Configuration Manual

MSc Research Project
Data Analytics

Abhishek Pawar
Student ID: X23214112

School of Computing
National College of Ireland

Supervisor: Barry Haycock

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	ABHISHEK MAHENDRA PAWAR		
Student ID:	X23214112		
Programme:	DATA ANALYTICS	**Year:**	2024 - 25
Module:	MSc Research Project ……………………………………………………………………………………………….………		
Supervisor:	BARRY HAYCOCK		
Submission Due Date:	12 December 2024		
Project Title:	Valorant Esports Pre-Match Betting Advisory System uses Machine Learning to Predict Winning Probability and Simulate Odds and Earning Projections.		
Word Count:	……………………………………… **Page Count**: 25		

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: ABHISHEK PAWAR ……………………………………………………………………………………………………

Date: 12/12/2024 ……………………………………………………………………………………………………

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Abhishek Pawar
Student ID: X23214112

1 Introduction

An implementation of Valorant pre-match betting advisory system using machine learning. this configuration manual includes system configuration, Data collection, Library package used, data merging and pre-processing. synthetic data generation, modeling. evaluation and comparing, deployment and user interface.

2 System Configuration

The project was performed on the local system, hardware, and software as mention in figure 1 and figure 2

2.1 Hardware and software requirement’s

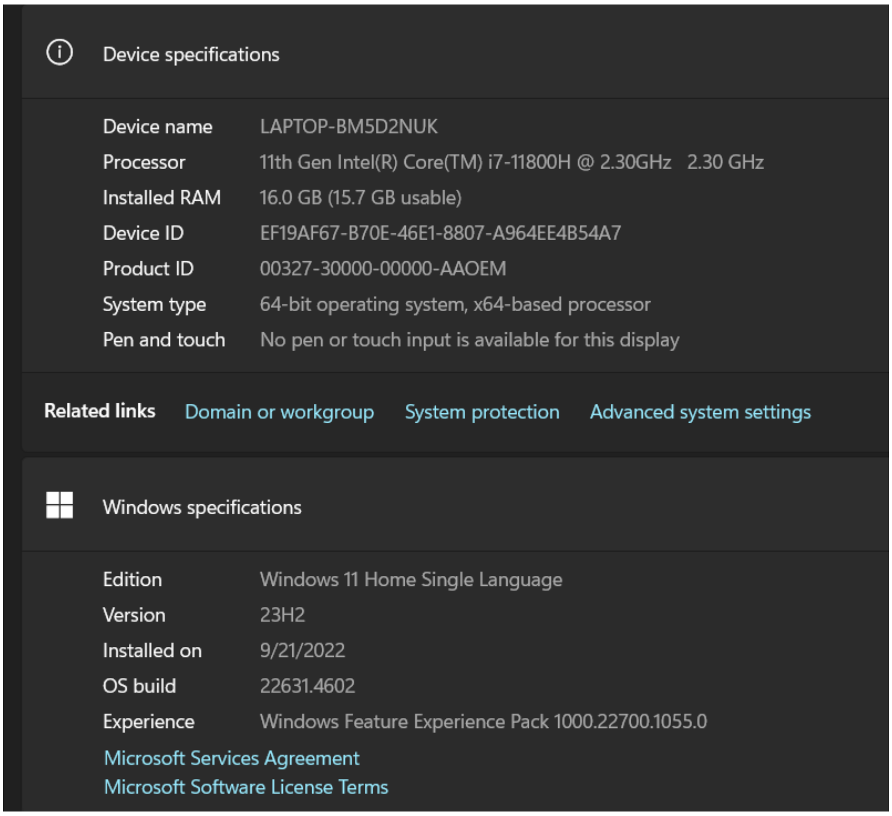


Figure 1: Hardware and software specification

2.2 Software used

```
In [53]: import sys
         import notebook

In [55]: print(" pyhton version" + sys.version)
         print(" jupyter notebook version" + notebook.__version__)

          pyhton version3.11.5 | packaged by Anaconda, Inc. | (main, Sep 11 2023, 13:26:23) [MSC v.1916 64 bit
         (AMD64)]
          jupyter notebook version6.5.4

In [ ]:
```

Figure 2: Jupyter notebook and python version

2.2.1 Jupyter Notebook Installation

1. Drag the cursor on the given link that will redirect to the download page of Anaconda. download the latest version.

   https://www.anaconda.com/download/

2. Install Anaconda on the desktop or else use Google Collabs. Once the installation has been completed the screen appears in the web browser.

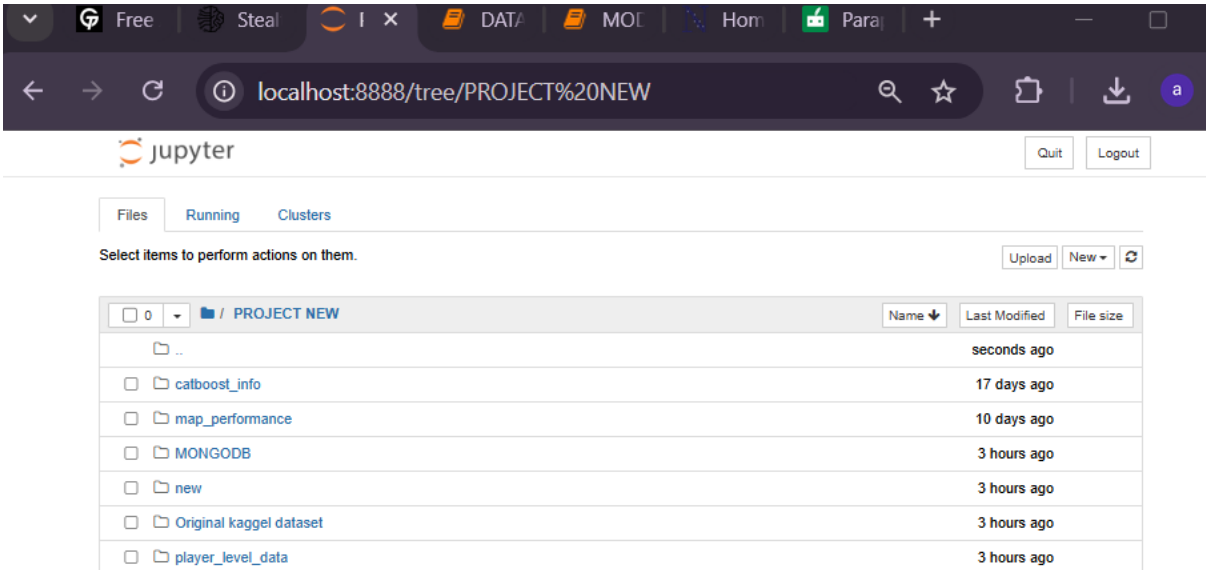


Figure 3: Jupyter Notebook on the local server localhost:8888/tree

3 Data collection

Valorant Champion Tour 2021-2024 Data contains the last four years' data which includes player, map, team, agents and IDs. To download this dataset, click on the given link that will redirect to the Kaggle page. download the CSV file of the dataset

