Bartosz Moczkowski
242249

Marta Haik
242243

Ansar Shilibekov
240342

League of Legends Winner Prediction

(photo from https://store.epicgames.com/pl/p/league-of-legends)

POLITECHNIKA ŁÓDZKA
Modelling and Data Science
Course: Big Data
Supervisor: dr inż. Paweł Drzymała
Academic Year: 2024/2025
Semester: winter
Time of lab: Friday 12:15-14:15

1. Introduction & Project Goal

League of Legends [1] is a 2009 multiplayer free-to-play online battle arena video game developed and published by Riot Games. In the game two teams of five players battle in player-versus-player combat, each team occupying and defending their half of the map. Each of the ten players controls a character, known as a "champion", with unique abilities and differing styles of play. During a match, champions become more powerful by collecting experience points, earning gold, and purchasing items to defeat the opposing team. In League's main mode, Summoner's Rift, a team wins by pushing through to the enemy base and destroying their "Nexus", a large structure located within.

Fig. 1 The map with arena where players battle against one another

The project aims to predict the winner of a League of Legends game based on the data obtained during the first 10 minutes of gameplay.

The requirements included:

- **ML Libraries**: Logistic Regression, Random Forest and Gradient Boosting as well as some other PySpark libraries are used,

- **Prediction Task**: given the nature of the problem, this will be framed as a prediction task. The model will predict the winner based on specific game metrics available in the first 10 minutes,

- **Big Data Processing**: since League of Legends generates large volumes of real-time data, we assume that Apache Spark is the optimal tool for handling, processing, and analyzing this data at scale.

2. About data

2.1 Data — basic information

To prepare a good model, the data must be of good quality. Our dataset has been found on the Kaggle platform [2], is in the CSV format and contains approx. 10 000 ranked games (from high Diamond to low Master). No columns have missing values. Each row is a unique game. Data has been collected after the first 10 minutes of the game, through the official Riot API. This collection approach aligns with the **one-time registration**, the data captures a snapshot of the game's state at a specific moment (after 10 minutes). The dataset is **secondary**, it was collected and made available by a third party (Kaggle) for analysis (although from a data collection standpoint it was sourced directly from the game's servers). It is reliable and clean, with enough rows to perform something on it.

Glossary of Terms:

League of Legends match: two teams of five players compete to destroy the opposing team's base, called "Nexus". Each player is able to play as a character (called "champion") with unique abilities. They can achieve it by defeating enemy players, controlling **map**, and defeating **creatures** of different difficulty. Creatures can be neutral(defeatable by both teams), or belong to specific team.

- *gameId* — unique RIOT ID of the game. Can be used with the Riot Games API.
- *blueWins* — the target column. 1 if the blue team has won, 0 otherwise.

Kills, Deaths, Assists: Track how well teams are performing in combat.

- *blueFirstBlood* — first kill of the game. 1 if the blue team did the first kill, 0 otherwise
- *blueKills* — number of enemies killed by the blue team
- *blueDeaths* — number of deaths (blue team)
- *blueAssists* — number of kill assists (blue team)

Gold: Indicates team wealth used for items.

- *blueTotalGold* — blue team total gold
- *blueGoldDiff* — blue team gold difference compared to the enemy team
- *blueGoldPerMin* — blue team gold per minute

Experience: Tracks team levels and champion power.

- *blueTotalExperience* — blue team total experience
- *blueExperienceDiff* — blue team experience difference compared to the enemy team
- *blueAvgLevel* — blue team average champion level

Map: The game is played on a map called "Summoner's Rift", divided into three lanes and a jungle. Each team controls half of the map lanes, where their teams' towers are located.
Lanes: different type of routes, on which towers
Jungle: the area between lanes, filled with neutral monsters

Towers: Powerful structures that attack enemies. Deal a lot of damage. Destroying them shows how teams are progressing in lane control.

- *blueTowersDestroyed* — number of structures destroyed by the blue team (towers...)

Wards: Teams place wards for vision and map control.

- *blueWardsPlaced* — number of warding totems placed by the blue team on the map
- *blueWardsDestroyed* — number of enemy warding totems the blue team has destroyed

Team-Sided Creatures:

Minions: these creatures are allied with a specific team and attack only enemies. They divide into types and bring different amounts of gold and experience to the specific person which defeated the minion. CS metric counts the amount of defeated minions in the game.

CS Per Minute: Shows how efficiently teams are farming minions.

- *blueCSPerMin* — blue team CS (minions) per minute
- *blueTotalMinionsKilled* — blue team total minions killed (CS)

Neutral:
Jungle Monsters: group of low level creatures that are easy to defeat. Essentially the same as minions, the difference is that they are not specified to any team.

- *blueTotalJungleMinionsKilled* — blue team total jungle monsters killed

Epic/Elite Monsters: group of harder to defeat monsters that grant high value reward based on the type. Usually requires several team members to defeat.

