NFL Draft Predictor

Introduction

Our project aims to predict the order of the NFL draft and provide valuable insights on player value. This is an important challenge, as the NFL draft is one of the most followed events in sports, with 47.5 million viewers tuning in to the 2019 draft as fans crave analysis on prospective players. Given that the NFL is a $16 billion business, optimizing drafting strategy is critical for teams, who invest heavily in scouting and analytics for the draft. Sports media also capitalizes on the draft, producing extensive coverage and mock drafts to drive viewership and engagement. By tackling the difficult task of accurately forecasting the draft order and player value, our prediction tool could offer meaningful insights to NFL teams, fans, and media alike.

Any changes since the proposal

No changes have been made to the dataset structure since the proposal.

Data

Our project involves a comprehensive set of features to predict NFL draft outcomes, covering various aspects of player performance. These features can be grouped into several categories, reflecting different roles and skills in football:

1. **Defense and Fumbles**: This category includes stats that measure a player's defensive capabilities, such as solo tackles, assisted tackles, sacks, interceptions, and fumbles. These statistics are crucial for evaluating defensive players, showcasing their ability to stop the opposing offense and create turnover opportunities.
2. **Passing**: Stats in this group evaluate the performance of quarterbacks, including completions, attempts, completion percentage, passing yards, touchdowns, interceptions, and passer rating. These statistics are key indicators of a quarterback's efficiency, accuracy, and overall ability to lead the offense.
3. **Receiving & Rushing**: This category encompasses the performance of running backs, wide receivers, and tight ends, detailing their contributions in both the passing and rushing aspects of the game. Metrics include receptions, receiving yards, yards per reception, rushing attempts, rushing yards, and touchdowns. These stats help assess a player's versatility and impact on the field.
4. **Punt & Kick Returns**: Special teams stats, such as punt and kickoff returns, including yards, yards per return, and touchdowns, evaluate a player's ability to contribute to field position and score in special teams play. This aspect is often considered for players with the ability to change the game's momentum through returns.
5. **Punting & Kicking**: This group of stats measures the performance of punters and kickers, including field goals made and attempted, extra points, punting yards, and

yards per punt. These statistics are essential for assessing a player's contribution to the team's scoring and field position through kicking and punting.

By analyzing these features, your project aims to predict NFL draft outcomes based on a holistic view of a player's performance across various aspects of the game. The data provides a detailed snapshot of each player's skills, efficiency, and impact, offering valuable insights into their potential success and draft prospects.

SCRAPER to generate the data!

-- All Members

```
import requests
from bs4 import BeautifulSoup
import pandas as pd
import glob
from sklearn.preprocessing import LabelEncoder
from fancyimpute import KNN
import math
from tqdm import tqdm
import warnings
warnings.filterwarnings("ignore")
from zenrows import ZenRowsClient

# Read API key from file
with open('zenrows_api.txt') as f:
    api_key = f.readline().strip()
    f.close()

client = ZenRowsClient(api_key)
```

Function to Retrieve Total Player Statistics from College Webpage

```
def get_total_stats(webpage, stat):
    """
    Given a player's college stats page, returns the total stats for
the player.
    """
    stat_html = webpage.select(f'td[data-stat="{stat}"]')
    if stat_html:
        return stat_html[-1].get_text()
    else:
        return None
```

Looping Over Years to Extract Player Data from NFL Draft Combine Pages and College Stats Pages

```python
# Loop over all years in the dataset.
current = False
START_YEAR = 2012
END_YEAR = 2023

STATS_LIST = [
    #Defense and Fumbles
    'tackles_solo',
    'tackles_assists',
    'tackles_total',
    'tackles_loss',
    'sacks',
    'def_int',
    'def_int_yds',
    'def_int_td',
    'pass_defended',
    'fumbles_rec',
    'fumbles_rec_yds',
    'fumbles_rec_td',
    'fumbles_forced',

    # Passing
    'pass_cmp',
    'pass_att',
    'pass_cmp_pct',
    'pass_yds',
    'pass_td',
    'pass_int',
    'pass_rating',

    # Receiving & Rushing
    'rec',
    'rec_yds',
    'rec_yds_per_rec',
    'rec_td',
    'rush_att',
    'rush_yds',
    'rush_yds_per_att',
    'rush_td',
    'scrim_att',
    'scrim_yds',
    'scrim_yds_per_att',
    'scrim_td',

    # Punt & Kick Returns
    'punt_ret',
    'punt_ret_yds',
    'punt_ret_yds_per_ret',
    'punt_ret_td',
    'kick_ret',
```

```
    'kick_ret_yds',
    'kick_ret_yds_per_ret',
    'kick_ret_td'

    # Punting & Kicking
    'xpm',
    'xpa',
    'xp_pct',
    'fgm',
    'fga',
    'fg_pct',
    'kick_points',
    'punt',
    'punt_yds',
    'punt_yds_per_punt'
    ]

# Looping over all years within a specified range to scrape data
for year in range(START_YEAR, END_YEAR + 1):
    # Initializing an empty DataFrame for each year
    df = pd.DataFrame()
    print(year)

    # Building URL for each year's draft combine data
    url = f"https://www.pro-football-reference.com/draft/{year}-
combine.htm"
    response = requests.get(url)
    webpage = BeautifulSoup(response.text, 'html.parser')

    # Extracting player names, positions, draft round, draft pick,
college, and stat URLs from the draft combine webpage
    names_html = webpage.select("tbody .left:nth-child(1)")
    all_names = [name.get_text() for name in names_html]
    names = [name for name in all_names if name != "Player"]
    num_players = len(names)

    # Get the position of the players
    pos_html = webpage.select("th+ td")
    pos = [pos.get_text() for pos in pos_html]
    pick = [0] * num_players
    round_ = [0] * num_players

    # Get draft data if this is not the current year.
    if not current:
        draft_html = webpage.select(".right+ .left")
        draft_info = [info.get_text() for info in draft_html]
        draft_info = ["Undrafted / 0th / 0th / 0" if info == "" else
info for info in draft_info]
        draft_spots = [info.split(" / ") for info in draft_info]
        round_ = [int(spot[1][0]) for spot in draft_spots]
```

```
        pick = [int(''.join(filter(str.isdigit, spot[2]))) for spot in
draft_spots]

    #Get school data
    college_elements = webpage.select('td.left + .left')
    college = [element.get_text() for element in college_elements]

    df["Name"] = names
    df["Position"] = pos
    df["College"] = college
    df["Round"] = round_
    df["Pick"] = pick

    # Get the links to the player's college stats
    stat_urls = []
    for link in webpage.select('td[data-stat="college"]'):
        if link.find('a'):
            stat_urls.append(link.find('a').get('href'))
        else:
            stat_urls.append(None)

    df["Stat URL"] = stat_urls

    # Get height
    height_html = webpage.select("td[data-stat='height']")
    height = [h.get_text() for h in height_html]
    height = [h.split("-") for h in height]
    new_height = []
    for h in height:
        if len(h) == 2:
            new_height.append((int(h[0]) * 12 + int(h[1])))
        else:
            new_height.append(math.nan)
    df["Height"] = new_height

    # Get weight
    weight_html = webpage.select("td[data-stat='weight']")
    weight = [w.get_text() for w in weight_html]
    weight = [int(w) if w != "" else math.nan for w in weight]
    df["Weight"] = weight

    # Get 40 yard dash
    forty_html = webpage.select("td[data-stat='forty_yd']")
    forty = [f.get_text() for f in forty_html]
    forty = [float(f) if f != "" else math.nan for f in forty]
    df["40 Yard Dash"] = forty

    # Get bench press
    bench_html = webpage.select("td[data-stat='bench_reps']")
    bench = [b.get_text() for b in bench_html]
```

```
    bench = [int(b) if b != "" else math.nan for b in bench]
    df["Bench Press"] = bench

    # Get vertical jump
    vertical_html = webpage.select("td[data-stat='vertical']")
    vertical = [v.get_text() for v in vertical_html]
    vertical = [float(v) if v != "" else math.nan for v in vertical]
    df["Vertical Jump"] = vertical

    # Get broad jump
    broad_html = webpage.select("td[data-stat='broad_jump']")
    broad = [b.get_text() for b in broad_html]
    broad = [int(b) if b != "" else math.nan for b in broad]
    df["Broad Jump"] = broad

    # Get 3 cone drill
    cone_html = webpage.select("td[data-stat='cone']")
    cone = [c.get_text() for c in cone_html]
    cone = [float(c) if c != "" else math.nan for c in cone]
    df["3 Cone Drill"] = cone

    # Get shuttle
    shuttle_html = webpage.select("td[data-stat='shuttle']")
    shuttle = [s.get_text() for s in shuttle_html]
    shuttle = [float(s) if s != "" else math.nan for s in shuttle]
    df["Shuttle"] = shuttle

    df.dropna(subset=["Stat URL"], inplace=True)
    df.reset_index(drop=True, inplace=True)

    urls  = df["Stat URL"]
    all_stats = {}

    for url in tqdm(urls):
        stats = {}
        response = client.get(url)
        webpage = BeautifulSoup(response.text, 'html.parser')

        # Get conference from stat page
        conf_html = webpage.select('td[data-stat="conf_abbr"]')
        if conf_html:
            conf = conf_html[0].get_text()
            stats['conf_abbr'] = conf
        else:
            stats['conf_abbr'] = None

        # Get games played and seasons played
        games_html = webpage.select('td[data-stat="g"]')

        if games_html:
            season = 0
```

```python
            games_played = 0
            for game in games_html:
                if game.get_text() != "":
                    games_played += int(game.get_text())
                    season += 1

            stats['games'] = games_played
            stats['seasons'] = season
        else:
            stats['games'] = None
            stats['seasons'] = None

        # Get total stats
        for stat in STATS_LIST:
            stats[stat] = get_total_stats(webpage, stat)

        all_stats[url] = stats

    stat_df = pd.DataFrame(all_stats).T
    stat_df.index.name = "Stat URL"
    new_df = pd.merge(df, stat_df, on="Stat URL")
    new_df["Year"] = year

    # Save the data to a CSV file
    new_df.to_csv(f"data/{year}.csv", index=False)
```

```
2021
['6', '2']
['6', '1']
['6', '3']
['']
['5', '7']
['6', '4']
['6', '5']
['6', '6']
['5', '8']
['5', '8']
['6', '3']
['6', '0']
['6', '7']
['6', '5']
['6', '1']
['6', '4']
['6', '0']
['6', '3']
['6', '0']
['6', '5']
['5', '11']
['5', '10']
['6', '3']
```

```
['6', '3']
['6', '2']
['6', '1']
['6', '5']
['5', '10']
['6', '6']
['6', '0']
['6', '4']
['6', '4']
['6', '3']
['6', '3']
['6', '5']
['6', '4']
['6', '3']
['6', '0']
['6', '0']
['6', '2']
['6', '3']
['6', '0']
['6', '6']
['6', '1']
['6', '5']
['6', '3']
['5', '9']
['5', '9']
['6', '3']
['6', '3']
['6', '0']
['5', '11']
['6', '0']
['5', '5']
['6', '6']
['6', '6']
['6', '8']
['6', '4']
['6', '1']
['6', '6']
['6', '7']
['5', '10']
['6', '2']
['6', '2']
['6', '5']
['6', '4']
['6', '7']
['6', '1']
['6', '1']
['5', '10']
['6', '4']
['6', '2']
```

```
['6', '6']
['5', '11']
['6', '0']
['6', '4']
['6', '5']
['6', '2']
['6', '3']
['5', '10']
['6', '3']
['6', '2']
['6', '3']

100%|██████████| 284/284 [00:13<00:00, 21.37it/s]
```

Data Imputation

--Aarsh Patel

```python
import pandas as pd
import numpy as np
import glob
from sklearn.preprocessing import LabelEncoder
from fancyimpute import KNN
```

Combine all years data into one csv file called "combined_data.csv"

```python
# Get a list of all csv files
csv_files = ['2012', '2013', '2014', '2015', '2016', '2017', '2018',
'2019', '2020', '2021', '2022', '2023']

# Create an empty list to store the dataframes
dfs = []

# Loop over the list of csv files
for csv in csv_files:
    # Read each csv file into a DataFrame and append it to the list
    dfs.append(pd.read_csv('data/' + csv + '.csv'))

# Concatenate all dataframes in the list into one dataframe
df = pd.concat(dfs, ignore_index=True)

df.to_csv('data/combined_data.csv', index=False)
```

Identifying and Dropping Columns with Less Than 10% Data Availability

```python
# Store original column names
original_columns = df.columns
```

```
# Drop columns with less than 10% data available
df = df.dropna(thresh=(0.1 * len(df)), axis=1)

# Get the remaining column names after dropping
remaining_columns = df.columns

# Find the dropped column names
dropped_columns = original_columns.difference(remaining_columns)

# Print the dropped column names
print(dropped_columns)
```

```
Index(['fg_pct', 'fga', 'fgm', 'kick_points', 'kick_ret',
'kick_ret_tdxpm',
       'kick_ret_yds', 'kick_ret_yds_per_ret', 'pass_att', 'pass_cmp',
       'pass_cmp_pct', 'pass_int', 'pass_rating', 'pass_td',
'pass_yds',
       'punt', 'punt_ret', 'punt_ret_td', 'punt_ret_yds',
       'punt_ret_yds_per_ret', 'punt_yds', 'punt_yds_per_punt',
'xp_pct',
       'xpa'],
      dtype='object')
```