https://www.kaggle.com/datasets/ryanluong1/valorant-champion-tour-2021-2023-data

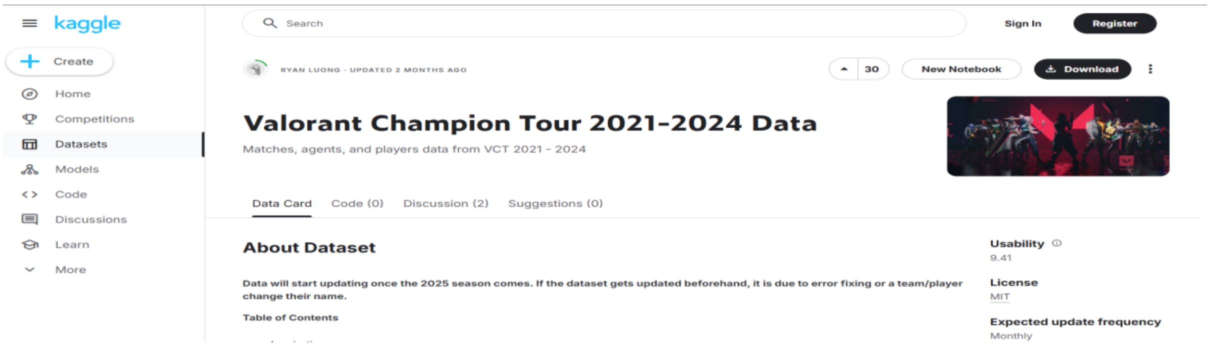


Figure 4: Kaggle dataset using project

4 Project Directory

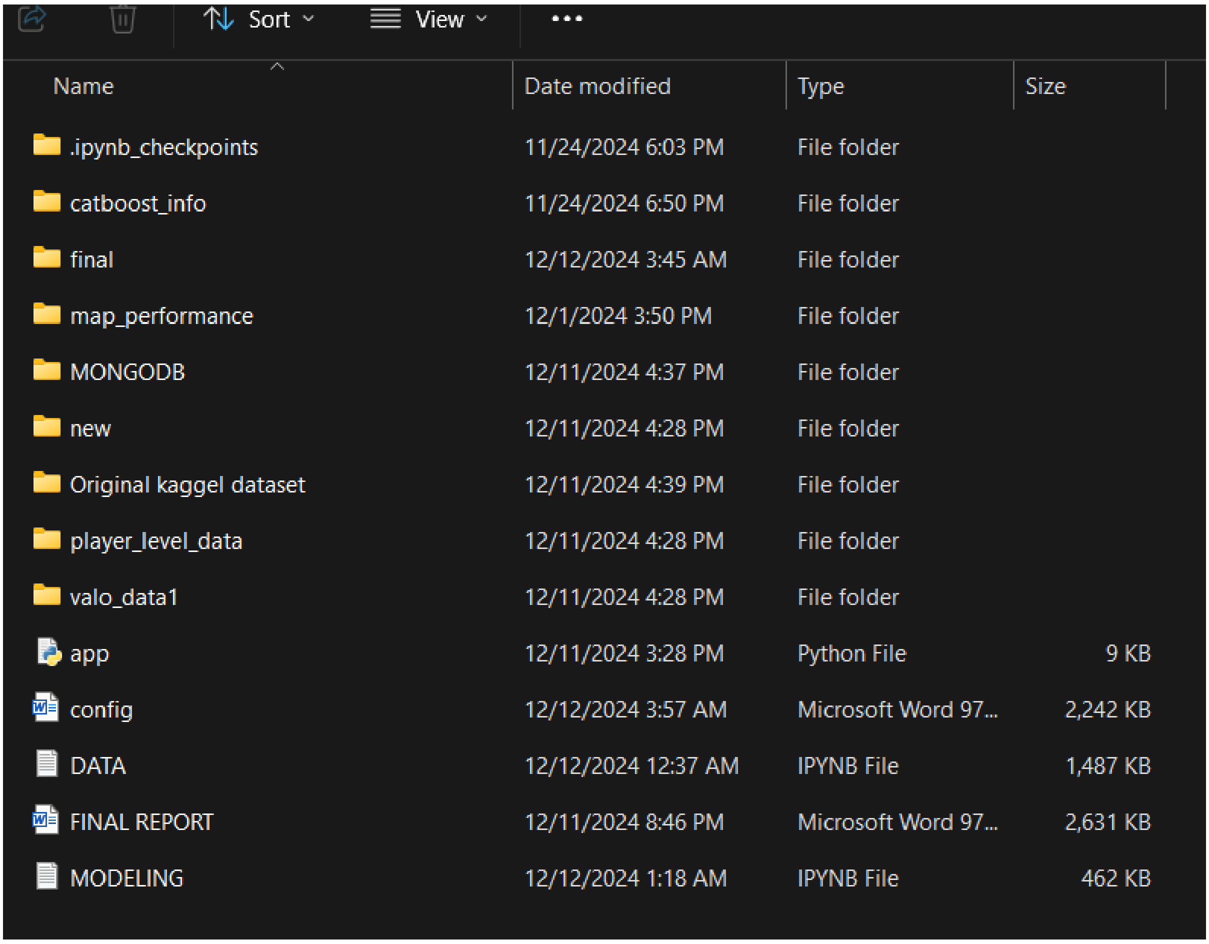


Figure 5 : project directory

5 Library Package Requirement

Important libraries must be imported before running any cells. if packages have never been installed use pip install "library name" in CMD or in jupyter cell. Figure 5 and figure 6 are libraries using data preparation and modeling.

```
In [ ]: #Library
import pandas as pd
# pip install ctgan
import pandas as pd
from ctgan import CTGAN
from sklearn.mixture import GaussianMixture
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
from sklearn.preprocessing import MinMaxScaler
import os
from pymongo import MongoClient
import seaborn as sns
import matplotlib.pyplot as plt
```

Figure 6: library used in data preparation

```
In [1]: # Import necessary libraries
import pandas as pd
from pymongo import MongoClient
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report, roc_auc_score
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import GradientBoostingClassifier, AdaBoostClassifier, ExtraTreesClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.linear_model import RidgeClassifier
import lightgbm as lgb
from catboost import CatBoostClassifier
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import SimpleRNN, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from sklearn.experimental import enable_hist_gradient_boosting  # noqa
from sklearn.ensemble import HistGradientBoostingClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_predict
import pandas as pd
from pymongo import MongoClient
from xgboost import XGBClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
import joblib
import pickle
import io
import bson
```

Figure 7: library used in modeling and deployment

6 Data Merging, Preprocessing, and Synthetic Data Generation

Data merging is a crucial part of this project because was segregated into different CSV files. so in this step, we will merge various data to form four import datasets which is player_level_data, map performance and team data, synthetic_odd_data, and match_data for merging datasets we use Pandas and NumPy. Once data is merged next is data preprocessing in this step data get cleaned removing missing values, duplicates, and nulls after that let's start with data is not perfect for modeling so in this project, synthetic data generation techniques were to tackle the issues like biased data, lack of data and imbalanced data

6.1 Player_level_data

We start off by getting the code to bring in all the required CSV files and merge the data together using Player IDs and Team IDs into a single dataset called player_level_data. Then, data preprocessing is done to remove rows with missing values, duplicates etc. Then, we store the cleaned dataset on a local device. Refer figures 7 and 8 .

In [1]:
```python
import pandas as pd

# Load the datasets
players_stats = pd.read_csv('player_level_data/M/players_stats.csv')
teams_ids = pd.read_csv('player_level_data/M/teams_ids.csv')
tournaments_stages_matches = pd.read_csv('player_level_data/M/tournaments_stages_matches_games_ids.csv')
player_ids = pd.read_csv('player_level_data/M/player_ids.csv')

# Step 1: Rename columns for consistency
players_stats.rename(columns={"Teams": "Team", "Player": "Player Name"}, inplace=True)
teams_ids.rename(columns={"Team": "Team Name"}, inplace=True)
player_ids.rename(columns={"Player": "Player Name"}, inplace=True)

# Step 2: Merge Player Stats with Player IDs
players_stats_merged = pd.merge(players_stats, player_ids, on="Player Name", how="left")

# Step 3: Merge with Team IDs
players_stats_merged = pd.merge(players_stats_merged, teams_ids, left_on="Team", right_on="Team Name", how="left")

# Step 4: Merge with Match Information using "Match Type"
players_stats_merged = pd.merge(players_stats_merged, tournaments_stages_matches, on="Match Type", how="left")

# Step 5: Select and rename relevant columns
final_player_data = players_stats_merged[[
    "Match ID", "Player ID", "Team ID", "Agents", "Kills", "Deaths", "Assists",
    "First Kills", "Average Combat Score", "Headshot %"
]].rename(columns={
    "Agents": "Agent",
    "First Kills": "First_Bloods",
    "Average Combat Score": "ACS",
    "Headshot %": "Headshot_Percentage"
})

# Step 6: Process the Agent column to retain only the first agent in cases of multiple agents
final_player_data["Agent"] = final_player_data["Agent"].apply(lambda x: x.split(',')[0].strip() if isinstance(x, str) else x)

# Step 7: Clean and reformat the data
# Convert percentage columns to float (removing % sign)
final_player_data["Headshot_Percentage"] = final_player_data["Headshot_Percentage"].str.rstrip('%').astype(float)

# Remove duplicates and rows with missing Match ID or Player ID
final_player_data = final_player_data.drop_duplicates().dropna(subset=["Match ID", "Player ID"])

# # Save the final dataset
# output_path = 'new/player_level_data.csv'
# final_player_data.to_csv(output_path, index=False)

# print(f"Cleaned dataset saved at: {output_path}")

final_player_data
```

Figure 8: data merging player level data code.

Out[1]:

	Match ID	Player ID	Team ID	Agent	Kills	Deaths	Assists	First_Bloods	ACS	Headshot_Percentage
0	235363.0	9558	7035.0	cypher	16	16	7	3	209.0	21.0
3	235364.0	9558	7035.0	cypher	16	16	7	3	209.0	21.0
6	235566.0	9558	7035.0	cypher	16	16	7	3	209.0	21.0
8	235567.0	9558	7035.0	cypher	16	16	7	3	209.0	21.0
11	235568.0	9558	7035.0	cypher	16	16	7	3	209.0	21.0
...	...	...	...	...	...	...	...	...	...	...
1747061	378672.0	301	12694.0	raze	64	44	14	7	290.0	23.0
1747063	295619.0	301	12694.0	raze	64	44	14	7	290.0	23.0
1747065	298699.0	301	12694.0	raze	64	44	14	7	290.0	23.0
1747067	297416.0	301	12694.0	raze	64	44	14	7	290.0	23.0
1747070	296749.0	301	12694.0	raze	64	44	14	7	290.0	23.0

689900 rows × 10 columns

Figure 9: Data merging player level data

Now will start with synthetic data generation using CTGAN. The metrics: kills, deaths, assists, ACS, and headshots percentage were merged as it was relative; higher kills contribute to a higher ACS value. Refer figure 9.

In [3]:

```python
import pandas as pd
from ctgan import CTGAN

# Load the uploaded data
data = pd.read_csv('new/player_level_data.csv')


# Step 1: Preprocess 'Headshot %' column
data['Headshot_Percentage'] = data['Headshot_Percentage']

# Step 2: Remove duplicates
data.drop_duplicates(inplace=True)

# Step 3: Limit Kills and Deaths to maximum thresholds
data.loc[data['Kills'] > 52, 'Kills'] = 52
data.loc[data['Deaths'] > 36, 'Deaths'] = 36

# Step 4: Separate original columns to retain
original_columns = ['Match ID', 'Player ID', 'Team ID', 'Agent']
original_data = data[original_columns].dropna()  # Drop rows with NaN values

# Select columns for synthetic data generation
columns_to_synthesize = ['Kills', 'Deaths', 'Assists', 'First_Bloods', 'ACS', 'Headshot_Percentage']
training_data = data[columns_to_synthesize].dropna().sample(n=3000, random_state=42)

# Initialize and train the CTGAN model with the specified constraints
ctgan = CTGAN(epochs=150)
ctgan.fit(training_data, columns_to_synthesize)

# Generate synthetic data and apply limits for consistency
synthetic_data = ctgan.sample(len(original_data))
synthetic_data.columns = columns_to_synthesize
synthetic_data['Kills'] = synthetic_data['Kills'].clip(upper=52)
synthetic_data['Deaths'] = synthetic_data['Deaths'].clip(upper=36)

# Ensure synthetic data reflects professional game dynamics: more Kills/Assists = higher Average Combat Score
synthetic_data['ACS'] = synthetic_data['Kills'] * 10 + synthetic_data['Assists'] * 5

# Step 5: Combine original and synthetic data
final_data = pd.concat([original_data.reset_index(drop=True), synthetic_data], axis=1)


final_data
```

Out[3]:

	Match ID	Player ID	Team ID	Agent	Kills	Deaths	Assists	First_Bloods	ACS	Headshot_Percentage
0	235363.0	9558	7035.0	cypher	13	17	29	1	275	27.0
1	235364.0	9558	7035.0	cypher	43	15	22	5	540	27.0
2	235566.0	9558	7035.0	cypher	45	14	25	3	575	30.0
3	235567.0	9558	7035.0	cypher	52	36	18	26	610	17.0
4	235568.0	9558	7035.0	cypher	52	34	9	8	565	37.0
...	...	...	...	...	...	...	...	...	...	...

Figure 10: CTGAN applied on player-level data

After the data is successfully generated, it is stored in a CSV file nsanew/player_level_data_main_syn.csv.

