Homework Three

Part One: Decision Trees

For this homework, I am taking a look at how team stats can be used as a playoff predictor. To start, I scraped data from the NBA API that featured a teams stats and merged that with data scraped indicating if a team made the plyoffs or not. Note, since the NBA added a Playin tournament in recent years, I had to go back in and manually adjust for teams who made the playoffs through the Playin tournament.

In [ ]:
```
import pandas as pd
from nba_api.stats.endpoints import leaguedashteamstats, leaguestandings
team_stats = leaguedashteamstats.LeagueDashTeamStats(season='2023-24').get_data_fra
standings = leaguestandings.LeagueStandings(season='2023-24').get_data_frames()[0]
playoff_data = standings[['TeamID', 'ClinchedPlayoffBirth']]
merged_data = pd.merge(team_stats, playoff_data, left_on='TEAM_ID', right_on='TeamI
merged_data.loc[
    merged_data['TEAM_NAME'].isin(['Los Angeles Lakers', 'New Orleans Pelicans', 'P
    'ClinchedPlayoffBirth'
] = 1
merged_data.to_csv(data_file, index=False)
```

First, I looked at the basic stats: Points, Rebounds, and Assists (per game):

In [ ]:
```
import pandas as pd
import os
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn.metrics import accuracy_score, classification_report
import numpy as np
data_directory = os.path.join(os.path.dirname(os.path.dirname(os.path.dirname(__fil
data_file = os.path.join(data_directory, 'team_stats_23_24.csv')
data = pd.read_csv(data_file)
data['PPG'] = data['PTS'] / 82
data['RPG'] = data['REB'] / 82
data['APG'] = data['AST'] / 82
features = ['PPG', 'RPG', 'APG']
X = data[features]
y = data['ClinchedPlayoffBirth']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
model = DecisionTreeClassifier(criterion='entropy', max_depth=3, random_state=42)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = np.mean(y_pred == y_test)
```

Which yielded: Accuracy: 33.33% Feature Importances: [0.56671216 0.43328784 0.] ... yikes. But these are very basic stats, it is no wonder they are not a good indicator of playoff stats. Efficiency and defensive stats may be a better place to look.

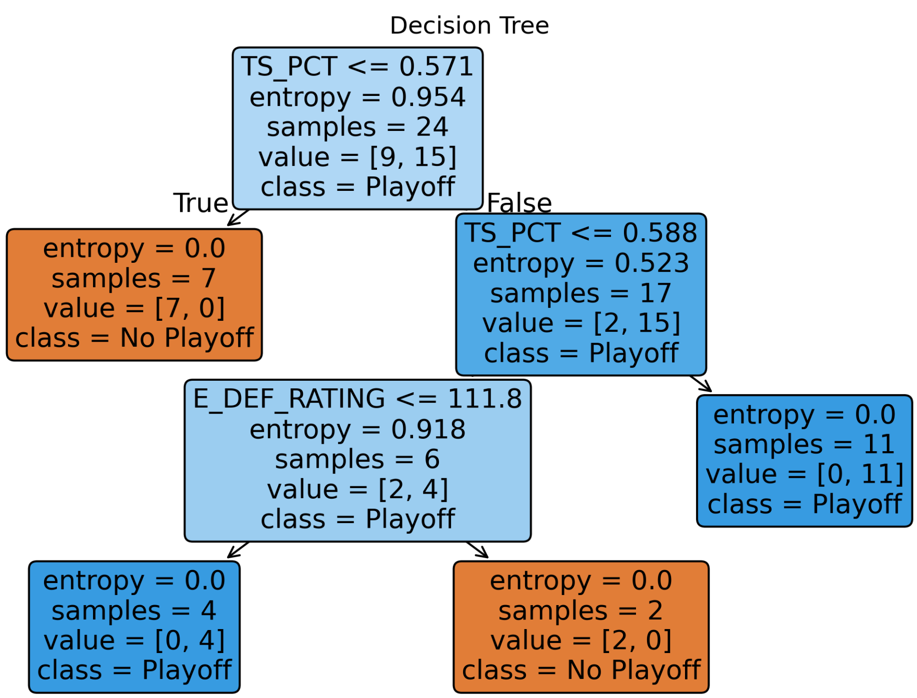
So next, I looked at 3P FG%, Steals per game, and Blocks per game:

In [ ]:
```
data = pd.read_csv(data_file)
data['SPG'] = data['STL'] / 82
data['BPG'] = data['BLK'] / 82
features = ['SPG', 'BPG', 'FG3_PCT']
X = data[features]
y = data['ClinchedPlayoffBirth']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
model = DecisionTreeClassifier(criterion='entropy', max_depth=4, random_state=42)
model.fit(X_train, y_train)
```

Which yielded: Accuracy: 83.33% Feature Importances: [0.2218242 0.11536556 0.66281025]

Much better (for context 53% of NBA teams make the playoffs). Lets visualize this tree:

In [ ]:
```
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 6))
plot_tree(model, feature_names=features, class_names=["No Playoff", "Playoff"], fil
plt.title("Decision Tree")
data_directory = os.path.join(os.path.dirname(os.path.dirname(os.path.dirname(__fil
data_file = os.path.join(data_directory, 'decision_tree_bpg_spg.png')
plt.savefig(data_file, dpi=300, bbox_inches="tight")
```



As you can see from the feature importances, this model weight 3PFG% very heavily, so it may be of interest to replace those with efficiency metrics. I decided to use Effective Field Goal Percentage and Effective Defensive Rating:

In [ ]:
```
data['RPG'] = data['REB'] / 82
features = ['E_DEF_RATING', 'RPG', 'FG3_PCT']
X = data[features]
```

```
y = data['ClinchedPlayoffBirth']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
model = DecisionTreeClassifier(criterion='entropy', max_depth=4, random_state=30)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = np.mean(y_pred == y_test)
```

which yielded: Accuracy: 83.33% Feature Importances: [0.31516414 0.22689069 0.45794517] Interesting, the model performed with the same accuracy. I think dropping 3PFG% may prove beneficial, as the model seems to latch on to 37% shooting from 3 as the de facto root. So next, I tried replacing 3PFG% with Effective Field Goal Percentage and Rebounds per game with True Shooting Percentage, which yielded: Accuracy: 100.00% Feature Importances: [0.24053414 0.75946586 0.] |--- TS_PCT <= 0.57 | |--- class: 0 |--- TS_PCT > 0.57 | |--- TS_PCT <= 0.59 | | |--- E_DEF_RATING <= 111.80 | | | |--- class: 1 | | |--- E_DEF_RATING > 111.80 | | | |--- class: 0 | |--- TS_PCT > 0.59 | | |--- class: 1

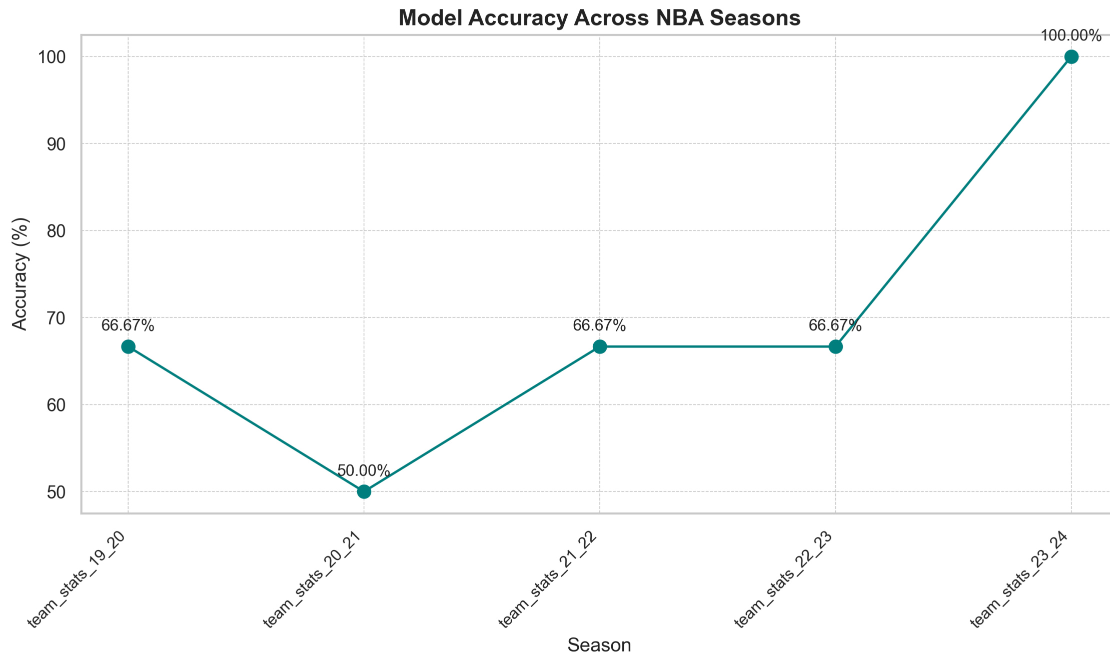
Oh wow and look at that, 100% only using Defensive Rating and True Shooting Percentage, I think I have found the playoff indicators.

Expanding the Dataset

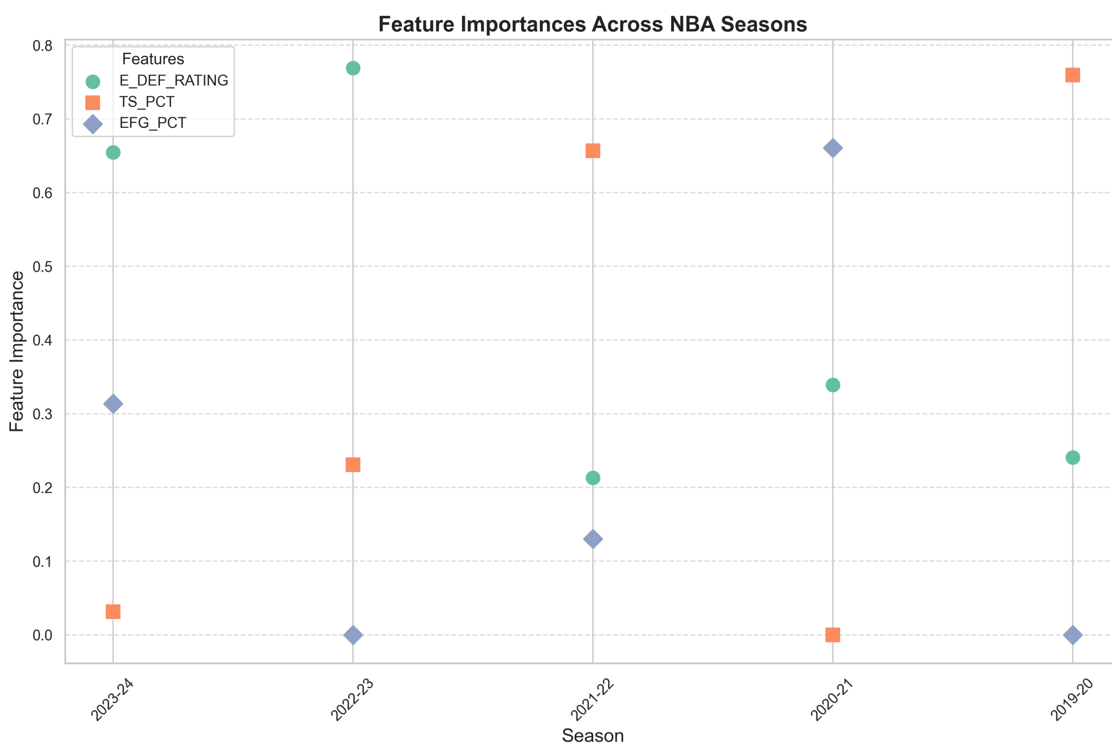
Let's see how this model holds up over the last 5 seasons, I created 5 csv files and ran the model on each of them:

In [ ]:
```
for csv_file in csv_files:
    data_file = os.path.join(data_directory, csv_file)
    data = pd.read_csv(data_file)

    data['PPG'] = data['PTS'] / 82
    features = ['E_DEF_RATING', 'TS_PCT', 'EFG_PCT']
    X = data[features]
    y = data['ClinchedPlayoffBirth']

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random

    model = DecisionTreeClassifier(criterion='entropy', max_depth=4, random_state=3
    model.fit(X_train, y_train)


    y_pred = model.predict(X_test)
    accuracy = np.mean(y_pred == y_test)

    feature_importances = model.feature_importances_

    accuracies.append(accuracy * 100)
    feature_importances_list.append(feature_importances)
    seasons.append(csv_file.split('.')[0])
```

And it did not do well, it's accuracy was not good apart from the recent season:

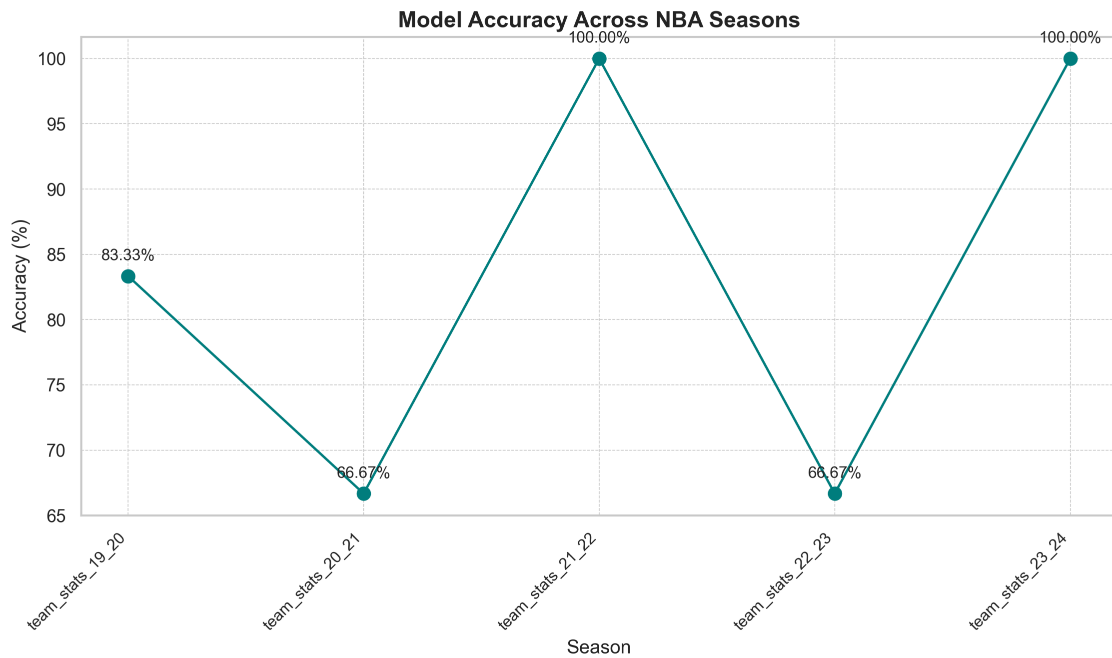


A look out how features were prioritized:



One Last Attempt

I went back and added points per game to model to see if that made it more consistent (it did not)



Part Two KNN Clustering

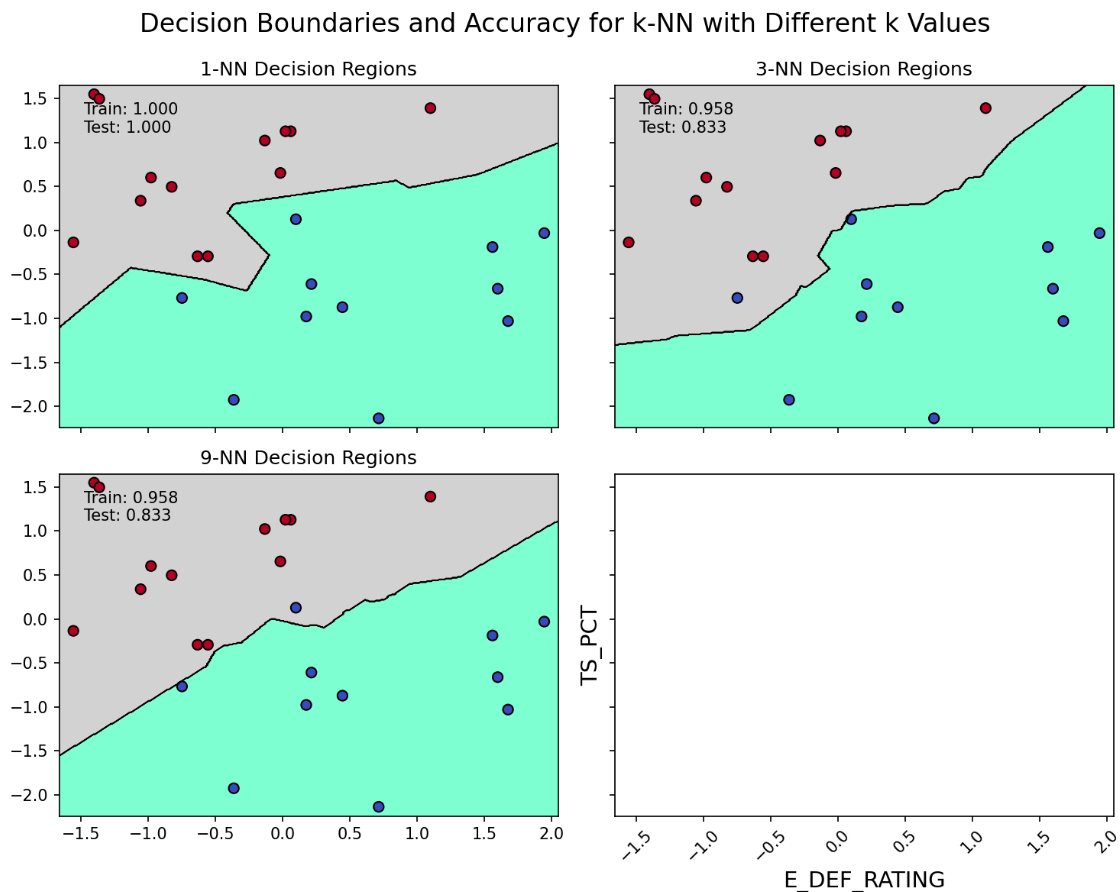
I ran a KNN cluster on the same dataset but just looking at True Shooting Percentage and Effective Defensive Rating, since those were the most used features

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier as KNN
from matplotlib.colors import ListedColormap
from sklearn.preprocessing import StandardScaler
import os

data_directory = os.path.join(os.path.dirname(os.path.dirname(os.path.dirname(__fil
data_file = os.path.join(data_directory, 'team_stats_23_24.csv')
data = pd.read_csv(data_file)

features = ['E_DEF_RATING', 'TS_PCT']
X = data[features]

y = data['ClinchedPlayoffBirth']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

mins = X_train.min(axis=0) - 0.1
maxs = X_train.max(axis=0) + 0.1
x = np.arange(mins[0], maxs[0], 0.01)
y = np.arange(mins[1], maxs[1], 0.01)
```

```python
X_grid, Y_grid = np.meshgrid(x, y)
coordinates = np.array([X_grid.ravel(), Y_grid.ravel()]).T

color = ('aquamarine', 'bisque', 'lightgrey')
cmap = ListedColormap(color)

K_vals = [1, 3, 9]

fig, axs = plt.subplots(2, 2, figsize=(10, 8), dpi=150, sharex=True, sharey=True)
fig.tight_layout()

for ax, K in zip(axs.ravel(), K_vals):
    knn = KNN(n_neighbors=K)
    knn.fit(X_train, y_train)

    Z = knn.predict(coordinates)
    Z = Z.reshape(X_grid.shape)

    # Plot the decision regions
    ax.pcolormesh(X_grid, Y_grid, Z, cmap=cmap, shading='nearest')
    ax.contour(X_grid, Y_grid, Z, colors='black', linewidths=0.5)

    scatter = ax.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap='coolwarm',

    ax.set_title(f'{K}-NN Decision Regions', fontsize=12)
    ax.tick_params(axis='both', labelsize=10)

    train_accuracy = knn.score(X_train, y_train)
    test_accuracy = knn.score(X_test, y_test)

    print('The accuracy for K={} on the train data is {:.3f}'.format(K, test_accura
    print('The accuracy for K={} on the test data is {:.3f}'.format(K, test_accurac
    ax.text(0.05, 0.95, f'Train: {train_accuracy:.3f}\nTest: {test_accuracy:.3f}',
            transform=ax.transAxes, fontsize=10, verticalalignment='top', horizonta


visualization_directory = os.path.join(os.path.dirname(os.path.dirname(os.path.dirn
save_file = os.path.join(visualization_directory, 'knn_practice.png')

plt.suptitle('Decision Boundaries and Accuracy for k-NN with Different k Values', f
plt.xlabel('E_DEF_RATING', fontsize=14)
plt.ylabel('TS_PCT', fontsize=14)
plt.xticks(rotation=45)

plt.tight_layout()
plt.subplots_adjust(top=0.9)
plt.savefig(save_file)
plt.show()
```

Which generated these returns: The accuracy for K=1 on the train data is 1.000 The accuracy for K=1 on the test data is 1.000 The accuracy for K=3 on the train data is 0.833 The accuracy for K=3 on the test data is 0.833 The accuracy for K=9 on the train data is 0.833 The accuracy for K=9 on the test data is 0.833

*Note I was unable to run for k=27 as there was not 27 nearest neighbors for a dataset of this size.



SVM

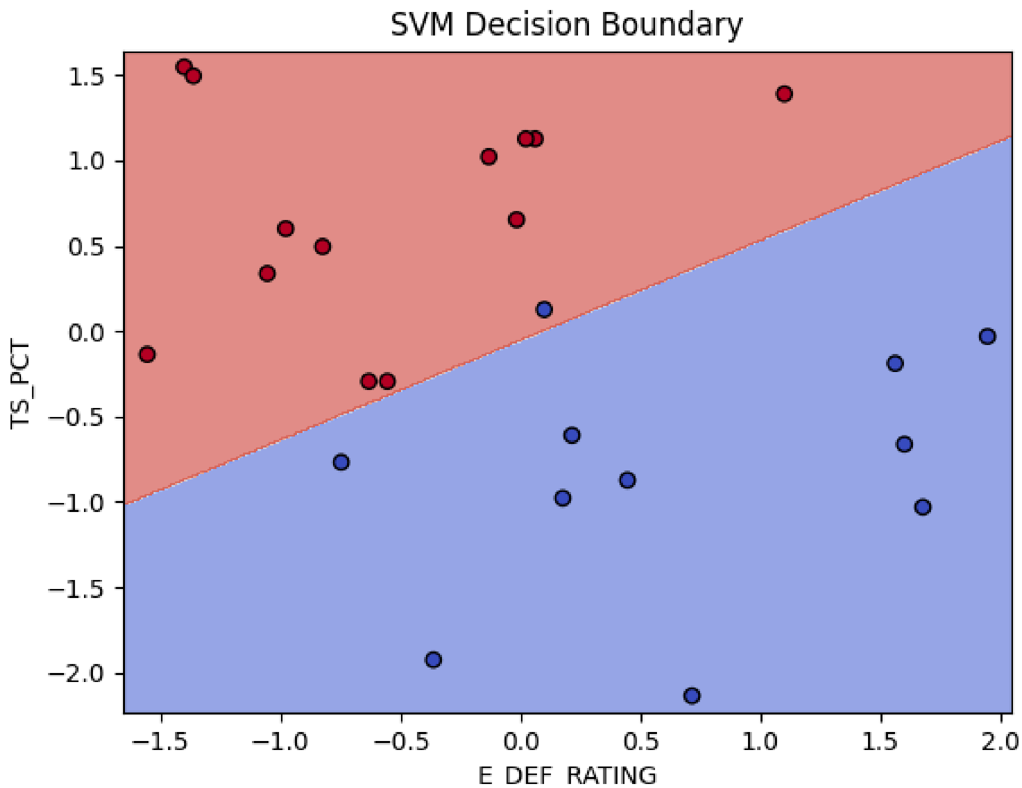
Once again using the same dataset, I created an SVM:

In [ ]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
import os

data_directory = os.path.join(os.path.dirname(os.path.dirname(os.path.dirname(__fil
data_file = os.path.join(data_directory, 'team_stats_23_24.csv')
data = pd.read_csv(data_file)
features = ['E_DEF_RATING', 'TS_PCT']
X = data[features]
y = data['ClinchedPlayoffBirth']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
svm_model = SVC(kernel='linear')
svm_model.fit(X_train, y_train)
train_accuracy = svm_model.score(X_train, y_train)
test_accuracy = svm_model.score(X_test, y_test)

print(f"Training accuracy: {train_accuracy:.3f}")
print(f"Testing accuracy: {test_accuracy:.3f}")
```

Which printed: Training accuracy: 0.958 Testing accuracy: 0.833

And yielded these decision boundaries:



Conclusion

I took a step back from LeBron this week and focused more on team data. I learned that efficiency metrics are generally better for predicting who will be in the NBA playoffs but that there can be a lot of variation from season to season.