Imputing Missing Values Using K-Nearest Neighbors (KNN) Algorithm and Label Encoding

```
# Selecting important columns from the original DataFrame
imp_df = df[['Position', 'Height', 'Weight', '40 Yard Dash', 'Bench
Press',
             'Vertical Jump', 'Broad Jump', '3 Cone Drill', 'Shuttle',
             'tackles_solo', 'tackles_assists', 'tackles_loss',
'sacks',
             'def_int', 'def_int_yds', 'def_int_td', 'pass_defended',
             'fumbles_rec', 'fumbles_rec_yds', 'fumbles_rec_td',
'fumbles_forced',
             'rec', 'rec_yds', 'rec_yds_per_rec', 'rec_td',
'rush_att', 'rush_yds',
             'rush_yds_per_att', 'rush_td', 'scrim_att', 'scrim_yds',
             'scrim_yds_per_att', 'scrim_td']]

# Initialize a label encoder for encoding categorical 'Position'
column
label_encoder = LabelEncoder()
imp_df.loc[:, 'Position'] =
label_encoder.fit_transform(imp_df['Position'])

# Impute missing values using KNN algorithm with k=5
imp_df = KNN(k=5).fit_transform(imp_df)
imp_df = pd.DataFrame(imp_df)
```

```
# Rename columns of the DataFrame
imp_df.columns = ['Position', 'Height', 'Weight', '40 Yard Dash',
'Bench Press',
             'Vertical Jump', 'Broad Jump', '3 Cone Drill', 'Shuttle',
             'tackles_solo', 'tackles_assists', 'tackles_loss',
'sacks',
             'def_int', 'def_int_yds', 'def_int_td', 'pass_defended',
             'fumbles_rec', 'fumbles_rec_yds', 'fumbles_rec_td',
'fumbles_forced',
             'rec', 'rec_yds', 'rec_yds_per_rec', 'rec_td',
'rush_att', 'rush_yds',
             'rush_yds_per_att', 'rush_td', 'scrim_att', 'scrim_yds',
             'scrim_yds_per_att', 'scrim_td']

# Round the values in the DataFrame to 2 decimal places
imp_df = imp_df.round(2)

# Replace the selected columns in the original DataFrame with the
imputed values
df[['Height', 'Weight', '40 Yard Dash', 'Bench Press', 'Vertical
Jump', 'Broad Jump',
    '3 Cone Drill', 'Shuttle', 'tackles_solo', 'tackles_assists',
'tackles_loss', 'sacks',
    'def_int', 'def_int_yds', 'def_int_td', 'pass_defended',
'fumbles_rec', 'fumbles_rec_yds',
    'fumbles_rec_td', 'fumbles_forced', 'rec', 'rec_yds',
'rec_yds_per_rec', 'rec_td', 'rush_att',
     'rush_yds', 'rush_yds_per_att', 'rush_td', 'scrim_att',
'scrim_yds','scrim_yds_per_att', 'scrim_td']] =
imp_df.drop('Position', axis=1)
```

```
Imputing row 1/3684 with 26 missing, elapsed time: 3.817
Imputing row 101/3684 with 25 missing, elapsed time: 3.933
Imputing row 201/3684 with 24 missing, elapsed time: 4.046
Imputing row 301/3684 with 20 missing, elapsed time: 4.149
Imputing row 401/3684 with 17 missing, elapsed time: 4.218
Imputing row 501/3684 with 13 missing, elapsed time: 4.286
Imputing row 601/3684 with 16 missing, elapsed time: 4.347
Imputing row 701/3684 with 14 missing, elapsed time: 4.412
Imputing row 801/3684 with 19 missing, elapsed time: 4.483
Imputing row 901/3684 with 12 missing, elapsed time: 4.549
Imputing row 1001/3684 with 12 missing, elapsed time: 4.616
Imputing row 1101/3684 with 15 missing, elapsed time: 4.684
Imputing row 1201/3684 with 14 missing, elapsed time: 4.752
Imputing row 1301/3684 with 15 missing, elapsed time: 4.818
Imputing row 1401/3684 with 12 missing, elapsed time: 4.881
Imputing row 1501/3684 with 20 missing, elapsed time: 4.947
Imputing row 1601/3684 with 14 missing, elapsed time: 5.018
Imputing row 1701/3684 with 25 missing, elapsed time: 5.084
Imputing row 1801/3684 with 18 missing, elapsed time: 5.156
```

```
Imputing row 1901/3684 with 15 missing, elapsed time: 5.226
Imputing row 2001/3684 with 16 missing, elapsed time: 5.300
Imputing row 2101/3684 with 14 missing, elapsed time: 5.370
Imputing row 2201/3684 with 16 missing, elapsed time: 5.441
Imputing row 2301/3684 with 15 missing, elapsed time: 5.508
Imputing row 2401/3684 with 19 missing, elapsed time: 5.580
Imputing row 2501/3684 with 12 missing, elapsed time: 5.650
Imputing row 2601/3684 with 15 missing, elapsed time: 5.720
Imputing row 2701/3684 with 12 missing, elapsed time: 5.787
Imputing row 2801/3684 with 12 missing, elapsed time: 5.853
Imputing row 2901/3684 with 19 missing, elapsed time: 5.923
Imputing row 3001/3684 with 24 missing, elapsed time: 5.989
Imputing row 3101/3684 with 24 missing, elapsed time: 6.059
Imputing row 3201/3684 with 30 missing, elapsed time: 6.135
Imputing row 3301/3684 with 17 missing, elapsed time: 6.212
Imputing row 3401/3684 with 18 missing, elapsed time: 6.291
Imputing row 3501/3684 with 14 missing, elapsed time: 6.365
Imputing row 3601/3684 with 30 missing, elapsed time: 6.438
```

Imputing Missing Games and Seasons Values Using KNN Algorithm

```
# Select the columns 'Position', 'games', and 'seasons' from the
original DataFrame
imp_df = df[["Position", "games", "seasons"]]

# Encode the 'Position' column using a label encoder
imp_df.loc[:, 'Position'] =
label_encoder.fit_transform(imp_df["Position"])

# Impute missing values for 'games' and 'seasons' columns using KNN
algorithm with k=10
imp_df=fancyimpute.KNN(k=10).fit_transform(imp_df)
imp_df = pd.DataFrame(imp_df)

# Round the values in the DataFrame to the nearest integer
imp_df = imp_df.round(0)

# Replace the missing values in the original DataFrame for 'Games' and
'Seasons' with the imputed values
df[["Games", "Seasons"]] = imp_df.drop(0, axis=1)

Imputing row 1/3684 with 2 missing, elapsed time: 1.832
Imputing row 101/3684 with 2 missing, elapsed time: 1.840
Imputing row 201/3684 with 2 missing, elapsed time: 1.848
Imputing row 301/3684 with 0 missing, elapsed time: 1.854
Imputing row 401/3684 with 0 missing, elapsed time: 1.854
Imputing row 501/3684 with 0 missing, elapsed time: 1.855
Imputing row 601/3684 with 0 missing, elapsed time: 1.855
Imputing row 701/3684 with 0 missing, elapsed time: 1.856
Imputing row 801/3684 with 0 missing, elapsed time: 1.857
```

```
Imputing row 901/3684 with 0 missing, elapsed time: 1.858
Imputing row 1001/3684 with 0 missing, elapsed time: 1.859
Imputing row 1101/3684 with 0 missing, elapsed time: 1.860
Imputing row 1201/3684 with 0 missing, elapsed time: 1.861
Imputing row 1301/3684 with 0 missing, elapsed time: 1.861
Imputing row 1401/3684 with 0 missing, elapsed time: 1.862
Imputing row 1501/3684 with 0 missing, elapsed time: 1.863
Imputing row 1601/3684 with 0 missing, elapsed time: 1.864
Imputing row 1701/3684 with 0 missing, elapsed time: 1.865
Imputing row 1801/3684 with 0 missing, elapsed time: 1.865
Imputing row 1901/3684 with 0 missing, elapsed time: 1.866
Imputing row 2001/3684 with 0 missing, elapsed time: 1.867
Imputing row 2101/3684 with 0 missing, elapsed time: 1.868
Imputing row 2201/3684 with 0 missing, elapsed time: 1.869
Imputing row 2301/3684 with 0 missing, elapsed time: 1.870
Imputing row 2401/3684 with 0 missing, elapsed time: 1.871
Imputing row 2501/3684 with 0 missing, elapsed time: 1.871
Imputing row 2601/3684 with 0 missing, elapsed time: 1.872
Imputing row 2701/3684 with 0 missing, elapsed time: 1.873
Imputing row 2801/3684 with 0 missing, elapsed time: 1.874
Imputing row 2901/3684 with 0 missing, elapsed time: 1.875
Imputing row 3001/3684 with 2 missing, elapsed time: 1.876
Imputing row 3101/3684 with 2 missing, elapsed time: 1.877
Imputing row 3201/3684 with 0 missing, elapsed time: 1.878
Imputing row 3301/3684 with 0 missing, elapsed time: 1.879
Imputing row 3401/3684 with 0 missing, elapsed time: 1.881
Imputing row 3501/3684 with 0 missing, elapsed time: 1.882
Imputing row 3601/3684 with 2 missing, elapsed time: 1.883
```

Calculating Total Tackles and Exporting Imputed Data to CSV

```
df['tackles_total'] = df['tackles_solo'] + df['tackles_assists']
df['tackles_total'] = df['tackles_total'].round(0)

# Export the imputed data to a CSV file
df.to_csv('data/imputed_data.csv', index=False)
```

Exploratory Data Analysis for the NFL Cut!

Dataset after Imputation: imputed_data.csv

```
import pandas as pd

file_path = './data/imputed_data.csv'
data = pd.read_csv(file_path)
data.head()
```

```
                Name Position         College  Round  Pick  \
0    Emmanuel Acho      OLB           Texas      6   204
1        Joe Adams       WR        Arkansas      4   104
2     Chas Alecxih       DT      Pittsburgh      0     0
3  Frank Alexander       DE        Oklahoma      4   103
4    Antonio Allen        S  South Carolina      7   242

                                            Stat URL  Height
Weight  \
0  https://www.sports-reference.com/cfb/players/e...    74.0   238.0

1  https://www.sports-reference.com/cfb/players/j...    71.0   179.0

2  https://www.sports-reference.com/cfb/players/c...    76.0   296.0

3  https://www.sports-reference.com/cfb/players/f...    76.0   270.0

4  https://www.sports-reference.com/cfb/players/a...    73.0   210.0


   40 Yard Dash  Bench Press  ...  rec_td  rush_att  rush_yds  \
0          4.64        24.00  ...    5.29    199.20   1282.58
1          4.51        14.59  ...    8.50      4.00     69.50
2          5.31        19.00  ...    0.00      1.19      5.20
3          4.80        24.48  ...    2.17     22.98     75.37
4          4.58        17.00  ...    1.68    374.69   2061.25

   rush_yds_per_att rush_td  scrim_att  scrim_yds
scrim_yds_per_att  \
0                8.83   14.91     239.71    1747.91                 8.22

1               11.65    0.00      96.00    1393.50                14.45

2               -0.68    0.36       1.36       5.55                 0.86

3                4.12    4.24      36.81     231.59                 6.49

4                4.94   19.21     420.39    2397.36                 6.43


   scrim_td  Year
0     20.20  2012
1      8.50  2012
2      0.36  2012
3      6.41  2012
4     20.89  2012

[5 rows x 43 columns]
```

Dataset Features

```
data.columns
```

```
Index(['Name', 'Position', 'College', 'Round', 'Pick', 'Stat URL',
'Height',
       'Weight', '40 Yard Dash', 'Bench Press', 'Vertical Jump',
'Broad Jump',
       '3 Cone Drill', 'Shuttle', 'conf_abbr', 'games', 'seasons',
       'tackles_solo', 'tackles_assists', 'tackles_total',
'tackles_loss',
       'sacks', 'def_int', 'def_int_yds', 'def_int_td',
'pass_defended',
       'fumbles_rec', 'fumbles_rec_yds', 'fumbles_rec_td',
'fumbles_forced',
       'rec', 'rec_yds', 'rec_yds_per_rec', 'rec_td', 'rush_att',
'rush_yds',
       'rush_yds_per_att', 'rush_td', 'scrim_att', 'scrim_yds',
       'scrim_yds_per_att', 'scrim_td', 'Year'],
      dtype='object')
```

Number of College Players Per Position

```
data_1= data.loc[:, ['Position']].value_counts().reset_index()
data_1
```

```
   Position  count
0        WR    538
1        CB    379
2        RB    342
3         S    237
4        DT    216
5        TE    210
6        DE    193
7        OT    193
8        QB    184
9        LB    176
10      OLB    148
11       OL    134
12       OG    120
13       DL    116
14      ILB     96
15     EDGE     86
16       DB     81
17        C     69
18        P     67
19        K     55
20       FB     25
21       LS     19
```

Number of College Players per College

```
data_2= data.loc[:, ['College']].value_counts().reset_index()
data_2
```

```
                     College  count
0                    Alabama    127
1                        LSU    107
2                    Georgia    101
3                    Florida     94
4                 Notre Dame     80
..                       ...    ...
158        North Dakota St.       1
159      Alabama-Birmingham       1
160        Northern Arizona       1
161          Ala-Birmingham       1
162   Northwestern St. (LA)       1

[163 rows x 2 columns]
```

Number of College Players per Position:

```
players_per_position = data.loc[:,
['Position']].value_counts().reset_index()
players_per_position
```

```
   Position  count
0        WR    538
1        CB    379
2        RB    342
3         S    237
4        DT    216
5        TE    210
6        DE    193
7        OT    193
8        QB    184
9        LB    176
10      OLB    148
11       OL    134
12       OG    120
13       DL    116
14      ILB     96
15     EDGE     86
16       DB     81
17        C     69
18        P     67
19        K     55
20       FB     25
21       LS     19
```

Visualizations

- Niketan

Visualization 1: Analyzing the Impact of Physical Metrics on Draft Rounds

Box plots showing the distribution of height and weight across different draft rounds

• **Visual 1:** Box plots showing the distribution of height and weight across different draft rounds.

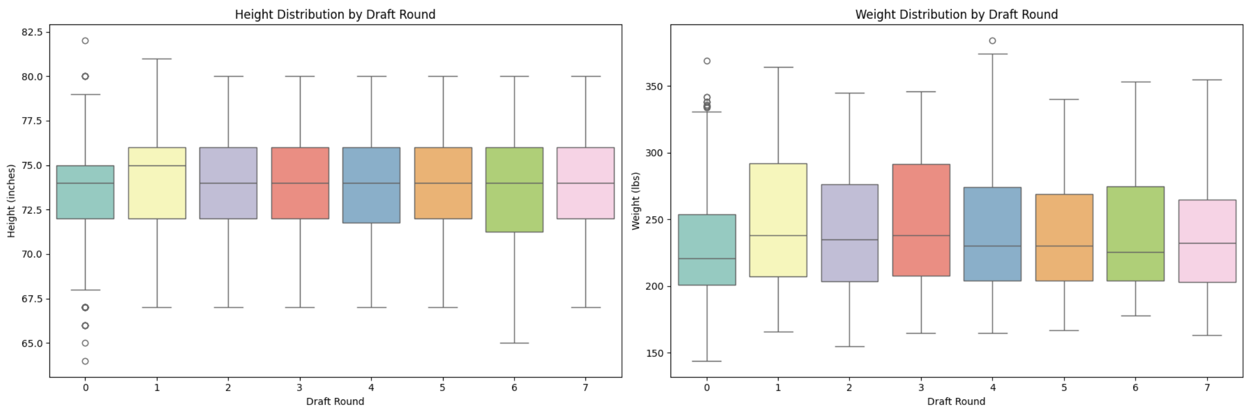
• **Hypothesis:** Players with stronger physical metrics (e.g., taller heights, heavier weights) are more likely to be drafted in earlier rounds.

