Task-2(a)

In [ ]:
```
!pip install seaborn
!pip install matplotlib
!pip install scikit-learn
!pip install tensorflow
!pip install tensorflow keras
```

```
Requirement already satisfied: seaborn in c:\users\jaskaran\appdata\local\programs\python\python310\lib\site-pac
kages (0.12.2)
Requirement already satisfied: numpy!=1.24.0,>=1.17 in c:\users\jaskaran\appdata\local\programs\python\python310
\lib\site-packages (from seaborn) (1.24.3)
Requirement already satisfied: pandas>=0.25 in c:\users\jaskaran\appdata\local\programs\python\python310\lib\sit
e-packages (from seaborn) (2.0.3)
Requirement already satisfied: matplotlib!=3.6.1,>=3.1 in c:\users\jaskaran\appdata\local\programs\python\python
310\lib\site-packages (from seaborn) (3.7.2)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\jaskaran\appdata\local\programs\python\python310\lib
\site-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (1.1.0)
Requirement already satisfied: cycler>=0.10 in c:\users\jaskaran\appdata\local\programs\python\python310\lib\sit
e-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\jaskaran\appdata\local\programs\python\python310\li
b\site-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (4.41.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\jaskaran\appdata\local\programs\python\python310\li
b\site-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (1.4.4)
Requirement already satisfied: packaging>=20.0 in c:\users\jaskaran\appdata\local\programs\python\python310\lib\
site-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (23.1)
Requirement already satisfied: pillow>=6.2.0 in c:\users\jaskaran\appdata\local\programs\python\python310\lib\si
te-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (10.0.0)
Requirement already satisfied: pyparsing<3.1,>=2.3.1 in c:\users\jaskaran\appdata\local\programs\python\python31
0\lib\site-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\jaskaran\appdata\local\programs\python\python310
\lib\site-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\users\jaskaran\appdata\local\programs\python\python310\lib\sit
e-packages (from pandas>=0.25->seaborn) (2023.3)
Requirement already satisfied: tzdata>=2022.1 in c:\users\jaskaran\appdata\local\programs\python\python310\lib\s
ite-packages (from pandas>=0.25->seaborn) (2023.3)
Requirement already satisfied: six>=1.5 in c:\users\jaskaran\appdata\local\programs\python\python310\lib\site-pa
ckages (from python-dateutil>=2.7->matplotlib!=3.6.1,>=3.1->seaborn) (1.16.0)
[notice] A new release of pip is available: 23.2 -> 23.2.1
[notice] To update, run: python.exe -m pip install --upgrade pip
Requirement already satisfied: matplotlib in c:\users\jaskaran\appdata\local\programs\python\python310\lib\site-
packages (3.7.2)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\jaskaran\appdata\local\programs\python\python310\lib
\site-packages (from matplotlib) (1.1.0)
Requirement already satisfied: cycler>=0.10 in c:\users\jaskaran\appdata\local\programs\python\python310\lib\sit
e-packages (from matplotlib) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\jaskaran\appdata\local\programs\python\python310\li
b\site-packages (from matplotlib) (4.41.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\jaskaran\appdata\local\programs\python\python310\li
b\site-packages (from matplotlib) (1.4.4)
Requirement already satisfied: numpy>=1.20 in c:\users\jaskaran\appdata\local\programs\python\python310\lib\site
-packages (from matplotlib) (1.24.3)
Requirement already satisfied: packaging>=20.0 in c:\users\jaskaran\appdata\local\programs\python\python310\lib\
site-packages (from matplotlib) (23.1)
Requirement already satisfied: pillow>=6.2.0 in c:\users\jaskaran\appdata\local\programs\python\python310\lib\si
te-packages (from matplotlib) (10.0.0)
Requirement already satisfied: pyparsing<3.1,>=2.3.1 in c:\users\jaskaran\appdata\local\programs\python\python31
0\lib\site-packages (from matplotlib) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\jaskaran\appdata\local\programs\python\python310
\lib\site-packages (from matplotlib) (2.8.2)
Requirement already satisfied: six>=1.5 in c:\users\jaskaran\appdata\local\programs\python\python310\lib\site-pa
ckages (from python-dateutil>=2.7->matplotlib) (1.16.0)
[notice] A new release of pip is available: 23.2 -> 23.2.1
[notice] To update, run: python.exe -m pip install --upgrade pip
Requirement already satisfied: scikit-learn in c:\users\jaskaran\appdata\local\programs\python\python310\lib\sit
e-packages (1.3.0)
Requirement already satisfied: numpy>=1.17.3 in c:\users\jaskaran\appdata\local\programs\python\python310\lib\si
te-packages (from scikit-learn) (1.24.3)
Requirement already satisfied: scipy>=1.5.0 in c:\users\jaskaran\appdata\local\programs\python\python310\lib\sit
e-packages (from scikit-learn) (1.11.1)
Requirement already satisfied: joblib>=1.1.1 in c:\users\jaskaran\appdata\local\programs\python\python310\lib\si
te-packages (from scikit-learn) (1.3.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\jaskaran\appdata\local\programs\python\python310
\lib\site-packages (from scikit-learn) (3.2.0)
[notice] A new release of pip is available: 23.2 -> 23.2.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

In [1]:
```
import pandas as pd
```

```
# Replace 'file_path.csv' with the actual file path of your CSV file
file_path = "C:/Users/Jaskaran/Downloads/2023_nba_player_stats.csv"

# Use the pd.read_csv() function to read the CSV file into a DataFrame
df_JS = pd.read_csv(file_path)
```

Displaying the first few rows of the dataset

In [2]:
```
print("Preview of the dataset:")
print(df_JS.head())
```

```
Preview of the dataset:
                   Player POS Team   Age    GP     W     L     Min     PTS  \
0             Jayson Tatum  SF  BOS  25.0  74.0  52.0  22.0  2732.2  2225.0
1              Joel Embiid   C  PHI  29.0  66.0  43.0  23.0  2284.1  2183.0
2              Luka Doncic  PG  DAL  24.0  66.0  33.0  33.0  2390.5  2138.0
3  Shai Gilgeous-Alexander  PG  OKC  24.0  68.0  33.0  35.0  2416.0  2135.0
4    Giannis Antetokounmpo  PF  MIL  28.0  63.0  47.0  16.0  2023.6  1959.0

     FGM  ...    REB    AST    TOV    STL    BLK     PF      FP   DD2   TD3  \
0  727.0  ...  649.0  342.0  213.0   78.0   51.0  160.0  3691.0  31.0   1.0
1  728.0  ...  670.0  274.0  226.0   66.0  112.0  205.0  3706.0  39.0   1.0
2  719.0  ...  569.0  529.0  236.0   90.0   33.0  166.0  3747.0  36.0  10.0
3  704.0  ...  329.0  371.0  192.0  112.0   65.0  192.0  3425.0   3.0   0.0
4  707.0  ...  742.0  359.0  246.0   52.0   51.0  197.0  3451.0  46.0   6.0

     +/-
0  470.0
1  424.0
2  128.0
3  149.0
4  341.0

[5 rows x 30 columns]
```

Understanding the basic information about the dataset

In [3]:
```
print("\nDataset information:")
print(df_JS.info())
```

```
Dataset information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 539 entries, 0 to 538
Data columns (total 30 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Player  539 non-null    object
 1   POS     534 non-null    object
 2   Team    539 non-null    object
 3   Age     539 non-null    float64
 4   GP      539 non-null    float64
 5   W       539 non-null    float64
 6   L       539 non-null    float64
 7   Min     539 non-null    float64
 8   PTS     539 non-null    float64
 9   FGM     539 non-null    float64
 10  FGA     539 non-null    float64
 11  FG%     539 non-null    float64
 12  3PM     539 non-null    float64
 13  3PA     539 non-null    float64
 14  3P%     539 non-null    float64
 15  FTM     539 non-null    float64
 16  FTA     539 non-null    float64
 17  FT%     539 non-null    float64
 18  OREB    539 non-null    float64
 19  DREB    539 non-null    float64
 20  REB     539 non-null    float64
 21  AST     539 non-null    float64
 22  TOV     539 non-null    float64
 23  STL     539 non-null    float64
 24  BLK     539 non-null    float64
 25  PF      539 non-null    float64
 26  FP      539 non-null    float64
 27  DD2     539 non-null    float64
 28  TD3     539 non-null    float64
 29  +/-     539 non-null    float64
dtypes: float64(27), object(3)
memory usage: 126.5+ KB
None
```

Printing summary statistics of the numerical columns

In [4]:
```
print("\nSummary Statistics:")
print(df_JS.describe())
```

```
Summary Statistics:
              Age          GP           W           L          Min  \
count  539.000000  539.000000  539.000000  539.000000   539.000000
mean    25.970315   48.040816   24.018553   24.022263  1103.617625
std      4.315513   24.650686   14.496366   13.445866   827.765114
min     19.000000    1.000000    0.000000    0.000000     1.000000
25%     23.000000   30.500000   12.000000   14.000000   329.000000
50%     25.000000   54.000000   25.000000   25.000000   970.200000
75%     29.000000   68.000000   36.000000   34.000000  1845.900000
max     42.000000   83.000000   57.000000   60.000000  2963.200000

               PTS         FGM          FGA         FG%         3PM  ...  \
count   539.000000  539.000000   539.000000  539.000000  539.000000  ...
mean    523.426716  191.576994   403.005566   46.325232   56.324675  ...
std     498.084360  178.351286   369.595909   10.967271   60.916821  ...
min       0.000000    0.000000     0.000000    0.000000    0.000000  ...
25%     120.500000   45.500000    93.500000   41.650000    5.000000  ...
50%     374.000000  138.000000   300.000000   45.500000   36.000000  ...
75%     769.500000  283.500000   598.500000   50.600000   92.000000  ...
max    2225.000000  728.000000  1559.000000  100.000000  301.000000  ...

