Rookie 5 Year Predictions

Seraj, Isaac, and Chris

Overview

In this Project we used various machine learning algorithms to accurately identify NBA rookies that had careers longer than 5 years. We start by importing neccassary packeges, cleaning/exploring the data and building our baseline model. We want to know what in game statitics for players in their rookie year lead to a career longer than 5 years to help our stakeholders get a better understanding on what stats they should focus on for their rookie players. Some of our recomendations include focusing on points per game, games played, rebounds per game, and blocks per game.

Business Problem

Every year NBA teams draft rookies. To predict if those rookies will be successful in the long run can be difficult since so many factors can play a role. Our goal is to limit those factors to just specific in-game stats to aid General Managers in their decision on what to focus on in their rookies.

We will predict which NBA players to keep by classifying them based on their stats of their first season (5-years).

Our goal is to help NBA front offices make better informed personnel decisions on their rookie players based on the historical data of NBA rookies from 1980 to 2012. These personnel decisions include but are not limited to:

1. Contract extensions
2. Trading players
3. Releasing players from their contracts

For context: The average NBA career is 4.5 to 4.8 years. NBA rookie contracts are 4 years in total so many draftees in the first round are expected to be able contributors (there are exceptions for very young players and foreign players).

To summarize: Based on a rookie’s stats, which specific stats will help us best predict their having an NBA career longer than 5 years?

Stakeholders

Our stakeholders are NBA front offices.

Specifically, NBA general managers can find this useful because these are the people in charge of team operations and personnel. They hold the ability to trade, release, and sign players in free agency or to extend contracts. A player’s length of career is not solely decided by the stats of their rookie season, but this insight can help GMs get a better picture of which players have a better chance of “sticking” in the league than others.

Data

We have used two datasets to make our predictive models, 'nba-players.csv' from Kaggle and 'NBA Rookies by Year.xlsx' from DataWorld. We concatted both dataset together to get the full array of statistics we were interested in exploring and saved it as nba.csv. Features are completely performance based and does not include some features that may lead to bias, such as Race. All features are an average per game statistic except for name(Player Name), gp(Games Played), target_5yrs(Career Duration > 5 Years), and Year Drafted. The dataframe column and descriptions are listed below:

Columns	Descriptions
name	Player Name
gp	Games played
min	Minutes Played
pts	Points per Game
fgm	Field Goals made
fga	Field Goal Attempts
fg	Field Goal Percentage
3p_made	3 Points made
3pa	3 Point Attempts
3p	3 Point Percentage
ftm	Free Throw made
fta	Free Throw Attempts

Columns	Descriptions
ft	Free Throw Percentage
oreb	Offensive Rebounds
dreb	Defensive Rebounds
reb	Rebounds
ast	Assists
stl	Steals
blk	Blocks
tov	Turnovers
target_5yrs	Career Duration > 5 years
Year Drafted	Year Drafted

