March Machine Learning Mania 2024

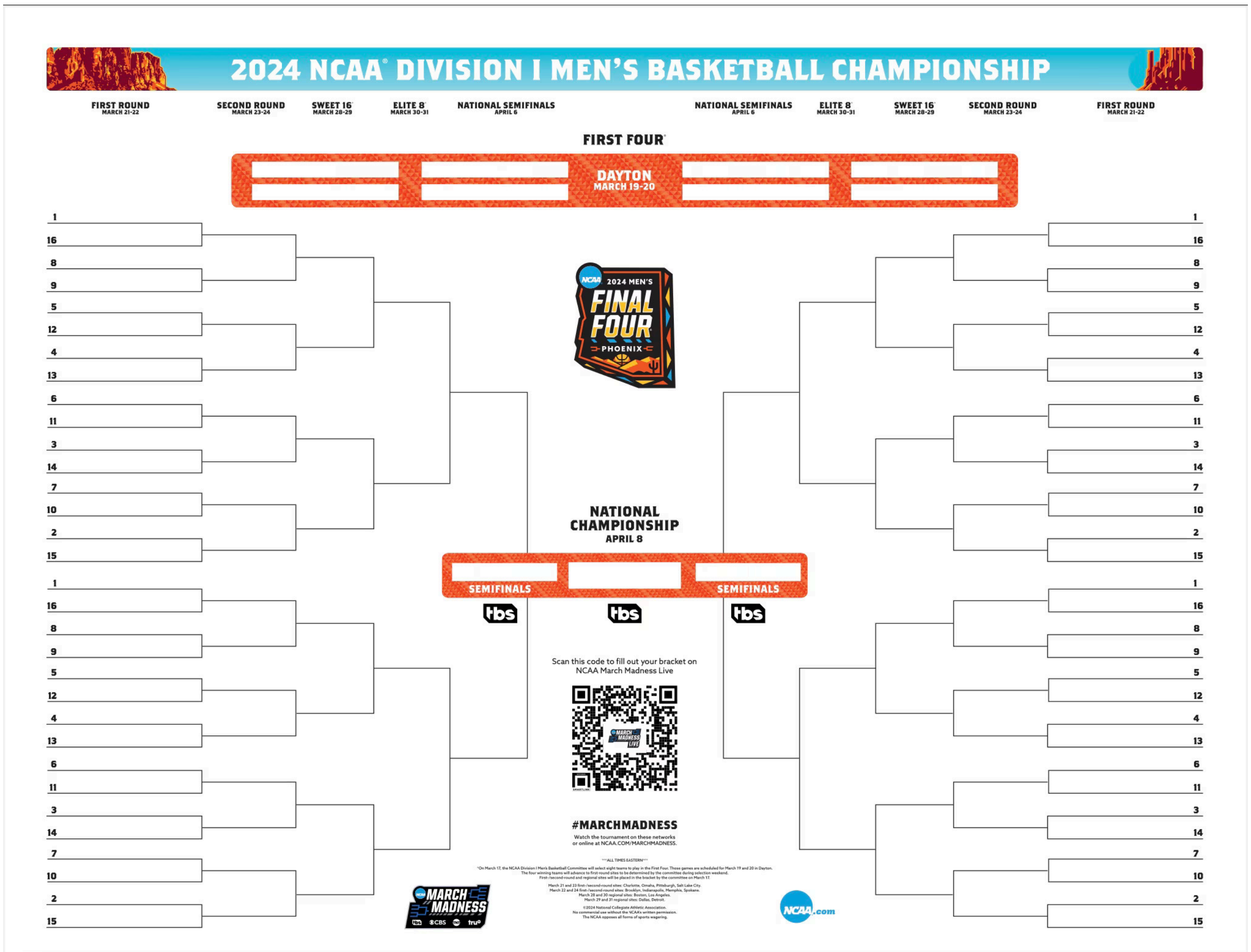
Code

AUTHOR
Shubham Palav

ABSTRACT

- The “March Machine Learning Mania 2024” project aims to forecast the outcomes of the 2024 NCAA Basketball Tournaments by analyzing historical data and presenting a portfolio of brackets for both the men’s and women’s divisions.
- The NCAA (National Collegiate Athletic Association) is a prominent American organization that governs college athletics and organizes the highly anticipated collegiate basketball tournaments, known as “March Madness.” This single-elimination tournament, held annually in March and April, features 68 elite college teams competing for the national championship. The project leverages machine learning techniques to predict game results, offering insights and forecasts for the 2024 tournaments.

You can check the Github repository for my project.



EDA

- pandas to be able to read data
- matplotlib to plot bar charts and basic histograms
- seaborn to plot box charts easily and then subplots
- sklearn for machine learning implementation

Data Loading.

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import os
```

```
# Data Section 1 - The Basics:
MTeams=
pd.read_csv('D:\Source\Orion_innovation_internship_repos\Basketball_Bracket_Forecasting_2024\Dat
asets\MTeams.csv')
WTeams=
pd.read_csv('D:\Source\Orion_innovation_internship_repos\Basketball_Bracket_Forecasting_2024\Dat
asets\WTeams.csv')

MSeasons=
pd.read_csv('D:\Source\Orion_innovation_internship_repos\Basketball_Bracket_Forecasting_2024\Dat
asets\MSeasons.csv')
WSeasons=
pd.read_csv('D:\Source\Orion_innovation_internship_repos\Basketball_Bracket_Forecasting_2024\Dat
asets\WSeasons.csv')

MNCAATourneySeeds=
pd.read_csv('D:\Source\Orion_innovation_internship_repos\Basketball_Bracket_Forecasting_2024\Dat
asets\MNCAATourneySeeds.csv')
WNCAATourneySeeds=
pd.read_csv('D:\Source\Orion_innovation_internship_repos\Basketball_Bracket_Forecasting_2024\Dat
asets\WNCAATourneySeeds.csv')

# Data Section 2 - Team Box Scores
MRegularSeasonCompactResults=
pd.read_csv('D:\Source\Orion_innovation_internship_repos\Basketball_Bracket_Forecasting_2024\Dat
asets\MRegularSeasonCompactResults.csv')
WRegularSeasonCompactResults=
pd.read_csv('D:\Source\Orion_innovation_internship_repos\Basketball_Bracket_Forecasting_2024\Dat
asets\WRegularSeasonCompactResults.csv')

MNCAATourneyCompactResults=
```

```
pd.read_csv('D:\Source\Orion_innovation_internship_repos\Basketball_Bracket_Forecasting_2024\Dat
asets\MNCAATourneyCompactResults.csv')
         WNCAATourneyCompactResults=
pd.read_csv('D:\Source\Orion_innovation_internship_repos\Basketball_Bracket_Forecasting_2024\Dat
asets\WNCAATourneyCompactResults.csv')

         MRegularSeasonDetailedResults=
pd.read_csv('D:\Source\Orion_innovation_internship_repos\Basketball_Bracket_Forecasting_2024\Dat
asets\MRegularSeasonDetailedResults.csv')
         WRegularSeasonDetailedResults=
pd.read_csv('D:\Source\Orion_innovation_internship_repos\Basketball_Bracket_Forecasting_2024\Dat
asets\WRegularSeasonDetailedResults.csv')

         MNCAATourneyDetailedResults=
pd.read_csv('D:\Source\Orion_innovation_internship_repos\Basketball_Bracket_Forecasting_2024\Dat
asets\MNCAATourneyDetailedResults.csv')
         WNCAATourneyDetailedResults=
pd.read_csv('D:\Source\Orion_innovation_internship_repos\Basketball_Bracket_Forecasting_2024\Dat
asets\WNCAATourneyDetailedResults.csv')

         # Data Section 2 - Team Box Scores
         MRegularSeasonCompactResults=
pd.read_csv('D:\Source\Orion_innovation_internship_repos\Basketball_Bracket_Forecasting_2024\Dat
asets\MRegularSeasonCompactResults.csv')
         WRegularSeasonCompactResults=
pd.read_csv('D:\Source\Orion_innovation_internship_repos\Basketball_Bracket_Forecasting_2024\Dat
asets\WRegularSeasonCompactResults.csv')

         MNCAATourneyCompactResults=
pd.read_csv('D:\Source\Orion_innovation_internship_repos\Basketball_Bracket_Forecasting_2024\Dat
asets\MNCAATourneyCompactResults.csv')
         WNCAATourneyCompactResults=
pd.read_csv('D:\Source\Orion_innovation_internship_repos\Basketball_Bracket_Forecasting_2024\Dat
asets\WNCAATourneyCompactResults.csv')

         MRegularSeasonDetailedResults=
pd.read_csv('D:\Source\Orion_innovation_internship_repos\Basketball_Bracket_Forecasting_2024\Dat
asets\MRegularSeasonDetailedResults.csv')
         WRegularSeasonDetailedResults=
pd.read_csv('D:\Source\Orion_innovation_internship_repos\Basketball_Bracket_Forecasting_2024\Dat
asets\WRegularSeasonDetailedResults.csv')

         MNCAATourneyDetailedResults=
pd.read_csv('D:\Source\Orion_innovation_internship_repos\Basketball_Bracket_Forecasting_2024\Dat
asets\MNCAATourneyDetailedResults.csv')
         WNCAATourneyDetailedResults=
pd.read_csv('D:\Source\Orion_innovation_internship_repos\Basketball_Bracket_Forecasting_2024\Dat
asets\WNCAATourneyDetailedResults.csv')

         # Data Section 3 - Geography
         Cities=
```

```
pd.read_csv('D:\Source\Orion_innovation_internship_repos\Basketball_Bracket_Forecasting_2024\Datasets\Cities.csv')

         MGameCities=
pd.read_csv('D:\Source\Orion_innovation_internship_repos\Basketball_Bracket_Forecasting_2024\Datasets\MGameCities.csv')
         WGameCities=
pd.read_csv('D:\Source\Orion_innovation_internship_repos\Basketball_Bracket_Forecasting_2024\Datasets\WGameCities.csv')


         # Data Section 5 - Supplements
         MTeamCoaches=
pd.read_csv('D:\Source\Orion_innovation_internship_repos\Basketball_Bracket_Forecasting_2024\Datasets\MTeamCoaches.csv')

         Conferences=
pd.read_csv('D:\Source\Orion_innovation_internship_repos\Basketball_Bracket_Forecasting_2024\Datasets\Conferences.csv')

         MTeamConferences=
pd.read_csv('D:\Source\Orion_innovation_internship_repos\Basketball_Bracket_Forecasting_2024\Datasets\MTeamConferences.csv')
         WTeamConferences=
pd.read_csv('D:\Source\Orion_innovation_internship_repos\Basketball_Bracket_Forecasting_2024\Datasets\WTeamConferences.csv')

         MConferenceTourneyGames=
pd.read_csv('D:\Source\Orion_innovation_internship_repos\Basketball_Bracket_Forecasting_2024\Datasets\MConferenceTourneyGames.csv')

         MSecondaryTourneyTeams=
pd.read_csv('D:\Source\Orion_innovation_internship_repos\Basketball_Bracket_Forecasting_2024\Datasets\MSecondaryTourneyTeams.csv')

         MSecondaryTourneyCompactResults=
pd.read_csv('D:\Source\Orion_innovation_internship_repos\Basketball_Bracket_Forecasting_2024\Datasets\MSecondaryTourneyCompactResults.csv')

         MTeamSpellings=
pd.read_csv('D:\Source\Orion_innovation_internship_repos\Basketball_Bracket_Forecasting_2024\Datasets\MTeamSpellings.csv', encoding='ISO-8859-1')
         WTeamSpellings=
pd.read_csv('D:\Source\Orion_innovation_internship_repos\Basketball_Bracket_Forecasting_2024\Datasets\WTeamSpellings.csv', encoding='ISO-8859-1')