              REB         AST         TOV         STL         BLK          PF  \
count  539.000000  539.000000  539.000000  539.000000  539.000000  539.000000
mean   198.254174  115.545455   61.300557   33.270872   21.241187   91.181818
std    181.819962  129.578453   58.279185   28.336745   26.529238   66.206731
min      0.000000    0.000000    0.000000    0.000000    0.000000    0.000000
25%     50.500000   22.000000   14.500000    8.500000    5.000000   32.000000
50%    159.000000   69.000000   44.000000   28.000000   13.000000   86.000000
75%    286.000000  162.500000   92.500000   51.000000   28.000000  140.000000
max    973.000000  741.000000  300.000000  128.000000  193.000000  279.000000

               FP         DD2         TD3         +/-
count   539.000000  539.000000  539.000000  539.000000
mean   1036.938776    4.011132    0.220779    0.000000
std     894.081896    8.770932    1.564432  148.223909
min      -1.000000    0.000000    0.000000 -642.000000
25%     254.000000    0.000000    0.000000  -70.000000
50%     810.000000    0.000000    0.000000   -7.000000
75%    1646.000000    3.000000    0.000000   57.000000
max    3842.000000   65.000000   29.000000  640.000000

[8 rows x 27 columns]
```

Check for missing values in the entire data frame

True indicating missing values and False indicates non missing values

In [5]:
```
missing_values = df_JS.isnull()
print(missing_values)
```

```
     Player    POS   Team    Age     GP      W      L    Min    PTS    FGM  \
0     False  False  False  False  False  False  False  False  False  False
1     False  False  False  False  False  False  False  False  False  False
2     False  False  False  False  False  False  False  False  False  False
3     False  False  False  False  False  False  False  False  False  False
4     False  False  False  False  False  False  False  False  False  False
..      ...    ...    ...    ...    ...    ...    ...    ...    ...    ...
534   False   True  False  False  False  False  False  False  False  False
535   False   True  False  False  False  False  False  False  False  False
536   False   True  False  False  False  False  False  False  False  False
537   False   True  False  False  False  False  False  False  False  False
538   False   True  False  False  False  False  False  False  False  False

     ...    REB    AST    TOV    STL    BLK     PF     FP    DD2    TD3    +/-
0    ...  False  False  False  False  False  False  False  False  False  False
1    ...  False  False  False  False  False  False  False  False  False  False
2    ...  False  False  False  False  False  False  False  False  False  False
3    ...  False  False  False  False  False  False  False  False  False  False
4    ...  False  False  False  False  False  False  False  False  False  False
..   ...    ...    ...    ...    ...    ...    ...    ...    ...    ...    ...
534  ...  False  False  False  False  False  False  False  False  False  False
535  ...  False  False  False  False  False  False  False  False  False  False
536  ...  False  False  False  False  False  False  False  False  False  False
537  ...  False  False  False  False  False  False  False  False  False  False
538  ...  False  False  False  False  False  False  False  False  False  False

[539 rows x 30 columns]
```

Get the count of missing values in each column

In [6]:
```
missing_count = df_JS.isnull().sum()
print(missing_count)
```