Data Preparation

Imports

In [449]:
```
# imports
import pandas as pd
import numpy as np
import math
import seaborn as sns
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.linear_model import LogisticRegression, LogisticRegressionCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn import tree
from xgboost import XGBClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, pl
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

In [450]:
```
df = pd.read_csv('Dataset/nba.csv', index_col=0)
```

Dropping Unnecessary Rows/Players

In [451]:
```
# dropping duplicate names
df.drop_duplicates(subset='name', keep=False, inplace=True)
```

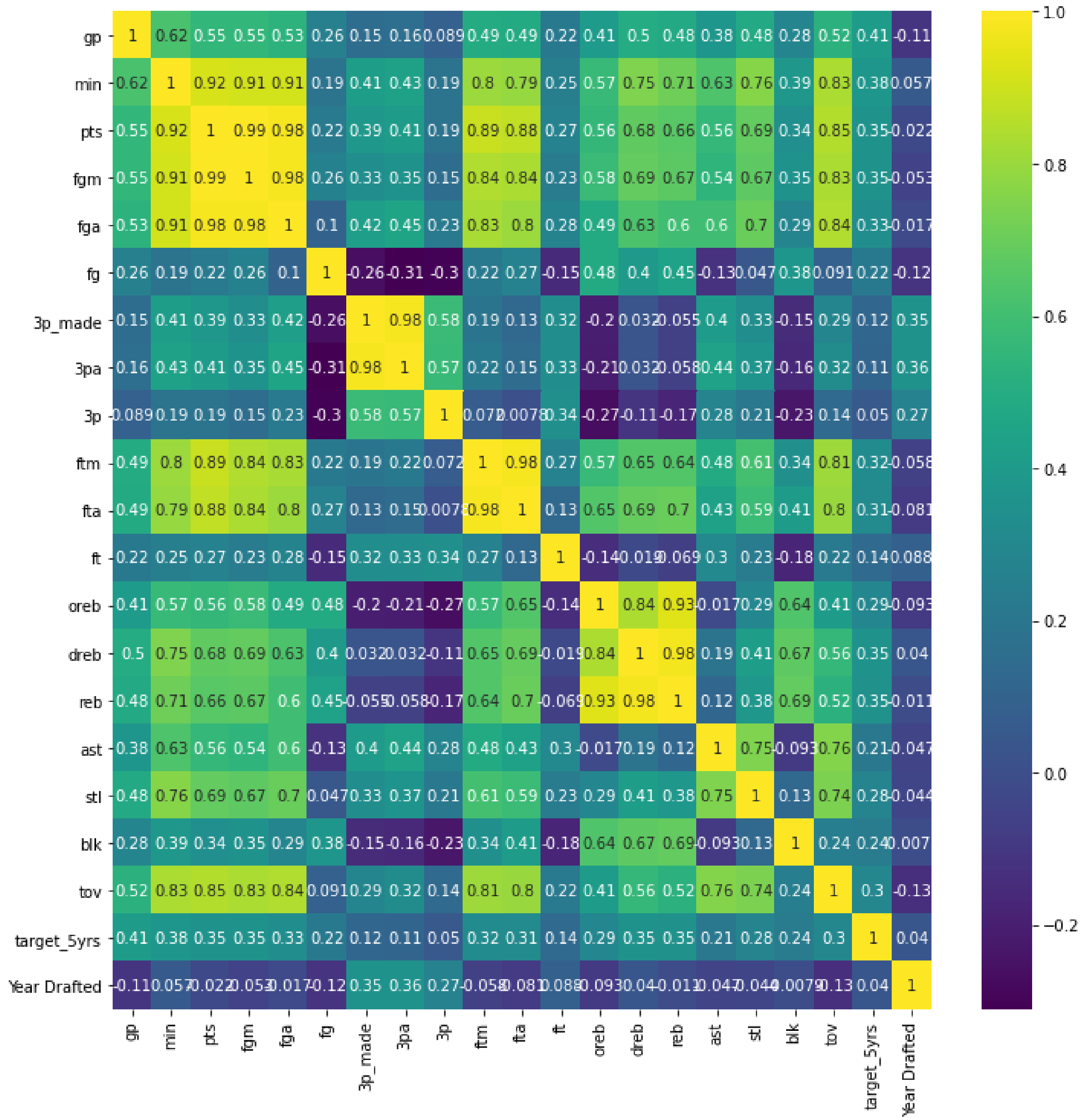
In [452]:
```
# dropping all rookies not in NBA for 5 years before creation of dataset
df = df[df['Year Drafted'] < 2013]
```

Determining correlation among features

In [453]:

```
plt.figure(figsize=(12,12))
sns.heatmap(df.corr(), annot=True, cmap='viridis')
```

Out[453]: <AxesSubplot:>



Many of the features available to us are very similar and highly correlated with each other. For this reason, we will drop the others but keep one. For example, there are stats for Offensive Rebounds, Defensive Rebounds, and Rebounds (total). In this case, we will keep the Rebounds (total).

Similarly --> FieldGoalsMade (FGM), FieldGoalAttempts(FGA), FieldGoalPercentage(FG%) --> keeping FG% even though FGM is more highly correlated with target_5yrs. We chose FG% because it is a measure of scoring efficiency while FGM is highly correlated with FGA. FGA can be explained by the role the rookie has on their team and we know that each rookie's role is different depending on the talent of the roster they are on.

3P_made, 3PA, 3P%, --> we will keep 3P% to stay consistent in our measures of a players' scoring efficiency.

Finally... FreeThrowsMade (FTM), FreeThrowsAttempted (FTA), FreeThrowPercentage(FT) --> keep FT to maintain consistency for player's scoring efficiency.

In [454]:
```
df = df.drop(['name','fgm','fga','3p_made','3pa','ftm','fta','oreb','dreb','Year Drafte
```

Data Exploration

In [455]:
```
df['target_5yrs'].value_counts(normalize=True)
```

Out[455]:
```
1    0.677074
0    0.322926
Name: target_5yrs, dtype: float64
```

In [456]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1121 entries, 98 to 1327
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   gp           1121 non-null   int64
 1   min          1121 non-null   float64
 2   pts          1121 non-null   float64
 3   fg           1121 non-null   float64
 4   3p           1121 non-null   float64
 5   ft           1121 non-null   float64
 6   reb          1121 non-null   float64
 7   ast          1121 non-null   float64
 8   stl          1121 non-null   float64
 9   blk          1121 non-null   float64
 10  tov          1121 non-null   float64
 11  target_5yrs  1121 non-null   int64
dtypes: float64(10), int64(2)
memory usage: 113.9 KB
```

We will be able to move forward with our modeling without artifically balancing the datasets because the target classifications are not significantly imbalanced. Neither will we need to make any major adjustments to the rest of the data as it is has no other major issues.