6.2 Map Performance And Team Data

It imports and cleans datasets related to map performance, team stats and pick rates, standardizing column names and converting percentages to floats. After this, it merges these datasets on keys such as "Tournament" and "Map", and creates a single, cleaned dataset that is stored locally new/map_performance_team_data.csv for further analysis. No synthetic data is generate in this step. As mentioned in the figure 10 and 11.

Map Performance And Team Data

In [6]:
```python
import pandas as pd

# File paths for datasets
maps_stats_path = 'map_performance/M/maps_stats.csv'
teams_picked_agents_path = 'map_performance/M/teams_picked_agents.csv'
agents_pick_rates_path = 'map_performance/M/agents_pick_rates.csv'
teams_ids_path = 'map_performance/M/teams_ids.csv'
tournaments_path = 'map_performance/M/tournaments_stages_matches_games_ids.csv'

# Step 1: Load the datasets
maps_stats = pd.read_csv(maps_stats_path)
teams_picked_agents = pd.read_csv(teams_picked_agents_path)
agents_pick_rates = pd.read_csv(agents_pick_rates_path)
teams_ids = pd.read_csv(teams_ids_path)
tournaments_stages_matches = pd.read_csv(tournaments_path)

# Step 2: Clean and Standardize Datasets

# Clean maps_stats
maps_stats = maps_stats.rename(columns={
    "Total Maps Played": "total_maps_played",
    "Attacker Side Win Percentage": "attacker_win_pct",
    "Defender Side Win Percentage": "defender_win_pct"
})
maps_stats["attacker_win_pct"] = maps_stats["attacker_win_pct"].str.rstrip('%').astype(float)
maps_stats["defender_win_pct"] = maps_stats["defender_win_pct"].str.rstrip('%').astype(float)

# Clean teams_picked_agents
teams_picked_agents = teams_picked_agents.rename(columns={
    "Total Wins By Map": "wins_by_map",
    "Total Loss By Map": "loss_by_map",
    "Total Maps Played": "maps_played"
})

# Clean agents_pick_rates
agents_pick_rates = agents_pick_rates.rename(columns={
    "Pick Rate": "pick_rate"
})
agents_pick_rates["pick_rate"] = agents_pick_rates["pick_rate"].str.rstrip('%').astype(float)

# Ensure consistency in teams_ids
teams_ids.rename(columns={"Team": "team"}, inplace=True)

# Step 3: Merge Datasets

# Merge teams_picked_agents with maps_stats
merged_maps_agents = pd.merge(
    teams_picked_agents, maps_stats,
    on=["Tournament", "Stage", "Match Type", "Map"],
    how="left"
)
```

Figure 11: Map Performance And Team Data code

Out[6]:

	Match ID	Tournament	Stage	Map	Team	Team ID	maps_played	wins_by_map	loss_by_map	total_maps_played	attacker_win_pct	defender_win_p
0	235363.0	Champions Tour 2023: EMEA Last Chance Qualifier	Playoffs	Fracture	Team Heretics	1001	1	0	1	1	71.0	29
1	235363.0	Champions Tour 2023: EMEA Last Chance Qualifier	Playoffs	Fracture	Team Heretics	1001	1	0	1	1	71.0	29
5	235363.0	Champions Tour 2023: EMEA Last Chance Qualifier	Playoffs	Fracture	KOI	7035	1	1	0	1	71.0	29
7	235363.0	Champions Tour 2023: EMEA Last Chance Qualifier	Playoffs	Fracture	KOI	7035	1	1	0	1	71.0	29
10	235363.0	Champions Tour 2023: EMEA Last Chance Qualifier	Playoffs	Pearl	Team Heretics	1001	1	1	0	1	26.0	74
...	...	...	...	...	...	...	...	...	...	...	...	
46869	296749.0	Champions Tour 2024: Pacific Kickoff	Group Stage	Breeze	Rex Regum Qeon	878	1	1	0	1	70.0	30
46872	296749.0	Champions Tour 2024: Pacific Kickoff	Group Stage	Icebox	Gen.G	17	1	1	0	1	63.0	38
46876	296749.0	Champions Tour 2024: Pacific Kickoff	Group Stage	Icebox	Gen.G	17	1	1	0	1	63.0	38
46877	296749.0	Champions Tour 2024: Pacific Kickoff	Group Stage	Icebox	Rex Regum Qeon	878	1	0	1	1	63.0	38
46880	296749.0	Champions Tour 2024: Pacific Kickoff	Group Stage	Icebox	Rex Regum Qeon	878	1	0	1	1	63.0	38

22279 rows × 15 columns

In [7]:
```
# Step 6: Save the final dataset
output_path = 'new/map_performance_team_data.csv'
final_dataset.to_csv(output_path, index=False)

print(f"Optimized dataset saved at: {output_path}")
```

Figure 12: Map Performance and Team Data output

6.3 Synthetic Data Generation

The code begins by loading all necessary CSVs specifying both tournament match details and team ID. Implied probabilities for each team are generated using Gaussian Mixture Models (GMM) and simulated odds are calculated by applying random adjustments. Entries in the dataset and the cleaned dataset are duplicated removed, and saved for analysis. As mentioned in the figure 12 and 13.

synthetic_odd_data

In [8]:
```
import numpy as np
import pandas as pd
from sklearn.mixture import GaussianMixture

# Load the datasets
tournaments_stages_matches_games_ids = pd.read_csv('valo_data1/tournaments_stages_matches_games_ids.csv')
teams_ids = pd.read_csv('valo_data1/teams_ids.csv')

#Split the 'Match Name' column to extract 'Team_A' and 'Team_B'
tournaments_stages_matches_games_ids[['Team_A', 'Team_B']] = tournaments_stages_matches_games_ids['Match Name'].str.split(' vs ',

#  Merge to get Team_A_ID and Team_B_ID
synthetic_dataset = tournaments_stages_matches_games_ids.merge(
    teams_ids, how='left', left_on='Team_A', right_on='Team'
).rename(columns={'Team ID': 'Team_A_ID'}).drop(columns=['Team'])

synthetic_dataset = synthetic_dataset.merge(
    teams_ids, how='left', left_on='Team_B', right_on='Team'
).rename(columns={'Team ID': 'Team_B_ID'}).drop(columns=['Team'])

 #Generate Implied Probabilities using GMM
np.random.seed(42)
gmm = GaussianMixture(n_components=2, random_state=42)
gmm.fit(np.array([0.35, 0.5, 0.65]).reshape(-1, 1))  # Adjusted range for a realistic spread

# Generate probabilities for Team A with randomness
implied_probs_a = gmm.sample(len(synthetic_dataset))[0].flatten()
synthetic_dataset['Implied_Prob_A'] = np.clip(implied_probs_a + np.random.normal(0, 0.02, len(implied_probs_a)), 0.35, 0.65)
synthetic_dataset['Implied_Prob_B'] = np.clip(1 - synthetic_dataset['Implied_Prob_A'] + np.random.normal(0, 0.02, len(implied_pro

# Step 4: Calculate Simulated Odds with Added Variability on both A and B odds
random_adjustment_a = np.random.normal(1.0, 0.05, size=len(synthetic_dataset))
random_adjustment_b = np.random.normal(1.0, 0.05, size=len(synthetic_dataset))

synthetic_dataset['Simulated_Odds_A'] = np.clip(1 / synthetic_dataset['Implied_Prob_A'] * random_adjustment_a, 1, 10)
synthetic_dataset['Simulated_Odds_B'] = np.clip(1 / synthetic_dataset['Implied_Prob_B'] * random_adjustment_b, 1, 10)

# Step 5: Remove duplicate Match IDs and select relevant columns for the final dataset
synthetic_dataset_cleaned = synthetic_dataset.drop_duplicates(subset=['Match ID'])
final_synthetic_dataset = synthetic_dataset_cleaned[[
    'Match ID', 'Match Name', 'Team_A', 'Team_A_ID', 'Team_B', 'Team_B_ID',
    'Implied_Prob_A', 'Implied_Prob_B', 'Simulated_Odds_A', 'Simulated_Odds_B'
]]

final_synthetic_dataset
```

Figure 13: Synthetic Data Generation using GMM

Out[8]:

	Match ID	Match Name	Team_A	Team_A_ID	Team_B	Team_B_ID	Implied_Prob_A	Implied_Prob_B	Simulated_Odds_A	Simulated_Odds_B
0	247100	Team Liquid vs Natus Vincere	Team Liquid	474	Natus Vincere	4915	0.483578	0.569309	2.082384	1.720461
2	247101	DRX vs LOUD	DRX	8185	LOUD	6961	0.420414	0.584661	2.213269	1.811065
5	247087	FUT Esports vs T1	FUT Esports	1184	T1	14	0.477948	0.524012	1.937772	1.873907
7	247086	Evil Geniuses vs FunPlus Phoenix	Evil Geniuses	5248	FunPlus Phoenix	11328	0.481102	0.511065	2.202433	1.934132
9	247102	Natus Vincere vs DRX	Natus Vincere	4915	DRX	8185	0.400931	0.592205	2.524499	1.810348
...	...	...	...	...	...	...	...	...	...	...
1922	297253	ZETA DIVISION vs Team Secret	ZETA DIVISION	5448	Team Secret	6199	0.630619	0.350000	1.456858	3.091030
1924	297255	Gen.G vs ZETA DIVISION	Gen.G	17	ZETA DIVISION	5448	0.650000	0.350000	1.561343	2.673486
1926	297256	T1 vs Paper Rex	T1	14	Paper Rex	624	0.600495	0.390466	1.641575	2.608762
1928	297257	DRX vs Gen.G	DRX	8185	Gen.G	17	0.650000	0.350000	1.776792	3.077534
1930	297258	Paper Rex vs Gen.G	Paper Rex	624	Gen.G	17	0.624944	0.367619	1.640958	2.665237

765 rows × 10 columns

Figure 14: Synthetic Data Generation using GMM Output

6.4 Match Data Using RF

First it imports required libraries and datasets to begin the code with tournament match details and synthetic odds. We have merged these datasets on Match ID derived a Simulated Winner column based on simulated odds and prepared the data for modeling by selecting features relevant to modeling and converting target labels to binary. The data is used to train the Random Forest Classifier that predicts wins. Files is stored locally at new/match_data.csv. Refer to figures 14 and 15

match_data using RF

In [10]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
import numpy as np

# Load the datasets
tournaments_stages_matches = pd.read_csv('valo_data1/tournaments_stages_matches_games_ids.csv')
simulated_odds = pd.read_csv('new/synthetic_odd_data.csv')