```
Player    0
POS       5
Team      0
Age       0
GP        0
W         0
L         0
Min       0
PTS       0
FGM       0
FGA       0
FG%       0
3PM       0
3PA       0
3P%       0
FTM       0
FTA       0
FT%       0
OREB      0
DREB      0
REB       0
AST       0
TOV       0
STL       0
BLK       0
PF        0
FP        0
DD2       0
TD3       0
+/-       0
dtype: int64
```

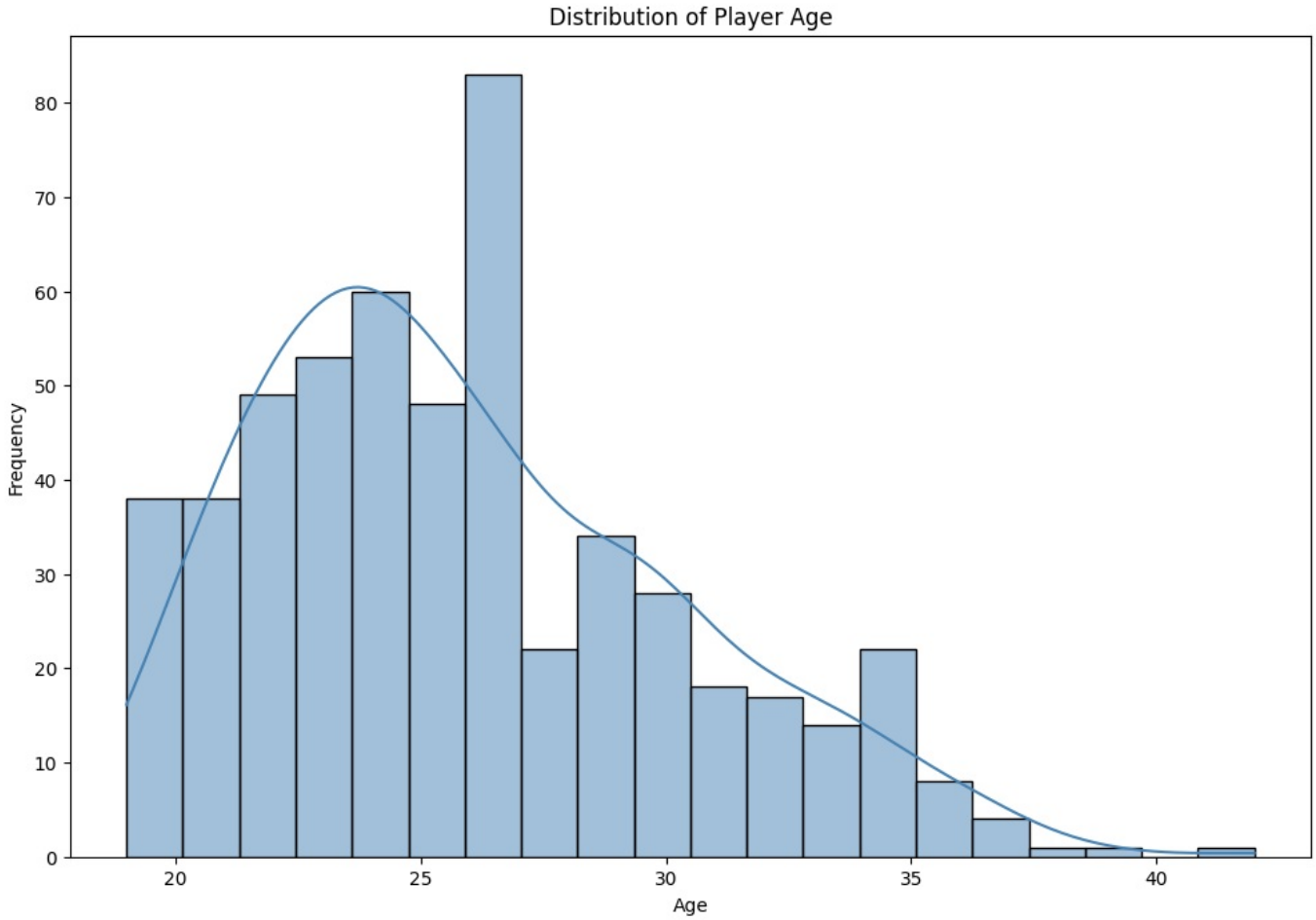
Printing the data types for all the columns

In [7]:
```
data_types = df_JS.dtypes
print(data_types)
```

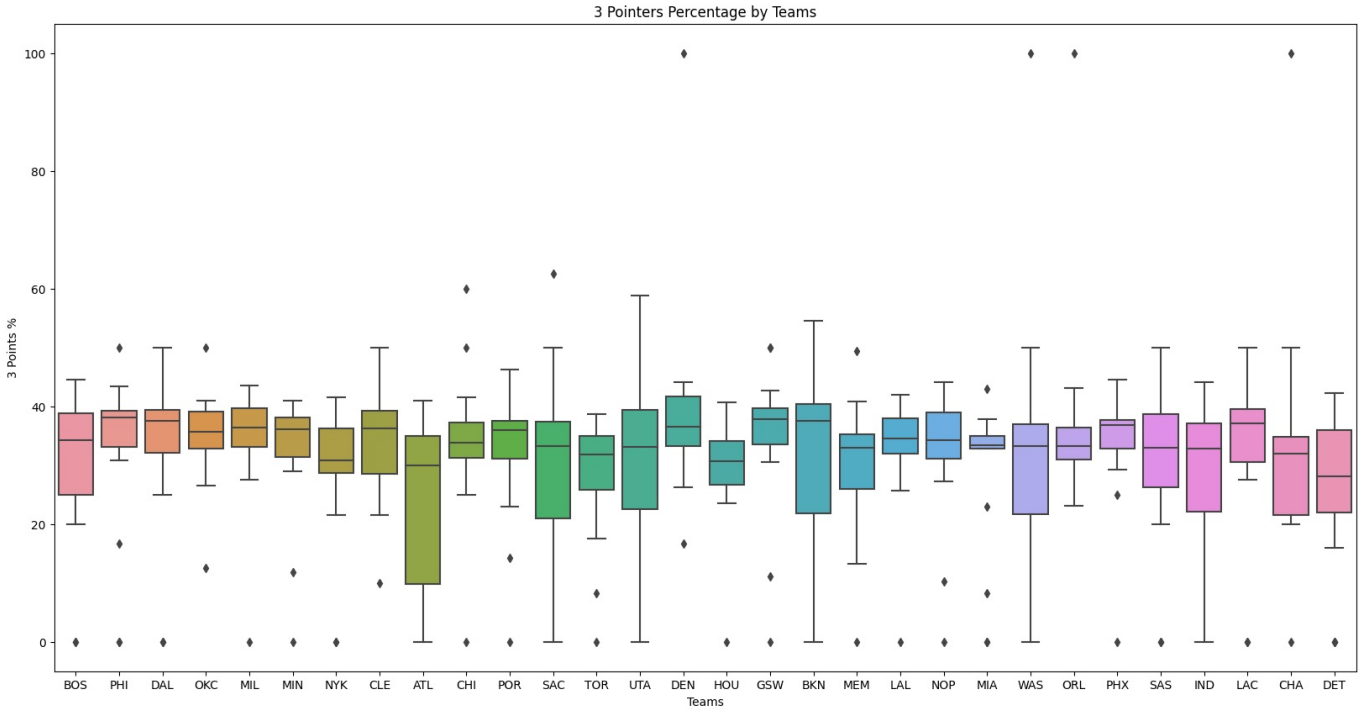
```
Player     object
POS        object
Team       object
Age       float64
GP        float64
W         float64
L         float64
Min       float64
PTS       float64
FGM       float64
FGA       float64
FG%       float64
3PM       float64
3PA       float64
3P%       float64
FTM       float64
FTA       float64
FT%       float64
OREB      float64
DREB      float64
REB       float64
AST       float64
TOV       float64
STL       float64
BLK       float64
PF        float64
FP        float64
DD2       float64
TD3       float64
+/-       float64
dtype: object
```

Visualizing the data distribuition using histogram, boxplot and scatter plot

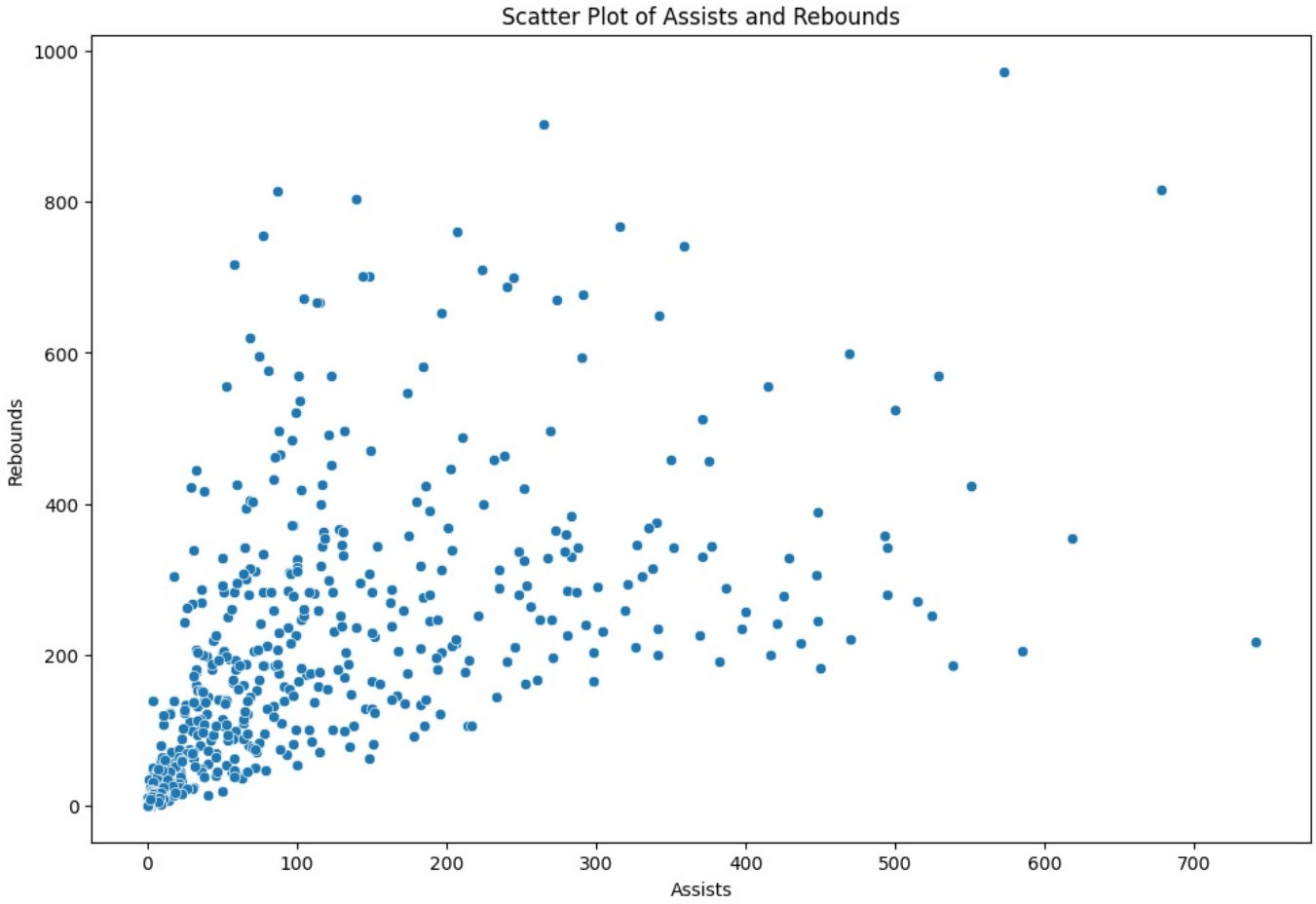
In [20]:
```
import matplotlib.pyplot as plt
import seaborn as sns


plt.figure(figsize=(12, 8))
sns.histplot(df_JS['Age'], bins=20, kde=True, color='steelblue')
plt.title("Distribution of Player Age")
plt.xlabel("Age")
plt.ylabel("Frequency")
plt.show()
```



In [22]:
```
plt.figure(figsize=(20, 10))
sns.boxplot(x='Team', y='3P%', data=df_JS)
plt.title("3 Pointers Percentage by Teams")
plt.xlabel("Teams")
plt.ylabel("3 Points %")
plt.show()
```



In [24]:
```
plt.figure(figsize=(12, 8))
sns.scatterplot(x='AST', y='REB', data=df_JS)
plt.title("Scatter Plot of Assists and Rebounds")
plt.xlabel("Assists")
plt.ylabel("Rebounds")
plt.show()
```



Task 3(a)

In [39]:

```python
import pandas as pd

# Load the data from the CSV file with the correct encoding
file_path = "C:/Users/Jaskaran/Downloads/2023_nba_player_stats.csv"
df_JS = pd.read_csv(file_path)

# Remove unnecessary columns
columns_to_drop = ['PF', 'FP', 'DD2', 'TD3', '+/-']
df_JS = df_JS.drop(columns=columns_to_drop)

# Clean player names by removing special characters
df_JS['Player'] = df_JS['Player'].str.replace('[^a-zA-Z\s]', '')

# Handle missing values by filling with appropriate values
df_JS.replace('N/A', '0', inplace=True)

# Convert percentage columns to float
try:
    df_JS['3P%'] = df_JS['3P%'].str.rstrip('%').astype(float)
    df_JS['FG%'] = df_JS['FG%'].str.rstrip('%').astype(float)
    df_JS['FT%'] = df_JS['FT%'].str.rstrip('%').astype(float)
except AttributeError:
    pass  # Column is already of type float or contains non-string values



# Define the path to save the preprocessed data
preprocessed_file_path = r"C:/Users/Jaskaran/Desktop/Preprocessed_NBA_Data.csv"

# Save the preprocessed data to a CSV file
df_JS.to_csv(preprocessed_file_path, index=False)
df_JS=pd.read_csv(r"C:/Users/Jaskaran/Desktop/Preprocessed_NBA_Data.csv")

print("Data preprocessing completed and saved to:", preprocessed_file_path)
print(df_JS.head())
```

```
Data preprocessing completed and saved to: C:/Users/Jaskaran/Desktop/Preprocessed_NBA_Data.csv
                   Player POS Team   Age    GP     W     L     Min     PTS  \
0            Jayson Tatum  SF  BOS  25.0  74.0  52.0  22.0  2732.2  2225.0
1             Joel Embiid   C  PHI  29.0  66.0  43.0  23.0  2284.1  2183.0
2             Luka Doncic  PG  DAL  24.0  66.0  33.0  33.0  2390.5  2138.0
3  Shai Gilgeous-Alexander  PG  OKC  24.0  68.0  33.0  35.0  2416.0  2135.0
4    Giannis Antetokounmpo  PF  MIL  28.0  63.0  47.0  16.0  2023.6  1959.0

     FGM  ...    FTM    FTA   FT%   OREB   DREB    REB    AST    TOV    STL  \
0  727.0  ...  531.0  622.0  85.4   78.0  571.0  649.0  342.0  213.0   78.0
1  728.0  ...  661.0  771.0  85.7  113.0  557.0  670.0  274.0  226.0   66.0
2  719.0  ...  515.0  694.0  74.2   54.0  515.0  569.0  529.0  236.0   90.0
3  704.0  ...  669.0  739.0  90.5   59.0  270.0  329.0  371.0  192.0  112.0
4  707.0  ...  498.0  772.0  64.5  137.0  605.0  742.0  359.0  246.0   52.0

     BLK
0   51.0
1  112.0
2   33.0
3   65.0
4   51.0

[5 rows x 25 columns]
```

Task3(b)

In [46]:

```python
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler

# Load the preprocessed data from the CSV file
preprocessed_file_path = "C:/Users/Jaskaran/Desktop/Preprocessed_NBA_Data.csv"
df_JS = pd.read_csv(preprocessed_file_path)

print(df_JS)

# Handling Missing Values
# missing values in POS column have already been handled in data preprocessing step.

# Identifying Outliers
# Let's identify outliers using the Interquartile Range (IQR) method.
# You can adjust the IQR multiplier based on your data characteristics.

def identify_outliers(column):
    Q1 = np.percentile(column, 25)
    Q3 = np.percentile(column, 75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    return (column < lower_bound) | (column > upper_bound)

outliers = identify_outliers(df_JS['PTS'])
df_JS = df_JS[~outliers]

# Feature Engineering
# Let's create a new feature 'Points Per Game' (PPG Ratio) which is PTS / GP.
df_JS['PPG RATIO'] = data['PTS'] / data['GP']
print(df_JS.head())
# Standardize Numeric Features
numeric_columns = ['Age', 'GP', 'W', 'L', 'Min', 'PTS', 'FGM', 'FGA', 'FG%', '3PM', '3PA',
                   '3P%', 'FTM', 'FTA', 'FT%', 'OREB', 'DREB', 'REB', 'AST', 'TOV', 'STL',
                   'BLK']

scaler = StandardScaler()
df_JS[numeric_columns] = scaler.fit_transform(df_JS[numeric_columns])

# Save the preprocessed and engineered data to a new CSV file
final_data_file_path = "C:/Users/Jaskaran/Desktop/Final_NBA_Data.csv"
df_JS.to_csv(final_data_file_path, index=False)

print("Data preprocessing, outlier removal, feature engineering, and scaling completed.")
print("Final data saved to:", final_data_file_path)
```

```
                    Player  POS Team   Age    GP     W     L     Min  \
0              Jayson Tatum   SF  BOS  25.0  74.0  52.0  22.0  2732.2
1               Joel Embiid    C  PHI  29.0  66.0  43.0  23.0  2284.1
2               Luka Doncic   PG  DAL  24.0  66.0  33.0  33.0  2390.5
3    Shai Gilgeous-Alexander   PG  OKC  24.0  68.0  33.0  35.0  2416.0
4      Giannis Antetokounmpo   PF  MIL  28.0  63.0  47.0  16.0  2023.6
..                       ...  ...  ...   ...   ...   ...   ...     ...
534         Alondes Williams  NaN  BKN  23.0   1.0   1.0   0.0     5.3
535            Deonte Burton  NaN  SAC  29.0   2.0   1.0   1.0     6.5
536            Frank Jackson  NaN  UTA  24.0   1.0   0.0   1.0     5.0
537       Michael Foster Jr.  NaN  PHI  20.0   1.0   1.0   0.0     1.0
538            Sterling Brown  NaN  LAL  28.0   4.0   2.0   2.0    24.4

        PTS    FGM  ...    FTM    FTA   FT%   OREB   DREB    REB    AST  \
0    2225.0  727.0  ...  531.0  622.0  85.4   78.0  571.0  649.0  342.0
1    2183.0  728.0  ...  661.0  771.0  85.7  113.0  557.0  670.0  274.0
2    2138.0  719.0  ...  515.0  694.0  74.2   54.0  515.0  569.0  529.0
3    2135.0  704.0  ...  669.0  739.0  90.5   59.0  270.0  329.0  371.0
4    1959.0  707.0  ...  498.0  772.0  64.5  137.0  605.0  742.0  359.0
..      ...    ...  ...    ...    ...   ...    ...    ...    ...    ...
534     0.0    0.0  ...    0.0    0.0   0.0    0.0    1.0    1.0    0.0
535     0.0    0.0  ...    0.0    0.0   0.0    0.0    0.0    0.0    0.0
536     0.0    0.0  ...    0.0    0.0   0.0    1.0    1.0    2.0    1.0
537     0.0    0.0  ...    0.0    0.0   0.0    0.0    0.0    0.0    0.0
538     0.0    0.0  ...    0.0    0.0   0.0    3.0    5.0    8.0    2.0

       TOV    STL    BLK
0    213.0   78.0   51.0
1    226.0   66.0  112.0
2    236.0   90.0   33.0
3    192.0  112.0   65.0
4    246.0   52.0   51.0
..     ...    ...    ...
534    2.0    0.0    0.0
535    0.0    0.0    0.0
536    0.0    0.0    0.0
537    0.0    0.0    0.0
538    0.0    3.0    0.0

[539 rows x 25 columns]
             Player POS Team   Age    GP     W     L     Min     PTS    FGM  \
14    Pascal Siakam  PF  TOR  29.0  71.0  35.0  36.0  2652.0  1720.0  630.0
15  Lauri Markkanen  PF  UTA  25.0  66.0  32.0  34.0  2272.5  1691.0  571.0
16     Nikola Jokic   C  DEN  28.0  69.0  48.0  21.0  2323.0  1690.0  646.0
17      Jalen Green  SG  HOU  21.0  76.0  20.0  56.0  2602.2  1683.0  566.0
18     Jordan Poole  SG  GSW  23.0  82.0  44.0  38.0  2458.1  1675.0  550.0

