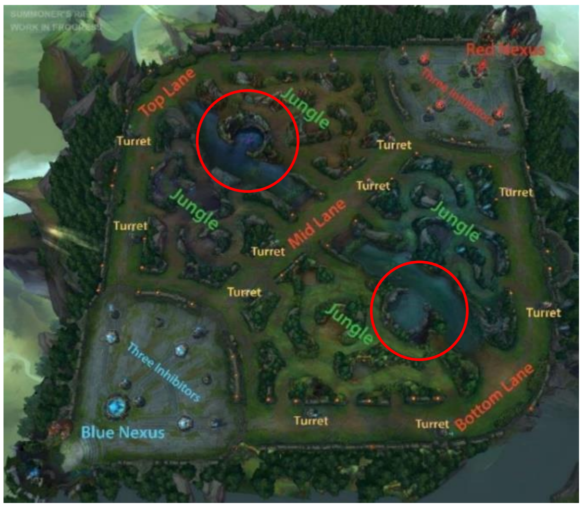
ST409_Final_1

- 'League of Legends' Win / Loss Prediction and Key Factor Analysis -

What is the 'League of Legends'?



This game sets up both sides by dividing the **symmetrical map**. On the each lanes, minions are created and advance toward the enemy base. Players use one side of the game character to raise their levels and abilities by capturing the minions and jungle monsters or destroying the defense towers. Players fight against the opponent's players with the goal of destroying the enemy's **'Nexus'**. On the red circled zone, rift heralds, barons and dragons are generated periodically. Such objectives provide benefit to the team that defeated those objectives.

There are many indicators of inferiority/dominance with the opposing team in the game. For instance, many professional players say that it is virtually impossible to reverse if the gold gap is more than 10000, if the first baron is given to the opponent, or if the opponent preoccupied the four dragons buff effect. It is clear that there are **key factors** that affect the win/loss in the game. Of course, we, who enjoy plain games, already know those factors in commonsense and experience, **but we wanted to verify them using statistical techniques and machine learning models. And, based on those factors, we wanted to see how accurately we could predict the outcome of the game through a trained model.**

1. Data loading

In order to load data, we used Riot API. Riotgames provides users with tools to access their database. Users can get API key from 'developer.riotgames.com'. Users who don't register product can send 20 requests in 1 seconds and 100 requests in 2minutes. Therefore, we needed time.sleep( ) to avoid this rate limit and spent two days to collect all data.

```
In [1]: import requests
        import pandas as pd
        import numpy as np
        import time

        api_key =  'RGAPI-4b716c74-5faf-4cff-88a4-617f81317dc5'
        sohwan = "https://kr.api.riotgames.com/lol/summoner/v4/summoners/by-name/" +'hide on bush' +'?api_key=' + api_key
        r = requests.get(sohwan)
        r

Out[1]: <Response [200]>
```

we can confirm that we succeeded in receiving data. HTTP status code 200 means standard response for successful HTTP requests.

We could receive data in json format using Riot API. So, we needed to convert them into pandas data frame. In addition, we combined 4 tables which are 'league table', 'summoner table', 'matchlist table' and 'match table'. First, league table contains challenger users' information such as a summoner name, league points and summoner Id.

```
In [2]: challenger = 'https://kr.api.riotgames.com/lol/league/v4/challengerleagues/by-queue/RANKED_SOLO_5x5?api_key=' + api_key
        r = requests.get(challenger)#챌린저데이터 호출
        league_df = pd.DataFrame(r.json())

        league_df.reset_index(inplace=True)#수집한 챌린저데이터 index정리
        league_entries_df = pd.DataFrame(dict(league_df['entries'])).T #dict구조로 되어 있는 entries컬럼 풀어주기
        league_df = pd.concat([league_df, league_entries_df], axis=1) #열끼리 결합

        league_df = league_df.drop(['index', 'queue', 'name', 'leagueId', 'entries', 'rank'], axis=1)
        league_df.info()
        league_df.to_csv('챌린저데이터.csv',index=False,encoding = 'cp949')
```

Summoner name is the key we need. We can get encrypted each challengers' account_id by using the key 'summoner name'. Summoner table contains account_id and we can find challenger's account id using 'summoner name' which is the key we got in the previous step.

```
In [5]: for i in range(len(league_df)):
            try:
                sohwan = 'https://kr.api.riotgames.com/lol/summoner/v4/summoners/by-name/' + league_df['summonerName'].iloc[i] + '?api_key=' + api_ke
        y
                r = requests.get(sohwan)

                while r.status_code == 429:
                    time.sleep(10)
                    sohwan = 'https://kr.api.riotgames.com/lol/summoner/v4/summoners/by-name/' + league_df['summonerName'].iloc[i] + '?api_key=' + ap
        i_key
                    r = requests.get(sohwan)

                account_id = r.json()['accountId']
                league_df.iloc[i, -1] = account_id

            except:
                pass
```

Next, we can gather game_id played by challengers. We need account_id as a key. Because we will find the games played by player who has that account_id.

```
In [11]: match_info_df = pd.DataFrame()
         season = str(13)
         for i in range(len(league_df2)):
             try:
                 match0 = 'https://kr.api.riotgames.com/lol/match/v4/matchlists/by-account/' + league_df2['account_id'].iloc[i]  +'?season=' + season
         + '&api_key=' + api_key
                 r = requests.get(match0)

                 while r.status_code == 429:
                     time.sleep(5)
                     match0 = 'https://kr.api.riotgames.com/lol/match/v4/matchlists/by-account/' + league_df2['account_id'].iloc[i]  +'?season=' + sea
         son + '&api_key=' + api_key
                     r = requests.get(match0)

                 match_info_df = pd.concat([match_info_df, pd.DataFrame(r.json()['matches'])])

             except Exception as e:
                 print(i)
                 print(e)
```

We succeeded in gathering gameId . This gameId identifies the games played by challengers. Now we can gather game data using this gameId. 'match table' contains game data and gameId.

```
match_fin = pd.DataFrame()


for i in range(16601, len(match_info_df)):

    api_url='https://kr.api.riotgames.com/lol/match/v4/matches/' + str(match_info_df['gameId'].iloc[i]) + '?api_key=' + api_key
    r = requests.get(api_url)

    if r.status_code == 200: # response가 정상이면 바로 맨 밑으로 이동하여 정상적으로 코드 실행
        pass

    elif r.status_code == 429:
        print('api cost full : infinite loop start')
        print('loop location : ',i)
        start_time = time.time()

        while True: # 429error가 끝날 때까지 무한 루프
            if r.status_code == 429:

                print('try 10 second wait time')
                time.sleep(10)

                r = requests.get(api_url)
                print(r.status_code)

            elif r.status_code == 200: #다시 response 200이면 loop escape
                print('total wait time : ', time.time() - start_time)
                print('recovery api cost')
                break
```

```
    elif r.status_code == 403: # api갱신이 필요
        print('you need api renewal')
        print('break')
        break

    # 위의 예외처리 코드를 거쳐서 내려왔을 때 해당 코드가 실행될 수 있도록 작성
    mat = pd.DataFrame(list(r.json().values()), index=list(r.json().keys())).T
    match_fin = pd.concat([match_fin,mat])


    if i%100 == 0:
        match_fin.to_csv('챌린저데이터(kr_match_fin)tmp4.csv',index=False,encoding = 'cp949')
        f = open('last_saved_index(challenger).txt', 'a')
        data = '₩nlast saved index in loop  : {}'.format(i)
        f.write(data)
        f.close()
```

```
challenger_tot = pd.concat([data1, match_fin])
challenger_tot.drop_duplicates(subset='gameId', inplace=True)
challenger_tot.to_csv('match_fin(챌린저데이터).csv', index=False, encoding='cp949')
```

```
challenger_tot.head()
```

	gameId	platformId	gameCreation	gameDuration	queueId	mapId	seasonId	gameVersion	gameMode	gameType	teams	participants
0	4.42418e+09	KR	1.59112e+12	2169	420	11	13	10.11.322.2991	CLASSIC	MATCHED_GAME	[{'teamId': 100, 'win': 'Win', 'firstBlood': T...	[{'participant 1, 'teamId': 100, 'champion...

We found 'teams' and 'participants' columns contain team and participants' stat in json format. So we needed to preprocess those columns. Participants' columns contain players' stat. Participant who has the number less than or equal to 5 represent blue player. Participant who has the number greater than 5 represents blue player. We can construct teams stat by combining each teams' players stat. For example, by summing up blue team players' vision score we created the variable *visionscore_blue.*

```python
stat_df = pd.DataFrame()
key_error_ls = []



for j in range(match_fin.shape[0]):
    blue_ply_df = pd.DataFrame()
    red_ply_df = pd.DataFrame()


    try :
        for i in range(10):
            if i<=5.5 :
                one_ply = pd.DataFrame(pd.DataFrame(match_fin.participants.values[j]).stats[i], index=[i])
                blue_ply_df = pd.concat([blue_ply_df, one_ply])
            else :
                one_ply = pd.DataFrame(pd.DataFrame(match_fin.participants.values[j]).stats[i], index=[i])
                red_ply_df = pd.concat([red_ply_df, one_ply])

    except KeyError :
        key_error_ls.append(j)
        continue

    blue_ply_df.drop(['item0', 'item1','item2', 'item3', 'item4', 'item5', 'item6'], axis=1, inplace=True)
    red_ply_df.drop(['item0', 'item1','item2', 'item3', 'item4', 'item5', 'item6'], axis=1, inplace=True)

    blue_tot_stat = pd.DataFrame(blue_ply_df.apply(sum)).T
    tmp_arr = np.array(blue_tot_stat.columns) +'_blue'
    blue_tot_stat.columns = tmp_arr.tolist()

    red_tot_stat = pd.DataFrame(red_ply_df.apply(sum)).T
    tmp_arr2 = np.array(red_tot_stat.columns) +'_red'
    red_tot_stat.columns = tmp_arr2.tolist()


    blue_team_stat = pd.DataFrame(pd.DataFrame(match_fin.teams.values[j]).iloc[0]).T
    blue_stat_index = blue_team_stat.columns.tolist()
    blue_team_stat.columns = [x +'_blue'for x in blue_stat_index]

    red_team_stat = pd.DataFrame(pd.DataFrame(match_fin.teams.values[j]).iloc[1]).T
    #red_team_stat.reset_index(inplace=True)
    red_stat_index = red_team_stat.columns.tolist()
    red_team_stat.columns = [x +'_red'for x in red_stat_index]

    team_tot_stat = pd.concat([blue_team_stat, red_team_stat.reset_index(drop=True)], axis=1)
    ply_tot_stat = pd.concat([blue_tot_stat, red_tot_stat.reset_index(drop=True)], axis=1)

    all_stat = pd.concat([team_tot_stat, ply_tot_stat], axis=1)
    all_stat.index = [j]

    try :
        stat_df = pd.concat([stat_df, all_stat])

    except :
        pass
        #print('fail to concat: the number of row in match_fin is ',j, 'problem is ',tmp_ls)
```

Finally, we collected league of legends game data played by challengers in korean server. We repeated this step for master and grandmaster and concatenated 3 tables into one. This is all how we gathered our data.

