(x, 25),
    percentile_75=lambda x: np.percentile(x, 75),
    percentile_90=lambda x: np.percentile(x, 90)
)

# print('Descriptive Statistics')
print(route_stats)
```

```
                    mean     median           std  count       min
\
play_action
0.0           511.334777  10.423522  12015.484400  11087  1.002880
1.0           341.968494   6.894603   5314.380377   3020  1.003527
NaN             8.360550   8.360550           NaN      1  8.360550

                      max  percentile_25  percentile_75  percentile
_90
play_action
0.0           1.042796e+06       3.625890      39.298192     326.119
493
1.0           2.774289e+05       2.712431      28.692393     275.374
088
NaN           8.360550e+00       8.360550       8.360550       8.360
550


/usr/local/lib/python3.11/dist-packages/pandas/io/formats/format.p
y:1458: RuntimeWarning: invalid value encountered in greater
  has_large_values = (abs_vals > 1e6).any()
/usr/local/lib/python3.11/dist-packages/pandas/io/formats/format.p
y:1459: RuntimeWarning: invalid value encountered in less
  has_small_values = ((abs_vals < 10 ** (-self.digits)) & (abs_vals
> 0)).any()
/usr/local/lib/python3.11/dist-packages/pandas/io/formats/format.p
y:1459: RuntimeWarning: invalid value encountered in greater
  has_small_values = ((abs_vals < 10 ** (-self.digits)) & (abs_vals
> 0)).any()
```

In [159]:

```
route_stats = eccentricity_df_new.groupby(['dropback_type'], dropna=Fal
se)['arxy'].agg(
    mean='mean',
    median='median',
    std='std',
    count='count',
    min='min',
    max='max',
    percentile_25=lambda x: np.percentile(x, 25),
    percentile_75=lambda x: np.percentile(x, 75),
    percentile_90=lambda x: np.percentile(x, 90)
)

# print('Descriptive Statistics')
print(route_stats)
```

```
                                mean     median           std  count
min  \
dropback_type
DESIGNED_ROLLOUT_LEFT   284.544187   8.947376   1855.991031    307
1.004262
DESIGNED_ROLLOUT_RIGHT   84.584689   4.171966    602.833338    534
1.006245
SCRAMBLE                123.072859   6.078875    707.059082    681
1.003236
SCRAMBLE_ROLLOUT_LEFT    12.576720   7.161330     15.280719     15
1.261507
SCRAMBLE_ROLLOUT_RIGHT   81.477193   3.233995    578.248008     80
1.055946
TRADITIONAL             518.724643  10.297478  11612.034600  12490
1.002880
NaN                       8.360550   8.360550           NaN      1
8.360550

                                 max  percentile_25  percentile_75
\
dropback_type
DESIGNED_ROLLOUT_LEFT   3.026143e+04       2.999065      36.328047
DESIGNED_ROLLOUT_RIGHT  1.096802e+04       2.225547      10.338762
SCRAMBLE                1.490929e+04       2.582781      19.830024
SCRAMBLE_ROLLOUT_LEFT   5.256270e+01       1.954244      13.807793
SCRAMBLE_ROLLOUT_RIGHT  5.148121e+03       1.952933       7.464373
TRADITIONAL             1.042796e+06       3.564304      40.158045
NaN                     8.360550e+00       8.360550       8.360550

                        percentile_90
dropback_type
DESIGNED_ROLLOUT_LEFT      377.393513
DESIGNED_ROLLOUT_RIGHT      33.461447
SCRAMBLE                   182.322340
SCRAMBLE_ROLLOUT_LEFT       33.988609
SCRAMBLE_ROLLOUT_RIGHT      17.153753
TRADITIONAL                350.958131
NaN                          8.360550
```