# NBA Statistics Final Report

Group 20
Sydney Truong, Austin Percy, Madan Thevar

# Introduction

The NBA has 30 teams with over 450 players total. The amount of data produced per game is staggering, and it's extremely important to understand how that data impacts the decisions of each team in the NBA. We applied machine learning and graphing to NBA statistics to analyze and predict decisions based on results from not only games but entire seasons. Our findings can be further applied to analyze different aspects of not only the NBA but all types of sports.

# Dataset Information

Data was scraped from a source that provides statistics on every player in the league through https://www.basketball-reference.com/leagues/NBA_2024_totals.html. This included information such as (but not limited to) ppg, rpg, mpg, FG, FG%, shot statistics, and rankings. We also extracted data on each team to find their win percentage, playoff status, and who won the championship for each year. The data on the website is presented differently depending on what information we want. For example, the win/loss percentages are presented in multiple datasets that are separated by conference. In contrast, the "per game" stats were separated by the amount of players in each section. This meant that we had to approach each data point separately and make sure we were extracting the correct information.

# Preliminary Tests and Comparison

Before applying a machine learning model, we wanted to make sure we were getting an accurate representation of the data. The first action we took was to print our data frames to ensure there were no missing values and that all of the information matched the website. After that, we made bar graphs to make a more clean and understandable presentation. This enabled us to think clearly about what approach to take when applying machine learning. We've made sure to include all diagrams and graphs used later in this report.

# Problems and Concerns

In our presentation, we briefly discussed our original idea (scraping real estate websites to predict the housing market) and why it didn't work. One of the main things we touched on was that most modern websites have bot/scraping detection. This was a small problem at first because we were testing the code often and creating requests for thousands of data points. We solved this by rotating the hardware we were using in order to split up the requests. Also, during the first stages of testing our code, we implemented a slight delay in the requests in order to lighten the load on the website. Doing these things fixed our main issues and let us work smoothly through the rest of the project. Fixing this was extremely important because we pivoted so late into the project.

# Code and Its Evaluations

```
import pandas as pd

years = [2019, 2020, 2021, 2022, 2023, 2024] #Select the years that will fill the open brackets in the url below
url_link = 'https://www.basketball-reference.com/leagues/NBA_{}_per_game.html' #this is where the data comes from

team_high_scorers_outliers = {} #initialize

for year in years: #iterate through chosen years
    url = url_link.format(year)
    df = pd.read_html(url, header=0)[0]
    df = df.drop(df[df['Age'] == 'Age'].index)

    #convert 'PTS' column to numeric
    df['PTS'] = pd.to_numeric(df['PTS'], errors='coerce')

    #calculate average points per game
    avg_pts_per_game = df['PTS'].mean()
    print(f"Average points per game in {year}: {avg_pts_per_game:.2f}")

    #determine outliers using IQR
    Q1 = df['PTS'].quantile(0.25)
    Q3 = df['PTS'].quantile(0.75)
    IQR = Q3 - Q1
    high_threshold = Q3 + 1.5 * IQR

    #find high outliers
    high_outliers = df[df['PTS'] > high_threshold]

    #group players by team for further organization and cleaning
    for index, row in high_outliers.iterrows():
        player_name = row['Player']
        team = row['Tm']
        if team in team_high_scorers_outliers:
            if player_name in team_high_scorers_outliers[team]:
                team_high_scorers_outliers[team][player_name].add(year)
            else:
                team_high_scorers_outliers[team][player_name] = {year}
        else:
            team_high_scorers_outliers[team] = {player_name: {year}}

print("\nHigh Scorers Outliers Grouped by Team:")
for team, player_outliers in team_high_scorers_outliers.items():
    print(f"\nTeam: {team}")
    for player, outlier_years in player_outliers.items():
        outlier_years_str = ', '.join(map(str, outlier_years))
        print(f"  Player: {player}, Outlier Years: {outlier_years_str}")
```

The above code displays a summary of high-scorer outliers by NBA teams, displaying each team's players with outlier years (years featuring very high points per game). It shows the names of each team's players and the times they were outliers in terms of points per game (NOTE: all of the statistics produced from this code were on a per-game basis). We used this as a way to clean the data and make the dataset smaller.

