Project Report: Predicting Player Kills in a 2024 League of Legends Match

Team Members: Haoxuan Qu, Jingting Chen, Yihan Yan, Huiyi He, Jiayi Mo

Introduction

League of Legends (LoL) is a highly popular multiplayer online battle arena (MOBA) game that requires players to demonstrate strategic thinking, teamwork, and individual skill. A key aspect of gameplay involves securing kills, which can significantly impact the outcome of a match. Understanding the factors that influence player kills provides valuable insights into gameplay dynamics, helping players improve their strategies and potentially offering predictive analytics for esports organizations.

In this project, we aim to predict the number of kills a player secures in a LoL match using various match-related features. The dataset includes player performance metrics such as gold earned, damage dealt, champion statistics, and map objectives. By analyzing these variables, we aim to identify patterns and build predictive models that capture the relationships between match features and player kills.

We evaluated the performance of three regression models—Linear Regression, Decision Tree Regressor, and Random Forest Regressor—using out-of-sample R-squared as the primary metric. Out-of-sample R-squared measures the proportion of variance in player kills explained by the model on unseen data, providing an effective way to assess each model's generalization ability. The report outlines our process, including exploratory data analysis, model development, and interpretation, demonstrating the potential of machine learning techniques in analyzing esports performance.

Data Overview

We extracted data from the Leaguepedia API using Python, focusing on players, teams, and match statistics from international tournaments in 2024. The dataset consists of 25 features, such as gold per game, opposing teams, and player age. For preprocessing, to address potential multicollinearity, we introduced features like tower differences and baron differences between two teams and removed the original columns for individual tower and baron counts. Incomplete entries in the Age and Country columns were dropped since we want these features to be applied in the model-building stage and we have sufficient training and test data, which is around 800 observations.

Exploratory Data Analysis (EDA)

Exploratory data analysis (EDA) was performed to understand the structure of the dataset and the relationships between features before building predictive models. For quantitative features, we first created a correlation heatmap (e.g., Figure 1) to detect multicollinearity. We noticed that Gold, CS (creep score), and DamageToChampions had high positive correlations (0.80) with one another. To avoid potential multicollinearity, we decided to keep only Gold as one of the features. In addition, BaronDifference and TowerDifference showed a strong positive correlation (0.77), which aligns with our expectation that these features jointly contribute to team advantages during games.

After removing the variables with high correlations, a Variance Inflation Factor (VIF) table (e.g., Figure 2) was created to further test for multicollinearity. Most of the VIF values remained under the threshold of 5, indicating that there is no strong presence of multicollinearity. Despite Gold having a relatively higher VIF value of 7.49 compared to the threshold, we decided to keep it in the model because its VIF is still under 10, meaning it might not significantly affect model performance. Also, Gold has high predictive value as a metric of a team's economic performance in games and can provide valuable information for our predictions.

For categorical features, we created boxplots to identify important predictors for our models. For example, the Kills Distribution by Role plot (e.g., Figure 3) illustrates that the medians are generally different for different roles. Specifically, Mid and Bot roles have a higher number of kills compared to other roles. On the other hand, the Support role has the fewest kills, reflecting its role in driving team performance rather than directly contributing to kills. These differences in distribution and spread in the kills distribution indicate that Role might be a significant feature in predicting the kills per game for a player.

Methodologies

1. **Linear Regression Model**

We began by training a linear regression model. As an initial step, the dataset was separated into features (X) and the target variable (y), with "Kills" as the dependent variable. The features were divided into numeric and categorical categories. For preprocessing, numeric features were standardized using StandardScaler, while categorical features were one-hot encoded using OneHotEncoder, with the first category dropped to avoid multicollinearity. These processed features were then combined and converted into a DataFrame with appropriate column names. Subsequently, a constant term was added to the feature set to account for the intercept in the regression model. The data was then split into training and test sets, ensuring that the indices were properly aligned. Finally, we utilized the statsmodels library to fit an ordinary least squares (OLS) regression model to the training data.

