INST 414 - Final Project
Alex Carcamo and AJ Cooper
May 10, 2021

Introduction

Dataset
2019 Dataset: https://www.basketball-reference.com/leagues/NBA_2019_per_game.html
2020 Dataset: https://www.basketball-reference.com/leagues/NBA_2020_per_game.html

These datasets of a combined 1059 rows and 30 variables were found from the basketball player references website and include statistical information on NBA players.

Variables
- Player: their name (object)
- Pos: their position (category)
- Age: their age (integer)
- Tm: their team (category)
- G: number of games played (integer)
- GS: number of games started (integer)
- MP: average minutes played per game (float)
- FG: field goals per game (float)
- FGA: field goal attempts per game (float)
- FG%: field goal percentage (float)
- 3P: 3-point field goals per game (float)
- 3PA: 3-point field goal attempts per game (float)
- 3P%: 3-point field goal percentage (float)
- 2P: 2-point field goals per game (float)
- 2PA: 2-point field goal attempts per game (float)
- 2P%: 2-point field goal percentage (float)
- eFG%: effective field goal percentage (float)
- FT: free throws per game (float)
- FTA: free throw attempts per game (float)
- FT%: free throw percentage (float)
- ORB: offensive rebounds per game (float)
- DRB: defensive rebounds per game (float)
- TRB: total rebounds per game (float)
- AST: assists per game (float)
- STL: steals per game (float)
- BLK: blocks per game (float)
- TOV: turnovers per game (float)
- PF: personal fouls per game (float)
- PTS: points per game (float)
- AllNBA: if they were awarded a spot on any of the All-NBA teams (categorical)

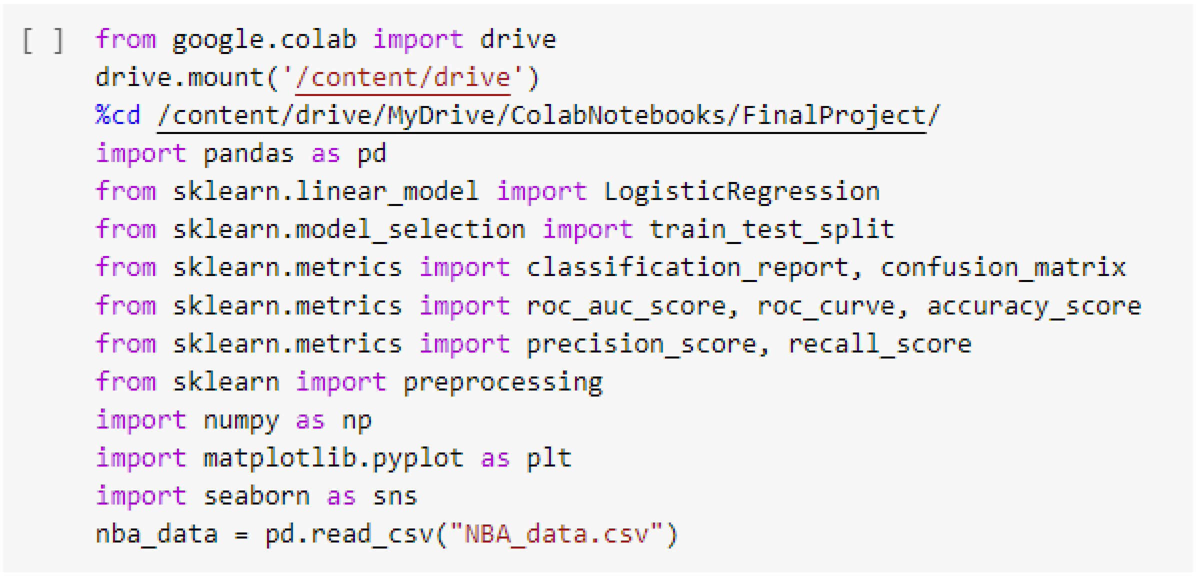
Research Objective:
We are trying to create a model that accurately predicts whether an individual player was awarded a spot on any of the All-NBA teams based on their player statistics for that season. The target variable (AllNBA) is a multiclass categorical variable with potential outcomes of not being on any All-NBA teams ('None'), being a member of the 1st place All-NBA team ('First Team'), being a member of the 2nd place All-NBA team ('Second Team'), and being a member of the 3rd place All-NBA team ('Third Team'). It is important to note that there are exactly 30 total players collectively between all three teams, so it is normal if the majority of our predictive results are 'None'.

1. Is there a correlation between seasonal player statistics and being awarded a spot on the different All-NBA teams?
2. Which variables affect this prediction the most?

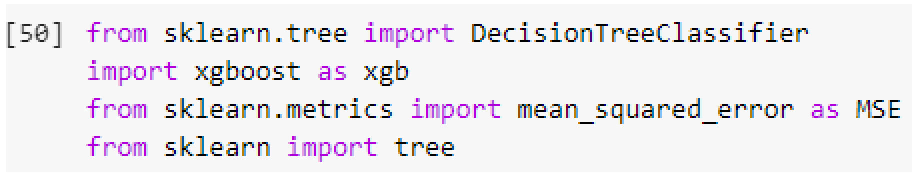
Preparing Data

First, we combined the two datasets, manually added the AllNBA variable as this was not initially included in the dataset, and converted the excel file to a shareable Google Spreadsheets csv file.

Next, the data was loaded in, and we imported all necessary packages for our project.



```
[ ]  from google.colab import drive
     drive.mount('/content/drive')
     %cd /content/drive/MyDrive/ColabNotebooks/FinalProject/
     import pandas as pd
     from sklearn.linear_model import LogisticRegression
     from sklearn.model_selection import train_test_split
     from sklearn.metrics import classification_report, confusion_matrix
     from sklearn.metrics import roc_auc_score, roc_curve, accuracy_score
     from sklearn.metrics import precision_score, recall_score
     from sklearn import preprocessing
     import numpy as np
     import matplotlib.pyplot as plt
     import seaborn as sns
     nba_data = pd.read_csv("NBA_data.csv")
```



```
[50] from sklearn.tree import DecisionTreeClassifier
     import xgboost as xgb
     from sklearn.metrics import mean_squared_error as MSE
     from sklearn import tree
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder
import keras
from keras.models import Sequential
from keras.layers import Dense
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score
```

The dataset was then created and purged of entries containing NaN values. The X dataset was created to include all variables except for Player (as the player name is not predictive or indicative of the target variable) and AllNBA (the target variable). The Y dataset was created to include our target variable, or the value we want to predict.

```
[ ]  nba_data = pd.read_csv("NBA_data.csv")
     clean_nba_data = nba_data.dropna()
     X = clean_nba_data.iloc[:,1:-1]
     y = clean_nba_data.AllNBA
     X.head(10)
```

```
  Pos Age  Tm   G  GS    MP   FG   FGA    FG%   3P  3PA    3P%   2P  2PA    2P%   eFG%   FT  FTA    FT%  ORB  DRB  TRB  AST  STL  BLK  TOV   PF   PTS
0  PF  23 ORL  78  78  33.8  6.0  13.4  0.449  1.6  4.4  0.349  4.5  9.0  0.499  0.507  2.4  3.2  0.731  1.7  5.7  7.4  3.7  0.7  0.7  2.1  2.2  16.0
1  PF  24 ORL  62  62  32.5  5.4  12.4  0.437  1.2  3.8  0.308  4.2  8.5  0.494  0.484  2.4  3.6  0.674  1.7  5.9  7.7  3.7  0.8  0.6  1.6  2.0  14.4
2  PG  22 IND  50   0  12.9  2.1   5.2  0.401  0.9  2.5  0.339  1.2  2.7  0.459  0.483  0.8  1.0  0.820  0.1  1.2  1.3  1.7  0.4  0.3  0.8  1.4   5.9
3  PG  23 IND  66  33  24.5  3.5   8.5  0.414  1.3  3.3  0.394  2.2  5.2  0.427  0.491  1.1  1.3  0.851  0.3  2.0  2.4  3.4  0.8  0.2  1.3  1.8   9.5
4  SF  25 OKC  61   1  11.4  1.5   3.5  0.423  0.5  1.6  0.320  1.0  1.9  0.513  0.498  0.4  0.6  0.750  0.2  1.7  1.9  0.3  0.3  0.2  0.4  1.1   4.0
5  SF  26 OKC  55   6  15.8  2.2   4.8  0.468  0.9  2.3  0.375  1.4  2.5  0.556  0.559  0.9  1.2  0.773  0.3  1.6  1.8  0.7  0.4  0.4  0.8  1.4   6.3
6  SF  21 CHI  11   0  10.2  1.1   2.5  0.429  0.5  1.4  0.400  0.5  1.2  0.462  0.536  0.2  0.4  0.500  0.6  0.3  0.9  0.4  0.4  0.0  0.2  1.5   2.9
7  PF  22 WAS  33   2  11.2  1.1   2.8  0.380  0.6  1.8  0.311  0.5  0.9  0.516  0.484  0.3  0.5  0.667  0.2  1.2  1.4  0.5  0.2  0.1  0.2  1.5   3.0
8   C  32 BOS  68  68  29.0  5.7  10.6  0.535  1.1  3.0  0.360  4.6  7.6  0.604  0.586  1.1  1.4  0.821  1.8  5.0  6.7  4.2  0.9  1.3  1.5  1.9  13.6
9   C  33 PHI  67  61  30.2  4.8  10.6  0.450  1.5  4.2  0.350  3.3  6.4  0.516  0.520  0.9  1.2  0.763  1.5  5.3  6.8  4.0  0.8  0.9  1.2  2.1  11.9
```

The original dataset inputs some variables as the ‘object’ data type and we want them to be the ‘category’ data type, so changes were made accordingly. They were then converted to integers to reflect the category codes to make analyzing these variables easier for our algorithms.

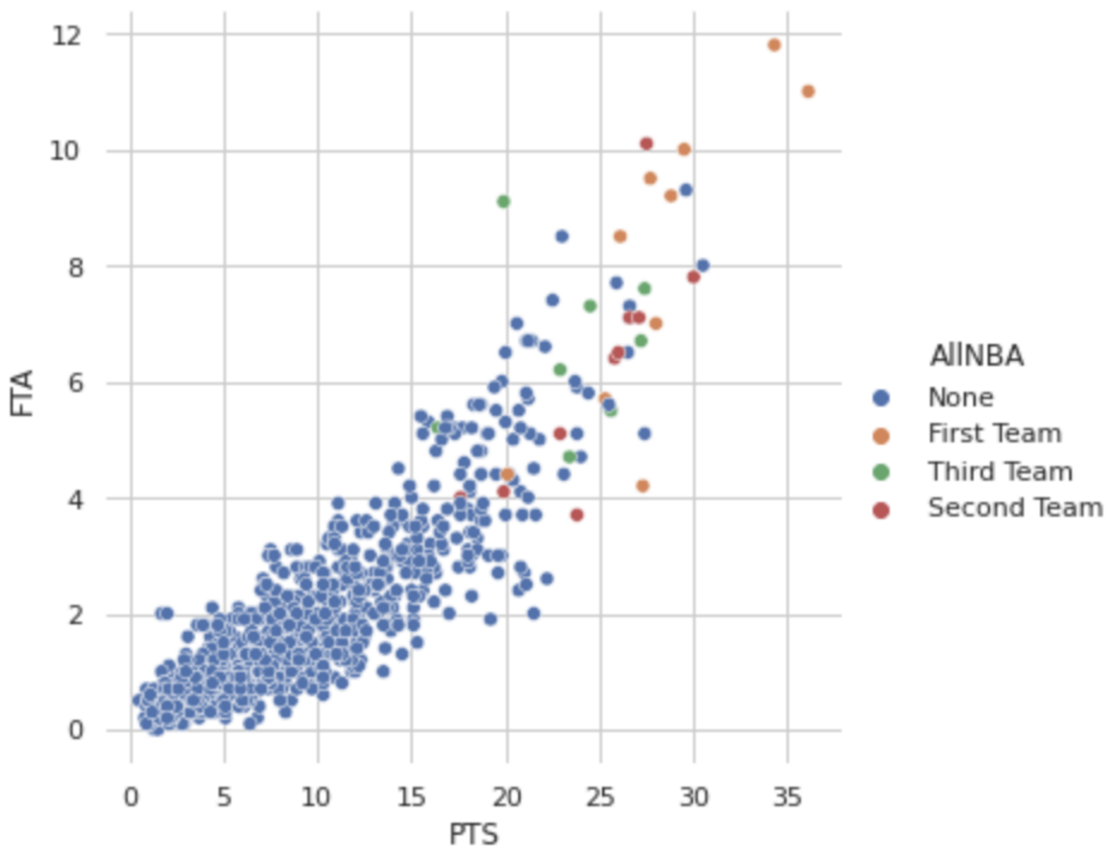
```
[32]  X["Pos"] = X["Pos"].astype('category')
      X["Tm"] = X["Tm"].astype('category')
      y = y.astype('category')

      X["Pos"] = X["Pos"].cat.codes
      X["Tm"] = X["Tm"].cat.codes
      y = y.cat.codes
```

An important note to make is that the AllNBA category codes were computed alphabetically.
0 = ‘First Team’
1 = ‘None’

2 = ‘Second Team’
3 = ‘Third Team’

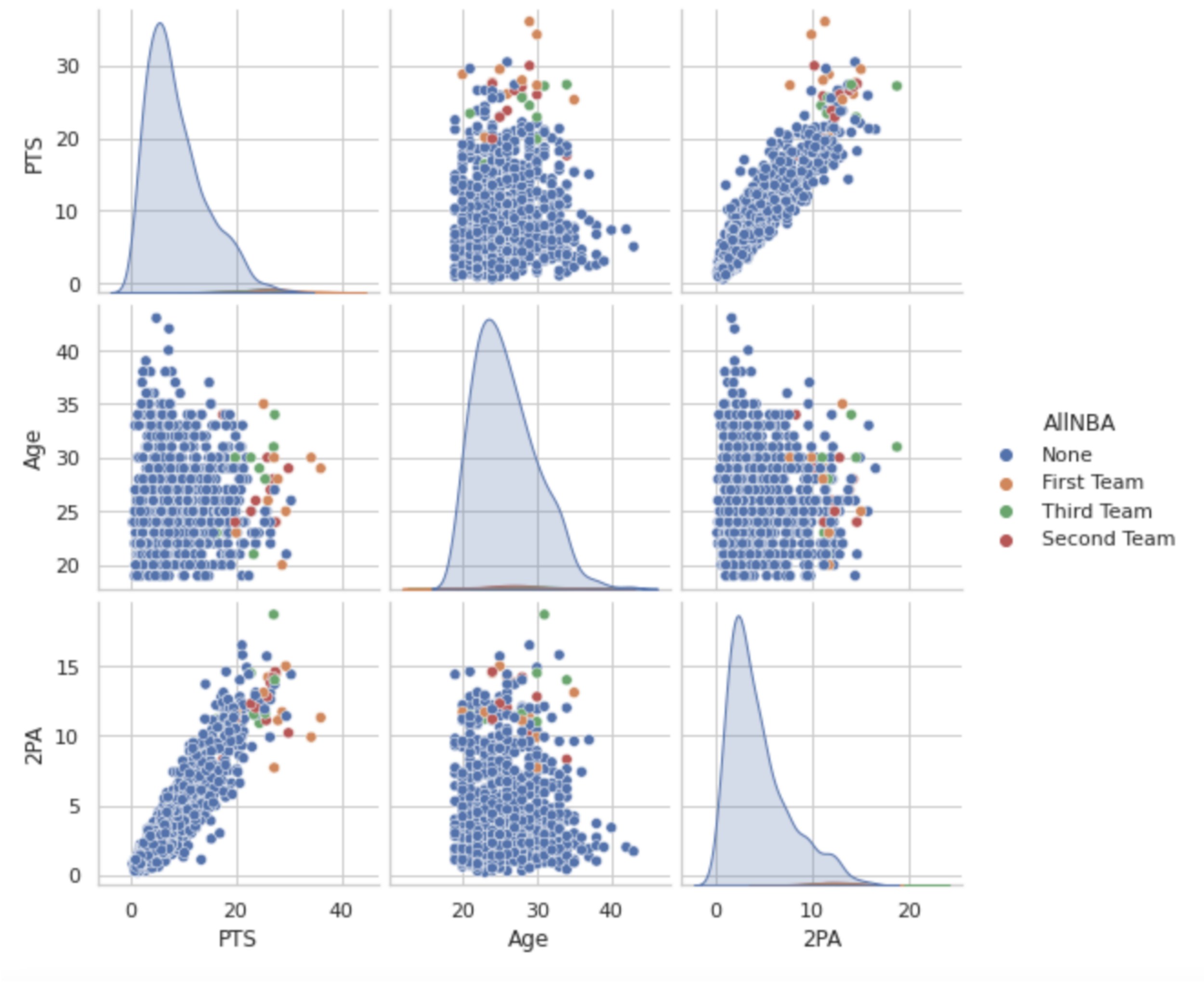
Initial Visualizations

Scatter plot of FTA vs PTS



We initially looked at our data with these variables because we felt as if the best NBA players scored the most points, and shot the most free throws. We wanted to see how this data looked when plotted. The graph shows our prediction was pretty correct as the higher you go on both, the more AllNBA players you see.

Pair Plot Charts

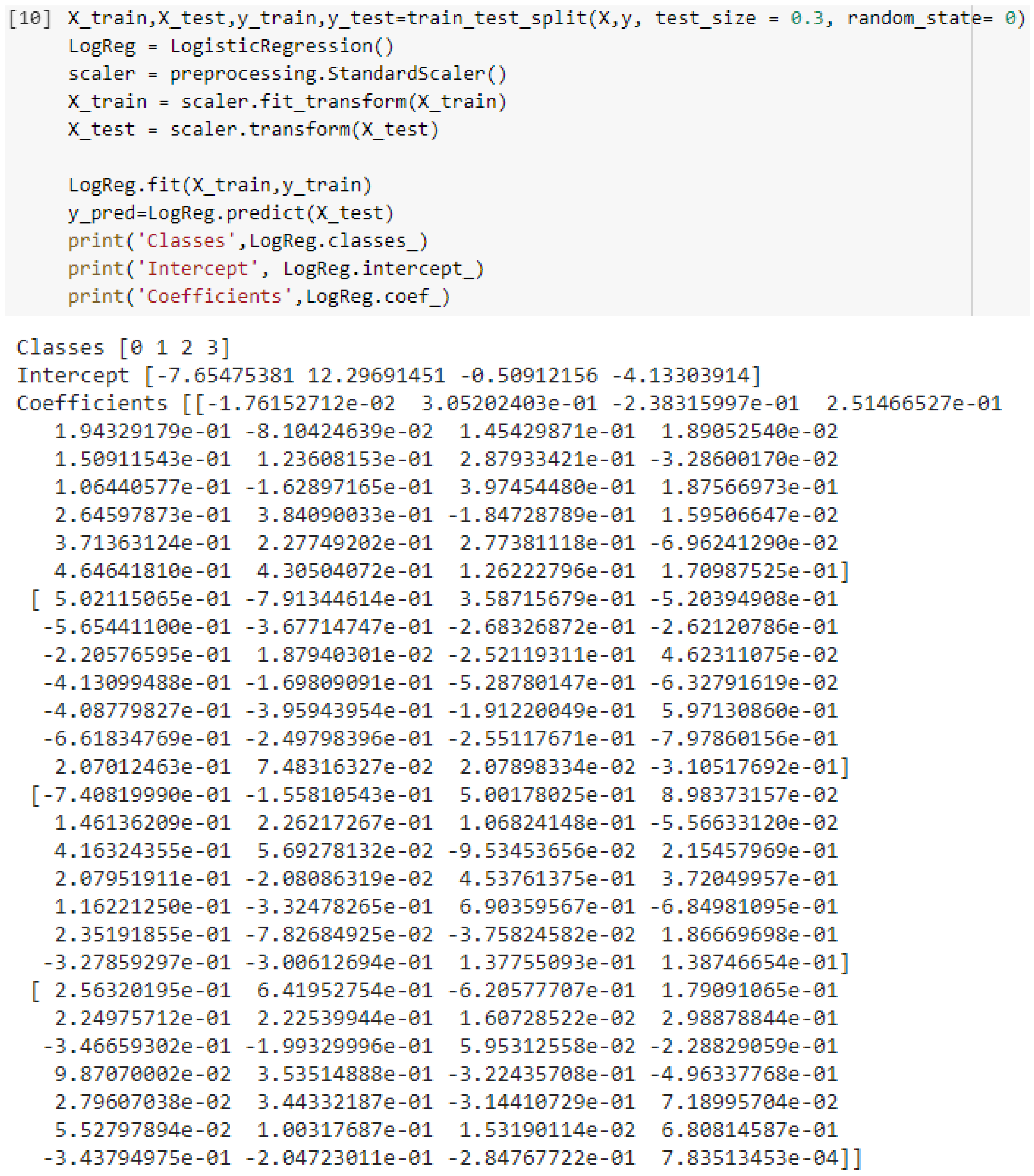


We created pair plot charts to see how some of the other variables can be used to predict the All NBA variable. Here we see that points and 2PA can be a good predictor, but age is difficult to tell. It is hard to say where the bulk of the All NBA players lie with age.

Performance Models

Logistic Regression

To create a logistic regression model, we split the X and y datasets into train and test subsets with a split of 0.3. We then fit the train data to a logistic regression model and predicted the y_test using this model. The code for this process and some statistics on the model are output below:



```
[10] X_train,X_test,y_train,y_test=train_test_split(X,y, test_size = 0.3, random_state= 0)
     LogReg = LogisticRegression()
     scaler = preprocessing.StandardScaler()
     X_train = scaler.fit_transform(X_train)
     X_test = scaler.transform(X_test)

     LogReg.fit(X_train,y_train)
     y_pred=LogReg.predict(X_test)
     print('Classes',LogReg.classes_)
     print('Intercept', LogReg.intercept_)
     print('Coefficients',LogReg.coef_)
```

```
Classes [0 1 2 3]
Intercept [-7.65475381 12.29691451 -0.50912156 -4.13303914]
Coefficients [[-1.76152712e-02  3.05202403e-01 -2.38315997e-01  2.51466527e-01
   1.94329179e-01 -8.10424639e-02  1.45429871e-01  1.89052540e-02
   1.50911543e-01  1.23608153e-01  2.87933421e-01 -3.28600170e-02
   1.06440577e-01 -1.62897165e-01  3.97454480e-01  1.87566973e-01
   2.64597873e-01  3.84090033e-01 -1.84728789e-01  1.59506647e-02
   3.71363124e-01  2.27749202e-01  2.77381118e-01 -6.96241290e-02
   4.64641810e-01  4.30504072e-01  1.26222796e-01  1.70987525e-01]
 [ 5.02115065e-01 -7.91344614e-01  3.58715679e-01 -5.20394908e-01
  -5.65441100e-01 -3.67714747e-01 -2.68326872e-01 -2.62120786e-01
  -2.20576595e-01  1.87940301e-02 -2.52119311e-01  4.62311075e-02
  -4.13099488e-01 -1.69809091e-01 -5.28780147e-01 -6.32791619e-02
  -4.08779827e-01 -3.95943954e-01 -1.91220049e-01  5.97130860e-01
  -6.61834769e-01 -2.49798396e-01 -2.55117671e-01 -7.97860156e-01
   2.07012463e-01  7.48316327e-02  2.07898334e-02 -3.10517692e-01]
 [-7.40819990e-01 -1.55810543e-01  5.00178025e-01  8.98373157e-02
   1.46136209e-01  2.26217267e-01  1.06824148e-01 -5.56633120e-02
   4.16324355e-01  5.69278132e-02 -9.53453656e-02  2.15457969e-01
   2.07951911e-01 -2.08086319e-02  4.53761375e-01  3.72049957e-01
   1.16221250e-01 -3.32478265e-01  6.90359567e-01 -6.84981095e-01
   2.35191855e-01 -7.82684925e-02 -3.75824582e-02  1.86669698e-01
  -3.27859297e-01 -3.00612694e-01  1.37755093e-01  1.38746654e-01]
 [ 2.56320195e-01  6.41952754e-01 -6.20577707e-01  1.79091065e-01
   2.24975712e-01  2.22539944e-01  1.60728522e-02  2.98878844e-01
  -3.46659302e-01 -1.99329996e-01  5.95312558e-02 -2.28829059e-01
   9.87070002e-02  3.53514888e-01 -3.22435708e-01 -4.96337768e-01
   2.79607038e-02  3.44332187e-01 -3.14410729e-01  7.18995704e-02
   5.52797894e-02  1.00317687e-01  1.53190114e-02  6.80814587e-01
  -3.43794975e-01 -2.04723011e-01 -2.84767722e-01  7.83513453e-04]]
```

To help evaluate the quality of our model, we generated some statistics on the model's accuracy, precision, and recall. A confusion matrix was also developed.

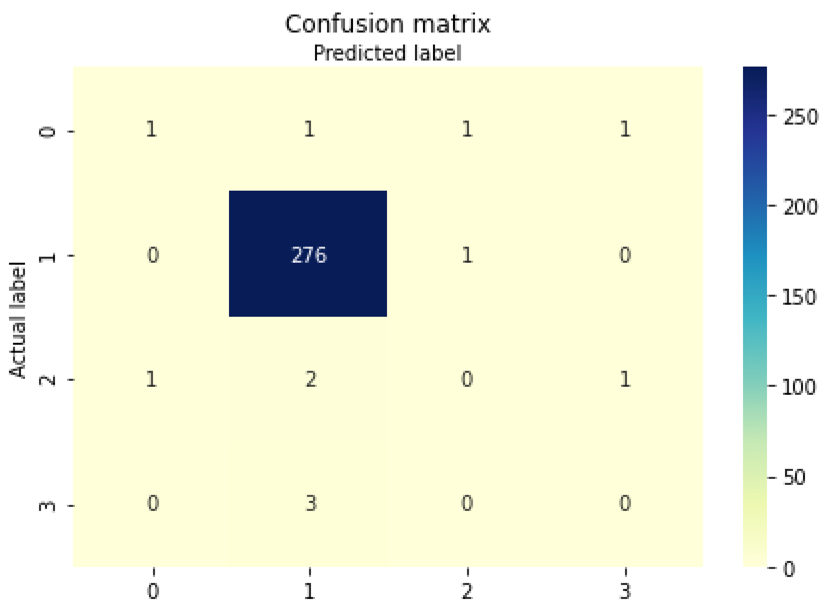
```
[11] print("Accuracy", LogReg.score(X_test, y_test))
     print(classification_report(y_test, LogReg.predict(X_test)))
```

```
Accuracy 0.9618055555555556
              precision    recall  f1-score   support

           0       0.50      0.25      0.33         4
           1       0.98      1.00      0.99       277
           2       0.00      0.00      0.00         4
           3       0.00      0.00      0.00         3

    accuracy                           0.96       288
   macro avg       0.37      0.31      0.33       288
weighted avg       0.95      0.96      0.95       288
```

```
[12] conf_mat = confusion_matrix(y_test, y_pred)
     conf_mat
```

```
array([[  1,   1,   1,   1],
       [  0, 276,   1,   0],
       [  1,   2,   0,   1],
       [  0,   3,   0,   0]])
```

```
[48] categories = [0,1,2,3]
     fig, ax= plt.subplots()
     plt.xticks([0,1,2,3], categories)
     plt.yticks([0,1,2,3],categories)
     sns.heatmap(pd.DataFrame(conf_mat), annot = True, cmap= 'YlGnBu', fmt = 'g')
     ax.xaxis.set_label_position("top")
     plt.tight_layout()
     plt.title('Confusion matrix', y = 1.1)
     plt.ylabel('Actual label')
     plt.xlabel('Predicted label')
```



```
[49] print('Accuracy', accuracy_score(y_test, y_pred))
     print('Precision', precision_score(y_test, y_pred, average= 'weighted'))
     print('Recall' , recall_score(y_test, y_pred, average = 'weighted'))
```

```
Accuracy 0.9618055555555556
Precision 0.9482860520094563
Recall 0.9618055555555556
```

The accuracy is calculated by dividing the number of correct predictions by the total number of predictions. With an accuracy of 0.9618, the logistic regression model appears to be fairly accurate. It is important to note that even though the majority of predictions may be correct, the severity of making an incorrect prediction isn't weighed into accuracy. It is essential to evaluate which values were misclassified.

Precision is measured by dividing the number of true positives by the total predictive positives. Precision is a good measurement when the cost of a false positive is high. A precision value of 0.9483 is fairly high.

Recall is the number of true positives divided by the total number of actual positives. Recall is a good measurement when the cost of a false negative is high. A recall value of 0.9618 is fairly high.

Decision Tree Classification

The Decision Tree Classification model was created using the DecisionTreeClassifier function and an entropy criterion. Once again, accuracy, precision, and recall were calculated to evaluate the difference between our models. A confusion matrix was also developed for further evaluation.

```
X_train,X_test,y_train,y_test=train_test_split(X,y, test_size = 0.3, random_state= 0)
clf = DecisionTreeClassifier(criterion='entropy', random_state=99)
clf.fit(X_train,y_train)
y_pred=clf.predict(X_test)

print('Accuracy', accuracy_score(y_test, y_pred))
print('Precision', precision_score(y_test, y_pred, average= 'weighted'))
print('Recall' , recall_score(y_test, y_pred, average = 'weighted'))
```

```
Accuracy 0.96875
Precision 0.9630127439267224
Recall 0.96875
```

```
[68] conf_mat = confusion_matrix(y_test, y_pred)
     conf_mat
```

```
array([[  1,   1,   0,   2],
       [  0, 277,   0,   0],
       [  2,   1,   1,   0],
       [  0,   0,   3,   0]])
```

With an accuracy and recall of 0.96875 and a precision of 0.9630, these high values are indicative of a good predictive model.

A decision tree plot was created with the following code. A png of the output will be submitted with our assignment separately as trying to fit the image into this document is too blurry to clearly see.

```
[69] fig, ax = plt.subplots(figsize = (25,25))
     tree.plot_tree(decision_tree = clf, max_depth= 5, fontsize = 12)
```

XGBoost

The XGBoost model was created using the XGBClassifier function and a multisolution objective in order to account for all categories in our target variable. Once again, accuracy, precision, and recall were calculated to evaluate the difference between our models. A confusion matrix was also developed for further evaluation.

```
[58] xg_cl = xgb.XGBClassifier(objective = 'multi:softprob', max_depth = 10, n_estimators = 20, seed = 99)
     xg_cl.fit(X_train, y_train)
     preds = xg_cl.predict(X_test)

     accuracy = float(np.sum(preds==y_test))/y_test.shape[0]
     print('Accuracy: %f' %(accuracy))
     print('Precision', precision_score(y_test, preds, average= 'weighted'))
     print('Recall' , recall_score(y_test, preds, average = 'weighted'))
```

```
     Accuracy: 0.961806
     Precision 0.9416568857589984
     Recall 0.9618055555555556
```

```
[63] conf_mat = confusion_matrix(y_test, preds)
     conf_mat
```

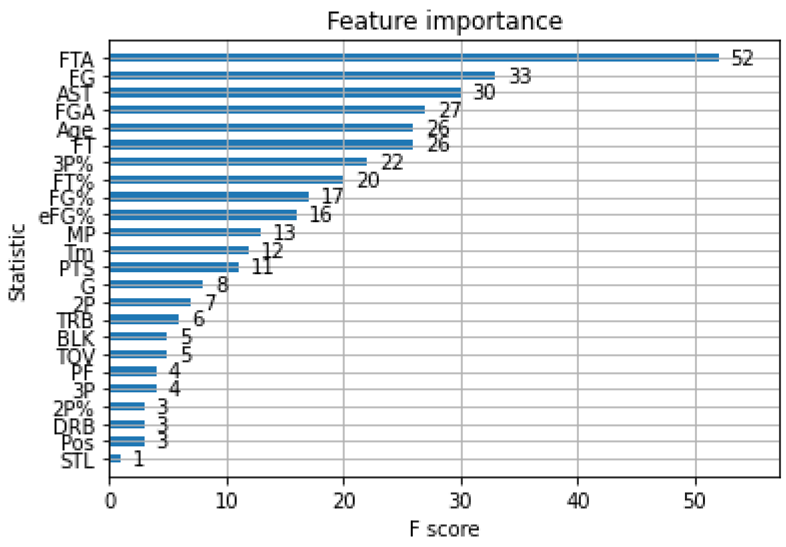
```
     array([[  1,   2,   1,   0],
            [  0, 276,   0,   1],
            [  1,   3,   0,   0],
            [  0,   3,   0,   0]])
```

With an accuracy and recall of 0.9618 and a precision of 0.9416, these high values are indicative of a good predictive model.

We also created an xgb plot to measure the importance of each variable in our predictive model. The weight of each individual variable will help us answer our second research question.

```
[ ]  xgb.plot_importance(xg_cl, ylabel = 'Statistic', height = 0.5)
```

```
     <matplotlib.axes._subplots.AxesSubplot at 0x7fe3579f7ed0>
```

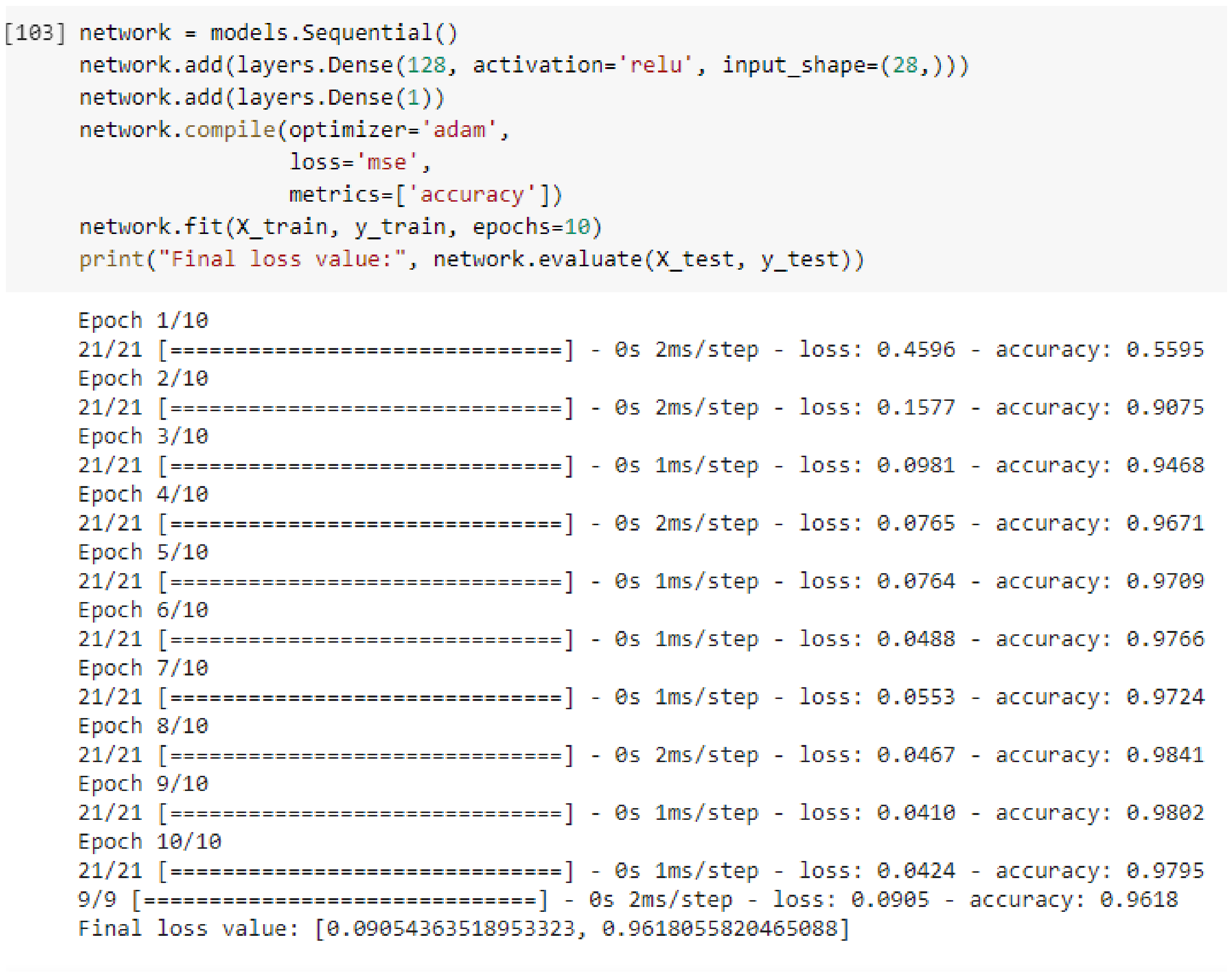


According to this chart, the five most influential variables on predicting whether a player will be on an All-NBA team are: free throw attempts per game (FTA), field goals per game (FG), assists per game (AST), field goal attempts per game (FGA), and the player's age (Age). The five least important variables are steals per game (STL), the player's position (Pos), defensive rebounds per game (DRB), 2-point field goal percentage (2P%), and 3-point field goals per game (3P).

If these models were to be recalculated with different combinations of variables, it may be viable to remove some of the variables with the lowest feature importance as they may not be heavily correlated with our target variable.

Keras Neural Network

The first neural network model was created using Keras. Ten epochs, or iterations through the dataset, were compiled with an input layer of 28 to signify the number of variables being used. Loss and accuracy values were also developed to evaluate the network's results.



```
[103] network = models.Sequential()
      network.add(layers.Dense(128, activation='relu', input_shape=(28,)))
      network.add(layers.Dense(1))
      network.compile(optimizer='adam',
                      loss='mse',
                      metrics=['accuracy'])
      network.fit(X_train, y_train, epochs=10)
      print("Final loss value:", network.evaluate(X_test, y_test))
```

```
Epoch 1/10
21/21 [==============================] - 0s 2ms/step - loss: 0.4596 - accuracy: 0.5595
Epoch 2/10
21/21 [==============================] - 0s 2ms/step - loss: 0.1577 - accuracy: 0.9075
Epoch 3/10
21/21 [==============================] - 0s 1ms/step - loss: 0.0981 - accuracy: 0.9468
Epoch 4/10
21/21 [==============================] - 0s 2ms/step - loss: 0.0765 - accuracy: 0.9671
Epoch 5/10
21/21 [==============================] - 0s 1ms/step - loss: 0.0764 - accuracy: 0.9709
Epoch 6/10
21/21 [==============================] - 0s 1ms/step - loss: 0.0488 - accuracy: 0.9766
Epoch 7/10
21/21 [==============================] - 0s 1ms/step - loss: 0.0553 - accuracy: 0.9724
Epoch 8/10
21/21 [==============================] - 0s 2ms/step - loss: 0.0467 - accuracy: 0.9841
Epoch 9/10
21/21 [==============================] - 0s 1ms/step - loss: 0.0410 - accuracy: 0.9802
Epoch 10/10
21/21 [==============================] - 0s 1ms/step - loss: 0.0424 - accuracy: 0.9795
9/9 [==============================] - 0s 2ms/step - loss: 0.0905 - accuracy: 0.9618
Final loss value: [0.09054363518953323, 0.9618055820465088]
```

This network resulted in an accuracy of 0.9618, this is indicative of a pretty successful model in terms of predictive probability.

Loss is a measurement of the prediction error of a neural net; it is a summation of the errors made for each epoch. The total test loss for this network is 0.0905. Unlike accuracy, precision, and recall, lower test loss scores are resulted from a better model.

Pytorch Neural Network

Comparing Results

All of our findings were similar. The logistic regression model and XGBoost were similar in accuracy. Both models were around 0.96 accurate in predicting the AllNBA value. The Decision Tree Classification was not as precise as the logistic regressions model. It had similar accuracy but a slightly smaller precision score. The Neural Network was the least accurate, and this one is the most difficult to use for our prediction. The accuracy metric starts off really bad but improves as the Epoch increases.