League of Legends Early Game Analysis & Prediction with ML Classifiers

League of Legends is a famous online game where 10 players are divided into 2 teams(Blue and Red), each consisting of 5 players. Generally, 5 players assume different roles, one of which is "Jungler"(JG). Game objective is to destroy the opponent"s Nexus to claim victory. Throughout the game, players need to gain an advantage by killing enemy players and contesting to access various resources.

This dataset contains the first 10 min of the game and players have roughly the same level in high ELO games(DIAMOND I to MASTER).

Dataset:[https://www.kaggle.com/datasets/bobbyscience/league-of-legends-diamond-ranked-games-10-min/data (https://www.kaggle.com/datasets/bobbyscience/league-of-legends-diamond-ranked-games-10-min/data)](https://www.kaggle.com/datasets/bobbyscience/league-of-legends-diamond-ranked-games-10-min/data)

※***The analyzed content is simply based on my preferences! :P***

Import Libraries

In [43]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import ttest_ind
from sklearn.model_selection import train_test_split, cross_validate
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, plot_roc_cur
from sklearn.linear_model import LogisticRegression
from sklearn import tree
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
import warnings
warnings.filterwarnings("ignore")
```

Importing Data and Preprocessing

Importing and Overviewing Data

In [44]:
```
data= pd.read_csv("LOLstats.csv")
data.head()
```

Out[44]:

	gameId	blueWins	blueWardsPlaced	blueWardsDestroyed	blueFirstBlood	blueKills	blueDeaths	blueAssists	blu
0	4519157822	0	28	2	1	9	6	11	
1	4523371949	0	12	1	0	5	5	5	
2	4521474530	0	15	0	0	7	11	4	
3	4524384067	0	43	1	0	4	5	5	
4	4436033771	0	75	4	0	6	6	6	

5 rows × 40 columns

In [45]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9879 entries, 0 to 9878
Data columns (total 40 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   gameId                        9879 non-null   int64  
 1   blueWins                      9879 non-null   int64  
 2   blueWardsPlaced               9879 non-null   int64  
 3   blueWardsDestroyed            9879 non-null   int64  
 4   blueFirstBlood                9879 non-null   int64  
 5   blueKills                     9879 non-null   int64  
 6   blueDeaths                    9879 non-null   int64  
 7   blueAssists                   9879 non-null   int64  
 8   blueEliteMonsters             9879 non-null   int64  
 9   blueDragons                   9879 non-null   int64  
 10  blueHeralds                   9879 non-null   int64  
 11  blueTowersDestroyed           9879 non-null   int64  
 12  blueTotalGold                 9879 non-null   int64  
 13  blueAvgLevel                  9879 non-null   float64
 14  blueTotalExperience           9879 non-null   int64  
 15  blueTotalMinionsKilled        9879 non-null   int64  
 16  blueTotalJungleMinionsKilled  9879 non-null   int64  
 17  blueGoldDiff                  9879 non-null   int64  
 18  blueExperienceDiff            9879 non-null   int64  
 19  blueCSPerMin                  9879 non-null   float64
 20  blueGoldPerMin                9879 non-null   float64
 21  redWardsPlaced                9879 non-null   int64  
 22  redWardsDestroyed             9879 non-null   int64  
 23  redFirstBlood                 9879 non-null   int64  
 24  redKills                      9879 non-null   int64  
 25  redDeaths                     9879 non-null   int64  
 26  redAssists                    9879 non-null   int64  
 27  redEliteMonsters              9879 non-null   int64  
 28  redDragons                    9879 non-null   int64  
 29  redHeralds                    9879 non-null   int64  
 30  redTowersDestroyed            9879 non-null   int64  
 31  redTotalGold                  9879 non-null   int64  
 32  redAvgLevel                   9879 non-null   float64
 33  redTotalExperience            9879 non-null   int64  
 34  redTotalMinionsKilled         9879 non-null   int64  
 35  redTotalJungleMinionsKilled   9879 non-null   int64  
 36  redGoldDiff                   9879 non-null   int64  
 37  redExperienceDiff             9879 non-null   int64  
 38  redCSPerMin                   9879 non-null   float64
 39  redGoldPerMin                 9879 non-null   float64
dtypes: float64(6), int64(34)
memory usage: 3.0 MB
```

In [46]:
```
data.isna().sum()
```

Out[46]:
```
gameId                          0
blueWins                        0
blueWardsPlaced                 0
blueWardsDestroyed              0
blueFirstBlood                  0
blueKills                       0
blueDeaths                      0
blueAssists                     0
blueEliteMonsters               0
blueDragons                     0
blueHeralds                     0
blueTowersDestroyed             0
blueTotalGold                   0
blueAvgLevel                    0
blueTotalExperience             0
blueTotalMinionsKilled          0
blueTotalJungleMinionsKilled    0
blueGoldDiff                    0
blueExperienceDiff              0
blueCSPerMin                    0
blueGoldPerMin                  0
redWardsPlaced                  0
redWardsDestroyed               0
redFirstBlood                   0
redKills                        0
redDeaths                       0
redAssists                      0
redEliteMonsters                0
redDragons                      0
redHeralds                      0
redTowersDestroyed              0
redTotalGold                    0
redAvgLevel                     0
redTotalExperience              0
redTotalMinionsKilled           0
redTotalJungleMinionsKilled     0
redGoldDiff                     0
redExperienceDiff               0
redCSPerMin                     0
redGoldPerMin                   0
dtype: int64
```

In [47]:
```
print("Duplicates : ", len(data[data.duplicated()]))
```

```
Duplicates :  0
```

There is no N/A or duplicated value.

Removing Unnecessary Data and Renaming Columns

In [48]:
```
data.drop(["gameId"], axis=1, inplace= True)
data.drop(["blueTotalGold","blueTotalExperience","blueEliteMonsters","blueTotalMinionsKilled","re
for col in data.columns:
    data.rename(columns={col: col.replace("blue", "B").replace("red", "R").replace("Experience",
data.columns
```

Out[48]:
```
Index(['BWins', 'BWardsPlaced', 'BWardsDestroyed', 'BFirstBlood', 'BKills',
       'BDeaths', 'BAssists', 'BDragons', 'BHeralds', 'BTowersDestroyed',
       'BAvgLevel', 'BTotalJungleMinionsKilled', 'BGoldDiff', 'BExpDiff',
       'BCSPerMin', 'BGoldPerMin', 'RWardsPlaced', 'RWardsDestroyed',
       'RFirstBlood', 'RKills', 'RDeaths', 'RAssists', 'RDragons', 'RHeralds',
       'RTowersDestroyed', 'RAvgLevel', 'RTotalJungleMinionsKilled',
       'RGoldDiff', 'RExpDiff', 'RCSPerMin', 'RGoldPerMin'],
      dtype='object')
```

Data Distribution

In [49]:
```
data.describe()
```

Out[49]:

	BWins	BWardsPlaced	BWardsDestroyed	BFirstBlood	BKills	BDeaths	BAssists	BDragons
count	9879.000000	9879.000000	9879.000000	9879.000000	9879.000000	9879.000000	9879.000000	9879.000000
mean	0.499038	22.288288	2.824881	0.504808	6.183925	6.137666	6.645106	0.361980
std	0.500024	18.019177	2.174998	0.500002	3.011028	2.933818	4.064520	0.480597
min	0.000000	5.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	14.000000	1.000000	0.000000	4.000000	4.000000	4.000000	0.000000
50%	0.000000	16.000000	3.000000	1.000000	6.000000	6.000000	6.000000	0.000000
75%	1.000000	20.000000	4.000000	1.000000	8.000000	8.000000	9.000000	1.000000
max	1.000000	250.000000	27.000000	1.000000	22.000000	22.000000	29.000000	1.000000

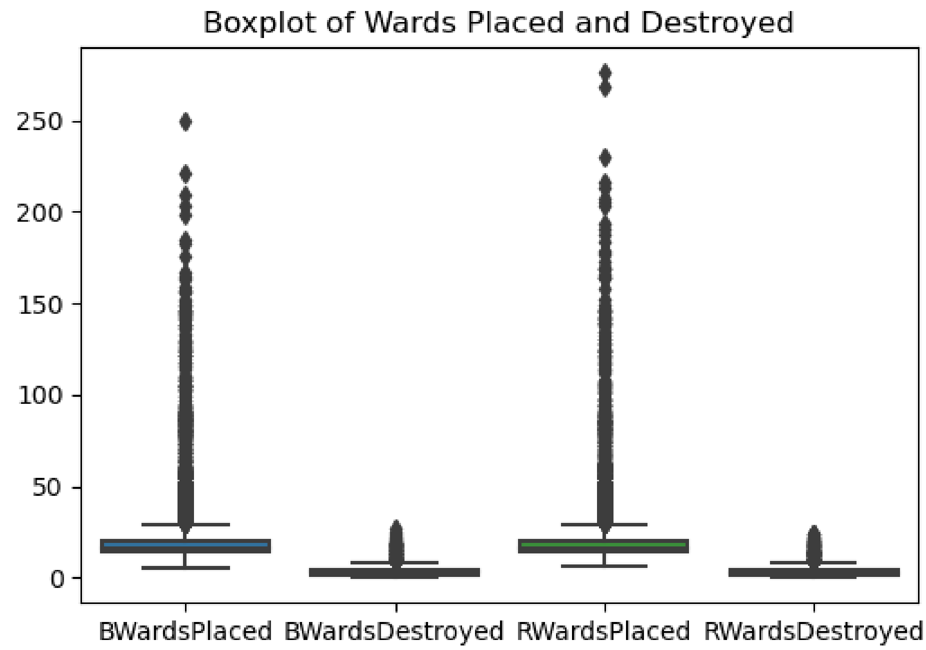
8 rows × 31 columns

Removing Outliers

From the describe() function, we can observe that the total num of wards placed on both sides appears to be significantly higher than expected.

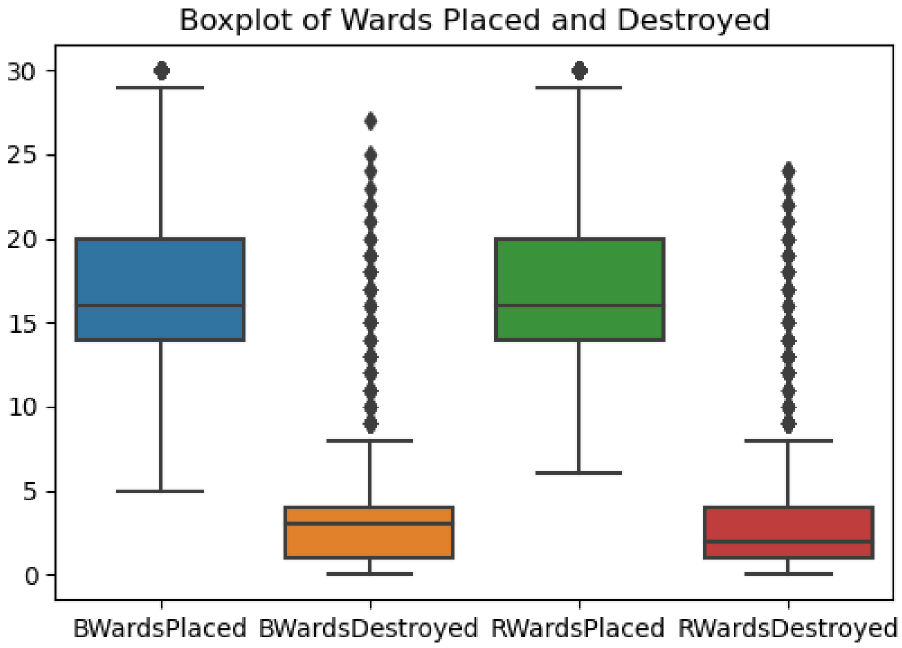
We assume that all players purchase the initial ward item (each ward has a 2-min CD).If each player places a ward at around 1 min into the game, the maximum num of wards a team can place in 10 mins is approx. 20. Assuming all players are proactive in ward placement, the maximum num of wards placed by a team in 10 mins should not exceed 30."