• **Observation:** The data shows that height and weight do not seem to be the main factors in determining which draft round a player is selected. They are fairly consistent across the different draft rounds, with some outliers but no clear trend. Teams might consider other factors over just the player's physical size when making draft decisions.

```
import matplotlib.pyplot as plt
import seaborn as sns

fig, axes = plt.subplots(1, 2, figsize=(18, 6))


sns.boxplot(ax=axes[0], x='Round', y='Height',
data=data,palette='Set3')
axes[0].set_title('Height Distribution by Draft Round')
axes[0].set_xlabel('Draft Round')
axes[0].set_ylabel('Height (inches)')


sns.boxplot(ax=axes[1], x='Round', y='Weight',
data=data,palette='Set3')
axes[1].set_title('Weight Distribution by Draft Round')
axes[1].set_xlabel('Draft Round')
axes[1].set_ylabel('Weight (lbs)')

plt.tight_layout()
plt.show()
```

```
/var/folders/zp/8ykdjlwd1qjdfz83znprmqp00000gn/T/
ipykernel_12660/1454464748.py:7: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.boxplot(ax=axes[0], x='Round', y='Height',
data=data,palette='Set3')
/var/folders/zp/8ykdjlwd1qjdfz83znprmqp00000gn/T/ipykernel_12660/14544
64748.py:13: FutureWarning:
```

```
Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.boxplot(ax=axes[1], x='Round', y='Weight',
data=data,palette='Set3')
```



Visualization 2: Historical Trends in Player Physical Metrics

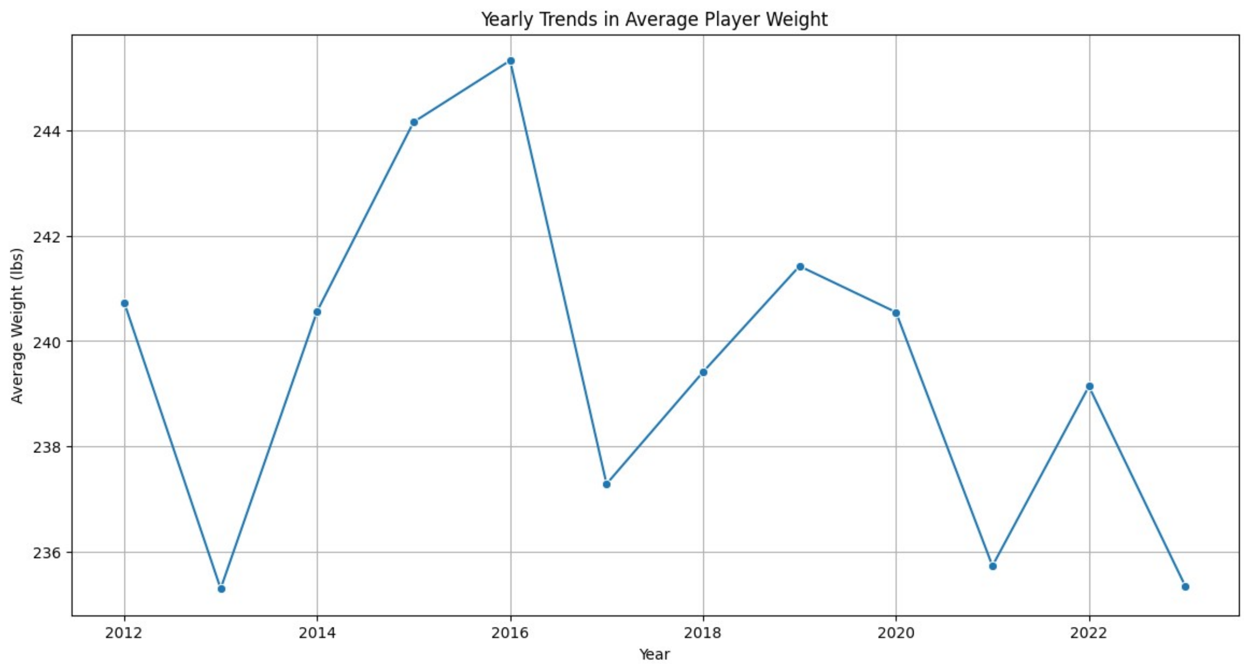
Line graphs depicting the evolution of average player heights and weights over the years.

• **Visual 2:** Line graphs showing the evolution of average player heights and weights over the years.

• **Hypothesis:** Over the years, there has been an increase in the average size and weight of drafted players.

• **Observation:** The line graph shows NFL player weight goes up and down over time. This data doesn't support the idea that players just keep getting bigger. But there was a peak in 2016 and a trend for 3 years (2013-16) where overall weight was increasing.

```
import matplotlib.pyplot as plt
import seaborn as sns


average_weight_per_year = data.groupby('Year')
['Weight'].mean().reset_index()


plt.figure(figsize=(14, 7))
sns.lineplot(x='Year', y='Weight', data=average_weight_per_year,
marker='o')
plt.title('Yearly Trends in Average Player Weight')
plt.xlabel('Year')
plt.ylabel('Average Weight (lbs)')
plt.grid(True)
plt.show()
```



Visualizations

- Nishant

Visualisation 3: College Performance and Its Correlation with Draft Success

Scatter plots comparing college performance metrics (e.g., rushing yards, receiving touchdowns) with draft rounds

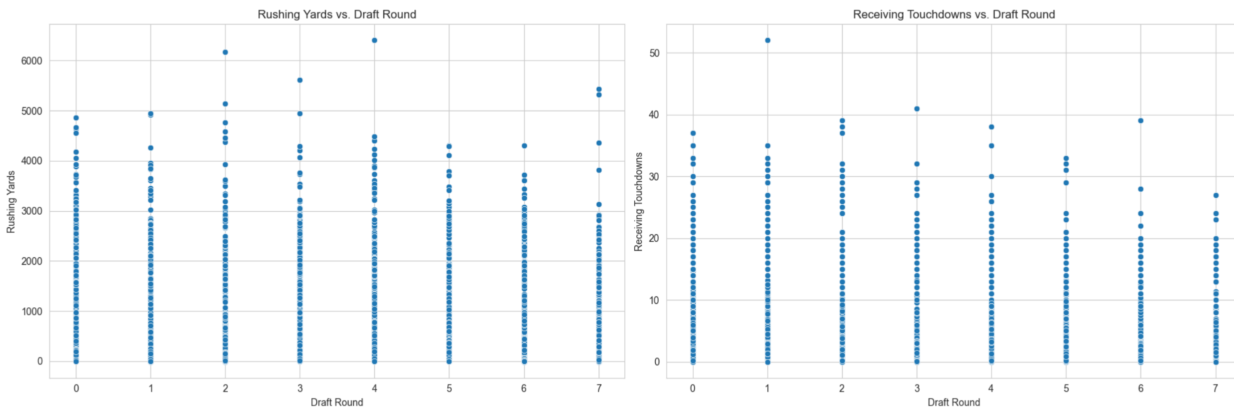
• **Visual 3:** Scatter plots comparing college performance metrics (e.g., rushing yards, receiving touchdowns) with draft rounds.

• **Hypothesis:** Outstanding college performance is positively correlated with being drafted in higher rounds.

• **Observation:** Early results suggest strong college performance in rushing yards and touchdowns might be linked to higher draft picks, highlighting the importance of college achievements. However, the data also shows players with good stats are spread across all draft rounds, suggesting that other factors might also impact during the draft.

```
import matplotlib.pyplot as plt
import seaborn as sns


sns.set_style("whitegrid")


fig, axes = plt.subplots(1, 2, figsize=(18, 6))
```

```
sns.scatterplot(ax=axes[0], x='Round', y='rush_yds', data=data)
axes[0].set_title('Rushing Yards vs. Draft Round')
axes[0].set_xlabel('Draft Round')
axes[0].set_ylabel('Rushing Yards')


sns.scatterplot(ax=axes[1], x='Round', y='rec_td', data=data)
axes[1].set_title('Receiving Touchdowns vs. Draft Round')
axes[1].set_xlabel('Draft Round')
axes[1].set_ylabel('Receiving Touchdowns')

plt.tight_layout()
plt.show()
```



Visualization 4: Impact of Combine Performance on Draft Outcomes

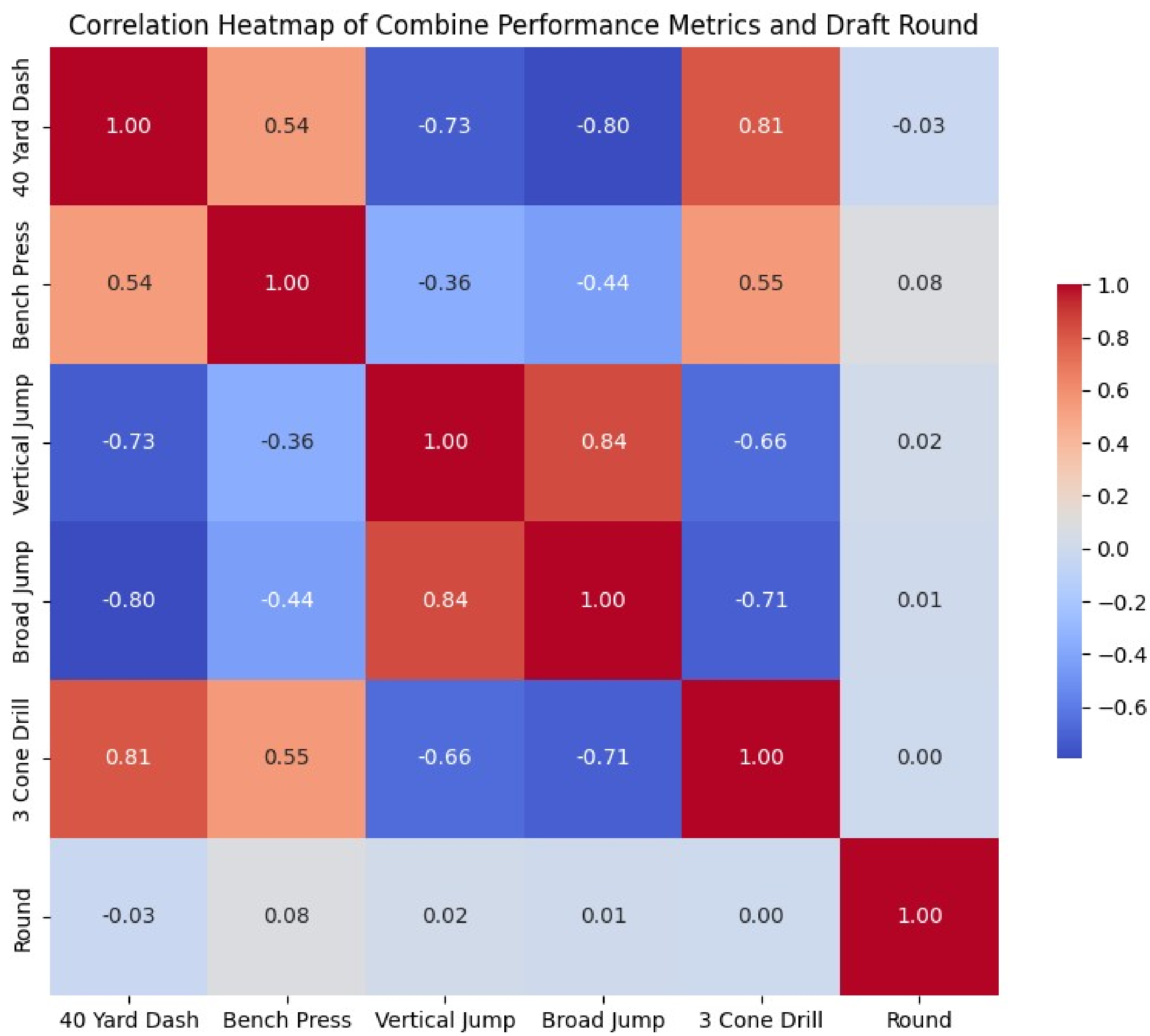
• **Visual 4:** Correlation heatmaps between combine performance metrics and draft rounds.

• **Hypothesis:** Good performance in combine drills correlates with higher draft selections.

