```
import numpy as np
import pandas as pd
from joblib import load
from sklearn.preprocessing import LabelBinarizer
from xgboost import XGBClassifier
import timeit
import warnings
warnings.filterwarnings('ignore')

##################################################################
def age_to_bucket(val):
  """
  Here we are mapping bucket value to the corresponding age value
  When a value enters the function the age_bucket_dict is looped individually.
  If the value is in the particular array then its corresponding key will be returned.
  Here each key value consist of 5 consecutive age while the last one consist og 6 consecutive age.

  parameter:
    val    : int

  returns:
    key    : string
  """
  age_bucket_dict={'bucket1':np.arange(15,20),'bucket2':np.arange(20,25),
                  'bucket3':np.arange(25,30),'bucket4':np.arange(30,35),
                  'bucket5':np.arange(35,40),'bucket6':np.arange(40,45),
                  'bucket7':np.arange(45,50),'bucket8':np.arange(50,55),
                  'bucket9':np.arange(55,60),'bucket10':np.arange(60,65),
                  'bucket11':np.arange(65,70),'bucket12':np.arange(70,75),
                  'bucket13':np.arange(75,80),'bucket14':np.arange(80,85),
                  'bucket15':np.arange(85,90),'bucket16':np.arange(90,95),
                  'bucket17':np.arange(95,100),'bucket18':np.arange(100,105),
                  'bucket19':np.arange(105,110),'bucket20':np.arange(110,116),
                  'nullbucket': [-1]
                  }
  for key,value in age_bucket_dict.items():
    if val in value:
      return key

##################################################################
def sessions_col_names():
  """
  Returns the column names of sessions data.

  parameter:
  index_of_action, index_of_action_details, index_of_action_type, index_of_device_type

  returns:
  column_names: list (column names of sessions dataset)

  """
  index_of_action=pd.read_csv('index_of_action.csv',index_col=0).T
  index_of_action_details=pd.read_csv('index_of_action_details.csv',index_col=0).T
  index_of_action_type=pd.read_csv('index_of_action_type.csv',index_col=0).T
  index_of_device_type=pd.read_csv('index_of_device_type.csv',index_col=0).T


  column_names=[]

  #count
  column_names.append('id')
  column_names.append('no_of_records')
  to_add_list=[0]*len(index_of_action.columns)
  for i in index_of_action.keys():
    to_add_list[index_of_action[i][0]]=i+' (count)'
  column_names=column_names+to_add_list
  to_add_list=[0]*len(index_of_action_details.columns)
  for i in index_of_action_details.keys():
    to_add_list[index_of_action_details[i][0]]=i+' (count)'
  column_names=column_names+to_add_list
  to_add_list=[0]*len(index_of_action_type.columns)
  for i in index_of_action_type.keys():
    to_add_list[index_of_action_type[i][0]]=i+' (count)'
  column_names=column_names+to_add_list
  to_add_list=[0]*len(index_of_device_type.columns)
  for i in index_of_device_type.keys():
    to_add_list[index_of_device_type[i][0]]=i+' (count)'
  column_names=column_names+to_add_list
  column_names.append('sum_of_secs_elapsed')
  column_names.append('mean_of_secs_elapsed')
  column_names.append('std_of_secs_elapsed')
  column_names.append('median_of_secs_elapsed')
  column_names.append('less_than_equal_60sec_count')
  column_names.append('1min_to_10min_count')
  column_names.append('10min_to_1hr_count')
  column_names.append('1hr_to_1day_count')
  column_names.append('more_than_1day_count')
  to_add_list=[0]*len(index_of_action.columns)
  for i in index_of_action.keys():
    to_add_list[index_of_action[i][0]]=i+' (action presence)%'
  column_names=column_names+to_add_list
  to_add_list=[0]*len(index_of_action_type.columns)
  for i in index_of_action_type.keys():
    to_add_list[index_of_action_type[i][0]]=i+' (action type presence)%'
  column_names=column_names+to_add_list

  return column_names

##################################################################
def dcg_score(y_true, y_score, k=5):
    """
    Discounted Cummulative Gain(dcg)= sum(i:1 to k) ((2**relavance(i))-1)/(log2(i+1))
    Here y_true is array which is label encoded and y_score is the probability score
    eg:
      y_true=[0,0,0,1,0,0,0,0,0,0,0,0] (For destination ES)
      y_score=[0,0,0,0.70,0,0.025,0,0.15,0,0,0.1,0.025]
    we argsort in highest order of y_score
    order=[3,6,9,5,12,......]
    now we take the top 5 index of the order and pick values of those top 5 index from y_true.
    gain=[(2**1)-1,(2**0)-1,(2**0)-1,...]
    discount=[log2(1+1),log2(2+1),log2(3+1),...]
    dcg=sum(gain/discount)

    parameter:
      y_true : numpy array of shape [n_samples,number_of_classes]
      y_score: numpy array of shape [n_samples,number_of_classes]
      k      : int (default=5)

    returns:
      dcg    : float
    """

    order = np.argsort(y_score)[::-1]
    y_true = np.take(y_true, order[:k])

    gain = 2 ** y_true - 1

    discounts = np.log2(np.arange(len(y_true)) + 2)
    return np.sum(gain / discounts)

def ndcg_score(ground_truth, predictions, k=5):
    """
    Normalized Discounted Cummulative Gain (ndcg)= Discounted Cummulative Gain/Ideal Discounted Cummulative Gain
    idcg=sum(i: 1 to k)((2**true_relavance(i))-1)/(log2(i+1))
    In idcg the probability score (ie y_score) will be y_true

    parameter:
      ground_truth: numpy array of shape [n_samples,]
      predictions : numpy array of shape [n_samples,number_of_classes]
      k           : int (default=5)

    returns:
      score       : float
      (average ndcg score)
    """

    lb = LabelBinarizer()
    lb.fit(range(predictions.shape[1] + 1))
    T = lb.transform(ground_truth)

    scores = []

    # Iterate over each y_true and compute the DCG score
    for y_true, y_score in zip(T, predictions):
        actual = dcg_score(y_true, y_score, k)
        best = dcg_score(y_true, y_true, k)
        if best == 0:
            best = 0.000000001
        score = float(actual) / float(best)
        scores.append(score)

    return np.mean(scores)
```

```
##################################################################
def preprocess_users(user_data):
  """Function that is defined to preprocess the users data"""

  user_data=user_data.reset_index()

  selected_language=np.load('selected_language.npz.npy',allow_pickle=True)
  selected_browsers=np.load('selected_browsers.npz.npy',allow_pickle=True)
  selected_aff_provider=np.load('selected_aff_provider.npz.npy',allow_pickle=True)
  selected_aff_track=np.load('selected_aff_track.npz.npy',allow_pickle=True)

  user_data['date_account_created']=pd.to_datetime(user_data['date_account_created'])
  user_data['day_account_created']=pd.DatetimeIndex(user_data['date_account_created']).dayofweek
  user_data['year_account_created']=pd.DatetimeIndex(user_data['date_account_created']).year
  user_data['timestamp_first_active']=pd.to_datetime(user_data['timestamp_first_active'],format='%Y%m%d%H%M%S')
  user_data['day_first_active']=pd.DatetimeIndex(user_data['timestamp_first_active']).dayofweek
  user_data['hour_first_active']=pd.DatetimeIndex(user_data['timestamp_first_active']).hour

  for i in range(len(user_data['age'])):
    if user_data['age'][i]>150 and user_data['age'][i]<2010:
      user_data['age'][i]=user_data['year_account_created'][i]-user_data['age'][i]
  user_data['age']=user_data.age.apply(lambda x: 115 if x>115 else x)
  user_data['age']=user_data.age.apply(lambda x: 15 if x<15 else x)
  user_data['age'].fillna(-1,inplace=True)
  user_data['first_affiliate_tracked'].fillna('unknown',inplace=True)
  user_data['age_bucket']=user_data.age.apply(age_to_bucket)
  user_data['language']=user_data.language.apply(lambda x: 'Other' if x not in selected_language else x)
  user_data['first_browser']=user_data.first_browser.apply(lambda x: 'Other' if x not in selected_browsers else x)
  user_data['affiliate_provider']=user_data.affiliate_provider.apply(lambda x: 'smallcontribution' if x not in selected_aff_p
  user_data['first_affiliate_tracked']=user_data.first_affiliate_tracked.apply(lambda x: 'others' if x not in selected_aff_tr
  user_data['no_of_nans']=0
  for i in range(len(user_data)):
    if user_data['age'][i]==-1:
      user_data['no_of_nans'][i]+=1
    if user_data['gender'][i]=='-unknown-':
      user_data['no_of_nans'][i]+=1
    if user_data['first_affiliate_tracked'][i]=='untracked':
      user_data['no_of_nans'][i]+=1

  user_data.drop(['date_account_created','timestamp_first_active',
                 'year_account_created','date_first_booking'],axis=1,inplace=True)

  age_bucket_ohe=load('age_bucket_ohe.bin')
  user_age_bucket_ohe=pd.DataFrame(age_bucket_ohe.transform(user_data['age_bucket'].values.reshape(-1,1)).todense())

  gender_ohe=load('gender_ohe.bin')
  user_gender_ohe=pd.DataFrame(gender_ohe.transform(user_data['gender'].values.reshape(-1,1)).todense())

  language_ohe=load('language_ohe.bin')
  user_language_ohe=pd.DataFrame(language_ohe.transform(user_data['language'].values.reshape(-1,1)).todense())

  signup_method_ohe=load('signup_method_ohe.bin')
  user_signup_method_ohe=pd.DataFrame(signup_method_ohe.transform(user_data['signup_method'].values.reshape(-1,1)).todense())

  affiliate_channel_ohe=load('affiliate_channel_ohe.bin')
  user_affiliate_channel_ohe=pd.DataFrame(affiliate_channel_ohe.transform(user_data['affiliate_channel'].values.reshape(-1,1)

  affiliate_provider_ohe=load('affiliate_provider_ohe.bin')
  user_affiliate_provider_ohe=pd.DataFrame(affiliate_provider_ohe.transform(user_data['affiliate_provider'].values.reshape(-1

  first_affiliate_tracked_ohe=load('first_affliate_tracked_ohe.bin')
  user_first_affiliate_tracked_ohe=pd.DataFrame(first_affiliate_tracked_ohe.transform(user_data['first_affiliate_tracked'].va

  signup_app_ohe=load('signup_app_ohe.bin')
  user_signup_app_ohe=pd.DataFrame(signup_app_ohe.transform(user_data['signup_app'].values.reshape(-1,1)).todense())

  first_device_type_ohe=load('first_device_type_ohe.bin')
  user_first_device_type_ohe=pd.DataFrame(first_device_type_ohe.transform(user_data['first_device_type'].values.reshape(-1,1)

  first_browser_ohe=load('first_browser_ohe.bin')
  user_first_browser_ohe=pd.DataFrame(first_browser_ohe.transform(user_data['first_browser'].values.reshape(-1,1)).todense())

  preprocessed_user=pd.concat([user_age_bucket_ohe,user_gender_ohe,user_language_ohe,user_signup_method_ohe,
                              user_affiliate_channel_ohe,user_affiliate_provider_ohe,user_first_affiliate_tracked_ohe,
                              user_signup_app_ohe,user_first_device_type_ohe,user_first_browser_ohe],axis=1)

  user_data.reset_index(inplace=True)
  user_data.drop('index',axis=1,inplace=True)

  if 'country_destination' in user_data.columns:
    preprocessed_user=pd.concat([user_data[['id','country_destination','day_account_created',
                                          'day_first_active','hour_first_active',
                                          'signup_flow','age','no_of_nans']],preprocessed_user],axis=1,join='inner')
  else:
    preprocessed_user=pd.concat([user_data[['id','day_account_created',
                                          'day_first_active','hour_first_active',
                                          'signup_flow','age','no_of_nans']],preprocessed_user],axis=1,join='inner')

  return preprocessed_user

#############################################################
def preprocess_sessions(sessions_data):
  """Function that preprocess sessions dataset"""

  sessions_data=sessions_data.reset_index()

  index_of_action=pd.read_csv('index_of_action.csv',index_col=0).T
  index_of_action_details=pd.read_csv('index_of_action_details.csv',index_col=0).T
  index_of_action_type=pd.read_csv('index_of_action_type.csv',index_col=0).T
  index_of_device_type=pd.read_csv('index_of_device_type.csv',index_col=0).T

  sessions_data.action.fillna("unkown_action",inplace=True)
  sessions_data.action_detail.fillna('unknow_action_detail',inplace=True)
  sessions_data.action_type.fillna('unkown_action_type',inplace=True)
  sessions_data.device_type.fillna('unkown_device_type',inplace=True)

  is_in_train={}
  for i in sessions_data.action.unique():
    is_in_train[i]=0
  for i in index_of_action.columns:
    is_in_train[i]=1
  sessions_data.action=sessions_data.action.apply(lambda x: 'OTHER' if is_in_train[x]==0 else x)
  is_in_train={}
  for i in sessions_data.action_type.unique():
    is_in_train[i]=0
  for i in index_of_action_type.columns:
    is_in_train[i]=1
  sessions_data.action_type=sessions_data.action_type.apply(lambda x: 'unkown_action_type' if is_in_train[x]==0 else x)
  is_in_train={}
  for i in sessions_data.action_detail.unique():
    is_in_train[i]=0
  for i in index_of_action_details.columns:
    is_in_train[i]=1
  sessions_data.action_detail=sessions_data.action_detail.apply(lambda x: 'unknow_action_detail' if is_in_train[x]==0 else x)
  is_in_train={}
  for i in sessions_data.device_type.unique():
    is_in_train[i]=0
  for i in index_of_device_type.columns:
    is_in_train[i]=1
  sessions_data.device_type=sessions_data.device_type.apply(lambda x: 'unkown_device_type' if is_in_train[x]==0 else x)

  sess_id_group=sessions_data.groupby(['user_id'])
  user_samples = []
  ln = len(sess_id_group)
  for id_session in sess_id_group:
    details_of_id = id_session[1]
    l = []

    #the id
    l.append(id_session[0])

    #The actual first feature is the number of values.
    l.append(len(details_of_id))

    #For Action
    #Number of times unique action value occurs
    feature_action=[0]*len(index_of_action.columns)
    for i,action in enumerate(details_of_id.action.values):
      feature_action[index_of_action[action][0]]+=1
    l=l+feature_action

    #For action_detail
    #Number of times unique action_detail value occurs
    feature_action_detail=[0]*len(index_of_action_details.columns)
    for i,action in enumerate(details_of_id.action_detail.values):
      feature_action_detail[index_of_action_details[action][0]]+=1
    l=l+feature_action_detail

    #For action_type
    #Number of times unique action_type value occurs
    feature_action_type=[0]*len(index_of_action_type.columns)
    for i,action in enumerate(details_of_id.action_type.values):
      feature_action_type[index_of_action_type[action][0]]+=1
    l=l+feature_action_type

    #For Device type
    #Number of times unique device_type value occurs
    feature_device_type=[0]*len(index_of_device_type.columns)
    for i,device in enumerate(details_of_id.device_type.values):
      feature_device_type[index_of_device_type[device][0]]+=1
    l=l+feature_device_type

    #For seconds elapsed
    secs=details_of_id.secs_elapsed.fillna(0).values
    feature_secs=[0]*4
    if len(secs)>0:
      feature_secs[0]=np.sum(secs)
      feature_secs[1]=np.mean(secs)
      feature_secs[2]=np.std(secs)
      feature_secs[3]=np.median(secs)
    l=l+feature_secs

    #Secs elapsed representation
    feature_action_secs_elapsed_represent=[0]*5
    details_of_id.secs_elapsed.fillna(0,inplace=True)
    if len(secs)>0:
      for i,action in enumerate(details_of_id.action.unique()):
        sec_elapsed_sum=details_of_id[details_of_id.action==action]['secs_elapsed'].values.sum()
        if sec_elapsed_sum<=60:#less than a minute
          feature_action_secs_elapsed_represent[0]+=1
        elif sec_elapsed_sum>60 and sec_elapsed_sum<=600:#between 1 minute to 10 minute
          feature_action_secs_elapsed_represent[1]+=1
        elif sec_elapsed_sum>600 and sec_elapsed_sum<=7200:#between 10 minutes to 2 hours
          feature_action_secs_elapsed_represent[2]+=1
        elif sec_elapsed_sum>7200 and sec_elapsed_sum<=86400:#between 2 hours to 24 hours
          feature_action_secs_elapsed_represent[3]+=1
        elif sec_elapsed_sum>86400: #greater than 24 hours
          feature_action_secs_elapsed_represent[4]+=1
    l=l+feature_action_secs_elapsed_represent

    #Percentage of presence of an action in the id
    feature_action_presence=[0]*len(index_of_action.columns)
    for i,action in enumerate(details_of_id.action.unique()):
      feature_action_presence[index_of_action[action][0]]+=details_of_id[details_of_id.action==action].count()['action']/len(
    l=l+feature_action_presence

    #percentage of presence of action type in id
    action_type_unique_counts=np.unique(details_of_id.action_type,return_counts=True)
    unknown_action_index=np.where(action_type_unique_counts[0]=='unkown_action_type')
    unique_action_type=np.delete(action_type_unique_counts[0],unknown_action_index)
    unique_action_type_count=np.delete(action_type_unique_counts[1],unknown_action_index)
    feature_action_type_presence=[0]*len(index_of_action_type.columns)
    for i,action in enumerate(details_of_id.action_type.unique()):
      if action!='unkown_action_type':
        action_type_index=np.where(unique_action_type==action)
        action_type_count=unique_action_type_count[action_type_index]
        feature_action_type_presence[index_of_action_type[action][0]]+=action_type_count[0]/np.sum(unique_action_type_count)
    l=l+feature_action_type_presence
    user_samples.append(l)

  user_samples=np.array(user_samples)

  column_names=sessions_col_names()

  sessions_user_preprocessed=pd.DataFrame(user_samples,columns=column_names)
  sessions_user_preprocessed.drop(['unkown_action_type (action type presence)%'],axis=1,inplace=True)

  return sessions_user_preprocessed

#####################################################################
def final(user_data,sessions_data=[]):
  """
  Takes in raw data that is user and session data and gives the top 5 preferred destination

  parameter:
    data   : pandas dataframe

  return:
    id,top 5 prefered destination
  """
  sessions_data_present=False
  if len(sessions_data)>=1:
    sessions_data_present=True

  preprocessed_user=preprocess_users(user_data)#preprocessing users data

  #if the sessions dataset is given the following code executes
  if sessions_data_present:
    user_counts=sessions_data['user_id'].value_counts()#Takes count of number of session record each user have
    present_ids=[]#Take ids of those whose session record is given
    null_ids=[]#Takes ids of those whose doesn't have sessions records
    for i in range(len(user_counts)):
        present_ids.append(user_counts.keys()[i])
    left_out_ids=list(set(user_data['id'])-set(sessions_data['user_id']))#Ids that are in users data but not even one record
    null_ids.extend(left_out_ids)

    #present_sessions variable preprocesses sessions dataset for those ids whose session record is given
    if len(present_ids)>=1:
      present_sessions=sessions_data.loc[sessions_data['user_id'].isin(present_ids)]
      preprocessed_present_sessions=preprocess_sessions(present_sessions)
      preprocessed_sessions=preprocessed_present_sessions

    #For those ids whose sessions dataset is not given are given zero values for all the features that are extracted from ses
    if len(null_ids)>=1:
      preprocessed_null_sessions=[]
      for i in range(len(null_ids)):
          null_session=[0]*667
          null_session=[null_ids[i]]+null_session
          preprocessed_null_sessions.append(null_session)
      preprocessed_null_sessions=np.array(preprocessed_null_sessions)
      column_names=sessions_col_names()
      preprocessed_null_sessions=pd.DataFrame(preprocessed_null_sessions,columns=column_names)
      preprocessed_null_sessions.drop(['unkown_action_type (action type presence)%'],axis=1,inplace=True)
      #we are checking if there are some users with session dataset
      #If so we have to concatenate with that of those whose session is not given
      if len(present_ids)>=1:
        preprocessed_sessions=pd.concat([preprocessed_sessions,preprocessed_null_sessions],axis=0)
      else:
        preprocessed_sessions=preprocessed_null_sessions

  #If no sessions data is given the following else code snippet executes
  else:
    null_ids=user_data['id'].tolist()
    preprocessed_null_sessions=[]
    for i in range(len(null_ids)):
        null_session=[0]*667
        null_session=[null_ids[i]]+null_session
        preprocessed_null_sessions.append(null_session)
    preprocessed_null_sessions=np.array(preprocessed_null_sessions)
    column_names=sessions_col_names()
    preprocessed_null_sessions=pd.DataFrame(preprocessed_null_sessions,columns=column_names)
    preprocessed_null_sessions.drop(['unkown_action_type (action type presence)%'],axis=1,inplace=True)
    preprocessed_sessions=preprocessed_null_sessions

  final_df=preprocessed_sessions.merge(preprocessed_user,how='inner',left_on='id',right_on='id')
  final_df.drop(['id'],axis=1,inplace=True)
  xgboost=XGBClassifier()
  xgboost.load_model('xgboost_model.json')
  le=load('class_label_encoder.bin')
  proba_prediction=xgboost.predict_proba(np.array(final_df)).argsort(axis=1)[:,::-1]
  prediction=le.classes_[proba_prediction[:,:5]]
  return preprocessed_sessions['id'],prediction

##########################################################################
def final_2(user_data,target_value,sessions_data=[]):
  """
  Takes in raw data that is user and session data and gives the ndcg score

  parameter:
    data        : pandas dataframe
    target value: pandas dataframe (having id and country_destination)

  return:
    ndcg score
  """
  sessions_data_present=False
  if len(sessions_data)>=1:
    sessions_data_present=True

  preprocessed_user=preprocess_users(user_data)#preprocessing users data

  user_to_target=[]
  #Creating a dataframe that has id and its corresponding destination
  for target_index in range(len(target_value)):
    user_to_target.append([user_data['id'].iloc[target_index],target_value[target_index]])
  user_to_target=pd.DataFrame(user_to_target,columns=['id','target'])

  #if the sessions dataset is given the following code executes
  if sessions_data_present:
    user_counts=sessions_data['user_id'].value_counts()#Takes count of number of session record each user have
    present_ids=[]#Take ids of those whose session record is present
    null_ids=[]#Takes ids of those whose session record is not there.
    for i in range(len(user_counts)):
        present_ids.append(user_counts.keys()[i])
    left_out_ids=list(set(user_data['id'])-set(sessions_data['user_id']))#Ids that are in users data but not even one record
    null_ids.extend(left_out_ids)

    #present_sessions variable preprocesses sessions dataset for those ids whose session record is present
    if len(present_ids)>=1:
      present_sessions=sessions_data.loc[sessions_data['user_id'].isin(present_ids)]
      preprocessed_present_sessions=preprocess_sessions(present_sessions)
      preprocessed_sessions=preprocessed_present_sessions

    #For those ids whose sessions dataset is not given are given zero values for all the features that are extracted from ses
    if len(null_ids)>=1:
      preprocessed_null_sessions=[]
      for i in range(len(null_ids)):
          null_session=[0]*667
          null_session=[null_ids[i]]+null_session
          preprocessed_null_sessions.append(null_session)
      preprocessed_null_sessions=np.array(preprocessed_null_sessions)
      column_names=sessions_col_names()
      preprocessed_null_sessions=pd.DataFrame(preprocessed_null_sessions,columns=column_names)
      preprocessed_null_sessions.drop(['unkown_action_type (action type presence)%'],axis=1,inplace=True)
      if len(present_ids)>=1:
        preprocessed_sessions=pd.concat([preprocessed_sessions,preprocessed_null_sessions],axis=0)
      else:
        preprocessed_sessions=preprocessed_null_sessions

  #If no sessions data is given the following else code snippet executes
  else:
    null_ids=user_data['id'].tolist()
    preprocessed_null_sessions=[]
    for i in range(len(null_ids)):
        null_session=[0]*667
        null_session=[null_ids[i]]+null_session
        preprocessed_null_sessions.append(null_session)
    preprocessed_null_sessions=np.array(preprocessed_null_sessions)
    column_names=sessions_col_names()
    preprocessed_null_sessions=pd.DataFrame(preprocessed_null_sessions,columns=column_names)
    preprocessed_null_sessions.drop(['unkown_action_type (action type presence)%'],axis=1,inplace=True)
    preprocessed_sessions=preprocessed_null_sessions

  final_df=preprocessed_sessions.merge(preprocessed_user,how='inner',left_on='id',right_on='id')
  final_df=final_df.merge(user_to_target,how='inner',left_on='id',right_on='id')
  target=final_df['target']
  final_df.drop(['target'],axis=1,inplace=True)
  final_df.drop(['id'],axis=1,inplace=True)
  xgboost=XGBClassifier()
  xgboost.load_model('xgboost_model.json')
  le=load('class_label_encoder.bin')
  target_value=le.transform(target)
  proba_prediction=xgboost.predict_proba(np.array(final_df))
  ndcg=ndcg_score(target_value,proba_prediction,k=5)
  return ndcg
```