Data 102 Final Project - NBA

By Xinming Li, Bill Tian, Weiqian Peng, Yanyu Chen

Set Up

In [14]:
```
pip install pandas scikit-learn statsmodels
```

```
Requirement already satisfied: pandas in /srv/conda/lib/python3.9/site-packa
ges (1.3.5)
Requirement already satisfied: scikit-learn in /srv/conda/lib/python3.9/site
-packages (1.1.1)
Requirement already satisfied: statsmodels in /srv/conda/lib/python3.9/site-
packages (0.13.5)
Requirement already satisfied: python-dateutil>=2.7.3 in /srv/conda/lib/pyth
on3.9/site-packages (from pandas) (2.8.0)
Requirement already satisfied: pytz>=2017.3 in /srv/conda/lib/python3.9/site
-packages (from pandas) (2021.1)
Requirement already satisfied: numpy>=1.17.3 in /srv/conda/lib/python3.9/sit
e-packages (from pandas) (1.26.2)
Requirement already satisfied: scipy>=1.3.2 in /srv/conda/lib/python3.9/site
-packages (from scikit-learn) (1.10.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /srv/conda/lib/python
3.9/site-packages (from scikit-learn) (3.2.0)
Requirement already satisfied: joblib>=1.0.0 in /srv/conda/lib/python3.9/sit
e-packages (from scikit-learn) (1.3.1)
Requirement already satisfied: packaging>=21.3 in /srv/conda/lib/python3.9/s
ite-packages (from statsmodels) (21.3)
Requirement already satisfied: patsy>=0.5.2 in /srv/conda/lib/python3.9/site
-packages (from statsmodels) (0.5.3)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /srv/conda/lib/py
thon3.9/site-packages (from packaging>=21.3->statsmodels) (3.1.1)
Requirement already satisfied: six in /srv/conda/lib/python3.9/site-packages
(from patsy>=0.5.2->statsmodels) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

In [31]:
```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
import statsmodels.api as sm
from sklearn.metrics import mean_squared_error, r2_score

import seaborn as sns

from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
```

Load the Dataset

In [32]:
```
# Load the dataset
df = pd.read_csv('NBA_Player_Salary_2022_2023.csv')
df = df[df['Salary'] > 1000000]
```

In [33]:
```
df.head()
```

Out[33]:

	Unnamed: 0	Player Name	Salary	Position	Age	Team	GP	GS	MP	FG	...
0	0	Stephen Curry	48070014	PG	34	GSW	56	56	34.7	10.0	...
1	1	John Wall	47345760	PG	32	LAC	34	3	22.2	4.1	...
2	2	Russell Westbrook	47080179	PG	34	LAL/LAC	73	24	29.1	5.9	...
3	3	LeBron James	44474988	PF	38	LAL	55	54	35.5	11.1	...
4	4	Kevin Durant	44119845	PF	34	BRK/PHO	47	47	35.6	10.3	...

5 rows × 52 columns

Feature Engineering

In [34]:
```
# Create new feature: Game Started / Game Played
df['GS/GP'] = df['GS'] / df['GP']

# Handling missing values
df.fillna(df.mean(), inplace=True)

# Handling infinite values
df.replace([np.inf, -np.inf], np.nan, inplace=True)
df.fillna(df.mean(), inplace=True)

df['Position'] = df['Position'].str.split('-').str[0]
```

```
/tmp/ipykernel_267/3473210936.py:5: FutureWarning: Dropping of nuisance colu
mns in DataFrame reductions (with 'numeric_only=None') is deprecated; in a f
uture version this will raise TypeError.  Select only valid columns before c
alling the reduction.
  df.fillna(df.mean(), inplace=True)
/tmp/ipykernel_267/3473210936.py:9: FutureWarning: Dropping of nuisance colu
mns in DataFrame reductions (with 'numeric_only=None') is deprecated; in a f
uture version this will raise TypeError.  Select only valid columns before c
alling the reduction.
  df.fillna(df.mean(), inplace=True)
```

EDA

Salary Distribution

In [41]:
```
# Salary Analysis
# Distribution of salaries
salary_distribution = df['Salary'].describe()

# Comparison of salaries across positions
salary_by_position = df.groupby('Position')['Salary'].describe()
salary_by_age = df.groupby('Position')['Age'].describe()

# Top 10 highest-paid players
top_paid_players = df.sort_values(by='Salary', ascending=False).head(10)[['F
```

In [42]:
```
print("\nSalary Distribution:")
print(salary_distribution)
```

