# project

December 12, 2022

[2]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import pickle

winners_df = pd.read_csv("./winners.csv")
losers_df = pd.read_csv("./losers.csv")
```

[3]:
```
total_matches_df = pd.concat([winners_df, losers_df])
```

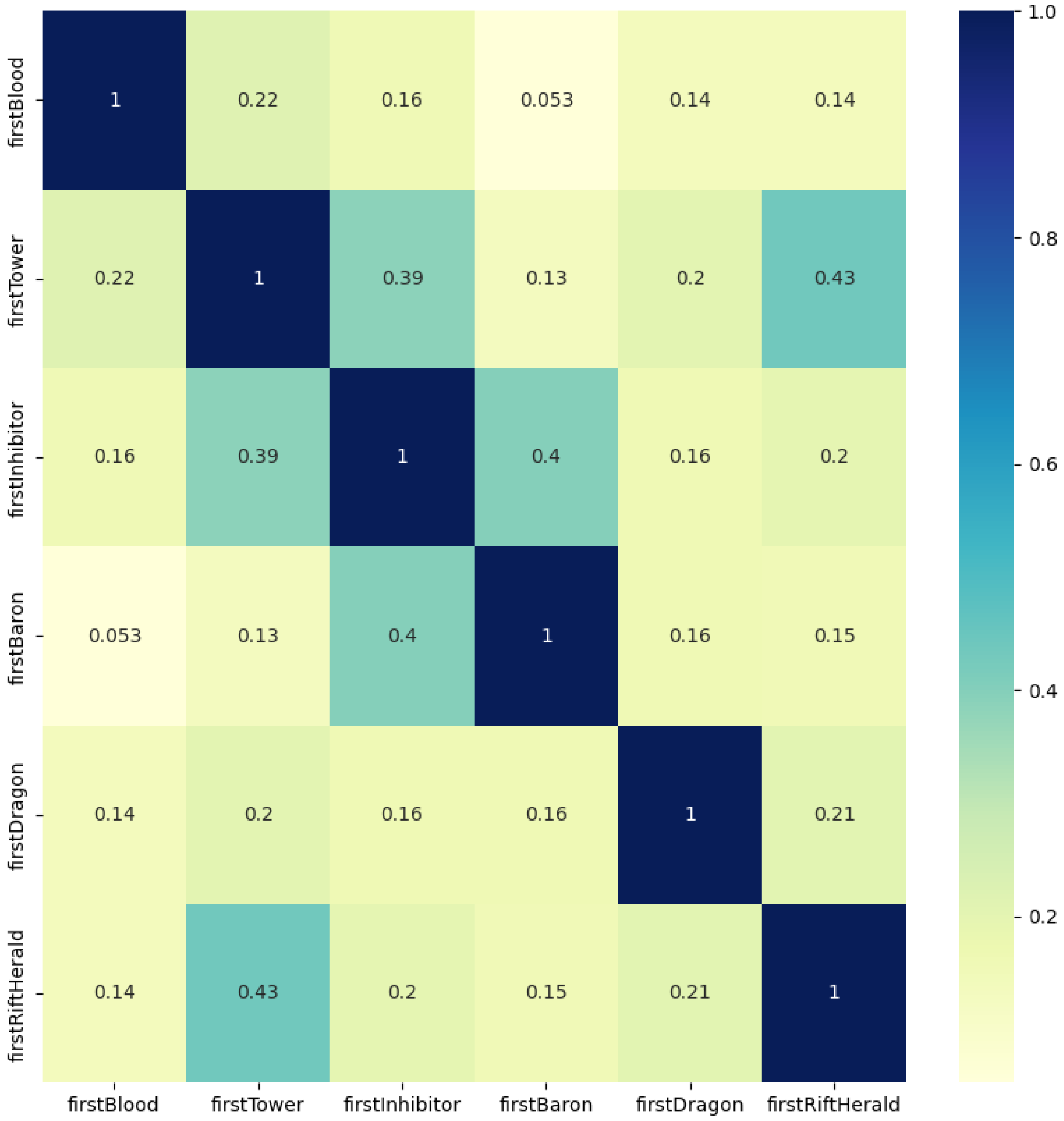
[4]:
```
total_matches_df.head()
```

[4]:
```
   Unnamed: 0  teamId  win  firstBlood  firstTower  firstInhibitor  \
0           0     200  Win       False        True            True
1           1     100  Win       False       False           False
2           2     200  Win        True        True            True
3           3     200  Win        True        True           False
4           4     100  Win        True        True            True

   firstBaron  firstDragon  firstRiftHerald  towerKills  inhibitorKills  \
0       False         True             True           9               1
1       False         True             True           4               0
2       False         True             True           5               1
3       False        False             True           6               0
4        True         True             True          11               3

   baronKills  dragonKills  vilemawKills  riftHeraldKills  \
0           0            3             0                2
1           0            2             0                2
2           0            2             0                2
3           1            3             0                1
4           2            2             0                2

   dominionVictoryScore                                                  bans  \
0                     0  [{'championId': 523, 'pickTurn': 6}, {'champio…
1                     0  [{'championId': 523, 'pickTurn': 1}, {'champio…
2                     0  [{'championId': 350, 'pickTurn': 6}, {'champio…
3                     0  [{'championId': 81, 'pickTurn': 6}, {'champion…
```

```
4                     0  [{'championId': 30, 'pickTurn': 1}, {'champion…

         gameId
0  4.247263e+09
1  4.247156e+09
2  4.243963e+09
3  4.241678e+09
4  4.241539e+09
```

[57]:

```
import seaborn as sns

total_matches_df = total_matches_df[["win", "firstBlood", "firstTower",␣
 ↪"firstInhibitor", "firstBaron", "firstDragon", "firstRiftHerald"]]
total_matches_df.dropna(inplace = True)
plt.figure(figsize = (10, 10))
sns.heatmap(total_matches_df.corr(numeric_only = True), cmap = "YlGnBu", annot␣
 ↪= True)
plt.show()
```



```
[6]: total_matches_df.groupby('win').mean()
```

```
[6]:       firstBlood  firstTower  firstInhibitor  firstBaron  firstDragon  \
     win
     Fail    0.395554    0.262477        0.066525    0.080713     0.298942
     Win     0.602321    0.725239        0.694962    0.394224     0.559704

           firstRiftHerald
     win
     Fail         0.274957
     Win          0.518823
```

Histograms here to show the win/fail chances based on each condition

(e.g. graph of firstBlood and win chances)

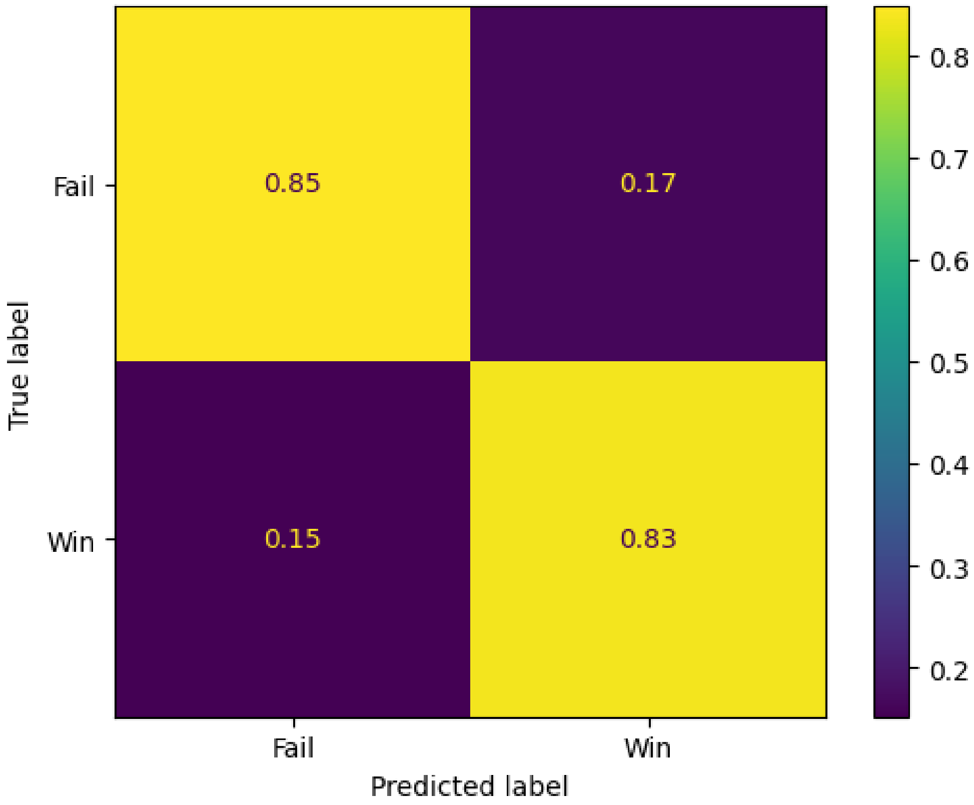
[49]:

```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import recall_score, balanced_accuracy_score,␣
 ↪confusion_matrix, ConfusionMatrixDisplay


y = total_matches_df["win"]
x = total_matches_df.drop(["win"], axis = 1)
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.4,␣
 ↪stratify=y)


dc = DecisionTreeClassifier().fit(x_train, y_train)
pred = dc.predict(x_test)


print(recall_score(y_true = y_test, y_pred = pred, pos_label = "Win"))
print(balanced_accuracy_score(y_true = y_test, y_pred = pred))
fn = ["firstBlood", "firstTower", "firstInhibitor", "firstBaron",␣
 ↪"firstDragon", "firstRiftHerald"]
print(dc.feature_importances_)
```

```
0.8509602131765138
0.8413024697191871
[0.00810117 0.13326591 0.78040934 0.03996116 0.03621114 0.00205127]
```

[51]:

```
import matplotlib.pyplot as plt

cfm = confusion_matrix(y_test, pred, normalize = "pred")

disp = ConfusionMatrixDisplay(cfm, display_labels = dc.classes_)

disp.plot()

plt.show()
```



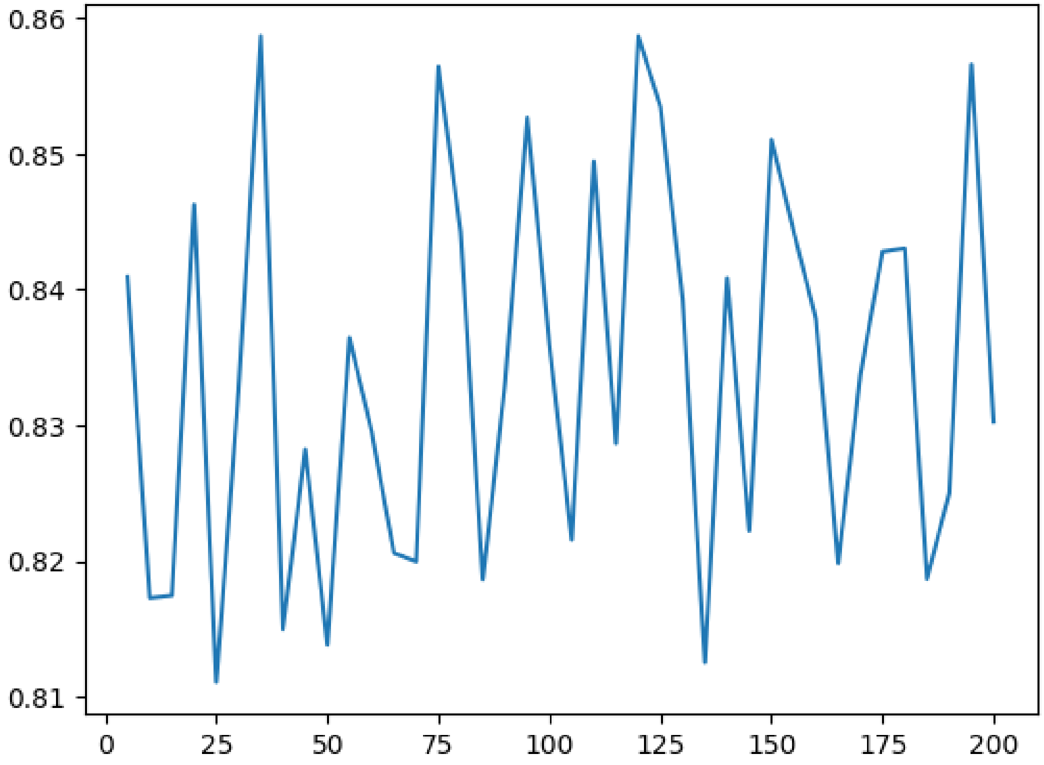
[28]:
```
from sklearn.neighbors import KNeighborsClassifier

y = total_matches_df["win"]
x = total_matches_df.drop(["win"], axis = 1)
indices = []
recalls = []
accuracies = []
for i in range(5, 201, 5):
    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.4,␣
 ↪stratify=y)

    knn = KNeighborsClassifier(n_neighbors = i, weights = "uniform").
 ↪fit(x_train, y_train)
    pred = knn.predict(x_test)
    indices.append(i)
    recalls.append(recall_score(y_true = y_test, y_pred = pred, pos_label =␣
 ↪"Win"))
    accuracies.append(balanced_accuracy_score(y_true = y_test, y_pred = pred))
```

[34]:
```
print(recalls)
print(accuracies)
```

```
[0.8409216208765965, 0.817283837177249, 0.8174676100339979, 0.8462510337223192,
0.8111044748690618, 0.8325599558945144, 0.8586786731599743, 0.8149637048607921,
0.8282183221538179, 0.8138380961132041, 0.8364421574933383, 0.8294817605439676,
0.820591748598732, 0.8199715152072039, 0.8564274556647983, 0.8442295322980796,
0.8186391619957732, 0.8330653312505742, 0.8526601121014427, 0.8359138105301847,
0.8215795277037582, 0.8494440871083341, 0.8286547826885969, 0.8586786731599743,
0.853441146742626, 0.8391987503445741, 0.812528714508867, 0.840829734448222,
0.8222227327023799, 0.8510291279977947, 0.8442525039051733, 0.8378204539189562,
0.8198336855646421, 0.8337085362491959, 0.8427823210511808, 0.8430350087292107,
0.8186621336028669, 0.8250022971607094, 0.8565882569144537, 0.8302627951851511]
[0.81820244206924, 0.841442249417765, 0.8324937700961452, 0.8401655150232199,
0.8378471348545231, 0.8400399430264034, 0.8386714752525353, 0.8392828035641751,
0.84167136384760‍3, 0.8402133251719557, 0.84225673477626, 0.8395346864152972,
0.8386277143067178, 0.8412812753282326, 0.8410264647543098, 0.8408893198743137,
0.840718482828934, 0.8405338602860724, 0.8404063764955028, 0.8432791190634243,
0.8406379040866758, 0.8408660461272826, 0.8392590429628342, 0.8424392515752774,
0.8419341189866782, 0.8425207802771645, 0.8400640677789355, 0.8403151589970335,
0.8423034999694039, 0.8418997961536518, 0.8425894127493334, 0.8405565614185495,
0.8413616931051092, 0.8392932140661975, 0.8407056090288183, 0.8403839550733608,
0.8412124277957588, 0.8412809863821789, 0.8408656357974984, 0.84180909189726]
```

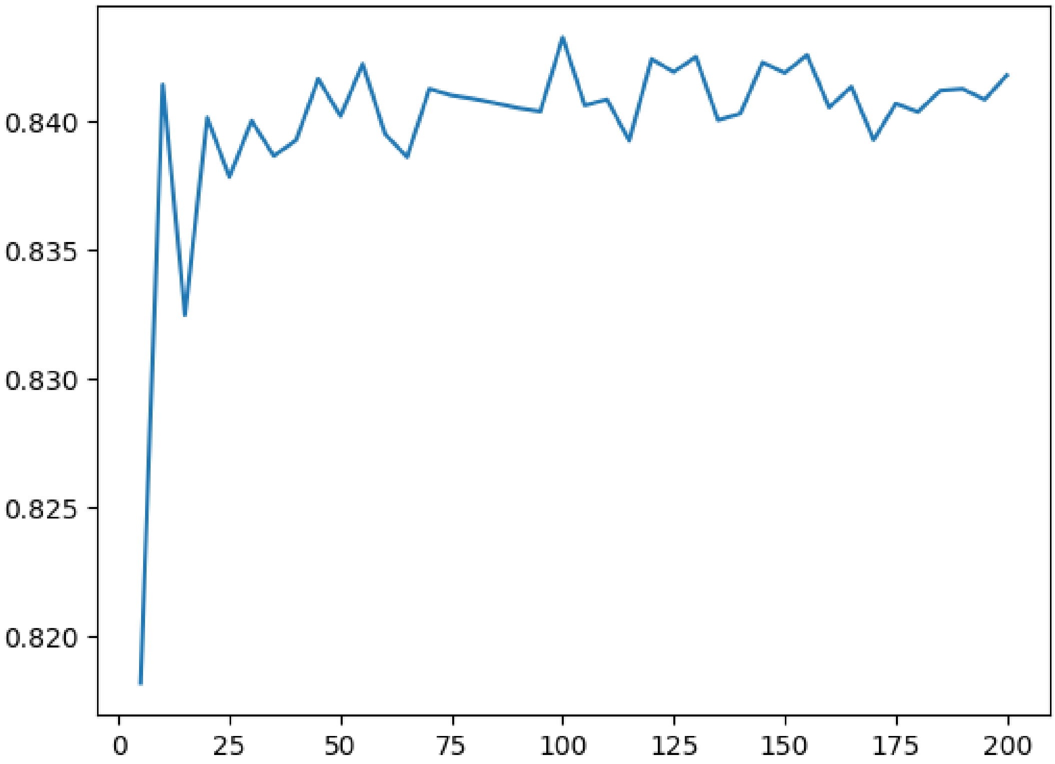
[35]:
```
sns.lineplot(x = indices, y = recalls)
```

[35]:
```
<AxesSubplot: >
```



```
[36]: sns.lineplot(x = indices, y = accuracies)
```

```
[36]: <AxesSubplot: >
```



```
[9]: matches_df = matches_df[["gameId", "participants"]]
team_champs_df = pd.DataFrame(columns = ["Blue_1", "Blue_2", "Blue_3",␣
 ↪"Blue_4", "Blue_5", "Red_1", "Red_2", "Red_3", "Red_4", "Red_5", "Win",␣
 ↪"gameId"])

for index, match in matches_df.iterrows():
    try:
        players = match["participants"]
        print(players)
        win = players[0]["teamId"] if players[0]["stats"]["win"] == True else␣
 ↪players[5]["teamId"]
        blue_team_champs = [players[i]["championId"] for i in range(5)]
        red_team_champs = [players[i + 5]["championId"] for i in range(5)]
        champs = blue_team_champs + red_team_champs
        champs.append("Blue" if win == 100 else "Red")
        champs.append(match["gameId"])
        team_champs_df.loc[len(team_champs_df)] = champs
        break
    except:
        continue

len(team_champs_df)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In [9], line 1
----> 1 matches_df = matches_df[["gameId", "participants"]]
      2 team_champs_df = pd.DataFrame(columns = ["Blue_1", "Blue_2", "Blue_3",␣
 ↪"Blue_4", "Blue_5", "Red_1", "Red_2", "Red_3", "Red_4", "Red_5", "Win",␣
 ↪"gameId"])
      4 for index, match in matches_df.iterrows():

NameError: name 'matches_df' is not defined
```

[ ]:

```
df = pd.merge(matches_df, team_champs_df, on = "gameId", how = "left").
 ↪drop(["participants"], axis = 1)
df.dropna(axis = 0, how = "any", inplace = True)
```

[ ]:

```
df.head()
```

[ ]:

```
y = df["Win"]
x = df.drop(["Win", "gameId"], axis = 1)
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.4,␣
 ↪stratify=y)


dc = DecisionTreeClassifier().fit(x_train, y_train)
pred = dc.predict(x_test)


print(recall_score(y_true = y_test, y_pred = pred, average = "micro"))
print(balanced_accuracy_score(y_true = y_test, y_pred = pred))
fn = ["Blue_1", "Blue_2", "Blue_3", "Blue_4", "Blue_5", "Red_1", "Red_2",␣
 ↪"Red_3", "Red_4", "Red_5"]
print(dc.feature_importances_)
```

[ ]:

```

```