    ...    FTA   FT%   OREB   DREB    REB    AST    TOV   STL   BLK  PPG RATIO
14  ...  474.0  77.4  131.0  425.0  556.0  415.0  169.0  65.0  36.0  24.225352
15  ...  399.0  87.5  130.0  440.0  570.0  123.0  127.0  42.0  38.0  25.621212
16  ...  415.0  82.2  167.0  650.0  817.0  678.0  247.0  87.0  47.0  24.492754
17  ...  463.0  78.6   43.0  241.0  284.0  281.0  200.0  59.0  18.0  22.144737
18  ...  415.0  87.0   32.0  193.0  225.0  369.0  252.0  63.0  21.0  20.426829

[5 rows x 26 columns]
Data preprocessing, outlier removal, feature engineering, and scaling completed.
Final data saved to: C:/Users/Jaskaran/Desktop/Final_NBA_Data.csv
```

In [50]:

```python
#4.1

#Machine learning Model -1 Using Random forest regressor


import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Load the dataset (replace 'data.csv' with the actual file path of your dataset)
data = pd.read_csv('C:/Users/Jaskaran/Desktop/Final_NBA_Data.csv')

# Preprocessing: Separate features (X) and target variable (y)
X = data.drop(columns=['Player', 'POS', 'Team', 'PPG RATIO'])
y = data['PPG RATIO']

# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Model: Random Forest Regressor
rf_model = RandomForestRegressor(random_state=42)
rf_model.fit(X_train, y_train)
rf_predictions = rf_model.predict(X_test)
```

```
# Evaluate the model using mean squared error, mean absolute error, and R-squared (coefficient of determination)
mse = mean_squared_error(y_test, rf_predictions)
mae = mean_absolute_error(y_test, rf_predictions)
r2 = r2_score(y_test, rf_predictions)

print("Random Forest Mean Squared Error (MSE):", mse)
print("Random Forest Mean Absolute Error (MAE):", mae)
print("Random Forest R-squared (R2):", r2)
```

```
Random Forest Mean Squared Error (MSE): 4.680750516085453
Random Forest Mean Absolute Error (MAE): 1.1951260825814995
Random Forest R-squared (R2): 0.9047317542362294
```

Performance of the Random Forest Regressor on this dataset.

The lower the MSE and MAE, and the higher the R2, the better the model's performance at predicting the target variable (PPG RATIO)

In [1]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow import keras
from tensorflow.keras import layers

# Load the dataset (replace 'data.csv' with the actual file path of your dataset)
data = pd.read_csv('C:/Users/Jaskaran/Desktop/Final_NBA_Data.csv')

# Preprocessing: Separate features (X) and target variable (y)
X = data.drop(columns=['Player', 'POS', 'Team', 'PPG RATIO'])
y = data['PPG RATIO']

# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Build the neural network model
model = keras.Sequential([
    layers.Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    layers.Dense(32, activation='relu'),
    layers.Dense(1)  # Output layer for regression
])

# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')

# Train the model
history = model.fit(X_train, y_train, epochs=100, batch_size=32, validation_split=0.1)

# Evaluate the model on the test set
mse = model.evaluate(X_test, y_test)
print("Neural Network Mean Squared Error (MSE):", mse)
```

```
Epoch 1/100
12/12 [==============================] - 1s 27ms/step - loss: 88.9878 - val_loss: 95.6214
Epoch 2/100
12/12 [==============================] - 0s 6ms/step - loss: 66.8432 - val_loss: 74.5905
Epoch 3/100
12/12 [==============================] - 0s 7ms/step - loss: 46.1015 - val_loss: 51.2537
Epoch 4/100
12/12 [==============================] - 0s 6ms/step - loss: 25.1479 - val_loss: 32.1146
Epoch 5/100
12/12 [==============================] - 0s 7ms/step - loss: 15.7502 - val_loss: 22.3354
Epoch 6/100
12/12 [==============================] - 0s 7ms/step - loss: 14.7435 - val_loss: 20.9186
Epoch 7/100
12/12 [==============================] - 0s 6ms/step - loss: 13.7827 - val_loss: 21.0339
Epoch 8/100
12/12 [==============================] - 0s 6ms/step - loss: 13.0465 - val_loss: 21.1688
Epoch 9/100
12/12 [==============================] - 0s 6ms/step - loss: 12.3549 - val_loss: 20.0726
Epoch 10/100
12/12 [==============================] - 0s 5ms/step - loss: 11.9421 - val_loss: 18.3308
Epoch 11/100
```

```
12/12 [==============================] - 0s 6ms/step - loss: 11.3227 - val_loss: 18.3225
Epoch 12/100
12/12 [==============================] - 0s 6ms/step - loss: 10.8282 - val_loss: 17.5922
Epoch 13/100
12/12 [==============================] - 0s 6ms/step - loss: 10.3832 - val_loss: 16.1365
Epoch 14/100
12/12 [==============================] - 0s 6ms/step - loss: 9.9553 - val_loss: 15.7982
Epoch 15/100
12/12 [==============================] - 0s 6ms/step - loss: 9.6199 - val_loss: 15.5031
Epoch 16/100
12/12 [==============================] - 0s 6ms/step - loss: 9.2315 - val_loss: 14.3855
Epoch 17/100
12/12 [==============================] - 0s 7ms/step - loss: 8.9057 - val_loss: 13.4763
Epoch 18/100
12/12 [==============================] - 0s 10ms/step - loss: 8.5365 - val_loss: 13.2159
Epoch 19/100
12/12 [==============================] - 0s 7ms/step - loss: 8.2532 - val_loss: 12.7552
Epoch 20/100
12/12 [==============================] - 0s 9ms/step - loss: 7.9406 - val_loss: 11.9912
Epoch 21/100
12/12 [==============================] - 0s 6ms/step - loss: 7.7160 - val_loss: 11.2167
Epoch 22/100
12/12 [==============================] - 0s 7ms/step - loss: 7.3986 - val_loss: 11.1544
Epoch 23/100
12/12 [==============================] - 0s 6ms/step - loss: 7.0979 - val_loss: 10.2527
Epoch 24/100
12/12 [==============================] - 0s 6ms/step - loss: 6.9131 - val_loss: 9.5855
Epoch 25/100
12/12 [==============================] - 0s 6ms/step - loss: 6.6427 - val_loss: 9.8929
Epoch 26/100
12/12 [==============================] - 0s 6ms/step - loss: 6.3937 - val_loss: 8.6069
Epoch 27/100
12/12 [==============================] - 0s 6ms/step - loss: 6.1220 - val_loss: 8.4415
Epoch 28/100
12/12 [==============================] - 0s 8ms/step - loss: 5.8975 - val_loss: 7.9091
Epoch 29/100
12/12 [==============================] - 0s 10ms/step - loss: 5.6799 - val_loss: 7.3669
Epoch 30/100
12/12 [==============================] - 0s 8ms/step - loss: 5.4389 - val_loss: 7.1376
Epoch 31/100
12/12 [==============================] - 0s 7ms/step - loss: 5.2244 - val_loss: 6.7081
Epoch 32/100
12/12 [==============================] - 0s 6ms/step - loss: 5.0593 - val_loss: 6.3746
Epoch 33/100
12/12 [==============================] - 0s 7ms/step - loss: 4.8544 - val_loss: 6.0352
Epoch 34/100
12/12 [==============================] - 0s 5ms/step - loss: 4.6498 - val_loss: 5.5207
Epoch 35/100
12/12 [==============================] - 0s 5ms/step - loss: 4.4733 - val_loss: 5.4145
Epoch 36/100
12/12 [==============================] - 0s 6ms/step - loss: 4.3280 - val_loss: 5.0068
Epoch 37/100
12/12 [==============================] - 0s 6ms/step - loss: 4.2071 - val_loss: 4.5583
Epoch 38/100
12/12 [==============================] - 0s 7ms/step - loss: 3.9742 - val_loss: 4.5047
Epoch 39/100
12/12 [==============================] - 0s 8ms/step - loss: 3.8101 - val_loss: 4.1302
Epoch 40/100
12/12 [==============================] - 0s 7ms/step - loss: 3.6339 - val_loss: 3.9030
Epoch 41/100
12/12 [==============================] - 0s 7ms/step - loss: 3.5224 - val_loss: 3.7613
Epoch 42/100
12/12 [==============================] - 0s 7ms/step - loss: 3.3822 - val_loss: 3.4220
Epoch 43/100
12/12 [==============================] - 0s 7ms/step - loss: 3.2469 - val_loss: 3.2727
Epoch 44/100
12/12 [==============================] - 0s 7ms/step - loss: 3.1693 - val_loss: 3.0578
Epoch 45/100
12/12 [==============================] - 0s 7ms/step - loss: 3.1145 - val_loss: 2.9568
Epoch 46/100
12/12 [==============================] - 0s 7ms/step - loss: 2.8962 - val_loss: 2.7926
Epoch 47/100
12/12 [==============================] - 0s 7ms/step - loss: 2.7884 - val_loss: 2.5499
Epoch 48/100
12/12 [==============================] - 0s 8ms/step - loss: 2.6909 - val_loss: 2.5586
Epoch 49/100
12/12 [==============================] - 0s 7ms/step - loss: 2.6762 - val_loss: 2.4189
Epoch 50/100
12/12 [==============================] - 0s 7ms/step - loss: 2.6233 - val_loss: 2.2447
Epoch 51/100
12/12 [==============================] - 0s 6ms/step - loss: 2.4492 - val_loss: 2.1889
Epoch 52/100
12/12 [==============================] - 0s 8ms/step - loss: 2.3780 - val_loss: 1.9823
```

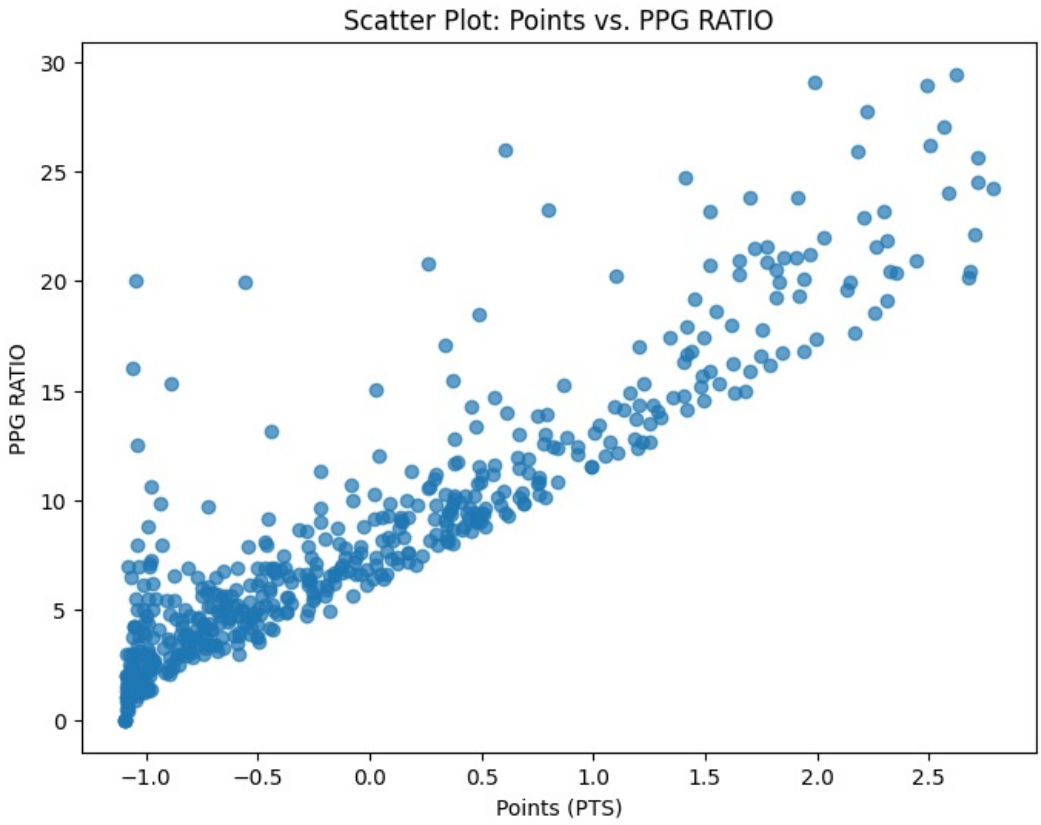
```
Epoch 53/100
12/12 [==============================] - 0s 5ms/step - loss: 2.3788 - val_loss: 1.9864
Epoch 54/100
12/12 [==============================] - 0s 5ms/step - loss: 2.2387 - val_loss: 1.8313
Epoch 55/100
12/12 [==============================] - 0s 5ms/step - loss: 2.2190 - val_loss: 1.8908
Epoch 56/100
12/12 [==============================] - 0s 5ms/step - loss: 2.2078 - val_loss: 1.6676
Epoch 57/100
12/12 [==============================] - 0s 5ms/step - loss: 2.0726 - val_loss: 1.6587
Epoch 58/100
12/12 [==============================] - 0s 8ms/step - loss: 2.0668 - val_loss: 1.6776
Epoch 59/100
12/12 [==============================] - 0s 6ms/step - loss: 2.0100 - val_loss: 1.6376
Epoch 60/100
12/12 [==============================] - 0s 7ms/step - loss: 2.1315 - val_loss: 1.6105
Epoch 61/100
12/12 [==============================] - 0s 7ms/step - loss: 1.9841 - val_loss: 1.4485
Epoch 62/100
12/12 [==============================] - 0s 8ms/step - loss: 1.9662 - val_loss: 1.4116
Epoch 63/100
12/12 [==============================] - 0s 9ms/step - loss: 1.8583 - val_loss: 1.4152
Epoch 64/100
12/12 [==============================] - 0s 8ms/step - loss: 1.8899 - val_loss: 1.4765
Epoch 65/100
12/12 [==============================] - 0s 7ms/step - loss: 1.9083 - val_loss: 1.4310
Epoch 66/100
12/12 [==============================] - 0s 7ms/step - loss: 1.8598 - val_loss: 1.4618
Epoch 67/100
12/12 [==============================] - 0s 6ms/step - loss: 1.7628 - val_loss: 1.2037
Epoch 68/100
12/12 [==============================] - 0s 5ms/step - loss: 1.7621 - val_loss: 1.2366
Epoch 69/100
12/12 [==============================] - 0s 6ms/step - loss: 1.6818 - val_loss: 1.2394
Epoch 70/100
12/12 [==============================] - 0s 6ms/step - loss: 1.7410 - val_loss: 1.1950
Epoch 71/100
12/12 [==============================] - 0s 6ms/step - loss: 1.6380 - val_loss: 1.3629
Epoch 72/100
12/12 [==============================] - 0s 6ms/step - loss: 1.6869 - val_loss: 1.2318
Epoch 73/100
12/12 [==============================] - 0s 6ms/step - loss: 1.6934 - val_loss: 1.1940
Epoch 74/100
12/12 [==============================] - 0s 5ms/step - loss: 1.6000 - val_loss: 1.1916
Epoch 75/100
12/12 [==============================] - 0s 5ms/step - loss: 1.6535 - val_loss: 1.1112
Epoch 76/100
12/12 [==============================] - 0s 7ms/step - loss: 1.6808 - val_loss: 1.1832
Epoch 77/100
12/12 [==============================] - 0s 7ms/step - loss: 1.6429 - val_loss: 1.0711
Epoch 78/100
12/12 [==============================] - 0s 8ms/step - loss: 1.5725 - val_loss: 1.0529
Epoch 79/100
12/12 [==============================] - 0s 7ms/step - loss: 1.5649 - val_loss: 1.0472
Epoch 80/100
12/12 [==============================] - 0s 6ms/step - loss: 1.4622 - val_loss: 1.0052
Epoch 81/100
12/12 [==============================] - 0s 6ms/step - loss: 1.4586 - val_loss: 1.0136
Epoch 82/100
12/12 [==============================] - 0s 5ms/step - loss: 1.4370 - val_loss: 1.0419
Epoch 83/100
12/12 [==============================] - 0s 6ms/step - loss: 1.4456 - val_loss: 0.9846
Epoch 84/100
12/12 [==============================] - 0s 6ms/step - loss: 1.4364 - val_loss: 1.1114
Epoch 85/100
12/12 [==============================] - 0s 6ms/step - loss: 1.4370 - val_loss: 0.9942
Epoch 86/100
12/12 [==============================] - 0s 6ms/step - loss: 1.3969 - val_loss: 0.9733
Epoch 87/100
12/12 [==============================] - 0s 6ms/step - loss: 1.3836 - val_loss: 1.0354
Epoch 88/100
12/12 [==============================] - 0s 6ms/step - loss: 1.4362 - val_loss: 0.9374
Epoch 89/100
12/12 [==============================] - 0s 6ms/step - loss: 1.4181 - val_loss: 0.9371
Epoch 90/100
12/12 [==============================] - 0s 6ms/step - loss: 1.4404 - val_loss: 1.0578
Epoch 91/100
12/12 [==============================] - 0s 6ms/step - loss: 1.4458 - val_loss: 0.9724
Epoch 92/100
12/12 [==============================] - 0s 6ms/step - loss: 1.4225 - val_loss: 0.9478
Epoch 93/100
12/12 [==============================] - 0s 6ms/step - loss: 1.3258 - val_loss: 0.9744
Epoch 94/100
```

