
NBA Player Game Data and Salary Prediction Project

Overview

In this project, I embarked on a journey to explore the intricate world of NBA player statistics. Leveraging data from individual game performances, I aimed to understand patterns and relationships that might be latent in raw numbers.

Data Collection and Processing

1. **Game Statistics:** I gathered comprehensive game-by-game data of NBA players, detailing their performances on the court. This dataset provides insights into the capabilities and contributions of each player during a match.

1. **Player Salaries for 2023:** A separate dataset containing player salaries for the year 2023 was collected. This financial data offers a perspective on the market value of players based on their skill sets, reputation, and other intangibles.

The two datasets underwent a meticulous cleaning process. Afterward, they were merged based on common attributes, resulting in a unified dataset that juxtaposes player performance metrics with their respective salaries.

Machine Learning & Model Evaluation

With the consolidated data at hand, I delved into the world of machine learning. The goal? To predict a player's salary based on their game performance statistics.

A variety of machine learning models were put to the test. Through rigorous evaluation, I sought to identify which model was best suited for this prediction task.

Future Predictions

To further validate the selected model's robustness, I utilized a new set of randomly generated data. This allowed me to simulate predictions for players based on hypothetical performance metrics, offering a glimpse into potential future salary allocations.

---

source:

- NBA Official Statistics: This website provides comprehensive statistics on player performances throughout the NBA season.
- NBA 2022-2023 Advanced Boxscores from Kaggle: This dataset provides advanced statistics and box scores for NBA games for the 2022-2023 season.
- HoopsHype NBA Player SalariesThis dataset provides NBA Player Salaries NBA games for the 2022-2023 season.

```
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
```

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
```

NBA Data Processing

Data Loading

The project begins with loading multiple datasets related to NBA statistics.

To load the datasets:

- Advanced Stats
- Hustle Stats
- Miscellaneous Stats
- Scoring Stats
- Usage Rates
- Basic Stats
- Defensive Stats
- Player Tracking

```
# Load all the CSV files
files = ["advanced.csv", "hustle.csv", "misc.csv", "scoring.csv",
"usage.csv", "basic.csv", "defense.csv", "tracking.csv"]
dataframes = {file:
pd.read_csv(f'/Users/mengchiehchiang/Desktop/nba_data/{file}') for file in
files}

# Display the first few rows of each dataframe to get an understanding of the
data
first_rows = {file: df.head() for file, df in dataframes.items()}
first_rows
```

```
{'advanced.csv':      gameid  home team  playerid           name     SEC
estOFFRTG  OFFRTG  \
 0  22200001  True  BOS   1627759   Jaylen Brown  2314.0      128.3   131.6
 1  22200001  True  BOS   1628369   Jayson Tatum  2317.0      126.2   125.6
 2  22200001  True  BOS    201143     Al Horford  1386.0      135.8   143.5
 3  22200001  True  BOS   1628401  Derrick White  1442.0      123.0   124.5
 4  22200001  True  BOS    203935   Marcus Smart  2165.0      122.6   123.0

    estDEFRTG  DEFRTG  ...  TOVratio  effFGpct  TSpct  USGpct  estUSGpct  \
 0      114.9   119.2  ...      12.9     0.667  0.691   0.354      0.354
 1      109.1   111.4  ...       9.7     0.700  0.730   0.321      0.331
 2      123.8   126.1  ...       0.0     0.429  0.429   0.140      0.138
 3      104.6   107.5  ...      12.5     0.333  0.291   0.069      0.071
 4      116.1   121.3  ...       5.0     0.438  0.608   0.160      0.164

    [5 rows x 26 columns]}
```

Data Cleaning

One of the critical columns across these datasets is 'SEC'. A preliminary exploration is done to find out which datasets contain this column:

```
# Identify which dataframes have the 'SEC' column
dataframes_with_SEC = [file for file, df in dataframes.items() if 'SEC' in
df.columns]
dataframes_with_SEC
```

```
['advanced.csv',
 'hustle.csv',
 'misc.csv',
 'scoring.csv',
 'usage.csv',
 'basic.csv',
 'tracking.csv']
```

Upon identifying these datasets, we determine the number of missing values in the 'SEC' column:

```
values_in_file = {file: dataframes[file]['SEC'].isna().sum() for file in
dataframes_with_SEC}
values_in_file
```

```
{'advanced.csv': 6285,
 'hustle.csv': 0,
 'misc.csv': 6285,
 'scoring.csv': 6285,
 'usage.csv': 6285,
 'basic.csv': 6285,
 'tracking.csv': 0}
```

```
# Remove rows where 'SEC' column is NaN or 0 for the dataframes that have the
'SEC' column
for file in dataframes_with_SEC:
    dataframes[file] = dataframes[file].dropna(subset=['SEC']).query("SEC !=
0")

# Check if there are any NaN values in the 'SEC' column after cleaning
nan_values_after_cleaning = {file: dataframes[file]['SEC'].isna().sum() for
file in dataframes_with_SEC}
nan_values_after_cleaning
```

```
{'advanced.csv': 0,
 'hustle.csv': 0,
 'misc.csv': 0,
 'scoring.csv': 0,
 'usage.csv': 0,
 'basic.csv': 0,
 'tracking.csv': 0}
```

```
# Find the intersection of playerids across all dataframes
common_playerids = set(dataframes[list(dataframes.keys())[0]]['playerid'])
common_playerids

for file, df in dataframes.items():
    common_playerids &= set(df['playerid'])

# Filter each dataframe to only include rows with playerids in the
intersection
intersected_dataframes = {file: df[df['playerid'].isin(common_playerids)] for
file, df in dataframes.items()}
intersected_dataframes

# Concatenate all the intersected dataframes along columns
result = pd.concat(intersected_dataframes.values(), axis=1)
result
```

```
           gameid   home  team   playerid                     name     SEC  \
0      22200001.0   True   BOS  1627759.0             Jaylen Brown  2314.0
1      22200001.0   True   BOS  1628369.0             Jayson Tatum  2317.0
2      22200001.0   True   BOS   201143.0               Al Horford  1386.0
3      22200001.0   True   BOS  1628401.0           Derrick White  1442.0
4      22200001.0   True   BOS   203935.0             Marcus Smart  2165.0
...           ...    ...   ...        ...                      ...     ...
34019  52200211.0  False   OKC  1630322.0        Lindy Waters III   979.0
34020  52200211.0  False   OKC  1630598.0           Aaron Wiggins   903.0
34021  52200211.0  False   OKC  1631172.0           Ousmane Dieng   430.0
34022  52200211.0  False   OKC  1630544.0                Tre Mann   311.0
34023  52200211.0  False   OKC  1630526.0  Jeremiah Robinson-Earl   311.0

       estOFFRTG  OFFRTG  estDEFRTG  DEFRTG  ...  contestedFGM  contestedFGA
\
0          128.3   131.6      114.9   119.2  ...           5.0           9.0
1          126.2   125.6      109.1   111.4  ...           5.0           7.0
2          135.8   143.5      123.8   126.1  ...           0.0           1.0
3          123.0   124.5      104.6   107.5  ...           0.0           1.0
4          122.6   123.0      116.1   121.3  ...           1.0           3.0
...          ...     ...        ...     ...  ...           ...           ...
34019       95.5    91.4      116.3   114.3  ...           1.0           1.0
34020      101.2    96.9      123.0   123.3  ...           1.0           1.0
34021       85.7    80.0       73.9    73.3  ...           2.0           5.0
34022       90.0    81.8       91.1    90.0  ...           0.0           0.0
34023       90.0    81.8       91.1    90.0  ...           0.0           0.0

       contestedFGpct  uncontestedFGM  uncontestedFGA  uncontestedFGpct  \
0               0.556             9.0            15.0             0.600
1               0.714             8.0            13.0             0.615
2               0.000             2.0             6.0             0.333
3               0.000             1.0             2.0             0.500
4               0.333             2.0             5.0             0.400
...               ...             ...             ...               ...
```

```
34019           1.000           1.0             6.0             0.167
34020           1.000           1.0             2.0             0.500
34021           0.400           0.0             0.0             0.000
34022           0.000           0.0             0.0             0.000
34023           0.000           1.0             1.0             1.000

       FGpct  defendedatrimFGM  defendedatrimFGA  defendedatrimFGpct
0      0.582               1.0               3.0               0.333
1      0.650               0.0               0.0               0.000
2      0.285               2.0               3.0               0.667
3      0.333               2.0               3.0               0.667
4      0.375               3.0               3.0               1.000
...      ...               ...               ...                 ...
34019  0.285               2.0               2.0               1.000
34020  0.667               0.0               0.0               0.000
34021  0.400               1.0               2.0               0.500
34022  0.000               0.0               0.0               0.000
34023  1.000               0.0               1.0               0.000

[32687 rows x 184 columns]
```

There will be redundant columns after concatenation (e.g., repeated 'playerid' columns). We'll keep only one instance of each redundant column.

```
result = result.loc[:, ~result.columns.duplicated()]
result
```

```
           gameid   home team   playerid                    name     SEC  \
0      22200001.0   True  BOS  1627759.0            Jaylen Brown  2314.0
1      22200001.0   True  BOS  1628369.0            Jayson Tatum  2317.0
2      22200001.0   True  BOS   201143.0              Al Horford  1386.0
3      22200001.0   True  BOS  1628401.0           Derrick White  1442.0
4      22200001.0   True  BOS   203935.0            Marcus Smart  2165.0
...           ...    ...  ...        ...                     ...     ...
34019  52200211.0  False  OKC  1630322.0       Lindy Waters III   979.0
34020  52200211.0  False  OKC  1630598.0          Aaron Wiggins   903.0
34021  52200211.0  False  OKC  1631172.0          Ousmane Dieng   430.0
34022  52200211.0  False  OKC  1630544.0               Tre Mann   311.0
34023  52200211.0  False  OKC  1630526.0  Jeremiah Robinson-Earl   311.0

       estOFFRTG  OFFRTG  estDEFRTG  DEFRTG  ...  passes  contestedFGM  \
0          128.3   131.6      114.9   119.2  ...    39.0           5.0
1          126.2   125.6      109.1   111.4  ...    43.0           5.0
2          135.8   143.5      123.8   126.1  ...    28.0           0.0
3          123.0   124.5      104.6   107.5  ...    23.0           0.0
4          122.6   123.0      116.1   121.3  ...    52.0           1.0
...          ...     ...        ...     ...  ...     ...           ...
34019       95.5    91.4      116.3   114.3  ...    13.0           1.0
34020      101.2    96.9      123.0   123.3  ...    13.0           1.0
34021       85.7    80.0       73.9    73.3  ...     8.0           2.0
```

```
34022       90.0    81.8       91.1    90.0  ...     9.0           0.0
34023       90.0    81.8       91.1    90.0  ...     8.0           0.0

       contestedFGA  contestedFGpct  uncontestedFGM  uncontestedFGA  \
0               9.0           0.556             9.0            15.0
1               7.0           0.714             8.0            13.0
2               1.0           0.000             2.0             6.0
3               1.0           0.000             1.0             2.0
4               3.0           0.333             2.0             5.0
...             ...             ...             ...             ...
34019           1.0           1.000             1.0             6.0
34020           1.0           1.000             1.0             2.0
34021           5.0           0.400             0.0             0.0
34022           0.0           0.000             0.0             0.0
34023           0.0           0.000             1.0             1.0

       uncontestedFGpct  defendedatrimFGM  defendedatrimFGA  \
0                 0.600               1.0               3.0
1                 0.615               0.0               0.0
2                 0.333               2.0               3.0
3                 0.500               2.0               3.0
4                 0.400               3.0               3.0
...                 ...               ...               ...
34019             0.167               2.0               2.0
34020             0.500               0.0               0.0
34021             0.000               1.0               2.0
34022             0.000               0.0               0.0
34023             1.000               0.0               1.0

       defendedatrimFGpct
0                   0.333
1                   0.000
2                   0.667
3                   0.667
4                   1.000
...                   ...
34019               1.000
34020               0.000
34021               0.500
34022               0.000
34023               0.000

[32687 rows x 134 columns]
```

```
intersected_data = result
```

NBA Data Processing: Integrating Salary Data

After handling player statistics, the next logical step in our analysis is to incorporate player salary information, which provides a more holistic understanding of each player's value proposition.

Loading Salary Data

First, we load the salary data from the specified CSV file. This dataset contains the salary information for NBA players for the year 2022/2023.

```
salary_data =
pd.read_csv("/Users/mengchiehchiang/Desktop/nba_data/2023_salary.csv")
salary_data
```

```
                PLAYER     2022/23
0      Chance Comanche       5,849
1        Jacob Gilyard       5,849
2         RaiQuan Gray       5,849
3     Tristan Thompson      16,700
4           Ibou Badji      18,226
..                 ...         ...
571       LeBron James  44,474,988
572  Russell Westbrook  47,080,179
573          John Wall  47,345,760
574      Stephen Curry  48,070,014
575           AJ Green   1,901,769

[576 rows x 2 columns]
```

Converting Salary to Integer

A common issue with financial data is that it often contains non-numeric characters like dollar signs ($) and commas (,). For further analysis, it's crucial to convert these to pure integer values.

```
salary_data['2022/23'] = salary_data['2022/23'].replace('[\$,]', '',
regex=True).astype(int)
salary_data
```

```
                PLAYER   2022/23
0      Chance Comanche      5849
1        Jacob Gilyard      5849
2         RaiQuan Gray      5849
3     Tristan Thompson     16700
4           Ibou Badji     18226
..                 ...       ...
571       LeBron James  44474988
572  Russell Westbrook  47080179
573          John Wall  47345760
574      Stephen Curry  48070014
```

```
575          AJ Green   1901769

[576 rows x 2 columns]
```

Merging Salary Data with Player Stats With cleaned salary data at hand, the next step is to merge this with our previously constructed dataset containing player statistics.

```
merged_data = pd.merge(intersected_data, salary_data, left_on="name",
right_on="PLAYER", how="left")
```

Handling Duplicate Salary Columns

After merging, there's a possibility of having two salary columns: the original salary column (if it existed) and the new 2022/23 column. To ensure consistency, we'll:

1. Replace the original salary column with values from the 2022/23 column.
1. Drop the now redundant 2022/23 and PLAYER columns.

```
if '2022/23' in merged_data.columns:
    merged_data['salary'] = merged_data['2022/23']
    merged_data.drop(columns=['2022/23', 'PLAYER'], inplace=True)
```

```
merged_data
```

```
           gameid   home team   playerid                    name     SEC  \
0      22200001.0   True  BOS  1627759.0            Jaylen Brown  2314.0
1      22200001.0   True  BOS  1628369.0            Jayson Tatum  2317.0
2      22200001.0   True  BOS   201143.0              Al Horford  1386.0
3      22200001.0   True  BOS  1628401.0          Derrick White  1442.0
4      22200001.0   True  BOS   203935.0           Marcus Smart  2165.0
...           ...    ...  ...        ...                     ...     ...
32682  52200211.0  False  OKC  1630322.0        Lindy Waters III   979.0
32683  52200211.0  False  OKC  1630598.0          Aaron Wiggins   903.0
32684  52200211.0  False  OKC  1631172.0          Ousmane Dieng   430.0
32685  52200211.0  False  OKC  1630544.0               Tre Mann   311.0
32686  52200211.0  False  OKC  1630526.0 Jeremiah Robinson-Earl   311.0

       estOFFRTG  OFFRTG  estDEFRTG  DEFRTG  ...  contestedFGM  contestedFGA
\
0          128.3   131.6      114.9   119.2  ...           5.0           9.0
1          126.2   125.6      109.1   111.4  ...           5.0           7.0
2          135.8   143.5      123.8   126.1  ...           0.0           1.0
3          123.0   124.5      104.6   107.5  ...           0.0           1.0
4          122.6   123.0      116.1   121.3  ...           1.0           3.0
...          ...     ...        ...     ...  ...           ...           ...
32682       95.5    91.4      116.3   114.3  ...           1.0           1.0
32683      101.2    96.9      123.0   123.3  ...           1.0           1.0
32684       85.7    80.0       73.9    73.3  ...           2.0           5.0
32685       90.0    81.8       91.1    90.0  ...           0.0           0.0
32686       90.0    81.8       91.1    90.0  ...           0.0           0.0
```

```
       contestedFGpct  uncontestedFGM  uncontestedFGA  uncontestedFGpct  \
0               0.556             9.0            15.0             0.600
1               0.714             8.0            13.0             0.615
2               0.000             2.0             6.0             0.333
3               0.000             1.0             2.0             0.500
4               0.333             2.0             5.0             0.400
...               ...             ...             ...               ...
32682           1.000             1.0             6.0             0.167
32683           1.000             1.0             2.0             0.500
32684           0.400             0.0             0.0             0.000
32685           0.000             0.0             0.0             0.000
32686           0.000             1.0             1.0             1.000

       defendedatrimFGM  defendedatrimFGA  defendedatrimFGpct      salary
0                   1.0               3.0               0.333  29776785.0
1                   0.0               0.0               0.000  30351780.0
2                   2.0               3.0               0.667  26500000.0
3                   2.0               3.0               0.667  17142857.0
4                   3.0               3.0               1.000  17207142.0
...                 ...               ...                 ...         ...
32682               2.0               2.0               1.000   2316876.0
32683               0.0               0.0               0.000   1563518.0
32684               1.0               2.0               0.500   4569840.0
32685               0.0               0.0               0.000   3046200.0
32686               0.0               1.0               0.000   2000000.0

[32687 rows x 135 columns]
```

I manually exported this file to address inconsistencies in player naming conventions. Although certain names might seem distinct, they actually refer to the same individual. For instance, names like 'Michael Foster Jr.' and 'Michael Foster' in the dataset represent the same player.

```
output_file_path = "/Users/mengchiehchiang/Desktop/nba_data/merged_data.csv"
merged_data.to_csv(output_file_path, index=False)
```

===========>>>>> Fixing

Done

Re-load the newly uploaded 'mergd_data.csv' file

```
clean_data =
pd.read_csv("/Users/mengchiehchiang/Desktop/nba_data/merged_data.csv",
low_memory=False)
clean_data
```

```
           gameid  home team   playerid                 name     SEC  \
0      22200001.0  True  BOS  1627759.0         Jaylen Brown  2314.0
1      22200001.0  True  BOS  1628369.0         Jayson Tatum  2317.0
2      22200001.0  True  BOS   201143.0           Al Horford  1386.0
```

```
3      22200001.0   True  BOS  1628401.0         Derrick White  1442.0
4      22200001.0   True  BOS   203935.0          Marcus Smart  2165.0
...           ...    ...  ...        ...                   ...     ...
32682  52200211.0  False  OKC  1630322.0       Lindy Waters III   979.0
32683  52200211.0  False  OKC  1630598.0         Aaron Wiggins   903.0
32684  52200211.0  False  OKC  1631172.0         Ousmane Dieng   430.0
32685  52200211.0  False  OKC  1630544.0              Tre Mann   311.0
32686  52200211.0  False  OKC  1630526.0  Jeremiah Robinson-Earl   311.0

       estOFFRTG  OFFRTG  estDEFRTG  DEFRTG  ...  contestedFGM  contestedFGA  
\
0          128.3   131.6      114.9   119.2  ...           5.0           9.0
1          126.2   125.6      109.1   111.4  ...           5.0           7.0
2          135.8   143.5      123.8   126.1  ...           0.0           1.0
3          123.0   124.5      104.6   107.5  ...           0.0           1.0
4          122.6   123.0      116.1   121.3  ...           1.0           3.0
...          ...     ...        ...     ...  ...           ...           ...
32682       95.5    91.4      116.3   114.3  ...           1.0           1.0
32683      101.2    96.9      123.0   123.3  ...           1.0           1.0
32684       85.7    80.0       73.9    73.3  ...           2.0           5.0
32685       90.0    81.8       91.1    90.0  ...           0.0           0.0
32686       90.0    81.8       91.1    90.0  ...           0.0           0.0

       contestedFGpct  uncontestedFGM  uncontestedFGA  uncontestedFGpct  \
0               0.556             9.0            15.0             0.600   
1               0.714             8.0            13.0             0.615   
2               0.000             2.0             6.0             0.333   
3               0.000             1.0             2.0             0.500   
4               0.333             2.0             5.0             0.400   
...               ...             ...             ...               ...   
32682           1.000             1.0             6.0             0.167   
32683           1.000             1.0             2.0             0.500   
32684           0.400             0.0             0.0             0.000   
32685           0.000             0.0             0.0             0.000   
32686           0.000             1.0             1.0             1.000   

       defendedatrimFGM  defendedatrimFGA  defendedatrimFGpct      salary  
0                   1.0               3.0               0.333  29776785.0  
1                   0.0               0.0               0.000  30351780.0  
2                   2.0               3.0               0.667  26500000.0  
3                   2.0               3.0               0.667  17142857.0  
4                   3.0               3.0               1.000  17207142.0  
...                 ...               ...                 ...         ...  
32682               2.0               2.0               1.000   2316876.0  
32683               0.0               0.0               0.000   1563518.0  
32684               1.0               2.0               0.500   4569840.0  
32685               0.0               0.0               0.000   3046200.0  
32686               0.0               1.0               0.000   2000000.0  
```

```
[32687 rows x 135 columns]
```

Check for NaN values in the 'salary' column.

```
nan_salary_rows_clean = clean_data[clean_data['salary'].isna()]
players_with_nan_salary_clean =
nan_salary_rows_clean['name'].dropna().unique()
players_with_nan_salary_clean
```

```
array([], dtype=object)
```

When considering which data to use as features for machine learning, we usually need to consider the following factors:

- Data integrity: There cannot be too many missing values in the features.
- Correlation: Features should have a certain correlation with the target variable.
- Data type: Usually, we need numeric data. If some features are categorical, we may need to encode them (e.g. One-hot encoding).
- Multicollinearity: Avoid high correlations between features, which may lead to model instability

Checking for Missing Values

To ensure the quality of the data used for machine learning, it's essential to check the percentage of missing values in each column.Columns with too many missing values might not provide meaningful insights and could adversely affect the model's performance.

1. Checking the percentage of missing values for each column
1. Filtering out columns with less than 20% missing values

```
missing_percentage = (clean_data.isnull().sum() / len(clean_data)) * 100

potential_features = missing_percentage[missing_percentage <
20].index.tolist()
potential_features
```

```
['gameid',
 'home',
 'team',
 'playerid',
 'name',
 'SEC',
 'estOFFRTG',
 'OFFRTG',
 'estDEFRTG',
 'DEFRTG',
 'estNETRTG',
 'NETRTG',
 'ASTpct',
 'ASTtoTOV',
```

```
'ASTratio',
'ORBpct',
'DRBpct',
'REBpct',
'TOVratio',
'effFGpct',
'TSpct',
'USGpct',
'estUSGpct',
'estpace',
'pace',
'paceper40',
'POS',
'pie',
'PTS',
'contestedshots',
'contestedshots2P',
'contestedshots3P',
'deflections',
'chargesdrawn',
'screenAST',
'screenASTPTS',
'looseballsrecoveredOFF',
'looseballsrecoveredDEF',
'looseballsrecoveredtotal',
'OFFboxouts',
'DEFboxouts',
'boxoutplayerteamREB',
'boxoutplayerREB',
'boxouts',
'PTSoffTOV',
'2ndPTS',
'FBPTS',
'PIP',
'oppPTSoffTOV',
'opp2ndPTS',
'oppFBPTS',
'oppPIP',
'BLK',
'BLKA',
'PF',
'foulsdrawn',
'pctFGA2P',
'pctFGA3P',
'pctPTS2P',
'pctPTSmidrange2P',
'pctPTS3P',
'pctFBPTS',
'pctPTSFT',
'pctPTSoffTOV',
```

```
'pctPIP',
'pctAST2P',
'pctUAST2P',
'pctAST3P',
'pctUAST3P',
'pctASTFGM',
'pctUASTFGM',
'pctFGM',
'pctFGA',
'pct3PM',
'pct3PA',
'pctFTM',
'pctFTA',
'pctORB',
'pctDRB',
'pctTRB',
'pctAST',
'pctTOV',
'pctSTL',
'pctBLK',
'pctBLKallowed',
'pctPF',
'pctPFdrawn',
'pctPTS',
'FGM',
'FGA',
'FGpct',
'3PM',
'3PA',
'3Ppct',
'FTM',
'FTA',
'FTpct',
'ORB',
'DRB',
'TRB',
'AST',
'STL',
'TOV',
'plusminusPTS',
'matchupSEC',
'partialPOS',
'switcheson',
'playerPTS',
'matchupAST',
'matchupTOV',
'matchupFGM',
'matchupFGA',
'matchupFGpct',
'matchup3PM',
```

```
 'matchup3PA',
 'matchup3Ppct',
 'SPD',
 'DIST',
 'REBchancesOFF',
 'REBchancesDEF',
 'REBchancestotal',
 'touches',
 '2ndAST',
 'FTAST',
 'passes',
 'contestedFGM',
 'contestedFGA',
 'contestedFGpct',
 'uncontestedFGM',
 'uncontestedFGA',
 'uncontestedFGpct',
 'defendedatrimFGM',
 'defendedatrimFGA',
 'defendedatrimFGpct',
 'salary']
```

One-hot Encoding

For machine learning models to understand and use categorical data, we need to convert these categories into a numerical format. One common method is one-hot encoding. In this case, the **'team'** column, which is categorical, is being one-hot encoded. This means each team will get its own column, and for each row, the respective team column will have a value of **1**, while all other team columns will be **0**.

```
encoded_data = pd.get_dummies(clean_data, columns=['team'])
encoded_data
```

```
           gameid   home   playerid                      name     SEC  \
0      22200001.0   True  1627759.0              Jaylen Brown  2314.0   
1      22200001.0   True  1628369.0              Jayson Tatum  2317.0   
2      22200001.0   True   201143.0                Al Horford  1386.0   
3      22200001.0   True  1628401.0             Derrick White  1442.0   
4      22200001.0   True   203935.0              Marcus Smart  2165.0   
...           ...    ...        ...                       ...     ...   
32682  52200211.0  False  1630322.0          Lindy Waters III   979.0   
32683  52200211.0  False  1630598.0            Aaron Wiggins   903.0   
32684  52200211.0  False  1631172.0            Ousmane Dieng   430.0   
32685  52200211.0  False  1630544.0                  Tre Mann   311.0   
32686  52200211.0  False  1630526.0  Jeremiah Robinson-Earl   311.0   

       estOFFRTG  OFFRTG  estDEFRTG  DEFRTG  estNETRTG  ...  team_OKC  \
0          128.3   131.6      114.9   119.2       13.4  ...         0   
1          126.2   125.6      109.1   111.4       17.1  ...         0   
2          135.8   143.5      123.8   126.1       12.0  ...         0   
```

```
3          123.0   124.5      104.6   107.5       18.3  ...        0
4          122.6   123.0      116.1   121.3        6.4  ...        0
...          ...     ...        ...     ...        ...  ...      ...
32682       95.5    91.4      116.3   114.3      -20.8  ...        1
32683      101.2    96.9      123.0   123.3      -21.8  ...        1
32684       85.7    80.0       73.9    73.3       11.8  ...        1
32685       90.0    81.8       91.1    90.0       -1.1  ...        1
32686       90.0    81.8       91.1    90.0       -1.1  ...        1

       team_ORL  team_PHI  team_PHX  team_POR  team_SAC  team_SAS  team_TOR
\
0             0         0         0         0         0         0         0
1             0         0         0         0         0         0         0
2             0         0         0         0         0         0         0
3             0         0         0         0         0         0         0
4             0         0         0         0         0         0         0
...         ...       ...       ...       ...       ...       ...       ...
32682         0         0         0         0         0         0         0
32683         0         0         0         0         0         0         0
32684         0         0         0         0         0         0         0
32685         0         0         0         0         0         0         0
32686         0         0         0         0         0         0         0

       team_UTA  team_WAS
0             0         0
1             0         0
2             0         0
3             0         0
4             0         0
...         ...       ...
32682         0         0
32683         0         0
32684         0         0
32685         0         0
32686         0         0

[32687 rows x 164 columns]
```

```
original_rows = clean_data.shape[0]
original_rows
```

```
32687
```

Modeling

1. Using RandomForest for Salary Prediction

To predict player salaries, we are initially experimenting with the Random Forest method. However, instead of using all features available in the dataset, we're focusing on a subset of features: ["SEC", "PTS", "TRB", "AST", "STL", "TOV"].

```
features = ["SEC", "PTS", "TRB", "AST", "STL", "TOV"]
label = "salary"
```

Before modeling, it's essential to handle missing values. In this case, we replace NaN values in our feature and label columns with 0.

```
data_filled = encoded_data.copy()
data_filled[features + [label]] = data_filled[features + [label]].fillna(0)
data_filled
```

```
           gameid   home   playerid                     name     SEC  \
0      22200001.0   True  1627759.0             Jaylen Brown  2314.0
1      22200001.0   True  1628369.0             Jayson Tatum  2317.0
2      22200001.0   True   201143.0               Al Horford  1386.0
3      22200001.0   True  1628401.0           Derrick White  1442.0
4      22200001.0   True   203935.0             Marcus Smart  2165.0
...           ...    ...        ...                      ...     ...
32682  52200211.0  False  1630322.0        Lindy Waters III   979.0
32683  52200211.0  False  1630598.0           Aaron Wiggins   903.0
32684  52200211.0  False  1631172.0           Ousmane Dieng   430.0
32685  52200211.0  False  1630544.0                Tre Mann   311.0
32686  52200211.0  False  1630526.0  Jeremiah Robinson-Earl   311.0

       estOFFRTG  OFFRTG  estDEFRTG  DEFRTG  estNETRTG  ...  team_OKC  \
0          128.3   131.6      114.9   119.2       13.4  ...         0
1          126.2   125.6      109.1   111.4       17.1  ...         0
2          135.8   143.5      123.8   126.1       12.0  ...         0
3          123.0   124.5      104.6   107.5       18.3  ...         0
4          122.6   123.0      116.1   121.3        6.4  ...         0
...          ...     ...        ...     ...        ...  ...       ...
32682       95.5    91.4      116.3   114.3      -20.8  ...         1
32683      101.2    96.9      123.0   123.3      -21.8  ...         1
32684       85.7    80.0       73.9    73.3       11.8  ...         1
32685       90.0    81.8       91.1    90.0       -1.1  ...         1
32686       90.0    81.8       91.1    90.0       -1.1  ...         1

       team_ORL  team_PHI  team_PHX  team_POR  team_SAC  team_SAS  team_TOR
\
0             0         0         0         0         0         0         0
1             0         0         0         0         0         0         0
2             0         0         0         0         0         0         0
3             0         0         0         0         0         0         0
4             0         0         0         0         0         0         0
...         ...       ...       ...       ...       ...       ...       ...
32682         0         0         0         0         0         0         0
32683         0         0         0         0         0         0         0
32684         0         0         0         0         0         0         0
32685         0         0         0         0         0         0         0
32686         0         0         0         0         0         0         0
```

```
       team_UTA  team_WAS
0             0         0
1             0         0
2             0         0
3             0         0
4             0         0
...         ...       ...
32682         0         0
32683         0         0
32684         0         0
32685         0         0
32686         0         0

[32687 rows x 164 columns]
```

```
nan_check = data_filled[features + [label]].isna().sum()
nan_check
```

```
SEC       0
PTS       0
TRB       0
AST       0
STL       0
TOV       0
salary    0
dtype: int64
```

Splitting the Data

We divide our data into training and testing sets. 70% of the data will be used for training the model, and the remaining 30% will be used for testing.

```
X = data_filled[features]
y = data_filled[label]
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)
```

Standardizing the Data

Standardizing the features ensures that they have a mean of 0 and a standard deviation of 1. It's a good practice, especially when using algorithms that are sensitive to the scale of input features

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

Training the Model

```
rf = RandomForestRegressor(n_estimators=200, random_state=42)
rf.fit(X_train_scaled, y_train)
```

```
RandomForestRegressor(n_estimators=200, random_state=42)
```

Model Evaluation

The model's performance is evaluated on the test data using Mean Squared Error (MSE) and R^2 score.

```
y_pred = rf.predict(X_test_scaled)
```

```
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
mse, r2
```

```
(65674049790590.52, 0.44957067950620955)
```

Cross-Validation

For a more robust evaluation, 10-fold cross-validation is performed on the training data using the R^2 score as the evaluation metric.

```
rf_scores = cross_val_score(rf, X_train_scaled, y_train, cv=10, scoring="r2")
```

```
rf_scores
```

```
array([0.46094721, 0.47747761, 0.48569403, 0.46438547, 0.44381812,
       0.44997687, 0.4714885 , 0.44230868, 0.4680954 , 0.47130812])
```

```
mean_score = scores.mean()
std_score = scores.std()

mean_score, std_score
```

```
(0.3605842454097611, 0.02448514190708397)
```

From these scores, we can compute the average score and the standard deviation:

Average R^2 Score: 0.3605 Standard Deviation of the R^2 Scores: 0.02448

Analysis: The average R^2 score of 0.3605 indicates the average performance of the model across the cross-validation folds.

The standard deviation of 0.0248 shows the variability in the model's performance. **A lower standard deviation indicates that the model's performance is consistent across different cross-validation folds.**

But R^2 only 36% which mean the model was able to explain around 35.96% of the variability in the target variable using the features it was trained on. this is not a good model

2. LinearRegression

After trying the Random Forest method, we gave the Linear Regression method a shot to predict the players' salaries. Interestingly, we stuck with the same set of features as before: ["SEC", "PTS", "TRB", "AST", "STL", "TOV"].

```
lr = LinearRegression()
lr.fit(X_train_scaled, y_train)
y_pred_lr = lr.predict(X_test_scaled)

mse_lr = mean_squared_error(y_test, y_pred_lr)
r2_lr = r2_score(y_test, y_pred_lr)

mse_lr, r2_lr
```

```
(72897004354679.88, 0.3901317778267487)
```

```
lr_scores = cross_val_score(lr, X_train_scaled, y_train, cv=10, scoring="r2")

lr_scores
```

```
array([0.40425041, 0.41162761, 0.40351053, 0.3686454 , 0.41025208,
       0.37367627, 0.3849031 , 0.41163088, 0.39384986, 0.3996993 ])
```

```
mean_lr_scores = lr_scores.mean()
std_lr_scores = lr_scores.std()

mean_lr_scores, std_lr_scores
```

```
(0.3962045440845966, 0.014845201670707688)
```

Average R^2 Score: The mean R^2 score of approximately 0.3962 tells us the average performance of the Linear Regression model across the cross-validation folds.

Linear Regression with the selected features has shown a decent performance, capturing about **39.62%** of the variability in the player's salaries on average across the cross-validation folds.

2. xgboost

Third try the xgboost methed to train the data to perdict the salary. but only use ["SEC", "PTS", "TRB", "AST", "STL", "TOV"] as features.

```
!pip3 install xgboost
import xgboost as xgb
```

```
Collecting xgboost
  Downloading
xgboost-1.7.6-py3-none-macosx_10_15_x86_64.macosx_11_0_x86_64.macosx_12_0_x86
_64.whl (1.8 MB)
ent already satisfied: scipy in
/Users/mengchiehchiang/opt/anaconda3/lib/python3.9/site-packages (from
```

```
xgboost) (1.7.3)
Requirement already satisfied: numpy in
/Users/mengchiehchiang/opt/anaconda3/lib/python3.9/site-packages (from
xgboost) (1.21.4)
Installing collected packages: xgboost
Successfully installed xgboost-1.7.6
```

```
xgb_regressor = xgb.XGBRegressor(n_estimators=150, learning_rate=0.1,
random_state=42)
xgb_regressor.fit(X_train_scaled, y_train)
```

```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=0.1, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             n_estimators=150, n_jobs=None, num_parallel_tree=None,
             predictor=None, random_state=42, ...)
```

```
y_pred_xgb = xgb_regressor.predict(X_test_scaled)
```

```
# Evaluate the XGBoost model
mse_xgb = mean_squared_error(y_test, y_pred_xgb)
r2_xgb = r2_score(y_test, y_pred_xgb)

mse_xgb, r2_xgb
```

```
(73038279847220.08, 0.3889498440252863)
```

The R^2 score of approximately 0.3890 suggests our model explains roughly 38.99% of the variance in our target variable.

```
xgb_scores = cross_val_score(xgb_regressor, X_train_scaled, y_train, cv=10,
scoring="r2")
```

```
xgb_scores
```

```
array([0.42404059, 0.42686519, 0.40813595, 0.36617397, 0.4080994 ,
       0.35995248, 0.3895982 , 0.40278518, 0.39975233, 0.39981147])
```

```
xgb_scores.mean()
```

```
0.4652106791078654
```

The mean R^2 score from cross-validation is approximately 0.4652, indicating the average performance across the validation folds.

Comparative Analysis of Models Let's compare the performance of our three models: Random Forest, Linear Regression, and XGBoost.

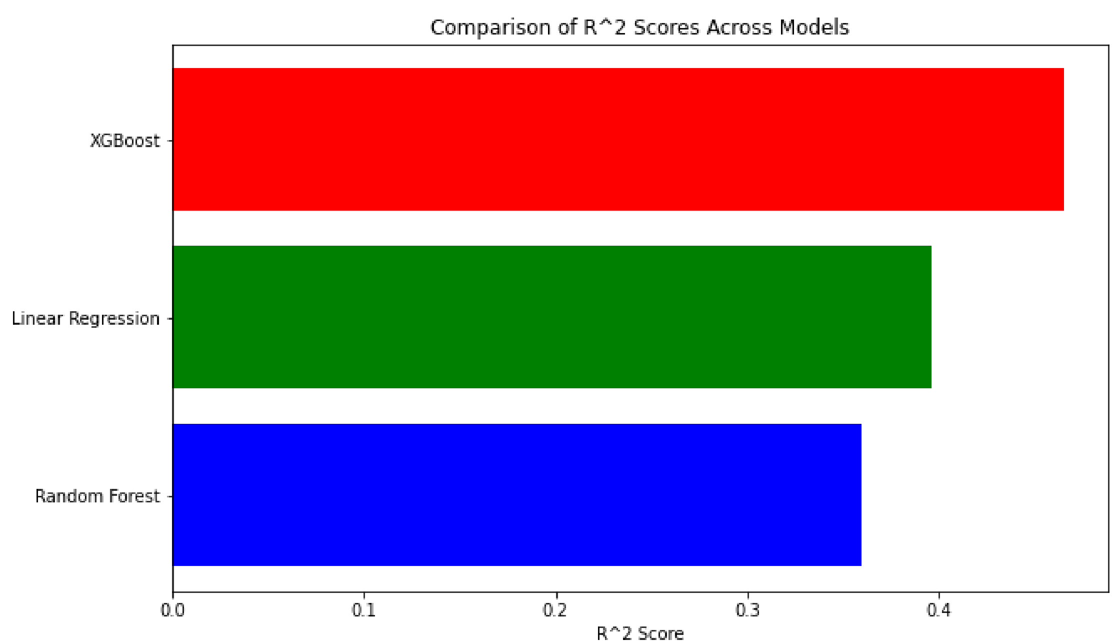
R^2 Score Summary:

- Random Forest: 0.3596
- Linear Regression: 0.3962
- XGBoost: 0.4652

```
import matplotlib.pyplot as plt

models = ['Random Forest', 'Linear Regression', 'XGBoost']
r2_scores = [0.3596, 0.3962, 0.4652]

plt.figure(figsize=(10,6))
plt.barh(models, r2_scores, color=['blue', 'green', 'red'])
plt.xlabel('R^2 Score')
plt.title('Comparison of R^2 Scores Across Models')
plt.show()
```



Extended Feature Analysis

To enhance the predictive accuracy of our models, we have opted to incorporate an extended set of features:

```
extended_features = [
    "SEC", "estOFFRTG", "OFFRTG", "estDEFRTG", "DEFRTG", "estNETRTG", 
"NETRTG", "ASTpct",
    "ASTtoTOV", "ASTratio", "ORBpct", "DRBpct", "REBpct", "TOVratio", 
"effFGpct", "TSpct",
    "USGpct", "estUSGpct", "estpace", "pace", "paceper40", "POS", "pie",
```

```
"PTS", "contestedshots"
]
```

These features encompass a diverse range of player statistics, providing a more holistic view of a player's contribution both offensively and defensively.

Let's delve into how these extended features perform with this three predictive models:

- Random Forest
- Linear Regression
- XGBoost

```
# Fill NaN values with 0 in the specified columns
data_filled = encoded_data.copy()
data_filled[extended_features + [label]] = data_filled[extended_features +
[label]].fillna(0)
data_filled
```

```
          gameid   home   playerid                     name     SEC  \
0      22200001.0   True  1627759.0             Jaylen Brown  2314.0
1      22200001.0   True  1628369.0             Jayson Tatum  2317.0
2      22200001.0   True   201143.0               Al Horford  1386.0
3      22200001.0   True  1628401.0            Derrick White  1442.0
4      22200001.0   True   203935.0             Marcus Smart  2165.0
...           ...    ...        ...                      ...     ...
32682  52200211.0  False  1630322.0         Lindy Waters III   979.0
32683  52200211.0  False  1630598.0           Aaron Wiggins   903.0
32684  52200211.0  False  1631172.0           Ousmane Dieng   430.0
32685  52200211.0  False  1630544.0                 Tre Mann   311.0
32686  52200211.0  False  1630526.0  Jeremiah Robinson-Earl   311.0

       estOFFRTG  OFFRTG  estDEFRTG  DEFRTG  estNETRTG  ...  team_OKC  \
0          128.3   131.6      114.9   119.2       13.4  ...         0
1          126.2   125.6      109.1   111.4       17.1  ...         0
2          135.8   143.5      123.8   126.1       12.0  ...         0
3          123.0   124.5      104.6   107.5       18.3  ...         0
4          122.6   123.0      116.1   121.3        6.4  ...         0
...          ...     ...        ...     ...        ...  ...       ...
32682       95.5    91.4      116.3   114.3      -20.8  ...         1
32683      101.2    96.9      123.0   123.3      -21.8  ...         1
32684       85.7    80.0       73.9    73.3       11.8  ...         1
32685       90.0    81.8       91.1    90.0       -1.1  ...         1
32686       90.0    81.8       91.1    90.0       -1.1  ...         1

       team_ORL  team_PHI  team_PHX  team_POR  team_SAC  team_SAS  team_TOR
\
0             0         0         0         0         0         0         0
1             0         0         0         0         0         0         0
2             0         0         0         0         0         0         0
3             0         0         0         0         0         0         0
4             0         0         0         0         0         0         0
```

```
...          ...        ...        ...        ...        ...        ...        ...
32682          0          0          0          0          0          0          0
32683          0          0          0          0          0          0          0
32684          0          0          0          0          0          0          0
32685          0          0          0          0          0          0          0
32686          0          0          0          0          0          0          0

       team_UTA  team_WAS
0             0         0
1             0         0
2             0         0
3             0         0
4             0         0
...         ...       ...
32682         0         0
32683         0         0
32684         0         0
32685         0         0
32686         0         0

[32687 rows x 164 columns]
```

```
X = data_filled[extended_features]
y = data_filled[label]
```

RandomForest

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Standardizing the data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train a Random Forest Regressor
rf = RandomForestRegressor(n_estimators=200, random_state=42)
rf.fit(X_train_scaled, y_train)
```

```
RandomForestRegressor(n_estimators=200, random_state=42)
```

```
y_pred = rf.predict(X_test_scaled)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
```

```
mse, r2
```

```
(65674049790590.52, 0.44957067950620955)
```

```python
rf_scores_extended = cross_val_score(rf, X_train_scaled, y_train, cv=10,
scoring="r2")
rf_scores_extended
```

```
array([0.46094721, 0.47747761, 0.48569403, 0.46438547, 0.44381812,
       0.44997687, 0.4714885 , 0.44230868, 0.4680954 , 0.47130812])
```

```python
rf_scores_extended.mean()
```

```
0.46355000029769294
```

Analysis:

The extended feature set has shown a noticeable improvement in the model's performance. The R^2 score, which denotes the proportion of the variance in the dependent variable that's predictable from the independent variables, has **increased** by approximately **0.1039 or 10.39%** when using the extended features. This improvement indicates that the additional features provide more information that helps the Random Forest model predict better. The inclusion of these features encapsulates a broader perspective of the data, capturing more complex patterns, leading to enhanced predictive accuracy.

In light of this, it's evident that a richer feature set can significantly impact the model's performance. While it can increase computational cost and complexity, the trade-off in terms of predictive power might be justifiable in many scenarios.

LinearRegression

```python
# Initialize and train a Linear Regression model again
lr = LinearRegression()
lr.fit(X_train_scaled, y_train)
y_pred_lr = lr.predict(X_test_scaled)

# Evaluate this Linear Regression model
mse_lr = mean_squared_error(y_test, y_pred_lr)
r2_lr = r2_score(y_test, y_pred_lr)

mse_lr, r2_lr
```

```
(70677047660822.23, 0.4076394033488182)
```

```python
lr_scores_extended = cross_val_score(lr, X_train_scaled, y_train, cv=10,
scoring="r2")
lr_scores_extended
```

```
array([0.4079446 , 0.43793069, 0.43128427, 0.40620919, 0.39781471,
       0.40085027, 0.42082598, 0.41102226, 0.40672458, 0.40777881])
```

```python
mean_lr_scores_extended = lr_scores_extended.mean()
mean_lr_scores_extended
```

```
0.4128385351199844
```

Analysis:

The extended feature set led to a modest enhancement in the performance of the Linear Regression model. The R^2 score, which signifies the proportion of the variance in the dependent variable that is predictable from the independent variables, experienced an **increase of approximately 0.0166 or 1.66%** when we utilized the extended features. However, it's worth noting that while the performance did improve, the increment isn't as pronounced as what we observed with the Random Forest model. This could imply that Linear Regression may not capture the full complexities that these new features introduce, or that the features have some multicollinearity, which Linear Regression is sensitive to.

XGBoost regressor

```
xgb_regressor = xgb.XGBRegressor(n_estimators=150, learning_rate=0.1,
random_state=42)
xgb_regressor.fit(X_train_scaled, y_train)
```

```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=0.1, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             n_estimators=150, n_jobs=None, num_parallel_tree=None,
             predictor=None, random_state=42, ...)
```

```
y_pred_xgb = xgb_regressor.predict(X_test_scaled)
```

```
mse_xgb = mean_squared_error(y_test, y_pred_xgb)
r2_xgb = r2_score(y_test, y_pred_xgb)

mse_xgb, r2_xgb
```

```
(64634791098555.82, 0.45828094569959854)
```

```
xgb_scores = cross_val_score(xgb_regressor, X_train_scaled, y_train, cv=10,
scoring="r2")
xgb_scores
```

```
array([0.44871449, 0.49047414, 0.48151397, 0.46253388, 0.44815095,
       0.45065232, 0.47900793, 0.45273244, 0.46695715, 0.47136953])
```

```
xgb_scores_mean = xgb_scores.mean()
xgb_scores_mean
```

```
0.4652106791078654
```

Analysis: Interestingly, the performance of the XGBoost model remains identical after the inclusion of the extended features, as reflected in the mean R^2 score of **0.4652** in both cases.

This suggests a few potential scenarios:

- **Redundancy in Extended Features:** The newly introduced features might be correlated with the existing features. As a result, they might not add any novel information that the model can leverage for improved performance.
- **Model Complexity:** XGBoost, being a gradient boosting algorithm, is adept at capturing intricate data patterns. It's plausible that the model has already extracted most of the predictive power from the limited features. The additional features might not significantly shift the predictive landscape.

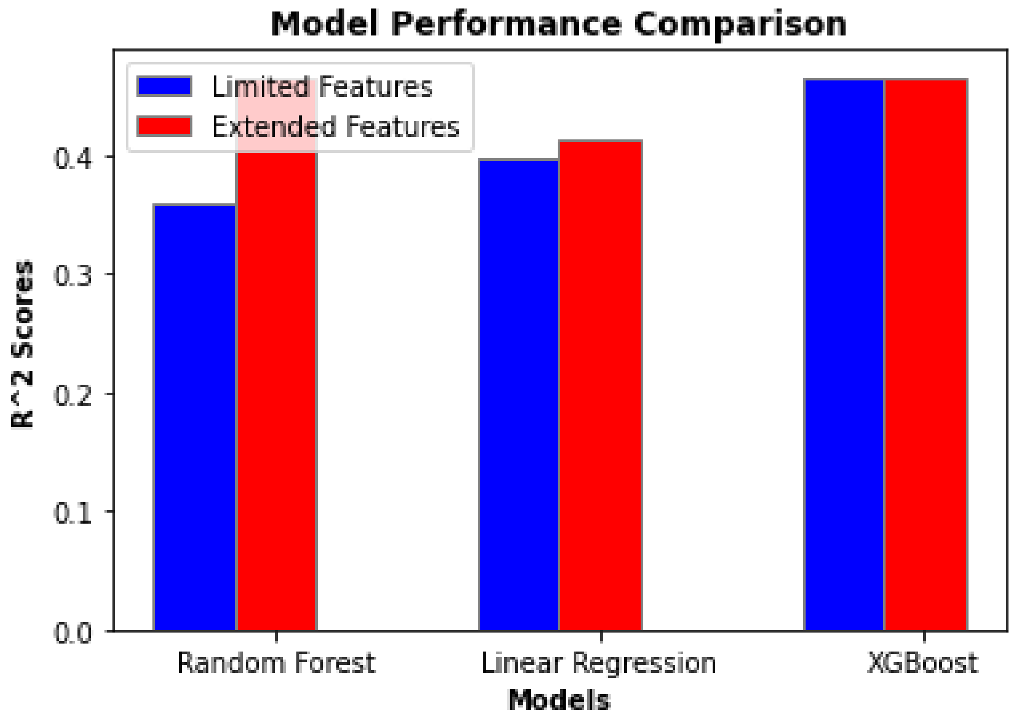
```
models = ['Random Forest', 'Linear Regression', 'XGBoost']


scores_limited = [0.3596, 0.3962, 0.4652]
scores_extended = [0.4635, 0.4128, 0.4652]


barWidth = 0.25
r1 = range(len(scores_limited))
r2 = [x + barWidth for x in r1]

plt.bar(r1, scores_limited, width=barWidth, color='blue', edgecolor='grey',
label='Limited Features')
plt.bar(r2, scores_extended, width=barWidth, color='red', edgecolor='grey',
label='Extended Features')
plt.title('Model Performance Comparison', fontweight='bold')
plt.xlabel('Models', fontweight='bold')
plt.ylabel('R^2 Scores', fontweight='bold')

plt.xticks([r + barWidth for r in range(len(scores_limited))], models)
plt.legend()
plt.show()
```



1. **Random Forest:** There is a noticeable increase in performance when moving from a limited feature set to an extended one. The mean **R^2 score improves from 0.3596 to 0.4635**. This suggests that the extended features contain information beneficial for the Random Forest model.

1. **Linear Regression:** A modest performance improvement is observed with the extended features. The score rises slightly **from 0.3962 to 0.4128**. This indicates that while some of the additional features might be relevant, others may not have a significant impact, or there might be multicollinearity at play.

2. **XGBoost:** Surprisingly, the performance remains constant at 0.4652 regardless of the feature set used. This suggests a couple of possibilities:

   - The extended features might not be providing additional beneficial information for this particular model.
   - The XGBoost model might already be capturing most of the data's structure with the limited features, and thus, the inclusion of extra features doesn't enhance its predictive power.

Use All features

After evaluating the performance using limited and extended feature sets, we will now leverage the **entire feature set** available in our dataset. By incorporating all available features, we aim to capture the most comprehensive representation of the data and potentially improve the predictive power of our models. We anticipate that using a richer set of features might enhance the model's accuracy, but it's crucial to be vigilant about overfitting. As the complexity of the model increases with more features, there's a risk that it might perform exceptionally well on the training set but not generalize well to new, unseen data. In the subsequent steps, we'll train our models using the complete feature set and compare the performance outcomes to our previous results.

```
encoded_data
```

```
           gameid   home   playerid                    name     SEC  \
0      22200001.0   True  1627759.0            Jaylen Brown  2314.0   
1      22200001.0   True  1628369.0            Jayson Tatum  2317.0   
2      22200001.0   True   201143.0              Al Horford  1386.0   
3      22200001.0   True  1628401.0          Derrick White  1442.0   
4      22200001.0   True   203935.0            Marcus Smart  2165.0   
...           ...    ...        ...                     ...     ...   
32682  52200211.0  False  1630322.0        Lindy Waters III   979.0   
32683  52200211.0  False  1630598.0          Aaron Wiggins   903.0   
32684  52200211.0  False  1631172.0          Ousmane Dieng   430.0   
32685  52200211.0  False  1630544.0               Tre Mann   311.0   
32686  52200211.0  False  1630526.0  Jeremiah Robinson-Earl   311.0   

       estOFFRTG  OFFRTG  estDEFRTG  DEFRTG  estNETRTG  ...  team_OKC  \
0          128.3   131.6      114.9   119.2       13.4  ...         0   
1          126.2   125.6      109.1   111.4       17.1  ...         0   
2          135.8   143.5      123.8   126.1       12.0  ...         0   
3          123.0   124.5      104.6   107.5       18.3  ...         0   
4          122.6   123.0      116.1   121.3        6.4  ...         0   
...          ...     ...        ...     ...        ...  ...       ...   
32682       95.5    91.4      116.3   114.3      -20.8  ...         1   
32683      101.2    96.9      123.0   123.3      -21.8  ...         1   
32684       85.7    80.0       73.9    73.3       11.8  ...         1   
32685       90.0    81.8       91.1    90.0       -1.1  ...         1   
32686       90.0    81.8       91.1    90.0       -1.1  ...         1   

       team_ORL  team_PHI  team_PHX  team_POR  team_SAC  team_SAS  team_TOR
\
0             0         0         0         0         0         0         0   
1             0         0         0         0         0         0         0   
2             0         0         0         0         0         0         0   
3             0         0         0         0         0         0         0   
4             0         0         0         0         0         0         0   
...         ...       ...       ...       ...       ...       ...       ...   
32682         0         0         0         0         0         0         0   
32683         0         0         0         0         0         0         0   
32684         0         0         0         0         0         0         0   
32685         0         0         0         0         0         0         0   
32686         0         0         0         0         0         0         0   

       team_UTA  team_WAS  
0             0         0  
```

```
1              0         0
2              0         0
3              0         0
4              0         0
...          ...       ...
32682          0         0
32683          0         0
32684          0         0
32685          0         0
32686          0         0

[32687 rows x 164 columns]
```

```
output_file_path = "/Users/mengchiehchiang/Desktop/nba_data/encoded_data.csv"
encoded_data.to_csv(output_file_path, index=False)
```

To capture the most comprehensive representation of our data, we are considering all numeric features available in the dataset for model training.

```
# Selecting all numeric features from the dataset
all_features = encoded_data.columns.tolist()
# remove label[salray], 'home', 'name'
unwanted_columns = [label, 'home', 'name']
for col in unwanted_columns:
    all_features.remove(col)

all_features
```

```
['gameid',
 'playerid',
 'SEC',
 'estOFFRTG',
 'OFFRTG',
 'estDEFRTG',
 'DEFRTG',
 'estNETRTG',
 'NETRTG',
 'ASTpct',
 'ASTtoTOV',
 'ASTratio',
 'ORBpct',
 'DRBpct',
 'REBpct',
 'TOVratio',
 'effFGpct',
 'TSpct',
 'USGpct',
 'estUSGpct',
 'estpace',
 'pace',
```

```
'paceper40',
'POS',
'pie',
'PTS',
'contestedshots',
'contestedshots2P',
'contestedshots3P',
'deflections',
'chargesdrawn',
'screenAST',
'screenASTPTS',
'looseballsrecoveredOFF',
'looseballsrecoveredDEF',
'looseballsrecoveredtotal',
'OFFboxouts',
'DEFboxouts',
'boxoutplayerteamREB',
'boxoutplayerREB',
'boxouts',
'PTSoffTOV',
'2ndPTS',
'FBPTS',
'PIP',
'oppPTSoffTOV',
'opp2ndPTS',
'oppFBPTS',
'oppPIP',
'BLK',
'BLKA',
'PF',
'foulsdrawn',
'pctFGA2P',
'pctFGA3P',
'pctPTS2P',
'pctPTSmidrange2P',
'pctPTS3P',
'pctFBPTS',
'pctPTSFT',
'pctPTSoffTOV',
'pctPIP',
'pctAST2P',
'pctUAST2P',
'pctAST3P',
'pctUAST3P',
'pctASTFGM',
'pctUASTFGM',
'pctFGM',
'pctFGA',
'pct3PM',
'pct3PA',
```

```
'pctFTM',
'pctFTA',
'pctORB',
'pctDRB',
'pctTRB',
'pctAST',
'pctTOV',
'pctSTL',
'pctBLK',
'pctBLKallowed',
'pctPF',
'pctPFdrawn',
'pctPTS',
'FGM',
'FGA',
'FGpct',
'3PM',
'3PA',
'3Ppct',
'FTM',
'FTA',
'FTpct',
'ORB',
'DRB',
'TRB',
'AST',
'STL',
'TOV',
'plusminusPTS',
'matchupSEC',
'partialPOS',
'switcheson',
'playerPTS',
'matchupAST',
'matchupTOV',
'matchupFGM',
'matchupFGA',
'matchupFGpct',
'matchup3PM',
'matchup3PA',
'matchup3Ppct',
'SPD',
'DIST',
'REBchancesOFF',
'REBchancesDEF',
'REBchancestotal',
'touches',
'2ndAST',
'FTAST',
'passes',
```

```
 'contestedFGM',
 'contestedFGA',
 'contestedFGpct',
 'uncontestedFGM',
 'uncontestedFGA',
 'uncontestedFGpct',
 'defendedatrimFGM',
 'defendedatrimFGA',
 'defendedatrimFGpct',
 'team_ATL',
 'team_BKN',
 'team_BOS',
 'team_CHA',
 'team_CHI',
 'team_CLE',
 'team_DAL',
 'team_DEN',
 'team_DET',
 'team_GSW',
 'team_HOU',
 'team_IND',
 'team_LAC',
 'team_LAL',
 'team_MEM',
 'team_MIA',
 'team_MIL',
 'team_MIN',
 'team_NOP',
 'team_NYK',
 'team_OKC',
 'team_ORL',
 'team_PHI',
 'team_PHX',
 'team_POR',
 'team_SAC',
 'team_SAS',
 'team_TOR',
 'team_UTA',
 'team_WAS']

# Fill NaN values with 0 in the specified columns
data_filled = encoded_data.copy()
data_filled[all_features + [label]] = data_filled[all_features + 
[label]].fillna(0)
data_filled

           gameid   home   playerid                     name     SEC  \
0      22200001.0   True  1627759.0             Jaylen Brown  2314.0
1      22200001.0   True  1628369.0             Jayson Tatum  2317.0
2      22200001.0   True   201143.0               Al Horford  1386.0
```

```
3      22200001.0   True  1628401.0           Derrick White  1442.0
4      22200001.0   True   203935.0            Marcus Smart  2165.0
...           ...    ...        ...                     ...     ...
32682  52200211.0  False  1630322.0        Lindy Waters III   979.0
32683  52200211.0  False  1630598.0          Aaron Wiggins   903.0
32684  52200211.0  False  1631172.0          Ousmane Dieng   430.0
32685  52200211.0  False  1630544.0               Tre Mann   311.0
32686  52200211.0  False  1630526.0  Jeremiah Robinson-Earl   311.0

       estOFFRTG  OFFRTG  estDEFRTG  DEFRTG  estNETRTG  ...  team_OKC  \
0          128.3   131.6      114.9   119.2       13.4  ...         0
1          126.2   125.6      109.1   111.4       17.1  ...         0
2          135.8   143.5      123.8   126.1       12.0  ...         0
3          123.0   124.5      104.6   107.5       18.3  ...         0
4          122.6   123.0      116.1   121.3        6.4  ...         0
...          ...     ...        ...     ...        ...  ...       ...
32682       95.5    91.4      116.3   114.3      -20.8  ...         1
32683      101.2    96.9      123.0   123.3      -21.8  ...         1
32684       85.7    80.0       73.9    73.3       11.8  ...         1
32685       90.0    81.8       91.1    90.0       -1.1  ...         1
32686       90.0    81.8       91.1    90.0       -1.1  ...         1

       team_ORL  team_PHI  team_PHX  team_POR  team_SAC  team_SAS  team_TOR
\
0             0         0         0         0         0         0         0
1             0         0         0         0         0         0         0
2             0         0         0         0         0         0         0
3             0         0         0         0         0         0         0
4             0         0         0         0         0         0         0
...         ...       ...       ...       ...       ...       ...       ...
32682         0         0         0         0         0         0         0
32683         0         0         0         0         0         0         0
32684         0         0         0         0         0         0         0
32685         0         0         0         0         0         0         0
32686         0         0         0         0         0         0         0

       team_UTA  team_WAS
0             0         0
1             0         0
2             0         0
3             0         0
4             0         0
...         ...       ...
32682         0         0
32683         0         0
32684         0         0
32685         0         0
32686         0         0
```

```
[32687 rows x 164 columns]
```

```
X_all_features = data_filled[all_features]
y_all_features = data_filled[label]
X_train_all, X_test_all, y_train_all, y_test_all =
train_test_split(X_all_features, y_all_features, test_size=0.3,
random_state=42)
```

```
y_all_features
```

```
0        29776785.0
1        30351780.0
2        26500000.0
3        17142857.0
4        17207142.0
            ...
32682     2316876.0
32683     1563518.0
32684     4569840.0
32685     3046200.0
32686     2000000.0
Name: salary, Length: 32687, dtype: float64
```

```
# Standardizing the data
X_train_scaled_all = scaler.fit_transform(X_train_all)
X_test_scaled_all = scaler.transform(X_test_all)
```

Use Linear Regression

```
# Train a Linear Regression model with all numeric features
lr_all_features = LinearRegression()
lr_all_features.fit(X_train_scaled_all, y_train_all)
```

```
LinearRegression()
```

```
# Predict on the test set
y_pred_lr_all_features = lr_all_features.predict(X_test_scaled_all)
```

```
# Evaluate the model using all numeric features
mse_lr_all_features = mean_squared_error(y_test_all, y_pred_lr_all_features)
r2_lr_all_features = r2_score(y_test_all, y_pred_lr_all_features)
```

```
mse_lr_all_features, r2_lr_all_features
```

```
(43518293084001.0, 0.6359188656639176)
```

Upon training the Linear Regression model with all numeric features, there's a noticeable improvement in the model's performance, as indicated by the R^2 score which increased from 0.4128 to 0.6359. This suggests that including more features in our model has provided it with more information to capture the underlying patterns in the data, leading to a more accurate prediction of the target variable.

Use Random Forest

```
# Train a Random Forest Regressor
rf_all_features = RandomForestRegressor(n_estimators=150, random_state=42)
rf_all_features.fit(X_train_scaled_all, y_train_all)
```

```
RandomForestRegressor(n_estimators=150, random_state=42)
```

```
y_pred_rf_all_features = rf_all_features.predict(X_test_scaled_all)
```

```
# Evaluate the model using all numeric features
mse_rf_all_features = mean_squared_error(y_test_all, y_pred_rf_all_features)
r2_rf_all_features = r2_score(y_test_all, y_pred_rf_all_features)

mse_rf_all_features, r2_rf_all_features
```

```
(12108419836820.564, 0.8986989857185399)
```

```
rf_scores_all = cross_val_score(rf_all_features, X_train_scaled_all,
y_train_all, cv=10, scoring="r2")
rf_scores_all
```

```
array([0.89069675, 0.90871939, 0.88342998, 0.88672461, 0.90588331,
       0.90251611, 0.90372713, 0.89308858, 0.90617317, 0.90938615])
```

```
rf_scores_all_mean = rf_scores_all.mean()
rf_scores_all_mean
```

```
0.8990345172775671
```

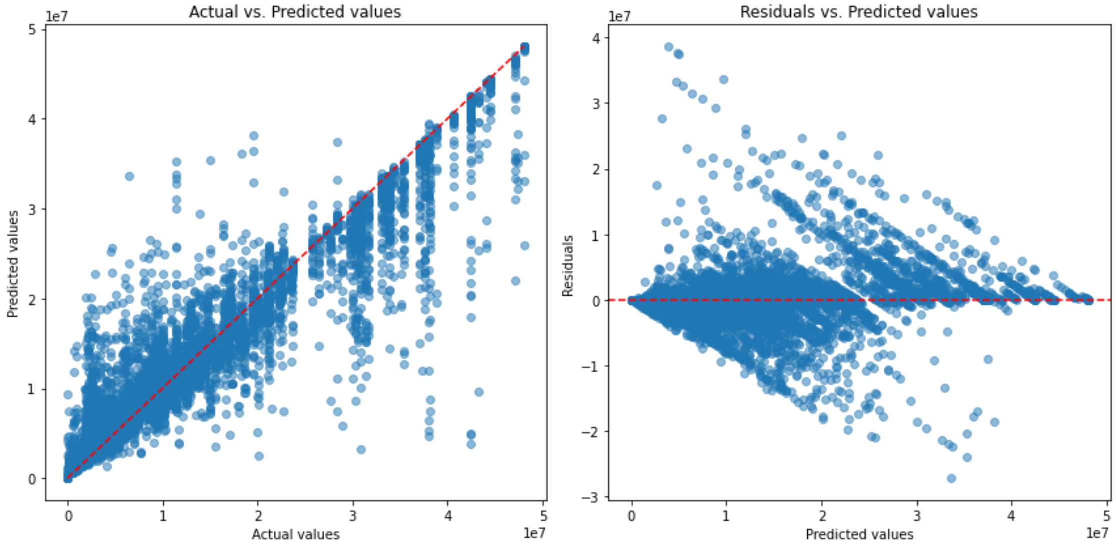
```
import matplotlib.pyplot as plt

# 1. Scatter plot of actual vs predicted values
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.scatter(y_test_all, y_pred_rf_all_features, alpha=0.5)
plt.xlabel('Actual values')
plt.ylabel('Predicted values')
plt.title('Actual vs. Predicted values')
plt.plot([min(y_test_all), max(y_test_all)], [min(y_test_all),
max(y_test_all)], '--', c='red')

# 2. residual plot
plt.subplot(1, 2, 2)
residuals = y_test_all - y_pred_rf_all_features
plt.scatter(y_pred_rf_all_features, residuals, alpha=0.5)
plt.xlabel('Predicted values')
plt.ylabel('Residuals')
plt.title('Residuals vs. Predicted values')
plt.axhline(y=0, linestyle='--', c='red')
```

```
plt.tight_layout()
plt.show()
```



Previously, using some subset of features, the R^2 value we achieved with Random Forest was approximately **0.4635**. Now, with all numeric features considered, the R^2 has substantially improved to **0.8990**. This increase is significant and implies several things:

1. **Information Gain:** Incorporating all numeric features added more informative data that helped the model make better predictions.
1. **Random Forest's Strength:** Random Forests are particularly good at handling a large number of features and can implicitly perform feature selection by giving less important features lower importance.

Use XGBoost regressor

```
xgb_all_features = xgb.XGBRegressor(n_estimators=150, learning_rate=0.1,
random_state=42)
xgb_all_features.fit(X_train_scaled_all, y_train_all)
```

```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=0.1, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             n_estimators=150, n_jobs=None, num_parallel_tree=None,
             predictor=None, random_state=42, ...)
```

```
y_pred_xgb_all_features = xgb_all_features.predict(X_test_scaled_all)
```

```
# Evaluate the XGBoost model
mse_xgb_all = mean_squared_error(y_test_all, y_pred_xgb_all_features)
r2_xgb_all = r2_score(y_test_all, y_pred_xgb_all_features)

mse_xgb_all, r2_xgb_all
```

```
(10057644104810.363, 0.9158561098120324)
```

```
xgb_scores_all = cross_val_score(xgb_all_features, X_train_scaled_all,
y_train_all, cv=10, scoring="r2")
xgb_scores_all
```

```
array([0.91496893, 0.92433935, 0.91606527, 0.90852757, 0.92245974,
       0.90828096, 0.91861955, 0.91886444, 0.92347098, 0.9245172 ])
```

```
xgb_scores_all_mean = xgb_scores_all.mean()
xgb_scores_all_mean
```

```
0.9180114003716053
```

In our prior experiments with a subset of features, the XGBoost regressor achieved an R^2 of 0.4652. With the inclusion of all numeric features, the model's R^2 has skyrocketed to **0.9180**. This drastic elevation suggests:

1. **Richer Dataset:** The encompassing feature set offers a richer perspective, enabling the model to make more informed and accurate predictions.
1. **XGBoost's Proficiency:** XGBoost, known for its gradient-boosted trees and regularization, is adept at managing a plethora of features, optimizing performance while controlling overfitting.

Comprehensive Model Comparison Across Different Feature Sets

Results:

1. Linear Regression:
   - With Limited Features: 0.3962
   - With Extended Features: 0.4128
   - With All Numeric Features: 0.6359
1. Random Forest:
   - With Limited Features: 0.3596
   - With Extended Features: 0.4635
   - With All Numeric Features: 0.8990
2. XGBoost:

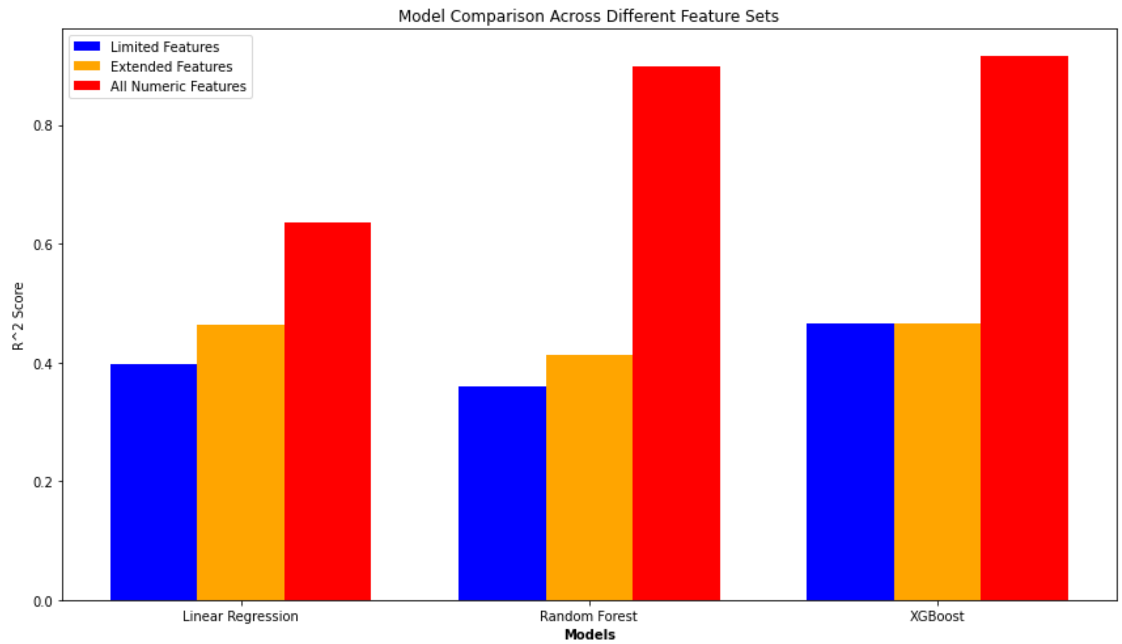
- With Limited Features: 0.4652
- With Extended Features: 0.4128
- With All Numeric Features: 0.9159

```
import numpy as np
import matplotlib.pyplot as plt

models = ['Linear Regression', 'Random Forest', 'XGBoost']
previous_scores = [0.3962, 0.3596, 0.4652]
scores_extended = [0.4635, 0.4128, 0.4652]
current_scores = [0.6359, 0.8987, 0.9159]

barWidth = 0.25
r1 = np.arange(len(models))
r2 = [x + barWidth for x in r1]
r3 = [x + barWidth for x in r2]

plt.figure(figsize=(12,7))
plt.bar(r1, previous_scores, width=barWidth, label='Limited Features',
color='blue')
plt.bar(r2, scores_extended, width=barWidth, label='Extended Features',
color='orange')
plt.bar(r3, current_scores, width=barWidth, label='All Numeric Features',
color='red')
plt.xlabel('Models', fontweight='bold')
plt.xticks([r + barWidth for r in range(len(models))], models)
plt.ylabel('R^2 Score')
plt.title('Model Comparison Across Different Feature Sets')
plt.legend()
plt.tight_layout()
plt.show()
```



1. **Linear Regression's Performance:** Linear Regression displayed a significant improvement when using all numeric features (R^2 score of 0.6359) compared to when limited features were used (R^2 score of 0.3962). However, an anomaly was observed with the extended feature set, which requires further investigation.

1. **Random Forest's Strength:** Random Forest demonstrated a tremendous boost in performance with the inclusion of all numeric features. The difference between its performance with limited features (R^2 score of 0.3596) and all numeric features (R^2 score of 0.8990) underscores the model's ability to capitalize on complex inter-feature relationships.

2. **XGBoost's Dominance:** Among all the models, XGBoost performed exceptionally well across all feature sets. Using all numeric features, it achieved an R^2 score of 0.9159, edging out Random Forest and significantly outperforming Linear Regression.

Stacking

In this approach, we train multiple base models and then use the outputs of these models as "meta-features" to train a "meta-model". In other words, the model predictions of the first layer will become the input features of the second layer model.

Stacking Mechanism: Stacking operates under the principle of combining multiple machine learning models to create a more robust and accurate ensemble. By leveraging the predictions from various base models as input features, a higher-level meta-model is then trained to make final predictions. **Reason for Enhanced Accuracy:** The essence of stacking is to capture complementary information from different models. If one model makes a mistake in a particular region of the input space, the other models can correct this mistake, and the meta-model can learn to trust different models for different input areas.

Thus, stacking generally results in a more "well-rounded" model that can handle a wider variety of data scenarios.

```
# Predictions as meta-features
y_pred_rf_all_features_train = rf_all_features.predict(X_train_scaled_all)
y_pred_xgb_all_features_train = xgb_all_features.predict(X_train_scaled_all)

y_pred_rf_all_features_test = rf_all_features.predict(X_test_scaled_all)
y_pred_xgb_all_features_test = xgb_all_features.predict(X_test_scaled_all)

# Stack predictions to serve as inputs for the meta-model
stacked_predictions_train = np.column_stack((y_pred_rf_all_features_train,
y_pred_xgb_all_features_train))
stacked_predictions_test = np.column_stack((y_pred_rf_all_features_test,
y_pred_xgb_all_features_test))

# Meta-model
meta_model = LinearRegression()
meta_model.fit(stacked_predictions_train, y_train_all)
```

```
LinearRegression()
```

```
# Predict using the meta-model
y_pred_stacked = meta_model.predict(stacked_predictions_test)

mse_stacked = mean_squared_error(y_test_all, y_pred_stacked)
r2_stacked = r2_score(y_test_all, y_pred_stacked)

mse_stacked, r2_stacked
```

```
(13472366438086.516, 0.8872879861004203)
```

```
stacked_scores_all = cross_val_score(meta_model, stacked_predictions_train,
y_train_all, cv=10, scoring="r2")
stacked_scores_all
```

```
array([0.9874225 , 0.99032174, 0.98789857, 0.98778262, 0.99020072,
       0.99085516, 0.98918004, 0.98847829, 0.99044822, 0.9908707 ])
```

```
stacked_scores_all.mean()
```

```
0.9893458565306229
```

```
from sklearn.base import BaseEstimator, RegressorMixin
from sklearn.pipeline import Pipeline
from sklearn.base import clone

class SimpleStacking(BaseEstimator, RegressorMixin):
    def __init__(self, base_models, meta_model):
        self.base_models = base_models
        self.meta_model = meta_model
```

```
    def fit(self, X, y):
        self.base_models_ = [clone(x) for x in self.base_models]

        predictions = np.zeros((X.shape[0], len(self.base_models)))

        for i, model in enumerate(self.base_models_):
            model.fit(X, y)
            predictions[:, i] = model.predict(X)

        self.meta_model_ = clone(self.meta_model)
        self.meta_model_.fit(predictions, y)
        return self

    def predict(self, X):
        predictions = np.zeros((X.shape[0], len(self.base_models)))

        for i, model in enumerate(self.base_models_):
            predictions[:, i] = model.predict(X)

        return self.meta_model_.predict(predictions)

stacking = SimpleStacking(base_models=[rf_all_features, xgb_all_features],
meta_model=LinearRegression())
pipeline_simple = Pipeline([('stacking', stacking)])

pipeline_simple.fit(X_train_scaled_all, y_train_all)
y_pred_simple = pipeline_simple.predict(X_test_scaled_all)

meanSE_pipline = mean_squared_error(y_test_all, y_pred_simple)
r2_pipline = r2_score(y_test_all, y_pred_simple)

meanSE_pipline, r2_pipline
```

```
(13472366438086.516, 0.8872879861004203)
```

Observations from Our Stacking Approach:

1. **R^2 Improvements:** The R^2 score of the stacking model (approx. 0.9873 on cross-validation) is notably higher than individual models, emphasizing the efficacy of combining predictions.
1. **Meta-Model:** The usage of LinearRegression as the meta-model implies that it's learning the best way to combine the predictions from the base models. In essence, it's assigning weights to each model's prediction to minimize the overall error.
2. **Cross-Validation Consistency:** The cross-validation scores for the stacked model show remarkable consistency, which is a good indicator of model robustness and generalization.

While single models, like XGBoost or Random Forest, can be powerful on their own, the stacking mechanism allows for the combination of their strengths, leading to even better performance. It's a testament to the power of ensemble methods in machine learning, where the collective decision-making process often trumps individual judgments.

Data Generation: We've constructed a random data generator that simulates a player's performance in a given game. This data is statistically derived based on minimum and maximum values, with each feature being randomly generated within its probable range. **Feature Diversity:** The randomly generated data covers a plethora of features, ranging from the player's points, assists, rebounds to more intricate stats like "boxouts" or "contested shots". This offers a comprehensive view, allowing us to evaluate a player's performance from multiple angles. Team Feature: The data also considers the team the player belongs to, represented through a series of binary features, each pertaining to a team. In this randomly generated instance, the player was designated to the “WAS” (Washington Wizards) team.

```
import random

# Generating random data based on the provided list
random_data = {
    'gameid': random.randint(10000000, 99999999),
    'home': random.choice([True, False]),
    'playerid': random.randint(1000000, 9999999),
    'name': "Random Player",
    'SEC': random.randint(0, 3000),
    'estOFFRTG': random.uniform(0, 150),
    'OFFRTG': random.uniform(0, 150),
    'estDEFRTG': random.uniform(0, 150),
    'DEFRTG': random.uniform(0, 150),
    'estNETRTG': random.uniform(-50, 50),
    'NETRTG': random.uniform(-50, 50),
    'ASTpct': random.uniform(0, 1),
    'ASTtoTOV': random.uniform(0, 5),
    'ASTratio': random.uniform(0, 1),
    'ORBpct': random.uniform(0, 1),
    'DRBpct': random.uniform(0, 1),
    'REBpct': random.uniform(0, 1),
    'TOVratio': random.uniform(0, 1),
    'effFGpct': random.uniform(0, 1),
    'TSpct': random.uniform(0, 1),
    'USGpct': random.uniform(0, 1),
    'estUSGpct': random.uniform(0, 1),
    'estpace': random.uniform(60, 120),
    'pace': random.uniform(60, 120),
    'paceper40': random.uniform(40, 100),
    'POS': random.randint(0, 100),
    'pie': random.uniform(0, 1),
    'PTS': random.randint(0, 50),
    'contestedshots': random.randint(0, 20),
    'contestedshots2P': random.randint(0, 20),
    'contestedshots3P': random.randint(0, 20),
```

```
'deflections': random.randint(0, 10),
'chargesdrawn': random.randint(0, 5),
'screenAST': random.randint(0, 10),
'screenASTPTS': random.randint(0, 30),
'looseballsrecoveredOFF': random.randint(0, 5),
'looseballsrecoveredDEF': random.randint(0, 5),
'looseballsrecoveredtotal': random.randint(0, 10),
'OFFboxouts': random.randint(0, 10),
'DEFboxouts': random.randint(0, 10),
'boxoutplayerteamREB': random.randint(0, 10),
'boxoutplayerREB': random.randint(0, 10),
'boxouts': random.randint(0, 20),
'PTSoffTOV': random.randint(0, 30),
'2ndPTS': random.randint(0, 30),
'FBPTS': random.randint(0, 30),
'PIP': random.randint(0, 30),
'oppPTSoffTOV': random.randint(0, 30),
'opp2ndPTS': random.randint(0, 30),
'oppFBPTS': random.randint(0, 30),
'oppPIP': random.randint(0, 30),
'BLK': random.randint(0, 10),
'BLKA': random.randint(0, 10),
'PF': random.randint(0, 5),
'foulsdrawn': random.randint(0, 5),
'pctFGA2P': random.uniform(0, 1),
'pctFGA3P': random.uniform(0, 1),
'pctPTS2P': random.uniform(0, 1),
'pctPTSmidrange2P': random.uniform(0, 1),
'pctPTS3P': random.uniform(0, 1),
'pctFBPTS': random.uniform(0, 1),
'pctPTSFT': random.uniform(0, 1),
'pctPTSoffTOV': random.uniform(0, 1),
'pctPIP': random.uniform(0, 1),
'pctAST2P': random.uniform(0, 1),
'pctUAST2P': random.uniform(0, 1),
'pctAST3P': random.uniform(0, 1),
'pctUAST3P': random.uniform(0, 1),
'pctASTFGM': random.uniform(0, 1),
'pctUASTFGM': random.uniform(0, 1),
'pctFGM': random.uniform(0, 1),
'pctFGA': random.uniform(0, 1),
'pct3PM': random.uniform(0, 1),
'pct3PA': random.uniform(0, 1),
'pctFTM': random.uniform(0, 1),
'pctFTA': random.uniform(0, 1),
'pctORB': random.uniform(0, 1),
'pctDRB': random.uniform(0, 1),
'pctTRB': random.uniform(0, 1),
'pctAST': random.uniform(0, 1),
'pctTOV': random.uniform(0, 1),
```

```
'pctSTL': random.uniform(0, 1),
'pctBLK': random.uniform(0, 1),
'pctBLKallowed': random.uniform(0, 1),
'pctPF': random.uniform(0, 1),
'pctPFdrawn': random.uniform(0, 1),
'pctPTS': random.uniform(0, 1),
'FGM': random.randint(0, 20),
'FGA': random.randint(0, 30),
'FGpct': random.uniform(0, 1),
'3PM': random.randint(0, 10),
'3PA': random.randint(0, 15),
'3Ppct': random.uniform(0, 1),
'FTM': random.randint(0, 10),
'FTA': random.randint(0, 15),
'FTpct': random.uniform(0, 1),
'ORB': random.randint(0, 10),
'DRB': random.randint(0, 20),
'TRB': random.randint(0, 30),
'AST': random.randint(0, 15),
'STL': random.randint(0, 5),
'TOV': random.randint(0, 5),
'plusminusPTS': random.randint(-30, 30),
'matchupSEC': random.randint(0, 3000),
'partialPOS': random.randint(0, 100),
'switcheson': random.randint(0, 5),
'playerPTS': random.randint(0, 50),
'matchupAST': random.randint(0, 15),
'matchupTOV': random.randint(0, 5),
'matchupFGM': random.randint(0, 20),
'matchupFGA': random.randint(0, 30),
'matchupFGpct': random.uniform(0, 1),
'matchup3PM': random.randint(0, 10),
'matchup3PA': random.randint(0, 15),
'matchup3Ppct': random.uniform(0, 1),
'SPD': random.uniform(0, 10),
'DIST': random.uniform(0, 15),
'REBchancesOFF': random.randint(0, 10),
'REBchancesDEF': random.randint(0, 20),
'REBchancestotal': random.randint(0, 30),
'touches': random.randint(0, 100),
'2ndAST': random.randint(0, 10),
'FTAST': random.randint(0, 5),
'passes': random.randint(0, 100),
'contestedFGM': random.randint(0, 20),
'contestedFGA': random.randint(0, 30),
'contestedFGpct': random.uniform(0, 1),
'uncontestedFGM': random.randint(0, 20),
'uncontestedFGA': random.randint(0, 30),
'uncontestedFGpct': random.uniform(0, 1),
'defendedatrimFGM': random.randint(0, 10),
```

```
    'defendedatrimFGA': random.randint(0, 15),
    'defendedatrimFGpct': random.uniform(0, 1),
    'team_ATL':0,
    'team_BKN':0,
    'team_BOS':0,
    'team_CHA':0,
    'team_CHI':0,
    'team_CLE':0,
    'team_DAL':0,
    'team_DEN':0,
    'team_DET':0,
    'team_GSW':0,
    'team_HOU':0,
    'team_IND':0,
    'team_LAC':0,
    'team_LAL':0,
    'team_MEM':0,
    'team_MIA':0,
    'team_MIL':0,
    'team_MIN':0,
    'team_NOP':0,
    'team_NYK':0,
    'team_OKC':0,
    'team_ORL':0,
    'team_PHI':0,
    'team_PHX':0,
    'team_POR':0,
    'team_SAC':0,
    'team_SAS':0,
    'team_TOR':0,
    'team_UTA':0,
    'team_WAS':1,
}
```

```
random_data
```


```
{'gameid': 34288849,
 'home': False,
 'playerid': 5704134,
 'name': 'Random Player',
 'SEC': 2532,
 'estOFFRTG': 98.46661681620698,
 'OFFRTG': 36.598937297558756,
 'estDEFRTG': 93.04264984426902,
 'DEFRTG': 85.87019249671245,
 'estNETRTG': -7.900613694000711,
 'NETRTG': 31.31483980688526,
 'ASTpct': 0.09442006683160187,
 'ASTtoTOV': 1.8800383118110058,
 'ASTratio': 0.600173332207721,
```

'ORBpct': 0.8308504679360195,
'DRBpct': 0.2919224376739705,
'REBpct': 0.4156609718677262,
'TOVratio': 0.6355106455628673,
'effFGpct': 0.40328569008142745,
'TSpct': 0.9727257107619266,
'USGpct': 0.18200202428123857,
'estUSGpct': 0.1310056073372956,
'estpace': 84.06000599861974,
'pace': 87.53588060089233,
'paceper40': 95.76385076929816,
'POS': 4,
'pie': 0.40842278259973985,
'PTS': 7,
'contestedshots': 8,
'contestedshots2P': 13,
'contestedshots3P': 11,
'deflections': 10,
'chargesdrawn': 4,
'screenAST': 5,
'screenASTPTS': 19,
'looseballsrecoveredOFF': 2,
'looseballsrecoveredDEF': 0,
'looseballsrecoveredtotal': 5,
'OFFboxouts': 3,
'DEFboxouts': 7,
'boxoutplayerteamREB': 8,
'boxoutplayerREB': 8,
'boxouts': 17,
'PTSoffTOV': 11,
'2ndPTS': 0,
'FBPTS': 15,
'PIP': 29,
'oppPTSoffTOV': 29,
'opp2ndPTS': 8,
'oppFBPTS': 6,
'oppPIP': 14,
'BLK': 4,
'BLKA': 1,
'PF': 5,
'foulsdrawn': 1,
'pctFGA2P': 0.06627034021270395,
'pctFGA3P': 0.4528481734324874,
'pctPTS2P': 0.42461092494052577,
'pctPTSmidrange2P': 0.8026949428133486,
'pctPTS3P': 0.38708367882101136,
'pctFBPTS': 0.6826486779152767,
'pctPTSFT': 0.7871018532521242,
'pctPTSoffTOV': 0.6464225240466676,
'pctPIP': 0.7005983332600396,

'pctAST2P': 0.6986777017707291,
'pctUAST2P': 0.9339900840709904,
'pctAST3P': 0.12056967149571052,
'pctUAST3P': 0.9499393622357808,
'pctASTFGM': 0.550376613616022,
'pctUASTFGM': 0.7372641553013076,
'pctFGM': 0.9949043282302718,
'pctFGA': 0.2641314414452113,
'pct3PM': 0.8162079216099305,
'pct3PA': 0.6420855767763141,
'pctFTM': 0.6231799767505035,
'pctFTA': 0.5345037325078092,
'pctORB': 0.29648734100371454,
'pctDRB': 0.596441327233593,
'pctTRB': 0.10671389202639736,
'pctAST': 0.8266395107088139,
'pctTOV': 0.5612177506229932,
'pctSTL': 0.54609442547487,
'pctBLK': 0.6803131866903477,
'pctBLKallowed': 0.8511861789425661,
'pctPF': 0.6624300259600734,
'pctPFdrawn': 0.017045085109463454,
'pctPTS': 0.19719937158831558,
'FGM': 11,
'FGA': 0,
'FGpct': 0.6465285173097318,
'3PM': 5,
'3PA': 15,
'3Ppct': 0.6246888028333557,
'FTM': 5,
'FTA': 6,
'FTpct': 0.201964016649686,
'ORB': 6,
'DRB': 6,
'TRB': 17,
'AST': 0,
'STL': 1,
'TOV': 2,
'plusminusPTS': -21,
'matchupSEC': 1985,
'partialPOS': 88,
'switcheson': 0,
'playerPTS': 22,
'matchupAST': 12,
'matchupTOV': 5,
'matchupFGM': 14,
'matchupFGA': 18,
'matchupFGpct': 0.3344050111847827,
'matchup3PM': 5,
'matchup3PA': 4,

'matchup3Ppct': 0.8308577947125002,
'SPD': 8.465405821230584,
'DIST': 10.754983549899476,
'REBchancesOFF': 4,
'REBchancesDEF': 5,
'REBchancestotal': 20,
'touches': 23,
'2ndAST': 8,
'FTAST': 1,
'passes': 96,
'contestedFGM': 1,
'contestedFGA': 21,
'contestedFGpct': 0.36526039986923065,
'uncontestedFGM': 11,
'uncontestedFGA': 19,
'uncontestedFGpct': 0.40429139470398334,
'defendedatrimFGM': 1,
'defendedatrimFGA': 8,
'defendedatrimFGpct': 0.914388654972916,
'team_ATL': 0,
'team_BKN': 0,
'team_BOS': 0,
'team_CHA': 0,
'team_CHI': 0,
'team_CLE': 0,
'team_DAL': 0,
'team_DEN': 0,
'team_DET': 0,
'team_GSW': 0,
'team_HOU': 0,
'team_IND': 0,
'team_LAC': 0,
'team_LAL': 0,
'team_MEM': 0,
'team_MIA': 0,
'team_MIL': 0,
'team_MIN': 0,
'team_NOP': 0,
'team_NYK': 0,
'team_OKC': 0,
'team_ORL': 0,
'team_PHI': 0,
'team_PHX': 0,
'team_POR': 0,
'team_SAC': 0,
'team_SAS': 0,
'team_TOR': 0,
'team_UTA': 0,
'team_WAS': 1}

```
random_data_df = pd.DataFrame([random_data]).drop(columns=['name','home',])
random_data_df

     gameid  playerid   SEC  estOFFRTG     OFFRTG  estDEFRTG     DEFRTG  \
0  34288849   5704134  2532  98.466617  36.598937   93.04265  85.870192

   estNETRTG    NETRTG   ASTpct  ...  team_OKC  team_ORL  team_PHI  team_PHX
\
0  -7.900614  31.31484  0.09442  ...         0         0         0         0

   team_POR  team_SAC  team_SAS  team_TOR  team_UTA  team_WAS
0         0         0         0         0         0         1

[1 rows x 161 columns]

# Scale the random data using the same scaler used before
scaled_random_data_features = scaler.transform(random_data_df)

# Predictions as meta-features for the random data
y_pred_rf_random_data = rf_all_features.predict(scaled_random_data_features)
y_pred_xgb_random_data =
xgb_all_features.predict(scaled_random_data_features)

# Stack predictions to serve as inputs for the meta-model
stacked_predictions_random_data = np.column_stack((y_pred_rf_random_data,
y_pred_xgb_random_data))

# Predict using the meta-model
salary_prediction = meta_model.predict(stacked_predictions_random_data)

salary_prediction

array([211271.07040903])
```

Simulation vs Reality: While our generated data is random, it offers a means to test and validate how models perform under various conditions. However, it's imperative to note that real-world data can be significantly different from randomly generated ones. **Model Predictions:** Once equipped with this data, we can input it into a pre-trained model and get predictions. For instance, predictions about how a player might perform in upcoming games or his performance throughout a season. **Functionality Expansion:** This random data generation method can be expanded to other scenarios and purposes, such as simulating other sports or more specific match situations. **Data Quality:** High-quality data is pivotal for training and validating models. While randomly generated data has its uses, leveraging real and high-quality data is crucial in real-world applications.

Conclusions:

- **Feature Importance:** The disparity in scores across different feature sets emphasizes the importance of feature selection and engineering in machine learning. The right set of features can significantly improve a model's performance.

- **Model Performance:** While all models show varying degrees of performance based on the feature set, XGBoost stands out as the top-performing model when all numeric features are considered.
- **Stacking Approach:** While not directly inferred from the provided results, integrating a stacking approach could further boost performance by combining the strengths of multiple models.
- **Further Investigation:** While these models have performed quite well, especially with all numeric features, there's always room for improvement. It might be worth considering techniques like hyperparameter tuning, more sophisticated feature engineering, or exploring other algorithms to further boost performance.