In [50]:
```
Outliers_Col= ["BWardsPlaced", "BWardsDestroyed", "RWardsPlaced", "RWardsDestroyed"]

plt.figure(figsize=(6,4))
ax= sns.boxplot(data= data[Outliers_Col])
ax.set(title="Boxplot of Wards Placed and Destroyed")
plt.show()
```



Data does have some non-logical outliers

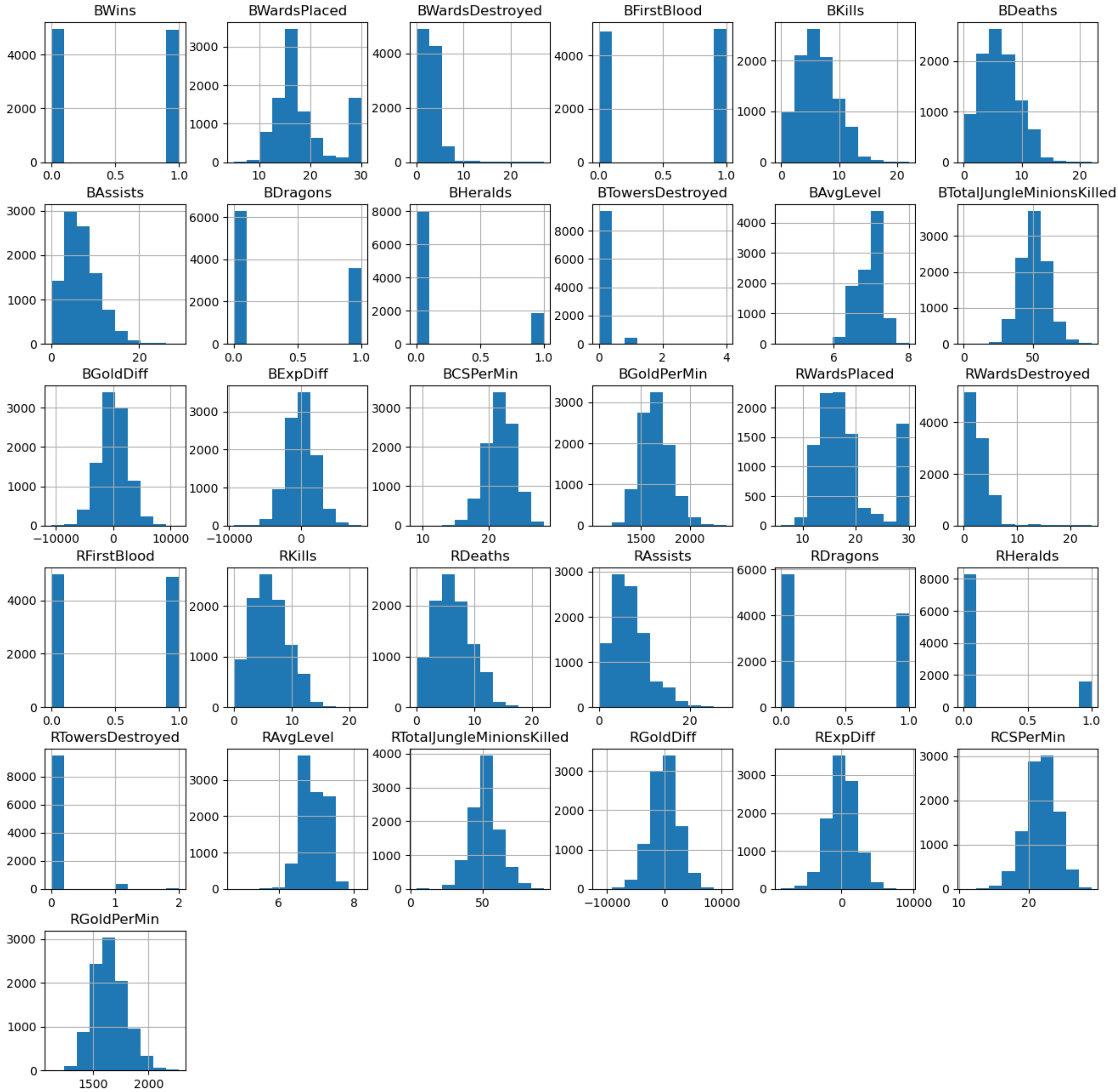
In [52]:
```
lower_l = 0
upper_l = 30
#Use np.clip to reset outlier to upperlimit
for col in Outliers_Col:
    data[col]= np.clip(data[col], lower_l, upper_l)

plt.figure(figsize= (6,4))
ax= sns.boxplot(data= data[Outliers_Col])
ax.set(title= "Boxplot of Wards Placed and Destroyed")
plt.show()
```



Exploratory Data Analysis

Visualizing Data Distributions

In [53]:
```
data.hist(figsize= (16,16))
plt.show()
```



Correlation Among Various Columns

In [54]:
```
data.drop("BWins",axis= 1, inplace= False).corr()
```

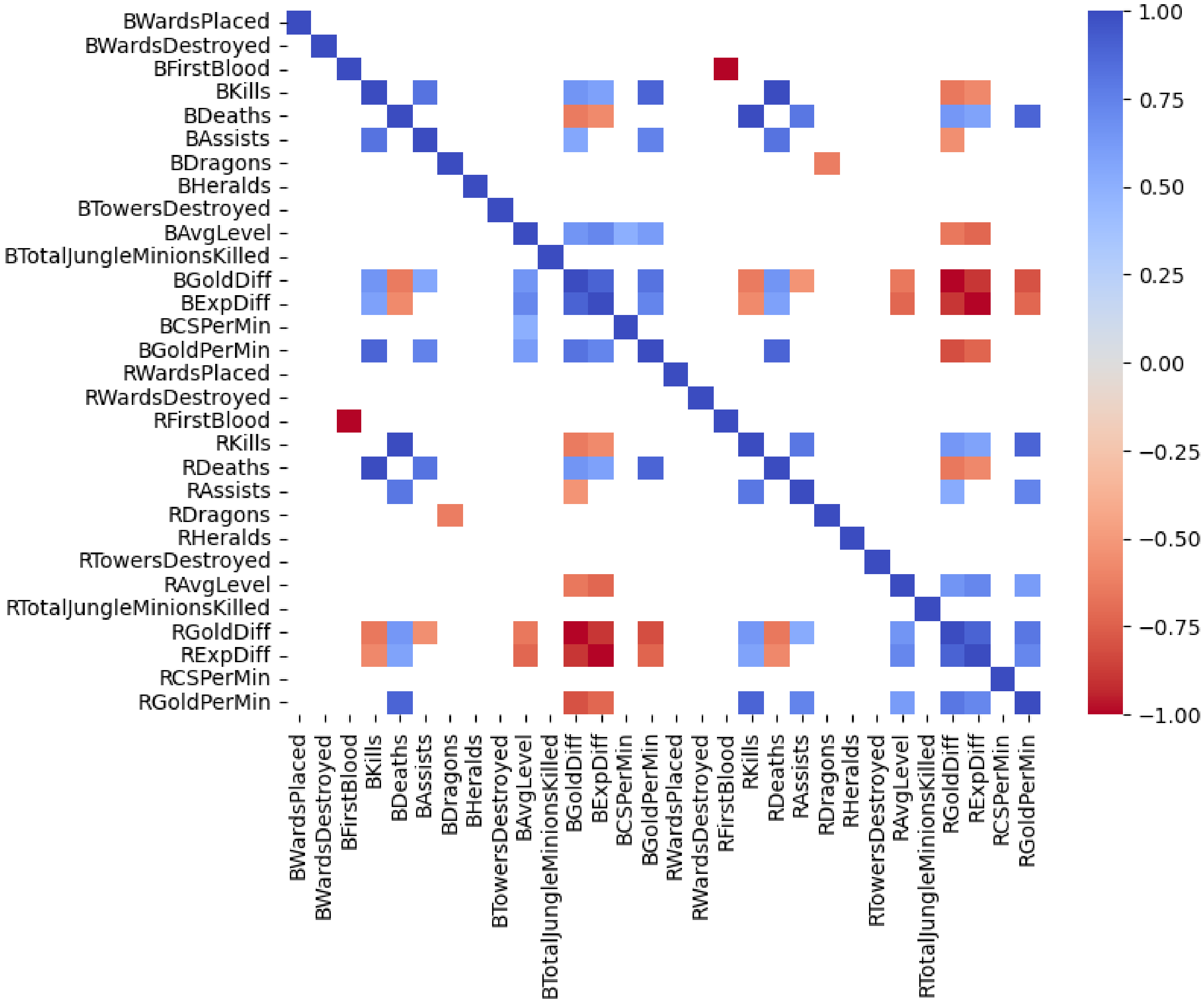
Out[54]:

	BWardsPlaced	BWardsDestroyed	BFirstBlood	BKills	BDeaths	BAssists	BDragons	BH
BWardsPlaced	1.000000	0.080709	0.015953	0.032652	-0.042695	0.066087	0.026716	0.0
BWardsDestroyed	0.080709	1.000000	0.017717	0.033748	-0.073182	0.067793	0.040504	0.0
BFirstBlood	0.015953	0.017717	1.000000	0.269425	-0.247929	0.229485	0.134309	0.0
BKills	0.032652	0.033748	0.269425	1.000000	0.004044	0.813667	0.170436	0.0
BDeaths	-0.042695	-0.073182	-0.247929	0.004044	1.000000	-0.026372	-0.188852	-0.0
BAssists	0.066087	0.067793	0.229485	0.813667	-0.026372	1.000000	0.170873	0.0
BDragons	0.026716	0.040504	0.134309	0.170436	-0.188852	0.170873	1.000000	0.0
BHeralds	0.018975	0.016940	0.077509	0.076195	-0.095527	0.028434	0.020381	1.0
BTowersDestroyed	-0.003873	-0.009150	0.083316	0.180314	-0.071441	0.123663	0.039750	0.2
BAvgLevel	0.060934	0.060294	0.177617	0.434867	-0.414755	0.292661	0.160683	0.1
BTotalJungleMinionsKilled	0.025711	-0.023452	0.018190	-0.112506	-0.228102	-0.134023	0.159595	0.1
BGoldDiff	0.047861	0.078585	0.378511	0.654148	-0.640000	0.549761	0.233875	0.1
BExpDiff	0.068422	0.077946	0.240665	0.583730	-0.577613	0.437002	0.211496	0.1
BCSPerMin	-0.018427	0.111028	0.125642	-0.030880	-0.468560	-0.062035	0.086686	0.0
BGoldPerMin	0.037676	0.060054	0.312058	0.888751	-0.162572	0.748352	0.186413	0.1
RWardsPlaced	-0.012547	0.265523	-0.030261	-0.078182	0.015353	-0.058399	-0.031033	0.0
RWardsDestroyed	0.251259	0.123919	-0.043304	-0.092278	0.038672	-0.064501	-0.023049	0.0
RFirstBlood	-0.015953	-0.017717	-1.000000	-0.269425	0.247929	-0.229485	-0.134309	-0.0
RKills	-0.042695	-0.073182	-0.247929	0.004044	1.000000	-0.026372	-0.188852	-0.0
RDeaths	0.032652	0.033748	0.269425	1.000000	0.004044	0.813667	0.170436	0.0
RAssists	-0.031501	-0.046212	-0.201140	-0.020344	0.804023	-0.007481	-0.162406	-0.0
RDragons	-0.036875	-0.034439	-0.135327	-0.207949	0.150746	-0.189563	-0.631930	0.0
RHeralds	-0.014977	-0.012712	-0.060246	-0.104423	0.076639	-0.058074	-0.016827	-0.2
RTowersDestroyed	-0.024194	-0.023943	-0.069584	-0.082491	0.156780	-0.060880	-0.032865	-0.0
RAvgLevel	-0.043241	-0.059090	-0.182602	-0.412219	0.433383	-0.356928	-0.149806	-0.0
RTotalJungleMinionsKilled	-0.016562	-0.035732	-0.024559	-0.214454	-0.100271	-0.160915	-0.098446	-0.0
RGoldDiff	-0.047861	-0.078585	-0.378511	-0.654148	0.640000	-0.549761	-0.233875	-0.1
RExpDiff	-0.068422	-0.077946	-0.240665	-0.583730	0.577613	-0.437002	-0.211496	-0.1
RCSPerMin	0.003263	0.040023	-0.156711	-0.472203	-0.040521	-0.337515	-0.059803	-0.0
RGoldPerMin	-0.039955	-0.067467	-0.301479	-0.161127	0.885728	-0.133948	-0.192871	-0.1

30 rows × 30 columns

Here we only show the high(>0.5) and low(<-0.5) corr graph.

