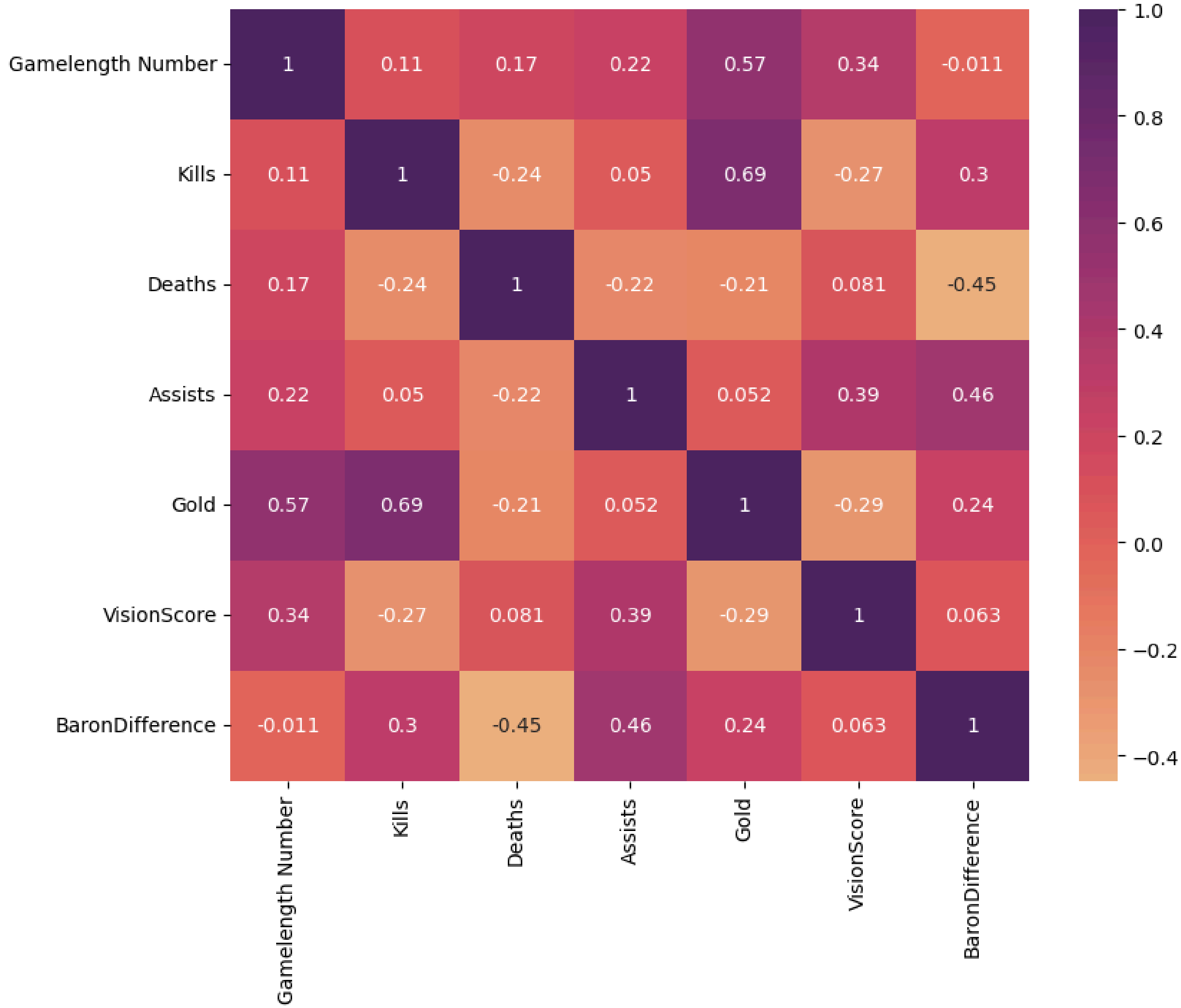
Killstreak Predictor: League of Legends 2024

In [ ]:
```
import numpy as np
import pandas as pd
```

In [6]:
```
data = pd.read_csv("LOL_matchdata_2024.csv")
data = data.iloc[:, :-1]
data = data.dropna()
data = data.drop(columns=['DateTime UTC','Items', 'CS', 'DamageToChampions', '(
```