         MNCAATourneySlots=
pd.read_csv('D:\Source\Orion_innovation_internship_repos\Basketball_Bracket_Forecasting_2024\Datasets\MNCAATourneySlots.csv')
         WNCAATourneySlots=
pd.read_csv('D:\Source\Orion_innovation_internship_repos\Basketball_Bracket_Forecasting_2024\Dat
```

```
asets\WNCAATourneySlots.csv')

        MNCAATourneySeedRoundSlots=
pd.read_csv('D:\Source\Orion_innovation_internship_repos\Basketball_Bracket_Forecasting_2024\Dat
asets\MNCAATourneySeedRoundSlots.csv')
```

```
        # Data Section 6 - Others
        tourney_seeds_2024=
pd.read_csv('D:\Source\Orion_innovation_internship_repos\Basketball_Bracket_Forecasting_2024\Dat
asets\seeds.csv')

        sample_submission=
pd.read_csv('D:\Source\Orion_innovation_internship_repos\Basketball_Bracket_Forecasting_2024\Dat
asets\sample_submission.csv')
```

1. Exploratory Data Analysis (EDA):

Men

```
        MTeams.head()
```

	TeamID	TeamName	FirstD1Season	LastD1Season
0	1101	Abilene Chr	2014	2024
1	1102	Air Force	1985	2024
2	1103	Akron	1985	2024
3	1104	Alabama	1985	2024
4	1105	Alabama A&M	2000	2024

```
        MTeams['D1Seasons'] = MTeams['LastD1Season'] - MTeams['FirstD1Season']
        df_d1_2024 = MTeams[MTeams['LastD1Season'] == 2024]
        fewest_d1_seasons = df_d1_2024.nsmallest(20, 'D1Seasons')

        plt.figure(figsize=(10, 6))
        plt.bar(fewest_d1_seasons['TeamName'], fewest_d1_seasons['D1Seasons'],
color='skyblue')
        plt.title('Teams with the Fewest D1 Seasons (up to 2024)')
        plt.xlabel('Team Name')
        plt.ylabel('Number of D1 Seasons')
        plt.xticks(rotation=90)
        plt.tight_layout()

        plt.show()
```

```python
def plot_regions(df):
    region_w_counts = df['RegionW'].value_counts()
    region_x_counts = df['RegionX'].value_counts()
    region_y_counts = df['RegionY'].value_counts()
    region_z_counts = df['RegionZ'].value_counts()

    region_counts = pd.DataFrame({
        'RegionW': region_w_counts,
        'RegionX': region_x_counts,
        'RegionY': region_y_counts,
        'RegionZ': region_z_counts
    })

    region_counts = region_counts.fillna(0).astype(int)

    fig, ax = plt.subplots(2, 2, figsize=(14, 10), sharey=True)

    ax[0, 0].bar(region_counts.index, region_counts['RegionW'], color='blue')
    ax[0, 0].set_title('Region W Counts')
    ax[0, 0].set_ylabel('Counts')
    ax[0, 0].tick_params(axis='x', rotation=90)

    ax[0, 1].bar(region_counts.index, region_counts['RegionX'], color='red')
    ax[0, 1].set_title('Region X Counts')
    ax[0, 1].tick_params(axis='x', rotation=90)

    ax[1, 0].bar(region_counts.index, region_counts['RegionY'], color='orange')
    ax[1, 0].set_title('Region Y Counts')
    ax[1, 0].set_ylabel('Counts')
    ax[1, 0].set_xlabel('Region')
    ax[1, 0].tick_params(axis='x', rotation=90)

    ax[1, 1].bar(region_counts.index, region_counts['RegionZ'], color='yellow')
    ax[1, 1].set_title('Region Z Counts')
    ax[1, 1].set_xlabel('Region')
    ax[1, 1].tick_params(axis='x', rotation=90)

    plt.tight_layout()
    plt.show()

plot_regions(MSeasons)
```

```python
tourney_seeds_2024_MTeam = tourney_seeds_2024[tourney_seeds_2024['Tournament'] == 'M']
tourney_seeds_2024_MTeam = pd.merge(tourney_seeds_2024_MTeam, MTeams, on='TeamID', how='left')
tourney_seeds_2024_MTeam.head()
```

	Tournament	Seed	TeamID	TeamName	FirstD1Season	LastD1Season	D1Seasons
0	M	W01	1163	Connecticut	1985	2024	39
1	M	W02	1235	Iowa St	1985	2024	39
2	M	W03	1228	Illinois	1985	2024	39
3	M	W04	1120	Auburn	1985	2024	39
4	M	W05	1361	San Diego St	1985	2024	39

```
        plt.figure(figsize=(10, 6))
        plt.hist(tourney_seeds_2024_MTeam['FirstD1Season'], bins=30, color='skyblue',
edgecolor='black')


        plt.xlabel('FirstD1Season')
        plt.ylabel('Frequency')
        plt.title('Histogram of FirstD1Season in 2024 Tournament Seeds')


        plt.show()
```

Women

```
        tourney_seeds_2024_WTeam = tourney_seeds_2024[tourney_seeds_2024['Tournament'] == 'W']
        tourney_seeds_2024_WTeam = pd.merge(tourney_seeds_2024_WTeam, WTeams, on='TeamID',
how='left')
        tourney_seeds_2024_WTeam.head()
```

	Tournament	Seed	TeamID	TeamName
0	W	W01	3376	South Carolina
1	W	W02	3323	Notre Dame
2	W	W03	3333	Oregon St
3	W	W04	3231	Indiana
4	W	W05	3328	Oklahoma

Seed Rank

Men

```
        MNCAATourneyCompactResults_2003 =
MNCAATourneyCompactResults[MNCAATourneyCompactResults['Season']>=2003]
        MNCAATourneyCompactResults_2003
```

```
        df_merged_seeds_M = pd.merge(MNCAATourneyCompactResults_2003,
MNCAATourneySeeds[['Season','TeamID','Seed']], left_on=['Season','WTeamID'], right_on=
['Season','TeamID'], how='left').rename(columns={'Seed':'WSeed'})
        df_merged_seeds_M = pd.merge(df_merged_seeds_M,
MNCAATourneySeeds[['Season','TeamID','Seed']], left_on=['Season','LTeamID'], right_on=
['Season','TeamID'], how='left').rename(columns={'Seed':'LSeed'})

        df_merged_seeds_M['WRank'] = df_merged_seeds_M['WSeed'].str[1:3].astype(int)
        df_merged_seeds_M['LRank'] = df_merged_seeds_M['LSeed'].str[1:3].astype(int)
        df_merged_seeds_M['RankDiff'] = df_merged_seeds_M['LRank'] -
df_merged_seeds_M['WRank']

        columns_to_delete = ['TeamID_x', 'TeamID_y']
        df_merged_seeds_M = df_merged_seeds_M.drop(columns=columns_to_delete)
        df_merged_seeds_M.head()
```

	Season	DayNum	WTeamID	WScore	LTeamID	LScore	WLoc	NumOT	WSeed	LSeed	WRank	LRank	RankDiff
0	2003	134	1421	92	1411	84	N	1	X16b	X16a	16	16	0
1	2003	136	1112	80	1436	51	N	0	Z01	Z16	1	16	15
2	2003	136	1113	84	1272	71	N	0	Z10	Z07	10	7	-3
3	2003	136	1141	79	1166	73	N	0	Z11	Z06	11	6	-5
4	2003	136	1143	76	1301	74	N	1	W08	W09	8	9	1

This histogram shows the difference in seed rank between the two teams (not considering the regions). Negative means the winner has a lower seed rank.

More than 60% of the games are won by the higher ranked team. But more than 10% of the games were won by teams ranked significantly lower. Therefore, the seed ranking of the competition team cannot accurately predict the outcome of the competition.

```
        plt.figure(figsize=(12, 6))
        plt.hist(df_merged_seeds_M['RankDiff'], bins=10, color='skyblue', edgecolor='black')
        plt.title('Distribution of Rank Differences')
        plt.xlabel('Rank Difference')
        plt.ylabel('Frequency')
        plt.grid(True)
        plt.show()
```

Women

```
        WNCAATourneyCompactResults_2010 =
WNCAATourneyCompactResults[WNCAATourneyCompactResults['Season']>=2010]
        WNCAATourneyCompactResults_2010
```

```
        df_merged_seeds_W = pd.merge(WNCAATourneyCompactResults_2010,
WNCAATourneySeeds[['Season','TeamID','Seed']], left_on=['Season','WTeamID'], right_on=
['Season','TeamID'], how='left').rename(columns={'Seed':'WSeed'})
        df_merged_seeds_W = pd.merge(df_merged_seeds_W,
WNCAATourneySeeds[['Season','TeamID','Seed']], left_on=['Season','LTeamID'], right_on=
['Season','TeamID'], how='left').rename(columns={'Seed':'LSeed'})

        df_merged_seeds_W['WRank'] = df_merged_seeds_W['WSeed'].str[1:3].astype(int)
        df_merged_seeds_W['LRank'] = df_merged_seeds_W['LSeed'].str[1:3].astype(int)
        df_merged_seeds_W['RankDiff'] = df_merged_seeds_W['LRank'] -
df_merged_seeds_W['WRank']

        columns_to_delete = ['TeamID_x', 'TeamID_y']
        df_merged_seeds_W = df_merged_seeds_W.drop(columns=columns_to_delete)
        df_merged_seeds_W.head()
```

	Season	DayNum	WTeamID	WScore	LTeamID	LScore	WLoc	NumOT	WSeed	LSeed	WRank	LRank	RankDiff
0	2010	138	3124	69	3201	55	N	0	X04	X13	4	13	9
1	2010	138	3173	67	3395	66	N	0	X08	X09	8	9	1
2	2010	138	3181	72	3214	37	H	0	X02	X15	2	15	13
3	2010	138	3199	75	3256	61	H	0	W03	W14	3	14	11
4	2010	138	3207	62	3265	42	N	0	X05	X12	5	12	7

```
        plt.figure(figsize=(12, 6))
        plt.hist(df_merged_seeds_W['RankDiff'], bins=10, color='skyblue', edgecolor='black')
        plt.title('Distribution of Rank Differences')
        plt.xlabel('Rank Difference')
        plt.ylabel('Frequency')
        plt.grid(True)
        plt.show()
```

Tourney Compact Results

Men

```
        # Calculate average scores for wins and losses each year
        average_scores = MNCAATourneyCompactResults_2003.groupby(['Season'])[['WScore',
'LScore']].mean().reset_index()
        average_scores
        # Melt the DataFrame to have separate columns for win and loss scores
        average_scores_melted = average_scores.melt(id_vars=['Season'], value_vars=['WScore',
'LScore'], var_name='Outcome', value_name='Average Score')
```

```
        average_scores_melted

        plt.figure(figsize=(12, 6))
        sns.barplot(x='Season', y='Average Score', hue='Outcome', data=average_scores_melted,
palette={'WScore': 'lightseagreen', 'LScore': 'plum'})
        plt.title('Average Win and Loss Scores by Year')
        plt.xlabel('Season')
        plt.ylabel('Average Score')
        plt.legend(title='Outcome', loc='upper right')
        plt.show()
```

```
        plt.figure(figsize=(12, 6))
        plt.subplot(1, 2, 1)
        plt.hist(MNCAATourneyCompactResults_2003['WScore'], bins=20, color='lightseagreen',
alpha=0.7, label='Winning Team')
        plt.hist(MNCAATourneyCompactResults_2003['LScore'], bins=20, color='plum', alpha=0.7,
label='Losing Team')
        plt.title('Distribution of Scores')
        plt.xlabel('Score')
        plt.ylabel('Frequency')
        plt.legend()