• **Observation:** NFL combine results (speed, jumps) do show some link to earlier draft picks, suggesting the combine is still important. But the connection isn't strong, so teams likely consider other factors like game film and interviews too.

```
import matplotlib.pyplot as plt
import seaborn as sns

combine_metrics = ['40 Yard Dash', 'Bench Press', 'Vertical Jump',
'Broad Jump', '3 Cone Drill']  # example metric columns

combine_metrics.append('Round')

# Calculating the correlation matrix
corr = data[combine_metrics].corr()

# Setting up the matplotlib figure
plt.figure(figsize=(10, 8))
```

```python
sns.heatmap(corr, annot=True, fmt=".2f", cmap='coolwarm',
cbar_kws={'shrink': .5}, square=True)
plt.title('Correlation Heatmap of Combine Performance Metrics and
Draft Round')
plt.show()
```



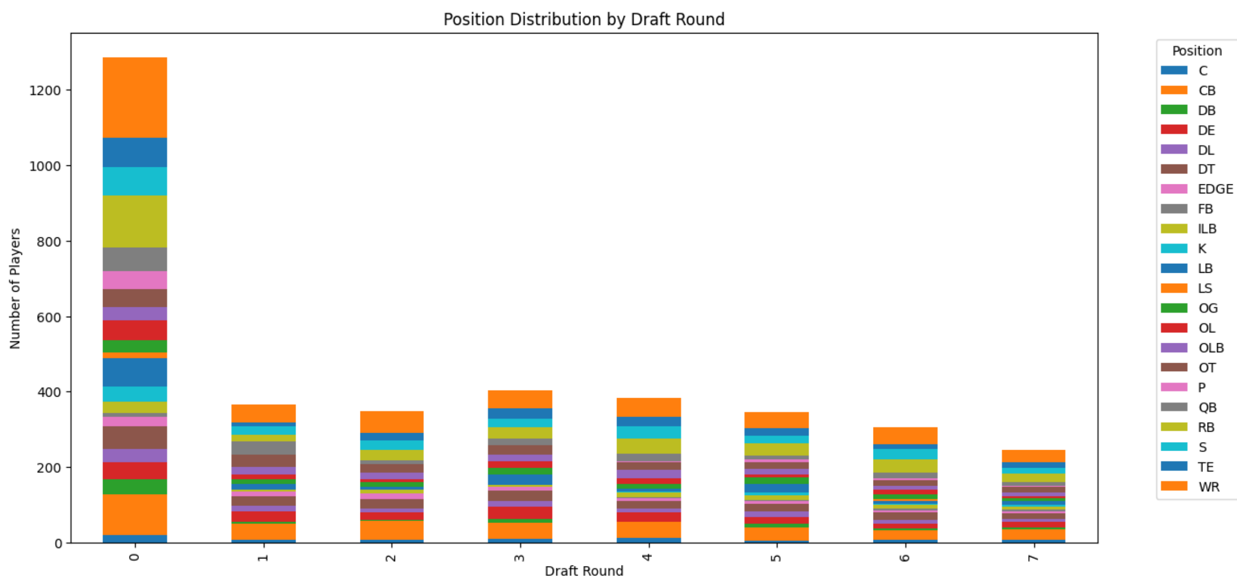
Visualization 5: Positional Value in Draft Rounds

• **Visual 5:** Stacked bar charts showing the distribution of drafted positions in each round.

• **Hypothesis:** Some positions (e.g., Quarterbacks, Offensive Tackles) are more likely to be drafted in earlier rounds due to their importance in the game.

• **Observation:** Most players go undrafted (Round 0). Some positions, like tackle and receiver, are usually drafted early because teams think they're valuable. Kickers and punters are drafted less often, and other positions can be drafted any round. This suggests teams consider more than just position when drafting.

```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

position_round_counts = data.groupby(['Round',
'Position']).size().unstack().fillna(0)

position_round_counts.plot(kind='bar', stacked=True, figsize=(14, 7))
plt.title('Position Distribution by Draft Round')
plt.xlabel('Draft Round')
plt.ylabel('Number of Players')
plt.legend(title='Position', bbox_to_anchor=(1.05, 1), loc='upper
left')
plt.show()
```



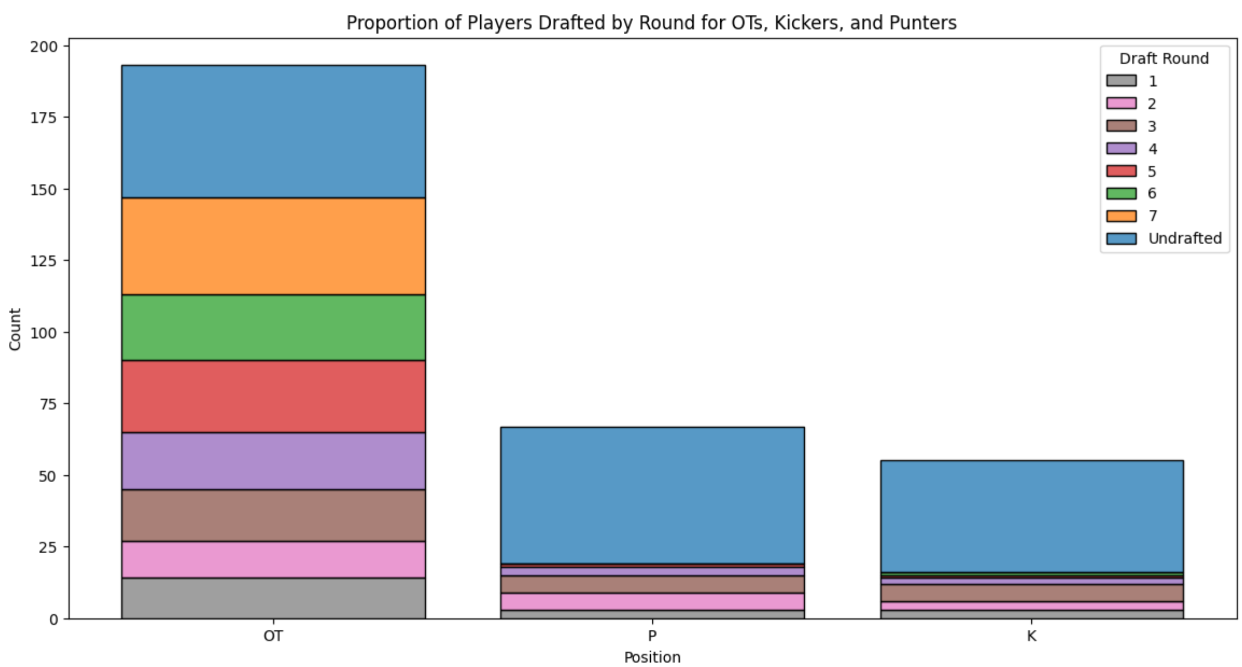
Visualization 6: Proportion of Players Drafted by Round for Selected Positions (OT vs. Kickers/Punters)

• **Visual 6:** The bar chart shows the count of players drafted by round for three distinct positions: Offensive Tackles (OT), Punters (P), and Kickers (K), along with those who were undrafted. We chose to explore these 3 positions more since we had some insights on these from in the EDA in Visual 5.

• **Hypothesis:** Players in key offensive and defensive positions, like Offensive Tackles, are likely to be drafted in earlier rounds, while specialized positions such as Kickers and Punters are often selected in later rounds.

• **Observation:** Offensive Tackles (OT) are heavily represented in the early rounds, with a significant number of players selected in the first round. Punters (P) and Kickers (K) show a contrasting distribution, with a substantial proportion undrafted, and the majority of those drafted are selected in the later rounds. The hypothesis is supported by the data, with OTs indeed being drafted earlier and more frequently, while Kickers and Punters are less prioritized in the draft process and are more likely to be undrafted.

```
data['Drafted'] = data['Round'].apply(lambda x: 'Undrafted' if x == 0
else 'Drafted')


positions_to_compare = ['OT', 'K', 'P']
draft_rounds_data = data[data['Position'].isin(positions_to_compare) &
(data['Round'] <= 7)]

plt.figure(figsize=(14, 7))
sns.histplot(data=draft_rounds_data, x='Position', hue='Round',
multiple='stack', palette='tab10', shrink=0.8)
plt.title('Proportion of Players Drafted by Round for OTs, Kickers,
and Punters')
plt.xlabel('Position')
plt.ylabel('Count')
plt.legend(title='Draft Round', loc='upper right', labels=['1', '2',
'3', '4', '5', '6', '7', 'Undrafted'])
plt.show()
```




```
# This is formatted as code
```

Rank prediction using Random Forest Classifier

-- Vishwa Sheth

Key components of the model

Data Preprocessing: After imputing missing values using KNN, we convert categorical data to numeric using the get_dummies function in pandas. This conversion helps to format the data in a way that is suitable for the model.

Feature selection: We remove irrelevant features such as "Name" and "College" from consideration for prediction. Additionally, "Round" and "Pick" are excluded as they are part of the target feature.

Target feature: Currently, we aim to predict ranking using the "Round" feature. In the future, we plan to incorporate "Pick" before final submission.

Dataset split: Given that this is a ranking problem, the training dataset includes all years except for 2023. Data from 2023 will be used solely for predicting the rank.

The hyperparameters are tuned using cross-validation. The disparity between baseline measurements and best-fit measurements demonstrates an improvement in accuracy and other metrics following 5-Fold cross-validation.

Comparative Analysis of Baseline and Best-Fit Random Forest Models for Ranking Prediction

Note: This work is in progress; we aim to improve measurement parameters, include ranking parameters in Cross Validation instead of accuracy and include "Pick" in the target feature.

```python
import pandas as pd

# Read the CSV file
df = pd.read_csv("data/imputed_data.csv")
print(df.columns)
```

```
Index(['Name', 'Position', 'College', 'Round', 'Pick', 'Stat URL',
'Height',
       'Weight', '40 Yard Dash', 'Bench Press', 'Vertical Jump',
'Broad Jump',
       '3 Cone Drill', 'Shuttle', 'conf_abbr', 'games', 'seasons',
       'tackles_solo', 'tackles_assists', 'tackles_total',
'tackles_loss',
       'sacks', 'def_int', 'def_int_yds', 'def_int_td',
'pass_defended',
       'fumbles_rec', 'fumbles_rec_yds', 'fumbles_rec_td',
'fumbles_forced',
       'rec', 'rec_yds', 'rec_yds_per_rec', 'rec_td', 'rush_att',
'rush_yds',
       'rush_yds_per_att', 'rush_td', 'scrim_att', 'scrim_yds',
       'scrim_yds_per_att', 'scrim_td', 'Year'],
      dtype='object')
```

```python
df.head
```

```
<bound method NDFrame.head of                    Name Position
College  Round  Pick  \
```

```
0       Emmanuel Acho      OLB             Texas      6   204
1           Joe Adams       WR          Arkansas      4   104
2         Chas Alecxih       DT        Pittsburgh      0     0
3     Frank Alexander       DE          Oklahoma      4   103
4       Antonio Allen        S   South Carolina      7   242
...               ...      ...              ...    ...   ...
3679      Luke Wypler        C          Ohio St.      6   190
3680      Bryce Young       QB           Alabama      1     1
3681      Byron Young       DT           Alabama      3    70
3682      Byron Young     EDGE         Tennessee      3    77
3683    Cameron Young       DT  Mississippi St.      4   123

                                               Stat URL  Height
Weight  \
0     https://www.sports-reference.com/cfb/players/e...    74.0
238.0
1     https://www.sports-reference.com/cfb/players/j...    71.0
179.0
2     https://www.sports-reference.com/cfb/players/c...    76.0
296.0
3     https://www.sports-reference.com/cfb/players/f...    76.0
270.0
4     https://www.sports-reference.com/cfb/players/a...    73.0
210.0
...                                                 ...     ...      ..
.
3679  https://www.sports-reference.com/cfb/players/l...    75.0
303.0
3680  https://www.sports-reference.com/cfb/players/b...    70.0
204.0
3681  https://www.sports-reference.com/cfb/players/b...    75.0
294.0
3682  https://www.sports-reference.com/cfb/players/b...    74.0
250.0
3683  https://www.sports-reference.com/cfb/players/c...    75.0
304.0

      40 Yard Dash  Bench Press  ...  rec_td  rush_att  rush_yds  \
0             4.64        24.00  ...    5.29    199.20   1282.58
1             4.51        14.59  ...    8.50      4.00     69.50
2             5.31        19.00  ...    0.00      1.19      5.20
3             4.80        24.48  ...    2.17     22.98     75.37
4             4.58        17.00  ...    1.68    374.69   2061.25
...            ...          ...  ...     ...       ...       ...
3679          5.14        25.26  ...    0.14      1.14     -1.96
3680          4.65        18.86  ...    0.00      1.00      1.02
3681          4.92        24.00  ...    0.13      1.53      6.99
3682          4.43        22.00  ...    1.42     67.49    319.31
3683          5.10        29.53  ...    0.21      1.08     -0.84
```

```
      rush_yds_per_att rush_td  scrim_att  scrim_yds
scrim_yds_per_att  \
0                 8.83   14.91     239.71    1747.91
8.22
1                11.65    0.00      96.00    1393.50
14.45
2                -0.68    0.36       1.36       5.55
0.86
3                 4.12    4.24      36.81     231.59
6.49
4                 4.94   19.21     420.39    2397.36
6.43
...                ...     ...        ...        ...                  ..
.
3679             -2.52    0.14       2.41       5.95                 -
2.27
3680              1.00    0.00       1.00       1.02
1.00
3681              1.28    0.53       2.67      14.08
1.51
3682              3.01    4.51      95.10     567.86
5.38
3683             -2.46    0.21       5.61      32.58                 -
0.22

      scrim_td  Year
0        20.20  2012
1         8.50  2012
2         0.36  2012
3         6.41  2012
4        20.89  2012
...        ...   ...
3679      0.28  2023
3680      0.00  2023
3681      0.65  2023
3682      5.94  2023
3683      0.42  2023

[3684 rows x 43 columns]>

# Classifing it in binary to train model and then predict
probabilities for ranking
df.loc[df.Round != 1, "Round"] = 0

# Dropping the columns which donot contribute in prediction
all_X = df.drop(["Name", "Round", "Pick", "College"], axis=1)
all_X = pd.get_dummies(all_X)

# Splitting testing and training sets
train_X = all_X[(all_X.Year != 2023)].drop(["Year"], axis=1)
```

```
test_X = all_X[all_X.Year == 2023].drop(["Year"], axis=1)
train_y = df[(df.Year != 2023)].Round
test_y = df[df.Year == 2023].Round


train_X.head()
```

```
   Height  Weight  40 Yard Dash  Bench Press  Vertical Jump  Broad
Jump  \
0    74.0   238.0          4.64        24.00          35.50
118.00
1    71.0   179.0          4.51        14.59          36.00
123.00
2    76.0   296.0          5.31        19.00          25.50
99.00
3    76.0   270.0          4.80        24.48          31.13
115.26
4    73.0   210.0          4.58        17.00          34.00
118.00

   3 Cone Drill  Shuttle  games  seasons  ...  conf_abbr_CUSA
conf_abbr_Ind  \
0          7.13     4.28   37.0      3.0  ...           False
False
1          7.09     4.12   40.0      4.0  ...           False
False
2          7.74     4.62   34.0      3.0  ...           False
False
3          7.19     4.48   37.0      4.0  ...           False
False
4          7.02     4.25   42.0      4.0  ...           False
False

   conf_abbr_MAC  conf_abbr_MVC  conf_abbr_MWC  conf_abbr_Pac-10  \
0          False          False          False             False
1          False          False          False             False
2          False          False          False             False
3          False          False          False             False
4          False          False          False             False

   conf_abbr_Pac-12  conf_abbr_SEC  conf_abbr_Sun Belt  conf_abbr_WAC

0             False          False               False          False

1             False          False               False          False

2             False          False               False          False

3             False          False               False          False

4             False          False               False          False
```

```
[5 rows x 3754 columns]
```

```
test_X.head()
```

```
      Height  Weight  40 Yard Dash  Bench Press  Vertical Jump  Broad
Jump  \
3400    70.0   216.0          4.51        19.42          33.64
115.58
3401    73.0   237.0          4.47        17.09          36.50
129.00
3402    69.0   188.0          4.32        14.92          33.00
119.26
3403    71.0   173.0          4.49        15.14          34.00
122.00
3404    74.0   282.0          4.49        27.00          37.50
125.00

      3 Cone Drill  Shuttle  games  seasons  ...  conf_abbr_CUSA  \
3400          7.03     4.28   31.0      3.0  ...           False
3401          7.22     4.25   53.0      5.0  ...           False
3402          7.02     4.19   30.0      3.0  ...           False
3403          7.00     4.16   35.0      3.0  ...           False
3404          7.22     4.47   36.0      4.0  ...           False

      conf_abbr_Ind  conf_abbr_MAC  conf_abbr_MVC  conf_abbr_MWC  \
3400          False          False          False          False
3401          False          False          False          False
3402          False          False          False          False
3403          False          False          False          False
3404          False          False          False          False

      conf_abbr_Pac-10  conf_abbr_Pac-12  conf_abbr_SEC  conf_abbr_Sun
Belt  \
3400             False             False          False
False
3401             False             False          False
False
3402             False             False           True
False
3403             False             False          False
False
3404             False             False          False
False

      conf_abbr_WAC
3400          False
3401          False
3402          False
```

```
3403          False
3404          False

[5 rows x 3754 columns]
```

```python
from sklearn.ensemble import RandomForestClassifier
# Define the parameter values as baseline
n_estimators = 1
max_depth = None
min_samples_split = 1000
min_samples_leaf = 1000
max_features = None
bootstrap = False


# Initialize the Random Forest classifier with custom parameters
baseline_rf = RandomForestClassifier(n_estimators=n_estimators,
                                    max_depth=max_depth,

min_samples_split=min_samples_split,
                                    min_samples_leaf=min_samples_leaf,
                                    max_features=max_features,
                                    bootstrap=bootstrap)

# Initialize and train Random Forest classifier as baseline
# baseline_rf = RandomForestClassifier()
baseline_rf.fit(train_X, train_y)
```

```
RandomForestClassifier(bootstrap=False, max_features=None,
                       min_samples_leaf=1000, min_samples_split=1000,
                       n_estimators=1)
```

```python
# Make predictions on test data
baseline_preds = preds = baseline_rf.predict_proba(test_X)
count = 1

# Ranking done according to the probability scores
for i in pd.DataFrame(baseline_preds).sort_values(by=1,
ascending=False).index:
    print(str(count) + " " + str(df[df.Year==2023].reset_index().at[i,
"Name"]))
    count += 1
```

```
1 Israel Abanikanda
2 Mike Morris
3 Tashawn Manning
4 Michael Mayer
5 Warren McClendon
6 Jordan McFadden
7 Tanner McKee
8 Kendre Miller
```

```
9 Marvin Mims
10 Keaton Mitchell
11 Wanya Morris
12 Calijah Kancey
13 Myles Murphy
14 Lukas Van Ness
15 John Ojukwu
16 BJ Ojulari
17 Jarrett Patterson
18 Kyle Patterson
19 Jack Podlesny
20 Asim Richards
21 Jaxson Kirkland
22 Darrell Luter Jr.
23 Anton Harrison
24 Clark Phillips III
25 Malik Heath
26 Nick Herbig
27 Ronnie Hickman
28 Brandon Hill
29 Xavier Hutchinson
30 Jalin Hyatt
31 Andre Carter II
32 Rashad Torrence II
33 Thomas Incoom
34 Paris Johnson Jr.
35 Rakim Jarrett
36 Antonio Johnson
37 Quentin Johnston
38 Broderick Jones
39 Dawand Jones
40 Jaylon Jones
41 Will Anderson Jr.
42 Emil Ekiyor Jr.
43 Anthony Richardson
44 Eli Ricks
45 Kelee Ringo
46 Parker Washington
47 DJ Turner
48 Carrington Valentine
49 Deuce Vaughn
50 Andrew Vorhees
51 Dalton Wagner
52 Alex Ward
53 Carter Warren
54 Darnell Washington
55 Tyrus Wheat
56 Tavius Robinson
57 Blake Whiteheart
```

```
58 Dontayvion Wicks
59 Garrett Williams
60 Darnell Wright
61 Rejzohn Wright
62 Luke Wypler
63 Bryce Young
64 Byron Young
65 Tuli Tuipulotu
66 Sean Tucker
67 O'Cyrus Torrence
68 Henry To'oTo'o
69 Bijan Robinson
70 Jaquelin Roy
71 Drew Sanders
72 John Michael Schmitz
73 Luke Schoonmaker
74 Tyler Scott
75 Juice Scruggs
76 Noah Sewell
77 Trenton Simpson
78 Peter Skoronski
79 Mazi Smith
80 Jaxon Smith-Njigba
81 Tyler Steen
82 Ricky Stromberg
83 C.J. Stroud
84 Mitchell Tinsley
85 Joe Tippmann
86 Ryan Hayes
87 Dalton Kincaid
88 Matthew Bergeron
89 Tiyon Evans
90 Ali Gaye
91 Malaesala Aumavae-Laulu
92 Ji'Ayir Brown
93 Connor Galvin
94 Henry Bainivalu
95 Jon Gaines
96 Gervon Dexter
97 Blake Freeland
98 Myles Brooks
99 Felix Anudike-Uzomah
100 Alex Forsyth
101 Zach Evans
102 Jalen Brooks
103 Nick Broeker
104 Emmanuel Forbes
105 Bryan Bresee
106 Nathaniel Dell
```

```
107 Jakorian Bennett
108 Jerrod Clark
109 Jahmyr Gibbs
110 Brian Branch
111 Jovaughn Gwyn
112 Grant DuBose
113 Josh Downs
114 Jalen Carter
115 Christian Gonzalez
116 Jordan Addison
117 Anthony Bradford
118 Kayshon Boutte
119 MJ Anderson
120 YaYa Diaby
121 Devon Achane
122 Jake Andrews
123 Tank Bigsby
124 Richard Gouraige
125 Alan Ali
126 Tyjae Spears
127 JL Skinner
128 Lance Boykin
129 Terell Smith
130 Christopher Smith
131 Cam Smith
132 Nolan Smith
133 Chase Brown
134 Julius Brents
135 Brad Robbins
136 Deneric Prince
137 Jalen Redmond
138 Jayden Reed
139 Rashee Rice
140 Zach Charbonnet
141 Anders Carlson
142 Jack Campbell
143 Arquon Bush
144 Justin Shorter
145 Sydney Brown
146 Jammie Robinson
147 Darius Rush
148 Chad Ryland
149 Dante Stills
150 Cameron Brown
151 Daniel Scott
152 Tyrique Stevenson
153 Jaren Hall
154 Brenton Strange
155 Bryce Baringer
```

```
156 Deonte Banks
157 Jay Ward
158 Habakkuk Baldonado
159 Alex Austin
160 Jalen Wayne
161 Keion White
162 Josh Whyle
163 Dorian Williams
164 Brayden Willis
165 Michael Wilson
166 Tyree Wilson
167 Dee Winters
168 Devon Witherspoon
169 Colby Wooden
170 Davis Allen
171 Adetomiwa Adebawore
172 Byron Young
173 Jeremy Banks
174 Travis Vokolek
175 Jake Bobo
176 Micah Baskerville
177 Mekhi Blackmon
178 Leonard Taylor
179 Noah Taylor
180 Charlie Thomas
181 Tavion Thomas
182 SaRodorick Thompson
183 Dorian Thompson-Robinson
184 Cedric Tillman
185 Keeanu Benton
186 Stetson Bennett
187 Cory Trice
188 Tre Tucker
189 Ronnie Bell
190 Clayton Tune
191 Michael Turk
192 Robert Beal
193 Jordan Battle
194 B.T. Potter
195 Gervarrius Owens
196 Zacch Pickens
197 Payne Durham
198 Jaray Jenkins
199 Anthony Johnson
200 Roschon Johnson
201 Isaiah Foskey
202 Cam Jones
203 Charlie Jones
204 Bryce Ford-Wheaton
```

```
205 Nic Jones
206 Tyreque Jones
207 Brandon Joseph
208 Anthony Johnson Jr.
209 Zay Flowers
210 Earl Bostick Jr.
211 Dontay Demus Jr.
212 Ikenna Enechukwu
213 Ivan Pace Jr.
214 Joey Porter Jr.
215 Travis Dye
216 Kyu Blu Kelly
217 Kearis Jackson
218 Siaki Ika
219 Mekhi Garner
220 Eric Gray
221 Derick Hall
222 Zach Harrison
223 Jadon Haselwood
224 Jake Haener
225 DeMarcco Hellams
226 Daiyan Henley
227 Antoine Green
228 Shaka Heyward
229 Elijah Higgins
230 Trey Dean III
231 Jalen Graham
232 Tre'Vius Hodges-Tomlinson
233 Hendon Hooker
234 Dylan Horton
235 Jordan Howden
236 Evan Hull
237 Mohamed Ibrahim
238 Jason Taylor II
239 Yasir Abdullah
240 Malik Knowles
241 Lonnie Phelps
242 Adam Korsak
243 DJ Dale
244 Luke Musgrave
245 PJ Mustipher
246 Puka Nacua
247 Malik Cunningham
248 Joseph Ngata
249 Aidan O'Connell
250 Moro Ojomo
251 Brenton Cox
252 Jacob Copeland
253 Anfernee Orji
```

```
254 DeMarvion Overshown
255 Nick Hampton
256 Trey Palmer
257 Owen Pappoe
258 Chamarri Conner
259 Keondre Coburn
260 Camerun Peoples
261 A.T. Perry
262 Riley Moss
263 Derius Davis
264 Isaiah Moore
265 Ochaun Mathis
266 Tyler Lacy
267 Matt Landers
268 Sam LaPorta
269 Cameron Latu
270 Will Levis
271 Will Mallory
272 Christopher Dunn
273 Jartavius Martin
274 Max Duggan
275 Jake Moody
276 Isaiah McGuire
277 Kenny McIntosh
278 Demario Douglas
279 Kaevon Merriweather
280 Ventrell Miller
281 Jonathan Mingo
282 Cameron Mitchell
283 SirVocea Dennis
284 Cameron Young
```

```python
from sklearn.metrics import accuracy_score, roc_auc_score
import numpy as np

# Convert predicted probabilities to binary predictions based on a
threshold (e.g., 0.5)
predicted_labels = (baseline_preds[:, 1] > 0.5).astype(int)

# Evaluation for ranking metrics
# Sort the predictions based on probability scores
sorted_indices = np.argsort(-preds[:, 1])
k = 10
num_relevant = sum(test_y)

def calculate_MRR(sorted_indices, test_y):
    # Calculate Mean Reciprocal Rank (MRR)
    mrr = 0
    for idx, i in enumerate(sorted_indices):
        if test_y.iloc[i] == 1:  # Use iloc to access test_y by index
```

```
            mrr = 1 / (idx + 1)
            break
    return mrr

def calculate_MAP(sorted_indices, test_y):
    # Calculate Mean Average Precision (MAP)
    ap = 0
    for idx, i in enumerate(sorted_indices):
        if test_y.iloc[i] == 1:
            ap += sum(test_y.iloc[:idx + 1]) / (idx + 1)
    map_score = ap / num_relevant
    return map_score

def calculate_NDCG(sorted_indices, test_y):
    # Calculate Normalized Discounted Cumulative Gain (NDCG) at k=10
    dcg = 0
    idcg = sum(1 / np.log2(np.arange(2, k + 2)))
    for idx, i in enumerate(sorted_indices[:k]):
        if test_y.iloc[i] == 1:
            dcg += 1 / np.log2(idx + 2)
    ndcg = dcg / idcg
    return ndcg

def calculate_PAK(sorted_indices, test_y):
    # Calculate Precision at k (P@k)
    tp_at_k = sum(test_y.iloc[sorted_indices[:k]])
    precision_at_k = tp_at_k / k
    return precision_at_k

def calculate_RAK(sorted_indices, test_y):
    # Calculate Recall at k (R@k)
    tp_at_k = sum(test_y.iloc[sorted_indices[:k]])
    recall_at_k = tp_at_k / num_relevant
    return recall_at_k

pip install tabulate

Requirement already satisfied: tabulate in
/opt/homebrew/anaconda3/lib/python3.11/site-packages (0.8.10)
Note: you may need to restart the kernel to use updated packages.

from tabulate import tabulate

# Calculate all measurements
baseline_measurements = [
    ("Accuracy", accuracy_score(test_y, predicted_labels)),
    ("ROC AUC Score", roc_auc_score(test_y, baseline_preds[:, 1])),
    ("Mean Reciprocal Rank (MRR)", calculate_MRR(sorted_indices,
test_y)),
    ("Mean Average Precision (MAP)", calculate_MAP(sorted_indices,
```

```
test_y)),
    ("Normalized Discounted Cumulative Gain (NDCG) at k=10",
calculate_NDCG(sorted_indices, test_y)),
    ("Precision at k (P@k) at k=10", calculate_PAK(sorted_indices,
test_y)),
    ("Recall at k (R@k) at k=10", calculate_RAK(sorted_indices,
test_y))
]

# Print measurements in a table format
print("Baseline measurements")
print(tabulate(baseline_measurements, headers=["Metric", "Value"]))
```

```
Baseline measurements
Metric                                                    Value
-----------------------------------------------------  ---------
Accuracy                                               0.897887
ROC AUC Score                                          0.696552
Mean Reciprocal Rank (MRR)                             0.125
Mean Average Precision (MAP)                           0.115021
Normalized Discounted Cumulative Gain (NDCG) at k=10  0.0694312
Precision at k (P@k) at k=10                           0.1
Recall at k (R@k) at k=10                              0.0344828
```

```
from sklearn.model_selection import GridSearchCV
# Training the model using Random Forest by using best parameters

param_grid = {
    'n_estimators': [100, 500, 1000]
}

# Initialize the Random Forest classifier
rf = RandomForestClassifier()

# Hypertuning parameters using 5-Fold Cross Validation method
clf = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5,
scoring='accuracy')
clf.fit(train_X, train_y)
```

```
GridSearchCV(cv=5, estimator=RandomForestClassifier(),
             param_grid={'n_estimators': [100, 500, 1000]},
scoring='accuracy')
```

```
# Get the best parameters
best_params = clf.best_params_
print("Best Parameters:", best_params)

# Use the best estimator to make predictions
best_rf = clf.best_estimator_
```

```
Best Parameters: {'n_estimators': 100}
```

```python
# Predicting the probabilities of Test set
preds = best_rf.predict_proba(test_X)
count = 1

# Ranking done according to the probability scores
for i in pd.DataFrame(preds).sort_values(by=1, ascending=False).index:
    print(str(count) + " " + str(df[df.Year==2023].reset_index().at[i,
"Name"]))
    count += 1
```

```
1 Bryce Young
2 C.J. Stroud
3 Jakorian Bennett
4 Dante Stills
5 Will Anderson Jr.
6 Emmanuel Forbes
7 Marvin Mims
8 Christian Gonzalez
9 Julius Brents
10 Tyler Steen
11 Darnell Wright
12 DJ Turner
13 Anthony Richardson
14 Deonte Banks
15 Nolan Smith
16 Blake Freeland
17 Lukas Van Ness
18 Jaren Hall
19 Tre'Vius Hodges-Tomlinson
20 Thomas Incoom
21 Owen Pappoe
22 Byron Young
23 Darnell Washington
24 Joe Tippmann
25 Kelee Ringo
26 Myles Brooks
27 Isaiah Foskey
28 Trenton Simpson
29 Charlie Thomas
30 Adetomiwa Adebawore
31 Dawand Jones
32 Riley Moss
33 Richard Gouraige
34 Ryan Hayes
35 Hendon Hooker
36 Carrington Valentine
37 Darius Rush
38 Tavius Robinson
39 Michael Mayer
40 Kayshon Boutte
```

```
41 Jason Taylor II
42 Zacch Pickens
43 Anthony Bradford
44 Yasir Abdullah
45 Rashee Rice
46 Jalin Hyatt
47 Rejzohn Wright
48 Nick Hampton
49 Joey Porter Jr.
50 Matt Landers
51 Quentin Johnston
52 Tanner McKee
53 Parker Washington
54 Anton Harrison
55 Andre Carter II
56 Josh Downs
57 Nick Herbig
58 Anfernee Orji
59 Wanya Morris
60 O'Cyrus Torrence
61 Jartavius Martin
62 Cam Smith
63 Jaxon Smith-Njigba
64 Bijan Robinson
65 A.T. Perry
66 Jay Ward
67 Daniel Scott
68 Malaesala Aumavae-Laulu
69 Xavier Hutchinson
70 Robert Beal
71 Jalen Redmond
72 YaYa Diaby
73 Mike Morris
74 Warren McClendon
75 Keaton Mitchell
76 Malik Cunningham
77 Cory Trice
78 Keion White
79 Earl Bostick Jr.
80 Sydney Brown
81 Jordan Battle
82 Matthew Bergeron
83 Mohamed Ibrahim
84 Tyler Lacy
85 Jerrod Clark
86 Jonathan Mingo
87 Eli Ricks
88 Tyreque Jones
89 Jordan Howden
```

```
90 Luke Schoonmaker
91 Jake Moody
92 Paris Johnson Jr.
93 Cameron Young
94 Jacob Copeland
95 Chamarri Conner
96 Calijah Kancey
97 Drew Sanders
98 Jaquelin Roy
99 Clayton Tune
100 Jack Campbell
101 Ikenna Enechukwu
102 Travis Dye
103 Bryan Bresee
104 Darrell Luter Jr.
105 Kyu Blu Kelly
106 Jon Gaines
107 Gervon Dexter
108 Dontayvion Wicks
109 Sam LaPorta
110 Cameron Brown
111 Moro Ojomo
112 Aidan O'Connell
113 Isaiah McGuire
114 Dorian Thompson-Robinson
115 Rakim Jarrett
116 Anthony Johnson Jr.
117 Carter Warren
118 Byron Young
119 Derick Hall
120 Kearis Jackson
121 Nathaniel Dell
122 Tyree Wilson
123 Jalen Brooks
124 Trey Palmer
125 Gervarrius Owens
126 MJ Anderson
127 Will Mallory
128 Tyler Scott
129 Brandon Hill
130 John Ojukwu
131 Zach Harrison
132 PJ Mustipher
133 Tyjae Spears
134 Myles Murphy
135 Ji'Ayir Brown
136 Lonnie Phelps
137 Jaxson Kirkland
138 Jahmyr Gibbs
```

```
139 Deuce Vaughn
140 Ali Gaye
141 Felix Anudike-Uzomah
142 Ricky Stromberg
143 Jaylon Jones
144 Clark Phillips III
145 Jadon Haselwood
146 Jeremy Banks
147 Shaka Heyward
148 Chase Brown
149 Anthony Johnson
150 Cedric Tillman
151 Garrett Williams
152 Jalen Carter
153 Anders Carlson
154 Mazi Smith
155 Rashad Torrence II
156 BJ Ojulari
157 Brian Branch
158 Tashawn Manning
159 Keeanu Benton
160 Mekhi Blackmon
161 Deneric Prince
162 Connor Galvin
163 Siaki Ika
164 Jovaughn Gwyn
165 Tyrique Stevenson
166 Tre Tucker
167 Arquon Bush
168 DeMarvion Overshown
169 Jake Bobo
170 John Michael Schmitz
171 Malik Heath
172 Isaiah Moore
173 Ivan Pace Jr.
174 Daiyan Henley
175 Tyrus Wheat
176 Antonio Johnson
177 Bryce Ford-Wheaton
178 Zay Flowers
179 Zach Evans
180 Tuli Tuipulotu
181 Tiyon Evans
182 Cameron Mitchell
183 Max Duggan
184 Trey Dean III
185 DeMarcco Hellams
186 Keondre Coburn
187 Mekhi Garner
```

```
188 Bryce Baringer
189 Broderick Jones
190 Puka Nacua
191 Habakkuk Baldonado
192 Alex Forsyth
193 Mitchell Tinsley
194 Luke Wypler
195 Luke Musgrave
196 Adam Korsak
197 Noah Sewell
198 Alex Ward
199 Jake Haener
200 Jordan Addison
201 Terell Smith
202 Devon Achane
203 Ronnie Bell
204 Will Levis
205 Christopher Dunn
206 Grant DuBose
207 Malik Knowles
208 Demario Douglas
209 Jordan McFadden
210 Brenton Cox
211 DJ Dale
212 Lance Boykin
213 Brandon Joseph
214 Kyle Patterson
215 Henry To'oTo'o
216 Camerun Peoples
217 Dorian Williams
218 Tavion Thomas
219 Colby Wooden
220 Dalton Wagner
221 SaRodorick Thompson
222 Noah Taylor
223 Brenton Strange
224 Dee Winters
225 Leonard Taylor
226 Israel Abanikanda
227 Christopher Smith
228 B.T. Potter
229 Peter Skoronski
230 Elijah Higgins
231 Emil Ekiyor Jr.
232 Antoine Green
233 Payne Durham
234 Kaevon Merriweather
235 Jarrett Patterson
236 Zach Charbonnet
```

```
237 Nick Broeker
238 Asim Richards
239 Tank Bigsby
240 Justin Shorter
241 Alan Ali
242 Roschon Johnson
243 Michael Wilson
244 Cam Jones
245 SirVocea Dennis
246 Josh Whyle
247 Stetson Bennett
248 Alex Austin
249 Ronnie Hickman
250 Davis Allen
251 Andrew Vorhees
252 Jaray Jenkins
253 Dalton Kincaid
254 Michael Turk
255 Nic Jones
256 JL Skinner
257 Juice Scruggs
258 Ochaun Mathis
259 Jammie Robinson
260 Jayden Reed
261 Jack Podlesny
262 Kendre Miller
263 Charlie Jones
264 Brayden Willis
265 Jake Andrews
266 Henry Bainivalu
267 Micah Baskerville
268 Devon Witherspoon
269 Brad Robbins
270 Ventrell Miller
271 Derius Davis
272 Kenny McIntosh
273 Jalen Graham
274 Jalen Wayne
275 Eric Gray
276 Sean Tucker
277 Dylan Horton
278 Evan Hull
279 Joseph Ngata
280 Blake Whiteheart
281 Cameron Latu
282 Chad Ryland
283 Dontay Demus Jr.
284 Travis Vokolek
```

```
from sklearn.metrics import accuracy_score, roc_auc_score
import numpy as np

# Convert predicted probabilities to binary predictions based on a
threshold (e.g., 0.5)
predicted_labels = (preds[:, 1] > 0.5).astype(int)

# Evaluation for ranking metrics
# Sort the predictions based on probability scores
sorted_indices = np.argsort(-preds[:, 1])

# Calculate all measurements
best_rf_measurements = [
    ("Accuracy", accuracy_score(test_y, predicted_labels)),
    ("ROC AUC Score", roc_auc_score(test_y, baseline_preds[:, 1])),
    ("Mean Reciprocal Rank (MRR)", calculate_MRR(sorted_indices,
test_y)),
    ("Mean Average Precision (MAP)", calculate_MAP(sorted_indices,
test_y)),
    ("Normalized Discounted Cumulative Gain (NDCG) at k=10",
calculate_NDCG(sorted_indices, test_y)),
    ("Precision at k (P@k) at k=10", calculate_PAK(sorted_indices,
test_y)),
    ("Recall at k (R@k) at k=10", calculate_RAK(sorted_indices,
test_y))
]

# Print measurements in a table format
print("Best Fit Random Forest measurements")
print(tabulate(best_rf_measurements, headers=["Metric", "Value"]))
```

```
Best Fit Random Forest measurements
Metric                                                   Value
----------------------------------------------------  --------
Accuracy                                              0.897887
ROC AUC Score                                         0.696552
Mean Reciprocal Rank (MRR)                            1
Mean Average Precision (MAP)                          0.111314
Normalized Discounted Cumulative Gain (NDCG) at k=10  0.591927
Precision at k (P@k) at k=10                          0.5
Recall at k (R@k) at k=10                             0.172414
```