# Merge the datasets on 'Match ID'
merged_data = pd.merge(tournaments_stages_matches, simulated_odds, on='Match ID')

# Define the winner based on the simulated odds
merged_data['Simulated_Winner'] = merged_data.apply(
    lambda row: 'Team_A' if row['Simulated_Odds_A'] < row['Simulated_Odds_B'] else 'Team_B', axis=1
)

# Prepare data for modeling
X = merged_data[['Simulated_Odds_A', 'Simulated_Odds_B', 'Implied_Prob_A', 'Implied_Prob_B']]
y = merged_data['Simulated_Winner'].apply(lambda x: 1 if x == 'Team_A' else 0)  # Convert to binary

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Train the Random Forest Classifier
rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X_train, y_train)

# Evaluate the model
y_pred = rf_model.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))

# Generate synthetic outcomes for the entire dataset
merged_data['Predicted_Team_A_Win'] = rf_model.predict(X)
merged_data['Predicted_Winner'] = merged_data['Predicted_Team_A_Win'].apply(lambda x: 'Team_A' if x == 1 else 'Team_B')
```

```
Accuracy: 1.0
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00       319
           1       1.00      1.00      1.00       262

    accuracy                           1.00       581
   macro avg       1.00      1.00      1.00       581
weighted avg       1.00      1.00      1.00       581
```

Figure 15: Generate winning using random forest

```
In [11]: # Prepare the final dataset with necessary columns, remove Team_A_Win and convert "Winner" to binary
synthetic_m = merged_data[['Match ID', 'Stage', 'Team_A', 'Team_B',
                           'Map', 'Tournament', 'Simulated_Odds_A', 'Simulated_Odds_B', 'Predicted_Winner']]

# Rename columns to the required format
synthetic_m.columns = ['Match ID', 'Date', 'Team_A', 'Team_B', 'Map', 'Tournament',
                       'Odds_A', 'Odds_B', 'Winner']

# Convert the "Winner" column to binary: Team_A -> 1, Team_B -> 0
synthetic_m['Winner'] = synthetic_m['Winner'].apply(lambda x: 1 if x == 'Team_A' else 0)

# Drop duplicate Match ID rows to ensure unique matches
synthetic_match = synthetic_m.drop_duplicates(subset=['Match ID'])

# Display the final synthetic match dataset
synthetic_match

C:\Users\ASUS\AppData\Local\Temp\ipykernel_21936\1739003082.py:10: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-v
rsus-a-copy
  synthetic_m['Winner'] = synthetic_m['Winner'].apply(lambda x: 1 if x == 'Team_A' else 0)
```

Out[11]:

	Match ID	Date	Team_A	Team_B	Map	Tournament	Odds_A	Odds_B	Winner
0	247100	Group Stage	Team Liquid	Natus Vincere	Fracture	Valorant Champions 2023	2.082384	1.720461	0
2	247101	Group Stage	DRX	LOUD	Lotus	Valorant Champions 2023	2.213269	1.811065	0
5	247087	Group Stage	FUT Esports	T1	Pearl	Valorant Champions 2023	1.937772	1.873907	0
7	247086	Group Stage	Evil Geniuses	FunPlus Phoenix	Ascent	Valorant Champions 2023	2.202433	1.934132	0
9	247102	Group Stage	Natus Vincere	DRX	Bind	Valorant Champions 2023	2.524499	1.810348	0
...	...	...	...	...	...	...	...	...	...
1922	297253	Play-Ins	ZETA DIVISION	Team Secret	Icebox	Champions Tour 2024: Pacific Kickoff	1.456858	3.091030	1
1924	297255	Play-Ins	Gen.G	ZETA DIVISION	Icebox	Champions Tour 2024: Pacific Kickoff	1.561343	2.673486	1
1926	297256	Playoffs	T1	Paper Rex	Split	Champions Tour 2024: Pacific Kickoff	1.641575	2.608762	1
1928	297257	Playoffs	DRX	Gen.G	Bind	Champions Tour 2024: Pacific Kickoff	1.776792	3.077534	1
1930	297258	Playoffs	Paper Rex	Gen.G	Ascent	Champions Tour 2024: Pacific Kickoff	1.640958	2.665237	1

765 rows × 9 columns

Figure 16: Merge generated synthetic data and output

All merging, preprocessing, and synthetic data creation files generated are moved to the valo1 folder in the project directory as well as additional CSV files like team_id, player_id, and tournaments stages matches games IDs.

7 Uploading All The Datasets In MongoDB

uploading all the datasets in MongoDB from the valo1 folder. use "mongodb://localhost:27017/" to connect with mongo server. Once the connection is established files are moved to the valorant database, in the valo1 collection. Refer figure 16 and 17.

uploading all the dataset in mongoDB

move all the generated dataset from new folder to valo1

```
In [9]: #pip install pandas pymongo
import os
from pymongo import MongoClient

client = MongoClient("mongodb://localhost:27017/")
db = client["valo1"]

In [10]: folder_path = r"C:\Users\ASUS\PROJECT NEW\valo_data1"

In [11]: player_file = "player_level_data_main_syn.csv"
map_perfo = "map_performance_team_data.csv"
odds_file = "synthetic_odd_data.csv"
playerid = "player_ids.csv"
teamsids = "teams_ids.csv"
tournaments_stages_matches_games_ids = "tournaments_stages_matches_games_ids.csv"
Match_Data_Team_Level = "match_data.csv"

In [12]: def extract_csv_to_mongodb(file_path, collection_name):
    try:
        # Read the CSV file into a DataFrame
        df = pd.read_csv(file_path)

        # Convert the DataFrame to a list of dictionaries
        data = df.to_dict(orient='records')

        # Insert data into the MongoDB collection
        db[collection_name].insert_many(data)

        print(f"Data from {file_path} has been successfully loaded into the collection '{collection_name}'.")
    except Exception as e:
        print(f"Error loading data from {file_path}: {e}")

In [27]: extract_csv_to_mongodb(os.path.join(folder_path, player_file), "player_level_data")
extract_csv_to_mongodb(os.path.join(folder_path, map_perfo), "map_performance")
extract_csv_to_mongodb(os.path.join(folder_path, odds_file), "simulated_odds")
extract_csv_to_mongodb(os.path.join(folder_path, playerid), "player_id")
extract_csv_to_mongodb(os.path.join(folder_path, tournaments_stages_matches_games_ids), "tournaments_stages_matches_games_ids")
extract_csv_to_mongodb(os.path.join(folder_path, teamsids), "teams_ids")
extract_csv_to_mongodb(os.path.join(folder_path, Match_Data_Team_Level), "Match_Data")

Data from C:\Users\ASUS\PROJECT NEW\valo_data1\player_level_data_main_syn.csv has been successfully loaded into the collection
'player_level_data'.
Data from C:\Users\ASUS\PROJECT NEW\valo_data1\map_performance_team_data.csv has been successfully loaded into the collection
'map_performance'.
```

Figure 17: Code to fetch file and insert into MongoDB

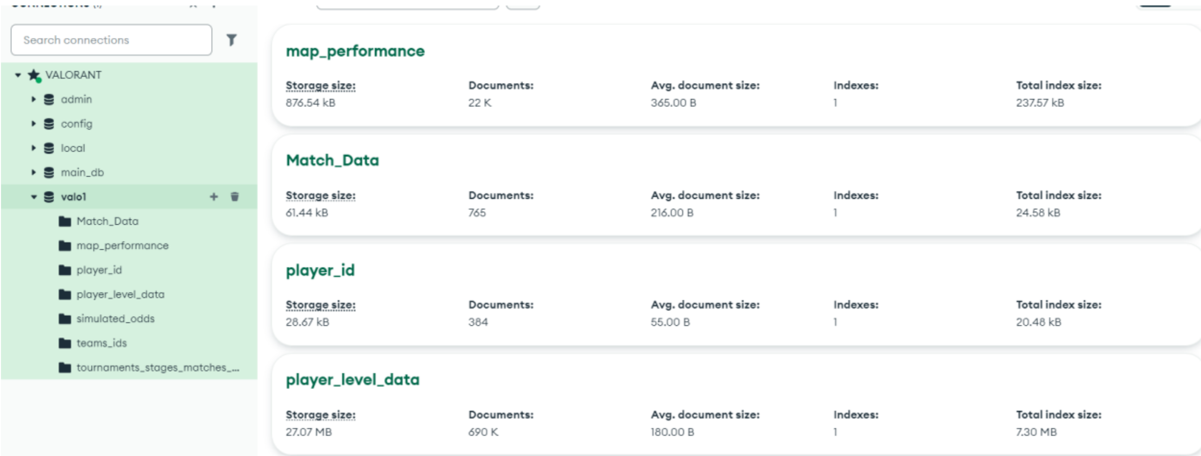


Figure 18: file successfull uploaded into database

8 Fetching data from MongoDB

Now data is uploaded successfully into the database it is time to fetch data from the database for further execution. Refer figure 18.

In [14]:
```
# Import necessary libraries
import pandas as pd
from pymongo import MongoClient

# Connect to MongoDB
client = MongoClient("mongodb://localhost:27017/")
db = client["valo1"]

# Function to fetch and display data from a specific collection
def display_collection_data(collection_name, limit=5):
    # Fetch the collection
    collection = db[collection_name]

    # Retrieve data with a limit for display
    data = list(collection.find().limit(limit))

    # Convert to DataFrame for better readability
    df = pd.DataFrame(data)

    # Display the data
    print(f"Data from collection '{collection_name}':")
    display(df)


display_collection_data("player_level_data")
```

```
Data from collection 'player_level_data':
```

	_id	Match ID	Player ID	Team ID	Agent	Kills	Deaths	Assists	First_Bloods	ACS	Headshot_Percentage
0	6759bdc67f877ea757c15fe2	235363.0	9558	7035.0	cypher	11	19	4	12	130	22.0
1	6759bdc67f877ea757c15fe3	235364.0	9558	7035.0	cypher	33	11	4	2	350	35.0
2	6759bdc67f877ea757c15fe4	235566.0	9558	7035.0	cypher	31	12	16	0	390	19.0
3	6759bdc67f877ea757c15fe5	235567.0	9558	7035.0	cypher	9	32	13	7	155	27.0
4	6759bdc67f877ea757c15fe6	235568.0	9558	7035.0	cypher	35	18	3	0	365	25.0

In [15]:
```
display_collection_data("simulated_odds")
display_collection_data("tournaments_stages_match_types_ids")
display_collection_data("tournaments_stages_matches_games_ids")
display_collection_data("teams_ids")
display_collection_data("Match_Data")
```