In [55]:
```
plt.figure(figsize= (8, 6))
corr= data.drop("BWins", axis= 1).corr()
filtered_corr= corr[(corr> 0.5) | (corr< -0.5)]
sns.heatmap(filtered_corr, cmap= "coolwarm_r")
plt.show()
```

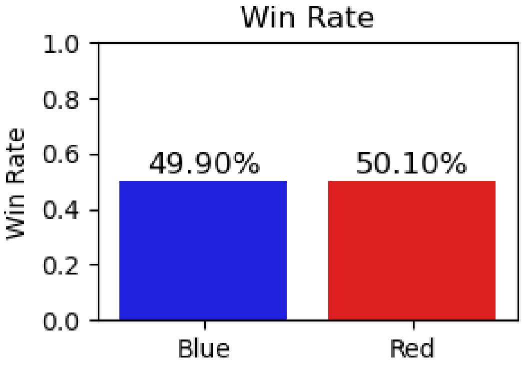


Win Rate

In [56]:
```
#Win Rate of 2 sides
B_wins_count= data["BWins"].sum()
R_wins_count= len(data)- B_wins_count
B_WR= (B_wins_count / len(data))
R_WR= 1 - B_WR
print(B_WR,R_WR)
```

```
0.4990383642069035 0.5009616357930965
```

In [57]:
```
plt.figure(figsize=(3,2))
barplot = sns.barplot(x= ["Blue", "Red"], y=[B_WR, R_WR], palette=["Blue", "Red"])
for index, value in enumerate([B_WR, R_WR]):
    barplot.text(index, value + 0.01, f"{value:.2%}", ha="center", va="bottom", fontsize=12)
plt.title("Win Rate")
plt.ylabel("Win Rate")
plt.ylim(0,1)
plt.show()
```

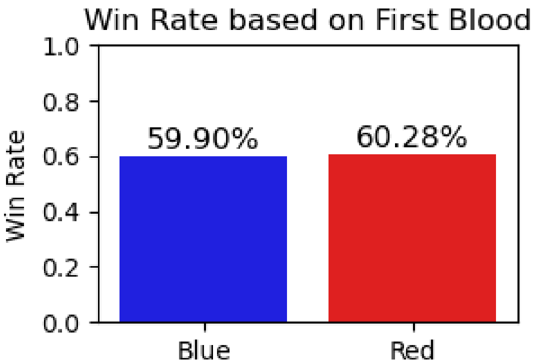


Win Rate with First Blood

In [58]:
```
B_FB_Wins= len(data[(data["BFirstBlood"]== 1) & (data["BWins"]== 1)])
B_FB= len(data[data["BFirstBlood"]== 1])
B_WR_FB= B_FB_Wins/ B_FB

R_FB_Wins= len(data[(data["RFirstBlood"]== 1) & (data["BWins"]== 0)])

R_FB= len(data[data["RFirstBlood"]== 1])
R_WR_FB= R_FB_Wins/ R_FB

print(f"Blue Teams' Win Rate If Secure First Blood: {B_WR_FB:.2%}")
print(f"Red Teams' Win Rate If Secure First Blood: {R_WR_FB:.2%}")
```

```
Blue Teams' Win Rate If Secure First Blood: 59.90%
Red Teams' Win Rate If Secure First Blood: 60.28%
```

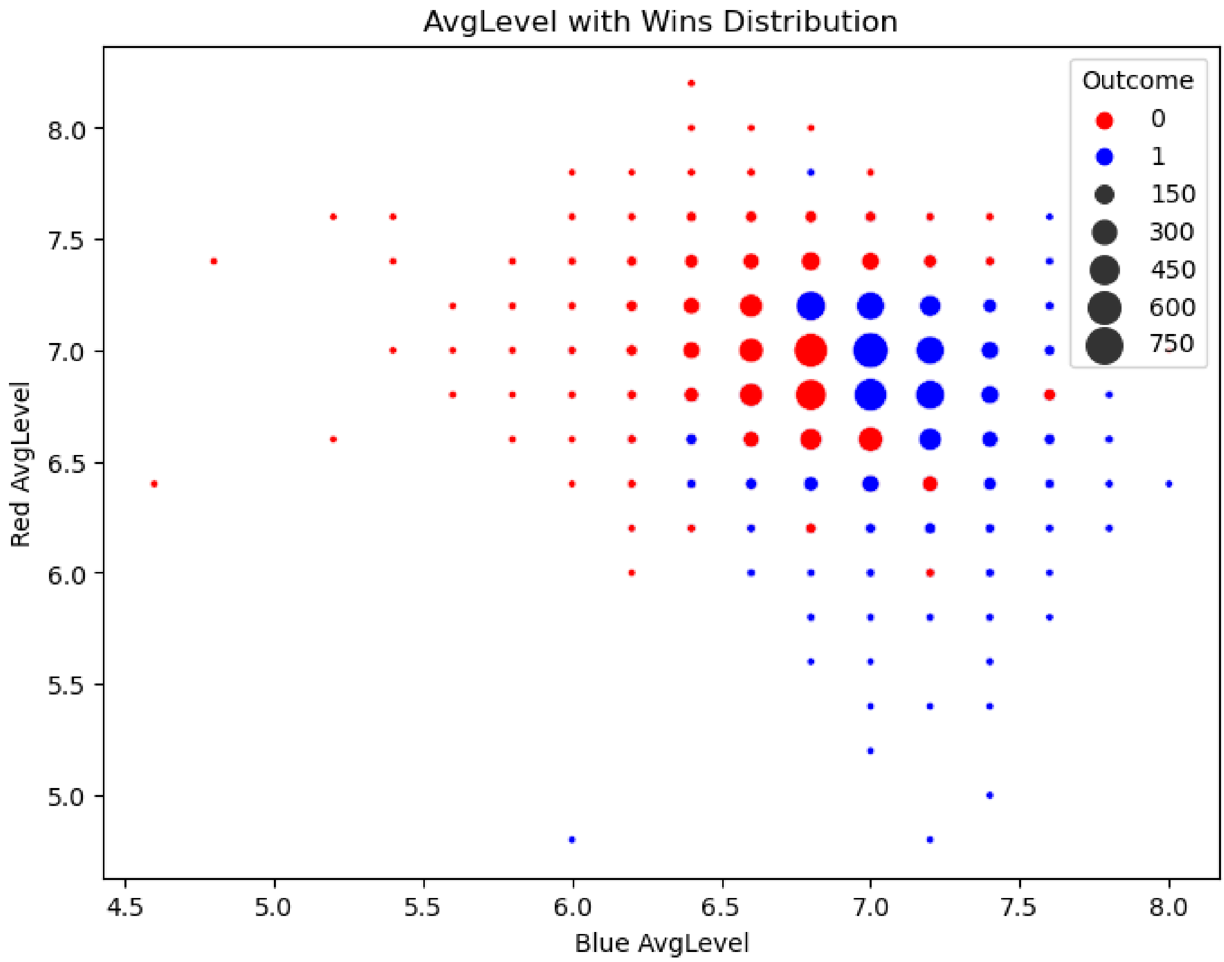
In [59]:
```
plt.figure(figsize=(3,2))
barplot = sns.barplot(x= ["Blue", "Red"], y=[B_WR_FB, R_WR_FB], palette=["Blue", "Red"])
for index, value in enumerate([B_WR_FB, R_WR_FB]):
    barplot.text(index, value + 0.01, f"{value:.2%}", ha="center", va="bottom", fontsize=12)
plt.title("Win Rate based on First Blood")
plt.ylabel("Win Rate")
plt.ylim(0,1)
plt.show()
```



If secures the FirstBlood in the first 10 min of the game, the Win Rate is approx.10% higher than the original Win Rate.

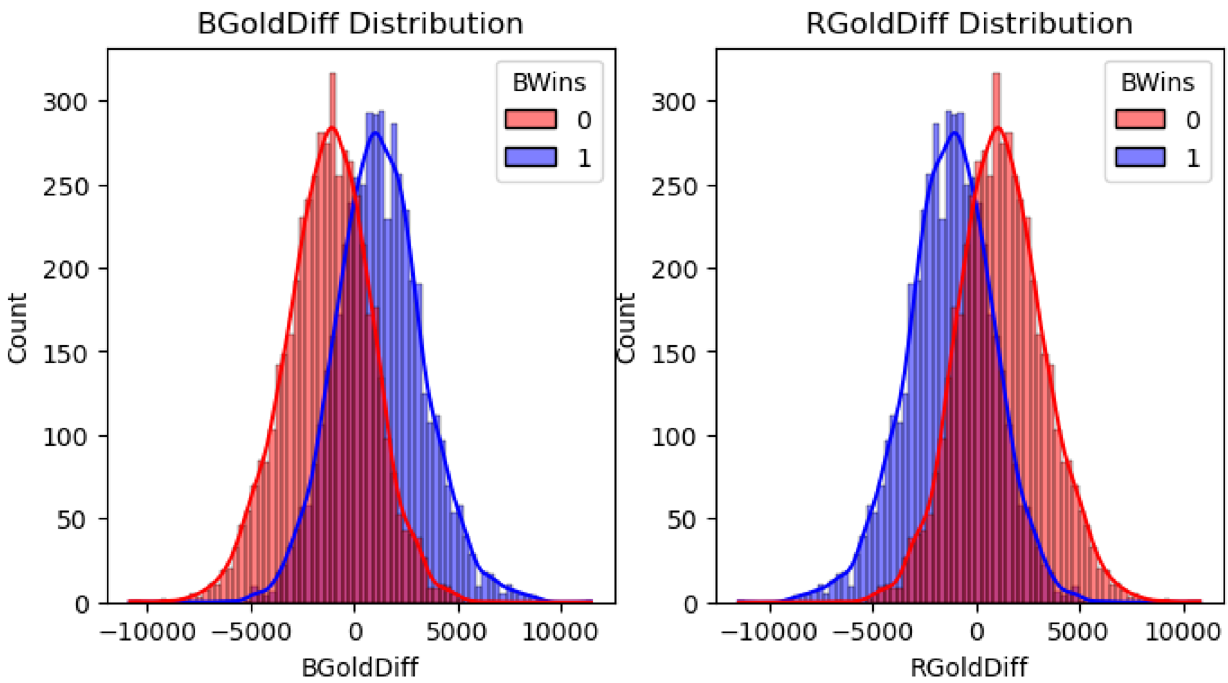
Average Level

In [63]:
```
data["BAvgLevel"]= pd.to_numeric(data["BAvgLevel"], errors= "coerce")
data["RAvgLevel"]= pd.to_numeric(data["RAvgLevel"], errors= "coerce")

lol_P ={1:"blue", 0:"red"}

plt.figure(figsize=(8, 6))
ax= sns.scatterplot(data= data, x= "BAvgLevel", y= "RAvgLevel", hue= "BWins",
                    size= data.groupby(["BAvgLevel", "RAvgLevel"])["BWins"].transform("count"),
                    sizes= (10,200), palette= lol_P)
ax.set(title= "AvgLevel with Wins Distribution",xlabel= "Blue AvgLevel", ylabel= "Red AvgLevel")

plt.legend(title= "Outcome", loc= "upper right")
plt.show()
```



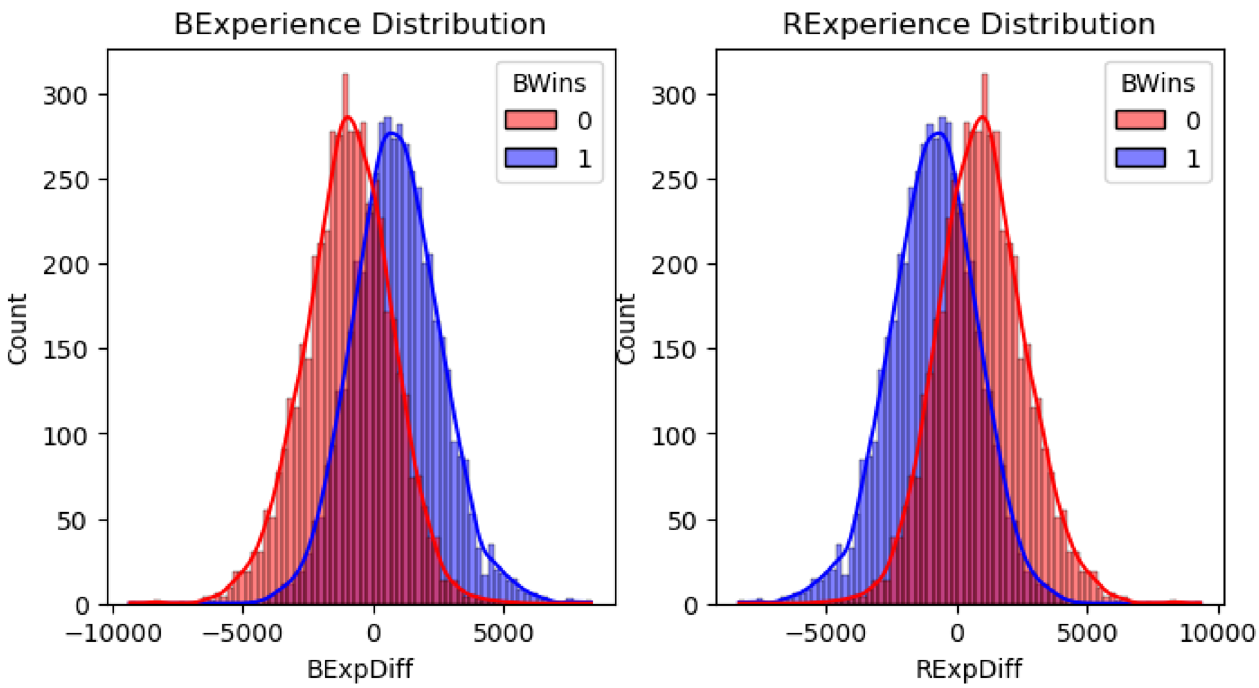
Gold Difference

In [65]:
```
fig, axes= plt.subplots(1,2 , figsize=(8, 4))
sns.histplot(x= data["BGoldDiff"],kde= True, hue= data["BWins"],palette= lol_P, ax= axes[0])
axes[0].set_title("BGoldDiff Distribution")
sns.histplot(x= data["RGoldDiff"],kde= True, hue= data["BWins"],palette= lol_P )
axes[1].set_title("RGoldDiff Distribution")
plt.show()
```



Experience

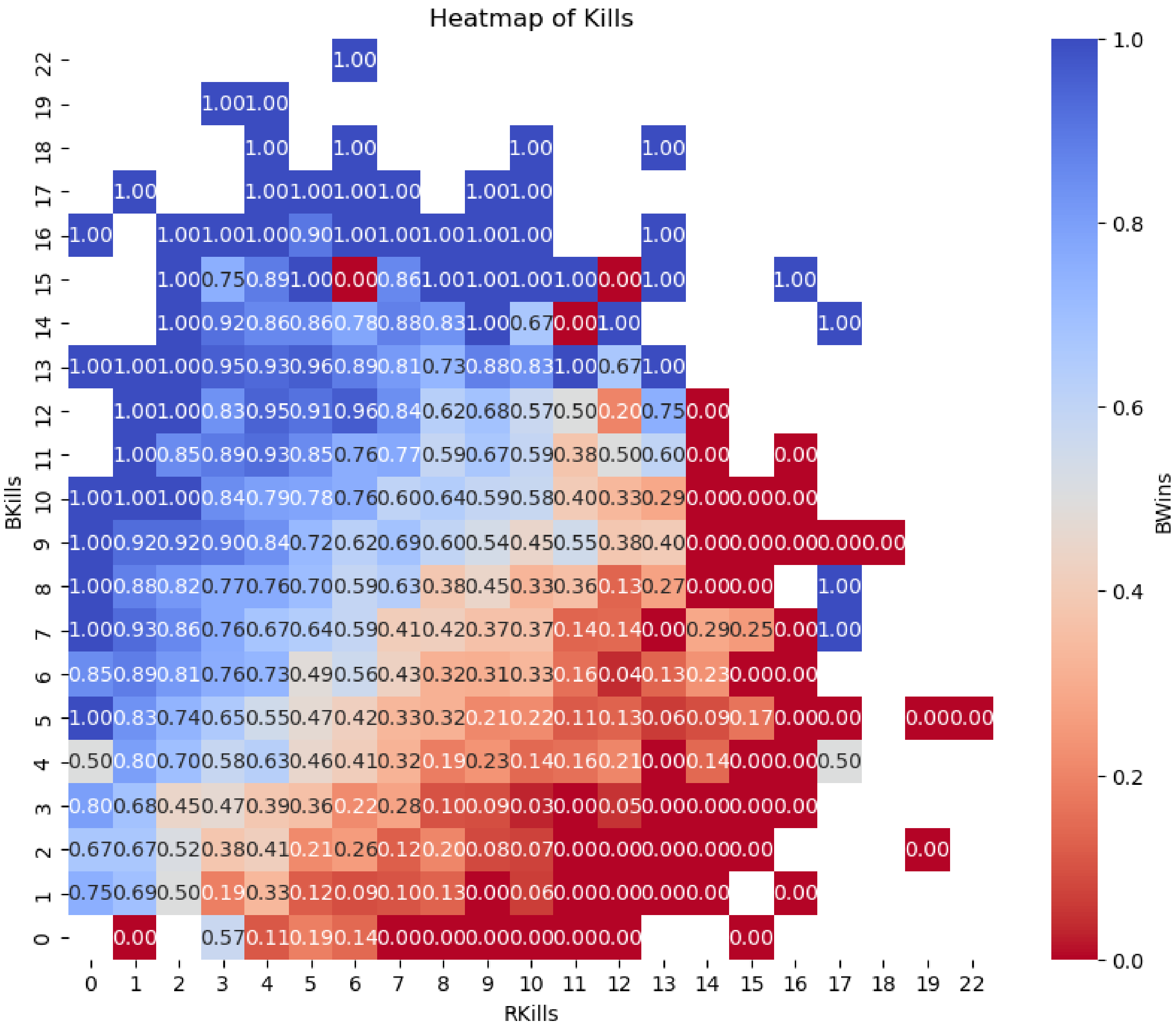
In [66]:
```
fig, axes= plt.subplots(1,2 , figsize= (8, 4))
sns.histplot(x= data["BExpDiff"],kde= True, hue= data["BWins"],palette= lol_P, ax= axes[0])
axes[0].set_title("BExperience Distribution")
sns.histplot(x= data["RExpDiff"],kde= True, hue= data["BWins"],palette= lol_P, )
axes[1].set_title("RExperience Distribution")
plt.show()
```



KDA Analysis - Kills

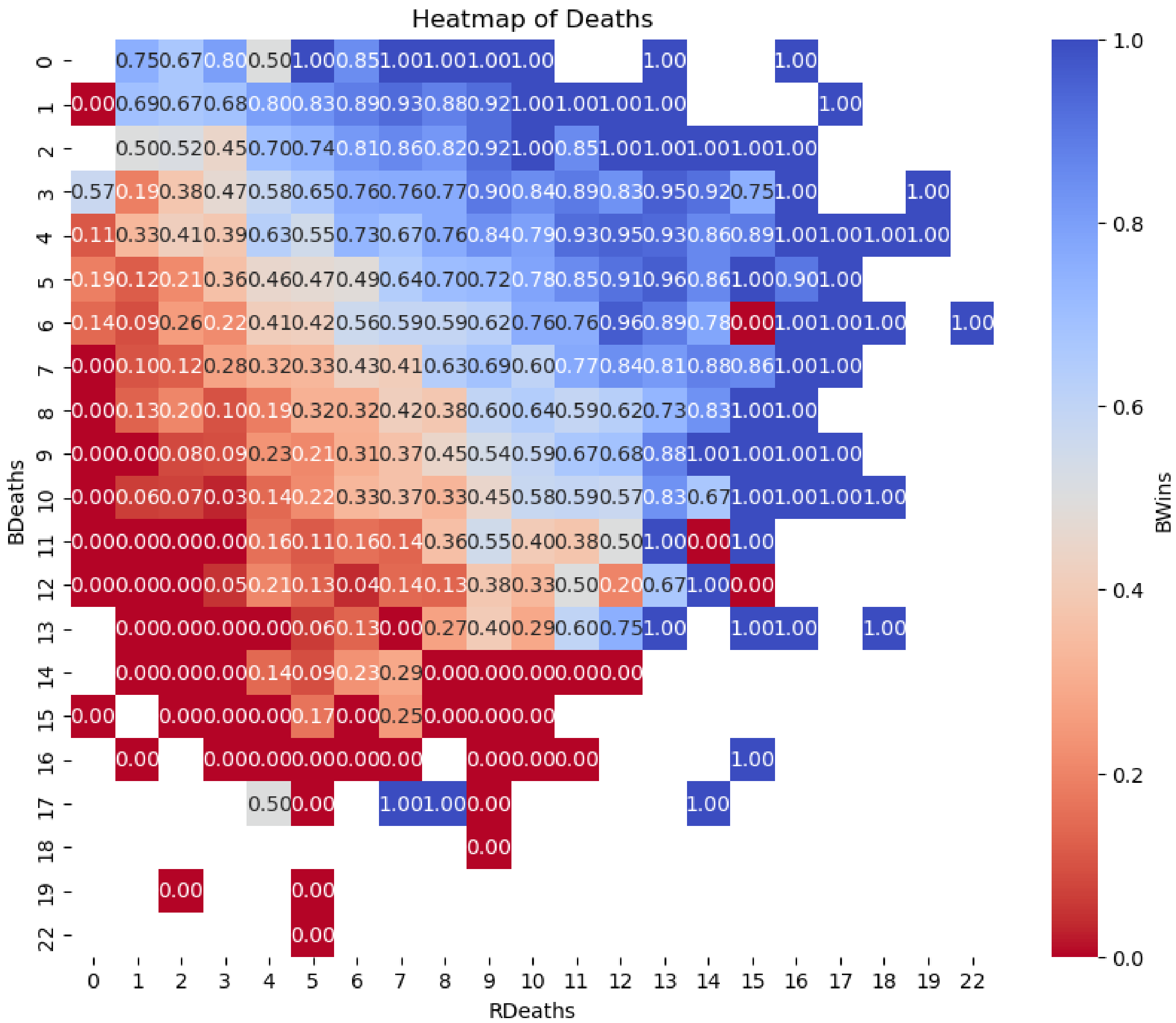
※Please noted that 1 =Blue wins, 0 = Red wins.

