50-master

March 22, 2020

This file, 50-master, is a combination of all 4 notebooks: 10-import, 20-Exploratory_Data_Analysis, 30-Feature_Engineering, 40-Modeling

0.0.1 10-import

Libraries

[6]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
sns.set()
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn import linear_model
from sklearn.metrics import roc_auc_score
```

Read in data

[7]:
```
df = pd.read_csv('cbb.csv')
```

Inspect data

[8]:
```
## view the first 5 rows of the dataset
df.head()
```

[8]:

	TEAM	CONF	G	W	ADJOE	ADJDE	BARTHAG	EFG_O	EFG_D	TOR	…	FTRD	2P_O	2P_D	3P_O	3P_D	ADJ_T	WAB	POSTSEASON	SEED	YEAR
0	North Carolina	ACC	40	33	123.3	94.9	0.9531	52.6	48.1	15.4	…	30.4	53.9	44.6	32.7	36.2	71.7	8.6	2ND	1.0	2016
1	Wisconsin	B10	40	36	129.1	93.6	0.9758	54.8	47.7	12.4	…	22.4	54.8	44.7	36.5	37.5	59.3	11.3	2ND	1.0	2015
2	Michigan	B10	40	33	114.4	90.4	0.9375	53.9	47.7	14.0	…	30.0	54.7	46.8	35.2	33.2	65.9	6.9	2ND	3.0	2018
3	Texas Tech	B12	38	31	115.2	85.2	0.9696	53.5	43.0	17.7	…	36.6	52.8	41.9	36.5	29.7	67.5	7.0	2ND	3.0	2019
4	Gonzaga	WCC	39	37	117.8	86.3	0.9728	56.6	41.1	16.2	…	26.9	56.3	40.0	38.2	29.0	71.5	7.7	2ND	1.0	2017

```
[5 rows x 24 columns]
```

[9]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1757 entries, 0 to 1756
Data columns (total 24 columns):
TEAM          1757 non-null object
CONF          1757 non-null object
G             1757 non-null int64
W             1757 non-null int64
ADJOE         1757 non-null float64
ADJDE         1757 non-null float64
BARTHAG       1757 non-null float64
EFG_O         1757 non-null float64
EFG_D         1757 non-null float64
TOR           1757 non-null float64
TORD          1757 non-null float64
ORB           1757 non-null float64
DRB           1757 non-null float64
FTR           1757 non-null float64
FTRD          1757 non-null float64
2P_O          1757 non-null float64
2P_D          1757 non-null float64
3P_O          1757 non-null float64
3P_D          1757 non-null float64
ADJ_T         1757 non-null float64
WAB           1757 non-null float64
POSTSEASON    340 non-null object
SEED          340 non-null float64
YEAR          1757 non-null int64
dtypes: float64(18), int64(3), object(3)
memory usage: 329.6+ KB
```

[10]:
```
df.describe()
```

[10]:

