

```
1 API_Key=""
```

```

1 import json
2 import requests
3 import time
4 import pandas as pd
5 import src.model.data_model as data
6 from dotenv import load_dotenv
7 import os
8
9 url_europe = "https://europe.api.riotgames.com"
10 url_euw1 = "https://euw1.api.riotgames.com"
11 load_dotenv()
12 key = os.getenv("API_Key")
13 if key is None:
14     key = ""
15
16 file = "../tmp/temp_match.csv" # for loading temporary matches
17
18 # file = "../data/matches6.csv" # for storing matches
19
20
21 def wait_exceeded():
22     for i in range(120):
23         time.sleep(1)
24         i = i + 1
25         if i % 10 == 0:
26             print("Noch " + str(120 - i) + ' Sekunden')
27
28
29 def http_call(url, api):
30     if "?" in api:
31         apikey = "&api_key=" + key
32     else:
33         apikey = "?api_key=" + key
34     r = requests.get(url + api + apikey, allow_redirects=False)
35     if r.status_code == 429:
36         print("Aufruf Limit erreicht. Bitte warten Sie 2 Minuten.")
37         wait_exceeded()
38         http_call(url, api)
39     elif r.status_code == 200:
40         return r
41     else:
42         print("something went wrong" + str(r.status_code))
43
44
45 def match_to_csv(new_data, file_src):
46     old_data = None
47     try:
48         old_data = pd.read_csv(file_src)
49     except:
50         pass
51     finally:
52         if old_data is not None and "temp_match" not in file_src:
53             pd.DataFrame(old_data.append(pd.json_normalize(json.loads(new_data)))).to_csv(
54                 file_src, index=False)
55         else:
56             pd.DataFrame(pd.json_normalize(json.loads(new_data))).to_csv(file_src, index=False)
57
58 def load_live_match(summoner_name, amount_matches=5):
59     summoner = []
60     s = http_call(url_euw1, "/lol/summoner/v4/summoners/by-name/" + str(summoner_name))

```

```

61  if s is None:
62      print("player might not be in game.")
63  else:
64      s_id = s.json()["id"]
65      m = http_call(url_euw1, "/lol/spectator/v4/active-games/by-summoner/" + str(s_id))
66  if m is None:
67      print("this didnt work for summoner id: " + str(s_id))
68  else:
69      match_id = m.json()["gameId"]
70      for p in m.json()["participants"]:
71          print(p["summonerName"], str(p["teamId"])[0])
72          player = http_call(url_euw1, "/lol/summoner/v4/summoners/" + p["summonerId"])
73  if player.json() is None:
74      print("cant find the player")
75      summoner.append("None")
76  else:
77      puuid = player.json()["puuid"]
78      summoner.append(data.Summoner(get_summoner(puuid, amount_matches), p["championId"]))
79      match_to_csv(data.Match(match_id, summoner[0], summoner[1], summoner[2],
80                           summoner[3], summoner[4],
81                           summoner[5], summoner[6], summoner[7], summoner[8], summoner[9],
82                           """).to_string(), file)
83
84 def load_one_match(summoner_name, amount_matches=5):
85     summoner = []
86     s = http_call(url_euw1, "/lol/summoner/v4/summoners/by-name/" + str(summoner_name))
87  if s is None:
88      print("cant find a game of this player.")