Model 1 - Baseline Decision Tree

For our baseline model, we decided to implement a decision tree and used field goal percentage (ft) as our feature. We chose to go with a percentage because statistics like games played, points per game, and minutes per game contain bias. This is because not every rookie's situation is the same when they come into the league. High draft picks (ex. 1-5 overall) usually have different roles to their team than players drafted (20-30 overall). Field goal percentage is a measure of a player's efficiency because it measures how many total shots they made (excluding free throws) divided by the total amount of shots they took (excluding free throws).

In [457]:
```
X = df[['fg']]
y = df['target_5yrs']
```

Splitting into training and validation sets

In [458]:
```python
# splitting the training data into training and validation data
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.25, rar
```

Creating, fitting, and predicting on baseline model

In [459]:
```python
base_tree = DecisionTreeClassifier(random_state=42)
```

In [460]:
```python
base_tree.fit(X_train, y_train)
```

Out[460]:
```
DecisionTreeClassifier(random_state=42)
```

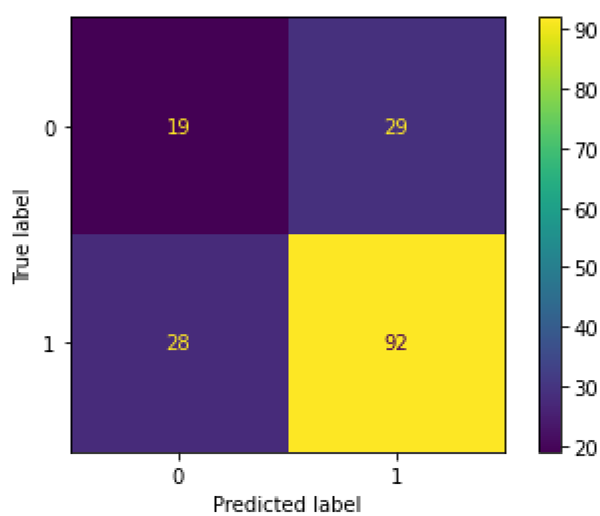
In [461]:
```python
tree_preds = base_tree.predict(X_val)
```

In [462]:
```python
print(classification_report(y_val, tree_preds))
```

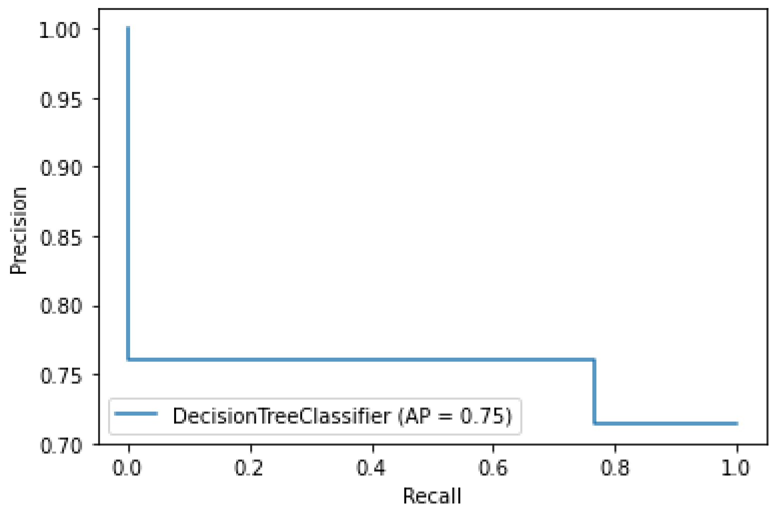
```
              precision    recall  f1-score   support

           0       0.40      0.40      0.40        48
           1       0.76      0.77      0.76       120

    accuracy                           0.66       168
   macro avg       0.58      0.58      0.58       168
weighted avg       0.66      0.66      0.66       168
```

In [463]:
```python
plot_confusion_matrix(base_tree,X_val,y_val)
```

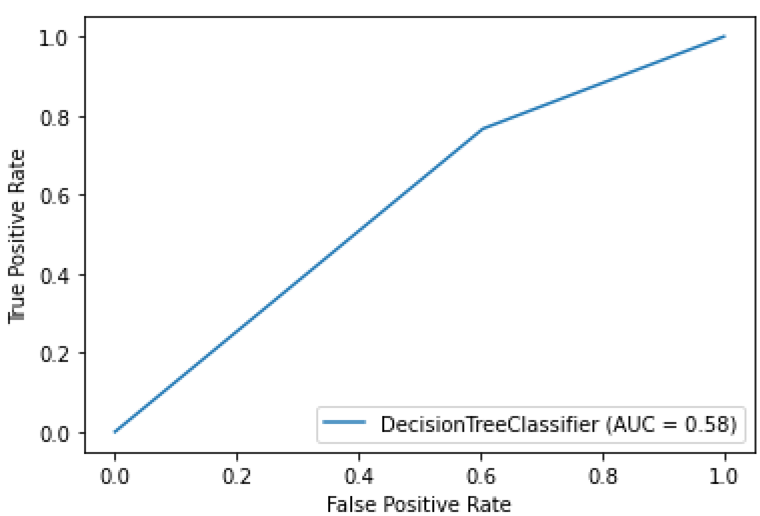
Out[463]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x28b5413ff40>
```



In [464]:
```
plot_precision_recall_curve(base_tree, X_val, y_val )
```
Out[464]:
```
<sklearn.metrics._plot.precision_recall_curve.PrecisionRecallDisplay at 0x28b541ef0a0
>
```



In [465]:
```
plot_roc_curve(base_tree,X_val,y_val)
```
Out[465]:
```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x28b5414dfd0>
```



Interpreting the Baseline Model

The baseline had an accuracy score of .59, hardly better than random chance. The ROC curve is nearly a straight line which shows poor performance.

Train | Test | Validation Split (all features)

In [466]:
```
# split into X and y
X = df.drop('target_5yrs',axis=1)
y = df['target_5yrs']
```

In [467]:
```
# first, split into training and testing data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=4

# splitting the training data into training and validation data",
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.25, rar
```

Moving onto Iterative Modeling Process

We will try out various applicable models, and according to their accuracy scores, determine which one to move forward and optimize with. We chose accuracy as the metric for our models as we were neither focused on minimizing False Positives or False Negatives, but getting as many accurate predictions as possible.

Model 1- Logistic Regression

In [468]:
```
log = LogisticRegression()
# scaler object
scaler = StandardScaler()
# knn operations
log_operations = [('scaler', scaler), ('log', log)]
# import pipeline object
log_pipe = Pipeline(log_operations)
```

In [469]:
```
log_pipe.fit(X_train, y_train)
```

Out[469]:
```
Pipeline(steps=[('scaler', StandardScaler()), ('log', LogisticRegression())])
```

In [470]:
```
log_pred = log_pipe.predict(X_val)
```

In [471]:
```
print(classification_report(y_val, log_pred))
```

```
              precision    recall  f1-score   support

           0       0.61      0.47      0.53        74
           1       0.77      0.85      0.81       150

    accuracy                           0.73       224
   macro avg       0.69      0.66      0.67       224
weighted avg       0.72      0.73      0.72       224
```

Model 2 - K-Nearest Neighbors

KNN Pipeline

In [472]:
```
knn = KNeighborsClassifier()
# knn operations
knn_operations = [('scaler', scaler), ('knn', knn)]
# import pipeline object
knn_pipe = Pipeline(knn_operations)
```

In [473]:
```
knn_pipe.fit(X_train, y_train)
```

Out[473]:
```
Pipeline(steps=[('scaler', StandardScaler()), ('knn', KNeighborsClassifier())])
```

In [474]:
```
knn_pred = knn_pipe.predict(X_val)
```

In [475]:
```
print(classification_report(y_val, knn_pred))
```

```
              precision    recall  f1-score   support

           0       0.53      0.43      0.48        74
           1       0.74      0.81      0.78       150

    accuracy                           0.69       224
   macro avg       0.64      0.62      0.63       224
weighted avg       0.67      0.69      0.68       224
```

Model 3 - Decision Tree

In [476]:
```
dtc = DecisionTreeClassifier()
```

In [477]:
```
dtc.fit(X_train, y_train)
```

Out[477]:
```
DecisionTreeClassifier()
```

In [478]:
```
dtc_pred = dtc.predict(X_val)
```

In [479]:
```
print(classification_report(y_val, dtc_pred))
```

```
              precision    recall  f1-score   support

           0       0.51      0.50      0.51        74
           1       0.76      0.77      0.76       150

    accuracy                           0.68       224
   macro avg       0.64      0.63      0.63       224
weighted avg       0.68      0.68      0.68       224
```

Model 4 - Random Forest

In [480]:
```
rfc_model = RandomForestClassifier(n_estimators=10,
                                   max_features='auto',)
```

In [481]:
```python
rfc_model.fit(X_train, y_train)
```
Out[481]:
```
RandomForestClassifier(n_estimators=10)
```

In [482]:
```python
rfc_preds = rfc_model.predict(X_val)
```

In [483]:
```python
print(classification_report(y_val, rfc_preds))
```
```
              precision    recall  f1-score   support

           0       0.52      0.45      0.48        74
           1       0.75      0.80      0.77       150

    accuracy                           0.68       224
   macro avg       0.63      0.62      0.63       224
weighted avg       0.67      0.68      0.68       224
```

Model 5 - XGBoost

In [484]:
```python
xgb1 = XGBClassifier(use_label_encoder=False, eval_metric='logloss')
xgb1.fit(X_train, y_train)
```
Out[484]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, eval_metric='logloss',
              gamma=0, gpu_id=-1, importance_type='gain',
              interaction_constraints='', learning_rate=0.300000012,
              max_delta_step=0, max_depth=6, min_child_weight=1, missing=nan,
              monotone_constraints='()', n_estimators=100, n_jobs=12,
              num_parallel_tree=1, random_state=0, reg_alpha=0, reg_lambda=1,
              scale_pos_weight=1, subsample=1, tree_method='exact',
              use_label_encoder=False, validate_parameters=1, verbosity=None)
```

In [485]:
```python
xgb1_preds = xgb1.predict(X_val)
```

In [486]:
```python
print(classification_report(y_val, xgb1_preds))
```
```
              precision    recall  f1-score   support

           0       0.57      0.45      0.50        74
           1       0.75      0.83      0.79       150

    accuracy                           0.71       224
   macro avg       0.66      0.64      0.65       224
weighted avg       0.69      0.71      0.69       224
```

We will move forward with logistic regression as our model as it performed the best out of all our initial models and is a fast and efficient model for binary classification with numeric features.

Model 6 - Optimized Logistic Regression

In [487]:
```
log2 = LogisticRegression()
# scaler object
scaler = StandardScaler()
# knn operations
log2_operations = [('scaler', scaler), ('log2', log2)]
# import pipeline object
log2_pipe = Pipeline(log2_operations)
```

In [488]:
```
param_grid_log2 = {
    'penalty': ['l2', 'l1', 'elasticnet'],
    'solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'],
    'max_iter': [100, 200, 300]
}
```

In [489]:
```
grid_log2 = GridSearchCV(log2, param_grid_log2, scoring='accuracy',n_jobs=1, cv=3)
```

In [490]:
```
grid_log2.fit(X_train, y_train)
```

Out[490]:
```
GridSearchCV(cv=3, estimator=LogisticRegression(), n_jobs=1,
             param_grid={'max_iter': [100, 200, 300],
                         'penalty': ['l2', 'l1', 'elasticnet'],
                         'solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag',
                                    'saga']},
             scoring='accuracy')
```

In [491]:
```
grid_log2.best_params_
```

Out[491]:
```
{'max_iter': 100, 'penalty': 'l2', 'solver': 'newton-cg'}
```

In [492]:
```
training_preds = grid_log2.predict(X_train)
val_preds = grid_log2.predict(X_val)

training_accuracy = accuracy_score(y_train, training_preds)
val_accuracy = accuracy_score(y_val, val_preds)
```

In [493]:
```
print(training_accuracy)
print(val_accuracy)
```

```
0.7544642857142857
0.7232142857142857
```

In [494]:
```
print(classification_report(y_val, val_preds))
```

```
              precision    recall  f1-score   support

           0       0.61      0.46      0.52        74
           1       0.76      0.85      0.81       150

    accuracy                           0.72       224
   macro avg       0.68      0.66      0.66       224
weighted avg       0.71      0.72      0.71       224
```

Here we create a new LogisticRegression model with the best performing hyperparameters to both confirm it's scores and access it's log-odd coefficients.