	G	W	ADJOE	ADJDE	BARTHAG	\
count	1757.000000	1757.000000	1757.000000	1757.000000	1757.000000	
mean	31.523051	16.513375	103.542402	103.542459	0.493398	
std	2.602819	6.545571	7.304975	6.472676	0.255291	
min	24.000000	0.000000	76.700000	84.000000	0.007700	
25%	30.000000	12.000000	98.600000	98.900000	0.283700	
50%	31.000000	16.000000	103.100000	103.800000	0.474000	
75%	33.000000	21.000000	108.100000	108.000000	0.710600	
max	40.000000	38.000000	129.100000	124.000000	0.984200	

```
              EFG_O        EFG_D          TOR         TORD          ORB  …  \
count  1757.000000  1757.000000  1757.000000  1757.000000  1757.000000  …
mean     50.120489    50.312806    18.591804    18.521286    29.277120  …
std       3.130430     2.859604     1.991637     2.108968     4.101782  …
min      39.400000    39.600000    12.400000    10.200000    15.000000  …
25%      48.100000    48.400000    17.200000    17.100000    26.600000  …
50%      50.000000    50.300000    18.500000    18.500000    29.400000  …
75%      52.100000    52.300000    19.800000    19.900000    31.900000  …
max      59.800000    59.500000    26.100000    28.000000    42.100000  …

              FTR         FTRD         2P_O         2P_D         3P_O  \
count  1757.000000  1757.000000  1757.000000  1757.000000  1757.000000
mean     35.097894    35.373307    49.135970    49.298065    34.563517
std       4.884599     5.900935     3.422136     3.288265     2.742323
min      21.600000    21.800000    37.700000    37.700000    25.200000
25%      31.700000    31.200000    46.900000    47.100000    32.600000
50%      34.900000    34.900000    49.000000    49.300000    34.600000
75%      38.300000    39.200000    51.400000    51.500000    36.400000
max      51.000000    58.500000    62.600000    61.200000    44.100000

              3P_D        ADJ_T          WAB        SEED         YEAR
count  1757.000000  1757.000000  1757.000000  340.000000  1757.000000
mean     34.744792    68.422254    -7.837109    8.791176  2017.002277
std       2.369727     3.258920     6.988694    4.674090     1.415419
min      27.100000    57.200000   -25.200000    1.000000  2015.000000
25%      33.100000    66.400000   -13.000000    5.000000  2016.000000
50%      34.700000    68.500000    -8.400000    9.000000  2017.000000
75%      36.300000    70.400000    -3.100000   13.000000  2018.000000
max      43.100000    83.400000    13.100000   16.000000  2019.000000

[8 rows x 21 columns]
```

[11]:
```
## size of dataset
df.shape
```

[11]:
```
(1757, 24)
```

[12]:
```
## any NA values
df.isnull().sum()
```

[12]:
```
TEAM            0
CONF            0
G               0
W               0
ADJOE           0
ADJDE           0
BARTHAG         0
```

```
EFG_O            0
EFG_D            0
TOR              0
TORD             0
ORB              0
DRB              0
FTR              0
FTRD             0
2P_O             0
2P_D             0
3P_O             0
3P_D             0
ADJ_T            0
WAB              0
POSTSEASON    1417
SEED          1417
YEAR             0
dtype: int64
```

From this, I can conclude/confirm that there are only two columns that contain NA values. In the case of this dataset and for the purposes of this project, these NA values actually tell us something: that "(1757-340)= 1417" teams have never made it to the March Madness tournament.

Inspecting the target column, "SEED"

[13]:
```
df['SEED'].values
```

[13]:
```
array([ 1.,  1.,  3., …,  2., 11.,  4.])
```

[14]:
```
df['SEED'].value_counts
```

[14]:
```
<bound method IndexOpsMixin.value_counts of 0        1.0
1        1.0
2        3.0
3        3.0
4        1.0
        …
1752     7.0
1753     3.0
1754     2.0
1755    11.0
1756     4.0
Name: SEED, Length: 1757, dtype: float64>
```

[15]:
```
print(np.min(df['SEED']))
print(np.max(df['SEED']))
```

```
1.0
16.0
```

This informs me that the highest SEED number is 16, for each basketball tournament. (Refer to *00_dataset-variables.ipynb* for more information on "SEED")

[16]:
```
df['YEAR'].unique()
```

[16]:
```
array([2016, 2015, 2018, 2019, 2017])
```

The dataset spans 5 years.

0.0.2 20-Exploratory Data Analysis

Which columns in the dataset contain numeric values?

[17]:
```
numeric_df = df.select_dtypes(include=['int', 'float'])

# Print the column names contained in df
print(numeric_df.columns)
```

```
Index(['G', 'W', 'ADJOE', 'ADJDE', 'BARTHAG', 'EFG_O', 'EFG_D', 'TOR', 'TORD',
       'ORB', 'DRB', 'FTR', 'FTRD', '2P_O', '2P_D', '3P_O', '3P_D', 'ADJ_T',
       'WAB', 'SEED', 'YEAR'],
      dtype='object')
```

Explore the values in each of these numeric columns to determine which ones are necessary for the purposes of my project:

[18]:
```
print(df['G'].values)
print(df['W'].values)
print(df['ADJOE'].values)
print(df['BARTHAG'].values)
print(df['EFG_O'].values)
```

```
[40 40 40 … 36 35 37]
[33 36 33 … 31 27 32]
[123.3 129.1 114.4 … 122.8 117.4 117.2]
[0.9531 0.9758 0.9375 … 0.9488 0.9238 0.9192]
[52.6 54.8 53.9 … 55.3 55.2 57. ]
```

From inspecting the values of only 5 columns fromthe dataset, I can tell that there is the need to either standardize or normalize the values in the various numeric columns of my dataset. I will do this in the feature emginnering notebook, "30-Feature Engineering"

Which columns in the dataset contain string values?