Here is the sample dataset after loading the data:

| | Rk | Player | Pos | Age | Tm | G | GS | MP | FG | FGA | ... | ORB | DRB | TRB | AST | STL | BLK | TOV | PF | PTS | Year |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 18 | Giannis Antetokounmpo | PF | 24 | MIL | 72 | 72 | 32.8 | 10.0 | 17.3 | ... | 2.2 | 10.3 | 12.5 | 5.9 | 1.3 | 1.5 | 3.7 | 3.2 | 27.7 | 2019 |
| **1** | 40 | Bradley Beal | SG | 25 | WAS | 82 | 82 | 36.9 | 9.3 | 19.6 | ... | 1.1 | 3.9 | 5.0 | 5.5 | 1.5 | 0.7 | 2.7 | 2.8 | 25.6 | 2019 |
| **2** | 61 | Devin Booker | SG | 22 | PHO | 64 | 64 | 35.0 | 9.2 | 19.6 | ... | 0.6 | 3.5 | 4.1 | 6.8 | 0.9 | 0.2 | 4.1 | 3.1 | 26.6 | 2019 |
| **3** | 124 | Stephen Curry | PG | 30 | GSW | 69 | 69 | 33.8 | 9.2 | 19.4 | ... | 0.7 | 4.7 | 5.3 | 5.2 | 1.3 | 0.4 | 2.8 | 2.4 | 27.3 | 2019 |
| **4** | 126 | Anthony Davis | C | 25 | NOP | 56 | 56 | 33.0 | 9.5 | 18.3 | ... | 3.1 | 8.9 | 12.0 | 3.9 | 1.6 | 2.4 | 2.0 | 2.4 | 25.9 | 2019 |
| **5** | 150 | Kevin Durant | SF | 30 | GSW | 78 | 78 | 34.6 | 9.2 | 17.7 | ... | 0.4 | 5.9 | 6.4 | 5.9 | 0.7 | 1.1 | 2.9 | 2.0 | 26.0 | 2019 |
| **6** | 155 | Joel Embiid | C | 24 | PHI | 64 | 64 | 33.7 | 9.1 | 18.7 | ... | 2.5 | 11.1 | 13.6 | 3.7 | 0.7 | 1.9 | 3.5 | 3.3 | 27.5 | 2019 |
| **7** | 184 | Paul George | SF | 28 | OKC | 77 | 77 | 36.9 | 9.2 | 21.0 | ... | 1.4 | 6.8 | 8.2 | 4.1 | 2.2 | 0.4 | 2.7 | 2.8 | 28.0 | 2019 |
| **8** | 203 | Blake Griffin | PF | 29 | DET | 75 | 75 | 35.0 | 8.3 | 17.9 | ... | 1.3 | 6.2 | 7.5 | 5.4 | 0.7 | 0.4 | 3.4 | 2.7 | 24.5 | 2019 |
| **9** | 207 | James Harden | PG | 29 | HOU | 78 | 78 | 36.8 | 10.8 | 24.5 | ... | 0.8 | 5.8 | 6.6 | 7.5 | 2.0 | 0.7 | 5.0 | 3.1 | 36.1 | 2019 |
| **10** | 250 | Kyrie Irving | PG | 26 | BOS | 67 | 67 | 33.0 | 9.0 | 18.5 | ... | 1.1 | 3.9 | 5.0 | 6.9 | 1.5 | 0.5 | 2.6 | 2.5 | 23.8 | 2019 |
| **11** | 259 | LeBron James | SF | 34 | LAL | 55 | 55 | 35.2 | 10.1 | 19.9 | ... | 1.0 | 7.4 | 8.5 | 8.3 | 1.3 | 0.6 | 3.6 | 1.7 | 27.4 | 2019 |