Comparative Analysis of Baseline and Best-Fit Random Forest Models for Ranking Prediction

The comparison between the baseline and best-fit Random Forest models reveals notable differences in performance across various metrics. In terms of accuracy and ROC AUC score, both models exhibit similar results. However, significant improvements are observed in the best-fit model for ranking-related metrics. The Mean Reciprocal Rank (MRR) shows a substantial increase, indicating that the best-fit model provides more relevant and accurate predictions at the top of the ranked list compared to the baseline. Similarly, the Mean Average Precision (MAP) and Precision at k (P@k) at k=10 metrics demonstrate considerable enhancements, implying better precision in predicting relevant instances within the top results. Moreover, the Normalized Discounted Cumulative Gain (NDCG) at k=10 reflects a notable improvement, suggesting that the best-fit model produces more relevant results at the top ranks, which is crucial for ranking tasks. Despite these improvements, the recall at k (R@k) at k=10 remains relatively low for both models, indicating a challenge in capturing all relevant instances within the top k results.

Overall, while the baseline model provides reasonable predictive performance, the best-fit Random Forest model significantly enhances the model's ability to accurately rank and prioritize instances, particularly at the top of the list, thereby improving its utility in predicting NFL Draft.

Rank prediction using XGBoost Classifier

-- Bhavya Batta

Key components of the model

Data Preprocessing: Missing values are filled in using K-nearest neighbors, and categorical data is transformed into numeric form through pandas' get_dummies method. This transformation ensures the data is properly formatted for model input.

Feature selection: Features deemed irrelevant, including "Name" and "College," are omitted from the prediction process. Furthermore, "Round" and "Pick" are not considered as they contribute to the target feature.

Target feature: Our current goal is to predict rankings based on the "Round" feature. We intend to include "Pick" as part of the target variable in the upcoming final version.

Dataset split: Given that this is a ranking problem, the training dataset includes all years except for 2023. Data from 2023 will be used solely for predicting the rank.

The **hyperparameters** are tuned using cross-validation. The disparity between baseline measurements and best-fit measurements demonstrates an improvement in accuracy and other metrics following 5-Fold cross-validation.