- *blueEliteMonsters* — number of elite monsters killed by the blue team (Dragons and Heralds)
- *blueDragons* — number of dragons killed by the blue team
- *blueHeralds* — number of heralds killed by the blue team

Similar stats but for the red team:

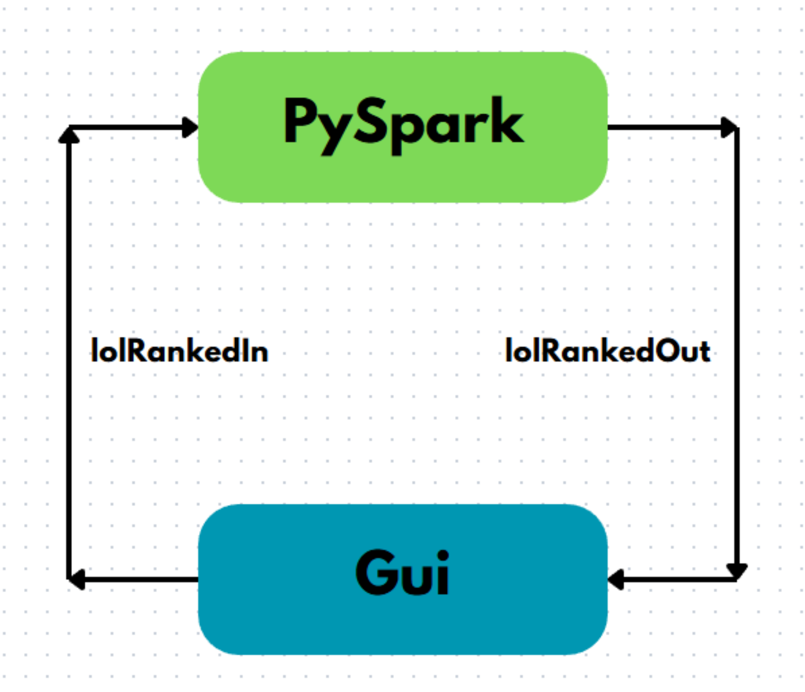
- *redWardsPlaced, redWardsDestroyed, redFirstBlood, redKills, redDeaths, redAssists, redEliteMonsters, redDragons, redHeralds, redTowersDestroyed, redTotalGold, redAvgLevel, redTotalExperience, redTotalMinionsKilled, redTotalJungleMinionsKilled, redGoldDiff, redExperienceDiff, redCSPerMin, redGoldPerMin*

To sum it up, the data includes:

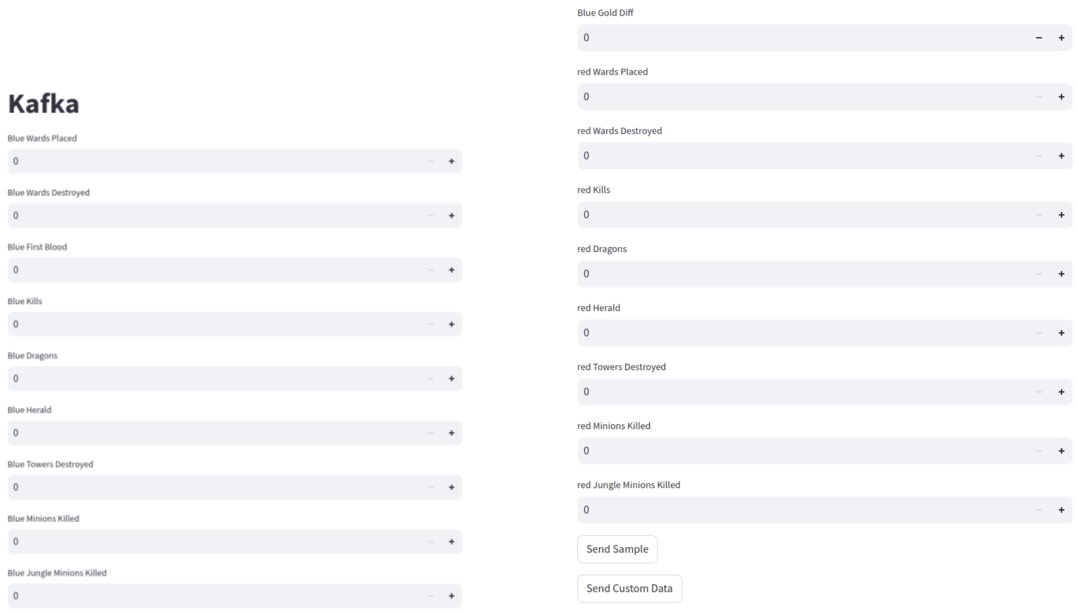
- Match outcome (win/loss)
- Teams' statistics (kills, deaths, assists, gold earned, damage dealt, etc.)
- Game objectives (dragons, towers, baron kills, etc.)

2.2 Kafka Setup

To simulate the theoretical setup of real time data streams in kafka we use two files. The first one has an user interface which allows us to send data to a topic lolRankedIn. And the second one which runs on listens to incoming data on the lolRankedIn topic, runs it through the model and sends the predictions back on the second topic lolRankedOut. The model is loaded as a pipeline from pre-saved data.



The user interface allows us to either send a random record from the dataset or our own custom one.



After sending either data we receive the predicted winner along with the probability of winning.

	winner	probability
0	Blue	0.8203

3. Investigation

To perform operations on data, we have used Google Collab due to its easy accessibility to PySpark, where we loaded the dataset and gained some basic insight — such as no missing values (9879 for each column), min. and max. values, etc.