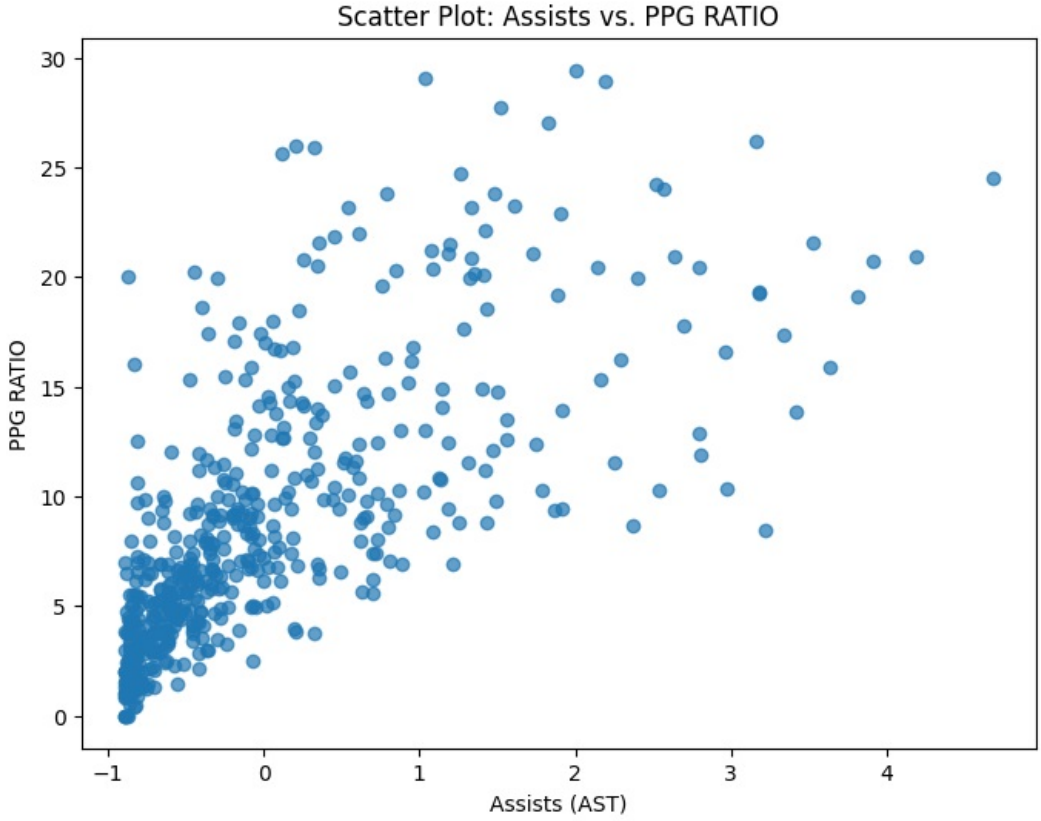
```
12/12 [==============================] - 0s 7ms/step - loss: 1.2990 - val_loss: 0.8888
Epoch 95/100
12/12 [==============================] - 0s 6ms/step - loss: 1.2839 - val_loss: 0.9200
Epoch 96/100
12/12 [==============================] - 0s 7ms/step - loss: 1.2631 - val_loss: 0.8828
Epoch 97/100
12/12 [==============================] - 0s 7ms/step - loss: 1.2683 - val_loss: 0.8617
Epoch 98/100
12/12 [==============================] - 0s 7ms/step - loss: 1.3010 - val_loss: 1.0884
Epoch 99/100
12/12 [==============================] - 0s 7ms/step - loss: 1.4040 - val_loss: 0.9451
Epoch 100/100
12/12 [==============================] - 0s 7ms/step - loss: 1.3139 - val_loss: 0.8077
4/4 [==============================] - 0s 3ms/step - loss: 2.5079
Neural Network Mean Squared Error (MSE): 2.507922410964966
```

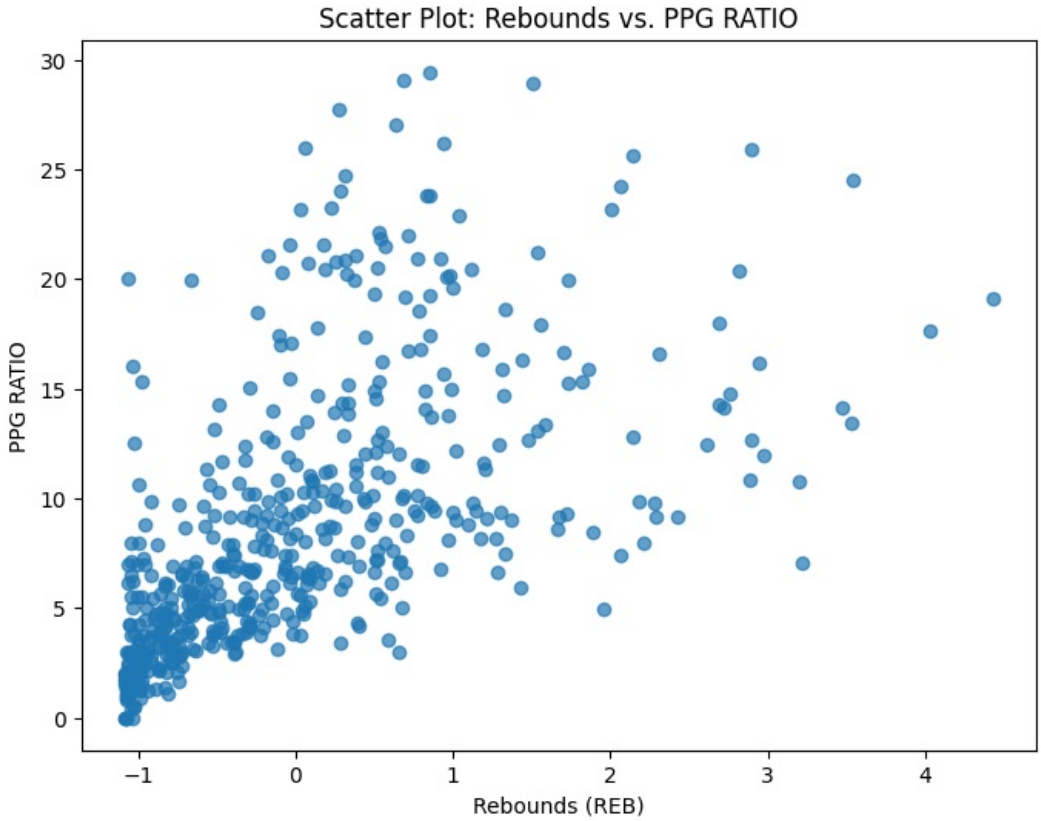
Task5 Visualization with Python (Sctterplot, Heatmap & Barchart)

In [3]:

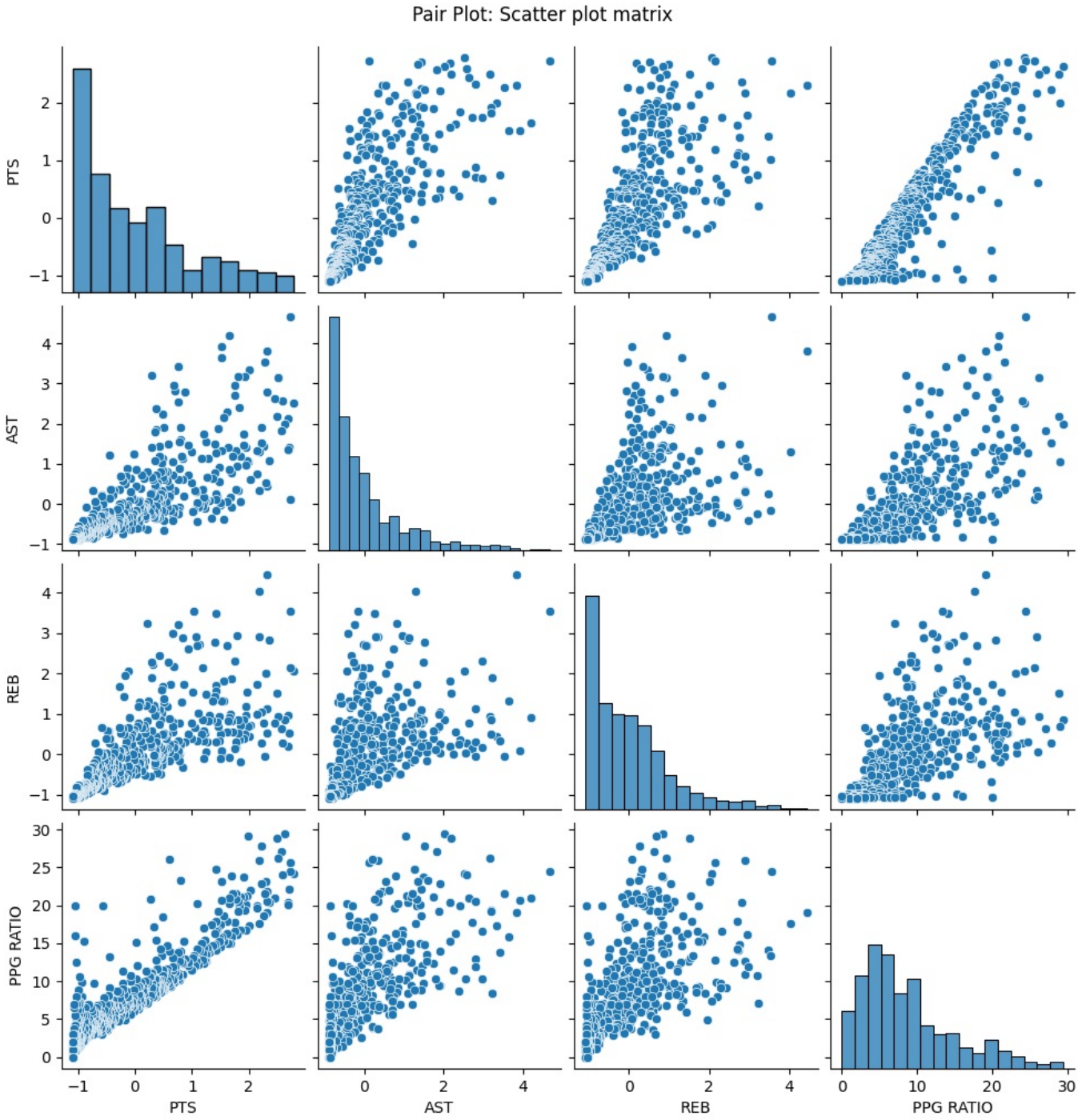
```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset
data = pd.read_csv('C:/Users/Jaskaran/Desktop/Final_NBA_Data.csv')

# Scatter Plot: Points (PTS) vs. PPG RATIO
plt.figure(figsize=(8, 6))
plt.scatter(data['PTS'], data['PPG RATIO'], alpha=0.7)
plt.xlabel('Points (PTS)')
plt.ylabel('PPG RATIO')
plt.title('Scatter Plot: Points vs. PPG RATIO')
plt.show()

# Scatter Plot: Assists (AST) vs. PPG RATIO
plt.figure(figsize=(8, 6))
plt.scatter(data['AST'], data['PPG RATIO'], alpha=0.7)
plt.xlabel('Assists (AST)')
plt.ylabel('PPG RATIO')
plt.title('Scatter Plot: Assists vs. PPG RATIO')
plt.show()

# Scatter Plot: Rebounds (REB) vs. PPG RATIO
plt.figure(figsize=(8, 6))
plt.scatter(data['REB'], data['PPG RATIO'], alpha=0.7)
plt.xlabel('Rebounds (REB)')
plt.ylabel('PPG RATIO')
plt.title('Scatter Plot: Rebounds vs. PPG RATIO')
plt.show()

# Pair Plot: Scatter plot matrix of 'PTS', 'AST', 'REB', and 'PPG RATIO'
sns.pairplot(data[['PTS', 'AST', 'REB', 'PPG RATIO']])
plt.suptitle('Pair Plot: Scatter plot matrix', y=1.02)
plt.show()

# Bar Chart: Team vs. PPG RATIO
plt.figure(figsize=(10, 6))
sns.barplot(x='Team', y='PPG RATIO', data=data, ci=None)
plt.xlabel('Team')
plt.ylabel('PPG RATIO')
plt.title('Bar Chart: Team vs. PPG RATIO')
plt.xticks(rotation=45, ha='right')
plt.show()
```


Scatter Plot: Points vs. PPG RATIO
PPG RATIO
Points (PTS)
30
25
20
15
10
5
0
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5

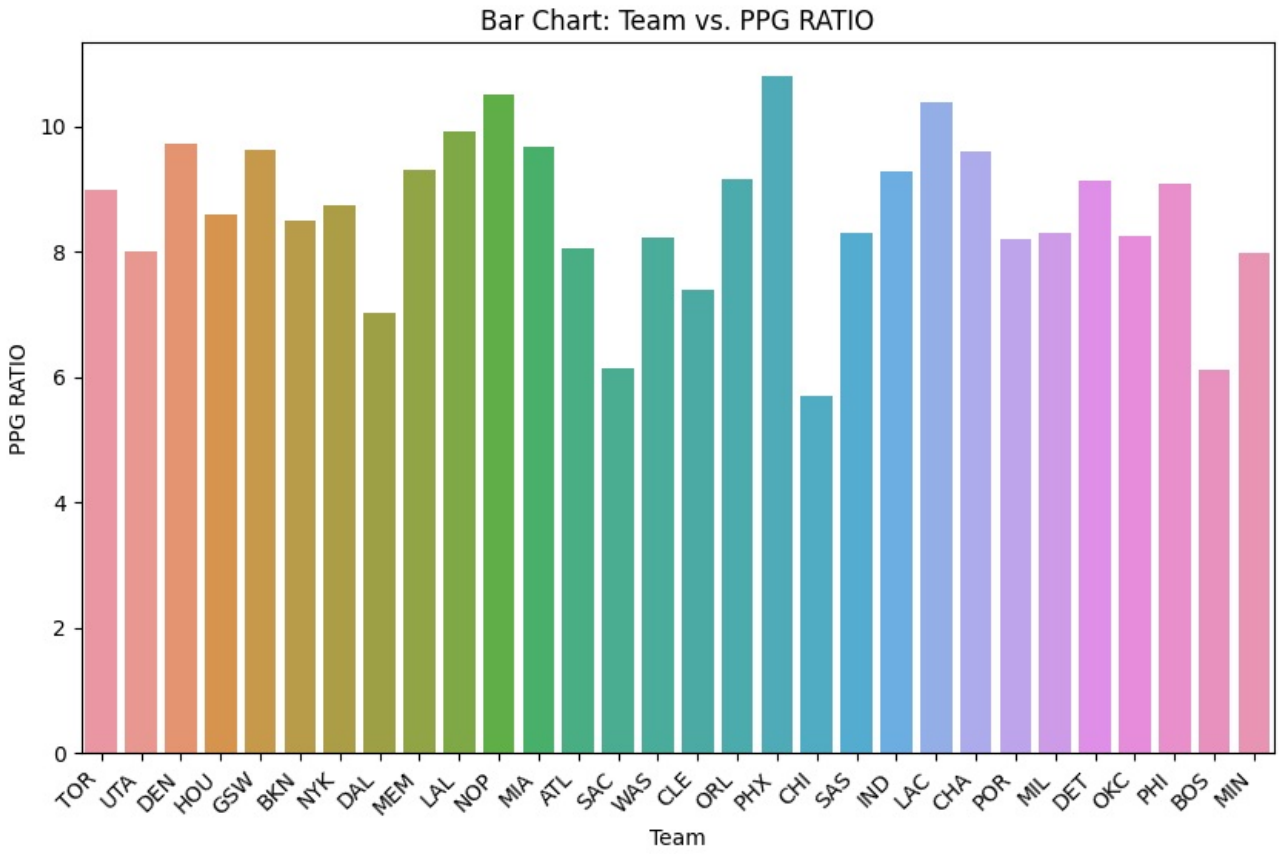

Scatter Plot: Assists vs. PPG RATIO
PPG RATIO
Assists (AST)
30
25
20
15
10
5
0
−1
0
1
2
3
4




```
C:\Users\Jaskaran\AppData\Local\Programs\Python\Python310\lib\site-packages\seaborn\axisgrid.py:118: UserWarning
: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```




```
C:\Users\Jaskaran\AppData\Local\Temp\ipykernel_7712\1131966536.py:39: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.

  sns.barplot(x='Team', y='PPG RATIO', data=data, ci=None)
```



Heatmap

In [1]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Load the dataset
data = pd.read_csv('C:/Users/Jaskaran/Desktop/Final_NBA_Data.csv')

# Drop non-numeric columns from the DataFrame
data_numeric = data.drop(columns=['Player', 'POS', 'Team'])

# Heatmap: Correlation Matrix
correlation_matrix = data_numeric.corr()

# Plot the heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5)
plt.title('Correlation Matrix Heatmap')
plt.show()
```