2. Features in the game & reduction of variables

We can easily identify the features contained in the data by using df.columns. There exist **221 variables** in the raw data. Intuitively, there are too many X variables. If excessive number of variables are used in data analysis, it can lead to a 'curse of dimension' and degrades generalization performance. In other words, the more information we prepare to specify each observatios (i.e, higher level) the more difficult it becomes to learn the model from our data. At this point, there is a need to use feature engineering to process and reduce variables in our raw data that contains 222 columns.

First of all, the encoding process of changing categorical (binary) target presented as character type to 0/1 should precede. As the case of blue win and red win are exclusive to each other, predicting the win/loss of blue is a forecasting of win/loss of the game. That is because the loss of blue means the win of red, and the win of blue is equal to the loss of red. (drop the *win_red*)

```
encoding = {'win_blue' : {'Win' : 1, 'Fail' : 0}}
df.replace(encoding, inplace=True)
df.dropna(inplace=True)

short_df.replace(encoding, inplace=True)
short_df.dropna(inplace=True)

long_df.replace(encoding, inplace=True)
long_df.dropna(inplace=True)
```

```
from sklearn.preprocessing import LabelEncoder

for i in range(2,8):
    le = LabelEncoder()
    y = list(df.iloc[:,i])

    le.fit(y)
    y2 = le.transform(y)

    df.iloc[:,i] = y2
```

We deleted all variables that are not related to the prediction of win/defeat. For instance, the unique identifiers such as *seasonId, mapId, queueId, platformid, participantId* are just id numbers. Therefore we assigned those columns to dropcols.

```
dropcols = ['gameId', # no need columns
            'platformId', 'gameCreation', 'queueId', 'mapId', 'seasonId',
            'gameVersion', 'gameMode', 'gameType', 'teams', 'participants',
            'participantIdentities', 'teamId_blue','teamId_red', 'bans_blue', 'bans_red',
```

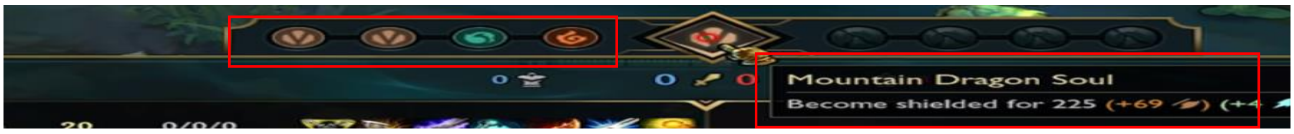
And there exist variables that can be included in other variables. For instance, *doubleKills_blue* (# of time the blue team has killed enemies twice in a row) is included in the *kills_blue* (total kills made by blue team). And *physicalDamageDealt_blue* (physical attack), *magicalDamageDealt_blue* (magical attack), *trueDamageDealt_blue* (unreducible fixed damage) are included in the *totalDamageDealtToChampions_blue* (all the damage done to the enemy). Those variables are assigned to dropcols2.

```
dropcols2=['doubleKills_blue', 'tripleKills_blue', 'quadraKills_blue','pentaKills_blue','unrealKills_blue','totalDamageDealt_blue',
          'magicDamageDealt_blue','physicalDamageDealt_blue','trueDamageDealt_blue','largestCriticalStrike_blue',
          'magicDamageDealtToChampions_blue','physicalDamageDealtToChampions_blue','trueDamageDealtToChampions_blue',
          'damageDealtToObjectives_blue','damageDealtToTurrets_blue','magicalDamageTaken_blue','physicalDamageTaken_blue',
          'trueDamageTaken_blue','neutralMinionsKilled_blue','neutralMinionsKilledEnemyJungle_blue',
```

dropcols are dropped from our raw data by drop method in python.

```
df.drop(columns=dropcols, inplace=True)
df.drop(columns=dropcols2, inplace=True)
```

After this, there still remains 86 columns. We conducted feature engineering again by using rules of game. One of the objectives, dragon provides more powerful advantage to the team that killed dragons 4 times. In addition, if the side that killed the dragon 4 times first, kills the elder dragon, it gives the effect of 'execution of the elder', that eliminates enemies below a certain level of HP immediately.



Therefore, we added variable that can contain this information by

```
# 4드래곤 변수와 장로드래곤 변수 만듦

df['blue4dragons'] = 0
df['red4dragons'] = 0
df['blue_olddragon'] = 0
df['red_olddragon'] = 0

for i in range(0, df.shape[0]):
    blue_dragon = df['dragonKills_blue']
    if blue_dragon[i] == 4:
        df['blue4dragons'][i] = 1
    if blue_dragon[i] >= 5:
        df['blue_olddragon'][i] = blue_dragon[i]-4

for i in range(0, df.shape[0]):
    red_dragon = df['dragonKills_red']
    if blue_dragon[i] == 4:
        df['red4dragons'][i] = 1
    if red_dragon[i] >= 5:
        df['red_olddragon'][i] = red_dragon[i]-4
```

Our objective is predicting the win/loss of blue by using not only the information of blue's but also that of corresponding red's. Because, whether you're doing well or bad at the game is relative. By defining the variables that identify the difference between the red and blue in the same variable, we can reduce the dimension of data.

```
df['gold_diff'] = df['goldEarned_blue'] - df['goldEarned_red']
df['level_diff'] = df['champLevel_blue'] - df['champLevel_red']
df['jg_gap'] = df['neutralMinionsKilledTeamJungle_blue'] - df['neutralMinionsKilledTeamJungle_red']
df['liner_gap'] = df['totalMinionsKilled_blue'] - df['totalMinionsKilled_red']
df['tower_gap'] = df['towerKills_blue'] - df['towerKills_red']
df['damage_gap'] = df['totalDamageDealtToChampions_blue'] - df['totalDamageDealtToChampions_red']
df['heal_gap'] = df['totalHeal_blue'] = df['totalHeal_red']
df['vision_gap'] = df['visionScore_blue'] - df['visionScore_red'] # vision score 변수에 다른 시야관련 변수들이 포함되어 있음
```

```
df = df.drop(['goldEarned_blue', 'goldEarned_red', 'champLevel_blue', 'champLevel_red',
             'neutralMinionsKilledTeamJungle_blue', 'neutralMinionsKilledTeamJungle_red', 'totalMinionsKilled_blue',
             'totalMinionsKilled_red', 'towerKills_blue', 'towerKills_red', 'totalHeal_blue', 'totalHeal_red', 'visionScore_blue',
             'visionScore_red', 'wardsKilled_blue', 'wardsKilled_red', 'kills_blue', 'assists_blue', 'deaths_blue',
              'kills_red', 'assists_red', 'deaths_red', 'wardsKilled_blue', 'wardsKilled_red',
```

# columns used to define the gap variables and variables that have all zero are removed

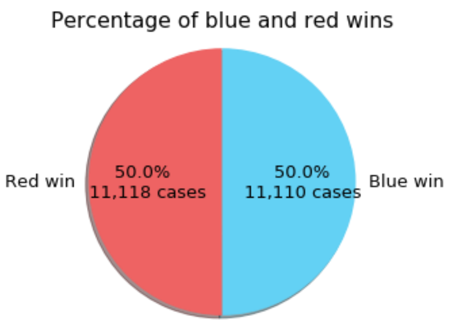
By recombining the original variables using the concept of difference, the identical information of both sides (2 columns needed) can be reflected in only variable. By using the game rule, we can corporate 3 variables into one variable. KDA score is defined by calculating (kill + assist) / death.

```
df['blue_kda'] = (df['kills_blue'] + df['assists_blue']) / (df['deaths_blue'] + 0.01) # 0으로 나누는 것을 방지
df['red_kda'] = (df['kills_red'] + df['assists_red']) / (df['deaths_red'] + 0.01)
```

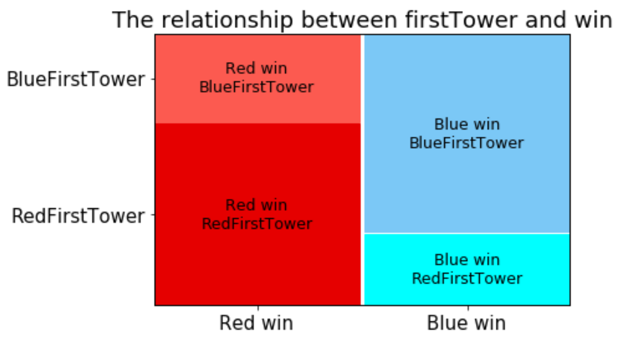
# epsilon value(0.01) was added to prevent dividing by zero

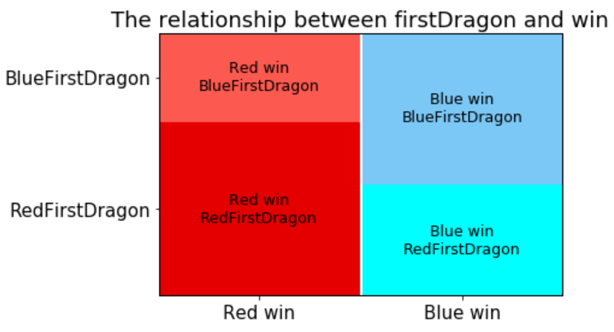
3. Data visualization

Before we start analyzing our data, we confirmed that our data consists of 50% blue win and 50% red win. This implies that the probability distribution without any given information, but team membership is Bernoulli distribution with success probability 0.5. The success is blue win.