In [7]:
```
data['BaronDifference'] = np.where(
    data['Team'] == data['Team1'],
    data['Team1Barons'] - data['Team2Barons'],
    np.where(
        data['Team'] == data['Team2'],
        data['Team2Barons'] - data['Team1Barons'], np.nan))
data['TowerDifference'] = np.where(
    data['Team'] == data['Team1'],
    data['Team1Towers'] - data['Team2Towers'],
    np.where(
        data['Team'] == data['Team2'],
        data['Team2Towers'] - data['Team1Towers'],
        np.nan))
data.drop(columns=['Team1Barons', 'Team2Barons', 'Team1Towers', 'Team2Towers',
```

In [8]:
```
import seaborn as sns
import matplotlib.pyplot as plt
numeric_data = data.select_dtypes(include=np.number)
correlation_matrix = numeric_data.corr()
plt.figure(figsize=(9, 7))
sns.heatmap(correlation_matrix, annot=True, cmap='flare')
plt.show()
```



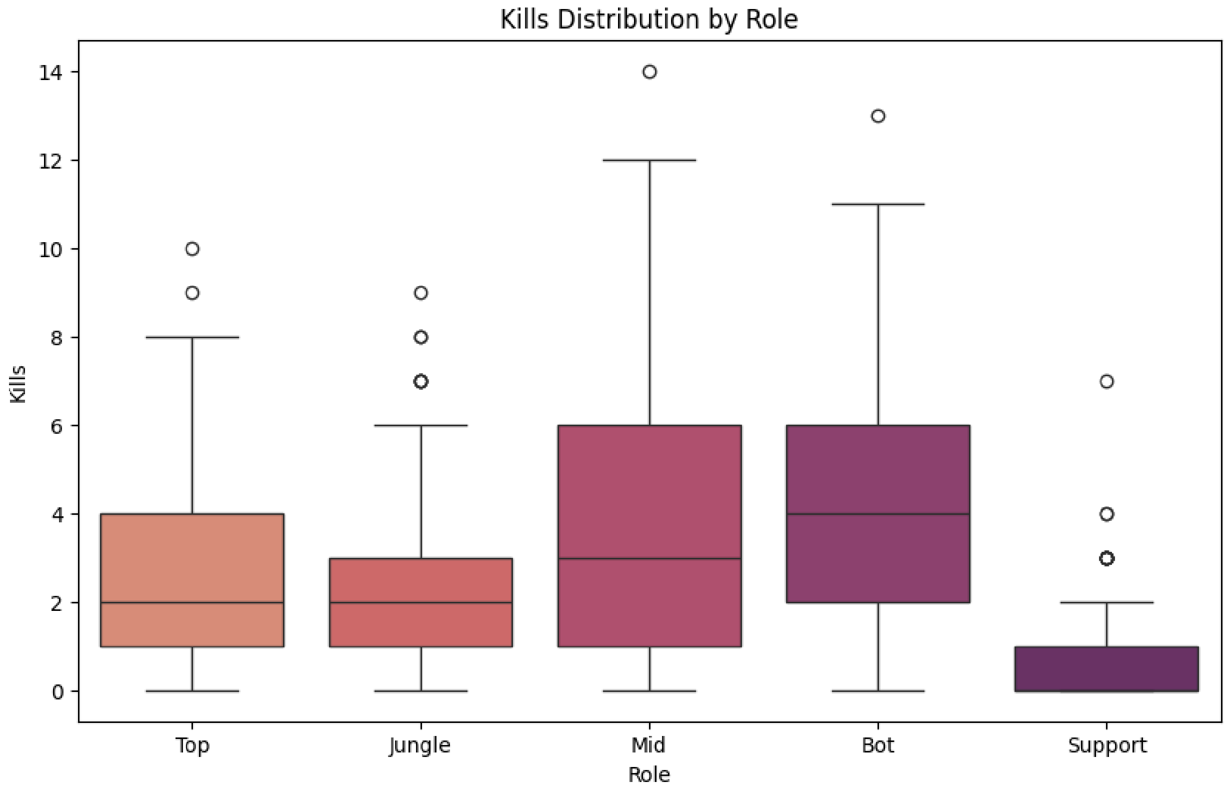
In [9]:

```
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize=(10, 6))
sns.boxplot(x=data['Role'], y=data['Kills'], palette='flare')
plt.title('Kills Distribution by Role')
plt.xlabel('Role')
plt.ylabel('Kills')
plt.show()
```

```
<ipython-input-9-2d0f8a95ca8b>:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.boxplot(x=data['Role'], y=data['Kills'], palette='flare')
```