| **12** | 294 | Zach LaVine | SG | 23 | CHI | 63 | 62 | 34.5 | 8.4 | 18.0 | ... | 0.6 | 4.0 | 4.7 | 4.5 | 1.0 | 0.4 | 3.4 | 2.2 | 23.7 | 2019 |
| **13** | 301 | Kawhi Leonard | SF | 27 | TOR | 60 | 60 | 34.0 | 9.3 | 18.8 | ... | 1.3 | 6.0 | 7.3 | 3.3 | 1.8 | 0.4 | 2.0 | 1.5 | 26.6 | 2019 |
| **14** | 305 | Damian Lillard | PG | 28 | POR | 80 | 80 | 35.5 | 8.5 | 19.2 | ... | 0.9 | 3.8 | 4.6 | 6.9 | 1.1 | 0.4 | 2.7 | 1.9 | 25.8 | 2019 |
| **15** | 354 | Donovan Mitchell | SG | 22 | UTA | 77 | 77 | 33.7 | 8.6 | 19.9 | ... | 0.8 | 3.3 | 4.1 | 4.2 | 1.4 | 0.4 | 2.8 | 2.7 | 23.8 | 2019 |
| **16** | 481 | Karl-Anthony Towns | C | 23 | MIN | 77 | 77 | 33.1 | 8.8 | 17.1 | ... | 3.4 | 9.0 | 12.4 | 3.4 | 0.9 | 1.6 | 3.1 | 3.8 | 24.4 | 2019 |
| **17** | 497 | Kemba Walker | PG | 28 | CHO | 82 | 82 | 34.9 | 8.9 | 20.5 | ... | 0.6 | 3.8 | 4.4 | 5.9 | 1.2 | 0.4 | 2.6 | 1.6 | 25.6 | 2019 |
| **18** | 506 | Russell Westbrook | PG | 30 | OKC | 73 | 73 | 36.0 | 8.6 | 20.2 | ... | 1.5 | 9.6 | 11.1 | 10.7 | 1.9 | 0.5 | 4.5 | 3.4 | 22.9 | 2019 |
| **19** | 13 | Giannis Antetokounmpo | PF | 25 | MIL | 63 | 63 | 30.4 | 10.9 | 19.7 | ... | 2.2 | 11.4 | 13.6 | 5.6 | 1.0 | 1.0 | 3.7 | 3.1 | 29.5 | 2020 |
| **20** | 35 | Bradley Beal | SG | 26 | WAS | 57 | 57 | 36.0 | 10.4 | 22.9 | ... | 0.9 | 3.3 | 4.2 | 6.1 | 1.2 | 0.4 | 3.4 | 2.2 | 30.5 | 2020 |
| **21** | 55 | Devin Booker | SG | 23 | PHO | 70 | 70 | 35.9 | 9.0 | 18.3 | ... | 0.4 | 3.8 | 4.2 | 6.5 | 0.7 | 0.3 | 3.8 | 3.0 | 26.6 | 2020 |
| **22** | 123 | Anthony Davis | PF | 26 | LAL | 62 | 62 | 34.4 | 8.9 | 17.7 | ... | 2.3 | 7.0 | 9.3 | 3.2 | 1.5 | 2.3 | 2.5 | 2.5 | 26.1 | 2020 |
| **23** | 134 | Luka Dončić | PG | 20 | DAL | 61 | 61 | 33.6 | 9.5 | 20.6 | ... | 1.3 | 8.1 | 9.4 | 8.8 | 1.0 | 0.2 | 4.3 | 2.5 | 28.8 | 2020 |
| **24** | 199 | James Harden | SG | 30 | HOU | 68 | 68 | 36.5 | 9.9 | 22.3 | ... | 1.0 | 5.5 | 6.6 | 7.5 | 1.8 | 0.9 | 4.5 | 3.3 | 34.3 | 2020 |

