Final_Submission_mdj120

April 27, 2025

1 Contributions

Mitch Johnson: I contributed to the following: EDA(Graphs to find the player/game identifiers), Data Pre-processing(Some of the dropping of features, the dummies variable creation, train/test splitting, and the scaling),Model Creation (Logistic Regression Model and Tuning), Verification of the Logistic Regression Model(through Known Dataset), Most of the markdown cell explinations, Analysis, Conclusion, and Recommendations/Next-Steps.

Kevin Morales Rosales: I contributed EDA scatterplots that demonstrate how post-game stats essentially give the models the answer and prevents them from learning. Feature engineering additional features for team leve statistics and normalizing team stats over a period of 5 minuntes from the 10 minute mark and 15 minute mark of the match. Model of choice was Random Forest Classifier witch GridSearchCV to find ideal set up for the models and the dataset.

2 Problem Definition

Prepared by: Mitch Johnson, Kevin Morales Rosales

PROJECT SUMMARY

We will attempt to predict the winner or loser of a "League of Legends" game based on the dataset. We will start by using Logistic Regression as our base model as we are trying to predict a binary label (win/lose). "League of Legends" is a computer game by "Riot Games".

PROBLEM STATEMENT

We are trying to predict if Team 1 will win or lose based on statistics from the game data.

DATASET

Dataset found here: https://www.kaggle.com/datasets/fernandorubiogarcia/league-of-legends-high-elo-patch-1016

Highlights for the dataset are as follows:

Instances: 60,156

Attributes: 609

Total number of data points: 36,635,004

3 Data Collection

We start by reading the data into a DataFrame.

[67]:
```python
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from matplotlib import pyplot as plt
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
↪f1_score, classification_report
from sklearn.metrics import confusion_matrix

# Hyperparameter Tuning
from sklearn.model_selection import GridSearchCV



df = pd.read_csv("../project/10.16_LeagueOfLegends_Games.csv", delimiter = ";")

df.head()
```

[67]:
```
       gameId                                  t1p1_accountId  t1p1_assists  \
0  4747415735  j7A9NCAnShAbuQuEtlPMwVGqUiHAyEUdI_OEe79ToX-gkeI             7
1  4747759040   LkRqeoIcbTZeArlKA80SnOuMcUhI4tPl_0rkVCKpyepB8g            21
2  4746336268   IhcGDHrSIFZc3c5g4r_ljnBNfe1bgcBqwg0KwkZdj_q5BA            11
3  4756259885  TC1II2bhaFaQ4q-jQi_GUxV7rgrRToiBC-3qAy18Melt25A            11
4  4756423982   JFMI-NomU4kwSVK5T7JvmYlIiTWQFPo47BoKnMRtz_cKNA            11

   t1p1_ban_champId  t1p1_champId  t1p1_champLevel  \
0               121            25               11
1               111            37               13
2               122            89               11
3                81            89               11
4                91            25               11

   t1p1_damageDealtToObjectives  t1p1_damageDealtToTurrets  \
0                           166                          0
1                          5397                       2879
2                          3030                       1038
3                          1253                       1253
4                           260                        260
```

```
   t1p1_damageSelfMitigated  t1p1_deaths  …  t2p5_wardsKilled  \
0                     12783            5  …                 0
1                      8793            5  …                 2
2                     23864            7  …                 5
3                     14012            4  …                 1
4                      9350            6  …                 2

   t2p5_wardsPlaced  gameCreation  gameDuration     gameVersion  platformId  \
0                10  1.596750e+12          1738  10.16.330.9186        EUW1
1                 6  1.596760e+12          1374  10.16.330.9186        EUW1
2                13  1.596720e+12          1692  10.16.330.9186        EUW1
3                 5  1.597160e+12          1383  10.16.330.9186        EUW1
4                 9  1.597160e+12          1672  10.16.330.9186        EUW1

   queueId  average_lp  t1_teamId  t1_win
0      420      3380.4        100       0
1      420      3284.9        100       1
2      420      3333.2        100       0
3      420      3258.1        100       1
4      420      3116.0        100       0

[5 rows x 609 columns]
```

We check to make sure there are no missing values, if there are missing values, we drop them.

[68]:
```
df.isnull().sum()

df = df.dropna()
```

Check the size of the dataframe.

[69]:
```
row,columns = df.shape
print(df.shape)
```

```
(60107, 609)
```

We grab some easy general info about the data.

[70]:
```
df.info()
```

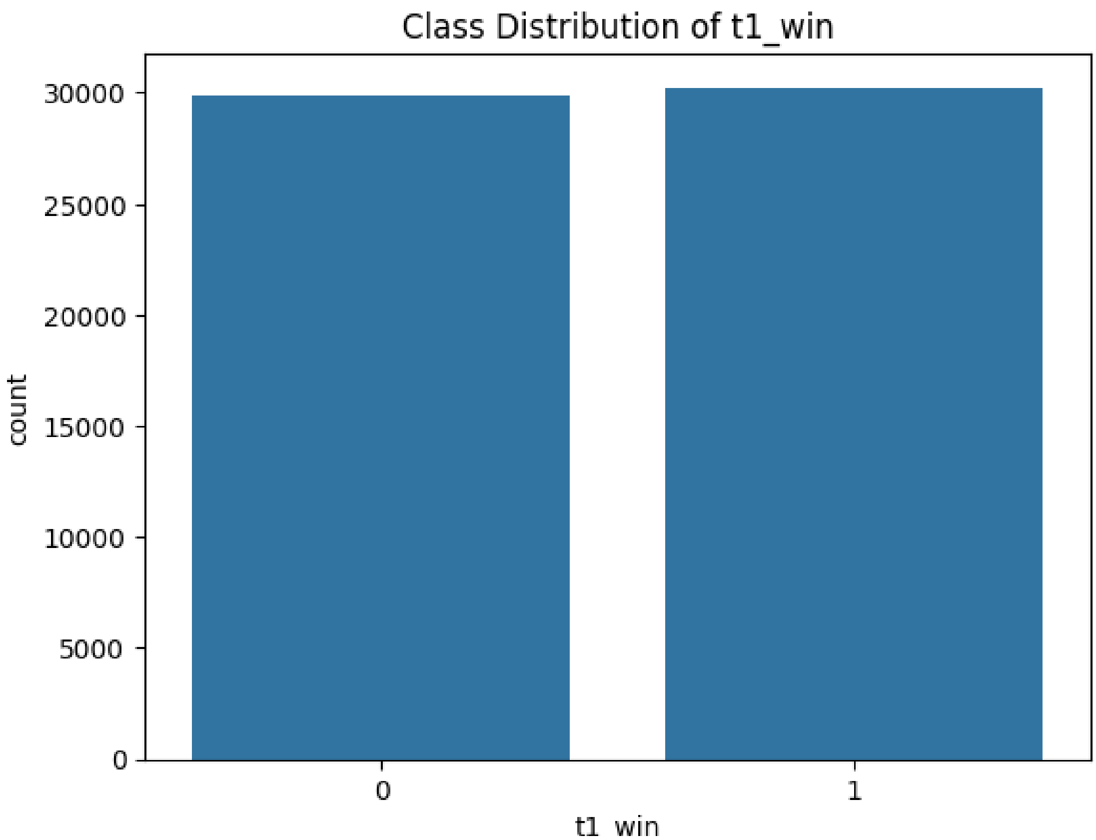
```
<class 'pandas.core.frame.DataFrame'>
Index: 60107 entries, 0 to 60155
Columns: 609 entries, gameId to t1_win
dtypes: float64(97), int64(470), object(42)
memory usage: 279.7+ MB
```

4 Exploratory Data Analysis

Check for good distribution of classes (Class Imbalance testing)

[71]:
```
target_dist = df['t1_win'].value_counts(normalize=True) * 100
print("Class Distribution in Target Variable (t1_win):")
print(target_dist)
sns.countplot(x='t1_win', data=df)
plt.title('Class Distribution of t1_win')
plt.show()
```

```
Class Distribution in Target Variable (t1_win):
t1_win
1    50.325253
0    49.674747
Name: proportion, dtype: float64
```

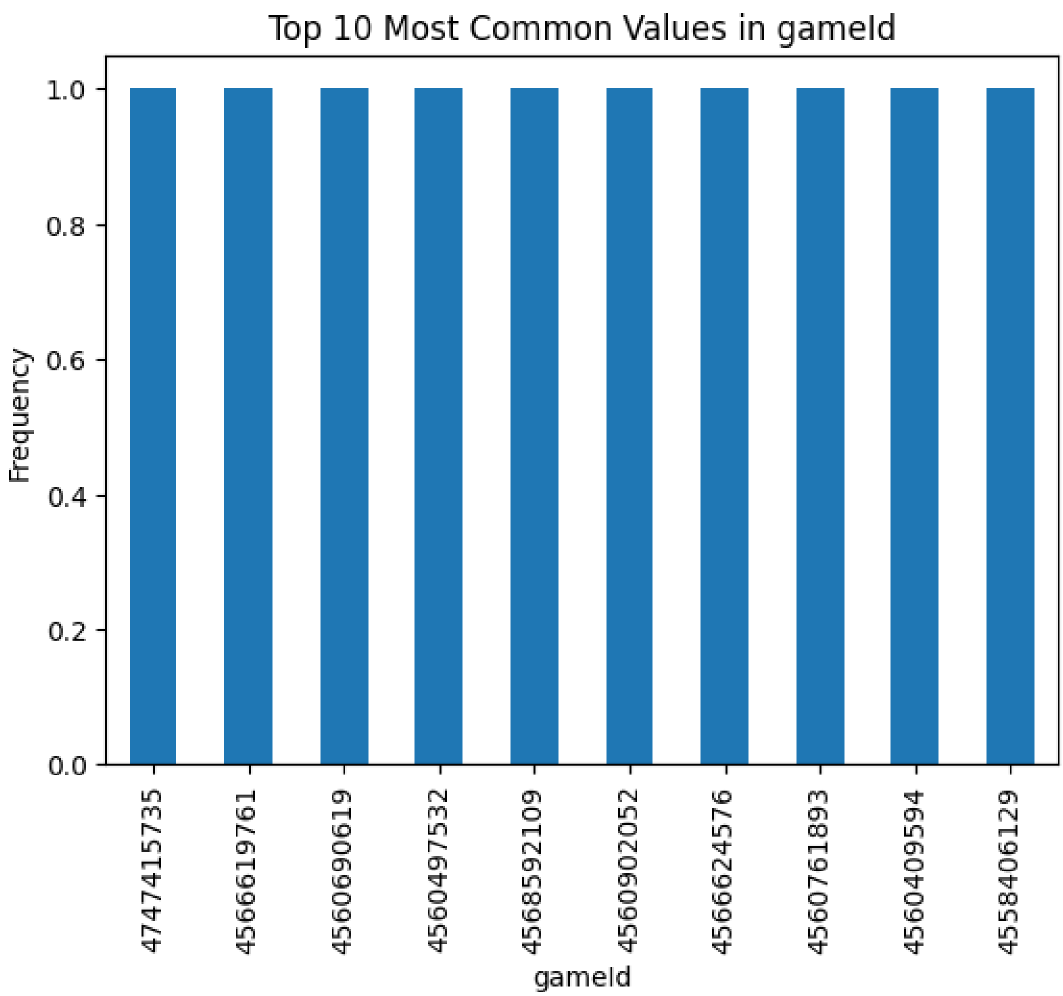


First we check gameId to see its uniqueness.

[72]:
```
gameId_column = [
    "gameId"
]

for col in gameId_column:
    unique_vals = df[col].nunique()
```

```
    total_vals = df[col].shape[0]
    print(f"{col}: {unique_vals:,} unique / {total_vals:,} total ({(unique_vals/
↪total_vals)*100:.2f}% unique)")

    df[col].value_counts().head(10).plot(kind='bar')
    plt.title(f"Top 10 Most Common Values in {col}")
    plt.ylabel("Frequency")
    plt.show()
```

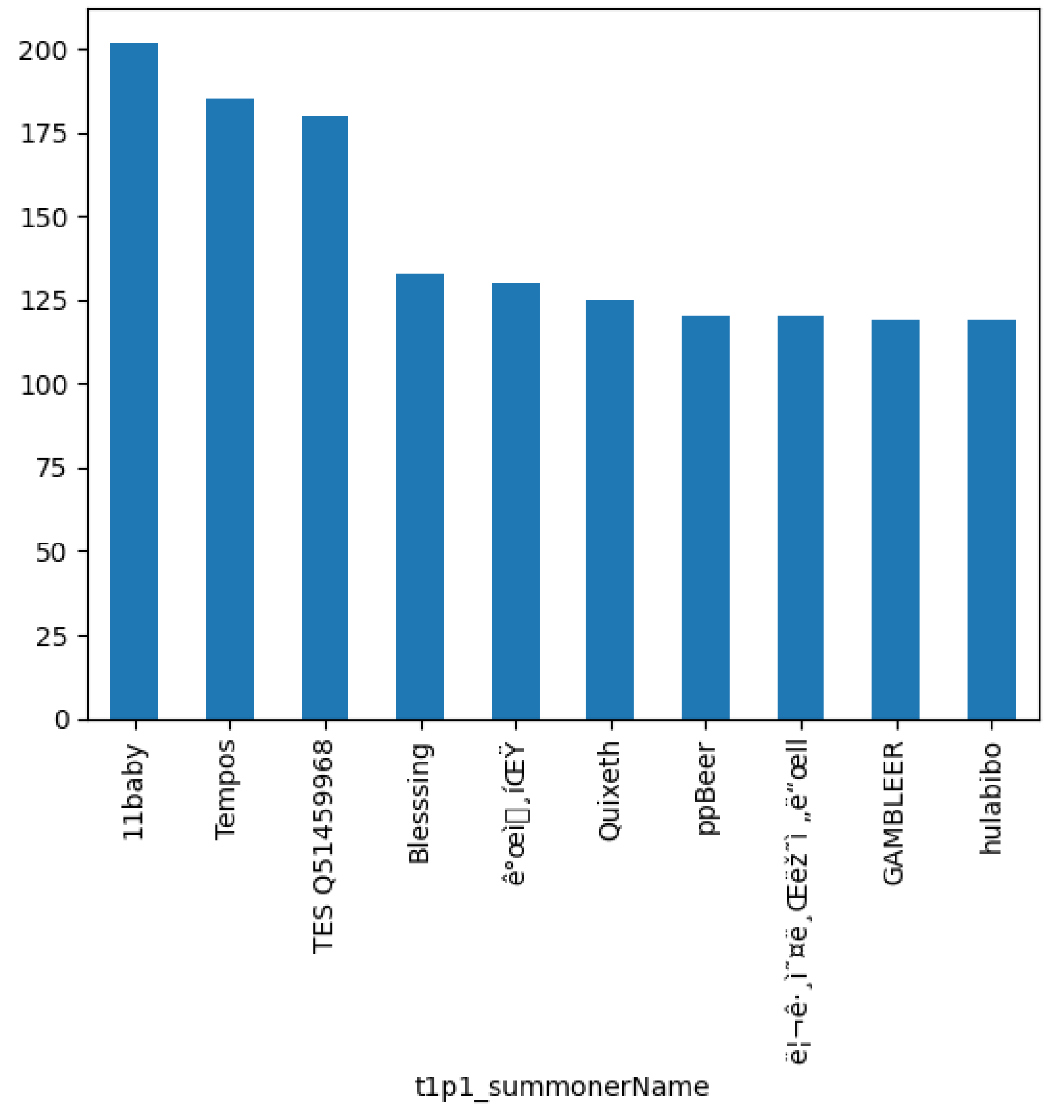
```
gameId: 60,107 unique / 60,107 total (100.00% unique)
```



Because gameID is completely unique to each row(there is no pattern for the model to look for), we will remove this column.

[73]:
```
df['t1p1_summonerName'].value_counts().head(10).plot(kind='bar')
```

```
[73]: <Axes: xlabel='t1p1_summonerName'>

C:\Users\maype\AppData\Roaming\Python\Python312\site-
packages\IPython\core\events.py:82: UserWarning: Glyph 157 (\x9d) missing from
font(s) DejaVu Sans.
  func(*args, **kwargs)
C:\Users\maype\AppData\Roaming\Python\Python312\site-
packages\IPython\core\pylabtools.py:170: UserWarning: Glyph 157 (\x9d) missing
from font(s) DejaVu Sans.
  fig.canvas.print_figure(bytes_io, **kw)
```



This graph shows that summonerName for all players will be repeated in multiple rows but the names of the players will add complexity to the model and noise to our data. So, we will remove the column. This is also bad for generalization.

[74]:
```
df.groupby('t1p1_summonerId')['t1_win'].agg(['mean', 'count']).
    ↪sort_values(by='count', ascending=False).head(10)
```

[74]:
```
                                                       mean  count
t1p1_summonerId
#NAME?                                             0.496112    643
2-ERsTYol6BKg2S6YkSdAWymMxS8jbFjcs0NU8bO8DKlmtE    0.509901    202
UWZq-HNQjD3jvjDvWCcKpC-0AXG-hgckNPdvWSFpUgT7rlM    0.526596    188
GsKHOcDm5NAcg5rINx0VufVDBnOec7iRc1oF7EnNA3UNEUVg  0.510753    186
1ojVDatkP2wFpGuDF5cZcwCIc9WyVKExt7fiu2rkk6HRfkM    0.529730    185
dzuPzDya96UuXkmN0597JnXOFC3Fgu7Bc7SebqrFY_WE9ks    0.479452    146
bhTVo0QEKhbq4zt3kmIBwytjjuhQVvsudXpgkZq1Wzk2-l8    0.481752    137
j1-dJMTIiytQyNLAwdCEL7oXugD0xWX-7pIB7WEX7Fl4sTI    0.451128    133
8ZOPrv5q0mYd1QMnEIYj-j6TJaXv7d_xCV-UzNZo_HyYrA     0.546154    130
QHbkiMKLIDwW57vNp8gplYGcw7O3JM2_ck8tL0iR2uv2WtI    0.531746    126
```

The summonerId only has a mean win of about .5 linked to most of the Id's so this will also not add useful information to the model for predictions. This is also bad for generalization.

[75]:
```
df.groupby('t1p1_accountId')['t1_win'].agg(['mean', 'count']).
    ↪sort_values(by='count', ascending=False).head(10)
```

[75]:
```
                                                         mean  count
t1p1_accountId
#NAME?                                               0.476589    598
yjEKl049QQ-EffE0r-pI40GJcW9vZxqyM_fpL5r7IhRIvwn…  0.509901    202
4PnREd5vdC06sXhouLBuYRqyoFnXt1gQyGx0NkFSRjRHBMo…  0.526596    188
VvET7tZvEPg9vJfVASv3BehO0oYLeJdrdeU12iDQgFytiNr…  0.510753    186
nmj-LPUtevhI9e2HmhMf95wbXtz0scqWVUt3wNijlViX1g       0.529730    185
Cf1URv3eOmhriCsbncfjL2JstZn_tfAAt1eTGroOmttv6QX…  0.479452    146
lrrmJ-tvn94a0yHxAp0JPzVaZIT-4NN6_AMXnIQIJjdWK-7…  0.481752    137
6HHZXGr6FqMC8R798c9dHMHQAaqdjEMYVFyDXLzuIPS4T6H…  0.451128    133
lKJjgnSIB5j-XvOCHYJ7n7681smTiV2akOQKBOSwh66t         0.546154    130
sx4QIG7Q9zOIwbSDc4P5DnnN0QeCDK_lPQAY2gL-Eje1hR7…  0.531746    126
```

We see the same thing as with SummonerId, we will remove the accountID columns as well. This is also bad for the generalization.