```
Data from collection 'simulated_odds':
```

Figure 19: Fetching data from MongoDB

```
In [20]: win_counts = match_data['Winner'].value_counts()
         sns.barplot(x=win_counts.index, y=win_counts.values)
         plt.title("Win Count by Team")
         plt.show()
```

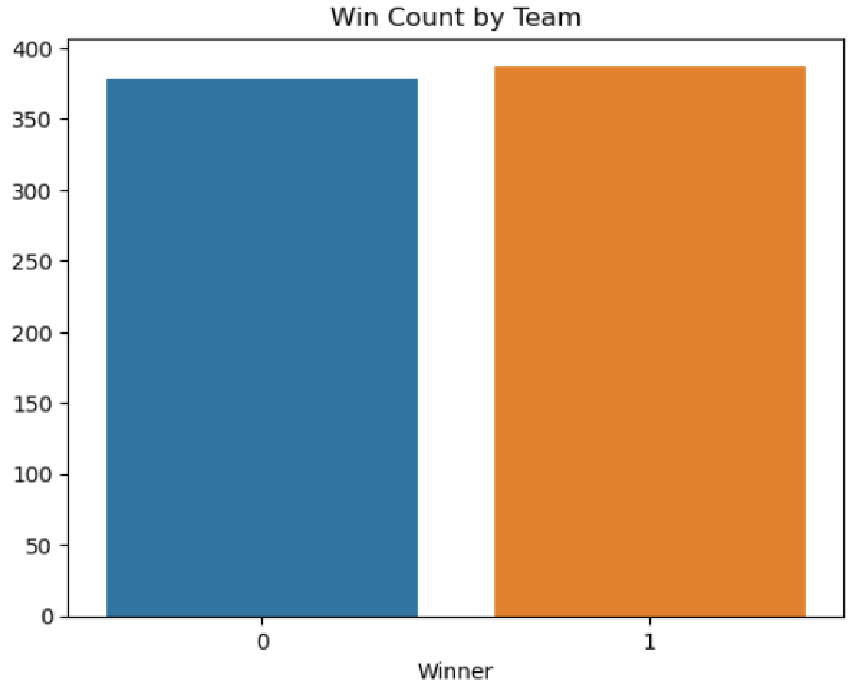
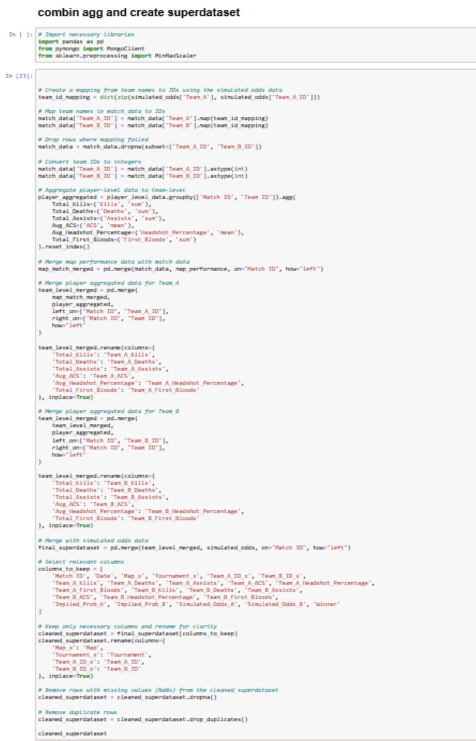


Figure 20: To check data is not bias or imbalance

9 SUPERDATASET



combin agg and create superdataset

```
In [ ]: # Import necessary libraries
import pandas as pd
from pymongo import MongoClient
from sklearn.preprocessing import MinMaxScaler
```

```
In [23]:

# Create a mapping from team names to IDs using the simulated odds data
team_id_mapping = dict(zip(simulated_odds['Team A'], simulated_odds['Team A ID']))

# Map team names in match data to IDs
match_data['Team A ID'] = match_data['Team A'].map(team_id_mapping)
match_data['Team B ID'] = match_data['Team B'].map(team_id_mapping)

# Drop rows where mapping failed
match_data = match_data.dropna(subset=['Team A ID', 'Team B ID'])

# Convert team IDs to integers
match_data['Team A ID'] = match_data['Team A ID'].astype(int)
match_data['Team B ID'] = match_data['Team B ID'].astype(int)

# Aggregate player-level data to team-level
player_aggregated = player_level_data.groupby(['Match ID', 'Team ID']).agg(
    Total_Kills=('Kills', 'sum'),
    Total_Deaths=('Deaths', 'sum'),
    Total_Assists=('Assists', 'sum'),
    Avg_ACS=('ACS', 'mean'),
    Avg_Headshot_Percentage=('Headshot Percentage', 'mean'),
    Total_First_Bloods=('First Bloods', 'sum')
).reset_index()

# Merge map performance data with match data
map_match_merged = pd.merge(match_data, map_performance, on="Match ID", how="left")

# Merge player aggregated data for Team A
team_level_merged = pd.merge(
    map_match_merged,
    player_aggregated,
    left_on=["Match ID", "Team A ID"],
    right_on=["Match ID", "Team ID"],
    how="left"
)

team_level_merged.rename(columns={
    'Total_Kills': 'Team A Kills',
    'Total_Deaths': 'Team A Deaths',
    'Total_Assists': 'Team A Assists',
    'Avg_ACS': 'Team A ACS',
    'Avg_Headshot_Percentage': 'Team A Headshot Percentage',
    'Total_First_Bloods': 'Team A First Bloods'
}, inplace=True)

# Merge player aggregated data for Team B
team_level_merged = pd.merge(
    team_level_merged,
    player_aggregated,
    left_on=["Match ID", "Team B ID"],
    right_on=["Match ID", "Team ID"],
    how="left"
)

team_level_merged.rename(columns={
    'Total_Kills': 'Team B Kills',
    'Total_Deaths': 'Team B Deaths',
    'Total_Assists': 'Team B Assists',
    'Avg_ACS': 'Team B ACS',
    'Avg_Headshot_Percentage': 'Team B Headshot Percentage',
    'Total_First_Bloods': 'Team B First Bloods'
}, inplace=True)

# Merge with simulated odds data
final_superdataset = pd.merge(team_level_merged, simulated_odds, on="Match ID", how="left")

# Select relevant columns
columns_to_keep = [
    'Match ID', 'Date', 'Map_x', 'Tournament_x', 'Team A ID_x', 'Team B ID_x',
    'Team A Kills', 'Team A Deaths', 'Team A Assists', 'Team A ACS', 'Team A Headshot Percentage',
    'Team A First Bloods', 'Team B Kills', 'Team B Deaths', 'Team B Assists',
    'Team B ACS', 'Team B Headshot Percentage', 'Team B First Bloods',
    'Implied Prob A', 'Implied Prob B', 'Simulated Odds A', 'Simulated Odds B', 'Winner'
]

# Keep only necessary columns and rename for clarity
cleaned_superdataset = final_superdataset[columns_to_keep]
cleaned_superdataset.rename(columns={
    'Map_x': 'Map',
    'Tournament_x': 'Tournament',
    'Team A ID_x': 'Team A ID',
    'Team B ID_x': 'Team B ID'
}, inplace=True)

# Remove rows with missing values (NaNs) from the cleaned_superdataset
cleaned_superdataset = cleaned_superdataset.dropna()