25 rows × 31 columns

The Python script uses the stats models as a linear regression analysis package whereby 'PTS' (Points Scored) is estimated based on 'FGA' (Field Goal Attempts) from the NBA dataset. The code takes the parameters from a data frame, inserts a constant for the intercept, and fits an Ordinary Least Squares (OLS) regression model. Significant results from the output show that 'FGA' significantly predicts 'PTS' with a coefficient of 1.0765, suggesting that each field goal attempt increases points scored by around 1.0765. The hypothesis explains 46% of the variance in points obtained, shown by an R-squared value of 0.460. The corresponding F-statistic and the p-value show the model's statistical significance. (from the below-attached image)

```python
import pandas as pd
import numpy as np
import statsmodels.api as sm

X = df['FGA']

#dependent variable (DV): Points Scored
y = df['PTS']

#add constant term to the independent variable
X = sm.add_constant(X)

#fit the linear regression model
model = sm.OLS(y, X).fit()

#print the model summary.
model.summary()
```

OLS Regression Results

| | | | |
|---|---|---|---|
| **Dep. Variable:** | PTS | **R-squared:** | 0.460 |
| **Model:** | OLS | **Adj. R-squared:** | 0.457 |
| **Method:** | Least Squares | **F-statistic:** | 141.6 |
| **Date:** | Tue, 30 Apr 2024 | **Prob (F-statistic):** | 5.27e-24 |
| **Time:** | 17:14:05 | **Log-Likelihood:** | -357.38 |
| **No. Observations:** | 168 | **AIC:** | 718.8 |
| **Df Residuals:** | 166 | **BIC:** | 725.0 |
| **Df Model:** | 1 | | |
| **Covariance Type:** | nonrobust | | |

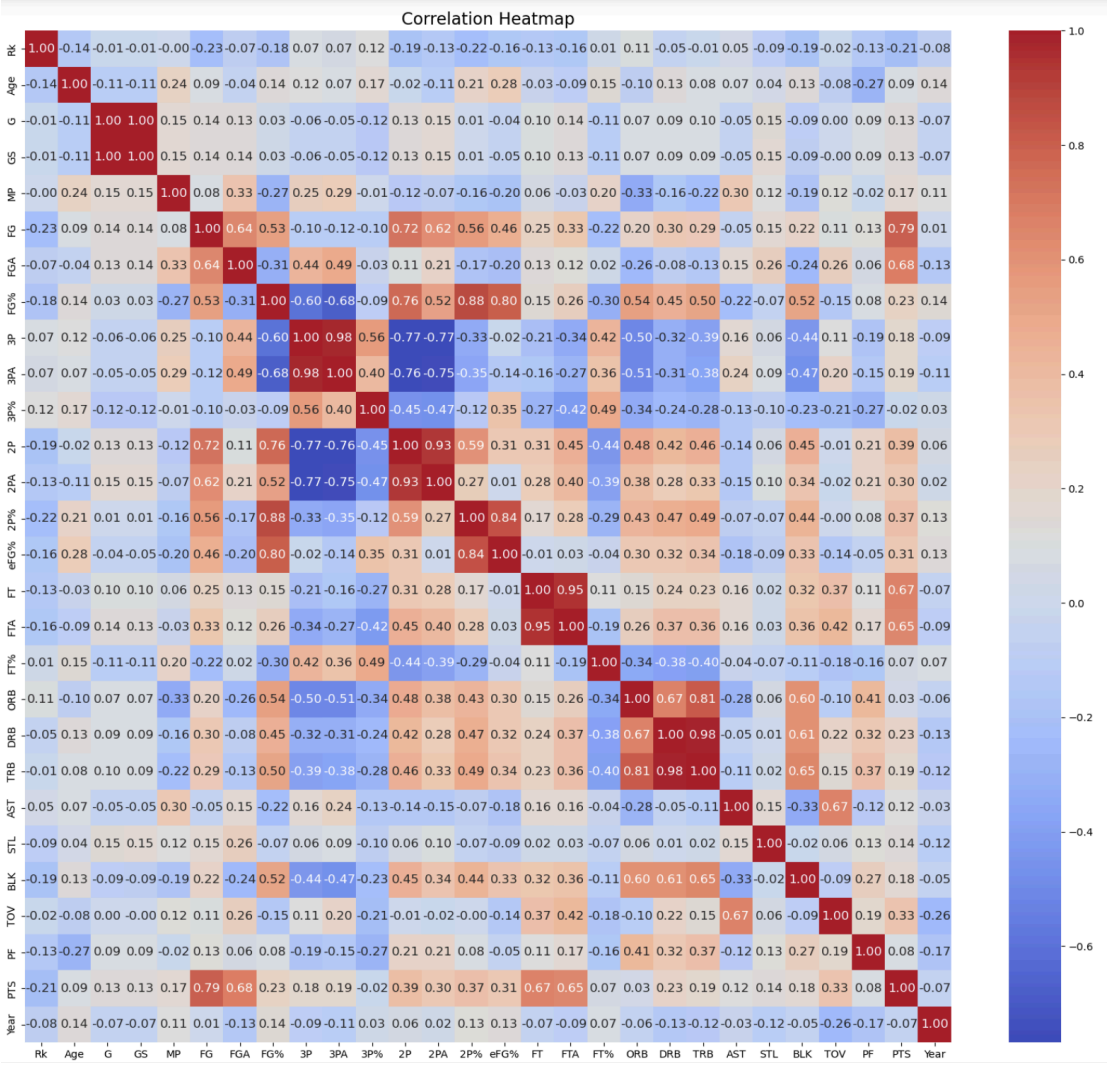
| | coef | std err | t | P>\|t\| | [0.025 | 0.975] |
|---|---|---|---|---|---|---|
| **const** | 5.7610 | 1.723 | 3.344 | 0.001 | 2.359 | 9.163 |
| **FGA** | 1.0765 | 0.090 | 11.900 | 0.000 | 0.898 | 1.255 |

| | | | |
|---|---|---|---|
| **Omnibus:** | 5.269 | **Durbin-Watson:** | 1.858 |
| **Prob(Omnibus):** | 0.072 | **Jarque-Bera (JB):** | 5.320 |
| **Skew:** | 0.433 | **Prob(JB):** | 0.0700 |
| **Kurtosis:** | 2.900 | **Cond. No.** | 209. |

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

# Correlation Heatmap

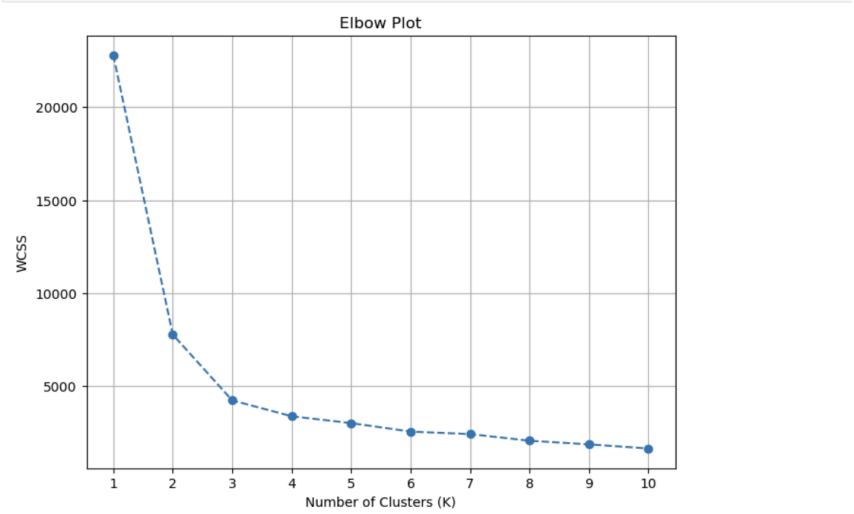
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

corr_matrix = df.corr() #make correlation matrix
warnings.filterwarnings("ignore") #this ignores warnings and we'll use it throughout the rest of the code

#plot heatmap
plt.figure(figsize=(20, 18))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f", annot_kws={"size": 12})
plt.title('Correlation Heatmap', fontsize=16)
plt.show()
```



We made the correlation heatmap using the Seaborn and Matplotlib packages, displaying the correlation coefficients through variables in the NBA dataset. This heatmap, with a color scale that spans blue (negative correlation) to red (positive correlation), offers an instant visual representation of how factors are associated. Strong associations can be seen in the provided heatmap image, such as among 'FGA' (Field Goal tries) and 'PTS' (Points Scored), which is compatible with our prediction that more tries will bring in more points. Another significant connection exists between 'DRB' (Defensive Rebounds) and 'TRB' (Total Rebounds), suggesting substantial overlap, as expected, given that all rebounds include defensive rebounds. The main purpose of this heatmap was to show relationships between statistics pulled from the entire dataset rather than just using points per game or FG attempts per game. This laid the foundation for future predictive models and machine learning.

# Elbow Graph

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

data = data[data['Year'] == 2024] #chose what year to show on the elbow plot

#select features
features = ['Age', 'G', 'GS', 'MP', 'FG', 'FGA', 'FG%', '3P', '3PA', '3P%', '2P', '2PA', '2P%', 'eFG%', 'FT', 'FTA',

X = data[features]

#initialize a list to store inertia values
inertia = []

warnings.filterwarnings("ignore")


#calculate inertia for different values of K
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X)
    inertia.append(kmeans.inertia_)

#plot the elbow plot
plt.figure(figsize=(8, 6))
plt.plot(range(1, 11), inertia, marker='o', linestyle='--')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('WCSS')
plt.title('Elbow Plot')
plt.xticks(np.arange(1, 11, 1))
plt.grid(True)
plt.show()
```




We used the elbow graph to help determine the optimal number of player archetypes or playing styles present in the dataset. Each cluster represents a distinct player archetype, such as scorers, playmakers, and defenders. Understanding these player archetypes can have various real-life applications, such as player evaluation, team composition, scouting, recruitment, player development, opponent analysis, and game planning.