The resulting linear regression model yielded an R-squared value of 0.708 and an adjusted R-squared of 0.676, indicating that the model explains a substantial portion of the data variability, though some predictors may have a more limited contribution to the overall fit. The out-of-sample R-squared value of 0.6692 demonstrates the model's ability to generalize its predictions to unseen data, albeit with slightly reduced explanatory power compared to the in-sample performance. This metric highlights the robustness of the model in maintaining predictive accuracy when applied to new observations. The F-statistic of 22.19, accompanied by a low p-value, provides strong evidence that the regression model is statistically significant. It confirms that the predictors collectively have a meaningful impact on the number of kills a player secures in a match, reinforcing the reliability of the model's conclusions.

The analysis of coefficients reveals that factors such as Deaths, Gold, and VisionScore positively impact the number of kills a player secures in a match, emphasizing the importance of resource accumulation, map control, and aggressive playstyles. Roles like Jungle and Support also show a positive association with kill counts, likely due to their critical roles in facilitating team coordination and creating opportunities for engagements. Conversely, features such as GameLength, Assists, BaronDifference, and the roles of Mid and Top are negatively associated with kill numbers. This suggests that longer games, objective-focused strategies, and roles with different priorities, like lane control or team support, tend to reduce individual kill contributions.

These findings offer valuable insights into the interplay between player behavior, role-specific responsibilities, and strategic decisions within the game. They highlight the importance of considering role dynamics and overarching team objectives when analyzing player performance. While the model provides a strong foundation for understanding these relationships, future analysis could be improved by incorporating interaction effects between features, such as the synergy between roles and team strategies, to uncover more nuanced patterns. Moreover, integrating time-series data to examine player performance trends across multiple matches could yield deeper insights into player behavior and team dynamics.

2. **Regression Tree Model**

Additionally, we applied a Decision Tree model to predict the number of kills in a game based on features such as Gold, Game Length, Player Win, and Assists. Decision trees are interpretable machine learning models that split the data into segments based on key thresholds. It allows us to easily identify the most influential features. Through the training process, the model determined that Gold was the most important predictor, likely due to its strong relationship with player performance and game outcomes. To optimize the model, we used GridSearchCV with 10-fold cross-validation to tune hyperparameters, which resulted in an optimal configuration with 33 nodes and a maximum depth of 4. This depth allowed us to balance the trade-off between model complexity and generalizability, which can not only avoid overfitting but also capturing significant patterns in the data.

Then, we visualize the decision tree (e.g., Figure 4) to allow us better understand the model's splits. First, the root node splits the data based on the Gold feature at a threshold of 0.503. The second-level split occurs on Game Length. We can see that higher Gold values combined with longer Game Length and winning status lead to higher predicted kills. Players with low Gold and shorter game durations perform poorly (value = 0.631), whereas those with high Gold and extended game times have higher kills (value = 9.286). This tree model effectively captures nonlinear relationships, showing that performance depends on a combination of Gold, match duration, and win status.

We evaluated our model's performance through calculating OSR^2, which is equal to 0.5530. This indicates that the model explains approximately 55.3% of the variance on test data. Such moderate accuracy suggests that the model captures important patterns but may benefit from additional optimization or more advanced models.

3. Random Forest

We then chose to build a Random Forest Regression model due to its ability to handle complex datasets and capture non-linear relationships between features and the target variable (Kills). Given the large number of features in our dataset, including both numerical and categorical variables, Random Forest is a natural choice because it can handle high-dimensional data without requiring extensive feature engineering. Additionally, its ability to provide feature importance helps us understand the key drivers of player kills in League of Legends matches.

After splitting the data into training and test sets, we used K-fold cross-validation (5 folds) to perform cross-validation. The decision to use K-fold instead of grid search cross-validation was driven by the need to balance model performance and runtime. Grid search, while effective for finding the best parameters, can be very computationally expensive and time-consuming. Therefore, we opted for a simpler approach, setting reasonable default values for hyperparameters like the number of estimators and tree depth based on prior knowledge. This allowed us to focus on assessing the model's general performance while minimizing runtime.

