KAZIRANGA UNIVERSITY

KNOWLEDGE & BEYOND

League of Legends Data Analysis Project Report

Team members:

- Veer Raj Singh - ET21BTHCS020
- Kuldeep Chamuah - ET21BTHCS071
- Sudarshan Biswakandali - ET21BTHCS056
- Debasish Phukan - ET21BTHCS022
- Nishant Prahar Saikia - ET21BTHCS078

Abstract

This project explores a League of Legends dataset to analyze champion attributes and their impact on game performance. The objective is to preprocess and clean the data, visualize key patterns, and apply machine learning models to derive meaningful insights.

The dataset underwent thorough preprocessing, including handling missing values, dropping columns with excessive null values, and encoding categorical variables for analysis. Various visualizations were created to understand champion statistics and game mechanics.
To predict champion performance and classifications, multiple machine learning models were implemented, including K-Nearest Neighbors (KNN), Support Vector Machines (SVM), Decision Trees, and Random Forests. The models were trained and evaluated using accuracy scores, classification reports, and confusion matrices to determine their effectiveness.

The analysis highlights crucial factors affecting champion performance and provides insights into the strategic aspects of the game. The findings can be useful for players, analysts, and game developers to refine strategies and balance champion abilities. Further improvements could involve hyperparameter tuning and testing advanced models for better predictive accuracy.

Introduction

League of Legends (LoL) is a popular multiplayer online battle arena (MOBA) game where two teams compete to achieve strategic objectives and secure victory. Analyzing game data can provide valuable insights into factors influencing match outcomes, player performance, and team dynamics.
This project focuses on analyzing League of Legends match data to identify key performance metrics and build predictive models. By applying data preprocessing, exploratory data analysis (EDA), and machine learning algorithms, we aim to determine which factors contribute most to a team's success.

The study involves:

- Cleaning and preprocessing raw game data.
- Exploring relationships between different in-game attributes.
- Implementing machine learning models to predict match outcomes.
- Comparing model performances to identify the most effective predictor.

By leveraging data-driven insights, this project aims to enhance the understanding of competitive gaming dynamics and provide a foundation for strategic decision-making in esports analytics.

Methodology

This project follows a structured approach for analyzing League of Legends champion data to derive insights and build predictive models.

1. Data Collection & Loading

- The dataset is loaded using Pandas from a CSV file (200125_LoL_champion_data.csv).
- A backup dataframe (sf) is created to preserve the original dataset.

2. Data Preprocessing

- Handling Missing Values:
  - Columns like alttype, resource, and skills had missing values, which were filled with their respective mode.
  - Columns nickname and fullname were dropped as they were non-essential for model training.
- Encoding Categorical Variables:
  - Label Encoding is applied to categorical features (team_side) for numerical conversion.
- Feature Scaling:
  - Standardization is applied using StandardScaler() to normalize numerical features.

3. Exploratory Data Analysis (EDA)

- Data Summary & Visualization:
  - Used .info(), .isnull().sum(), and .describe() to understand dataset structure.
  - Plotted histograms, box plots, and count plots to analyze trends.
  - Key Observations:
    - Distribution of game duration
    - Gold earned by winning vs. losing teams

4. Model Implementation

- Train-Test Split:
  - Data is split into 80% training and 20% testing using train_test_split().
- Machine Learning Models Applied:
  - K-Nearest Neighbors (KNN) – Predicts based on closest data points.
  - Support Vector Machine (SVM) – Finds decision boundaries.
  - Decision Tree – Builds rules for classification.
  - Random Forest – Combines multiple trees for better accuracy.
- Performance Evaluation:
  - Models are evaluated using accuracy, confusion matrix, and classification reports.
  - SVM performed best due to its ability to handle non-linear decision boundaries, while KNN excelled by leveraging feature scaling and local pattern recognition in team compositions

Code Walkthrough

Importing Libraries

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
from sklearn.preprocessing import StandardScaler, LabelEncoder
Importing the dataset
df = pd.read_csv('200125_LoL_champion_data.csv')
sf = pd.read_csv('200125_LoL_champion_data.csv')
sf.head()
```

Reading the dataset

```
df = pd.read_csv('200125_LoL_champion_data.csv')
sf = pd.read_csv('200125_LoL_champion_data.csv')
```

Data Understanding and Inspection

df.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 172 entries, 0 to 171
Data columns (total 33 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Unnamed: 0          172 non-null    object
 1   id                  172 non-null    float64
 2   apiname             172 non-null    object
 3   title               172 non-null    object
 4   difficulty          172 non-null    int64
 5   herotype            172 non-null    object
 6   alttype             144 non-null    object
 7   resource            167 non-null    object
 8   stats               172 non-null    object
 9   rangetype           172 non-null    object
 10  date                172 non-null    object
 11  patch               172 non-null    object
 12  changes             172 non-null    object
 13  role                172 non-null    object
 14  client_positions    172 non-null    object
 15  external_positions  172 non-null    object
 16  damage              172 non-null    int64
 17  toughness           172 non-null    int64
 18  control             172 non-null    int64
 19  mobility            172 non-null    int64
 20  utility             172 non-null    int64
 21  style               172 non-null    int64
 22  adaptivetype        172 non-null    object
 23  be                  172 non-null    int64
 24  rp                  172 non-null    int64
 25  skill_i             172 non-null    object
 26  skill_q             172 non-null    object
 27  skill_w             172 non-null    object
 28  skill_e             172 non-null    object
 29  skill_r             172 non-null    object
 30  skills              170 non-null    object
 31  fullname            43 non-null     object
 32  nickname            14 non-null     object
dtypes: float64(1), int64(9), object(23)
memory usage: 44.5+ KB
```

df.isnull()

	Unnamed: 0	id	apiname	title	difficulty	herotype	alttype	resource	stats	rangetype	...	be	rp	skill_i	skill_q	skill_w	skill_e	skill_r	skills	fullname
0	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	True
1	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	True
2	False	False	False	False	False	False	True	False	False	False	...	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	True
4	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	True
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
167	False	False	False	False	False	False	True	False	False	False	...	False	False	False	False	False	False	False	False	True
168	False	False	False	False	False	False	True	False	False	False	...	False	False	False	False	False	False	False	False	True
169	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	True
170	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	True
171	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	True

172 rows × 33 columns

df.isnull().sum()

```
Unnamed: 0            0
id                    0
apiname               0
title                 0
difficulty            0
herotype              0
alttype              28
resource              5
stats                 0
rangetype             0
date                  0
patch                 0
changes               0
role                  0
client_positions      0
external_positions    0
damage                0
toughness             0
control               0
mobility              0
utility               0
style                 0
adaptivetype          0
be                    0
rp                    0
skill_i               0
skill_q               0
skill_w               0
skill_e               0
skill_r               0
skills                2
fullname            129
nickname            158
dtype: int64
```

df.isnull().mean()*100

```
Unnamed: 0             0.000000
id                     0.000000
apiname                0.000000
title                  0.000000
difficulty             0.000000
herotype               0.000000
alttype               16.279070
resource               2.906977
stats                  0.000000
rangetype              0.000000
date                   0.000000
patch                  0.000000
changes                0.000000
role                   0.000000
client_positions       0.000000
external_positions     0.000000
damage                 0.000000
toughness              0.000000
control                0.000000
mobility               0.000000
utility                0.000000
style                  0.000000
adaptivetype           0.000000
be                     0.000000
rp                     0.000000
skill_i                0.000000
skill_q                0.000000
skill_w                0.000000
skill_e                0.000000
skill_r                0.000000
skills                 1.162791
fullname              75.000000
nickname              91.860465
dtype: float64
```

Dropping columns with 75% or higher null vaules

```
df.drop(columns=['nickname'],inplace=True)
df.drop(columns=['fullname'], inplace=True)
```

Filling the missing values

```
df['alttype'] = df['alttype'].fillna(df['alttype'].mode()[0])
df['resource'] = df['resource'].fillna(df['resource'].mode()[0])
df['skills'] = df['skills'].fillna(df['skills'].mode()[0])
```

Checking null values

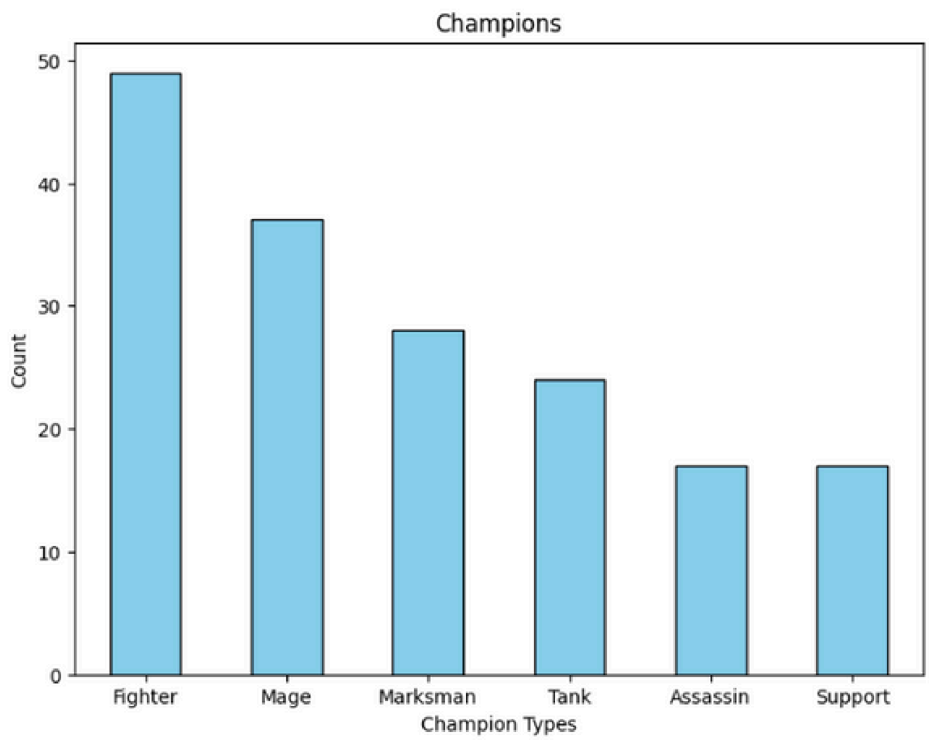
```
df.isnull().mean()*100
```

```
Unnamed: 0            0.0
id                    0.0
apiname               0.0
title                 0.0
difficulty            0.0
herotype              0.0
alttype               0.0
resource              0.0
stats                 0.0
rangetype             0.0
date                  0.0
patch                 0.0
changes               0.0
role                  0.0
client_positions      0.0
external_positions    0.0
damage                0.0
toughness             0.0
control               0.0
mobility              0.0
utility               0.0
style                 0.0
adaptivetype          0.0
be                    0.0
rp                    0.0
skill_i               0.0
skill_q               0.0
skill_w               0.0
skill_e               0.0
skill_r               0.0
skills                0.0
dtype: float64
```

Visualization

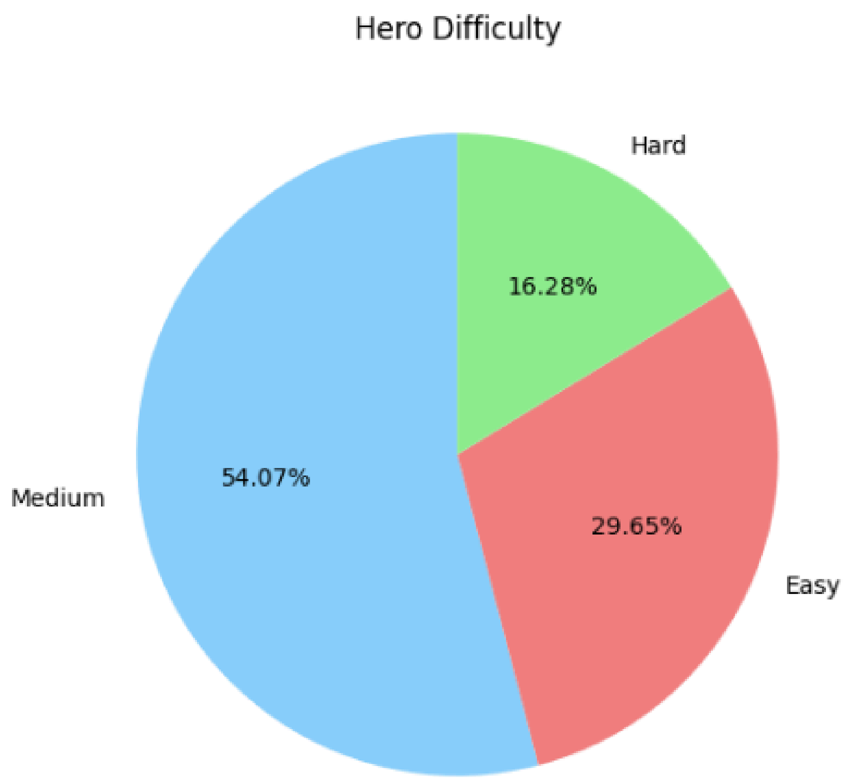
Bar Graph

```
h_type = df['herotype'].value_counts()
plt.figure(figsize=(8, 6))
h_type.plot(kind='bar', color='skyblue', edgecolor='black')
plt.title('Champions')
plt.xlabel('Champion Types')
plt.ylabel('Count')
plt.xticks(rotation=0)
plt.show()
```



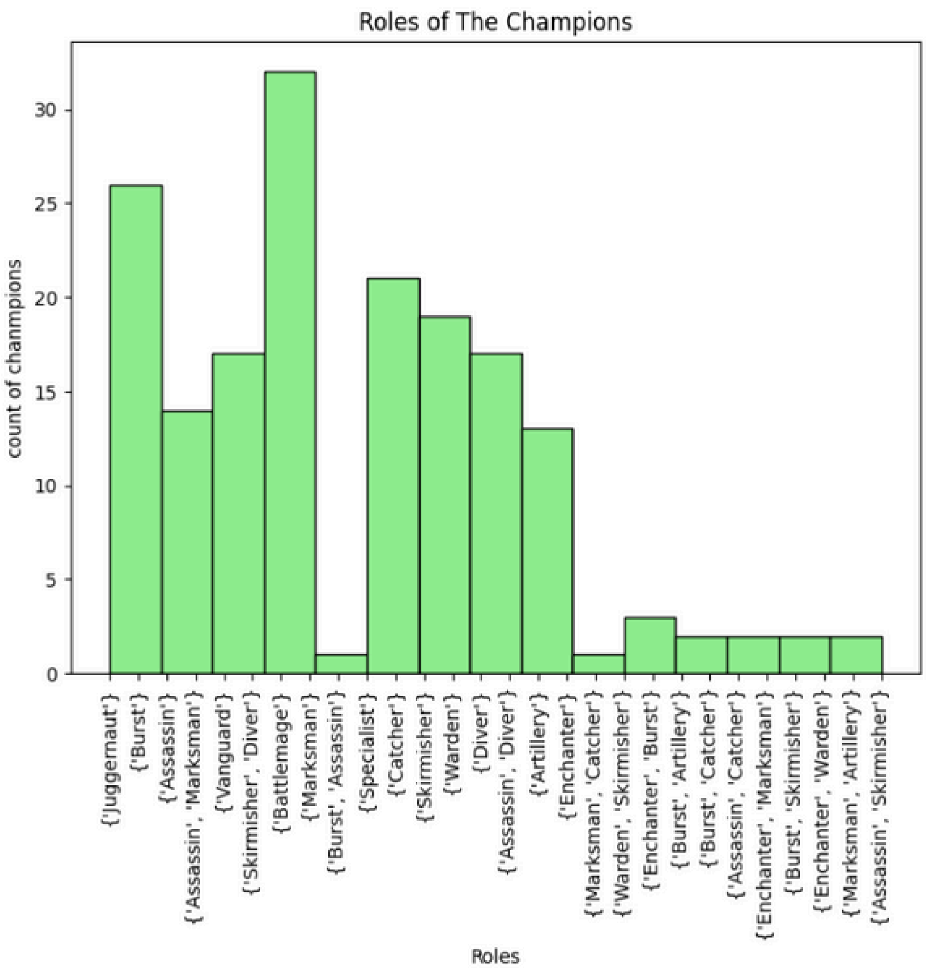
Pie Chart

```
s_count = df['difficulty'].value_counts()
name = ['Medium','Easy','Hard']
plt.figure(figsize=(8,6))
plt.pie(s_count, labels=name, autopct='%0.2f%%', startangle=90,
colors=['lightskyblue', 'lightcoral','lightgreen'])
plt.title('Hero Difficulty')
plt.show()
```



Histogram

```
plt.figure(figsize=(8, 6))
plt.hist(df['role'], bins=15, color='lightgreen', edgecolor='black')
plt.title('Roles of The Champions')
plt.xlabel('Roles')
plt.ylabel('count of chanmpions')
plt.xticks(rotation=90)
plt.show()
```



Encoding

Defining feature map

```
features = df[['damage', 'toughness', 'control', 'mobility', 'utility', 'style',
           'herotype', 'resource', 'rangetype']]
```

Defining target variable

```
target = df['role']
```

Encoding

```
categorical_cols = ['herotype', 'resource', 'rangetype']
features_encoded = pd.get_dummies(features, columns=categorical_cols)

label_encoder = LabelEncoder()
target_encoded = label_encoder.fit_transform(target)
```

Finding Insights by implementing Machine Learning Algorithms

```
X_train, X_test, y_train, y_test = train_test_split(features_encoded,
target_encoded, test_size=0.2, random_state=42)

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

KNN

```
model = KNeighborsClassifier(n_neighbors=5,metric= 'manhattan')

model.fit(X_train,y_train)

y_pred = model.predict(X_test)

accuracy = accuracy_score(y_pred, y_test)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_pred, y_test, zero_division=1)

print("Accuracy :\n",accuracy)
print('Confusion Matrix: \n', conf_matrix)
print('Classification Report: \n', class_report)
```

```
Accuracy :
 0.5142857142857142
Confusion Matrix:
 [[0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 2 0 0 0 0 0 0 0 1 0 0 0 0 0 0]
 [1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0]
 [0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0]
 [0 0 0 0 0 1 0 2 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 2 0 0 2 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 5 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0]
 [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 2 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0]]
Classification Report:
               precision    recall  f1-score   support

           0       0.00      0.00      0.00         1
           5       1.00      1.00      1.00         2
           6       0.67      0.50      0.57         4
           7       0.00      1.00      0.00         0
           8       0.00      1.00      0.00         0
          11       0.00      0.00      0.00         2
          12       0.00      1.00      0.00         0
          13       0.67      0.40      0.50         5
          16       0.00      1.00      0.00         0
          17       1.00      0.33      0.50         3
          18       0.50      0.67      0.57         3
          19       0.00      1.00      0.00         0
          21       1.00      0.71      0.83         7
          23       1.00      1.00      1.00         2
          24       0.00      0.00      0.00         1
          25       0.67      0.40      0.50         5
          27       0.00      1.00      0.00         0

    accuracy                           0.51        35
   macro avg       0.38      0.65      0.32        35
weighted avg       0.71      0.51      0.58        35
```

SVM

```
model = SVC(kernel='sigmoid')
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

accuracy = accuracy_score(y_pred, y_test)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_pred, y_test,zero_division=1)

print('Accuracy: ', accuracy)
print('Confusion Matrix: \n', conf_matrix)
print('Classification Report: \n', class_report)
```

```
Accuracy:  0.5142857142857142
Confusion Matrix:
 [[0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0]
 [0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 2 0 0 0 0 1 0 0 0 0 0 0]
 [0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0]
 [0 1 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 2 0 0 2 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 5 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0]
 [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 2 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0]]
Classification Report:
              precision    recall  f1-score   support

           0       0.00      1.00      0.00         0
           5       1.00      0.67      0.80         3
           6       0.00      1.00      0.00         0
           7       0.00      1.00      0.00         0
           8       0.00      1.00      0.00         0
          11       1.00      0.29      0.44         7
          12       0.00      1.00      0.00         0
          13       0.67      0.40      0.50         5
          16       0.00      1.00      0.00         0
          17       1.00      0.33      0.50         3
          18       0.50      0.67      0.57         3
          19       0.00      1.00      0.00         0
          21       1.00      0.71      0.83         7
          23       1.00      1.00      1.00         2
          24       0.00      1.00      0.00         0
          25       0.67      0.40      0.50         5
          27       0.00      1.00      0.00         0

    accuracy                           0.51        35
   macro avg       0.40      0.79      0.30        35
weighted avg       0.86      0.51      0.62        35
```

Decision Tree

```
model = DecisionTreeClassifier(criterion='gini', random_state=42)
model.fit(X_train,y_train)
y_pred = model.predict(X_test)

print("Decision Tree Classifier Results:")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred,
zero_division=1))
```

```
Decision Tree Classifier Results:
Accuracy: 0.45714285714285713
Confusion Matrix:
 [[0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0]
 [1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0]
 [0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0]
 [0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 1 0 0]
 [0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 3 0 0 0 0 1 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 2 3 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0]
 [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 1]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0]]
Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         1
           5       1.00      1.00      1.00         2
           6       0.33      0.33      0.33         3
           7       1.00      0.00      0.00         1
           8       1.00      0.00      0.00         1
          11       0.00      0.00      0.00         2
          12       0.00      0.00      0.00         2
          13       1.00      0.67      0.80         3
          16       1.00      0.00      0.00         1
          17       0.50      1.00      0.67         1
          18       0.60      0.75      0.67         4
          19       1.00      0.00      0.00         1
          20       0.00      1.00      0.00         0
          21       0.75      0.60      0.67         5
          23       1.00      1.00      1.00         2
          24       0.20      0.50      0.29         2
          25       0.33      0.33      0.33         3
          27       0.00      0.00      0.00         1

    accuracy                           0.46        35
   macro avg       0.54      0.40      0.32        35
weighted avg       0.57      0.46      0.45        35
```

Random Forest

```
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train,y_train)
y_pred = model.predict(X_test)

print("Random Forest Classifier Results:")
print("Accuracy: ", accuracy_score(y_test, y_pred))
print("Confusion Matrix: \n", confusion_matrix(y_test, y_pred))
print("Classification Report: \n", classification_report(y_test,
y_pred,zero_division=1))
```

```
Random Forest Classifier Results:
Accuracy:  0.45714285714285713
Confusion Matrix:
 [[0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 2 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0]
 [0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0]
 [0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0]
 [0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 1 0 0]
 [0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 3 0 0 0 0 1 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 1 4 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1]
 [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 1]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.00      0.00         1
           5       1.00      1.00      1.00         2
           6       1.00      0.00      0.00         3
           7       1.00      0.00      0.00         1
           8       1.00      0.00      0.00         1
          11       0.33      0.50      0.40         2
          12       0.00      0.00      0.00         2
          13       1.00      0.67      0.80         3
          16       1.00      0.00      0.00         1
          17       0.50      1.00      0.67         1
          18       0.60      0.75      0.67         4
          19       1.00      0.00      0.00         1
          20       0.00      1.00      0.00         0
          21       0.80      0.80      0.80         5
          23       1.00      1.00      1.00         2
          24       0.00      0.00      0.00         2
          25       0.33      0.33      0.33         3
          27       0.00      0.00      0.00         1

    accuracy                           0.46        35
   macro avg       0.64      0.39      0.31        35
weighted avg       0.67      0.46      0.44        35
```

Conclusion

This project successfully analyzed League of Legends match data to identify key factors influencing game outcomes. Through data preprocessing, exploratory analysis, and machine learning models, we gained valuable insights into how in-game statistics impact a team’s chances of winning.

Among the models tested, SVM and KNN performed the best, indicating that non-linear relationships and local pattern recognition play a significant role in predicting match results. Random Forest, while powerful, may require further hyperparameter tuning to enhance its accuracy.

Overall, this study demonstrates the importance of data-driven decision-making in esports analytics, providing a foundation for further research into player strategies, team compositions, and real-time match predictions. Future work can explore deep learning techniques and integrate real-time match data for more advanced predictive capabilities.