1. **Source Code**
2. **import** **numpy** **as** **np**
3. **import** **pandas** **as** **pd**
4. **from** **sklearn.preprocessing** **import** LabelEncoder
5. **from** **sklearn.preprocessing** **import** StandardScaler
6. **from** **sklearn.model\_selection** **import** train\_test\_split
7. **from** **sklearn.tree** **import** DecisionTreeClassifier
8. **from** **sklearn.ensemble** **import** RandomForestClassifier, BaggingClassifier
9. **from** **sklearn.linear\_model** **import** LinearRegression, LogisticRegression
10. **from** **sklearn.model\_selection** **import** cross\_val\_score
11. **from** **sklearn.metrics** **import** confusion\_matrix, accuracy\_score
12. **from** **sklearn.metrics** **import** classification\_report
13. **import** **matplotlib.pyplot** **as** **plt**
14. **import** **seaborn** **as** **sns**
15. *# Load data with using columns*
16. original\_data = pd.read\_csv("match\_winner\_data\_version1.csv",
17. usecols=['win', 'firstBlood', 'firstTower', 'firstInhibitor', 'towerKills',
18. 'inhibitorKills',
19. 'baronKills', 'dragonKills', 'riftHeraldKills', 'gameId'])
20. **print**(original\_data.head())
21. *# Categorical Feature: win, firstBlood, firstTower, firstInhibitor*
22. *# Numerical Feature: towerKills, inhibitorKills, baronKills, dragonKills, riftHeraldKills, gameId*
23. *# Missing Data detection*
24. *# Find columns with missing value*
25. columnMD = original\_data.columns[original\_data.isnull().any()]
26. **print**(columnMD)
27. *# Among the missing values, numerical data: towerKills, inhibitorKills, baronKills, dragonKills*
28. *# Fill with mean (Rounds mean to nearest integer)*
29. original\_data['towerKills'].fillna((original\_data['towerKills'].mean().round(0)), inplace=True)
30. original\_data['inhibitorKills'].fillna((original\_data['inhibitorKills'].mean().round(0)), inplace=True)
31. original\_data['baronKills'].fillna((original\_data['baronKills'].mean().round(0)), inplace=True)
32. original\_data['dragonKills'].fillna((original\_data['dragonKills'].mean().round(0)), inplace=True)
33. *# Finding the wrong game that is not divided by win or lose and any others (Check using towerkills)*
34. **for** i **in** range(len(original\_data)):
35. **print**(i)
36. **if** original\_data['towerKills'][i] < 5.0:
37. original\_data.drop(i, inplace=True)
38. original\_data.to\_csv("original\_data\_remove\_wrongGame.csv", mode='w') *# 1차...*
39. second\_data = pd.read\_csv("original\_data\_remove\_wrongGame.csv")
40. second\_data.columns = ['index', 'win', 'firstBlood', 'firstTower', 'firstInhibitor', 'towerKills',
41. 'inhibitorKills',
42. 'baronKills', 'dragonKills', 'riftHeraldKills', 'gameId']
43. **print**(second\_data.head())
44. *# Find rows with missing value*
45. rowMD = second\_data[second\_data.isnull().any(axis=1)]
46. **def** findCategoricalMD(fIndex, fgameId):
47. tempData = pd.read\_csv("match\_loser\_data\_version1.csv",
48. usecols=['firstBlood', 'firstTower', 'firstInhibitor', 'gameId'
49. ])
50. **for** i **in** range(len(tempData)):
51. **if** tempData['gameId'][i] == fgameId:
52. **print**(tempData.loc[[i], :])
53. firstBlood = tempData['firstBlood'][i]
54. firstTower = tempData['firstTower'][i]
55. firstInhibitor = tempData['firstInhibitor'][i]
56. *# nan일 경우 값 변경*
57. *# loser data T --> F // F --> T*
58. **if** second\_data.loc[fIndex, 'firstBlood']:
59. **if** firstBlood:
60. firstBlood = False
61. **elif** **not** firstBlood:
62. firstBlood = True