```
Salary Distribution:
count    3.920000e+02
mean     9.955994e+06
std      1.103874e+07
min      1.000001e+06
25%      2.293039e+06
50%      5.092736e+06
75%      1.297000e+07
max      4.807001e+07
Name: Salary, dtype: float64
```

Top 10 Highest-Paid Players

In [43]:
```
print("\nTop 10 Highest-Paid Players:")
print(top_paid_players)
```

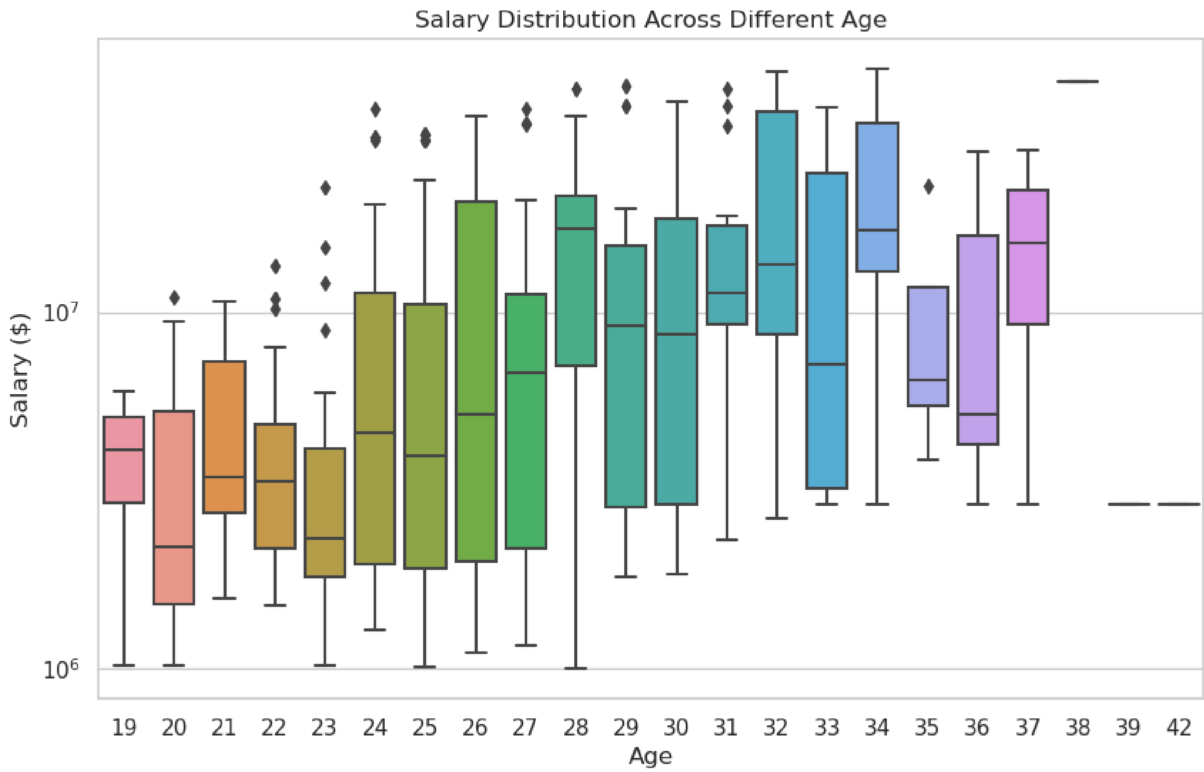
```
Top 10 Highest-Paid Players:
             Player Name    Salary     Team Position
0          Stephen Curry  48070014      GSW       PG
1              John Wall  47345760      LAC       PG
2      Russell Westbrook  47080179  LAL/LAC       PG
3           LeBron James  44474988      LAL       PF
4           Kevin Durant  44119845  BRK/PHO       PF
5           Bradley Beal  43279250      WAS       SG
6          Kawhi Leonard  42492492      LAC       SF
7            Paul George  42492492      LAC       SF
8  Giannis Antetokounmpo  42492492      MIL       PF
9         Damian Lillard  42492492      POR       PG
```

Salary Distribution Across Different Age

In [44]:
```
print("\nSalary_by_age:")
print(salary_by_age)
```

```
plt.figure(figsize=(10, 6))
sns.boxplot(x='Age', y='Salary', data=df)
plt.title('Salary Distribution Across Different Age')
plt.ylabel('Salary ($)')
plt.xlabel('Age')
plt.yscale('log')
plt.show()
```

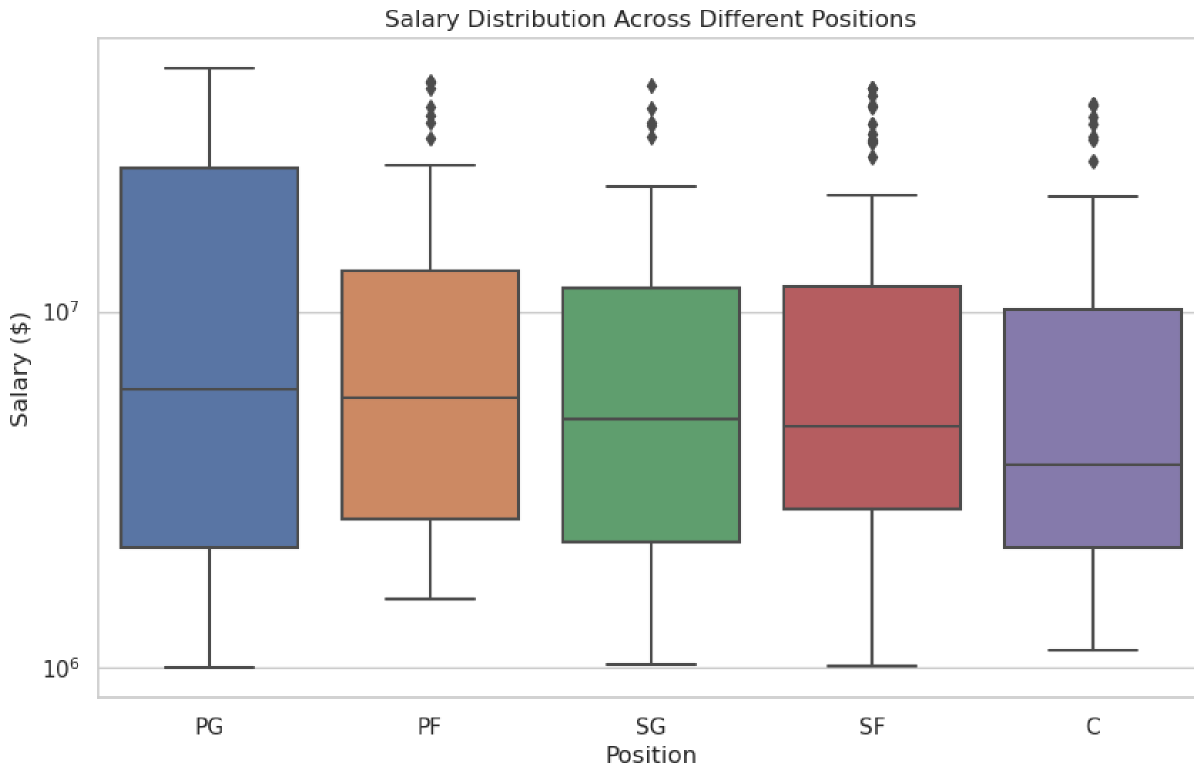
```
Salary_by_age:
          count       mean       std   min   25%   50%   75%   max
Position
C          81.0  26.506173  4.574723  19.0  23.0  25.0  29.0  42.0
PF         73.0  26.931507  5.006465  19.0  23.0  27.0  30.0  39.0
PG         67.0  26.582090  4.668233  19.0  23.0  26.0  30.0  37.0
SF         76.0  25.881579  3.808647  19.0  23.0  25.0  28.0  36.0
SG         95.0  25.084211  4.101539  19.0  22.0  24.0  28.0  36.0
```



Salaries by Position

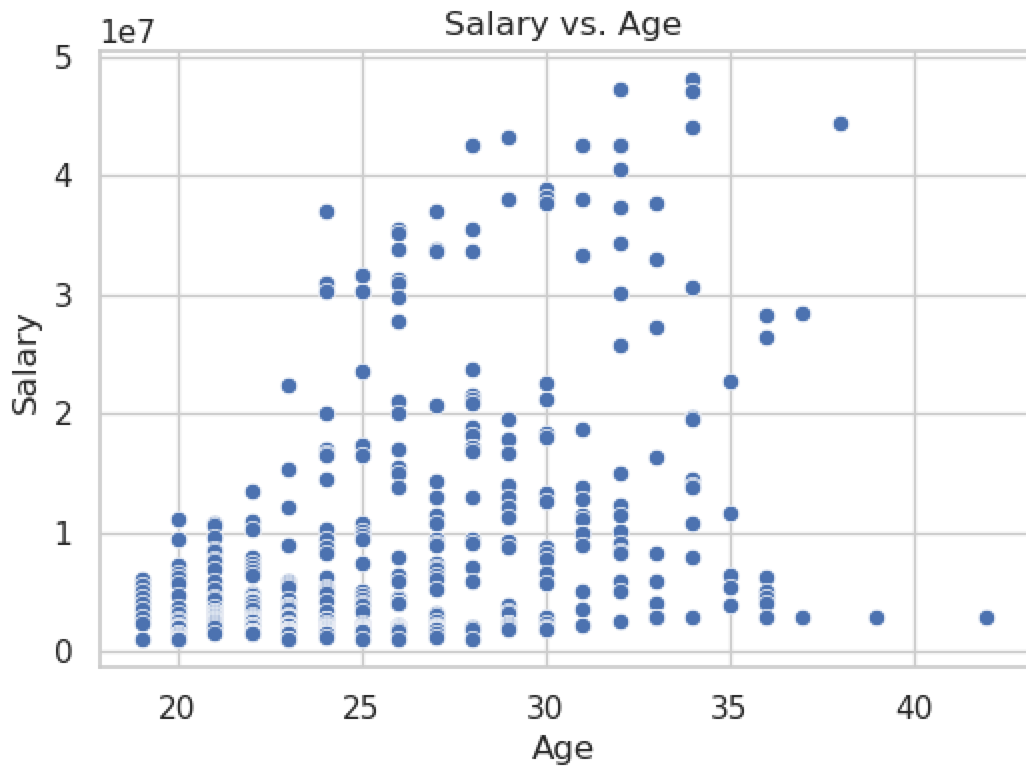
In [45]:
```
print("\nSalaries by Position:")
print(salary_by_position)

