```
API_Key=""
```

```python
import json
import requests
import time
import pandas as pd
import src.model.data_model as data
from dotenv import load_dotenv
import os

url_europe = "https://europe.api.riotgames.com"
url_euw1 = "https://euw1.api.riotgames.com"
load_dotenv()
key = os.getenv("API_Key")
if key is None:
    key = ""

file = "../tmp/temp_match.csv"  # for loading temporary matches

# file = "../data/matches6.csv"  # for storing matches


def wait_exceeded():
    for i in range(120):
        time.sleep(1)
        i = i + 1
        if i % 10 == 0:
            print('Noch ' + str(120 - i) + ' Sekunden')


def http_call(url, api):
    if "?" in api:
        apikey = "&api_key=" + key
    else:
        apikey = "?api_key=" + key
    r = requests.get(url + api + apikey, allow_redirects=False)
    if r.status_code == 429:
        print("Aufruf Limit erreicht. Bitte warten Sie 2 Minuten.")
        wait_exceeded()
        http_call(url, api)
    elif r.status_code == 200:
        return r
    else:
        print("something went wrong" + str(r.status_code))


def match_to_csv(new_data, file_src):
    old_data = None
    try:
        old_data = pd.read_csv(file_src)
    except:
        pass
    finally:
        if old_data is not None and "temp_match" not in file_src:
            pd.DataFrame(old_data.append(pd.json_normalize(json.loads(new_data)))).to_csv(file_src, index=False)
        else:
            pd.DataFrame(pd.json_normalize(json.loads(new_data))).to_csv(file_src, index=False)


def load_live_match(summoner_name, amount_matches=5):
    summoner = []
    s = http_call(url_euw1, "/lol/summoner/v4/summoners/by-name/" + str(summoner_name))
```

```
    if s is None:
        print("player might not be in game.")
    else:
        s_id = s.json()["id"]
        m = http_call(url_euw1, "/lol/spectator/v4/active-games/by-summoner/" + str(s_id))
        if m is None:
            print("this didnt work for summoner id: " + str(s_id))
        else:
            match_id = m.json()["gameId"]
            for p in m.json()["participants"]:
                print(p["summonerName"], str(p["teamId"])[0])
                player = http_call(url_euw1, "/lol/summoner/v4/summoners/" + p["summonerId"])
                if player.json() is None:
                    print("cant find the player")
                    summoner.append("'None'")
                else:
                    puuid = player.json()["puuid"]
                    summoner.append(data.Summoner(get_summoner(puuid, amount_matches), p["championId"]))
            match_to_csv(data.Match(match_id, summoner[0], summoner[1], summoner[2], summoner[3], summoner[4],
                                    summoner[5], summoner[6], summoner[7], summoner[8], summoner[9],
                                    "'None'").to_string(), file)


def load_one_match(summoner_name, amount_matches=5):
    summoner = []
    s = http_call(url_euw1, "/lol/summoner/v4/summoners/by-name/" + str(summoner_name))
    if s is None:
        print("cant find a game of this player.")
    else:
        s_id = s.json()["puuid"]
        m = http_call(url_europe, "/lol/match/v5/matches/by-puuid/" + s_id + "/ids?start=0&count=1")
        if m is None:
            print("this didnt work for summoner id: " + str(s_id))
        else:
            match_id = m.json()[0]
            match = http_call(url_europe, "/lol/match/v5/matches/" + str(match_id))
            for p in match.json()["info"]["participants"]:
                print(p["summonerName"], str(p["teamId"])[0])
                player = http_call(url_euw1, "/lol/summoner/v4/summoners/" + p["summonerId"])
                if player.json() is None:
                    print("cant find the player")
                    summoner.append("'None'")
                else:
                    puuid = player.json()["puuid"]
                    summoner.append(data.Summoner(get_summoner(puuid, amount_matches), p["championId"]))
            match_to_csv(data.Match(match_id, summoner[0], summoner[1], summoner[2], summoner[3], summoner[4],
                                    summoner[5], summoner[6], summoner[7], summoner[8], summoner[9],
                                    "'None'").to_string(), file)


def load_match(match_id, amount_matches=5):
    summoner = []
    m = http_call(url_europe, "/lol/match/v5/matches/" + match_id)
    if m is None:
        print("retry load match: " + match_id)
```

```python
        return load_match(match_id, amount_matches)
    else:
        win = str(m.json()["info"]["participants"][0]["win"])
        for p in m.json()["info"]["participants"]:
            print(p["summonerName"], str(p["teamId"])[0])
            summoner.append(data.Summoner(get_summoner(p["puuid"], amount_matches), p["championId"]))

    match_to_csv(data.Match(match_id, summoner[0], summoner[1], summoner[2], summoner[3], summoner[4],
                            summoner[5], summoner[6], summoner[7], summoner[8], summoner[9], win).to_string(), file)


def get_summoner(puu_id, amount_matches):
    r = http_call(url_europe, "/lol/match/v5/matches/by-puuid/" + puu_id + "/ids?start=0&count=" + str(amount_matches))
    if r is not None:
        match_ids = r.json()
        summoner_matches = []
        for ma in match_ids:
            r = http_call(url_europe, "/lol/match/v5/matches/" + str(ma))
            stats = None
            if r is not None:
                for p in r.json()["info"]["participants"]:
                    if p["puuid"] == puu_id:
                        stats = p
                if stats is None:
                    stats = "'None'"
                game_duration = r.json()["info"]["gameDuration"]
                if game_duration is None:
                    game_duration = "'None'"
                summoner_matches.append(data.SummonerMatch(puu_id, game_duration, stats))
            else:
                print("retry loading summoner_match : " + str(ma) + "; " + puu_id)
        if summoner_matches is None:
            summoner_matches = "'None'"
        return summoner_matches
    else:
        print("retry loading match_history : " + puu_id)
        return get_summoner(puu_id, amount_matches)


def load_summoner_history(summoner_name, amount_matches):
    r = http_call(url_euw1, "/lol/summoner/v4/summoners/by-name/" + summoner_name)
    if r is not None:
        puuid = r.json()["puuid"]
        r = http_call(url_europe,
                      "/lol/match/v5/matches/by-puuid/" + puuid + "/ids?start=0&count=" + str(amount_matches))
        if r is not None:
            matches = r.json()
            return matches
        else:
            load_summoner_history(summoner_name, amount_matches)
    else:
        load_summoner_history(summoner_name, amount_matches)
```

```python
from src.api.api import load_live_match, load_one_match, load_match, load_summoner_history
from src.model.model_training import load_model, predict_with_model

model = load_model("random_forest")

loaders = [{'1': "live"}, {'2': "csv"}, {'3': "letztes"}, {'4': "match id"}]
loader_chosen = False
loader = None
matches = 5
while not loader_chosen:
    print("Wie soll das Match geladen werden? Bitte die Nummer eingeben.")
    print(loaders)
    try:
        input1 = int(input())
        if 0 < input1 <= len(loaders):
            loader = loaders[input1 - 1]
            loader = loader.get(str(input1))
            loader_chosen = True
        else:
            print("Bitte eine der angegebenen Nummern wählen")
    except:
        print("Bitte eine der angegebenen Nummern wählen")
print(loader)
matches_chosen = False
while not matches_chosen:
    print("Wie viele vergangene Spiele sollen für die Bewertung der Leistung beachtet werden? [1-100]")
    try:
        matches = int(input())
        if 101 > matches > 0:
            matches_chosen = True
        else:
            print("Bitte geben sie eine Zahl zwischen 1 und 100 ein.")
    except:
        print("Bitte geben sie eine Zahl zwischen 1 und 100 ein.")

if loader == "csv":
    print("Bitte den Namen einer Match Datei im Ordner tmp eingeben.")
    csv_src = input()
    print("Vorhersage für Sieg von Team "+str(2-predict_with_model(model, csv_src)[0]))
else:
    print("Bitte den Summoner Namen eingeben.")
    summoner_name = input()
    if loader == "letztes":
        print("lade letztes Spiel von " + summoner_name + ":")
        load_one_match(summoner_name, matches)
    elif loader == "live":
        print("lade live Spiel von " + summoner_name + ":")
        load_live_match(summoner_name, matches)
    elif loader == "match id":
        print("Lade die letzten 20 match ids:")
        print(load_summoner_history(summoner_name, 20))
        print("Bitte die match id eingeben.")
        match_id = input()
        print("lade Spiel " + match_id + " von " + summoner_name + ":")
        load_match(match_id, matches)
    print("Vorhersage für Sieg von Team "+str(2-predict_with_model(model, "../tmp/temp_match.csv")[0]))

```

```python
import pandas as pd
from src.model.file_writer import write


def create_features_1(df, features):
    pd.set_option('display.max_columns', None)
    match_df = df[['matchId', "game_result"]]
    # array with selected game stats
    avg_features = features

    # calculate features in feature_df
    feature_df = pd.DataFrame()

    for avg_feature in avg_features:
        # team blue
        feature_df[avg_feature + '_avg_team_blue'] = (df['summoner_1_' + avg_feature + '_avg']
                                                      + df['summoner_2_' + avg_feature + '_avg']
                                                      + df['summoner_3_' + avg_feature + '_avg']
                                                      + df['summoner_4_' + avg_feature + '_avg']
                                                      + df['summoner_5_' + avg_feature + '_avg']) / 5

        # team red
        feature_df[avg_feature + '_avg_team_red'] = (df['summoner_6_' + avg_feature + '_avg']
                                                     + df['summoner_7_' + avg_feature + '_avg']
                                                     + df['summoner_8_' + avg_feature + '_avg']
                                                     + df['summoner_9_' + avg_feature + '_avg']
                                                     + df['summoner_10_' + avg_feature + '_avg']) / 5

        feature_df[avg_feature] = ''
        for index, row in feature_df.iterrows():
            if row[avg_feature + '_avg_team_blue'] == row[avg_feature + '_avg_team_red']:
                feature_df.at[index, avg_feature] = 1
            if row[avg_feature + '_avg_team_blue'] > row[avg_feature + '_avg_team_red']:
                feature_df.at[index, avg_feature] = 1 + (
                            row[avg_feature + '_avg_team_red'] / row[avg_feature + '_avg_team_blue'])
            else:
                feature_df.at[index, avg_feature] = (
                            row[avg_feature + '_avg_team_blue'] / row[avg_feature + '_avg_team_red'])

    # select features from feature_df
    feature_result_df = df[['matchId']]
    for avg_feature in avg_features:
        feature_result_df.insert(1, avg_feature, feature_df[avg_feature])
        #feature_result_df[avg_feature] = feature_df[avg_feature]


    write('\n-------------------- features dataframe (1) --------------------\n')
    result_df = match_df.merge(feature_result_df, on='matchId', how='inner')
    result_df.to_csv('../tmp/features_1.csv')
    write(result_df.columns)
    write(' \n')
    # write(result_df.head())
    # return result_df

def create_features_2(df, features):
    pd.set_option('display.max_columns', None)
    match_df = df[['matchId', "game_result"]]
    # array with selected game stats
    avg_features = features

    # calculate features in feature_df
```

```
    feature_df = pd.DataFrame()

    for avg_feature in avg_features:
        # team blue
        feature_df[avg_feature + '_avg_team_blue'] = (df['summoner_1_' + avg_feature + '_avg']
                                                      + df['summoner_2_' + avg_feature + '_avg']
                                                      + df['summoner_3_' + avg_feature + '_avg']
                                                      + df['summoner_4_' + avg_feature + '_avg']
                                                      + df['summoner_5_' + avg_feature + '_avg']) / 5

        # team red
        feature_df[avg_feature + '_avg_team_red'] = (df['summoner_6_' + avg_feature + '_avg']
                                                     + df['summoner_7_' + avg_feature + '_avg']
                                                     + df['summoner_8_' + avg_feature + '_avg']
                                                     + df['summoner_9_' + avg_feature + '_avg']
                                                     + df['summoner_10_' + avg_feature + '_avg']) / 5

        feature_df[avg_feature] = ''
        for index, row in feature_df.iterrows():
            if row[avg_feature + '_avg_team_blue'] == row[avg_feature + '_avg_team_red']:
                feature_df.at[index, avg_feature] = 0
            if row[avg_feature + '_avg_team_blue'] > row[avg_feature + '_avg_team_red']:
                feature_df.at[index, avg_feature] = \
                    row[avg_feature + '_avg_team_red'] / row[avg_feature + '_avg_team_blue']
            else:
                feature_df.at[index, avg_feature] = \
                    (row[avg_feature + '_avg_team_blue'] / row[avg_feature + '_avg_team_red']) * (-1)

    # select features from feature_df
    feature_result_df = df[['matchId']]
    for avg_feature in avg_features:
        feature_result_df.insert(1, avg_feature, feature_df[avg_feature])
        #feature_result_df[avg_feature] = feature_df[avg_feature]


    write('\n-------------------- features dataframe (2) --------------------\n')
    result_df = match_df.merge(feature_result_df, on='matchId', how='inner')
    result_df.to_csv('../tmp/features_2.csv')
    write(result_df.columns)
    write('\n')
    # write(result_df.head())
    # return result_df
```

```python
class SummonerMatch:

    def __init__(self, puuid, game_time, in_game_stats):
        self.puuid = puuid  # identifier of the summoner playing the match
        self.game_time = game_time  # the time until one team won
        self.in_game_stats = in_game_stats  # the summoners in game stats, such as damage done, etc

    def to_string(self):
        return '{ "puuid": "' + self.puuid + '", "game_time": ' + str(self.game_time) \
               + ', "in_game_stats": ' + \
               str(self.in_game_stats) + "}"


class Summoner:

    def __init__(self, match_history, champion):
        self.match_history = match_history  # array of last matches
        self.champion = champion  # champion in active game

    def to_string(self):
        match_history = "["
        for m in self.match_history:
            match_history += m.to_string() + ","
        match_history = match_history[0:len(match_history) - 1]
        match_history += "]"
        return '{ "match_history": ' + match_history + ', "champion": ' + str(self.champion) + "}"


class Match:

    def __init__(self, match_id, summoner1, summoner2, summoner3, summoner4,
                 summoner5, summoner6, summoner7, summoner8, summoner9, summoner10, win):
        self.match_id = match_id  # unique match id
        self.summoner1 = summoner1  # game participants
        self.summoner2 = summoner2
        self.summoner3 = summoner3
        self.summoner4 = summoner4
        self.summoner5 = summoner5
        self.summoner6 = summoner6
        self.summoner7 = summoner7
        self.summoner8 = summoner8
        self.summoner9 = summoner9
        self.summoner10 = summoner10
        self.win = win  # summoner 1-5 game won?

    def to_string(self):
        res = '{ "matchId":"' + str(self.match_id) + '", "Summoner1":' + self.summoner1.to_string() + ', "Summoner2":' \
              + self.summoner2.to_string() + ', "Summoner3":' + self.summoner3.to_string() + ', "Summoner4":' \
              + self.summoner4.to_string() + ', "Summoner5":' + self.summoner5.to_string() + ', "Summoner6":' + self.summoner6.to_string() \
               + ', "Summoner7":' + self.summoner7.to_string() + ', "Summoner8":' + self.summoner8.to_string() \
               + ', "Summoner9":' + self.summoner9.to_string() + ', "Summoner10":' + self.summoner10.to_string() \
               + ', "wins":' + str(self.win) + "}"
        return res.replace("'", '"').replace("True", "true").replace("False", "false")

```

```python
def write(text):
    f = open('../data/log.txt', 'a')
    f.write(str(text))
    f.close()


def read_log():
    f = open('../data/log.txt', 'r')
    return f.read()
```

```python
import pandas as pd
from joblib import dump, load
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, accuracy_score, classification_report
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier

from src.model.features import create_features_1
from src.model.file_writer import write
from src.model.data_processing import process_data


def train_model(df, classifier):
    # select features
    X = df[['kills', 'deaths', 'assists', 'damageDealtToObjectives', 'dragonKills', 'goldEarned',
            'totalDamageDealt', 'turretKills', 'visionScore', 'win']]

    # select target object
    y = df.game_result

    # split into validation and training data
    train_X, val_X, train_y, val_y = train_test_split(X, y, test_size=0.35, random_state=2)

    # specify model
    prediction_model = None

    if classifier == 1:
        prediction_model = RandomForestClassifier(random_state=2)
        write('---------- Random Forest Classifier ----------\n')
    elif classifier == 2:
        prediction_model = DecisionTreeClassifier(random_state=3)
        write('---------- Decision Tree Classifier ----------\n')
    elif classifier == 3:
        prediction_model = MLPClassifier(random_state=3)
        write('---------- MLP Classifier ----------\n')
    elif classifier == 4:
        prediction_model = KNeighborsClassifier(n_neighbors=5)
        write('---------- K Neighbors Classifier ----------\n')
    else:
        print('\n---------- Select Classifier! ----------\n')

    # fit model with train_X and train_y
    if prediction_model is not None:
        prediction_model.fit(train_X, train_y)

        # predictions on val_X with model
        val_predictions = prediction_model.predict(val_X)

        # calculate accuracy

        write('\naccuracy:\n')
        write(accuracy_score(val_y, val_predictions))

        write('\nmean absolute error:\n')
        write(mean_absolute_error(val_y, val_predictions))

        write('\nclassification report:\n')
        write(classification_report(val_y, val_predictions))
        return prediction_model
```

```python
    else:
        write('\n---------- prediction_model is none ----------\n')


def save_model(model, name):
    dump(model, '../model/trained_models/' + name + '.joblib')


def load_model(name):
    return load('../model/trained_models/' + name + '.joblib')


def predict_with_model(model, csv_path):

    # select features
    features = ['kills', 'deaths', 'assists', 'damageDealtToObjectives', 'dragonKills', 'goldEarned',
                'totalDamageDealt', 'turretKills', 'visionScore', 'win']

    # read and process data of match
    match_df = pd.read_csv(csv_path)
    processed_match_df = process_data(match_df, features)

    # create features (1)
    create_features_1(processed_match_df, features)
    features_1 = pd.read_csv('../tmp/features_1.csv')

    feature_result = features_1.drop(['Unnamed: 0', 'matchId', 'game_result'], axis=1)

    # return prediction on input features
    return model.predict(feature_result)

```

```python
import json
import pandas as pd
from src.model.file_writer import write


def process_data(df, features):
    pd.set_option('display.max_columns', None)

    # create matches_df, game_result = 1 => blue team wins
    matches_df = df[['matchId', 'wins']]
    matches_df.insert(2, 'game_result', 'null')
    # print(matches_df)
    for index, row in matches_df.iterrows():
        # print(row['wins'])
        if row.wins == True:
            matches_df.at[index, 'game_result'] = '1'
        else:
            matches_df.at[index, 'game_result'] = '0'
    #print(matches_df.head())
    matches_df = matches_df.drop('wins', 1)

    # create summoner dataframes
    summoner_1 = df[['matchId', 'Summoner1.match_history']]
    summoner_2 = df[['matchId', 'Summoner2.match_history']]
    summoner_3 = df[['matchId', 'Summoner3.match_history']]
    summoner_4 = df[['matchId', 'Summoner4.match_history']]
    summoner_5 = df[['matchId', 'Summoner5.match_history']]
    summoner_6 = df[['matchId', 'Summoner6.match_history']]
    summoner_7 = df[['matchId', 'Summoner7.match_history']]
    summoner_8 = df[['matchId', 'Summoner8.match_history']]
    summoner_9 = df[['matchId', 'Summoner9.match_history']]
    summoner_10 = df[['matchId', 'Summoner10.match_history']]

    # array with selected game stats
    avg_features = features

    # create avg features for each summoner (player)
    summoner_1 = create_avg_features(summoner_1, '1', avg_features)
    summoner_2 = create_avg_features(summoner_2, '2', avg_features)
    summoner_3 = create_avg_features(summoner_3, '3', avg_features)
    summoner_4 = create_avg_features(summoner_4, '4', avg_features)
    summoner_5 = create_avg_features(summoner_5, '5', avg_features)
    summoner_6 = create_avg_features(summoner_6, '6', avg_features)
    summoner_7 = create_avg_features(summoner_7, '7', avg_features)
    summoner_8 = create_avg_features(summoner_8, '8', avg_features)
    summoner_9 = create_avg_features(summoner_9, '9', avg_features)
    summoner_10 = create_avg_features(summoner_10, '10', avg_features)

    # join matches_df with summoner dataframes
    matches_df = pd.merge(matches_df, summoner_1, on='matchId', how='inner')
    matches_df = pd.merge(matches_df, summoner_2, on='matchId', how='inner')
    matches_df = pd.merge(matches_df, summoner_3, on='matchId', how='inner')
    matches_df = pd.merge(matches_df, summoner_4, on='matchId', how='inner')
    matches_df = pd.merge(matches_df, summoner_5, on='matchId', how='inner')
    matches_df = pd.merge(matches_df, summoner_6, on='matchId', how='inner')
    matches_df = pd.merge(matches_df, summoner_7, on='matchId', how='inner')
    matches_df = pd.merge(matches_df, summoner_8, on='matchId', how='inner')
    matches_df = pd.merge(matches_df, summoner_9, on='matchId', how='inner')
    matches_df = pd.merge(matches_df, summoner_10, on='matchId', how='inner')

    # write csv file
```

```
    write('-------------------- result dataframe --------------------\n')
    matches_df.to_csv('../data/result_data_model.csv')
    write(matches_df.columns)
    # write(matches_df.head())
    return matches_df


def create_avg_features(summoner_dataframe, summoner_number, avg_features):
    # iterate over avg_features
    for avg_feature in avg_features:

        # create feature columns
        summoner_dataframe.insert(2, 'summoner_' + summoner_number + '_' + avg_feature + '_avg', '')

        # iterate over
        for index, row in summoner_dataframe.iterrows():

            # transform and load json
            match_history = '{"match_history":' + row['Summoner' + summoner_number + '.match_history'] \
                .replace("'", '"').replace("True", "true").replace("False", "false") + '}'
            match_history_json = json.loads(match_history)['match_history']

            # set counters
            match_count = 0
            features_count = 0
            none_count = 0

            # iterate over in game stats
            for x in match_history_json:
                if x['in_game_stats'] == 'None':
                    none_count += 1
                else:
                    match_count += 1
                    features_count += x['in_game_stats'][str(avg_feature)]
                    # if avg_feature == 'win': print('features_count: '+str(x['in_game_stats'][str(avg_feature)]))

            # calculate avg
            summoner_dataframe.at[index, 'summoner_' + summoner_number + '_' + avg_feature + '_avg'] = features_count / (
                        match_count - none_count)
            # if avg_feature == 'win': print('AVG : ' + str(features_count / (match_count - none_count)))
    return summoner_dataframe.drop('Summoner' + summoner_number + '.match_history', 1)

```

```python
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from src.model.data_processing import process_data
from src.model.features import create_features_1, create_features_2
from src.model.file_writer import write, read_log
from src.model.model_training import train_model, save_model

write('\n--------------- START ---------------\n')

# select in game statistics features
features = ['kills', 'deaths', 'assists', 'damageDealtToObjectives', 'dragonKills', 'goldEarned',
            'totalDamageDealt', 'turretKills', 'visionScore', 'win']

# read and process csv data generated by api script '../tmp/features_1.csv'
# result_matches = pd.read_csv('../tmp/temp_match.csv')

result_matches = pd.read_csv('../data/matches.csv')
# matches_2 = pd.read_csv('../data/matches2.csv')
# matches_3 = pd.read_csv('../data/matches3.csv')
# matches_4 = pd.read_csv('../data/matches4.csv')
# matches_5 = pd.read_csv('../data/matches5.csv')
# result_matches = pd.concat([matches_2, matches_3, matches_4, matches_5], ignore_index=True)
# result_matches.to_csv('../data/matches.csv')
result_data_model = process_data(result_matches, features)

# check result data model
write('\ndescription:')
write(result_data_model[['matchId', 'game_result', 'summoner_1_win_avg']].describe())
write(result_data_model.shape[0])

# create features with result_data_model
create_features_1(result_data_model, features)
create_features_2(result_data_model, features)

# analyze features
features_for_graph = ['kills', 'deaths', 'assists', 'damageDealtToObjectives', 'dragonKills', 'goldEarned',
                      'totalDamageDealt', 'turretKills', 'visionScore', 'win']
features_df = pd.read_csv('../tmp/features_1.csv')
sns.relplot(x="value", y="game_result", col="variable",
            data=features_df.melt(id_vars='game_result', value_vars=features_for_graph),
            facet_kws=dict(sharex=False))
plt.show()

# train prediction model
model_1 = train_model(pd.read_csv('../tmp/features_1.csv'), 1)  # RandomForestClassifier
# save_model(model_1, 'random_forest')
model_2 = train_model(pd.read_csv('../tmp/features_1.csv'), 2)  # DecisionTreeClassifier
# save_model(model_2, 'decision_tree')
# model_3 = train_model(pd.read_csv('../tmp/features_1.csv'), 3)  # MLPClassifier
# save_model(model_3, 'mlp')
# model_4 = train_model(pd.read_csv('../tmp/features_1.csv'), 4)  # KNeighborsClassifier
# save_model(model_4, 'k_neighbors')

print(read_log())
```