In [10]:
```
import pandas as pd
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.model_selection import train_test_split
import statsmodels.api as sm

# Separate features (X) and target (y)
target = 'Kills'
X = data.drop(columns=[target])
y = data[target]

# Identify categorical and numeric columns
categorical_features = X.select_dtypes(include=['object']).columns
numeric_features = X.select_dtypes(include=['int64', 'float64']).columns

# Preprocessing: One-hot encoding for categorical and scaling for numeric
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numeric_features),
        ('cat', OneHotEncoder(handle_unknown='ignore', drop ='first'), categor
    ]
)

# Apply preprocessing
X_processed = preprocessor.fit_transform(X)

# Check if the result is sparse, and convert to dense if necessary
if hasattr(X_processed, "toarray"):  # If it's sparse
    X_processed = X_processed.toarray()

# Get column names for the transformed data
# Numeric columns remain unchanged
numeric_columns = numeric_features.tolist()
```

```
# For the one-hot encoded columns, get feature names from OneHotEncoder
categorical_columns = preprocessor.transformers_[1][1].get_feature_names_out(ca

# Combine the column names
column_names = numeric_columns + categorical_columns

# Convert the result to a DataFrame
X_processed_df = pd.DataFrame(X_processed, columns=column_names)

# Ensure the indices of X_processed_df and y are aligned
X_processed_df = X_processed_df.reset_index(drop=True)
y = y.reset_index(drop=True)

# Add constant to X_processed for OLS
X_processed_with_const = sm.add_constant(X_processed_df)

# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X_processed_with_const, y,

# Ensure that the indices of X_train and y_train are aligned
X_train = X_train.reset_index(drop=True)
y_train = y_train.reset_index(drop=True)

# Linear regression using statsmodels
model = sm.OLS(y_train, X_train)
results = model.fit()

# Display the summary
print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  Kills   R-squared:                       0.704
Model:                            OLS   Adj. R-squared:                  0.672
Method:                 Least Squares   F-statistic:                     21.89
Date:                Mon, 30 Dec 2024   Prob (F-statistic):          1.86e-112
Time:                        20:59:48   Log-Likelihood:                -1082.7
No. Observations:                 624   AIC:                             2289.
Df Residuals:                     562   BIC:                             2564.
Df Model:                          61
Covariance Type:            nonrobust
==============================================================================
=================================
                                                 coef    std err          t
P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
---------------------------------
const                                          4.6045      0.887      5.191
0.000       2.862       6.347
Gamelength Number                             -1.8007      0.139    -12.996
0.000      -2.073      -1.529
Deaths                                         0.4314      0.081      5.336
0.000       0.273       0.590
Assists                                       -0.4018      0.093     -4.310
0.000      -0.585      -0.219
Gold                                           3.4273      0.166     20.642
0.000       3.101       3.753
VisionScore                                    0.4281      0.149      2.874
0.004       0.136       0.721
BaronDifference                               -0.2322      0.105     -2.215
0.027      -0.438      -0.026
Role_Jungle                                    0.5942      0.304      1.952
0.051      -0.004       1.192
Role_Mid                                      -0.5079      0.281     -1.810
0.071      -1.059       0.043
Role_Support                                   1.1903      0.475      2.504
0.013       0.257       2.124
Role_Top                                      -0.1367      0.302     -0.452
0.651      -0.731       0.457
Team_Bilibili Gaming                           1.3749      1.288      1.067
0.286      -1.155       3.905
Team_Dplus KIA                                -1.6927      1.227     -1.380
0.168      -4.102       0.717
Team_FlyQuest                                  0.1675      0.654      0.256
0.798      -1.117       1.452
Team_Fnatic                                   -0.9076      1.300     -0.698
0.485      -3.460       1.645
Team_Fukuoka SoftBank HAWKS gaming             1.0517      0.843      1.248
0.213      -0.604       2.707
Team_G2 Esports                               -1.8798      0.723     -2.600
0.010      -3.300      -0.460
Team_GAM Esports                              -1.3348      0.396     -3.368
0.001      -2.113      -0.556
Team_Gen.G                                    -0.6553      0.623     -1.052
0.293      -1.879       0.569
Team_Hanwha Life Esports                       0.0910      0.902      0.101
0.920      -1.681       1.863
Team_LNG Esports                               1.4003      1.189      1.178
0.239      -0.935       3.736
Team_MAD Lions KOI                            -0.7997      1.192     -0.671
```

```
0.502      -3.140       1.541
Team_Movistar R7                                   -1.2926      0.731     -1.768