plt.figure(figsize=(10, 6))
sns.boxplot(x='Position', y='Salary', data=df)
plt.title('Salary Distribution Across Different Positions')
plt.ylabel('Salary ($)')
plt.xlabel('Position')
plt.yscale('log')  # Using a logarithmic scale due to wide range in salaries
plt.show()
```

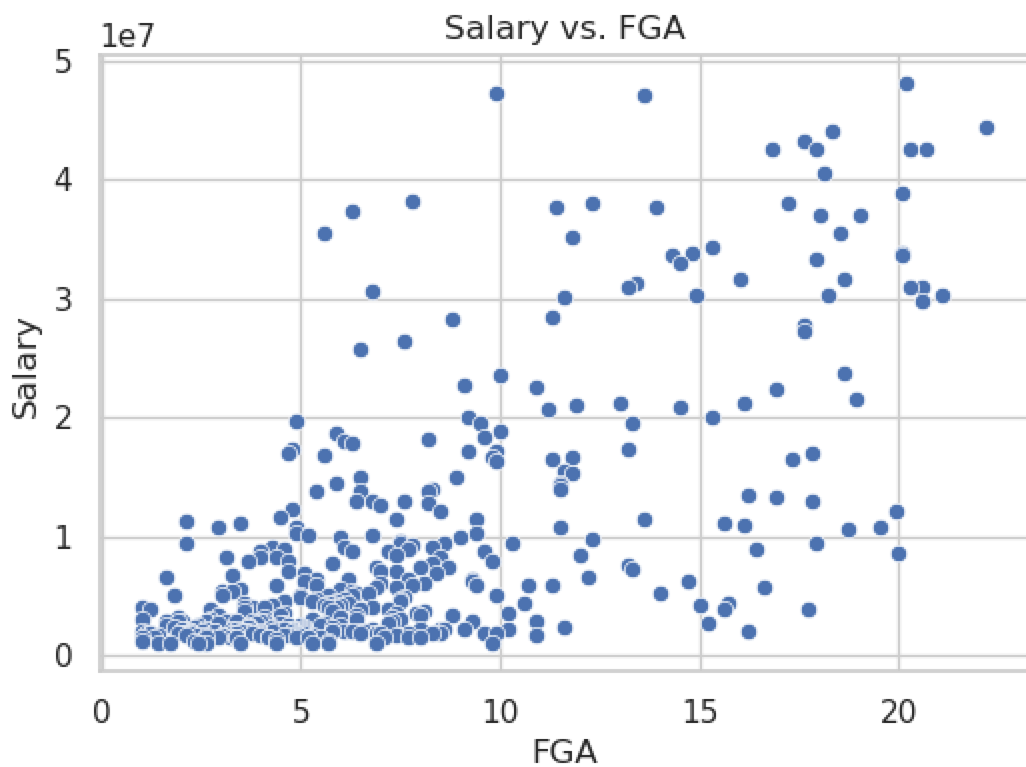
```
Salaries by Position:
          count          mean           std         min         25%         50%  \
Position
C          81.0  8.119211e+06  9.183682e+06  1116112.0  2174880.0  3722040.0
PF         73.0  1.040581e+07  1.116614e+07  1563518.0  2617800.0  5739840.0
PG         67.0  1.389486e+07  1.434398e+07  1000001.0  2180760.0  6025000.0
SF         76.0  9.996459e+06  1.146101e+07  1013119.0  2798160.0  4753920.0
SG         95.0  8.366136e+06  8.580999e+06  1017781.0  2260555.5  5000000.0