Table 1. Descriptive statistics of in-game metrics

column	count	mean	stddev	min	max
gameId	9879	4500084045	27573278	4295358071	4527990640
blueWins	9879	0.5	0.5	0	1
blueWardsPlaced	9879	22.29	18.02	5	250
blueWardsDestroyed	9879	2.82	2.17	0	27
blueFirstBlood	9879	0.5	0.5	0	1
blueKills	9879	6.18	3.01	0	22

blueDeaths	9879	6.14	2.93	0	22
blueAssists	9879	6.65	4.06	0	29
blueEliteMonsters	9879	0.55	0.63	0	2
blueDragons	9879	0.36	0.48	0	1
blueHeralds	9879	0.19	0.39	0	1
blueTowersDestroyed	9879	0.05	0.24	0	4
blueTotalGold	9879	16503.46	1535.45	10730	23701
blueAvgLevel	9879	6.92	0.31	4.6	8
blueTotalExperience	9879	17928.11	1200.52	10098	22224
blueTotalMinionsKilled	9879	216.7	21.86	90	283
blueTotalJungleMinionsKilled	9879	50.51	9.9	0	92
blueGoldDiff	9879	14.41	2453.35	-10830	11467
blueExperienceDiff	9879	-33.62	1920.37	-9333	8348
blueGoldPerMin	9879	1650.35	153.54	1073	2370.1
redWardsPlaced	9879	22.37	18.46	6	276
redWardsDestroyed	9879	2.72	2.14	0	24
redFirstBlood	9879	0.5	0.5	0	1
redKills	9879	6.14	2.93	0	22
redDeaths	9879	6.18	3.01	0	22
redAssists	9879	6.66	4.06	0	28
redEliteMonsters	9879	0.57	0.63	0	2
redDragons	9879	0.41	0.49	0	1
redHeralds	9879	0.16	0.37	0	1
redTowersDestroyed	9879	0.04	0.22	0	2
redTotalGold	9879	16489.04	1490.89	11212	22732
redAvgLevel	9879	6.93	0.31	4.8	8.2
redTotalExperience	9879	17961.73	1198.58	10465	22269

redTotalMinionsKilled	9879	217.35	21.91	107	289
redTotalJungleMinionsKilled	9879	51.31	10.03	4	92
redGoldDiff	9879	-14.41	2453.35	-11467	10830
redExperienceDiff	9879	33.62	1920.37	-8348	9333
redCSPerMin	9879	21.73	2.19	10.7	28.9
redGoldPerMin	9879	1648.9	149.09	1121.2	2273.2

```
df.printSchema()
df.describe().show()
```

```
+-------+------------------+------------------+------------------+------------------+------------------+------------------+------------------+--
|summary|          blueWins|   blueWardsPlaced|blueWardsDestroyed|     blueFirstBlood|         blueKills|       blueDragons|       blueHeralds|b
+-------+------------------+------------------+------------------+------------------+------------------+------------------+------------------+--
|  count|              9879|              9879|              9879|              9879|              9879|              9879|              9879|
|   mean|0.4990383642069035| 22.28828828828829| 2.824881060836117|0.5048081789654824|  6.18392549853224|0.3619799574855755|0.18797449134527786|0
| stddev|0.5000243833353937|18.019176515802513|2.1749983822222023|0.5000021878367942|3.011027975365158|0.4805973867496533|0.39071157453780536|0
|    min|                 0|                 5|                 0|                 0|                 0|                 0|                 0|
|    max|                 1|               250|                27|                 1|                22|                 1|                 1|
+-------+------------------+------------------+------------------+------------------+------------------+------------------+------------------+--
```

Fig. 2 Screenshot of the descriptive statistics

The analysis of the schema prompted us to drop some columns, as they would be insignificant and useless later on.

```
df = df.drop("gameId","blueDeaths","redDeaths","blueEliteMonsters","redEliteMonsters", "redCSPerMin", "blueCSPerMin")
df = df.drop("redFirstBlood","blueGoldPerMin","redGoldPerMin","blueTotalExperience","redTotalExperience","redExperienceDiff","redGoldDiff")
df = df.drop("blueExperienceDiff","blueTotalGold","redTotalGold","blueAssists","redAssists","blueAvgLevel","redAvgLevel")
```

We consulted among each other what could potentially impact the game and disregarded (most) columns. Some values do not give any information to our model (such as *gameId* or *red/blueCSPerMin* and many more), some are indirect duplicates (*redDeaths* are the same as *blueKills*). Initial cleaning of the data allowed us to create a correlation matrix (Fig. 2):