The Random Forest model provided us with an R-squared value of 0.5530, indicating that about 55% of the variance in the number of kills is explained by the model. While this suggests the model captures a significant portion of the variance, there is still room for improvement. The Mean Squared Error of 3.1154 reflects the average deviation between the predicted and actual kill counts, which suggests that while the model performs reasonably well, it still has some degree of error. Feature importance analysis revealed that the most influential features in predicting kills were Gold (0.6125) and Game Length (0.1435), which aligns with the understanding that a player's economic advantage and match duration are key drivers of number of kills in League of Legends. These insights could be valuable in understanding player performance and formulating strategies.

Given the moderate R^2 score and the reasonable MSE, we are moderately confident in the Random Forest model's ability to predict player kills. While it doesn't explain all of the variance, it provides valuable insights into the factors affecting kill counts. The model's performance on the test set, coupled with an out-of-sample R^2 of 0.5625, shows that it generalizes well to unseen data, further boosting our confidence in its predictive ability.

For future extensions, we could explore further hyperparameter tuning by adjusting parameters like n_estimators, max_depth, and min_samples_split more extensively to improve model performance. Additionally, we could consider using out-of-bag (OOB) error for validation to save computation time and compare it against the current cross-validation results. Lastly, testing the model with more diverse features or ensemble methods like Gradient Boosting could lead to improved predictions.

Evaluation and Conclusion

The performance of the three models—Linear Regression, Random Forest, and Decision Tree (CART)—was assessed using out-of-sample R-squared (OSR^2) to evaluate their ability to generalize to unseen data. This metric is particularly suitable for our application because it directly reflects the model's ability to generalize beyond the training data, which is critical when making predictions about player performance in new and varied League of Legends matches. Unlike training R^2, which can overestimate a model's performance due to overfitting, OSR^2 accounts for potential discrepancies between training and test sets, offering a realistic evaluation. The results are summarized below:

Model	OSR^2
Linear Regression	0.6692
CART	0.5530
Random Forest	0.5625

Linear Regression achieved the highest OSR^2 score of 0.6692, demonstrating its effectiveness in capturing the relationship between features and kill counts. Both Random Forest and CART provided additional insights through feature importance analysis, identifying key predictors such as Gold, Game Length, and Deaths. However, these models underperformed compared to Linear Regression, with OSR^2 scores of 0.5625 and 0.5530, respectively, likely due to limited non-linear interactions in the dataset.

These results demonstrate that simpler models can outperform complex ones when the dataset lacks significant nonlinear interactions. However, the relatively moderate OSR^2 scores across all models suggest opportunities for improvement. Future work could focus on incorporating richer features, such as champion-specific stats, team compositions, or time-series data, and experimenting with advanced methods like Gradient Boosting to uncover deeper gameplay insights. Beyond game-specific performance metrics, we can incorporate other factors such as leadership, personality, and the coach-player relationship. While these are qualitative variables, they can be quantified through analysis of the team's weekly documentaries. By combining these factors with traditional performance data, we can extend our focus to predicting a player's salary. This provides a more comprehensive analysis of a player's value, which teams may find useful when making decisions.

In conclusion, this project highlights the utility of machine learning in esports analytics, offering valuable insights into player performance and a strong foundation for further research in competitive gaming.

Appendix

Data Reference: https://lol.fandom.com/wiki/Help:Leaguepedia_API

Code Link:
https://colab.research.google.com/drive/1WHC5POqrXdAszfUzP0FEV182ryHtLE4t?usp=sharing

Youtube Link: https://youtu.be/hzmD8B_5Mjk

Feature Table:

Feature Name	Type	Description
Kills (Dependent)	Numerical	Number of kills secured by the player
BaronDifference	Numerical	Difference in the number of Baron Nashor objectives (Player's Team Baron - Emeny Team Baron)
GameLength	Numerical	Duration of the game in minutes
Deaths	Numerical	Number of times the player died during the game
Assists	Numerical	Number of assists made by the player
Gold	Numerical	Total gold earned by the player during the game
VisionScore	Numerical	How much vision a player has influenced in the game, including the vision they granted and denied
KeystoneRune	Categorical	Primary rune equipped by the player in the game
Country	Categorical	Nationality of player
Role	Categorical	Player's role in the game
Team	Categorical	Name of the team the player belongs to
PlayerWin	Binary	Indicates whether the player's team won the game

Description Reference:https://leagueoflegends.fandom.com/wiki/Category:Gameplay_elements

Figure 1: Correlation Heatmap

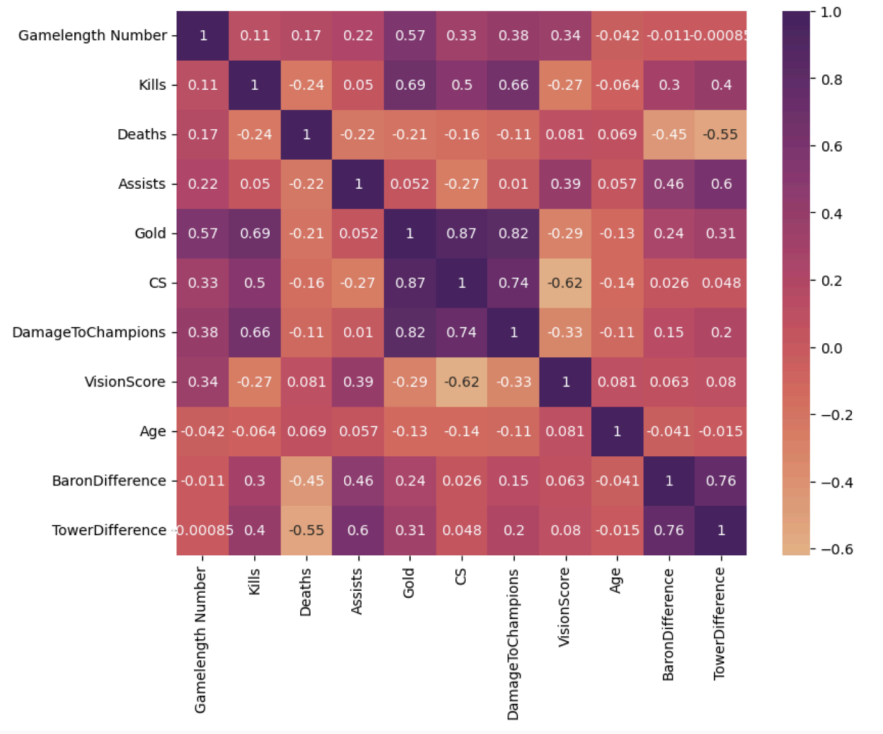


Figure 2: Variance Inflation Factor (VIF) table

Gamelength Number	4.869059
Kills	2.775555
Deaths	1.546634
Assists	1.591315
Gold	7.497193
VisionScore	2.568353
Age	1.026872
BaronDifference	1.679407