For example, coaches and analysts can use the clustering results to identify common characteristics and performance trends among players within each cluster. They can then tailor strategies, training programs, and game plans to leverage the strengths of each player archetype and maximize team performance.

# Random Forest Classifier (Supervised Machine Learning)

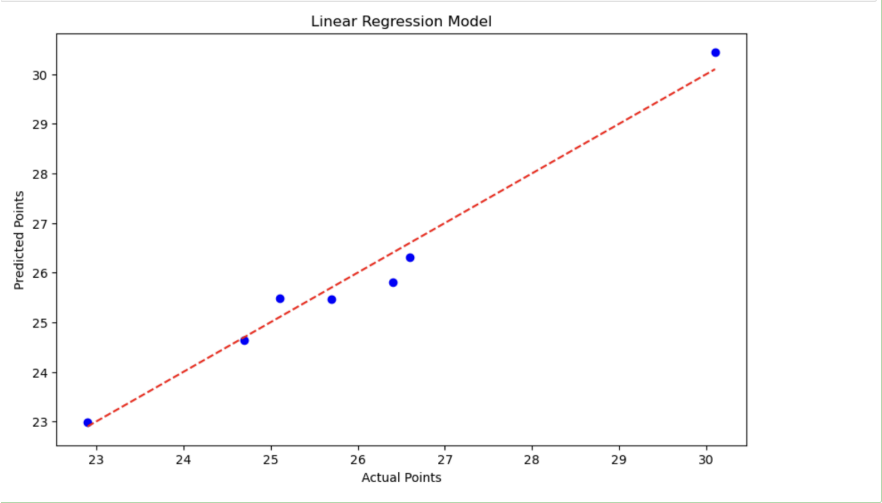
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

data = df[df['Year'] == 2024]

features = ['Age', 'G', 'GS', 'MP', 'FG', 'FGA', 'FG%', '3P', '3PA', '3P%', '2P', '2PA', '2P%', 'eFG%', 'FT', 'FTA',
target = 'Pos'

X = data[features]
y = data[target]

#split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

#initialize and train the Random Forest Classifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
rf_classifier.fit(X_train, y_train)

#predictions
y_pred = rf_classifier.predict(X_test)

#evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.42857142857142855
```

The machine learning framework uses the sklearn package to forecast NBA player positions in 2024, depending on their performance. It decides various features, such as age, games played, shots made, shooting percentages, and additional on-court analytics. The dataset has been split into training (80%) and testing (20%). The model is a RandomForestClassifier, which is capable of handling non-linear data. After training, the model has an accuracy of around 42.86%, suggesting modest predictive potential. This level of accuracy suggests an association between the features and player positions but also emphasizes the challenges and unpredictability of displaying positions merely based on game data.

# Linear Regression Model (Supervised Machine Learning)

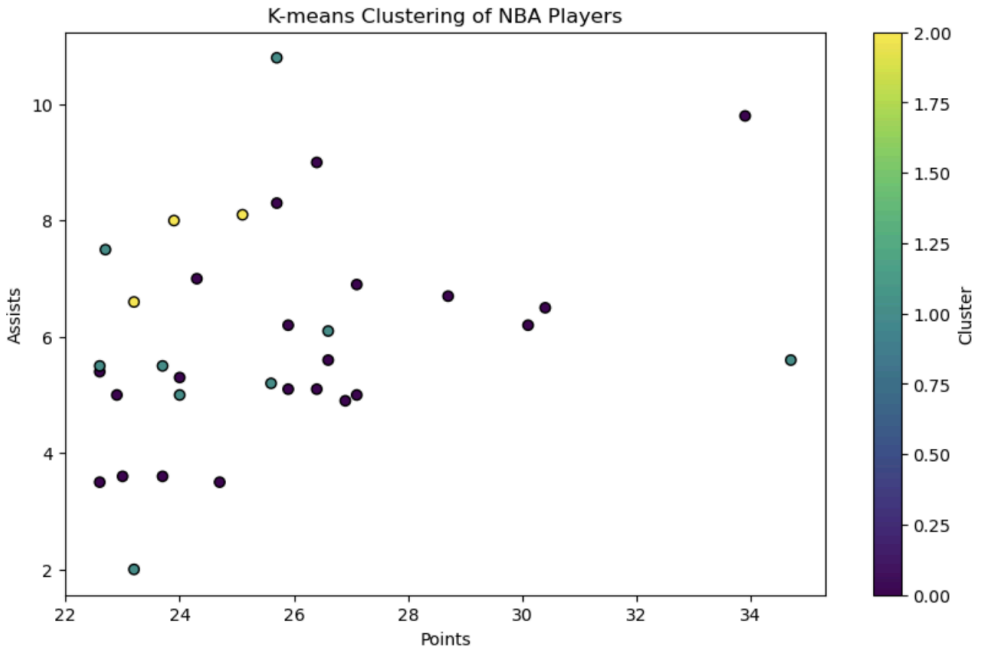
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt

#load the data again from the CSV
df = pd.read_csv('nba_high_scorers_outliers.csv')

data = df[df['Year'] == 2024]

features = ['Age', 'G', 'GS', 'MP', 'FG', 'FGA', '3P', '3PA', '2P', '2PA', 'FT', 'FTA', 'ORB', 'DRB', 'TRB', 'AST',
target = 'PTS'

X = data[features]
y = data[target]

#split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

#initialize and train the Linear Regression model
linear_reg = LinearRegression()
linear_reg.fit(X_train, y_train)

#predictions
y_pred_linear = linear_reg.predict(X_test)

plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred_linear, color='blue')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], linestyle='--', color='red')
plt.xlabel('Actual Points')
plt.ylabel('Predicted Points')
plt.title('Linear Regression Model')
plt.show()
```