# Remove duplicate rows
cleaned_superdataset = cleaned_superdataset.drop_duplicates()

cleaned_superdataset
```

Figure 21: SUPERDATASET

The data is combine and aggregated to create a superdataaset using match data, player stats, map, and simulated odds. Combine using player IDs, match IDs, and team IDs. And then save it in a new database named main_db and collection name superdataset1. Refer figure 20 and 21.

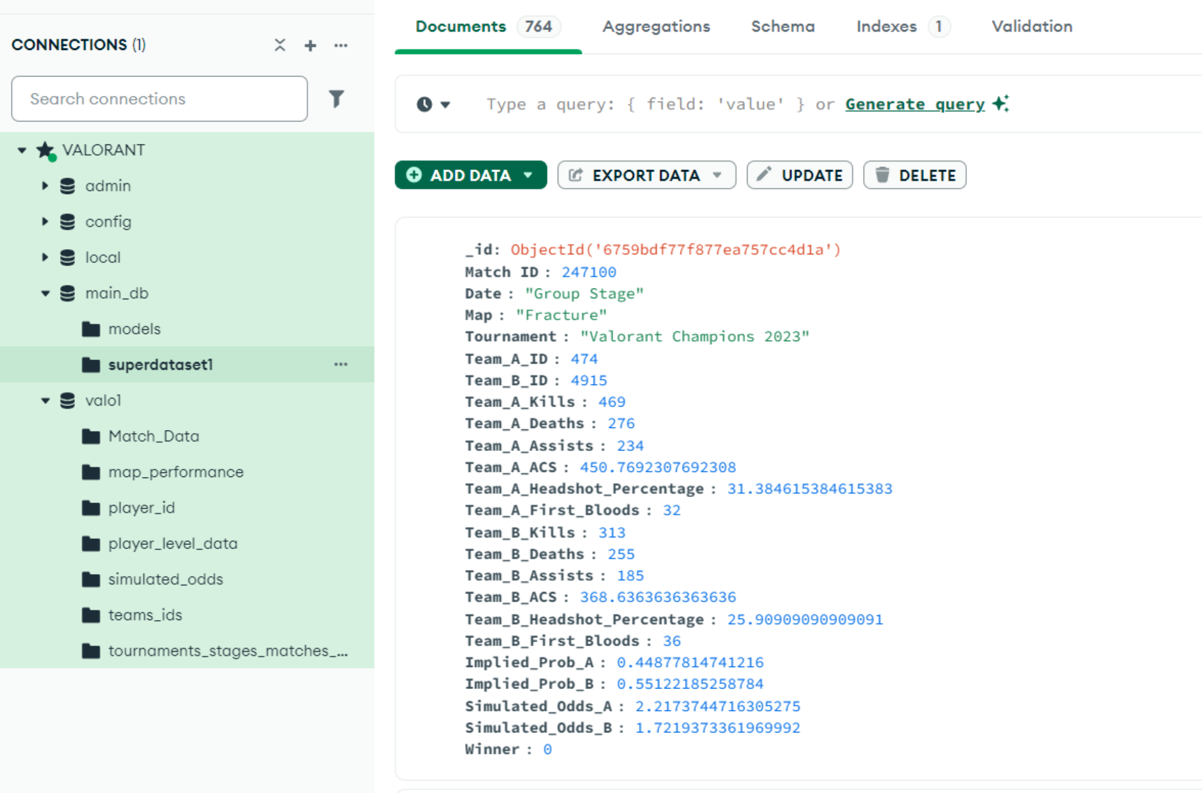


Figure 22: Superdataset move to MongoDB

10 Modeling

Firstly, connect to the new database name superdataset1 then fetch data. Drop MongoDB's default `_id` column if present then Relevant columns for features and targets are selected and displayed data frame. Refer figure 22.

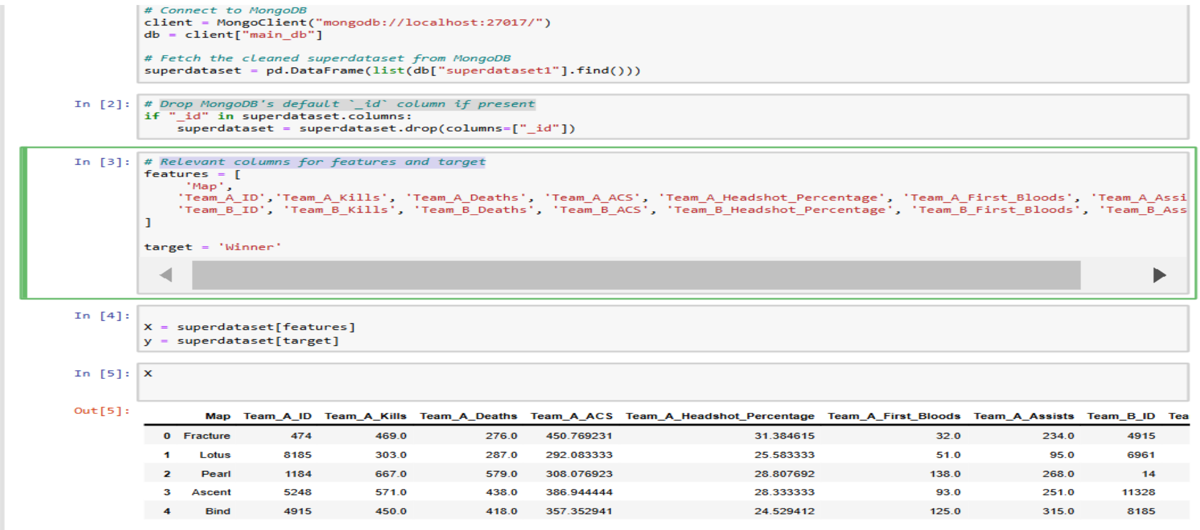


Figure 23: Modeling connection and feature selection

Labelencoder is used to convert map names into numerical refer figure 23.

In [6]:
```python
from sklearn.preprocessing import LabelEncoder, StandardScaler

# Encode categorical features
categorical_features = ['Map']
label_encoders = {col: LabelEncoder() for col in categorical_features}

for col in categorical_features:
    X[col] = label_encoders[col].fit_transform(X[col])
```

```
C:\Users\ASUS\AppData\Local\Temp\ipykernel_24600\3655108482.py:8: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ve
rsus-a-copy
  X[col] = label_encoders[col].fit_transform(X[col])
```

In [7]:
```python
X
```

Out[7]:

	Map	Team_A_ID	Team_A_Kills	Team_A_Deaths	Team_A_ACS	Team_A_Headshot_Percentage	Team_A_First_Bloods	Team_A_Assists	Team_B_ID	Team_
0	4	474	469.0	276.0	450.769231	31.384615	32.0	234.0	4915	
1	7	8185	303.0	287.0	292.083333	25.583333	51.0	95.0	6961	
2	8	1184	667.0	579.0	308.076923	28.807692	138.0	268.0	14	
3	1	5248	571.0	438.0	386.944444	28.333333	93.0	251.0	11328	
4	2	4915	450.0	418.0	357.352941	24.529412	125.0	315.0	8185	
...	...	...	...	...	...	...	...	...	...	
759	6	5448	250.0	196.0	300.909091	27.000000	60.0	162.0	6199	
760	6	17	633.0	441.0	436.388889	29.611111	83.0	305.0	5448	
761	9	14	295.0	294.0	340.000000	31.545455	37.0	158.0	624	
762	2	8185	933.0	786.0	333.382353	30.470588	132.0	401.0	17	
763	1	624	2060.0	1765.0	329.610390	27.779221	365.0	956.0	17	

764 rows × 15 columns

Figure 24 Labelencoder

In [8]:
```python
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
```

In [9]:
```python
# Scale the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Figure 25: Splitting data training and test and using standard scaler

Superdataset is split into train and test 80:20 ratio also used standard scaler for reference figure 24.

11 Model pipeline

Model pipeline is created using This script either evaluates a machine learning model's performance by using accuracy, F1 score, ROC AUC, precision, and recall or it visualizes a confusion matrix. Refer figure 25.

MODEL PIPELINE

```
In [10]: from sklearn.metrics import accuracy_score, f1_score, roc_auc_score, precision_score, recall_score, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

def evaluate_model(model, X_test, y_test, model_name):
    y_pred = model.predict(X_test)
    y_pred_proba = model.predict_proba(X_test)[:, 1]

    # Metrics
    accuracy = accuracy_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    roc_auc = roc_auc_score(y_test, y_pred_proba)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    cm = confusion_matrix(y_test, y_pred)

    # Print metrics
    print(f"{model_name} Performance:")
    print(f"Accuracy: {accuracy:.2f}")
    print(f"F1sn-Score: {f1:.2f}")
    print(f"ROC-AUC: {roc_auc:.2f}")
    print(f"Precision: {precision:.2f}")
    print(f"Recall: {recall:.2f}")

    # Plot confusion matrix
    plt.figure(figsize=(6, 4))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Class 0', 'Class 1'], yticklabels=['Class 0', 'Class 1'])
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.title(f"{model_name} Confusion Matrix")
    plt.show()

    return accuracy, f1, roc_auc, precision, recall

# Initialize the results dictionary to store evaluation metrics
results = {}
```

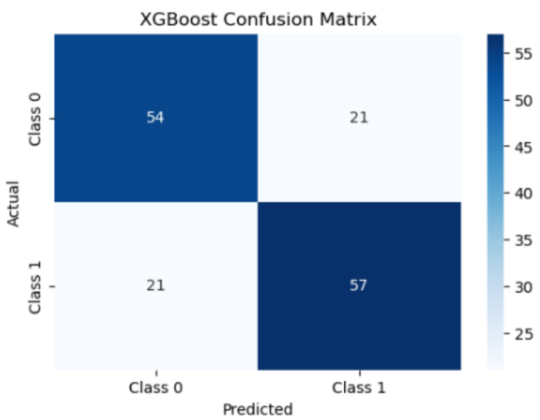
Figure 26: Model pipeline evaluation.

Multiple machine models were implemented such as Logistic Regression, Random Forest and Gradient Boosting, Bagging, AdaBoost, Recurrent Neural Network and hybrid model, etc., and evaluated using accuracy, f1, precision, and recall refer to figures 26 and 27.

```
In [11]:
# Convert results to DataFrame
results_df = pd.DataFrame(results, index=['Accuracy', 'F1-Score', 'ROC-AUC', 'Precision', 'Recall'])


# Model 1: XGBoost
xgb_model = XGBClassifier(n_estimators=200, max_depth=4, learning_rate=0.05, random_state=42)
xgb_model.fit(X_train, y_train)
results['XGBoost'] = evaluate_model(xgb_model, X_test, y_test, "XGBoost")
```

```
XGBoost Performance:
Accuracy: 0.73
F1-Score: 0.73
ROC-AUC: 0.76
Precision: 0.73
Recall: 0.73
```




```
In [12]: # Model 3: Logistic Regression
from sklearn.linear_model import LogisticRegression
lr_model = LogisticRegression(random_state=42, max_iter=100)
lr_model.fit(X_train, y_train)
results['Logistic Regression'] = evaluate_model(lr_model, X_test, y_test, "Logistic Regression")
```

Figure 27 machine learning model

```
Model Comparison:
            XGBoost  Logistic Regression  Naive Bayes  Decision Tree  \
Accuracy   0.725490             0.660131     0.673203       0.549020
F1-Score   0.734177             0.657895     0.683544       0.549020
ROC-AUC    0.742735             0.708547     0.714701       0.549231
Precision  0.725000             0.675676     0.675000       0.560000
Recall     0.743590             0.641026     0.692308       0.538462

           Gradient Boosting  AdaBoost       SVM       KNN  CatBoost  \
Accuracy            0.705882  0.627451  0.705882  0.620915  0.718954
F1-Score            0.701987  0.636943  0.716981  0.632911  0.726115
ROC-AUC             0.747863  0.673333  0.745128  0.656068  0.762735
Precision           0.726027  0.632911  0.703704  0.625000  0.721519
Recall              0.679487  0.641026  0.730769  0.641026  0.730769

            Bagging       HGB       RNN
Accuracy   0.673203  0.699346  0.686275
F1-Score   0.687500  0.705128  0.700000
ROC-AUC    0.717607  0.739829  0.734359
Precision  0.670732  0.705128  0.682927
Recall     0.705128  0.705128  0.717949
```

