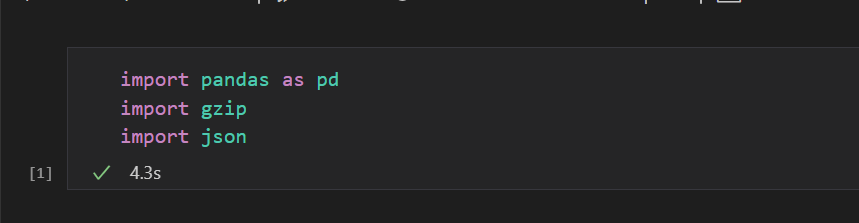
**NAME: VANSHAJ SHARMA**

**ROLLNO: MT23103**

**COURSE: M.TECH CSE**

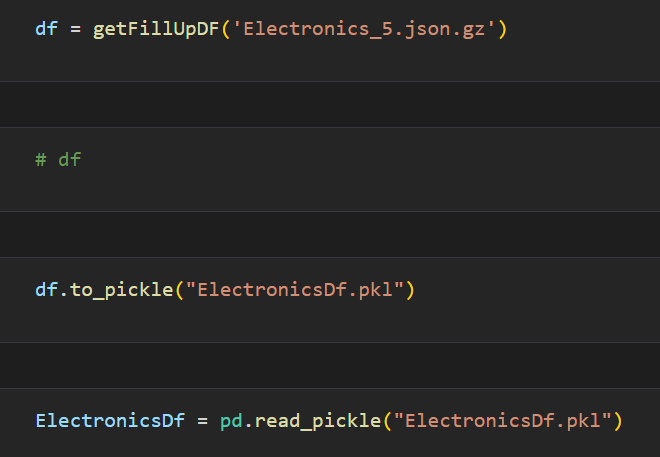
**YEAR: FIRST YEAR**

**Libraries Used :-**

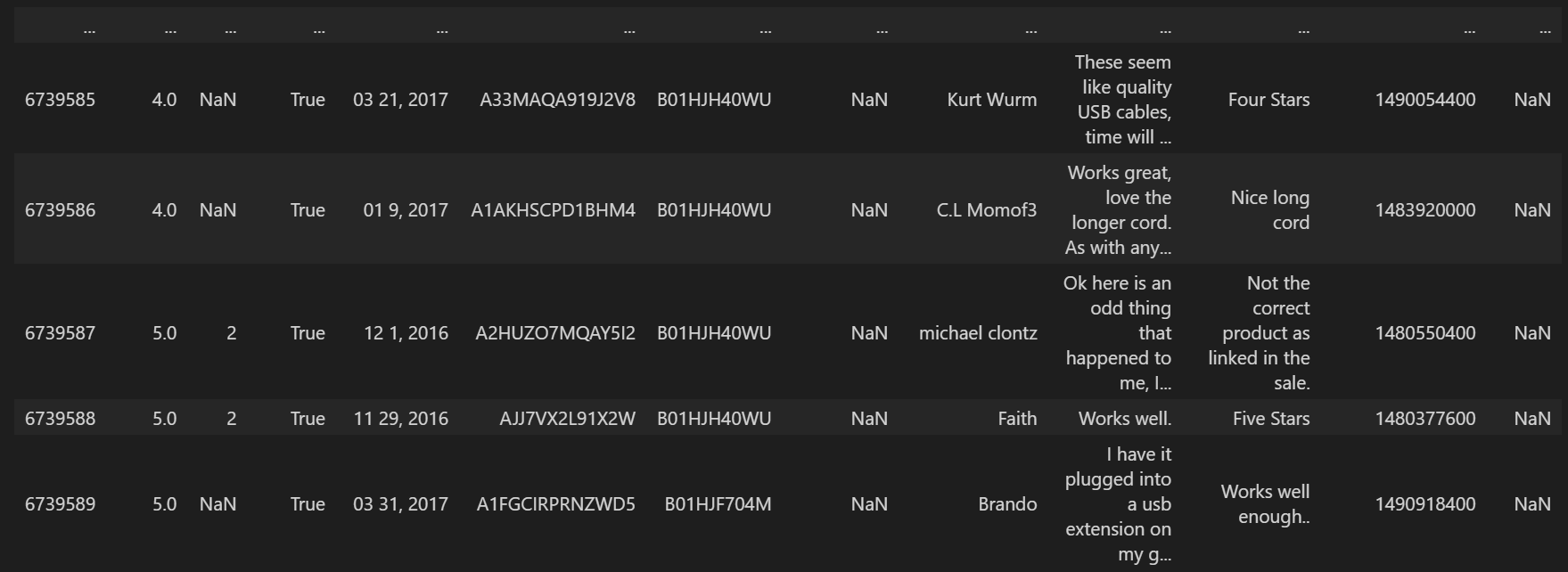
****

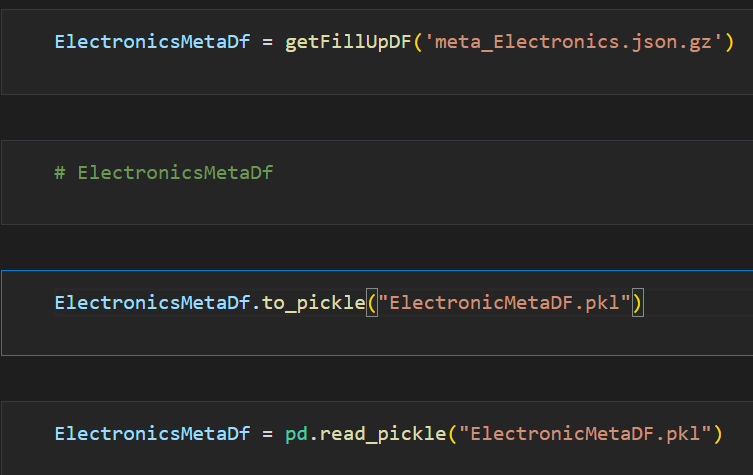
1) Read the file to a dataframe. Remember to keep the product metadata in a distinct dataframe as well.

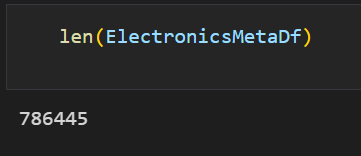








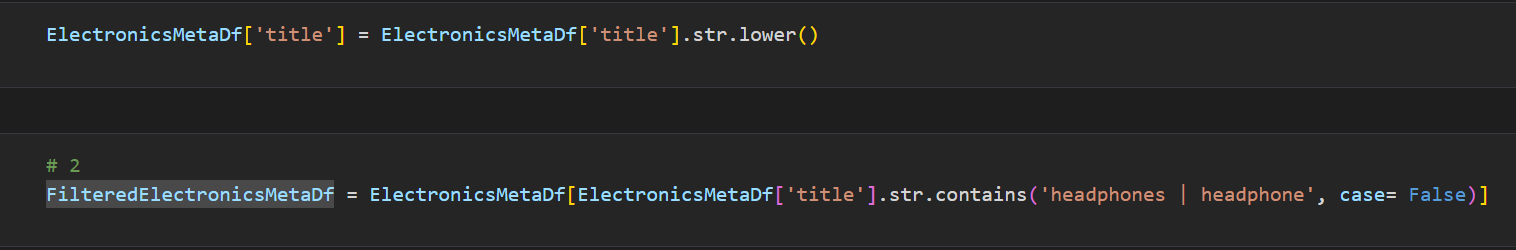




For loading and storing data from Electronic\_5.json.gz and meta\_Electronic.gz into dataframe from getFillUpDf function is used which parse the whole zip file and read the content into dataframe and for parsing each zip file it calls function called parseZip .

It took me 5 hours and 75 min to store data into dataframes.

For storing meta\_data ElectronicsMetaDf dataframe is created and for storing reviews data ElectronicDf is used.

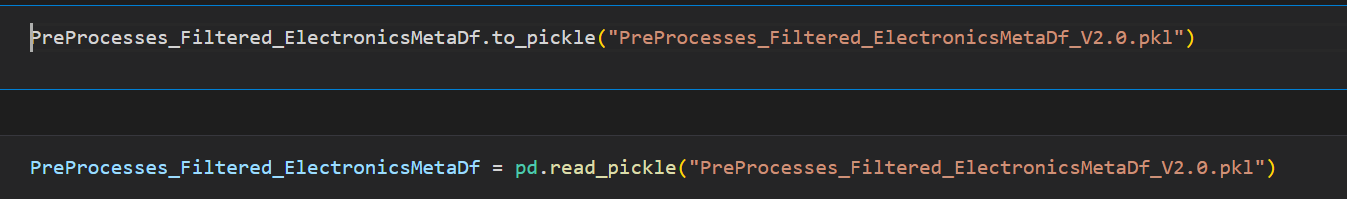
2) Choose a product of your choice. Let's say ‘Headphones’.

In this part we have to choose a product . So I choose headphone . So we can get product ids related to the headphones from ElectronicsMetaDf’s column name title . But there could be many variations of headphones like Heaphones and HeAdphones or headphone. So for that all the titles are converted to lowercase then rows that are having values related to headphones or headphone in title are extracted and saved into PreProcess\_FilteredMetaDf . But this can might have duplicate rows in this dataframe so For removing duplicacy below mentioned line is used.

#citation:- https://stackoverflow.com/a/43855963

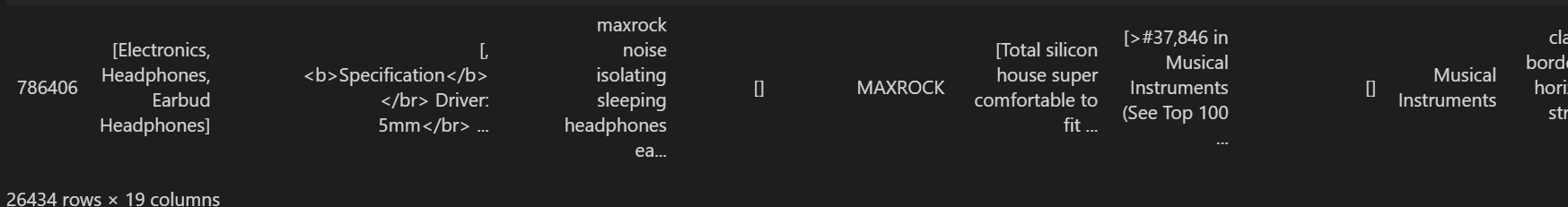
PreProcesses\_Filtered\_ElectronicsMetaDf = PreProcesses\_Filtered\_ElectronicsMetaDf.loc[PreProcesses\_Filtered\_ElectronicsMetaDf.astype(str).drop\_duplicates().index]

As the doing this processing is very time consuming so pickle files are used to store them in local and later when it is required it is fetched from there.









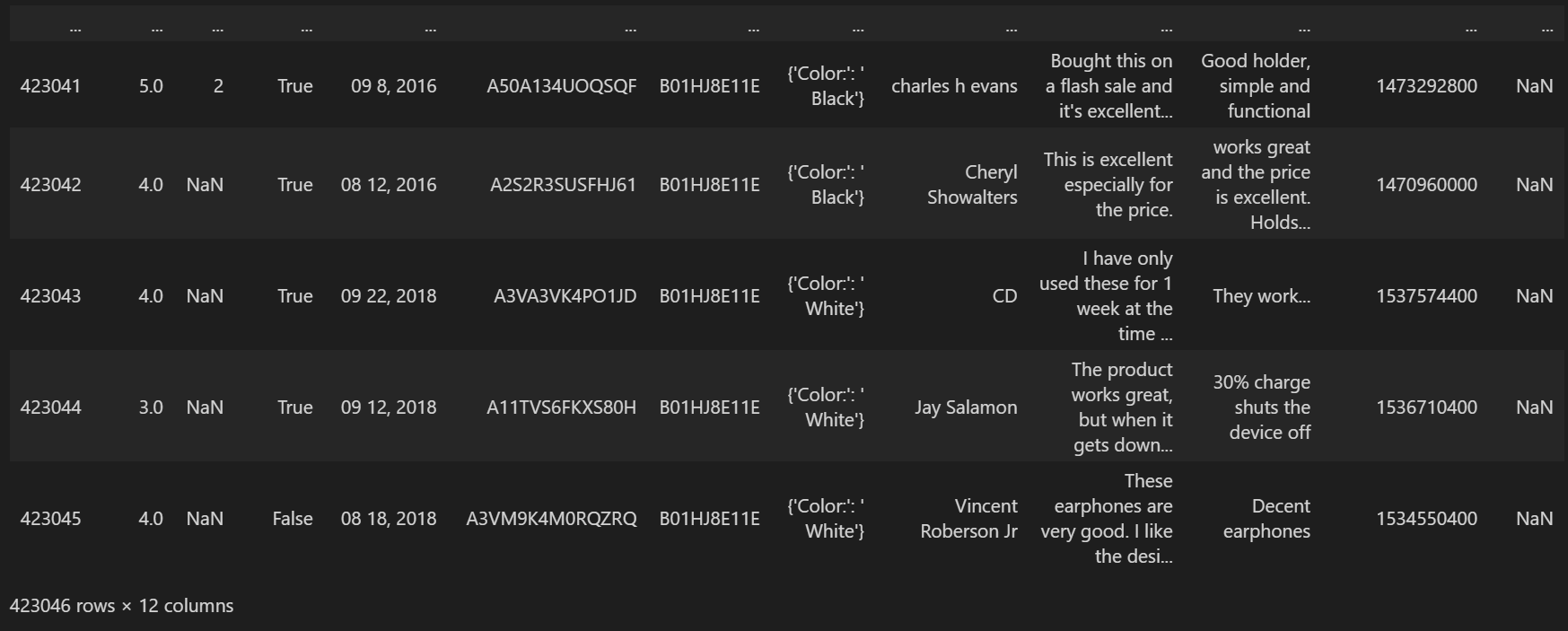
3. Report the total number of rows for the product. Perform appropriate pre-processing as handling missing values, duplicates and other.

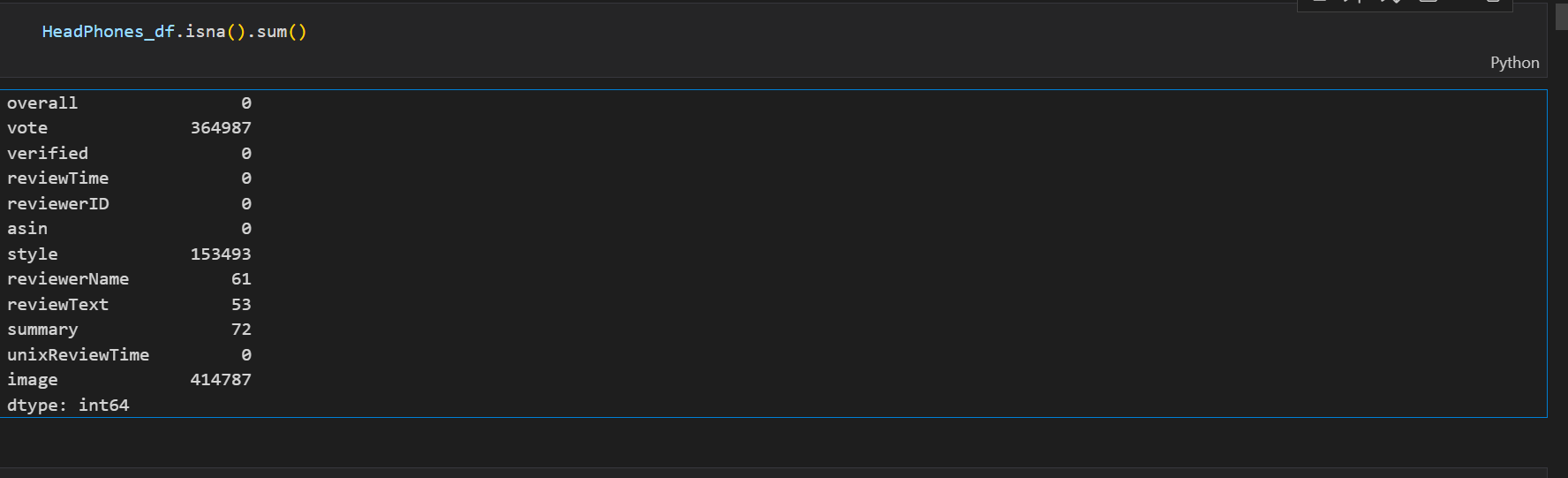


Now for further processing as in later parts review Text required . so, It is essential to create a dataframe that is having reviews that belongs to only headphones product. So for that product is in metadata are used and corresponding reviews are stored on to the dataframe called HeadPhone\_df.

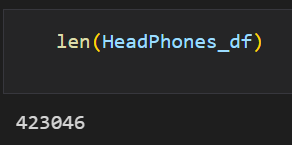
Later that HeadPhone\_df is also stored into a pickle file called Headphones\_review\_df.pkl

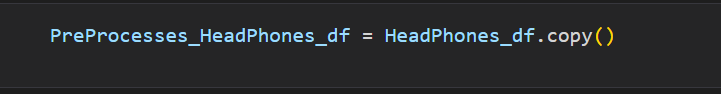


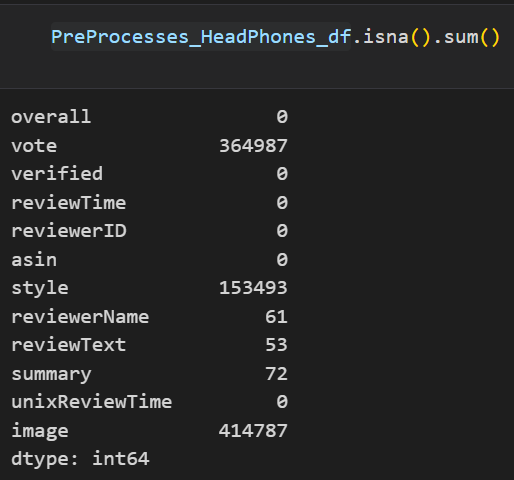


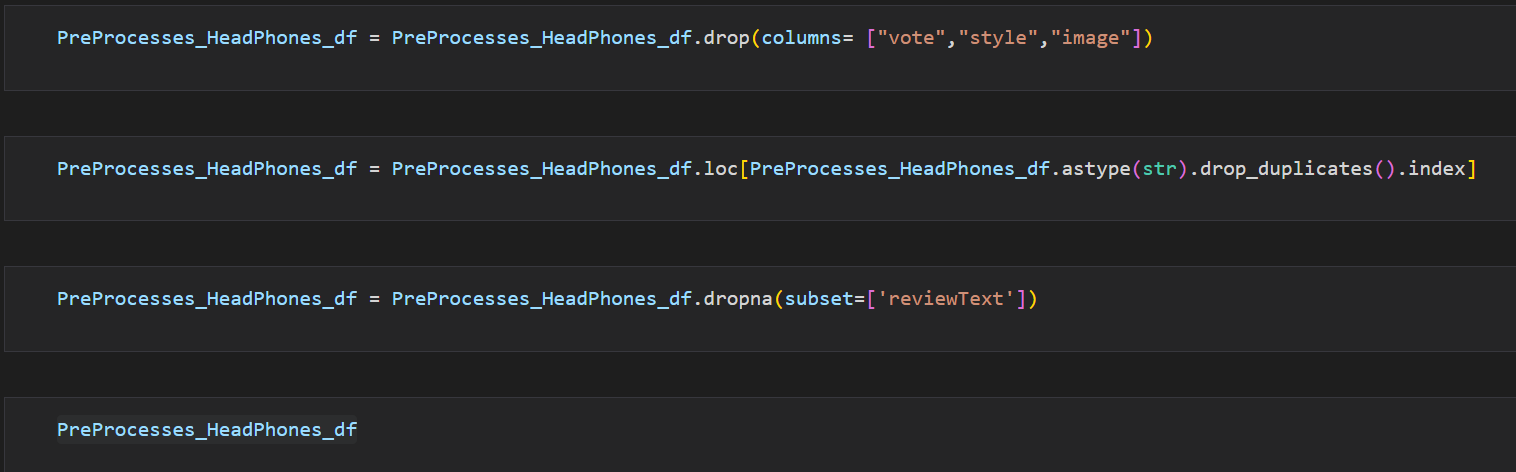


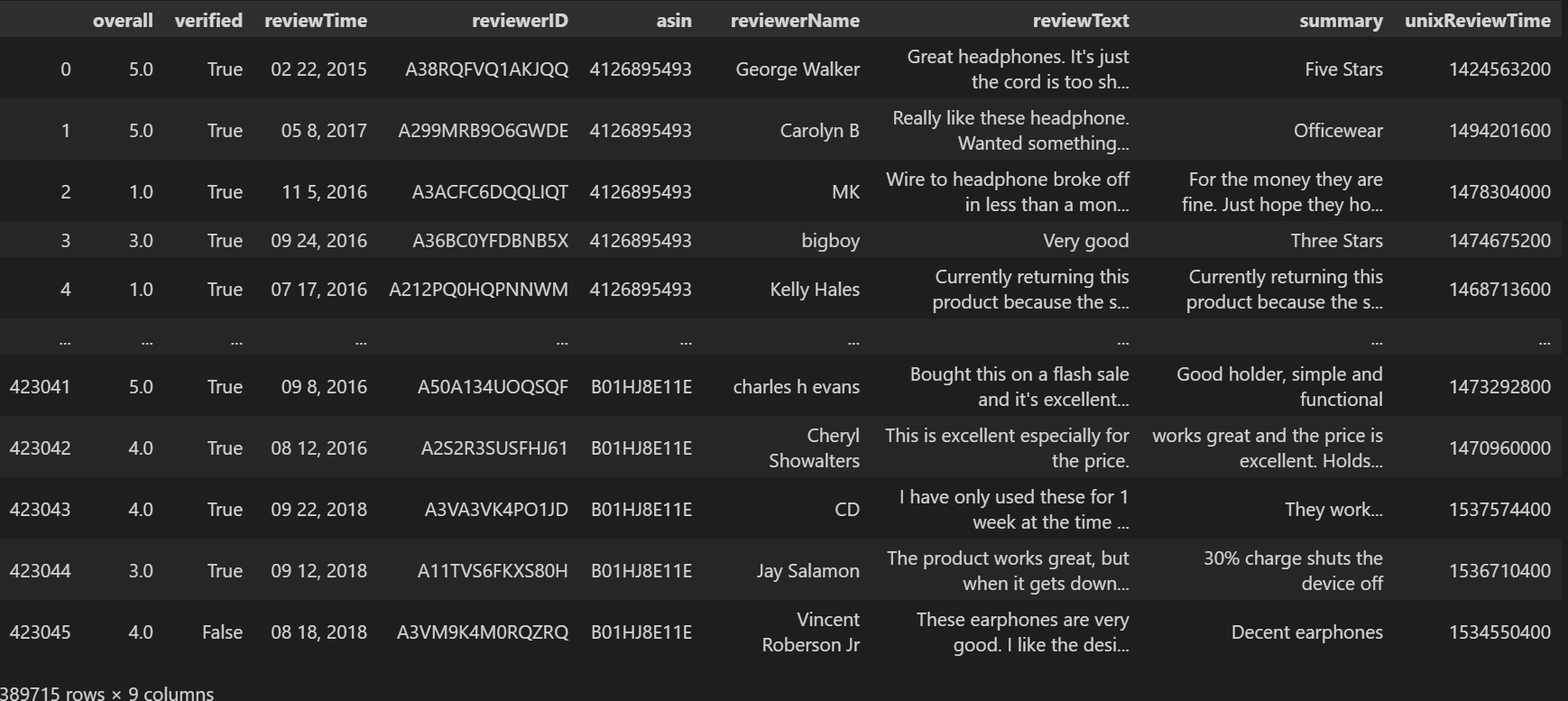
Now in headphones\_df , As there are too many Nan values in vote , style and image , But we don’t required that columns in later parts so only these columns are removed and result is stored into PreProcess\_HeadPhones\_df.

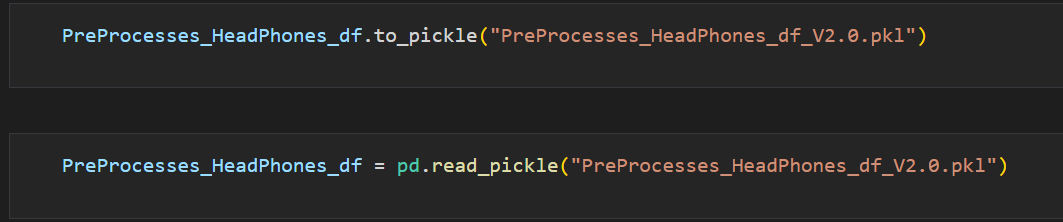


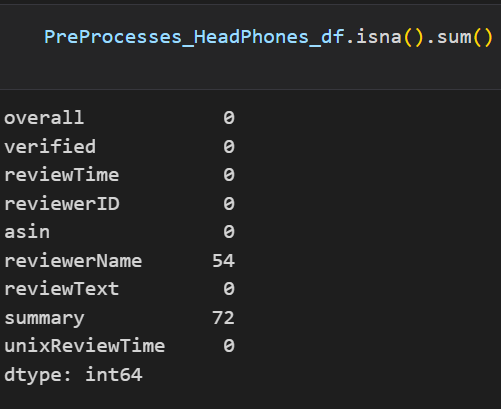












Now after removing columns that had Nan Values , Dataframe seems to be clean and but still some Nan values are here but that doent make any effect that much on the dataset so not removing them.

4. Obtain the Descriptive Statistics of the product as : -

a. Number of Reviews.

b. Average Rating Score.

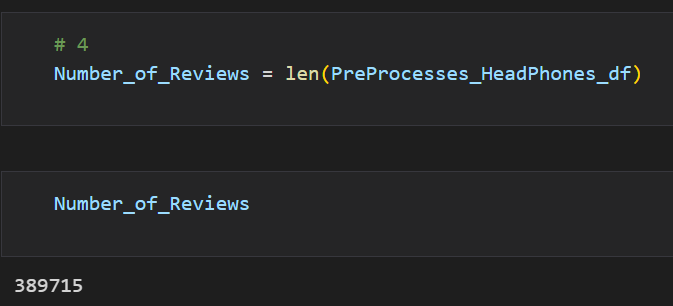
c. Number of Unique Products.

d. Number of Good Rating.

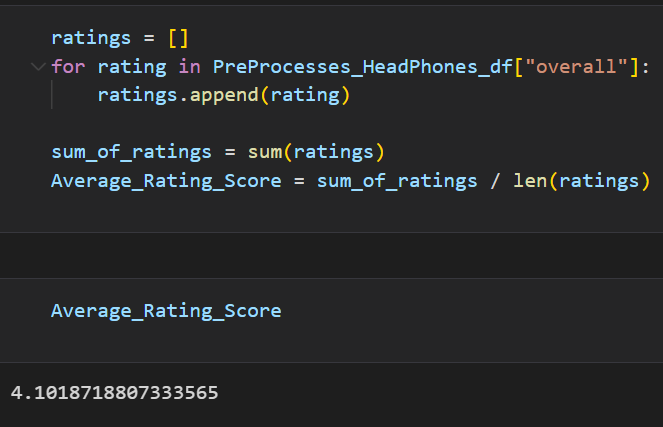
e. Number of Bad Ratings ( Set a threshold of >=3 as ‘Good’ and rest as

‘Bad’), and

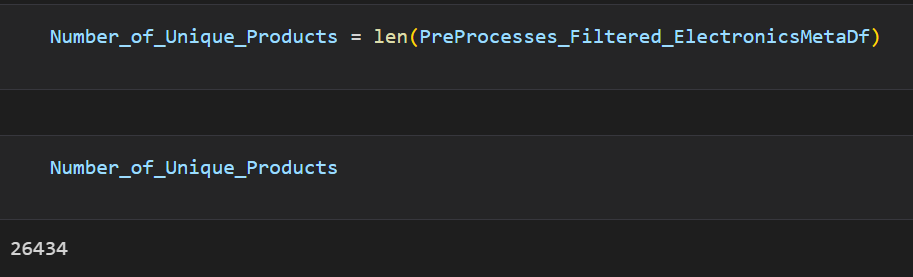
f. Number of Reviews corresponding to each Rating.



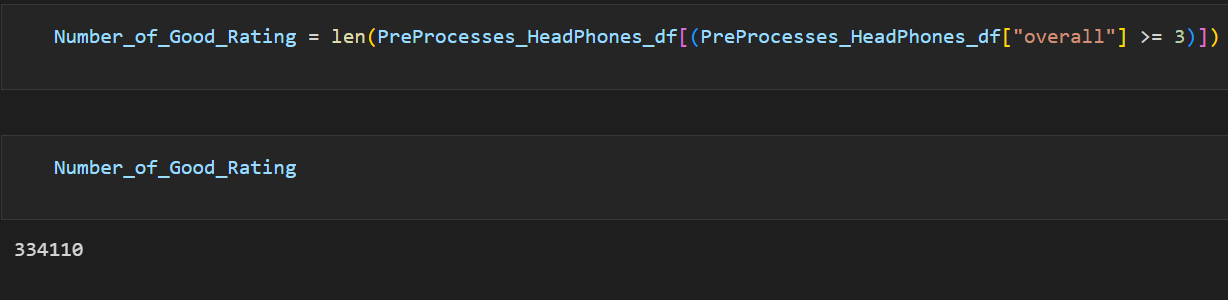
Total number of reviews related to headphones are the only Total Number of Reviews . So the length of PreProcess\_Headphones\_df is the answer.



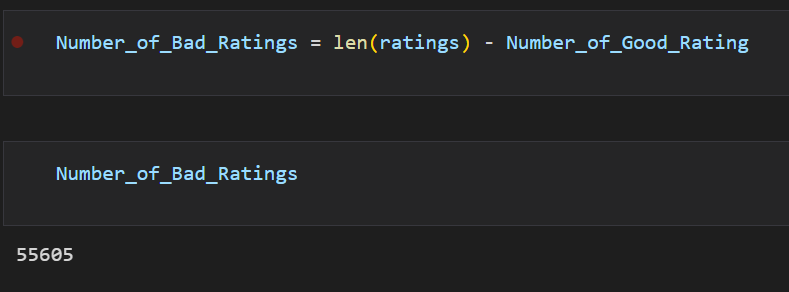
For calculating average rating all the rating among all the reviews related to headphones are averaged out.



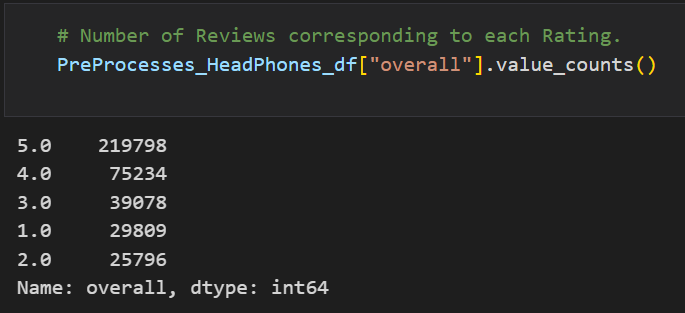
Total number of products in metadata related to heaphones are the total number of Unique products.



Among all reviews related to headphones whose column named overall having value greater than or equal to 3 are fetched and there total number is the answer for number of good rating.



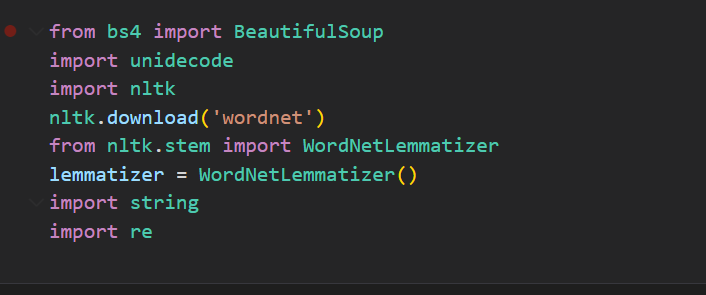
For finding number of bad rating we can subtract Total number of good ratings by total number of ratings and can get total number of bad ratings.



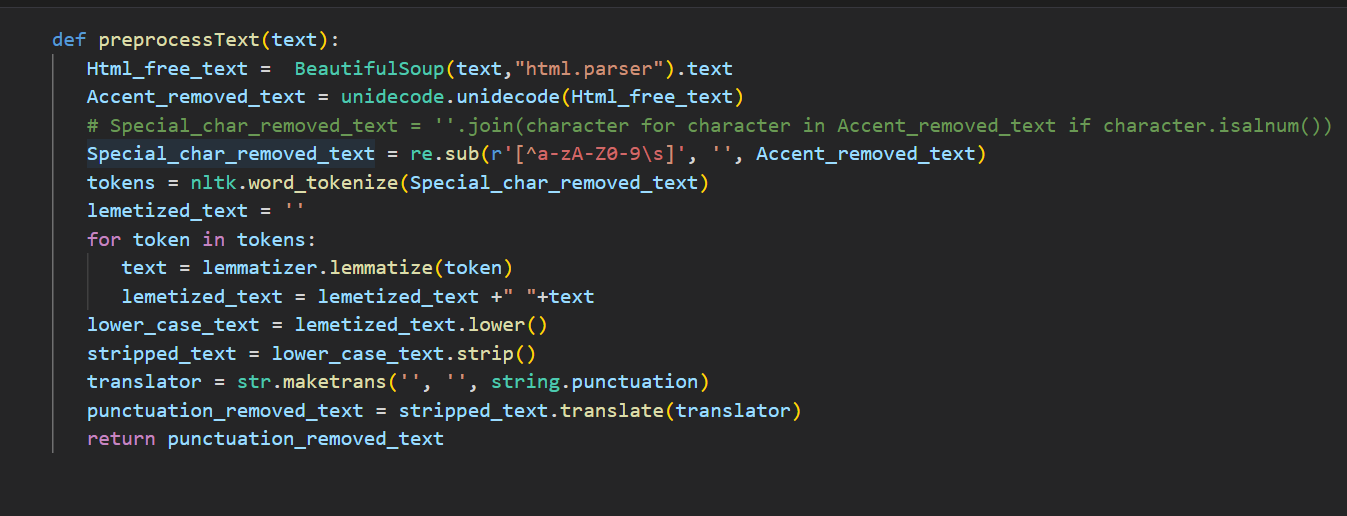
Here we have showed how many reviews have given 5 ,4 ,3 ,2, 1 ratings.

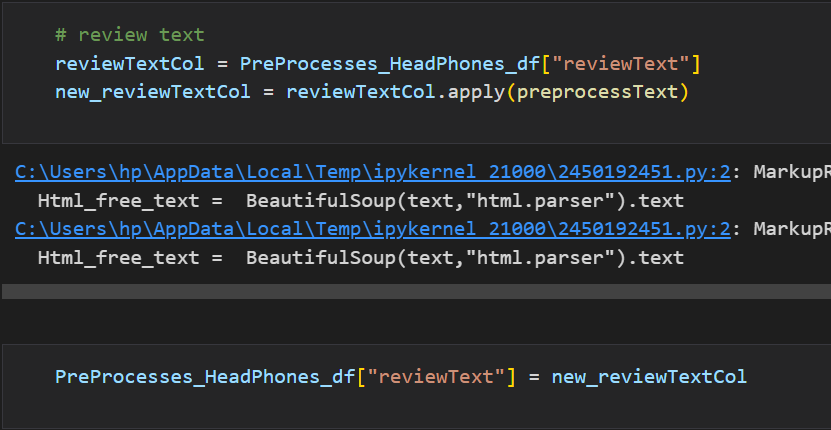
5. Preprocess the Text

Libraries used:-



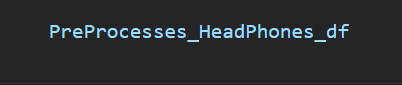
All parts are combined into one function called preprocessText that takes text as input and return preprocessed text.

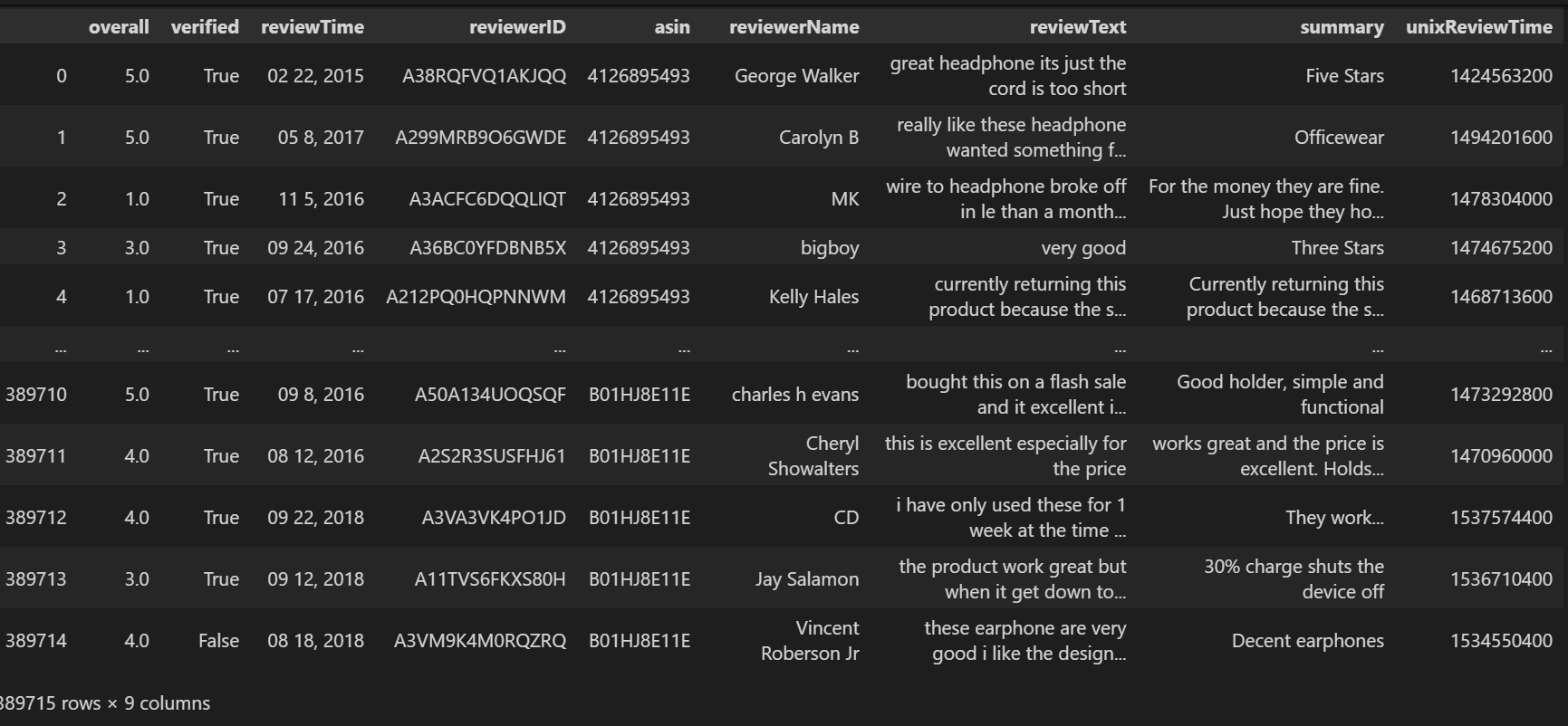
The function prepocessText preprocess the text given to it and return the preprocessed text. In preprocessing specifically it removes Html tags and Accent characters, It also does lemmatization, It removes punctuation. For removing Html tags Beautiful soap’s Html parser is used. For removing accent characters unidecode is used. For doing lemmatization text is first converted Into tokens using NLTK library and then By using NLTK library’s lemmatise all the tokens are being lemmatized. Then after that all the tokens are combined and becomes string text then that string text converted into lowercase and then that lowercased text’s punctuations are removed using maketrans and translate functions of string as you can see in screenshot.

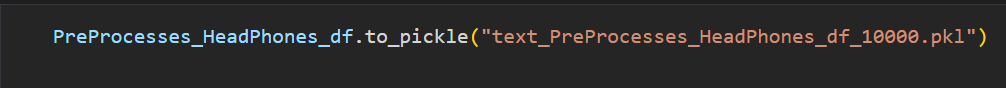


The function Preprocess text is applied to each reviewText into each review in PreProcess headphones which are for only headphones category.

You can see in output what changes are being done on review text





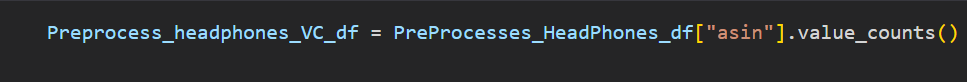


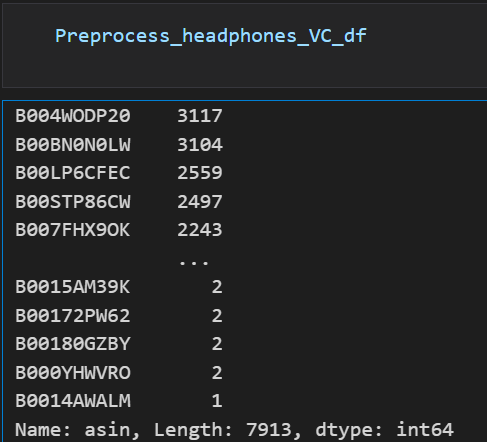


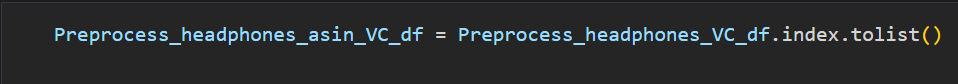
After doing text preprocessing the result is stored into pickle file so that again we don’t need to do the same for using preprocessing reviewtext in later parts.

6)

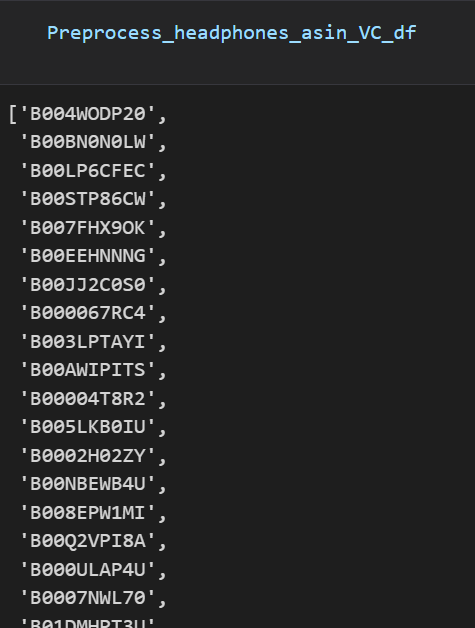
For finding Top 20 brands. I first calculated how many brands are reviewed because we can only say about a brand which is reviewed at least one time.



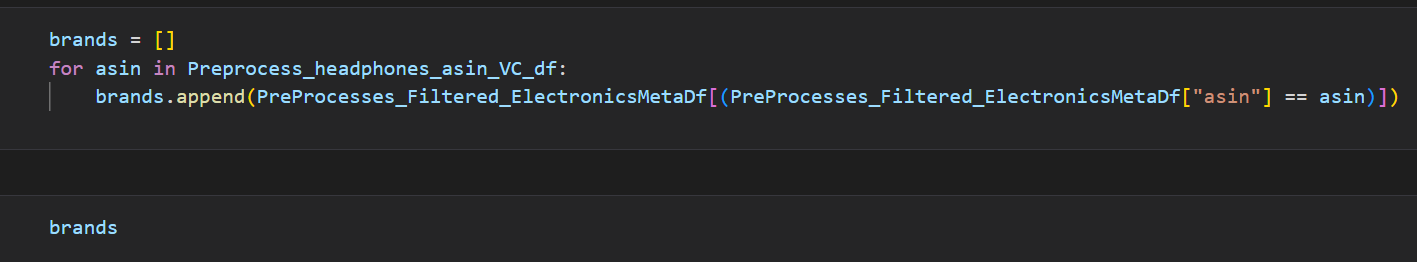


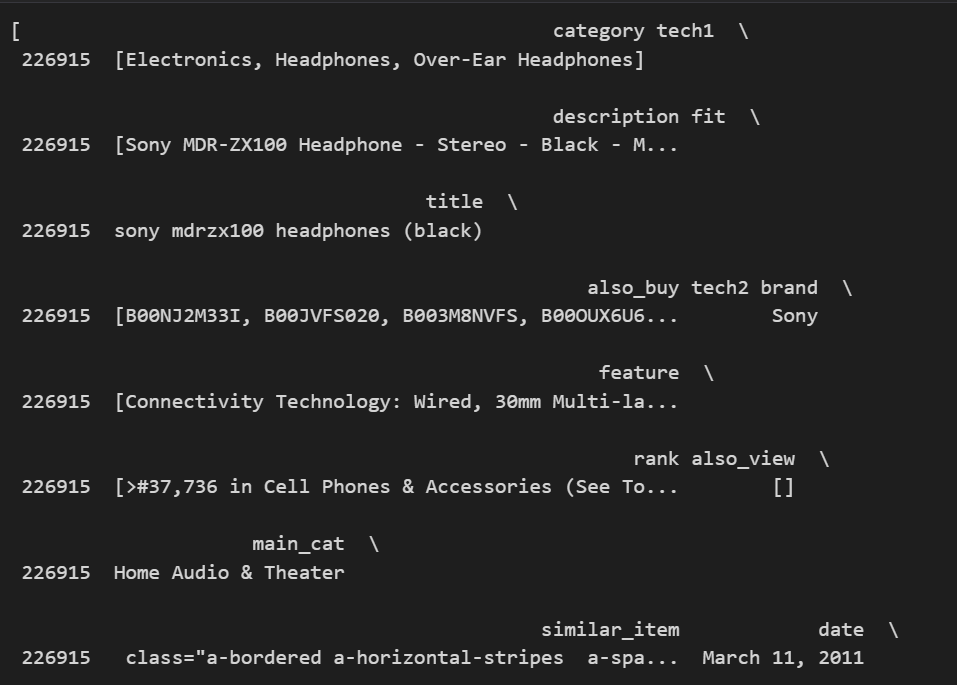


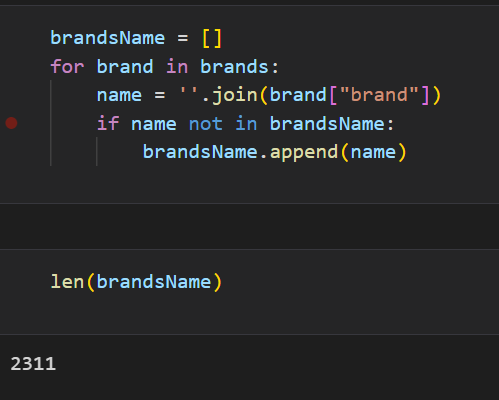
All the brands that are being reviewed converted into list and stored into Preprocess\_headphones\_asin\_VC\_df



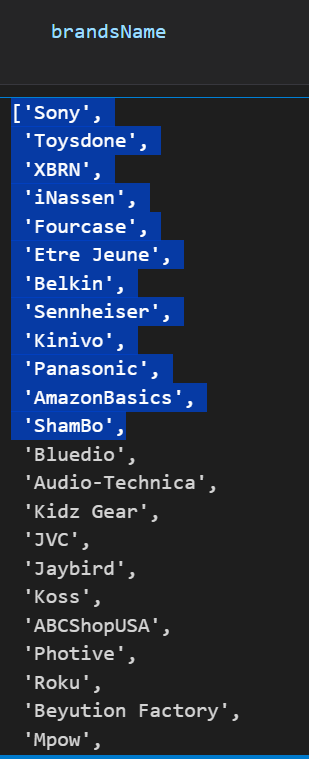
Now we have find to out of these many products how many are the products that are specifically for headphones. In below screen shit we calculated that.

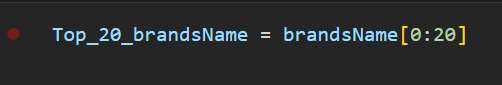




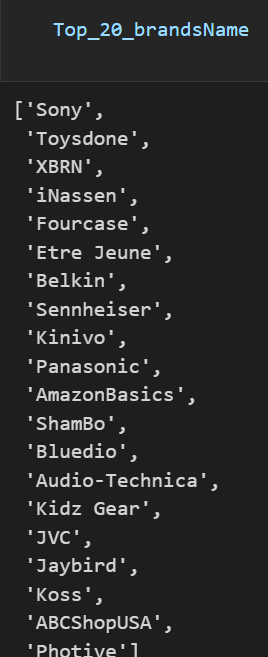


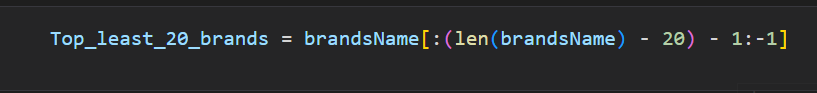
Now the corresponding brand from the product is retrieved and stored into the brandsName list

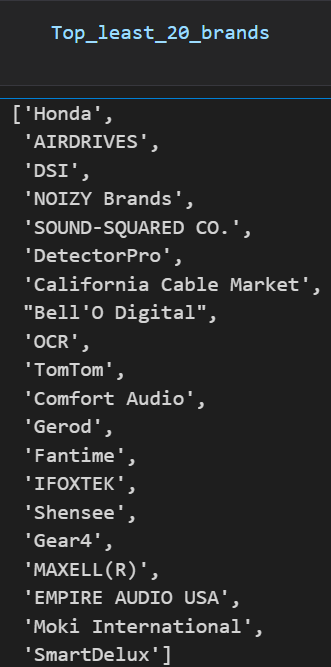




Now the first twenty are the most reviwed brand in it.

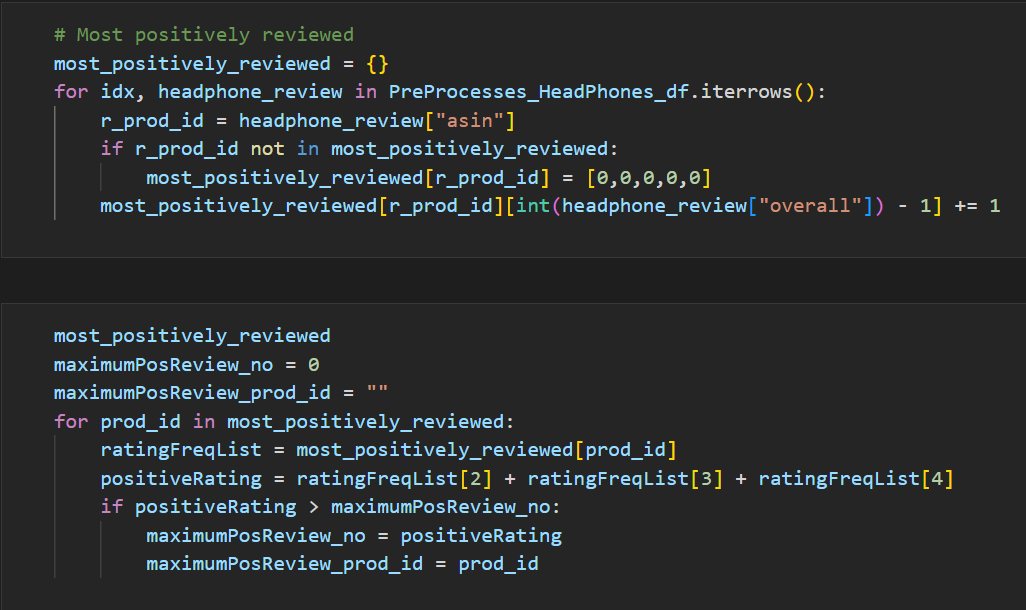


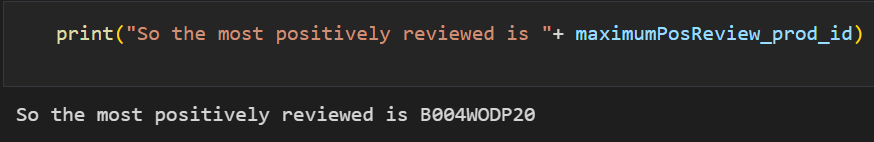




Last twenty are least 20 brands.

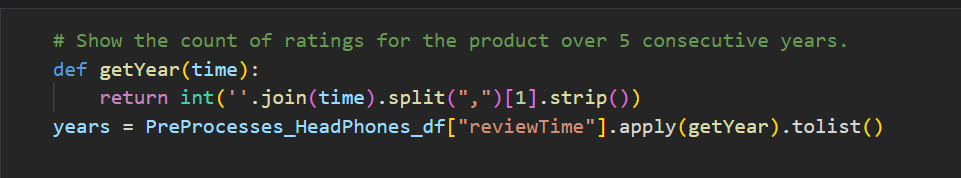
For finding most positively reviewed product , number of reviews corresponding to each rating i.e

5 4 3 2 1 is calculated then product which is having more positive reviews such as no of reviews at 3 rating + no of rating at 4 rating and no of rating at 5 star is added and the product which is having the maximum sum is the most positively reviwed.



For displaying count of ratings for the product over 5 consecutive years.

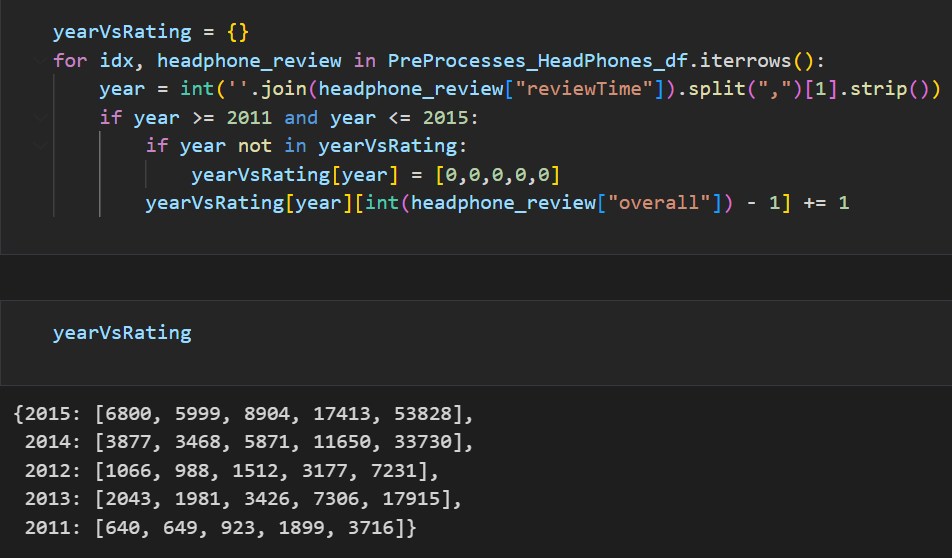
For calculating year from the reviewTime column in the dataframe that is having all the reviews a function getYear is created theat return year



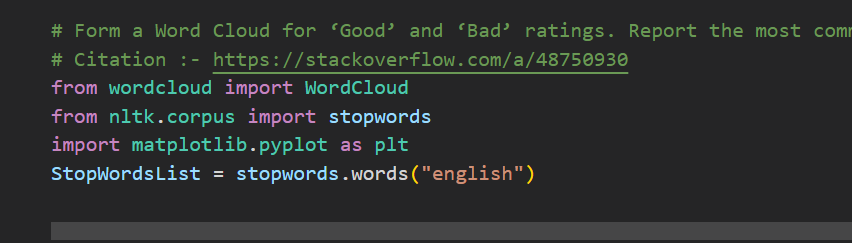
So we found year for all reviewTime values in PreProcess\_Headphones\_df and stored as list into year variable

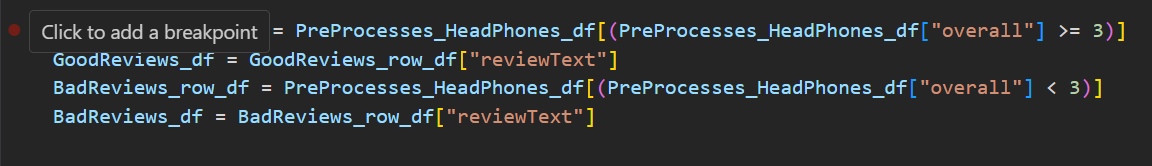


From the sorted list I got idea that we are having ratings consecutive from year 2000 to 2016 for headphones . So I choosed 5 years from 2011 to 2015 and calculated for each chosen year that who many reviews they got for this rating and printed.



For wordCloud the libraries used are wordcloud and nltk

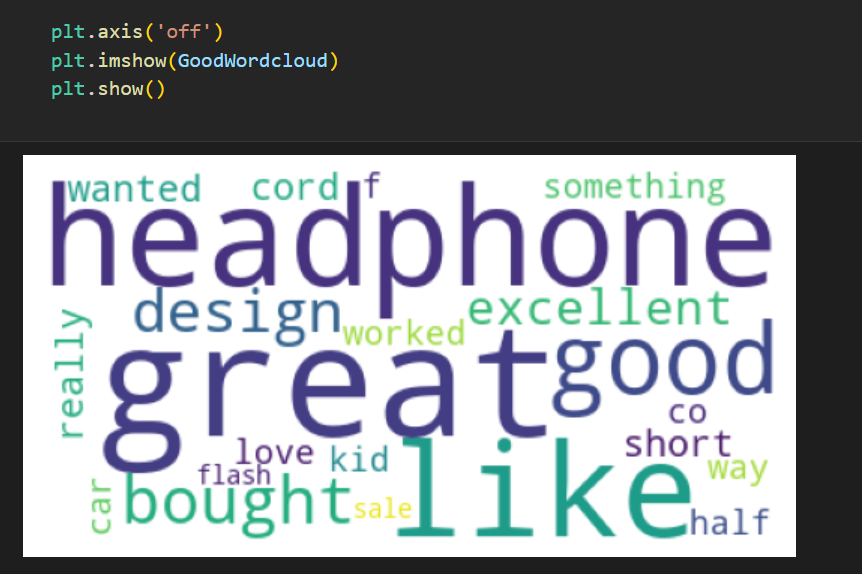


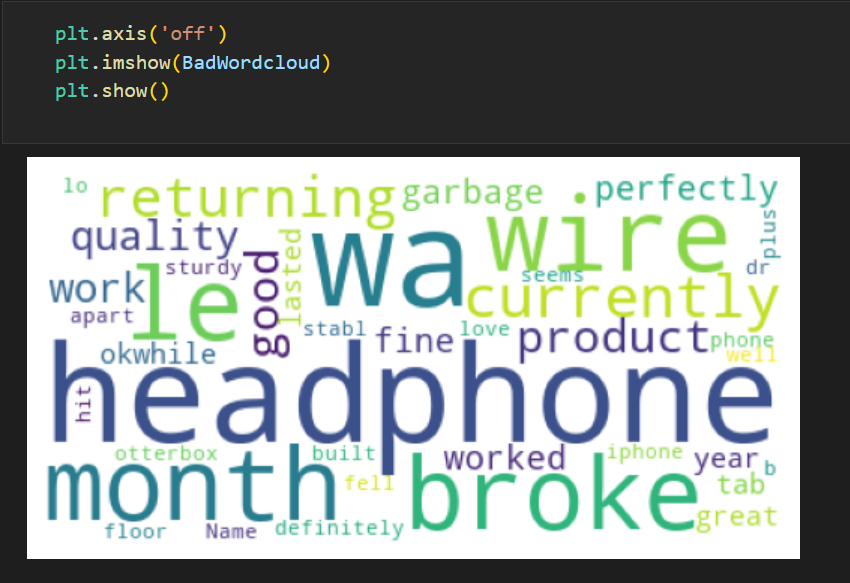


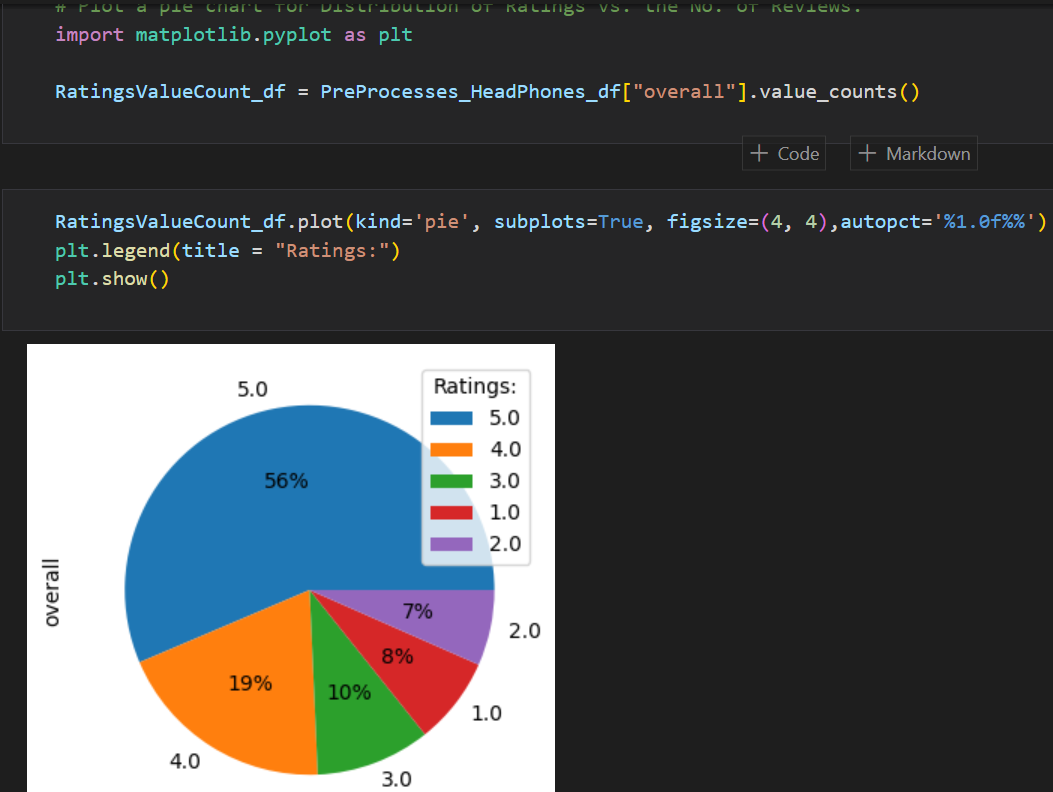
Here we have extract all the review text from the reviews that are positively reviewed and stored into the GoodReviews\_df and we also have extracted all the review text from the reviews which badly rated the product means gave rating less than 3 and we stored bad reviews into BadReviews\_df.



Now by using WordCloud object and reviewTexts list we created GoodwordCloud and BadWordcloud. The review text given into word cloud is correxponding to the type of wordcloud we want.

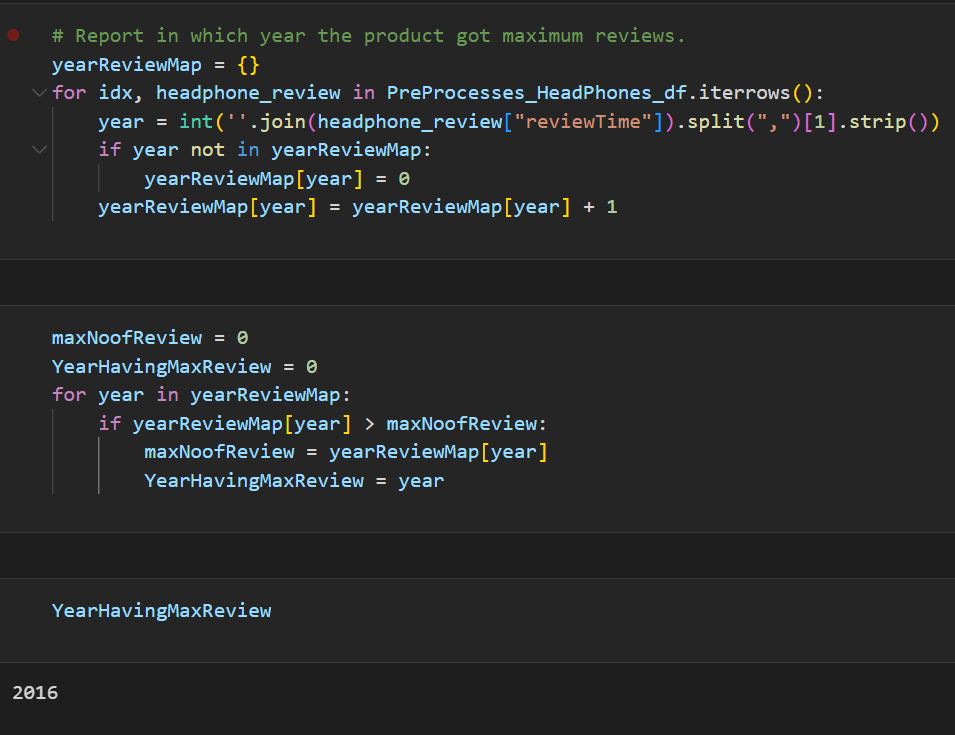




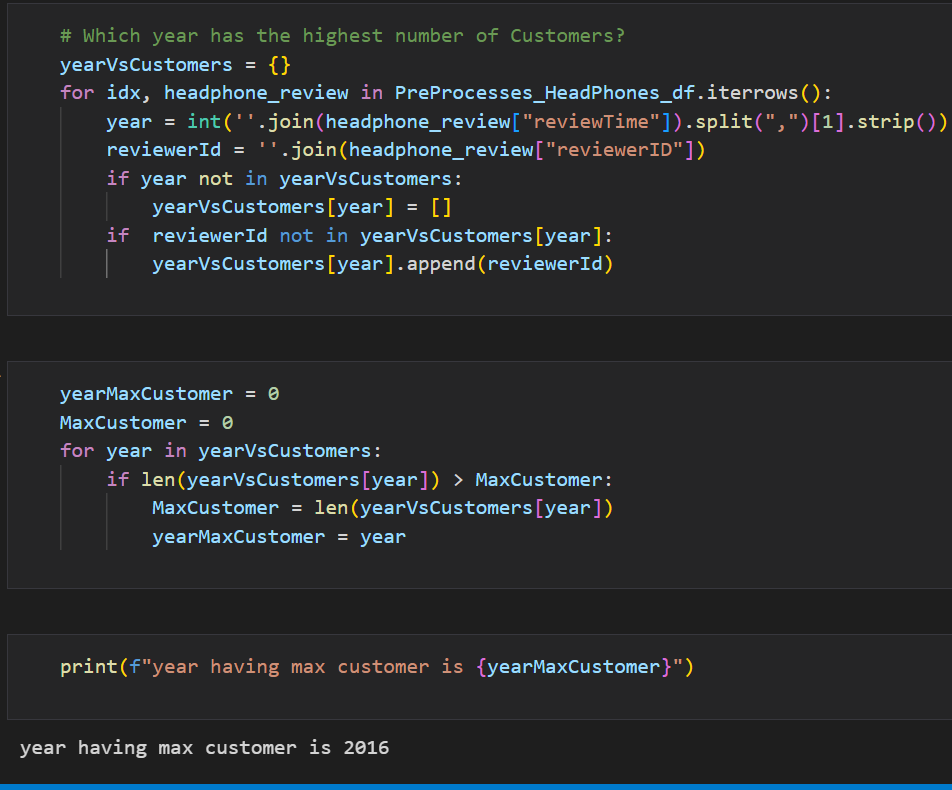


Matplotlib is used to form the pie chat between the distribution of no of ratings vs no of reviews.

Here we calculated the count of no of reviews for each rating i.e 5 , 4, 3, 2, 1 and given the values into the pie chart and formed it.

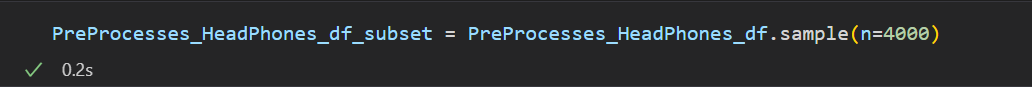


For finding which year have maximum review I created a dictionary of year vs no of reviews. I looped over all the reviews corresponding to headphones and calculated no of reviews each year has and printed year which is having maximum reviews which is 2016.

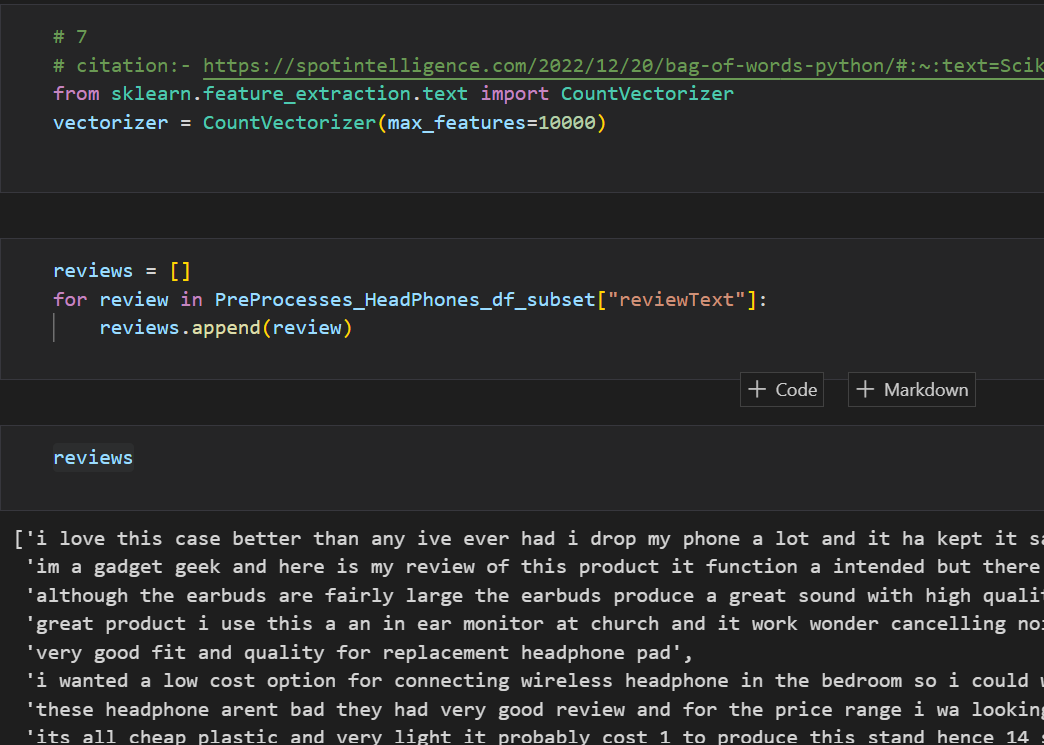


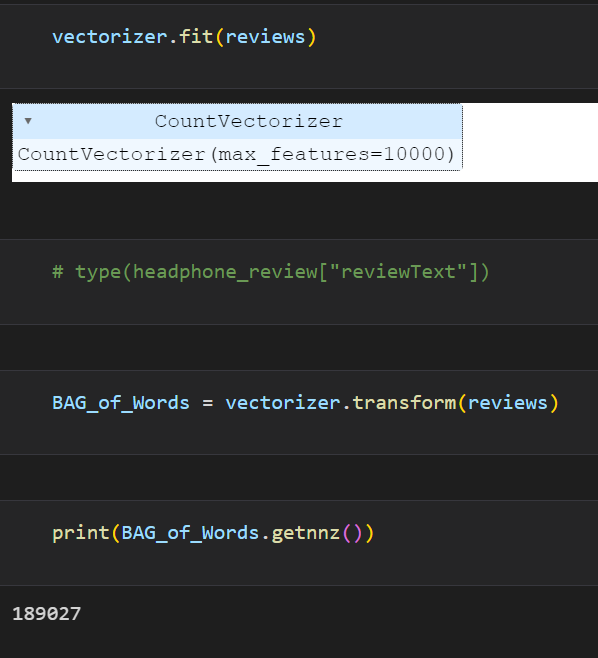
For finding customers, I found no of unique reviewers in a year and the year which is having maximum reviewers will the year which is having maximum reviews.

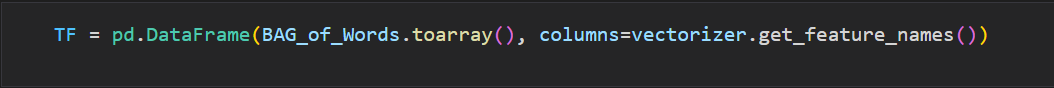
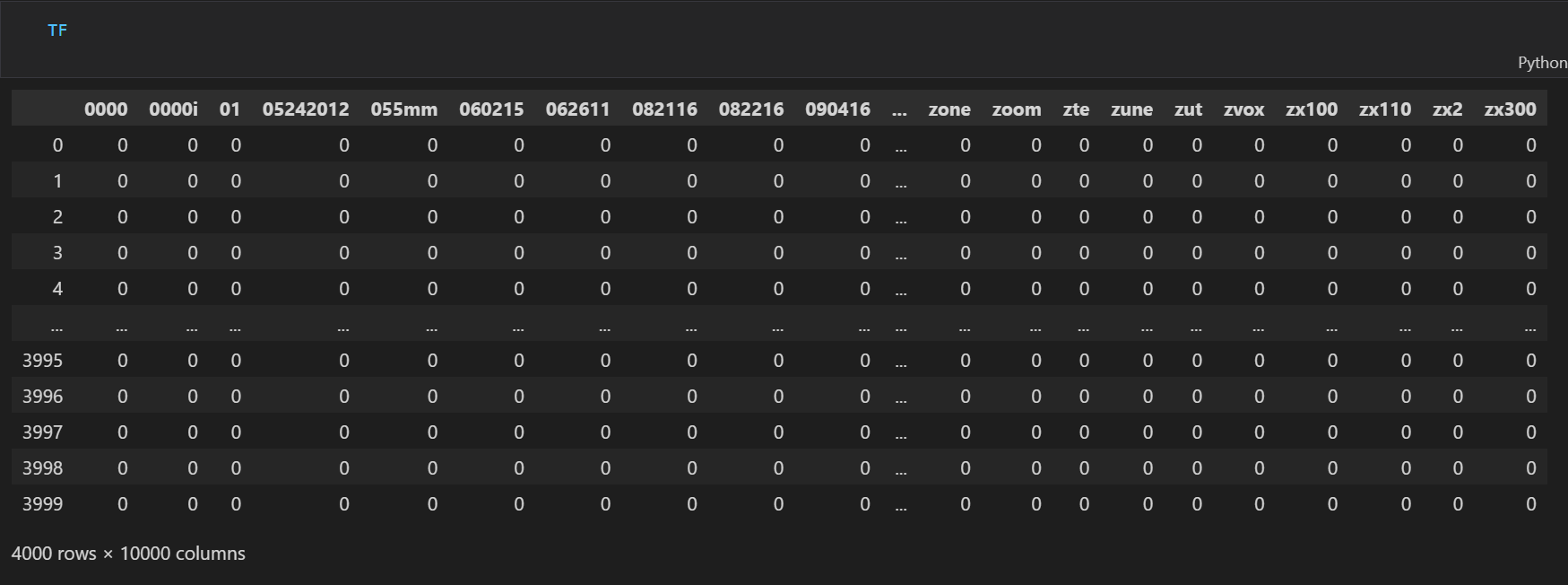
7)

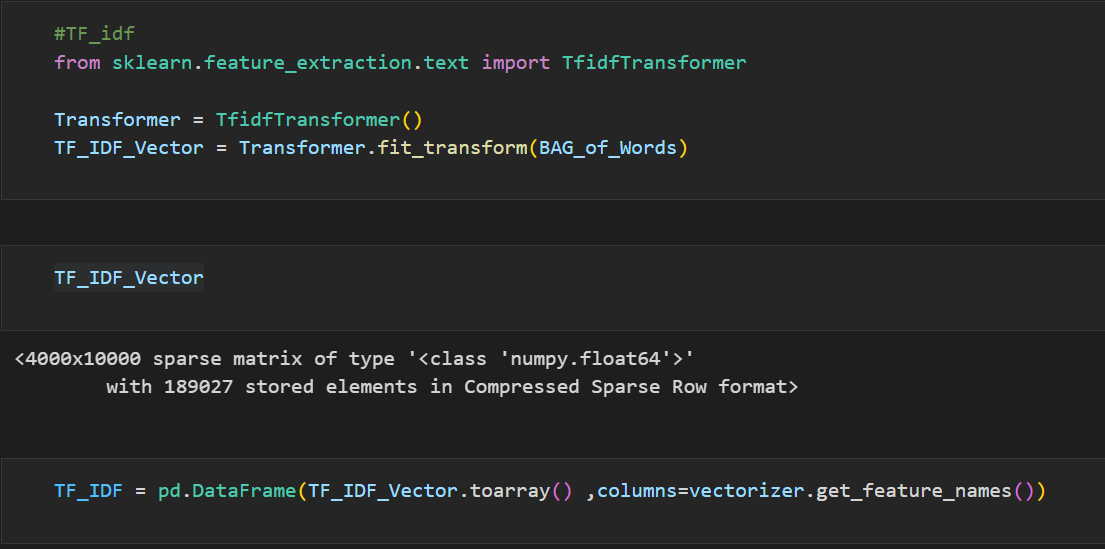


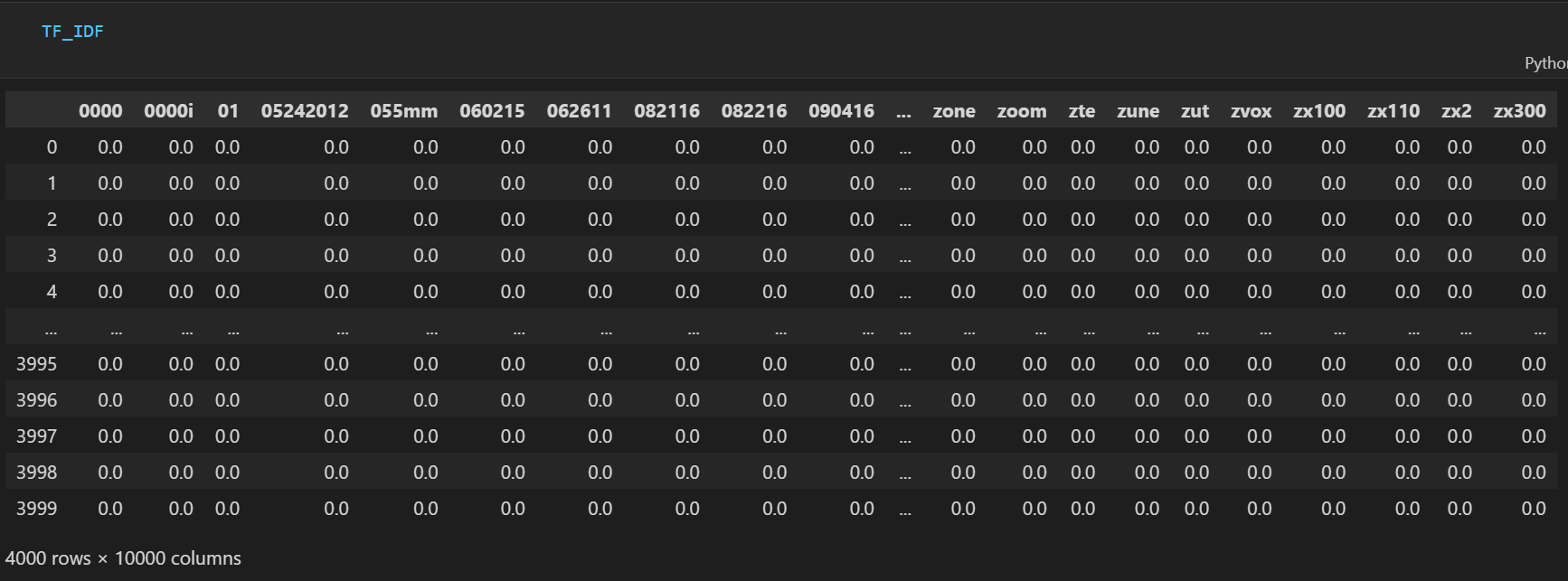
Now we have to convert review text into vector embeddings. So for vector embeddings I chose tf\_idf Because I am here only using 4000 samples of data and which is later used for training the Machine learning models and then these machine learning models predict is reviewText is good bad or average. Since I am using only 4000 review text for training the model out of 4 lakh total review text so it is required to use TF\_IDF because TF\_IDF embeddings provides the importance of each word in review text. Also in smaller dataset TF\_IDF works better because TF\_IDF consider importance of term but in we see Word2vec this gives context aware embeddings but It only works better if we are having larger data because it consider nearest word for a word to construct embeddings but in smaller dataset it is not feasible to construct embeddings this way because of smaller dataset.



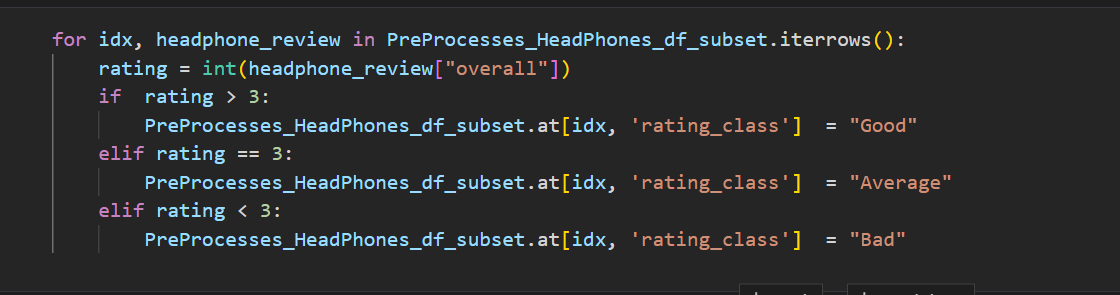


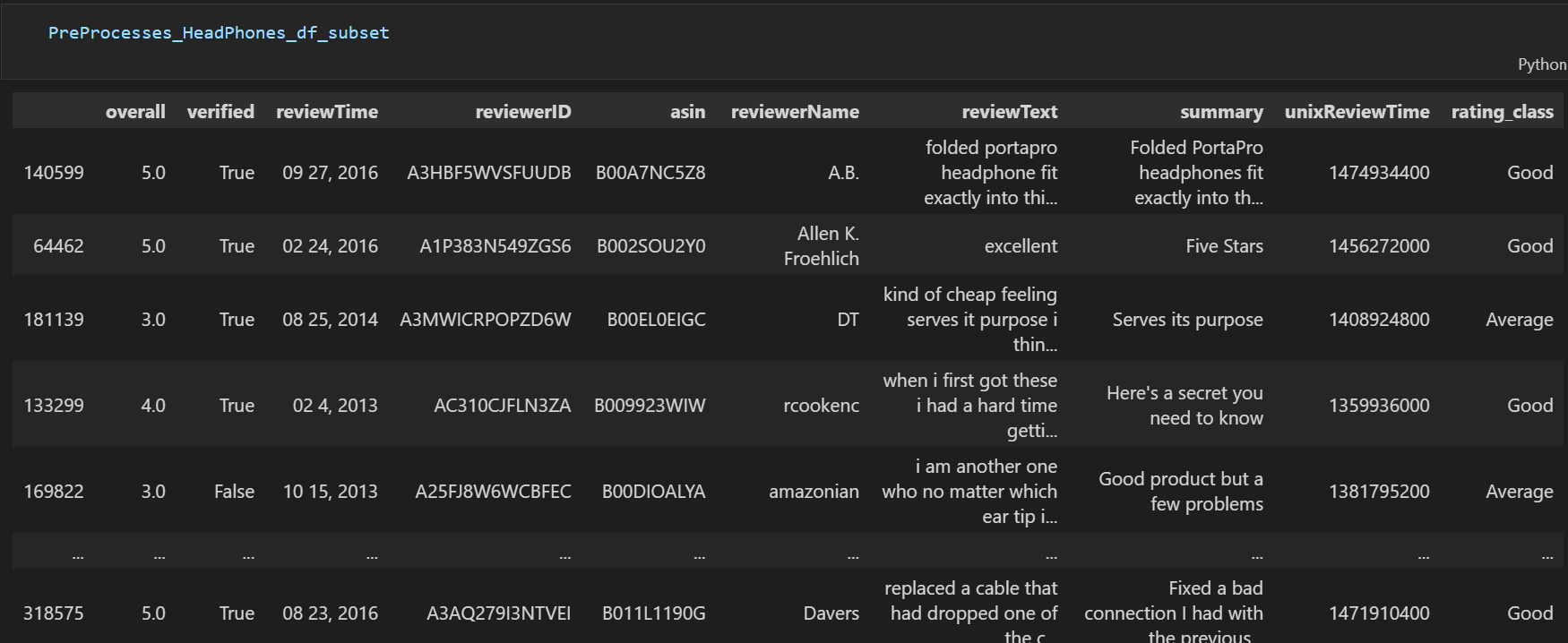






8)

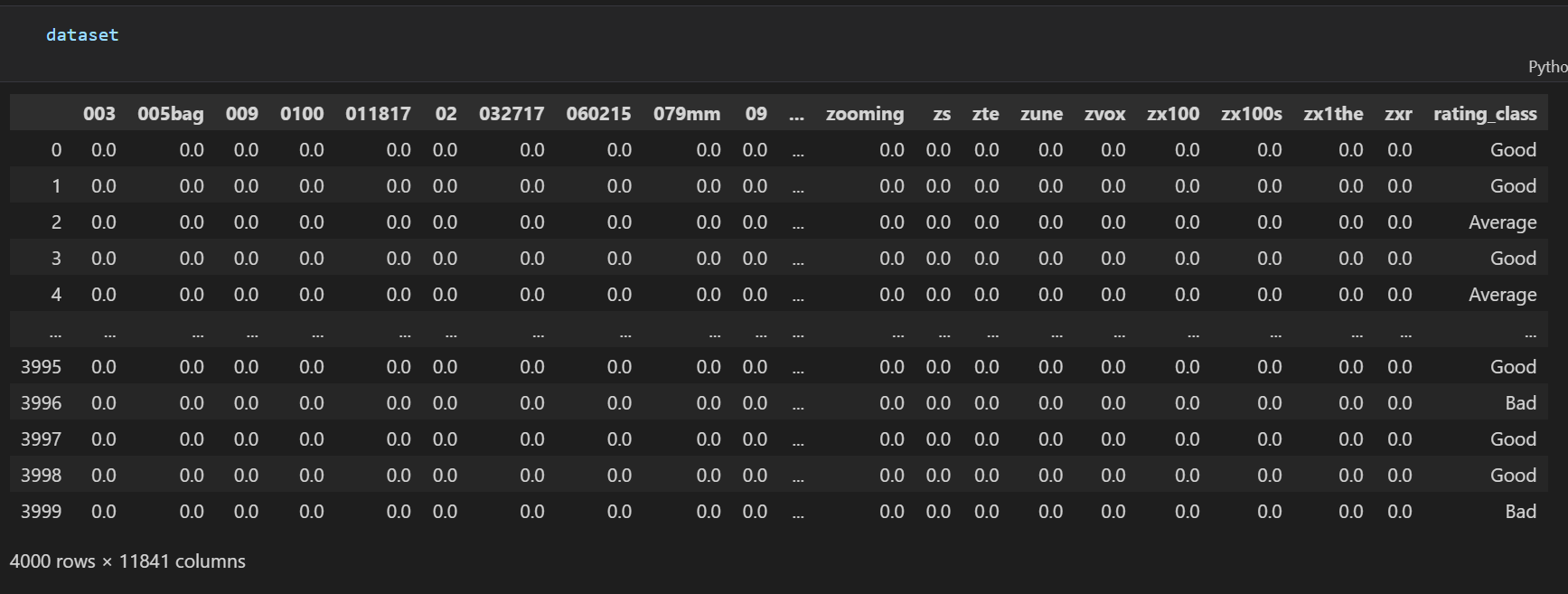




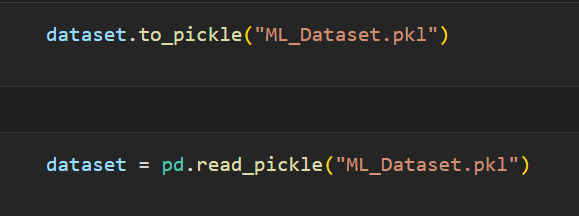
By using rule that ratings above 3 are considered to be good, equal to 3 are considered to be Average and less than 3 are considered to be bad . we classified reviews in to three classes good average and bad.

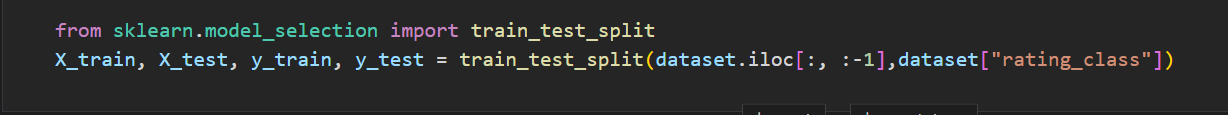
9)





Now we have taken TF\_IDF embedding as input features rather than taking review text in plain and attached rating class with them according to which embedding belongs to which reviewText and attached its rating to it.

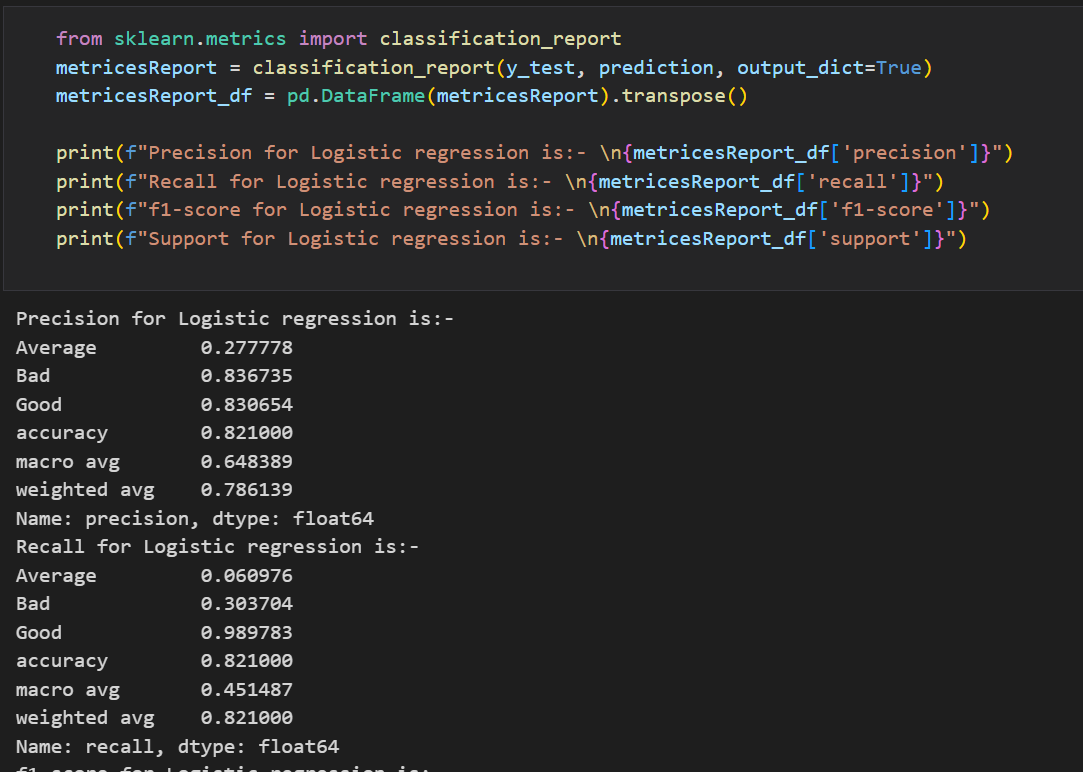


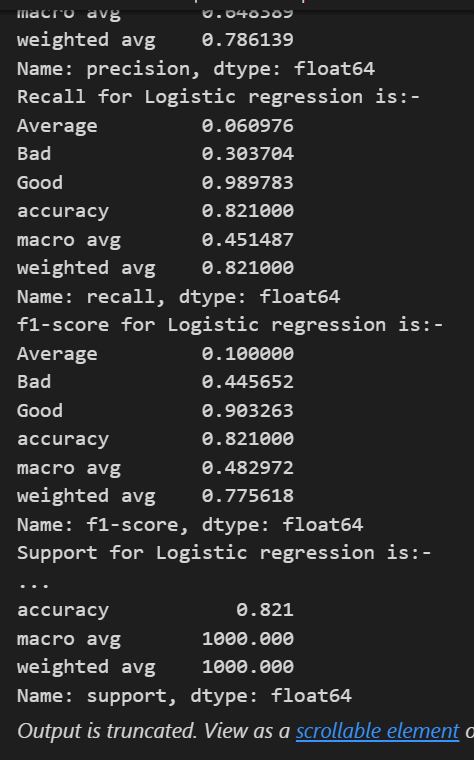


Here I have divided 75% data into training dataset and 25% into testing dataset.

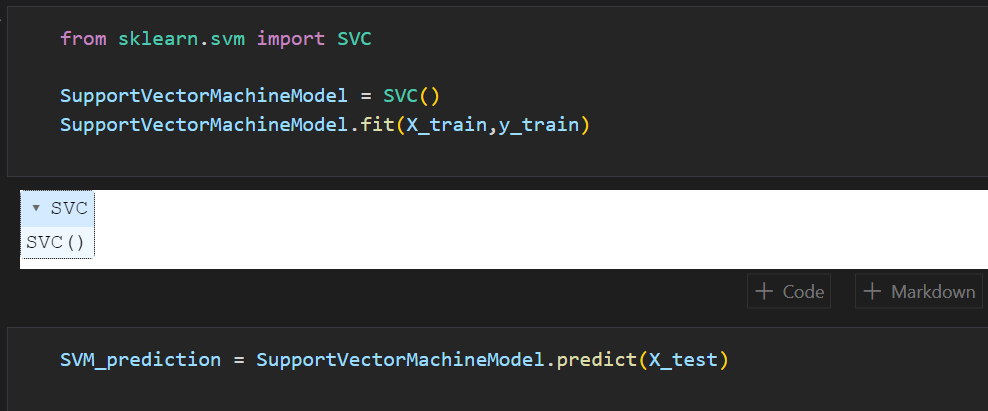
10)

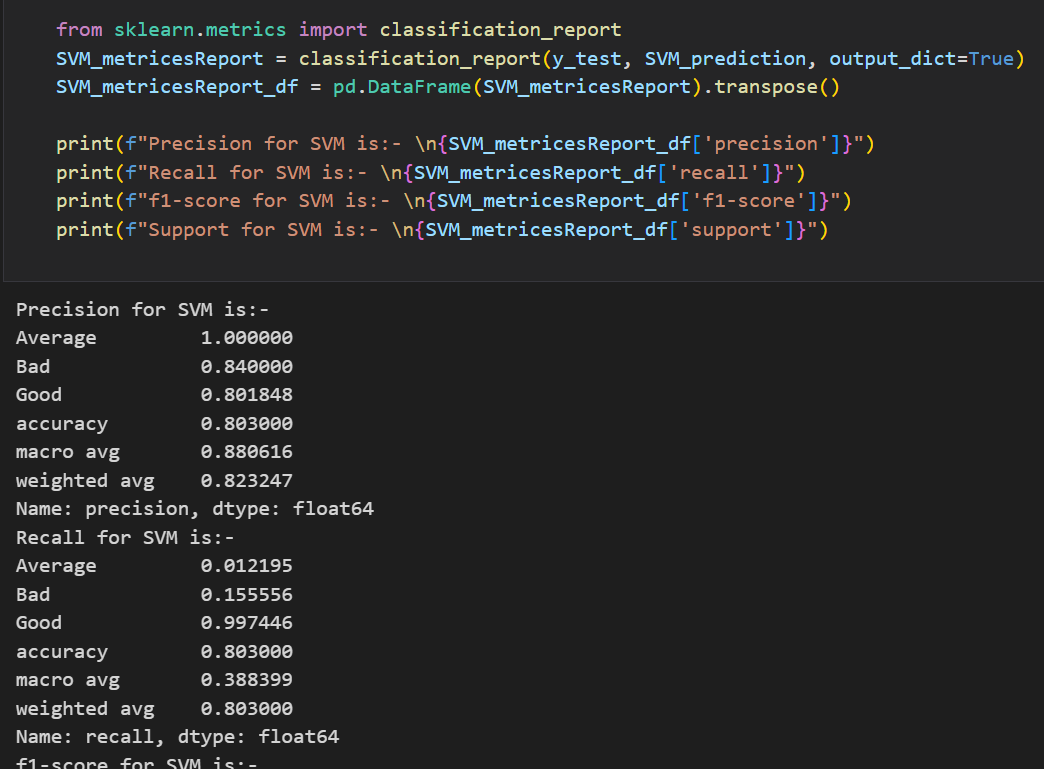
1) Used Logistic regression on it.

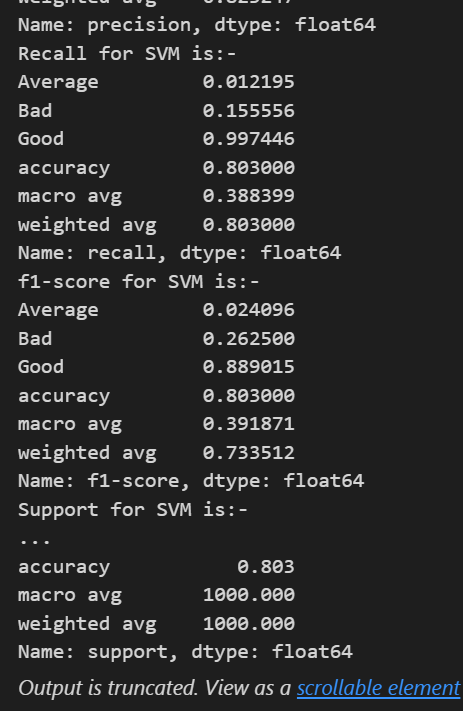




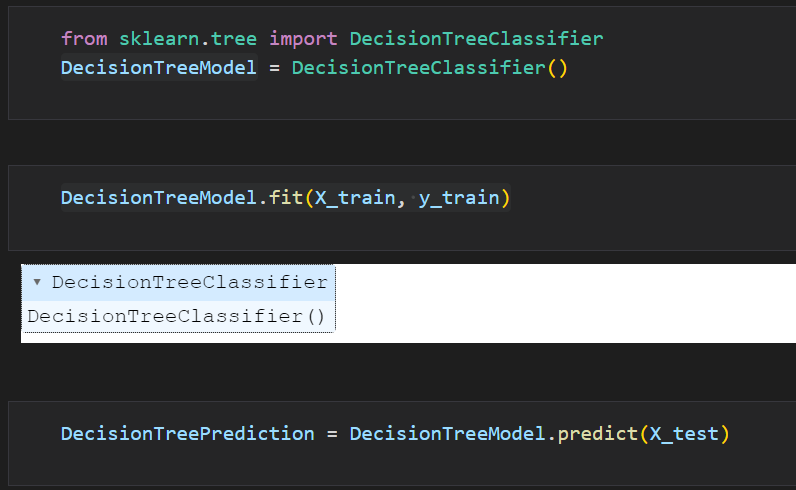
2) used Support vector machine on to the training dataset and predicted rating class for text TF\_IDF embeddings.

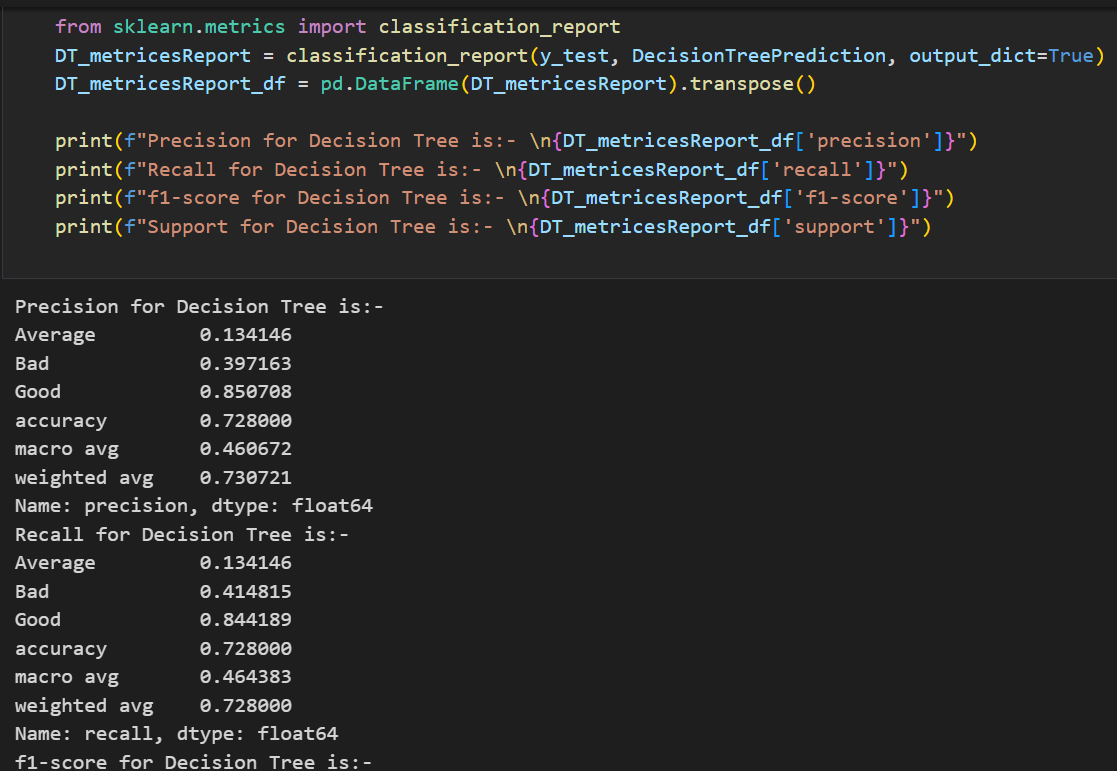


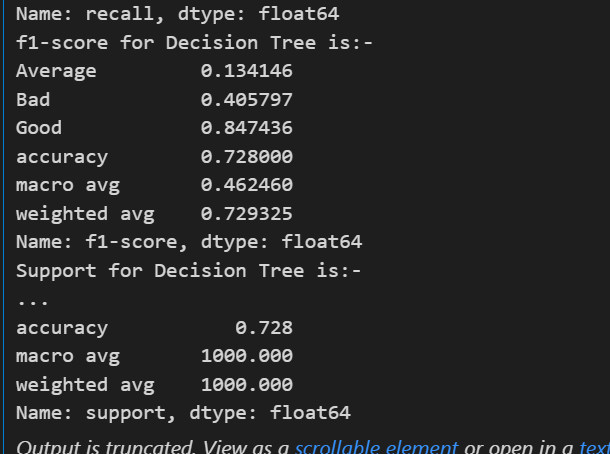




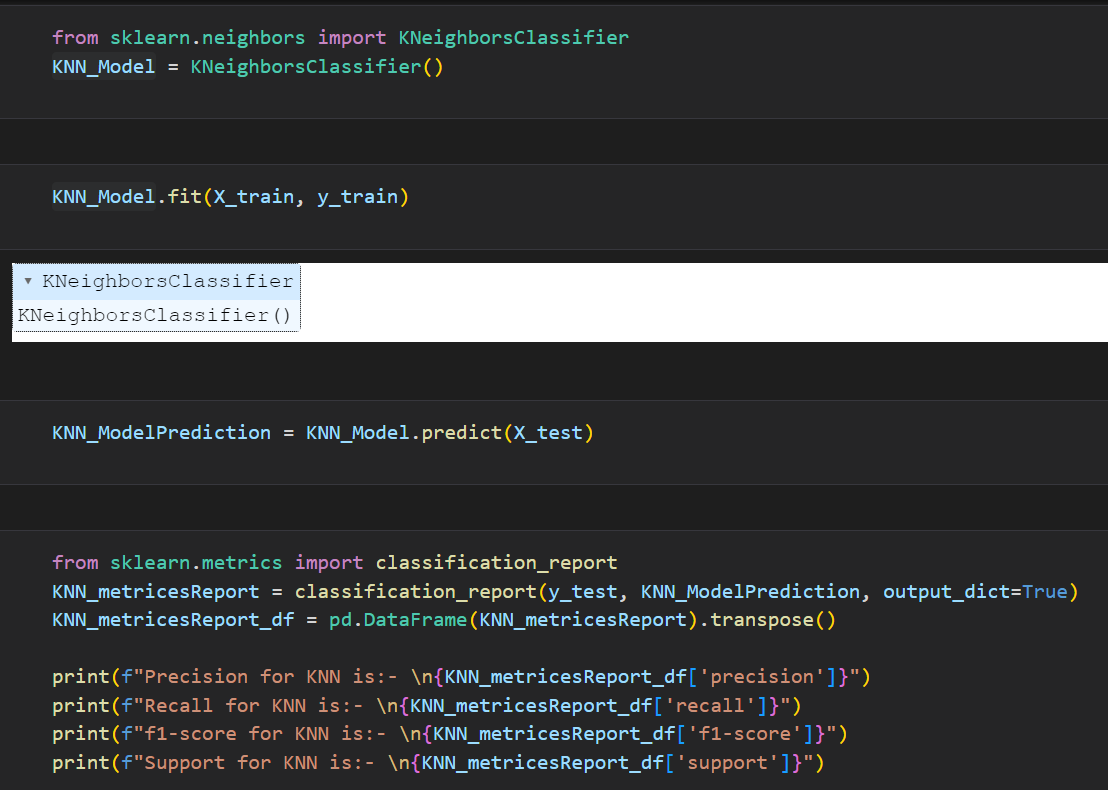
c) Used decision tree on dataset and finds testing metrices for it

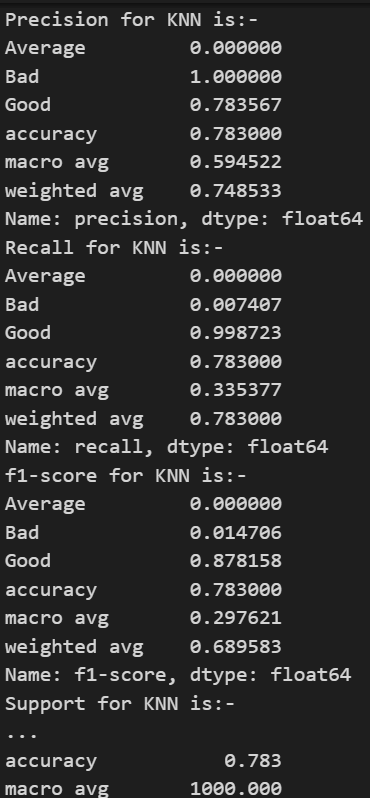




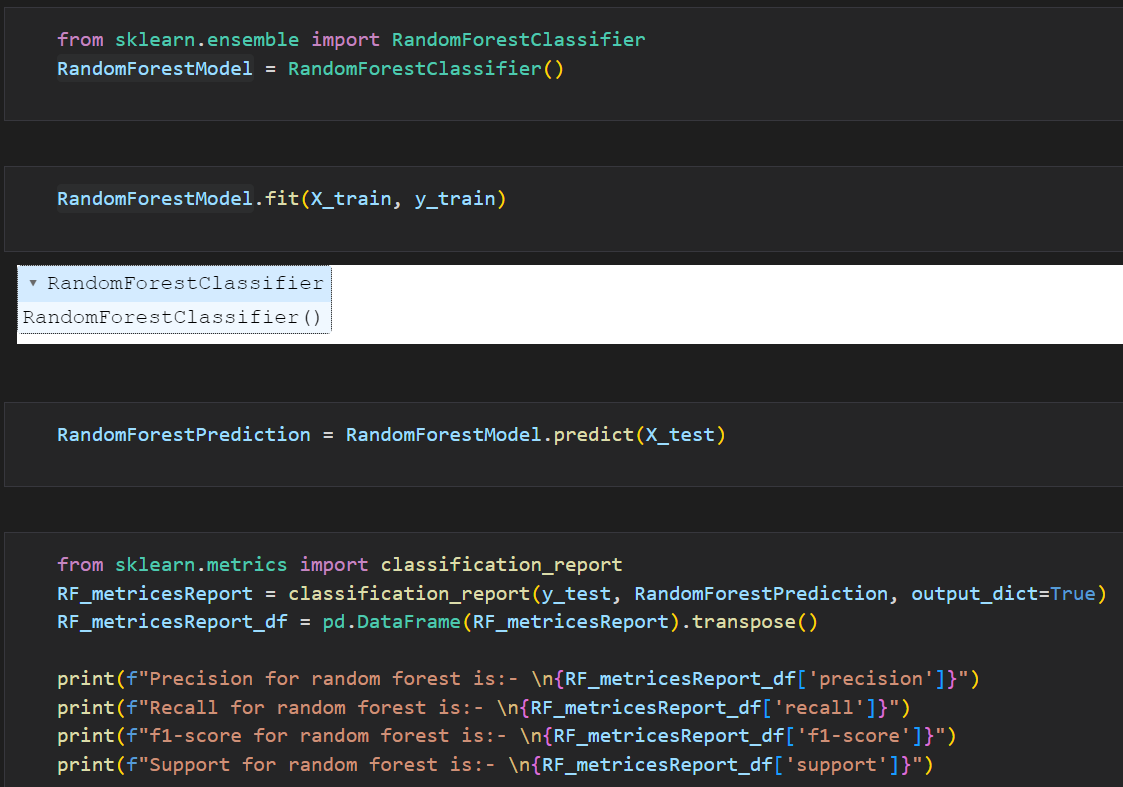


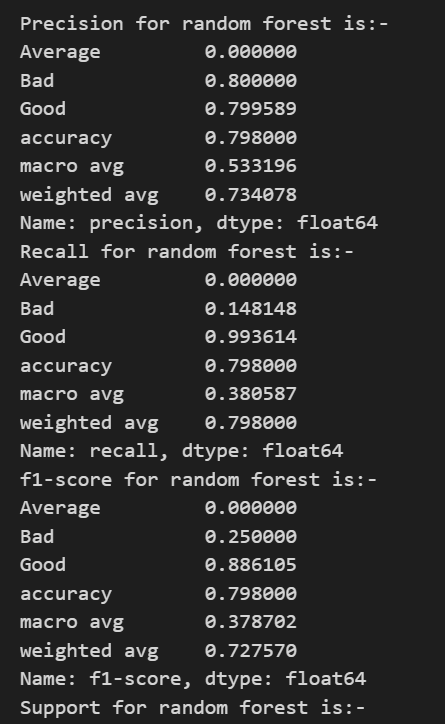
d) Also used KNN I dataset

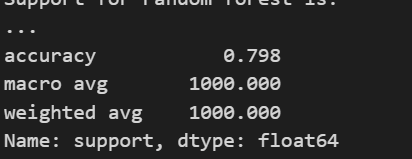




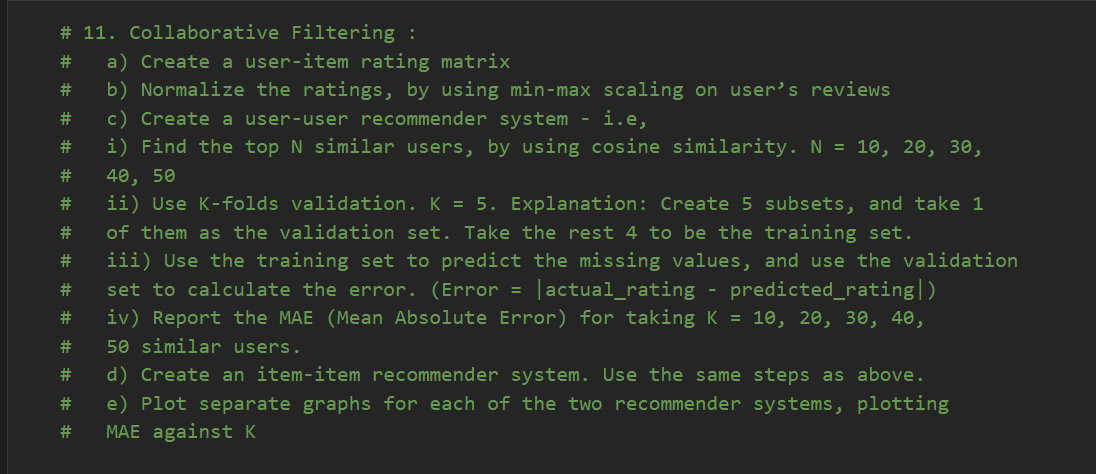
e) Used Random forest on the dataset to finds its evaluation metrices





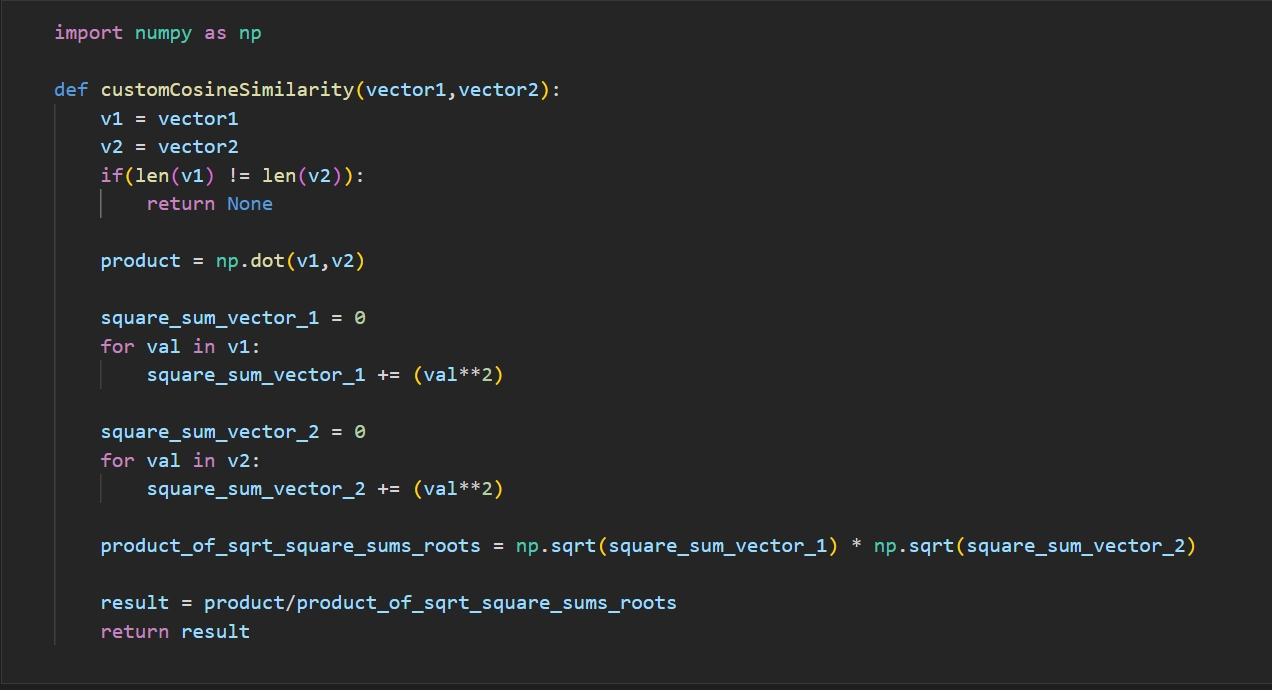


11) Here I have created User user recommender system and item item recommeder system and compare their results based by plotting graph.

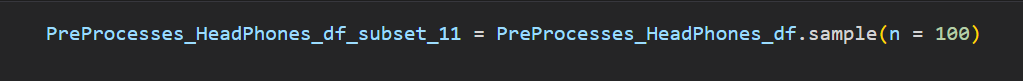
****



Created custom cosine similarity function



Taken 100 reviews for the testing purpose but can take whatever reviews any body wants



from sklearn.preprocessing import MinMaxScaler

# citattion:- https://stackoverflow.com/a/55129763

K\_10 = []

K\_20 = []

K\_30 = []

K\_40 = []

K\_50 = []

scaler = MinMaxScaler()

user\_item\_matrix\_Validation = PreProcesses\_HeadPhones\_df\_subset\_11.pivot\_table(index='reviewerID', columns='asin', values='overall', aggfunc='first')

user\_item\_matrix\_Validation\_Inverse = user\_item\_matrix\_Validation.T

user\_item\_matrix\_Validation\_Inverse\_scaled = pd.DataFrame(scaler.fit\_transform(user\_item\_matrix\_Validation\_Inverse.values), columns=user\_item\_matrix\_Validation\_Inverse.columns, index=user\_item\_matrix\_Validation\_Inverse.index)

user\_item\_matrix\_Validation\_normalized = user\_item\_matrix\_Validation\_Inverse\_scaled.T

user\_item\_matrix\_Validation\_normalized.fillna(-1, inplace=True)

i = 1

for train\_idx, test\_idx in k\_fold.split(PreProcesses\_HeadPhones\_df\_subset\_11):

    print(i)

    i += 1

    X\_F\_train, X\_F\_test = PreProcesses\_HeadPhones\_df\_subset\_11.iloc[train\_idx,:],PreProcesses\_HeadPhones\_df\_subset\_11.iloc[test\_idx,:]

    print("X\_F\_train", X\_F\_train)

    print("X\_F\_test", X\_F\_test)

    user\_item\_matrix = X\_F\_train.pivot\_table(index='reviewerID', columns='asin', values='overall', aggfunc='first')

    scaler = MinMaxScaler()

    user\_item\_matrix\_Inverse = user\_item\_matrix.T

    user\_item\_matrix\_Inverse\_scaled = pd.DataFrame(scaler.fit\_transform(user\_item\_matrix\_Inverse.values), columns=user\_item\_matrix\_Inverse.columns, index=user\_item\_matrix\_Inverse.index)

    user\_item\_matrix\_normalized = user\_item\_matrix\_Inverse\_scaled.T

    user\_item\_matrix\_normalized.fillna(-1, inplace=True)

    UservsUser\_similairity\_matrix = pd.DataFrame(index=user\_item\_matrix\_normalized.index, columns=user\_item\_matrix\_normalized.index)

    for idx1, row1 in user\_item\_matrix\_normalized.iterrows():

        for idx2, row2 in user\_item\_matrix\_normalized.iterrows():

            cosine\_similarity\_withall\_values = cosine\_similarity(row1.values.reshape(1, -1), row2.values.reshape(1, -1))

            print(cosine\_similarity\_withall\_values)

            UservsUser\_similairity\_matrix.at[idx1, idx2] = cosine\_similarity\_withall\_values[0][0]

    UserVsUser\_dict\_10 = {}

    for idx, row in UservsUser\_similairity\_matrix.iterrows():

        top\_11\_users = row.sort\_values(ascending=False).head(11).index.tolist()

        print(top\_11\_users)

        UserVsUser\_dict\_10[idx] = top\_11\_users

        print(idx, UserVsUser\_dict\_10[idx])

        # print(idx,UservsUser\_similairity\_matrix.loc[row] )

        UserVsUser\_dict\_10[idx].remove(idx)

    UserVsUser\_dict\_20 = {}

    for idx, row in UservsUser\_similairity\_matrix.iterrows():

        top\_21\_users = row.sort\_values(ascending=False).head(21).index.tolist()

        print(top\_21\_users)

        UserVsUser\_dict\_20[idx] = top\_21\_users

        print(idx, UserVsUser\_dict\_20[idx])

        # print(idx,UservsUser\_similairity\_matrix.loc[row] )

        UserVsUser\_dict\_20[idx].remove(idx)

    # print("UserVsUser\_dict\_20")

    # print(UserVsUser\_dict\_20)

    UserVsUser\_dict\_30 = {}

    for idx, row in UservsUser\_similairity\_matrix.iterrows():

        top\_31\_users = row.sort\_values(ascending=False).head(31).index.tolist()

        print(top\_31\_users)

        UserVsUser\_dict\_30[idx] = top\_31\_users

        print(idx, UserVsUser\_dict\_30[idx])

        # print(idx,UservsUser\_similairity\_matrix.loc[row] )

        UserVsUser\_dict\_30[idx].remove(idx)

    # print("UserVsUser\_dict\_30")

    # print(UserVsUser\_dict\_30)

    UserVsUser\_dict\_40 = {}

    for idx, row in UservsUser\_similairity\_matrix.iterrows():

        top\_41\_users = row.sort\_values(ascending=False).head(41).index.tolist()

        print(top\_41\_users)

        UserVsUser\_dict\_40[idx] = top\_41\_users

        print(idx, UserVsUser\_dict\_40[idx])

        # print(idx,UservsUser\_similairity\_matrix.loc[row] )

        UserVsUser\_dict\_40[idx].remove(idx)

    # print("UserVsUser\_dict\_40")

    # print(UserVsUser\_dict\_40)

    UserVsUser\_dict\_50 = {}

    for idx, row in UservsUser\_similairity\_matrix.iterrows():

        top\_51\_users = row.sort\_values(ascending=False).head(51).index.tolist()

        print(top\_51\_users)

        UserVsUser\_dict\_50[idx] = top\_51\_users

        print(idx, UserVsUser\_dict\_50[idx])

        # print(idx,UservsUser\_similairity\_matrix.loc[row] )

        UserVsUser\_dict\_50[idx].remove(idx)

    # print("UserVsUser\_dict\_50")

    # print(UserVsUser\_dict\_50)

    print("UservsUser\_similairity\_matrix",UservsUser\_similairity\_matrix)

    user\_item\_matrix\_normalized\_10 = user\_item\_matrix\_normalized.copy()

    # mae = mean\_absolute\_error(user\_item\_matrix\_Validation\_normalized.iloc[0].values, user\_item\_matrix\_normalized\_10.iloc[0].values)

    print(user\_item\_matrix\_Validation\_normalized.iloc[0].values, user\_item\_matrix\_normalized\_10.iloc[0].values)

    print("user\_item\_matrix\_normalized", user\_item\_matrix\_normalized)

    print("user\_item\_matrix\_normalized\_10", user\_item\_matrix\_normalized\_10)

    print("user\_item\_matrix\_Validation", user\_item\_matrix\_Validation\_normalized)

    for userId in UserVsUser\_dict\_10:

        similar\_Users = UserVsUser\_dict\_10[userId]

        predicted\_values = user\_item\_matrix\_normalized\_10.loc[similar\_Users].mean(axis=0)

        row = user\_item\_matrix\_normalized\_10.loc[userId]

        for column\_name, column\_data in row.iteritems():

            user\_item\_matrix\_normalized\_10[column\_name] = predicted\_values[column\_name]

    mae\_list\_10 = []

    for idx, row in user\_item\_matrix\_normalized\_10.iterrows():

        print("I am outside")

        print(user\_item\_matrix\_Validation.index)

        print(idx)

        if idx in user\_item\_matrix\_Validation\_normalized.index:

            print("I am inside")

            mae = 0

            for column\_name, column\_data in row.iteritems():

                if column\_name in user\_item\_matrix\_Validation.columns:

                    mae += abs(user\_item\_matrix\_Validation\_normalized.loc[idx,column\_name] - user\_item\_matrix\_normalized\_10.loc[idx,column\_name])

            mae = mae /  user\_item\_matrix\_normalized\_10.shape[1]

            mae\_list\_10.append(mae)

    print(mae\_list\_10)

    if len(mae\_list\_10) > 0:

        mae\_10 = sum(mae\_list\_10) / len(mae\_list\_10)

        K\_10.append(mae\_10)

    user\_item\_matrix\_normalized\_20 = user\_item\_matrix\_normalized.copy()

    for userId in UserVsUser\_dict\_20:

        similar\_Users = UserVsUser\_dict\_20[userId]

        predicted\_values = user\_item\_matrix\_normalized\_20.loc[similar\_Users].mean(axis=0)

        row = user\_item\_matrix\_normalized\_20.loc[userId]

        for column\_name, column\_data in row.iteritems():

            user\_item\_matrix\_normalized\_20[column\_name] = predicted\_values[column\_name]

    mae\_list\_20 = []

    for idx, row in user\_item\_matrix\_normalized\_20.iterrows():

        print("I am outside")

        print(user\_item\_matrix\_Validation.index)

        print(idx)

        if idx in user\_item\_matrix\_Validation\_normalized.index:

            print("I am inside")

            mae = 0

            for column\_name, column\_data in row.iteritems():

                if column\_name in user\_item\_matrix\_Validation.columns:

                    mae += abs(user\_item\_matrix\_Validation\_normalized.loc[idx,column\_name] - user\_item\_matrix\_normalized\_20.loc[idx,column\_name])

            mae = mae /  user\_item\_matrix\_normalized\_20.shape[1]

            mae\_list\_20.append(mae)

    print(mae\_list\_20)

    if len(mae\_list\_20) > 0:

        mae\_20 = sum(mae\_list\_20) / len(mae\_list\_20)

        K\_20.append(mae\_20)

    user\_item\_matrix\_normalized\_30 = user\_item\_matrix\_normalized.copy()

    for userId in UserVsUser\_dict\_30:

        similar\_Users = UserVsUser\_dict\_30[userId]

        predicted\_values = user\_item\_matrix\_normalized\_30.loc[similar\_Users].mean(axis=0)

        row = user\_item\_matrix\_normalized\_30.loc[userId]

        for column\_name, column\_data in row.iteritems():

            user\_item\_matrix\_normalized\_30[column\_name] = predicted\_values[column\_name]

    mae\_list\_30 = []

    for idx, row in user\_item\_matrix\_normalized\_30.iterrows():

        print("I am outside")

        print(user\_item\_matrix\_Validation.index)

        print(idx)

        if idx in user\_item\_matrix\_Validation\_normalized.index:

            print("I am inside")

            mae = 0

            for column\_name, column\_data in row.iteritems():

                if column\_name in user\_item\_matrix\_Validation.columns:

                    mae += abs(user\_item\_matrix\_Validation\_normalized.loc[idx,column\_name] - user\_item\_matrix\_normalized\_30.loc[idx,column\_name])

            mae = mae /  user\_item\_matrix\_normalized\_30.shape[1]

            mae\_list\_30.append(mae)

    print(mae\_list\_30)

    if len(mae\_list\_30) > 0:

        mae\_30 = sum(mae\_list\_30) / len(mae\_list\_30)

        K\_30.append(mae\_30)

    user\_item\_matrix\_normalized\_40 = user\_item\_matrix\_normalized.copy()

    for userId in UserVsUser\_dict\_40:

        similar\_Users = UserVsUser\_dict\_40[userId]

        predicted\_values = user\_item\_matrix\_normalized\_40.loc[similar\_Users].mean(axis=0)

        row = user\_item\_matrix\_normalized\_40.loc[userId]

        for column\_name, column\_data in row.iteritems():

            user\_item\_matrix\_normalized\_40[column\_name] = predicted\_values[column\_name]

    mae\_list\_40 = []

    for idx, row in user\_item\_matrix\_normalized\_40.iterrows():

        print("I am outside")

        print(user\_item\_matrix\_Validation.index)

        print(idx)

        if idx in user\_item\_matrix\_Validation\_normalized.index:

            print("I am inside")

            mae = 0

            for column\_name, column\_data in row.iteritems():

                if column\_name in user\_item\_matrix\_Validation.columns:

                    mae += abs(user\_item\_matrix\_Validation\_normalized.loc[idx,column\_name] - user\_item\_matrix\_normalized\_40.loc[idx,column\_name])

            mae = mae /  user\_item\_matrix\_normalized\_40.shape[1]

            mae\_list\_40.append(mae)

    print(mae\_list\_40)

    if len(mae\_list\_40) > 0:

        mae\_40 = sum(mae\_list\_40) / len(mae\_list\_40)

        K\_40.append(mae\_40)

    user\_item\_matrix\_normalized\_50 = user\_item\_matrix\_normalized.copy()

    for userId in UserVsUser\_dict\_50:

        similar\_Users = UserVsUser\_dict\_50[userId]

        predicted\_values = user\_item\_matrix\_normalized\_50.loc[similar\_Users].mean(axis=0)

        row = user\_item\_matrix\_normalized\_50.loc[userId]

        for column\_name, column\_data in row.iteritems():

            user\_item\_matrix\_normalized\_50[column\_name] = predicted\_values[column\_name]

    mae\_list\_50 = []

    for idx, row in user\_item\_matrix\_normalized\_50.iterrows():

        print("I am outside")

        print(user\_item\_matrix\_Validation.index)

        print(idx)

        if idx in user\_item\_matrix\_Validation\_normalized.index:

            print("I am inside")

            mae = 0

            for column\_name, column\_data in row.iteritems():

                if column\_name in user\_item\_matrix\_Validation.columns:

                    mae += abs(user\_item\_matrix\_Validation\_normalized.loc[idx,column\_name] - user\_item\_matrix\_normalized\_50.loc[idx,column\_name])

            mae = mae /  user\_item\_matrix\_normalized\_50.shape[1]

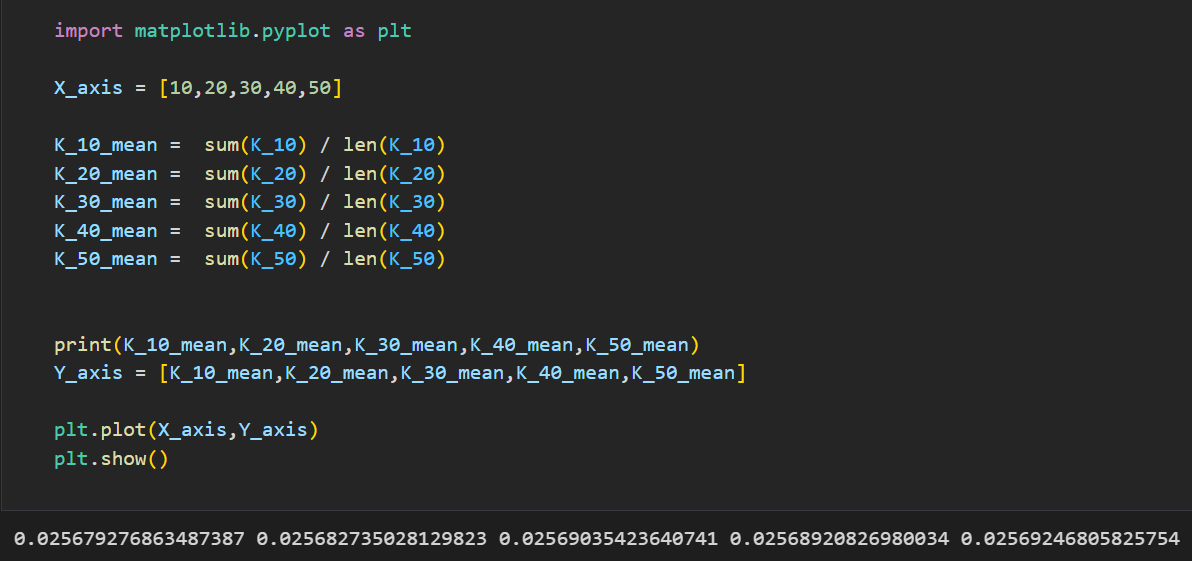
            mae\_list\_50.append(mae)

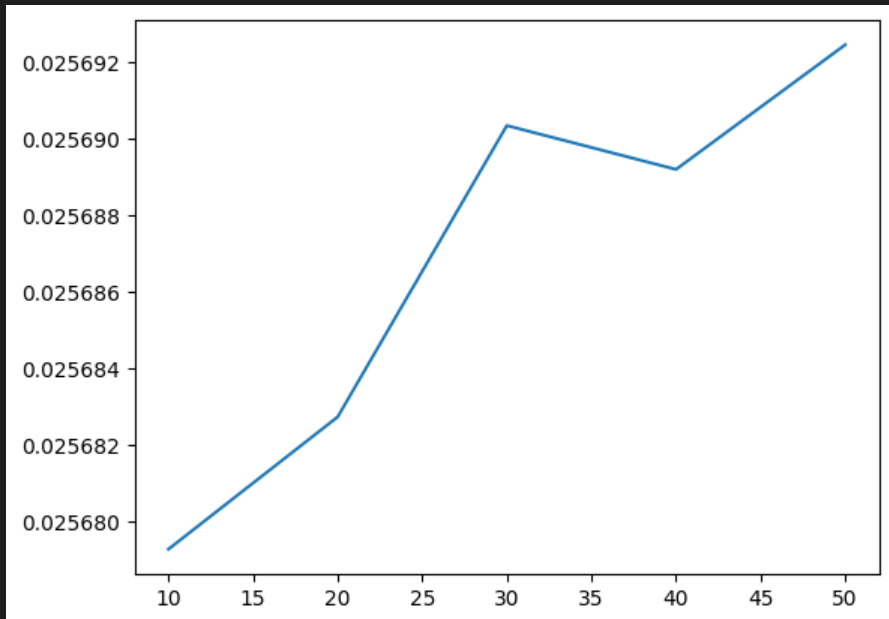
    print(mae\_list\_50)

    if len(mae\_list\_50) > 0:

        mae\_50 = sum(mae\_list\_50) / len(mae\_list\_50)

        K\_50.append(mae\_50)





from sklearn.preprocessing import MinMaxScaler

# citattion:- https://stackoverflow.com/a/55129763

K\_10 = []

K\_20 = []

K\_30 = []

K\_40 = []

K\_50 = []

scaler = MinMaxScaler()

item\_user\_matrix\_Validation = PreProcesses\_HeadPhones\_df\_subset\_11.pivot\_table(index='asin', columns='reviewerID', values='overall', aggfunc='first')

item\_user\_matrix\_Validation\_Inverse = item\_user\_matrix\_Validation.T

item\_user\_matrix\_Validation\_Inverse\_scaled = pd.DataFrame(scaler.fit\_transform(item\_user\_matrix\_Validation\_Inverse.values), columns=item\_user\_matrix\_Validation\_Inverse.columns, index=item\_user\_matrix\_Validation\_Inverse.index)

item\_user\_matrix\_Validation\_normalized = item\_user\_matrix\_Validation\_Inverse\_scaled.T

item\_user\_matrix\_Validation\_normalized.fillna(-1, inplace=True)

i = 1

for train\_idx, test\_idx in k\_fold.split(PreProcesses\_HeadPhones\_df\_subset\_11):

    print(i)

    i += 1

    X\_F\_train, X\_F\_test = PreProcesses\_HeadPhones\_df\_subset\_11.iloc[train\_idx,:],PreProcesses\_HeadPhones\_df\_subset\_11.iloc[test\_idx,:]

    print("X\_F\_train", X\_F\_train)

    print("X\_F\_test", X\_F\_test)

    item\_user\_matrix = X\_F\_train.pivot\_table(index='asin', columns='reviewerID', values='overall', aggfunc='first')

    scaler = MinMaxScaler()

    item\_user\_matrix\_Inverse = item\_user\_matrix.T

    item\_user\_matrix\_Inverse\_scaled = pd.DataFrame(scaler.fit\_transform(item\_user\_matrix\_Inverse.values), columns=item\_user\_matrix\_Inverse.columns, index=item\_user\_matrix\_Inverse.index)

    item\_user\_matrix\_normalized = item\_user\_matrix\_Inverse\_scaled.T

    item\_user\_matrix\_normalized.fillna(-1, inplace=True)

    ItemVsItem\_similairity\_matrix = pd.DataFrame(index=item\_user\_matrix\_normalized.index, columns=item\_user\_matrix\_normalized.index)

    for idx1, row1 in item\_user\_matrix\_normalized.iterrows():

        for idx2, row2 in item\_user\_matrix\_normalized.iterrows():

            cosine\_similarity\_withall\_values = cosine\_similarity(row1.values.reshape(1, -1), row2.values.reshape(1, -1))

            print(cosine\_similarity\_withall\_values)

            ItemVsItem\_similairity\_matrix.at[idx1, idx2] = cosine\_similarity\_withall\_values[0][0]

    ItemVsItem\_dict\_10 = {}

    for idx, row in ItemVsItem\_similairity\_matrix.iterrows():

        top\_11\_items = row.sort\_values(ascending=False).head(11).index.tolist()

        print(top\_11\_items)

        ItemVsItem\_dict\_10[idx] = top\_11\_items

        print(idx, ItemVsItem\_dict\_10[idx])

        # print(idx,UservsUser\_similairity\_matrix.loc[row] )

        ItemVsItem\_dict\_10[idx].remove(idx)

    ItemVsItem\_dict\_20 = {}

    for idx, row in ItemVsItem\_similairity\_matrix.iterrows():

        top\_21\_items = row.sort\_values(ascending=False).head(21).index.tolist()

        print(top\_21\_items)

        ItemVsItem\_dict\_20[idx] = top\_21\_items

        print(idx, ItemVsItem\_dict\_20[idx])

        # print(idx,UservsUser\_similairity\_matrix.loc[row] )

        ItemVsItem\_dict\_20[idx].remove(idx)

    # print("UserVsUser\_dict\_20")

    # print(UserVsUser\_dict\_20)

    ItemVsItem\_dict\_30 = {}

    for idx, row in ItemVsItem\_similairity\_matrix.iterrows():

        top\_31\_items = row.sort\_values(ascending=False).head(31).index.tolist()

        print(top\_31\_items)

        ItemVsItem\_dict\_30[idx] = top\_31\_items

        print(idx, ItemVsItem\_dict\_30[idx])

        # print(idx,UservsUser\_similairity\_matrix.loc[row] )

        ItemVsItem\_dict\_30[idx].remove(idx)

    # print("UserVsUser\_dict\_30")

    # print(UserVsUser\_dict\_30)

    ItemVsItem\_dict\_40 = {}

    for idx, row in ItemVsItem\_similairity\_matrix.iterrows():

        top\_41\_items = row.sort\_values(ascending=False).head(41).index.tolist()

        print(top\_41\_items)

        ItemVsItem\_dict\_40[idx] = top\_41\_items

        print(idx, ItemVsItem\_dict\_40[idx])

        # print(idx,UservsUser\_similairity\_matrix.loc[row] )

        ItemVsItem\_dict\_40[idx].remove(idx)

    # print("UserVsUser\_dict\_40")

    # print(UserVsUser\_dict\_40)

    ItemVsItem\_dict\_50 = {}

    for idx, row in ItemVsItem\_similairity\_matrix.iterrows():

        top\_51\_items = row.sort\_values(ascending=False).head(51).index.tolist()

        print(top\_51\_items)

        ItemVsItem\_dict\_50[idx] = top\_51\_items

        print(idx, ItemVsItem\_dict\_50[idx])

        # print(idx,UservsUser\_similairity\_matrix.loc[row] )

        ItemVsItem\_dict\_50[idx].remove(idx)

    # print("UserVsUser\_dict\_50")

    # print(UserVsUser\_dict\_50)

    print("UservsUser\_similairity\_matrix",ItemVsItem\_similairity\_matrix)

    item\_user\_matrix\_normalized\_10 = item\_user\_matrix\_normalized.copy()

    # mae = mean\_absolute\_error(user\_item\_matrix\_Validation\_normalized.iloc[0].values, user\_item\_matrix\_normalized\_10.iloc[0].values)

    print(item\_user\_matrix\_Validation\_normalized.iloc[0].values, item\_user\_matrix\_normalized\_10.iloc[0].values)

    print("user\_item\_matrix\_normalized", item\_user\_matrix\_normalized)

    print("user\_item\_matrix\_normalized\_10", item\_user\_matrix\_normalized\_10)

    print("user\_item\_matrix\_Validation", item\_user\_matrix\_Validation\_normalized)

    for itemId in ItemVsItem\_dict\_10:

        similar\_Items = ItemVsItem\_dict\_10[itemId]

        predicted\_values = item\_user\_matrix\_normalized\_10.loc[similar\_Items].mean(axis=0)

        row = item\_user\_matrix\_normalized\_10.loc[itemId]

        for column\_name, column\_data in row.iteritems():

            item\_user\_matrix\_normalized\_10[column\_name] = predicted\_values[column\_name]

    mae\_list\_10 = []

    for idx, row in item\_user\_matrix\_normalized\_10.iterrows():

        print("I am outside")

        print(item\_user\_matrix\_Validation.index)

        print(idx)

        if idx in item\_user\_matrix\_Validation\_normalized.index:

            print("I am inside")

            mae = 0

            for column\_name, column\_data in row.iteritems():

                if column\_name in item\_user\_matrix\_Validation.columns:

                    mae += abs(item\_user\_matrix\_Validation\_normalized.loc[idx,column\_name] - item\_user\_matrix\_normalized\_10.loc[idx,column\_name])

            mae = mae /  item\_user\_matrix\_normalized\_10.shape[1]

            mae\_list\_10.append(mae)

    print(mae\_list\_10)

    if len(mae\_list\_10) > 0:

        mae\_10 = sum(mae\_list\_10) / len(mae\_list\_10)

        K\_10.append(mae\_10)

    item\_user\_matrix\_normalized\_20 = item\_user\_matrix\_normalized.copy()

    for itemId in ItemVsItem\_dict\_20:

        similar\_Items = ItemVsItem\_dict\_20[itemId]

        predicted\_values = item\_user\_matrix\_normalized\_20.loc[similar\_Items].mean(axis=0)

        row = item\_user\_matrix\_normalized\_20.loc[itemId]

        for column\_name, column\_data in row.iteritems():

            item\_user\_matrix\_normalized\_20[column\_name] = predicted\_values[column\_name]

    mae\_list\_20 = []

    for idx, row in item\_user\_matrix\_normalized\_20.iterrows():

        print("I am outside")

        print(item\_user\_matrix\_Validation.index)

        print(idx)

        if idx in item\_user\_matrix\_Validation\_normalized.index:

            print("I am inside")

            mae = 0

            for column\_name, column\_data in row.iteritems():

                if column\_name in item\_user\_matrix\_Validation.columns:

                    mae += abs(item\_user\_matrix\_Validation\_normalized.loc[idx,column\_name] - item\_user\_matrix\_normalized\_20.loc[idx,column\_name])

            mae = mae /  item\_user\_matrix\_normalized\_20.shape[1]

            mae\_list\_20.append(mae)

    print(mae\_list\_20)

    if len(mae\_list\_20) > 0:

        mae\_20 = sum(mae\_list\_20) / len(mae\_list\_20)

        K\_20.append(mae\_20)

    item\_user\_matrix\_normalized\_30 = item\_user\_matrix\_normalized.copy()

    for itemId in ItemVsItem\_dict\_30:

        similar\_Items = ItemVsItem\_dict\_30[itemId]

        predicted\_values = item\_user\_matrix\_normalized\_30.loc[similar\_Items].mean(axis=0)

        row = item\_user\_matrix\_normalized\_30.loc[itemId]

        for column\_name, column\_data in row.iteritems():

            item\_user\_matrix\_normalized\_30[column\_name] = predicted\_values[column\_name]

    mae\_list\_30 = []

    for idx, row in item\_user\_matrix\_normalized\_30.iterrows():

        print("I am outside")

        print(item\_user\_matrix\_Validation.index)

        print(idx)

        if idx in item\_user\_matrix\_Validation\_normalized.index:

            print("I am inside")

            mae = 0

            for column\_name, column\_data in row.iteritems():

                if column\_name in item\_user\_matrix\_Validation.columns:

                    mae += abs(item\_user\_matrix\_Validation\_normalized.loc[idx,column\_name] - item\_user\_matrix\_normalized\_30.loc[idx,column\_name])

            mae = mae /  item\_user\_matrix\_normalized\_30.shape[1]

            mae\_list\_30.append(mae)

    print(mae\_list\_30)

    if len(mae\_list\_30) > 0:

        mae\_30 = sum(mae\_list\_30) / len(mae\_list\_30)

        K\_30.append(mae\_30)

    item\_user\_matrix\_normalized\_40 = item\_user\_matrix\_normalized.copy()

    for itemId in ItemVsItem\_dict\_40:

        similar\_Items = ItemVsItem\_dict\_40[itemId]

        predicted\_values = item\_user\_matrix\_normalized\_40.loc[similar\_Items].mean(axis=0)

        row = item\_user\_matrix\_normalized\_40.loc[itemId]

        for column\_name, column\_data in row.iteritems():

            item\_user\_matrix\_normalized\_40[column\_name] = predicted\_values[column\_name]

    mae\_list\_40 = []

    for idx, row in item\_user\_matrix\_normalized\_40.iterrows():

        print("I am outside")

        print(item\_user\_matrix\_Validation.index)

        print(idx)

        if idx in item\_user\_matrix\_Validation\_normalized.index:

            print("I am inside")

            mae = 0

            for column\_name, column\_data in row.iteritems():

                if column\_name in item\_user\_matrix\_Validation.columns:

                    mae += abs(item\_user\_matrix\_Validation\_normalized.loc[idx,column\_name] - item\_user\_matrix\_normalized\_40.loc[idx,column\_name])

            mae = mae /  item\_user\_matrix\_normalized\_40.shape[1]

            mae\_list\_40.append(mae)

    print(mae\_list\_40)

    if len(mae\_list\_40) > 0:

        mae\_40 = sum(mae\_list\_40) / len(mae\_list\_40)

        K\_40.append(mae\_40)

    item\_user\_matrix\_normalized\_50 = item\_user\_matrix\_normalized.copy()

    for itemId in ItemVsItem\_dict\_50:

        similar\_Items = ItemVsItem\_dict\_50[itemId]

        predicted\_values = item\_user\_matrix\_normalized\_50.loc[similar\_Items].mean(axis=0)

        row = item\_user\_matrix\_normalized\_50.loc[itemId]

        for column\_name, column\_data in row.iteritems():

            item\_user\_matrix\_normalized\_50[column\_name] = predicted\_values[column\_name]

    mae\_list\_50 = []

    for idx, row in item\_user\_matrix\_normalized\_50.iterrows():

        print("I am outside")

        print(item\_user\_matrix\_Validation.index)

        print(idx)

        if idx in item\_user\_matrix\_Validation\_normalized.index:

            print("I am inside")

            mae = 0

            for column\_name, column\_data in row.iteritems():

                if column\_name in item\_user\_matrix\_Validation.columns:

                    mae += abs(item\_user\_matrix\_Validation\_normalized.loc[idx,column\_name] - item\_user\_matrix\_normalized\_50.loc[idx,column\_name])

            mae = mae /  item\_user\_matrix\_normalized\_50.shape[1]

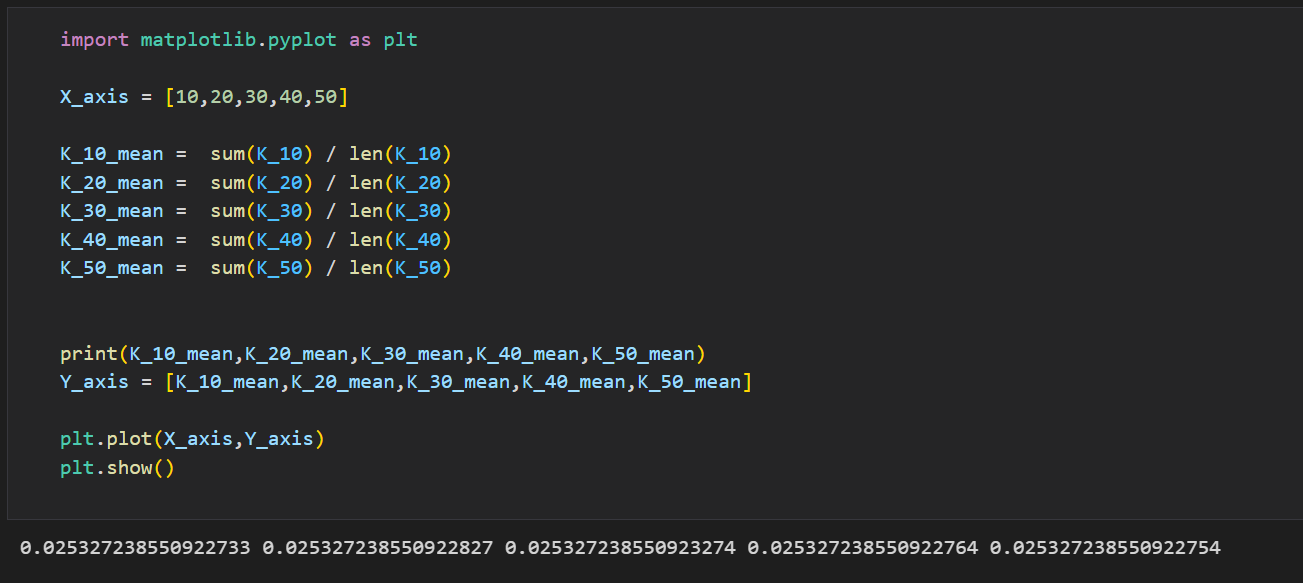
            mae\_list\_50.append(mae)

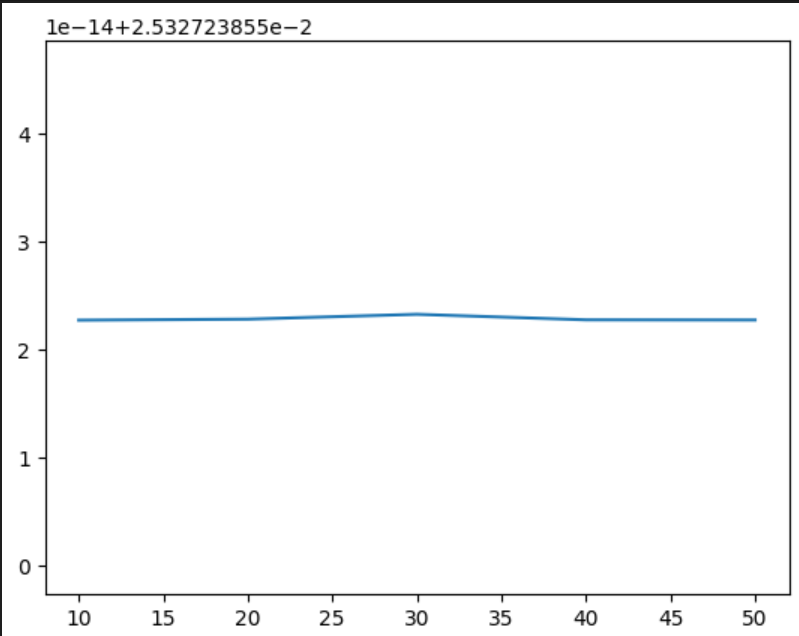
    print(mae\_list\_50)

    if len(mae\_list\_50) > 0:

        mae\_50 = sum(mae\_list\_50) / len(mae\_list\_50)

        K\_50.append(mae\_50)





In User User recommender system:-

I firstly calculated user item matrix that have ratings as value in it for all the reviews dataset.

Then I used 5 fold validation.

For each fold’s training set I created user\_item rating matrix

then found out 10 similar user based on the cosine similairity for each user.

Based on top 10 similar users calculated missing values for the items for each user by taking mean of 10 similar users rating on that items.

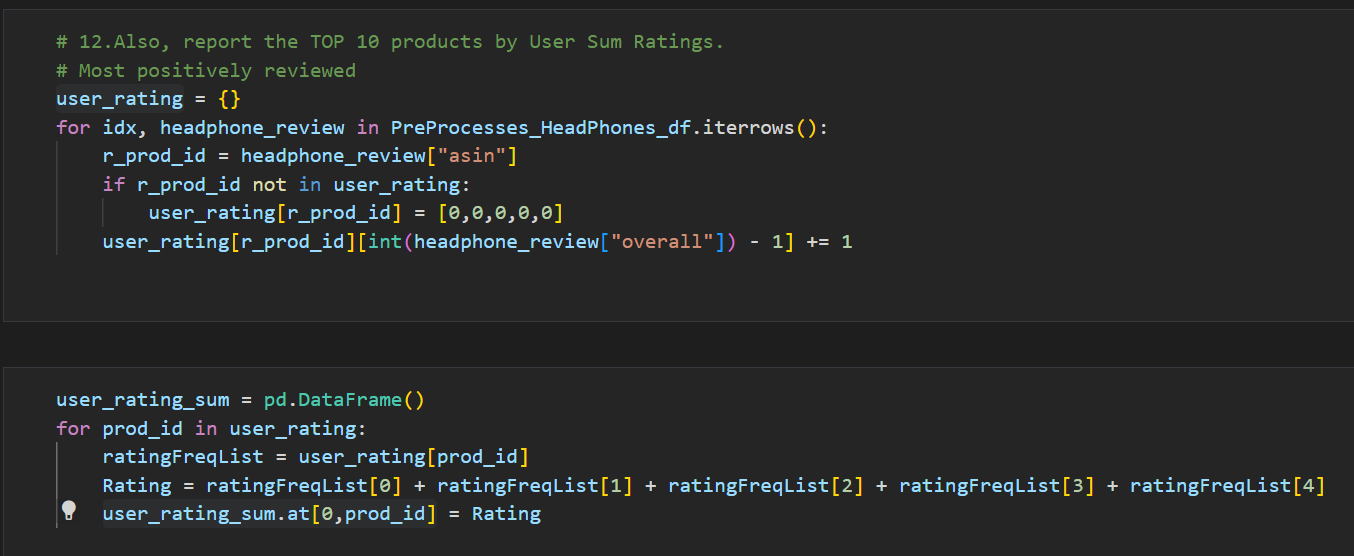
After that I calculated MAE by taking into account the values of ratings for the items for each user in first or globally calculated user item matrix and training set user\_item\_matrix and calculated MAE for each user item rating in user\_item matrix and average out MAE for each user and calculated for whole training set and stored into K\_10 list which stores MAE value for each fold

And did this for 10,20,30,40 and 50.

After finishing all the folds I averaged out the K\_10 and got MAE for K = 10 , do same for all other K values such as 20,30,40,50

Also did the same in item item recommender system and calculated MAE for each k values.

Later from them plot the graphs for each system of K vs mae for each recommender system and compare

12) Also, report the TOP 10 products by User Sum Ratings.