Note: This project is ongoing, with objectives to enhance measurement criteria, replace accuracy with ranking metrics in Cross-Validation, and incorporate "Pick" into the target feature for improvement.

```
import xgboost as xgb
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score
import numpy as np
```

```
# Read the CSV file
df = pd.read_csv("data/imputed_data.csv")
print(df.columns)
```

```
Index(['Name', 'Position', 'College', 'Round', 'Pick', 'Stat URL',
'Height',
       'Weight', '40 Yard Dash', 'Bench Press', 'Vertical Jump',
'Broad Jump',
       '3 Cone Drill', 'Shuttle', 'conf_abbr', 'games', 'seasons',
       'tackles_solo', 'tackles_assists', 'tackles_total',
'tackles_loss',
       'sacks', 'def_int', 'def_int_yds', 'def_int_td',
'pass_defended',
       'fumbles_rec', 'fumbles_rec_yds', 'fumbles_rec_td',
'fumbles_forced',
       'rec', 'rec_yds', 'rec_yds_per_rec', 'rec_td', 'rush_att',
'rush_yds',
       'rush_yds_per_att', 'rush_td', 'scrim_att', 'scrim_yds',
       'scrim_yds_per_att', 'scrim_td', 'Year'],
      dtype='object')
```

```
df.head()
```

{"type":"dataframe","variable_name":"df"}

```
df.loc[df.Round != 1, "Round"] = 0
```

```
# Dropping the columns which donot contribute in prediction
all_X = df.drop(["Name", "Round", "Pick", "College"], axis=1)
all_X = pd.get_dummies(all_X)
```

```
# Splitting testing and training sets
train_X = all_X[(all_X.Year != 2023)].drop(["Year"], axis=1)
test_X = all_X[all_X.Year == 2023].drop(["Year"], axis=1)
train_y = df[(df.Year != 2023)].Round
test_y = df[df.Year == 2023].Round
```

```
train_X.head()
```

{"type":"dataframe","variable_name":"train_X"}

```
test_X.head()
```

{"type":"dataframe","variable_name":"test_X"}

```
# Initialize the baseline XGBoost classifier with custom parameters
baseline_XGB = xgb.XGBClassifier(colsample_bytree=0.7,
 eta= 0.001,
 eval_metric= 'mae',
 max_depth= 6,
 min_child_weight= 15,
```

```
 objective= 'binary:logistic',
 subsample= 0.7)

baseline_XGB.fit(train_X, train_y)

XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=0.7, device=None,
early_stopping_rounds=None,
              enable_categorical=False, eta=0.001, eval_metric='mae',
              feature_types=None, gamma=None, grow_policy=None,
              importance_type=None, interaction_constraints=None,
              learning_rate=None, max_bin=None,
max_cat_threshold=None,
              max_cat_to_onehot=None, max_delta_step=None,
max_depth=6,
              max_leaves=None, min_child_weight=15, missing=nan,
              monotone_constraints=None, multi_strategy=None,
n_estimators=None,
              n_jobs=None, num_parallel_tree=None, ...)

# Predict on the testing data
XGBbaseline_pred = baseline_XGB.predict(test_X)

# Calculate accuracy
accuracy = accuracy_score(test_y, XGBbaseline_pred)
print("Accuracy:", accuracy)

Accuracy: 0.897887323943662

XGBbaseline_preds = baseline_XGB.predict_proba(test_X)
count = 1

# Ranking done according to the probability scores
for i in pd.DataFrame(XGBbaseline_preds).sort_values(by=1,
ascending=False).index:
    print(str(count) + " " + str(df[df.Year==2023].reset_index().at[i,
"Name"]))
    count += 1

1 Christian Gonzalez
2 Marvin Mims
3 Jakorian Bennett
4 Jalin Hyatt
5 DJ Turner
6 Anthony Richardson
7 Emmanuel Forbes
8 Byron Young
9 Keaton Mitchell
10 Kelee Ringo
11 Brandon Hill
```

```
12 Devon Achane
13 Trenton Simpson
14 Jahmyr Gibbs
15 Tyler Scott
16 Carrington Valentine
17 Dawand Jones
18 Quentin Johnston
19 Tyler Steen
20 Wanya Morris
21 Tavius Robinson
22 Eli Ricks
23 Lukas Van Ness
24 Rejzohn Wright
25 Asim Richards
26 Blake Freeland
27 Gervon Dexter
28 YaYa Diaby
29 Will Anderson Jr.
30 John Ojukwu
31 Rakim Jarrett
32 Joe Tippmann
33 Jon Gaines
34 Malaesala Aumavae-Laulu
35 Josh Downs
36 Anton Harrison
37 Nick Herbig
38 Carter Warren
39 Ali Gaye
40 Nolan Smith
41 Luke Schoonmaker
42 Isaiah Foskey
43 Yasir Abdullah
44 BJ Ojulari
45 C.J. Stroud
46 Broderick Jones
47 Matt Landers
48 Peter Skoronski
49 Bryan Bresee
50 Tanner McKee
51 Nick Hampton
52 Bijan Robinson
53 Bryce Ford-Wheaton
54 Thomas Incoom
55 Myles Murphy
56 Michael Mayer
57 Tre'Vius Hodges-Tomlinson
58 Darnell Wright
59 Deonte Banks
60 Ryan Hayes
```

```
61 Darrell Luter Jr.
62 Tyrus Wheat
63 Zay Flowers
64 Dontayvion Wicks
65 Anthony Bradford
66 Dante Stills
67 Isaiah McGuire
68 Darnell Washington
69 Adetomiwa Adebawore
70 Jacob Copeland
71 Tre Tucker
72 Chase Brown
73 Cam Smith
74 Owen Pappoe
75 Derick Hall
76 Parker Washington
77 Andre Carter II
78 Jalen Carter
79 Deneric Prince
80 Trey Palmer
81 Bryce Young
82 Sydney Brown
83 Jalen Redmond
84 Tuli Tuipulotu
85 Charlie Jones
86 Antonio Johnson
87 Derius Davis
88 Tyree Wilson
89 Darius Rush
90 Jonathan Mingo
91 Cameron Brown
92 Xavier Hutchinson
93 Noah Sewell
94 Dylan Horton
95 Charlie Thomas
96 Brenton Cox
97 Felix Anudike-Uzomah
98 Alex Ward
99 Richard Gouraige
100 Robert Beal
101 Colby Wooden
102 Shaka Heyward
103 Dee Winters
104 Jarrett Patterson
105 Jason Taylor II
106 Terell Smith
107 Tyler Lacy
108 Jalen Brooks
109 A.T. Perry
```

```
110 Habakkuk Baldonado
111 Mike Morris
112 Jartavius Martin
113 Ricky Stromberg
114 Isaiah Moore
115 O'Cyrus Torrence
116 Nathaniel Dell
117 Myles Brooks
118 Blake Whiteheart
119 Jaxon Smith-Njigba
120 Jaylon Jones
121 Rashee Rice
122 Calijah Kancey
123 Dalton Kincaid
124 Sean Tucker
125 Tiyon Evans
126 Ikenna Enechukwu
127 MJ Anderson
128 Daniel Scott
129 Grant DuBose
130 Cory Trice
131 Garrett Williams
132 Demario Douglas
133 Jerrod Clark
134 Tashawn Manning
135 Mazi Smith
136 Matthew Bergeron
137 SirVocea Dennis
138 Ronnie Hickman
139 Deuce Vaughn
140 Julius Brents
141 Israel Abanikanda
142 Zach Harrison
143 Kyu Blu Kelly
144 Drew Sanders
145 Jaquelin Roy
146 Alan Ali
147 Dorian Williams
148 Zach Evans
149 Jordan McFadden
150 Jake Bobo
151 Malik Heath
152 Jordan Addison
153 Joey Porter Jr.
154 Luke Wypler
155 Alex Forsyth
156 Brian Branch
157 Jovaughn Gwyn
158 Jack Podlesny
```

```
159 Jalen Wayne
160 Malik Knowles
161 Kayshon Boutte
162 Cedric Tillman
163 Paris Johnson Jr.
164 Sam LaPorta
165 Earl Bostick Jr.
166 Chamarri Conner
167 Riley Moss
168 Mohamed Ibrahim
169 Nick Broeker
170 Antoine Green
171 Keion White
172 John Michael Schmitz
173 Puka Nacua
174 Max Duggan
175 Jack Campbell
176 Connor Galvin
177 Jeremy Banks
178 Andrew Vorhees
179 Jake Andrews
180 Davis Allen
181 Tank Bigsby
182 Mekhi Garner
183 Evan Hull
184 Ochaun Mathis
185 Anfernee Orji
186 Jaxson Kirkland
187 Emil Ekiyor Jr.
188 DeMarvion Overshown
189 Clark Phillips III
190 Josh Whyle
191 Gervarrius Owens
192 Tyjae Spears
193 Eric Gray
194 Keeanu Benton
195 Warren McClendon
196 Jayden Reed
197 Malik Cunningham
198 Tyrique Stevenson
199 Rashad Torrence II
200 Will Levis
201 Cameron Mitchell
202 Mekhi Blackmon
203 Dalton Wagner
204 Kyle Patterson
205 Henry To'oTo'o
206 Juice Scruggs
207 Mitchell Tinsley
```

```
208 Tyreque Jones
209 Jordan Howden
210 Zach Charbonnet
211 Jay Ward
212 Kendre Miller
213 Zacch Pickens
214 Hendon Hooker
215 Devon Witherspoon
216 Aidan O'Connell
217 Elijah Higgins
218 JL Skinner
219 Luke Musgrave
220 Justin Shorter
221 Anthony Johnson Jr.
222 Brenton Strange
223 Jaren Hall
224 Lance Boykin
225 Anders Carlson
226 Clayton Tune
227 Byron Young
228 Joseph Ngata
229 Ji'Ayir Brown
230 Henry Bainivalu
231 Daiyan Henley
232 Ronnie Bell
233 Nic Jones
234 Moro Ojomo
235 Cameron Young
236 Noah Taylor
237 Dorian Thompson-Robinson
238 Siaki Ika
239 PJ Mustipher
240 Lonnie Phelps
241 Stetson Bennett
242 Keondre Coburn
243 Kearis Jackson
244 Will Mallory
245 Michael Wilson
246 Payne Durham
247 Jaray Jenkins
248 Arquon Bush
249 DJ Dale
250 Anthony Johnson
251 Cameron Latu
252 Jake Moody
253 Ivan Pace Jr.
254 Jammie Robinson
255 Jadon Haselwood
256 Jordan Battle
```

```
257 Dontay Demus Jr.
258 Trey Dean III
259 Jake Haener
260 Camerun Peoples
261 Ventrell Miller
262 Micah Baskerville
263 Leonard Taylor
264 DeMarcco Hellams
265 Christopher Smith
266 Brayden Willis
267 Travis Vokolek
268 SaRodorick Thompson
269 Kaevon Merriweather
270 Roschon Johnson
271 Brandon Joseph
272 Jalen Graham
273 Bryce Baringer
274 Tavion Thomas
275 Kenny McIntosh
276 Travis Dye
277 Christopher Dunn
278 Adam Korsak
279 Alex Austin
280 Brad Robbins
281 Michael Turk
282 Cam Jones
283 B.T. Potter
284 Chad Ryland
```

```
# Convert predicted probabilities to binary predictions based on a
threshold (e.g., 0.5)
XGBpredicted_labels = (XGBbaseline_preds[:, 1] > 0.5).astype(int)

# Evaluation for ranking metrics
# Sort the predictions based on probability scores
sorted_indices = np.argsort(-XGBbaseline_preds[:, 1])
k = 10
num_relevant = sum(test_y)

def calculate_MRR(sorted_indices, test_y):
    # Calculate Mean Reciprocal Rank (MRR)
    mrr = 0
    for idx, i in enumerate(sorted_indices):
        if test_y.iloc[i] == 1:  # Use iloc to access test_y by index
            mrr = 1 / (idx + 1)
            break
    return mrr

def calculate_MAP(sorted_indices, test_y):
    # Calculate Mean Average Precision (MAP)
```

```
    ap = 0
    for idx, i in enumerate(sorted_indices):
        if test_y.iloc[i] == 1:
            ap += sum(test_y.iloc[:idx + 1]) / (idx + 1)
    map_score = ap / num_relevant
    return map_score

def calculate_NDCG(sorted_indices, test_y):
    # Calculate Normalized Discounted Cumulative Gain (NDCG) at k=10
    dcg = 0
    idcg = sum(1 / np.log2(np.arange(2, k + 2)))
    for idx, i in enumerate(sorted_indices[:k]):
        if test_y.iloc[i] == 1:
            dcg += 1 / np.log2(idx + 2)
    ndcg = dcg / idcg
    return ndcg

def calculate_PAK(sorted_indices, test_y):
    # Calculate Precision at k (P@k)
    tp_at_k = sum(test_y.iloc[sorted_indices[:k]])
    precision_at_k = tp_at_k / k
    return precision_at_k

def calculate_RAK(sorted_indices, test_y):
    # Calculate Recall at k (R@k)
    tp_at_k = sum(test_y.iloc[sorted_indices[:k]])
    recall_at_k = tp_at_k / num_relevant
    return recall_at_k

from tabulate import tabulate
from sklearn.metrics import accuracy_score, roc_auc_score

# Calculate all measurements
baseline_measurements = [
    ("Accuracy", accuracy_score(test_y, XGBpredicted_labels)),
    ("ROC AUC Score", roc_auc_score(test_y, XGBbaseline_preds[:, 1])),
    ("Mean Reciprocal Rank (MRR)", calculate_MRR(sorted_indices,
test_y)),
    ("Mean Average Precision (MAP)", calculate_MAP(sorted_indices,
test_y)),
    ("Normalized Discounted Cumulative Gain (NDCG) at k=10",
calculate_NDCG(sorted_indices, test_y)),
    ("Precision at k (P@k) at k=10", calculate_PAK(sorted_indices,
test_y)),
    ("Recall at k (R@k) at k=10", calculate_RAK(sorted_indices,
test_y))
]

# Print measurements in a table format
```

```python
print("Baseline measurements")
print(tabulate(baseline_measurements, headers=["Metric", "Value"]))
```

```
Baseline measurements
Metric                                                      Value
----------------------------------------------------  --------
Accuracy                                              0.897887
ROC AUC Score                                         0.764841
Mean Reciprocal Rank (MRR)                            1
Mean Average Precision (MAP)                          0.110987
Normalized Discounted Cumulative Gain (NDCG) at k=10  0.371854
Precision at k (P@k) at k=10                          0.3
Recall at k (R@k) at k=10                             0.103448
```

```python
best_XGB = xgb.XGBClassifier(
    colsample_bytree=0.8,
    eta=0.1,
    eval_metric='logloss',
    max_depth=6,
    min_child_weight=1,
    objective='binary:logistic',
    subsample=0.8
)

# Hypertuning parameters using 5-Fold Cross Validation method
scores = cross_val_score(best_XGB, train_X, train_y, cv=5)

best_XGB.fit(train_X, train_y)
```

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=0.8, device=None,
early_stopping_rounds=None,
              enable_categorical=False, eta=0.1,
eval_metric='logloss',
              feature_types=None, gamma=None, grow_policy=None,
              importance_type=None, interaction_constraints=None,
              learning_rate=None, max_bin=None,
max_cat_threshold=None,
              max_cat_to_onehot=None, max_delta_step=None,
max_depth=6,
              max_leaves=None, min_child_weight=1, missing=nan,
              monotone_constraints=None, multi_strategy=None,
n_estimators=None,
              n_jobs=None, num_parallel_tree=None, ...)
```

```python
# Predict on the testing data
y_pred = best_XGB.predict(test_X)

# Calculate accuracy
```

```
accuracy = accuracy_score(test_y, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.8943661971830986
```