Figure 28 Evaluation

Create a histogram for model comparison refer figure 28

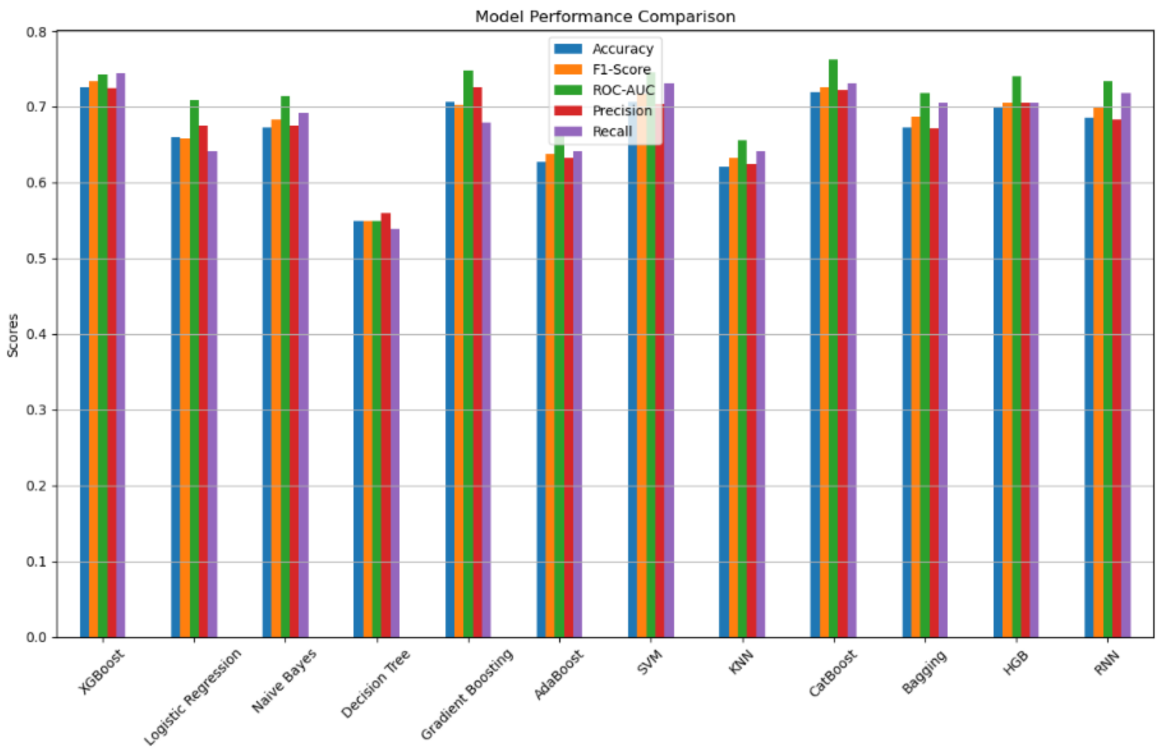


Figure 29 Model comparison histogram

XGBOOST performs very well out of all models also model is fast and can handle big data. The XGBOOST is the best model to deploy.

12 Save final model

This scripts connects to a MongoDB databases to fetch a dataset, preprocess the dataset by encoding categorical feature and scaling numerical values, trains an xgboost model, and save the trained model, and label encoder and scaler to MongoDB for later use. Refer image 29

DEPLOYMENT

In [26]:

```python
# Import necessary libraries
import pandas as pd
from pymongo import MongoClient
from xgboost import XGBClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
import joblib
import pickle
import io
import bson

# Connect to MongoDB
client = MongoClient("mongodb://localhost:27017/")
db = client["main_db"]

# Fetch the superdataset from MongoDB
superdataset = pd.DataFrame(list(db["superdataset1"].find()))

# Drop MongoDB's default `_id` column if present
if "_id" in superdataset.columns:
    superdataset = superdataset.drop(columns=["_id"])

# Relevant columns for features and target
features = [
    'Map',
    'Team_A_ID', 'Team_A_Kills', 'Team_A_Deaths', 'Team_A_ACS',
    'Team_A_Headshot_Percentage', 'Team_A_First_Bloods', 'Team_A_Assists',
    'Team_B_ID', 'Team_B_Kills', 'Team_B_Deaths', 'Team_B_ACS',
    'Team_B_Headshot_Percentage', 'Team_B_First_Bloods', 'Team_B_Assists'
]
target = 'Winner'

# Split data into features (X) and target (y)
X = superdataset[features]
y = superdataset[target]

# Encode categorical features
label_encoder = LabelEncoder()
X['Map'] = label_encoder.fit_transform(X['Map'])

# Scale features
scaler = StandardScaler()
X = scaler.fit_transform(X)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train the XGBoost model
xgb_model = XGBClassifier(n_estimators=300, max_depth=4, learning_rate=0.05, random_state=42)
xgb_model.fit(X_train, y_train)

# Save the XGBoost model to MongoDB
model_buffer = io.BytesIO()
joblib.dump(xgb_model, model_buffer)
model_bson = bson.Binary(model_buffer.getvalue())

# Serialize the label encoder and scaler
label_encoder_bson = bson.Binary(pickle.dumps(label_encoder))
scaler_bson = bson.Binary(pickle.dumps(scaler))

# Insert the XGBoost model, label encoder, and scaler into MongoDB
db["models"].insert_one({
```

Figure 30: Train and store xgboost model in MongoDB

13 Deploy final model

The code then from the MongoDB loads the XGBoost model, the label encoder, and the scaler by deserializing them. That's why it defines the calculate_odds_and_earnings function that transforms probability into bookmaker odds, takes operating margin into consideration, and calculates potential earnings depending on the bet size. This makes a certain their betting calculations are precise. Refer to figure 30.

```
# Retrieve the XGBoost model from MongoDB
model_data = db["models"].find_one({"model_name": "xgboost"})

# Deserialize the XGBoost model
model_buffer = io.BytesIO(model_data["model_data"])
saved_xgb_model = joblib.load(model_buffer)

# Deserialize the label encoder and scaler
label_encoder = pickle.loads(model_data["label_encoder"])
scaler = pickle.loads(model_data["scaler"])

print("XGBoost model, label encoder, and scaler loaded from MongoDB.")
```

```
XGBoost model, label encoder, and scaler loaded from MongoDB.
```

[29]:

```
def calculate_odds_and_earnings(prob_a, prob_b, bet_amount, margin=0.05):
    """
    Convert probabilities into realistic bookmaker odds and then calculate potential earnings.

    Parameters:
    - prob_a (float): Probability of Team A winning.
    - prob_b (float): Probability of Team B winning.
    - bet_amount (float): Amount of money bet.
    - margin (float): Overround margin the bookmaker adds.

    Returns:
    - odds_a (float): Bookmaker odds for Team A.
    - odds_b (float): Bookmaker odds for Team B.
    - earnings_a (float): Potential earnings if you bet on Team A and they win.
    - earnings_b (float): Potential earnings if you bet on Team B and they win.
    """
    # Compute fair odds
    if prob_a <= 0 or prob_b <= 0:
        raise ValueError("Probabilities must be greater than 0.")

    fair_odds_a = 1.0 / prob_a
    fair_odds_b = 1.0 / prob_b

    # To apply a margin, we distribute it proportionally.
    # One approach is to adjust each fair odds downward so that
    # the implied probabilities sum to more than 1.
    # Implied probabilities from fair odds:
    implied_a = 1.0 / fair_odds_a
    implied_b = 1.0 / fair_odds_b
    sum_implied = implied_a + implied_b

    # With a margin, we want sum_implied > 1. For example,
    # if we want a 5% margin, sum_implied should be 1.05.
    # We'll scale probabilities so their sum is equal to 1 + margin.
    desired_sum = 1 + margin
    scale_factor = desired_sum / sum_implied

    # Adjusted probabilities with margin
    adj_prob_a = implied_a * scale_factor
    adj_prob_b = implied_b * scale_factor

    # Convert back to odds
    odds_a = 1.0 / adj_prob_a
    odds_b = 1.0 / adj_prob_b
```

Figure 31: Fetching model from MongoDB and defines calculate_odds_and_earnings

The predict_match_outcome function calculates match probability using the XGBoost model, checks validation and fetches team stats. data gets processed and predicts the probability of winning, odds, and earnings. Refer to figure 32 and 33.

```python
def predict_match_outcome(team_a_id, team_b_id, map_name, bet_amount=30):
    """
    Predict the probabilities for a match using the saved XGBoost model and
    calculate realistic betting odds and potential earnings.

    Parameters:
    - team_a_id (int): ID of Team A.
    - team_b_id (int): ID of Team B.
    - map_name (str): Map name.
    - bet_amount (float): Amount of money to place as a bet.

    Returns:
    None. Prints out probabilities, odds, and potential earnings.
    """
    # Validate teams
    if team_a_id not in superdataset['Team_A_ID'].values:
        raise ValueError(f"Team A ID {team_a_id} is not valid.")
    if team_b_id not in superdataset['Team_B_ID'].values:
        raise ValueError(f"Team B ID {team_b_id} is not valid.")
    if team_a_id == team_b_id:
        raise ValueError("Team A ID and Team B ID cannot be the same.")

    # Validate map
    if map_name not in label_encoder.classes_:
        raise ValueError(f"Map name '{map_name}' not recognized by the label encoder.")

    # Compute team-based medians
    team_a_kills = superdataset[superdataset['Team_A_ID'] == team_a_id]["Team_A_Kills"].median()
    if pd.isna(team_a_kills):
        print(f"Warning: No historical data for Team A (ID {team_a_id}), defaulting stats to 0.")
        team_a_kills = 0
    team_a_deaths = superdataset[superdataset['Team_A_ID'] == team_a_id]["Team_A_Deaths"].median() or 0
    team_a_acs = superdataset[superdataset['Team_A_ID'] == team_a_id]["Team_A_ACS"].median() or 0
    team_a_hs = superdataset[superdataset['Team_A_ID'] == team_a_id]["Team_A_Headshot_Percentage"].median() or 0
    team_a_fb = superdataset[superdataset['Team_A_ID'] == team_a_id]["Team_A_First_Bloods"].median() or 0
    team_a_asst = superdataset[superdataset['Team_A_ID'] == team_a_id]["Team_A_Assists"].median() or 0

    team_b_kills = superdataset[superdataset['Team_B_ID'] == team_b_id]["Team_B_Kills"].median()
    if pd.isna(team_b_kills):
        print(f"Warning: No historical data for Team B (ID {team_b_id}), defaulting stats to 0.")
        team_b_kills = 0
    team_b_deaths = superdataset[superdataset['Team_B_ID'] == team_b_id]["Team_B_Deaths"].median() or 0
    team_b_acs = superdataset[superdataset['Team_B_ID'] == team_b_id]["Team_B_ACS"].median() or 0
    team_b_hs = superdataset[superdataset['Team_B_ID'] == team_b_id]["Team_B_Headshot_Percentage"].median() or 0
    team_b_fb = superdataset[superdataset['Team_B_ID'] == team_b_id]["Team_B_First_Bloods"].median() or 0
    team_b_asst = superdataset[superdataset['Team_B_ID'] == team_b_id]["Team_B_Assists"].median() or 0

    # Prepare input data
    input_data = {
        "Map": label_encoder.transform([map_name])[0],
        "Team_A_ID": team_a_id,
        "Team_A_Kills": team_a_kills,
        "Team_A_Deaths": team_a_deaths,
        "Team_A_ACS": team_a_acs,
        "Team_A_Headshot_Percentage": team_a_hs,
        "Team_A_First_Bloods": team_a_fb,
        "Team_A_Assists": team_a_asst,
        "Team_B_ID": team_b_id,
        "Team_B_Kills": team_b_kills,
        "Team_B_Deaths": team_b_deaths,
        "Team_B_ACS": team_b_acs,
        "Team_B_Headshot_Percentage": team_b_hs,
        "Team_B_First_Bloods": team_b_fb,
        "Team_B_Assists": team_b_asst
    }

    input_df = pd.DataFrame([input_data])
    input_df = input_df.fillna(0)

    # Ensure feature order matches training set
    input_df = input_df[features]

    # Scale the input
    input_scaled = scaler.transform(input_df)

    # Predict probabilities
    probabilities = saved_xgb_model.predict_proba(input_scaled)
```