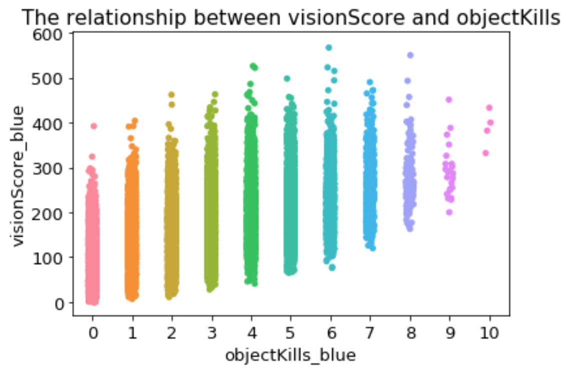


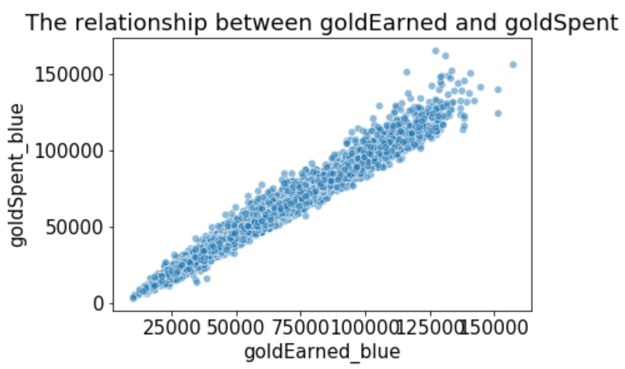
To identify covariate which show distinct pattern with target, we plotted mosaic plot.





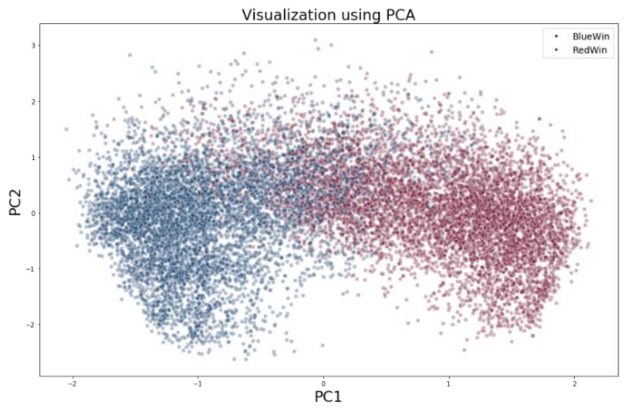
And we found some covariates which varies with target. *'FirstTower'* and *'FirstDragon'* are identified as variable indicating win. In addition, we try to identify the relationship between explanatory variables.

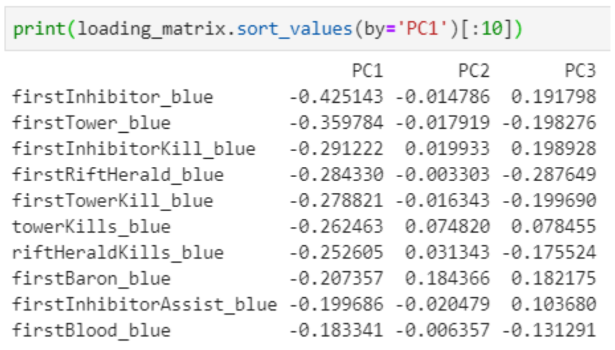




We found several explanatory variables correlated with each other. One of them is the correlation between vision score and object kills. Another one is correlation between gold earned and gold spent. So, we thought we need feature engineering and dimensionality reduction in order to avoid multicollinearity and curse of dimensionality.

In order to look over high dimensionality data at a one glance, we used Principal Component Analysis. We found we can distinguish blue win game from red win game by PC1. PC1 represents how much blue lost initiative in the front of game.





```
print(loading_matrix.sort_values(by='PC1')[:10])

                              PC1       PC2       PC3
firstInhibitor_blue       -0.425143 -0.014786  0.191798
firstTower_blue           -0.359784 -0.017919 -0.198276
firstInhibitorKill_blue   -0.291222  0.019933  0.198928
firstRiftHerald_blue      -0.284330 -0.003303 -0.287649
firstTowerKill_blue       -0.278821 -0.016343 -0.199690
towerKills_blue           -0.262463  0.074820  0.078455
riftHeraldKills_blue      -0.252605  0.031343 -0.175524
firstBaron_blue           -0.207357  0.184366  0.182175
firstInhibitorAssist_blue -0.199686 -0.020479  0.103680
firstBlood_blue           -0.183341 -0.006357 -0.131291
```

4. Identify relationship between variables

As in the above case, if a variable is defined as a linear combination of different variables, it means that the variables are correlated each other. The data that covered in courses usually assume the independence of X variables. However, such assumptions are for the convenience of learners, and it is right to say that the actual data are different. Models that using those highly correlated variables can be problematic because inverse of (X^TX) is not defined (one of the columns is defined by the linear combination of others = determinant is zero = no inverse). Therefore, we conducted VIF and factor analysis to identify the relationship between the variables.

VIF (Variance Inflation Factor) is obtained by calculating $1/1-R_i^2$, where R_i is obtained from the linear regression equation $X_i = B_1X_1 + B_2X_2 + \ldots + B_{i-1}X_{i-1} + B_{i+1}X_{i+1} + \ldots + B_pX_p$. It considers X_i as a target and set variables except X_i as independent variables. Then we can calculate R_i^2, the higher the R_i^2, the better the X_i variable is explained by regression of other variables (dependency). If $R_i^2 > 0.9$ ($VIF_i > 10$), X_i is a highly correlated variable which implies the existence of multicollinearity. VIF can be obtained by simple coding below

```python
from statsmodels.stats.outliers_influence import variance_inflation_factor

x = df.drop(columns=['gameDuration', 'win_blue'])
vif = pd.DataFrame()
vif["VIF Factor"] = [variance_inflation_factor(x.values, i) for i in range(x.shape[1])]
vif["features"] = x.columns
vif.sort_values(by='VIF Factor', ascending=False)

# 변수들간의 상관관계 존재
```

	VIF Factor	features
18	inf	totalDamageDealtToChampions_blue
36	inf	red4dragons
35	inf	blue4dragons
30	inf	totalDamageDealtToChampions_red
46	inf	damage_gap
20	371.965744	totalDamageTaken_blue
21	338.817946	goldSpent_blue
32	218.679486	totalDamageTaken_red
33	212.813856	goldSpent_red
39	146.853861	gold_diff

Variables in our data is highly correlated. Therefore there exist the need to conduct feature selection methods such as Lasso, Ridge and PCA etc.

It is natural to think that the elements in the game are related to each other. For instance, obtaining dragons (dragonKills), destroying tower (towerKills) and killing enemies (Kills) provide gold (goldSpent).

To identify the latent relationships between the variables, we conducted factor analysis.

```python
from factor_analyzer import FactorAnalyzer

df1 = df.iloc[:, 2:] # with 'first' columns
df2 = df.iloc[:, 16:] # without 'first' columns

df1_train, df1_test = train_test_split(df1, test_size=0.25, random_state=1234)
df2_train, df2_test = train_test_split(df2, test_size=0.25, random_state=1234)
```

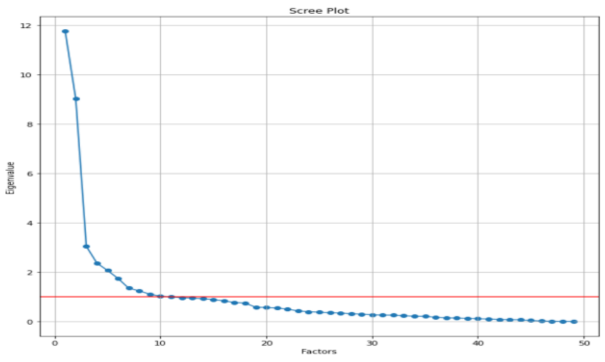
First divide the data into train set and test set. And we can see the statistical significance of the dimension reduction by bartlett test. It calculates the value of a determinant of correlation matrix derived from sample, and test (chi-square test) whether the correlation matrix is a unit matrix or not. If correlation matrix is similar to a unit matrix, it means all variables have little relationships with other variables (correlation close to 0).

```python
from factor_analyzer.factor_analyzer import calculate_bartlett_sphericity

chi_square_value, p_value=calculate_bartlett_sphericity(df2_train)
chi_square_value, p_value # 차원축소를 할 통계적 유의성이 존재
```

The result was (1712910.491423723, 0.0), as the p-value is zero we reject H0 (correlation matrix is a unit matrix), and there is a statistical significance of dimension reduction. Appropriate number of reduction is identified by scree-plot.

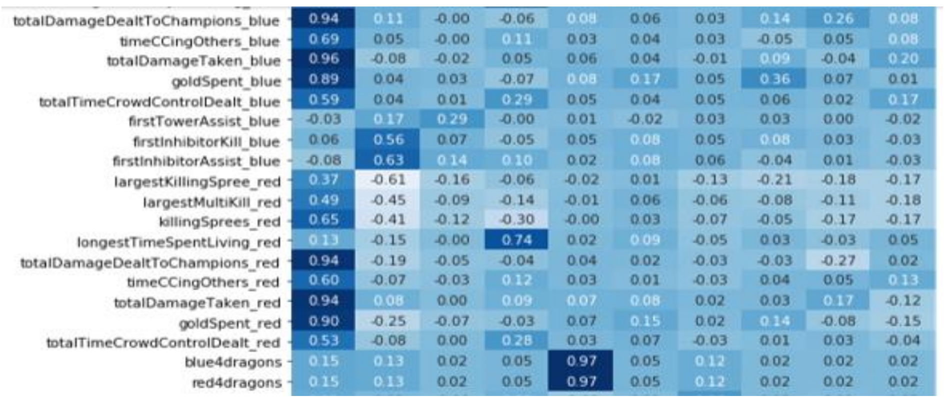
```python
from factor_analyzer.factor_analyzer import calculate_bartlett_sphericity

chi_square_value, p_value=calculate_bartlett_sphericity(df2_train)
chi_square_value, p_value # 차원축소를 할 통계적 유의성이 존재
```

```python
import matplotlib.pyplot as plt

plt.figure(figsize=(10,10))
plt.scatter(range(1,df1.shape[1]+1),ev1)
plt.plot(range(1,df1.shape[1]+1),ev1)
plt.title('Scree Plot')
plt.xlabel('Factors')
plt.ylabel('Eigenvalue')
plt.axhline(y=1, color='r', linestyle='-')
plt.grid()
plt.show()
```



According to the scree plot, 10 factor model is appropriate.

```
fa1 = FactorAnalyzer(n_factors=10, method='ml', rotation='varimax')
fa1.fit(df1_train)
efa_result= pd.DataFrame(fa1.loadings_, index=df1_train.columns)

plt.figure(figsize=(10,15))
sns.heatmap(efa_result, cmap="Blues", annot=True, fmt='.2f')
```



totalDamageDealtToChampions_blue	0.94	0.11	-0.00	-0.06	0.08	0.06	0.03	0.14	0.26	0.08
timeCCingOthers_blue	0.69	0.05	-0.00	0.11	0.03	0.04	0.03	-0.05	0.05	0.08
totalDamageTaken_blue	0.96	-0.08	-0.02	0.05	0.06	0.04	-0.01	0.09	-0.04	0.20
goldSpent_blue	0.89	0.04	0.03	-0.07	0.08	0.17	0.05	0.36	0.07	0.01
totalTimeCrowdControlDealt_blue	0.59	0.04	0.01	0.29	0.05	0.04	0.05	0.06	0.02	0.17
firstTowerAssist_blue	-0.03	0.17	0.29	-0.00	0.01	-0.02	0.03	0.03	0.00	-0.02
firstInhibitorKill_blue	0.06	0.56	0.07	-0.05	0.05	0.08	0.05	0.08	0.03	-0.03
firstInhibitorAssist_blue	-0.08	0.63	0.14	0.10	0.02	0.08	0.06	-0.04	0.01	-0.03
largestKillingSpree_red	0.37	-0.61	-0.16	-0.06	-0.02	0.01	-0.13	-0.21	-0.18	-0.17
largestMultiKill_red	0.49	-0.45	-0.09	-0.14	-0.01	0.06	-0.06	-0.08	-0.11	-0.18
killingSprees_red	0.65	-0.41	-0.12	-0.30	-0.00	0.03	-0.07	-0.05	-0.17	-0.17
longestTimeSpentLiving_red	0.13	-0.15	-0.00	0.74	0.02	0.09	-0.05	0.03	-0.03	0.05
totalDamageDealtToChampions_red	0.94	-0.19	-0.05	-0.04	0.04	0.02	-0.03	-0.03	-0.27	0.02
timeCCingOthers_red	0.60	-0.07	-0.03	0.12	0.03	0.01	-0.03	0.04	0.05	0.13
totalDamageTaken_red	0.94	0.08	0.00	0.09	0.07	0.08	0.02	0.03	0.17	-0.12
goldSpent_red	0.90	-0.25	-0.07	-0.03	0.07	0.15	0.02	0.14	-0.08	-0.15
totalTimeCrowdControlDealt_red	0.53	-0.08	0.00	0.28	0.03	0.07	-0.03	0.01	0.03	-0.04
blue4dragons	0.15	0.13	0.02	0.05	0.97	0.05	0.12	0.02	0.02	0.02
red4dragons	0.15	0.13	0.02	0.05	0.97	0.05	0.12	0.02	0.02	0.02

# Part of the FA loadings heatmap

```
pd.DataFrame(fa1.get_factor_variance(), index = ['SS Loadings', 'Proportion Var', 'Cumulative Var'])
```

# 4 factors can explain the half of the total variance

	0	1	2	3	4	5	6	7	8	9
SS Loadings	10.338418	6.698949	2.683211	2.581737	2.201519	1.742404	1.540786	1.407341	1.024785	0.632525
Proportion Var	0.210988	0.136713	0.054759	0.052689	0.044929	0.035559	0.031445	0.028721	0.020914	0.012909
Cumulative Var	0.210988	0.347701	0.402461	0.455149	0.500078	0.535637	0.567082	0.595803	0.616717	0.629626

From this we can group the similar variable into the same factor. We derived important 5 factors from above result. Factor 0 is a team performance (kda, damage, heal, level), factor 1 is a structure destruction and resource (first tower/inhibitor destruction, gold difference) factor 2 is tower destruction (tower destruction, rift herald that destroy the tower), factor 3 is a survival time and factor 4 is a dragon buff (4 dragons).

5. Model Fit

The entire data set was randomly divided into 8:2 ratios to create train sets and test sets. After fitting the 'Factor Analyzer' to the train set, a data set was created with the variables reduced to 10 by using the factor scores obtained by applying the model to the train set and the test set.

```
from factor_analyzer import FactorAnalyzer

df_train, df_test = train_test_split(df, test_size = 0.25, random_state = 77)
```

```
X_train, y_train = df_train.iloc[:, 2:], df_train.win_blue
X_test, y_test = df_test.iloc[:, 2:], df_test.win_blue
```

# Split Train set and Test set

```
fa_score1 = pd.DataFrame(fa1.transform(X_train))
fa_score1['win_blue'] = y_train.ravel()
fa_score11_train = fa_score1.copy()
```

•••

```
fa_score11_test = pd.DataFrame(fa1.transform(X_test))
fa_score11_test['win_blue'] = y_test.ravel()
```

# Use factor scores to create data set with reduced variables

We used a data set that uses EFA to reduce explanatory variables to 10 and a data set that uses all variables for making predictive model. And the performance of each model was compared. After that, important variables from each model were identified and the results were analyzed. The models used are Random Forest, Logistic Regression, and SVM.

In case of using all variables, 49 features except game duration and target variable win_blue of train set were assigned as X_train and win_blue as y_train, and X_test and y_test were assigned in the same way in the test set..

In the case of using a variable that was reduced using EFA, factor 0 to factor 9 were assigned to X_train and win_blue to y_train in the train set, and X_test and y_test were assigned in the same way in the test set.

5_1. Random Forest

First, when all variables are used. The optimal hyper-parameter was found using the grid search and the corresponding parameter was used. Grid search is a numerical optimization method that finds parameters that satisfy the optimization requirements by repeated trial and error. That is, after setting the parameters of a certain range or a specific candidate group, the results in all cases are compared to find the parameter having the best result. As a result of prediction using the hyperparameters derived through the grid search, the accuracy was 0.985. The accuracy is calculated by calculating the ratio of the actual value (y_test) and the predicted value.

```
from sklearn.model_selection import GridSearchCV

params = { 'n_estimators' : [0, 100, 150, 200],
           'max_depth' : [0, 5, 10, 15, 20],
           'min_samples_leaf' : [0, 5, 10, 15, 20],
           'min_samples_split' : [0, 5, 10, 15, 20]
            }

# RandomForestClassifier 객체 생성 후 GridSearchCV 수행
rf = RandomForestClassifier(random_state = 0, n_jobs = -1)
grid_cv = GridSearchCV(rf, param_grid = params, cv = 3, n_jobs = -1)
grid_cv.fit(X_train, y_train)

print('최적 하이퍼 파라미터: ', grid_cv.best_params_)
print('최고 예측 정확도: {:.4f}'.format(grid_cv.best_score_))
```

```
최적 하이퍼 파라미터:  {'max_depth': 20, 'min_samples_leaf': 5, 'min_samples_split': 5, 'n_estimators': 150}
최고 예측 정확도: 0.9837
```

```
rf_clf = RandomForestClassifier(max_depth=20, min_samples_leaf=5, min_samples_split=5, n_estimators=150, random_state=628)
rf_clf.fit(X_train, y_train)

predicted = rf_clf.predict(X_test)
accuracy = accuracy_score(y_test, predicted)

print(f'Mean accuracy score: {accuracy:.3}')
```

```
Mean accuracy score: 0.985
```

The same process is applied to the reduced dataset in which the variables are reduced using EFA, and the Random Forest fit results show an accuracy of 0.961. There is no significant difference from the case of using the whole variable, and despite the fact that about 50 variables are reduced to 10, the accuracy is still over 95%

.

```
print('최적 하이퍼 파라미터: ', grid_cv.best_params_)
print('최고 예측 정확도: {:.4f}'.format(grid_cv.best_score_))
```

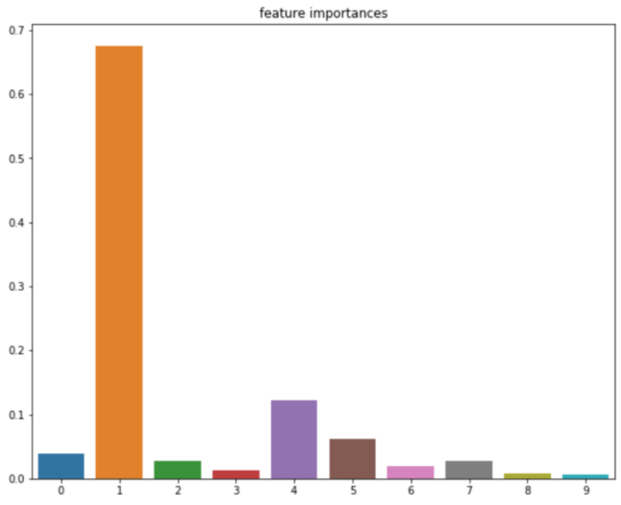
```
최적 하이퍼 파라미터:  {'max_depth': 10, 'min_samples_leaf': 5, 'min_samples_split': 5, 'n_estimators': 100}
최고 예측 정확도: 0.9577
```

```
rf_clf = RandomForestClassifier(max_depth=10, min_samples_leaf=5, min_samples_split=5, n_estimators=100, random_state=628)
rf_clf.fit(X_train, y_train)

predicted = rf_clf.predict(X_test)
accuracy = accuracy_score(y_test, predicted)

print(f'Mean accuracy score: {accuracy:.3}')
```

```
Mean accuracy score: 0.961
```

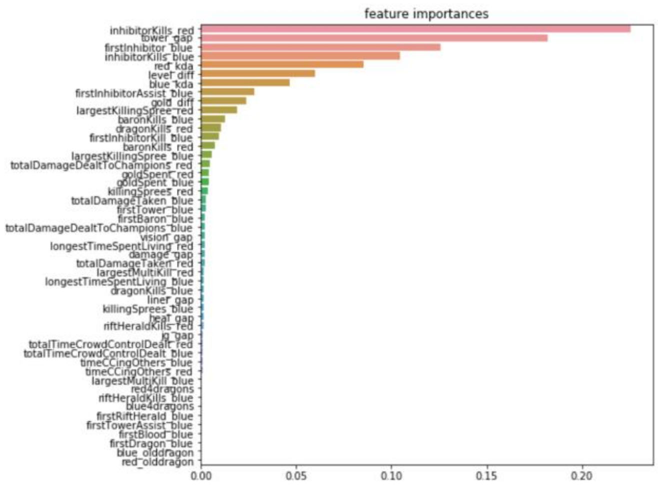
Thereafter, the result of visualizing the importance of each variable in both cases using the 'feature_importances_' method of random forest is as follows.

```
feature_importances = rf_clf.feature_importances_

ft_importances=pd.Series(feature_importances, index = X_train.columns)
ft_importances = ft_importances.sort_values(ascending=False)

plt.figure(figsize = (10,8))
plt.title('feature importances')
sns.barplot(x=ft_importances.index, y = ft_importances)
plt.show()
```



```
ft_importances

1     0.675673
4     0.121406
5     0.061966
0     0.039534
2     0.028128
7     0.027382
6     0.019415
3     0.013030
8     0.007328
9     0.006138
```

# Use factor score data set



```
inhibitorKills_red                  0.225869
tower_gap                           0.182187
firstInhibitor_blue                 0.125719
inhibitorKills_blue                 0.104450
red_kda                             0.085535
level_diff                          0.060098
blue_kda                            0.046851
firstInhibitorAssist_blue           0.028319
gold_diff                           0.023743
largestKillingSpree_red             0.019231
baronKills_blue                     0.012668
dragonKills_red                     0.010440
firstInhibitorKill_blue             0.009711
baronKills_red                      0.007420
largestKillingSpree_blue            0.005991
totalDamageDealtToChampions_red     0.004779
goldSpent_red                       0.004318
goldSpent_blue                      0.004159
```

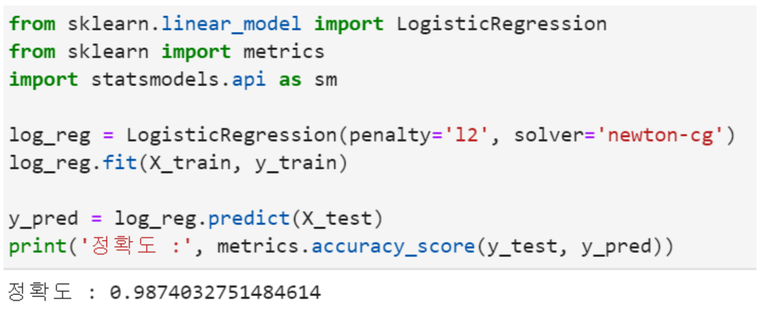
# Use all variables

In the variable reduction model, the importance of factor 1 and factor 4 was found to be overwhelmingly high. As previously explained, these two factors refer to the destruction of the structure, the degree of gold acquisition, and the buff caused by the dragon treatment. In all variables, the destruction of the structure such as *inhibitorKills_red, tower_gap, firstInhibitor_blue* and the gain of goals such as *gold_diff* derived from it were found in the upper level.

That is, when the variables were reduced using EFA due to the high correlation between the variables, it was possible to implement a high-accuracy predictive model with only 10 factors and to identify variables that significantly influenced win/loss.

5_2. Logistic Regression

The results predicted using the full explanatory variables through logistic regression are as follows.

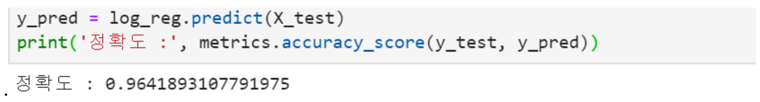


```
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
import statsmodels.api as sm

log_reg = LogisticRegression(penalty='l2', solver='newton-cg')
log_reg.fit(X_train, y_train)

y_pred = log_reg.predict(X_test)
print('정확도 :', metrics.accuracy_score(y_test, y_pred))
```

```
정확도 : 0.9874032751484614
```

Logistic regression was used because the target variable was categorical, and L2 (Lasso) regulation was used to completely remove the weight of the less important variable in the process. The algorithm to be used for optimization was set to 'newton-cg'. One of the optimization algorithms that can be used when penalty='l2' in the Logistic Regression function is 'newton-cg'. As a 'Quasi-Newton Method' that compensates for the shortcomings of the Newton method, it is a method of directly calculating a modified gradient vector without requiring a Hessian matrix. The accuracy is about 99%, and like the random forest, the accuracy was calculated through the ratio of the actual value and the predicted value are same.

The prediction results using the reduced variables are as follows.

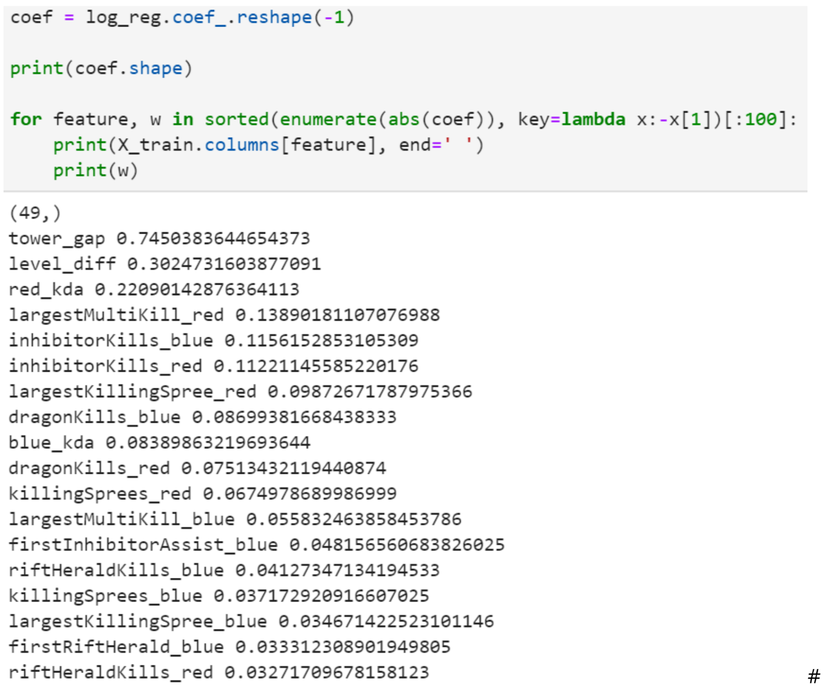


```
y_pred = log_reg.predict(X_test)
print('정확도 :', metrics.accuracy_score(y_test, y_pred))
```

```
정확도 : 0.9641893107791975
```

As with the random forest model, it can be seen that although the accuracy is slightly lower than the case where the entire variable is used, the accuracy is still over 95%.

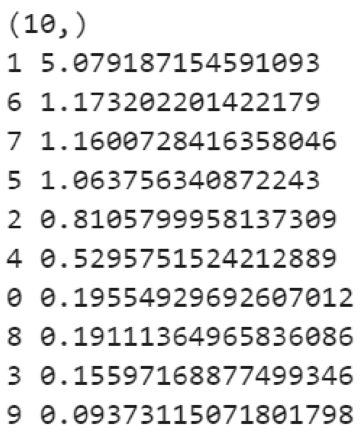
The results of deriving the importance of variables by sorting the absolute values of the regression coefficients in both cases are as follows.



```
coef = log_reg.coef_.reshape(-1)

print(coef.shape)

for feature, w in sorted(enumerate(abs(coef)), key=lambda x:-x[1])[:100]:
    print(X_train.columns[feature], end=' ')
    print(w)
```

```
(49,)
tower_gap 0.7450383644654373
level_diff 0.3024731603877091
red_kda 0.22090142876364113
largestMultiKill_red 0.13890181107076988
inhibitorKills_blue 0.1156152853105309
inhibitorKills_red 0.11221145585220176
largestKillingSpree_red 0.09872671787975366
dragonKills_blue 0.08699381668438333
blue_kda 0.08389863219693644
dragonKills_red 0.07513432119440874
killingSprees_red 0.0674978689986999
largestMultiKill_blue 0.055832463858453786
firstInhibitorAssist_blue 0.048156560683826025
riftHeraldKills_blue 0.04127347134194533
killingSprees_blue 0.037172920916607025
largestKillingSpree_blue 0.034671422523101146
firstRiftHerald_blue 0.033312308901949805
riftHeraldKills_red 0.03271709678158123
```

# Use full variables



```
(10,)
1 5.079187154591093
6 1.173202201422179
7 1.1600728416358046
5 1.063756340872243
2 0.8105799958137309
4 0.5295751524212889
0 0.19554929692607012
8 0.19111364965836086
3 0.15597168877499346
9 0.09373115071801798
```

# Use reduced variables

When looking at important variables among all variables, there are many variables related to structure destruction such as *tower_gap, inhibitorKills_blue, inhibitorKills_red*, and variables related to performance of team members and object treatment such as *'red_kda', 'largestMultiKills_red', and 'drgondKills_blue'*. This is reflected in the increased importance of factors 6 and 7 related to level differences, differences in gold gains, and damage done to opponents than in the case of random forests. However, it can be seen that the importance of factor 1 is overwhelmingly the same as the result of random forest fit..

5_3. SVM (Support Vector Machine)

The prediction result using the entire explanatory variable through SVM is as follows.

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn import svm

svm = svm.SVC(kernel='linear')
svm.fit(X_train, y_train)
y_pred = svm.predict(X_test)
print('정확도 :', metrics.accuracy_score(y_test, y_pred))
```

```
정확도 : 0.9641893107791975
```

Linear SVM was used and the accuracy was calculated in the same way as other models. The accuracy when using all variables is about 0.96.

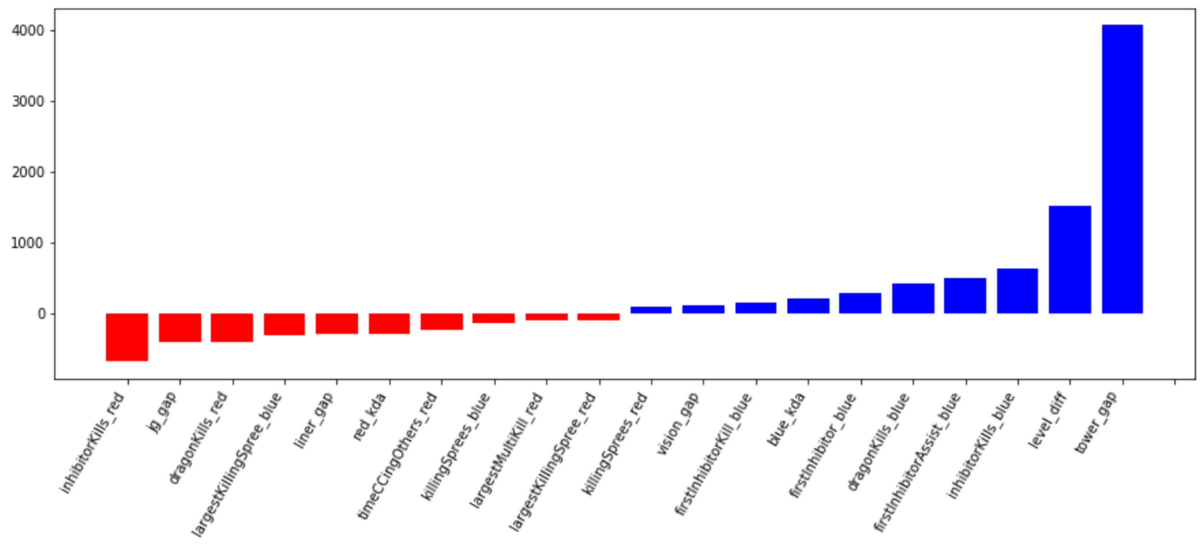
The prediction results using the reduced variables are as follows.

```
svm = svm.SVC(kernel='linear')
svm.fit(X_train, y_train)
y_pred = svm.predict(X_test)
print('정확도 :', metrics.accuracy_score(y_test, y_pred))
```

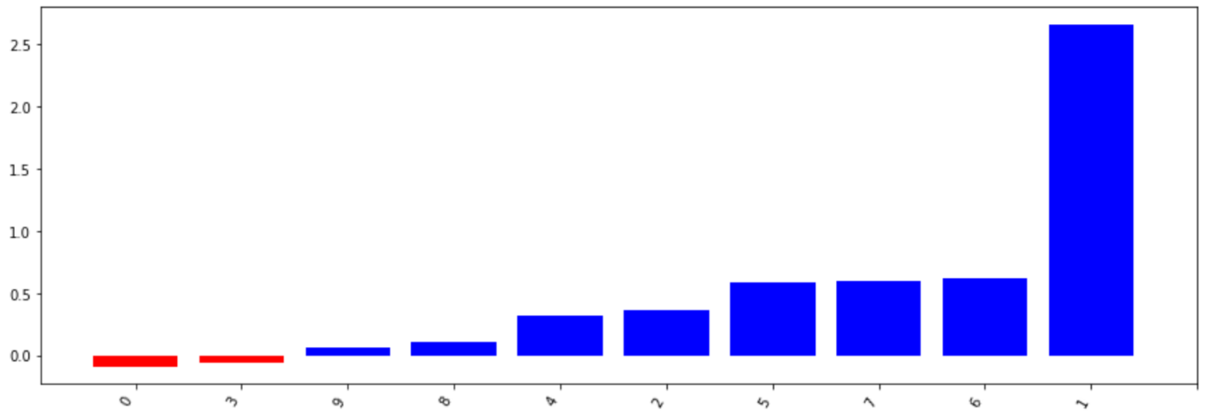
```
정확도 : 0.9643692639913622
```

In the case of SVM, it is confirmed that the accuracy when using the reduced variable is almost the same as when using the entire variable. Through this, high accuracy can be maintained even when prediction is performed with only the reduced variables in above models. The result of visualizing the importance of the variable when using the entire variable and the reduced variable in SVM is as follows.

```
def plot_coefficients(classifier, feature_names, top_features=6):
    coef = classifier.coef_.ravel()
    top_positive_coefficients = np.argsort(coef)[-top_features:]
    top_negative_coefficients = np.argsort(coef)[:top_features]
    top_coefficients = np.hstack([top_negative_coefficients, top_positive_coefficients])
    # create plot
    plt.figure(figsize=(15, 5))
    colors = ['red' if c < 0 else 'blue' for c in coef[top_coefficients]]
    plt.bar(np.arange(2 * top_features), coef[top_coefficients], color=colors)
    feature_names = np.array(feature_names)
    plt.xticks(np.arange(0, 1 + 2 * top_features), feature_names[top_coefficients], rotation=60, ha='right')
    plt.show()
```



```
plot_coefficients(svm, X_train.columns, top_features = 5)
```



We could use the linear svm 'coef_' method to determine the important factors of winning or losing. In the random forest or logistic regression model, only the degree of affecting the winning or losing itself could be confirmed, but in the SVM model, the factors affecting the winning of the blue team and the red team were separately identified. However, it was difficult to give a meaningful insight to the analysis because the importance was symmetric in both camps unless it was a randomly generated variable. In the reduced variable, the importance of factor 1 was remarkably high, and the destruction of the tower and the inhibitor, the level difference, etc. were derived as important variables in all variables, showing that the results are almost the same as those of other models.

As a result, even when about 50 variables were reduced to 10, the accuracy of the model remained low and remained high. In addition, as a result of analyzing the important factors of winning and losing from each model, the factors related to the destruction of the structures were found to be overwhelmingly important, followed by the level and gold gain gap from the destruction of the structures, and the treatment of objects to help the destruction of the structures. This is because destroying the innermost Nexus of the enemy camp is the goal of the "League of Legends", and the destruction of various structures that keep the road to the Nexus leads directly to victory.

6. Case Study

Through various situation settings and case classification, we examined the changes in factors affecting win or loss.

6_1. Classification by game duration

After classifying the total game time of 15 to 25 minutes as a short game and a long game over 30 minutes, the prediction was conducted using a random forest and the variable importance was visualized. The reason for setting the short game's game duration to 15 minutes or more is that the game's surrender function is activated after 15 minutes. Games that are decided to win or lose 15 minutes ago may have ended the game quickly due to an overwhelming power difference, but it is highly likely that the game did not run smoothly due to system errors or teammates' escape.

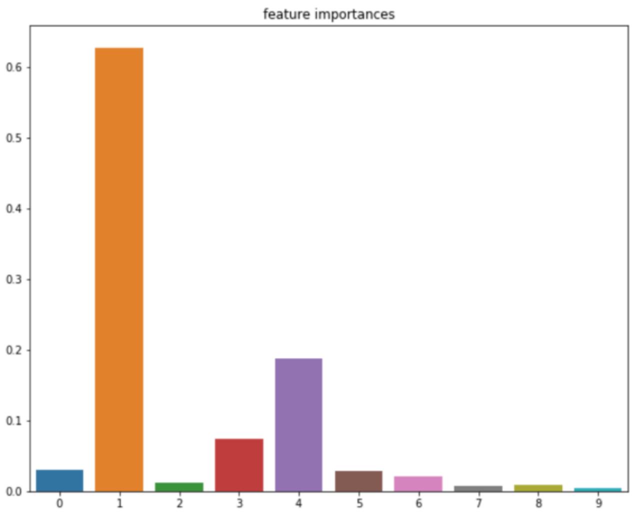
```
short_game, long_game = df.loc[(df['gameDuration'] > 60*15) & (df['gameDuration'] < 60*25)], df.loc[df['gameDuration'] > 60*30]
```

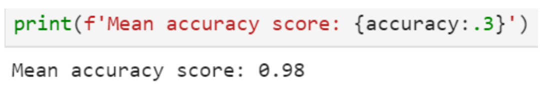
```
print(short_game.shape)
print(long_game.shape)
```

```
(8522, 51)
(5733, 51)
```

According to the above criteria, there are a total of 8522 short games and 5733 long games.

Looking at the results below, the prediction accuracy is 0.98 for a short game and the prediction accuracy is 0.916 for a long game. This is because the longer the game, the more anomalous factors occur in various parts, such as the treatment of objects, the destruction of structures, and the preservation of the initial leadership. However, there was no significant difference in the two cases in terms of the importance of variables. In both cases, the importance of factor 1 related to the destruction of structures and gold acquisition was overwhelmingly high, and factor 4 related to the effect from dragon treatment was the next important variable. However, in the long game, the importance of factor 2 related to the destruction of towers has been greatly increased. That because the League of Legends destroys the tower and approaches the enemy's camp, it is interpreted as a result that naturally appears as the number of towers destroyed increases as the game gets longer.

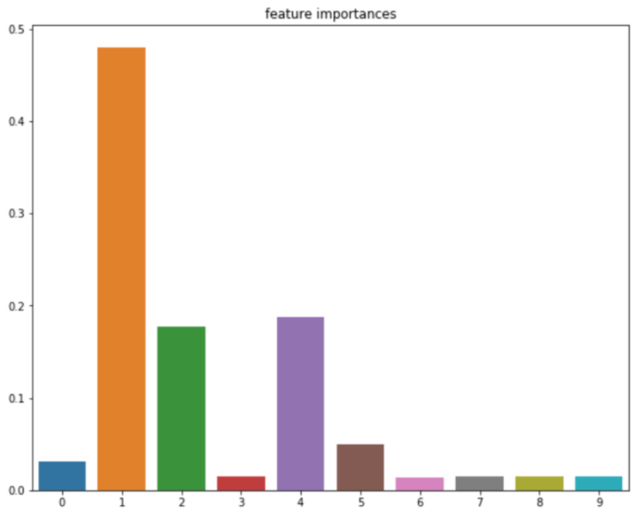


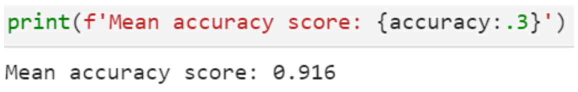


```
print(f'Mean accuracy score: {accuracy:.3}')
```

```
Mean accuracy score: 0.98
```

# short game





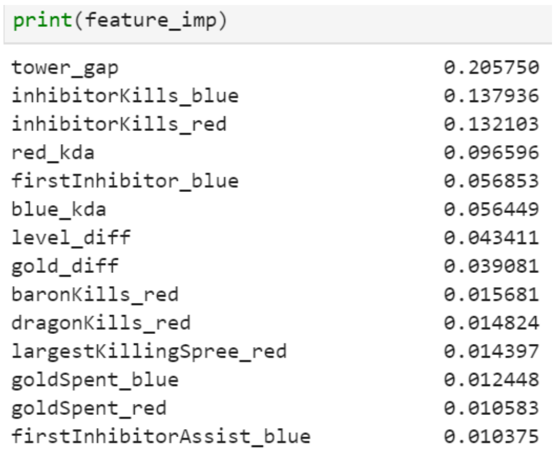
```
print(f'Mean accuracy score: {accuracy:.3}')
```

```
Mean accuracy score: 0.916
```

# long game

6_2. Assuming adverse conditions

Assume a variety of cases where the opponent takes the initial lead, such as when the enemy team destroys the first tower, kills the first dragon, or kills rift Herald for the first time that helps destroy the tower. At this time, if the opponent defeats the first dragon, the variable importance derived through random forest is as follows.



```
print(feature_imp)
```

```
tower_gap                          0.205750
inhibitorKills_blue                0.137936
inhibitorKills_red                 0.132103
red_kda                            0.096596
firstInhibitor_blue                0.056853
blue_kda                           0.056449
level_diff                         0.043411
gold_diff                          0.039081
baronKills_red                     0.015681
dragonKills_red                    0.014824
largestKillingSpree_red            0.014397
goldSpent_blue                     0.012448
goldSpent_red                      0.010583
firstInhibitorAssist_blue          0.010375
```

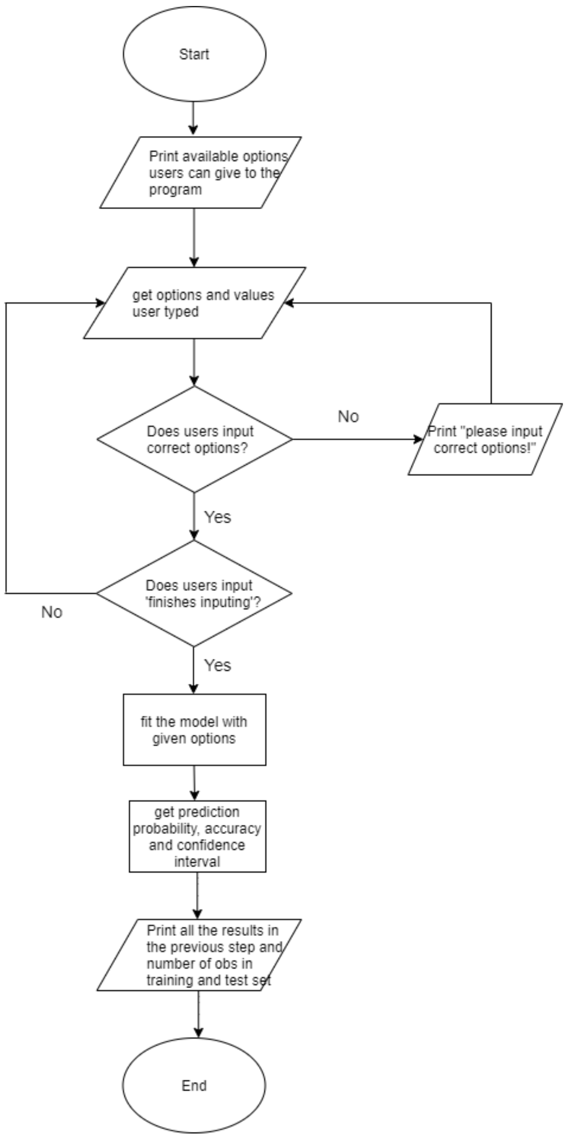
There were many variables related to the destruction of structures such as *tower_gap, firstInhibitor_blue, inhibitorKills_blue,* and *inhibitorKills_red*. Besides, it showed a similar pattern to important factors in general situations such as level difference, difference in gold gain, and buff caused by dragon. In addition, in all other adverse conditions mentioned above, the aspects of the important factors were almost the same.

As a result, it was concluded that under a variety of condition settings and total game time, the action the player would have to take to win or the elements to focus on remained unchanged. In other words, the factor determining the win or loss is fixed to some extent.

7. Win probability prediction program

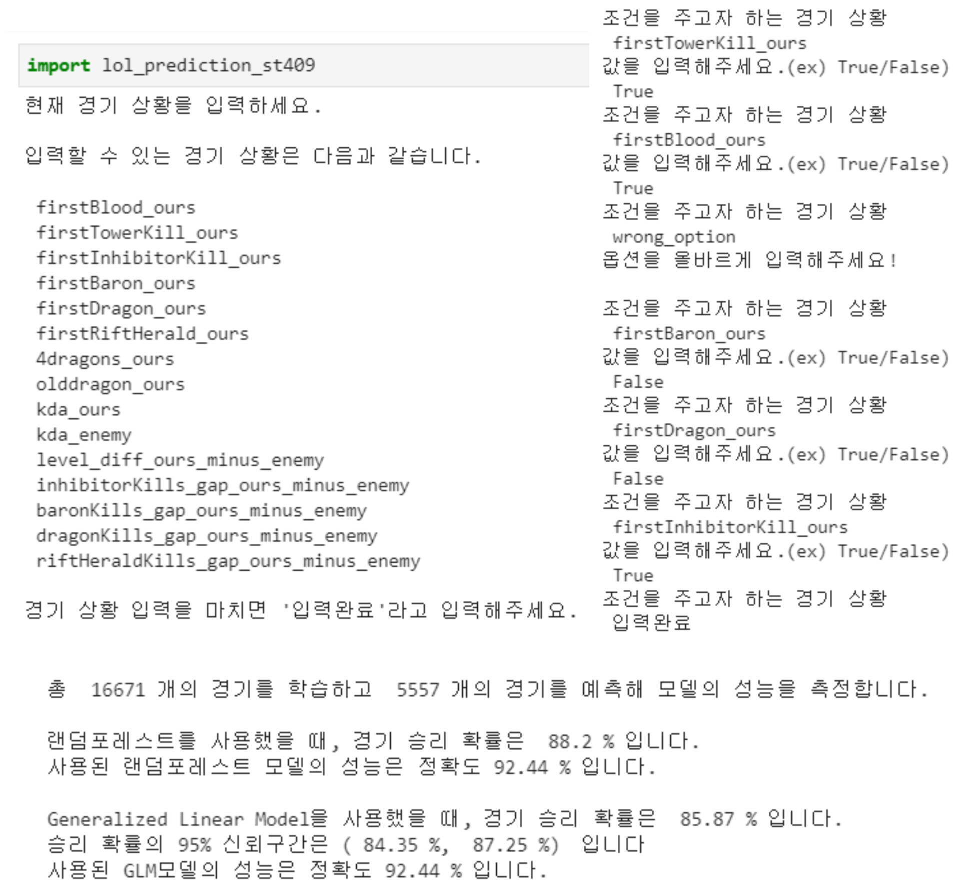
However, as we proceeded with the analysis, we questioned some of the variables that had a major impact on the win/loss. It is about whether it is meaningful to predict the factors of winning or losing using the indicators of the already finished game. A structure called a Inhibitor in a game is a structure located inside the enemy camp, close to the enemy team's Nexus, which must be destroyed to determine victory. Obviously, the game that destroyed the opposing team's Inhibitor will have a higher probability of winning, and the analysis result that the Inhibitor's destruction is an important factor is natural. In addition, the fifth dragon which is an object called 'elder dragon' or an object "Baron" have a strong effect that greatly aids the team's victory, and these objects are structured to be easily defeated by favorable teams who have almost won the match. Therefore, like the inhibitor, the treatment of this object will greatly increase the chances of winning and appear as an important variable. The above analysis may be effective in solving the problem of classifying wins and losses in the finished game, but it will not be a practical help for players who are playing directly in the game. We thought a different approach would be needed to achieve a **'practical win/loss prediction'** where players facing various anomalies and situations could effectively predict their win/loss.

Therefore, we implemented a program that predicts the probability of winning through competence of team members and the degree of gold supply from the early and the middle of the game. Through this program, players can set the situation at the beginning and the middle of each match and determine which factors can have a positive effect on victory, or the relative importance of specific factors. The user-friendly program, which allows players to find the best action to take under various assumptions, perfectly fits into the purpose of this report, 'Winning prediction and understanding of important factors'. The actual implementation process and results of the program are as follows.


Start
Print available options users can give to the program
get options and values user typed
Does users input correct options?
No
Print "please input correct options!"
Yes
Does users input 'finishes inputing'?
No
Yes
fit the model with given options
get prediction probability, accuracy and confidence interval
Print all the results in the previous step and number of obs in training and test set
End

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
import statsmodels.api as sm
import statsmodels.formula.api as smf
import distutils
from distutils import util

print('현재 경기 상황을 입력하세요.\n')
print('입력할 수 있는 경기 상황은 다음과 같습니다.\n')
print(' firstBlood_ours\n', 'firstTowerKill_ours\n', 'firstInhibitorKill_ours\n', 'firstBaron_ours\n',
      'firstDragon_ours\n',
      'firstRiftHerald_ours\n', '4dragons_ours\n', 'olddragon_ours\n', 'kda_ours\n', 'kda_enemy\n',
      'level_diff_ours_minus_enemy\n',
      'inhibitorKills_gap_ours_minus_enemy\n', 'baronKills_gap_ours_minus_enemy\n',
      'dragonKills_gap_ours_minus_enemy\n',
      'riftHeraldKills_gap_ours_minus_enemy\n')

print("경기 상황 입력을 마치면 '입력완료'라고 입력해주세요.\n")

option_dict = {}
total_option_ls = ['firstBlood_ours', 'firstTowerKill_ours', 'firstInhibitorKill_ours', 'firstBaron_ours',
                   'firstDragon_ours',
                   'firstRiftHerald_ours', '4dragons_ours', 'olddragon_ours', 'kda_ours', 'kda_enemy',
                   'level_diff_ours_minus_enemy',
                   'inhibitorKills_gap_ours_minus_enemy', 'baronKills_gap_ours_minus_enemy',
                   'dragonKills_gap_ours_minus_enemy',
                   'riftHeraldKills_gap_ours_minus_enemy']
```

```python
  while True:
      option = input('조건을 주고자 하는 경기 상황\n')

      if option == '입력완료':
          break

      if option in ['firstBlood_ours', 'firstTowerKill_ours', 'firstInhibitorKill_ours',
                    'firstBaron_ours', 'firstDragon_ours', 'firstRiftHerald_ours', '4dragons_ours', 'olddragon_ours']:

          state = input('값을 입력해주세요.(ex) True/False)\n')

          if state == '입력완료':
              break

          value = [bool(distutils.util.strtobool(state))]

      if option in ['kda_ours', 'kda_enemy', 'level_diff_ours_minus_enemy', 'inhibitorKills_gap_ours_minus_enemy',
                    'baronKills_gap_ours_minus_enemy',
                    'dragonKills_gap_ours_minus_enemy', 'riftHeraldKills_gap_ours_minus_enemy']:

          state = input('값을 입력해주세요. (ex)3)\n')
          value = [float(state)]

          if state == '입력완료':
              break


      elif option not in total_option_ls:
          print('옵션을 올바르게 입력해주세요!\n')
          continue

      option_dict[option] = value

  option_df = pd.DataFrame.from_dict(option_dict)

model_data = pd.read_csv('model_data.csv')
model_data = model_data[['win_blue'] + option_df.columns.tolist()]
train, test = train_test_split(model_data, test_size=0.25, random_state=1234)

X_train, y_train = train.drop('win_blue', axis=1), train['win_blue']
X_test, y_test = test.drop('win_blue', axis=1), test['win_blue']

rf = RandomForestClassifier(n_estimators=100, random_state=123456)
rf.fit(X_train, y_train)

predicted1 = rf.predict(X_test)
accuracy1 = accuracy_score(y_test, predicted1)

print('\n\n총 ', X_train.shape[0], '개의 경기를 학습하고 ', X_test.shape[0], '개의 경기를 예측해 모델의 성능을 측정합니다.')

print('\n랜덤포레스트를 사용했을 때, 경기 승리 확률은 ', round(rf.predict_proba(option_df)[:, 1][0] * 100, 2), '% 입니다.')
print(f'사용된 랜덤포레스트 모델의 성능은 정확도 {accuracy1 * 100:.4} % 입니다.')

all_variables = '+'.join(X_train.columns)
formula = 'win_blue~' + all_variables
glm = smf.glm(formula=formula, data=train, family=sm.families.Binomial()).fit()
glm.summary()

predicted2 = glm.predict(test)
accuracy2 = accuracy_score(y_test, np.where(predicted2 > 0.5, 1, 0))
ci_lower = glm.get_prediction(option_df).summary_frame()[['mean_ci_lower', 'mean_ci_upper']]['mean_ci_lower'][0]
ci_upper = glm.get_prediction(option_df).summary_frame()[['mean_ci_lower', 'mean_ci_upper']]['mean_ci_upper'][0]

print('\nGeneralized Linear Model을 사용했을 때, 경기 승리 확률은 ', round(glm.predict(option_df)[0] * 100, 2), '% 입니다.')
print('승리 확률의 95% 신뢰구간은 (', round(ci_lower * 100, 2), '%, ', round(ci_upper * 100, 2), '%)', ' 입니다')
print(f'사용된 GLM모델의 성능은 정확도 {accuracy2 * 100:.4} % 입니다.')
```



```
import lol_prediction_st409
현재 경기 상황을 입력하세요.

입력할 수 있는 경기 상황은 다음과 같습니다.

 firstBlood_ours
 firstTowerKill_ours
 firstInhibitorKill_ours
 firstBaron_ours
 firstDragon_ours
 firstRiftHerald_ours
 4dragons_ours
 olddragon_ours
 kda_ours
 kda_enemy
 level_diff_ours_minus_enemy
 inhibitorKills_gap_ours_minus_enemy
 baronKills_gap_ours_minus_enemy
 dragonKills_gap_ours_minus_enemy
 riftHeraldKills_gap_ours_minus_enemy

경기 상황 입력을 마치면 '입력완료'라고 입력해주세요.
```

```
조건을 주고자 하는 경기 상황
 firstTowerKill_ours
값을 입력해주세요.(ex) True/False)
 True
조건을 주고자 하는 경기 상황
 firstBlood_ours
값을 입력해주세요.(ex) True/False)
 True
조건을 주고자 하는 경기 상황
 wrong_option
옵션을 올바르게 입력해주세요!

조건을 주고자 하는 경기 상황
 firstBaron_ours
값을 입력해주세요.(ex) True/False)
 False
조건을 주고자 하는 경기 상황
 firstDragon_ours
값을 입력해주세요.(ex) True/False)
 False
조건을 주고자 하는 경기 상황
 firstInhibitorKill_ours
값을 입력해주세요.(ex) True/False)
 True
조건을 주고자 하는 경기 상황
 입력완료
```

```
 총  16671 개의 경기를 학습하고  5557 개의 경기를 예측해 모델의 성능을 측정합니다.

 랜덤포레스트를 사용했을 때, 경기 승리 확률은  88.2 % 입니다.
 사용된 랜덤포레스트 모델의 성능은 정확도 92.44 % 입니다.

 Generalized Linear Model을 사용했을 때, 경기 승리 확률은  85.87 % 입니다.
 승리 확률의 95% 신뢰구간은 ( 84.35 %,  87.25 %)  입니다
 사용된 GLM모델의 성능은 정확도 92.44 % 입니다.
```

8. Conclusion

As mentioned at the beginning, this report examined the actual importance of in-game indicators that are considered to be common sense through statistical methods and the implementation of machine learning models. Through the rules in the game, the variables were deleted, combined, and created, and it was confirmed that the important factors that greatly influenced the game's victory or defeat were fixed to some extent due to the existence of an exact purpose for victory. During the analysis process, it was possible to identify a high correlation between variables and reduce the variables to implement a predictive model with a high degree of generalization and high accuracy. As a result of examining the changes in important variables assuming various situations, the actions to be taken and factors to be taken by the player did not change significantly depending on the conditions. While analyzing from the player's point of view through various case classifications, it was found that there are limitations in predicting wins and losses and analyzing factors by collected data. In order to implement an effective model from the perspective of the players participating in the game, it was necessary to overcome the limitation of analysis using indicators after the game ended. In order to overcome the limitations, the indicators of the middle and early games were used, and the implementation of a highly practical win prediction program from the player's point of view can be considered as the biggest harvest of this report.