63. second\_data['firstBlood'] = second\_data['firstBlood'].replace(np.nan, firstBlood)
64. **if** second\_data.loc[fIndex, 'firstTower']:
65. **if** firstTower:
66. firstTower = False
67. **elif** **not** firstTower:
68. firstTower = True
69. second\_data['firstTower'] = second\_data['firstTower'].replace(np.nan, firstTower)
70. **if** second\_data.loc[fIndex, 'firstInhibitor']:
71. **if** firstInhibitor:
72. firstInhibitor = False
73. **elif** **not** firstInhibitor:
74. firstInhibitor = True
75. second\_data['firstInhibitor'] = second\_data['firstInhibitor'].replace(np.nan, firstInhibitor)
76. **for** i **in** range(len(second\_data)):
77. **try**:
78. **if** rowMD['index'][i]:
79. findCategoricalMD(i, rowMD['gameId'][i])
80. **except** **KeyError**:
81. **continue**
82. second\_data.to\_csv("fix\_missing\_data.csv", mode='w') *# 2차....*
83. win\_data = pd.read\_csv("fix\_missing\_data.csv",
84. usecols=['win', 'firstBlood', 'firstTower', 'firstInhibitor', 'towerKills', 'inhibitorKills',
85. 'baronKills', 'dragonKills', 'riftHeraldKills', 'gameId'])
86. lose\_data = pd.read\_csv("match\_loser\_data\_version1.csv",
87. usecols=['win', 'firstBlood', 'firstTower', 'firstInhibitor', 'towerKills', 'inhibitorKills',
88. 'baronKills', 'dragonKills', 'riftHeraldKills', 'gameId'])
89. merged = pd.merge(win\_data, lose\_data, on='gameId')
90. merged.to\_csv("win\_lose\_final\_data.csv", mode='w') *# 3차....*
91. data = pd.read\_csv("win\_lose\_final\_data.csv",
92. usecols=['win\_x', 'firstBlood\_x', 'firstTower\_x', 'firstInhibitor\_x', 'towerKills\_x',
93. 'inhibitorKills\_x',
94. 'baronKills\_x', 'dragonKills\_x', 'riftHeraldKills\_x', 'win\_y', 'firstBlood\_y',
95. 'firstTower\_y', 'firstInhibitor\_y', 'towerKills\_y', 'inhibitorKills\_y',
96. 'baronKills\_y', 'dragonKills\_y', 'riftHeraldKills\_y'])
97. **print**(data.head())
98. columnMD = data.columns[data.isnull().any()]
99. **print**(columnMD)
100. data['win\_y'] = data['win\_y'].fillna('Fail')
101. win\_data = pd.DataFrame(columns=['win', 'firstBlood', 'firstTower', 'firstInhibitor', 'towerKills',
102. 'inhibitorKills',
103. 'baronKills', 'dragonKills', 'riftHeraldKills'])
104. lose\_data = pd.DataFrame(columns=['win', 'firstBlood', 'firstTower', 'firstInhibitor', 'towerKills',
105. 'inhibitorKills',
106. 'baronKills', 'dragonKills', 'riftHeraldKills'])
107. win\_data['win'] = data['win\_x']
108. win\_data['win'] = win\_data['win'].replace('Win', 1)
109. win\_data['firstBlood'] = data['firstBlood\_x']
110. win\_data['firstTower'] = data['firstTower\_x']
111. win\_data['firstInhibitor'] = data['firstInhibitor\_x']
112. win\_data['towerKills'] = data['towerKills\_x']
113. win\_data['inhibitorKills'] = data['inhibitorKills\_x']
114. win\_data['baronKills'] = data['baronKills\_x']
115. win\_data['dragonKills'] = data['dragonKills\_x']
116. win\_data['riftHeraldKills'] = data['riftHeraldKills\_x']
117. lose\_data['win'] = data['win\_y']
118. lose\_data['win'] = lose\_data['win'].replace('Fail', 0)
119. lose\_data['firstBlood'] = data['firstBlood\_y']
120. lose\_data['firstTower'] = data['firstTower\_y']
121. lose\_data['firstInhibitor'] = data['firstInhibitor\_y']
122. lose\_data['towerKills'] = data['towerKills\_y']
123. lose\_data['inhibitorKills'] = data['inhibitorKills\_y']
124. lose\_data['baronKills'] = data['baronKills\_y']
125. lose\_data['dragonKills'] = data['dragonKills\_y']
126. lose\_data['riftHeraldKills'] = data['riftHeraldKills\_y']
127. final\_data = pd.concat([win\_data, lose\_data], ignore\_index=True)
128. final\_data.to\_csv("Final\_data.csv", mode='w') *# 4차....*
129. *# ---------------Complete Preprocessing--------------------*
130. *# ---------------------------------------------------------*
131. *# Read csv file*
132. data = pd.read\_csv('Final\_data.csv')
133. data = df.loc[:, ~df.columns.str.contains('^Unnamed')]
134. **print**(data.head())
135. # Using Pearson Correlation
136. plt.figure(figsize=(12, 10))
137. cor = data.corr()
138. sns.heatmap(cor, annot=True, cmap=plt.cm.Reds)
139. plt.show()
140. # Correlation with output variable
141. cor\_target = abs(cor[“win”])
142. # Selectinng highly correlated features
143. relevant\_features = cor\_target[cor\_target > 0.4]
144. print(relevant\_features)
145. # win: 1.000000 / firstTower: 0.468613 / firstInhibitor: 0.784156 / towerKills: 0.833098 / inhibitorKills: 0.673570 / baronKills: 0.419577 / dragonKills: 0.514895
146. df = pd.DataFrame(data, columns=['win', 'firstTower', 'firstInhibitor', 'towerKills', 'inhibitorKills', 'baronKills',
147. 'dragonKills'])
148. print(df.head)
149. *# Probability of win according to a feature that has a numeric data type*
150. **def** probStatus(dataset, group\_by):
151. df = pd.crosstab(index=dataset[group\_by], columns=dataset.win).reset\_index()
152. df['probWin'] = df[1] / (df[1] + df[0])
153. **print**(df)
154. **return** df[[group\_by, 'probWin']]
155. *# Visualization: Win ratio changes with respect to towerKills changes.*
156. sns.lmplot(data=probStatus(df, 'towerKills'), x='towerKills', y='probWin', fit\_reg=False)
157. plt.xlim(0, 11)
158. plt.title('Win ratio on tower kills')
159. plt.show()
160. *# Visualization: Win ratio changes with respect to dragonKills changes.*
161. sns.lmplot(data=probStatus(df, 'dragonKills'), x='dragonKills', y='probWin', fit\_reg=False)
162. plt.xlim(0, 7)
163. plt.title('Win ratio on dragon kills')
164. plt.show()
165. *# Find the categorical columns*
166. **def** categoricalColumns(data):
167. cateCols = []
168. **for** i **in** data.columns:
169. **if** data[i].dtypes == bool:
170. cateCols.append(i)
171. **return** cateCols
172. *# LabelEncoder (Categorical -> Numeric)*
173. **def** labelEncoder(data, cateCols):
174. label = LabelEncoder()
175. **for** i **in** cateCols:
176. label.fit\_transform(list(data[i].values))
177. data[i] = label.transform(list(data[i].values))
178. *# Use a LabelEncoder to encode Categorical values to numeric values.*
179. labelEncoder(df, categoricalColumns(df))
180. X = df.drop(['win'], axis=1)
181. y = df['win']
182. *# Holdout*
183. X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=1)
184. *# StandardScaler*
185. scaler = StandardScaler()
186. X[['towerKills', 'inhibitorKills', 'baronKills', 'dragonKills', 'riftHeraldKills']] = scaler.fit\_transform(
187. X[['towerKills', 'inhibitorKills', 'baronKills', 'dragonKills', 'riftHeraldKills']])
188. *# kdeplot - Before, After Scaling*
189. X = pd.DataFrame(X, columns=['towerKills', 'inhibitorKills', 'baronKills', 'dragonKills', 'riftHeraldKills'])
190. fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(10, 10))
191. ax1.set\_title('Before Scaling')
192. sns.kdeplot(X['towerKills'], ax=ax1)
193. sns.kdeplot(X['inhibitorKills'], ax=ax1)
194. sns.kdeplot(X['baronKills'], ax=ax1)
195. sns.kdeplot(X['dragonKills'], ax=ax1)
196. sns.kdeplot(X['riftHeraldKills'], ax=ax1)
197. ax2.set\_title('After Standard Scaler')
198. sns.kdeplot(df['towerKills'], ax=ax2)
199. sns.kdeplot(df['inhibitorKills'], ax=ax2)
200. sns.kdeplot(df['baronKills'], ax=ax2)
201. sns.kdeplot(df['dragonKills'], ax=ax2)
202. sns.kdeplot(df['riftHeraldKills'], ax=ax2)
203. plt.show()
204. *# Draw result*
205. **def** showResult(confusionMatrix, name):
206. target\_names = ['win', 'lose']
207. fig, ax = plt.subplots()
208. tick\_marks = np.arange(len(target\_names))
209. plt.xticks(tick\_marks, target\_names)
210. plt.yticks(tick\_marks, target\_names)
211. sns.heatmap(pd.DataFrame(confusionMatrix), annot=True, cmap='YlGnBu', fmt='g')
212. ax.xaxis.set\_label\_position("top")
213. plt.tight\_layout()
214. plt.title("Confusion matrix : {0}".format(name), y=1.1)
215. plt.xlabel("Predict label")
216. plt.ylabel("Actual label")
217. plt.show()
218. **print**("Classification report on {0}".format(name))
219. **print**(classification\_report(y\_test, y\_pred\_LG, target\_names=target\_names))
220. *################### Analysis & Evaluation ####################*
221. *# LogisticRegression*
222. **print**("LogisticRegression")
223. model\_LG = LogisticRegression()
224. model\_LG.fit(X\_train, y\_train)
225. y\_pred\_LG = model\_LG.predict(X\_test)
226. **print**('Accuracy :', accuracy\_score(y\_test, y\_pred\_LG))
227. *# Display confusin\_matrix*
228. cm\_LG = confusion\_matrix(y\_test, y\_pred\_LG)
229. **print**(cm\_LG)
230. *# Draw heatmap*
231. showResult(cm\_LG, "LogisticRegression")
232. *# Use 10-fold cross validation*
233. cvs\_LG = cross\_val\_score(model\_LG, X, y, scoring=None, cv=10)
234. **print**("Accuracy score using 10-fold cv:", (np.mean(cvs\_LG)))
235. **print**("-" \* 60)
236. *#################################################################*
237. *# DecisionTreeClassifer : entropy-based*
238. **print**("DecisionTreeClassifer")
239. model\_DT = DecisionTreeClassifier(criterion='entropy')
240. model\_DT.fit(X\_train, y\_train)
241. y\_pred\_DT = model\_DT.predict(X\_test)
242. **print**('Accuracy :', accuracy\_score(y\_test, y\_pred\_DT))
243. *# Display confusion\_matrix*
244. cm\_DT = confusion\_matrix(y\_test, y\_pred\_DT)
245. **print**(cm\_DT)
246. *# Draw heatmap*
247. showResult(cm\_DT, "DecisionTree")
248. *# Use 10-fold cross validation*
249. cvs\_DT = cross\_val\_score(model\_DT, X, y, scoring=None, cv=10)
250. **print**("Accuracy score using 10-fold cv:", (np.mean(cvs\_DT)))
251. **print**("-" \* 60)
252. **print**("-" \* 60)
253. *# Ensemble Learning - BaggingClassifier, RandomForestClassifier*
254. **def** essemble(name, n\_estimators):
255. **if** name == "BaggingClassifier":
256. essembleModel = BaggingClassifier(base\_estimator=model\_DT,
257. n\_estimators=n\_estimators,
258. n\_jobs=-1,
259. max\_samples=1.0,
260. max\_features=1.0,
261. random\_state=1).fit(X\_train, y\_train)
262. **else**:
263. essembleModel = RandomForestClassifier(n\_estimators=n\_estimators, max\_leaf\_nodes=16, n\_jobs=-1, random\_state=42).fit(X\_train, y\_train)
265. y\_pred = essembleModel.predict(X\_test)
266. accuracyResult.append(accuracy\_score(y\_test, y\_pred))
267. cm = confusion\_matrix(y\_test, y\_pred)
268. showResult(cm, name)
269. essembleName = ["BaggingClassifier", "RandomForestClassifier"]
270. n\_estimators = [5, 10, 20, 50, 100]
271. **for** i **in** essembleName:
272. accuracyResult=[]
273. **for** j **in** n\_estimators:
274. essemble(i, j)
275. **print**("{0} Max Score : ".format(i), max(accuracyResult))
276. **print**("-" \* 60)
277. **Objective Setting**

* People are always wondering if they will win or lose while watching the game.
* Classify the wins or losses with the events in the game.

**Used Data**

* Kaggle : https://www.kaggle.com/gyejr95/league-of-legendslol-ranked-games-2020-ver1
* Year 2020
* challenger, grandmaster, master game data (Korea, 2020)
* Win/Lose

1. **Data Curation**

**Dataset**

* match\_winner\_data\_version1
* match\_loser\_data\_version1

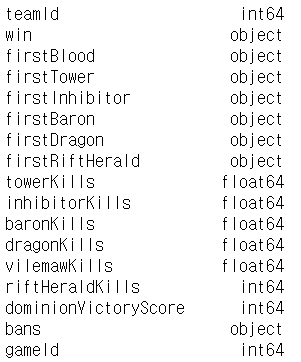
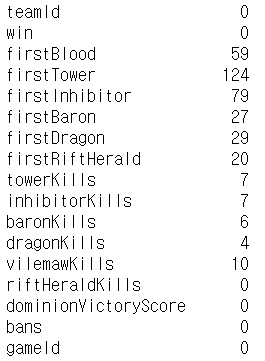
**Size**

* each 108,829 rows
* each 17 columns

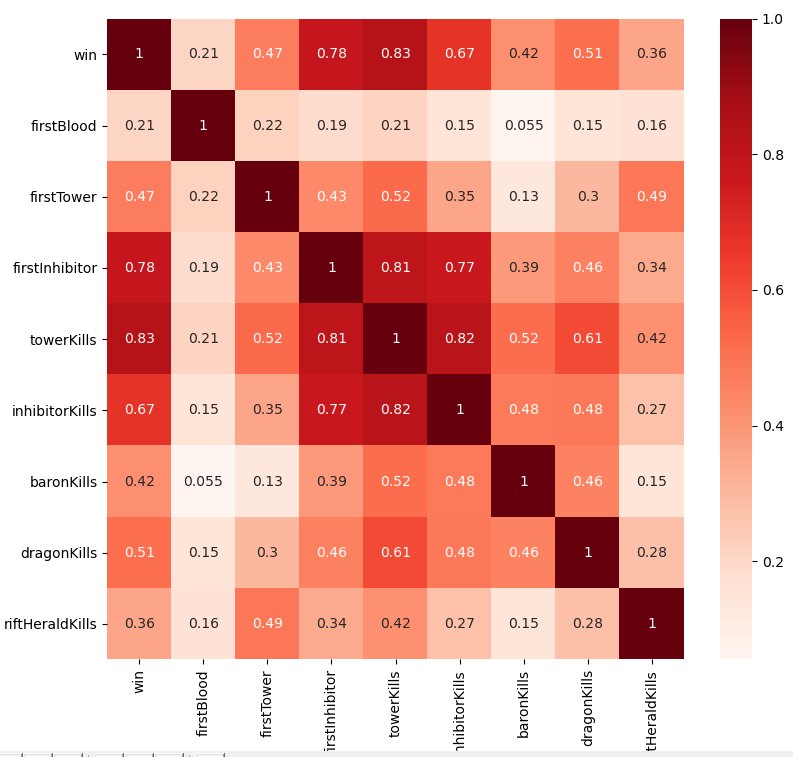
**Features**

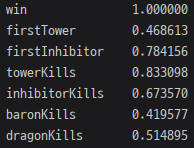


**<Missing Data>** **<Categorical Data>**

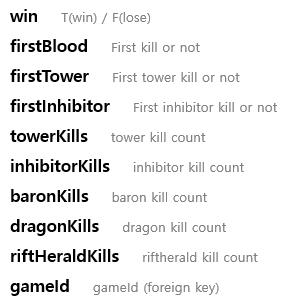
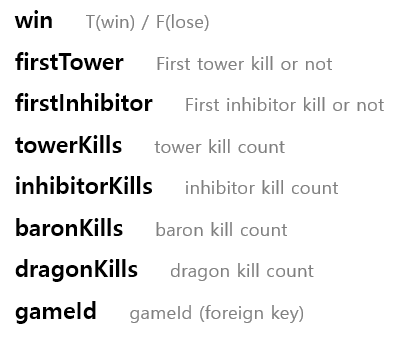


**Additional work (Using Pearson Correlation)**

****

****

**Used Features** (**17** cols 🡪 **10** cols 🡪 8 cols)

****

****

1. **Data Inspection**

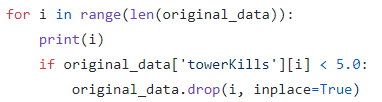
* Contains label results Win / Lose
* Sufficient number of features, rows 108,829x17
* Dirty data Missing data
* Categorical data ex) firstTower

****

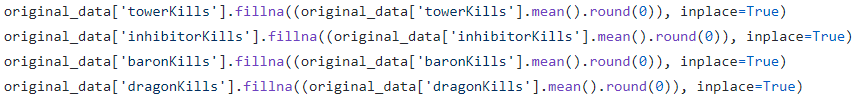
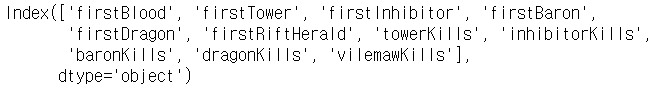
🡪 **Suitability**

1. **Data Preprocessing**
2. **Unusable Data Cleaning**

Finding the wrong game that is not divided by win or lose and any others. (Check using towerkills)

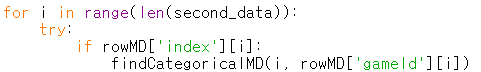
****

(Winner Data) **108,829** rows 🡪 **78,163** rows

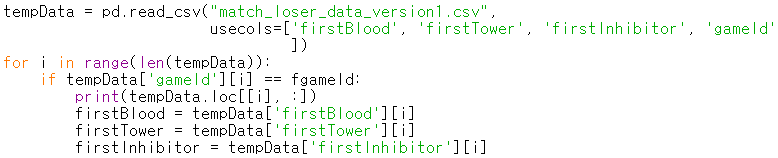
1. ******Missing Data Cleaning – Numerical Data**

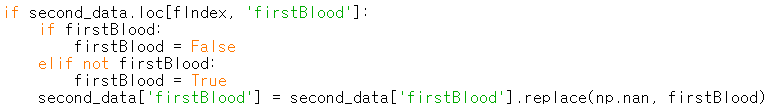
**Fill with mean** (Rounds mean to nearest integer)

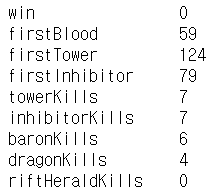
1. **Missing Data Cleaning – Categorical Data**
2. Find the ‘index’ with missing value and the ‘gameId’ value of the row in ‘original\_data\_remove\_wrongGame.csv’.



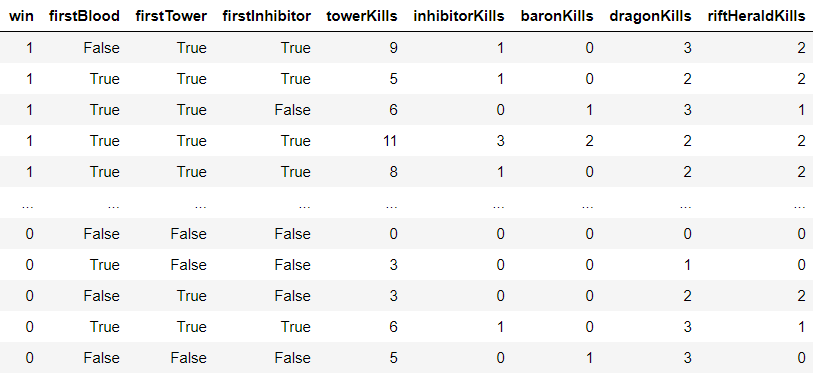
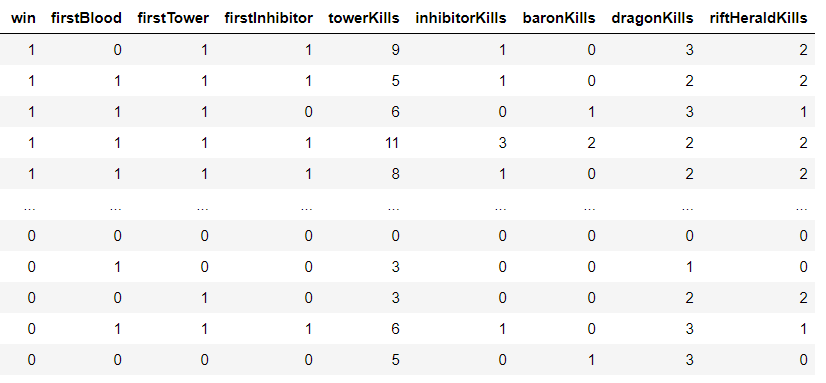
1. Find **empty data** using ‘gameId’ among existing loser data.



1. Last, since the data are **Boolean data** 🡪 The **opposite value** of the found value is entered.
2. **Missing Data Cleaning – Result**



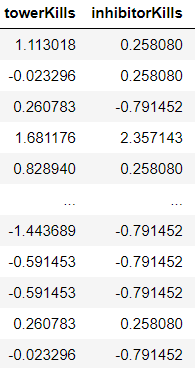
1. **Label Encoder**

****

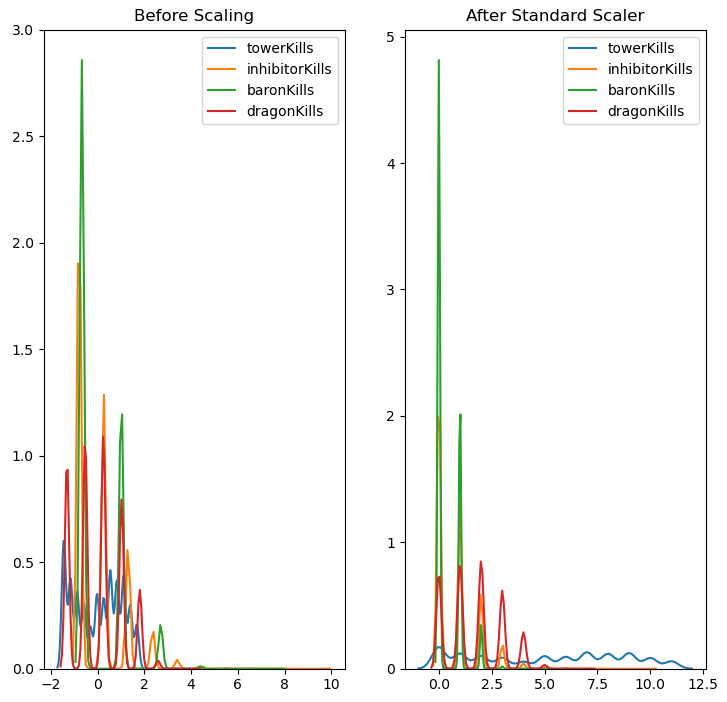
**Numeric**

**Categorical**

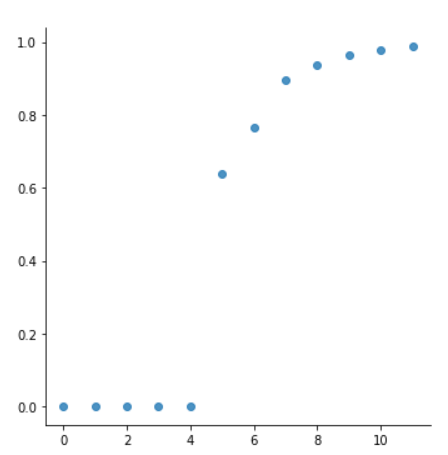
1. **Standard Scaler**

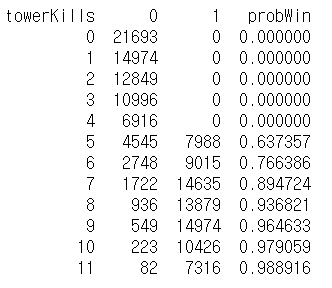
****

**\* Before vs After Standard Scaler**

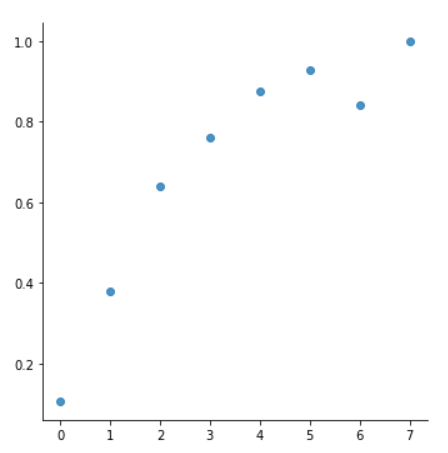
****

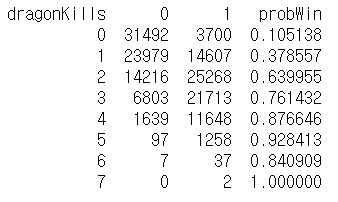
1. **Visualization**
2. **Probability victory based on the number of tower kill**

****



1. **Probability victory based on the number of dragon kill**





1. **Data Analysis**

# separate target from data

X = df.drop(['win'], axis=1)

y = df['win']

# Logistic Regression

model\_LG= LogisticRegression()

# DecisionTreeClassifer : entropy-based

model\_DT = DecisionTreeClassifier(criterion='entropy')

# Holdout: Dataset split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=1)

# train them on same training\_set

model\_LG.fit(X\_train, y\_train)

model\_DT.fit(X\_train, y\_train)

# predict show/no-show using same test\_set

y\_pred\_LG = model\_LG.predict(X\_test)

y\_pred\_DT = model\_DT.predict(X\_test)

# Ensemble Learning - BaggingClassifier, RandomForestClassifier

BaggingClassifier(base\_estimator=model\_DT,

n\_estimators=n\_estimators,

n\_jobs=-1,

max\_samples=1.0,

max\_features=1.0,

random\_state=1).fit(X\_train, y\_train)

RandomForestClassifier(n\_estimators=n\_estimators,

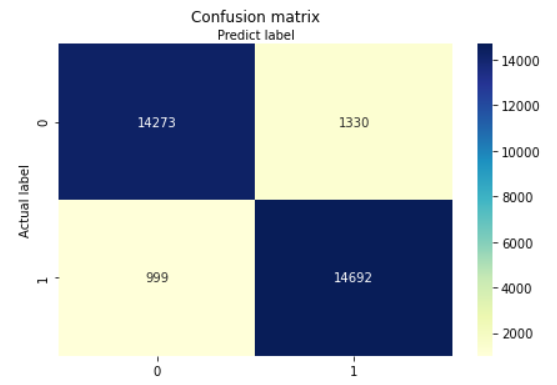
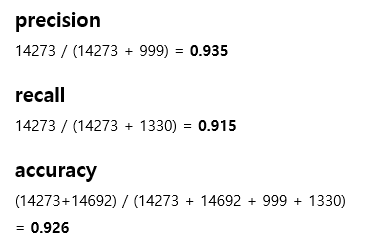
max\_leaf\_nodes=16,

n\_jobs=-1,

random\_state=42).fit(X\_train, y\_train)

n\_estimators = [5, 10, 20, 50, 100]

1. **Evaluation**
2. **Logistic Regression**

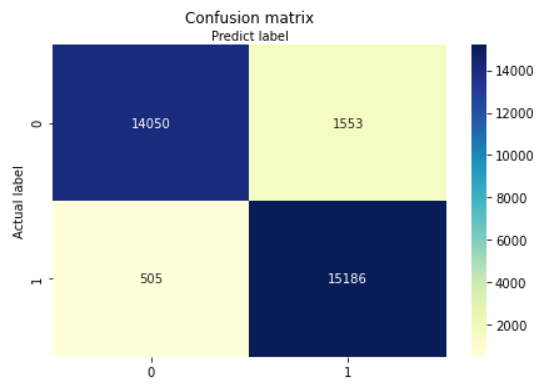
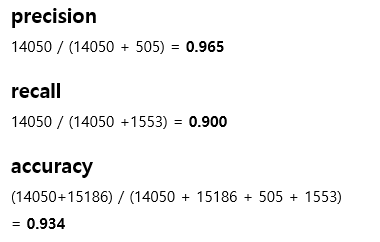
****

**\* 10-Fold Cross Validation**

cvs\_LG = cross\_val\_score(model\_LG, X, y, scoring=None, cv=10)

* ****

1. **Decision Tree**

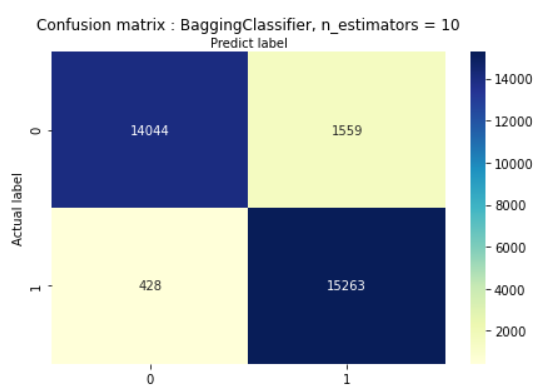
****

**\* 10-Fold Cross Validation**

cvs\_DT = cross\_val\_score(model\_DT, X, y, scoring=None, cv=10)

* ****

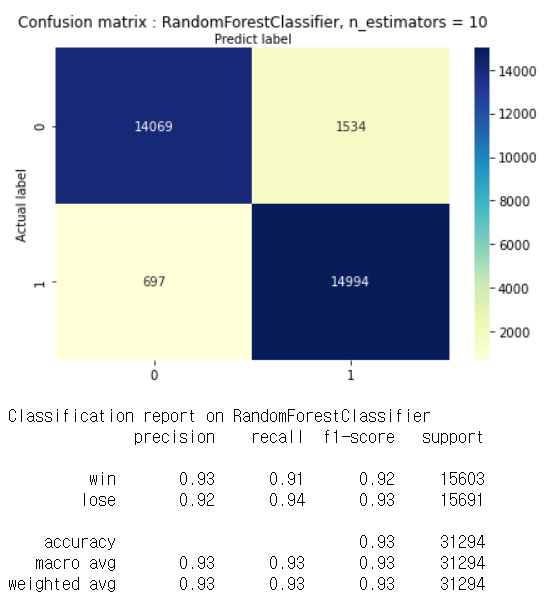
1. **Ensemble Learning**
2. Bagging Classifier

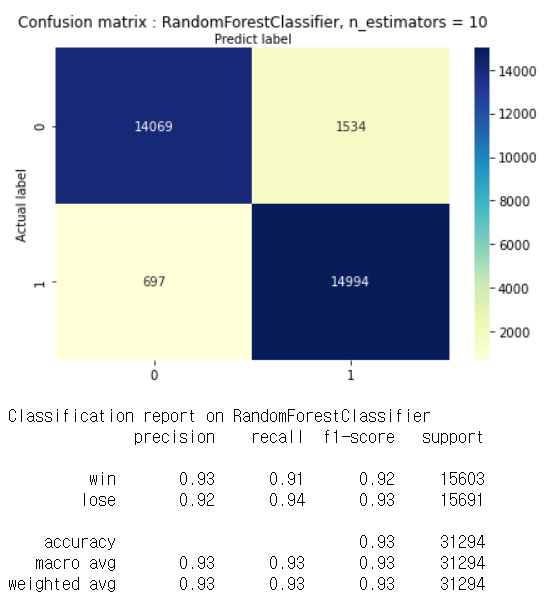






1. RandomForest Classifier

****

****

****