NAME: VANSHAJ SHARMA

ROLLNO: MT23103

COURSE: M.TECH CSE

YEAR: FIRST YEAR

Libraries Used :-

```
import pandas as pd
import gzip
import json
```

[1] ✓ 4.3s

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

```
def parseZip(path):
    zip = gzip.open(path, 'rb')
    for content in zip:
        yield json.loads(content)
```

```
#1 Saving data into dataFrame

def getFillUpDF(path):
    dataframe = {}
    i = 0
    for content in parseZip(path):
        dataframe[i] = content
        i += 1
    return pd.DataFrame.from_dict(dataframe, orient='index')
```

```python
df = getFillUpDF('Electronics_5.json.gz')
```

```python
# df
```

```python
df.to_pickle("ElectronicsDf.pkl")
```

```python
ElectronicsDf = pd.read_pickle("ElectronicsDf.pkl")
```

	overall	vote	verified	reviewTime	reviewerID	asin	style	reviewerName	reviewText	summary	unixReviewTime	image
0	5.0	67	True	09 18, 1999	AAP7PPBU72QFM	0151004714	{'Format:': ' Hardcover'}	D. C. Carrad	This is the best novel I have read in 2 or 3 y...	A star is born	937612800	NaN
1	3.0	5	True	10 23, 2013	A2E168DTVGE6SV	0151004714	{'Format:': ' Kindle Edition'}	Evy	Pages and pages of introspection, in the style...	A stream of consciousness novel	1382486400	NaN
2	5.0	4	False	09 2, 2008	A1ER5AYS3FQ9O3	0151004714	{'Format:': ' Paperback'}	Kcorn	This is the kind of novel to read when you hav...	I'm a huge fan of the author and this one did ...	1220313600	NaN
3	5.0	13	False	09 4, 2000	A1T17LMQABMBN5	0151004714	{'Format:': ' Hardcover'}	Caf Girl Writes	What gorgeous language! What an incredible wri...	The most beautiful book I have ever read!	968025600	NaN
4	3.0	8	True	02 4, 2000	A3QHJ0FXK33OBE	0151004714	{'Format:': ' Hardcover'}	W. Shane Schmidt	I was taken in by reviews that compared this b...	A dissenting view--In part.	949622400	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...	...	...	...	...
6739585	4.0	NaN	True	03 21, 2017	A33MAQA919J2V8	B01HJH40WU	NaN	Kurt Wurm	These seem like quality USB cables, time will ...	Four Stars	1490054400	NaN
6739586	4.0	NaN	True	01 9, 2017	A1AKHSCPD1BHM4	B01HJH40WU	NaN	C.L Momof3	Works great, love the longer cord. As with any...	Nice long cord	1483920000	NaN
6739587	5.0	2	True	12 1, 2016	A2HUZO7MQAY5I2	B01HJH40WU	NaN	michael clontz	Ok here is an odd thing that happened to me, I...	Not the correct product as linked in the sale.	1480550400	NaN
6739588	5.0	2	True	11 29, 2016	AJJ7VX2L91X2W	B01HJH40WU	NaN	Faith	Works well.	Five Stars	1480377600	NaN
6739589	5.0	NaN	True	03 31, 2017	A1FGCIRPRNZWD5	B01HJF704M	NaN	Brando	I have it plugged into a usb extension on my g...	Works well enough..	1490918400	NaN

```
ElectronicsMetaDf = getFillUpDF('meta_Electronics.json.gz')
```

```
# ElectronicsMetaDf
```

```
ElectronicsMetaDf.to_pickle("ElectronicMetaDF.pkl")
```

```
ElectronicsMetaDf = pd.read_pickle("ElectronicMetaDF.pkl")
```

```
len(ElectronicsMetaDf)
```

```
786445
```

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'.

```
ElectronicsMetaDf['title'] = ElectronicsMetaDf['title'].str.lower()
```

```
# 2
FilteredElectronicsMetaDf = ElectronicsMetaDf[ElectronicsMetaDf['title'].str.contains('headphones | headphone', case= False)]
```

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_ElectronicsM
etaDf.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.

```
PreProcesses_Filtered_ElectronicsMetaDf.to_pickle("PreProcesses_Filtered_ElectronicsMetaDf_V2.0.pkl")
```

```
PreProcesses_Filtered_ElectronicsMetaDf = pd.read_pickle("PreProcesses_Filtered_ElectronicsMetaDf_V2.0.pkl")
```

	category	tech1	description	fit	title	also_buy	tech2	brand	feature	rank	also_view	main_cat	simila
8	[Electronics, Headphones, Earbud Headphones]		[, True High Definition Sound: With ...		wireless bluetooth headphones earbuds with mic...	[]		Enter The Arena	[Superb Sound Quality: Plays crystal clear aud...	[>#950 in Cell Phones & Accessories (See Top 1...	[]	Home Audio & Theater	
47	[Electronics, Headphones]		[Use these high quality headphones for interne...		polaroid pbm2200 pc / gaming stereo headphones...	[]		Polaroid	[Ideal for PC Internet chatting, PC / Console ...	[>#3,548,269 in Cell Phones & Accessories ...	[]	All Electronics	
132	[Electronics, Headphones, Earbud Headphones]		[, True High Definition Sound: With ...		bluetooth workout headphones for running and g...	[]		Enter The Arena	[Superb Sound Quality: Plays crystal clear aud...	[>#4,626,934 in Cell Phones & Accessories (See...	[]	Home Audio & Theater	
223	[Electronics, Headphones, Earbud Headphones]		[, True High Definition Sound: Wit...		bluetooth workout headphones for running and g...	[]		Enter The Arena	[Superb Sound Quality: Plays crystal clear aud...	[>#2,654,020 in Cell Phones & Accessories ...	[]	Home Audio & Theater	
229	[Electronics, Headphones,		[, True High Definition Sound:		bluetooth workout headphones	[]		Enter The	[Superb Sound Quality: Plays	[>#5,289,289 in Cell Phones	[]	Home Audio	

	...	...	...	...	...	...	...	...	...	...	...	...
786395	[Electronics, Headphones, Earbud Headphones]	[, Specification </br> Driver: 5mm</br> ...	maxrock noise isolating sleeping headphones ea...	[]	MAXROCK	[Unique patented silicone design headphones, s...	[>#21,087 in Musical Instruments (See Top 100 ...	[B071WRSL38, B00XCDOGY8, B00V9FN1R4, B00SRAV6V...	Musical Instruments	cla borde hori stri
786400	[Electronics, Accessories & Supplies, Audio & ...	[, Compatible Headphones: - SONY MDR...	geekria® elite headphone shoulder bag / ca...	[]	Geekria	[Saffiano Leather, lightweight and fashionable...	[>#4,760 in Electronics > Accessories & Suppli...	[B0796LWMCR, B019Z81V3M, B00TBELD02, B01CJJ2IF...	Home Audio & Theater	cla borde hori stri
786404	[Electronics, Headphones, Earbud Headphones]	[About the product Rhapsody & Mogan H9 is a m...	wireless bluetooth headset, handsfree wireless...	[]	snorain	[COMFORTABLE CUSTOM FIT Rhapsody & Mogan nois...	[>#343,752 in Cell Phones & Accessories (See T...	[B01D3QZB2Y, B079GFF4HZ, B00XBZY0EI, B00S2P0M1...	All Electronics	cla borde hori stri
786405	[Electronics, Headphones, Earbud Headphones]	[, Specification </br> Driver: 5mm</br> ...	maxrock wired headphones in-ear headphone spor...	[]	MAXROCK	[Unique patented silicone design headphones, s...	[>#59,366 in Musical Instruments (See Top 100 ...	[]	Musical Instruments	cla borde hori stri
786406	[Electronics, Headphones, Earbud Headphones]	[, Specification </br> Driver: 5mm</br> ...	maxrock noise isolating sleeping headphones ea...	[]	MAXROCK	[Total silicon house super comfortable to fit ...	[>#37,846 in Musical Instruments (See Top 100 ...	[]	Musical Instruments	cla borde hori stri

26434 rows × 19 columns

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

```
HeadPhones_df = pd.DataFrame()
for product_id in PreProcesses_Filtered_ElectronicsMetaDf["asin"]:
    HeadPhones_df = pd.concat([HeadPhones_df, ElectronicsDf[(ElectronicsDf["asin"] == product_id)]], ignore_index = True)
```

```
# len(HeadPhones_df)
```

```
HeadPhones_df.to_pickle("HeadPhones_reviews_df.pkl")
```

```
HeadPhones_df = pd.read_pickle("HeadPhones_reviews_df.pkl")
```

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

	overall	vote	verified	reviewTime	reviewerID	asin	style	reviewerName	reviewText	summary	unixReviewTime	image
0	5.0	NaN	True	02 22, 2015	A38RQFVQ1AKJQQ	4126895493	{'Color:': ' Blue W/Mic'}	George Walker	Great headphones. It's just the cord is too sh...	Five Stars	1424563200	NaN
1	5.0	NaN	True	05 8, 2017	A299MRB9O6GWDE	4126895493	{'Color:': ' Blue Zebra W/Mic'}	Carolyn B	Really like these headphone. Wanted something...	Officewear	1494201600	NaN
2	1.0	NaN	True	11 5, 2016	A3ACFC6DQQLIQT	4126895493	{'Color:': ' Blue W/Mic'}	MK	Wire to headphone broke off in less than a mon...	For the money they are fine. Just hope they ho...	1478304000	NaN
3	3.0	NaN	True	09 24, 2016	A36BC0YFDBNB5X	4126895493	{'Color:': ' Green'}	bigboy	Very good	Three Stars	1474675200	NaN
4	1.0	NaN	True	07 17, 2016	A212PQ0HQPNNWM	4126895493	{'Color:': ' Violet Purple'}	Kelly Hales	Currently returning this product because the s...	Currently returning this product because the s...	1468713600	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...
423041	5.0	2	True	09 8, 2016	A50A134UOQSQF	B01HJ8E11E	{'Color:': ' Black'}	charles h evans	Bought this on a flash sale and it's excellent...	Good holder, simple and functional	1473292800	NaN

...	...	...	...	...	...	...	...	...	...	...	...	...
423041	5.0	2	True	09 8, 2016	A50A134UOQSQF	B01HJ8E11E	{'Color:': ' Black'}	charles h evans	Bought this on a flash sale and it's excellent...	Good holder, simple and functional	1473292800	NaN

423042	4.0	NaN	True	08 12, 2016	A2S2R3SUSFHJ61	B01HJ8E11E	{'Color:': ' Black'}	Cheryl Showalters	This is excellent especially for the price.	works great and the price is excellent. Holds...	1470960000	NaN
423043	4.0	NaN	True	09 22, 2018	A3VA3VK4PO1JD	B01HJ8E11E	{'Color:': ' White'}	CD	I have only used these for 1 week at the time ...	They work...	1537574400	NaN
423044	3.0	NaN	True	09 12, 2018	A11TVS6FKXS80H	B01HJ8E11E	{'Color:': ' White'}	Jay Salamon	The product works great, but when it gets down...	30% charge shuts the device off	1536710400	NaN
423045	4.0	NaN	False	08 18, 2018	A3VM9K4M0RQZRQ	B01HJ8E11E	{'Color:': ' White'}	Vincent Roberson Jr	These earphones are very good. I like the desi...	Decent earphones	1534550400	NaN

423046 rows × 12 columns

```python
HeadPhones_df.isna().sum()
```

Python

```
overall                0
vote              364987
verified               0
reviewTime             0
reviewerID             0
asin                   0
style             153493
reviewerName          61
reviewText            53
summary               72
unixReviewTime         0
image             414787
dtype: int64
```

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.

```
len(HeadPhones_df)
```

```
423046
```

```
PreProcesses_HeadPhones_df = HeadPhones_df.copy()
```

```
PreProcesses_HeadPhones_df.isna().sum()
```

```
overall              0
vote            364987
verified             0
reviewTime           0
reviewerID           0
asin                 0
style           153493
reviewerName        61
reviewText          53
summary             72
unixReviewTime       0
image           414787
dtype: int64
```

```python
PreProcesses_HeadPhones_df = PreProcesses_HeadPhones_df.drop(columns= ["vote","style","image"])
```

```python
PreProcesses_HeadPhones_df = PreProcesses_HeadPhones_df.loc[PreProcesses_HeadPhones_df.astype(str).drop_duplicates().index]
```

```python
PreProcesses_HeadPhones_df = PreProcesses_HeadPhones_df.dropna(subset=['reviewText'])
```

```python
PreProcesses_HeadPhones_df
```

	overall	verified	reviewTime	reviewerID	asin	reviewerName	reviewText	summary	unixReviewTime
0	5.0	True	02 22, 2015	A38RQFVQ1AKJQQ	4126895493	George Walker	Great headphones. It's just the cord is too sh...	Five Stars	1424563200
1	5.0	True	05 8, 2017	A299MRB9O6GWDE	4126895493	Carolyn B	Really like these headphone. Wanted something...	Officewear	1494201600
2	1.0	True	11 5, 2016	A3ACFC6DQQLIQT	4126895493	MK	Wire to headphone broke off in less than a mon...	For the money they are fine. Just hope they ho...	1478304000
3	3.0	True	09 24, 2016	A36BC0YFDBNB5X	4126895493	bigboy	Very good	Three Stars	1474675200
4	1.0	True	07 17, 2016	A212PQ0HQPNNWM	4126895493	Kelly Hales	Currently returning this product because the s...	Currently returning this product because the s...	1468713600
...	...	...	...	...	...	...	...	...	...
423041	5.0	True	09 8, 2016	A50A134UOQSQF	B01HJ8E11E	charles h evans	Bought this on a flash sale and it's excellent...	Good holder, simple and functional	1473292800
423042	4.0	True	08 12, 2016	A2S2R3SUSFHJ61	B01HJ8E11E	Cheryl Showalters	This is excellent especially for the price.	works great and the price is excellent. Holds...	1470960000
423043	4.0	True	09 22, 2018	A3VA3VK4PO1JD	B01HJ8E11E	CD	I have only used these for 1 week at the time ...	They work...	1537574400
423044	3.0	True	09 12, 2018	A11TVS6FKXS80H	B01HJ8E11E	Jay Salamon	The product works great, but when it gets down...	30% charge shuts the device off	1536710400
423045	4.0	False	08 18, 2018	A3VM9K4M0RQZRQ	B01HJ8E11E	Vincent Roberson Jr	These earphones are very good. I like the desi...	Decent earphones	1534550400

389715 rows × 9 columns

```python
PreProcesses_HeadPhones_df.to_pickle("PreProcesses_HeadPhones_df_V2.0.pkl")
```

```python
PreProcesses_HeadPhones_df = pd.read_pickle("PreProcesses_HeadPhones_df_V2.0.pkl")
```

```python
PreProcesses_HeadPhones_df.isna().sum()
```

```
overall            0
verified           0
reviewTime         0
reviewerID         0
asin               0
reviewerName      54
reviewText         0
summary           72
unixReviewTime     0
dtype: int64
```

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.

```python
# 4
Number_of_Reviews = len(PreProcesses_HeadPhones_df)
```

```python
Number_of_Reviews
```

```
389715
```

Total number of reviews related to headphones are the only Total Number of Reviews . So the length of PreProcess_Headphones_df is the answer.

```python
ratings = []
for rating in PreProcesses_HeadPhones_df["overall"]:
    ratings.append(rating)

sum_of_ratings = sum(ratings)
Average_Rating_Score = sum_of_ratings / len(ratings)
```

```python
Average_Rating_Score
```

```
4.1018718807333565
```

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

```python
Number_of_Unique_Products = len(PreProcesses_Filtered_ElectronicsMetaDf)
```

```python
Number_of_Unique_Products
```

```
26434
```

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

```
Number_of_Good_Rating = len(PreProcesses_HeadPhones_df[(PreProcesses_HeadPhones_df["overall"] >= 3)])
```

```
Number_of_Good_Rating
```

```
334110
```

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.

```
Number_of_Bad_Ratings = len(ratings) - Number_of_Good_Rating
```

```
Number_of_Bad_Ratings
```

```
55605
```

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.

```
# Number of Reviews corresponding to each Rating.
PreProcesses_HeadPhones_df["overall"].value_counts()
```

```
5.0    219798
4.0     75234
3.0     39078
1.0     29809
2.0     25796
Name: overall, dtype: int64
```

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

5. Preprocess the Text

Libraries used:-

```
from bs4 import BeautifulSoup
import unidecode
import nltk
nltk.download('wordnet')
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
import string
import re
```

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

```
def preprocessText(text):
    Html_free_text =  BeautifulSoup(text,"html.parser").text
    Accent_removed_text = unidecode.unidecode(Html_free_text)
    # Special_char_removed_text = ''.join(character for character in Accent_removed_text if character.isalnum())
    Special_char_removed_text = re.sub(r'[^a-zA-Z0-9\s]', '', Accent_removed_text)
    tokens = nltk.word_tokenize(Special_char_removed_text)
    lemetized_text = ''
    for token in tokens:
        text = lemmatizer.lemmatize(token)
        lemetized_text = lemetized_text +" "+text
    lower_case_text = lemetized_text.lower()
    stripped_text = lower_case_text.strip()
    translator = str.maketrans('', '', string.punctuation)
    punctuation_removed_text = stripped_text.translate(translator)
    return punctuation_removed_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.

```
# review text
reviewTextCol = PreProcesses_HeadPhones_df["reviewText"]
new_reviewTextCol = reviewTextCol.apply(preprocessText)
```

```
C:\Users\hp\AppData\Local\Temp\ipykernel_21000\2450192451.py:2: MarkupR
  Html_free_text =  BeautifulSoup(text,"html.parser").text
C:\Users\hp\AppData\Local\Temp\ipykernel_21000\2450192451.py:2: MarkupR
  Html_free_text =  BeautifulSoup(text,"html.parser").text
```

```
PreProcesses_HeadPhones_df["reviewText"] = new_reviewTextCol
```

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

```
PreProcesses_HeadPhones_df
```

	overall	verified	reviewTime	reviewerID	asin	reviewerName	reviewText	summary	unixReviewTime
0	5.0	True	02 22, 2015	A38RQFVQ1AKJQQ	4126895493	George Walker	great headphone its just the cord is too short	Five Stars	1424563200
1	5.0	True	05 8, 2017	A299MRB9O6GWDE	4126895493	Carolyn B	really like these headphone wanted something f...	Officewear	1494201600
2	1.0	True	11 5, 2016	A3ACFC6DQQLIQT	4126895493	MK	wire to headphone broke off in le than a month...	For the money they are fine. Just hope they ho...	1478304000
3	3.0	True	09 24, 2016	A36BC0YFDBNB5X	4126895493	bigboy	very good	Three Stars	1474675200
4	1.0	True	07 17, 2016	A212PQ0HQPNNWM	4126895493	Kelly Hales	currently returning this product because the s...	Currently returning this product because the s...	1468713600
...	...	...	...	...	...	...	...	...	...
389710	5.0	True	09 8, 2016	A50A134UOQSQF	B01HJ8E11E	charles h evans	bought this on a flash sale and it excellent i...	Good holder, simple and functional	1473292800
389711	4.0	True	08 12, 2016	A2S2R3SUSFHJ61	B01HJ8E11E	Cheryl Showalters	this is excellent especially for the price	works great and the price is excellent. Holds...	1470960000
389712	4.0	True	09 22, 2018	A3VA3VK4PO1JD	B01HJ8E11E	CD	i have only used these for 1 week at the time ...	They work...	1537574400
389713	3.0	True	09 12, 2018	A11TVS6FKXS80H	B01HJ8E11E	Jay Salamon	the product work great but when it get down to...	30% charge shuts the device off	1536710400
389714	4.0	False	08 18, 2018	A3VM9K4M0RQZRQ	B01HJ8E11E	Vincent Roberson Jr	these earphone are very good i like the design...	Decent earphones	1534550400

389715 rows × 9 columns

```python
PreProcesses_HeadPhones_df.to_pickle("text_PreProcesses_HeadPhones_df_10000.pkl")
```

```python
PreProcesses_HeadPhones_df = pd.read_pickle("text_PreProcesses_HeadPhones_df_10000.pkl")
```
✓ 3.5s

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.

```python
Preprocess_headphones_VC_df = PreProcesses_HeadPhones_df["asin"].value_counts()
```

```python
Preprocess_headphones_VC_df
```

```
B004WODP20    3117
B00BN0N0LW    3104
B00LP6CFEC    2559
B00STP86CW    2497
B007FHX9OK    2243
              ...
B0015AM39K       2
B00172PW62       2
B00180GZBY       2
B000YHWVRO       2
B0014AWALM       1
Name: asin, Length: 7913, dtype: int64
```

```python
Preprocess_headphones_asin_VC_df = Preprocess_headphones_VC_df.index.tolist()
```

All the brands that are being reviewed converted into list and stored into Preprocess_headphones_asin_VC_df

```
Preprocess_headphones_asin_VC_df
```

```
['B004WODP20',
 'B00BN0N0LW',
 'B00LP6CFEC',
 'B00STP86CW',
 'B007FHX9OK',
 'B00EEHNNNG',
 'B00JJ2C0S0',
 'B000067RC4',
 'B003LPTAYI',
 'B00AWIPITS',
 'B00004T8R2',
 'B005LKB0IU',
 'B0002H02ZY',
 'B00NBEWB4U',
 'B008EPW1MI',
 'B00Q2VPI8A',
 'B000ULAP4U',
 'B0007NWL70',
```

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.

```
brands = []
for asin in Preprocess_headphones_asin_VC_df:
    brands.append(PreProcesses_Filtered_ElectronicsMetaDf[(PreProcesses_Filtered_ElectronicsMetaDf["asin"] == asin)])
```

```
brands
```

```
[                                              category tech1  \
 226915  [Electronics, Headphones, Over-Ear Headphones]

                                              description fit  \
 226915  [Sony MDR-ZX100 Headphone - Stereo - Black - M...

                                   title  \
 226915  sony mdrzx100 headphones (black)

                                                 also_buy tech2 brand  \
 226915  [B00NJ2M33I, B00JVFS020, B003M8NVFS, B00OUX6U6...        Sony

                                                  feature  \
 226915  [Connectivity Technology: Wired, 30mm Multi-la...

                                                      rank also_view  \
 226915  [>#37,736 in Cell Phones & Accessories (See To...        []

                       main_cat  \
 226915  Home Audio & Theater

                                              similar_item            date  \
 226915   class="a-bordered a-horizontal-stripes  a-spa...  March 11, 2011
```

```python
brandsName = []
for brand in brands:
    name = ''.join(brand["brand"])
    if name not in brandsName:
        brandsName.append(name)
```

```python
len(brandsName)
```

```
2311
```

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

```
brandsName
```

```
['Sony',
 'Toysdone',
 'XBRN',
 'iNassen',
 'Fourcase',
 'Etre Jeune',
 'Belkin',
 'Sennheiser',
 'Kinivo',
 'Panasonic',
 'AmazonBasics',
 'ShamBo',
 'Bluedio',
 'Audio-Technica',
 'Kidz Gear',
 'JVC',
 'Jaybird',
 'Koss',
 'ABCShopUSA',
 'Photive',
 'Roku',
 'Beyution Factory',
 'Mpow',
```

```
Top_20_brandsName = brandsName[0:20]
```

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

```
Top_20_brandsName
```

```
['Sony',
 'Toysdone',
 'XBRN',
 'iNassen',
 'Fourcase',
 'Etre Jeune',
 'Belkin',
 'Sennheiser',
 'Kinivo',
 'Panasonic',
 'AmazonBasics',
 'ShamBo',
 'Bluedio',
 'Audio-Technica',
 'Kidz Gear',
 'JVC',
 'Jaybird',
 'Koss',
 'ABCShopUSA',
 'Photive']
```

```
Top_least_20_brands = brandsName[:(len(brandsName) - 20) - 1:-1]
```

```
Top_least_20_brands
```

```
['Honda',
 'AIRDRIVES',
 'DSI',
 'NOIZY Brands',
 'SOUND-SQUARED CO.',
 'DetectorPro',
 'California Cable Market',
 "Bell'O Digital",
 'OCR',
 'TomTom',
 'Comfort Audio',
 'Gerod',
 'Fantime',
 'IFOXTEK',
 'Shensee',
 'Gear4',
 'MAXELL(R)',
 'EMPIRE AUDIO USA',
 'Moki International',
 'SmartDelux']
```

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.

```
# Most positively reviewed
most_positively_reviewed = {}
for idx, headphone_review in PreProcesses_HeadPhones_df.iterrows():
    r_prod_id = headphone_review["asin"]
    if r_prod_id not in most_positively_reviewed:
        most_positively_reviewed[r_prod_id] = [0,0,0,0,0]
    most_positively_reviewed[r_prod_id][int(headphone_review["overall"]) - 1] += 1
```

```
most_positively_reviewed
maximumPosReview_no = 0
maximumPosReview_prod_id = ""
for prod_id in most_positively_reviewed:
    ratingFreqList = most_positively_reviewed[prod_id]
    positiveRating = ratingFreqList[2] + ratingFreqList[3] + ratingFreqList[4]
    if positiveRating > maximumPosReview_no:
        maximumPosReview_no = positiveRating
        maximumPosReview_prod_id = prod_id
```

```
print("So the most positively reviewed is "+ maximumPosReview_prod_id)
```

```
So the most positively reviewed is B004WODP20
```

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

```
# Show the count of ratings for the product over 5 consecutive years.
def getYear(time):
    return int(''.join(time).split(",")[1].strip())
years = PreProcesses_HeadPhones_df["reviewTime"].apply(getYear).tolist()
```

So we found year for all reviewTime values in PreProcess_Headphones_df and stored as list into year variable

```
sorted(years)
```

```
[2000,
 2000,
 2000,
 2000,
 2000,
 2000,
 2000,
 2000,
 2000,
 2000,
 2000,
 2000,
 2000,
 2000,
 2000,
 2000,
 2000,
 2000,
 2000,
 2000,
 2000,
```

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.

```
yearVsRating = {}
for idx, headphone_review in PreProcesses_HeadPhones_df.iterrows():
    year = int(''.join(headphone_review["reviewTime"]).split(",")[1].strip())
    if year >= 2011 and year <= 2015:
        if year not in yearVsRating:
            yearVsRating[year] = [0,0,0,0,0]
        yearVsRating[year][int(headphone_review["overall"]) - 1] += 1
```

```
yearVsRating
```

```
{2015: [6800, 5999, 8904, 17413, 53828],
 2014: [3877, 3468, 5871, 11650, 33730],
 2012: [1066, 988, 1512, 3177, 7231],
 2013: [2043, 1981, 3426, 7306, 17915],
 2011: [640, 649, 923, 1899, 3716]}
```

For wordCloud the libraries used are wordcloud and nltk

```
# Form a Word Cloud for 'Good' and 'Bad' ratings. Report the most comm
# Citation :- https://stackoverflow.com/a/48750930
from wordcloud import WordCloud
from nltk.corpus import stopwords
import matplotlib.pyplot as plt
StopWordsList = stopwords.words("english")
```

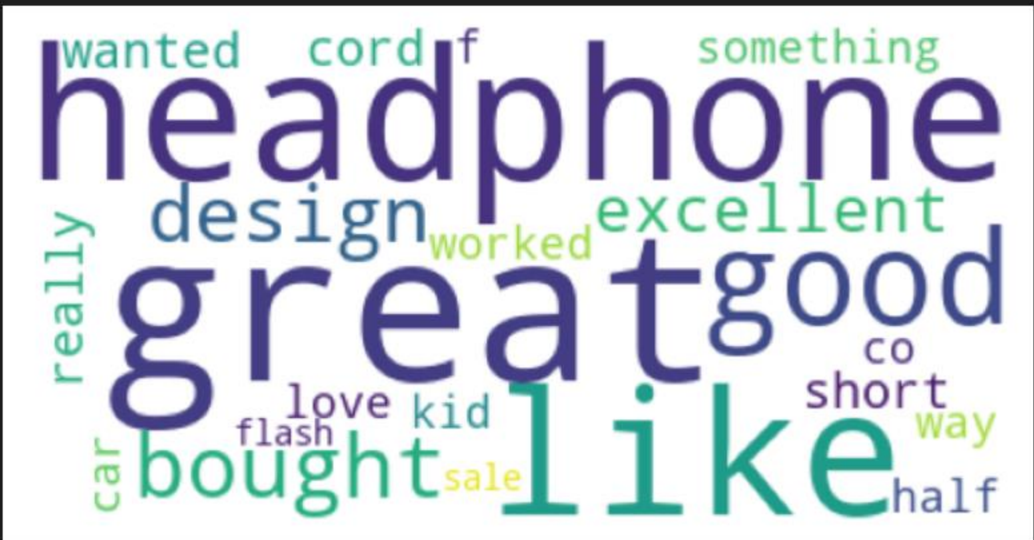
```
Click to add a breakpoint = PreProcesses_HeadPhones_df[(PreProcesses_HeadPhones_df["overall"] >= 3)]
GoodReviews_df = GoodReviews_row_df["reviewText"]
BadReviews_row_df = PreProcesses_HeadPhones_df[(PreProcesses_HeadPhones_df["overall"] < 3)]
BadReviews_df = BadReviews_row_df["reviewText"]
```

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.

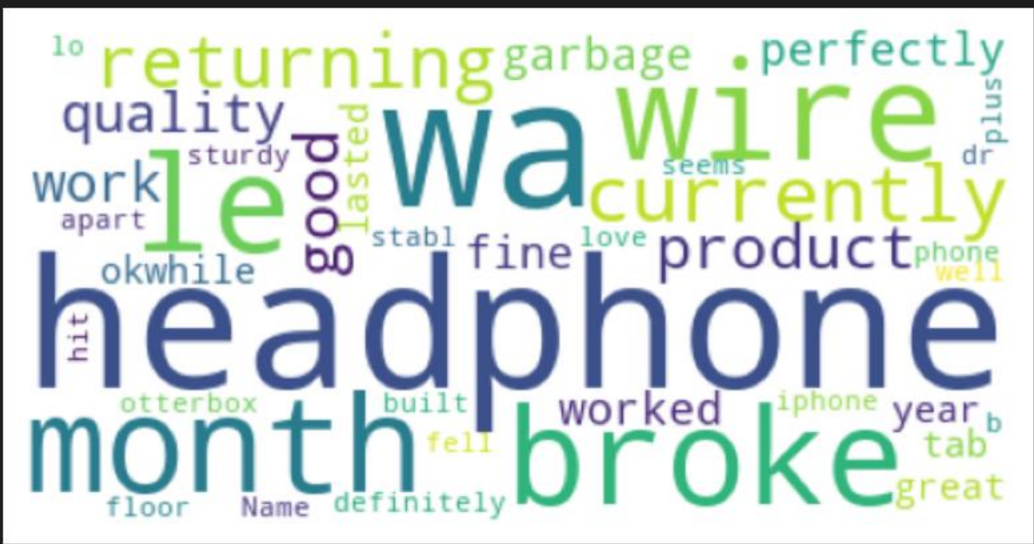
```
GoodWordcloud = WordCloud(
        background_color='white',
        stopwords=StopWordsList,
        min_font_size = 10
    ).generate(str(GoodReviews_df))

BadWordcloud = WordCloud(
        background_color='white',
        stopwords=StopWordsList,
        min_font_size = 10
    ).generate(str(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.

```
plt.axis('off')
plt.imshow(GoodWordcloud)
plt.show()
```



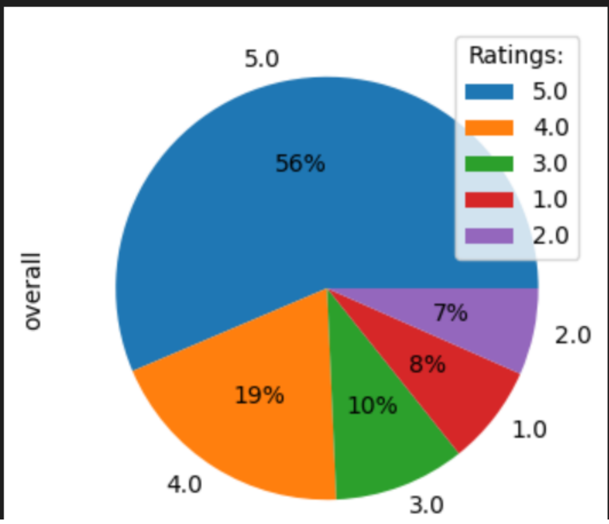

```
plt.axis('off')
plt.imshow(BadWordcloud)
plt.show()
```



```python
# Plot a pie chart for Distribution of Ratings vs. the No. of Reviews.
import matplotlib.pyplot as plt

RatingsValueCount_df = PreProcesses_HeadPhones_df["overall"].value_counts()
```

+ Code + Markdown

```python
RatingsValueCount_df.plot(kind='pie', subplots=True, figsize=(4, 4),autopct='%1.0f%%')
plt.legend(title = "Ratings:")
plt.show()
```



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.

```python
# Report in which year the product got maximum reviews.
yearReviewMap = {}
for idx, headphone_review in PreProcesses_HeadPhones_df.iterrows():
    year = int(''.join(headphone_review["reviewTime"]).split(",")[1].strip())
    if year not in yearReviewMap:
        yearReviewMap[year] = 0
    yearReviewMap[year] = yearReviewMap[year] + 1
```

```python
maxNoofReview = 0
YearHavingMaxReview = 0
for year in yearReviewMap:
    if yearReviewMap[year] > maxNoofReview:
        maxNoofReview = yearReviewMap[year]
        YearHavingMaxReview = year
```

```python
YearHavingMaxReview
```

```
2016
```

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.

```python
# Which year has the highest number of Customers?
yearVsCustomers = {}
for idx, headphone_review in PreProcesses_HeadPhones_df.iterrows():
    year = int(''.join(headphone_review["reviewTime"]).split(",")[1].strip())
    reviewerId = ''.join(headphone_review["reviewerID"])
    if year not in yearVsCustomers:
        yearVsCustomers[year] = []
    if  reviewerId not in yearVsCustomers[year]:
        yearVsCustomers[year].append(reviewerId)
```

```python
yearMaxCustomer = 0
MaxCustomer = 0
for year in yearVsCustomers:
    if len(yearVsCustomers[year]) > MaxCustomer:
        MaxCustomer = len(yearVsCustomers[year])
        yearMaxCustomer = year
```

```python
print(f"year having max customer is {yearMaxCustomer}")
```

```
year having max customer 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)

```python
PreProcesses_HeadPhones_df_subset = PreProcesses_HeadPhones_df.sample(n=4000)
```

✓ 0.2s

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.

```
# 7
# citation:- https://spotintelligence.com/2022/12/20/bag-of-words-python/#:~:text=Scik
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(max_features=10000)
```

```
reviews = []
for review in PreProcesses_HeadPhones_df_subset["reviewText"]:
    reviews.append(review)
```

+ Code + Markdown

```
reviews
```

```
['i love this case better than any ive ever had i drop my phone a lot and it ha kept it s
 'im a gadget geek and here is my review of this product it function a intended but there
 'although the earbuds are fairly large the earbuds produce a great sound with high quali
 'great product i use this a an in ear monitor at church and it work wonder cancelling no
 'very good fit and quality for replacement headphone pad',
 'i wanted a low cost option for connecting wireless headphone in the bedroom so i could
 'these headphone arent bad they had very good review and for the price range i wa lookin
 'its all cheap plastic and very light it probably cost 1 to produce this stand hence 14
```

```python
vectorizer.fit(reviews)
```

```
▾ CountVectorizer
CountVectorizer(max_features=10000)
```

```python
# type(headphone_review["reviewText"])
```

```python
BAG_of_Words = vectorizer.transform(reviews)
```

```python
print(BAG_of_Words.getnnz())
```

```
189027
```

```python
TF = pd.DataFrame(BAG_of_Words.toarray(), columns=vectorizer.get_feature_names())
```

```python
TF
```

Python

	0000	0000i	01	05242012	055mm	060215	062611	082116	082216	090416	...	zone	zoom	zte	zune	zut	zvox	zx100	zx110	zx2	zx300
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
3995	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3996	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3997	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3998	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3999	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

4000 rows × 10000 columns

```python
#TF_idf
from sklearn.feature_extraction.text import TfidfTransformer

Transformer = TfidfTransformer()
TF_IDF_Vector = Transformer.fit_transform(BAG_of_Words)
```

```python
TF_IDF_Vector
```

```
<4000x10000 sparse matrix of type '<class 'numpy.float64'>'
	with 189027 stored elements in Compressed Sparse Row format>
```

```python
TF_IDF = pd.DataFrame(TF_IDF_Vector.toarray() ,columns=vectorizer.get_feature_names())
```

```python
TF_IDF
```

Pytho

	0000	0000i	01	05242012	055mm	060215	062611	082116	082216	090416	...	zone	zoom	zte	zune	zut	zvox	zx100	zx110	zx2	zx300
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
3995	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3996	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3997	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3998	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3999	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

4000 rows × 10000 columns

8)

```python
for idx, headphone_review in PreProcesses_HeadPhones_df_subset.iterrows():
    rating = int(headphone_review["overall"])
    if  rating > 3:
        PreProcesses_HeadPhones_df_subset.at[idx, 'rating_class']  = "Good"
    elif rating == 3:
        PreProcesses_HeadPhones_df_subset.at[idx, 'rating_class']  = "Average"
    elif rating < 3:
        PreProcesses_HeadPhones_df_subset.at[idx, 'rating_class']  = "Bad"
```

```
PreProcesses_HeadPhones_df_subset
```

Python

	overall	verified	reviewTime	reviewerID	asin	reviewerName	reviewText	summary	unixReviewTime	rating_class
140599	5.0	True	09 27, 2016	A3HBF5WVSFUUDB	B00A7NC5Z8	A.B.	folded portapro headphone fit exactly into thi...	Folded PortaPro headphones fit exactly into th...	1474934400	Good
64462	5.0	True	02 24, 2016	A1P383N549ZGS6	B002SOU2Y0	Allen K. Froehlich	excellent	Five Stars	1456272000	Good
181139	3.0	True	08 25, 2014	A3MWICRPOPZD6W	B00EL0EIGC	DT	kind of cheap feeling serves it purpose i thin...	Serves its purpose	1408924800	Average
133299	4.0	True	02 4, 2013	AC310CJFLN3ZA	B009923WIW	rcookenc	when i first got these i had a hard time getti...	Here's a secret you need to know	1359936000	Good
169822	3.0	False	10 15, 2013	A25FJ8W6WCBFEC	B00DIOALYA	amazonian	i am another one who no matter which ear tip i...	Good product but a few problems	1381795200	Average
...	...	...	...	...	...	...	...	...	...	...
318575	5.0	True	08 23, 2016	A3AQ279I3NTVEI	B011L1190G	Davers	replaced a cable that had dropped one of the c...	Fixed a bad connection I had with the previous...	1471910400	Good

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)

```
# 9. From the dataset, take the Review Text as input feature and Rating Class as target
# variable. Divide the data into Train and Test Data in the ratio of 75:25.
# Making dataset of TF_IDF as input feature vs rating class Target Label
dataset = TF_IDF.copy()
```

```
dataset['rating_class'] = PreProcesses_HeadPhones_df_subset['rating_class'].to_numpy()
```

```
dataset
```

Python

	003	005bag	009	0100	011817	02	032717	060215	079mm	09	...	zooming	zs	zte	zune	zvox	zx100	zx100s	zx1the	zxr	rating_class
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	Good
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	Good
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	Average
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	Good
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	Average
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
3995	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	Good
3996	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	Bad
3997	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	Good
3998	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	Good
3999	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	Bad

4000 rows × 11841 columns

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.

```
dataset.to_pickle("ML_Dataset.pkl")
```

```
dataset = pd.read_pickle("ML_Dataset.pkl")
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(dataset.iloc[:, :-1],dataset["rating_class"])
```

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

10)

1) Used Logistic regression on it.

```
from sklearn.linear_model import LogisticRegression

logisticRegressionModel = LogisticRegression()
logisticRegressionModel.fit(X_train,y_train)
prediction = logisticRegressionModel.predict(X_test)
```

```
from sklearn.metrics import classification_report
metricesReport = classification_report(y_test, prediction, output_dict=True)
metricesReport_df = pd.DataFrame(metricesReport).transpose()

print(f"Precision for Logistic regression is:- \n{metricesReport_df['precision']}")
print(f"Recall for Logistic regression is:- \n{metricesReport_df['recall']}")
print(f"f1-score for Logistic regression is:- \n{metricesReport_df['f1-score']}")
print(f"Support for Logistic regression is:- \n{metricesReport_df['support']}")
```

```
Precision for Logistic regression is:-
Average         0.277778
Bad             0.836735
Good            0.830654
accuracy        0.821000
macro avg       0.648389
weighted avg    0.786139
Name: precision, dtype: float64
Recall for Logistic regression is:-
Average         0.060976
Bad             0.303704
Good            0.989783
accuracy        0.821000
macro avg       0.451487
weighted avg    0.821000
Name: recall, dtype: float64
f1-score for Logistic regression is:-
```

```
macro avg       0.648589
weighted avg    0.786139
Name: precision, dtype: float64
Recall for Logistic regression is:-
Average         0.060976
Bad             0.303704
Good            0.989783
accuracy        0.821000
macro avg       0.451487
weighted avg    0.821000
Name: recall, dtype: float64
f1-score for Logistic regression is:-
Average         0.100000
Bad             0.445652
Good            0.903263
accuracy        0.821000
macro avg       0.482972
weighted avg    0.775618
Name: f1-score, dtype: float64
Support for Logistic regression is:-
...
accuracy           0.821
macro avg       1000.000
weighted avg    1000.000
Name: support, dtype: float64
```

Output is truncated. View as a scrollable element o

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

```
from sklearn.svm import SVC

SupportVectorMachineModel = SVC()
SupportVectorMachineModel.fit(X_train,y_train)
```

```
▾ SVC
SVC()
```

+ Code   + Markdown

```
SVM_prediction = SupportVectorMachineModel.predict(X_test)
```

```python
from sklearn.metrics import classification_report
SVM_metricesReport = classification_report(y_test, SVM_prediction, output_dict=True)
SVM_metricesReport_df = pd.DataFrame(SVM_metricesReport).transpose()

print(f"Precision for SVM is:- \n{SVM_metricesReport_df['precision']}")
print(f"Recall for SVM is:- \n{SVM_metricesReport_df['recall']}")
print(f"f1-score for SVM is:- \n{SVM_metricesReport_df['f1-score']}")
print(f"Support for SVM is:- \n{SVM_metricesReport_df['support']}")
```

```
Precision for SVM is:-
Average         1.000000
Bad             0.840000
Good            0.801848
accuracy        0.803000
macro avg       0.880616
weighted avg    0.823247
Name: precision, dtype: float64
Recall for SVM is:-
Average         0.012195
Bad             0.155556
Good            0.997446
accuracy        0.803000
macro avg       0.388399
weighted avg    0.803000
Name: recall, dtype: float64
f1-score for SVM is:-
```

```
Name: precision, dtype: float64
Recall for SVM is:-
Average         0.012195
Bad             0.155556
Good            0.997446
accuracy        0.803000
macro avg       0.388399
weighted avg    0.803000
Name: recall, dtype: float64
f1-score for SVM is:-
Average         0.024096
Bad             0.262500
Good            0.889015
accuracy        0.803000
macro avg       0.391871
weighted avg    0.733512
Name: f1-score, dtype: float64
Support for SVM is:-
...
accuracy           0.803
macro avg       1000.000
weighted avg    1000.000
Name: support, dtype: float64
```

Output is truncated. View as a [scrollable element]

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

```
from sklearn.tree import DecisionTreeClassifier
DecisionTreeModel = DecisionTreeClassifier()
```

```
DecisionTreeModel.fit(X_train, y_train)
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

```
DecisionTreePrediction = DecisionTreeModel.predict(X_test)
```

```python
from sklearn.metrics import classification_report
DT_metricesReport = classification_report(y_test, DecisionTreePrediction, output_dict=True)
DT_metricesReport_df = pd.DataFrame(DT_metricesReport).transpose()

print(f"Precision for Decision Tree is:- \n{DT_metricesReport_df['precision']}")
print(f"Recall for Decision Tree is:- \n{DT_metricesReport_df['recall']}")
print(f"f1-score for Decision Tree is:- \n{DT_metricesReport_df['f1-score']}")
print(f"Support for Decision Tree is:- \n{DT_metricesReport_df['support']}")
```

```
Precision for Decision Tree is:-
Average         0.134146
Bad             0.397163
Good            0.850708
accuracy        0.728000
macro avg       0.460672
weighted avg    0.730721
Name: precision, dtype: float64
Recall for Decision Tree is:-
Average         0.134146
Bad             0.414815
Good            0.844189
accuracy        0.728000
macro avg       0.464383
weighted avg    0.728000
Name: recall, dtype: float64
f1-score for Decision Tree is:-
```

```
Name: recall, dtype: float64
f1-score for Decision Tree is:-
Average         0.134146
Bad             0.405797
Good            0.847436
accuracy        0.728000
macro avg       0.462460
weighted avg    0.729325
Name: f1-score, dtype: float64
Support for Decision Tree is:-
...
accuracy           0.728
macro avg       1000.000
weighted avg    1000.000
Name: support, dtype: float64
```

Output is truncated. View as a scrollable element or open in a text

d) Also used KNN I dataset

```
from sklearn.neighbors import KNeighborsClassifier
KNN_Model = KNeighborsClassifier()
```

```
KNN_Model.fit(X_train, y_train)
```

```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

```
KNN_ModelPrediction = KNN_Model.predict(X_test)
```

```
from sklearn.metrics import classification_report
KNN_metricesReport = classification_report(y_test, KNN_ModelPrediction, output_dict=True)
KNN_metricesReport_df = pd.DataFrame(KNN_metricesReport).transpose()

print(f"Precision for KNN is:- \n{KNN_metricesReport_df['precision']}")
print(f"Recall for KNN is:- \n{KNN_metricesReport_df['recall']}")
print(f"f1-score for KNN is:- \n{KNN_metricesReport_df['f1-score']}")
print(f"Support for KNN is:- \n{KNN_metricesReport_df['support']}")
```

```
Precision for KNN is:-
Average         0.000000
Bad             1.000000
Good            0.783567
accuracy        0.783000
macro avg       0.594522
weighted avg    0.748533
Name: precision, dtype: float64
Recall for KNN is:-
Average         0.000000
Bad             0.007407
Good            0.998723
accuracy        0.783000
macro avg       0.335377
weighted avg    0.783000
Name: recall, dtype: float64
f1-score for KNN is:-
Average         0.000000
Bad             0.014706
Good            0.878158
accuracy        0.783000
macro avg       0.297621
weighted avg    0.689583
Name: f1-score, dtype: float64
Support for KNN is:-
...
accuracy           0.783
macro avg       1000.000
```

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

```
from sklearn.ensemble import RandomForestClassifier
RandomForestModel = RandomForestClassifier()
```

```
RandomForestModel.fit(X_train, y_train)
```

```
▾ RandomForestClassifier
RandomForestClassifier()
```

```
RandomForestPrediction = RandomForestModel.predict(X_test)
```

```
from sklearn.metrics import classification_report
RF_metricesReport = classification_report(y_test, RandomForestPrediction, output_dict=True)
RF_metricesReport_df = pd.DataFrame(RF_metricesReport).transpose()

print(f"Precision for random forest is:- \n{RF_metricesReport_df['precision']}")
print(f"Recall for random forest is:- \n{RF_metricesReport_df['recall']}")
print(f"f1-score for random forest is:- \n{RF_metricesReport_df['f1-score']}")
print(f"Support for random forest is:- \n{RF_metricesReport_df['support']}")
```

```
Precision for random forest is:-
Average         0.000000
Bad             0.800000
Good            0.799589
accuracy        0.798000
macro avg       0.533196
weighted avg    0.734078
Name: precision, dtype: float64
Recall for random forest is:-
Average         0.000000
Bad             0.148148
Good            0.993614
accuracy        0.798000
macro avg       0.380587
weighted avg    0.798000
Name: recall, dtype: float64
f1-score for random forest is:-
Average         0.000000
Bad             0.250000
Good            0.886105
accuracy        0.798000
macro avg       0.378702
weighted avg    0.727570
Name: f1-score, dtype: float64
Support for random forest is:-
```

```
Support for random forest is:-
...
accuracy           0.798
macro avg       1000.000
weighted avg    1000.000
Name: support, dtype: float64
```

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

```
# 11. Collaborative Filtering :
#   a) Create a user-item rating matrix
#   b) Normalize the ratings, by using min-max scaling on user’s reviews
#   c) Create a user-user recommender system - i.e,
#   i) Find the top N similar users, by using cosine similarity. N = 10, 20, 30,
#   40, 50
#   ii) Use K-folds validation. K = 5. Explanation: Create 5 subsets, and take 1
#   of them as the validation set. Take the rest 4 to be the training set.
#   iii) Use the training set to predict the missing values, and use the validation
#   set to calculate the error. (Error = |actual_rating - predicted_rating|)
#   iv) Report the MAE (Mean Absolute Error) for taking K = 10, 20, 30, 40,
#   50 similar users.
#   d) Create an item-item recommender system. Use the same steps as above.
#   e) Plot separate graphs for each of the two recommender systems, plotting
#   MAE against K
```

```
# citation:- https://www.javatpoint.com/k-fold-cross-validation-in-sklearn
from sklearn.model_selection import KFold
k_fold = KFold(n_splits = 5)
```

✓ 0.1s

Created custom cosine similarity function

```
import numpy as np

def customCosineSimilarity(vector1,vector2):
    v1 = vector1
    v2 = vector2
    if(len(v1) != len(v2)):
        return None

    product = np.dot(v1,v2)

    square_sum_vector_1 = 0
    for val in v1:
        square_sum_vector_1 += (val**2)

    square_sum_vector_2 = 0
    for val in v2:
        square_sum_vector_2 += (val**2)

    product_of_sqrt_square_sums_roots = np.sqrt(square_sum_vector_1) * np.sqrt(square_sum_vector_2)

    result = product/product_of_sqrt_square_sums_roots
    return result
```

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

```
PreProcesses_HeadPhones_df_subset_11 = PreProcesses_HeadPhones_df.sample(n = 100)
```

```
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]
```

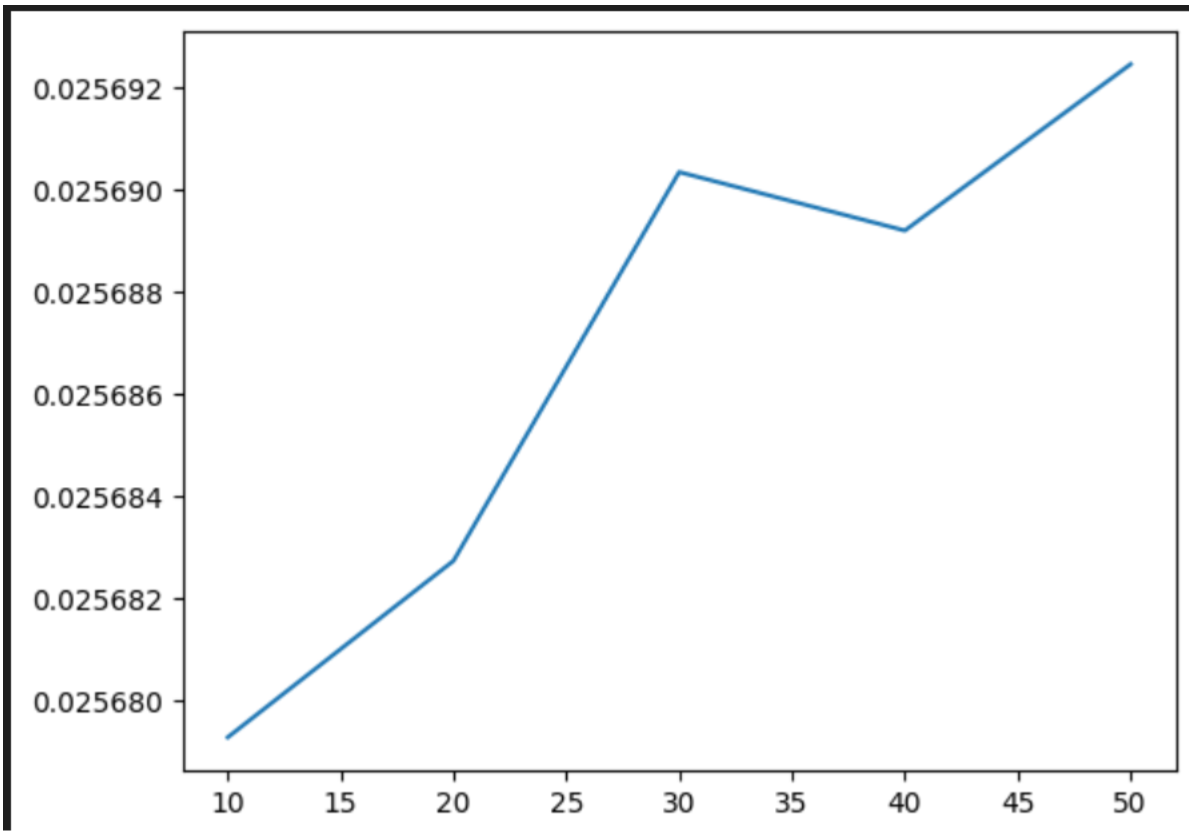
```
        predicted_values = 
user_item_matrix_normalized_10.loc[similar_Users].mean(axis=0)
        row = user_item_matrix_normalized_10.loc[userId]
        for column_name, column_data in row.iteritems():
            user_item_matrix_normalized_10[column_name] = 
predicted_values[column_name]




    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)
```

```python
import matplotlib.pyplot as plt

X_axis = [10,20,30,40,50]

K_10_mean =  sum(K_10) / len(K_10)
K_20_mean =  sum(K_20) / len(K_20)
K_30_mean =  sum(K_30) / len(K_30)
K_40_mean =  sum(K_40) / len(K_40)
K_50_mean =  sum(K_50) / len(K_50)


print(K_10_mean,K_20_mean,K_30_mean,K_40_mean,K_50_mean)
Y_axis = [K_10_mean,K_20_mean,K_30_mean,K_40_mean,K_50_mean]

plt.plot(X_axis,Y_axis)
plt.show()

0.025679276863487387 0.025682735028129823 0.02569035423640741 0.02568920826980034 0.02569246805825754
```



```
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]
```

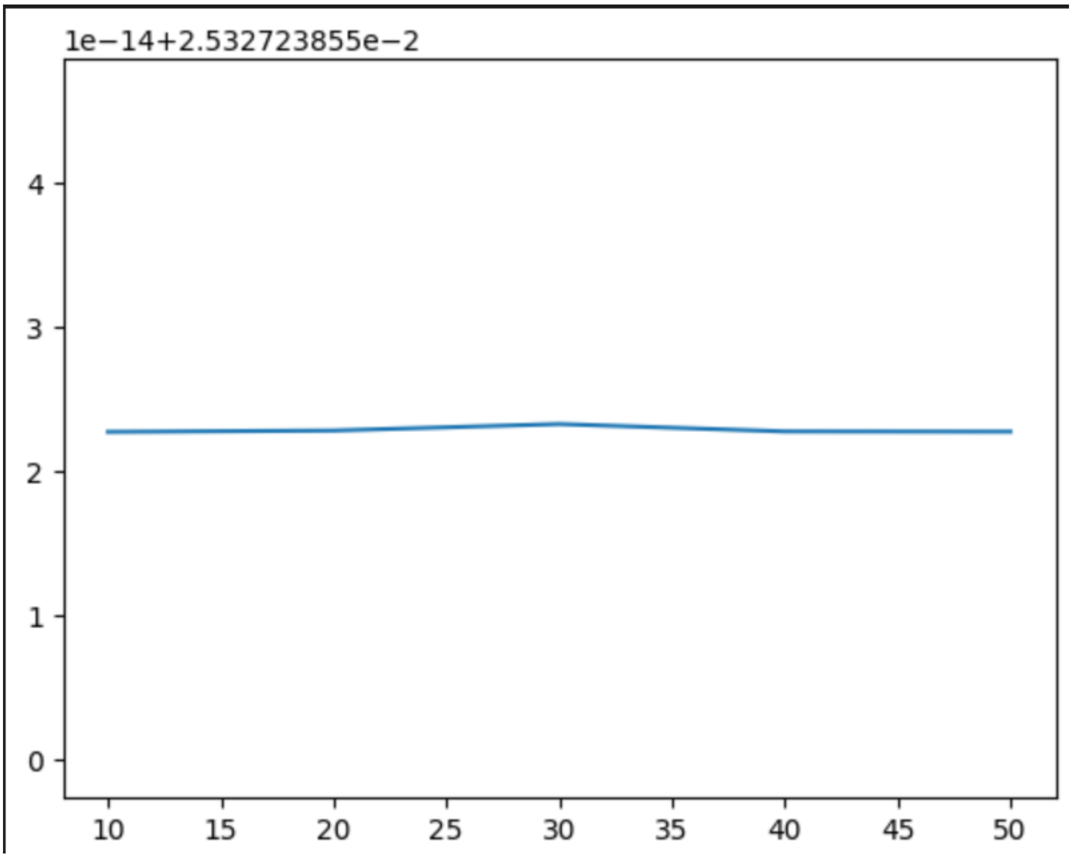
```
    mae_list_20 = []
    for idx, row in item_user_matrix_normalized_20.iterrows():
        print("I am outside")
        print(item_user_matrix_Validation.index)
        print(idx)
        if idx in item_user_matrix_Validation_normalized.index:
            print("I am inside")
            mae = 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)
```

```python
import matplotlib.pyplot as plt

X_axis = [10,20,30,40,50]

K_10_mean =  sum(K_10) / len(K_10)
K_20_mean =  sum(K_20) / len(K_20)
K_30_mean =  sum(K_30) / len(K_30)
K_40_mean =  sum(K_40) / len(K_40)
K_50_mean =  sum(K_50) / len(K_50)


print(K_10_mean,K_20_mean,K_30_mean,K_40_mean,K_50_mean)
Y_axis = [K_10_mean,K_20_mean,K_30_mean,K_40_mean,K_50_mean]

plt.plot(X_axis,Y_axis)
plt.show()
```

```
0.025327238550922733 0.025327238550922827 0.025327238550923274 0.025327238550922764 0.025327238550922754
```



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.

```
# 12.Also, report the TOP 10 products by User Sum Ratings.
# Most positively reviewed
user_rating = {}
for idx, headphone_review in PreProcesses_HeadPhones_df.iterrows():
    r_prod_id = headphone_review["asin"]
    if r_prod_id not in user_rating:
        user_rating[r_prod_id] = [0,0,0,0,0]
    user_rating[r_prod_id][int(headphone_review["overall"]) - 1] += 1
```

```
user_rating_sum = pd.DataFrame()
for prod_id in user_rating:
    ratingFreqList = user_rating[prod_id]
    Rating = ratingFreqList[0] + ratingFreqList[1] + ratingFreqList[2] + ratingFreqList[3] + ratingFreqList[4]
    user_rating_sum.at[0,prod_id] = Rating
```

```python
top_10_users_rating = user_rating_sum.iloc[0].sort_values(ascending=False).head(10).index.tolist()
```

```python
top_10_users_rating
```

```
['B004WODP20',
 'B00BN0N0LW',
 'B00LP6CFEC',
 'B00STP86CW',
 'B007FHX9OK',
 'B00EEHNNNG',
 'B00JJ2C0S0',
 'B000067RC4',
 'B003LPTAYI',
 'B00AWIPITS']
```