Figure 32 : Predict_match_outcome using XGB

```python
# Example usage:
team_a_id = 17
team_b_id = 1001
map_name = "Abyss"
bet_amount = 22

try:
    predict_match_outcome(team_a_id, team_b_id, map_name, bet_amount)
except ValueError as e:
    print("Error:", str(e))
except Exception as e:
    print("Unexpected error:", str(e))
```

```
Winning Probabilities - Team A: 0.59, Team B: 0.41
Realistic Odds (with margin) - Team A: 1.61, Team B: 2.34
Potential Earnings - Bet €22 on Team A: €35.32, on Team B: €51.50
```

Figure 33: deployment prediction

14 USER INTERFACE

- Firstly, install streamlit for that use the command prompt and enter ‘ pip install streamlit ‘

```
:\Users\ASUS>pip install streamlit
```

- After Streamlit is successfully installed check version

```
C:\Users\ASUS>streamlit --version
Streamlit, version 1.40.1
```

- If a check occurs reinstall

Figure 34 shows the script for the user interface created using Streamlit.

jupyter app.py 13 hours ago

File Edit View Language

```python
import streamlit as st
import pandas as pd
from pymongo import MongoClient
from sklearn.preprocessing import LabelEncoder, StandardScaler
import joblib
import pickle
import io
import matplotlib.pyplot as plt
import numpy as np

# MongoDB connection
client = MongoClient("mongodb://localhost:27017/")
db = client["main_db"]

# Load the superdataset
superdataset = pd.DataFrame(list(db["superdataset1"].find()))
if "_id" in superdataset.columns:
    superdataset = superdataset.drop(columns=["_id"])

# Extract unique values for dropdowns
team_ids = sorted(superdataset['Team_A_ID'].unique().tolist())
map_names = sorted(superdataset['Map'].unique().tolist())

# Features and target column
features = [
    'Map',
    'Team_A_ID', 'Team_A_Kills', 'Team_A_Deaths', 'Team_A_ACS',
    'Team_A_Headshot_Percentage', 'Team_A_First_Bloods', 'Team_A_Assists',
    'Team_B_ID', 'Team_B_Kills', 'Team_B_Deaths', 'Team_B_ACS',
    'Team_B_Headshot_Percentage', 'Team_B_First_Bloods', 'Team_B_Assists'
]
target = 'Winner'

# Load model, scaler, and label encoder from MongoDB
model_data = db["models"].find_one({"model_name": "xgboost"})
model_buffer = io.BytesIO(model_data["model_data"])
saved_xgb_model = joblib.load(model_buffer)
label_encoder = pickle.loads(model_data["label_encoder"])
scaler = pickle.loads(model_data["scaler"])


def predict_match_outcome(team_a_id, team_b_id, map_name):
    """
    Predict the winning probabilities for a Valorant match using an XGBoost model.
    """
    if team_a_id not in superdataset['Team_A_ID'].values:
        raise ValueError(f"Team A ID {team_a_id} is not valid.")
    if team_b_id not in superdataset['Team_B_ID'].values:
```

Figure 34: Streamlit script for model deployment

To run the project deployment using Streamlit enter the command in the command prompt firstly open the project directory using the "cd" command and then type the command "streamlit run app.py" to run the application. Refer to figure 35.

```
C:\Users\ASUS>cd PROJECT NEW

C:\Users\ASUS\PROJECT NEW>streamlit run app.py
```

Figure 35: Command to run application using streamlit

Then select team A and team B, select map and enter the amount you want to bet and press predict. Refer to figure 36.

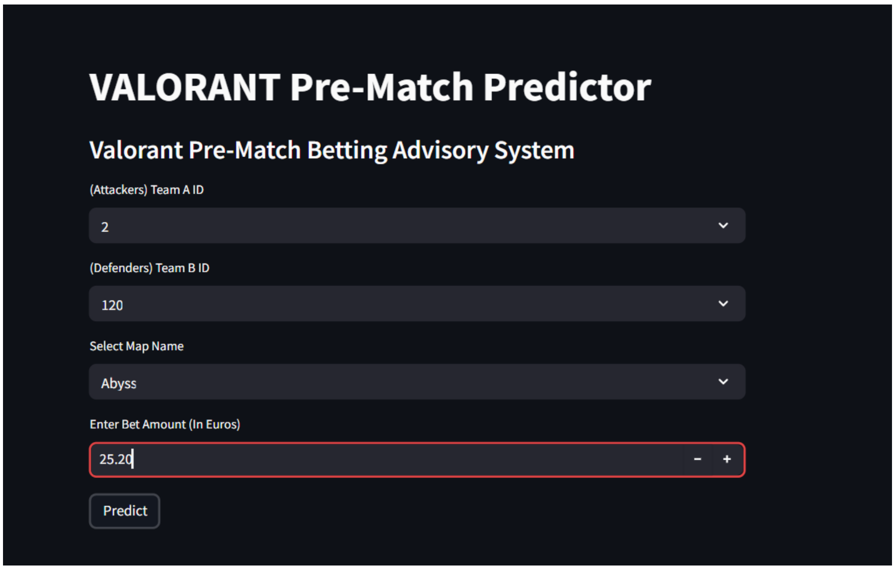


Figure 36: Valorant pre match betting advisory system UI

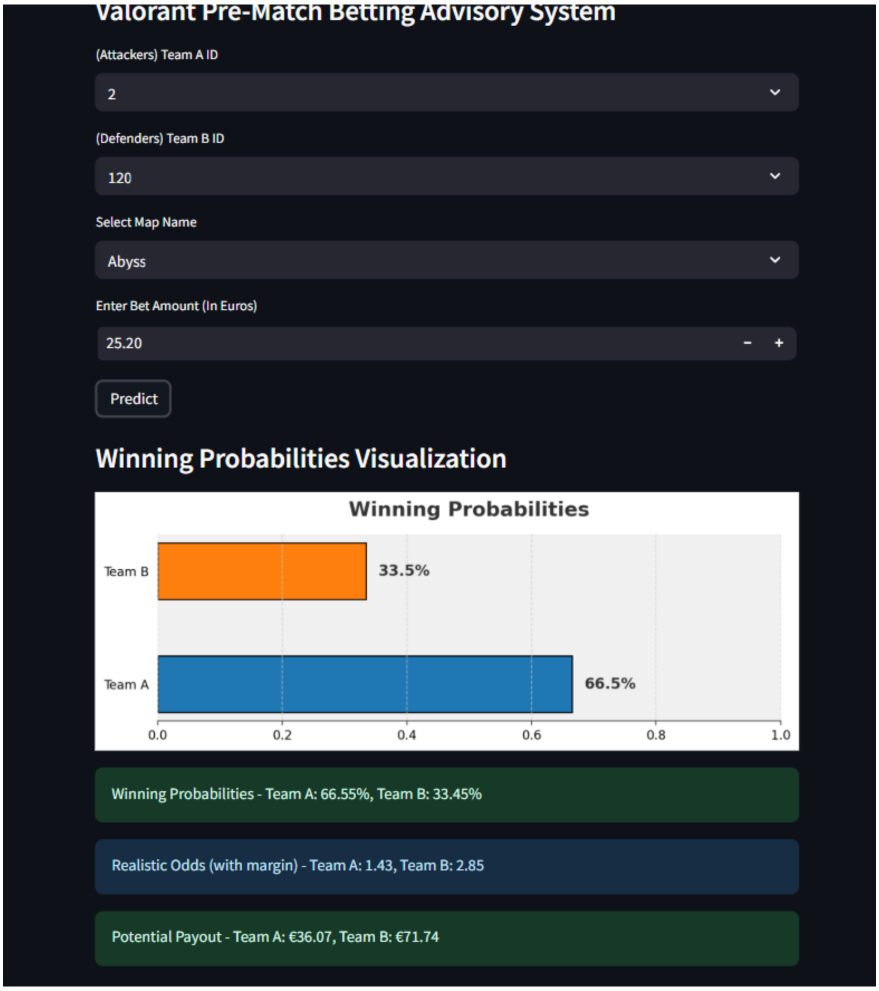


Figure 37: Final prediction

After pressing "Predict" the model predicts and calculates the winning probability for each team, simulates odd with margin and shows the earnings possibilities. it also displays graphics of winning probability. Refer to figure 37.

Provides a framework and the techniques can be applied to other popular esports games like CSGO and League of Legends to make the system more relevant.

15 References

Bahrololloomi, F., et al. (2022). A Machine Learning Based Predictive Analysis Use Case for eSports Games. Dergipark. https://dergipark.org.tr/en/download/article-file/2990904.

Coretto, P., 2021. Estimation and computations for Gaussian mixtures with uniform noise under separation constraints. *Journal of the Italian Statistical Society*, 31(3). Available at: https://doi.org/10.1007/s10260-021-00578-2

Kaggle.com, Valorant Champion Tour 2021-2024 Data, Ryan Luong. https://www.kaggle.com/datasets/ryanluong1/valorant-champion-tour-2021-2023-data

Xu, L., Skoularidou, M., Cuesta-Infante, A., & Veeramachaneni, K. (2019). Modeling tabular data using conditional GAN. *Advances in Neural Information Processing Systems*, *32*.