0.078      -2.728       0.143
Team_PSG Talon                                     -1.2188      0.435     -2.803
0.005      -2.073      -0.365
Team_T1                                            -0.7398      0.617     -1.199
0.231      -1.952       0.472
Team_Team Liquid                                    0.3451      0.660      0.523
0.601      -0.951       1.642
Team_Top Esports                                    1.2425      1.280      0.971
0.332      -1.272       3.757
Team_Vikings Esports (2023 Vietnamese Team)        -0.5049      0.418     -1.209
0.227      -1.325       0.316
Team_Weibo Gaming                                   1.6832      1.342      1.254
0.210      -0.954       4.320
Team_paiN Gaming                                   -0.1288      0.732     -0.176
0.860      -1.567       1.309
PlayerWin_Yes                                       1.5343      0.260      5.899
0.000       1.023       2.045
KeystoneRune_Arcane Comet                          -0.6086      0.328     -1.857
0.064      -1.252       0.035
KeystoneRune_Conqueror                             -0.5802      0.275     -2.113
0.035      -1.119      -0.041
KeystoneRune_Electrocute                            0.1792      0.388      0.462
0.644      -0.582       0.940
KeystoneRune_First Strike                          -1.1139      0.485     -2.298
0.022      -2.066      -0.162
KeystoneRune_Fleet Footwork                        -0.8656      0.331     -2.611
0.009      -1.517      -0.215
KeystoneRune_Glacial Augment                       -0.7360      0.624     -1.179
0.239      -1.962       0.490
KeystoneRune_Grasp of the Undying                  -0.6768      0.339     -1.997
0.046      -1.342      -0.011
KeystoneRune_Guardian                              -0.5173      0.301     -1.720
0.086      -1.108       0.074
KeystoneRune_Hail of Blades                        -0.2956      0.411     -0.720
0.472      -1.102       0.511
KeystoneRune_Lethal Tempo                          -1.5247      0.596     -2.560
0.011      -2.695      -0.355
KeystoneRune_Phase Rush                            -0.5758      0.379     -1.521
0.129      -1.320       0.168
KeystoneRune_Press the Attack                      -1.2073      0.350     -3.446
0.001      -1.896      -0.519
KeystoneRune_Summon Aery                            0.2359      0.364      0.648
0.517      -0.479       0.951
KeystoneRune_Unsealed Spellbook                    -1.0941      1.091     -1.003
0.317      -3.238       1.049
Country_Australia                                  -2.2709      1.012     -2.245
0.025      -4.258      -0.284
Country_Belgium                                    -3.0504      1.029     -2.966
0.003      -5.071      -1.030
Country_Brazil                                     -2.6805      0.816     -3.286
0.001      -4.283      -1.078
Country_Canada                                     -0.5823      1.094     -0.532
0.595      -2.732       1.567
Country_China                                      -4.0360      1.290     -3.128
0.002      -6.571      -1.502
Country_Czech Republic                             -0.9027      1.475     -0.612
0.541      -3.800       1.995
Country_Denmark                                     0.0225      0.455      0.049
```

```
0.961      -0.871      0.916
Country_France                                      -0.8118      0.484     -1.677
0.094      -1.763      0.139
Country_Germany                                      0.0950      0.467      0.203
0.839      -0.823      1.013
Country_Iraq                                        -1.7671      1.016     -1.740
0.082      -3.762      0.228
Country_Japan                                       -2.8921      1.232     -2.347
0.019      -5.313     -0.472
Country_Peru                                        -1.1871      0.708     -1.677
0.094      -2.578      0.204
Country_Poland                                      -3.1012      1.050     -2.955
0.003      -5.163     -1.040
Country_Slovenia                                    -0.6975      0.434     -1.605
0.109      -1.551      0.156
Country_South Korea                                 -2.2039      0.624     -3.532
0.000      -3.430     -0.978
Country_Spain                                       -1.1633      1.379     -0.844
0.399      -3.871      1.545
Country_Sweden                                      -0.4880      0.482     -1.013
0.312      -1.434      0.458
Country_Taiwan                                      -1.2188      0.435     -2.803
0.005      -2.073     -0.365
Country_United States                               -2.2031      0.881     -2.502
0.013      -3.933     -0.473
Country_Vietnam                                     -1.8397      0.586     -3.140
0.002      -2.990     -0.689
==============================================================================
Omnibus:                       10.308   Durbin-Watson:                   1.998
Prob(Omnibus):                  0.006   Jarque-Bera (JB):               13.361
Skew:                           0.177   Prob(JB):                      0.00126
Kurtosis:                       3.624   Cond. No.                     1.50e+16
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correct
ly specified.
[2] The smallest eigenvalue is 6.26e-30. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

In [11]:

```python
def LR_OSR2(model, X_test, y_test, y_train):
    y_pred = model.predict(X_test)
    SSE = np.sum((y_test - y_pred)**2)  # Sum of Squared Errors
    SST = np.sum((y_test - np.mean(y_train))**2)  # Total Sum of Squares
    return (1 - SSE/SST)

# Calculate OSR² for Linear Regression
lr_osr2 = LR_OSR2(results, X_test, y_test, y_train)

# Print the OSR² for the Linear Regression model
print(f"OSR² for the Linear Regression model: {lr_osr2:.4f}")
```

```
OSR² for the Linear Regression model: 0.6752
```

In [12]:

```python
# calculate Variance Inflation Factor for each explanatory variable
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor

def VIF(df, columns):
    values = sm.add_constant(df[columns]).values
```

```
    num_columns = len(columns)+1
    vif = [variance_inflation_factor(values, i) for i in range(num_columns)]
    return pd.Series(vif[1:], index=columns)

VIF(X, numeric_features)
```

Out[12]:

	0
Gamelength Number	3.518042
Deaths	1.498649
Assists	1.579354
Gold	3.544281
VisionScore	2.337927
BaronDifference	1.675623

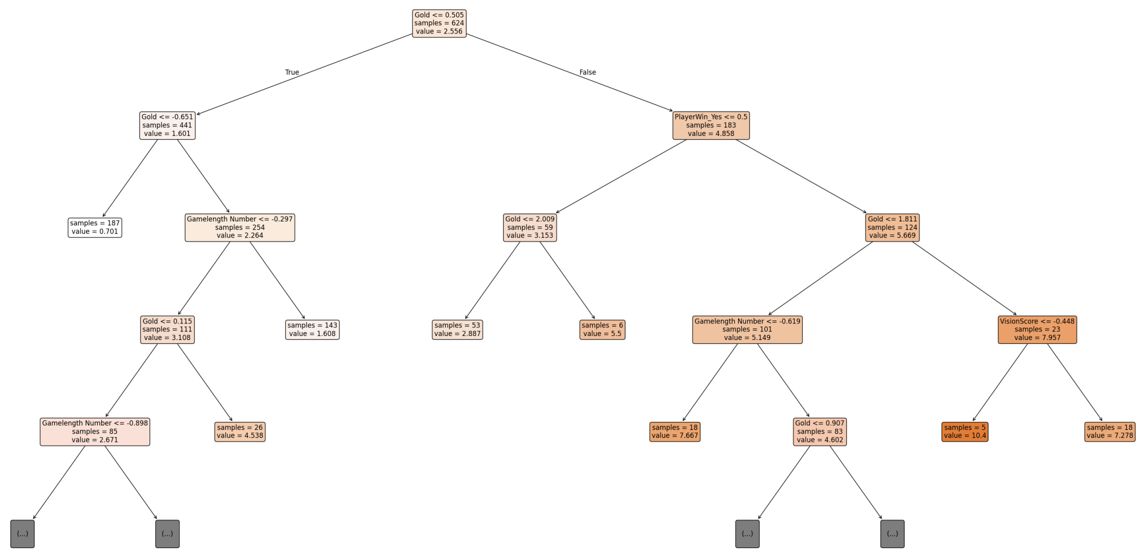
dtype: float64

In [13]:

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import GridSearchCV
import numpy as np

grid_values = {'ccp_alpha': np.linspace(0, 0.10, 201),
               'min_samples_leaf': [5],
               'min_samples_split': [20],
               'max_depth': [30],
               'random_state': [2024]}
dtr = DecisionTreeRegressor()
dtr_cv_nmse = GridSearchCV(dtr, param_grid = grid_values,
                           scoring='neg_mean_squared_error', cv=10, verbose=1)
dtr_cv_nmse.fit(X_train, y_train)
import matplotlib.pyplot as plt
from sklearn.tree import plot_tree
print('Node count =', dtr_cv_nmse.best_estimator_.tree_.node_count)
plt.figure(figsize=(40,20))
plot_tree(dtr_cv_nmse.best_estimator_,
          feature_names=X_train.columns,
          class_names=['0','1'],
          filled=True,
          impurity=False,
          rounded=True,
          fontsize=12,
          max_depth=4)
plt.show()
```

```
Fitting 10 folds for each of 201 candidates, totalling 2010 fits
Node count = 25
```