[76]:
```
df.groupby('platformId')['t1_win'].agg(['mean', 'count']).
    ↪sort_values(by='count', ascending=False).head(10)
```

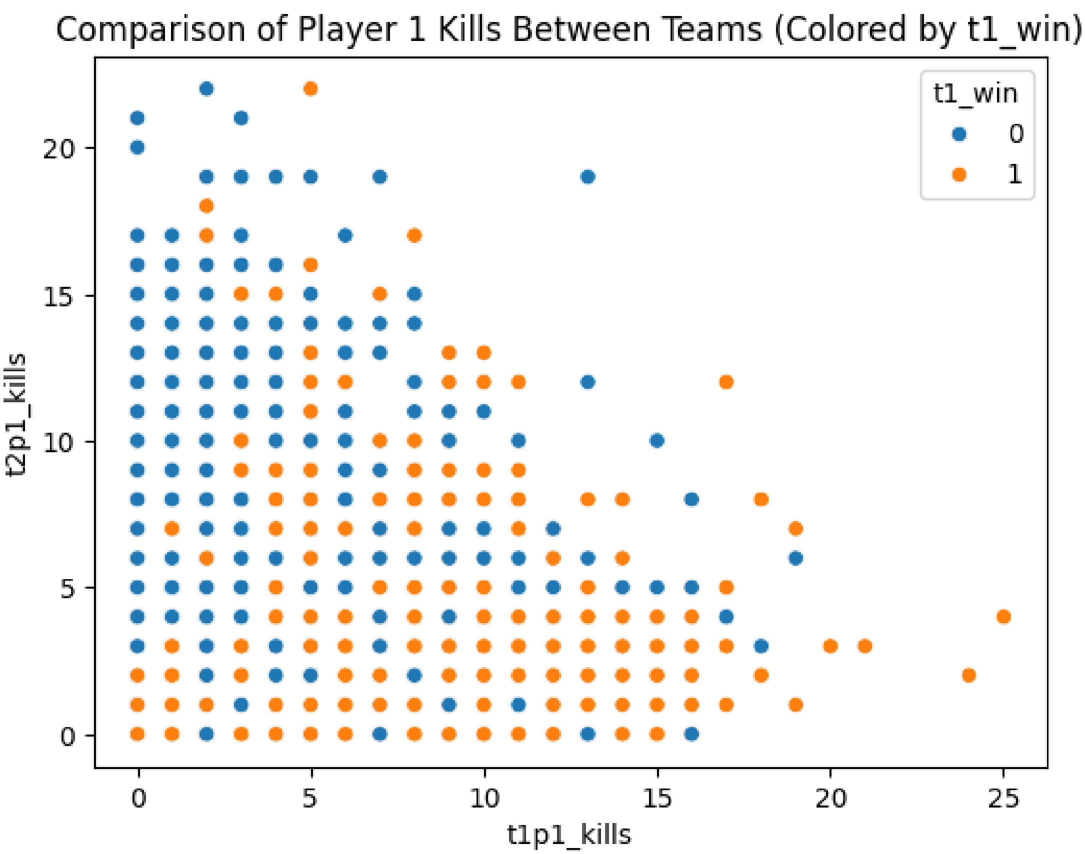
[76]:
```
                mean  count
platformId
KR          0.503208  28056
EUW1        0.507775  21864
NA1         0.493668  10187
```

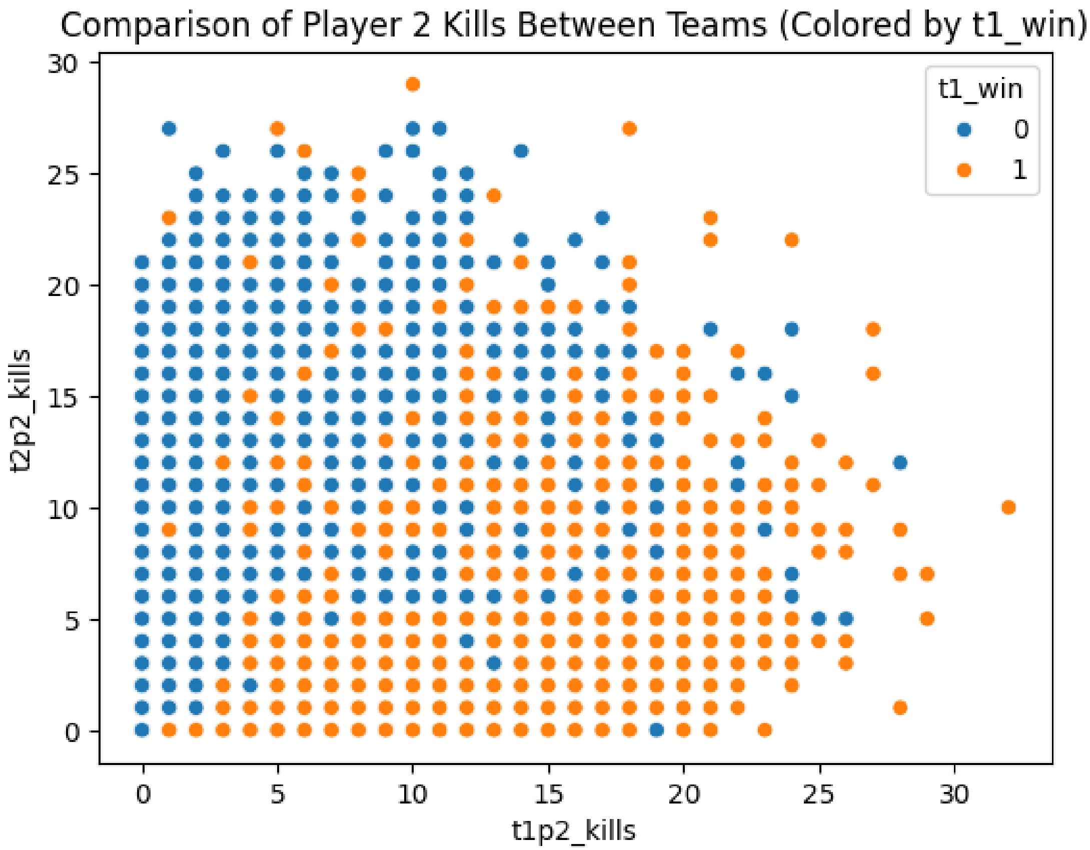
Again the mean is at .5 so the platformId is not likely to help the model.

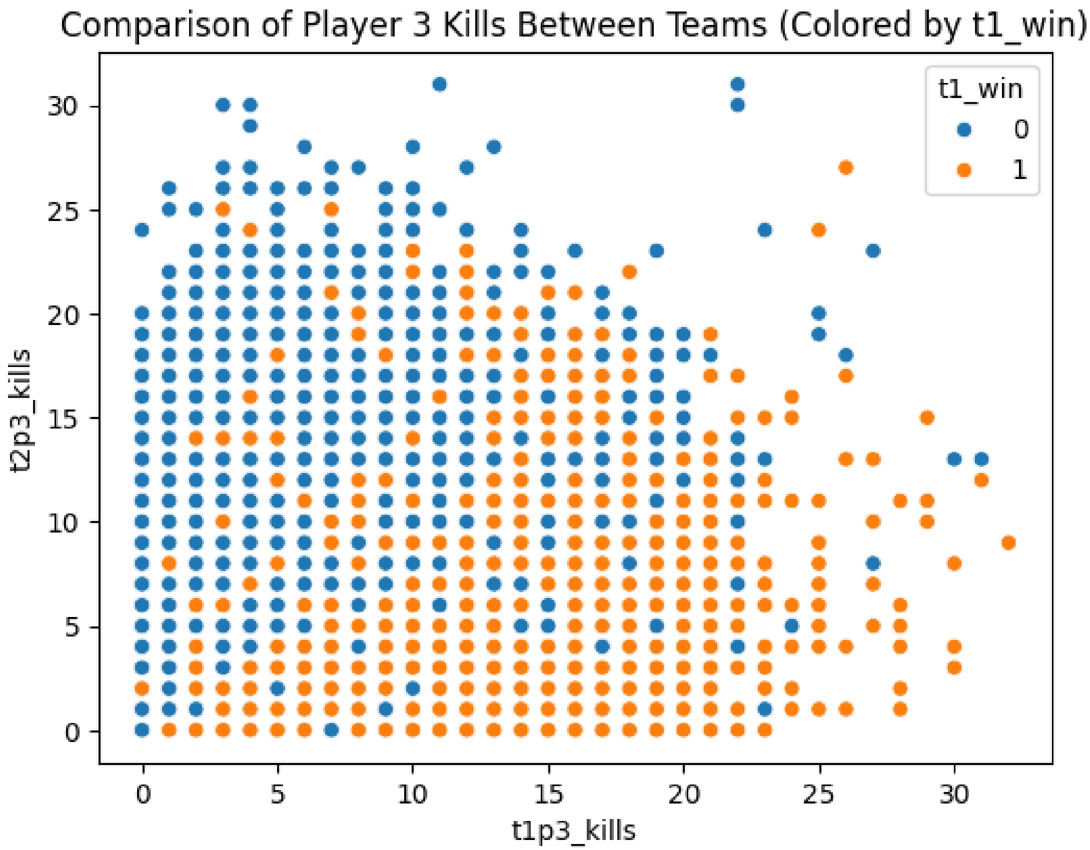
4.0.1 We now check some comparison statistics

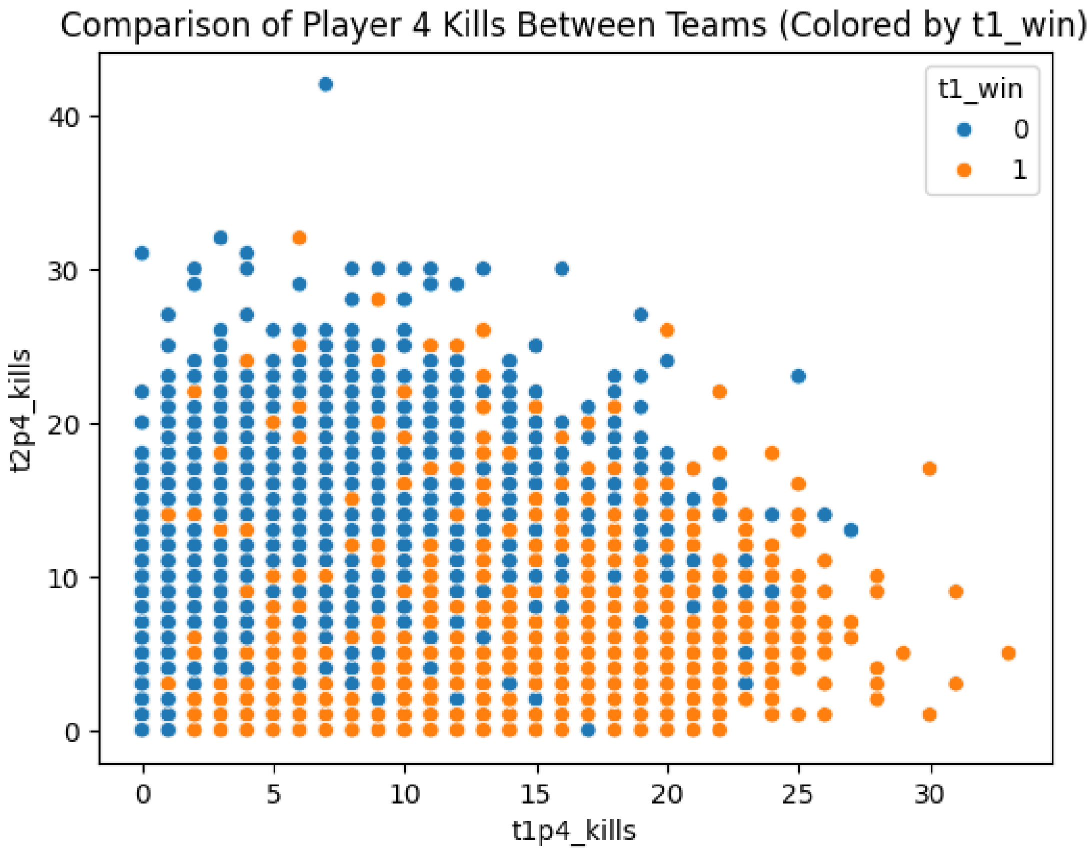
We check kill statistics. (comparision is from "player 1 on team 1" to "player 1 on team 2", this ensures the comparison is done based on the role each player is playing)

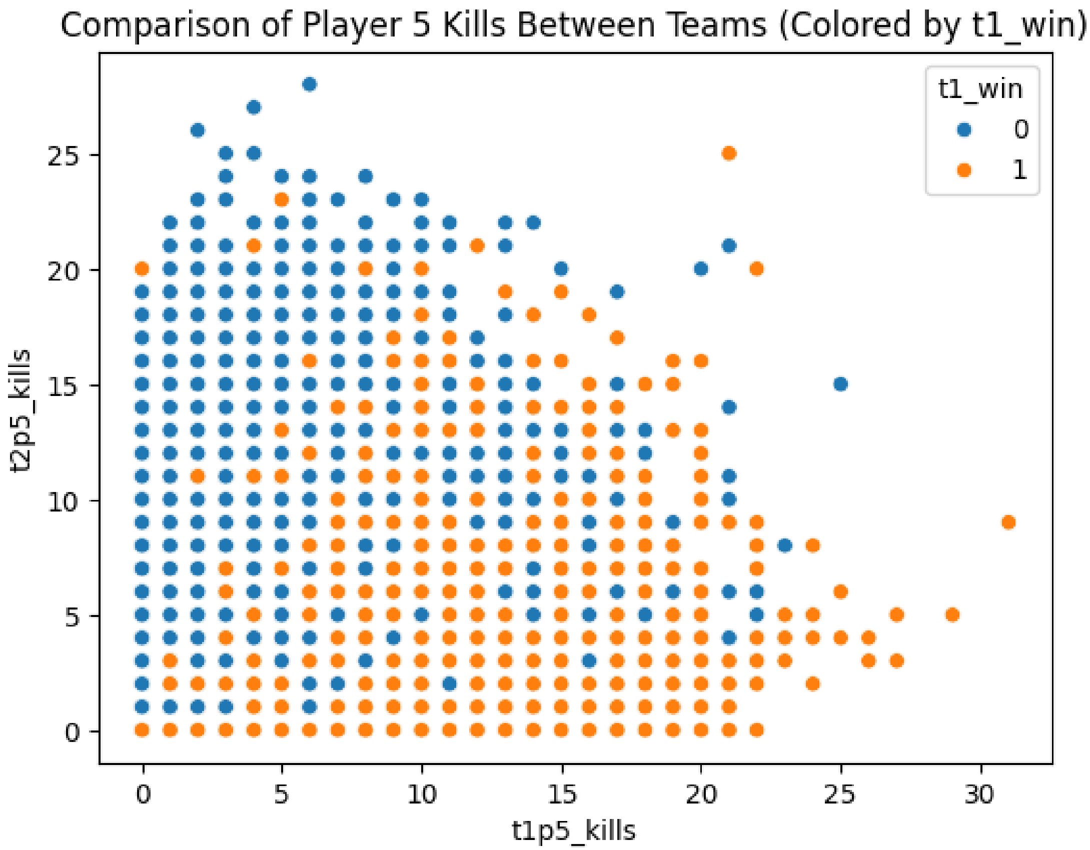
[77]:
```
sns.scatterplot(x='t1p1_kills', y='t2p1_kills', data=df, hue='t1_win')
plt.title('Comparison of Player 1 Kills Between Teams (Colored by t1_win)')
plt.show()


sns.scatterplot(x='t1p2_kills', y='t2p2_kills', data=df, hue='t1_win')
plt.title('Comparison of Player 2 Kills Between Teams (Colored by t1_win)')
plt.show()

sns.scatterplot(x='t1p3_kills', y='t2p3_kills', data=df, hue='t1_win')
plt.title('Comparison of Player 3 Kills Between Teams (Colored by t1_win)')
plt.show()

sns.scatterplot(x='t1p4_kills', y='t2p4_kills', data=df, hue='t1_win')
plt.title('Comparison of Player 4 Kills Between Teams (Colored by t1_win)')
plt.show()

sns.scatterplot(x='t1p5_kills', y='t2p5_kills', data=df, hue='t1_win')
plt.title('Comparison of Player 5 Kills Between Teams (Colored by t1_win)')
plt.show()
```


Comparison of Player 1 Kills Between Teams (Colored by t1_win)
t1_win
0
1
t2p1_kills
t1p1_kills
0
5
10
15
20
25


Comparison of Player 2 Kills Between Teams (Colored by t1_win)
t1_win
0
1
t2p2_kills
t1p2_kills
0
5
10
15
20
25
30


Comparison of Player 3 Kills Between Teams (Colored by t1_win)
t1_win
0
1
t2p3_kills
t1p3_kills
0
5
10
15
20
25
30

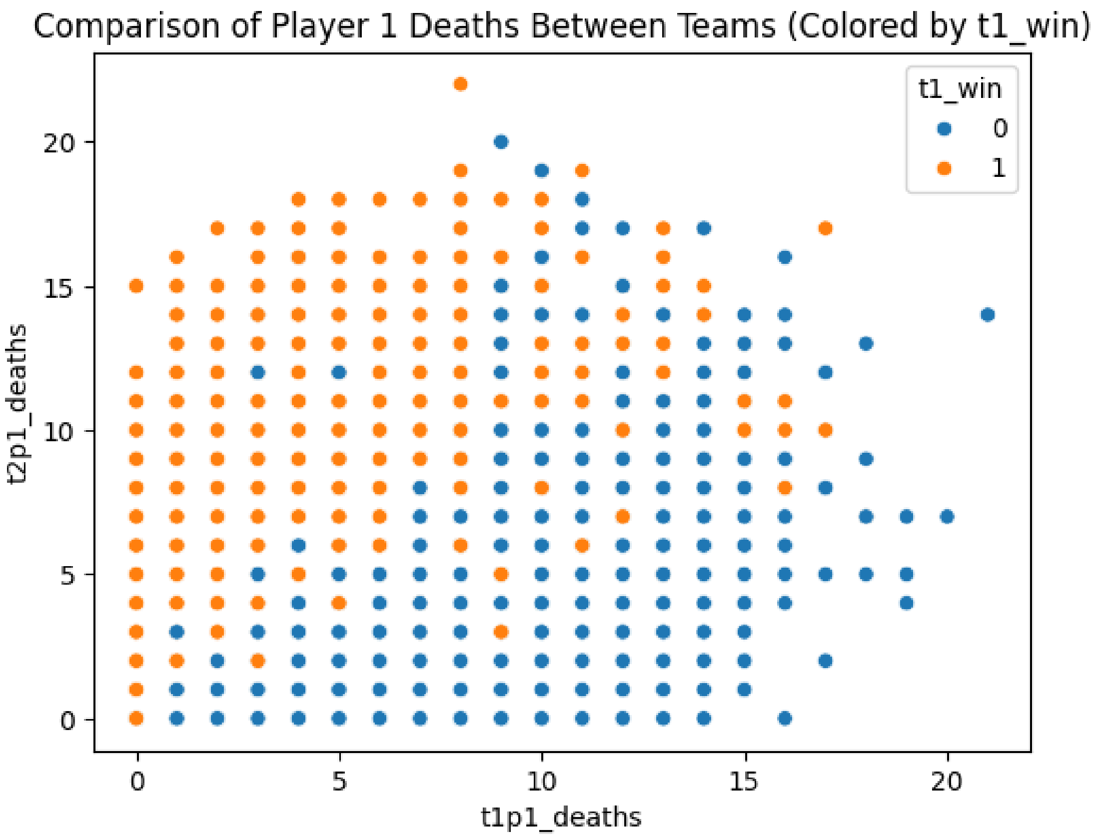

Comparison of Player 4 Kills Between Teams (Colored by t1_win)
t1_win
0
1
t2p4_kills
t1p4_kills
0
10
20
30
40
0
5
10
15
20
25
30