89  else:
90      s_id = s.json()["puuid"]
91      m = http_call(url_europe, "/lol/match/v5/matches/by-puuid/" + s_id + "/ids?start=0&
92 count=1")
93  if m is None:
94      print("this didnt work for summoner id: " + str(s_id))
95  else:
96      match_id = m.json()[0]
97      match = http_call(url_europe, "/lol/match/v5/matches/" + str(match_id))
98      for p in match.json()["info"]["participants"]:
99          print(p["summonerName"], str(p["teamId"])[0])
100         player = http_call(url_euw1, "/lol/summoner/v4/summoners/" + p["summonerId"])
101  if player.json() is None:
102      print("cant find the player")
103      summoner.append("None")
104  else:
105      puuid = player.json()["puuid"]
106      summoner.append(data.Summoner(get_summoner(puuid, amount_matches), p["championId"]))
107      match_to_csv(data.Match(match_id, summoner[0], summoner[1], summoner[2],
108                           summoner[3], summoner[4],
109                           summoner[5], summoner[6], summoner[7], summoner[8], summoner[9],
110                           """).to_string(), file)
111
112 def load_match(match_id, amount_matches=5):
113     summoner = []
114     m = http_call(url_europe, "/lol/match/v5/matches/" + match_id)
115  if m is None:
116      print("retry load match: " + match_id)

```

```

116     return load_match(match_id, amount_matches)
117 else:
118     win = str(m.json()["info"]["participants"][0]["win"])
119     for p in m.json()["info"]["participants"]:
120         print(p["summonerName"], str(p["teamId"])[0])
121         summoner.append(data.Summoner(get_summoner(p["puuid"]), amount_matches), p["championId"]))
122
123     match_to_csv(data.Match(match_id, summoner[0], summoner[1], summoner[2], summoner[3], summoner[4],
124                             summoner[5], summoner[6], summoner[7], summoner[8], summoner[9], win
125                             ).to_string(), file)
126
127 def get_summoner(puu_id, amount_matches):
128     r = http_call(url_europe, "/lol/match/v5/matches/by-puuid/" + puu_id + "/ids?start=0&
129 count=" + str(amount_matches))
130     if r is not None:
131         match_ids = r.json()
132         summoner_matches = []
133         for ma in match_ids:
134             r = http_call(url_europe, "/lol/match/v5/matches/" + str(ma))
135             stats = None
136             if r is not None:
137                 for p in r.json()["info"]["participants"]:
138                     if p["puuid"] == puu_id:
139                         stats = p
140                     if stats is None:
141                         stats = "None"
142                     game_duration = r.json()["info"]["gameDuration"]
143                     if game_duration is None:
144                         game_duration = "None"
145                     summoner_matches.append(data.SummonerMatch(puu_id, game_duration, stats))
146             else:
147                 print("retry loading summoner_match : " + str(ma) + "; " + puu_id)
148             if summoner_matches is None:
149                 summoner_matches = "None"
150             return summoner_matches
151     else:
152         print("retry loading match_history : " + puu_id)
153     return get_summoner(puu_id, amount_matches)
154
155 def load_summoner_history(summoner_name, amount_matches):
156     r = http_call(url_euw1, "/lol/summoner/v4/summoners/by-name/" + summoner_name)
157     if r is not None:
158         puuid = r.json()["puuid"]
159         r = http_call(url_europe,
160                       "/lol/match/v5/matches/by-puuid/" + puuid + "/ids?start=0&count=" + str(
161                           amount_matches))
162         if r is not None:
163             matches = r.json()
164             return matches
165         load_summoner_history(summoner_name, amount_matches)
166     else:
167         load_summoner_history(summoner_name, amount_matches)
168

```

```

1 from src.api.api import load_live_match, load_one_match, load_match,
2 load_summoner_history
3
4 model = load_model("random_forest")
5
6 loaders = [{"1": "live"}, {"2": "csv"}, {"3": "letztes"}, {"4": "match id"}]
7 loader_chosen = False
8 loader = None
9 matches = 5
10 while not loader_chosen:
11     print("Wie soll das Match geladen werden? Bitte die Nummer eingeben.")
12     print(loaders)
13     try:
14         input1 = int(input())
15         if 0 < input1 <= len(loaders):
16             loader = loaders[input1 - 1]
17             loader = loader.get(str(input1))
18             loader_chosen = True
19         else:
20             print("Bitte eine der angegebenen Nummern wählen")
21     except:
22         print("Bitte eine der angegebenen Nummern wählen")
23     print(loader)
24     matches_chosen = False
25     while not matches_chosen:
26         print("Wie viele vergangene Spiele sollen für die Bewertung der Leistung beachtet werden? [1-100]")
27         try:
28             matches = int(input())
29             if 101 > matches > 0:
30                 matches_chosen = True
31             else:
32                 print("Bitte geben sie eine Zahl zwischen 1 und 100 ein.")
33         except:
34             print("Bitte geben sie eine Zahl zwischen 1 und 100 ein.")
35
36 if loader == "csv":
37     print("Bitte den Namen einer Match Datei im Ordner tmp eingeben.")
38     csv_src = input()
39     print("Vorhersage für Sieg von Team "+str(2-predict_with_model(model, csv_src)[0]))
40 else:
41     print("Bitte den Summoner Namen eingeben.")
42     summoner_name = input()
43     if loader == "letztes":
44         print("lade letztes Spiel von " + summoner_name + ":")
45         load_one_match(summoner_name, matches)
46     elif loader == "live":
47         print("lade live Spiel von " + summoner_name + ":")
48         load_live_match(summoner_name, matches)
49     elif loader == "match id":
50         print("Lade die letzten 20 match ids:")
51         print(load_summoner_history(summoner_name, 20))
52         print("Bitte die match id eingeben.")
53         match_id = input()
54         print("lade Spiel " + match_id + " von " + summoner_name + ":")
55         load_match(match_id, matches)
56     print("Vorhersage für Sieg von Team "+str(2-predict_with_model(model, "../tmp/
temp_match.csv")[0]))
57

```

```

1 import pandas as pd
2 from src.model.file_writer import write
3
4
5 def create_features_1(df, features):
6     pd.set_option('display.max_columns', None)
7     match_df = df[['matchId', "game_result"]]
8     # array with selected game stats
9     avg_features = features
10
11    # calculate features in feature_df
12    feature_df = pd.DataFrame()
13
14    for avg_feature in avg_features:
15        # team blue
16        feature_df[avg_feature + '_avg_team_blue'] = (df['summoner_1_' + avg_feature + '_avg']
17                                            + df['summoner_2_' + avg_feature + '_avg']
18                                            + df['summoner_3_' + avg_feature + '_avg']
19                                            + df['summoner_4_' + avg_feature + '_avg']
20                                            + df['summoner_5_' + avg_feature + '_avg']) / 5
21
22        # team red
23        feature_df[avg_feature + '_avg_team_red'] = (df['summoner_6_' + avg_feature + '_avg']
24                                            + df['summoner_7_' + avg_feature + '_avg']
25                                            + df['summoner_8_' + avg_feature + '_avg']
26                                            + df['summoner_9_' + avg_feature + '_avg']
27                                            + df['summoner_10_' + avg_feature + '_avg']) / 5
28
29        feature_df[avg_feature] = ""
30        for index, row in feature_df.iterrows():
31            if row[avg_feature + '_avg_team_blue'] == row[avg_feature + '_avg_team_red']:
32                feature_df.at[index, avg_feature] = 1
33            if row[avg_feature + '_avg_team_blue'] > row[avg_feature + '_avg_team_red']:
34                feature_df.at[index, avg_feature] = 1 + (
35                    row[avg_feature + '_avg_team_red'] / row[avg_feature + '_avg_team_blue'])
36            else:
37                feature_df.at[index, avg_feature] = (
38                    row[avg_feature + '_avg_team_blue'] / row[avg_feature + '_avg_team_red'])
39
40    # select features from feature_df
41    feature_result_df = df[['matchId']]
42    for avg_feature in avg_features:
43        feature_result_df.insert(1, avg_feature, feature_df[avg_feature])
44        #feature_result_df[avg_feature] = feature_df[avg_feature]
45
46
47    write("\n----- features dataframe (1) -----")
48    result_df = match_df.merge(feature_result_df, on='matchId', how='inner')
49    result_df.to_csv('../tmp/features_1.csv')
50    write(result_df.columns)
51    write('\n')
52    # write(result_df.head())
53    # return result_df
54
55 def create_features_2(df, features):
56     pd.set_option('display.max_columns', None)
57     match_df = df[['matchId', "game_result"]]
58     # array with selected game stats
59     avg_features = features
60
61    # calculate features in feature_df

```

```

62     feature_df = pd.DataFrame()
63
64     for avg_feature in avg_features:
65         # team blue
66         feature_df[avg_feature + '_avg_team_blue'] = (df['summoner_1_' + avg_feature + ' '
67                                         + df['summoner_2_' + avg_feature + '_avg']
68                                         + df['summoner_3_' + avg_feature + '_avg']
69                                         + df['summoner_4_' + avg_feature + '_avg']
70                                         + df['summoner_5_' + avg_feature + '_avg']) / 5
71
72         # team red
73         feature_df[avg_feature + '_avg_team_red'] = (df['summoner_6_' + avg_feature + '_avg'
74                                         + df['summoner_7_' + avg_feature + '_avg']
75                                         + df['summoner_8_' + avg_feature + '_avg']
76                                         + df['summoner_9_' + avg_feature + '_avg']
77                                         + df['summoner_10_' + avg_feature + '_avg']) / 5
78
79     feature_df[avg_feature] = "
80     for index, row in feature_df.iterrows():
81         if row[avg_feature + '_avg_team_blue'] == row[avg_feature + '_avg_team_red']:
82             feature_df.at[index, avg_feature] = 0
83         if row[avg_feature + '_avg_team_blue'] > row[avg_feature + '_avg_team_red']:
84             feature_df.at[index, avg_feature] = \
85                 row[avg_feature + '_avg_team_red'] / row[avg_feature + '_avg_team_blue']
86         else:
87             feature_df.at[index, avg_feature] = \
88                 (row[avg_feature + '_avg_team_blue'] / row[avg_feature + '_avg_team_red']) * (-
89                 1)
90
91     # select features from feature_df
92     feature_result_df = df[['matchId']]
93     for avg_feature in avg_features:
94         feature_result_df.insert(1, avg_feature, feature_df[avg_feature])
95
96
97     write("\n----- features dataframe (2) -----\\n")
98     result_df = match_df.merge(feature_result_df, on='matchId', how='inner')
99     result_df.to_csv('..tmp/features_2.csv')
100    write(result_df.columns)
101    write("\\n")
102    # write(result_df.head())
103    # return result_df

```

```

1  class SummonerMatch:
2
3      def __init__(self, puuid, game_time, in_game_stats):
4          self.puuid = puuid # identifier of the summoner playing the match
5          self.game_time = game_time # the time until one team won
6          self.in_game_stats = in_game_stats # the summoners in game stats, such as damage
done, etc
7
8      def to_string(self):
9          return '{ "puuid": "' + self.puuid + '", "game_time": ' + str(self.game_time) \
10             + ', "in_game_stats": ' + \
11             str(self.in_game_stats) + "}"
12
13
14  class Summoner:
15
16      def __init__(self, match_history, champion):
17          self.match_history = match_history # array of last matches
18          self.champion = champion # champion in active game
19
20      def to_string(self):
21          match_history = "["
22          for m in self.match_history:
23              match_history += m.to_string() + ","
24          match_history = match_history[0:len(match_history) - 1]
25          match_history += "]"
26          return '{ "match_history": ' + match_history + ', "champion": ' + str(self.champion) + "}"
27
28
29  class Match:
30
31      def __init__(self, match_id, summoner1, summoner2, summoner3, summoner4,
32                  summoner5, summoner6, summoner7, summoner8, summoner9, summoner10, win
33      ):
34          self.match_id = match_id # unique match id
35          self.summoner1 = summoner1 # game participants
36          self.summoner2 = summoner2
37          self.summoner3 = summoner3
38          self.summoner4 = summoner4
39          self.summoner5 = summoner5
40          self.summoner6 = summoner6
41          self.summoner7 = summoner7
42          self.summoner8 = summoner8
43          self.summoner9 = summoner9
44          self.summoner10 = summoner10
45          self.win = win # summoner 1-5 game won?
46
47      def to_string(self):
48          res = '{ "matchId": "' + str(self.match_id) + '", "Summoner1": ' + self.summoner1.
49          to_string() + ', "Summoner2":' \
50             + self.summoner2.to_string() + ', "Summoner3":' + self.summoner3.to_string() + ', "Sum
51             moner4":' \
52             + self.summoner4.to_string() + ', "Summoner5":' + self.summoner5.to_string() + ', "Sum
53             moner6":' + self.summoner6.to_string() \
54             + ', "Summoner7":' + self.summoner7.to_string() + ', "Summoner8":' + self.
summoner8.to_string() \
55             + ', "Summoner9":' + self.summoner9.to_string() + ', "Summoner10":' + self.
summoner10.to_string() \
56             + ', "wins":' + str(self.win) + "}"
57
58      return res.replace("\"", "").replace("True", "true").replace("False", "false")

```

```
1 def write(text):
2     f = open('../data/log.txt', 'a')
3     f.write(str(text))
4     f.close()
5
6
7 def read_log():
8     f = open('../data/log.txt', 'r')
9     return f.read()
10
```

```

1 import pandas as pd
2 from joblib import dump, load
3 from sklearn.model_selection import train_test_split
4 from sklearn.metrics import mean_absolute_error, accuracy_score, classification_report
5 from sklearn.neighbors import KNeighborsClassifier
6 from sklearn.neural_network import MLPClassifier
7 from sklearn.tree import DecisionTreeClassifier
8 from sklearn.ensemble import RandomForestClassifier
9
10 from src.model.features import create_features_1
11 from src.model.file_writer import write
12 from src.model.data_processing import process_data
13
14
15 def train_model(df, classifier):
16     # select features
17     X = df[['kills', 'deaths', 'assists', 'damageDealtToObjectives', 'dragonKills', 'goldEarned',
18             'totalDamageDealt', 'turretKills', 'visionScore', 'win']]
19
20     # select target object
21     y = df.game_result
22
23     # split into validation and training data
24     train_X, val_X, train_y, val_y = train_test_split(X, y, test_size=0.35, random_state=2)
25
26     # specify model
27     prediction_model = None
28
29     if classifier == 1:
30         prediction_model = RandomForestClassifier(random_state=2)
31         write('----- Random Forest Classifier -----\\n')
32     elif classifier == 2:
33         prediction_model = DecisionTreeClassifier(random_state=3)
34         write('----- Decision Tree Classifier -----\\n')
35     elif classifier == 3:
36         prediction_model = MLPClassifier(random_state=3)
37         write('----- MLP Classifier -----\\n')
38     elif classifier == 4:
39         prediction_model = KNeighborsClassifier(n_neighbors=5)
40         write('----- K Neighbors Classifier -----\\n')
41     else:
42         print('\\n----- Select Classifier! -----\\n')
43
44     # fit model with train_X and train_y
45     if prediction_model is not None:
46         prediction_model.fit(train_X, train_y)
47
48     # predictions on val_X with model
49     val_predictions = prediction_model.predict(val_X)
50
51     # calculate accuracy
52
53     write('\\naccuracy:\\n')
54     write(accuracy_score(val_y, val_predictions))
55
56     write('\\nmean absolute error:\\n')
57     write(mean_absolute_error(val_y, val_predictions))
58
59     write('\\nclassification report:\\n')
60     write(classification_report(val_y, val_predictions))
61     return prediction_model

```

```
62     else:  
63         write('\n----- prediction_model is none -----\\n')  
64  
65  
66     def save_model(model, name):  
67         dump(model, '../model/trained_models/' + name + '.joblib')  
68  
69  
70     def load_model(name):  
71         return load('../model/trained_models/' + name + '.joblib')  
72  
73  
74     def predict_with_model(model, csv_path):  
75  
76         # select features  
77         features = ['kills', 'deaths', 'assists', 'damageDealtToObjectives', 'dragonKills', '  
goldEarned',  
78             'totalDamageDealt', 'turretKills', 'visionScore', 'win']  
79  
80         # read and process data of match  
81         match_df = pd.read_csv(csv_path)  
82         processed_match_df = process_data(match_df, features)  
83  
84         # create features (1)  
85         create_features_1(processed_match_df, features)  
86         features_1 = pd.read_csv('../tmp/features_1.csv')  
87  
88         feature_result = features_1.drop(['Unnamed: 0', 'matchId', 'game_result'], axis=1)  
89  
90         # return prediction on input features  
91         return model.predict(feature_result)  
92
```

```

1 import json
2 import pandas as pd
3 from src.model.file_writer import write
4
5
6 def process_data(df, features):
7     pd.set_option('display.max_columns', None)
8
9     # create matches_df, game_result = 1 => blue team wins
10    matches_df = df[['matchId', 'wins']]
11    matches_df.insert(2, 'game_result', 'null')
12    # print(matches_df)
13    for index, row in matches_df.iterrows():
14        # print(row['wins'])
15        if row.wins == True:
16            matches_df.at[index, 'game_result'] = '1'
17        else:
18            matches_df.at[index, 'game_result'] = '0'
19    #print(matches_df.head())
20    matches_df = matches_df.drop('wins', 1)
21
22    # create summoner dataframes
23    summoner_1 = df[['matchId', 'Summoner1.match_history']]
24    summoner_2 = df[['matchId', 'Summoner2.match_history']]
25    summoner_3 = df[['matchId', 'Summoner3.match_history']]
26    summoner_4 = df[['matchId', 'Summoner4.match_history']]
27    summoner_5 = df[['matchId', 'Summoner5.match_history']]
28    summoner_6 = df[['matchId', 'Summoner6.match_history']]
29    summoner_7 = df[['matchId', 'Summoner7.match_history']]
30    summoner_8 = df[['matchId', 'Summoner8.match_history']]
31    summoner_9 = df[['matchId', 'Summoner9.match_history']]
32    summoner_10 = df[['matchId', 'Summoner10.match_history']]
33
34    # array with selected game stats
35    avg_features = features
36
37    # create avg features for each summoner (player)
38    summoner_1 = create_avg_features(summoner_1, '1', avg_features)
39    summoner_2 = create_avg_features(summoner_2, '2', avg_features)
40    summoner_3 = create_avg_features(summoner_3, '3', avg_features)
41    summoner_4 = create_avg_features(summoner_4, '4', avg_features)
42    summoner_5 = create_avg_features(summoner_5, '5', avg_features)
43    summoner_6 = create_avg_features(summoner_6, '6', avg_features)
44    summoner_7 = create_avg_features(summoner_7, '7', avg_features)
45    summoner_8 = create_avg_features(summoner_8, '8', avg_features)
46    summoner_9 = create_avg_features(summoner_9, '9', avg_features)
47    summoner_10 = create_avg_features(summoner_10, '10', avg_features)
48
49    # join matches_df with summoner dataframes
50    matches_df = pd.merge(matches_df, summoner_1, on='matchId', how='inner')
51    matches_df = pd.merge(matches_df, summoner_2, on='matchId', how='inner')
52    matches_df = pd.merge(matches_df, summoner_3, on='matchId', how='inner')
53    matches_df = pd.merge(matches_df, summoner_4, on='matchId', how='inner')
54    matches_df = pd.merge(matches_df, summoner_5, on='matchId', how='inner')
55    matches_df = pd.merge(matches_df, summoner_6, on='matchId', how='inner')
56    matches_df = pd.merge(matches_df, summoner_7, on='matchId', how='inner')
57    matches_df = pd.merge(matches_df, summoner_8, on='matchId', how='inner')
58    matches_df = pd.merge(matches_df, summoner_9, on='matchId', how='inner')
59    matches_df = pd.merge(matches_df, summoner_10, on='matchId', how='inner')
60
61    # write csv file

```

```
62 write('----- result dataframe -----\\n')
63 matches_df.to_csv('../data/result_data_model.csv')
64 write(matches_df.columns)
65 # write(matches_df.head())
66 return matches_df
67
68
69 def create_avg_features(summoner_dataframe, summoner_number, avg_features):
70     # iterate over avg_features
71     for avg_feature in avg_features:
72         # create feature columns
73         summoner_dataframe.insert(2, 'summoner_' + summoner_number + '_' + avg_feature
74 + '_avg', '')
75
76         # iterate over
77         for index, row in summoner_dataframe.iterrows():
78
79             # transform and load json
80             match_history = '{"match_history":"' + row['Summoner' + summoner_number + '.'
81 match_history'] \
82             .replace("{}", "").replace("True", "true").replace("False", "false") + '}'
83             match_history_json = json.loads(match_history)['match_history']
84
85             # set counters
86             match_count = 0
87             features_count = 0
88             none_count = 0
89
90             # iterate over in game stats
91             for x in match_history_json:
92                 if x['in_game_stats'] == 'None':
93                     none_count += 1
94                 else:
95                     match_count += 1
96                     features_count += x['in_game_stats'][str(avg_feature)]
97                     # if avg_feature == 'win': print('features_count: '+str(x['in_game_stats'][str(
98 avg_feature)]))
99
100                # calculate avg
101                summoner_dataframe.at[index, 'summoner_' + summoner_number + '_' +
102 avg_feature + '_avg'] = features_count / (
103                         match_count - none_count)
104                # if avg_feature == 'win': print('AVG : ' + str(features_count / (match_count -
105 none_count)))
106                return summoner_dataframe.drop('Summoner' + summoner_number + '.match_history',
107 1)
108
```

```

1 import pandas as pd
2 import seaborn as sns
3 import matplotlib.pyplot as plt
4
5 from src.model.data_processing import process_data
6 from src.model.features import create_features_1, create_features_2
7 from src.model.file_writer import write, read_log
8 from src.model.model_training import train_model, save_model
9
10 write("\n----- START -----\\n")
11
12 # select in game statistics features
13 features = ['kills', 'deaths', 'assists', 'damageDealtToObjectives', 'dragonKills', 'goldEarned',
14             'totalDamageDealt', 'turretKills', 'visionScore', 'win']
15
16 # read and process csv data generated by api script './tmp/features_1.csv'
17 # result_matches = pd.read_csv('./tmp/temp_match.csv')
18
19 result_matches = pd.read_csv('../data/matches.csv')
20 # matches_2 = pd.read_csv('../data/matches2.csv')
21 # matches_3 = pd.read_csv('../data/matches3.csv')
22 # matches_4 = pd.read_csv('../data/matches4.csv')
23 # matches_5 = pd.read_csv('../data/matches5.csv')
24 # result_matches = pd.concat([matches_2, matches_3, matches_4, matches_5], ignore_index=True)
25 # result_matches.to_csv('../data/matches.csv')
26 result_data_model = process_data(result_matches, features)
27
28 # check result data model
29 write('Indescription:')
30 write(result_data_model[['matchId', 'game_result', 'summoner_1_win_avg']].describe())
31 write(result_data_model.shape[0])
32
33 # create features with result_data_model
34 create_features_1(result_data_model, features)
35 create_features_2(result_data_model, features)
36
37 # analyze features
38 features_for_graph = ['kills', 'deaths', 'assists', 'damageDealtToObjectives', 'dragonKills', 'goldEarned',
39                         'totalDamageDealt', 'turretKills', 'visionScore', 'win']
40 features_df = pd.read_csv('../tmp/features_1.csv')
41 sns.relplot(x="value", y="game_result", col="variable",
42              data=features_df.melt(id_vars='game_result', value_vars=features_for_graph),
43              facet_kws=dict(sharesx=False))
44 plt.show()
45
46 # train prediction model
47 model_1 = train_model(pd.read_csv('../tmp/features_1.csv'), 1) # RandomForestClassifier
48 # save_model(model_1, 'random_forest')
49 model_2 = train_model(pd.read_csv('../tmp/features_1.csv'), 2) # DecisionTreeClassifier
50 # save_model(model_2, 'decision_tree')
51 # model_3 = train_model(pd.read_csv('../tmp/features_1.csv'), 3) # MLPClassifier
52 # save_model(model_3, 'mlp')
53 # model_4 = train_model(pd.read_csv('../tmp/features_1.csv'), 4) # KNeighborsClassifier
54 # save_model(model_4, 'k_neighbors')
55
56 print(read_log())
57

```