Figure 3: Kills Distribution by Role plot

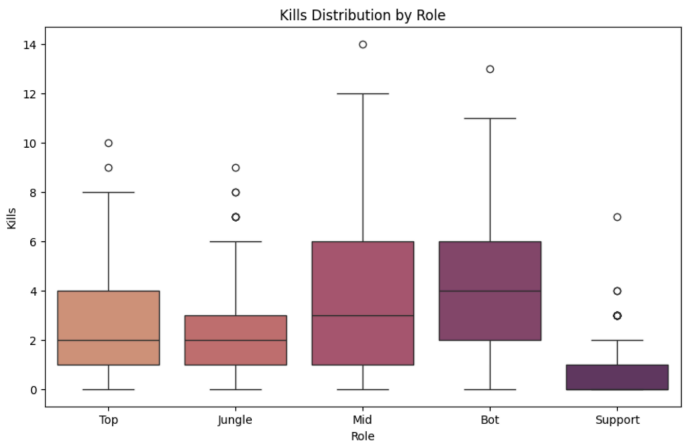
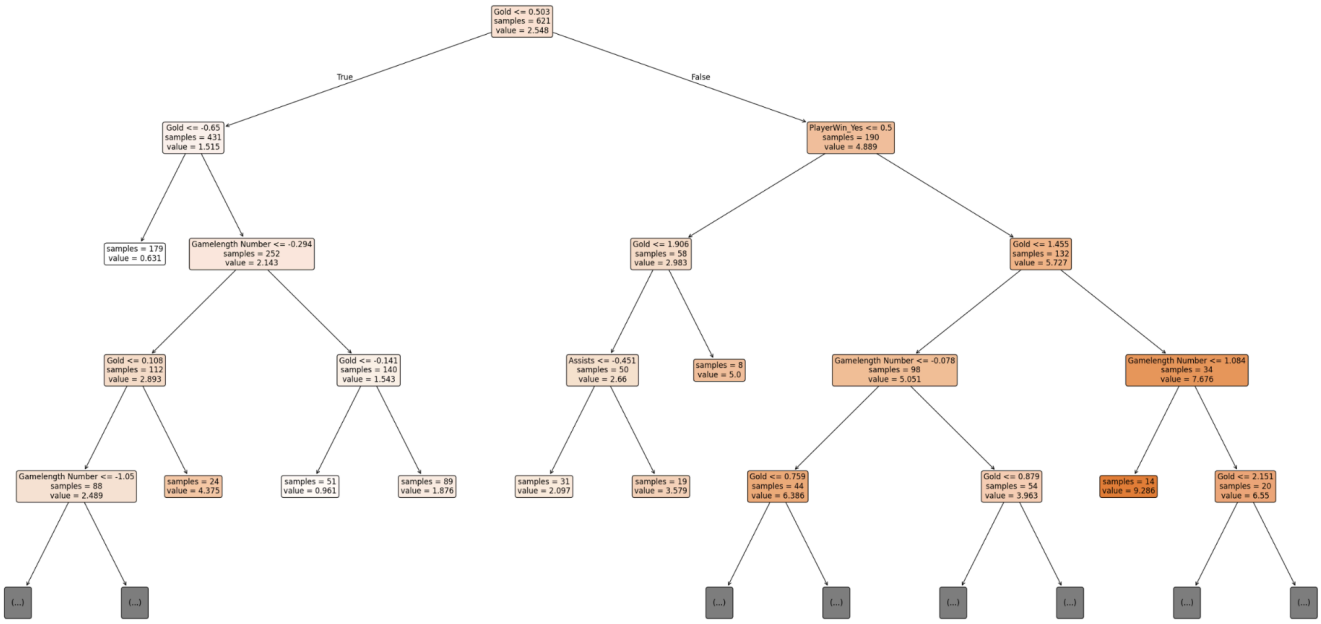


Figure 4: Decision Tree Visualization



Data import:

```
import pandas as pd
import mwclient

# Site
site = mwclient.Site('lol.fandom.com', path='/')

# Tournaments
tournaments = [
    "Worlds 2024 Play-In",
    "Worlds 2024 Main Event",
    "Esports World Cup 2024",
    "MSI 2024"
]

# Loop through each tournament
all_data = []
for tournament in tournaments:
    response = site.api(
        'cargoquery',
        limit=1000,
        tables="ScoreboardGames=SG, ScoreboardPlayers=SP, ScoreboardTeams=ST,PlayerRedirects=PR, Players=P",
        fields="SG.GameId, SG.Tournament, SG.DateTime_UTC, SG.Team1, SG.Team2, SG.Gamelength_Number,"
               "SG.Team1Barons, SG.Team2Barons, SG.Team1Towers, SG.Team2Towers," # Team Stats
               "SP.Name, SP.Role, SP.Team, SP.TeamVs, SP.PlayerWin,"
               "SP.Kills, SP.Deaths, SP.Assists, SP.Gold, SP.CS, SP.DamageToChampions, SP.VisionScore, SP.Items, SP.KeystoneRune,"
               "P.Age, P.Country", # Player Info
        join_on="SG.GameId=SP.GameId, SP.GameId=ST.GameId, SP.Name=PR.AllName, PR.OverviewPage=P.OverviewPage",
        where=f'SG.Tournament = "{tournament}"'
    )
    df = pd.json_normalize(response, record_path=['cargoquery'], sep='-')
    all_data.append(df)

# Combined dataframes
combined_df = pd.concat(all_data, ignore_index=True).drop_duplicates()
combined_df.columns = [col.replace('title-', '') for col in combined_df.columns]

# Output csv
combined_df.to_csv('LOL_matchdata_2024.csv', index=False)
print("success")
```