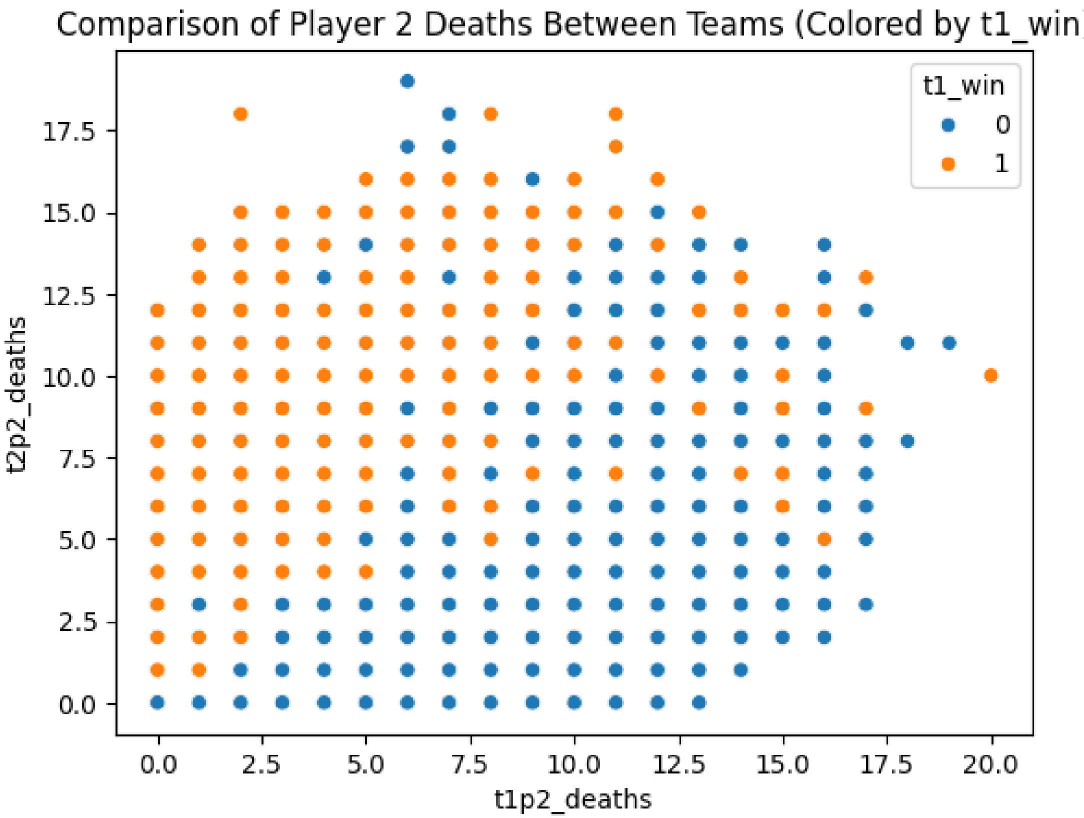


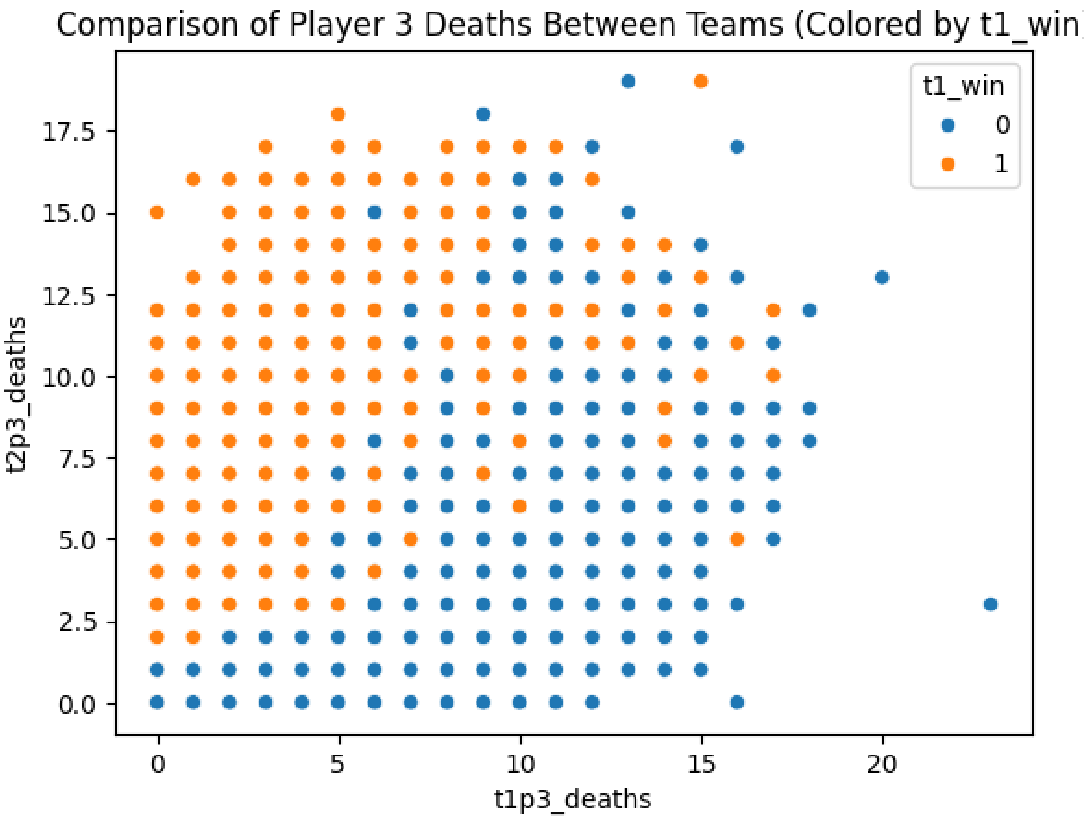
We check death statistics. (comparision is from “player 1 on team 1” to “player 1 on team 2”, this ensures the comparison is done based on the role each player is playing)

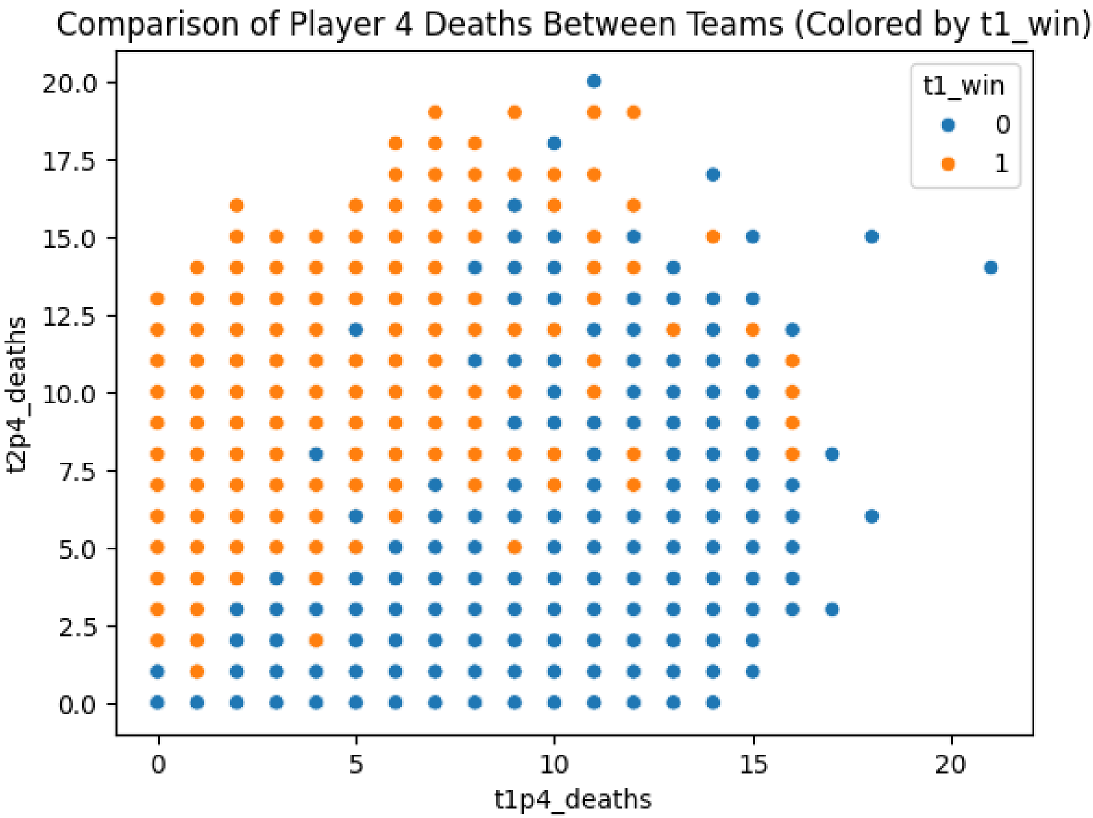
[78]:
```
sns.scatterplot(x='t1p1_deaths', y='t2p1_deaths', data=df, hue='t1_win')
plt.title('Comparison of Player 1 Deaths Between Teams (Colored by t1_win)')
plt.show()


sns.scatterplot(x='t1p2_deaths', y='t2p2_deaths', data=df, hue='t1_win')
plt.title('Comparison of Player 2 Deaths Between Teams (Colored by t1_win)')
plt.show()

sns.scatterplot(x='t1p3_deaths', y='t2p3_deaths', data=df, hue='t1_win')
plt.title('Comparison of Player 3 Deaths Between Teams (Colored by t1_win)')
plt.show()

sns.scatterplot(x='t1p4_deaths', y='t2p4_deaths', data=df, hue='t1_win')
plt.title('Comparison of Player 4 Deaths Between Teams (Colored by t1_win)')
plt.show()
```

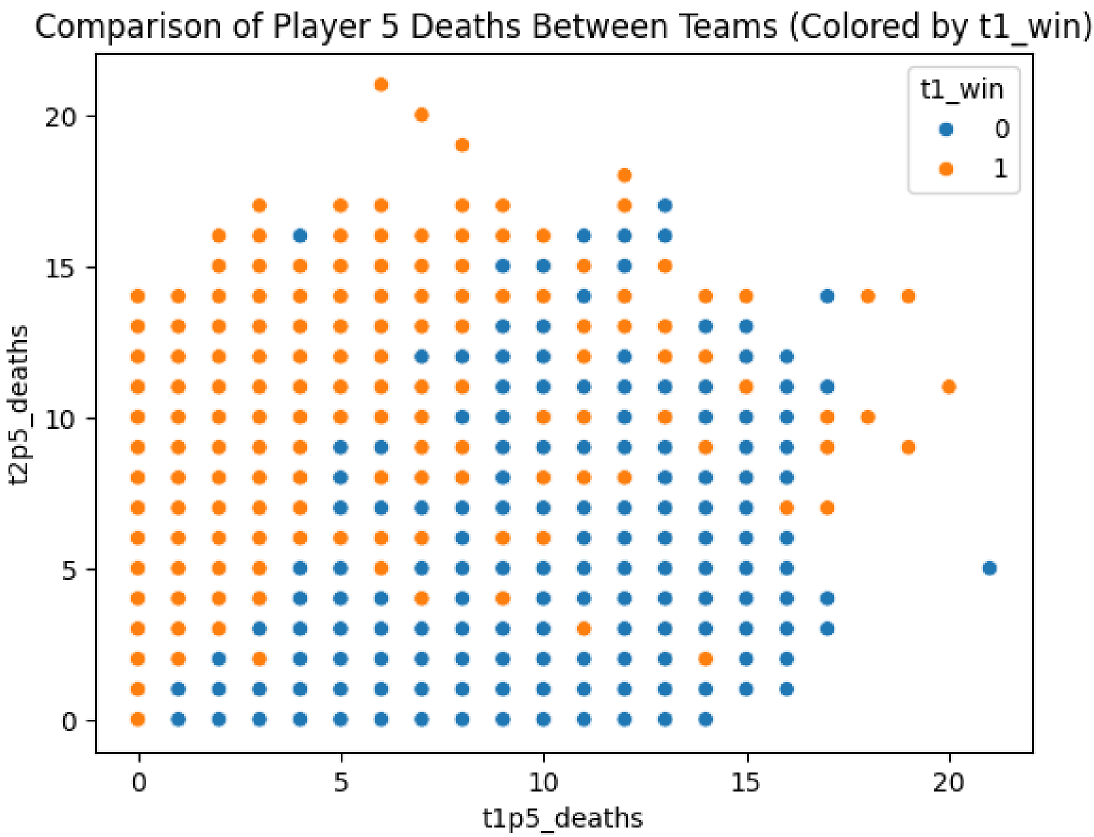
```python
sns.scatterplot(x='t1p5_deaths', y='t2p5_deaths', data=df, hue='t1_win')
plt.title('Comparison of Player 5 Deaths Between Teams (Colored by t1_win)')
plt.show()
```




Comparison of Player 2 Deaths Between Teams (Colored by t1_win)
t1_win
0
1
t2p2_deaths
t1p2_deaths
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0


Comparison of Player 3 Deaths Between Teams (Colored by t1_win)
t1_win
0
1
t2p3_deaths
t1p3_deaths
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
0
5
10
15
20

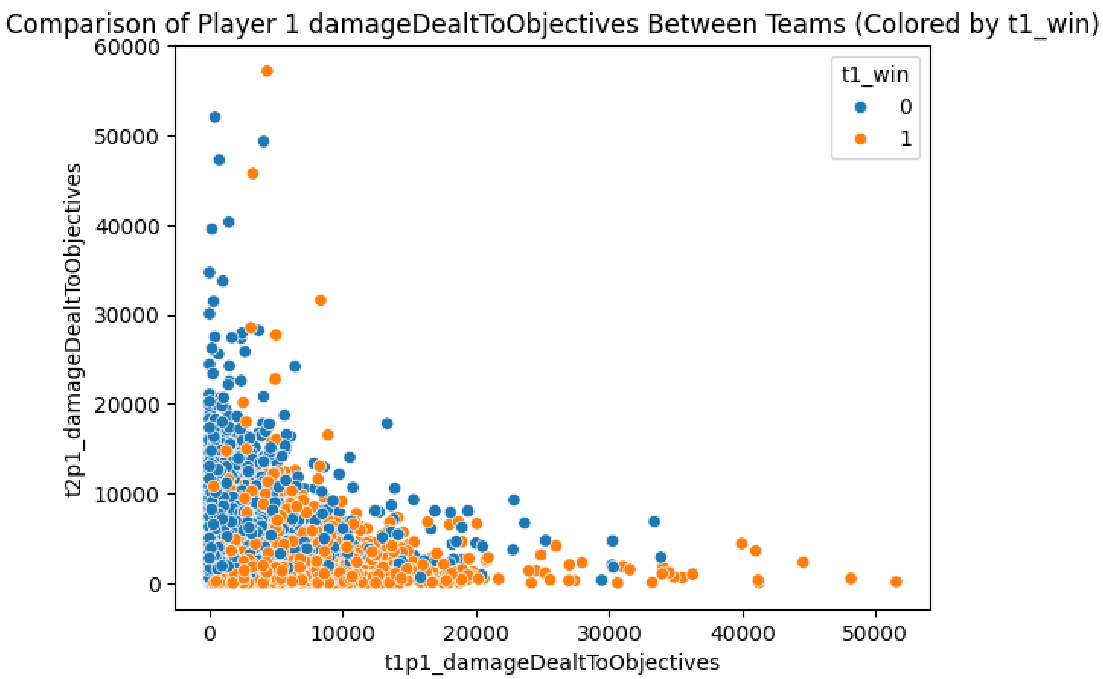

Comparison of Player 4 Deaths Between Teams (Colored by t1_win)
t1_win
0
1
t2p4_deaths
t1p4_deaths
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
0
5
10
15
20



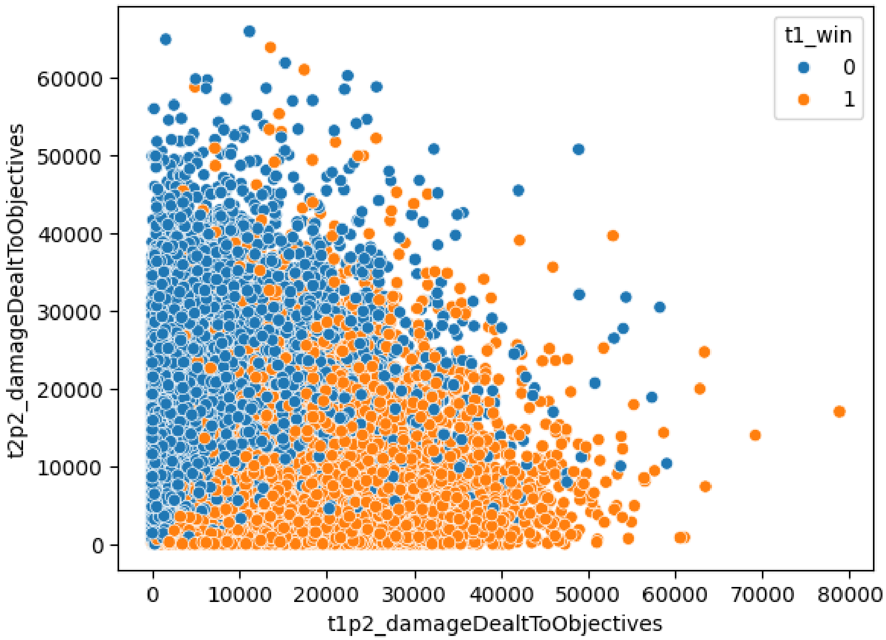
We check damage dealt to objectives. (comparision is from "player 1 on team 1" to "player 1 on team 2", this ensures the comparison is done based on the role each player is playing)

[79]:
```
sns.scatterplot(x='t1p1_damageDealtToObjectives',␣
  ↪y='t2p1_damageDealtToObjectives', data=df, hue='t1_win')
plt.title('Comparison of Player 1 damageDealtToObjectives Between Teams␣
  ↪(Colored by t1_win)')
plt.show()


sns.scatterplot(x='t1p2_damageDealtToObjectives',␣
  ↪y='t2p2_damageDealtToObjectives', data=df, hue='t1_win')
plt.title('Comparison of Player 2 damageDealtToObjectives Between Teams␣
  ↪(Colored by t1_win)')
plt.show()

sns.scatterplot(x='t1p3_damageDealtToObjectives',␣
  ↪y='t2p3_damageDealtToObjectives', data=df, hue='t1_win')
plt.title('Comparison of Player 3 damageDealtToObjectives Between Teams␣
  ↪(Colored by t1_win)')
```

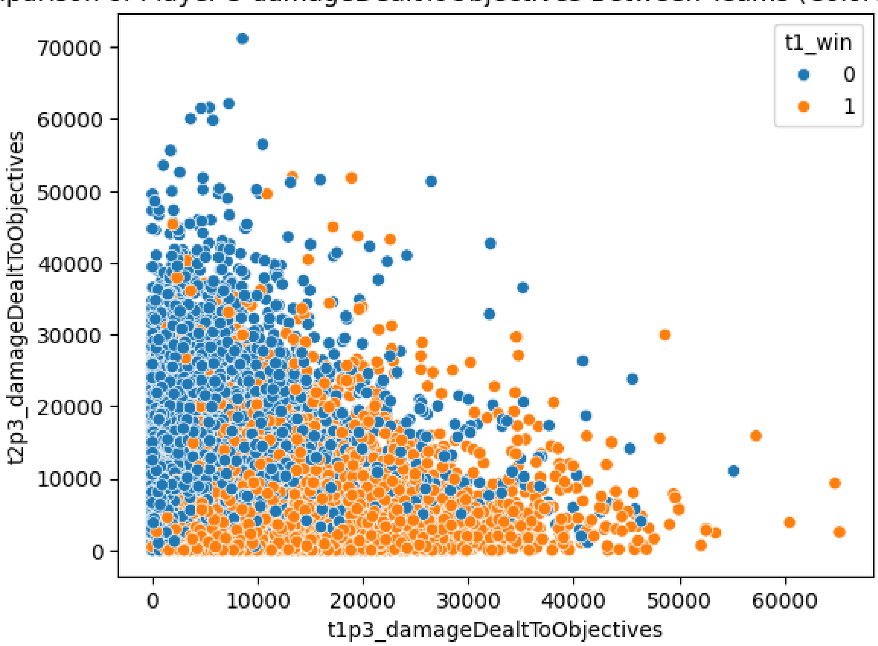
```
plt.show()

sns.scatterplot(x='t1p4_damageDealtToObjectives',␣
↪y='t2p4_damageDealtToObjectives', data=df, hue='t1_win')
plt.title('Comparison of Player 4 damageDealtToObjectives Between Teams␣
↪(Colored by t1_win)')
plt.show()

sns.scatterplot(x='t1p5_damageDealtToObjectives',␣
↪y='t2p5_damageDealtToObjectives', data=df, hue='t1_win')
plt.title('Comparison of Player 5 damageDealtToObjectives Between Teams␣
↪(Colored by t1_win)')
plt.show()
```