[19]:
```
string_df = df.select_dtypes(include=['object'])
print(string_df.columns)
```

```
Index(['TEAM', 'CONF', 'POSTSEASON'], dtype='object')
```

Explore the values in each of these three columns to determine if they should be used for predictions:

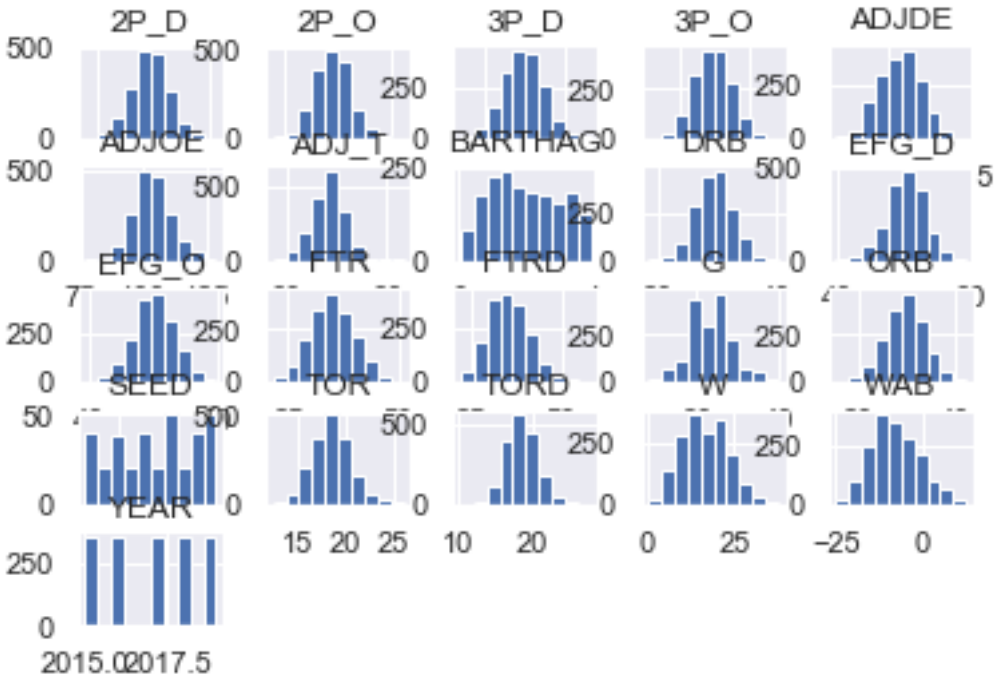
[20]:
```
print(df['TEAM'].values)
print(df['CONF'].values)
print(df['POSTSEASON'].values)
```

```
['North Carolina' 'Wisconsin' 'Michigan' … 'Tennessee' 'Gonzaga'
 'Gonzaga']
['ACC' 'B10' 'B10' … 'SEC' 'WCC' 'WCC']
['2ND' '2ND' '2ND' … 'S16' 'S16' 'S16']
```

It could be helpful to explore the number of teams that make it to the tournament from each league (Ex. ACC, B10, SEC, etc), how highly they're ranked in the national tournament, and how their performaces vary from year to year. However, these explorations would not be necessary, for the purposes of my project.

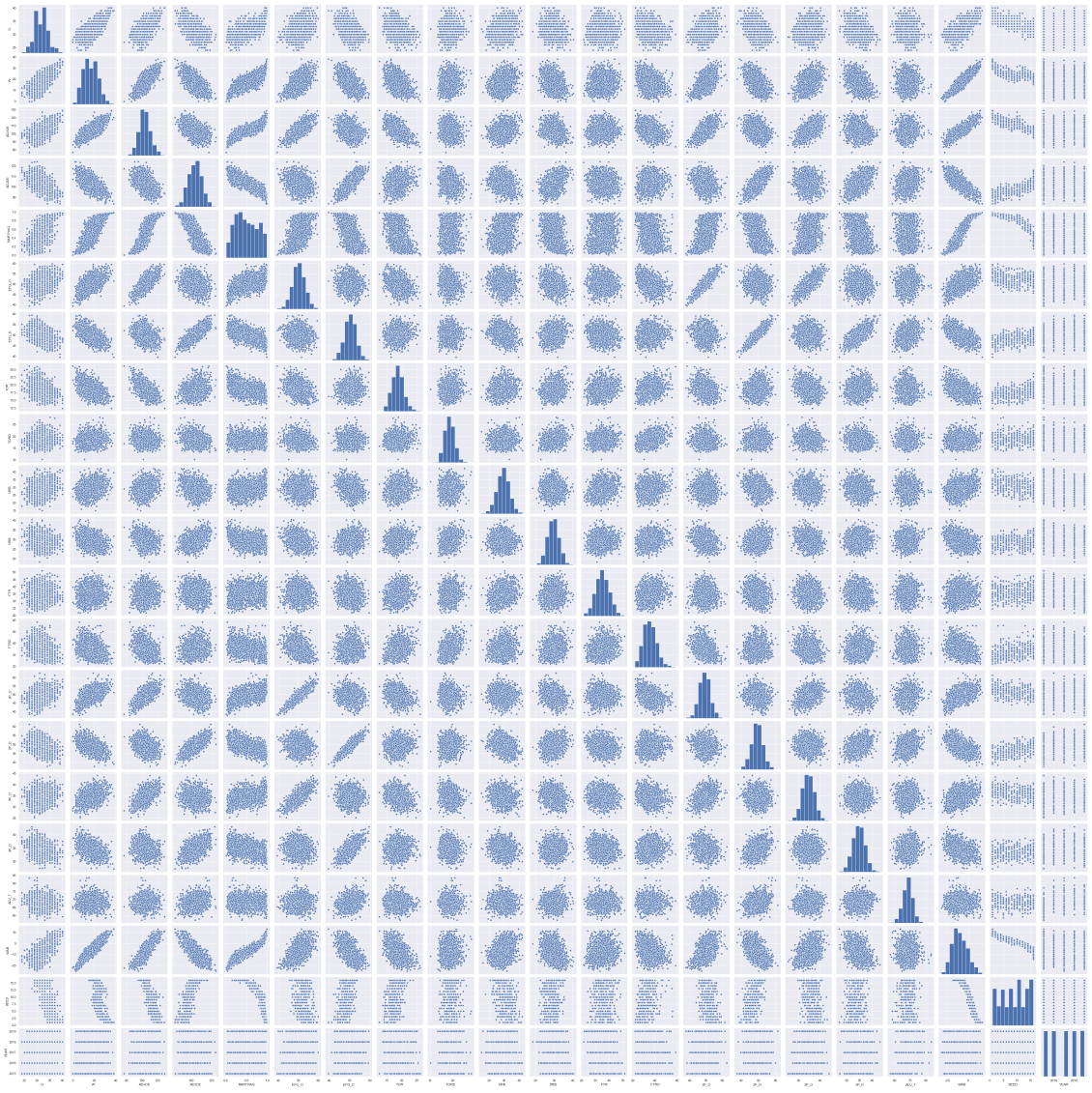
Check the distribution of the features in the dataset because most ML models assume that the data is normally distributed:

[21]:
```
df.hist()
plt.show()
```



I'd like to explore the possibility of some of the features in the dataset being correlated/having a relationship:

[22]:
```
sns.pairplot(df)
plt.show()
```



From quick observation, some of the positive correlations make a lot of sense. For example:

- BARTHAG, Power Rating & W, Number of games won. A team with a higher chance of beating an average Division I team can be expected to have a high number of wins.
- 2P_D, Two-Point Shooting Percentage Allowed & ADJDE, Adjusted Defensive Efficiency. A team that prevents the opposing team from making baskets can be said to be efficient, defensively.

0.0.3 30-Feature Engineering

label the target column, 'Y'

[23]:
```
Y = df['SEED'].copy()
```

Repalce the null values with 0 since it was confirmed that the null values in the dataset are non-trivial: they represent teams that never made it to the basketball tournament

[24]:
```
Y.fillna(0, inplace = True)
```

[25]:
```
Y.value_counts()
```

[25]:
```
0.0     1417
16.0      30
11.0      30
3.0       21
15.0      20
10.0      20
2.0       20
6.0       20
9.0       20
7.0       20
14.0      20
5.0       20
8.0       20
12.0      20
13.0      20
1.0       20
4.0       19
Name: SEED, dtype: int64
```

Replace values ranging from 1-16 with '1' so that '1' will refer to teams that made it to the tournament and '0' will represent teams that did not make it

[26]:
```
Y.replace([1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16], 1, inplace=True)
```

Remove string columns since it was determined in "20-Exploratory_Data_Analysis" that these columns will not be used for the purposes of this project, as well as the target column, 'SEED'/'Y

[27]:
```
string_df
```

[27]:

	TEAM	CONF	POSTSEASON
0	North Carolina	ACC	2ND
1	Wisconsin	B10	2ND
2	Michigan	B10	2ND
3	Texas Tech	B12	2ND
4	Gonzaga	WCC	2ND

```
…              …  …         …
1752       Texas A&M  SEC        S16
1753             LSU  SEC        S16
1754       Tennessee  SEC        S16
1755         Gonzaga  WCC        S16
1756         Gonzaga  WCC        S16

[1757 rows x 3 columns]
```

[28]:
```
strings = ['TEAM', 'CONF', 'POSTSEASON', 'SEED']
df.drop(columns = strings, inplace = True)
```

[29]:
```
df.columns
```

[29]:
```
Index(['G', 'W', 'ADJOE', 'ADJDE', 'BARTHAG', 'EFG_O', 'EFG_D', 'TOR', 'TORD',
       'ORB', 'DRB', 'FTR', 'FTRD', '2P_O', '2P_D', '3P_O', '3P_D', 'ADJ_T',
       'WAB', 'YEAR'],
      dtype='object')
```

[30]:
```
df
```

[30]:

	G	W	ADJOE	ADJDE	BARTHAG	EFG_O	EFG_D	TOR	TORD	ORB	DRB	\
0	40	33	123.3	94.9	0.9531	52.6	48.1	15.4	18.2	40.7	30.0	
1	40	36	129.1	93.6	0.9758	54.8	47.7	12.4	15.8	32.1	23.7	
2	40	33	114.4	90.4	0.9375	53.9	47.7	14.0	19.5	25.5	24.9	
3	38	31	115.2	85.2	0.9696	53.5	43.0	17.7	22.8	27.4	28.7	
4	39	37	117.8	86.3	0.9728	56.6	41.1	16.2	17.1	30.0	26.2	
…	..	..	…	…	…	…	…	…	…	…	…	
1752	35	22	111.2	94.7	0.8640	51.4	46.9	19.2	15.3	33.9	27.3	
1753	35	28	117.9	96.6	0.9081	51.2	49.9	17.9	20.1	36.7	30.8	
1754	36	31	122.8	95.2	0.9488	55.3	48.1	15.8	18.0	31.6	30.2	
1755	35	27	117.4	94.5	0.9238	55.2	44.8	17.1	15.1	32.1	26.0	
1756	37	32	117.2	94.9	0.9192	57.0	47.1	16.1	17.4	33.0	23.1	

	FTR	FTRD	2P_O	2P_D	3P_O	3P_D	ADJ_T	WAB	YEAR
0	32.3	30.4	53.9	44.6	32.7	36.2	71.7	8.6	2016
1	36.2	22.4	54.8	44.7	36.5	37.5	59.3	11.3	2015
2	30.7	30.0	54.7	46.8	35.2	33.2	65.9	6.9	2018
3	32.9	36.6	52.8	41.9	36.5	29.7	67.5	7.0	2019
4	39.0	26.9	56.3	40.0	38.2	29.0	71.5	7.7	2017
…	…	…	…	…	…	…	…	…	…
1752	32.0	27.6	52.5	45.7	32.9	32.6	70.3	1.9	2018
1753	37.1	33.1	52.9	49.4	31.9	33.7	71.2	7.3	2019
1754	33.3	34.9	55.4	44.7	36.7	35.4	68.8	9.9	2019
1755	34.4	28.1	54.3	44.4	37.8	30.3	68.2	2.1	2016
1756	32.1	29.1	58.2	44.1	36.8	35.0	70.5	4.9	2018

```
[1757 rows x 20 columns]
```

standardizing the remaining columns, the predictors. First, let's check for variance:

[31]:
```
df.var().round(3)
```

[31]:
```
G           6.775
W          42.844
ADJOE      53.363
ADJDE      41.896
BARTHAG     0.065
EFG_O       9.800
EFG_D       8.177
TOR         3.967
TORD        4.448
ORB        16.825
DRB         9.375
FTR        23.859
FTRD       34.821
2P_O       11.711
2P_D       10.813
3P_O        7.520
3P_D        5.616
ADJ_T      10.621
WAB        48.842
YEAR        2.003
dtype: float64
```

[32]:
```
x = df.values #returns a numpy array
min_max_scaler = preprocessing.MinMaxScaler()
x_scaled = min_max_scaler.fit_transform(x)
df = pd.DataFrame(x_scaled)
```

[33]:
```
df.var().round(3)
```

[33]:
```
0     0.026
1     0.030
2     0.019
3     0.026
4     0.068
5     0.024
6     0.021
7     0.021
8     0.014
9     0.023
10    0.019
11    0.028
```

```
12    0.026
13    0.019
14    0.020
15    0.021
16    0.022
17    0.015
18    0.033
19    0.125
dtype: float64
```

[34]:
```
df = df.assign(Y=Y.values)
```

[35]:
```
df
```

[35]:

	0	1	2	3	4	5	6	\
0	1.0000	0.868421	0.889313	0.2725	0.968152	0.647059	0.427136	
1	1.0000	0.947368	1.000000	0.2400	0.991398	0.754902	0.407035	
2	1.0000	0.868421	0.719466	0.1600	0.952176	0.710784	0.407035	
3	0.8750	0.815789	0.734733	0.0300	0.985049	0.691176	0.170854	
4	0.9375	0.973684	0.784351	0.0575	0.988326	0.843137	0.075377	
…	…	…	…	…	…	…	…	
1752	0.6875	0.578947	0.658397	0.2675	0.876907	0.588235	0.366834	
1753	0.6875	0.736842	0.786260	0.3150	0.922069	0.578431	0.517588	
1754	0.7500	0.815789	0.879771	0.2800	0.963748	0.779412	0.427136	
1755	0.6875	0.710526	0.776718	0.2625	0.938146	0.774510	0.261307	
1756	0.8125	0.842105	0.772901	0.2725	0.933436	0.862745	0.376884	

	7	8	9	…	11	12	13	\
0	0.218978	0.449438	0.948339	…	0.363946	0.234332	0.650602	
1	0.000000	0.314607	0.630996	…	0.496599	0.016349	0.686747	
2	0.116788	0.522472	0.387454	…	0.309524	0.223433	0.682731	
3	0.386861	0.707865	0.457565	…	0.384354	0.403270	0.606426	
4	0.277372	0.387640	0.553506	…	0.591837	0.138965	0.746988	
…	…	…	…	…	…	…	…	
1752	0.496350	0.286517	0.697417	…	0.353741	0.158038	0.594378	
1753	0.401460	0.556180	0.800738	…	0.527211	0.307902	0.610442	
1754	0.248175	0.438202	0.612546	…	0.397959	0.356948	0.710843	
1755	0.343066	0.275281	0.630996	…	0.435374	0.171662	0.666667	
1756	0.270073	0.404494	0.664207	…	0.357143	0.198910	0.823293	

	14	15	16	17	18	19	Y
0	0.293617	0.396825	0.56875	0.553435	0.882507	0.25	1.0
1	0.297872	0.597884	0.65000	0.080153	0.953003	0.00	1.0
2	0.387234	0.529101	0.38125	0.332061	0.838120	0.75	1.0
3	0.178723	0.597884	0.16250	0.393130	0.840731	1.00	1.0
4	0.097872	0.687831	0.11875	0.545802	0.859008	0.50	1.0
…	…	…	…	…	…	…	…

```
1752  0.340426  0.407407  0.34375  0.500000  0.707572  0.75  1.0
1753  0.497872  0.354497  0.41250  0.534351  0.848564  1.00  1.0
1754  0.297872  0.608466  0.51875  0.442748  0.916449  1.00  1.0
1755  0.285106  0.666667  0.20000  0.419847  0.712794  0.25  1.0
1756  0.272340  0.613757  0.49375  0.507634  0.785901  0.75  1.0

[1757 rows x 21 columns]
```

0.0.4 40-Modeling

Split the dataset into train and test sets

it is important to remember that the number of teams that make it to the tournament and those that don't is imbalanced. I will account for this imbalance during the splitting process through the "stratify" method:

[36]:
```
# Split DataFrame into
# X_train, X_test, y_train and y_test datasets,
# stratifying on the `target` column
X_train, X_test, y_train, y_test = train_test_split(
    df.drop(columns='Y'),
    df.Y,
    test_size=0.25,
    random_state=42,
    stratify=df.Y
)
```

[37]:
```
X_train.head(2)
```

[37]:
```
          0    1         2       3         4         5         6         7   \
1037  0.6875  0.5  0.589695  0.3625  0.743881  0.612745  0.507538  0.598540
613   0.5625  0.5  0.553435  0.5550  0.490425  0.617647  0.562814  0.416058

             8         9         10        11        12        13        14  \
1037  0.320225  0.656827  0.390909  0.680272  0.201635  0.506024  0.434043
613   0.376404  0.479705  0.463636  0.472789  0.163488  0.550201  0.459574

            15      16        17        18    19
1037  0.619048  0.5000  0.561069  0.545692  0.75
613   0.529101  0.6375  0.564885  0.449086  0.75
```

[38]:
```
X_test.head(2)
```

[38]:
```
        0         1         2       3         4         5         6   \
997  0.375  0.236842  0.421756  0.6825  0.199898  0.411765  0.728643
596  0.375  0.236842  0.475191  0.7325  0.218945  0.470588  0.703518

            7         8         9    10        11        12        13  \
```

```
997  0.510949  0.410112  0.527675  0.5  0.387755  0.656676  0.381526
596  0.233577  0.308989  0.284133  0.7  0.278912  0.291553  0.313253

            14        15       16        17        18    19
997  0.672340  0.380952  0.60625  0.507634  0.258486  0.75
596  0.714894  0.624339  0.46875  0.610687  0.292428  0.25
```

I will be using a simple Logistic Regression model for this project, at least for now. My reason is that logistic regression is suitable for when a Y variable takes on only two values. Such a variable is referred to a "binary" or "dichotomous." "Dichotomous" basically means two categories such as yes/no, defective/non-defective, success/failure, and so on. For this project, my goal is to predict whether a college basketball team will make it to the March Madness tournament (1) or not (0)

[39]:
```
# Instantiate LogisticRegression
logreg = linear_model.LogisticRegression(
    solver='liblinear',
    random_state=42
)

# Train the model
logreg.fit(X_train, y_train)

logreg_auc_score = roc_auc_score(y_test, logreg.predict_proba(X_test)[:, 1])
print(f'\nAUC score: {logreg_auc_score:.4f}')
```

```
AUC score: 0.9444
```

This is actually impressive and it makes me wonder if I missed anything. The fact that I didn't do much to arrive at this level of accuracy makes me wonder why "Bracketology" is such an unconquered beast in the sports world. But I guess it gets more complicated when people try to predict every single game's outcome. I plan on trying a few other models soon.