In [67]:
```
plt.figure(figsize= (10,8))
ax= sns.heatmap(data.pivot_table(index= "BKills", columns= "RKills", values= "BWins")[::-1], cmap
                fmt= ".2f", cbar_kws= {"label": "BWins"})
ax.set(title= "Heatmap of Kills", xlabel= "RKills", ylabel= "BKills")
plt.show()
```



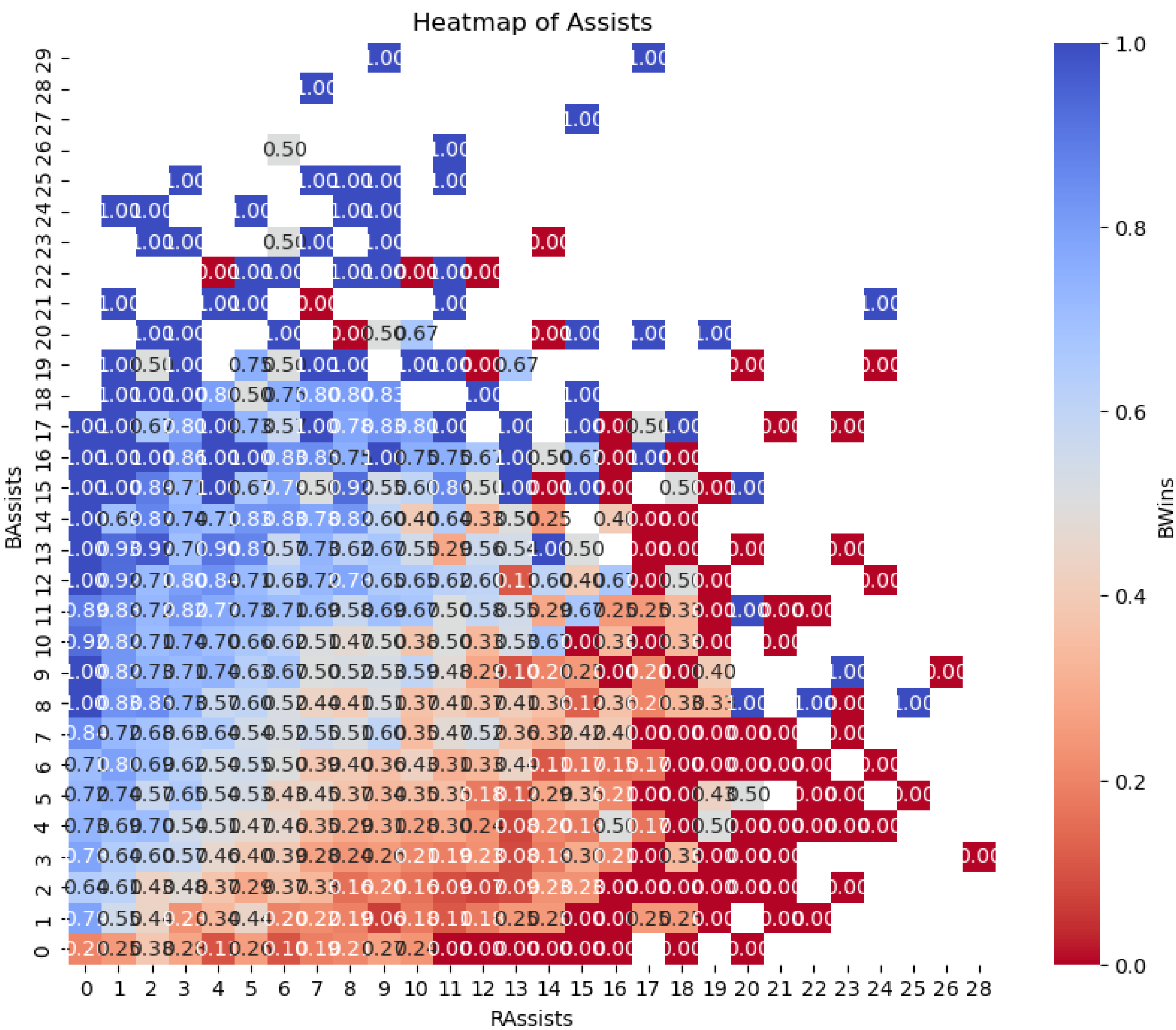
KDA Analysis - Deaths

In [68]:
```
plt.figure(figsize= (10,8))
ax= sns.heatmap(data.pivot_table(index= "BDeaths", columns= "RDeaths", values= "BWins"), cmap= "c
                fmt= ".2f", cbar_kws= {"label": "BWins"})
ax.set(title= "Heatmap of Deaths", xlabel= "RDeaths", ylabel= "BDeaths")
plt.show()
```



KDA Analysis - Assists

In [69]:
```
plt.figure(figsize= (10,8))
ax= sns.heatmap(data.pivot_table(index= "BAssists", columns= "RAssists", values= "BWins")[::-1],
                fmt= ".2f", cbar_kws= {"label": "BWins"})
ax.set(title= "Heatmap of Assists", xlabel= "RAssists", ylabel= "BAssists")
plt.show()
```



KDA Analysis

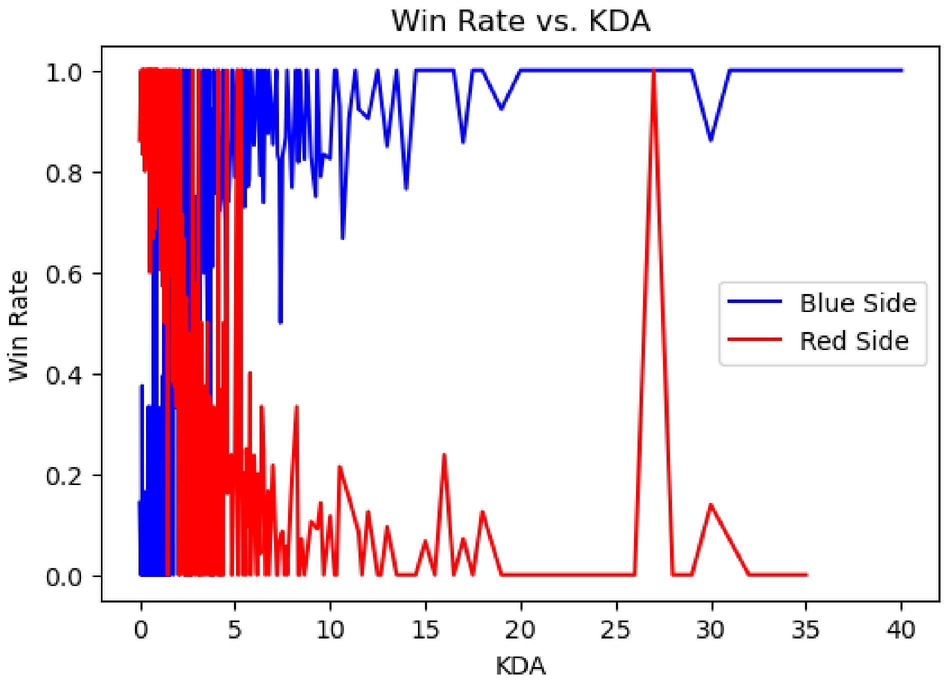
KDA =(Kills + Assists) / Deaths, its a indicator that usually used to evaluate a player"s performance in many online game.

A high KDA usually means that the player achieved more kills and reduced the number of deaths in the battle, that is more helpful in the game.

We set the upper limit to 30, lower limit to 0 to avoid inf or -inf values when dealing with data that might be zero.

In [70]:
```
data["BKDA"]= (data["BKills"] + data["BAssists"]) / data["BDeaths"]
data["RKDA"]= (data["RKills"] + data["RAssists"]) / data["RDeaths"]

# To Avoid inf or -inf output.
data.loc[np.isinf(data["BKDA"]), "BKDA"]= 30
data.loc[np.isinf(data["RKDA"]), "RKDA"]= 30
data.loc[np.isneginf(data["BKDA"]), "BKDA"]= 0
data.loc[np.isneginf(data["RKDA"]), "RKDA"]= 0
```

In [73]:
```
plt.figure(figsize= (6,4))
ax= sns.lineplot(data= data, x= "BKDA", y= "BWins", label= "Blue Side", ci= None, color= "blue")
sns.lineplot(data= data, x= "RKDA", y= "BWins", label= "Red Side", ci= None, color= "red")

ax.set(title= "Win Rate vs. KDA", xlabel= "KDA", ylabel= "Win Rate")
ax.legend()
plt.show()
```

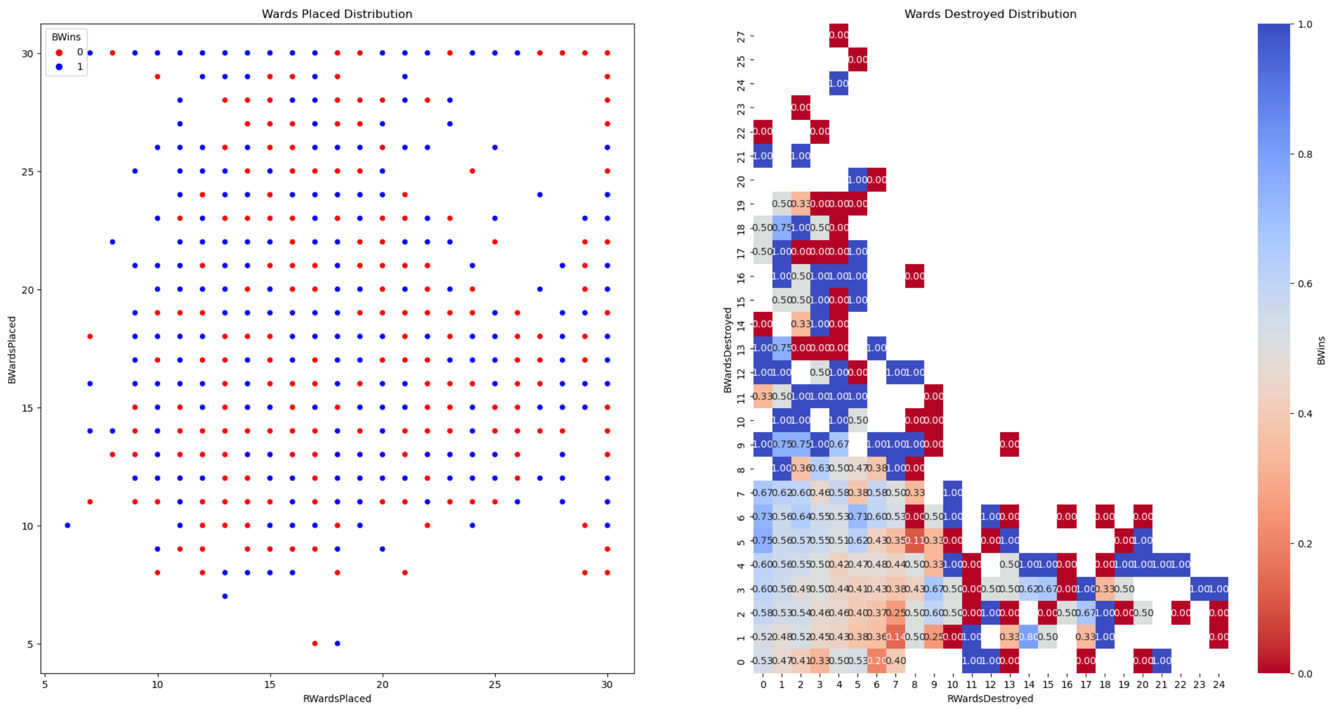


Wards Placed

Wards are items placed on the map to enhance visibility in the game. (In the games, much of the map is shrouded in the fog of war, preventing players from seeing the movements of enemies.)

Having ample vision is crucial, enabling teams to identify enemy positions, control key map locations, and anticipate enemy's actions in advance.

In [74]:
```
fig, axes= plt.subplots(1,2 , figsize= (24,12))
sns.scatterplot(data= data, x= "RWardsPlaced", y= "BWardsPlaced", hue= "BWins", palette= lol_P, a
axes[0].set_title("Wards Placed Distribution")
sns.heatmap(data.pivot_table(index= "BWardsDestroyed", columns= "RWardsDestroyed", values= "BWins
            cmap= "coolwarm_r", annot= True, fmt= ".2f", cbar_kws= {"label": "BWins"})
axes[1].set_title("Wards Destroyed Distribution")

plt.show()
```



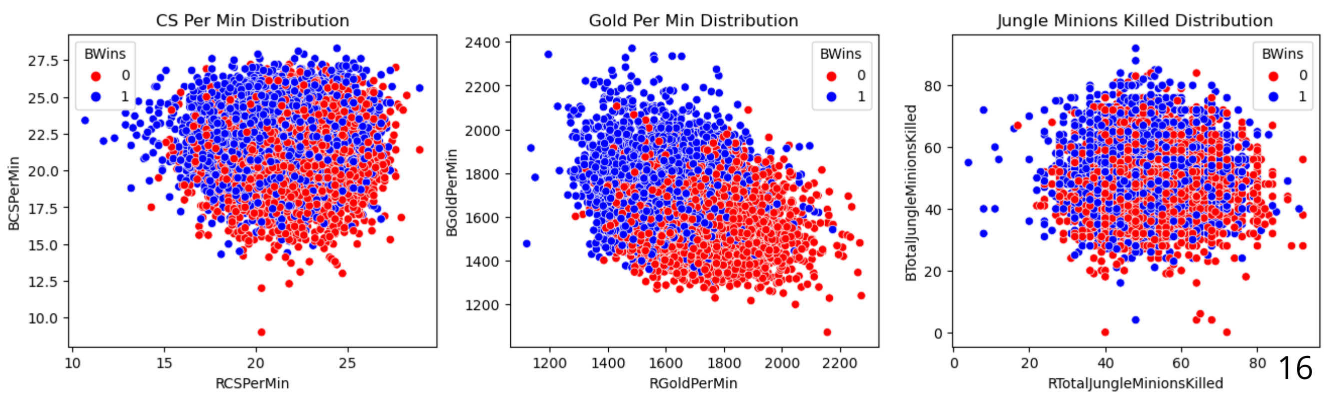
Gold & Income

In LOL, 2 teams' Nexuses periodically spawn minions, providing players with a key source of gold income. Players can strategically eliminate these minions to get advantage in the game.

1 player in the team often assumes the role of the "Jungler"(JG). In the early stages, JGs strategically navigate the map's periphery, relying not on minions but on jungle monsters as their main source of gold income.

※Here CS Per Min = Minions been killed by the team per min

In [75]:
```
fig, axes= plt.subplots(1, 3 , figsize= (16, 4))
sns.scatterplot(data= data, x= "RCSPerMin", y= "BCSPerMin", hue= "BWins", palette= lol_P, ax= axe
axes[0].set_title("CS Per Min Distribution")

sns.scatterplot(data= data, x= "RGoldPerMin", y= "BGoldPerMin", hue= "BWins", palette= lol_P, ax=
axes[1].set_title("Gold Per Min Distribution")

sns.scatterplot( data= data, x= "RTotalJungleMinionsKilled", y= "BTotalJungleMinionsKilled", hue=
axes[2].set_title("Jungle Minions Killed Distribution")

plt.show()
```



Other Objectives

In LOL, the map features elite monsters known as "Dragon" and "Herald". Killing them provides huge buffs or benefits, significantly enhancing advantages or turning the tide in a team's favor. Both sides frequently engage in contested battles to secure these valuable objectives.

Dragons spawn every 5 mins, starting from the 5th min of the game, and respawn every 5 mins after being killed.

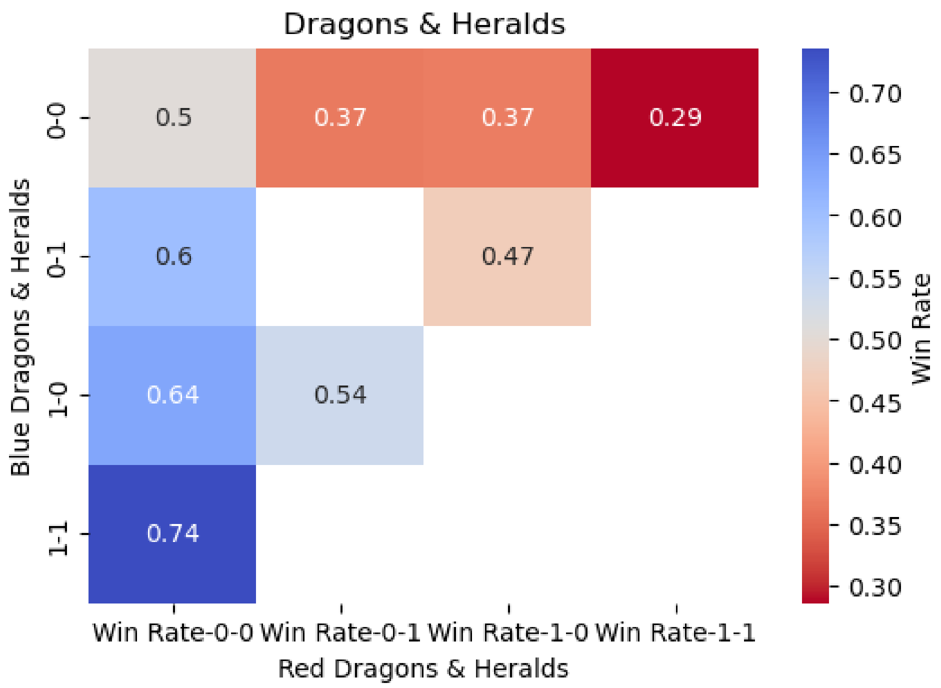
Heralds appear at the 8th min and similarly respawn every 5 mins.

Consequently, within the first 10 mins of a match, only 1 Dragon and 1 Herald can be slain.

In [76]:
```
Object_Scenario= [(b_dragon, b_herald, r_dragon, r_herald)
                  for b_dragon in [0, 1]
                  for b_herald in [0, 1]
                  for r_dragon in [0, 1]
                  for r_herald in [0, 1]]
Object_WR= []
for s in Object_Scenario:
    b_dragon, b_herald, r_dragon, r_herald= s
    condition= (data["BDragons"]== b_dragon) & (data["BHeralds"]== b_herald) & (data["RDragons"]=
    o_wr= data[condition]["BWins"].mean()
    Object_WR.append(o_wr)


Object_df= pd.DataFrame(Object_WR, index= pd.MultiIndex.from_tuples(Object_Scenario, names= ["BDr
Object_df= Object_df.sort_index(level= ["BDragons", "BHeralds", "RDragons", "RHeralds"])
print(Object_df)
```

```
                                   Win Rate
BDragons BHeralds RDragons RHeralds
0        0        0        0       0.504442
                           1       0.366883
                  1        0       0.369001
                           1       0.286301
         1        0        0       0.600592
                           1            NaN
                  1        0       0.469716
                           1            NaN
1        0        0        0       0.636677
                           1       0.535912
                  1        0            NaN
                           1            NaN
         1        0        0       0.735211
                           1            NaN
                  1        0            NaN
                           1            NaN
```

In [78]:
```
plt.figure(figsize= (6,4))
ax= sns.heatmap(Object_df.unstack(level= ["RDragons", "RHeralds"]), annot= True, cmap= "coolwarm_
ax.set(title= "Dragons & Heralds", xlabel= "Red Dragons & Heralds", ylabel= "Blue Dragons & Heral
plt.show()
```



Dominant Factors in the First 10 Minutes and Their Impact on Game Outcome

In [79]:
```
BR_WR = pd.DataFrame({"Blue's WR": B_WR_FB, "Red's WR": R_WR_FB}, index=["FirstBlood"])
print(BR_WR)
```

```
            Blue's WR  Red's WR
FirstBlood   0.598957  0.602821
```

In [80]:
```
KDA_Columns= ["Kills", "Deaths", "Assists", "KDA"]

for col in KDA_Columns:
    B_adv_col= f"B{col}"
    R_adv_col= f"R{col}"
    B_WR_Adv= data[data[B_adv_col] > data[R_adv_col]]["BWins"].mean()
    R_WR_Adv= 1 - data[data[R_adv_col] > data[B_adv_col]]["BWins"].mean()
    BR_WR.loc[col, "Blue's WR"]= B_WR_Adv
    BR_WR.loc[col, "Red's WR"]= R_WR_Adv

print(BR_WR)
```

```
            Blue's WR  Red's WR
FirstBlood   0.598957  0.602821
Kills        0.724107  0.728665
Deaths       0.271335  0.275893
Assists      0.675652  0.677349
KDA          0.700919  0.705226
```

In [81]:
```
KDA_Columns= ["WardsPlaced", "AvgLevel","CSPerMin", "GoldPerMin"]

for col in KDA_Columns:
    B_adv_col= f"B{col}"
    R_adv_col= f"R{col}"
    B_WR_Adv= data[data[B_adv_col] > data[R_adv_col]]["BWins"].mean()
    R_WR_Adv= 1 - data[data[R_adv_col] > data[B_adv_col]]["BWins"].mean()
    BR_WR.loc[col, "Blue's WR"]= B_WR_Adv
    BR_WR.loc[col, "Red's WR"]= R_WR_Adv

print(BR_WR)
```

```
             Blue's WR  Red's WR
FirstBlood    0.598957  0.602821
Kills         0.724107  0.728665
Deaths        0.271335  0.275893
Assists       0.675652  0.677349
KDA           0.700919  0.705226
WardsPlaced   0.531076  0.533454
AvgLevel      0.731775  0.726245
CSPerMin      0.629684  0.626584
GoldPerMin    0.720968  0.724832
```

In [82]:
```
Object_Columns= ["Dragons", "Heralds", "TowersDestroyed"]

for col in Object_Columns:
    B_Adv_Col= f"B{col}"
    R_Adv_Col= f"R{col}"

    B_WR_Adv= data[data[B_Adv_Col] > data[R_Adv_Col]]["BWins"].mean()
    R_WR_Adv= 1 - data[data[R_Adv_Col] > data[B_Adv_Col]]["BWins"].mean()

    BR_WR.loc[col, "Blue's WR"]= B_WR_Adv
    BR_WR.loc[col, "Red's WR"]= R_WR_Adv

print(BR_WR)
BR_corr= BR_WR["Blue's WR"].corr(BR_WR["Red's WR"])
print(f"Correlation : {BR_corr}")
```

```
                 Blue's WR  Red's WR
FirstBlood        0.598957  0.602821
Kills             0.724107  0.728665
Deaths            0.271335  0.275893
Assists           0.675652  0.677349
KDA               0.700919  0.705226
WardsPlaced       0.531076  0.533454
AvgLevel          0.731775  0.726245
CSPerMin          0.629684  0.626584
GoldPerMin        0.720968  0.724832
Dragons           0.640940  0.625827
Heralds           0.595046  0.612271
TowersDestroyed   0.759637  0.786096
Correlation : 0.9969642648617048
```

In [83]:
```
B_Top5,R_Top5= BR_WR["Blue's WR"].nlargest(5).index, BR_WR["Red's WR"].nlargest(5).index
are_equal= set(B_Top5)== set(R_Top5)

print(f"Blue's WR Highest 5 Factors:: {B_Top5}")
print(f"Red's WR Highest 5 Factors: {R_Top5}")
print(f"Same or not: {are_equal}")
```

```
Blue's WR Highest 5 Factors:: Index(['TowersDestroyed', 'AvgLevel', 'Kills', 'GoldPerMin', 'KD
A'], dtype='object')
Red's WR Highest 5 Factors: Index(['TowersDestroyed', 'Kills', 'AvgLevel', 'GoldPerMin', 'KDA'],
dtype='object')
Same or not: True
```

Once your team gets an advantage at TowersDestroyed in the first 10 min, you"re very likely to win the game (75%+ WR)

Does Being on Red side or Blue side Have a Significant Impact on Win Rate?

In [84]:
```
#T Test
T_stat, P_value= ttest_ind(BR_WR["Blue's WR"],BR_WR["Red's WR"])
#If P-Value < 0.05
alpha= 0.05
print("p_value:", P_value)
if P_value< alpha:
    print("2 sets of Win Rate data exhibit significant diff.(Reject Null Hypothesis)")
else:
    print("There's no significant diff between 2 teams's Win Rate Data.(Null hypothesis cannot be
```

```
p_value: 0.9452662043772597
There's no significant diff between 2 teams's Win Rate Data.(Null hypothesis cannot be rejected)
```

Feature Engineering

Feature Creation

We selected some features to avoid multicollinearity, all analyzed in relation to whether Blue team wins / loses. In the following analysis, we adopt the perspective of the Blue team as the primary viewpoint.

Wins: Binary value,1 = Blue team wins, 0 = Red team Wins.

FirstBlood: Binary value, 1 = Blue team secures the first kill in the game, and 0 = Red team gets the first kill.

GoldDiff: Blue teams' gold - Red teams' gold. (In the 10th min)

KDADiff: Blue teams' KDA - Red teams' KDA. (In the 10th min)

And so forth.

In [85]:
```
df= pd.DataFrame()
df["Wins"]= data["BWins"]
df["FirstBlood"]= data["BFirstBlood"]
df["GoldDiff"]= data["BGoldDiff"]
features= ["KDA", "WardsPlaced", "WardsDestroyed", "TowersDestroyed",
           "AvgLevel", "CSPerMin", "Dragons", "Heralds"]
for feature in features:
    df[feature + "Diff"]= data["B" + feature]- data["R" + feature]


df.head()
```

Out[85]:

	Wins	FirstBlood	GoldDiff	KDADiff	WardsPlacedDiff	WardsDestroyedDiff	TowersDestroyedDiff	AvgLevelDiff	CSPe
0	0	1	643	1.777778	13	-4	0	-0.2	
1	0	0	-2908	0.600000	0	0	-1	-0.2	
2	0	0	-1172	-2.571429	0	-3	0	-0.4	
3	0	0	-1321	-1.950000	15	-1	0	0.0	
4	0	0	-1004	-0.166667	13	2	0	0.0	

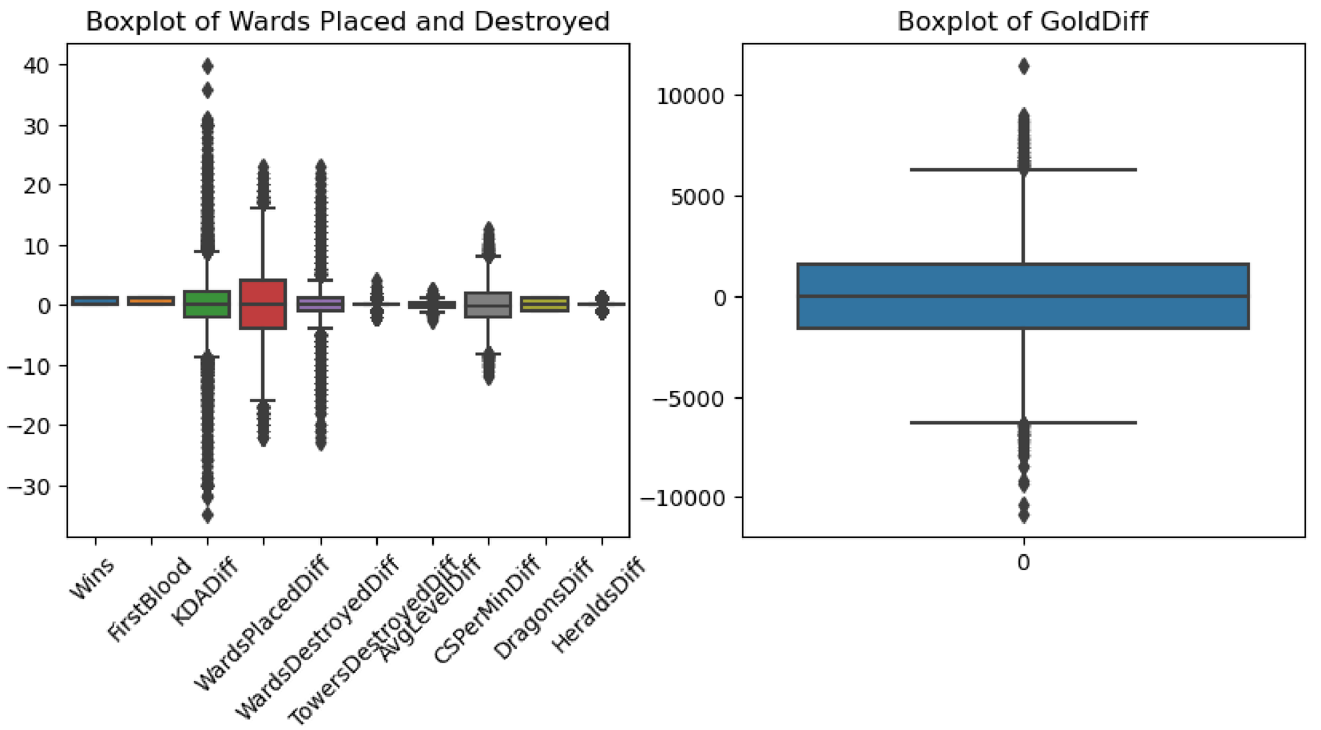
Check The Data Distribution of Features

In [86]:
```
df.describe()
```

Out[86]:

	Wins	FirstBlood	GoldDiff	KDADiff	WardsPlacedDiff	WardsDestroyedDiff	TowersDestroyedDiff
count	9879.000000	9879.000000	9879.000000	9879.000000	9879.000000	9879.000000	9879.000000
mean	0.499038	0.504808	14.414111	0.009623	-0.020144	0.101731	0.008402
std	0.500024	0.500002	2453.349179	6.049175	8.453528	2.854910	0.324835
min	0.000000	0.000000	-10830.000000	-34.857143	-22.000000	-23.000000	-2.000000
25%	0.000000	0.000000	-1585.500000	-2.200000	-4.000000	-1.000000	0.000000
50%	0.000000	1.000000	14.000000	0.000000	0.000000	0.000000	0.000000
75%	1.000000	1.000000	1596.000000	2.171429	4.000000	1.000000	0.000000
max	1.000000	1.000000	11467.000000	39.823529	23.000000	23.000000	4.000000

In [87]:
```
fig, axes= plt.subplots(1, 2, figsize= (10, 4))

sns.boxplot(data= df[df.columns[df.columns!= "GoldDiff"]], ax= axes[0])
axes[0].set(title= "Boxplot of Wards Placed and Destroyed")
axes[0].set_xticklabels(df.columns[df.columns!= "GoldDiff"], rotation= 45)

sns.boxplot(data= df["GoldDiff"], ax= axes[1])
axes[1].set(title= "Boxplot of GoldDiff")

plt.show()
```



There"s no non-logical outliers.

Model Building

Scaling Data

In [88]:
```
X= df.drop(["Wins"], axis= 1).values
Y= df["Wins"].values
scaler= MinMaxScaler().fit(X)
X= scaler.transform(X)
```

Splitting Data

Splitting Data into Trainging and Teststing Sets.

In [89]:
```
X_train, X_test, Y_train, Y_test= train_test_split(X, Y, test_size= .2, random_state= 11)
for n, d in [("X_train", X_train), ("X_test", X_test), ("Y_train", Y_train), ("Y_test", Y_test)]:
    print(f"{n} Shape= {d.shape}")
```

```
X_train Shape= (7903, 10)
X_test Shape= (1976, 10)
Y_train Shape= (7903,)
Y_test Shape= (1976,)
```

Model Training and Prediction

- **Logistic Regression (LR)**
- **Decision Tree (Tree)**
- **Random Forest (RF)**
- **Support Vector Machine (SVC)**

We test these models and evaluate the predictive performance of each model.

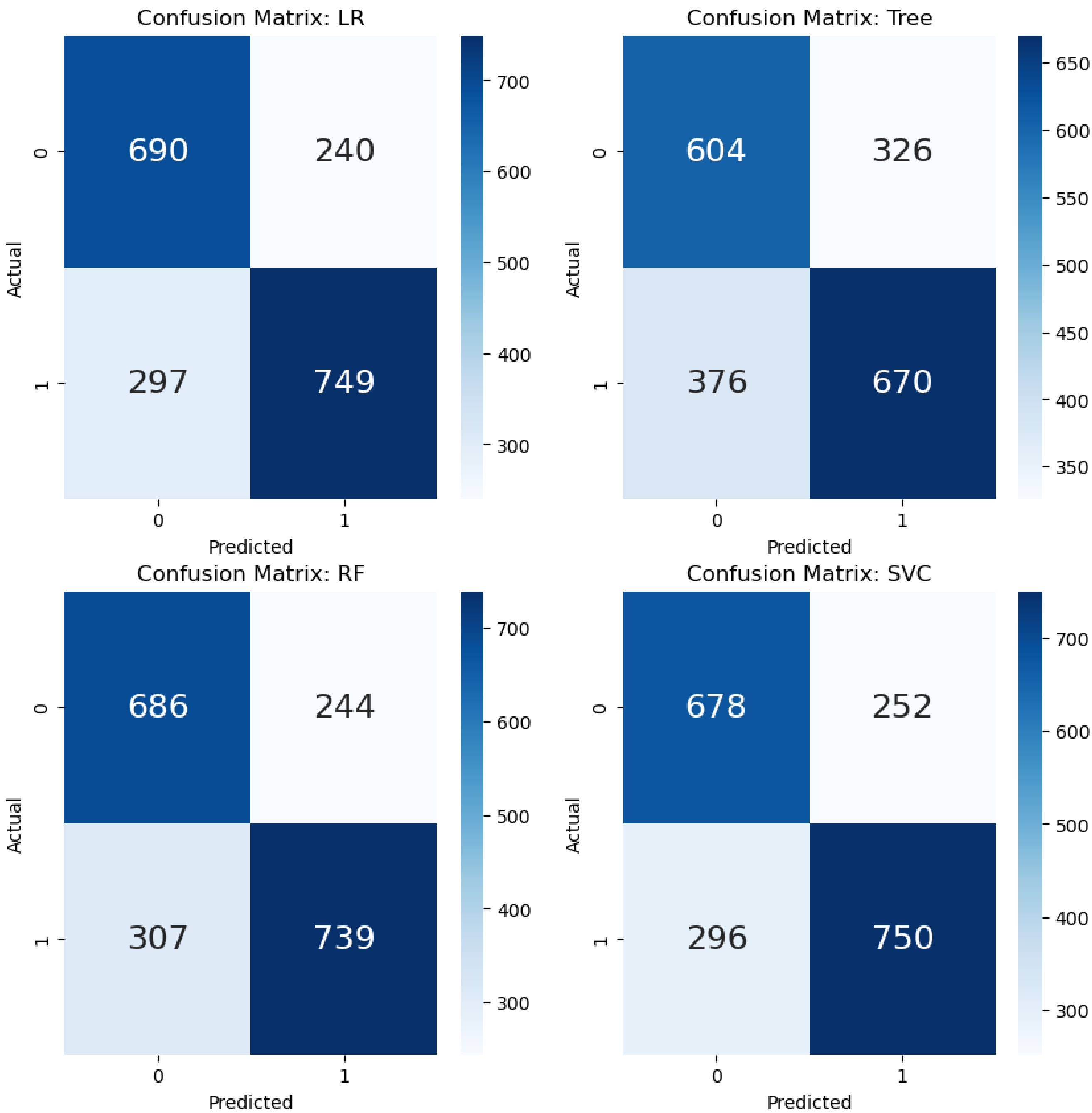
In [90]:
```
Models= {
    "LR": LogisticRegression(),
    "Tree": tree.DecisionTreeClassifier(),
    "RF": RandomForestClassifier(),
    "SVC": SVC()}
```

In [91]:
```
# LRModel= Models["LR"].fit(X_train, Y_train)
# LRPred= LRModel.predict(X_test)
# TreeModel= Models["Tree"].fit(X_train, Y_train)
# TreePred= TreeModel.predict(X_test)
# RFModel= Models["RF"].fit(X_train, Y_train)
# RFPred= RFModel.predict(X_test)
# SVCModel= Models["SVC"].fit(X_train, Y_train)
# SVCPred= SVCModel.predict(X_test)
prediction = {}
for model_name, model in Models.items():
    fitted_model= model.fit(X_train, Y_train)
    prediction[model_name]= fitted_model.predict(X_test)
    print(f"{model_name}: { round(model.score(X_test, Y_test),4)*100} %")
    print(classification_report(prediction[model_name],Y_test))
```

```
LR: 72.82 %
              precision    recall  f1-score   support

           0       0.74      0.70      0.72       987
           1       0.72      0.76      0.74       989

    accuracy                           0.73      1976
   macro avg       0.73      0.73      0.73      1976
weighted avg       0.73      0.73      0.73      1976

Tree: 64.47 %
              precision    recall  f1-score   support

           0       0.65      0.62      0.63       980
           1       0.64      0.67      0.66       996

    accuracy                           0.64      1976
   macro avg       0.64      0.64      0.64      1976
weighted avg       0.64      0.64      0.64      1976

RF: 72.11999999999999 %
              precision    recall  f1-score   support

           0       0.74      0.69      0.71       993
           1       0.71      0.75      0.73       983

    accuracy                           0.72      1976
   macro avg       0.72      0.72      0.72      1976
weighted avg       0.72      0.72      0.72      1976

SVC: 72.27 %
              precision    recall  f1-score   support

           0       0.73      0.70      0.71       974
           1       0.72      0.75      0.73      1002

    accuracy                           0.72      1976
   macro avg       0.72      0.72      0.72      1976
weighted avg       0.72      0.72      0.72      1976
```

Let's check the Confusion Matrix of these outputs.

In [92]:
```
fig, axes= plt.subplots(2, 2, figsize= (10, 10))
for (model_name, predicted_labels), ax in zip(prediction.items(), axes.flatten()):
    cm= confusion_matrix(Y_test, predicted_labels)
    sns.heatmap(cm, annot= True, annot_kws= {"size": 18}, fmt= "d", cmap= "Blues", ax= ax)
    ax.set(title= f"Confusion Matrix: {model_name}", xlabel="Predicted", ylabel="Actual")
plt.show()
```



From the Classification Report and Confusion Matrix, we can observe that the performance of the Decision Tree model is noticeably inferior to the other three.

These models have an accuracy rate of 70% or above in their predictions, except for the decision tree.

Cross Validation

We utilize cross validation to assess models' performance and generalization capabilities.

Let's try Logistic Regression model at first.

In [93]:
```
Cv= 10
Model_LR= Models["LR"]
Score_LR= cross_validate(Model_LR, X_train, Y_train, cv= Cv, scoring= "accuracy")["test_score"]
AvgAcc_LR= np.mean(Score_LR)
SD_LR= np.std(Score_LR)
print(f"Logistic Regression Cross-val Score = {AvgAcc_LR}, ", f"Standard Deviation = {SD_LR}")
```

```
Logistic Regression Cross-val Score = 0.7317460673078461,  Standard Deviation = 0.01243028270048
9926
```

Let"s try other models

In [94]:
```
Model_df= pd.DataFrame(columns= ["Model", "Cross-val Score", "Standard Deviation"])
for model_name, model in Models.items():
    Scores= cross_validate(model, X_train, Y_train, cv= Cv, scoring="accuracy")["test_score"]
    Avg_acc= np.mean(Scores)
    Std_acc= np.std(Scores)

    Model_df= Model_df.append({"Model": model_name, "Cross-val Score": Avg_acc, "Standard Deviati

print(Model_df)
```

```
  Model  Cross-val Score  Standard Deviation
0    LR         0.731746            0.012430
1  Tree         0.640528            0.026605
2    RF         0.718084            0.014840
3   SVC         0.724533            0.014008
```

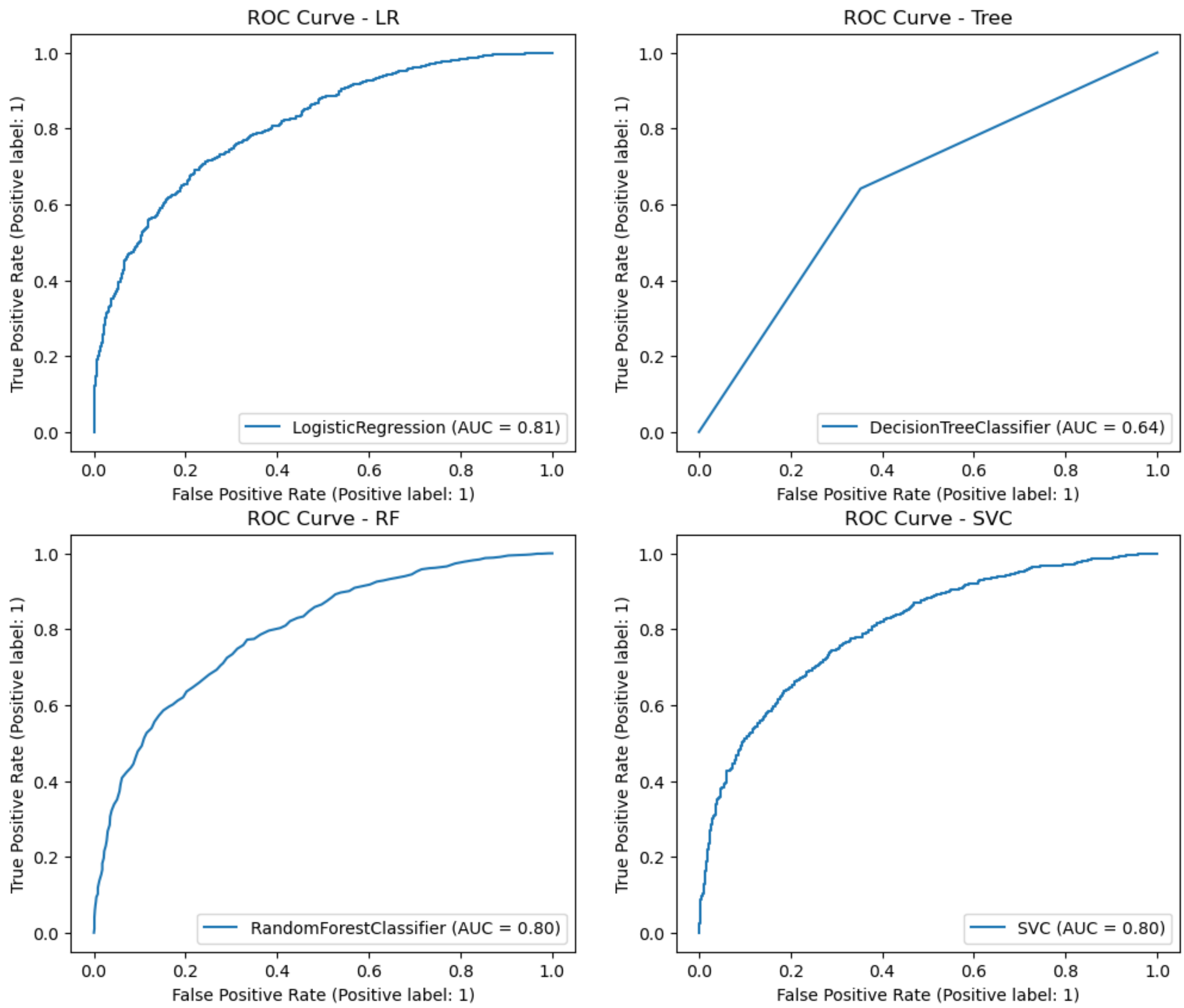
ROC Visualization

ROC (Receiver Operating Characteristic) curve is a representation of a binary classification model's performance across different classification thresholds.

AUC (Area Under the Curve) provides a single scalar value summarizing the performance of a classifier across various threshold settings, ranging from 0 to 1.(A higher value indicates better model performance)

※An 0.5 AUC suggests that the model performs no better than random, and an 1.0 AUC indicates perfect classification.

In [95]:
```
fig, axes= plt.subplots(2, 2, figsize= (12, 10))

for (model_name, model), ax in zip(Models.items(), axes.flatten()):
    fitted_model= model.fit(X_train, Y_train)
    plot_roc_curve(model, X_test, Y_test, ax= ax)
    ax.set_title(f"ROC Curve - {model_name}")

plt.show()
```

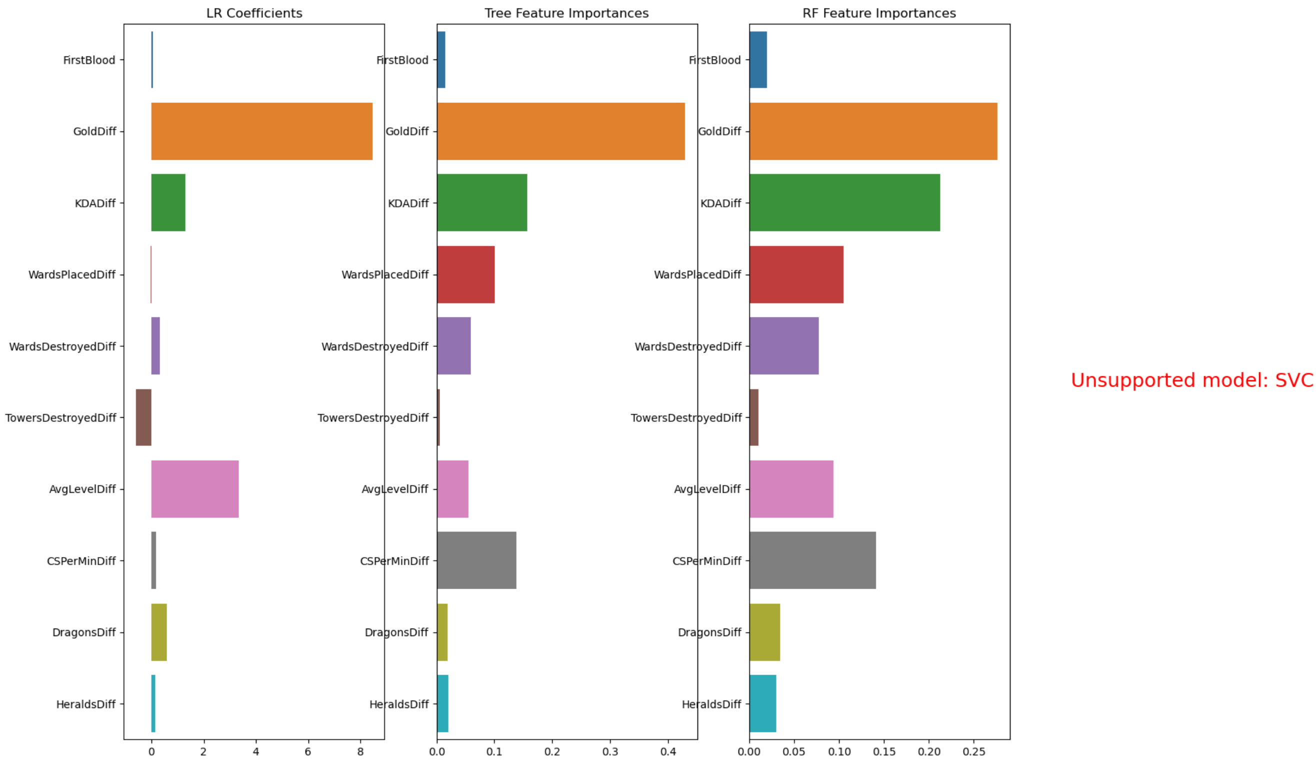


Weights Visualization

Visualizing coefficients or feature importance of the model provides insights into which features contribute the most to the model's predictions.

In [96]:
```
fig, axes= plt.subplots(1, len(Models), figsize= (20, 12))

for idx, (model_name, model) in enumerate(Models.items()):
    model.fit(X_train, Y_train)
    if hasattr(model, "coef_"):
        sns.barplot(x= model.coef_[0], y= df.columns[1:], ax= axes[idx])
        axes[idx].set_title(f"{model_name} Coefficients")
    elif hasattr(model, "feature_importances_"):
        sns.barplot(x= model.feature_importances_, y= df.columns[1:], ax= axes[idx])
        axes[idx].set_title(f"{model_name} Feature Importances")
    else:
        axes[idx].text(0.5, 0.5, f"Unsupported model: {model_name}", ha= "center", va= "center",
        axes[idx].axis("off")
        print(f"Unsupported model: {model_name}")


plt.show()
```

```
Unsupported model: SVC
```



Conclusion

From EDA ,

we have obtained valuable information, such as:

Within the first 10 mins

1. The win rate for the team that secures firstblood has increased by approx. 10% compared to the original win rate.

2. The distribution of average levels for both sides falls between 6.5 and 7.5. If the average level exceeds 7.5, it can be considered a huge advantage.

3. The distribution of gold and level data is very similar.

4. The explanatory power of jungle monster kills is weaker than that of minion kills.

5. Teams that kills 1Dragon + 1Herald have an increased win rate of approx. 20% compared to the original win rate.

6. KDA is an informative indicator for game outcomes.

7.Considering the analyzed factors, there is no significant difference in win rates between Blue or Red sides.

From the Weights Visualization,

it can be observed that GoldDiff is the most crucial factor in determining the game outcome. However, in LOL, many movements contribute to the acquisition of Gold, such as eliminating enemy players, destroying enemy towers, killing minions, jungle minions, and etc. all fundamentally aimed at obtaining Gold (and Experience).

Besides, we can also observe that WardsPlaced and WardsDestroyed carry significant weight (Noted that destroying wards provides only a minimal amount of gold, which can be disregarded), indicating that vision control is also one of the important factors in predicting the outcome of the game.

Based on the issues mentioned above, there may have some collinearity problems in this prediction. But due to limitations in the data provided by the dataset, achieving flawless feature engineering is a challenging task. I believe there is room for further improvement."