In [495]:
```
opt_log = LogisticRegression(max_iter = 100, penalty = 'l2',  solver = 'newton-cg')
```

In [496]:
```
opt_log.fit(X_train, y_train)
training_preds = opt_log.predict(X_train)
val_preds = opt_log.predict(X_val)

training_accuracy = accuracy_score(y_train, training_preds)
val_accuracy = accuracy_score(y_val, val_preds)
```

In [497]:
```
print(training_accuracy)
print(val_accuracy)
```

```
0.7544642857142857
0.7232142857142857
```

In [498]:
```
print(classification_report(y_val, val_preds))
```

```
              precision    recall  f1-score   support

           0       0.61      0.46      0.52        74
           1       0.76      0.85      0.81       150

    accuracy                           0.72       224
   macro avg       0.68      0.66      0.66       224
weighted avg       0.71      0.72      0.71       224
```

In [499]:
```
#Finding Log_odds
log_model_cv_coefs = pd.Series(index=X_train.columns, data=opt_log.coef_[0]).sort_value
log_model_cv_coefs = pd.DataFrame(log_model_cv_coefs)
#Converting to Predicted Odds Ratio
log_model_odds_ratio= math.e ** log_model_cv_coefs
```

In [500]:
```
log_model_odds_ratio
```

Out[500]:

	0
blk	1.936474
stl	1.773498
reb	1.703232
ast	1.428382
pts	1.111598
fg	1.041158
gp	1.030084
3p	1.015564
ft	1.012775
min	0.907622
tov	0.645994

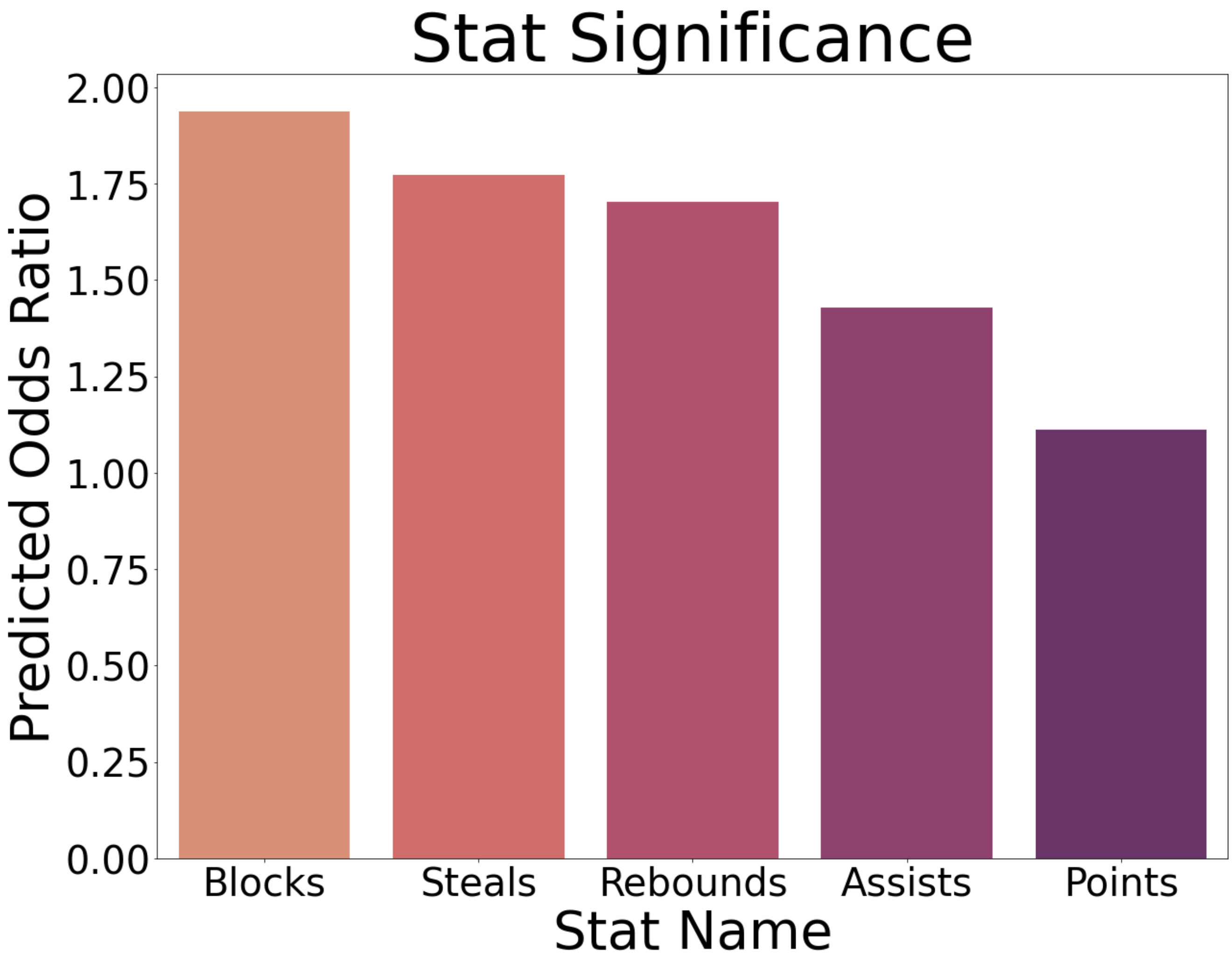
Creating visualization for selected Stats based on Predicted Odds Ratio for final model.

In [501]:
```python
y_ax = np.array(log_model_odds_ratio.values).ravel()
y_ax[:5]
```

Out[501]:
```
array([1.93647371, 1.77349812, 1.70323173, 1.4283817 , 1.11159793])
```

In [502]:
```python
columns = ['Blocks','Steals','Rebounds','Assists','Points','FieldGoal% ',' Games Playec
           'FreeThrow%','Minutes','Turnovers']
```

In [503]:
```python
plt.figure(figsize=(20,15))
sns.barplot(x=columns[:5], y=y_ax[:5], palette='flare')
plt.title('Stat Significance', fontsize=70)
plt.xticks(fontsize=40)
plt.xlabel('Stat Name',fontsize=55)
plt.yticks(fontsize=40)
plt.ylabel('Predicted Odds Ratio',fontsize=55)
# plt.show()
plt.savefig('stats_sig.png')
```



By calulating Predicted Odds Ratio for the various features in the models, we can analyze the impact each feature has on the target(Career lasting longer than 5 years). We will use the highest impact features to reduce the complexity of our model and minimize computational costs while not sacrificing significant accuracy.

Final Model - Optimized Features and Parameters Logistic Regression

In [504]:
```
X = df[['blk', 'stl', 'reb', 'ast', 'pts']]
y = df['target_5yrs']
```

In [505]:
```
# first, split into training and testing data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=4

# splitting the training data into training and validation data\n",
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.25, rar
```

In [506]:
```
best_log = LogisticRegression(max_iter = 100, penalty = 'l2',  solver = 'newton-cg')
```

In [507]:
```
best_log.fit(X_train, y_train)
training_preds = best_log.predict(X_train)
val_preds = best_log.predict(X_val)

training_accuracy = accuracy_score(y_train, training_preds)
val_accuracy = accuracy_score(y_val, val_preds)
```

In [508]:
```
print(training_accuracy)
print(val_accuracy)
```

```
0.7291666666666666
0.7098214285714286
```

In [509]:
```
print(classification_report(y_val, val_preds))
```

```
              precision    recall  f1-score   support

           0       0.59      0.41      0.48        74
           1       0.75      0.86      0.80       150

    accuracy                           0.71       224
   macro avg       0.67      0.63      0.64       224
weighted avg       0.69      0.71      0.69       224
```

In [510]:
```
test_preds = best_log.predict(X_test)

test_accuracy = accuracy_score(y_test, test_preds)
```

In [511]:
```
print(test_accuracy)
```

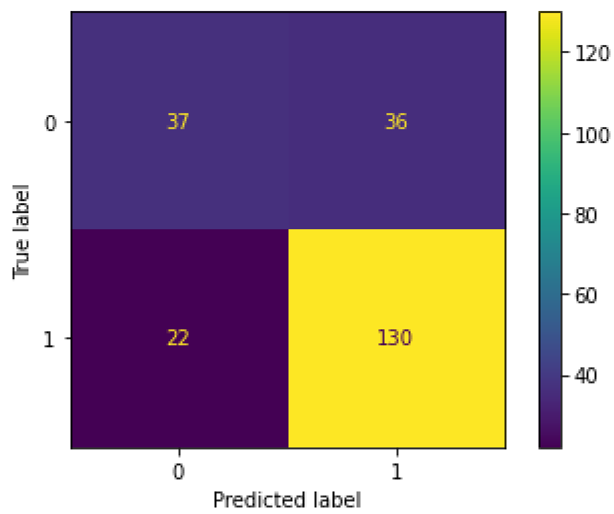
```
0.7422222222222222
```

In [512]:
```
print(classification_report(y_test, test_preds))
```

```
              precision    recall  f1-score   support

           0       0.63      0.51      0.56        73
           1       0.78      0.86      0.82       152

    accuracy                           0.74       225
   macro avg       0.71      0.68      0.69       225
weighted avg       0.73      0.74      0.73       225
```

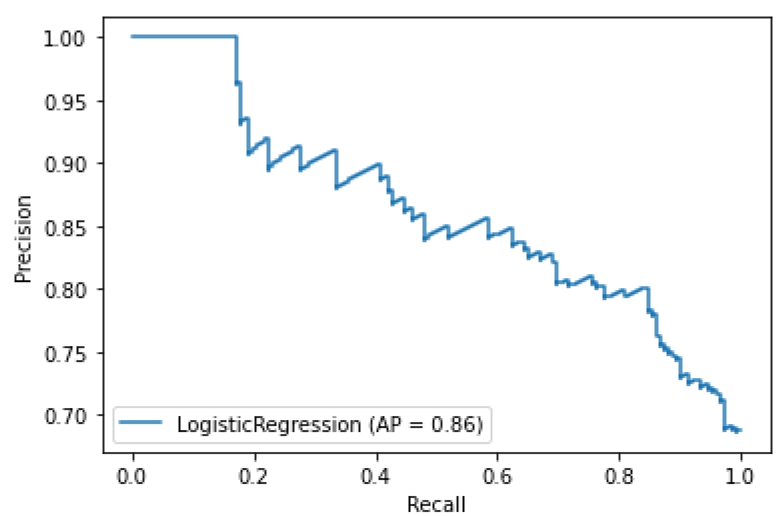
In [513]:
```
plot_confusion_matrix(best_log,X_test,y_test)
```

Out[513]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x28b546c3520>



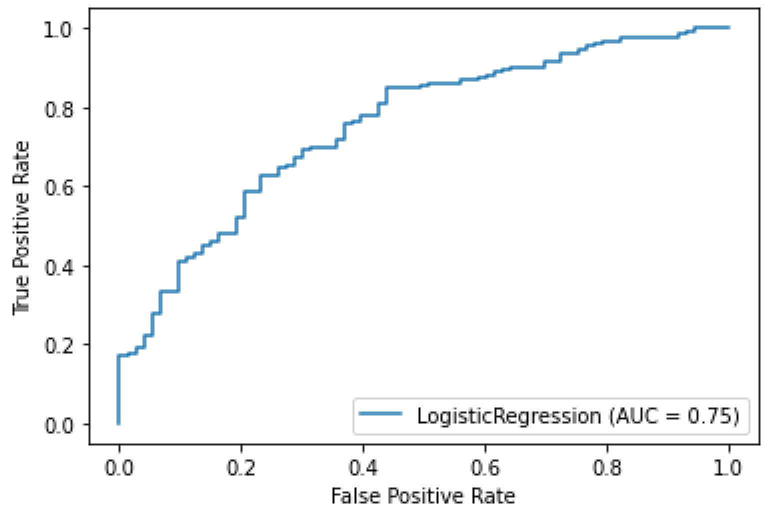
In [514]:
```
plot_precision_recall_curve(best_log, X_test, y_test)
```

Out[514]: <sklearn.metrics._plot.precision_recall_curve.PrecisionRecallDisplay at 0x28b54629790
>



```
In [515]: plot_roc_curve(best_log,X_test,y_test)
```

Out[515]: <sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x28b54799790>



Conclusion

By using the the features with the greatest predicted odds ratio, the best performing hyper-parameters, and removing excessive features we arrive at a final model that balances high accuracy with reasonable computational costs. As a bonus, with an f1-score of .82 it also does a good job minimizing false-positives and false-negatives, despite it not being our main focus. The ROC curve also has good results with an AUC of .75.

As for our recommendations, the key attributes to hone in on for a player who will have a long Career in the NBA:

1. Blocks
2. Steals
3. Rebounds
4. Assists
5. Points Blocks, steals, and rebounds being the top 3 indicates that good defenders are top candidates for long careers, and drives home that "defense wins champions", making those players more valuable.

For future investigation, our current dataset is limited to only in-game statistics and so does not cover all aspects of an aspiring rookie. For some examples:

1. Draft Pick - More draft capital is spent on first and second round players, making teams reluctant to let go of players they have already invested highly in.
2. Financial Impact - Since all NBA teams work under the salary cap, you as a GM have to find a balance between expensive veterans and rookies that are on the cheaper side when building your roster.
3. Injuries - Players who suffer major injuries in the beginning of their professional career may be dealing with long term affects that can affect their ability and career longevity.
4. Evolution of the game - In recent years, the success of Steph Curry and the Golden State Warriors has caused NBA organizations to place a great emphasis on players that can shoot. In particular, NBA teams covet 3-point shooting ability so we would weigh a player's 3 point ability more heavily in the model to predict their career longevity.