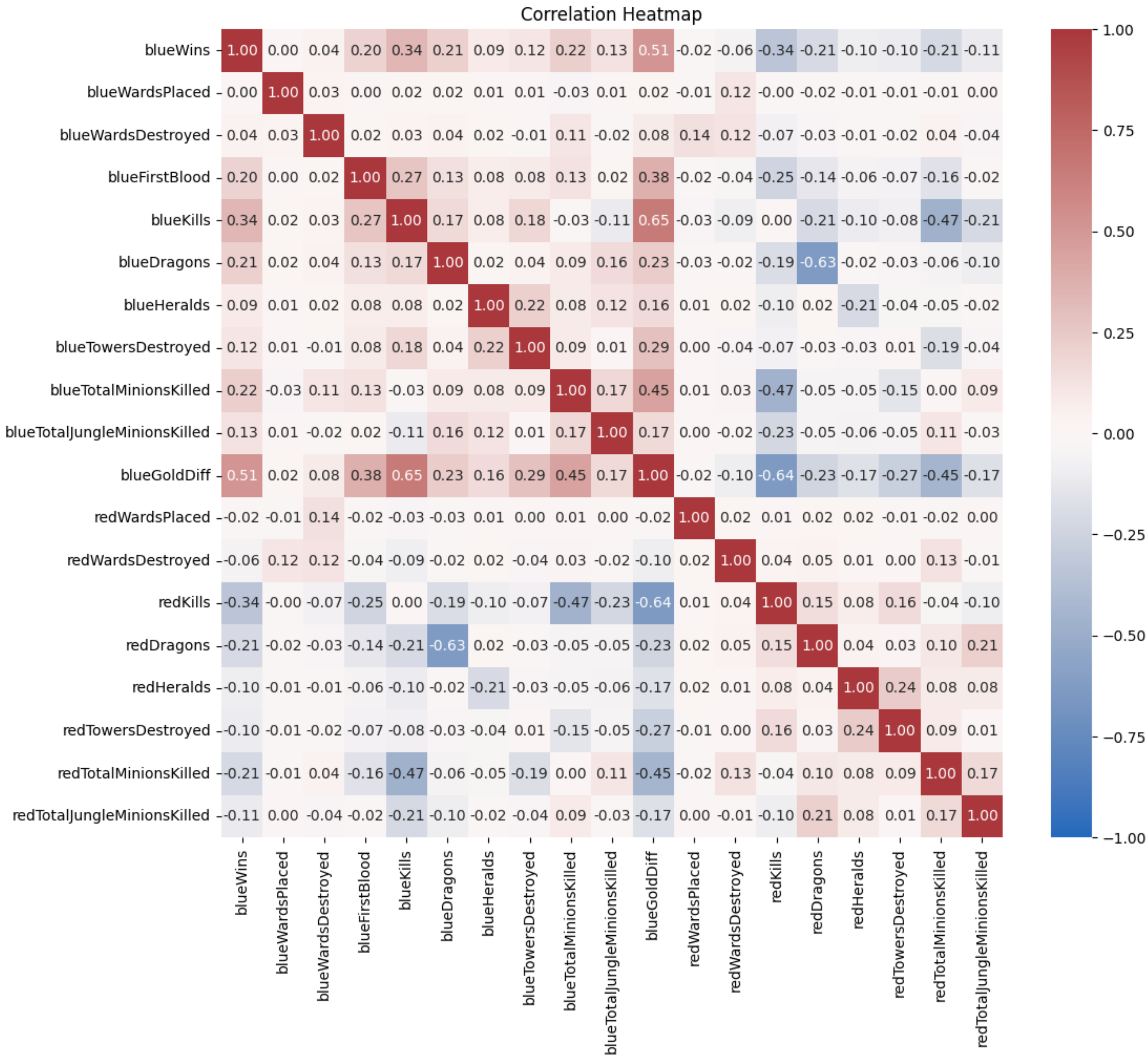


Fig. 2 Screenshot of Correlation Heatmap

To visualize it to ourselves better, the columns have been additionally plotted (Fig. 3, 4):