                 75%         max
Position
C         10123457.0  38172414.0
PF        13000000.0  44474988.0
PG        25206666.0  48070014.0
SF        11790000.0  42492492.0
SG        11605264.0  43279250.0
```

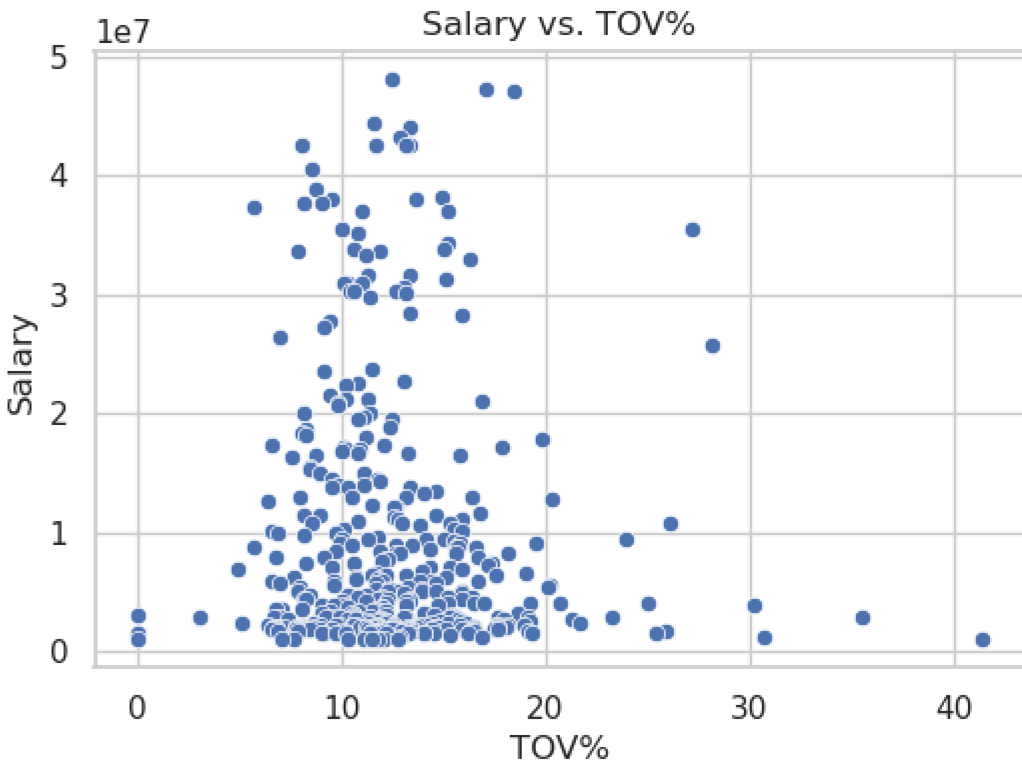


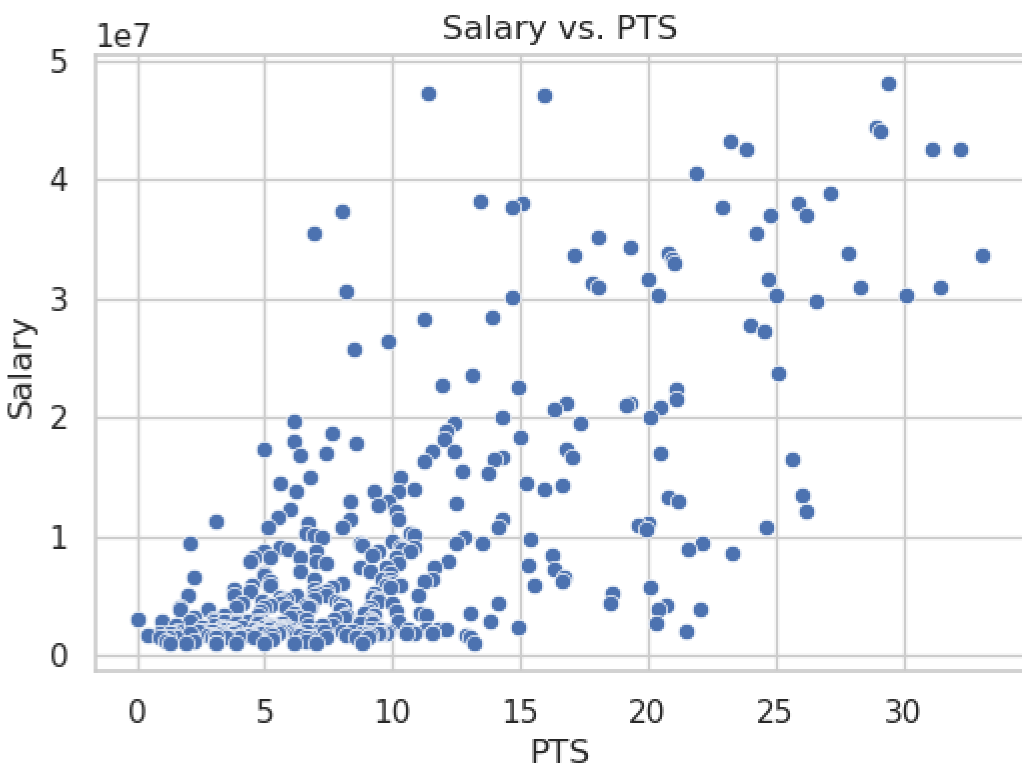
Others

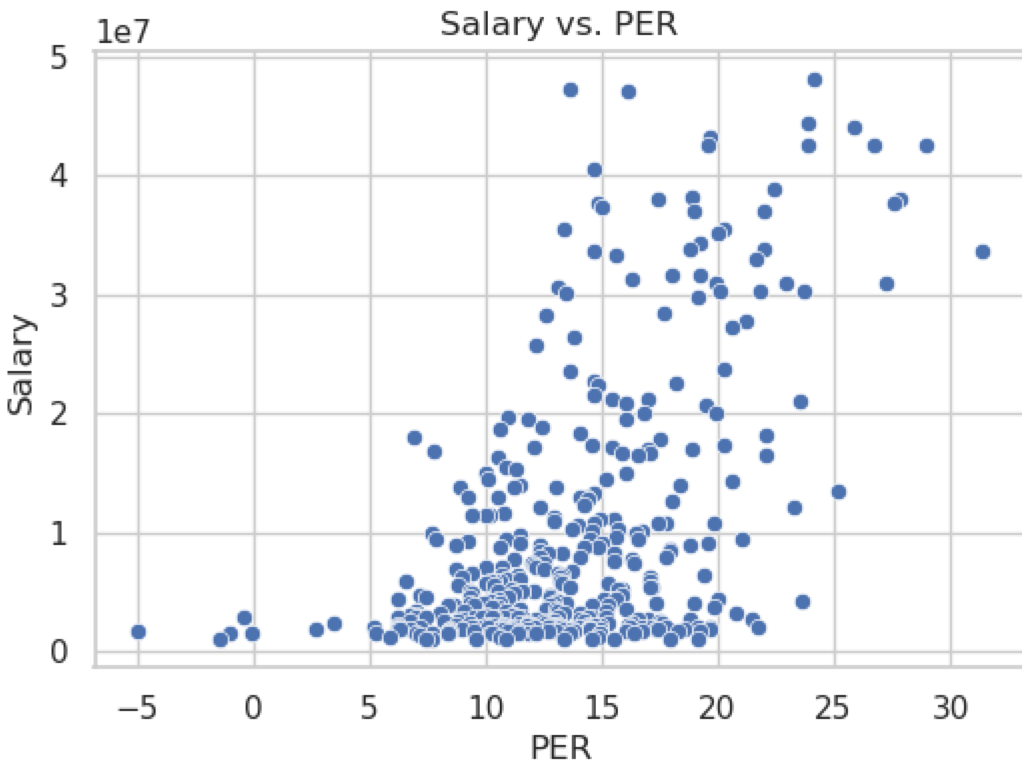
In [46]:
```
sns.set(style="whitegrid")

performance_metrics = ['Age', 'FGA', 'TOV%', 'PTS', 'PER', 'VORP', 'FG%', 'U
for metric in performance_metrics:
    plt.figure(figsize=(6, 4))
    sns.scatterplot(x=df[metric], y=df['Salary'])
    plt.title(f'Salary vs. {metric}')
    plt.xlabel(metric)
    plt.ylabel('Salary')
    plt.show()
```

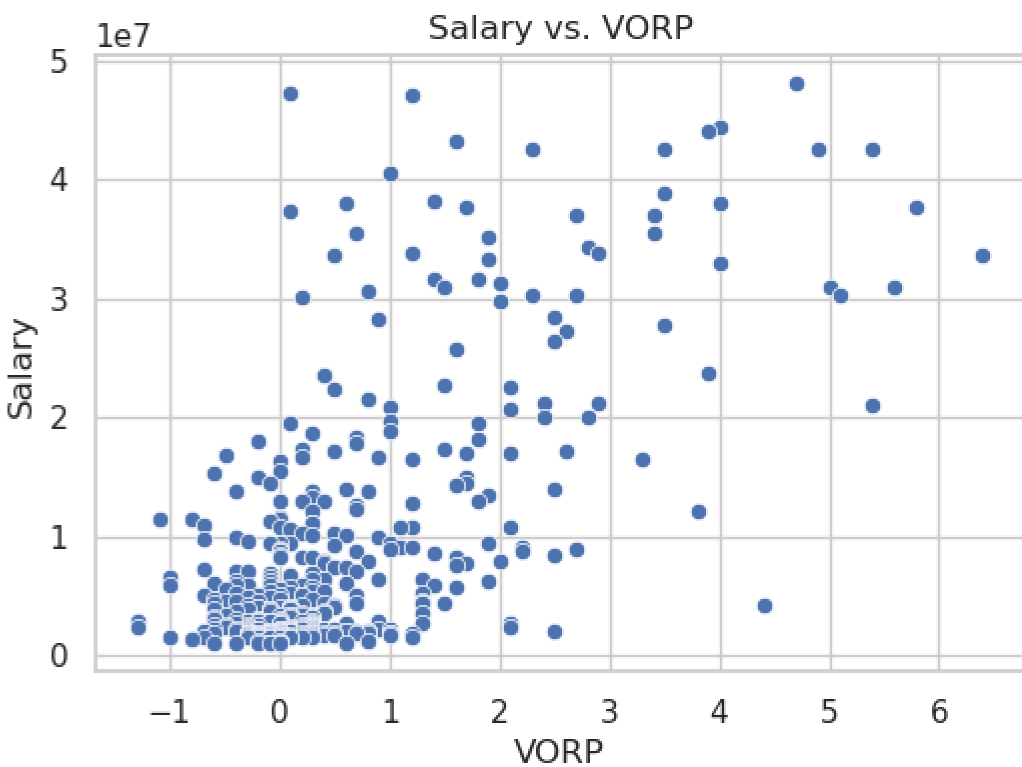

Salary vs. Age
1e7
5
4
3
2
1
0
Salary
20
25
30
35
40
Age

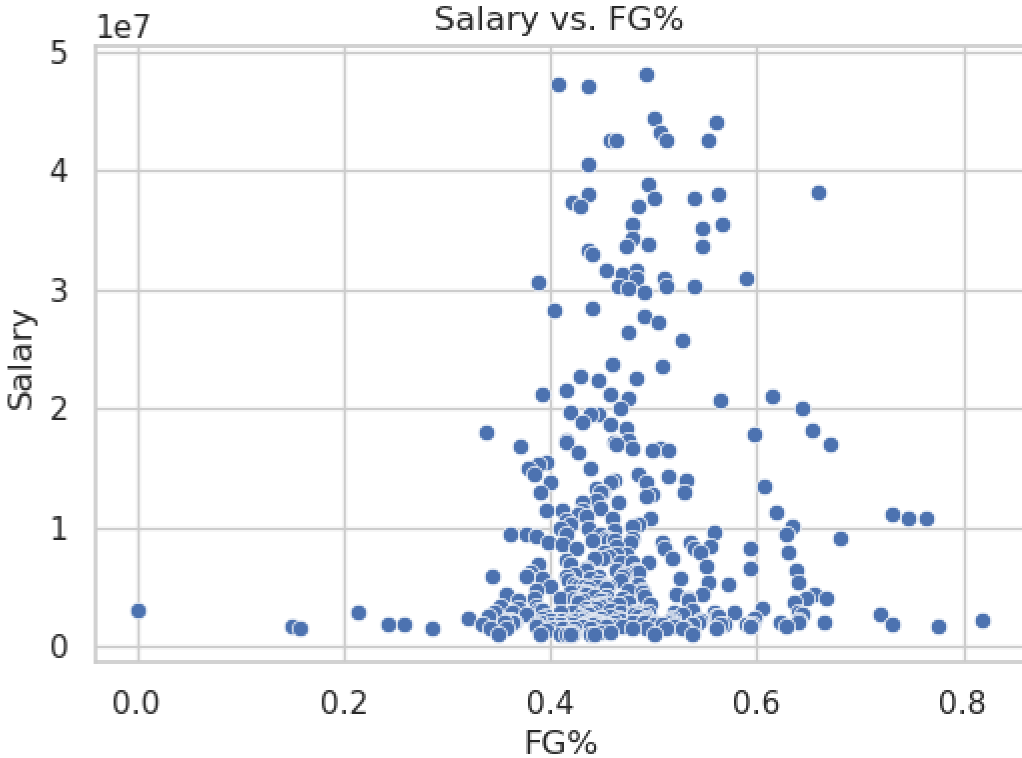

Salary vs. FGA
1e7
5
4
3
2
1
0
Salary
0
5
10
15
20
FGA

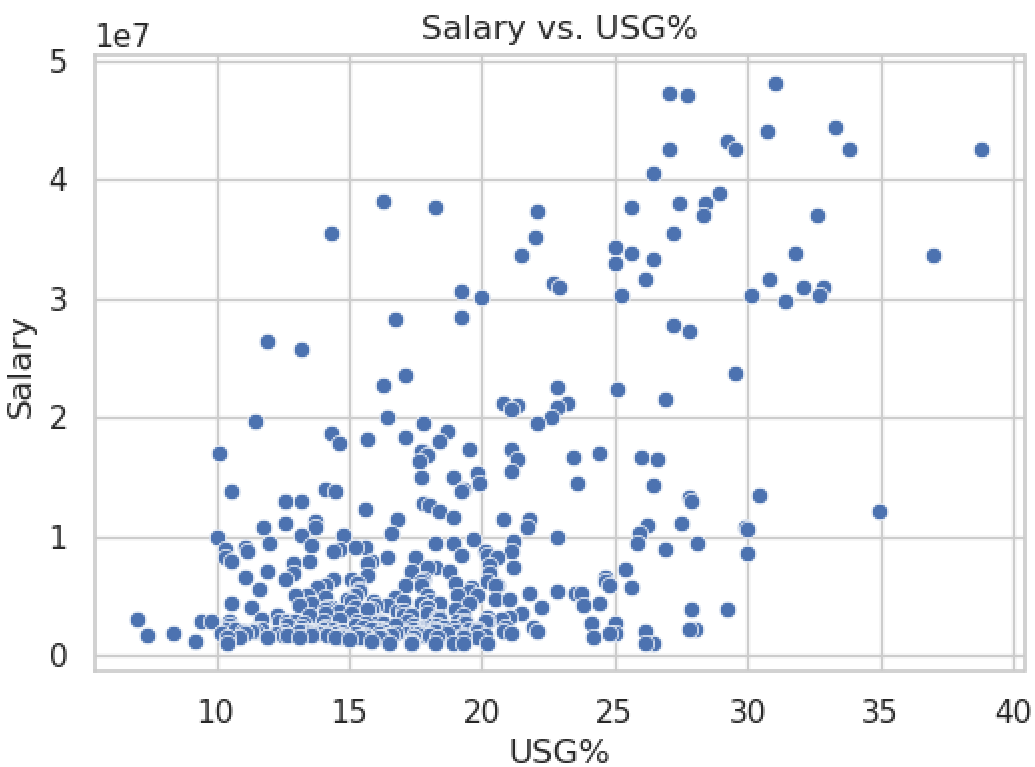

Salary vs. TOV%
1e7
Salary
TOV%
0
1
2
3
4
5
0
10
20
30
40

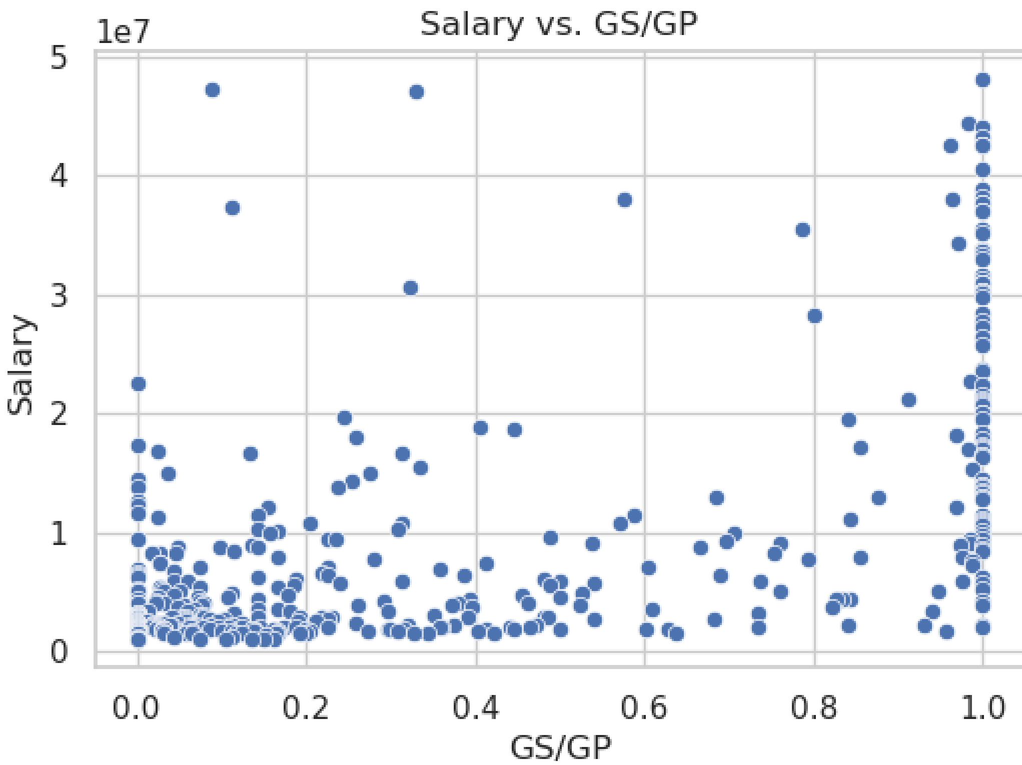

Salary vs. PTS
1e7
Salary
PTS
0
1
2
3
4
5
0
5
10
15
20
25
30


Salary vs. PER
1e7
Salary
PER
5
4
3
2
1
0
−5
0
5
10
15
20
25
30


Salary vs. VORP
1e7
Salary
VORP
5
4
3
2
1
0
−1
0
1
2
3
4
5
6


Salary vs. FG%
1e7
Salary
FG%
0
1
2
3
4
5
0.0
0.2
0.4
0.6
0.8


Salary vs. USG%
1e7
Salary
USG%
0
1
2
3
4
5
10
15
20
25
30
35
40



Feature Selection

In [5]:
```
selected_features = ['Position', 'Age', 'FGA', 'TOV%', 'PTS', 'PER', 'VORP',
X = df[selected_features]
y = df['Salary']
```

Splitting Train and Test Set

In [6]:
```
X_train, X_test, y_train, y_test = train_test_split(df[selected_features], y
```

Applying preprocessor to the dataset

In [7]:
```
# Applying preprocessor to the dataset
preprocessor = ColumnTransformer(
    transformers=[
        ('cat', OneHotEncoder(), ['Position']),
        ('num', StandardScaler(), ['Age', 'FGA', 'TOV%', 'PTS', 'PER', 'VORP
    ])

# Fit and transform the training data
X_train_transformed = preprocessor.fit_transform(X_train)

# Transform the test data
X_test_transformed = preprocessor.transform(X_test)
```

```
# Create a DataFrame from the transformed data with correct column names
one_hot_feature_names = preprocessor.named_transformers_['cat'].get_feature_
all_feature_names = np.concatenate([one_hot_feature_names, ['Age', 'FGA', 'T

X_train_df = pd.DataFrame(X_train_transformed, columns=all_feature_names)
X_test_df = pd.DataFrame(X_test_transformed, columns=all_feature_names)

X_train_df = X_train_df.reset_index(drop=True)
X_test_df = X_test_df.reset_index(drop=True)
y_train = y_train.reset_index(drop=True)
y_test = y_test.reset_index(drop=True)
```

GLM Model

In [8]:
```
# Fit the GLM model using the DataFrame with named columns
glm_model = sm.GLM(y_train, sm.add_constant(X_train_df), family=sm.families.
glm_results = glm_model.fit()

# Predict on the test set and evaluate
y_pred = glm_results.predict(sm.add_constant(X_test_df))
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error:", mse)
print("R-squared:", r2)

# Print the model summary with original feature names
print(glm_results.summary())
```

```
Mean Squared Error: 38244011656143.516
R-squared: 0.7315872068392756
                 Generalized Linear Model Regression Results
============================================================================
==
Dep. Variable:                 Salary   No. Observations:                  2
74
Model:                            GLM   Df Residuals:                      2
60
Model Family:                Gaussian   Df Model:
13
Link Function:               identity   Scale:                    4.4051e+
13
Method:                          IRLS   Log-Likelihood:                 -468
5.6
Date:                Thu, 07 Dec 2023   Deviance:                 1.1453e+
16
Time:                        19:48:26   Pearson chi2:               1.15e+
16
No. Iterations:                     3   Pseudo R-squ. (CS):           0.79
41
Covariance Type:            nonrobust
============================================================================
===
                  coef    std err          z      P>|z|      [0.025      0.9
75]
----------------------------------------------------------------------------
---
const        7.894e+06   3.37e+05     23.427      0.000    7.23e+06    8.55e
+06
Position_C    8.31e+05   1.08e+06      0.768      0.443   -1.29e+06    2.95e
+06
Position_PF  1.395e+06   8.26e+05      1.688      0.091   -2.24e+05    3.01e
+06
Position_PG  2.994e+06   9.75e+05      3.071      0.002    1.08e+06     4.9e
+06
Position_SF  2.528e+06   8.86e+05      2.855      0.004    7.93e+05    4.26e
+06
Position_SG   1.46e+05   8.08e+05      0.181      0.857   -1.44e+06    1.73e
+06
Age          3.781e+06   4.13e+05      9.164      0.000    2.97e+06    4.59e
+06
FGA         -2.581e+05   3.23e+06     -0.080      0.936   -6.58e+06    6.07e
+06
TOV%          5.37e+05   5.08e+05      1.057      0.290   -4.58e+05    1.53e
+06
PTS          2.479e+06    3.5e+06      0.708      0.479   -4.39e+06    9.34e
+06
PER           7.85e+05   1.27e+06      0.620      0.535    -1.7e+06    3.27e
+06
VORP         1.063e+06   9.39e+05      1.132      0.258   -7.78e+05     2.9e
+06
FG%         -5.134e+05    8.6e+05     -0.597      0.550    -2.2e+06    1.17e
+06
USG%         2.016e+06   9.98e+05      2.020      0.043    5.96e+04    3.97e
+06
```

```
GS/GP          2.58e+06   7.98e+05      3.233      0.001    1.02e+06    4.14e
+06
==============================================================================
===
```

Non-Parametric: Random Forest

In [10]:

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Create the Random Forest model
rf_model = RandomForestRegressor(random_state=42)

# Fit the model to the training data
rf_result = rf_model.fit(X_train_df, y_train)

# Predict on the test set
y_pred_rf = rf_model.predict(X_test_df)

# Evaluate the model
mse_rf = mean_squared_error(y_test, y_pred_rf)
r2_rf = r2_score(y_test, y_pred_rf)

print("Random Forest Mean Squared Error:", mse_rf)
print("Random Forest R-squared:", r2_rf)
```

```
Random Forest Mean Squared Error: 28857100989525.09
Random Forest R-squared: 0.7974685514490141
```

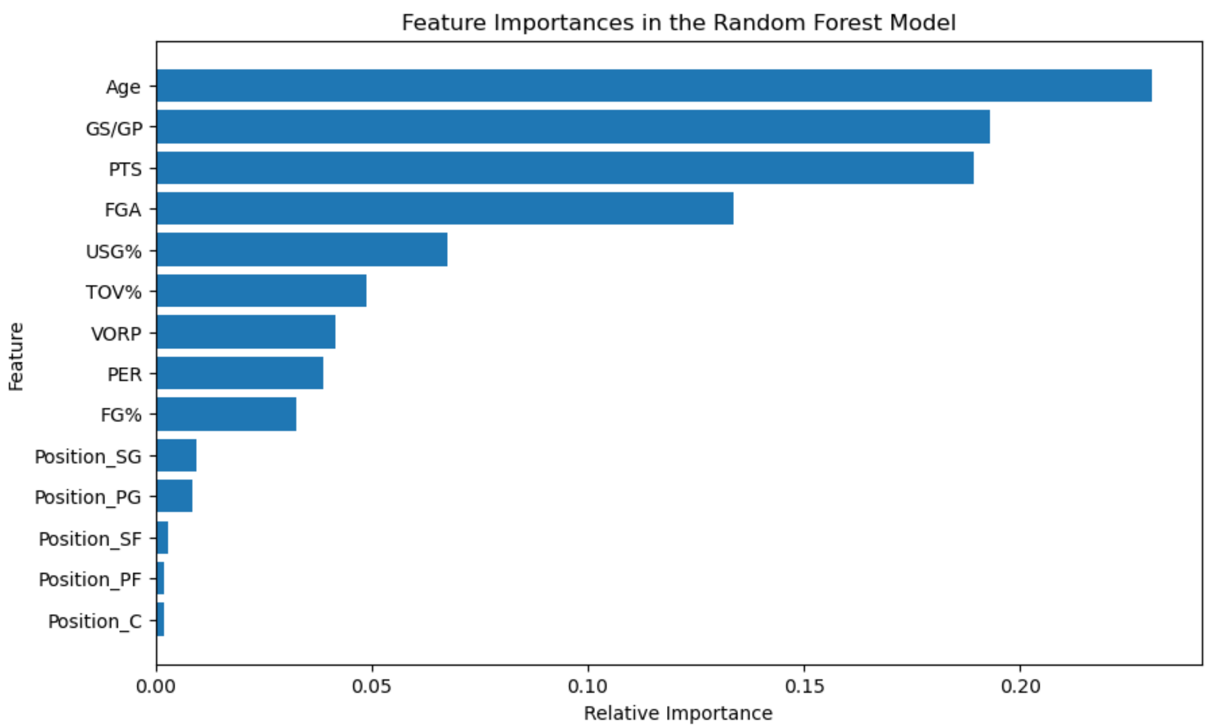
Feature Importance

In [21]:

```
import matplotlib.pyplot as plt

# Predict on the test set
y_pred_rf = rf_model.predict(X_test_df)

# Calculate performance metrics
mse_rf = mean_squared_error(y_test, y_pred_rf)
r2_rf = r2_score(y_test, y_pred_rf)

print("Random Forest Model Summary Statistics")
print("----------------------------------------")
print("Mean Squared Error:", mse_rf)
print("R-squared:", r2_rf)

# Feature Importances
feature_importances = rf_model.feature_importances_

# Create a DataFrame to display feature importances
features_df = pd.DataFrame({'Feature': X_train_df.columns, 'Importance': fea
features_df = features_df.sort_values(by='Importance', ascending=False)

print("\nFeature Importances")
```

```
print(features_df)

# Plotting feature importances
plt.figure(figsize=(10, 6))
plt.title("Feature Importances in the Random Forest Model")
plt.barh(features_df['Feature'], features_df['Importance'])
plt.xlabel('Relative Importance')
plt.ylabel('Feature')
plt.gca().invert_yaxis()  # Invert y-axis to have the most important feature
plt.show()
```

```
Random Forest Model Summary Statistics
--------------------------------------
Mean Squared Error: 28857100989525.09
R-squared: 0.7974685514490141

Feature Importances
         Feature  Importance
5            Age    0.230504
13         GS/GP    0.193022
8            PTS    0.189201
6            FGA    0.133569
12          USG%    0.067521
7           TOV%    0.048677
10          VORP    0.041573
9            PER    0.038855
11           FG%    0.032613
4    Position_SG    0.009246
2    Position_PG    0.008444
3    Position_SF    0.002921
1    Position_PF    0.001984
0     Position_C    0.001870
```



In [ ]: