Show code

Show hidden output

Show code

Show hidden output

Import Libraries

Show code

Identifying the top 5% of NBA Players using Unsupervised Machine Learning Algorithms:

- **KMeans Clustering**
- **Gaussian Mixture Models**
- **Agglomerative Clustering**

A Clustering Analysis of NBA Players Statistics from the 2021-22 Season

by John K. Hancock
jkhancock@gmail.com

Show code

Introduction

Every year, NBA journalists vote on the best 15 players in the NBA. The NBA elite. The vote is spread across First, Second, and Third All-NBA teams. Each team must have 2 guards and forwards and a center. These are the historical and prototypical positions on a NBA team even though in the modern game player positions are more fluid.

All-NBA team votes are critical because they can impact how much a player can be paid. Players who make the All-NBA team can get higher maximum value contracts. Thus, the possibility of subjectiveness and biases may come into play. A voter may vote for one player over another if it means that the player would get a better contract. This subjectiveness leads to contentious debates where we argue which player deserves to be voted on the team. What if there was a better method to decide which players are elite. A method purely based on the statistical performance of the players no matter their position.

Unsupervised machine learning, an algorithm that find patterns in unlabeled data, may provide us with sucg a method. Clustering data into similar groups is the most common usage. Using this method, players would be partitioned into clusters based on how similar they are to each other within a cluster and dissimilar to other clusters. A practice that has been a long established. Marketers cluster customers in order to target advertising campaigns. Banks may use them to segment borrowers based on their likelihood of default. Gene researchers may use them to identify reactions of different groups to drug therapies..

Unsupervised Maching Learning

Unsupervised machine learning is the practice of training a computer algorithm to identify cluster among unlabeled data and to discover how different data points are related to one another within a cluster and unrelated to each other outside of a cluster. Intra-cluster similarity is high while inter-cluster similarity is low.

This type of learning is different from supervised machine learning where the algorithm is trained on labeled data in order for it to make predictions on unseen data. Often, data projects involve using unsupervised machine learning to recognize patterns to create clusters which then can be used as labels for supervised learning models.

Clustering Algorithms

There are several different types of Unsupervised Machine learning clustering algorithms. They can be grouped based on their clustering methodology. For this project, I focused on three primary types:

1. Centroid based clustering: KMeans Clustering
2. Distribution based clustering: Gaussian Mixture Models
3. Hierarchical clustering: Agglomerative Hierarchial Clustering

Centroid based clustering groups data into clusters based on their closeness to central, representative data point called a centroid. The centroid is calculated as the average or mean of the data points within a cluster. One of the more prominent algorithms.

Distribution based clustering groups data into clusters based on the underlying distribution of the data. Data points within the same cluster are expected to have similar probability distributions whereas data points from different clusters are expected to have different distributions. Gaussian Mixture Models ("GMM") models the data points as a mixture of Gaussian distributions, where each Gaussian component represents a cluster. The algorithm estimates the parameters of the Gaussian components, such as the mean, covariance, and mixture weights, to identify the clusters in the data.

Hierarchical clustering groups data into clusters by building a hierarchy of clusters using a graphical diagram called a dendrogram. It recursively splits or merges clusters until a stopping criterion is met. Agglomerative hierarchical clustering uses a bottom-up clustering methodology. Each data point starts out as a separate cluster and iteratively merges clusters based on a similarity or dissimilarity measure until a stopping criterion is reached. The similarity or dissimilarity between clusters can be calculated using various distance metrics such as Euclidean distance, Manhattan distance, or cosine similarity.

To find the elite 5% of NBA players, I will use all three of these approaches.

Project Objective

The objective of this project is to see if the three unsupervised machine learning algorithms, KMeans Clustering, Gaussian Mixture Models, and Agglomerative Hierarchial clustering will be able to identify a cluster of NBA players whose statistics will indicate that they are the elite 5% of the NBA. Players in this elite cluster will be compared to players voted on the NBA All Pro Teams.

Project Plan

I. Collect the Data

> A. Data Source
> B. Discussion of Data Source

II. Data Import and Cleaning

> A. Read in data from website
> B. Inspect and describe imported data
> C. Remove RANK Feature and Correct Column Names
> D. Update Duplicate Player Observations

III. Data Exploration

> A. Position Feature ("POS")
> B. Games Played ("GP")
> C. Missing Data
> D. Distribution of Data
> E. Multicollinearity

IV. Update and Scale the Features

> A.Handling Object Features
> B. Scale the Features using sklearn Standard Scaler

V. Dimensionality Reduction Using PCA and Manifold Learning

> A. Principal Component Analysis ("PCA")
> B. Manifold Learning

VI. Application of Unsupervised Learning Algorithms

> A. KMeans Clustering
> B. Gaussian Mixture Models ("GMM")
> C. Agglomerative Hierarchy Clustering("AGC")
> D. Elite Players

VII. Finding the Elite of the Elite

> A. KMeans Clustering
> B. Gaussian Mixture Models ("GMM")
> C. Agglomerative Hierarchy Clustering("AGC")
> D. Identifying the top 5% (top tier elite)

VIII. Summary of Findings

I. Collect the Data

For this project, I will use NBA player statistics from the 2021–22 regular season. The data was collected from the website www.nbastuffer.com. NBA Stuffer provides robust NBA player statistics that help more than 1 million people who want to analyze NBA. I excluded data from the NBA 2020–21 playoffs as that might bias the clusters to those players on those teams that made the playoffs. Also, players are voted to the All NBA teams prior to the playoffs.

II. Data Import and Cleaning

A. Read in data from website

The website no longer allows users to read in data directly, but it does allow for the exportation of data into an excel spreadsheet titled, NBA Stats 202122 All Player Statistics.xlsx

B. Inspect and describe imported data

1. Reading the data from NBAStuffer into a pandas dataframe, import_df, shows that there are 29 features and 716 observations where each observation represents a player in the NBA.
2. All of the features are numeric except for three, the Full Name of the player, his position, and his Team.
3. The column names include the name of the feature along with a description which will have to be cleaned.

▶ Import excel file from NBA Stuffer

Show code

▶ **Info Printout**

Show code

Show hidden output

▶ **Sample of a Column Name which includes Description**

Show code

```
USG%Usage RateUsage rate, a.k.a., usage percentage is an estimate of the percentage of
```

▼ C. Remove RANK Feature, Correct Column Names, and Review Dataset Features

> 1. Remove the RANK column and save it into a new dataframe, df
> 2. Rename the columns to shorter names
> 3. Review the updated column names and Dataset Features

1. Remove the RANK column and save it into a new dataframe, df

Show code

2. Rename the columns to shorter names

Show code

3. Review the updated column names and Dataset Features

Show code

Show hidden output

Dataset Features

FULL NAME = Full name of the player
TEAM = Name of the player's team
POS = Position of the player
AGE = Age of the player
GP = No. of games the player played
MPG = No. of minutes a player played per game
MIN% = Minutes Percentage is the percentage of team minutes used by a player while he was on the floor
USG% = Usage rate percentage is an estimate of the percentage of team plays used by a player while he was on the floor
TO% = Turnover Rate estimates the number of turnovers a player commits per 100 possessions
FTA = Free throw attempts
FT% = Percentage of free throws made to free throws attempted
2PA = Two point shots attempted
2P% = Percentage of two points made to two point shots attempted
3PA = Three point shots attempted
3P% = Percentage of three points made to two point shots attempted
eFG% = Effective Shooting Percentage. Three-point shots made are worth 50% more than two-point shots made. eFG% Formula=(FGM+ (0.5 x 3PM))/FGA
TS% = True Shooting Percentage is a measure of shooting efficiency that takes into account field goals 3-point field goals and free throws.
PPG = No. of points per game
RPG = No. of rebounds per game
TRB% = Total Rebound Percentage is estimated percentage of available rebounds grabbed by the player while the player is on the court.
APG = No. of assists per game.
AST% = Assist Percentage is an estimated percentage of teammate field goals a player assisted while the player is on the court
SPG = No. of steals per game
BPG = No. of Blocks per game
TOPG = No. of Turnovers per game
VI = Versatility index is a metric that measures a player’s ability to produce in points, assists, and rebounds. The average player will score around a five on the index while top players score above 10
ORTG = Offensive Rating Individual is the number of points produced by a player per 100 total

individual possessions

DRTG = Defensive Rating Individual estimates how many points the player allowed per 100 possessions he individually faced while staying on the court

D. Update Duplicate Player Observations

Data is duplicated for players that were traded during the season. In the Sample of Players Traded, there are two entries for Seth Curry, James Harden, and Kristaps Porzingas. These players were all traded during the seaon. Alize Johnson was traded twice so he appears three times in the dataset. Their statistics are split across each team they played for.

Out of the original imported dataset containing 716 observations, 209 were duplicate player entries. To correct this, I grouped and aggregated accordingly so that each player is represented once in the dataset.

Sample of Players Traded

Show code

	FULL NAME	TEAM
140	Seth Curry	Phi
141	Seth Curry	Bro
251	James Harden	Bro
252	James Harden	Phi
328	Alize Johnson	Chi
329	Alize Johnson	Was
330	Alize Johnson	Nor
533	Kristaps Porzingis	Dal
534	Kristaps Porzingis	Was

Create an index of players with multiple records

Show code

```
FULL NAME
Aaron Holiday     True
Alize Johnson     True
Andre Drummond    True
```

```
Armoni Brooks     True
Name: FULL NAME, dtype: bool
```

▶ Create a dataframe, player_dupes, for the duplicate records. Drop the TEAM feature.

Show code

	FULL NAME	POS	AGE	GP	MPG	MIN%	USG%	TO%
284	Aaron Holiday	G	25.53	41	16.2	33.7	18.0	14.5
285	Aaron Holiday	G	25.53	22	16.3	33.9	19.9	16.3
330	Alize Johnson	F	25.97	4	7.1	14.8	19.2	7.8
329	Alize Johnson	F	25.97	3	6.1	12.6	23.1	31.8
328	Alize Johnson	F	25.97	16	7.5	15.7	12.5	23.5
172	Andre Drummond	C	28.67	24	22.3	46.4	21.5	13.5
171	Andre Drummond	C	28.67	49	18.4	38.4	17.6	22.4

Group the duplicate records by Full Name and aggregate their statistical measures accordingly. For example, to get the traded players full number of games played, you have to sum their games played across multiple teams.

▼ Update the player_dupes dataframe by aggregate the columns

```
#@title ####Update the player_dupes dataframe by aggregate the columns
cols = player_dupes.columns.to_list()
player_dupes = player_dupes.groupby(['FULL NAME']).agg({cols[0] : 'min',
                                                        cols[1] : 'min',
                                                        cols[2] : 'mean',
                                                        cols[3] : 'sum',
                                                        cols[4] : 'mean',
                                                        cols[5] : 'mean',
                                                        cols[6] : 'mean',
                                                        cols[7] : 'mean',
                                                        cols[8] : 'mean',
                                                        cols[9] : 'mean',
                                                        cols[10] : 'mean',
                                                        cols[11] : 'mean',
                                                        cols[12] : 'mean',
                                                        cols[13] : 'mean',
                                                        cols[14] : 'mean',
                                                        cols[15] : 'mean',
                                                        cols[16] : 'mean',
                                                        cols[17] : 'mean',
```

```
                                                    cols[18] : 'mean',
                                                    cols[19] : 'mean',
                                                    cols[20] : 'mean',
                                                    cols[21] : 'mean',
                                                    cols[22] : 'mean',
                                                    cols[23] : 'mean',
                                                    cols[24] : 'mean',
                                                    cols[25] : 'mean',
                                                    cols[26] : 'mean'
  })

player_dupes.reset_index(drop=True,inplace=True)
```

Now duplicate player entries now only appear once.

Show code

	FULL NAME
1	Alize Johnson
39	James Harden
57	Kristaps Porzingis
82	Seth Curry

- Remove the 209 duplicates from the main dataframe, df

Show code

```
(507, 28)
```

There are now 98 records in the player_dupes df and 507 in the main df.

```
player_dupes.shape
```

```
(98, 27)
```

```
df.shape
```

```
(507, 28)
```

- Drop the 'TEAM' from the main dataframe, df, and concat the player_dupes dataframe

Show code

	FULL NAME	POS	AGE	GP	MPG	MIN%	USG%	TO%	FTA	FT%	...	RPG	TRB%	APG	AST
0	Precious Achiuwa	F	22.56	73	23.6	49.2	18.5	11.3	131.0	0.595	...	6.5	14.9	1.1	6.
1	Steven Adams	C	28.73	76	26.3	54.8	12.0	19.6	199.0	0.543	...	10.0	19.9	3.4	16.
2	Bam Adebayo	C-F	24.73	56	32.6	67.9	25.0	14.4	340.0	0.753	...	10.1	17.5	3.4	17.

There are now 605 observations over 27 features.

Show code

Show hidden output

III. Data Exploration

A. Position Feature ("POS")
B. Games Played ("GP")
C. Minutes per Game ("MPG")
D. Check for Missing Data
E. Distribution of Key Data
F. Multicollinearity

A. Position Feature ("POS")

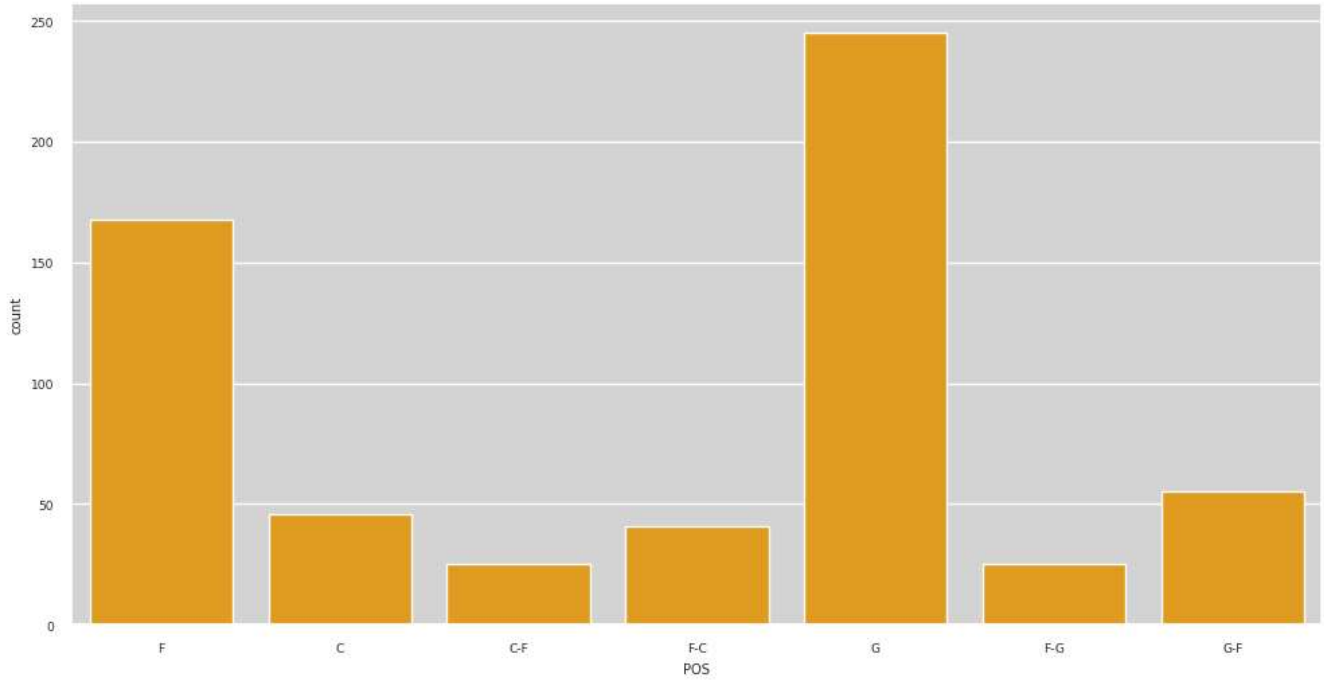
Traditionally, a basketball team consists of 5 players, two Guards ("G"), two Forwards ("F"), and a Center ("C"). In today's NBA, there are hybrid positions such as a "Guard-Forward" like ("GF") or "Forward-Center" ("FC"). A countplot of positions ("POS") shows that there are more pure guards and forwards in the NBA than at any other position.

Show code

```
POS
G      245
F      168
G-F     55
C       46
```

```
F-C     41
C-F     25
F-G     25
Name: POS, dtype: int64
```

Show code



B. Games Played ("GP")

The number of games played by players is not normally distributed. Out of the 605 players, 104 played in 10 games or less. This is expected as teams use temporary contract players as fill ins mostly for injured players.

Additionally, 146 players are in the bottom 25% quartile of games played. In order to not skew the results, these players will be removed from the dataset.

Games Played Statistics

Show code

	count	mean	std	min	25%	50%	75%	max
GP	605.0	43.042975	25.807966	1.0	17.0	48.0	66.0	82.0

No. of Players who played less than 10 games

Show code

	count	mean	std	min	25%	50%	75%	max
GP	104.0	3.836538	2.481428	1.0	2.0	3.0	5.0	9.0

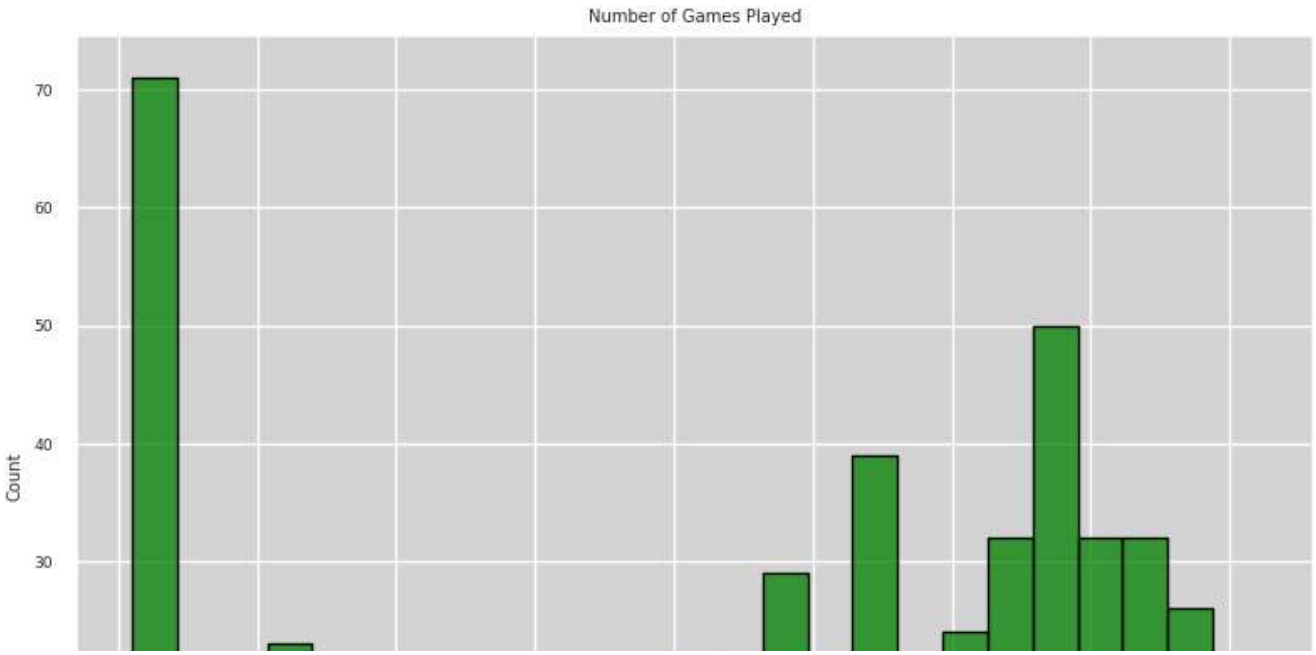
Double-click (or enter) to edit

No. of Games Played by Players in the bottom 25% quartile

Show code

	count	mean	std	min	25%	50%	75%	max
GP	146.0	6.363014	4.598171	1.0	2.0	5.0	10.0	16.0

```
#@title
plt.figure(figsize=(8,6))
sns.histplot(x='GP', data=df, bins=25, color='green', edgecolor='black')
plt.title("Number of Games Played")
plt.tight_layout()
plt.show()
```



Sample of Removed Players

Show code

	FULL NAME	POS	GP
14	Ryan Arcidiacono	G	10
17	Joel Ayayi	G	7
25	Cat Barber	G	3
31	Paris Bass	F	2
38	Jordan Bell	F	1

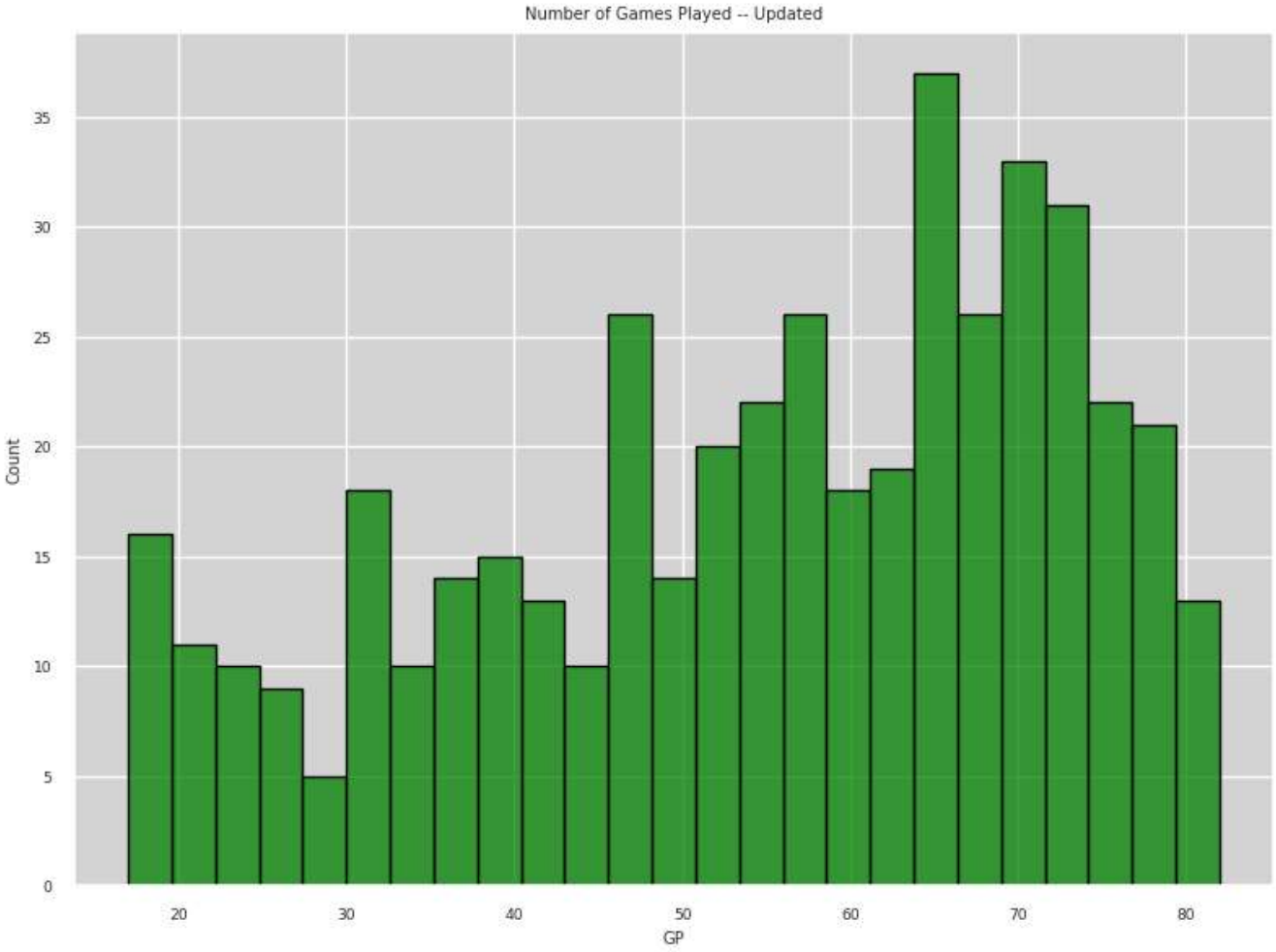
The updated number of players in the dataset is now 459.

Shape of Updated Dataset

Show code

```
(459, 27)
```

Show code



C. Minutes per Game ("MPG")

The average number of minutes played per game was 21.67 minutes. The bottom 25% played less than 15 minutes per game. For example, Kai Jones played only 3 minutes per the 21 games that he played. Players with low number of minutes makes their offensive and defensive ratings NaN.

Show code

	count	mean	std	min	25%	50%	75%	max
MPG	459.0	21.675454	8.438692	3.0	14.95	21.5	28.85	37.9

Kai Jones Minutes Played

Show code

Players in the bottom quartile of the number of minutes played are removed from the dataset.

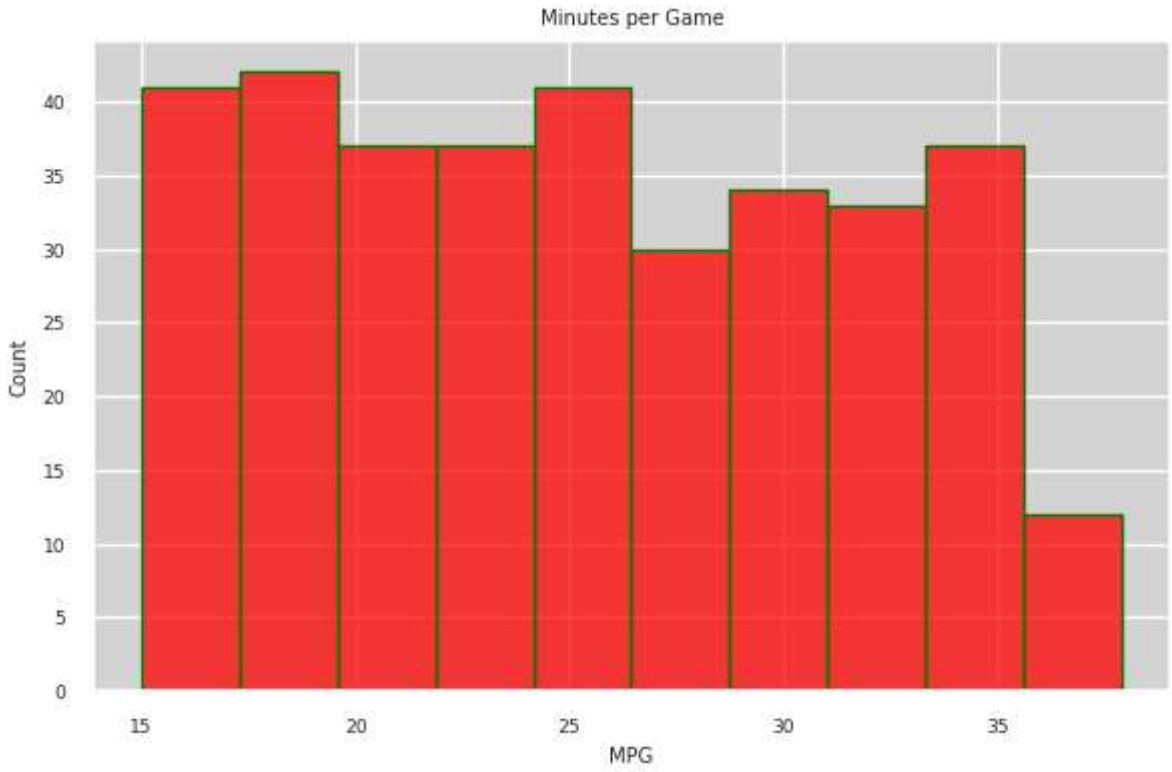
Show code

```
(261, 27)
```

Show code

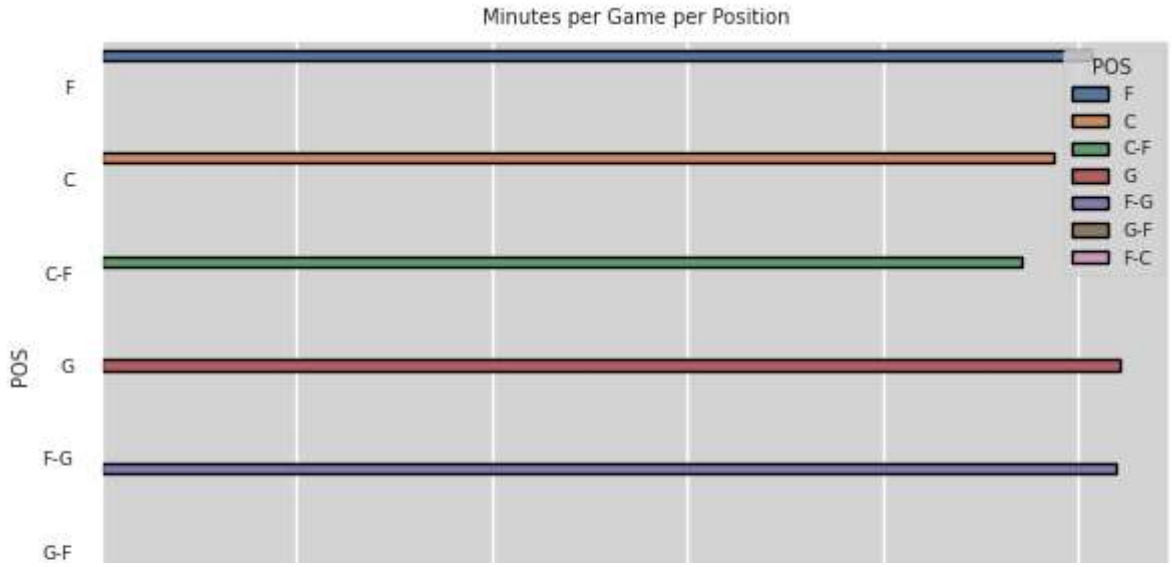
```
(344, 27)
```

Show code



Guards, and hybrid guard positions, account for the most minutes played per game.

Show code



D. Check for Missing Data

Given that players with a low amount of games played and minutes per game have been removed, there are no missing data. The updated total number of players in the dataset is now 344.

```
df.isna().sum()
```

```
FULL NAME    0
POS          0
AGE          0
GP           0
MPG          0
MIN%         0
USG%         0
TO%          0
FTA          0
FT%          0
2PA          0
2P%          0
3PA          0
3P%          0
eFG%         0
TS%          0
PPG          0
RPG          0
TRB%         0
APG          0
AST%         0
SPG          0
BPG          0
TOPG         0
VI           0
ORTG         0
DRTG         0
dtype: int64
```

Show code

Show code

```
(344, 27)
```

E. Distribution of Key Features

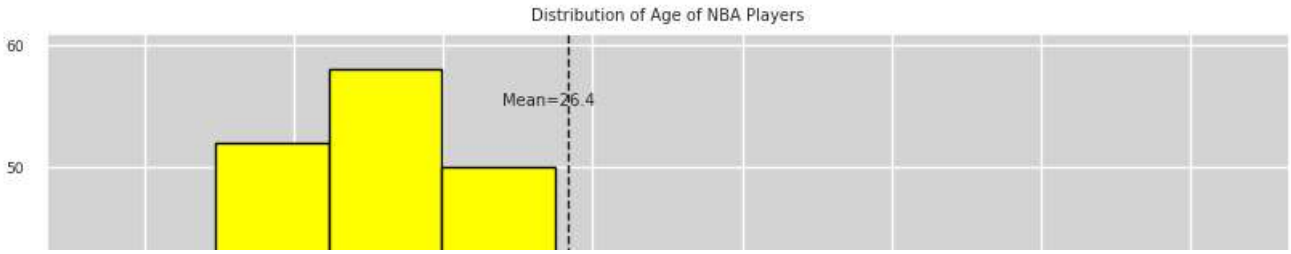
Age

The average age of the players is 26.4 years old with 25% of the players aged 23.6 and under. The distribution of age is right skewed meaning that there are far more youger players than older players.

Show code

	count	mean	std	min	25%	50%	75%	max
AGE	344.0	27.077442	4.479344	19.3	23.5825	26.415	30.2175	38.2

Show code



Games Played

Of the players that played in more than 25% of the games, their average number of games played was 60 and the median number of games played was 64.5.

Show code

	count	mean	std	min	25%	50%	75%	max
GP	344.0	59.944767	15.402846	17.0	51.0	64.5	71.25	82.0

Show code

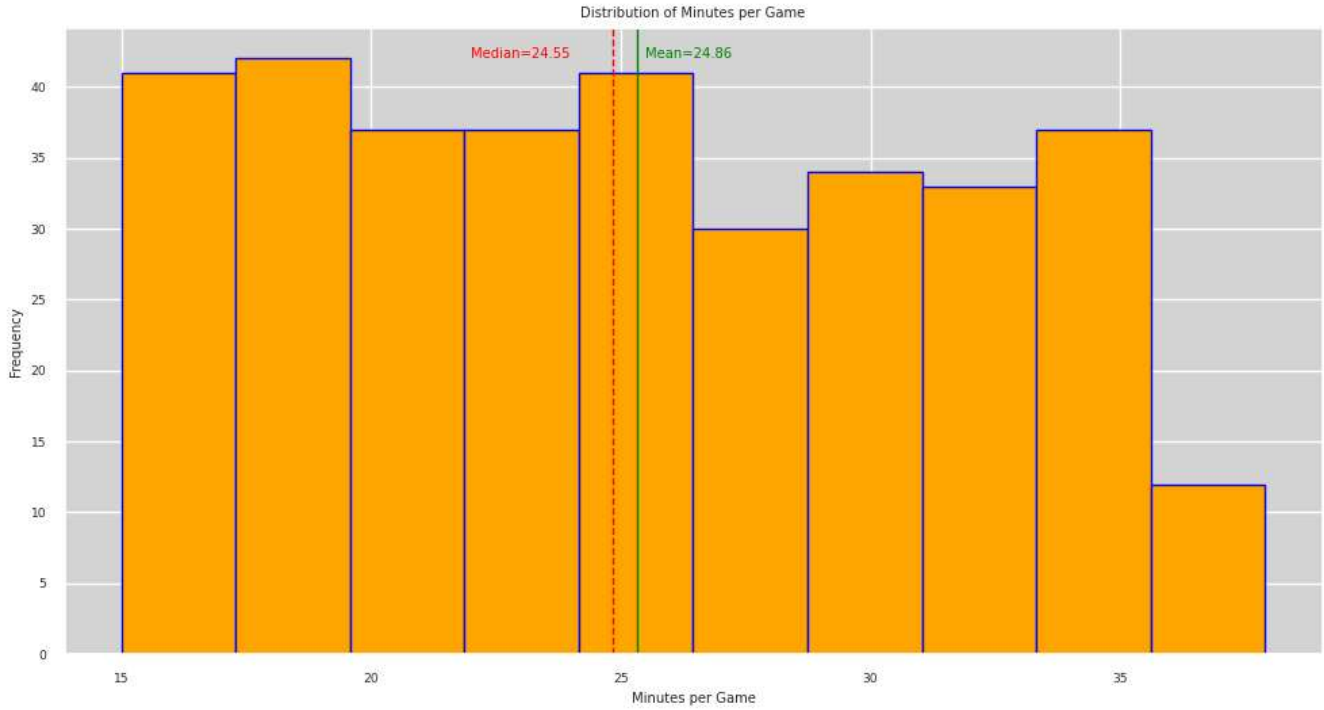
Minutes per Game

For players that averaged above the bottom quartile in Minutes per Game("MPG"), their MPG is normally distributed with the mean(25.3) being nearly identical to the median (24.85).

Show code

	count	mean	std	min	25%	50%	75%	max
MPG	344.0	25.313469	6.297458	15.0	19.975	24.85	30.725	37.9

Show code

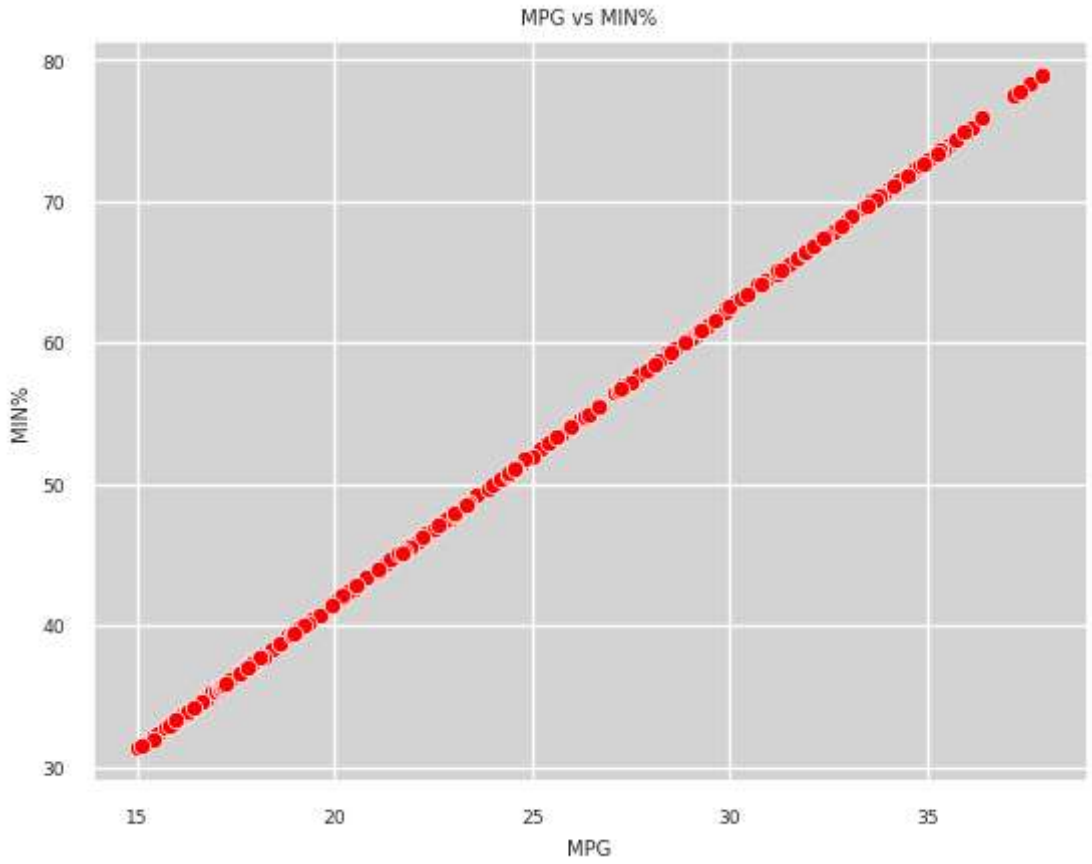


F. Multicollinearity

Often in sports statistics, there will be some correlation between indidivual features. For example, 'MPG' (the number of minutes played per game) is highly correlated with 'MIN%' (the percentage of team minutes used by a player while he was on the floor). Additionally, MPG has high correlations with Points per Game ("PPG"), Turnovers per Game ("TOPG"), and two point attempts ("2A") as well as other features. The pairplot shows the strong correlation between these features.

The Heatmap further confirms that there are strong correlations among other features suggesting that there is **Multicollinearity**. Multicollinearity occurs when there are two or more correlated independent or predictor variables. If the dataset has highly correlated variables due to multicollinearity, it can affect the similarity or distance metrics used for clustering, leading to biased or distorted clustering results.

Show code



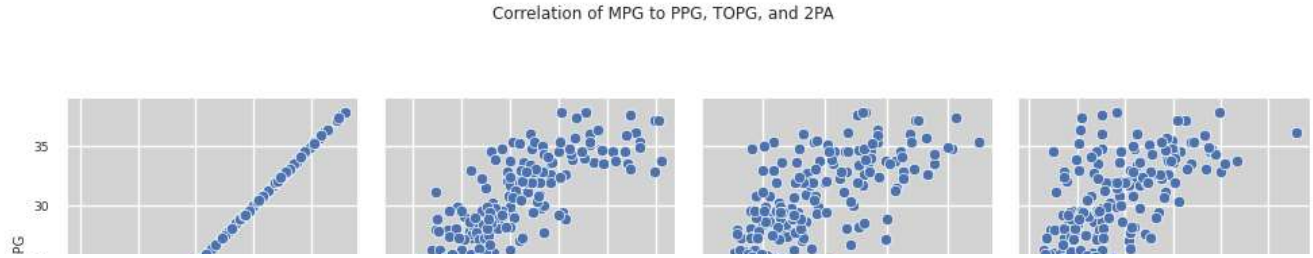
Show code

```
<ipython-input-281-e8ddef53137f>:2: FutureWarning: The default value of numeric_only in
  df.corr()['MPG'].sort_values(ascending=False).head()
MPG     1.000000
MIN%    0.999988
PPG     0.843148
TOPG    0.711129
```

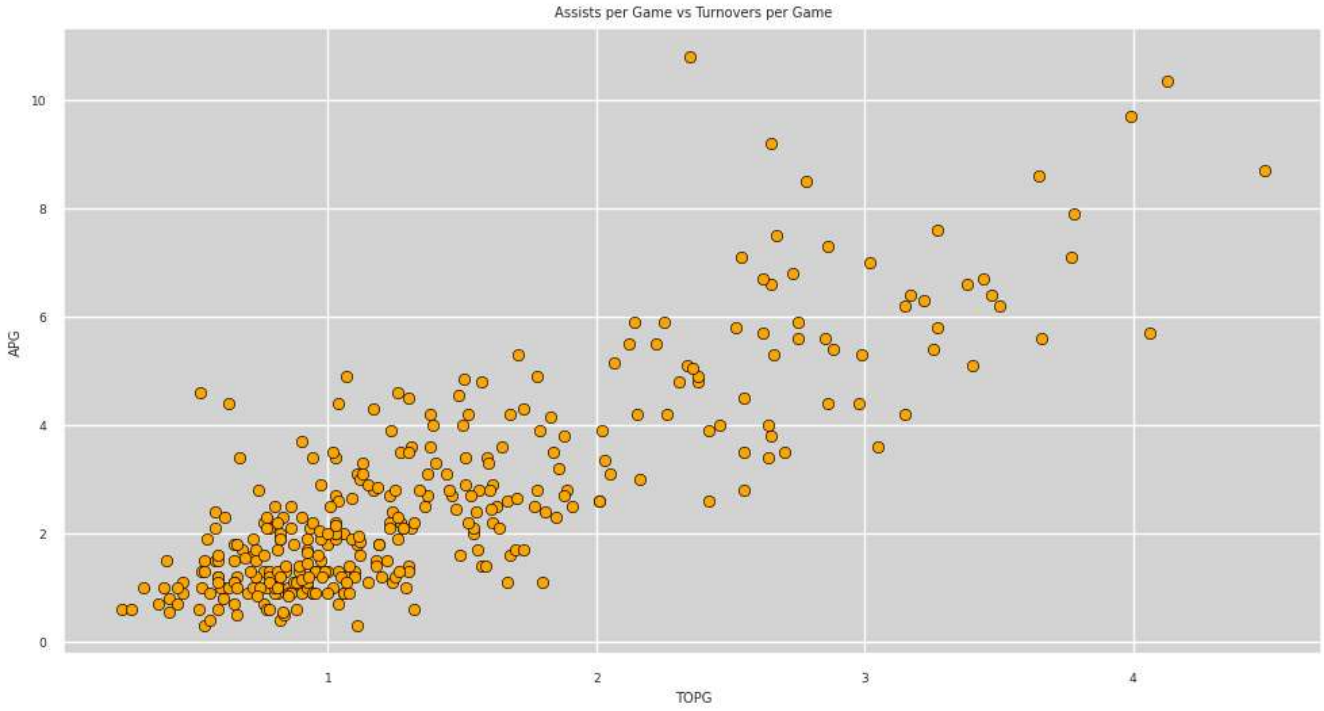
```
2PA     0.706566
Name: MPG, dtype: float64
```

Show code

```
<Figure size 1800x600 with 0 Axes>
```



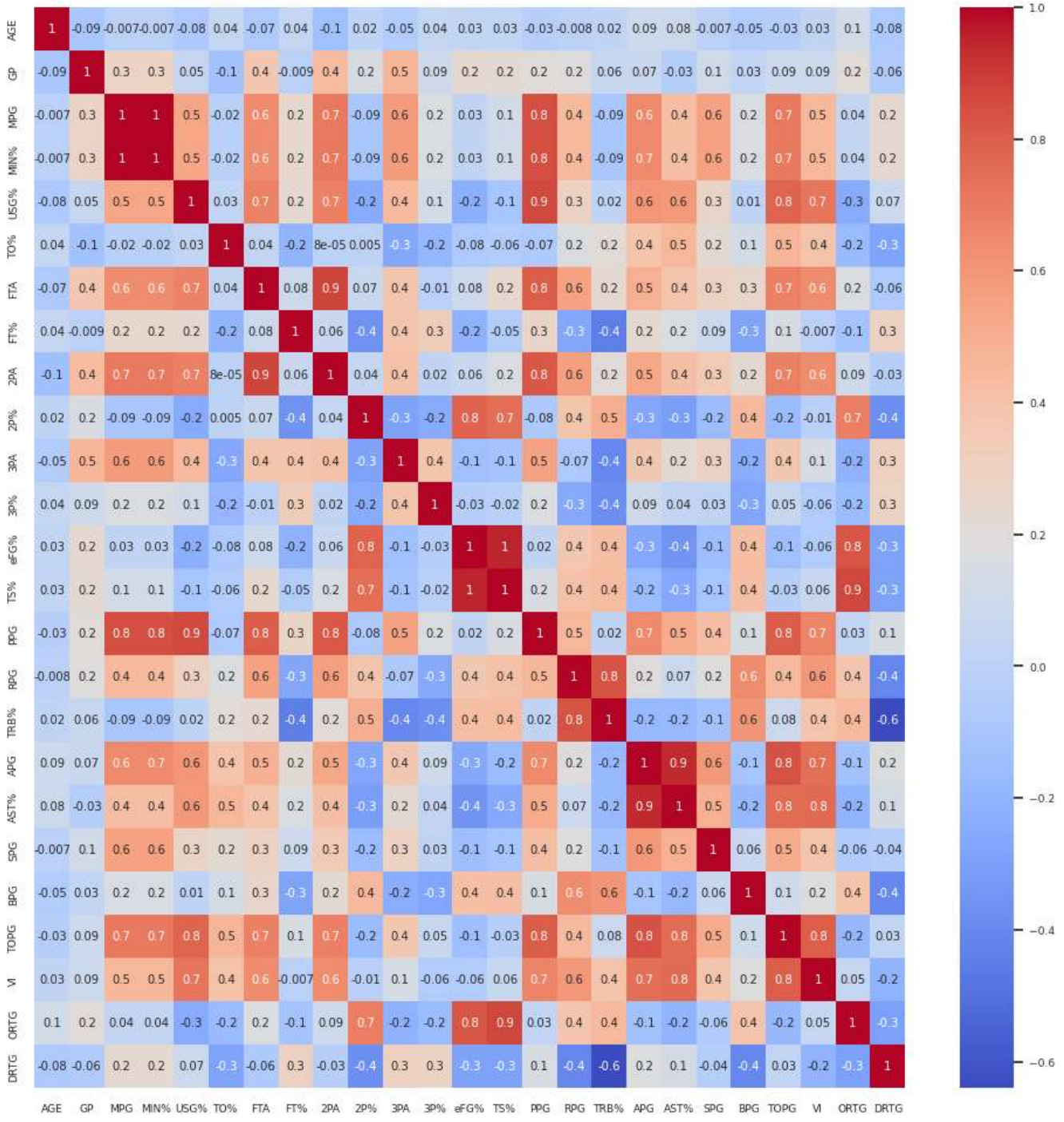
Show code



The heatmap below shows the correlation for all of the variables in the dataset. This shows that Multicollinearity is a problem that needs to be addressed.

Show code

```
<ipython-input-284-f95f7e0698b4>:5: FutureWarning: The default value of numeric_only in
  sns.heatmap(data = df[2:].corr(),
```



IV. Handling Object Features and Scale Features

A.Handling Object Features

In the dataset, the two object features, "FULL NAME" and "POS" will be preserved in its own dataframe and dropped from the main dataframe. Later, this new dataframe will be re-joined with the final dataframe.

Show code

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 344 entries, 0 to 343
Data columns (total 2 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   FULL NAME  344 non-null    object
 1   POS        344 non-null    object
dtypes: object(2)
memory usage: 5.5+ KB
```

Show code

	FULL NAME	POS
0	Precious Achiuwa	F
1	Steven Adams	C
2	Bam Adebayo	C-F

Show code

Show code

B. Scale the Features using sklearn Standard Scaler

Scaling features is essential for machine learning algorithms. Many algorithms assume that all of the features are centered around zero. Features with large variances may inhibit the algorithm's ability to learn.

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
df_scaled = scaler.fit_transform(df)
```

```
df.shape
```

```
(344, 25)
```

V. Dimensionality Reduction Using PCA and Manifold Learning

The Curse of Dimensionality

The term, "Curse of Dimensionality" was coined by the mathematician, Richard E. Bellman. to describe the problem caused by the exponential increase in volume associated with adding extra dimensions to a (mathematical) space. As the number of features increase, our data become sparser, which results in overfitting, and we therefore need more data to avoid it.

This dataset now consists of 25 features or "dimensions". Having a high number of dimensions can cause problems with clustering since clustering algorithms depend on measuring the distances between observations in order to identify the clusters. If the distances are all roughly equal, then all the observations appear equally alike (or equally different) and no meaningful clusters can be formed. High dimensionality means that space between each observation becomes equidistant.

In this project, each player (or observation) represents a combination of the 25 dimensions. The statistical distances between each player become equally alike and no cluster with any meaningful information can be formed.

High Correlation of Features

As previously discussed in the Data Exploration, Section III, Multicollinearity, we see that there are features in the dataset that are highly correlated with each other. See the Heatmap. For example, "MPG" and "MIN%" are perfectly, positively correlated. High correlations create redundancies among the features which may bias the alogrithms to give more weight to these features. Additionally, this could lead to overfitting on the training data.

To address both of these problems, this project will reduce the dimensionality space using two methods:

A. Principal Component Analysis ("PCA")

B. Manifold Learning

The project will evaluate the results of each method and then select the best approach for dimensionality reduction. Both PCA and Manifold were applied to scaled features.

A. Principal Component Analysis ("PCA")

PCA provides a method to reduce the overall dimensionality of the features while still retaining the information of the larger feature set. Here, I reduce the number features need to retain 95% of the information in the dataset.

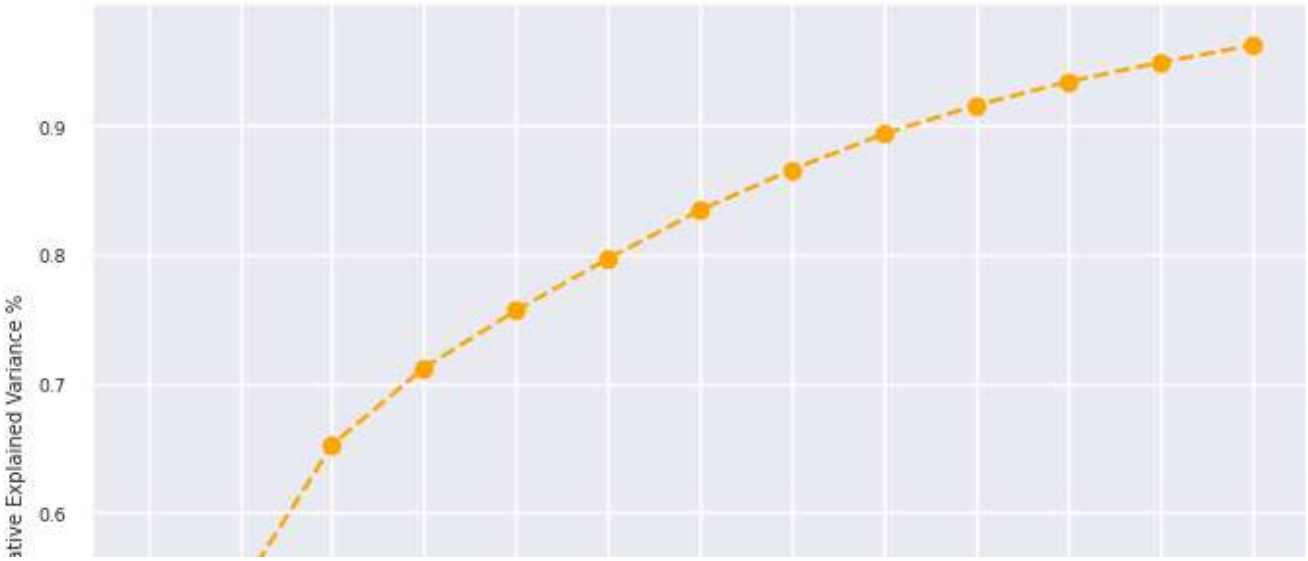
For more detailed information on how PCA analysis works, please see the article, A One-Stop Shop for Principal Component Analysis by Matt Brems (April 17, 2017) TowardsDataScience.com

```
pca = PCA(n_components = .95)
# Fit the pca object to the standardized data
principalComponents = pca.fit_transform(df_scaled)
pca.explained_variance_ratio_
```

```
array([0.32270492, 0.21640281, 0.11323169, 0.05994224, 0.04453623,
       0.0402728 , 0.03781319, 0.03140261, 0.02790885, 0.02210657,
       0.01832836, 0.01497584, 0.01314284])
```

Using 95% of the explained variance,the 29 features have now been reduced to 12.

```
#@title
plt.figure(figsize = (8,6))
features = range(pca.n_components_)
plt.plot(features, pca.explained_variance_ratio_.cumsum(), marker='o', linestyle='--', color=
plt.xlabel('PCA components')
plt.ylabel('Cumulative Explained Variance %')
plt.xticks(features);
```



PCA has its limitations. "[I]t it often fails in that it assumes that the data can be modelled linearly. PCA expressed new features as linear combinations of existing ones by multiplying each by a coefficient...Whereas PCA attempts to create several linear hyperplanes to represent dimensions, much like multiple regression constructs as an estimation of the data, **Manifold learning** attempts to learn manifolds, which are smooth, curved surfaces within the multidimensional space."

Manifold Learning t-SNE, LLE, Isomap Made Easy, by Andre Ye (08/12/2020) Towards Data Science

B. Manifold Learning

Manifold learning generally refers to an unsupervised reduction, where the class is not presented to the algorithm (but may exist). Manifold learning attempts to learn manifolds, which are smooth, curved surfaces within the multidimensional space.

For this project, I used sklearn's manifold library, **Locally Linear Embedding** or "LLE". Seeks a lower-dimensional projection of the data which preserves distances within local neighborhoods. *It can be thought of as a series of local Principal Component Analyses which are globally compared to find the best non-linear embedding.*

"In general, LLE is a more efficient algorithm as it eliminates the need to estimate pairwise distances between widely separated data points. Furthermore, it assumes that the manifold is linear when viewed locally. Thus it recovers the non-linear structure from locally linear fits", LLE: Locally Linear Embedding — A Nifty Way to Reduce Dimensionality in Python by Saul Dobilas (10/10/2021) TowardsDataScience.com

```
from sklearn.manifold import LocallyLinearEmbedding as LLE
lle = LLE(n_components=2)
df_lle = lle.fit_transform(df_scaled)
```

```
len(df_lle)
```

```
344
```

```
df.columns
```

```
Index(['AGE', 'GP', 'MPG', 'MIN%', 'USG%', 'TO%', 'FTA', 'FT%', '2PA', '2P%',
       '3PA', '3P%', 'eFG%', 'TS%', 'PPG', 'RPG', 'TRB%', 'APG', 'AST%', 'SPG',
       'BPG', 'TOPG', 'VI', 'ORTG', 'DRTG'],
      dtype='object')
```

Using LLE, I reduced the 29 dimension feature set down to just two features which was fitted and transformed on the scaled data. These two features will now be concatenated with the original dataset and loaded into the final_df.

```
final_df = pd.concat([df, pd.DataFrame(df_lle)],axis=1)
final_df.columns.values[-2:] = ['LLE_Component 1', 'LLE_Component 2']
```

```
final_df.head(2)
```

	AGE	GP	MPG	MIN%	USG%	TO%	FTA	FT%	2PA	2P%	...	APG	AST%	SPG	BPG
0	22.56	73	23.6	49.2	18.5	11.3	131.0	0.595	447.0	0.468	...	1.1	6.9	0.51	0.56
1	28.73	76	26.3	54.8	12.0	19.6	199.0	0.543	383.0	0.548	...	3.4	16.1	0.87	0.79

VI. Application of Unsupervised Learning Algorithms:

> A. KMeans Clustering
> B. Gaussian Mixture Models ("GMM")
> C. Agglomerative Hierarchy Clustering("AGC")**

As stated previously, there are various forms of unsupervised machine learning algorithms. For this project, my goal is to use three different forms and see if they agree on an elite cluster. The three forms are as follows:

- **Centroid based Clustering (KMeans)** divides clusters into non-overlapping groups. Each observation belongs to only one cluster
- **Distribution based clustering (GMM)** uses probabilistic models to group data points into clusters based on their similarity

- **Hierarchical Clustering ("AGC")** determines cluster assignments by building a hierarchy which is implemented by either a bottom-up or a top-down approach. AGC uses a bottom up approach which merges points that are most similar until one or two main clusters are left.

A. KMeans Clustering

The algorithm works by grouping data together into a fixed number of clusters. This number of clusters is the "K" in KMeans clustering.

Data points are assigned to clusters based on their distance from the centroid of the cluster. It then calculates the means of each cluster. It iterates this process by taking the variation of the cluster.

"The quality of the cluster assignments is determined by computing the sum of the squared error (SSE) after the centroids converge, or match the previous iteration's assignment. The SSE is defined as the sum of the squared Euclidean distances of each point to its closest centroid. Since this is a measure of error, the objective of k-means is to try to minimize this value."

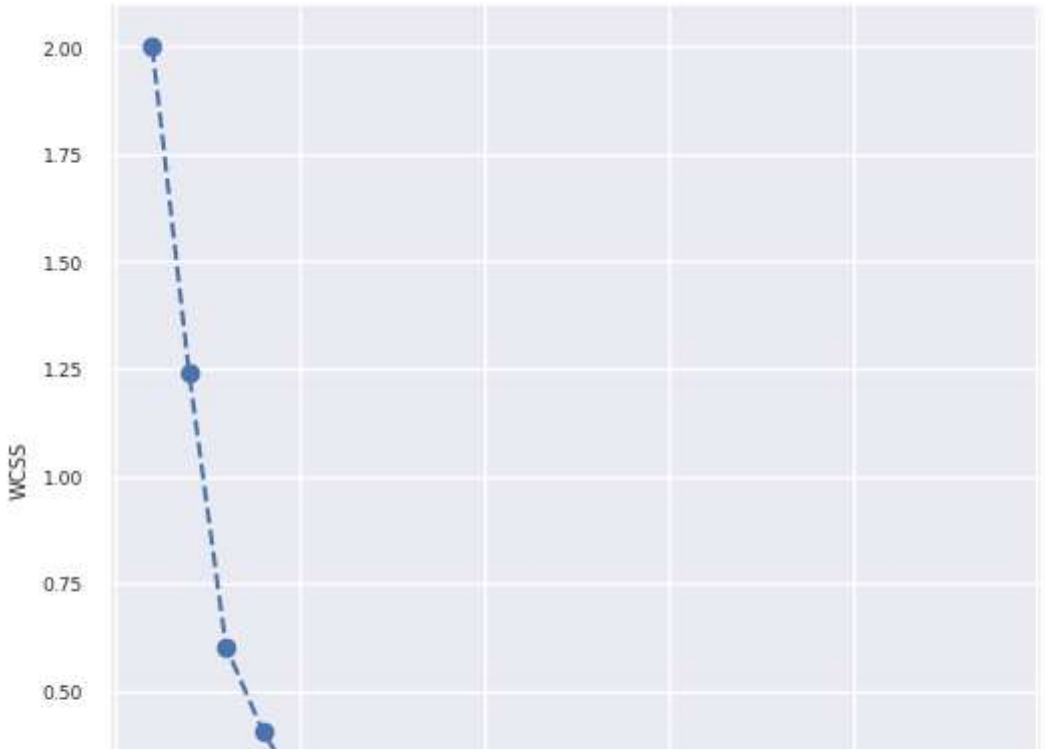
Kevin Arvai, *K-Means Clustering in Python: A Practical Guide*, RealPython.com(2021)

```
#Below, I tried out a number of clusters ranging from 1 to 25.
#Each cluster was fit onto the reduced number of features df_lle.
clusters = []
for i in range(1,25):
  kmeans_pca = KMeans(n_clusters=i, init='k-means++', n_init=10, random_state=42)
  kmeans_pca.fit(df_lle)
  clusters.append(kmeans_pca.inertia_)
```

The number of clusters is determined by plotting each cluster against the within cluster sum of squares (WCSS) sum of the squared deviations from each observation and the cluster centroid.

When the graph levels out after steep declines shows us the number of clusters.

```
plt.figure(figsize=(6,6))
plt.plot(range(1,25), clusters, marker='o', linestyle='--')
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS')
plt.show();
```




```
from kneed.knee_locator import KneeLocator
k1 = KneeLocator(range(1, 25), clusters, curve="convex", direction="decreasing")
k1.elbow
```

```
5
```

Number of Clusters

The number of clusters bend at 5.

```
kmeans = KMeans(n_clusters=5, init='k-means++', n_init='auto', random_state=42,verbose=0)
kmeans.fit(df_lle)
```

```
KMeans
KMeans(n_clusters=5, n_init='auto', random_state=42)
```

```
kmeans.labels_
```

```
array([1, 4, 2, 3, 3, 4, 1, 1, 2, 0, 2, 0, 1, 3, 0, 2, 0, 3, 0, 0, 0, 2,
       0, 3, 3, 1, 2, 3, 0, 0, 3, 3, 1, 0, 0, 2, 3, 3, 2, 3, 1, 0, 0, 1,
       3, 2, 0, 3, 3, 0, 2, 0, 1, 4, 0, 1, 1, 4, 0, 4, 3, 0, 3, 0, 3, 3,
       2, 2, 0, 1, 3, 2, 1, 2, 0, 3, 1, 0, 1, 2, 2, 1, 1, 3, 2, 3, 3, 0,
       2, 1, 4, 0, 2, 3, 0, 3, 2, 2, 4, 0, 0, 0, 0, 3, 0, 3, 0, 3, 3, 3,
       1, 3, 3, 1, 0, 3, 0, 0, 3, 3, 1, 0, 3, 2, 3, 2, 3, 0, 1, 3, 0, 0,
       1, 1, 3, 2, 0, 0, 1, 3, 0, 2, 1, 0, 3, 0, 3, 2, 3, 3, 3, 1, 0, 1,
       0, 3, 3, 3, 1, 1, 3, 0, 2, 3, 1, 3, 0, 3, 3, 4, 3, 0, 2, 1, 0, 1,
       0, 3, 3, 3, 3, 3, 0, 1, 3, 3, 3, 4, 1, 1, 1, 2, 3, 1, 1, 2, 0, 0,
       2, 0, 1, 0, 2, 1, 3, 3, 3, 3, 0, 1, 3, 4, 3, 3, 3, 3, 0, 2, 1, 3,
       4, 0, 1, 2, 2, 3, 0, 3, 1, 3, 1, 2, 3, 3, 3, 3, 1, 4, 3, 1, 1, 0,
       1, 2, 3, 3, 1, 2, 0, 0, 1, 3, 3, 3, 1, 1, 0, 2, 3, 3, 1, 0, 3, 3,
```

```
2, 0, 3, 3, 0, 3, 2, 0, 3, 0, 3, 0, 3, 1, 1, 1, 3, 1, 3, 2, 0, 4,
3, 0, 4, 1, 3, 3, 3, 3, 0, 3, 2, 4, 1, 4, 1, 3, 0, 0, 1, 0, 3, 3,
3, 1, 0, 0, 1, 3, 1, 1, 1, 3, 2, 1, 0, 3, 3, 1, 1, 0, 1, 3, 3, 4,
1, 1, 0, 1, 3, 3, 0, 0, 3, 3, 3, 3, 3, 2], dtype=int32)
```

The KMeans cluster labels will be added to the dataframe.

```
final_df['KMeans Clusters'] = kmeans.labels_
```

B. Gaussian Mixture Models (GMM)

GMM models look for a mixture of multi-dimensional Gaussian probability distributions to fit the data. It's more of a probability distribution or a mixture of different distributions.

"GMMs can be used to find clusters in data sets where the clusters may not be clearly defined. Additionally, GMMs can be used to estimate the probability that a new data point belongs to each cluster. Gaussian mixture models are also relatively robust to outliers, meaning that they can still yield accurate results even if there are some data points that do not fit neatly into any of the clusters. This makes GMMs a flexible and powerful tool for clustering data. It can be understood as a probabilistic model where Gaussian distributions are assumed for each group and they have means and covariances which define their parameters."

Ajitesh Kumar, *Gaussian Mixture Models: What are they & when to use?*, Data Analytics (April 14, 2022)

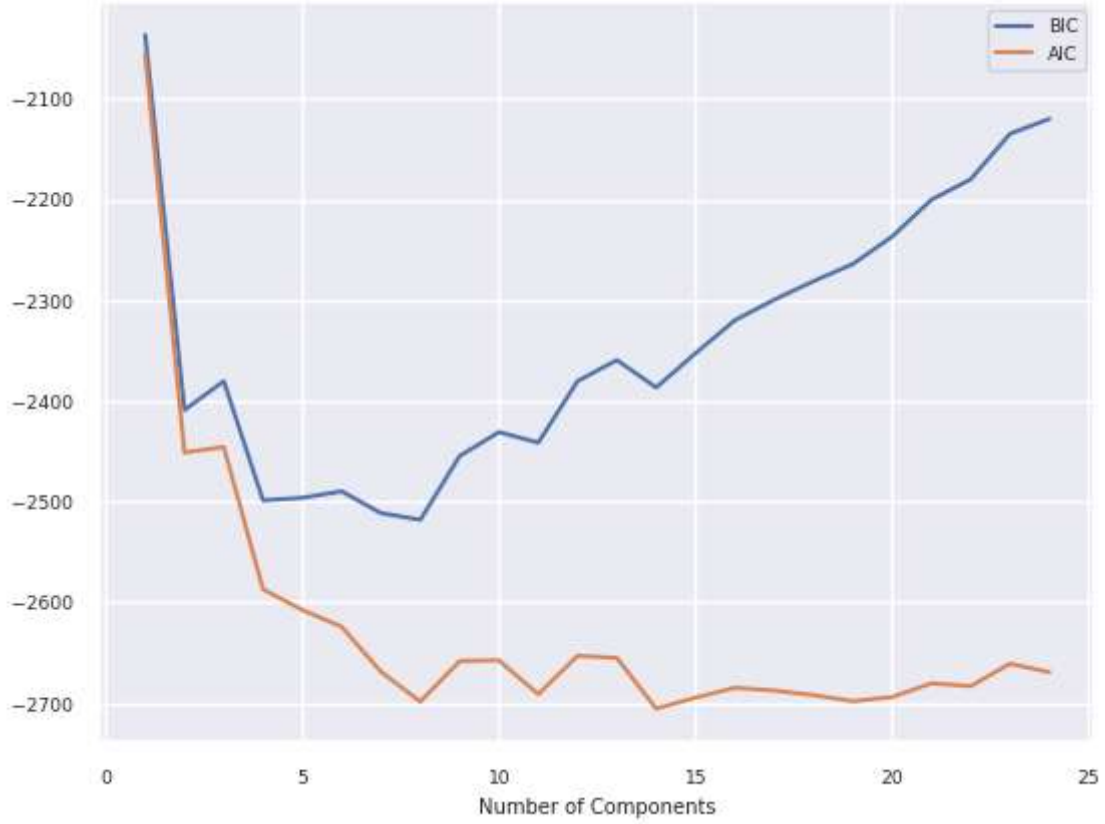
Finding the Optimal number of components

Similar to the KMeans approach, to discover the correct number of components for the algorithm is a trial and error process. I created a list comprehension which applied the Gaussian Mixture algorithm for 25 components.

Below, I created a list comprehension that fits the model on the two features per each component. The **Akaike Information Criterion ("AIC")** and **Bayesian Information Criteria ("BIC")** are used to find the optimal number of components. Output from the AIC and BIC are plotted.The optimal number of components is determined by choosing the model with the lowest AIC or BIC value.In the model below, the best GMM model for the data is with 4 components.

```
#Find the number of optimal components

n_components = np.arange(1,25)

#Create a GMM model
```

```
model = [GaussianMixture(n_components=n,
                         random_state=42).fit(df_lle) for n in n_components ]
#Plot the model
plt.plot(n_components, [m.bic(df_lle) for m in model], label="BIC")
plt.plot(n_components, [m.aic(df_lle) for m in model], label="AIC")
plt.legend()
plt.xlabel('Number of Components');
```



Building the Model and Adding the GMM Segments

The GMM model is fitted and predicted which produces 4 clusters.

```
model = GaussianMixture(n_components=4,
                        random_state=1502).fit(df_lle)
```

```
#Predict for each cluster
segments = pd.Series(model.predict(df_lle))
final_df['GMM Clusters'] = segments
```

```
final_df['GMM Clusters'].value_counts().sort_index()
```

```
0    119
1     46
2     45
```

```
3    134
Name: GMM Clusters, dtype: int64
```

C. Agglomerative Hierarchy clustering

Agglomerative hierarchy clustering is bottom-up hierarchial clustering algorithm. It starts with each data point as a separate cluster and iteratively merge the closest pairs of clusters until a stopping criterion is met. This creates a hierarchy of clusters that can be visualized as a dendrogram. Common linkage methods for merging clusters include single linkage, complete linkage, and average linkage, among others.

```
from scipy.cluster.hierarchy import dendrogram, linkage
plt.figure(figsize=(12,6))
linkage_data = linkage(df_lle, method='ward', metric='euclidean')
dendrogram(linkage_data)
plt.tight_layout()
plt.show()
```

The above plot is a dendrogram which is a type of tree diagram showing hierarchical clustering — relationships between similar sets of data. Using this diagram, we can train the algorithm with 4 clusters. These clusters are added to the final dataframe.

```
import random
random.seed(42)
from sklearn.cluster import AgglomerativeClustering as AGC
agc = AGC(n_clusters=4, linkage="ward")
agc.fit(df_lle)
```

```
AgglomerativeClustering(n_clusters=4)
```

```
final_df['AGC Clusters'] = agc.labels_
```

```
final_df['AGC Clusters'].value_counts().sort_index()
```

```
0    205
1     59
2     63
3     17
Name: AGC Clusters, dtype: int64
```

```
final_df = pd.concat([FULL_NAME_POS, final_df], axis=1)
```

All of the clusters have been added to the final dataframe.

```
final_df[["FULL NAME", "LLE_Component 1", "LLE_Component 2", "KMeans Clusters","GMM Clusters'
```

	FULL NAME	LLE_Component 1	LLE_Component 2	KMeans Clusters	GMM Clusters	AGC Clusters
0	Precious Achiuwa	0.039434	-0.047847	1	3	2
1	Steven Adams	0.026499	0.093110	4	3	3
2	Bam Adebayo	-0.082642	0.006002	2	2	1
3	LaMarcus Aldridge	0.023765	0.014292	3	0	0

NBA MVPs

Every year, the NBA names the Most Valuable Player ("MVP) of the league. The last 10 NBA MVPs are:

- LeBron James
- Kevin Durant
- Stephen Curry
- Russell Westbrook
- James Harden
- Giannis Antetokounmpo
- Nikola Jokic

Using the last 10 NBA MVPs (some were repeat), we can see that they all belong to the same clusters for each algorithm, KMeans and GMM which are cluster 2 and AGC which is cluster one.

Show code

	FULL NAME	KMeans Clusters	GMM Clusters	AGC Clusters
8	Giannis Antetokounmpo	2	2	1
318	James Harden	2	2	1
79	Kevin Durant	2	2	1
141	LeBron James	2	2	1
147	Nikola Jokic	2	2	1
283	Russell Westbrook	2	2	1
67	Stephen Curry	2	2	1

The function elite_tag tags each player with "elite" if they meet this criteria.

```
def elite_tag(df):
  if df['KMeans Clusters'] == 2 and df['GMM Clusters'] ==2 and df['AGC Clusters'] == 1:
    return "elite"
  else:
    return "non_elite"
```

D. Elite Players

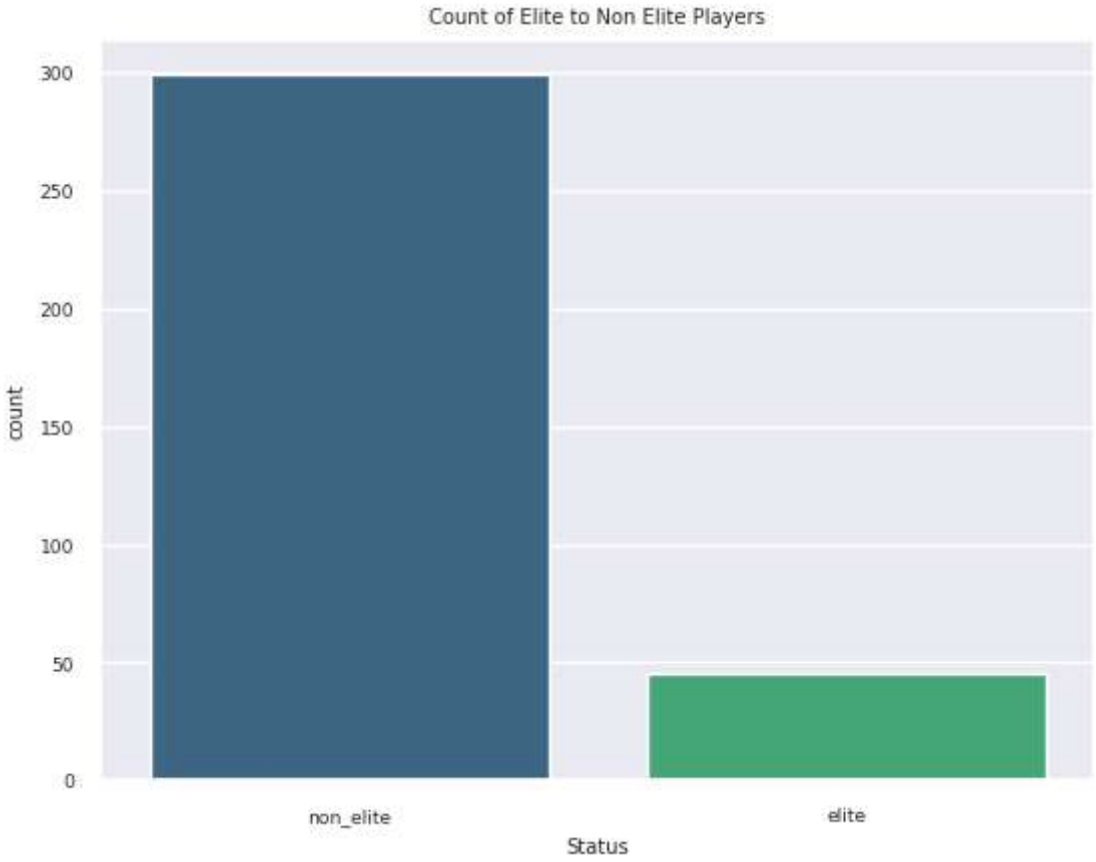
Of the 344 players in the dataset, 45 are elite. Visually, we can see why these players are elite.

Show code

Show code

```
non_elite    299
elite         45
Name: Status, dtype: int64
```

Show code



Characteristics of Elite Players -- Younger and Play More

To evaluate the three algorithms' ability to identify the elite cluster of NBA players, let's start by looking at key characteristics

POS = Position of the player
AGE = Age of the player

GP = No. of games the player played

MPG = No. of minutes a player played per game

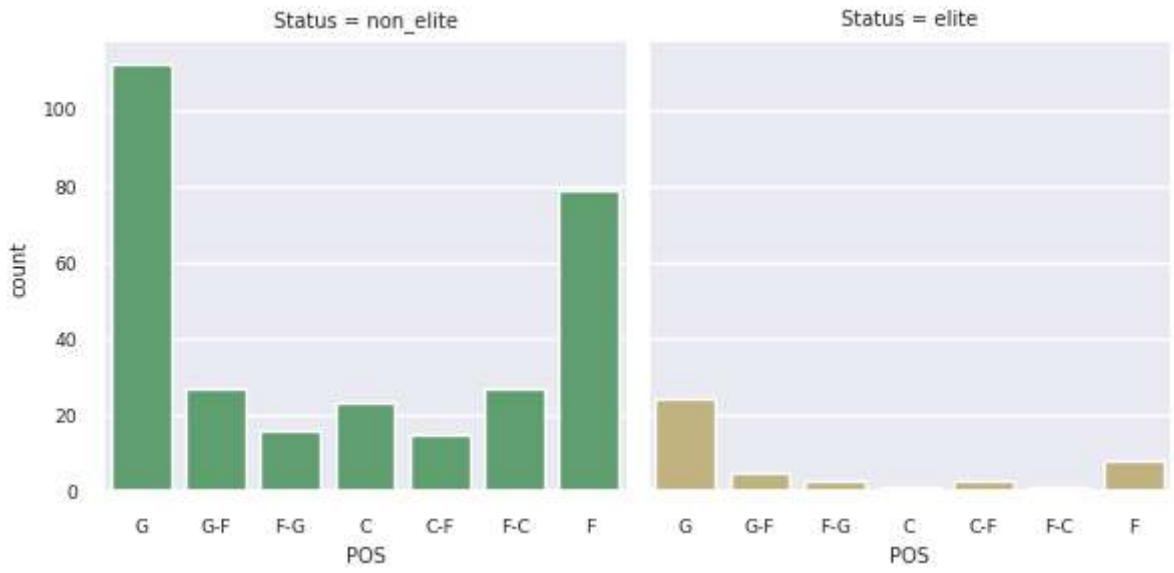
MIN% = Minutes Percentage is the percentage of team minutes used by a player while he was on the floor

USG% = Usage rate percentage is an estimate of the percentage of team plays used by a player while he was on the floor

Player Positions

Most of the elite players play the pure guard position. Beyond that position, it becomes rare to find elite players, especially at the pure center position. Out of 24 players at that position, there is only one player, Nikola Jokic, who is elite.

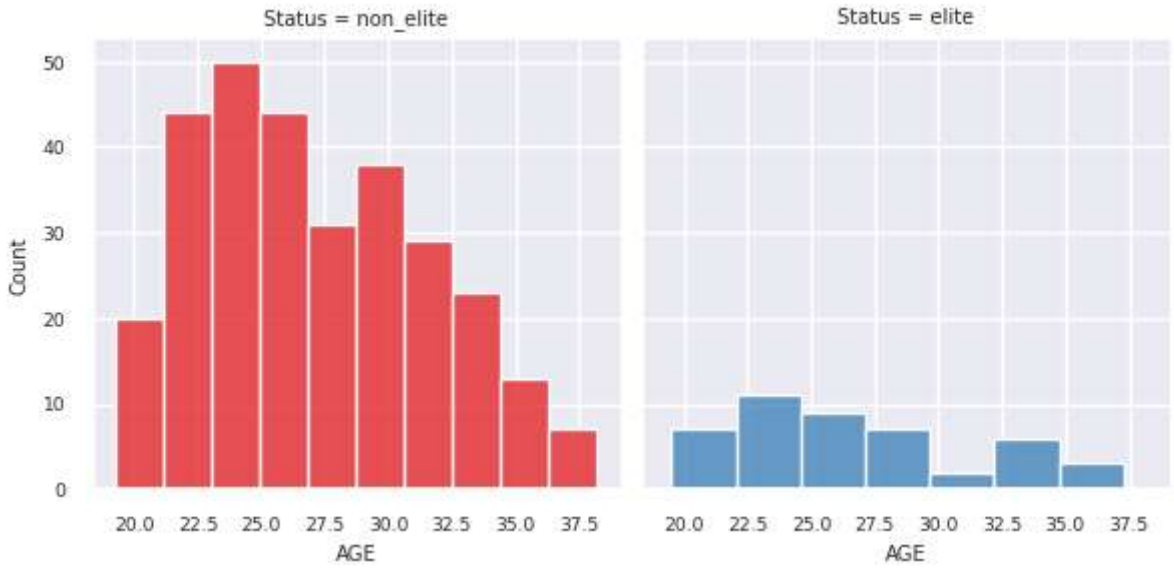
```
#@title
POS = final_df[["POS", "Status"]]
order = ["G", "G-F", "F-G", "C", "C-F", "F-C", "F"]
colors = {'color': ['g', 'y']}
g = sns.FacetGrid(POS, col="Status", hue_kws=colors, hue='Status')
g.map(sns.countplot,"POS",order=order)
plt.show()
```



Age The age of non-elite players follows a pattern. Generally, the number of players is higher for players under 27 and then trails off as players age.

For elite players, there's a sharp drop off in the number of players after age 27.

Show code



Games Played and Minutes per Game

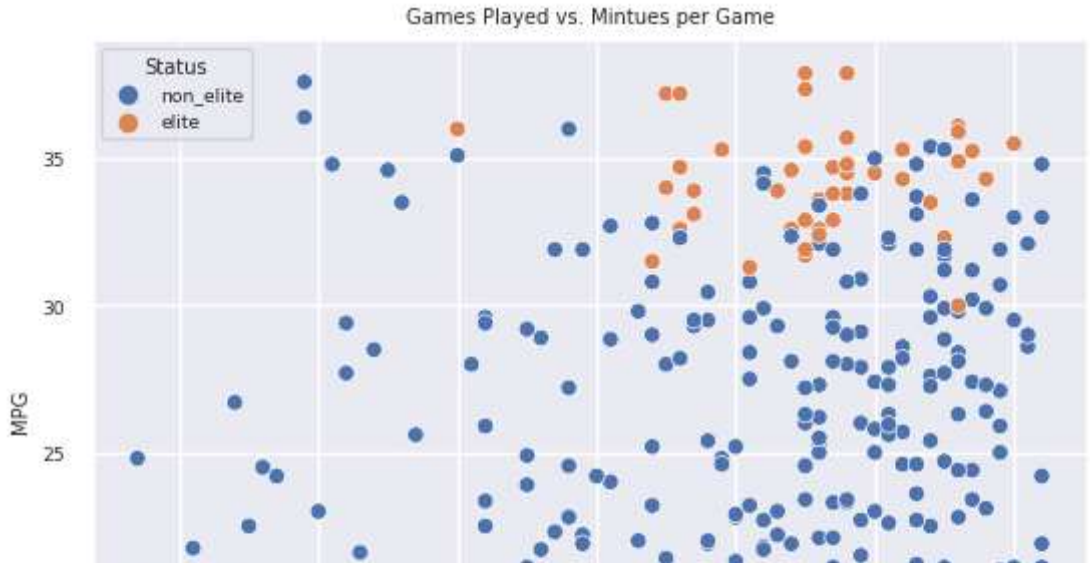
Elite players average seven more games played and 10 more minutes per game than their non-elite counterparts. These differences represent 8.5% of all games played and 21% of minutes per game.

Show code

Status	GP	MPG
elite	65.977778	34.222222
non_elite	59.036789	23.972687

The scatter plot of Games Played vs Minutes per Game shows the upper right quadrant has a higher number of elite players. Although there are non-elite in this quadrant as well, the elite players play more games and more minutes. As seen earlier, Minutes per Game and Points per Game are highly correlated.

Show code



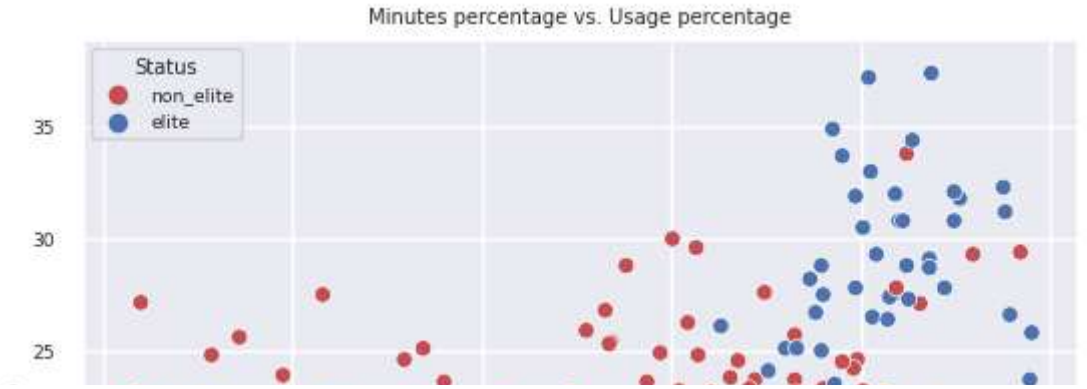
MIN% and USG%

Minutes Percentage is the percentage of team minutes used by a player while he was on the floor. ***Usage rate percentage*** is an estimate of the percentage of team plays used by a player while he was on the floor. Taken together these two features demonstrate how important a player is to the team's objective. Elite player average 71% of the team minutes and 28% of the team's plays.

Show code

Status	MIN%	USG%
elite	71.300000	28.215556
non_elite	49.940245	18.234894

Show code



Characteristics of Elite Players -- Take More Shots

FTA = Free throw attempts
FT% = Percentage of free throws made to free throws attempted
2PA = Two point shots attempted
2P% = Percentage of two points made to two point shots attempted
3PA = Three point shots attempted
3P% = Percentage of three points made to two point shots attempted

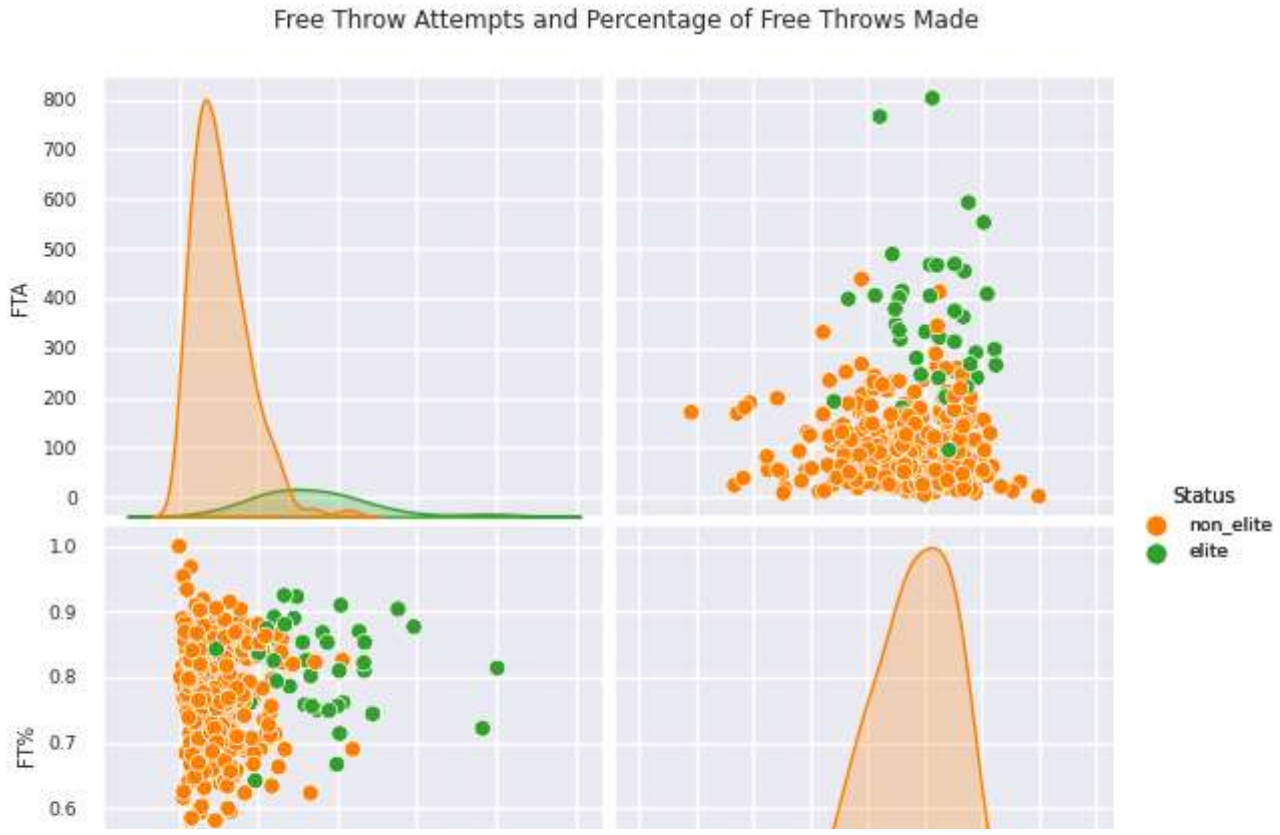
Elite players, on average, get more free throws than non-elite players by a margin of 3:1. They take more Two and Three point attempts as well. However, there's little difference on percentage made on Free Throws, Two and Three point attempts on average. Yet, when looking at the standard deviation on percentages made, the elite players show more consistency. The violin plots for FT%, 2P%, and 3P% show a narrower distritbution for elite player vs. non-elite players. In other words, outliers among the non-elite has an impact on the averages for these dimensions.

Show code

Status	FTA	FT%	2PA	2P%	3PA	3P%
elite	339.711111	0.8157	721.933333	0.520400	378.211111	0.340633
non_elite	105.144928	0.7678	268.707358	0.535128	190.741360	0.325491

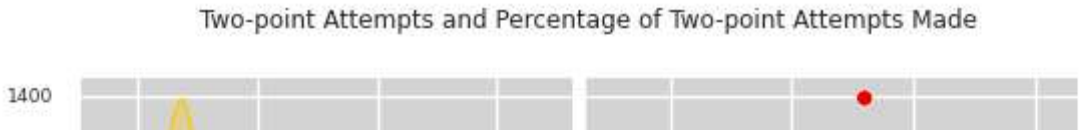
Show code

```
<Figure size 1800x600 with 0 Axes>
```



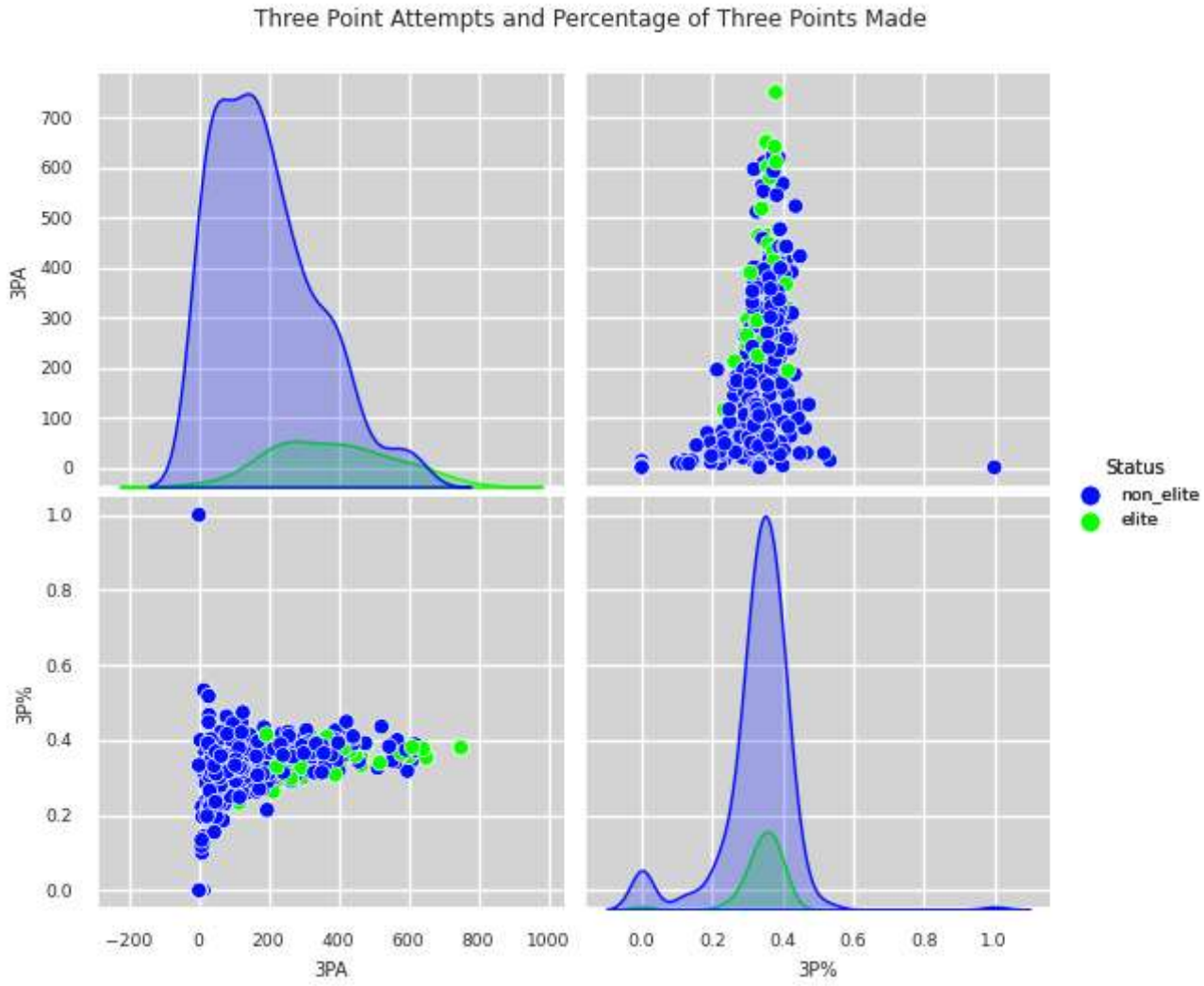
Show code

```
<Figure size 1800x600 with 0 Axes>
```




```
#@title
plt.figure(figsize=(18,6))
g = sns.pairplot(final_df, vars=["3PA","3P%"], hue="Status", palette="hsv_r")
g.fig.suptitle("Three Point Attempts and Percentage of Three Points Made", y=1.05)
g.add_legend()
plt.show()
```

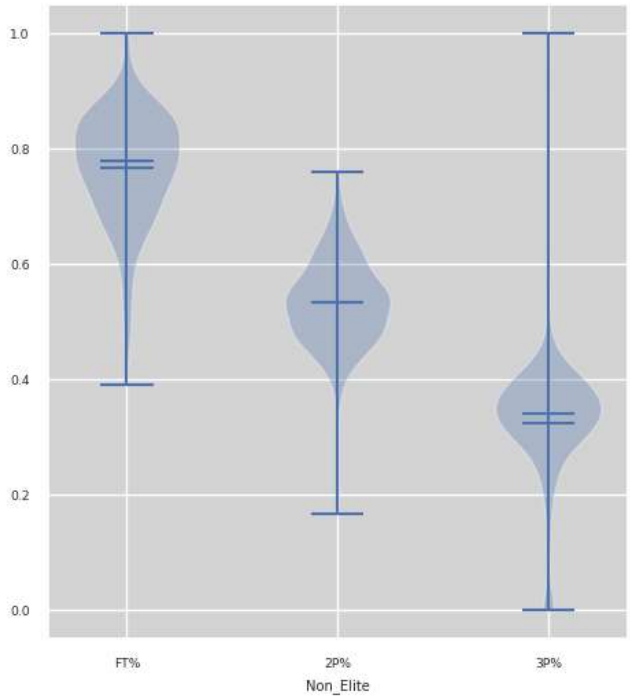
```
<Figure size 1800x600 with 0 Axes>
```

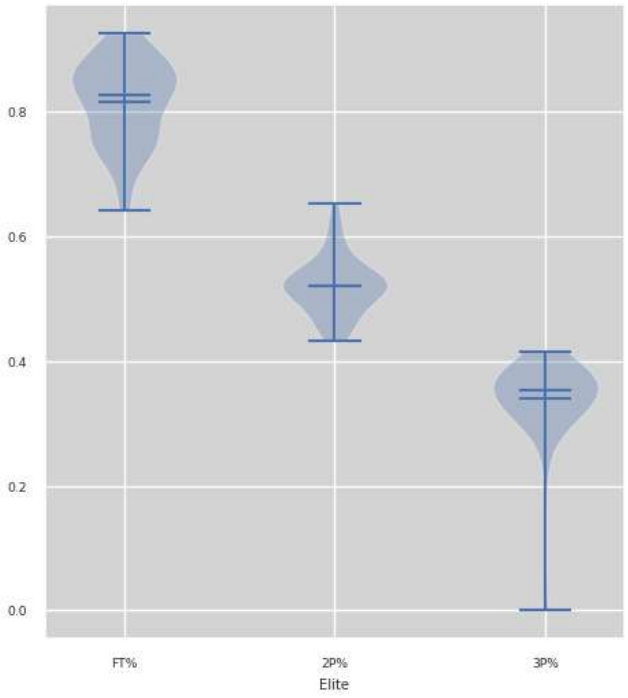


Show code

Status	FT%	2P%	3P%
elite	0.067855	0.046874	0.064933

Show code





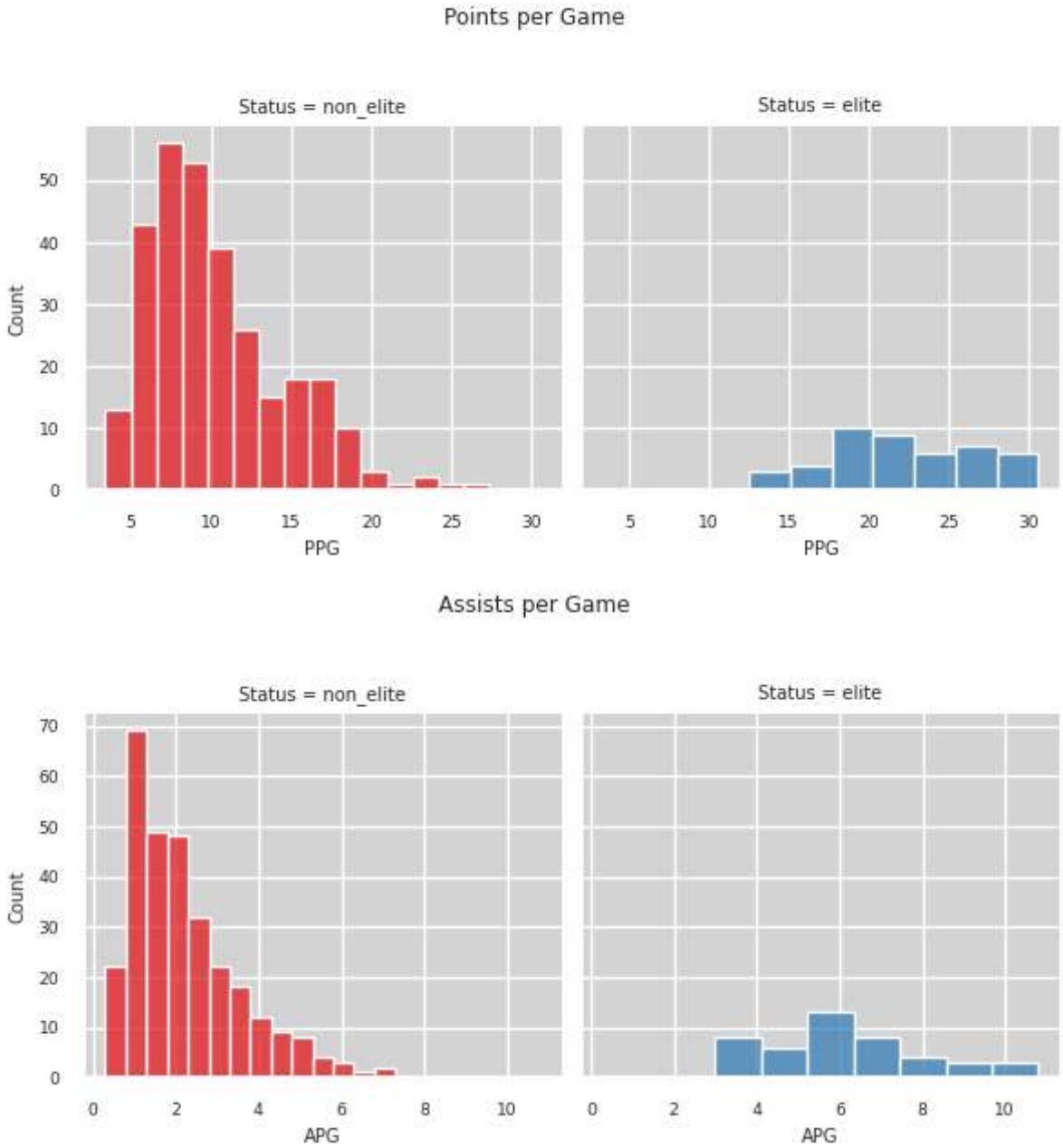
Characteristics of Elite Players -- Score, Rebounds, Steals, Assists, and Blocks

PPG = No. of points per game
RPG = No. of rebounds per game
APG = No. of assists per game.
SPG = No. of steals per game
BPG = No. of Blocks per game

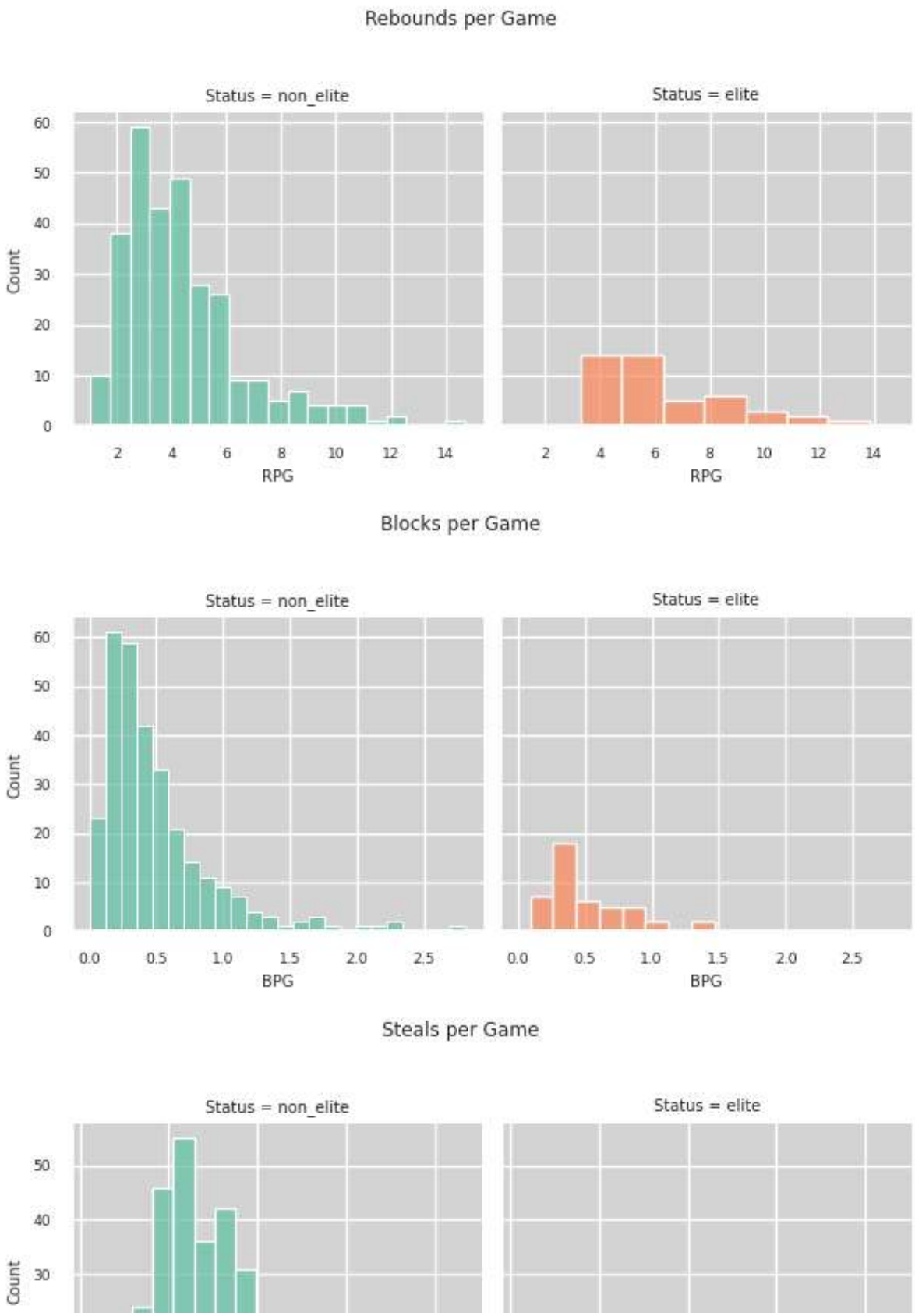
Except for Blocks per game, players in the elite cluster score more points per game, more rebounds, more assists and more steals.

Show code

Show code



Show code



Characteristics of Elite Players -- Effective Shooting %, True Shooting %, Offensive and Defensive Rating

Effective Shooting Percentage ("eFG%") Three-point shots made are worth 50% more than two-point shots made. eFG% Formula=(FGM+ (0.5 x 3PM))/FGA

True Shooting Percentage ("TS%") is a measure of shooting efficiency that takes into account field goals 3-point field goals and free throws.

Offensive Rating ("ORTG") is the number of points produced by a player per 100 total individual possessions

Defensive Rating ("DRTG") estimates how many points the player allowed per 100 possessions he individually faced while staying on the court

Averages

Show code

Status	eFG%	TS%	ORTG	DRTG
elite	0.526222	0.573189	113.086667	107.065556
non_elite	0.537698	0.566973	113.702341	106.744259

Median

Show code

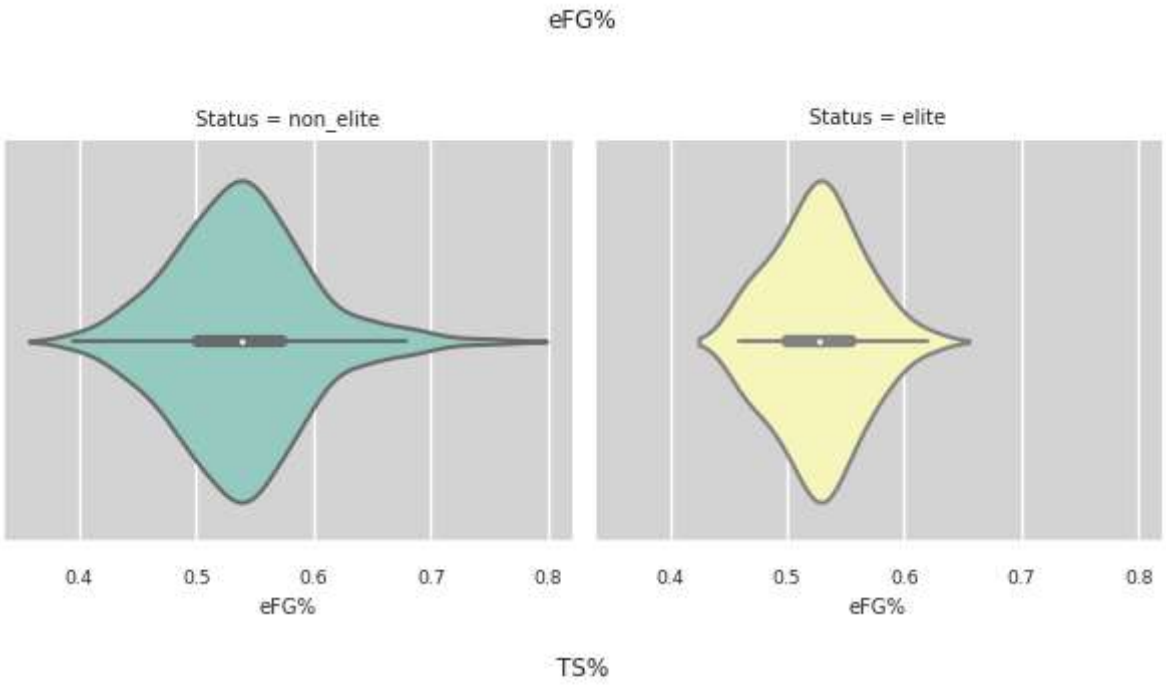
Status	eFG%	TS%	ORTG	DRTG
elite	0.528	0.576	113.4	107.55
non_elite	0.539	0.564	112.9	106.90

Standard Deviations

Show code

Status	eFG%	TS%	ORTG	DRTG
elite	0.037142	0.039254	6.983710	3.829696
non_elite	0.058879	0.053590	9.512182	4.714658

Show code



Characteristics of Elite Players -- Versatility Index ("VI")

Versatility index is a metric that measures a player's ability to produce in points, assists, and rebounds. The average player will score around a five on the index while top players score above 10.

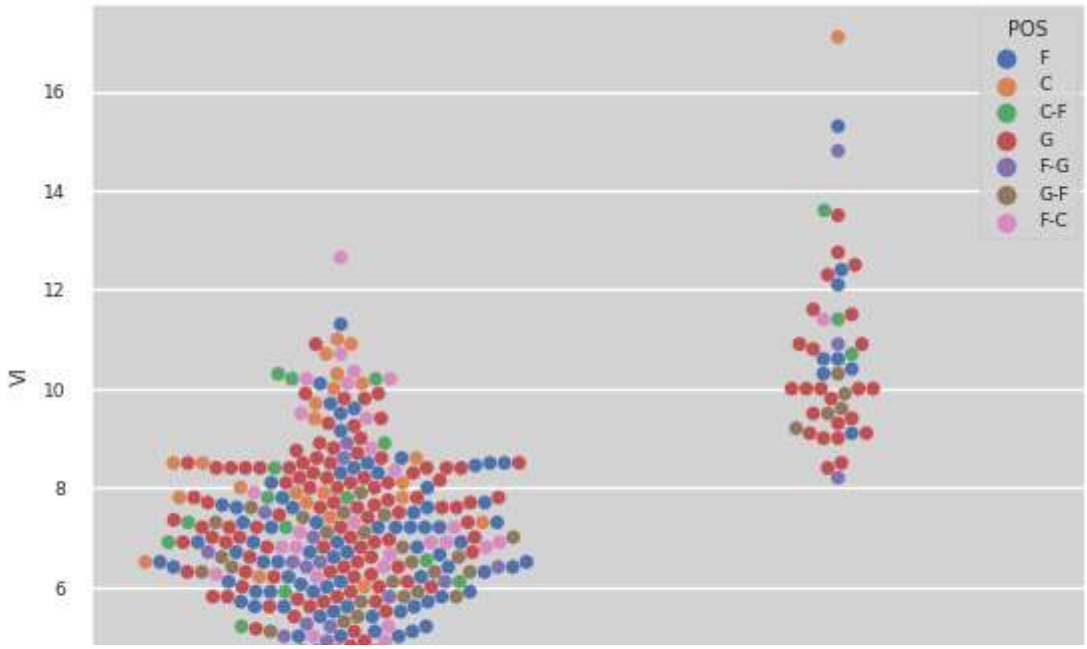
As shown in the swarmplot, elite players average over 10 on this measure.

Show code

Status	VI
elite	10.783333
non_elite	7.134392

Versatility Index Comparison between Elite and Non-Elite Players

Show code



Summary of Findings

Using unsupervised machine learning tools to identify elite players appears to be successful. Each algorithm correctly clustered the better players into their own category, and this was proven by comparing the statistical performance of elite players to non-elite players.

So far, the top 7.4% of players have been identified, but can it find the top 5% of players? In the next section, I re-run the process on just the elite cluster to find the elite cluster within the elite cluster.

VII. Finding the Elite of the Elite

At this point, three unsupervised machine learning algorithms have found the top 7.4% of elite NBA players have been identified. Using this same process but this time using only on the elite cluster, will these three algorithms find the elite among the elite?

The Shape of the Elite DataFrame

Show code

```
(45, 33)
```

Start by filtering out the elite players into their own dataframe. Once complete, save the Locally Linear Embedding components from the Manifold learning process into their series.

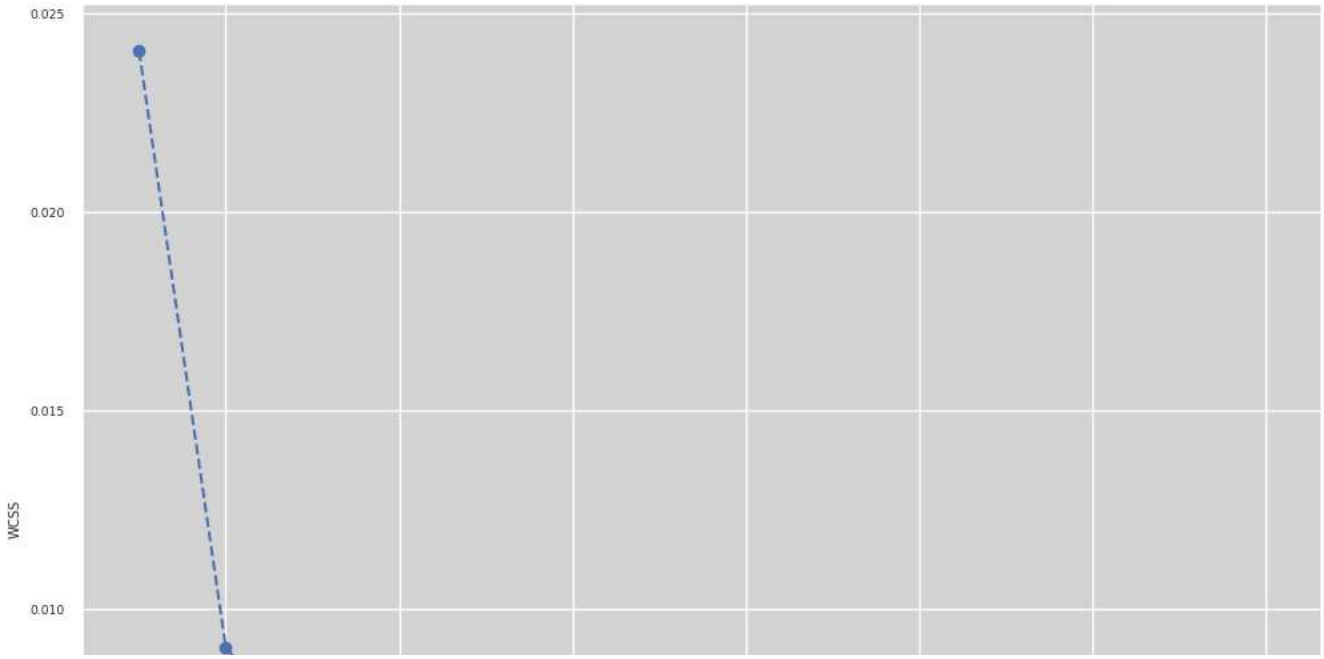
Show code

Next, using the three unsupervised algorithms on the Locally Linear Embedding components for the elite players only, find the KMeans, GMM, and AGC clusters for theses elite players.

A. KMeans

The KMeans algorithm identified three clusters among the elite cluster.

```
#@title
clusters = []
for i in range(1,15):
  kmeans_pca = KMeans(n_clusters=i, init='k-means++', n_init=10, random_state=42)
  kmeans_pca.fit(elite_lle)
  clusters.append(kmeans_pca.inertia_)
plt.figure(figsize=(12,10))
plt.plot(range(1,15), clusters, marker='o', linestyle='--')
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS')
plt.show();
```




```
k1 = KneeLocator(range(1, 15), clusters, curve="convex", direction="decreasing")
k1.elbow
```

```
3
```

```
random.seed(42)
kmeans = KMeans(n_clusters=3, init='k-means++', n_init='auto', random_state=42,verbose=0)
kmeans.fit(elite_lle)
```

```
KMeans(n_clusters=3, n_init='auto', random_state=42)
```

```
elite_df["KMeans Elite Clusters"] = kmeans.labels_
```

```
elite_df["KMeans Elite Clusters"].value_counts()
```

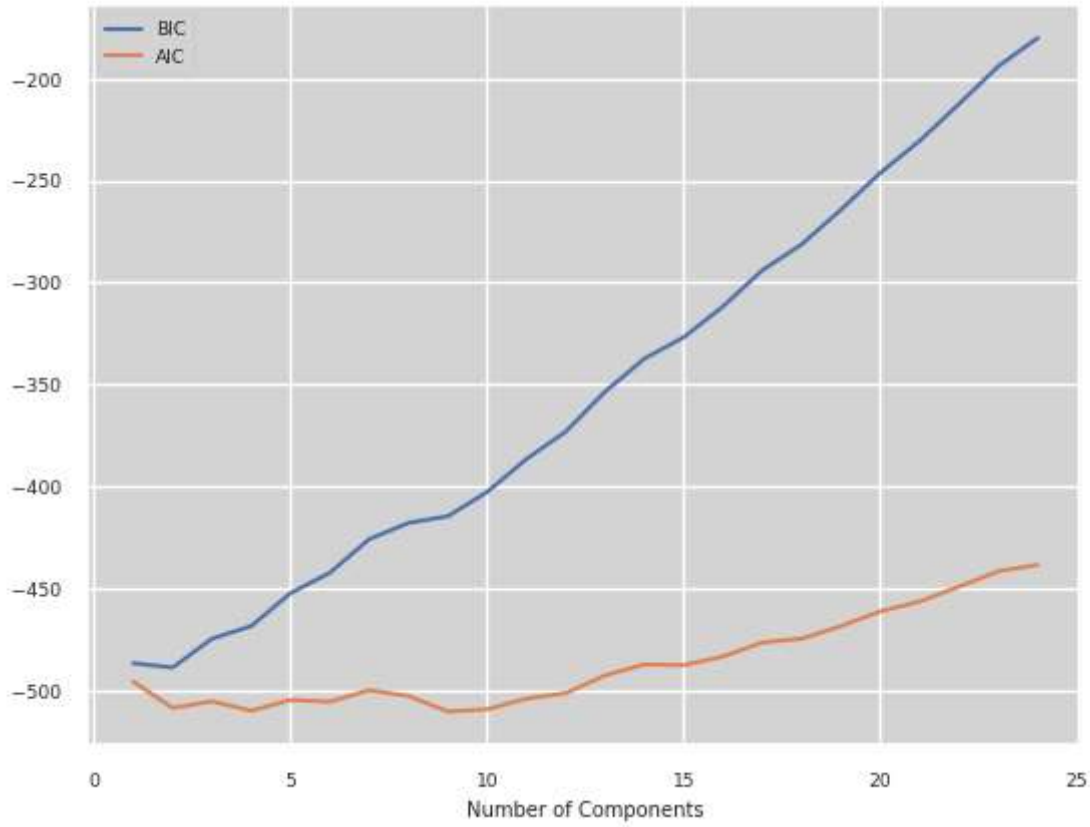
```
0    31
1     8
2     6
Name: KMeans Elite Clusters, dtype: int64
```

B. GMM

The Gaussian Mixture Model only found two clusters.

```
#Find the number of optimal components

n_components = np.arange(1,25)
```

```
#Create a GMM model
model = [GaussianMixture(n_components=n,
                         random_state=42).fit(elite_lle) for n in n_components ]
#Plot the model
plt.plot(n_components, [m.bic(elite_lle) for m in model], label="BIC")
plt.plot(n_components, [m.aic(elite_lle) for m in model], label="AIC")
plt.legend()
plt.xlabel('Number of Components');
```



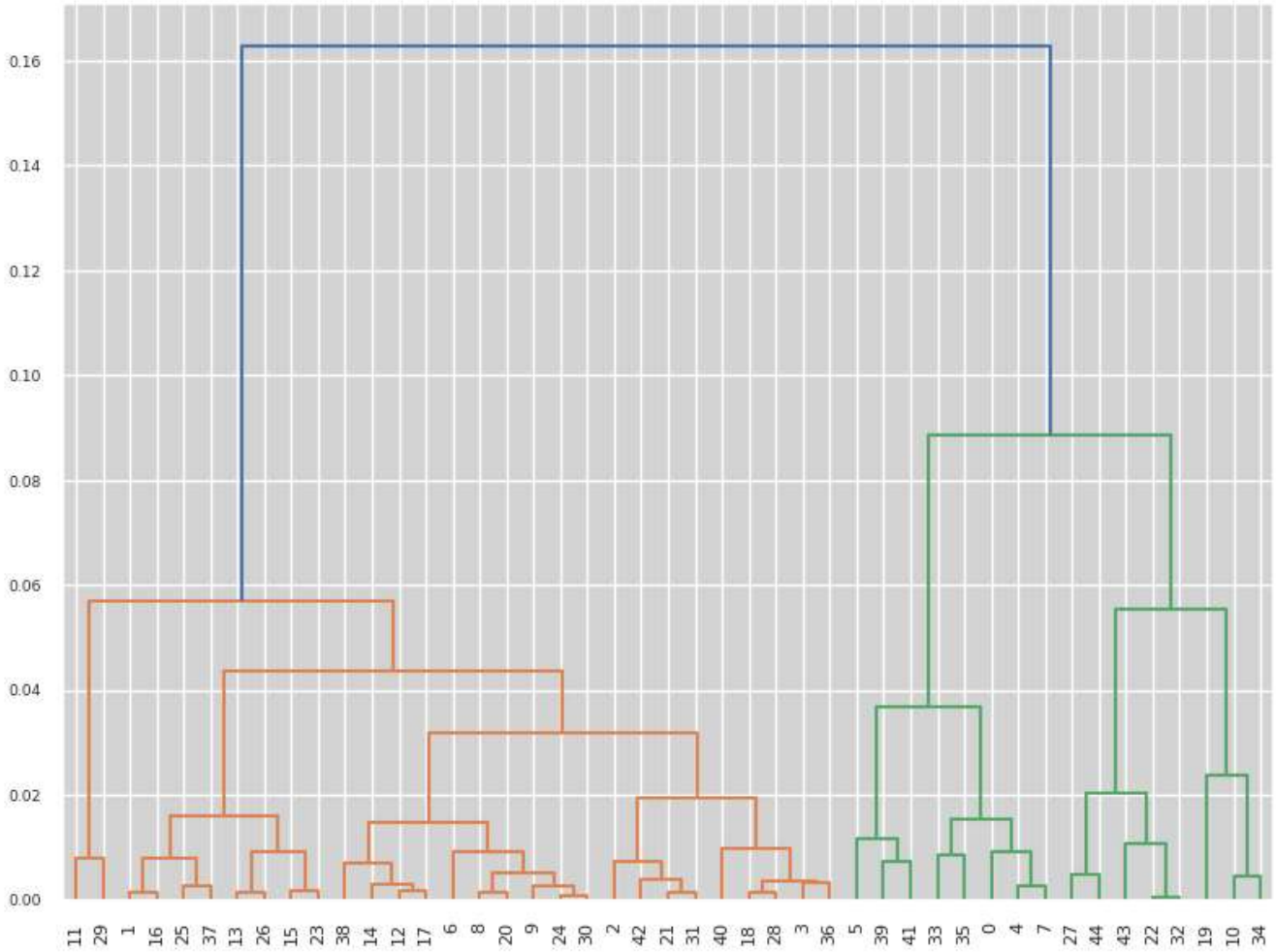

```
model = GaussianMixture(n_components=2,
                        random_state=1502).fit(elite_lle)


#Predict for each cluster
segments = pd.Series(model.predict(elite_lle))
elite_df['GMM Elite Clusters'] = segments



elite_df['GMM Elite Clusters'].value_counts()
```

```
0    31
1    14
Name: GMM Elite Clusters, dtype: int64
```

C. AGC

The Agglomerative Hierarchial cluster algorithm also only found two clusters.

```
plt.figure(figsize=(8,6))
linkage_data = linkage(elite_lle, method='ward', metric='euclidean')
dendrogram(linkage_data)
plt.tight_layout()
plt.show()
```




```
random.seed(42)
agc2 = AGC(n_clusters=2, linkage="ward")
agc2.fit(elite_lle)
```

▾ AgglomerativeClustering

AgglomerativeClustering()

```
elite_df['AGC Elite Clusters'] = agc2.labels_
elite_df['AGC Elite Clusters'].value_counts()
```

```
1    29
0    16
Name: AGC Elite Clusters, dtype: int64
```

D. Identifying the top 5% (top tier elite)

Using the previous and current MVPs as a test, the elite elite cluster is comprised of KMeans and GMM Elite Clusters equaling 0 and the AGC Elite Cluster equaling 1. Players who are in these three clusters are tagged as "top tier elite" while players not in these three clusters are tagged as "second tier elite".

```
elite_df[elite_df['FULL NAME'].str.contains('LeBron James|Stephen Curry|Giannis|Nikola Jokic|
```

	FULL NAME	KMeans Elite Clusters	GMM Elite Clusters	AGC Elite Clusters
1	Giannis Antetokounmpo	0	0	1
11	Stephen Curry	0	0	1
14	Kevin Durant	0	0	1
16	Joel Embiid	0	0	1
24	LeBron James	0	0	1
25	Nikola Jokic	0	0	1

```
def tier_elite_tag(df):
  if df['KMeans Elite Clusters'] == 0 and df['GMM Elite Clusters'] == 0 and df['AGC Elite Clu
    return "top tier elite"
  else:
    return "second tier elite"
```

```
elite_df["Elite Status"] = elite_df.apply(tier_elite_tag, axis=1)
```

All three algorithms show that there are 29 players in the elite of the elite segment.

```
elite_df["Elite Status"].value_counts()
```

```
top tier elite       29
second tier elite    16
Name: Elite Status, dtype: int64
```

The players names, their elite clusters and elite status will be merged with the main dataframe. Players without any elite clusters or status will be tagged as None.

```
merge_df = elite_df[['FULL NAME','KMeans Elite Clusters', 'GMM Elite Clusters', 'AGC Elite Cl

final_df = pd.merge(final_df, merge_df, how='left', on="FULL NAME")
final_df = final_df.fillna("None")

final_df[['FULL NAME','KMeans Elite Clusters', 'GMM Elite Clusters', 'AGC Elite Clusters', 'E
```

	FULL NAME	KMeans Elite Clusters	GMM Elite Clusters	AGC Elite Clusters	Elite Status
0	Precious Achiuwa	None	None	None	None
1	Steven Adams	None	None	None	None
2	Bam Adebayo	1.0	1.0	0.0	second tier elite
3	LaMarcus	N	N	N	N

Top Tier Elite

The 29 top tier elite and the 16 second tier elite have now been added to the main dataframe.

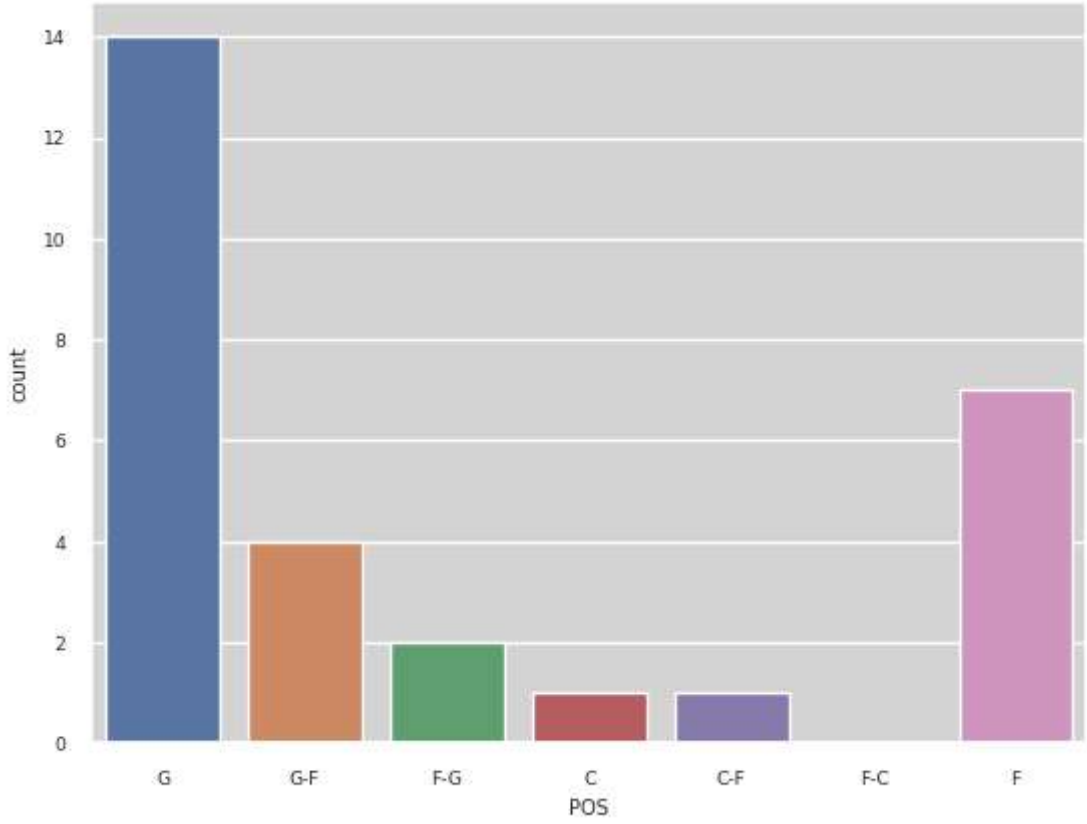
Show code

Characteristics of Top Tier Elite - Position

Of the 29 top tier elite players, 14 of them purely play the Guard position. There are no hybrid Forward-Centers, and there is only pure center, Nikola Jokic, and one hybrid Center-Forward, Joel Embiid.

Show code

```
<Axes: xlabel='POS', ylabel='count'>
```



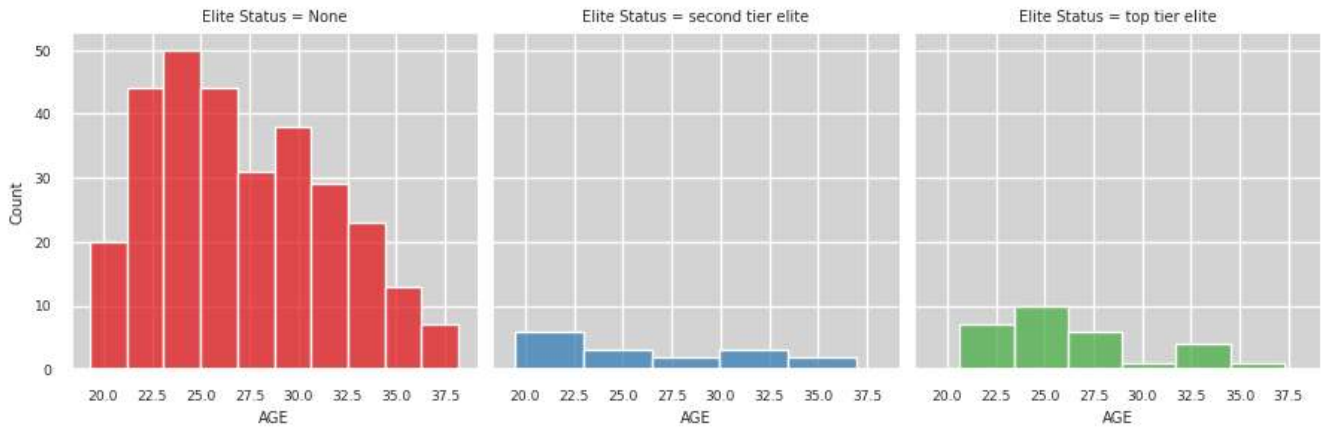
Show code

	FULL NAME	POS	Elite Status
16	Joel Embiid	C-F	top tier elite
25	Nikola Jokic	C	top tier elite

Characteristics of Top Tier Elite - Age

For the top tier elite, the age drop off is even more pronounced. There's a significant drop off after age 27.5 compared to the more gradual drop off of other players. The average age is 29 with only 9 players are aged 27.5 or higher.

Show code



Age Distribution of Top Tier Elite Players

Show code

	count	mean	std	min	25%	50%	75%	max
AGE	29.0	26.501724	4.270812	20.64	23.56	25.56	28.07	37.28

Age Distribution of Top Tier Elite Players beyond age 27.5

Show code

	count	mean	std	min	25%	50%	75%	max
AGE	9.0	31.667778	3.208464	28.03	28.12	32.57	33.53	37.28

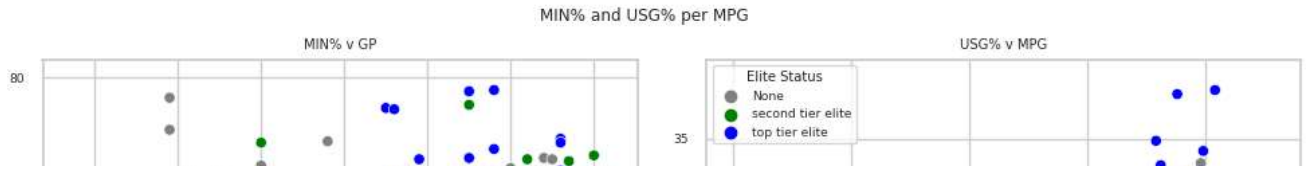
Characteristics of Top Tier Elite - MIN% and USG%

On average, top tier elite players have the highest percentage of their team's minutes and the highest percentage usage rate than any other players including second tier elite players. The scatter plot below which plots MIN% to USG% shows how heavily top tier elite players are used.

```
final_df.groupby('Elite Status')[['MIN%','USG%']].mean()
```

Elite Status	MIN%	USG%
None	49.940245	18.234894
second tier elite	70.231250	24.818750
top tier elite	71.889655	30.089655

```
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(10,6))
sns.set_style("whitegrid")
colors = ["grey", "green", "blue"]

fig.suptitle('MIN% and USG% per MPG')
axes[0].set_title("MIN% v GP")
sns.scatterplot(ax=axes[0], data=final_df, x='GP', y='MIN%', hue="Elite Status", palette=colo
axes[1].set_title("USG% v MPG")
sns.scatterplot(ax=axes[1], data=final_df, x='MPG', y='USG%', hue="Elite Status", palette=co
plt.tight_layout()
plt.show()
```



Characteristics of Top Tier Elite Players -- Take More Shots

Top tier elite players take more free throws, two and three point attempts than second tier and lower players. In the histograms below, for the top tier elite the distribution for the number of attempts is left skewed.

Additionally, looking at the distribution for percentages made for free throws, two and three point shots there is lower variability around the mean for the top tier players when compared to second tier and non-elite players. For example, for free throws made, top tier elite players have a 6% standard deviation while second tier elite players have 8% standard deviation. In other words, statistics for non top tier players are heavily influenced by outliers as also seen in the violin plots.

Averages for Free Throws, 2P, and 3P

Show code

Elite Status	FTA	FT%	2PA	2P%	3PA	3P%
None	105.144928	0.767800	268.707358	0.535128	190.74136	0.325491
second tier elite	260.875000	0.790031	589.750000	0.515563	312.21875	0.322094
top tier elite	383.206897	0.829862	794.862069	0.523069	414.62069	0.350862

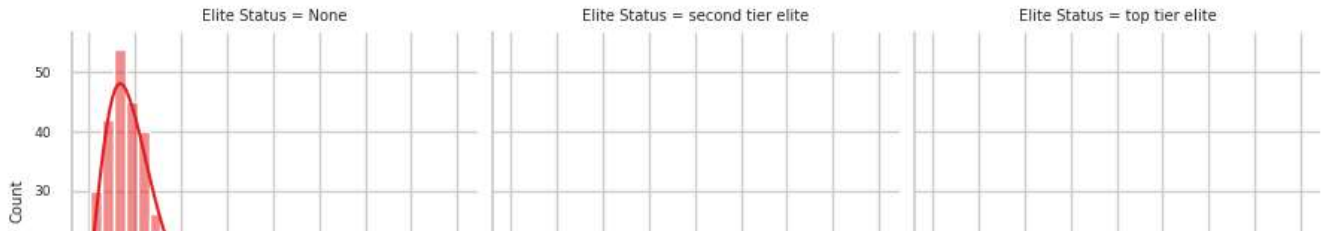
Standard Deviations for Free Throws, 2P, and 3P

```
#@title ######**Standard Deviations for Free Throws, 2P, and 3P**
final_df.groupby("Elite Status")[["FTA","2PA","3PA"]].std()
```

Elite Status	FTA	2PA	3PA
None	70.286266	165.696827	147.798229
second tier elite	117.907238	226.771397	136.785107
top tier elite	148.900345	206.992533	171.850311

Free Throws, 2P, and 3P Attempts

```
#@title #####**Free Throws, 2P, and 3P Attempts**
Attempts = final_df[["FULL NAME", "Elite Status", "FTA", "FT%","2PA", "2P%", "3PA", "3P%"]]
g = sns.FacetGrid(Attempts, col="Elite Status", hue='Elite Status', palette='Set1')
g.map(sns.histplot, "FTA", kde=True)
g = sns.FacetGrid(Attempts, col="Elite Status", hue='Elite Status', palette='Set1')
g.map(sns.histplot, "2PA", kde=True)
g = sns.FacetGrid(Attempts, col="Elite Status", hue='Elite Status', palette='Set1')
g.map(sns.histplot, "3PA", kde=True)
plt.show()
```


Elite Status = None
Elite Status = second tier elite
Elite Status = top tier elite
Count
50
40
30

Show code

Characteristics of Top Tier Elite Players -- More Points, Rebounds, Assists, and Steals

Across the production categories of Points, Assists, Rebounds, and Steals, the top tier elite well out-perform the non-elite players and only slightly out-perform the second tier elite.

Averages for Points, Assists, Rebounds, and Steals

```
#@title ######**Averages for Points, Assists, Rebounds, and Steals**
final_df.groupby("Elite Status")[["PPG","APG", "RPG", "SPG"]].mean()
```

Elite Status	PPG	APG	RPG	SPG
None	10.313434	2.165775	4.370847	0.751198
second tier elite	18.353125	6.171875	6.306250	1.135000
top tier elite	24.389655	6.044828	6.389655	1.175862

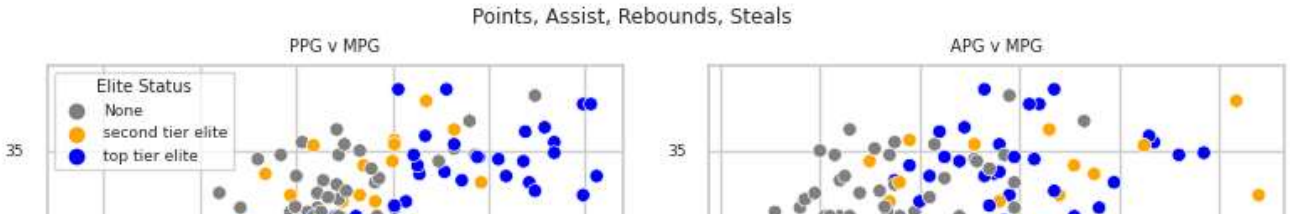
Standard Deviations for Points, Assists, Rebounds, and Steals

```
#@title ######**Standard Deviations for Points, Assists, Rebounds, and Steals**
final_df.groupby("Elite Status")[["PPG","APG", "RPG", "SPG"]].std()
```

Elite Status	PPG	APG	RPG	SPG
None	4.251604	1.330328	2.197167	0.323259
second tier elite	3.367663	2.362623	2.253433	0.366424
top tier elite	3.904516	1.630203	2.612107	0.337983

```
#@title
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(8,8))
```

```
    sns.set_style("whitegrid")
    colors = ["grey", "orange", "blue"]

    fig.suptitle('Points, Assist, Rebounds, Steals')
    axes[0,0].set_title("PPG v MPG")
    sns.scatterplot(ax=axes[0, 0], data=final_df, x='PPG', y='MPG', hue="Elite Status", palette=c
    axes[0,1].set_title("APG v MPG")
    sns.scatterplot(ax=axes[0, 1], data=final_df, x='APG', y='MPG', hue="Elite Status", palette=c
    axes[1,0].set_title("RPG v MPG")
    sns.scatterplot(ax=axes[1, 0], data=final_df, x='RPG', y='MPG', hue="Elite Status", palette=c
    axes[1,1].set_title("SPG v MPG")
    sns.scatterplot(ax=axes[1, 1], data=final_df, x='SPG', y='MPG', hue="Elite Status", palette=c
    plt.tight_layout()
    plt.show()
```



Characteristics of Top Tier Elite Players -- eFG%, TS%, Offensive and Defensive Rating

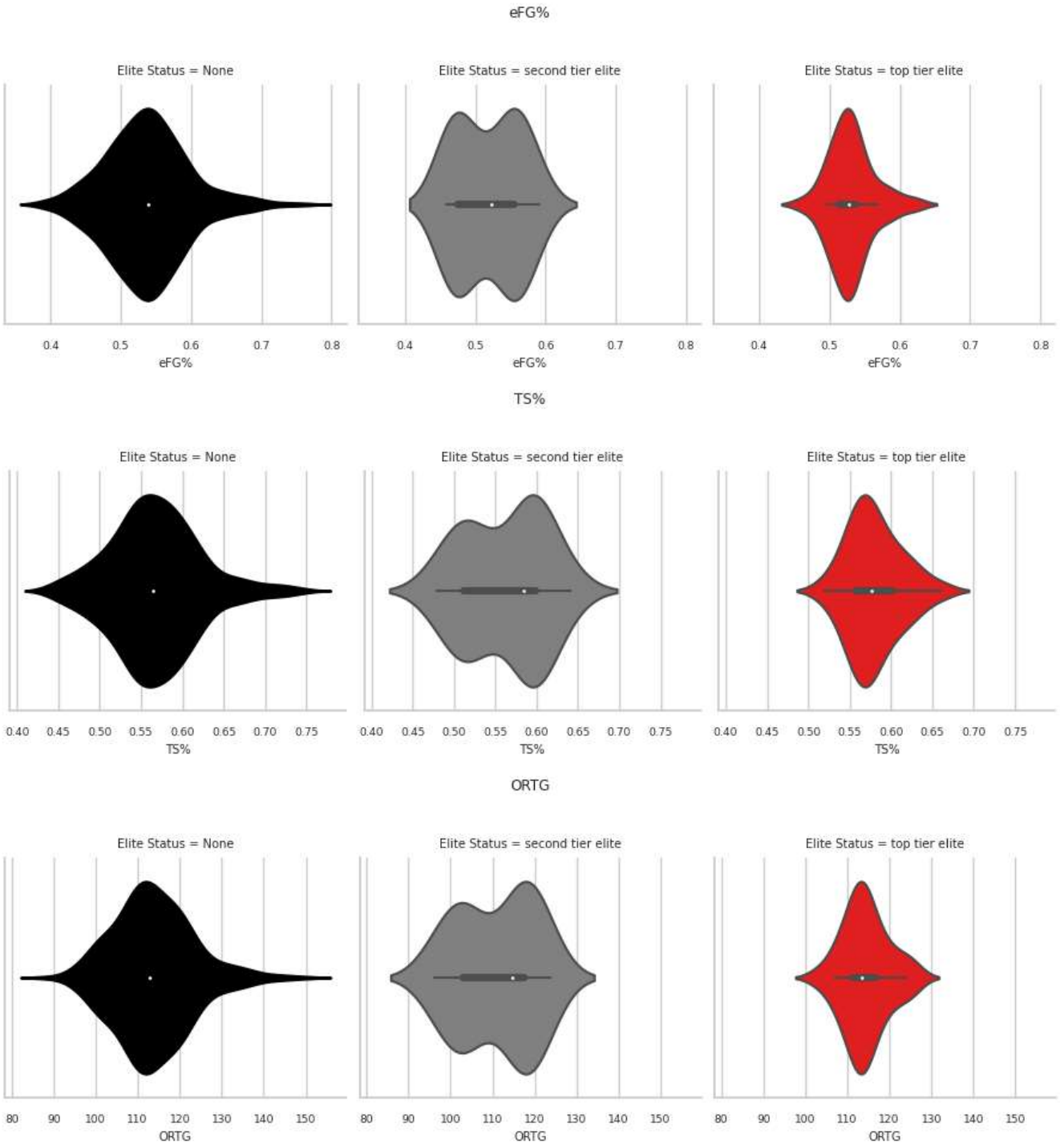
Show code

Elite Status	eFG%	TS%	ORTG	DRTG
None	0.537698	0.566973	113.702341	106.744259
second tier elite	0.518813	0.561906	110.937500	106.984375
top tier elite	0.530310	0.579414	114.272414	107.110345

Show code

Elite Status	eFG%	TS%	ORTG	DRTG
None	0.058879	0.053590	9.512182	4.714658
second tier elite	0.045708	0.048762	9.090150	4.225940
top tier elite	0.031622	0.032168	5.317887	3.670767

Show code



Characteristics of Top Tier Elite Players -- Versatility Index

As stated previously, the versatility index is a metric that measures a player's ability to produce in points, assists, and rebounds. The average player will score around a five on the index while top players score above 10.

The average for top tier elite players is higher for other players. The median is higher as well. However, the standard deviation is higher for the top tier elite. This is the result of outliers in the top tier elite cluster.

Versatility Index Mean

Show code

Elite Status	VI
None	7.134392
second tier elite	10.203125
top tier elite	11.103448

Versatility Index Median

Show code

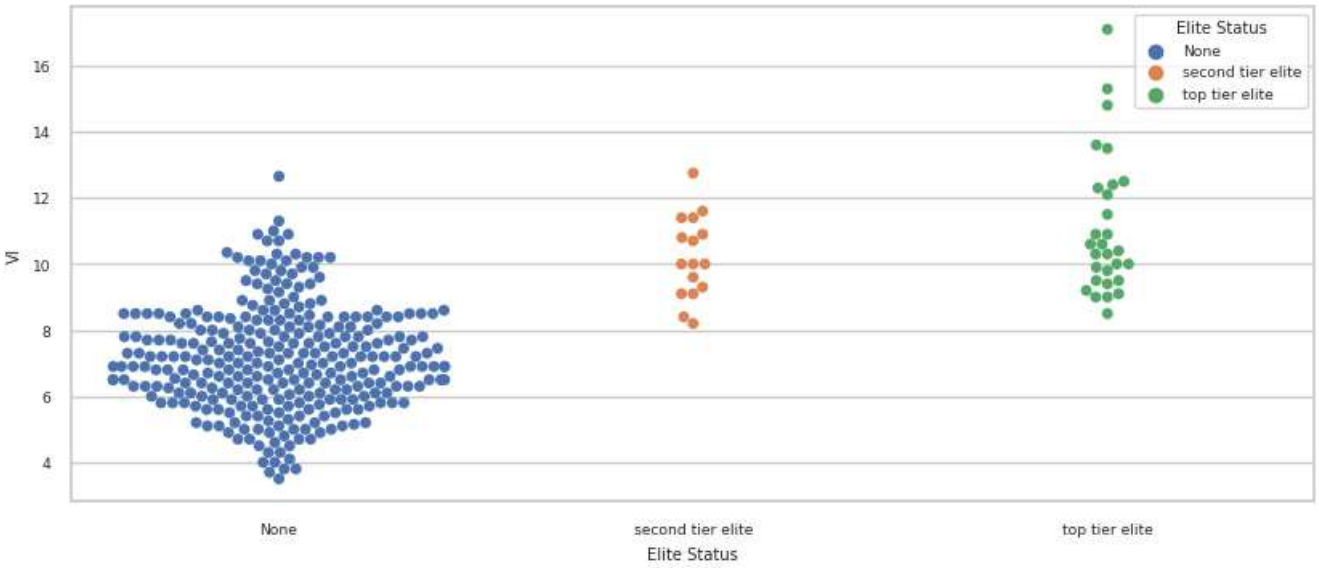
Elite Status	VI
None	7.0
second tier elite	10.0
top tier elite	10.4

Versatility Index Standard Deviation

Show code

Elite Status	VI
None	1.573485
second tier elite	1.248795
top tier elite	2.110261

```
#@title
df1 = final_df[["Elite Status","VI","POS"]]
plt.figure(figsize=(10,4))
ax = sns.swarmplot(x="Elite Status", y="VI", data=df1, hue="Elite Status")
```




```
final_df.to_excel("tblNBAPlayerSegmentAnalysis.xlsx")
```

VIII. Summary of Findings

The project was successful. Data was imported from a credible source, cleaned, and scaled. The 25 features dimensional space was reduced to two by using Manifold Learning, an approach to non-linear dimensionality reduction. Specifically the project used Locally Linear Embedding which seeks a lower-dimensional projection of the data which preserves distances within local neighborhoods.

Next three different categories of unsupervised machine learning algorithms:

- **Centroid based Clustering (KMeans)**
- **Distribution based clustering (GMM)**
- **Hierarchical Clustering ("AGC")**

were applied to the dataset twice. The first time was used to find the elite cluster, and the second time to find the elite within the elite for the top 5% of NBA players.

A. Final Comparison

Below is a table of the 29 players deemed as Top Tier Elite along with a field indicating where they were voted to an NBA All Pro team. Of the 29 players, 13 were selected to an All NBA team by NBA journalists.

The two exceptions were:

1. **Karl Anthony Town**s, who was tagged as second tier elite, was chosen to the All NBA team because their selections are by positions. Towns was one of only four Centers that the project deemed either first or second tier elite. Two centers, Jokic and Embiid made the first and second All NBA teams respectively.
2. The other excption was **Chris Paul**. This was clearly a sentimental, biased choice. There are 13 other Guards that are ranked higher than Paul, including Jaylen Brown, Donovan Mitchell, and Fred VanVleet.

The project's results were inline with the consensus opinion of NBA journalists with only one clear disagreement.

Show code

	FULL NAME	POS	NBA_ALL_Pro	LLE_Component 1	LLE_Component 2	KMeans Clusters	GMM Clusters	C
0	Luka Doncic	F-G	1st	0.117749	0.018927	2	2	
1	Nikola Jokic	C	1st	0.118687	0.012801	2	2	
2	Giannis Antetokounmpo	F	1st	0.124541	0.013214	2	2	
3	Jayson Tatum	F-G	1st	0.132305	0.017730	2	2	
4	Devin Booker	G	1st	0.139815	0.025503	2	2	

References

- Richard Bellman, Wikipedia Article, https://en.wikipedia.org/wiki/Richard_E._Bellman, Last Updated April 9, 2023
- Manifold Learning t-SNE, LLE, Isomap Made Easy, by Andre Ye (08/12/2020) Towards Data Science
- LLE: Locally Linear Embedding — A Nifty Way to Reduce Dimensionality in Python by Saul Dobilas (10/10/2021) TowardsDataScience.com
- A One-Stop Shop for Principal Component Analysis by Matt Brems (April 17, 2017)