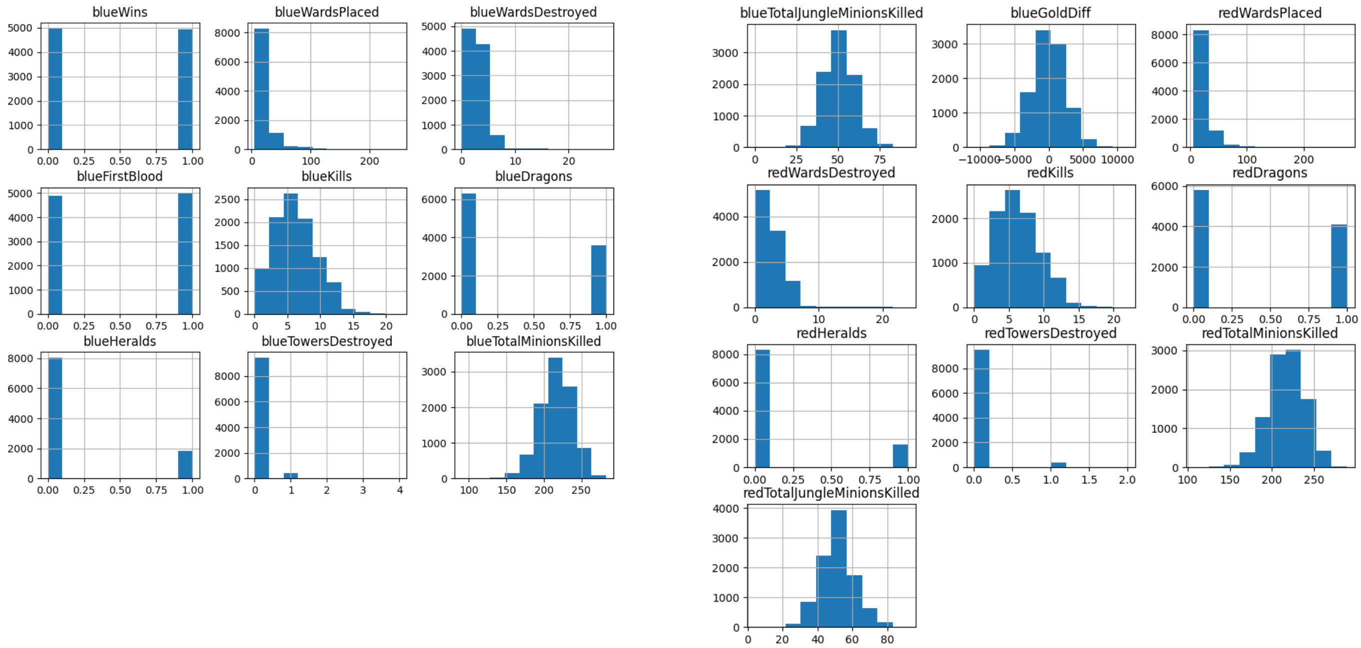


Fig. 3 Histograms of the remaining columns

Nothing took our particular attention, the values were as expected. However, in the very beginning while watching descriptive statistics we noticed that there exists a game with 250 placed wards. From our experience, there usually should have been around 20-25 wards (also confirmed by mean value oscillating at 22) placed after 10 minutes.

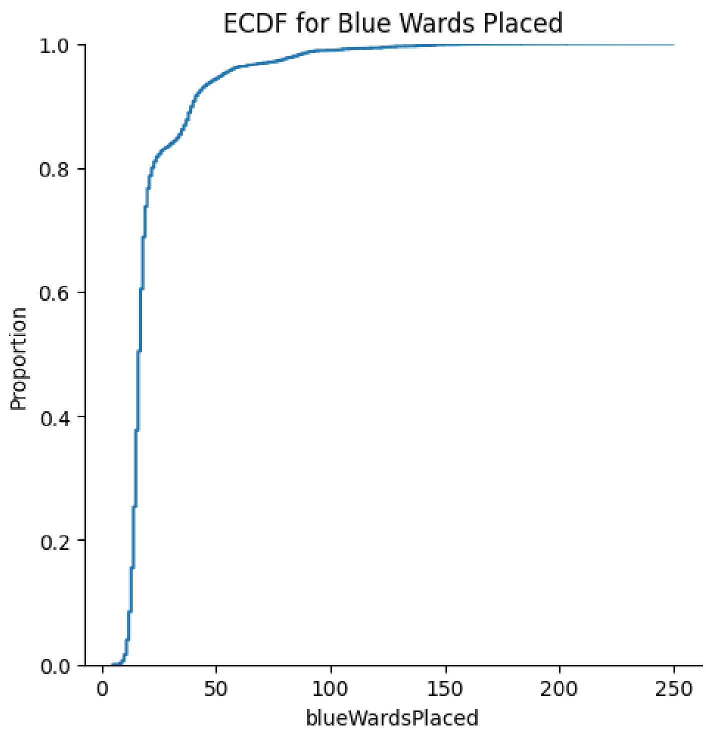


Fig. 4 Curve depicting distribution of wards

An Empirical Cumulative Distribution Function (ECDF) [3] provides a way to visualize the distribution of a dataset. It plots the proportion (or percentage) of data points that are less than or equal to a particular value. As seen on the plot, we considered very high values of wards as outliers and removed them from analyzed columns.

```
from pyspark.sql import functions as F

df_no_outliers = df

percentile_99 = df_no_outliers.approxQuantile('blueWardsPlaced', [0.99], 0.001)[0]
df_no_outliers = df_no_outliers.filter(
    df_no_outliers['blueWardsPlaced'] <= percentile_99
)
print(f"After filtering blueWardsPlaced, {df_no_outliers.count()} rows remain.")

percentile_99 = df_no_outliers.approxQuantile('redWardsPlaced', [0.99], 0.001)[0]
df_no_outliers = df_no_outliers.filter(
    df_no_outliers['redWardsPlaced'] <= percentile_99
)
print(f"After filtering redWardsPlaced, {df_no_outliers.count()} rows remain.")
```

```
After filtering blueWardsPlaced, 9771 rows remain.
After filtering redWardsPlaced, 9666 rows remain.
```

Having done all that, we checked the Target balance, which turned out to be fine.

```
# Target balance

from pyspark.sql.functions import col

df_balanced = df_no_outliers

# To create a relevant model, let's check if the target is well balanced between 0 & 1

total_games = df_balanced.count()
won_games = df_balanced.filter(col('blueWins') == 1).count()

won_percentage = (won_games / total_games) * 100

print(f"In this current Dataset, there is {won_percentage:.3f}% of won games")
```

```
In this current Dataset, there is 49.948% of won games
```

This proves that assigning 1 to blueWins and 0 otherwise (loss) is reasonable. Next, using PySpark's **VectorAssembler**, the features were combined into a column *features* (required for PySpark model).

```
# Target: 'blueWins'
target = df_balanced.select('blueWins')

# Features: everything else
features = df_balanced.drop('blueWins')
```

```
from pyspark.ml.feature import VectorAssembler, StandardScaler
from pyspark.ml.classification import LogisticRegression
from pyspark.ml import Pipeline
from pyspark.sql.functions import col
from pyspark.ml.evaluation import BinaryClassificationEvaluator


feature_columns = features.columns

assembler = VectorAssembler(inputCols=feature_columns,
outputCol='features')
data_assembled = assembler.transform(df)
print(data_assembled)
```