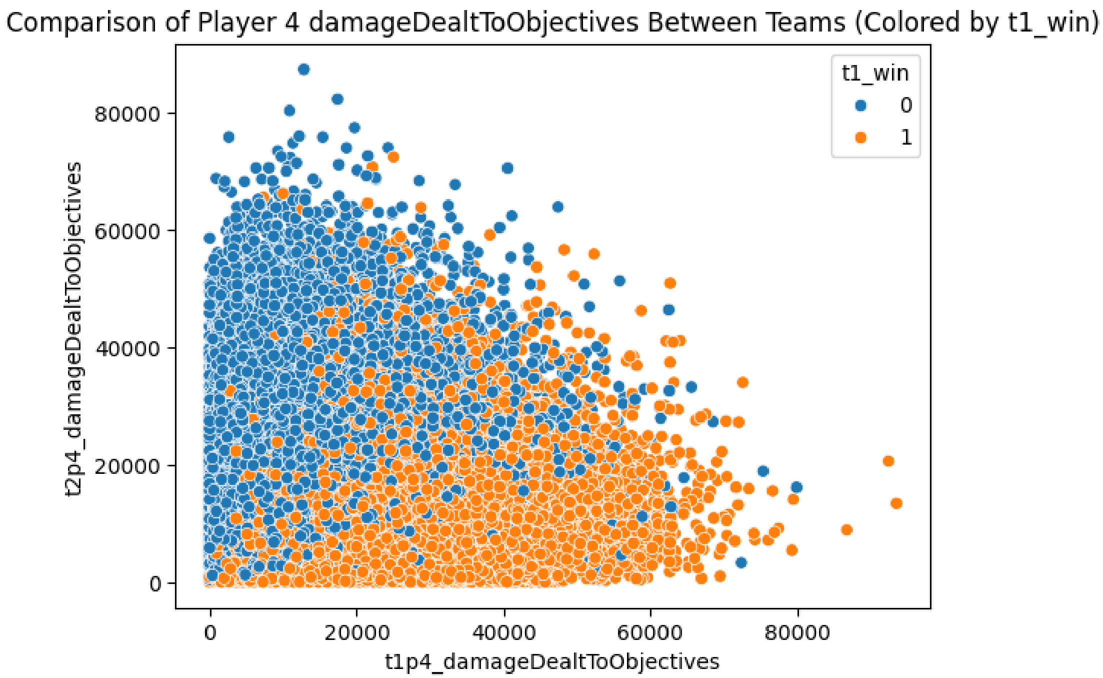


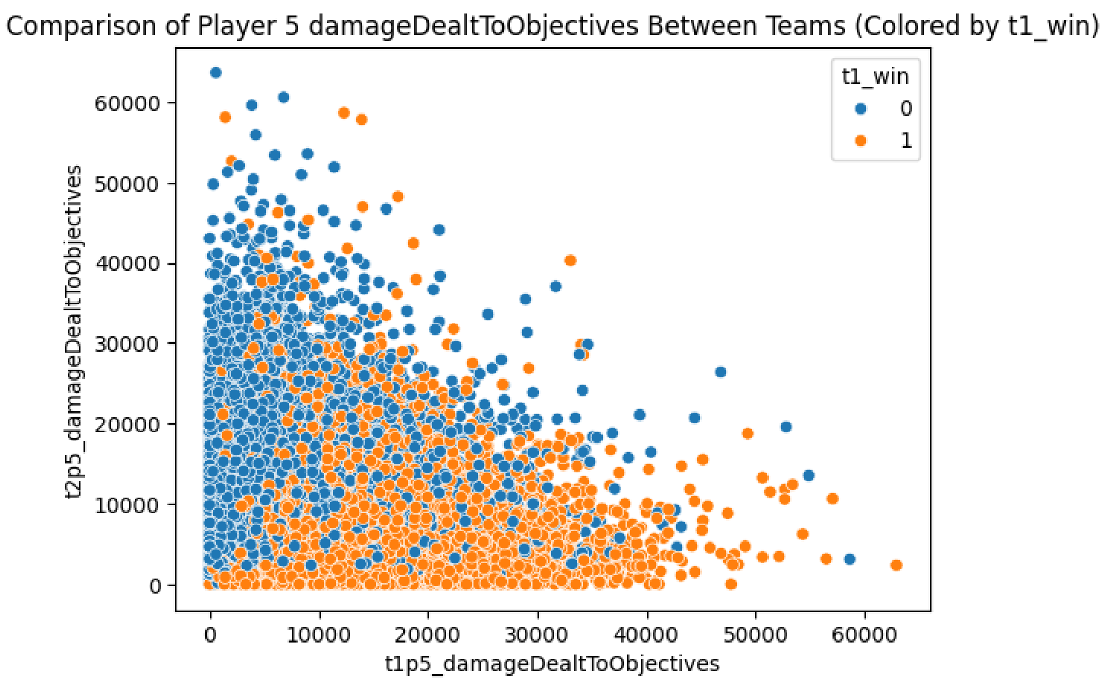
Comparison of Player 2 damageDealtToObjectives Between Teams (Colored by t1_win)



Comparison of Player 3 damageDealtToObjectives Between Teams (Colored by t1_win)

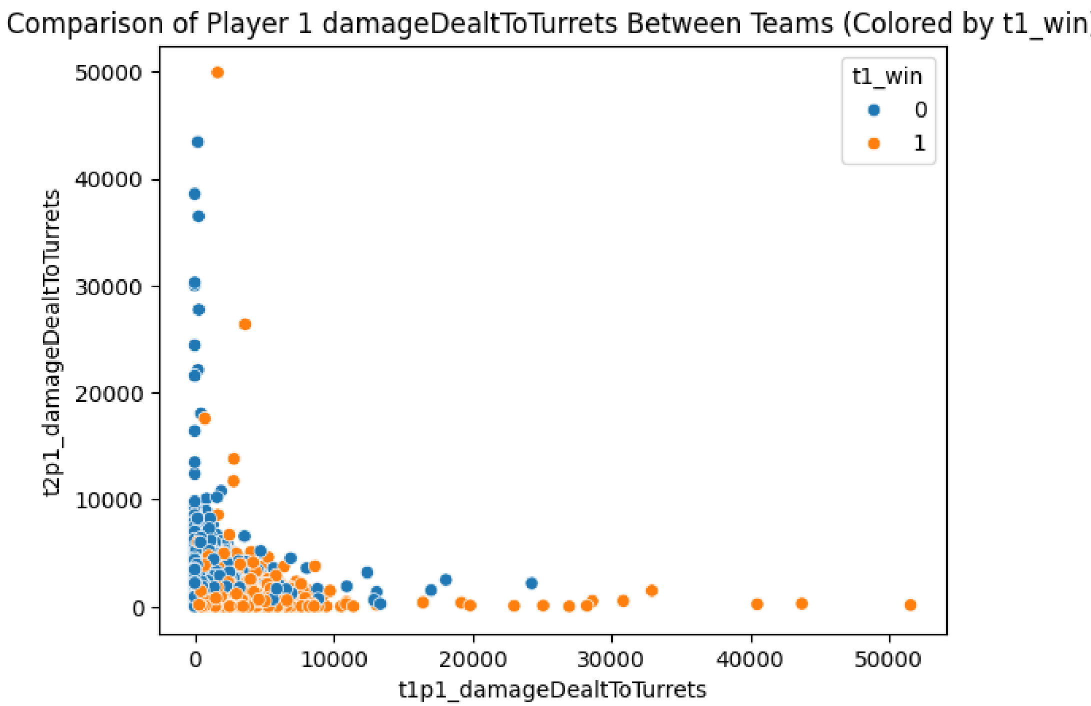




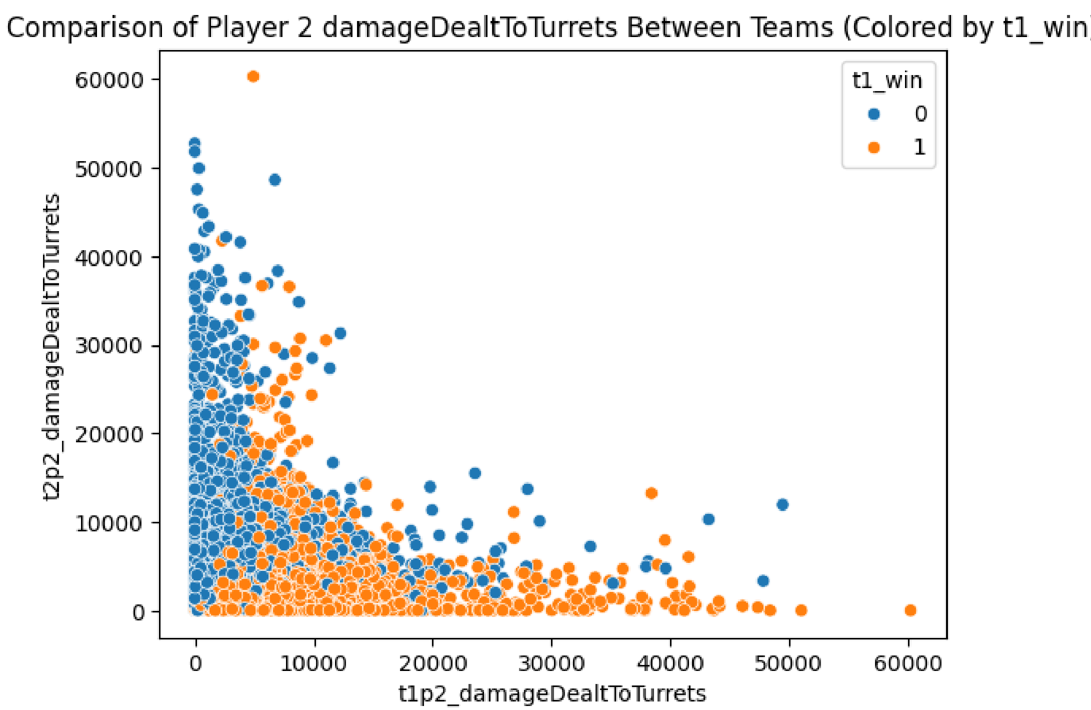


We check damage dealt to turrets. (comparision is from “player 1 on team 1” to “player 1 on team 2”, this ensures the comparison is done based on the role each player is playing)

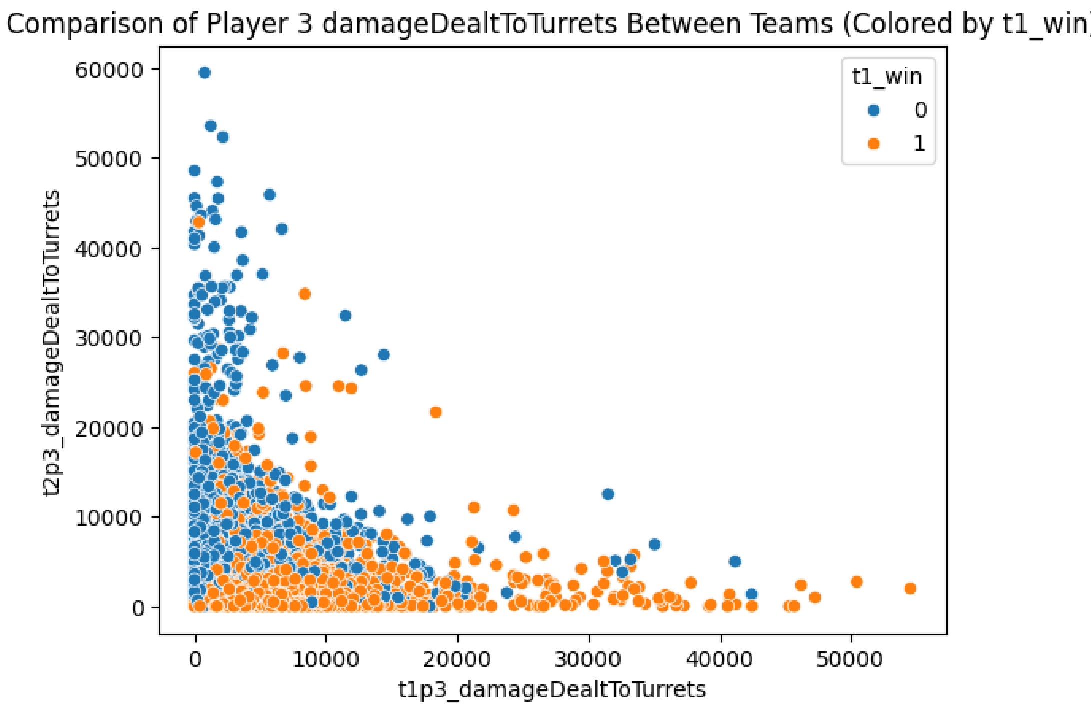
[80]:
```
sns.scatterplot(x='t1p1_damageDealtToTurrets', y='t2p1_damageDealtToTurrets',␣
↪data=df, hue='t1_win')
plt.title('Comparison of Player 1 damageDealtToTurrets Between Teams (Colored␣
↪by t1_win)')
plt.show()


sns.scatterplot(x='t1p2_damageDealtToTurrets', y='t2p2_damageDealtToTurrets',␣
↪data=df, hue='t1_win')
plt.title('Comparison of Player 2 damageDealtToTurrets Between Teams (Colored␣
↪by t1_win)')
plt.show()

sns.scatterplot(x='t1p3_damageDealtToTurrets', y='t2p3_damageDealtToTurrets',␣
↪data=df, hue='t1_win')
plt.title('Comparison of Player 3 damageDealtToTurrets Between Teams (Colored␣
↪by t1_win)')
plt.show()

sns.scatterplot(x='t1p4_damageDealtToTurrets', y='t2p4_damageDealtToTurrets',␣
↪data=df, hue='t1_win')
plt.title('Comparison of Player 4 damageDealtToTurrets Between Teams (Colored␣
↪by t1_win)')
plt.show()

sns.scatterplot(x='t1p5_damageDealtToTurrets', y='t2p5_damageDealtToTurrets',␣
↪data=df, hue='t1_win')
plt.title('Comparison of Player 5 damageDealtToTurrets Between Teams (Colored␣
↪by t1_win)')
plt.show()
```

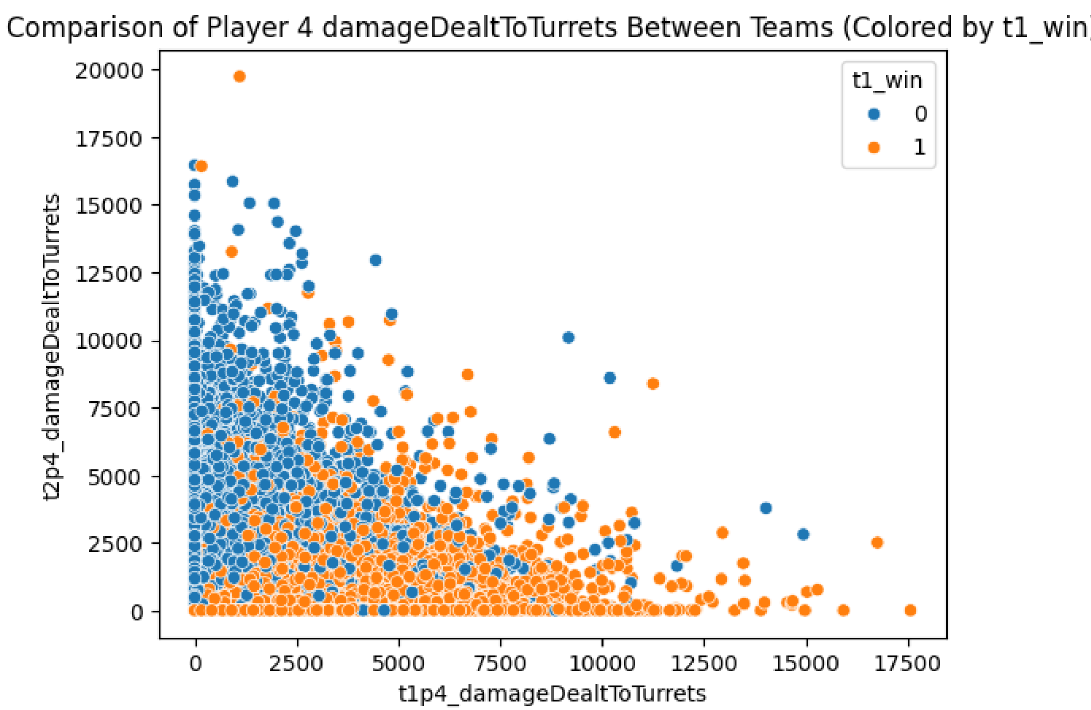


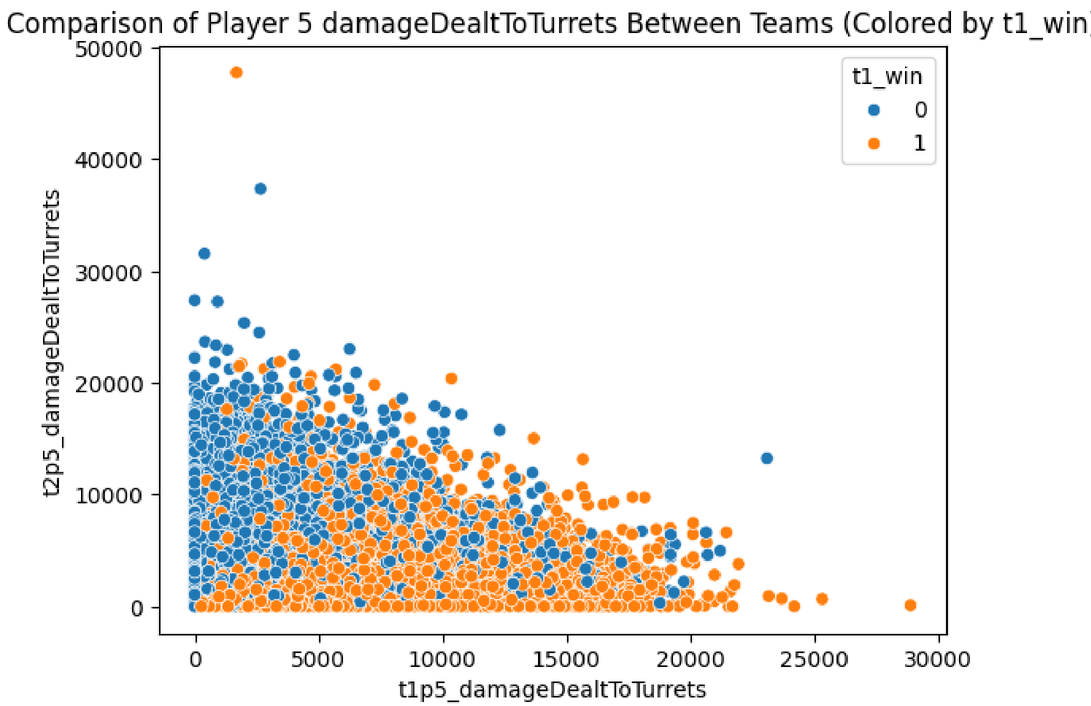
Comparison of Player 1 damageDealtToTurrets Between Teams (Colored by t1_win)
t1_win
0
1
t2p1_damageDealtToTurrets
t1p1_damageDealtToTurrets
0
10000
20000
30000
40000
50000



Comparison of Player 2 damageDealtToTurrets Between Teams (Colored by t1_win)
t1_win
0
1
t2p2_damageDealtToTurrets
t1p2_damageDealtToTurrets
0
10000
20000
30000
40000
50000
60000

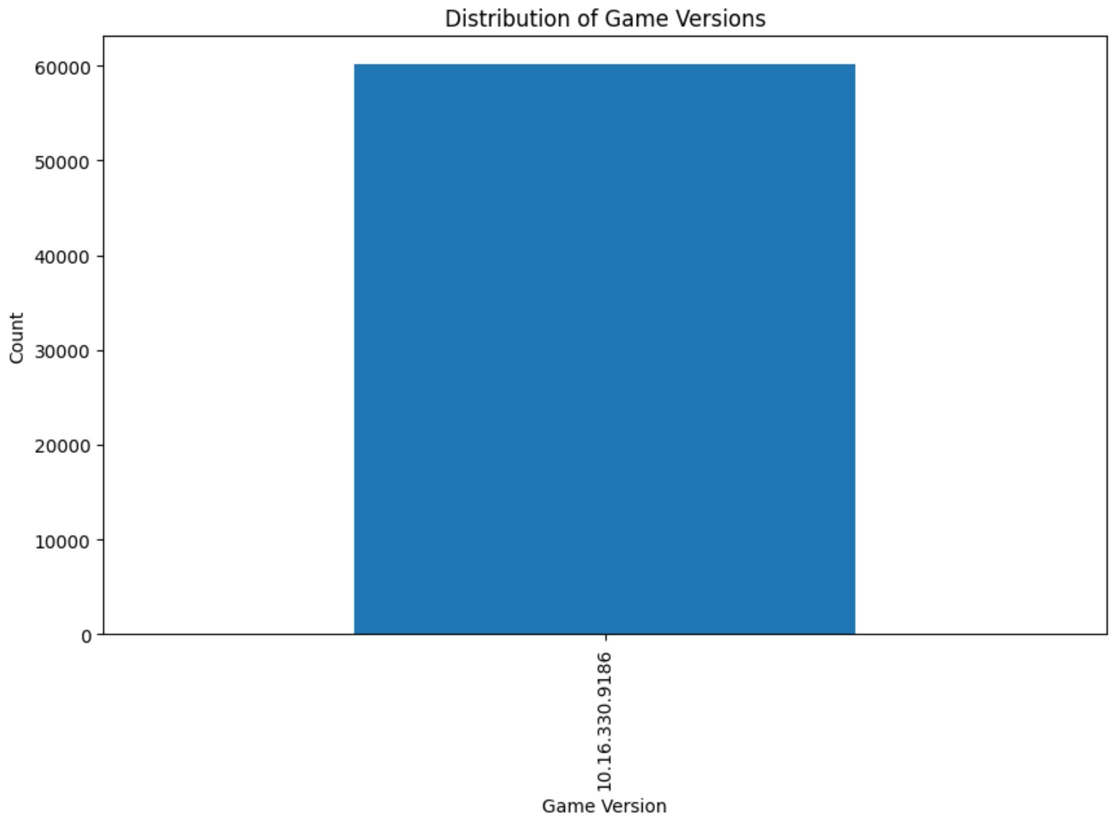

Comparison of Player 3 damageDealtToTurrets Between Teams (Colored by t1_win)
t1_win
0
1
60000
50000
40000
30000
20000
10000
0
10000
20000
30000
40000
50000
t2p3_damageDealtToTurrets
t1p3_damageDealtToTurrets


Comparison of Player 4 damageDealtToTurrets Between Teams (Colored by t1_win)
t1_win
0
1
20000
17500
15000
12500
10000
7500
5000
2500
0
2500
5000
7500
10000
12500
15000
17500
t2p4_damageDealtToTurrets
t1p4_damageDealtToTurrets

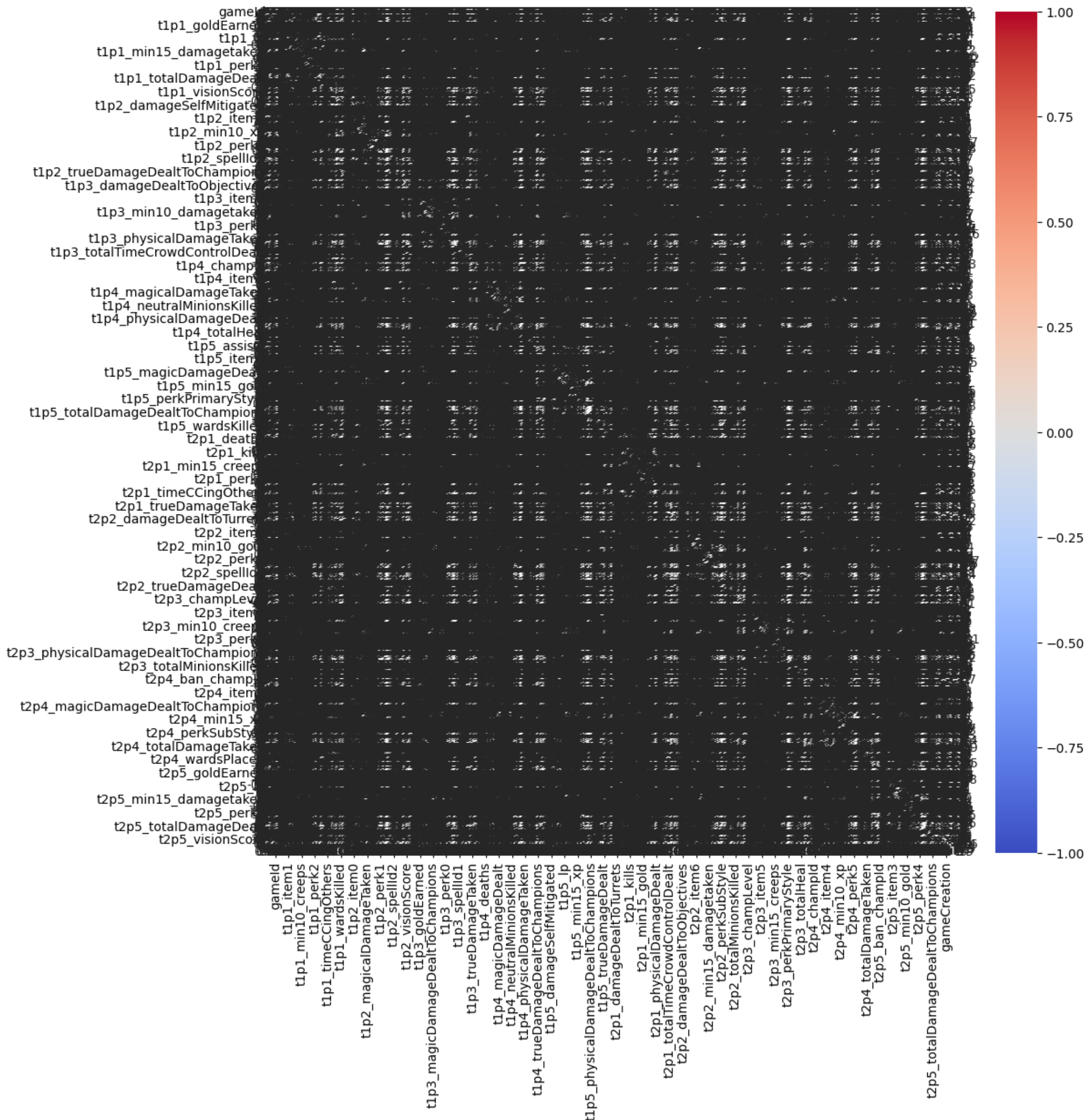


Lastly, we check to see that all games(rows) are played on the same game version, if so we will remove this column as it not meaningful.

[81]:
```
game_version_counts = df['gameVersion'].value_counts()

plt.figure(figsize=(10,6))
game_version_counts.plot(kind='bar')
plt.title("Distribution of Game Versions")
plt.xlabel("Game Version")
plt.ylabel("Count")
plt.show()
```



[82]:
```
corr_matrix = df.select_dtypes(include=['number']).corr()
plt.figure(figsize=(12, 12))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', vmin=-1)
plt.show()
```



Unfortunaty, due to such high dimensionality we can't physically see anything from the heatmap of all remaining columns together.

5 Data Preprocessing

We drop irrelevant features such as players name's, gameID's, etc.(these features hold no bearing on the winner or loser). We also dropped many features that are be very deterministic (leading to data leakage) and would only be available after the game is over. We would like to predict the winner while a game is still happening so we must get rid of those features, and do some Feature Engineering after.

[83]:
```
df = df.drop(columns=["gameId", "t1p1_accountId", "t1p2_accountId",␣
 ↪"t1p3_accountId", "t1p4_accountId", "t1p5_accountId", "t1p1_summonerId",␣
 ↪"t1p2_summonerId", "t1p3_summonerId", "t1p4_summonerId", "t1p5_summonerId",
               ␣
 ↪"t1p1_summonerName","t1p2_summonerName","t1p3_summonerName","t1p4_summonerName","t1p5_summo
 ↪"t2p1_accountId", "t2p2_accountId", "t2p3_accountId", "t2p4_accountId",␣
 ↪"t2p5_accountId", "t1p1_assists",
                "t1p2_assists", "t1p3_assists", "t1p4_assists", "t1p5_assists",␣
 ↪"t2p1_assists", "t2p2_assists", "t2p3_assists", "t2p4_assists",␣
 ↪"t2p5_assists", "t1p1_deaths", "t1p2_deaths", "t1p3_deaths", "t1p4_deaths",␣
 ↪"t1p5_deaths",
                "t2p1_deaths", "t2p2_deaths", "t2p3_deaths", "t2p4_deaths",␣
 ↪"t2p5_deaths", "t2p1_summonerId", "t2p2_summonerId", "t2p3_summonerId",␣
 ↪"t2p4_summonerId",␣
 ↪"t2p5_summonerId","t2p1_summonerName","t2p2_summonerName","t2p3_summonerName",
                "t2p4_summonerName","t2p5_summonerName",␣
 ↪"t1p1_damageDealtToObjectives", "t1p2_damageDealtToObjectives",␣
 ↪"t1p3_damageDealtToObjectives", "t1p4_damageDealtToObjectives",␣
 ↪"t1p5_damageDealtToObjectives", "t2p1_damageDealtToObjectives",
                "t2p2_damageDealtToObjectives", "t2p3_damageDealtToObjectives",␣
 ↪"t2p4_damageDealtToObjectives", "t2p5_damageDealtToObjectives",␣
 ↪"t1p1_damageDealtToTurrets", "t1p2_damageDealtToTurrets",␣
 ↪"t1p3_damageDealtToTurrets", "t1p4_damageDealtToTurrets",
                "t1p5_damageDealtToTurrets", "t2p1_damageDealtToTurrets",␣
 ↪"t2p2_damageDealtToTurrets", "t2p3_damageDealtToTurrets",␣
 ↪"t2p4_damageDealtToTurrets", "t2p5_damageDealtToTurrets",␣
 ↪"t1p1_damageSelfMitigated", "t1p2_damageSelfMitigated",
                "t1p3_damageSelfMitigated", "t1p4_damageSelfMitigated",␣
 ↪"t1p5_damageSelfMitigated", "t2p1_damageSelfMitigated",␣
 ↪"t2p2_damageSelfMitigated", "t2p3_damageSelfMitigated",␣
 ↪"t2p4_damageSelfMitigated", "t2p5_damageSelfMitigated",
                "t1p1_neutralMinionsKilled", "t1p2_neutralMinionsKilled",␣
 ↪"t1p3_neutralMinionsKilled", "t1p4_neutralMinionsKilled",␣
 ↪"t1p5_neutralMinionsKilled", "t2p1_neutralMinionsKilled",␣
 ↪"t2p2_neutralMinionsKilled",
                "t2p3_neutralMinionsKilled", "t2p4_neutralMinionsKilled",␣
 ↪"t2p5_neutralMinionsKilled", "t1p1_magicDamageDealtToChampions",␣
 ↪"t1p2_magicDamageDealtToChampions", "t1p3_magicDamageDealtToChampions",␣
 ↪"t1p4_magicDamageDealtToChampions",
                "t1p5_magicDamageDealtToChampions",␣
 ↪"t2p1_magicDamageDealtToChampions", "t2p2_magicDamageDealtToChampions",␣
 ↪"t2p3_magicDamageDealtToChampions", "t2p4_magicDamageDealtToChampions",␣
 ↪"t2p5_magicDamageDealtToChampions",
```

```
               "t1p1_magicDamageDealt", "t1p2_magicDamageDealt",␣
↪"t1p3_magicDamageDealt", "t1p4_magicDamageDealt", "t1p5_magicDamageDealt",␣
↪"t2p1_magicDamageDealt", "t2p2_magicDamageDealt", "t2p3_magicDamageDealt",␣
↪"t2p4_magicDamageDealt",
               "t2p5_magicDamageDealt", "t1p1_magicalDamageTaken",␣
↪"t1p2_magicalDamageTaken", "t1p3_magicalDamageTaken",␣
↪"t1p4_magicalDamageTaken", "t1p5_magicalDamageTaken",␣
↪"t2p1_magicalDamageTaken", "t2p2_magicalDamageTaken",
               "t2p3_magicalDamageTaken", "t2p4_magicalDamageTaken",␣
↪"t2p5_magicalDamageTaken", "t1p1_physicalDamageDealt",␣
↪"t1p2_physicalDamageDealt", "t1p3_physicalDamageDealt",␣
↪"t1p4_physicalDamageDealt", "t1p5_physicalDamageDealt",
               "t2p1_physicalDamageDealt", "t1p2_physicalDamageDealt",␣
↪"t2p2_physicalDamageDealt", "t2p3_physicalDamageDealt",␣
↪"t2p4_physicalDamageDealt", "t2p5_physicalDamageDealt",␣
↪"t1p1_physicalDamageDealtToChampions",
               "t1p2_physicalDamageDealtToChampions",␣
↪"t1p3_physicalDamageDealtToChampions",␣
↪"t1p4_physicalDamageDealtToChampions",␣
↪"t1p5_physicalDamageDealtToChampions",␣
↪"t2p1_physicalDamageDealtToChampions", "t2p2_physicalDamageDealtToChampions",
               "t2p3_physicalDamageDealtToChampions",␣
↪"t2p4_physicalDamageDealtToChampions",␣
↪"t2p5_physicalDamageDealtToChampions", "t1p1_physicalDamageTaken",␣
↪"t1p2_physicalDamageTaken", "t1p3_physicalDamageTaken",␣
↪"t1p4_physicalDamageTaken",
               "t1p5_physicalDamageTaken", "t2p1_physicalDamageTaken",␣
↪"t2p2_physicalDamageTaken", "t2p3_physicalDamageTaken",␣
↪"t2p4_physicalDamageTaken", "t2p5_physicalDamageTaken",␣
↪"t1p1_timeCCingOthers", "t1p2_timeCCingOthers", "t1p3_timeCCingOthers",
               "t1p4_timeCCingOthers", "t1p5_timeCCingOthers",␣
↪"t2p1_timeCCingOthers", "t2p2_timeCCingOthers", "t2p3_timeCCingOthers",␣
↪"t2p4_timeCCingOthers", "t2p5_timeCCingOthers", "t1p1_totalDamageDealt",␣
↪"t1p1_totalDamageDealt", "t1p2_totalDamageDealt",
               "t1p3_totalDamageDealt", "t1p4_totalDamageDealt",␣
↪"t1p5_totalDamageDealt", "t2p1_totalDamageDealt", "t2p2_totalDamageDealt",␣
↪"t2p3_totalDamageDealt", "t2p4_totalDamageDealt", "t2p5_totalDamageDealt",␣
↪"t1p1_totalDamageDealtToChampions",
               "t1p2_totalDamageDealtToChampions",␣
↪"t1p3_totalDamageDealtToChampions", "t1p4_totalDamageDealtToChampions",␣
↪"t1p5_totalDamageDealtToChampions", "t2p1_totalDamageDealtToChampions",␣
↪"t2p2_totalDamageDealtToChampions", "t2p3_totalDamageDealtToChampions",
               "t2p4_totalDamageDealtToChampions",␣
↪"t2p5_totalDamageDealtToChampions", "t1p1_totalDamageTaken",␣
↪"t1p2_totalDamageTaken", "t1p3_totalDamageTaken", "t1p4_totalDamageTaken",␣
↪"t1p5_totalDamageTaken", "t2p1_totalDamageTaken", "t2p2_totalDamageTaken",
```

```
            "t2p3_totalDamageTaken", "t2p4_totalDamageTaken",␣
↪"t2p5_totalDamageTaken", "t1p1_totalHeal", "t1p2_totalHeal",␣
↪"t1p3_totalHeal", "t1p4_totalHeal", "t1p5_totalHeal", "t2p1_totalHeal",␣
↪"t2p2_totalHeal", "t2p3_totalHeal", "t2p4_totalHeal",
            "t2p5_totalHeal", "t1p1_totalMinionsKilled",␣
↪"t1p2_totalMinionsKilled", "t1p3_totalMinionsKilled",␣
↪"t1p4_totalMinionsKilled", "t1p5_totalMinionsKilled",␣
↪"t2p1_totalMinionsKilled", "t2p2_totalMinionsKilled",␣
↪"t2p3_totalMinionsKilled",
            "t2p4_totalMinionsKilled", "t2p5_totalMinionsKilled",␣
↪"t1p1_totalTimeCrowdControlDealt", "t1p2_totalTimeCrowdControlDealt",␣
↪"t1p3_totalTimeCrowdControlDealt", "t1p4_totalTimeCrowdControlDealt",␣
↪"t1p5_totalTimeCrowdControlDealt",
            "t2p1_totalTimeCrowdControlDealt",␣
↪"t2p2_totalTimeCrowdControlDealt", "t2p3_totalTimeCrowdControlDealt",␣
↪"t2p4_totalTimeCrowdControlDealt", "t2p5_totalTimeCrowdControlDealt",␣
↪"t1p1_trueDamageDealt", "t1p2_trueDamageDealt",
            "t1p3_trueDamageDealt", "t1p4_trueDamageDealt",␣
↪"t1p5_trueDamageDealt", "t2p1_trueDamageDealt", "t2p2_trueDamageDealt",␣
↪"t2p3_trueDamageDealt", "t2p4_trueDamageDealt", "t2p5_trueDamageDealt",␣
↪"t1p1_trueDamageDealtToChampions",
            "t1p2_trueDamageDealtToChampions",␣
↪"t1p3_trueDamageDealtToChampions", "t1p4_trueDamageDealtToChampions",␣
↪"t1p5_trueDamageDealtToChampions", "t2p1_trueDamageDealtToChampions",␣
↪"t2p2_trueDamageDealtToChampions", "t2p3_trueDamageDealtToChampions",
            "t2p4_trueDamageDealtToChampions",␣
↪"t2p5_trueDamageDealtToChampions", "t1p1_trueDamageTaken",␣
↪"t1p2_trueDamageTaken", "t1p3_trueDamageTaken", "t1p4_trueDamageTaken",␣
↪"t1p5_trueDamageTaken", "t2p1_trueDamageTaken", "t2p2_trueDamageTaken",
            "t2p3_trueDamageTaken", "t2p4_trueDamageTaken",␣
↪"t2p5_trueDamageTaken", "t1p1_wardsKilled", "t1p2_wardsKilled",␣
↪"t1p3_wardsKilled", "t1p4_wardsKilled", "t1p5_wardsKilled",␣
↪"t2p1_wardsKilled", "t2p2_wardsKilled", "t2p3_wardsKilled",
            "t2p4_wardsKilled", "t2p5_wardsKilled", "t1p1_wardsPlaced",␣
↪"t1p2_wardsKilled", "t1p3_wardsKilled", "t1p4_wardsKilled",␣
↪"t1p5_wardsKilled", "t2p1_wardsKilled", "t2p2_wardsKilled",␣
↪"t2p3_wardsKilled", "t2p4_wardsKilled", "t2p5_wardsKilled",
            "t1p1_wardsPlaced", "t1p2_wardsPlaced", "t1p3_wardsPlaced",␣
↪"t1p4_wardsPlaced", "t1p5_wardsPlaced", "t2p1_wardsPlaced",␣
↪"t2p2_wardsPlaced", "t2p3_wardsPlaced", "t2p4_wardsPlaced",␣
↪"t2p5_wardsPlaced", "t1p1_goldEarned", "t1p2_goldEarned",
            "t1p3_goldEarned", "t1p4_goldEarned", "t1p5_goldEarned",␣
↪"t2p1_goldEarned", "t2p2_goldEarned", "t2p3_goldEarned", "t2p4_goldEarned",␣
↪"t2p5_goldEarned", "t1p1_kills", "t1p2_kills", "t1p3_kills", "t1p4_kills",␣
↪"t1p5_kills", "t2p1_kills",
```

```
                "t2p2_kills", "t2p3_kills", "t2p4_kills", "t2p5_kills",␣
↪"t1p1_deaths", "t1p2_deaths", "t1p3_deaths", "t1p4_deaths", "t1p5_deaths",␣
↪"t2p1_deaths", "t2p2_deaths", "t2p3_deaths", "t2p4_deaths", "t2p5_deaths",␣
↪"gameVersion",
                "platformId", "gameCreation", "queueId", "gameDuration"])
df.head()
```

[83]:

```
   t1p1_ban_champId  t1p1_champId  t1p1_champLevel  t1p1_item0  t1p1_item1  \
0               121            25               11        3158        3157
1               111            37               13        1001        3174
2               122            89               11        3193        3857
3                81            89               11        3860        3050
4                91            25               11           0        3028

   t1p1_item2  t1p1_item3  t1p1_item4  t1p1_item5  t1p1_item6  …  \
0        3860        3114        2055        1028        3364  …
1        3504        3853        3114        1082        3364  …
2        3105        3009        2055        1033        3364  …
3        3109        2055        3047           0        3364  …
4        3860        2065        3158        3108        3364  …

   t2p5_perk5  t2p5_perkPrimaryStyle  t2p5_perkSubStyle  t2p5_role  \
0        8316                   8200             8300.0        TOP
1        8234                   8000             8200.0        TOP
2        8451                   8000             8400.0        TOP
3        8473                   8000             8400.0        TOP
4        8345                   8000             8300.0        TOP

   t2p5_spellId1  t2p5_spellId2  t2p5_visionScore  average_lp  t1_teamId  \
0              4             12                13      3380.4        100
1              4              6                11      3284.9        100
2             12              4                27      3333.2        100
3              4              6                 7      3258.1        100
4              4             12                14      3116.0        100

   t1_win
0       0
1       1
2       0
3       1
4       0

[5 rows x 313 columns]
```

[84]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 60107 entries, 0 to 60155
```

```
Columns: 313 entries, t1p1_ban_champId to t1_win
dtypes: float64(96), int64(207), object(10)
memory usage: 144.0+ MB
```

Feature engieering on team level statistics

[85]:
```
for stat in ['creeps', 'damagetaken', 'gold', 'xp']:
    #Compute team totals for 10 min and 15 min
    df[f't1_{stat}_10'] = df[[f't1p{i}_min10_{stat}' for i in range(1,6)]].
 ↪sum(axis=1)
    df[f't2_{stat}_10'] = df[[f't2p{i}_min10_{stat}' for i in range(1,6)]].
 ↪sum(axis=1)
    df[f't1_{stat}_15'] = df[[f't1p{i}_min15_{stat}' for i in range(1,6)]].
 ↪sum(axis=1)
    df[f't2_{stat}_15'] = df[[f't2p{i}_min15_{stat}' for i in range(1,6)]].
 ↪sum(axis=1)
    #Compute the gains of a team (momentum)
    df[f't1_{stat}_gain'] = df[f't1_{stat}_15'] - df[f't1_{stat}_10']
    df[f't2_{stat}_gain'] = df[f't2_{stat}_15'] - df[f't2_{stat}_10']
    df[f'{stat}_gain_diff'] = df[f't1_{stat}_gain'] - df[f't2_{stat}_gain']
    #Compute difference between teams at 10 min and 15 min
    df[f'{stat}_diff_10'] = df[f't1_{stat}_10'] - df[f't2_{stat}_10']
    df[f'{stat}_diff_15'] = df[f't1_{stat}_15'] - df[f't2_{stat}_15']
    #Normalize the gains over 5 min window
    df[f't1_{stat}_gain_per_min'] = df[f't1_{stat}_gain'] / 5
    df[f't2_{stat}_gain_per_min'] = df[f't2_{stat}_gain'] / 5
    #Compute total gains difference per minute
    df[f'{stat}_gain_diff_per_min'] = df[f't1_{stat}_gain_per_min'] -␣
 ↪df[f't2_{stat}_gain_per_min']
```

[86]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 60107 entries, 0 to 60155
Columns: 361 entries, t1p1_ban_champId to xp_gain_diff_per_min
dtypes: float64(144), int64(207), object(10)
memory usage: 166.0+ MB
```

We create dummy variables for the roles players are playing. Currently a string value, we change to a boolean. Why did we keep roles column though? Because the "stats" of each player are tied to their role, which will be helpful for the model.

[87]:
```
df_dummies = pd.get_dummies(df, columns=['t1p1_role', 't1p2_role', 't1p3_role',␣
 ↪'t1p4_role', 't1p5_role','t2p1_role', 't2p2_role', 't2p3_role', 't2p4_role',␣
 ↪'t2p5_role'], drop_first=False)
print(df_dummies)
```

```
      t1p1_ban_champId  t1p1_champId  t1p1_champLevel  t1p1_item0  \
0                  121            25               11        3158
```

```
1                    111          37              13        1001
2                    122          89              11        3193
3                     81          89              11        3860
4                     91          25              11           0
…                  …          …              …          …
60151                111         350               7        3850
60152                104         235               8        1001
60153                238          80              15        3190
60154                141         432               9        3859
60155                 -1         432               8        2055

       t1p1_item1  t1p1_item2  t1p1_item3  t1p1_item4  t1p1_item5  t1p1_item6  \
0            3157        3860        3114        2055        1028        3364
1            3174        3504        3853        3114        1082        3364
2            3857        3105        3009        2055        1033        3364
3            3050        3109        2055        3047           0        3364
4            3028        3860        2065        3158        3108        3364
…          …           …           …           …           …           …
60151        3028        3108           0        1004           0        3340
60152        2031        3863        1011        1028        1036        3340
60153        3009        3179        3857        3071        2055        3364
60154        1011        1029        3117        1004        2055        3364
60155        3028        3117        3108        1028        3858        3364

       …  t1p1_role_SUPPORT  t1p2_role_ADC  t1p3_role_MIDDLE  \
0      …               True           True              True
1      …               True           True              True
2      …               True           True              True
3      …               True           True              True
4      …               True           True              True
…    …                 …              …                 …
60151  …               True           True              True
60152  …               True           True              True
60153  …               True           True              True
60154  …               True           True              True
60155  …               True           True              True

       t1p4_role_JUNGLE  t1p5_role_TOP  t2p1_role_SUPPORT  t2p2_role_ADC  \
0                  True           True               True           True
1                  True           True               True           True
2                  True           True               True           True
3                  True           True               True           True
4                  True           True               True           True
…                  …            …                  …             …
60151              True           True               True           True
60152              True           True               True           True
60153              True           True               True           True
60154              True           True               True           True
```

```
60155                True             True                True             True

       t2p3_role_MIDDLE  t2p4_role_JUNGLE  t2p5_role_TOP
0                  True              True           True
1                  True              True           True
2                  True              True           True
3                  True              True           True
4                  True              True           True
…                   …                …               …
60151              True              True           True
60152              True              True           True
60153              True              True           True
60154              True              True           True
60155              True              True           True

[60107 rows x 361 columns]
```

[88]:

```
y = df_dummies["t1_win"]
x = df_dummies.drop("t1_win", axis = 1)
x
```

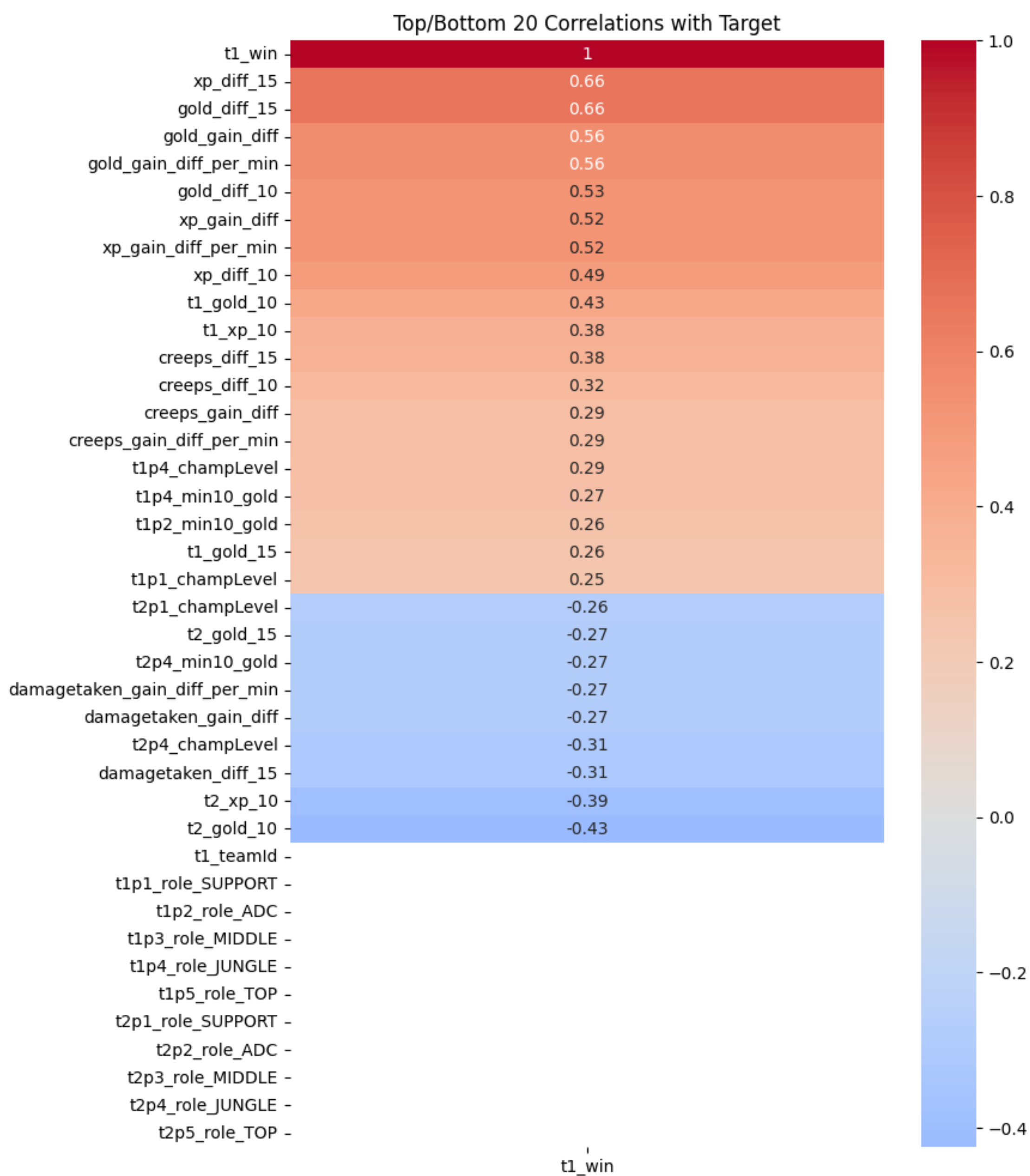
[88]:

```
       t1p1_ban_champId  t1p1_champId  t1p1_champLevel  t1p1_item0  \
0                   121            25               11        3158
1                   111            37               13        1001
2                   122            89               11        3193
3                    81            89               11        3860
4                    91            25               11           0
…                    …             …                …           …
60151               111           350                7        3850
60152               104           235                8        1001
60153               238            80               15        3190
60154               141           432                9        3859
60155                -1           432                8        2055

       t1p1_item1  t1p1_item2  t1p1_item3  t1p1_item4  t1p1_item5  t1p1_item6  \
0            3157        3860        3114        2055        1028        3364
1            3174        3504        3853        3114        1082        3364
2            3857        3105        3009        2055        1033        3364
3            3050        3109        2055        3047           0        3364
4            3028        3860        2065        3158        3108        3364
…            …           …           …           …           …
60151        3028        3108           0        1004           0        3340
60152        2031        3863        1011        1028        1036        3340
60153        3009        3179        3857        3071        2055        3364
60154        1011        1029        3117        1004        2055        3364
60155        3028        3117        3108        1028        3858        3364
```

```
       …  t1p1_role_SUPPORT  t1p2_role_ADC  t1p3_role_MIDDLE  \
0      …               True           True              True
1      …               True           True              True
2      …               True           True              True
3      …               True           True              True
4      …               True           True              True
…    …                 …              …               …
60151  …               True           True              True
60152  …               True           True              True
60153  …               True           True              True
60154  …               True           True              True
60155  …               True           True              True

       t1p4_role_JUNGLE  t1p5_role_TOP  t2p1_role_SUPPORT  t2p2_role_ADC  \
0                  True           True               True           True
1                  True           True               True           True
2                  True           True               True           True
3                  True           True               True           True
4                  True           True               True           True
…                  …              …                 …              …
60151              True           True               True           True
60152              True           True               True           True
60153              True           True               True           True
60154              True           True               True           True
60155              True           True               True           True

       t2p3_role_MIDDLE  t2p4_role_JUNGLE  t2p5_role_TOP
0                  True              True           True
1                  True              True           True
2                  True              True           True
3                  True              True           True
4                  True              True           True
…                  …                 …              …
60151              True              True           True
60152              True              True           True
60153              True              True           True
60154              True              True           True
60155              True              True           True

[60107 rows x 360 columns]
```

[89]:
```python
# Calculate correlations with target
corr_with_target = df_dummies.corr()[['t1_win']].sort_values('t1_win',␣
↪ascending=False)

# Top 20 and Bottom 20 features
top_20 = corr_with_target.head(20)
```

```
bottom_20 = corr_with_target.tail(20)
extreme_corr = pd.concat([top_20, bottom_20])

# Plot
plt.figure(figsize=(8, 12))
sns.heatmap(extreme_corr, annot=True, cmap='coolwarm', center=0)
plt.title("Top/Bottom 20 Correlations with Target")
plt.show()
```



Split the data into test/train sets

[90]:
```
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2,␣
↪random_state=1)
```

We scale AFTER splitting the test/train as we dont want to model to "peak" at the data early(Info on Data leakage from https://scikit-learn.org/stable/common_pitfalls.html). Scaling is still applied to both the test and train sets.

[91]:
```
#scale the features
scalar = StandardScaler()
X_train_scaled = scalar.fit_transform(X_train)
X_test_scaled = scalar.transform(X_test)
```

6 Logistic Regression

Creation of the logistic Regression model

[92]:
```
lr_model = LogisticRegression(penalty=None)
```

We fit the model.

[93]:
```
lr_model.fit(X_train_scaled, y_train)
```

[93]:
```
LogisticRegression(penalty=None)
```

[94]:
```
print("Coefficients:", lr_model.coef_)
print("Intercept:", lr_model.intercept_)
```

```
Coefficients: [[ 5.32047019e-02 -9.91487752e-02  3.30646862e+00 -1.44671371e-01
  -9.37654680e-02 -1.63359863e-01 -1.40351644e-01 -7.12023064e-02
  -1.99081355e-02  4.36669322e-02  5.00180135e-01 -1.39409626e-01
   3.81129443e-02  2.77948341e-01  7.01642875e-02  1.84319315e-01
  -8.52142082e-02 -3.04888524e-01 -3.14247671e-01 -5.10396986e-02
  -6.09877282e-02 -2.19865947e-02  8.13457344e-02  4.85292686e-02
   3.37198646e-02  7.84147401e-02 -4.68384375e-02  4.10416183e-02
  -1.52606797e-02  8.98802462e-02 -3.74512864e-02  6.55982275e-02
   2.89524689e+00 -1.40599036e-01 -2.97736032e-02 -4.24621918e-02
  -2.79461402e-04  1.90765932e-02 -9.57178982e-03  6.61578678e-02
   2.90525982e-01  2.31807683e-02  5.16380729e-02 -1.45730452e-01
   5.98655046e-02 -1.86572903e-02  4.16936703e-02  4.20162128e-01
  -2.18198727e-01 -1.10478011e-02  7.35324273e-02 -7.72267863e-02
  -1.38579976e-02 -1.25939089e-01 -9.46321027e-02  1.28006927e-01
   1.26114714e-01  1.77537086e-02 -9.29262448e-03  2.37607794e-02
   5.00069997e-02 -4.74082845e-02  2.47072249e+00 -9.30279173e-02
   2.61421399e-02 -6.41731896e-02 -5.69124783e-02 -7.77985163e-02
  -5.01950769e-03 -2.17398588e-02  3.61781049e-01 -5.34216687e-02
  -2.78930314e-03  4.12991023e-02 -1.86600402e-02  1.08731299e-01
  -3.86971004e-02  2.13227757e-01 -1.01834296e-01 -9.20413263e-02
```

-2.08995456e-02	8.61468378e-02	1.14529135e-01	2.84433893e-03
-8.80921545e-03	-2.30772082e-02	-2.68365145e-02	-5.42926072e-03
3.40257046e-02	8.58260934e-02	-2.71748736e-02	2.04243085e-02
2.37717009e+00	-5.54484585e-02	-2.28830832e-02	-6.15822341e-02
-2.88445914e-02	3.98886060e-02	-3.91062950e-03	-1.31382659e-02
4.42520742e-01	4.74774453e-02	2.52447092e-02	2.87666490e-02
-6.48872016e-02	-1.09039994e-01	-1.30045805e-01	2.02123651e-01
-5.06612763e-03	-4.62826679e-02	-9.94570758e-02	1.18949384e-01
7.51803145e-02	9.26622511e-02	-6.15008574e-02	1.04370475e-02
7.89026658e-02	1.22675677e-01	1.17575830e-01	1.28225572e-02
-5.75613637e-02	-2.16614667e-02	2.86518703e+00	-9.91768395e-02
-1.84528638e-02	7.59236440e-02	1.08707530e-02	-2.47611421e-02
2.83065110e-02	4.39444463e-02	3.89743999e-01	1.41853591e-02
-6.42338036e-02	1.11895012e-01	-9.48707058e-02	-9.25925200e-02
1.93963546e-01	1.19919359e-01	1.24087888e-01	9.44318823e-02
-9.41192098e-02	7.49444813e-03	1.23422012e-02	-1.21259275e-01
1.55084661e-02	-4.59933466e-02	5.73028614e-02	-1.52284873e-01
-1.38186531e-01	9.62740924e-02	-7.25620335e-02	8.04534821e-02
-3.26302764e+00	5.92712716e-02	1.48616019e-01	2.02236459e-01
1.59940795e-01	2.95207042e-02	5.79227692e-02	-3.65624859e-02
-3.71200174e-01	-1.22059968e-01	-6.45694443e-02	-7.98364217e-02
4.75096697e-02	9.92046014e-02	1.60020417e-01	1.53893111e-01
2.06835857e-01	3.56838488e-02	6.29245846e-02	-2.69230208e-02
-4.13884463e-02	5.08854209e-02	-8.38767930e-03	-1.75509863e-03
-1.62528656e-02	-5.44782114e-02	-3.73652989e-02	-4.21965302e-02
4.49924534e-02	-5.72320482e-02	-2.93553190e+00	1.43826034e-01
6.87109129e-02	8.72674808e-02	-8.16981557e-02	5.39392866e-02
1.78044692e-02	-1.36822055e-01	-2.79578815e-01	-1.53087093e-01
-2.01800374e-01	-1.38965405e-01	-6.75947355e-02	2.23793232e-01
8.41275136e-02	-2.25804630e-01	1.84492408e-01	-9.20414617e-02
2.26983891e-02	-2.21250468e-01	2.15275406e-02	-5.35413162e-02
3.63387899e-03	-2.22448145e-01	-4.57910465e-02	-3.14801839e-02
-7.29095207e-02	-1.02690097e-01	-2.82873940e-02	1.13192734e-03
-2.46242748e+00	6.24002266e-02	1.94561720e-02	3.25252815e-02
-3.02906969e-02	2.13673834e-03	2.35981380e-02	2.81760697e-02
-4.05806161e-01	-8.79944260e-02	-1.01671253e-01	1.66172837e-02
9.15399893e-02	5.58503429e-02	7.66206907e-02	-1.80782358e-01
-1.81099791e-01	-6.33035537e-02	3.79495759e-02	1.21957044e-03
-4.14004933e-02	6.05022222e-03	1.81048576e-02	6.10298774e-02
3.74967398e-02	5.11224965e-02	1.54440903e-02	-6.93966171e-02
-1.67869008e-02	-8.35105428e-03	-2.70229723e+00	5.08789328e-02
1.88045607e-02	1.33610904e-01	2.15764117e-02	1.88399219e-02
1.14557281e-01	3.28933605e-02	-4.08566068e-01	1.50449008e-01
1.46566089e-01	-2.06735491e-01	2.97635898e-02	-6.20411039e-02
-3.56018640e-02	1.52160226e-01	-6.97053415e-02	7.13140828e-02
-8.07251944e-03	-6.95982441e-02	4.48743249e-02	-9.07005941e-02
7.12595632e-02	-8.88142719e-02	-4.04846530e-02	-2.04292753e-03
7.95797647e-03	-8.47585442e-02	2.59220494e-02	7.35563727e-02

```
 -2.87301251e+00  3.09202875e-02 -4.43677308e-02  3.33692650e-02
  1.84373440e-02  5.61887624e-02  1.35065801e-02 -4.24313928e-03
 -4.85022559e-01  2.77879609e-01  1.54896380e-01  9.27877788e-02
 -8.87593010e-02 -1.96912691e-01 -2.12047375e-01 -3.39753570e-01
  5.58914356e-02 -1.48761530e-01 -1.53175023e-02  4.67536026e-02
 -2.38855693e-02  3.75826576e-02 -7.49040468e-02  5.93692692e-02
  5.53003765e-03 -1.13696754e-01 -1.85155335e-01 -4.10959169e-02
 -4.46622323e-03  0.00000000e+00 -4.06606826e-02  4.45439895e-02
  9.84997001e-03  3.72300938e-02  2.87382600e-02  2.34123630e-02
  1.96985260e-02 -6.07920115e-02 -4.47355743e-02  2.87382600e-02
  2.34123630e-02  1.96985260e-02  1.21392269e-02  3.00053903e-03
 -4.36037014e-03 -1.78072523e-03 -9.42359945e-03 -3.09843452e-03
 -1.90538417e-02  8.66488345e-03 -4.55534125e-03 -9.42359945e-03
 -3.09843452e-03 -1.90538417e-02  8.98221957e-02 -1.13982209e-01
  1.95814024e-01 -1.27877550e-01  1.77454193e-01 -9.61608661e-02
  6.02547763e-01  1.25824169e-01  4.09053600e-01  1.77454193e-01
 -9.61608661e-02  6.02547763e-01 -3.75794756e-02  4.48747701e-03
 -9.51398631e-02  2.66728144e-02 -9.10783596e-02  2.72032680e-02
 -3.16757940e-01 -2.68963430e-02 -2.16206018e-01 -9.10783596e-02
  2.72032680e-02 -3.16757940e-01  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]]
Intercept: [0.10424083]
```

We make predictions on the test set.

[95]:
```python
y_pred = lr_model.predict(X_test_scaled)
#y_prob = lr_model.predict_proba(X_test)
```

7 Random Forrest

[96]:
```python
# Initialize Random Forest Model
rf_model = RandomForestClassifier(n_estimators=200, max_depth=15,␣
↪min_samples_split=5, random_state=42)

# Train the model
rf_model.fit(X_train, y_train)

# Make predictions
y_pred_rf = rf_model.predict(X_test)
```

8 Model Evaluation

8.0.1 Logistic Regression Eval.

We check the classification report for the Logistic Regression model.

[97]:
```
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred, digits=4)

print(f"Test Accuracy: {accuracy:.4f}")
print("Classification Report:\n", report)
```

```
Test Accuracy: 0.9820
Classification Report:
               precision    recall  f1-score   support

           0     0.9824    0.9821    0.9822      6077
           1     0.9817    0.9820    0.9818      5945

    accuracy                         0.9820     12022
   macro avg     0.9820    0.9820    0.9820     12022
weighted avg     0.9820    0.9820    0.9820     12022
```

Cross-Validtion check of the Logistic Regression model (Using pipeline to ensure correct scaling)

[98]:
```
#pipeline to ensure the scaling of each fold happens appropriatly
lr_model = make_pipeline(StandardScaler(), LogisticRegression(max_iter=1000))

#cross-validation (10 folds)
cv_scores = cross_val_score(lr_model, x, y, cv=10, scoring='f1')

print("Cross-validation f1 scores for each fold:", cv_scores)
print("Mean cross-validation f1:", np.mean(cv_scores))
```

```
Cross-validation f1 scores for each fold: [0.97798378 0.9781746  0.97870233
0.98044415 0.97887091 0.98153034
 0.9829442  0.98207766 0.98285526 0.98115079]
Mean cross-validation f1: 0.9804734027886562
```

Dummy Classifer to show the model is learning and not just guessing (like guessing the majority class)

[99]:
```
from sklearn.dummy import DummyClassifier
dummy = DummyClassifier(strategy='most_frequent').fit(X_train, y_train)
print(f"Dummy F1: {f1_score(y_test, dummy.predict(X_test))}")
```

```
Dummy F1: 0.6617687983525352
```

Checks for class imbalance after splitting data into train/test sets.

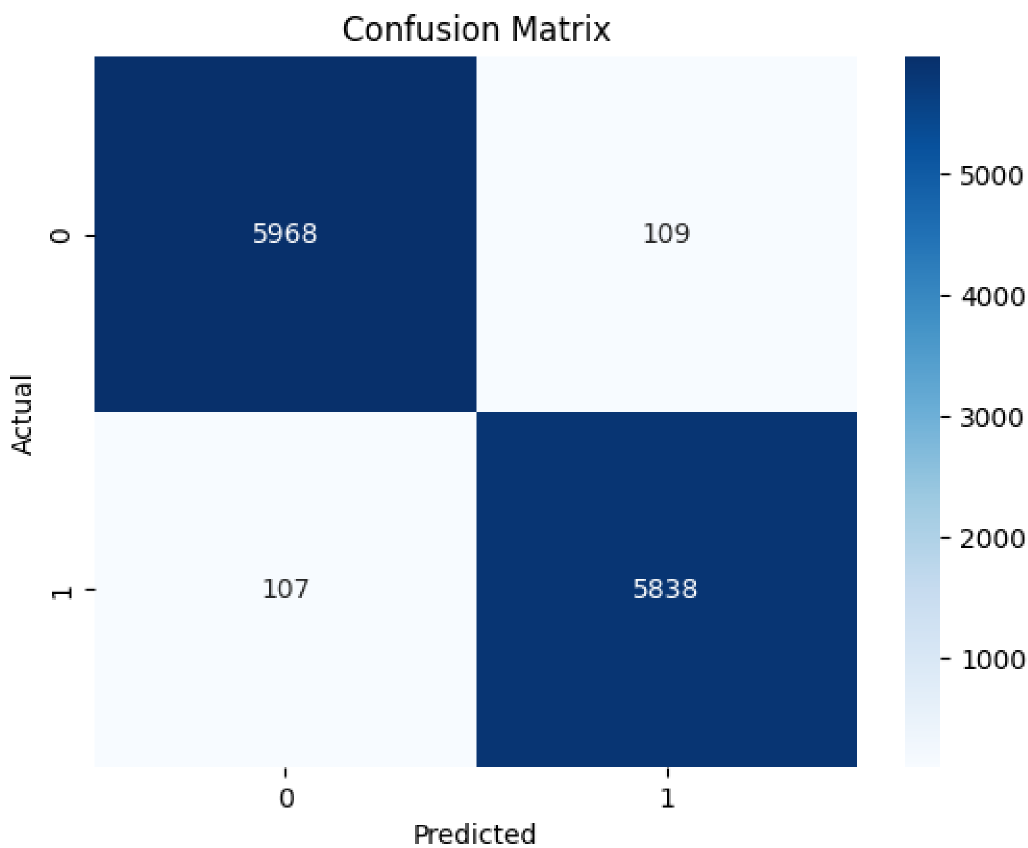
[100]:
```
# Check the distribution of classes in the training set
print("Class distribution in training data:")
print(y_train.value_counts())

# Check the distribution of classes in the test set
```

```
print("Class distribution in test data:")
print(y_test.value_counts())
```

```
Class distribution in training data:
t1_win
1    24304
0    23781
Name: count, dtype: int64
Class distribution in test data:
t1_win
0    6077
1    5945
Name: count, dtype: int64
```

We check the Confusion matrix of the Logistic Regression model

[101]:
```
cm = confusion_matrix(y_test, y_pred)

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')

plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```



8.0.2 Random Forrest Eval.

We check the classification report for the Random Forrest model.

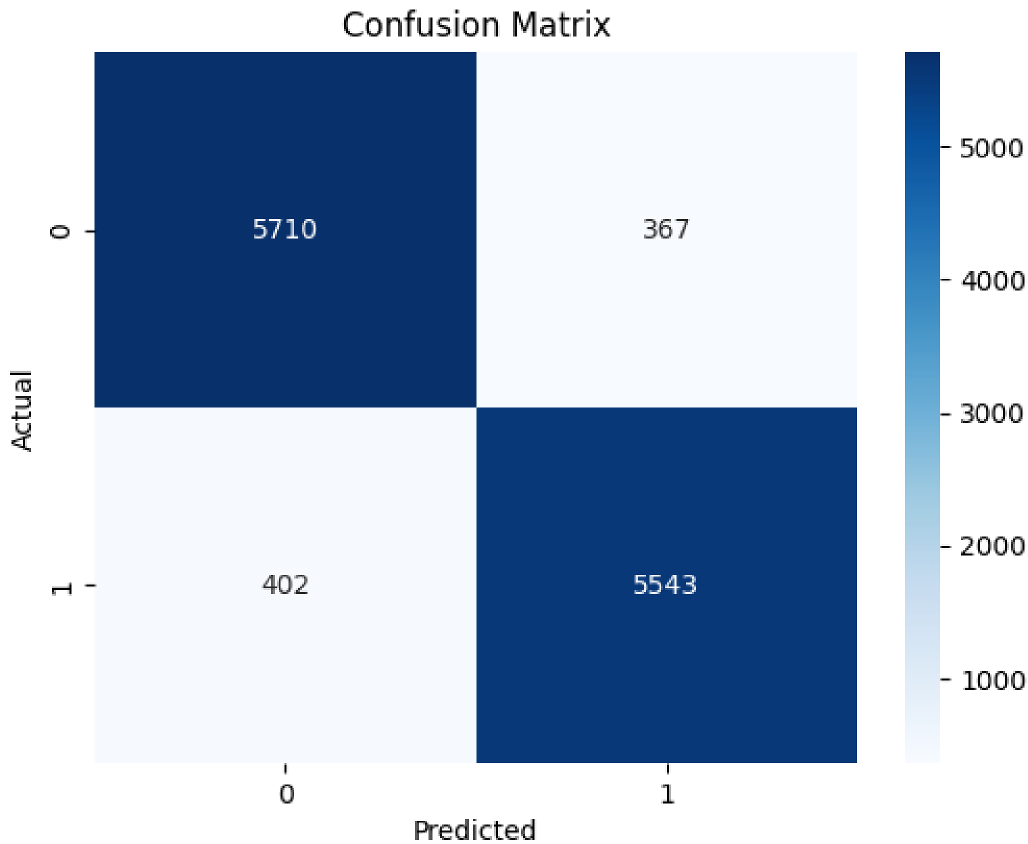
[102]:
```python
# Calculate metrics
accuracy = accuracy_score(y_test, y_pred_rf)
precision = precision_score(y_test, y_pred_rf)
recall = recall_score(y_test, y_pred_rf)
f1 = f1_score(y_test, y_pred_rf)

# Print the results
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {f1:.4f}")

# Show full classification report
print("\nClassification Report:\n", classification_report(y_test, y_pred_rf))
```

```
Accuracy: 0.9360
Precision: 0.9379
```

```
Recall: 0.9324
F1 Score: 0.9351

Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.94      0.94      6077
           1       0.94      0.93      0.94      5945

    accuracy                           0.94     12022
   macro avg       0.94      0.94      0.94     12022
weighted avg       0.94      0.94      0.94     12022
```

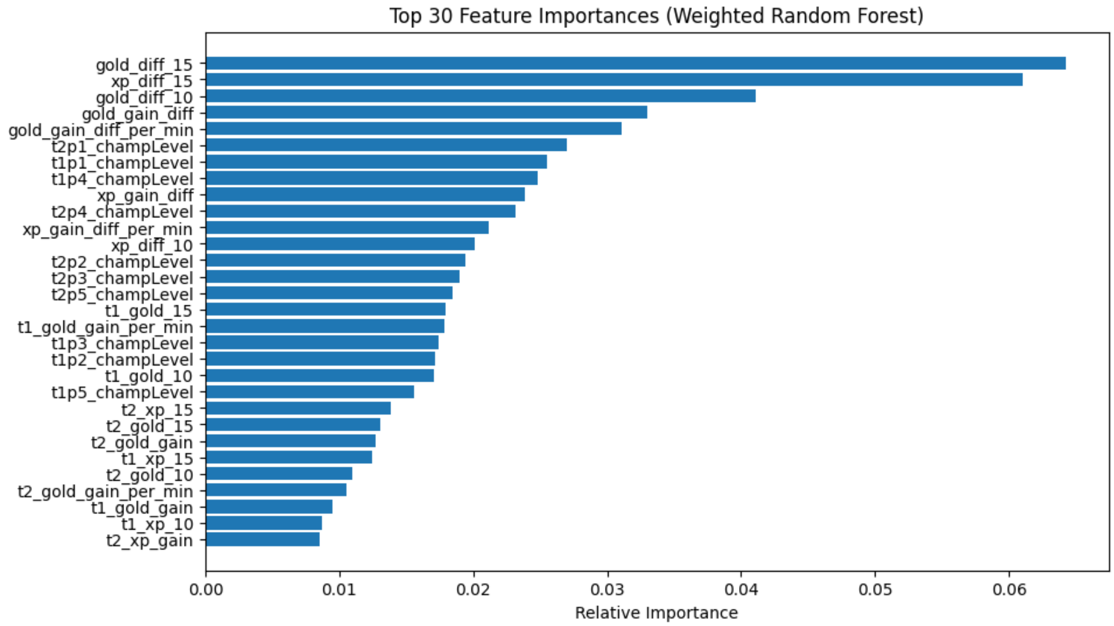
We check the Confusion Matrix for the Random Forrest model.

[103]:
```
# Confusion Matrix
cm_rf = confusion_matrix(y_test, y_pred_rf)

sns.heatmap(cm_rf, annot=True, fmt='d', cmap='Blues')

plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```



Feature importance for the Random Forrest model

[104]:
```
importances = rf_model.feature_importances_  # For the non-weighted model

# Get indices of top 30 most important features
indices = np.argsort(importances)[::-1][:30]  # Sort and take top 30 features

# Plot the top 30 features
plt.figure(figsize=(10, 6))
plt.title("Top 30 Feature Importances (Weighted Random Forest)")
plt.barh(range(30), importances[indices], align="center")
plt.yticks(range(30), X_train.columns[indices])  # Using the column names for
 ↪the top 30
plt.xlabel("Relative Importance")
plt.gca().invert_yaxis()  # To display the highest importance at the top
plt.show()
```



9 Model Tuning

Logistic Regression tuning using class_weight to balance weight on predictions.

[105]:
```
lr_model = LogisticRegression(class_weight='balanced')

lr_model.fit(X_train_scaled, y_train)

y_pred = lr_model.predict(X_test_scaled)
```

Random Forrest tuning via GridSearchCV

[106]:
```
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [10, 15, 20],
    'min_samples_split': [2, 5, 10, 15]
}
grid_search = GridSearchCV(
    estimator=RandomForestClassifier(random_state=42),
    param_grid=param_grid,
    cv=5,
    scoring='accuracy',
    n_jobs=-1,
    verbose=1
)
grid_search.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 36 candidates, totalling 180 fits
```

[106]:
```
GridSearchCV(cv=5, estimator=RandomForestClassifier(random_state=42), n_jobs=-1,
             param_grid={'max_depth': [10, 15, 20],
                         'min_samples_split': [2, 5, 10, 15],
                         'n_estimators': [100, 200, 300]},
             scoring='accuracy', verbose=1)
```

[107]:
```python
# Initialize Random Forest Model
rf_model = grid_search.best_estimator_

# Make predictions
y_pred_rf = rf_model.predict(X_test)
```

10 Results after tuning

Logistic Regression results after tuning

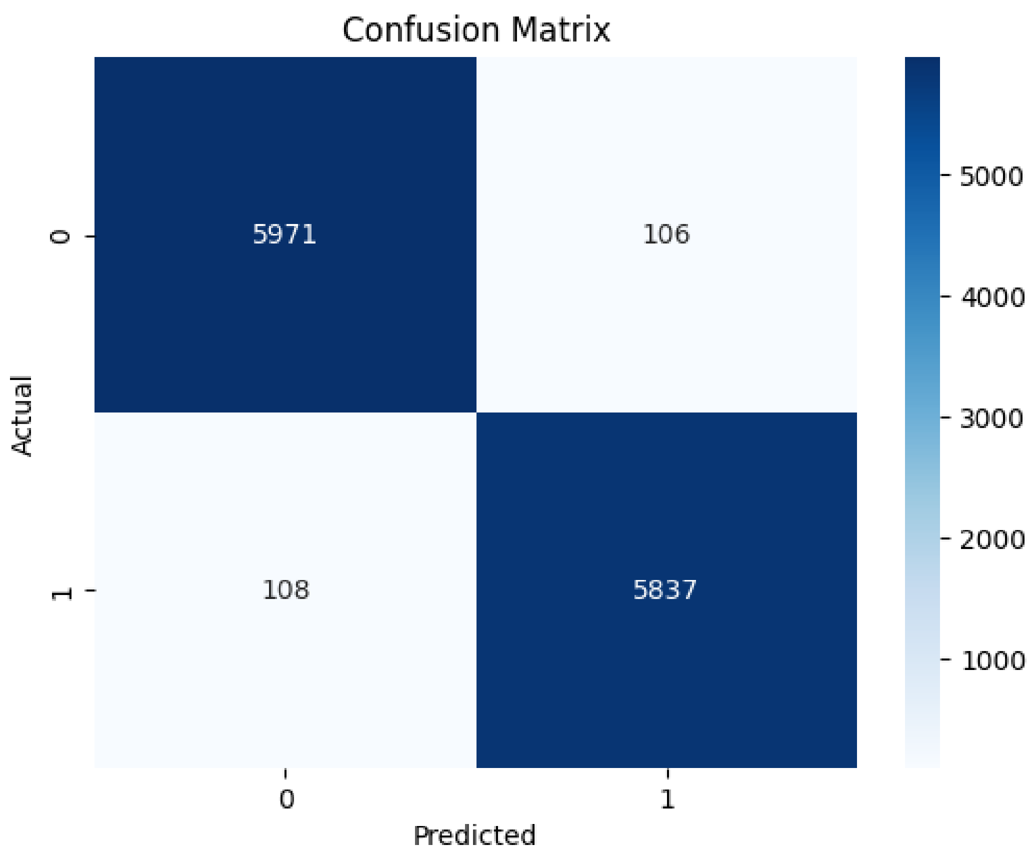
[108]:
```python
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred, digits=4)

print(f"Test Accuracy: {accuracy:.4f}")
print("Classification Report:\n", report)
```

```
Test Accuracy: 0.9822
Classification Report:
               precision    recall  f1-score   support

           0     0.9822    0.9826    0.9824      6077
           1     0.9822    0.9818    0.9820      5945

    accuracy                         0.9822     12022
   macro avg     0.9822    0.9822    0.9822     12022
weighted avg     0.9822    0.9822    0.9822     12022
```

[109]:
```python
#pipeline to ensure the scaling of each fold happens appropriatly
lr_model = make_pipeline(StandardScaler(), LogisticRegression(max_iter=1000))

#cross-validation (10 folds)
cv_scores = cross_val_score(lr_model, x, y, cv=10, scoring='f1')

print("Cross-validation f1 scores for each fold:", cv_scores)
print("Mean cross-validation f1-score:", np.mean(cv_scores))
```

```
Cross-validation f1 scores for each fold: [0.97798378 0.9781746  0.97870233
0.98044415 0.97887091 0.98153034
 0.9829442  0.98207766 0.98285526 0.98115079]
Mean cross-validation f1-score: 0.9804734027886562
```

[110]:
```
cm = confusion_matrix(y_test, y_pred)

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')

plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```



Overall, Logistic regression saw a very very small increase.

Random Forrest results after tuning

We check the classification report for the Random Forrest model.

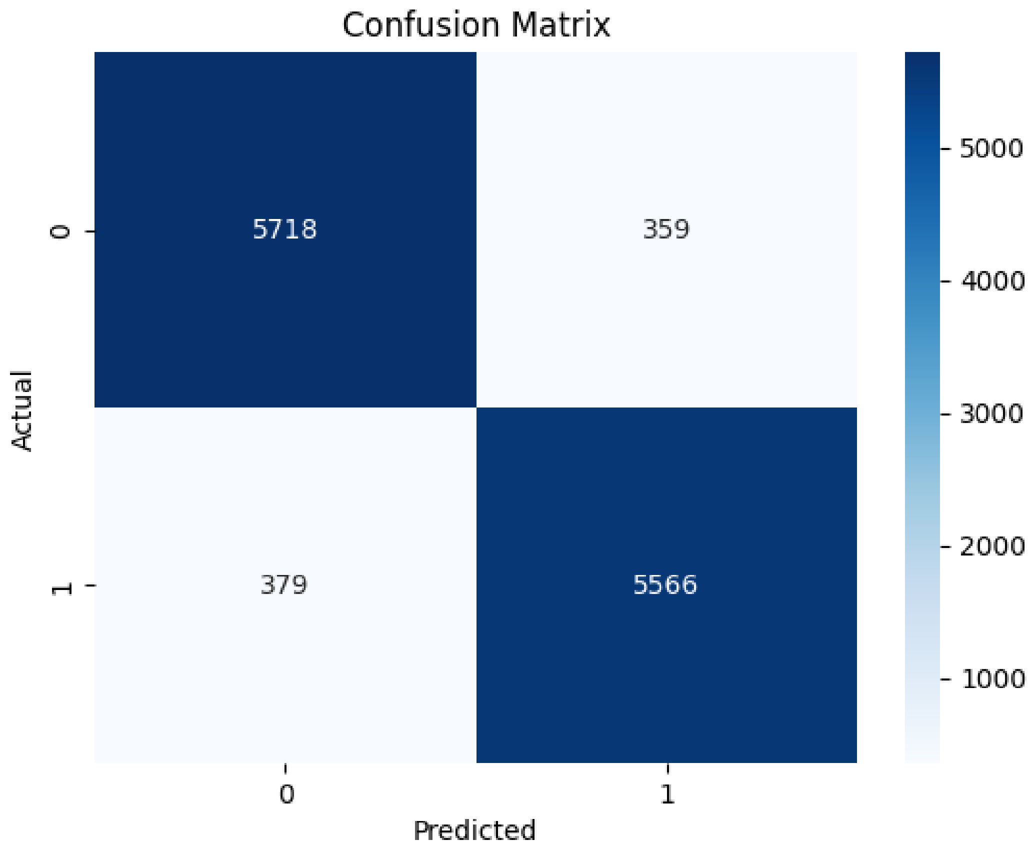
[111]:
```
# Calculate metrics
accuracy = accuracy_score(y_test, y_pred_rf)
precision = precision_score(y_test, y_pred_rf)
recall = recall_score(y_test, y_pred_rf)
```

```
f1 = f1_score(y_test, y_pred_rf)

# Print the results
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {f1:.4f}")

# Show full classification report
print("\nClassification Report:\n", classification_report(y_test, y_pred_rf))
```

```
Accuracy: 0.9386
Precision: 0.9394
Recall: 0.9362
F1 Score: 0.9378

Classification Report:
               precision    recall  f1-score   support

           0       0.94      0.94      0.94      6077
           1       0.94      0.94      0.94      5945

    accuracy                           0.94     12022
   macro avg       0.94      0.94      0.94     12022
weighted avg       0.94      0.94      0.94     12022
```

We check the Confusion Matrix for the Random Forrest model.

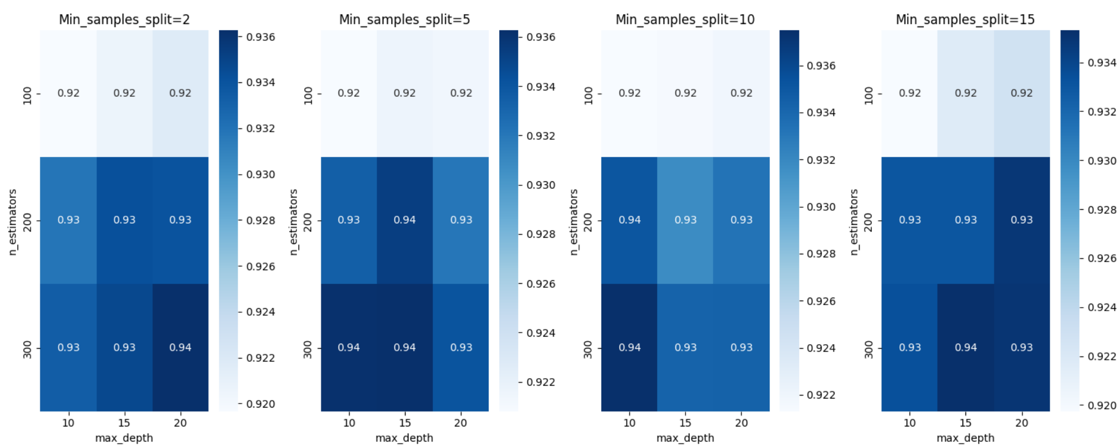
[112]:
```
# Confusion Matrix
cm_rf = confusion_matrix(y_test, y_pred_rf)

sns.heatmap(cm_rf, annot=True, fmt='d', cmap='Blues')

plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```



Visualizing the results from a GridSearchCV over a 3-parameter grid for Random Forest Classifier

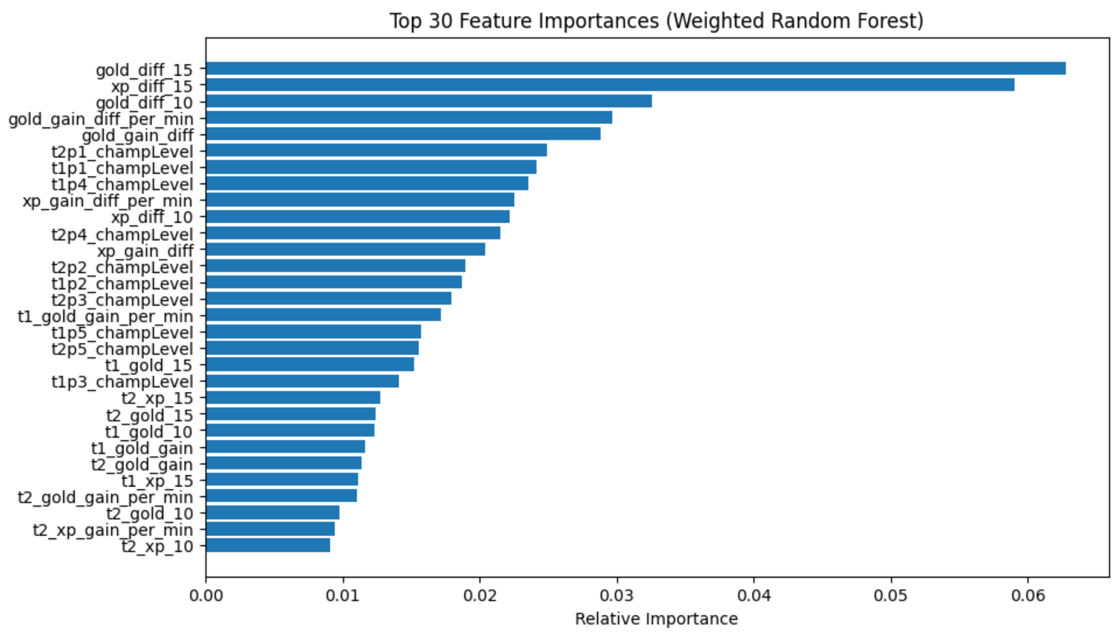
[113]:
```
# Get the results from GridSearchCV
results = grid_search.cv_results_

# Extract the mean test scores and reshape them into a 3D array
mean_test_scores = results['mean_test_score']

# Reshape mean_test_scores into a 3D matrix
scores_matrix_3d = mean_test_scores.reshape(
    len(param_grid['n_estimators']),
    len(param_grid['max_depth']),
    len(param_grid['min_samples_split'])
)

# Create subplots for each value of 'min_samples_split'
fig, axes = plt.subplots(1, len(param_grid['min_samples_split']), figsize=(15,␣
 ↪6))

for idx, min_samples_split_value in enumerate(param_grid['min_samples_split']):
```

```
    z = scores_matrix_3d[:, :, idx]  # Slice for the current 'min_samples_split' value
    sns.heatmap(z, annot=True, cmap='Blues', xticklabels=param_grid['max_depth'],
                yticklabels=param_grid['n_estimators'], ax=axes[idx])
    axes[idx].set_title(f'Min_samples_split={min_samples_split_value}')
    axes[idx].set_xlabel('max_depth')
    axes[idx].set_ylabel('n_estimators')

plt.tight_layout()
plt.show()
```



Feature importance for the Random Forrest model

[114]:
```
importances = rf_model.feature_importances_  # For the non-weighted model

# Get indices of top 30 most important features
indices = np.argsort(importances)[::-1][:30]  # Sort and take top 30 features

# Plot the top 30 features
plt.figure(figsize=(10, 6))
plt.title("Top 30 Feature Importances (Weighted Random Forest)")
plt.barh(range(30), importances[indices], align="center")
plt.yticks(range(30), X_train.columns[indices])  # Using the column names for the top 30
plt.xlabel("Relative Importance")
plt.gca().invert_yaxis()  # To display the highest importance at the top
plt.show()
```



11 Analysis of Results

Logistic Regression worked best at .98 F1 and saw little inprovement from our tuning of the class_weight. Logistic Regression also balanced both precision and recall very well, showing that the model is effective at making its prediction. Random Forrest also performed well at .94 F1 after tuning using GridSearchCV. Random Forrest also helped us to find the most impactful features which was gold_diff_15 and xp_diff_15. We also found that our Feature Selection and Engineering was well done. We were able to allow the model to better generalize and prevent overfitting with our EDA and Data Pre-processing. Overall our approach to the data, task, and models were appropriate and effective.

12 Conclusion

In conclusion, Logistic Regression was a great pick for the task. The task we chose("To predict whether Team 1 will win a game or not") in general lined up very well with our dataset as well. We also learned alot about model and data validation along the way.

13 Next Steps / Recommendation

Our recommened next steps would be to attempt to reduce the features even more to allow the model to be deployed. We still have approximately 300 features so reducing that even more could help the model to run more efficiently when in production. Ofcourse, it works very well now so it could also go into production as is. If the model did have more features dropped it would need to be retested to ensure reliablity and accuracy are maintained before release. We could also collect more data using the Riot Gaming api, to allow the model to train on much more data. The api does require approval from Riot Gaming to use, so that would also be a hurdle to overcome.

14 Extra: Validation Logistic Regression is correctly built using a "known dataset".

[124]:
```
columns = ["age",
 ↪"sex","chestpain","bp","cholestorel","sugar","ecg","heartrate","angina","oldpeak","slope","
df = pd.read_csv("../data/heart.dat",names=columns, sep=' ')

df.head()
```

[124]:
```
    age  sex  chestpain     bp  cholestorel  sugar  ecg  heartrate  angina  \
0  70.0  1.0        4.0  130.0        322.0    0.0  2.0      109.0     0.0
1  67.0  0.0        3.0  115.0        564.0    0.0  2.0      160.0     0.0
2  57.0  1.0        2.0  124.0        261.0    0.0  0.0      141.0     0.0
3  64.0  1.0        4.0  128.0        263.0    0.0  0.0      105.0     1.0
4  74.0  0.0        2.0  120.0        269.0    0.0  2.0      121.0     1.0

   oldpeak  slope  vessels  thal  presence
0      2.4    2.0      3.0   3.0         2
1      1.6    2.0      0.0   7.0         1
2      0.3    1.0      0.0   7.0         2
3      0.2    2.0      1.0   7.0         1
4      0.2    1.0      1.0   3.0         1
```

[125]:
```
df.isnull().sum()

df = df.dropna()
```

[126]:
```
dummy_list = ['chestpain','slope', 'ecg', 'thal']
df = pd.get_dummies(df, columns=dummy_list,
 ↪prefix=['chestpain','slope','ecg','thal'], prefix_sep='-')
df.head()
```

[126]:
```
    age  sex     bp  cholestorel  sugar  heartrate  angina  oldpeak  vessels  \
0  70.0  1.0  130.0        322.0    0.0      109.0     0.0      2.4      3.0
1  67.0  0.0  115.0        564.0    0.0      160.0     0.0      1.6      0.0
2  57.0  1.0  124.0        261.0    0.0      141.0     0.0      0.3      0.0
3  64.0  1.0  128.0        263.0    0.0      105.0     1.0      0.2      1.0
4  74.0  0.0  120.0        269.0    0.0      121.0     1.0      0.2      1.0

   presence  …  chestpain-4.0  slope-1.0  slope-2.0  slope-3.0  ecg-0.0  \
0         2  …           True      False       True      False    False
1         1  …          False      False       True      False    False
2         2  …          False       True      False      False     True
3         1  …           True      False       True      False     True
4         1  …          False       True      False      False    False
```

```
   ecg-1.0  ecg-2.0  thal-3.0  thal-6.0  thal-7.0
0    False     True      True     False     False
1    False     True     False     False      True
2    False    False     False     False      True
3    False    False     False     False      True
4    False     True      True     False     False

[5 rows x 23 columns]
```

[127]:
```
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2,␣
↪random_state=1)
```

[128]:
```
#scale the features
scalar = StandardScaler()
X_train_scaled = scalar.fit_transform(X_train)
X_test_scaled = scalar.transform(X_test)
```

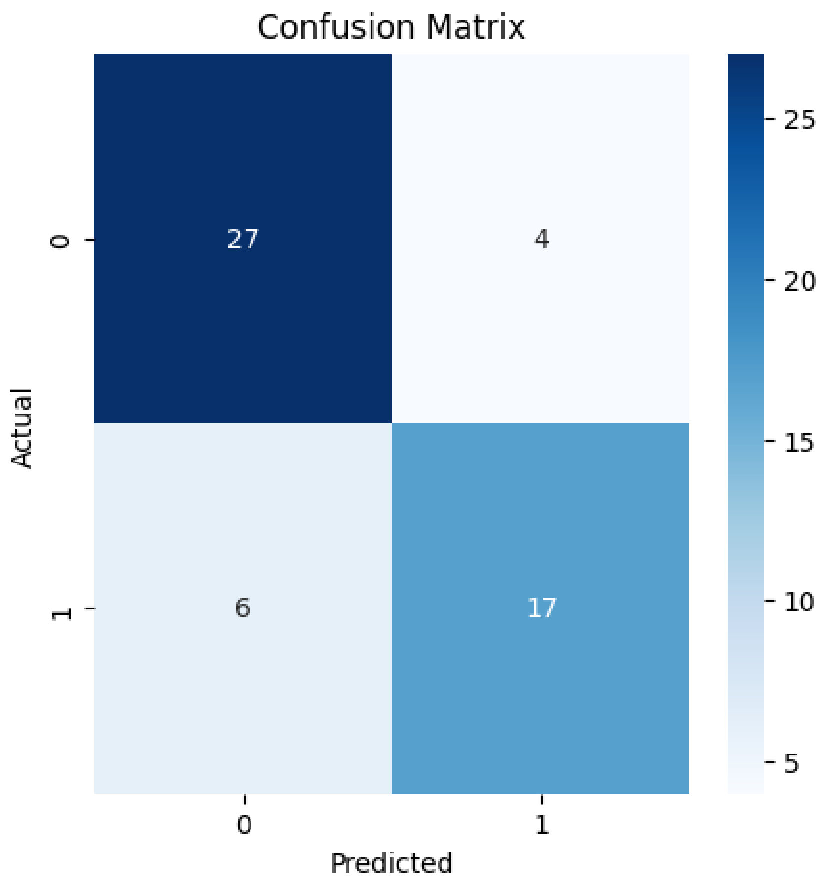
[129]:
```
y = df.presence.values
# Drop 'presence' column from data frame,
df.drop(columns=['presence'], inplace=True)
# Assign df values to x
x = df.values
```

[130]:
```
lr_model_known_data = LogisticRegression(penalty=None)

lr_model_known_data.fit(X_train_scaled, y_train)

y_pred_known_data = lr_model_known_data.predict(X_test_scaled)
```

[131]:
```
accuracy = accuracy_score(y_test, y_pred_known_data)
report = classification_report(y_test, y_pred_known_data)

print(f"Test Accuracy: {accuracy:.4f}")
print("Classification Report:\n", report)
```

```
Test Accuracy: 0.8148
Classification Report:
               precision    recall  f1-score   support

           1       0.82      0.87      0.84        31
           2       0.81      0.74      0.77        23

    accuracy                           0.81        54
   macro avg       0.81      0.81      0.81        54
weighted avg       0.81      0.81      0.81        54
```

[132]:
```
cm = confusion_matrix(y_test, y_pred_known_data)
plt.figure(figsize=(5,5))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
```



The result confirms our model is working correctly and is a high performing model for the League of Legends dataset!