Using various player statistics as features, we used a linear regression analysis to anticipate NBA points scored for 2024 (We used 2024 because the season isn't over yet, which created an unfinished dataset for this year). It imports data, selects features, splits into training and testing sets, constructs a Linear Regression model, and evaluates predictions to actual scores. The associated plot shows a good alignment of projected points with actual points, as evidenced by the data points' proximity to the red dashed line, which indicates perfect prediction. This implies that the model has a significant predictive potential for this sample of data, which is especially evident when the predicted and fundamental values increase, implying a viable model for projecting more outlying scores.

# K-Means Clustering Model (Unsupervised Machine Learning)

```
import pandas as pd
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import warnings

df = pd.read_csv('nba_high_scorers_outliers.csv')

data = df[df['Year'] == 2024]

features = ['Age', 'G', 'GS', 'MP', 'FG', 'FGA', '3P', '3PA', '2P', '2PA', 'FT', 'FTA', 'ORB', 'DRB', 'TRB', 'AST', '
X = data[features]

warnings.filterwarnings("ignore")

#initialize and fit clustering
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(X)

#get the cluster labels and add them to the dataframe
data['Cluster'] = kmeans.labels_

#scatter plot
plt.figure(figsize=(10, 6))
plt.scatter(data['PTS'], data['AST'], c=data['Cluster'], cmap='viridis', edgecolor='k')
plt.xlabel('Points')
plt.ylabel('Assists')
plt.title('K-means Clustering of NBA Players')
plt.colorbar(label='Cluster')
plt.show()
```




Using K-means clustering, the code evaluates and divides players based on their 2024 statistics. Clusters use essential features like points, assists, etc. The model is set up with three clusters, and the model is trained to utilize these features. Cluster labels are then added to the dataset and shown in a scatter plot centered on 'Points' and 'Assists.' The resultant plot, colored by cluster, depicts how players are classified based on these characteristics, with the color bar reflecting the cluster's identity. This visualization aids in analyzing player performance patterns by displaying how players with comparable points and assists are clustered together and provides insights into player roles or styles.

# Conclusion

In conclusion, we used multiple models and graphing techniques to analyze data from NBA players. Of these, the OLS and linear regression models were the most accurate and applicable. They both did well in predicting and correlating the data based on points per game and other statistics such as assists per game, FG attempts per game, etc.

Moving forward, this approach holds promise for various facets of the NBA and other sports. By employing predictive models and graphing techniques, franchises can glean invaluable insights into player evaluations and team strategies. Furthermore, harnessing data-driven insights can not only provide teams with a competitive advantage on the court but also enhance the experiences of fans globally. While the use of predictive models in sports is not groundbreaking, fully realizing their potential stands to revolutionize the operational dynamics of teams.