Code reference: https://lol.fandom.com/wiki/Help:Leaguepedia_API;
https://blog.csdn.net/lazymark2/article/details/117935338

Preprocessing:

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

```
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', 'Team1', 'Team2', 'TeamVs', 'TowerDifference'], inplace=True)
```

Heatmap:

```
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()
```

Code Reference: Asked GPT now to select numeric variables

```
import pandas as pd

# Example dataframe
data = pd.DataFrame({
    'A': [1, 2, 3],
    'B': [1.1, 2.2, 3.3],
    'C': ['text1', 'text2', 'text3'],
    'D': [True, False, True]
})

# Select numeric columns (integers and floats)
numeric_data = data.select_dtypes(include=['number'])
```

VIF table:

```
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)
features = [
    "Gamelength Number", "Kills", "Deaths", "Assists", "Gold",  "VisionScore", "Age",
    "BaronDifference"]
VIF(data, features)
```

Code Reference: Asked GPT ways to compute VIF

```
# Example dataframe
data = pd.DataFrame({
    'X1': [1, 2, 3, 4, 5],
    'X2': [2, 4, 6, 8, 10],
    'X3': [5, 10, 15, 20, 25]
})

# Add a constant (intercept) to the data
X = add_constant(data)

# Compute VIF for each feature
vif = pd.DataFrame()
vif["Variable"] = X.columns
vif["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
```

Boxplot:

```
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()
```

Linear regression:

```
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'), categorical_features)
    ]
)

# 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(categorical_features).tolist()

# 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, test_size=0.2, random_state=42)

# 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())
```

Code reference: asked GPT to do the preprocessing and handle the transformed feature matrix output

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

# 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()
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  Kills   R-squared:                       0.708
Model:                            OLS   Adj. R-squared:                  0.676
Method:                 Least Squares   F-statistic:                     22.19
Date:                Tue, 17 Dec 2024   Prob (F-statistic):          2.85e-113
Time:                        03:33:46   Log-Likelihood:                -1080.4
No. Observations:                 621   AIC:                             2285.
Df Residuals:                     559   BIC:                             2560.
Df Model:                          61
Covariance Type:            nonrobust
=================================================================================================================
                                                  coef    std err          t      P>|t|      [0.025      0.975]
-----------------------------------------------------------------------------------------------------------------
const                                           3.6933      0.878      4.207      0.000       1.969       5.418
Gamelength Number                              -1.8441      0.145    -12.751      0.000      -2.128      -1.560
Deaths                                          0.4451      0.080      5.542      0.000       0.287       0.603
Assists                                        -0.2744      0.095     -2.877      0.004      -0.462      -0.087
Gold                                            3.4622      0.171     20.273      0.000       3.127       3.798
VisionScore                                     0.5318      0.157      3.379      0.001       0.223       0.841
BaronDifference                                -0.3013      0.105     -2.858      0.004      -0.508      -0.094
Role_Jungle                                     0.9385      0.299      3.140      0.002       0.351       1.526
Role_Mid                                       -0.2189      0.274     -0.800      0.424      -0.757       0.319
Role_Support                                    1.4087      0.478      2.947      0.003       0.470       2.348
Role_Top                                       -0.0996      0.303     -0.329      0.742      -0.694       0.495
Team_Bilibili Gaming                            0.0425      1.327      0.032      0.974      -2.564       2.649
Team_Dplus KIA                                 -0.9556      0.994     -0.962      0.337      -2.907       0.996
Team_FlyQuest                                  -0.2037      0.690     -0.295      0.768      -1.559       1.151
Team_Fnatic                                    -0.9244      1.229     -0.752      0.452      -3.338       1.489
Team_Fukuoka SoftBank HAWKS gaming              1.2470      0.856      1.457      0.146      -0.434       2.928
Team_G2 Esports                                -1.5268      0.716     -2.131      0.034      -2.934      -0.120
Team_GAM Esports                               -0.9962      0.427     -2.335      0.020      -1.834      -0.158
Team_Gen.G                                     -0.4880      0.639     -0.764      0.445      -1.742       0.766
Team_Hanwha Life Esports                        0.6185      0.817      0.757      0.449      -0.986       2.223
Team_LNG Esports                                0.1099      1.236      0.089      0.929      -2.318       2.538
Team_MAD Lions KOI                             -0.7607      1.122     -0.678      0.498      -2.965       1.444
Team_Movistar R7                               -0.7780      0.706     -1.102      0.271      -2.165       0.609
Team_PSG Talon                                 -0.9237      0.429     -2.153      0.032      -1.766      -0.081
Team_T1                                        -0.3393      0.626     -0.542      0.588      -1.568       0.890
Team_Team Liquid                                0.4185      0.691      0.606      0.545      -0.938       1.775
Team_Top Esports                               -0.0302      1.316     -0.023      0.982      -2.614       2.554
Team_Vikings Esports (2023 Vietnamese Team)    -0.2542      0.486     -0.523      0.601      -1.209       0.700
Team_Weibo Gaming                               0.2217      1.369      0.162      0.871      -2.467       2.910
Team_paiN Gaming                                0.3756      0.749      0.501      0.616      -1.096       1.847
PlayerWin_Yes                                   1.4729      0.266      5.528      0.000       0.950       1.996
KeystoneRune_Arcane Comet                       0.0251      0.337      0.074      0.941      -0.636       0.687
KeystoneRune_Conqueror                         -0.4357      0.263     -1.656      0.098      -0.953       0.081
KeystoneRune_Electrocute                        0.0675      0.420      0.161      0.872      -0.757       0.892
KeystoneRune_First Strike                      -0.5585      0.441     -1.266      0.206      -1.425       0.308
KeystoneRune_Fleet Footwork                    -0.3892      0.334     -1.164      0.245      -1.046       0.267
KeystoneRune_Glacial Augment                   -0.3176      0.681     -0.466      0.641      -1.656       1.020
KeystoneRune_Grasp of the Undying              -0.2727      0.334     -0.815      0.415      -0.930       0.384
KeystoneRune_Guardian                          -0.3991      0.310     -1.286      0.199      -1.009       0.210
KeystoneRune_Hail of Blades                    -0.2207      0.413     -0.534      0.594      -1.033       0.591
KeystoneRune_Lethal Tempo                      -1.1887      0.606     -1.961      0.050      -2.379       0.002
KeystoneRune_Phase Rush                        -0.3928      0.354     -1.108      0.268      -1.089       0.303
KeystoneRune_Press the Attack                  -0.5812      0.340     -1.711      0.088      -1.248       0.086
KeystoneRune_Summon Aery                        0.5634      0.414      1.361      0.174      -0.249       1.376
KeystoneRune_Unsealed Spellbook                -0.8998      1.084     -0.830      0.407      -3.029       1.229
Country_Australia                              -2.1664      1.190     -1.821      0.069      -4.504       0.171
Country_Belgium                                -2.0496      1.060     -1.934      0.054      -4.131       0.032
Country_Brazil                                 -2.7179      0.817     -3.329      0.001      -4.322      -1.114
Country_Canada                                 -0.7464      1.081     -0.690      0.490      -2.870       1.377
Country_China                                  -2.2926      1.305     -1.757      0.079      -4.855       0.270
Country_Czech Republic                         -0.7893      1.413     -0.558      0.577      -3.565       1.987
Country_Denmark                                 0.1939      0.459      0.422      0.673      -0.708       1.095
Country_France                                 -0.6171      0.494     -1.249      0.212      -1.588       0.353
Country_Germany                                 0.4385      0.453      0.968      0.334      -0.451       1.329
Country_Iraq                                   -0.6131      1.032     -0.594      0.553      -2.640       1.413
Country_Japan                                  -2.7770      1.156     -2.403      0.017      -5.047      -0.507
Country_Peru                                   -1.2378      0.696     -1.779      0.076      -2.604       0.129
Country_Poland                                 -2.2350      1.008     -2.218      0.027      -4.214      -0.256
Country_Slovenia                               -0.8637      0.464     -1.861      0.063      -1.775       0.048
Country_South Korea                            -2.0666      0.612     -3.375      0.001      -3.269      -0.864
Country_Spain                                  -0.5789      1.299     -0.446      0.656      -3.131       1.973
Country_Sweden                                 -0.6784      0.437     -1.552      0.121      -1.537       0.180
Country_Taiwan                                 -0.9237      0.429     -2.153      0.032      -1.766      -0.081
Country_United States                          -1.5005      0.892     -1.681      0.093      -3.253       0.252
Country_Vietnam                                -1.2504      0.596     -2.099      0.036      -2.420      -0.080
==============================================================================
Omnibus:                       16.245   Durbin-Watson:                   1.853
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               19.049
Skew:                           0.310   Prob(JB):                     7.30e-05
```

```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: 0.6692
```

Decision Tree:

```python
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()
```

Code reference: asked GPT to set the grid values for GridSearchCV for me

```python
grid_values = {
    'ccp_alpha': np.linspace(0, 0.10, 201),  # Fine-tuned values for pruning
    'min_samples_leaf': [5],                 # Fixed value for minimum samples per l
    'min_samples_split': [20],               # Fixed value for minimum samples to sp
    'max_depth': [30],                       # Fixed depth of the tree
    'random_state': [2024]                   # Random state for reproducibility
}
```

Random Forest Regression:

```
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}")
```

Code reference: asked GPT to set regressor parameters for me and helped me print the result

```
# Initialize Random Forest Regressor with specified parameters
rf_regressor = RandomForestRegressor(
    n_estimators=100,       # Number of trees in the forest
    max_depth=10,           # Maximum depth of the tree
    min_samples_split=5,    # Minimum samples required to split a node
    min_samples_leaf=2,     # Minimum samples required at each leaf node
    random_state=2024       # Random state for reproducibility
)
```

```
Random Forest Regression Results:
Mean Squared Error (MSE): 3.1154
R-squared (R²): 0.5530

Top 10 Feature Importances:
                              Feature  Importance
3                                Gold    0.612471
0                   Gamelength Number    0.143520
29                      PlayerWin_Yes    0.083164
4                         VisionScore    0.037168
2                             Assists    0.026204
1                              Deaths    0.019647
5                     BaronDifference    0.008361
58                Country_South Korea    0.005393
6                          Role_Jungle    0.005330
36  KeystoneRune_Grasp of the Undying    0.004412
```

```python
# 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: 0.5625
```

Random forest future extension analysis reference: asked GPT to help me identify future improvements such as OOB validation.

For future extensions of the Random Forest model, we could explore several avenues to enhance its performance and interpretability:

1. **Hyperparameter Optimization**: Implement advanced hyperparameter tuning methods, such as Randomized Search or Bayesian Optimization, to identify the best configuration for parameters like `n_estimators`, `max_features`, and `min_samples_split`.
2. **Feature Selection**: Investigate feature importance scores more deeply and refine the input dataset by removing less significant features to simplify the model and reduce overfitting.
3. **Out-of-Bag (OOB) Evaluation**: Utilize OOB scoring to evaluate model performance more efficiently, providing an additional validation measure without relying on a separate test set.
4. **Model Interpretation**: Use SHAP (SHapley Additive exPlanations) or LIME (Local Interpretable Model-Agnostic Explanations) to better understand the influence of individual features on the model's predictions.
5. **Ensemble Techniques**: Combine Random Forest with other models, such as Gradient Boosting or Neural Networks, in an ensemble framework to capture different aspects of the data.