In [14]:

```
# CART OSR^2 Calculation
def CART_OSR2(model, X_test, y_test, y_train):
  y_pred = model.predict(X_test)
  SSE = np.sum((y_test - y_pred)**2)
  SST = np.sum((y_test - np.mean(y_train))**2)
  return (1 - SSE/SST)
cart_osr2 = CART_OSR2(dtr_cv_nmse, X_test, y_test, y_train)
print('The OSR^2 for the tree regression model is: ', cart_osr2)
```

```
The OSR^2 for the tree regression model is:  0.537184386710543
```

In [15]:

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

# Split data into training and testing sets
X_train_rf, X_test_rf, y_train_rf, y_test_rf = train_test_split(
    X_processed_df, y, test_size=0.2, random_state=42
)
# Initialize Random Forest Regressor with fixed parameters
rf_regressor = RandomForestRegressor(
    n_estimators=100,
    max_depth=10,
    min_samples_split=5,
    min_samples_leaf=2,
    random_state=2024
)
# Fit the model
rf_regressor.fit(X_train_rf, y_train_rf)
# Predictions
y_pred_rf = rf_regressor.predict(X_test_rf)
# Evaluation
mse = mean_squared_error(y_test_rf, y_pred_rf)
r2 = r2_score(y_test_rf, y_pred_rf)
print("\nRandom Forest Regression Results:")
print(f"Mean Squared Error (MSE): {mse:.4f}")
print(f"R-squared (R²): {r2:.4f}")

# Feature Importance
importances = rf_regressor.feature_importances_
feature_importances = pd.DataFrame({
```

```
    'Feature': X_train_rf.columns,
    'Importance': importances
}).sort_values(by='Importance', ascending=False)

print("\nTop 10 Feature Importances:")
print(feature_importances.head(10))
```

```
Random Forest Regression Results:
Mean Squared Error (MSE): 2.6335
R-squared (R²): 0.6487

Top 10 Feature Importances:
                     Feature  Importance
3                       Gold    0.604208
0          Gamelength Number    0.154622
29             PlayerWin_Yes    0.053638
4                VisionScore    0.044177
2                    Assists    0.028694
1                     Deaths    0.023243
5            BaronDifference    0.010760
32  KeystoneRune_Electrocute    0.006668
9                   Role_Top    0.005393
58       Country_South Korea    0.005054
```

In [16]:

```
# OSR² Calculation (out-of-sample R²)
def RF_OSR2(model, X_test, y_test, y_train):
    y_pred = model.predict(X_test)
    SSE = np.sum((y_test - y_pred)**2)
    SST = np.sum((y_test - np.mean(y_train))**2)
    return (1 - SSE/SST)

rf_osr2 = RF_OSR2(rf_regressor, X_test_rf, y_test_rf, y_train_rf)

print(f"OSR² for the Random Forest model: {rf_osr2:.4f}")
```

```
OSR² for the Random Forest model: 0.6530
```

In [17]:

```
Comparison = pd.DataFrame({
    "Model": ["Linear", "CART", "Random Forest"],
    "OSR^2": [lr_osr2, r2, rf_osr2]
})
Comparison
```

Out[17]:

	Model	OSR^2
0	Linear	0.675186
1	CART	0.648695
2	Random Forest	0.652968