        # Visualize the distribution of locations
        plt.subplot(1, 2, 2)
        MNCAATourneyCompactResults_2003['WLoc'].value_counts().plot(kind='bar',
color='skyblue', alpha=0.7)
        plt.title('Distribution of Locations')
        plt.xlabel('Location')
        plt.ylabel('Count')

        plt.show()
```

Finding

This historical data of team-level box scores for NCAA men tournaments starts with the 2003 season.

A more ideal normal distribution can be seen in the past scores of the winning and losing teams.

There is no 'A' in the 'Location', and all most all of them are 'N'. Which means we might don't need consider the effect by location.

Women

```
        average_scores = WNCAATourneyCompactResults_2010.groupby(['Season'])[['WScore',
'LScore']].mean().reset_index()
        average_scores
        # Melt the DataFrame to have separate columns for win and loss scores
        average_scores_melted = average_scores.melt(id_vars=['Season'], value_vars=['WScore',
'LScore'], var_name='Outcome', value_name='Average Score')
        average_scores_melted

        plt.figure(figsize=(12, 6))
        sns.barplot(x='Season', y='Average Score', hue='Outcome', data=average_scores_melted,
palette={'WScore': 'lightseagreen', 'LScore': 'plum'})
        plt.title('Average Win and Loss Scores by Year')
        plt.xlabel('Season')
        plt.ylabel('Average Score')
        plt.legend(title='Outcome', loc='upper right')
        plt.show()
```

```
        plt.figure(figsize=(12, 6))
        plt.subplot(1, 2, 1)
        plt.hist(WNCAATourneyCompactResults_2010['WScore'], bins=20, color='lightseagreen',
alpha=0.7, label='Winning Team')
        plt.hist(WNCAATourneyCompactResults_2010['LScore'], bins=20, color='plum', alpha=0.7,
label='Losing Team')
        plt.title('Distribution of Scores')
        plt.xlabel('Score')
        plt.ylabel('Frequency')
        plt.legend()

        # Visualize the distribution of locations
        plt.subplot(1, 2, 2)
        WNCAATourneyCompactResults_2010['WLoc'].value_counts().plot(kind='bar',
color='skyblue', alpha=0.7)
        plt.title('Distribution of Locations')
        plt.xlabel('Location')
        plt.ylabel('Count')

        plt.show()
```

Tourney Compact Results

```
        sns.set(style="whitegrid")
```

```
fig, axes = plt.subplots(3, 2, figsize=(15, 15))

# Men's Winning Scores
sns.histplot(MNCAATourneyCompactResults['WScore'], bins=30, kde=True, ax=axes[0, 0], color='blue')
axes[0, 0].set_title('Men\'s Winning Scores Distribution')
axes[0, 0].set_xlabel('Score')
axes[0, 0].set_ylabel('Frequency')

# Men's Losing Scores
sns.histplot(MNCAATourneyCompactResults['LScore'], bins=30, kde=True, ax=axes[0, 1], color='red')
axes[0, 1].set_title('Men\'s Losing Scores Distribution')
axes[0, 1].set_xlabel('Score')
axes[0, 1].set_ylabel('Frequency')

# Men's Number of Overtimes
sns.histplot(MNCAATourneyCompactResults['NumOT'], bins=30, kde=False, ax=axes[1, 0], color='purple')
axes[1, 0].set_title('Men\'s Number of Overtimes Distribution')
axes[1, 0].set_xlabel('Number of Overtimes')
axes[1, 0].set_ylabel('Frequency')

# Women's Winning Scores
sns.histplot(WNCAATourneyCompactResults['WScore'], bins=30, kde=True, ax=axes[1, 1], color='blue')
axes[1, 1].set_title('Women\'s Winning Scores Distribution')
axes[1, 1].set_xlabel('Score')
axes[1, 1].set_ylabel('Frequency')

# Women's Losing Scores
sns.histplot(WNCAATourneyCompactResults['LScore'], bins=30, kde=True, ax=axes[2, 0], color='red')
axes[2, 0].set_title('Women\'s Losing Scores Distribution')
axes[2, 0].set_xlabel('Score')
axes[2, 0].set_ylabel('Frequency')

# Women's Number of Overtimes
sns.histplot(WNCAATourneyCompactResults['NumOT'], bins=30, kde=False, ax=axes[2, 1], color='purple')
axes[2, 1].set_title('Women\'s Number of Overtimes Distribution')
axes[2, 1].set_xlabel('Number of Overtimes')
axes[2, 1].set_ylabel('Frequency')

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

Both men's and women's tournaments have similar patterns in terms of the distribution of scores.

Winning scores tend to be higher and more spread out compared to losing scores.

Overtime games are rare in both tournaments, indicating that most games are decided within the regular time.

```
# Plotting histograms for WScore, LScore, and NumOT
fig, axes = plt.subplots(2, 3, figsize=(18, 12))
fig.suptitle('Score and Overtime Distributions')

# Women's dataset
sns.histplot(WRegularSeasonCompactResults['WScore'], bins=30, kde=True, ax=axes[0, 0]).set_title('Women - Winning Score')
sns.histplot(WRegularSeasonCompactResults['LScore'], bins=30, kde=True, ax=axes[0, 1]).set_title('Women - Losing Score')
sns.histplot(WRegularSeasonCompactResults['NumOT'], bins=30, kde=True, ax=axes[0, 2]).set_title('Women - Number of Overtimes')

# Men's dataset
sns.histplot(MRegularSeasonCompactResults['WScore'], bins=30, kde=True, ax=axes[1, 0]).set_title('Men - Winning Score')
sns.histplot(MRegularSeasonCompactResults['LScore'], bins=30, kde=True, ax=axes[1, 1]).set_title('Men - Losing Score')
sns.histplot(MRegularSeasonCompactResults['NumOT'], bins=30, kde=True, ax=axes[1, 2]).set_title('Men - Number of Overtimes')

plt.tight_layout(rect=[0, 0.03, 1, 0.95])
```

-The x-axis represents the range of scores achieved by teams.

-The y-axis represents the frequency (count) of games that fall within each score range.

```
# Trends over seasons - Average Winning Score per Season
womens_trend = WRegularSeasonCompactResults.groupby('Season')['WScore'].mean().reset_index()
mens_trend = MRegularSeasonCompactResults.groupby('Season')['WScore'].mean().reset_index()

# Plotting trends over seasons
fig, axes = plt.subplots(1, 2, figsize=(18, 6))
fig.suptitle('Average Winning Score per Season')

sns.lineplot(data=womens_trend, x='Season', y='WScore', ax=axes[0]).set_title('Women - Average Winning Score per Season')
sns.lineplot(data=mens_trend, x='Season', y='WScore', ax=axes[1]).set_title('Men - Average Winning Score per Season')

plt.xticks(rotation=90)
```

```
    plt.tight_layout(rect=[0, 0.03, 1, 0.95])
    plt.show()
```

MODELING

```
import pandas as pd
import numpy as np
import matplotlib.pylab as plt
import matplotlib as mpl
from matplotlib.patches import Circle, Rectangle, Arc
import seaborn as sns

from sklearn.metrics import accuracy_score, log_loss
import xgboost as xgb
from sklearn.model_selection import GroupKFold

plt.style.use("fivethirtyeight")
mypal = plt.rcParams["axes.prop_cycle"].by_key()["color"]
```

```
!ls -GFlash ../input/march-machine-learning-mania-2024/
```

```
total 144M
   0 drwxr-xr-x 2 nobody    0 May 13 08:49 ./
4.0K drwxr-xr-x 3 root   4.0K Jun 12 12:38 ../
4.0K -rw-r--r-- 1 nobody 1.4K May 13 08:49 2024_tourney_seeds.csv
 12K -rw-r--r-- 1 nobody 9.1K May 13 08:49 Cities.csv
4.0K -rw-r--r-- 1 nobody 1.7K May 13 08:49 Conferences.csv
168K -rw-r--r-- 1 nobody 168K May 13 08:49 MConferenceTourneyGames.csv
2.5M -rw-r--r-- 1 nobody 2.5M May 13 08:49 MGameCities.csv
111M -rw-r--r-- 1 nobody 111M May 13 08:49 MMasseyOrdinals_thruSeason2024_day128.csv
 72K -rw-r--r-- 1 nobody  72K May 13 08:49 MNCAATourneyCompactResults.csv
132K -rw-r--r-- 1 nobody 129K May 13 08:49 MNCAATourneyDetailedResults.csv
 16K -rw-r--r-- 1 nobody  16K May 13 08:49 MNCAATourneySeedRoundSlots.csv
 40K -rw-r--r-- 1 nobody  38K May 13 08:49 MNCAATourneySeeds.csv
 52K -rw-r--r-- 1 nobody  50K May 13 08:49 MNCAATourneySlots.csv
5.3M -rw-r--r-- 1 nobody 5.3M May 13 08:49 MRegularSeasonCompactResults.csv
 11M -rw-r--r-- 1 nobody  11M May 13 08:49 MRegularSeasonDetailedResults.csv
4.0K -rw-r--r-- 1 nobody 1.8K May 13 08:49 MSeasons.csv
 60K -rw-r--r-- 1 nobody  59K May 13 08:49 MSecondaryTourneyCompactResults.csv
 28K -rw-r--r-- 1 nobody  27K May 13 08:49 MSecondaryTourneyTeams.csv
388K -rw-r--r-- 1 nobody 385K May 13 08:49 MTeamCoaches.csv
220K -rw-r--r-- 1 nobody 220K May 13 08:49 MTeamConferences.csv
```

```
 24K -rw-r--r-- 1 nobody  23K May 13 08:49 MTeamSpellings.csv
 12K -rw-r--r-- 1 nobody 9.8K May 13 08:49 MTeams.csv
2.4M -rw-r--r-- 1 nobody 2.4M May 13 08:49 WGameCities.csv
 48K -rw-r--r-- 1 nobody  47K May 13 08:49 WNCAATourneyCompactResults.csv
 84K -rw-r--r-- 1 nobody  82K May 13 08:49 WNCAATourneyDetailedResults.csv
 28K -rw-r--r-- 1 nobody  25K May 13 08:49 WNCAATourneySeeds.csv
 36K -rw-r--r-- 1 nobody  34K May 13 08:49 WNCAATourneySlots.csv
3.7M -rw-r--r-- 1 nobody 3.7M May 13 08:49 WRegularSeasonCompactResults.csv
7.3M -rw-r--r-- 1 nobody 7.3M May 13 08:49 WRegularSeasonDetailedResults.csv
4.0K -rw-r--r-- 1 nobody 1.4K May 13 08:49 WSeasons.csv
156K -rw-r--r-- 1 nobody 154K May 13 08:49 WTeamConferences.csv
 24K -rw-r--r-- 1 nobody  22K May 13 08:49 WTeamSpellings.csv
8.0K -rw-r--r-- 1 nobody 6.1K May 13 08:49 WTeams.csv
4.0K -rw-r--r-- 1 nobody 2.1K May 13 08:49 sample_submission.csv
```

Files we are interested in:

- **MRegularSeasonCompactResults.csv & WMRegularSeasonCompactResults.csv**

  All game results from the regular season.

- **MNCAATourneyCompactResults.csv & WNCAATourneyCompactResults.csv**

  All game results from past tournaments.

- **MNCAATourneySeeds.csv & MNCAATourneySeeds.csv**

  The seeding for the tournaments

- 2024_tourney_seeds.csv

```
File that will be updated with 2024 seeds once released (2023 seeds
prior to that)
```

```
DATA_PATH = "../input/march-machine-learning-mania-2024/"
```

```
df_seeds = pd.concat(
    [
        pd.read_csv(DATA_PATH + "MNCAATourneySeeds.csv").assign(League="M"),
        pd.read_csv(DATA_PATH + "WNCAATourneySeeds.csv").assign(League="W"),
    ],
).reset_index(drop=True)

df_season_results = pd.concat(
    [
        pd.read_csv(DATA_PATH +
"MRegularSeasonCompactResults.csv").assign(League="M"),
```

```
            pd.read_csv(DATA_PATH +
"WRegularSeasonCompactResults.csv").assign(League="W"),
        ]
    ).reset_index(drop=True)

    df_tourney_results = pd.concat(
        [
            pd.read_csv(DATA_PATH +
"MNCAATourneyCompactResults.csv").assign(League="M"),
            pd.read_csv(DATA_PATH +
"WNCAATourneyCompactResults.csv").assign(League="W"),
        ]
    ).reset_index(drop=True)
```

```
Season = 2024
filtered_wins =  df_seeds[ df_seeds['Season'] == Season]
filtered_wins
```

	Season	Seed	TeamID	League
2490	2024	W01	1163	M
2491	2024	W02	1235	M
2492	2024	W03	1228	M
2493	2024	W04	1120	M
2494	2024	W05	1361	M
...	...	...	...	...
4229	2024	Z12b	3435	W
4230	2024	Z13	3267	W
4231	2024	Z14	3238	W
4232	2024	Z15	3263	W
4233	2024	Z16	3394	W

136 rows × 4 columns

Creating Team Season Results

- We the the data from the existing format with 1 row per game
- New format has 1 row for each team's game - win or loss.
- This data can be aggregated for season metrics

```
df_team_season_results = pd.concat(
    [
        df_season_results[["Season", "League", "WTeamID", "DayNum", "WScore",
"LScore"]]
        .assign(GameResult="W")
        .rename(
            columns={"WTeamID": "TeamID", "WScore": "TeamScore", "LScore":
"OppScore"}
        ),
        df_season_results[["Season", "League", "LTeamID", "DayNum", "WScore",
"LScore"]]
        .assign(GameResult="L")
        .rename(
            columns={"LTeamID": "TeamID", "LScore": "TeamScore", "WScore":
"OppScore"}
        ),
    ]
).reset_index(drop=True)
```

Create Season Features

- add some features to this data like the score differential

```
# Score Differential
df_team_season_results["ScoreDiff"] = (
    df_team_season_results["TeamScore"] - df_team_season_results["OppScore"]
)
df_team_season_results["Win"] = (df_team_season_results["GameResult"] ==
"W").astype(
    "int"
)
```

```
df_team_season_results.sample(10, random_state=529)
```

	Season	League	TeamID	DayNum	TeamScore	OppScore	GameResult	ScoreDiff	Win
493232	2022	M	1444	96	64	77	L	-13	0
71811	2002	M	1281	79	74	50	W	24	1
555548	2008	W	3276	115	67	69	L	-2	0
84226	2005	M	1393	40	86	56	W	30	1
129439	2014	M	1368	8	63	62	W	1	1

	Season	League	TeamID	DayNum	TeamScore	OppScore	GameResult	ScoreDiff	Win
364900	1996	M	1148	85	46	116	L	-70	0
587485	2015	W	3394	18	66	68	L	-2	0
397470	2003	M	1266	129	76	83	L	-7	0
99374	2008	M	1177	47	93	88	W	5	1
532105	2003	W	3414	130	50	83	L	-33	0

Aggregate for team's total season stats

```
# Aggregate the data
team_season_agg = (
    df_team_season_results.groupby(["Season", "TeamID", "League"])
    .agg(
        AvgScoreDiff=("ScoreDiff", "mean"),
        MedianScoreDiff=("ScoreDiff", "median"),
        MinScoreDiff=("ScoreDiff", "min"),
        MaxScoreDiff=("ScoreDiff", "max"),
        Wins=("Win", "sum"),
        Losses=("GameResult", lambda x: (x == "L").sum()),
        WinPercentage=("Win", "mean"),
    )
    .reset_index()
)
```

```
team_season_agg.head()
```

	Season	TeamID	League	AvgScoreDiff	MedianScoreDiff	MinScoreDiff	MaxScoreDiff	Wins	Losses	WinPer
0	1985	1102	M	-5.791667	-5.5	-41	29	5	19	0.20833
1	1985	1103	M	-3.043478	-2.0	-22	16	9	14	0.39130
2	1985	1104	M	7.800000	6.5	-12	25	21	9	0.70000
3	1985	1106	M	-3.791667	-1.5	-35	28	10	14	0.41666
4	1985	1108	M	7.960000	4.0	-15	35	19	6	0.76000

```
df_seeds["ChalkSeed"] = (
    df_seeds["Seed"].str.replace("a", "").str.replace("b",
"").str[1:].astype("int")
)
```

```
team_season_agg = team_season_agg.merge(
    df_seeds, on=["Season", "TeamID", "League"], how="left"
)
```

```
team_season_agg.shape, df_seeds.shape
```

```
((22150, 12), (4234, 5))
```

Tournament Results Aggregation

```
df_team_tourney_results = pd.concat(
    [
        df_tourney_results[
            ["Season", "League", "WTeamID", "LTeamID", "WScore", "LScore"]
        ]
        .assign(GameResult="W")
        .rename(
            columns={
                "WTeamID": "TeamID",
                "LTeamID": "OppTeamID",
                "WScore": "TeamScore",
                "LScore": "OppScore",
            }
        ),
        df_tourney_results[
            ["Season", "League", "LTeamID", "WTeamID", "LScore", "WScore"]
        ]
        .assign(GameResult="L")
        .rename(
            columns={
                "LTeamID": "TeamID",
                "WTeamID": "OppTeamID",
                "LScore": "TeamScore",
                "WScore": "OppScore",
            }
        ),
    ]
).reset_index(drop=True)

df_team_tourney_results["Win"] = (df_team_tourney_results["GameResult"] ==
"W").astype(
```

```
            "int"
        )
```

```
        df_team_tourney_results.head()
```

	Season	League	TeamID	OppTeamID	TeamScore	OppScore	GameResult	Win
0	1985	M	1116	1234	63	54	W	1
1	1985	M	1120	1345	59	58	W	1
2	1985	M	1207	1250	68	43	W	1
3	1985	M	1229	1425	58	55	W	1
4	1985	M	1242	1325	49	38	W	1

Tourney Dataset with Features

- merge our team's regular season features with our tourney dataframe.
- This gives us the data format that we will use to train our model.
- target column is the "Winner" and the features are the regular season stats.

```
        df_historic_tourney_features = df_team_tourney_results.merge(
            team_season_agg[
                ["Season", "League", "TeamID", "WinPercentage", "MedianScoreDiff",
"ChalkSeed"]
            ],
            on=["Season", "League", "TeamID"],
            how="left",
        ).merge(
            team_season_agg[
                ["Season", "League", "TeamID", "WinPercentage", "MedianScoreDiff",
"ChalkSeed"]
            ].rename(
                columns={
                    "TeamID": "OppTeamID",
                    "WinPercentage": "OppWinPercentage",
                    "MedianScoreDiff": "OppMedianScoreDiff",
                    "ChalkSeed": "OppChalkSeed",
                }
            ),
            on=["Season", "League", "OppTeamID"],
        )
```

```
df_historic_tourney_features.head()
```

	Season	League	TeamID	OppTeamID	TeamScore	OppScore	GameResult	Win	WinPercentage	MedianScor
0	1985	M	1116	1234	63	54	W	1	0.636364	5.0
1	1985	M	1120	1345	59	58	W	1	0.620690	2.0
2	1985	M	1207	1250	68	43	W	1	0.925926	14.0
3	1985	M	1229	1425	58	55	W	1	0.740741	6.0
4	1985	M	1242	1325	49	38	W	1	0.766667	5.5

```
df_historic_tourney_features.columns
```

```
Index(['Season', 'League', 'TeamID', 'OppTeamID', 'TeamScore', 'OppScore',
       'GameResult', 'Win', 'WinPercentage', 'MedianScoreDiff', 'ChalkSeed',
       'OppWinPercentage', 'OppMedianScoreDiff', 'OppChalkSeed'],
      dtype='object')
```

```
df_historic_tourney_features["WinPctDiff"] = (
    df_historic_tourney_features["WinPercentage"]
    - df_historic_tourney_features["OppWinPercentage"]
)

df_historic_tourney_features["ChalkSeedDiff"] = (
    df_historic_tourney_features["ChalkSeed"]
    - df_historic_tourney_features["OppChalkSeed"]
)

df_historic_tourney_features["MedianScoreDiffDiff"] = (
    df_historic_tourney_features["MedianScoreDiff"]
    - df_historic_tourney_features["OppMedianScoreDiff"]
)
```

```
df_historic_tourney_features
```

	Season	League	TeamID	OppTeamID	TeamScore	OppScore	GameResult	Win	WinPercentage	MedianS
0	1985	M	1116	1234	63	54	W	1	0.636364	5.0

	Season	League	TeamID	OppTeamID	TeamScore	OppScore	GameResult	Win	WinPercentage	MedianS
1	1985	M	1120	1345	59	58	W	1	0.620690	2.0
2	1985	M	1207	1250	68	43	W	1	0.925926	14.0
3	1985	M	1229	1425	58	55	W	1	0.740741	6.0
4	1985	M	1242	1325	49	38	W	1	0.766667	5.5
...	...	...	...	...	...	...	...	...	...	...
8063	2023	W	3268	3376	75	86	L	0	0.806452	11.0
8064	2023	W	3326	3439	74	84	L	0	0.781250	12.0
8065	2023	W	3376	3234	73	77	L	0	1.000000	28.0
8066	2023	W	3439	3261	72	79	L	0	0.870968	13.0
8067	2023	W	3234	3261	85	102	L	0	0.812500	13.5

8068 rows × 17 columns

```
        TeamID = 1116
        filtered_wins =  df_historic_tourney_features[
df_historic_tourney_features['TeamID'] == TeamID]
        filtered_wins
```

	Season	League	TeamID	OppTeamID	TeamScore	OppScore	GameResult	Win	WinPercentage	MedianS
0	1985	M	1116	1234	63	54	W	1	0.636364	5.0
253	1989	M	1116	1258	120	101	W	1	0.800000	16.5
315	1990	M	1116	1343	68	64	W	1	0.866667	15.0
347	1990	M	1116	1173	86	84	W	1	0.866667	15.0
363	1990	M	1116	1314	96	73	W	1	0.866667	15.0
371	1990	M	1116	1400	88	85	W	1	0.866667	15.0
396	1991	M	1116	1209	117	76	W	1	0.909091	20.0
419	1991	M	1116	1113	97	90	W	1	0.909091	20.0
426	1991	M	1116	1104	93	70	W	1	0.909091	20.0
442	1992	M	1116	1293	80	69	W	1	0.758621	11.0
504	1993	M	1116	1221	94	64	W	1	0.714286	8.0
536	1993	M	1116	1385	80	74	W	1	0.714286	8.0
584	1994	M	1116	1299	94	79	W	1	0.888889	13.0
608	1994	M	1116	1207	85	73	W	1	0.888889	13.0
619	1994	M	1116	1409	103	84	W	1	0.888889	13.0
625	1994	M	1116	1276	76	68	W	1	0.888889	13.0

	Season	League	TeamID	OppTeamID	TeamScore	OppScore	GameResult	Win	WinPercentage	MedianS
627	1994	M	1116	1112	91	82	W	1	0.888889	13.0
629	1994	M	1116	1181	76	72	W	1	0.888889	13.0
646	1995	M	1116	1411	79	78	W	1	0.812500	7.5
670	1995	M	1116	1393	96	94	W	1	0.812500	7.5
682	1995	M	1116	1272	96	91	W	1	0.812500	7.5
688	1995	M	1116	1438	68	61	W	1	0.812500	7.5
690	1995	M	1116	1314	75	68	W	1	0.812500	7.5
693	1996	M	1116	1336	86	80	W	1	0.600000	7.0
725	1996	M	1116	1266	65	56	W	1	0.600000	7.0
820	1998	M	1116	1304	74	65	W	1	0.766667	7.5
882	1999	M	1116	1373	94	80	W	1	0.687500	5.0
1473	2008	M	1116	1231	86	72	W	1	0.666667	6.0
1921	2015	M	1116	1459	56	53	W	1	0.764706	7.0
2070	2017	M	1116	1371	77	71	W	1	0.735294	8.5
2255	2021	M	1116	1159	85	68	W	1	0.785714	11.5
2286	2021	M	1116	1403	68	66	W	1	0.785714	11.5
2302	2021	M	1116	1331	72	70	W	1	0.785714	11.5
2321	2022	M	1116	1436	75	71	W	1	0.757576	8.0
2353	2022	M	1116	1308	53	48	W	1	0.757576	8.0
2369	2022	M	1116	1211	74	68	W	1	0.757576	8.0
2389	2023	M	1116	1228	73	63	W	1	0.606061	6.0
2421	2023	M	1116	1242	72	71	W	1	0.606061	6.0
4073	1985	M	1116	1385	65	68	L	0	0.636364	5.0
4254	1988	M	1116	1437	74	82	L	0	0.724138	10.0
4321	1989	M	1116	1257	84	93	L	0	0.800000	16.5
4409	1990	M	1116	1181	83	97	L	0	0.866667	15.0
4468	1991	M	1116	1242	81	93	L	0	0.909091	20.0
4511	1992	M	1116	1272	80	82	L	0	0.758621	11.0
4592	1993	M	1116	1314	74	80	L	0	0.714286	8.0
4726	1995	M	1116	1417	78	89	L	0	0.812500	7.5
4777	1996	M	1116	1269	63	79	L	0	0.600000	7.0
4890	1998	M	1116	1428	69	75	L	0	0.766667	7.5
4952	1999	M	1116	1234	72	82	L	0	0.687500	5.0
5001	2000	M	1116	1274	71	75	L	0	0.575758	7.0

	Season	League	TeamID	OppTeamID	TeamScore	OppScore	GameResult	Win	WinPercentage	MedianS
5046	2001	M	1116	1207	61	63	L	0	0.666667	5.5
5381	2006	M	1116	1137	55	59	L	0	0.709677	6.0
5454	2007	M	1116	1425	60	77	L	0	0.617647	8.0
5534	2008	M	1116	1314	77	108	L	0	0.666667	6.0
5989	2015	M	1116	1314	78	87	L	0	0.764706	7.0
6132	2017	M	1116	1314	65	72	L	0	0.735294	8.5
6172	2018	M	1116	1139	62	79	L	0	0.676471	4.5
6344	2021	M	1116	1124	72	81	L	0	0.785714	11.5
6411	2022	M	1116	1181	69	78	L	0	0.757576	8.0
6470	2023	M	1116	1163	65	88	L	0	0.606061	6.0

```
import warnings
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import math

# Suppress all warnings
warnings.filterwarnings('ignore')

# Replace infinite values with NaN
df_historic_tourney_features.replace([np.inf, -np.inf], np.nan, inplace=True)


# Get numerical features
numerical_features = df_historic_tourney_features.select_dtypes(include=
['float64', 'int64']).columns

# Calculate the number of rows and columns for subplots
num_features = len(numerical_features)
num_cols = 3
num_rows = math.ceil(num_features / num_cols)

plt.figure(figsize=(15, num_rows * 5))
for i, col in enumerate(numerical_features):
    plt.subplot(num_rows, num_cols, i + 1)
    sns.histplot(df_historic_tourney_features[col], kde=True)
    plt.title(col)

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

Baseline - Higher Seed Wins

To do this we will simply score the accuracy on historic tournaments assuming the higher seed always wins.

```
df_historic_tourney_features["BaselinePred"] = (
    df_historic_tourney_features["ChalkSeed"]
    < df_historic_tourney_features["OppChalkSeed"]
)

df_historic_tourney_features.loc[
    df_historic_tourney_features["ChalkSeed"]
    == df_historic_tourney_features["OppChalkSeed"],
    "BaselinePred",
] = (
    df_historic_tourney_features["WinPercentage"]
    > df_historic_tourney_features["OppWinPercentage"]
)
```

```
from sklearn.metrics import accuracy_score, log_loss

cv_scores_baseline = []
for season in df_historic_tourney_features["Season"].unique():
    pred = df_historic_tourney_features.query("Season == @season")[
        "BaselinePred"
    ].astype("int")
    y = df_historic_tourney_features.query("Season == @season")["Win"]
    score = accuracy_score(y, pred)
    score_ll = log_loss(y, pred)
    cv_scores_baseline.append(score)
    print(f"Holdout season {season} - Accuracy {score:0.4f} Log Loss {score_ll:0.4f}")

print(f"Baseline accuracy {np.mean(cv_scores_baseline):0.4f}")
```

```
Holdout season 1985 - Accuracy 0.7143 Log Loss 10.2982
Holdout season 1986 - Accuracy 0.7143 Log Loss 10.2982
Holdout season 1987 - Accuracy 0.6984 Log Loss 10.8703
Holdout season 1988 - Accuracy 0.7143 Log Loss 10.2982
Holdout season 1989 - Accuracy 0.6667 Log Loss 12.0146
Holdout season 1990 - Accuracy 0.6825 Log Loss 11.4424
Holdout season 1991 - Accuracy 0.7460 Log Loss 9.1539
```

```
Holdout season 1992 - Accuracy 0.7619 Log Loss 8.5818
Holdout season 1993 - Accuracy 0.7937 Log Loss 7.4376
Holdout season 1994 - Accuracy 0.7143 Log Loss 10.2982
Holdout season 1995 - Accuracy 0.7619 Log Loss 8.5818
Holdout season 1996 - Accuracy 0.7460 Log Loss 9.1539
Holdout season 1997 - Accuracy 0.7302 Log Loss 9.7261
Holdout season 1998 - Accuracy 0.7143 Log Loss 10.2982
Holdout season 1999 - Accuracy 0.7222 Log Loss 10.0121
Holdout season 2000 - Accuracy 0.7302 Log Loss 9.7261
Holdout season 2001 - Accuracy 0.7047 Log Loss 10.6428
Holdout season 2002 - Accuracy 0.7480 Log Loss 9.0819
Holdout season 2003 - Accuracy 0.7402 Log Loss 9.3657
Holdout season 2004 - Accuracy 0.7244 Log Loss 9.9333
Holdout season 2005 - Accuracy 0.7165 Log Loss 10.2171
Holdout season 2006 - Accuracy 0.7480 Log Loss 9.0819
Holdout season 2007 - Accuracy 0.7717 Log Loss 8.2304
Holdout season 2008 - Accuracy 0.8031 Log Loss 7.0952
Holdout season 2009 - Accuracy 0.7402 Log Loss 9.3657
Holdout season 2010 - Accuracy 0.7402 Log Loss 9.3657
Holdout season 2011 - Accuracy 0.7000 Log Loss 10.8131
Holdout season 2012 - Accuracy 0.7923 Log Loss 7.4860
Holdout season 2013 - Accuracy 0.7154 Log Loss 10.2586
Holdout season 2014 - Accuracy 0.7000 Log Loss 10.8131
Holdout season 2015 - Accuracy 0.7923 Log Loss 7.4860
Holdout season 2016 - Accuracy 0.6846 Log Loss 11.3676
Holdout season 2017 - Accuracy 0.7769 Log Loss 8.0405
Holdout season 2018 - Accuracy 0.7000 Log Loss 10.8131
Holdout season 2019 - Accuracy 0.7538 Log Loss 8.8723
Holdout season 2021 - Accuracy 0.7519 Log Loss 8.9411
Holdout season 2022 - Accuracy 0.7015 Log Loss 10.7593
Holdout season 2023 - Accuracy 0.7164 Log Loss 10.2213
Baseline accuracy 0.7325
```

XGBoost Model

```
FEATURES = [
       "WinPercentage",
        "MedianScoreDiff",
        "ChalkSeed",
       "OppWinPercentage",
        "OppMedianScoreDiff",
        "OppChalkSeed",
    "WinPctDiff",
        "ChalkSeedDiff"
]
TARGET = "Win"
```

```
    X = df_historic_tourney_features[FEATURES]
    y = df_historic_tourney_features[TARGET]
    groups = df_historic_tourney_features["Season"]
    seasons = df_historic_tourney_features["Season"].unique()

    # Setup cross-validation
    gkf = GroupKFold(n_splits=df_historic_tourney_features["Season"].nunique())
    cv_results = []
    models = []

    season_idx = 0
    for train_index, test_index in gkf.split(X, y, groups):
        X_train, X_test = X.iloc[train_index], X.iloc[test_index]
        y_train, y_test = y.iloc[train_index], y.iloc[test_index]

        # Prepare the model
        model = xgb.XGBRegressor(
            eval_metric="logloss",
            n_estimators=1_000,
            learning_rate=0.001,
        )
        holdout_season = seasons[season_idx]
        print(f"Holdout Season: {holdout_season}")
        # Train the model
        model.fit(X_train, y_train, eval_set=[(X_test, y_test)], verbose=100)

        # Predict on the test set
        y_pred = model.predict(X_test)
        score_ll = log_loss(y_test, y_pred)
        y_pred = y_pred > 0.5
        # Evaluate the model
        accuracy = accuracy_score(y_test, y_pred)
        cv_results.append(accuracy)
        season_idx += 1
        print(f"Season {holdout_season}: {accuracy} {score_ll}")
        models.append(model)
    # Print the average accuracy across all folds
    print("Average CV Accuracy:", np.mean(cv_results))
```

```
Holdout Season: 1985
[0] validation_0-logloss:0.69289
[100]   validation_0-logloss:0.66959
[200]   validation_0-logloss:0.65069
[300]   validation_0-logloss:0.63538
[400]   validation_0-logloss:0.62238
[500]   validation_0-logloss:0.61101
[600]   validation_0-logloss:0.60165
[700]   validation_0-logloss:0.59384
[800]   validation_0-logloss:0.58727
```

```
[900]   validation_0-logloss:0.58168
[999]   validation_0-logloss:0.57783
Season 1985: 0.7388059701492538 0.5778297953897061
Holdout Season: 1986
[0] validation_0-logloss:0.69288
[100]   validation_0-logloss:0.66882
[200]   validation_0-logloss:0.64906
[300]   validation_0-logloss:0.63319
[400]   validation_0-logloss:0.61992
[500]   validation_0-logloss:0.60787
[600]   validation_0-logloss:0.59805
[700]   validation_0-logloss:0.59025
[800]   validation_0-logloss:0.58413
[900]   validation_0-logloss:0.57886
[999]   validation_0-logloss:0.57464
Season 1986: 0.6902985074626866 0.5746410916790947
Holdout Season: 1987
[0] validation_0-logloss:0.69279
[100]   validation_0-logloss:0.65966
[200]   validation_0-logloss:0.63105
[300]   validation_0-logloss:0.60721
[400]   validation_0-logloss:0.58759
[500]   validation_0-logloss:0.57118
[600]   validation_0-logloss:0.55699
[700]   validation_0-logloss:0.54410
[800]   validation_0-logloss:0.53302
[900]   validation_0-logloss:0.52317
[999]   validation_0-logloss:0.51405
Season 1987: 0.75 0.5140454237560299
Holdout Season: 1988
[0] validation_0-logloss:0.69285
[100]   validation_0-logloss:0.66623
[200]   validation_0-logloss:0.64427
[300]   validation_0-logloss:0.62591
[400]   validation_0-logloss:0.61106
[500]   validation_0-logloss:0.59889
[600]   validation_0-logloss:0.58833
[700]   validation_0-logloss:0.57878
[800]   validation_0-logloss:0.57083
[900]   validation_0-logloss:0.56448
[999]   validation_0-logloss:0.55938
Season 1988: 0.7038461538461539 0.5593773261339369
Holdout Season: 1989
[0] validation_0-logloss:0.69280
[100]   validation_0-logloss:0.66095
[200]   validation_0-logloss:0.63427
[300]   validation_0-logloss:0.61216
[400]   validation_0-logloss:0.59319
[500]   validation_0-logloss:0.57735
[600]   validation_0-logloss:0.56368
[700]   validation_0-logloss:0.55191
```

```
[800]   validation_0-logloss:0.54149
[900]   validation_0-logloss:0.53248
[999]   validation_0-logloss:0.52479
Season 1989: 0.7884615384615384 0.5247887917788466
Holdout Season: 1990
[0] validation_0-logloss:0.69287
[100]   validation_0-logloss:0.66713
[200]   validation_0-logloss:0.64599
[300]   validation_0-logloss:0.62820
[400]   validation_0-logloss:0.61317
[500]   validation_0-logloss:0.60001
[600]   validation_0-logloss:0.58916
[700]   validation_0-logloss:0.58034
[800]   validation_0-logloss:0.57328
[900]   validation_0-logloss:0.56775
[999]   validation_0-logloss:0.56280
Season 1990: 0.7269230769230769 0.5628005594382917
Holdout Season: 1991
[0] validation_0-logloss:0.69279
[100]   validation_0-logloss:0.66054
[200]   validation_0-logloss:0.63394
[300]   validation_0-logloss:0.61099
[400]   validation_0-logloss:0.59190
[500]   validation_0-logloss:0.57585
[600]   validation_0-logloss:0.56179
[700]   validation_0-logloss:0.54953
[800]   validation_0-logloss:0.53909
[900]   validation_0-logloss:0.52938
[999]   validation_0-logloss:0.52094
Season 1991: 0.7807692307692308 0.5209394005663719
Holdout Season: 1992
[0] validation_0-logloss:0.69286
[100]   validation_0-logloss:0.66644
[200]   validation_0-logloss:0.64507
[300]   validation_0-logloss:0.62768
[400]   validation_0-logloss:0.61345
[500]   validation_0-logloss:0.60177
[600]   validation_0-logloss:0.59163
[700]   validation_0-logloss:0.58264
[800]   validation_0-logloss:0.57539
[900]   validation_0-logloss:0.56918
[999]   validation_0-logloss:0.56388
Season 1992: 0.6961538461538461 0.5638759436656668
Holdout Season: 1993
[0] validation_0-logloss:0.69286
[100]   validation_0-logloss:0.66692
[200]   validation_0-logloss:0.64527
[300]   validation_0-logloss:0.62694
[400]   validation_0-logloss:0.61169
[500]   validation_0-logloss:0.59876
[600]   validation_0-logloss:0.58816
```

```
[700]   validation_0-logloss:0.57937
[800]   validation_0-logloss:0.57203
[900]   validation_0-logloss:0.56525
[999]   validation_0-logloss:0.55973
Season 1993: 0.7038461538461539 0.5597309496798991
Holdout Season: 1994
[0] validation_0-logloss:0.69284
[100]   validation_0-logloss:0.66468
[200]   validation_0-logloss:0.63918
[300]   validation_0-logloss:0.61844
[400]   validation_0-logloss:0.60206
[500]   validation_0-logloss:0.58784
[600]   validation_0-logloss:0.57519
[700]   validation_0-logloss:0.56416
[800]   validation_0-logloss:0.55484
[900]   validation_0-logloss:0.54798
[999]   validation_0-logloss:0.54230
Season 1994: 0.75 0.5423035564199855
Holdout Season: 1995
[0] validation_0-logloss:0.69284
[100]   validation_0-logloss:0.66487
[200]   validation_0-logloss:0.64208
[300]   validation_0-logloss:0.62291
[400]   validation_0-logloss:0.60702
[500]   validation_0-logloss:0.59393
[600]   validation_0-logloss:0.58276
[700]   validation_0-logloss:0.57300
[800]   validation_0-logloss:0.56453
[900]   validation_0-logloss:0.55728
[999]   validation_0-logloss:0.55142
Season 1995: 0.7038461538461539 0.5514155650375784
Holdout Season: 1996
[0] validation_0-logloss:0.69286
[100]   validation_0-logloss:0.66706
[200]   validation_0-logloss:0.64582
[300]   validation_0-logloss:0.62855
[400]   validation_0-logloss:0.61431
[500]   validation_0-logloss:0.60184
[600]   validation_0-logloss:0.59106
[700]   validation_0-logloss:0.58220
[800]   validation_0-logloss:0.57470
[900]   validation_0-logloss:0.56827
[999]   validation_0-logloss:0.56252
Season 1996: 0.748062015503876 0.5625201829225002
Holdout Season: 1997
[0] validation_0-logloss:0.69281
[100]   validation_0-logloss:0.66213
[200]   validation_0-logloss:0.63632
[300]   validation_0-logloss:0.61529
[400]   validation_0-logloss:0.59747
[500]   validation_0-logloss:0.58238
```

```
[600]   validation_0-logloss:0.56959
[700]   validation_0-logloss:0.55879
[800]   validation_0-logloss:0.54948
[900]   validation_0-logloss:0.54136
[999]   validation_0-logloss:0.53445
Season 1997: 0.6968503937007874 0.5344523644145192
Holdout Season: 1998
[0] validation_0-logloss:0.69283
[100]   validation_0-logloss:0.66432
[200]   validation_0-logloss:0.64074
[300]   validation_0-logloss:0.62166
[400]   validation_0-logloss:0.60571
[500]   validation_0-logloss:0.59174
[600]   validation_0-logloss:0.57976
[700]   validation_0-logloss:0.57015
[800]   validation_0-logloss:0.56236
[900]   validation_0-logloss:0.55623
[999]   validation_0-logloss:0.55118
Season 1998: 0.7007874015748031 0.5511808710089903
Holdout Season: 1999
[0] validation_0-logloss:0.69283
[100]   validation_0-logloss:0.66377
[200]   validation_0-logloss:0.63985
[300]   validation_0-logloss:0.61951
[400]   validation_0-logloss:0.60174
[500]   validation_0-logloss:0.58692
[600]   validation_0-logloss:0.57416
[700]   validation_0-logloss:0.56353
[800]   validation_0-logloss:0.55441
[900]   validation_0-logloss:0.54638
[999]   validation_0-logloss:0.53948
Season 1999: 0.7598425196850394 0.5394789986657479
Holdout Season: 2000
[0] validation_0-logloss:0.69282
[100]   validation_0-logloss:0.66268
[200]   validation_0-logloss:0.63775
[300]   validation_0-logloss:0.61673
[400]   validation_0-logloss:0.59911
[500]   validation_0-logloss:0.58420
[600]   validation_0-logloss:0.57191
[700]   validation_0-logloss:0.56153
[800]   validation_0-logloss:0.55283
[900]   validation_0-logloss:0.54518
[999]   validation_0-logloss:0.53916
Season 2000: 0.7480314960629921 0.5391647286090452
Holdout Season: 2001
[0] validation_0-logloss:0.69281
[100]   validation_0-logloss:0.66195
[200]   validation_0-logloss:0.63569
[300]   validation_0-logloss:0.61374
[400]   validation_0-logloss:0.59482
```

```
[500]   validation_0-logloss:0.57860
[600]   validation_0-logloss:0.56502
[700]   validation_0-logloss:0.55295
[800]   validation_0-logloss:0.54328
[900]   validation_0-logloss:0.53569
[999]   validation_0-logloss:0.52856
Season 2001: 0.7362204724409449 0.5285587446199357
Holdout Season: 2002
[0] validation_0-logloss:0.69282
[100]   validation_0-logloss:0.66278
[200]   validation_0-logloss:0.63832
[300]   validation_0-logloss:0.61846
[400]   validation_0-logloss:0.60142
[500]   validation_0-logloss:0.58682
[600]   validation_0-logloss:0.57428
[700]   validation_0-logloss:0.56378
[800]   validation_0-logloss:0.55461
[900]   validation_0-logloss:0.54673
[999]   validation_0-logloss:0.53948
Season 2002: 0.7598425196850394 0.5394769853566181
Holdout Season: 2003
[0] validation_0-logloss:0.69278
[100]   validation_0-logloss:0.65820
[200]   validation_0-logloss:0.62854
[300]   validation_0-logloss:0.60321
[400]   validation_0-logloss:0.58185
[500]   validation_0-logloss:0.56368
[600]   validation_0-logloss:0.54842
[700]   validation_0-logloss:0.53497
[800]   validation_0-logloss:0.52316
[900]   validation_0-logloss:0.51282
[999]   validation_0-logloss:0.50337
Season 2003: 0.8110236220472441 0.5033689635420715
Holdout Season: 2004
[0] validation_0-logloss:0.69279
[100]   validation_0-logloss:0.66004
[200]   validation_0-logloss:0.63298
[300]   validation_0-logloss:0.61002
[400]   validation_0-logloss:0.59091
[500]   validation_0-logloss:0.57495
[600]   validation_0-logloss:0.56084
[700]   validation_0-logloss:0.54809
[800]   validation_0-logloss:0.53646
[900]   validation_0-logloss:0.52657
[999]   validation_0-logloss:0.51806
Season 2004: 0.7598425196850394 0.518058388778077
Holdout Season: 2005
[0] validation_0-logloss:0.69285
[100]   validation_0-logloss:0.66593
[200]   validation_0-logloss:0.64316
[300]   validation_0-logloss:0.62386
```

```
[400]	validation_0-logloss:0.60806
[500]	validation_0-logloss:0.59532
[600]	validation_0-logloss:0.58458
[700]	validation_0-logloss:0.57532
[800]	validation_0-logloss:0.56777
[900]	validation_0-logloss:0.56184
[999]	validation_0-logloss:0.55686
Season 2005: 0.7204724409448819 0.5568630854801879
Holdout Season: 2006
[0]	validation_0-logloss:0.69279
[100]	validation_0-logloss:0.66073
[200]	validation_0-logloss:0.63402
[300]	validation_0-logloss:0.61214
[400]	validation_0-logloss:0.59401
[500]	validation_0-logloss:0.57845
[600]	validation_0-logloss:0.56532
[700]	validation_0-logloss:0.55379
[800]	validation_0-logloss:0.54412
[900]	validation_0-logloss:0.53580
[999]	validation_0-logloss:0.52888
Season 2006: 0.7834645669291339 0.5288818500432665
Holdout Season: 2007
[0]	validation_0-logloss:0.69280
[100]	validation_0-logloss:0.66140
[200]	validation_0-logloss:0.63545
[300]	validation_0-logloss:0.61412
[400]	validation_0-logloss:0.59582
[500]	validation_0-logloss:0.58045
[600]	validation_0-logloss:0.56727
[700]	validation_0-logloss:0.55605
[800]	validation_0-logloss:0.54700
[900]	validation_0-logloss:0.53899
[999]	validation_0-logloss:0.53207
Season 2007: 0.7301587301587301 0.5320677444183068
Holdout Season: 2008
[0]	validation_0-logloss:0.69282
[100]	validation_0-logloss:0.66283
[200]	validation_0-logloss:0.63764
[300]	validation_0-logloss:0.61655
[400]	validation_0-logloss:0.59906
[500]	validation_0-logloss:0.58414
[600]	validation_0-logloss:0.57135
[700]	validation_0-logloss:0.56081
[800]	validation_0-logloss:0.55198
[900]	validation_0-logloss:0.54486
[999]	validation_0-logloss:0.53912
Season 2008: 0.7063492063492064 0.5391169694796966
Holdout Season: 2009
[0]	validation_0-logloss:0.69285
[100]	validation_0-logloss:0.66541
[200]	validation_0-logloss:0.64289
```

```
[300]	validation_0-logloss:0.62259
[400]	validation_0-logloss:0.60512
[500]	validation_0-logloss:0.59025
[600]	validation_0-logloss:0.57842
[700]	validation_0-logloss:0.56846
[800]	validation_0-logloss:0.55988
[900]	validation_0-logloss:0.55266
[999]	validation_0-logloss:0.54624
Season 2009: 0.7341269841269841 0.5462413883044884
Holdout Season: 2010
[0]	validation_0-logloss:0.69287
[100]	validation_0-logloss:0.66895
[200]	validation_0-logloss:0.64925
[300]	validation_0-logloss:0.63285
[400]	validation_0-logloss:0.61851
[500]	validation_0-logloss:0.60678
[600]	validation_0-logloss:0.59829
[700]	validation_0-logloss:0.59128
[800]	validation_0-logloss:0.58547
[900]	validation_0-logloss:0.58057
[999]	validation_0-logloss:0.57621
Season 2010: 0.746031746031746 0.5762088515730862
Holdout Season: 2011
[0]	validation_0-logloss:0.69287
[100]	validation_0-logloss:0.66799
[200]	validation_0-logloss:0.64904
[300]	validation_0-logloss:0.63373
[400]	validation_0-logloss:0.62165
[500]	validation_0-logloss:0.61168
[600]	validation_0-logloss:0.60233
[700]	validation_0-logloss:0.59461
[800]	validation_0-logloss:0.58855
[900]	validation_0-logloss:0.58447
[999]	validation_0-logloss:0.58129
Season 2011: 0.7063492063492064 0.5812932656654312
Holdout Season: 2012
[0]	validation_0-logloss:0.69289
[100]	validation_0-logloss:0.66963
[200]	validation_0-logloss:0.65096
[300]	validation_0-logloss:0.63522
[400]	validation_0-logloss:0.62203
[500]	validation_0-logloss:0.61131
[600]	validation_0-logloss:0.60321
[700]	validation_0-logloss:0.59699
[800]	validation_0-logloss:0.59160
[900]	validation_0-logloss:0.58604
[999]	validation_0-logloss:0.58108
Season 2012: 0.6825396825396826 0.5810827664532549
Holdout Season: 2013
[0]	validation_0-logloss:0.69287
[100]	validation_0-logloss:0.66859
```

```
[200]   validation_0-logloss:0.64898
[300]   validation_0-logloss:0.63316
[400]   validation_0-logloss:0.61976
[500]   validation_0-logloss:0.60907
[600]   validation_0-logloss:0.60078
[700]   validation_0-logloss:0.59385
[800]   validation_0-logloss:0.58772
[900]   validation_0-logloss:0.58348
[999]   validation_0-logloss:0.58025
Season 2013: 0.6904761904761905 0.5802509797933194
Holdout Season: 2014
[0] validation_0-logloss:0.69287
[100]   validation_0-logloss:0.66770
[200]   validation_0-logloss:0.64557
[300]   validation_0-logloss:0.62656
[400]   validation_0-logloss:0.61011
[500]   validation_0-logloss:0.59621
[600]   validation_0-logloss:0.58392
[700]   validation_0-logloss:0.57417
[800]   validation_0-logloss:0.56614
[900]   validation_0-logloss:0.55869
[999]   validation_0-logloss:0.55304
Season 2014: 0.7142857142857143 0.553043633254014
Holdout Season: 2015
[0] validation_0-logloss:0.69291
[100]   validation_0-logloss:0.67182
[200]   validation_0-logloss:0.65480
[300]   validation_0-logloss:0.64066
[400]   validation_0-logloss:0.62913
[500]   validation_0-logloss:0.61952
[600]   validation_0-logloss:0.61138
[700]   validation_0-logloss:0.60411
[800]   validation_0-logloss:0.59840
[900]   validation_0-logloss:0.59297
[999]   validation_0-logloss:0.58755
Season 2015: 0.6666666666666666 0.5875460362566335
Holdout Season: 2016
[0] validation_0-logloss:0.69285
[100]   validation_0-logloss:0.66582
[200]   validation_0-logloss:0.64238
[300]   validation_0-logloss:0.62351
[400]   validation_0-logloss:0.60794
[500]   validation_0-logloss:0.59474
[600]   validation_0-logloss:0.58378
[700]   validation_0-logloss:0.57469
[800]   validation_0-logloss:0.56706
[900]   validation_0-logloss:0.55932
[999]   validation_0-logloss:0.55255
Season 2016: 0.7698412698412699 0.5525456201369173
Holdout Season: 2017
[0] validation_0-logloss:0.69288
```

```
[100]	validation_0-logloss:0.66787
[200]	validation_0-logloss:0.64759
[300]	validation_0-logloss:0.62961
[400]	validation_0-logloss:0.61464
[500]	validation_0-logloss:0.60223
[600]	validation_0-logloss:0.59172
[700]	validation_0-logloss:0.58283
[800]	validation_0-logloss:0.57539
[900]	validation_0-logloss:0.56904
[999]	validation_0-logloss:0.56344
Season 2017: 0.7698412698412699 0.5634443193271796
Holdout Season: 2018
[0]	validation_0-logloss:0.69281
[100]	validation_0-logloss:0.66275
[200]	validation_0-logloss:0.63768
[300]	validation_0-logloss:0.61566
[400]	validation_0-logloss:0.59696
[500]	validation_0-logloss:0.58147
[600]	validation_0-logloss:0.56827
[700]	validation_0-logloss:0.55691
[800]	validation_0-logloss:0.54721
[900]	validation_0-logloss:0.53843
[999]	validation_0-logloss:0.53049
Season 2018: 0.7936507936507936 0.530490176309942
Holdout Season: 2019
[0]	validation_0-logloss:0.69286
[100]	validation_0-logloss:0.66659
[200]	validation_0-logloss:0.64485
[300]	validation_0-logloss:0.62537
[400]	validation_0-logloss:0.60911
[500]	validation_0-logloss:0.59551
[600]	validation_0-logloss:0.58377
[700]	validation_0-logloss:0.57349
[800]	validation_0-logloss:0.56437
[900]	validation_0-logloss:0.55681
[999]	validation_0-logloss:0.55022
Season 2019: 0.7301587301587301 0.5502209399838572
Holdout Season: 2021
[0]	validation_0-logloss:0.69285
[100]	validation_0-logloss:0.66611
[200]	validation_0-logloss:0.64414
[300]	validation_0-logloss:0.62468
[400]	validation_0-logloss:0.60717
[500]	validation_0-logloss:0.59322
[600]	validation_0-logloss:0.58191
[700]	validation_0-logloss:0.57141
[800]	validation_0-logloss:0.56251
[900]	validation_0-logloss:0.55529
[999]	validation_0-logloss:0.54939
Season 2021: 0.7936507936507936 0.5493942297502401
Holdout Season: 2022
```

```
[0] validation_0-logloss:0.69289
[100]   validation_0-logloss:0.66851
[200]   validation_0-logloss:0.64874
[300]   validation_0-logloss:0.63198
[400]   validation_0-logloss:0.61734
[500]   validation_0-logloss:0.60451
[600]   validation_0-logloss:0.59320
[700]   validation_0-logloss:0.58400
[800]   validation_0-logloss:0.57707
[900]   validation_0-logloss:0.57161
[999]   validation_0-logloss:0.56720
Season 2022: 0.7301587301587301 0.5671985749078063
Holdout Season: 2023
[0] validation_0-logloss:0.69288
[100]   validation_0-logloss:0.66923
[200]   validation_0-logloss:0.64961
[300]   validation_0-logloss:0.63407
[400]   validation_0-logloss:0.62084
[500]   validation_0-logloss:0.61033
[600]   validation_0-logloss:0.60186
[700]   validation_0-logloss:0.59503
[800]   validation_0-logloss:0.58943
[900]   validation_0-logloss:0.58486
[999]   validation_0-logloss:0.58053
Season 2023: 0.7142857142857143 0.5805315287572473
Average CV Accuracy: 0.7351568954812976
```

Predict on Test Set

Now that we've trained our models. We can use them to predict on our future data.

```
TEST_SEASON = 2024  # Change to 2024 when it comes out!

seeds_2024 = pd.read_csv(DATA_PATH + "2024_tourney_seeds.csv")

seeds_2024["ChalkSeed"] = (
    seeds_2024["Seed"].str.replace("a", "").str.replace("b",
"").str[1:].astype("int")
)
```

Tourney Pairs

- We don't know which teams will play each other in later rounds, so we create a tourney_pairs dataframe.
- This dataframe has all possible combinations of games. We will use our model to predict these.

```python
tourney_pairs = (
    seeds_2024.merge(seeds_2024, on=["Tournament"], suffixes=("", "Opp"))
    .assign(Season=TEST_SEASON)
    .query("TeamID != TeamIDOpp")
    .rename(columns={"Tournament": "League"})
)

tourney_pairs = (
    tourney_pairs.merge(
        team_season_agg[
            ["Season", "League", "TeamID", "WinPercentage", "MedianScoreDiff"]
        ],
        on=["Season", "League", "TeamID"],
        how="left",
    )
    .merge(
        team_season_agg[
            ["Season", "League", "TeamID", "WinPercentage", "MedianScoreDiff"]
        ].rename(
            columns={
                "TeamID": "TeamIDOpp",
                "WinPercentage": "OppWinPercentage",
                "MedianScoreDiff": "OppMedianScoreDiff",
            }
        ),
        on=["Season", "League", "TeamIDOpp"],
    )
    .reset_index(drop=True)
)

tourney_pairs["OppChalkSeed"] = (
    tourney_pairs["SeedOpp"]
    .str.replace("a", "")
    .str.replace("b", "")
    .str[1:]
    .astype("int")
)
```

Add Features to 2024

```python
tourney_pairs["BaselinePred"] = (
    tourney_pairs["ChalkSeed"] < tourney_pairs["OppChalkSeed"]
)

tourney_pairs.loc[
```

```
        tourney_pairs["ChalkSeed"] == tourney_pairs["OppChalkSeed"],
        "BaselinePred",
    ] = (
        tourney_pairs["WinPercentage"] > tourney_pairs["OppWinPercentage"]
    )

    tourney_pairs["WinPctDiff"] = (
        tourney_pairs["WinPercentage"] - tourney_pairs["OppWinPercentage"]
    )

    tourney_pairs["ChalkSeedDiff"] = (
        tourney_pairs["ChalkSeed"] - tourney_pairs["OppChalkSeed"]
    )

    tourney_pairs["MedianScoreDiffDiff"] = (
        tourney_pairs["MedianScoreDiff"] - tourney_pairs["OppMedianScoreDiff"]
    )
```

```
tourney_pairs.head()
```

	League	Seed	TeamID	ChalkSeed	SeedOpp	TeamIDOpp	ChalkSeedOpp	Season	WinPercentage	MedianSc
0	M	W01	1163	1	W02	1235	2	2024	0.911765	14.0
1	M	W01	1163	1	W03	1228	3	2024	0.911765	14.0
2	M	W01	1163	1	W04	1120	4	2024	0.911765	14.0
3	M	W01	1163	1	W05	1361	5	2024	0.911765	14.0
4	M	W01	1163	1	W06	1140	6	2024	0.911765	14.0

Create Predictions and Aggregate

- Loop through each of the models we trained before and predict on the latest tourney seed data.

```
for i, model in enumerate(models):
    tourney_pairs[f"pred_model{i}"] = model.predict(tourney_pairs[FEATURES])
```

```
tourney_pairs["Pred"] = tourney_pairs[
    [f for f in tourney_pairs.columns if "model" in f]
].mean(axis=1)

tourney_pairs["ID"] = (
```

```
        tourney_pairs["Season"].astype("str")
        + "_"
        + tourney_pairs["TeamID"].astype("str")
        + "_"
        + tourney_pairs["TeamIDOpp"].astype("str")
    )

    preds = tourney_pairs.copy()
```

Simulate Bracket

- Now we have probabilites for every possible combination of possible games in the tournament.
- We want to convert this into a standard “bracket” format.
- To do this we simulate each round and select the highest scored team.

```
from tqdm import tqdm

# Load and filter data
round_slots = pd.read_csv(
    "/kaggle/input/march-machine-learning-mania-2024/MNCAATourneySlots.csv"
)
round_slots = round_slots[round_slots["Season"] == 2024]
round_slots = round_slots[
    round_slots["Slot"].str.contains("R")
]  # Filter out First Four

seeds = pd.read_csv(
    "/kaggle/input/march-machine-learning-mania-2024/2024_tourney_seeds.csv"
)
seeds_m = seeds[seeds["Tournament"] == "M"]
seeds_w = seeds[seeds["Tournament"] == "W"]

preds["ID"] = preds["ID"].str.split("_")
```

```
def prepare_data(seeds, preds):
    # Function preparing the data for the simulation
    seed_dict = seeds.set_index("Seed")["TeamID"].to_dict()
    inverted_seed_dict = {value: key for key, value in seed_dict.items()}
    probas_dict = {}

    for teams, proba in zip(preds["ID"], preds["Pred"]):
        team1, team2 = teams[1], teams[2]

        probas_dict.setdefault(team1, {})[team2] = proba
        probas_dict.setdefault(team2, {})[team1] = 1 - proba
```

```
        return seed_dict, inverted_seed_dict, probas_dict


    def simulate(round_slots, seeds, inverted_seeds, probas, sim=True):

        winners = []
        slots = []

        for slot, strong, weak in zip(
            round_slots.Slot, round_slots.StrongSeed, round_slots.WeakSeed
        ):
            team_1, team_2 = seeds[strong], seeds[weak]

            # Get the probability of team_1 winning
            proba = probas[str(team_1)][str(team_2)]

            if sim:
                # Randomly determine the winner based on the probability
                winner = np.random.choice([team_1, team_2], p=[proba, 1 - proba])
            else:
                # Determine the winner based on the higher probability
                winner = [team_1, team_2][np.argmax([proba, 1 - proba])]

            # Append the winner and corresponding slot to the lists
            winners.append(winner)
            slots.append(slot)

            seeds[slot] = winner

        # Convert winners to original seeds using the inverted_seeds dictionary
        return [inverted_seeds[w] for w in winners], slots


    def run_simulation(brackets=1, seeds=None, preds=None, round_slots=None,
sim=True):

        # Get relevant data for the simulation
        seed_dict, inverted_seed_dict, probas_dict = prepare_data(seeds, preds)
        # Lists to store simulation results
        results = []
        bracket = []
        slots = []

        # Iterate through the specified number of brackets
        for b in tqdm(range(1, brackets + 1)):
            # Run single simulation
            r, s = simulate(round_slots, seed_dict, inverted_seed_dict, probas_dict,
sim)

            # Update results
```

```
            results.extend(r)
            bracket.extend([b] * len(r))
            slots.extend(s)

        # Create final DataFrame
        result_df = pd.DataFrame({"Bracket": bracket, "Slot": slots, "Team":
results})

        return result_df


    n_brackets = 1
    result_m = run_simulation(
        brackets=n_brackets, seeds=seeds_m, preds=preds, round_slots=round_slots,
sim=False
    )
    result_m["Tournament"] = "M"
    result_w = run_simulation(
        brackets=n_brackets, seeds=seeds_w, preds=preds, round_slots=round_slots,
sim=False
    )
    result_w["Tournament"] = "W"
    submission = pd.concat([result_m, result_w])
    submission = submission.reset_index(drop=True)
    submission.index.names = ["RowId"]
    submission = submission.reset_index()
```

```
100%|██████████| 1/1 [00:00<00:00, 653.73it/s]
100%|██████████| 1/1 [00:00<00:00, 601.85it/s]
```

```
    ss = pd.read_csv(DATA_PATH + "sample_submission.csv")
    submission[ss.columns] = submission[ss.columns]
    submission[ss.columns].to_csv("submission.csv", index=False)
```

```
    submission_with_names = submission.rename(columns={"Team": "Seed"}).merge(
        seeds, on=["Seed", "Tournament"], how="left"
    )

    teams = pd.concat(
        [
            pd.read_csv(DATA_PATH + "MTeams.csv").assign(Tournament="M"),
            pd.read_csv(DATA_PATH + "WTeams.csv").assign(Tournament="W"),
        ]
    )

    submission_with_names = submission_with_names.merge(
```

```
    teams[["Tournament", "TeamID", "TeamName"]], how="left"
)
```

```
submission_with_names.to_csv("submission_with_names.csv")
```

```
submission_with_names
```

	RowId	Bracket	Slot	Seed	Tournament	TeamID	TeamName
0	0	1	R1W1	W01	M	1163	Connecticut
1	1	1	R1W2	W02	M	1235	Iowa St
2	2	1	R1W3	W03	M	1228	Illinois
3	3	1	R1W4	W04	M	1120	Auburn
4	4	1	R1W5	W05	M	1361	San Diego St
...	...	...	...	...	...	...	...
121	121	1	R4Y1	Y01	W	3234	Iowa
122	122	1	R4Z1	Z03	W	3163	Connecticut
123	123	1	R5WX	W01	W	3376	South Carolina
124	124	1	R5YZ	Y01	W	3234	Iowa
125	125	1	R6CH	W01	W	3376	South Carolina

126 rows × 7 columns

```
output_file_path = '/kaggle/working/submission_with_names.csv'
submission_with_names.to_csv(output_file_path, index=False)
```

```
men_teams = submission_with_names[submission_with_names['Tournament'] == 'M']

# Regions: W, X, Y, Z
regions = ['W', 'X', 'Y', 'Z']

# Create a dictionary to hold the bracket mapping
bracket_mapping = {region: {} for region in regions}

# Populate the bracket mapping
for _, row in men_teams.iterrows():
    region = row['Seed'][0]
```

```
        seed = int(row['Seed'][1:])
        team_name = row['TeamName']

        bracket_mapping[region][seed] = team_name

bracket_mapping
```

```
{'W': {1: 'Connecticut',
  2: 'Iowa St',
  3: 'Illinois',
  4: 'Auburn',
  5: 'San Diego St',
  6: 'BYU',
  7: 'Washington St',
  8: 'FL Atlantic'},
 'X': {1: 'North Carolina',
  2: 'Arizona',
  3: 'Baylor',
  4: 'Alabama',
  5: "St Mary's CA",
  6: 'Clemson',
  7: 'Dayton',
  8: 'Mississippi St'},
 'Y': {1: 'Purdue',
  2: 'Tennessee',
  3: 'Creighton',
  4: 'Kansas',
  5: 'Gonzaga',
  6: 'South Carolina',
  7: 'Texas',
  8: 'Utah St'},
 'Z': {1: 'Houston',
  2: 'Marquette',
  3: 'Kentucky',
  4: 'Duke',
  5: 'Wisconsin',
  6: 'Texas Tech',
  7: 'Florida',
  8: 'Nebraska'}}
```

```
def print_bracket(bracket_mapping):
    for region, seeds in bracket_mapping.items():
        print(f"Region {region}:")
        for seed in sorted(seeds.keys()):
            print(f"  {seed}: {seeds[seed]}")
        print()
```

```
    # Print the bracket representation
    print_bracket(bracket_mapping)
```

```
Region W:
  1: Connecticut
  2: Iowa St
  3: Illinois
  4: Auburn
  5: San Diego St
  6: BYU
  7: Washington St
  8: FL Atlantic

Region X:
  1: North Carolina
  2: Arizona
  3: Baylor
  4: Alabama
  5: St Mary's CA
  6: Clemson
  7: Dayton
  8: Mississippi St

Region Y:
  1: Purdue
  2: Tennessee
  3: Creighton
  4: Kansas
  5: Gonzaga
  6: South Carolina
  7: Texas
  8: Utah St

Region Z:
  1: Houston
  2: Marquette
  3: Kentucky
  4: Duke
  5: Wisconsin
  6: Texas Tech
  7: Florida
  8: Nebraska
```

```
women_teams = submission_with_names[submission_with_names['Tournament'] == 'w']

# Regions: W, X, Y, Z
regions = ['W', 'X', 'Y', 'Z']
```

```python
    # Create a dictionary to hold the bracket mapping
    bracket_mapping = {region: {} for region in regions}

    # Populate the bracket mapping
    for _, row in men_teams.iterrows():
        region = row['Seed'][0]
        seed = int(row['Seed'][1:])
        team_name = row['TeamName']

        bracket_mapping[region][seed] = team_name

    bracket_mapping
```

```
{'W': {1: 'Connecticut',
  2: 'Iowa St',
  3: 'Illinois',
  4: 'Auburn',
  5: 'San Diego St',
  6: 'BYU',
  7: 'Washington St',
  8: 'FL Atlantic'},
 'X': {1: 'North Carolina',
  2: 'Arizona',
  3: 'Baylor',
  4: 'Alabama',
  5: "St Mary's CA",
  6: 'Clemson',
  7: 'Dayton',
  8: 'Mississippi St'},
 'Y': {1: 'Purdue',
  2: 'Tennessee',
  3: 'Creighton',
  4: 'Kansas',
  5: 'Gonzaga',
  6: 'South Carolina',
  7: 'Texas',
  8: 'Utah St'},
 'Z': {1: 'Houston',
  2: 'Marquette',
  3: 'Kentucky',
  4: 'Duke',
  5: 'Wisconsin',
  6: 'Texas Tech',
  7: 'Florida',
  8: 'Nebraska'}}
```

```python
    def print_bracket(bracket_mapping):
        for region, seeds in bracket_mapping.items():
```

```
            print(f"Region {region}:")
            for seed in sorted(seeds.keys()):
                print(f"  {seed}: {seeds[seed]}")
            print()

    # Print the bracket representation
    print_bracket(bracket_mapping)
```

```
Region W:
  1: Connecticut
  2: Iowa St
  3: Illinois
  4: Auburn
  5: San Diego St
  6: BYU
  7: Washington St
  8: FL Atlantic

Region X:
  1: North Carolina
  2: Arizona
  3: Baylor
  4: Alabama
  5: St Mary's CA
  6: Clemson
  7: Dayton
  8: Mississippi St

Region Y:
  1: Purdue
  2: Tennessee
  3: Creighton
  4: Kansas
  5: Gonzaga
  6: South Carolina
  7: Texas
  8: Utah St

Region Z:
  1: Houston
  2: Marquette
  3: Kentucky
  4: Duke
  5: Wisconsin
  6: Texas Tech
  7: Florida
  8: Nebraska
```