# NCAA Championship Statistics

Can we determine March Madness contestants based on historical data?

## Overview

The purpose of this analysis was to obtain greater understanding of NCAA Division 1 Statistics using KNN, Random Forest & Logistic Regression

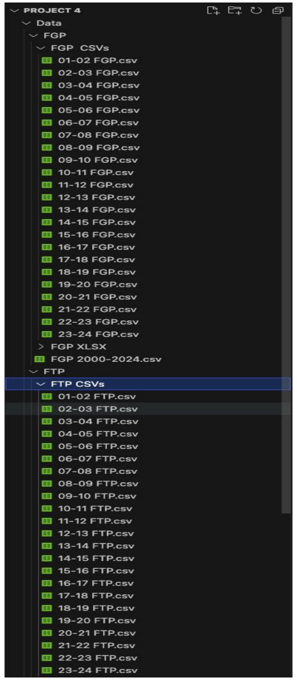
Top Accuracy scores:

- KNN: 66%
- Random Forest: 72%
- Logistic Regression: 75%

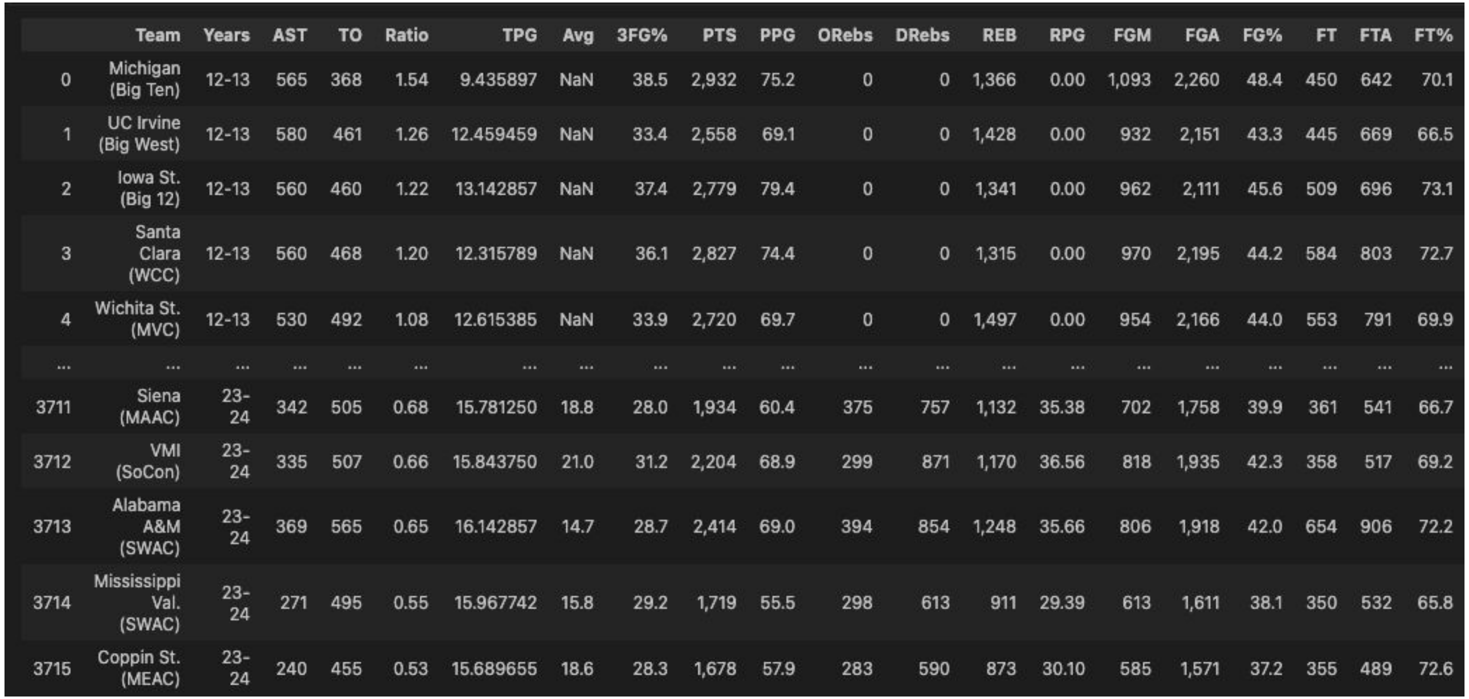
## Datasets Analyzed

The main dataset used was several data pulls of team stats going back 10 to 20 years, and included such stats as Assist/Turnover Ratio, Three-Point Attempts, and Rebounds, etc. These were all pulled from the stats page of the NCAA website and combined into one main csv file using Spark. The combined data then includes Team Name, Season of the Stats in format of 1011 for example, meaning years 2010 through 2011. Overall, about 3700 records combined to analyze.

# Data Cleaning



```
PROJECT 4
  Data
    FGP
      FGP  CSVs
        01-02 FGP.csv
        02-03 FGP.csv
        03-04 FGP.csv
        04-05 FGP.csv
        05-06 FGP.csv
        06-07 FGP.csv
        07-08 FGP.csv
        08-09 FGP.csv
        09-10 FGP.csv
        10-11 FGP.csv
        11-12 FGP.csv
        12-13 FGP.csv
        13-14 FGP.csv
        14-15 FGP.csv
        15-16 FGP.csv
        16-17 FGP.csv
        17-18 FGP.csv
        18-19 FGP.csv
        19-20 FGP.csv
        20-21 FGP.csv
        21-22 FGP.csv
        22-23 FGP.csv
        23-24 FGP.csv
      > FGP XLSX
      FGP 2000-2024.csv
    FTP
      FTP CSVs
        01-02 FTP.csv
        02-03 FTP.csv
        03-04 FTP.csv
        04-05 FTP.csv
        05-06 FTP.csv
        06-07 FTP.csv
        07-08 FTP.csv
        08-09 FTP.csv
        09-10 FTP.csv
        10-11 FTP.csv
        11-12 FTP.csv
        12-13 FTP.csv
        13-14 FTP.csv
        14-15 FTP.csv
        15-16 FTP.csv
        16-17 FTP.csv
        17-18 FTP.csv
        18-19 FTP.csv
        19-20 FTP.csv
        20-21 FTP.csv
        21-22 FTP.csv
        22-23 FTP.csv
        23-24 FTP.csv
```



| | Team | Years | AST | TO | Ratio | TPG | Avg | 3FG% | PTS | PPG | ORebs | DRebs | REB | RPG | FGM | FGA | FG% | FT | FTA | FT% |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | Michigan (Big Ten) | 12-13 | 565 | 368 | 1.54 | 9.435897 | NaN | 38.5 | 2,932 | 75.2 | 0 | 0 | 1,366 | 0.00 | 1,093 | 2,260 | 48.4 | 450 | 642 | 70.1 |
| 1 | UC Irvine (Big West) | 12-13 | 580 | 461 | 1.26 | 12.459459 | NaN | 33.4 | 2,558 | 69.1 | 0 | 0 | 1,428 | 0.00 | 932 | 2,151 | 43.3 | 445 | 669 | 66.5 |
| 2 | Iowa St. (Big 12) | 12-13 | 560 | 460 | 1.22 | 13.142857 | NaN | 37.4 | 2,779 | 79.4 | 0 | 0 | 1,341 | 0.00 | 962 | 2,111 | 45.6 | 509 | 696 | 73.1 |
| 3 | Santa Clara (WCC) | 12-13 | 560 | 468 | 1.20 | 12.315789 | NaN | 36.1 | 2,827 | 74.4 | 0 | 0 | 1,315 | 0.00 | 970 | 2,195 | 44.2 | 584 | 803 | 72.7 |
| 4 | Wichita St. (MVC) | 12-13 | 530 | 492 | 1.08 | 12.615385 | NaN | 33.9 | 2,720 | 69.7 | 0 | 0 | 1,497 | 0.00 | 954 | 2,166 | 44.0 | 553 | 791 | 69.9 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 3711 | Siena (MAAC) | 23-24 | 342 | 505 | 0.68 | 15.781250 | 18.8 | 28.0 | 1,934 | 60.4 | 375 | 757 | 1,132 | 35.38 | 702 | 1,758 | 39.9 | 361 | 541 | 66.7 |
| 3712 | VMI (SoCon) | 23-24 | 335 | 507 | 0.66 | 15.843750 | 21.0 | 31.2 | 2,204 | 68.9 | 299 | 871 | 1,170 | 36.56 | 818 | 1,935 | 42.3 | 358 | 517 | 69.2 |
| 3713 | Alabama A&M (SWAC) | 23-24 | 369 | 565 | 0.65 | 16.142857 | 14.7 | 28.7 | 2,414 | 69.0 | 394 | 854 | 1,248 | 35.66 | 806 | 1,918 | 42.0 | 654 | 906 | 72.2 |
| 3714 | Mississippi Val. (SWAC) | 23-24 | 271 | 495 | 0.55 | 15.967742 | 15.8 | 29.2 | 1,719 | 55.5 | 298 | 613 | 911 | 29.39 | 613 | 1,611 | 38.1 | 350 | 532 | 65.8 |
| 3715 | Coppin St. (MEAC) | 23-24 | 240 | 455 | 0.53 | 15.689655 | 18.6 | 28.3 | 1,678 | 57.9 | 283 | 590 | 873 | 30.10 | 585 | 1,571 | 37.2 | 355 | 489 | 72.6 |

## Problems in training

At first, the goal was to train the model based on individual matchups in an effort to predict a teams performance throughout the NCAA March Madness tournament.

We experienced some issues here, as our data did not include individual matchups so we had to pivot.

We landed on predicting contestants who make it to the tournament, based on season stats.

Contestants who made it into the tournament that year would be given a "Madness" label of 1, and those who did not make it to the tournament would be given a 0.

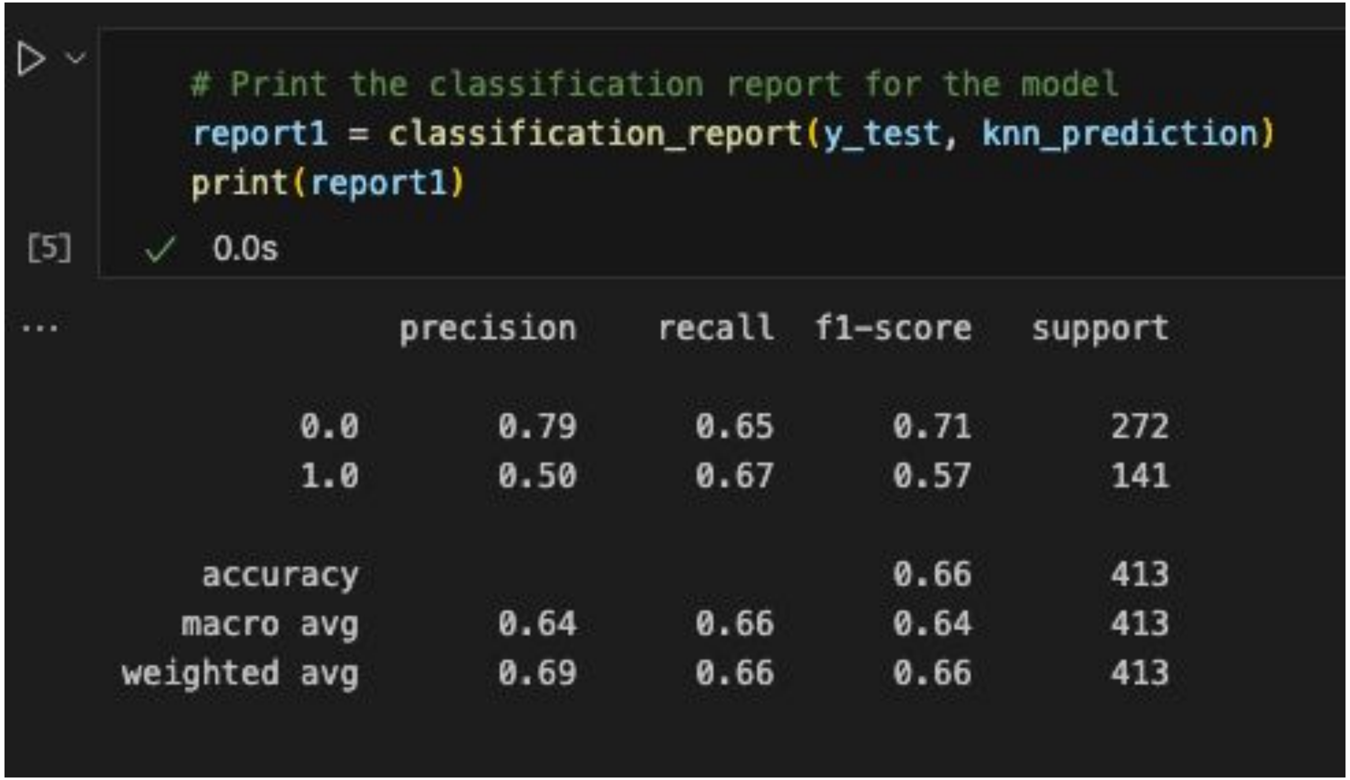
## Resolving Imbalanced Data

Another obstacle we ran into when determining March Madness contestants, is that of 351 Division I NCAA basketball teams, only 68 are considered for the tournament. This presented us with a very imbalanced data set, and in order to fix this imbalance we reduced the number of teams' stats we considered by about half. We kept the top 150 teams by points scored for each season, and then joined the 68 Madness Contestants with the remaining 82 Division I teams.

## KNN

- Top accuracy reached: 66%
- Although we tried resampling and scaling our data, KNN was still unable to reach the levels of accuracy we were experiencing with both Random Forest and Logistic Regression
- A couple reasons for this are that KNN is highly sensitive to feature scaling, and imbalanced classes. This causes KNN to struggle in classifying our minority class accurately.
- Finally, when researching optimization, we discovered KNN is sensitive to noisy data.

## KNN Classification Report



```
# Print the classification report for the model
report1 = classification_report(y_test, knn_prediction)
print(report1)
```

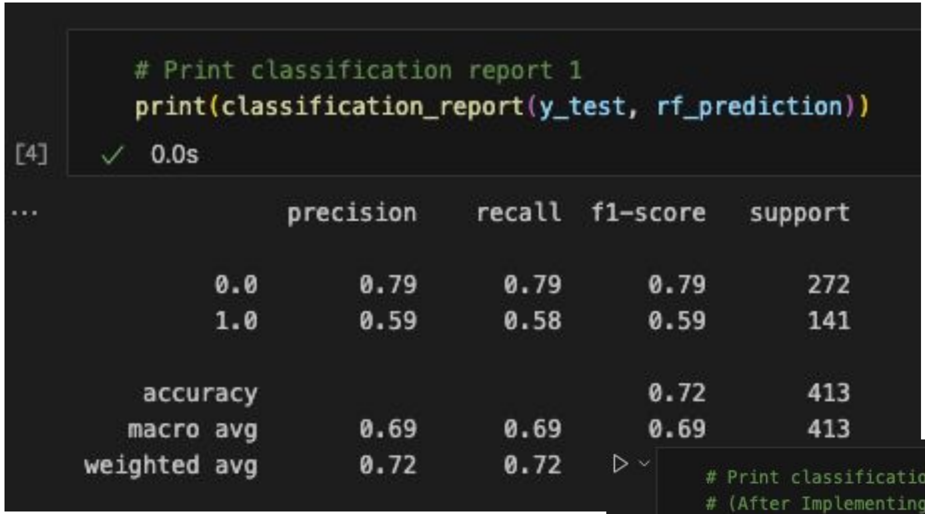
[5] ✓ 0.0s

```
              precision    recall  f1-score   support

         0.0       0.79      0.65      0.71       272
         1.0       0.50      0.67      0.57       141

    accuracy                           0.66       413
   macro avg       0.64      0.66      0.64       413
weighted avg       0.69      0.66      0.66       413
```

## Random Forest

- Top Accuracy Reached: 72%
- Random Forest was the second classification model we used while trying to optimize Logistic Regression.
- Most optimizations we tried here did not work to improve accuracy and even negatively impacted accuracy in some cases.
- When trying to apply the same optimizations that helped Logistic Regression, we were unsuccessful in reaching the 75% target. This may be because Random Forest is a an ensemble of decision trees and could require more tuning and feature engineering for optimal performance.

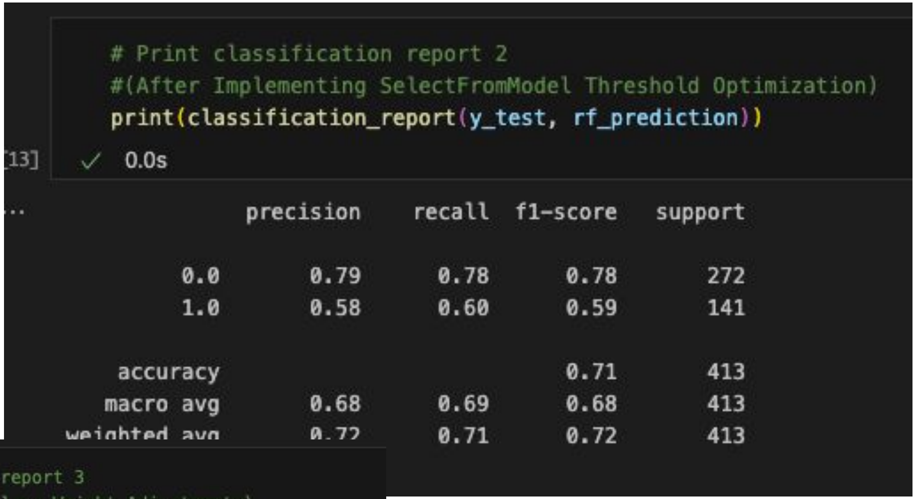
# Random Forest Classification Reports



```
# Print classification report 1
print(classification_report(y_test, rf_prediction))
```

[4] ✓ 0.0s

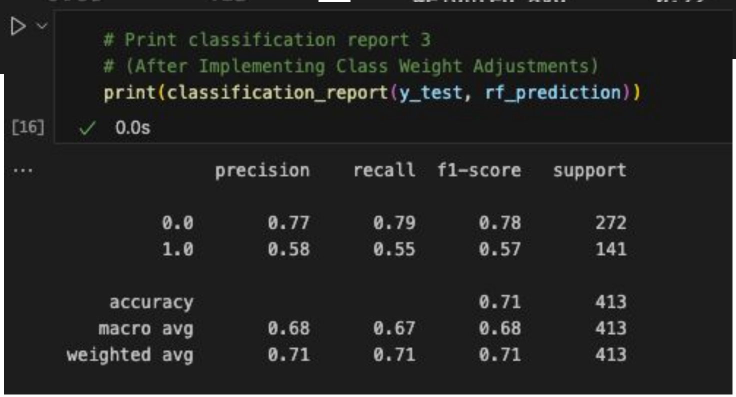
```
              precision    recall  f1-score   support

         0.0       0.79      0.79      0.79       272
         1.0       0.59      0.58      0.59       141

    accuracy                           0.72       413
   macro avg       0.69      0.69      0.69       413
weighted avg       0.72      0.72
```



```
# Print classification report 2
#(After Implementing SelectFromModel Threshold Optimization)
print(classification_report(y_test, rf_prediction))
```

[13] ✓ 0.0s

```
              precision    recall  f1-score   support

         0.0       0.79      0.78      0.78       272
         1.0       0.58      0.60      0.59       141

    accuracy                           0.71       413
   macro avg       0.68      0.69      0.68       413
weighted avg       0.72      0.71      0.72       413
```



```
# Print classification report 3
# (After Implementing Class Weight Adjustments)
print(classification_report(y_test, rf_prediction))
```

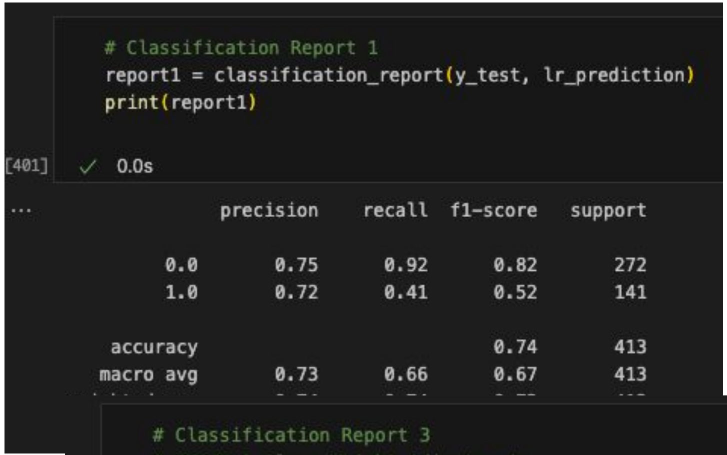
[16] ✓ 0.0s

```
              precision    recall  f1-score   support

         0.0       0.77      0.79      0.78       272
         1.0       0.58      0.55      0.57       141

    accuracy                           0.71       413
   macro avg       0.68      0.67      0.68       413
weighted avg       0.71      0.71      0.71       413
```

## Logistic Regression

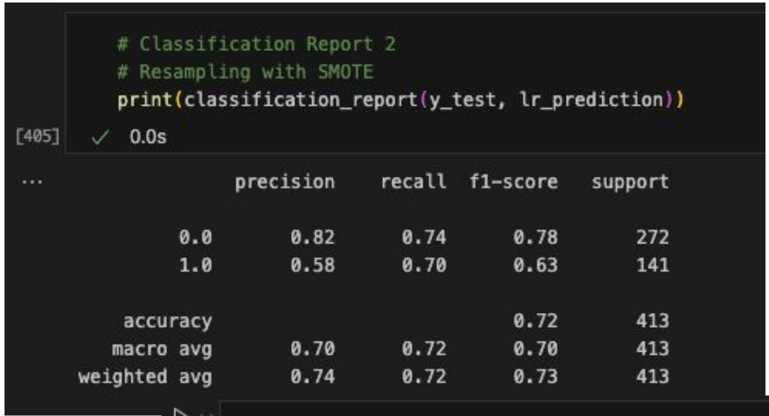
- Top Accuracy Reached: 75%
- Logistic Regression was the first model we tried, and originally ran at 74%
- After some “optimizations”, we were not improving the accuracy and this is when we started working on Random Forest and KNN
- Finally, the optimization that got us to our goal, was adjusting the class weights
- We tried adjusting class weights on two different versions, one resampling imbalanced data with SMOTE and one resampling with ADASYN. Both versions came back at 75%. We believe the class weight made the most impact and the resampling had insignificant impact.

# Logistic Regression Classification Reports



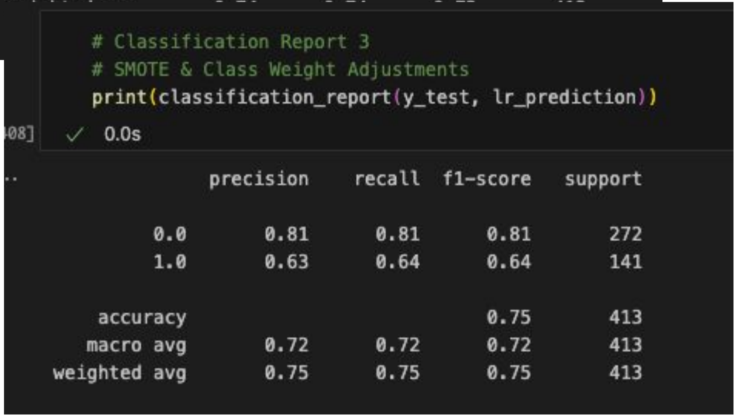
```
# Classification Report 1
report1 = classification_report(y_test, lr_prediction)
print(report1)
```

```
[401]  ✓  0.0s

...               precision    recall  f1-score   support

           0.0       0.75      0.92      0.82       272
           1.0       0.72      0.41      0.52       141

      accuracy                           0.74       413
     macro avg       0.73      0.66      0.67       413
```



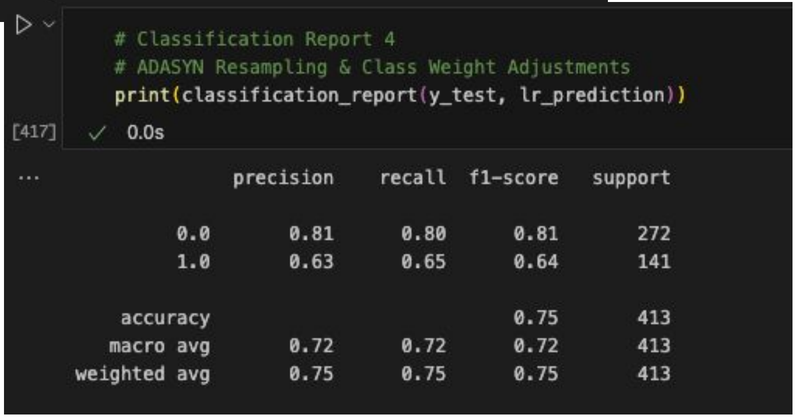
```
# Classification Report 2
# Resampling with SMOTE
print(classification_report(y_test, lr_prediction))
```

```
[405]  ✓  0.0s

...               precision    recall  f1-score   support

           0.0       0.82      0.74      0.78       272
           1.0       0.58      0.70      0.63       141

      accuracy                           0.72       413
     macro avg       0.70      0.72      0.70       413
  weighted avg       0.74      0.72      0.73       413
```



```
# Classification Report 3
# SMOTE & Class Weight Adjustments
print(classification_report(y_test, lr_prediction))
```

```
08]  ✓  0.0s

..                precision    recall  f1-score   support

           0.0       0.81      0.81      0.81       272
           1.0       0.63      0.64      0.64       141

      accuracy                           0.75       413
     macro avg       0.72      0.72      0.72       413
  weighted avg       0.75      0.75      0.75       413
```



```
# Classification Report 4
# ADASYN Resampling & Class Weight Adjustments
print(classification_report(y_test, lr_prediction))
```

```
[417]  ✓  0.0s

...               precision    recall  f1-score   support

           0.0       0.81      0.80      0.81       272
           1.0       0.63      0.65      0.64       141

      accuracy                           0.75       413
     macro avg       0.72      0.72      0.72       413
  weighted avg       0.75      0.75      0.75       413
```

## Conclusion

- Had we understood the nature of the problem we wanted to solve a little better at the beginning, we might have spent less time experimenting with incorrect models
- With a more complete data set that included individual season matchups, or more time to put that data together manually, we could have predicted individual matchups
- We are glad our model reached 75% accuracy, however it was clear during optimization that our data is imbalanced. Even at 75% accuracy, our model is predicting teams who will not make the tournament (0s) better than teams that will (1s)