```
# Predicting the probabilities of Test set
preds = best_XGB.predict_proba(test_X)
count = 1

# Ranking done according to the probability scores
for i in pd.DataFrame(preds).sort_values(by=1, ascending=False).index:
    print(str(count) + " " + str(df[df.Year==2023].reset_index().at[i,
"Name"]))
    count += 1
```

```
1 Dawand Jones
2 Darnell Wright
3 Byron Young
4 Marvin Mims
5 Anthony Richardson
6 Emmanuel Forbes
7 C.J. Stroud
8 Kelee Ringo
9 Anton Harrison
10 Jakorian Bennett
11 Will Anderson Jr.
12 Tyler Steen
13 Rejzohn Wright
14 Thomas Incoom
15 Richard Gouraige
16 Lukas Van Ness
17 Adetomiwa Adebawore
18 Christian Gonzalez
19 Michael Mayer
20 Anthony Bradford
21 YaYa Diaby
22 Joe Tippmann
23 Wanya Morris
24 Quentin Johnston
25 Blake Freeland
26 Bryce Young
27 Asim Richards
28 Carrington Valentine
29 Broderick Jones
30 Malaesala Aumavae-Laulu
31 Calijah Kancey
32 DJ Turner
33 Isaiah Foskey
34 Matthew Bergeron
35 Gervon Dexter
```

```
36 Mazi Smith
37 Ryan Hayes
38 Trenton Simpson
39 Carter Warren
40 Nolan Smith
41 John Ojukwu
42 Jalin Hyatt
43 Henry To'oTo'o
44 Jon Gaines
45 Zach Charbonnet
46 Zach Harrison
47 Tyree Wilson
48 Jonathan Mingo
49 Tavius Robinson
50 Luke Schoonmaker
51 Tanner McKee
52 Bijan Robinson
53 Peter Skoronski
54 Rakim Jarrett
55 Robert Beal
56 Darrell Luter Jr.
57 Paris Johnson Jr.
58 Nathaniel Dell
59 Hendon Hooker
60 Tyler Scott
61 Ali Gaye
62 Jalen Redmond
63 Devon Achane
64 A.T. Perry
65 Sydney Brown
66 Owen Pappoe
67 Josh Downs
68 Yasir Abdullah
69 Zacch Pickens
70 Anfernee Orji
71 Darnell Washington
72 Kayshon Boutte
73 Rashee Rice
74 Bryan Bresee
75 Warren McClendon
76 Dontayvion Wicks
77 Riley Moss
78 Jaxson Kirkland
79 Myles Brooks
80 Sam LaPorta
81 BJ Ojulari
82 Dante Stills
83 Will Levis
84 Jordan Battle
```

```
85 Nick Broeker
86 Jalen Carter
87 Brian Branch
88 Nick Herbig
89 Keaton Mitchell
90 Earl Bostick Jr.
91 Cedric Tillman
92 Dalton Kincaid
93 Jason Taylor II
94 Jaylon Jones
95 Nick Hampton
96 Tashawn Manning
97 Matt Landers
98 Jeremy Banks
99 Jaren Hall
100 Darius Rush
101 Moro Ojomo
102 Brandon Hill
103 Emil Ekiyor Jr.
104 Tre'Vius Hodges-Tomlinson
105 Jake Andrews
106 Tyrus Wheat
107 MJ Anderson
108 Jordan Addison
109 Garrett Williams
110 Jartavius Martin
111 Myles Murphy
112 Elijah Higgins
113 Ji'Ayir Brown
114 Jerrod Clark
115 Zay Flowers
116 Shaka Heyward
117 Clayton Tune
118 Davis Allen
119 Derick Hall
120 Alan Ali
121 Mike Morris
122 Andrew Vorhees
123 Byron Young
124 Sean Tucker
125 Tyreque Jones
126 Brenton Cox
127 Mohamed Ibrahim
128 Isaiah McGuire
129 Tiyon Evans
130 Aidan O'Connell
131 Ronnie Bell
132 Deonte Banks
133 Jarrett Patterson
```

```
134 Dorian Williams
135 Jacob Copeland
136 Jordan McFadden
137 Xavier Hutchinson
138 Chamarri Conner
139 Keeanu Benton
140 Parker Washington
141 Rashad Torrence II
142 Jalen Brooks
143 Malik Knowles
144 Noah Sewell
145 Jack Podlesny
146 Jack Campbell
147 Tuli Tuipulotu
148 Gervarrius Owens
149 Puka Nacua
150 Colby Wooden
151 Eli Ricks
152 Tank Bigsby
153 Luke Wypler
154 Ikenna Enechukwu
155 Jay Ward
156 Habakkuk Baldonado
157 Keion White
158 Payne Durham
159 Ricky Stromberg
160 Antonio Johnson
161 Jake Bobo
162 Malik Cunningham
163 Felix Anudike-Uzomah
164 Kyu Blu Kelly
165 Lonnie Phelps
166 Josh Whyle
167 John Michael Schmitz
168 Demario Douglas
169 Ronnie Hickman
170 Daniel Scott
171 Kearis Jackson
172 Jalen Wayne
173 Cam Smith
174 Dee Winters
175 Brenton Strange
176 Charlie Thomas
177 O'Cyrus Torrence
178 Ochaun Mathis
179 Tyrique Stevenson
180 Michael Wilson
181 Jayden Reed
182 Bryce Ford-Wheaton
```

```
183 Dorian Thompson-Robinson
184 Blake Whiteheart
185 Chase Brown
186 Jovaughn Gwyn
187 Deneric Prince
188 Cameron Brown
189 Deuce Vaughn
190 Jammie Robinson
191 Tyjae Spears
192 Luke Musgrave
193 Zach Evans
194 Tre Tucker
195 Joey Porter Jr.
196 Anthony Johnson
197 Cameron Mitchell
198 Grant DuBose
199 Juice Scruggs
200 Tyler Lacy
201 Antoine Green
202 Julius Brents
203 Arquon Bush
204 Cameron Young
205 Keondre Coburn
206 Henry Bainivalu
207 Jaxon Smith-Njigba
208 Jake Haener
209 Kendre Miller
210 Israel Abanikanda
211 Anders Carlson
212 JL Skinner
213 Joseph Ngata
214 Jadon Haselwood
215 DeMarcco Hellams
216 Jordan Howden
217 Alex Forsyth
218 Jaquelin Roy
219 Anthony Johnson Jr.
220 Connor Galvin
221 Nic Jones
222 Will Mallory
223 Kaevon Merriweather
224 Noah Taylor
225 Jahmyr Gibbs
226 Jake Moody
227 Daiyan Henley
228 Trey Palmer
229 Stetson Bennett
230 Drew Sanders
231 Evan Hull
```

```
232 Mekhi Blackmon
233 Malik Heath
234 Roschon Johnson
235 Dontay Demus Jr.
236 Eric Gray
237 Terell Smith
238 Christopher Smith
239 Lance Boykin
240 Clark Phillips III
241 Andre Carter II
242 Micah Baskerville
243 Dalton Wagner
244 Dylan Horton
245 Charlie Jones
246 Cory Trice
247 Jaray Jenkins
248 Alex Ward
249 Leonard Taylor
250 Cameron Latu
251 Devon Witherspoon
252 Kenny McIntosh
253 Max Duggan
254 Travis Dye
255 Ivan Pace Jr.
256 Mitchell Tinsley
257 Alex Austin
258 Brayden Willis
259 Justin Shorter
260 SirVocea Dennis
261 DJ Dale
262 Brandon Joseph
263 Bryce Baringer
264 Derius Davis
265 PJ Mustipher
266 Trey Dean III
267 Camerun Peoples
268 Siaki Ika
269 Christopher Dunn
270 Adam Korsak
271 DeMarvion Overshown
272 Isaiah Moore
273 Travis Vokolek
274 Kyle Patterson
275 SaRodorick Thompson
276 Ventrell Miller
277 Mekhi Garner
278 Tavion Thomas
279 Michael Turk
280 Brad Robbins
281 Chad Ryland
```

```
282 B.T. Potter
283 Jalen Graham
284 Cam Jones
```

```python
from sklearn.metrics import accuracy_score, roc_auc_score

# Convert predicted probabilities to binary predictions based on a
threshold (e.g., 0.5)
predicted_labels = (preds[:, 1] > 0.5).astype(int)

# Evaluation for ranking metrics
# Sort the predictions based on probability scores
sorted_indices = np.argsort(-preds[:, 1])

# Calculate all measurements
best_rf_measurements = [
    ("Accuracy", accuracy_score(test_y, predicted_labels)),
    ("ROC AUC Score", roc_auc_score(test_y, preds[:, 1])),
    ("Mean Reciprocal Rank (MRR)", calculate_MRR(sorted_indices,
test_y)),
    ("Mean Average Precision (MAP)", calculate_MAP(sorted_indices,
test_y)),
    ("Normalized Discounted Cumulative Gain (NDCG) at k=10",
calculate_NDCG(sorted_indices, test_y)),
    ("Precision at k (P@k) at k=10", calculate_PAK(sorted_indices,
test_y)),
    ("Recall at k (R@k) at k=10", calculate_RAK(sorted_indices,
test_y))
]

# Print measurements in a table format
print("Best Fit measurements")
print(tabulate(best_rf_measurements, headers=["Metric", "Value"]))
```

```
Best Fit measurements
Metric                                                   Value
----------------------------------------------------  --------
Accuracy                                              0.894366
ROC AUC Score                                         0.766329
Mean Reciprocal Rank (MRR)                            0.5
Mean Average Precision (MAP)                          0.114885
Normalized Discounted Cumulative Gain (NDCG) at k=10  0.442022
Precision at k (P@k) at k=10                          0.5
Recall at k (R@k) at k=10                             0.172414
```

Comparative Analysis of Baseline and Best-Fit XGBoost model for Ranking Prediction

Accuracy: Baseline is slightly higher, indicating it correctly classified a marginally higher percentage of the total. ROC AUC Score: Both results are identical, showing the same ability to discriminate between classes.

MRR: Baseline is perfect, indicating it always ranks the correct item highest. Baseline result shows a significant drop, which could be critical if the goal is to rank a correct item as high as possible.

MAP: Best fit is slightly better, indicating a slight improvement in the ranking of relevant items across queries.

NDCG at k=10: Best fit is higher, showing it ranks relevant items more effectively within the top 10 positions.

P@k at k=10: Best fit is significantly higher, suggesting it has a better top-10 precision.

R@k at k=10: Best fit is also higher here, indicating it retrieves a higher proportion of relevant items within its top 10 predictions.

Conclusion

For Ranking Tasks: If the focus is on ranking performance, particularly in retrieving and ranking the most relevant items as high as possible, Best fit is better. It shows superior performance in MAP, NDCG, P@k, and R@k, which are critical for ranking and recommendation systems.

Reflection

What is the most challenging part of the project that you've encountered so far?

Tackling the use of the "Round" feature in our NFL data to predict which players would make the cut was a significant challenge, diverging notably from traditional classification problems. The ordinal nature of draft rounds required an approach that recognized the inherent ranking, not just discrete categories. Standard classification models and accuracy metrics fell short in addressing the nuanced complexity of predicting players' success based on draft rounds, due to their inability to grasp the ordered significance of the data. By pivoting to strategies that accommodate the ordinality in predictions and employing ranking-specific evaluation metrics, we successfully navigated this challenge. This adaptation underscored the importance of innovative problem-solving and marked a significant achievement in our project, demonstrating our capacity to extend beyond conventional methodologies to yield meaningful insights in the context of sports analytics.

What are your initial insights?

Our first look at the data shows that teams often choose Offensive Tackles in the early rounds of the draft, showing that these players are really important for the team's game plan. On the other hand, doing well at the combine - where players show off their physical skills - doesn't always mean a player will be picked early in the draft. This tells us that being in great shape and showing good skills at the combine helps, but it's not the only thing teams think about when deciding who to pick. They also consider how well players have played in the past, what the team needs, and other special qualities a player might have.

Are there any concrete results you can show at this point? If not, why not?

The data has been properly imputed and prepared for analysis. The results so far indicate the project is viable and has potential for further refinement and enhancement. The specific hypothesis -*certain key offensive and defensive positions, like Offensive Tackles, tend to be drafted earlier, while specialized positions such as Kickers and Punters are often selected in later rounds*- explored during the exploratory data analysis (EDA) is proving to be a distinguishing and key factor for the model to predict and rank the draft. Concrete results have been obtained and improvements in the ranking measurements are achieved through the implementation of two machine learning models.

Going forward, what are the current biggest problems you're facing? Moving forward, the current biggest hurdle lies in incorporating the "Pick" feature alongside "Round" to predict ranking. This presents a unique challenge due to the interplay between these variables and the need for nuanced handling to ensure accurate predictions.

Do you think you are on track with your project? If not, what parts do you need to dedicate more time to?

Despite the challenges encountered, I believe the project is on track, with significant progress made towards achieving the objectives outlined. Moving forward, dedicating more time to fine-tuning the model architecture and optimising feature selection strategies will be crucial to further advancing the project's outcomes.

Given your initial exploration of the data, is it worth proceeding with your project, why? If not, how will you move forward (method, data etc)?

Given the initial exploration of the data and the promising results obtained thus far, it is certainly worth proceeding with the project. The insights gained and the improvements observed in ranking measurements underscore the project's potential for meaningful impact and contribute to its continued pursuit. Moving forward, continued data analysis, model refinement, and iterative experimentation will be key to realising the project's full potential

Next Step: Concrete plans and goals for the next month

The focus will be on implementing additional machine learning models to further enhance the predictive capabilities for ranking. Specifically, the plan involves developing and evaluating at least three more models tailored to predict rank, leveraging various algorithms and techniques to explore the dataset comprehensively. Additionally, significant attention will be devoted to hyperparameter tuning across all models to optimise performance and maximise predictive accuracy. A notable advancement in the approach will be the incorporation of both "Round" and "Pick" features to predict ranking, aiming to capitalise on the combined predictive power of these variables. Furthermore, an innovative strategy will be explored, involving the integration of features extracted from different models to create a composite predictive framework, thereby potentially enhancing the accuracy and robustness of the ranking predictions. These concrete plans and goals underscore a strategic and iterative approach towards advancing the project's objectives and refining the predictive models for optimal performance.