To ensure the features have a consistent scale, we used **StandardScaler** to center the data around zero (subtract the mean) and scale features so they have a standard deviation of 1. We proceeded with the first model (Logistic Regression):

```
# Standardization

scaler = StandardScaler(inputCol="features", outputCol="scaled_features",
withStd=True, withMean=True)

scaler_model = scaler.fit(data_assembled)

data_scaled = scaler_model.transform(data_assembled)

# Split data

train_data, test_data = data_scaled.randomSplit([0.9, 0.1], seed=42)

# Train Logistic Reg

log_reg = LogisticRegression(featuresCol="scaled_features",
labelCol="blueWins")  # Assuming 'blueWins' is the target column
```

```
# Train model

log_reg_model = log_reg.fit(train_data)
```

We then tested some more models, like Random Forest, Gradient Boosted Trees, Decision Tree, Multilayer Perceptron and Naive Bayes, which had varying accuracies (all evaluations included in Jupyter Notebook). We decided to not analyze further Naive Bayes Model given that it was worse than a random guess.

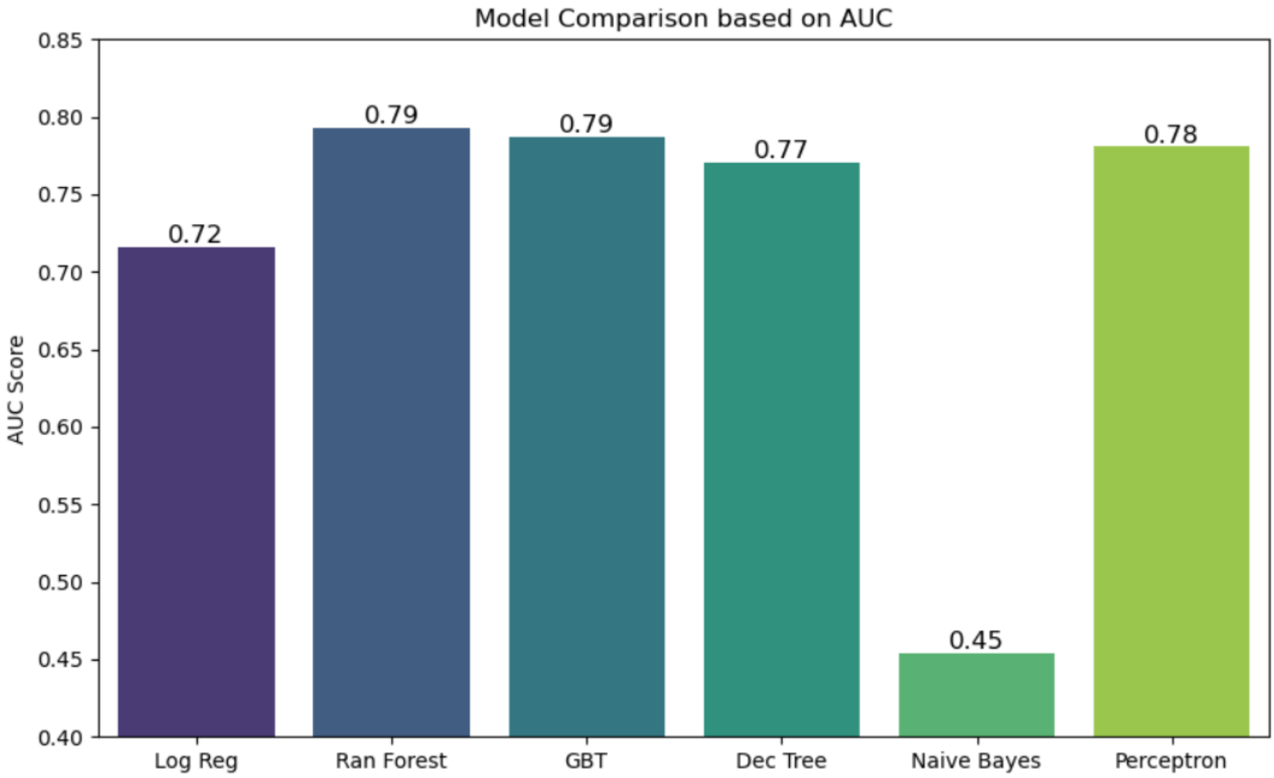


Fig. 5 Plot of tested models' accuracies

The Hyper parameter tuning was very inconsistent. For Most models it took less than 10min, but for some reason for GBT it took more than an hour, which was very strange. Regardless the optimized results are as follows.

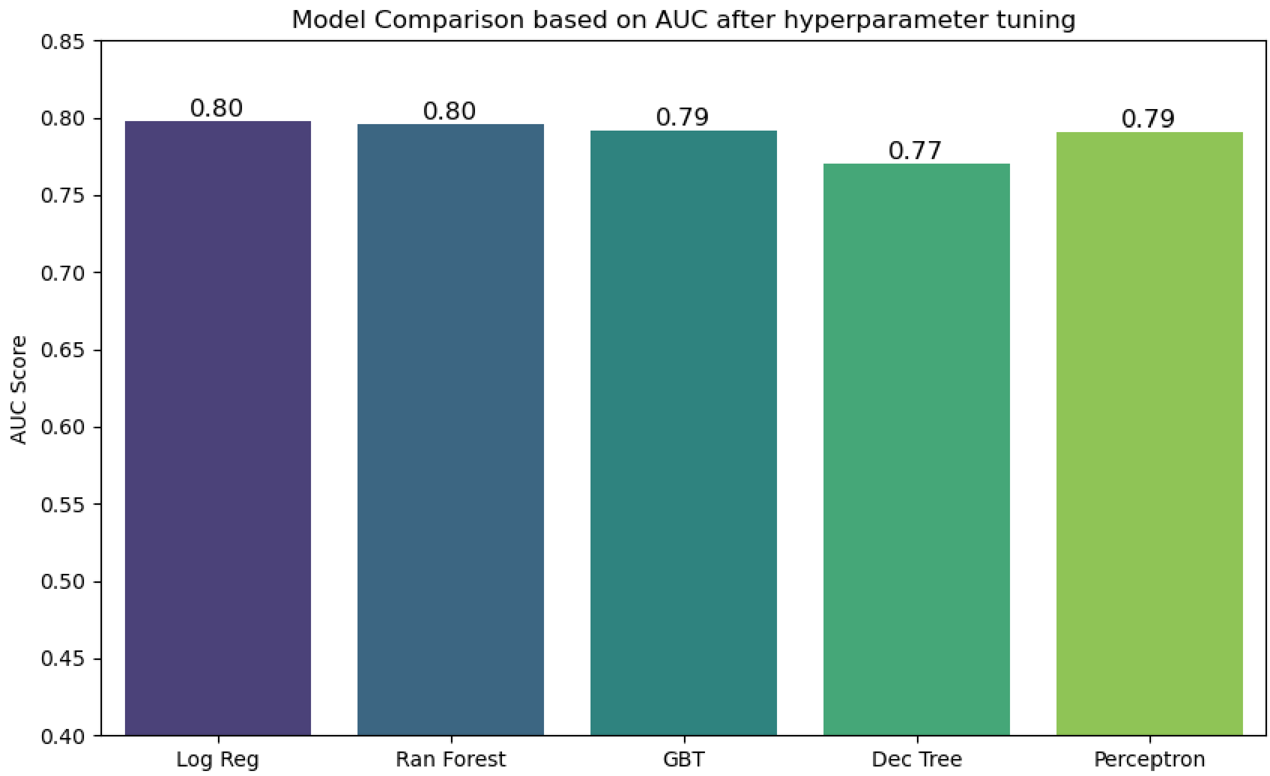


Fig. 6 New accuracies after tuning

The Hyper parameter tuning didn't bring significant increase in accuracy (which was sort of expected), but since the main purpose of this project was to get familiar with PySpark, we proceeded with the model to make some predictions and get some results.

```
Tuned Random Forest AUC: 79.58%
```

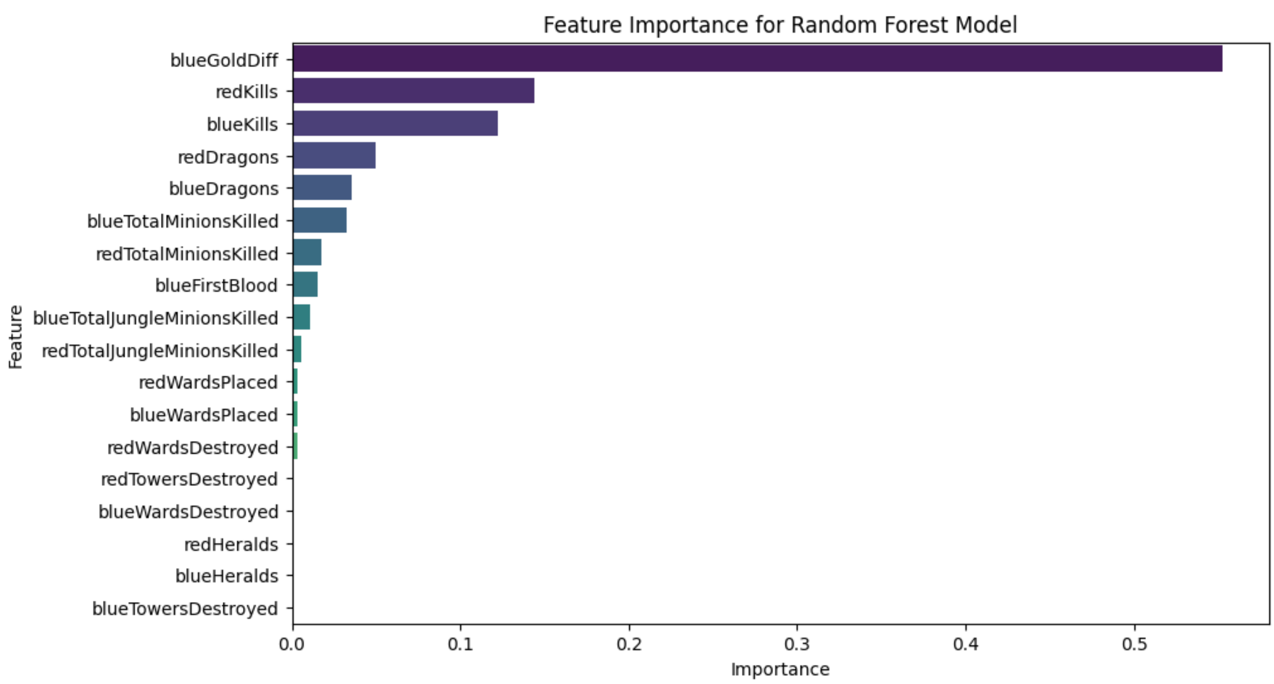


Fig. 7 Plot of Feature Importance of our Random Forest Model

We calculated probabilities of blue wins based on specific scenarios using PySpark's filtering and aggregation functions:

- **Gold Difference:** The probability of winning when the blue team has a gold lead of 1000 or more,
- **Kills:** The likelihood of winning if the blue team has more kills than the red team,
- **Dragons, Turrets, and Heralds:** Similar analyses were done for objectives like dragons, turrets, and heralds, something that players would be interested to know

This gave deeper insights into how certain in-game metrics impact win rates:

- *The probability of the blue team winning when blueGoldDiff is >= 1000 is about **0.2020***
- *The probability of the blue team winning when blueKills > redKills is about **0.7242***
- *The probability of the blue team winning when blueDragons > redDragons is about **0.6417***
- *The probability of the blue team winning when they got First Blood is about **0.6001***
- *The probability of the blue team winning when they destroyed more turrets than red team is about **0.7611***
- *The probability of the blue team winning when they destroyed more wards than red team is about **0.5460***

```
The probability to win based on turrets destroyed at 10 minutes:

+------------------+------------------+

|blueTowersDestroyed|          win_rate|

+------------------+------------------+

|                 0|0.48692633177823585|

|                 1| 0.7397590361445783|

|                 2| 0.9629629629629629|

|                 3|               1.0|

|                 4|               1.0|

+------------------+------------------+



The probability to win based on drakes taken at 10 minutes:

+----------+-----------------+

|blueDragons|         win_rate|

+----------+-----------------+

|         0|0.4187479727538112|

|         1|0.6417142857142857|
```

```
+----------+----------------+



The probability to win based on heralds taken at 10 minutes:

+----------+----------------+

|blueHeralds|         win_rate|

+----------+----------------+

|          0|0.4776347648782974|

|          1|0.5937328202308961|

...

Win rate for low ward destruction teams: 0.4815

The probability of winning with 0 turrets at 10 minutes is 0.4869

The probability of winning with 1 turret at 10 minutes is 0.7398

The probability of winning with 2 turrets at 10 minutes is 0.9630
```

4. Results & Conclusion

4.1 Results

To summarize a bit what we achieved:

- **Model development**: Several machine learning models including Logistic Regression, Random Forest, and Gradient Boosted Trees were developed using PySpark. Among these, Random Forest delivered the best performance with an AUC of 79.58%, which highlights its ability to accurately predict match outcomes based on early game metrics. Hyperparameter tuning was attempted, though it yielded limited improvements.

- **Insights**: By analyzing probabilities of winning based on in-game metrics (e.g. gold advantage, kills, objectives), we provided actionable insights for players and teams:

  For example, blueKills > redKills resulted in a win probability of 72.42%, and blueDragons > redDragons corresponded to a win probability of 64.17%, showcasing the importance of early-game objectives and combat efficiency. Objectives like turrets destroyed showed a particularly strong correlation with success. Winning probabilities rose from 48.69% (0 turrets) to 100% (4 turrets) within the first 10 minutes, underscoring the strategic significance of map control.

4.2 Conclusions

The primary goal of this project was to apply PySpark in a practical manner, gaining experience with its capabilities while exploring a machine learning model — in our case for determining the winner of a League of Legends match based on some game metrics. This objective was successfully achieved: we implemented important PySpark functionalities, performed data cleaning and built predictive model on a relatively big dataset.

The analysis highlights that **early-game performance is a reliable indicator of match outcomes**, with metrics like gold, kills, objectives, and map control playing pivotal roles. These insights can support players, coaches, and analysts in making informed decisions during matches. For example, focusing on early-game strategies to secure objectives like dragons and turrets could dramatically improve the likelihood of success.

The integration of both predictive analytics and real-time data processing demonstrates the potential of this approach for other esports or real-world applications where quick, data-driven decisions are crucial.

5. Bibliography

[1] **Wikipedia contributors**, *League of Legends [Article]*. Wikipedia. Retrieved: 20.01.2025, https://pl.wikipedia.org/wiki/League_of_Legends

[2] **Yi Lan Ma**, *League of Legends Diamond ranked games (10 min) [Dataset].* Kaggle. Retrieved: 20.01.2025, https://www.kaggle.com/datasets/bobbyscience/league-of-legends-diamond-ranked-games-10-min

[3] **Michael Waskom,** *Seaborn Documentation.* Retrieved: 22.01.2025 seaborn.ecdfplot — seaborn 0.13.2 documentation

Table of Figures: