**BrainyBeam Technologies**

**Data Science using python Internship Batch-1**

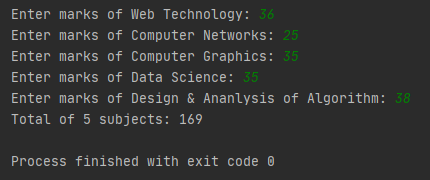
**Task-1**

**Write a python program to take input from the user, enter marks subjects and give total of marks that subjects as output.**

wt = int(input("Enter marks of Web Technology: "))  
cn = int(input("Enter marks of Computer Networks: "))  
cg = int(input("Enter marks of Computer Graphics: "))  
ds = int(input("Enter marks of Data Science: "))  
daa = int(input("Enter marks of Design & Ananlysis of Algorithm: "))  
total = wt + cn + cg + ds + daa

print("Total of 5 subjects:", total)

**Output:**

****

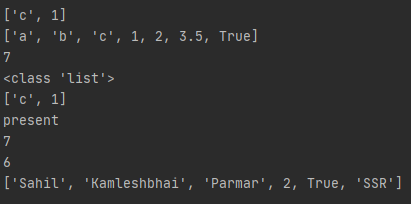
**Task 2**

**List out the methods used commonly in list, set, tuple, dictionary with their rules.**

**List:**

l1=['a', 'b', 'c', 1, 2, 3.5, True]  
print(l1[2:4]) # slicing  
l2=['SSR'] # updating  
print(l1)  
print(len(l1)) # length  
print(type(l1)) # type   
print(l1[2:4])   
  
if 'a' in l1: # membership operator  
 print('present')  
else:  
 print('absent')  
  
l1[0]="Sahil"  
l1[1:3]=['Kamleshbhai', 'Parmar']  
print(len(l1))  
l1[1:3]=['Kamleshbhai']  
print(len(l1))  
l1.insert(2,"Parmar") # insert  
l1.extend(l2) # extend  
l1.remove(3.5) # remove  
l1.pop(3) # pop  
print(l1)

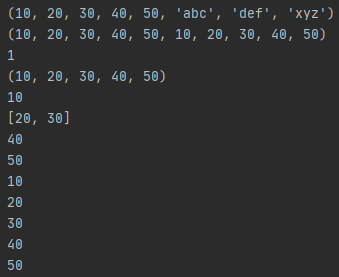
**Output:**

****

**Tuple:**

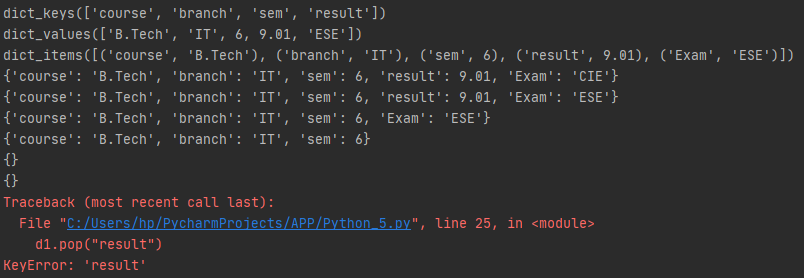
t1=(10, 20, 30, 40, 50)  
t2=('abc', 'def', 'xyz')  
t3=t1+t2 # concatenate  
print(t3)  
print(t1\*2) # \* operator  
print(t1.count(10)) # count  
print(t1)  
  
(a, \*c, b, d) = t1  
print(a)  
print(c)  
print(b)  
print(d)  
  
for x in (t1):  
 print(x)

**Output:**

****

**Dictionary:**

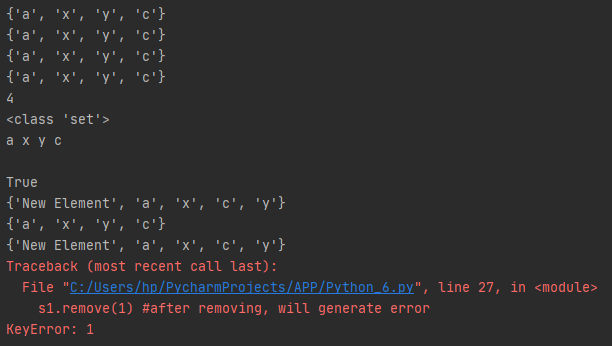
d1={  
 "course": "B.Tech",  
 "branch": "IT",  
 "sem": 6,  
 "result": 9.01  
 }  
print(d1.keys()) #keys  
d1["Exam"]="ESE" #update  
print(d1.values()) #output values key: value  
print(d1.items())#output value   
d1["Exam"]="CIE" #update  
print(d1)  
d1.update({"Exam": "ESE"})  
print(d1)  
del d1["result"] # delete  
print(d1)  
d1.pop("Exam") #pop  
print(d1)  
d1.popitem() #popitem  
d1.clear() #clear  
print(d1)  
d2=d1 #equality  
print(d2)  
d2=d1.copy() #copy  
d1.pop("result")  
print(d1)  
print(d2)  
d3=dict(d1)  
print(d1)  
if "course" in d1: #membership operator  
 print('Yes')  
else:  
 print('No')  
for i, j in d1.items():  
 print(i, j)

****

**Set:**

#Set: Store multiple data in single variable, in {} brackets are unordered, unchanged, not allowed duplocates. Not supoort key: value frm.  
  
s1 = {"a", "b", "c", "d"}  
s2 = {"a", "x", "y", "c"}  
s1.update(s2) #update  
s1.union(s2) #union  
s3=s1.intersection(s2) #intersection  
print(s3)  
s1.intersection\_update(s2) #intersection\_update  
print(s1)  
print(s2)  
print(s1)  
print(len(s1)) #length  
print(type(s1)) #type  
for i in s1: #membership  
 print(i, end=" ")  
print("\n")  
if "c" in s1:  
 print("True")  
else:  
 print("False")  
s1.add("New Element") #add  
print(s1)  
s1.update(s2) #update  
print(s2)  
print(s1)  
s1.remove(1) #after removing, will generate error  
print(s1)  
s1.discard(2) #after removing, will not generate error  
print(s1)  
s1.pop() #pop  
print(s1)   
s1.clear()#clear  
print(s1)  
del s1 #del  
print(s1)  
s3 = s1.symmetric\_difference(s2) #symmetric\_difference  
print(s1)  
print(s2)  
print(s3)  
s1.symmetric\_difference\_update(s2) # symmetric\_difference\_update  
print(s1)  
print(s2)

**Output:**



**Task 3**

1. **Explain any 5 functions of random module**
2. **random()**

first of all import random module to use any function of random module by writing **import random**.

random() function creates any random float value.

import random  
print(random.random()) #random()



1. **randint()**

randint() function creates any random integers within given range. It takes two positional arguments in form of border. Within given number it will create integers.

print(random.randint(1, 10)) #randint()



1. **randrange()**

randrange() function also returns the integer values. Within range user has to pass another argument as in which form it want numbers.

Here, I write (0, 100, 10) means 0 is starting range, 100 is ending range, 10 is gap. Now because 10 is gap every time function returns the integer with gap of 10. As you can see here, output is 30.

print(random.randrange(0, 100, 10)) #randrange()

****

1. **choice()**

choice() function returns the random latter from string and random number from list. It takes one argument as a list or a string.

print(random.choice("Sahil")) #choice()  
print(random.choice([148, 183, 101, 107, 109, 124, 134, 139, 139, 113]))



1. **shuffle()**

shuffle() function shuffles the elements inside the list, either they can be numbers or can be string also. It will take one argument as a list.

numbers = [148, 183, 101, 107, 109, 124, 134, 139, 139, 113]  
random.shuffle(numbers) #shuffle()  
print(numbers)



1. **Build password generator program contains numbers, alphabets, special characters.**

import random

length = int(input("Enter password length: "))  
number = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz@!#$%^&\_-()"  
print("".join(random.sample(number, length)))



1. **Write a note about NLP, NLU, NLG with examples**

* **NLP (Natural Language Processing)**
* How we humans are communicate with each other, by verbal and non-verbal communication okay! Non-verbal communications are the signs of particular things. i.e. blinking of eyes, move around your eyes, hand directions etc. Verbal communication is the communication done by any medium i.e. language. e.g. I ask you, “How are you?”, you replied me, “I am fine!” This is also known as the normal communication.
* We want, robots or our computer talk like this manner in now a days, so that computer/replicate this things and computer-human language processing can be possible.
* **Applications of NLP:**

Speech Recognition e.g. Google Assistant, Siri, Alexa etc. It interprets our speech and replies us in such a manner.

Sentimental Analysis e.g. Twitter (reviews on particular tweet)

Machine Translator e.g. Google Translator

Chatbots e.g. Chatting with machines

Spellchecks

* **How it works?**
* Let’s take an example of speech recognition. You ask or tell something to your robot, it will covert that speech into text.
* Then it will move further to NLU to understand the speech and then move forward to NLG to generate the reply answer.
* **NLU (Natural Language Understand)**
* When user asks or tell something, machine collects that words and understands it.
* What do the user say? Intention of user. Wheather machine is getting your language or not. It is known as ambiguity in machine.
* There are several types of ambiguity:

1. **Lexical Ambiguity:** Speech of user is divides into tokens. e.g. “The tank is full of water.” Here what is challenge for NLU is! What is tank? We use to store water that tank or the tanks which are used in armies. Here confusing of words because of tokenization is known as Lexical Ambiguity.
2. **Synthetic Ambiguity:** The syntax of sentence must be valid! Otherwise it will not give proper reply according what user wants! e.g. Old men and women were taken to the safe place. Here, old men and women means old men and old women is create confusion for NLU. It is easy to understand for humans, not for machines.
3. **Semantic Ambiguity:** What user is actually saying? What is meaning of that speech? e.g. The car hit the poll while it was moving. Here, ambiguity is for NLU is what is moving car or poll? For solving this problem, it is divided into verbs, adjective, nouns, verbal etc.
4. **Pragmatic Ambiguity:** Context of the phrase gives multiple interpretation, means one phrase, multiple meanings. e.g. The police are coming. As a human it is easy to understand why police is coming, for whom police is coming? But the same thing is difficult to understand for the machines.

* **NLG (Natural Language Generator)**
* On the basis of NLU, it will reply in proper manner.
* What should we say to user?
* Reply should intelligent and understandable. Give proper answer of user’s questions.
* Text/sentence planning. Take words from the database and structure the sentences.

1. **Perform text to speech example using gtts**

from gtts import gTTS

commentry = gTTS("""Dhoni finishes off in style! Magnificent strike into the crowd. India lift the world cup after 28 years.

And It's an Indian captain who has been absolutely magnificent in the night of the final.""")

commentry.save("Ravi\_Shastri.mp3")



**Task 4**

1. **List out any 5 methods of numpy and pandas with output.**

* **Numpy**

import numpy as np

s = np.array([1, 2, 3, 4, 5])

s

****

type(s)

****

a = np.array([[1, 2, 3],

[10, 20, 30]])

a

****

np.zeros((2, 3))

****

np.arange(10)

****

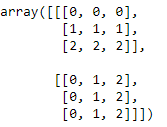
np.arange(2, 3, 0.1)

****

np.linspace(1, 4, 6)

****

np.indices((3, 3))

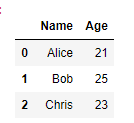
****

* **Pandas**

import pandas as pd

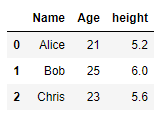
df = pd.DataFrame({"Name": pd.Series(["Alice","Bob", "Chris"]), "Age": pd.Series([21, 25, 23])})

df



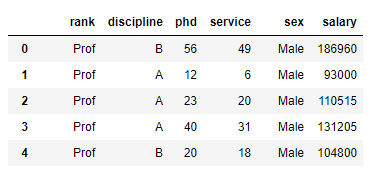
df["height"] = pd.Series([5.2, 6, 5.6])

df



sal = pd.read\_csv(r"C:\Users\hp\Downloads\Salaries.csv")

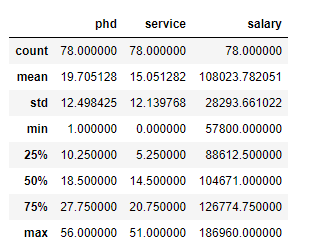
sal.head()



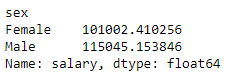
sal.columns



sal.describe()



sal.groupby(["sex"])["salary"].mean()

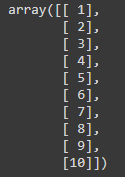


1. **reshape(-1, 1) explanation**

* reshape() function is used to convert an array shape. It converts the shape within the range of number of elements
* reshape() function takes two arguments. First arguments represents row and second argument represents column.
* Now, in reshape(-1, 1) row -1 means it represents all the rows except last one and column 1 means only one column should be there.

a = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])

a.reshape(-1, 1)



1. **Linear regression working with mathematical equation**

import pandas as pd

import numpy as np

from sklearn.linear\_model import LinearRegression

x = np.array([1, 2, 3, 4, 5])

y = np.array([14, 26, 37, 48, 59])

testing = np.array([6, 7, 8])

model = LinearRegression()

model.fit(x.reshape(-1, 1), y)

model.predict(testing.reshape(-1, 1))

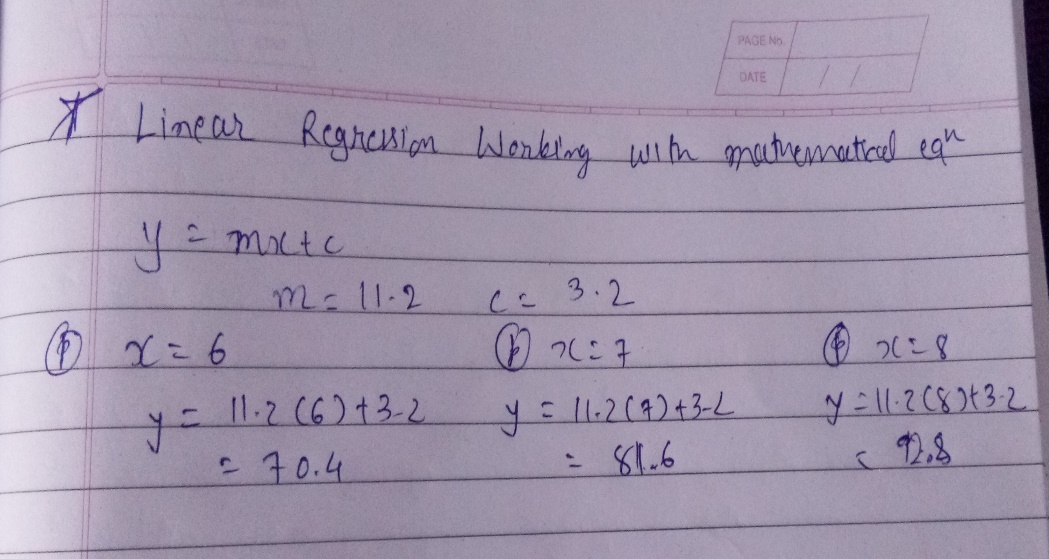
****

model.coef\_

****

model.intercept\_

****



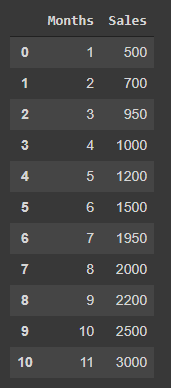
1. **Sales prediction using Linear Regression**

import numpy as np

import pandas as pd

sales = pd.read\_csv("/content/drive/MyDrive/BrainyBeam/Sales.csv")

sales



x = sales["Months"]

y = sales["Sales"]

print(x.shape)

print(y.shape)



x = sales.iloc[:, : -1]

y = sales.iloc[:, 1]

print(x.shape)

print(y.shape)



import sklearn

from sklearn.model\_selection import train\_test\_split

xtrain, xtest, ytrain, ytest = train\_test\_split(x, y, test\_size = 0.7, random\_state = 0)

from sklearn.linear\_model import LinearRegression

lin = LinearRegression()

lin.fit(xtrain, ytrain)

ypred = lin.predict(xtest)

print(ypred)



lin.coef\_



lin.intercept\_



from sklearn.metrics import r2\_score

acc = r2\_score(ypred, ytest)

print(acc\*100)



**Task 5**

1. **Clean the data**

* Import necessary libraries

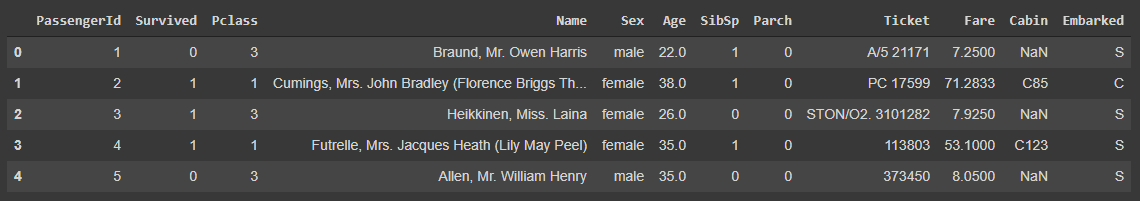
import numpy as np

import pandas as pd

* Read the data

train = pd.read\_csv("/content/drive/MyDrive/BrainyBeam/train.csv")

train.head()

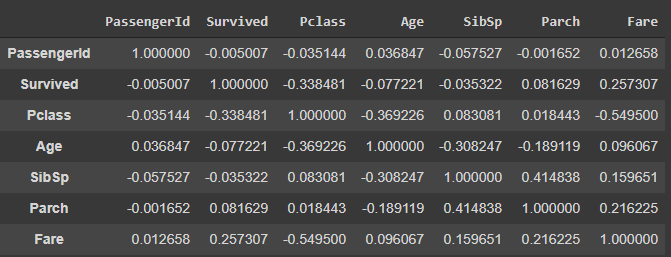


* Summary of data

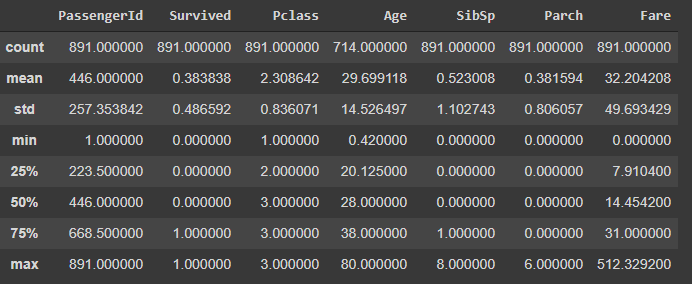
train.shape



train.corr()

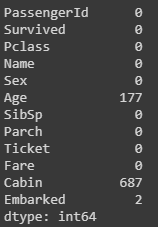


train.describe()



* Null values inside data

train.isnull().sum()



isnull().sum() functions gives the null values in short manner. It will give total null values inside each columns.

* Drop a column contains high no. of null values

drop\_col = train.isnull().sum()[train.isnull().sum()>(35/100 \* train.shape[0])]

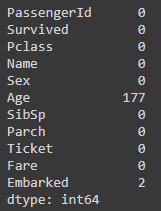
drop\_col

Here, if null value ratio of any column is greater than 35%, it will be removed. User can give any number for removing columns.



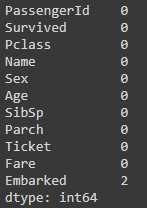
train.drop(drop\_col.index, axis = 1, inplace = True)

train.isnull().sum()



train.fillna(train.mean(), inplace = True)

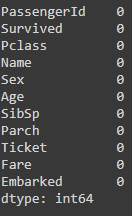
train.isnull().sum()



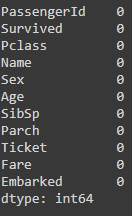
* Fill less no. of values by giving certain value.

train["Embarked"].fillna("S", inplace = True)

train.isnull().sum()



* Data after cleaning



1. **Decision Tree Explanation**

* Decision Tree is classification algorithm. It is form of Unsupervised Learning. It is also used in Regression.
* It is mainly binary classifier i.e. yes/no, 0/1, high/low
* Based on trained dataset algorithm generates a model for classification, it is in tree structure that’s why it is known as decision tree.
* In model, user has to give an input, after input, algorithm learns itself and predicts this input is belongs to which class.
* There are two nodes in decision tree: 1. Decision Node 2. Leaf Node
* 1. Decision Node: From where, new branches are begun is known as Decision Node.
* 2. Leaf Node: From where, new branches are not begun is known as Leaf Node.
* Based on root node decision, data has been split. This process is known as splitting action.
* Leaf node is classification label.

1. **Random State Explanation**

* Random state is parameter to fix the training data. I can be any positive integer value. User can any value as random state of model.
* If we are not providing any random state value, model will take any of indexes for training and sends remaining for testing. It can be affect accuracy of model.
* For example, I have 30 entries in my data. I have to send 20 rows for training every time. For that I used random state, after that, it will send that 20 rows only for training every time.
* If I will not provide any random state, it will take any 20 rows by itself for training and remaining will for testing. Just like:

1st time training: 1-20 rows

2nd time training: 5-25 rows

3rd time training: 7-27 rows and so on…

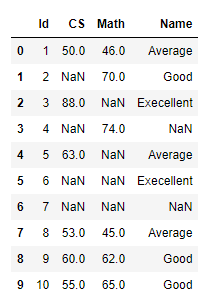
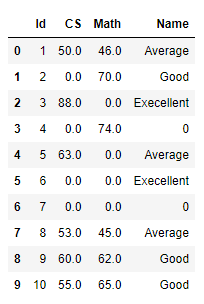
**Task 6**

1. **Explain 5 parameters used in decision tree model.**

* **max\_depth**
* max\_depth is determined depthness of decision tree.
* It can be any integer or float value.
* Default depth is 2.
* **min\_samples\_split**
* Minimum number of samples required to split an internal node.
* It can be any integer or float value.
* Its default value is 2.
* **max\_leaf\_nodes**
* Grow a tree with max\_leaf\_nodes in best first fashion. Best nodes are defined as relative reduction in impurity. If none then unlimited number of leaf nodes.
* It can be any integer value.
* Its default value is None.
* **min\_samples\_leaf**
* The minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least min\_samples\_leaf training samples in each of the left and right branches. This may have the effect of smoothing the model, especially in regression.
* It can be any integer or float value.
* Its default value is 2.
* **criterion**
* The function to measure the quality of a split. Supported criteria are “gini” for the Gini impurity and “entropy” for the information gain.
* It can be either “gini” or “entropy”.
* **splitter**
* The strategy used to choose the split at each node. Supported strategies are “best” to choose the best split and “random” to choose the best random split.
* It is in form of string value It can be either best or random.

1. **Mention Data cleaning methods and its working**

* In datasets, we generally find some missing values which are known as null values.
* It displayed in dataset as **NaN**.
* We can either drop those values or can fill those values by some appropriate methods supported by pandas.
* This process is known as Data Cleaning.
* **Some Data cleaning methods**
* **fillna() mtethod**
* This method is used to fill NaN values by filling values by choices.
* Syntax: df.fillna(value)
* df.fillna(0)

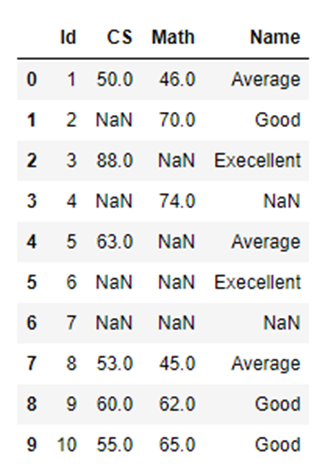
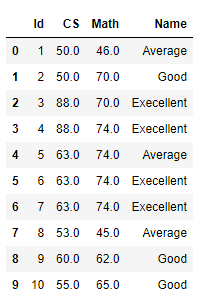
 

* **ffill and bfill method**
* These methods are used in fillna() method as a parameter.
* ffill fills the value in forward manner, whereas bfill fills the value in backward manner
* Syntax: df.fillna(method = “ffill”), for forward manner

df.fillna(method = “bfill”), for forward manner

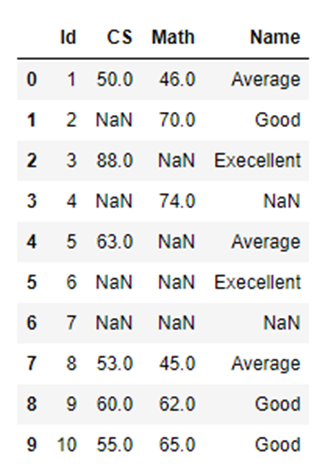
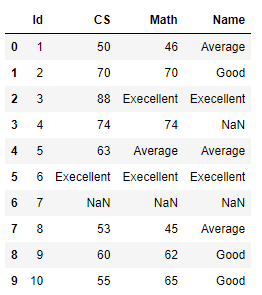
* new\_df1 = df.fillna(method = "ffill")

new\_df1

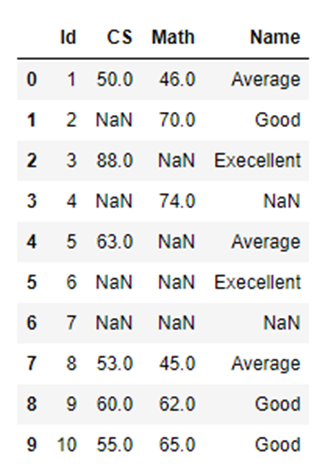
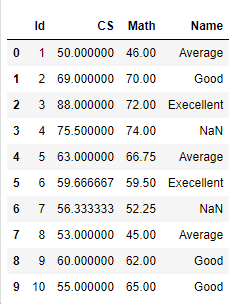
* new\_df = df.fillna(method = "bfill", axis = "columns") # index = row, axis = column

new\_df

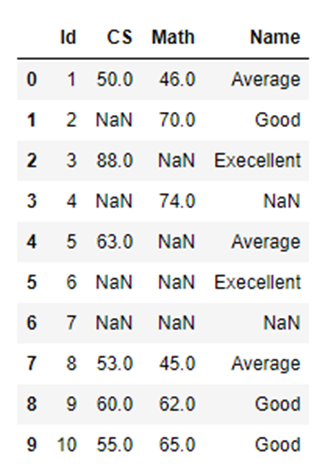
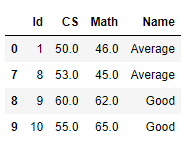
* **interpolate() method**
* It fills the values by converting into float.
* Syntax: df.interpolate()
* newdf\_interpolate = df.interpolate()

newdf\_interpolate

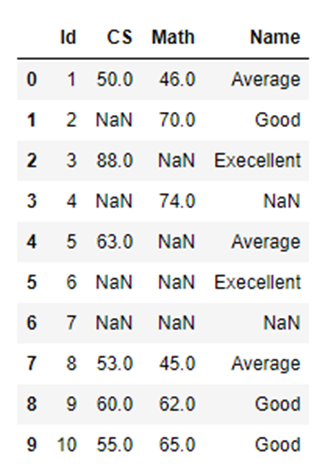
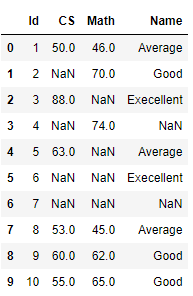
* **dropna() method**
* dropna() method used to remove the values conists of NaN values.
* It removes the rows by default to removes the values.
* Syntax: df.dropna()
* dropped\_df = df.dropna()

dropped\_df

 ****

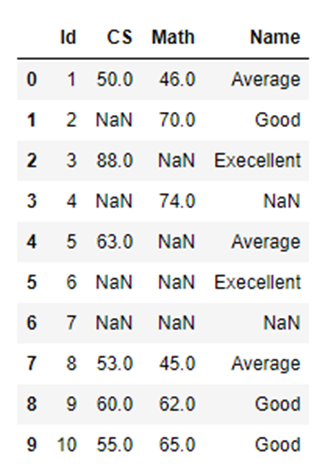
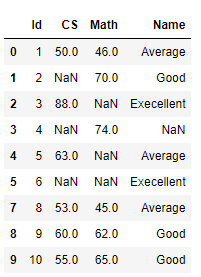
* **how method**
* how method is used in dropna() method as parameter. In the parameter according to user requirement, values has been removed by function.
* Syntax: df.dropna(how = “method\_name/pattern”)
* drop\_all = df.dropna(how = "all")

drop\_all

 ****

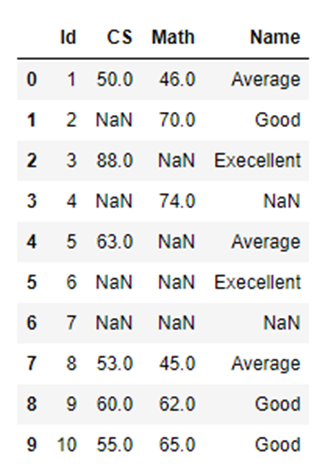
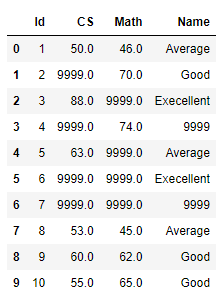
* **threshold method**
* thresh method removes the rows which have value according to thresh value.
* Syntax: df.dropna(thresh = any integer)
* thresh\_df = df.dropna(thresh = 2)

thresh\_df

* **replace() method**
* replace() method replaces the NaN value by given value into specific position.
* Syntax: df.replace([“old\_value”], new\_value)
* replaced\_df = df.replace(np.NaN, 9999)

replaced\_df

* changed\_df = df.replace(

np.NaN,

{

"CS": 99,

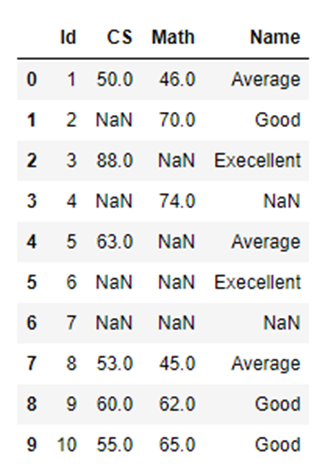
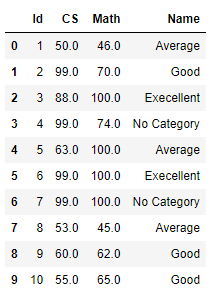
"Math": 100,

"Name": "No Category"

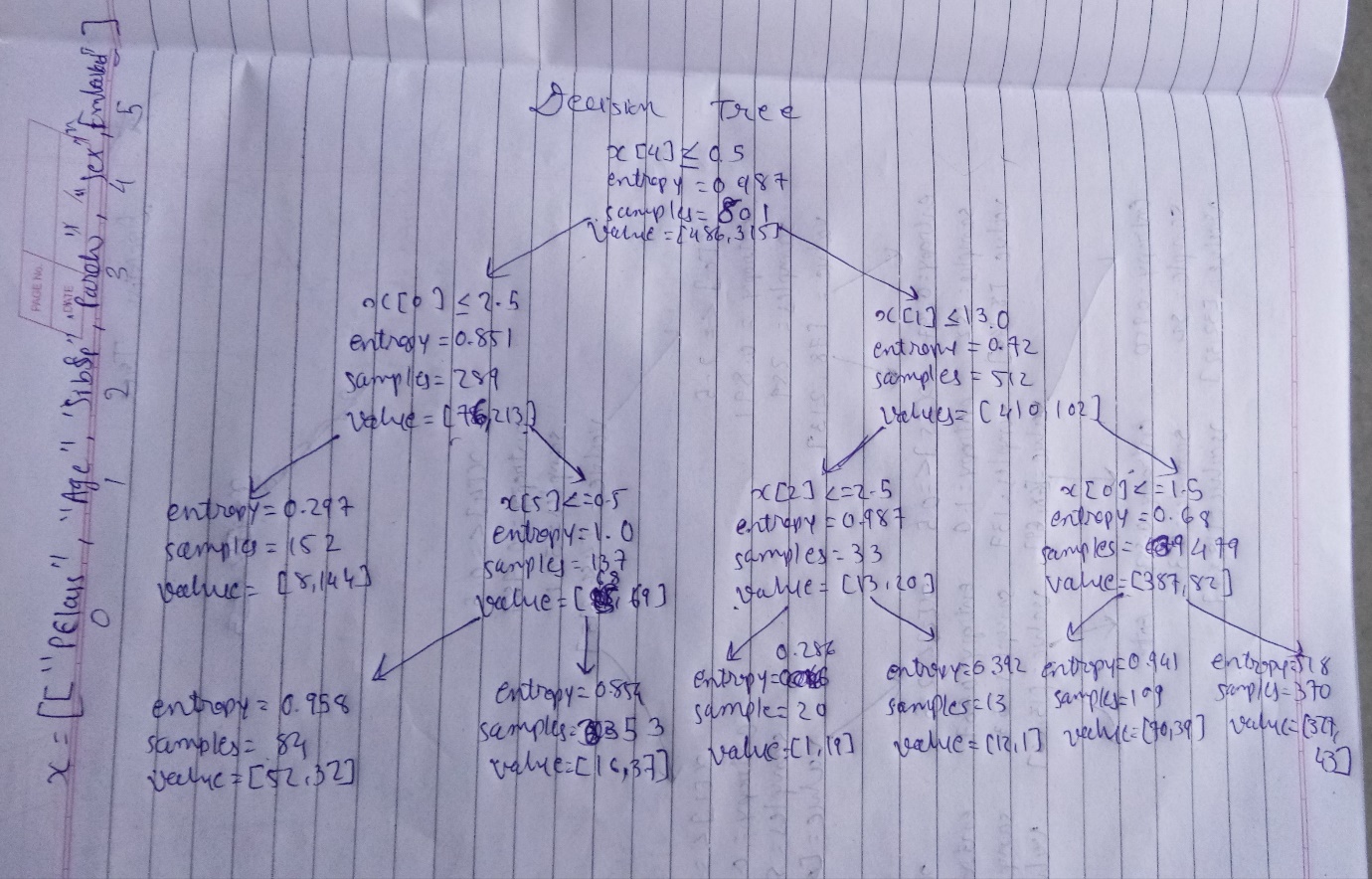
}

)

changed\_df

1. **Draw the Decision Tree**



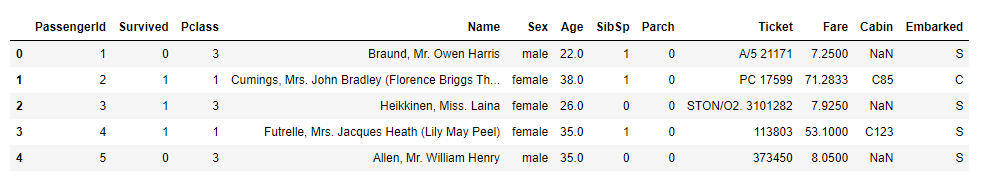
1. **Reach 90% accuracy**

import numpy as np

import pandas as pd

train = pd.read\_csv(r"C:\Users\hp\Downloads\train.csv")

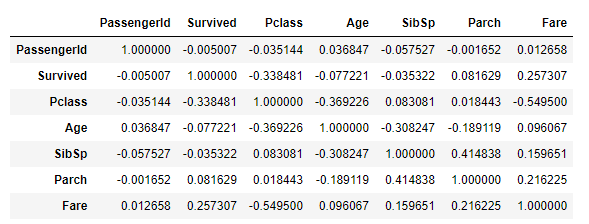
train.head()



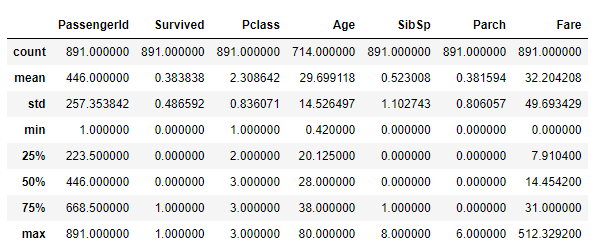
train.shape



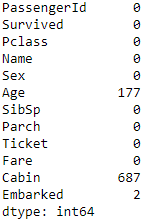
train.corr()



train.describe()

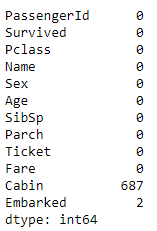


train.isnull().sum()



train.fillna(train.mean(), inplace = True)

train.isnull().sum()

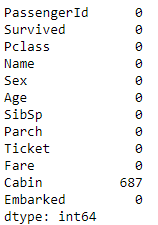


train['Embarked'].fillna('S', inplace = True)

train['Embarked'].unique()



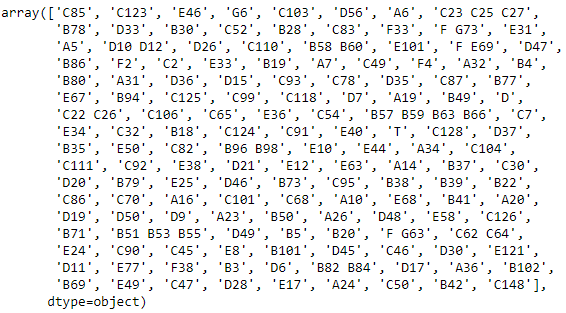
train.isnull().sum()



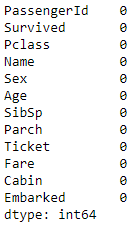
train['Cabin'].fillna(method = "ffill", inplace = True)

train['Cabin'].replace(np.NaN, 'C85', inplace = True)

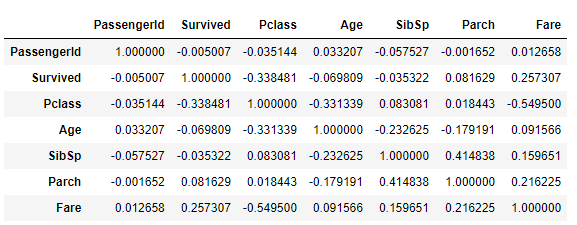
train['Cabin'].unique()



train.isnull().sum()



train.corr()



from sklearn import preprocessing

label\_encoding = preprocessing.LabelEncoder()

train["Sex"] = label\_encoding.fit\_transform(train["Sex"])

train["Sex"].unique()



train["Embarked"].fillna("S", inplace = True)

train["Embarked"] = train["Embarked"].replace(["S"], 0)

train["Embarked"] = train["Embarked"].replace(["C"], 1)

train["Embarked"] = train["Embarked"].replace(["Q"], 2)

train["Embarked"].unique()



from sklearn.tree import DecisionTreeClassifier

feature = train[["Pclass", "Age", "SibSp", "Parch", "Survived", "Sex", "Embarked"]]

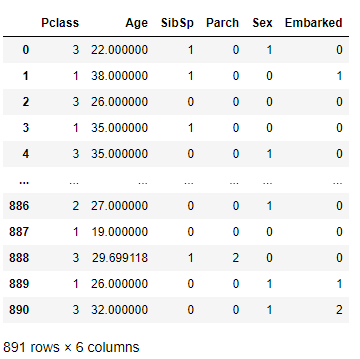
feature = feature.dropna()

feature.shape



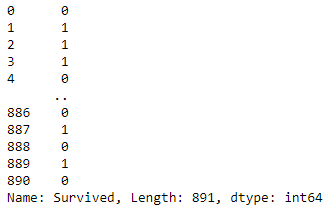
x = feature.drop("Survived", axis = 1)

x



y = feature["Survived"]

y



from sklearn.model\_selection import train\_test\_split

xtrain, xtest, ytrain, ytest = train\_test\_split(x, y, test\_size = 0.1, random\_state = 45) # random\_state = training and testing data should be same

model = DecisionTreeClassifier(max\_depth = 4, min\_samples\_leaf = 7, max\_leaf\_nodes = 7, min\_samples\_split = 2,

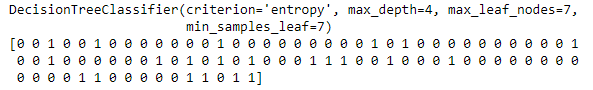
criterion = "entropy", splitter = "best")

model.fit(xtrain, ytrain)

print(model)

ypred = model.predict(xtest)

print(ypred)



from sklearn import metrics

accuracy = metrics.accuracy\_score(ytest, ypred)

print(accuracy\*100)



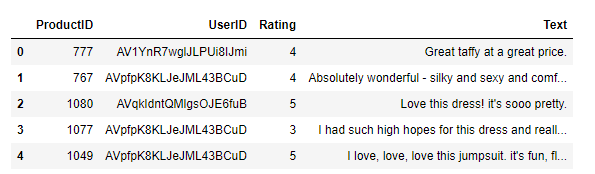
**Task 7**

1. **Find polarity of unique products using apply with function.**

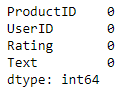
* import pandas as pd

rating = pd.read\_csv(r"C:\Users\hp\Downloads\cloths-rating.csv")

rating.head()



rating.isnull().sum()



import nltk

from nltk.tokenize import word\_tokenize

from nltk.tokenize import sent\_tokenize

from textblob import TextBlob

from nltk.corpus import stopwords

stop\_words = set(stopwords.words())

def polarity(text):

blob = TextBlob(text)

words = blob.words

sentence = [word for word in words if word not in stop\_words]

sent = " ".join(sentence)

blob2 = TextBlob(sent)

return blob2.sentiment.polarity

apply = {"Rating": "mean", "Text": lambda x: ".".join(x)}

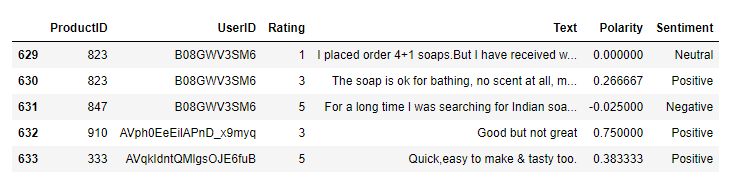
rating["Polarity"] = rating["Text"].apply(polarity)

rating.loc[rating["Polarity"] > 0, "Sentiment"] = "Positive"

rating.loc[rating["Polarity"] < 0, "Sentiment"] = "Negative"

rating.loc[rating["Polarity"] == 0, "Sentiment"] = "Neutral"

rating.tail()



1. **Use frequency distribution and pos tag from nltk.**

* import nltk

from textblob import TextBlob

from collections import Counter

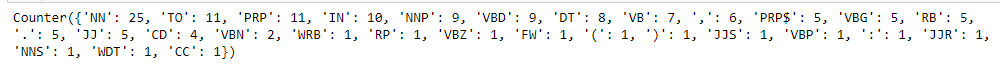
text = """On 16 November 2013, my cricketing journey finally came to an end at the Wankhede Stadium. After somehow managing to complete my farewell speech, I was having a conversation with my family, trying to soak in every moment, when my team-mate Virat Kohli walked up to me. He said, ‘Paaji aapne kaha tha aap ko yaad dilane ke liye ki aapko pitch pe jana hain.’ (You asked me to remind you that you had to go to the pitch one final time.) To be honest, I hadn’t forgotten; I was just trying to put the moment off for a little longer. It was to be my final visit to the 22 yards that had nurtured and cared for me for so long."""

tokens = nltk.word\_tokenize(text)

tags = nltk.pos\_tag(tokens)

counts = Counter(tag for word, tag in tags)

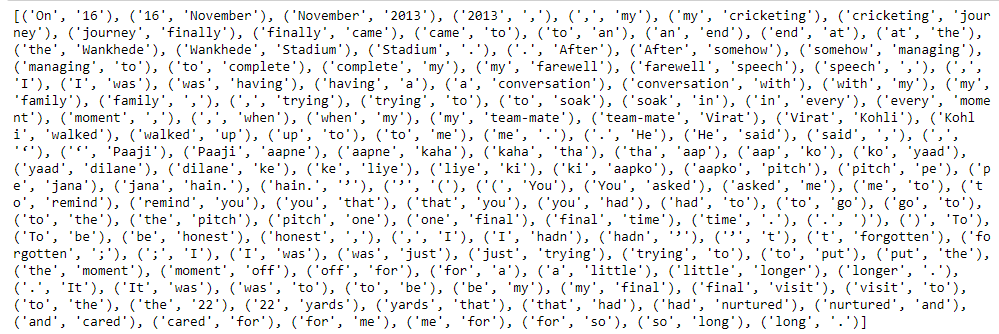
print(counts)



* tokens = nltk.word\_tokenize(text)

output = list(nltk.bigrams(tokens))

print(output)



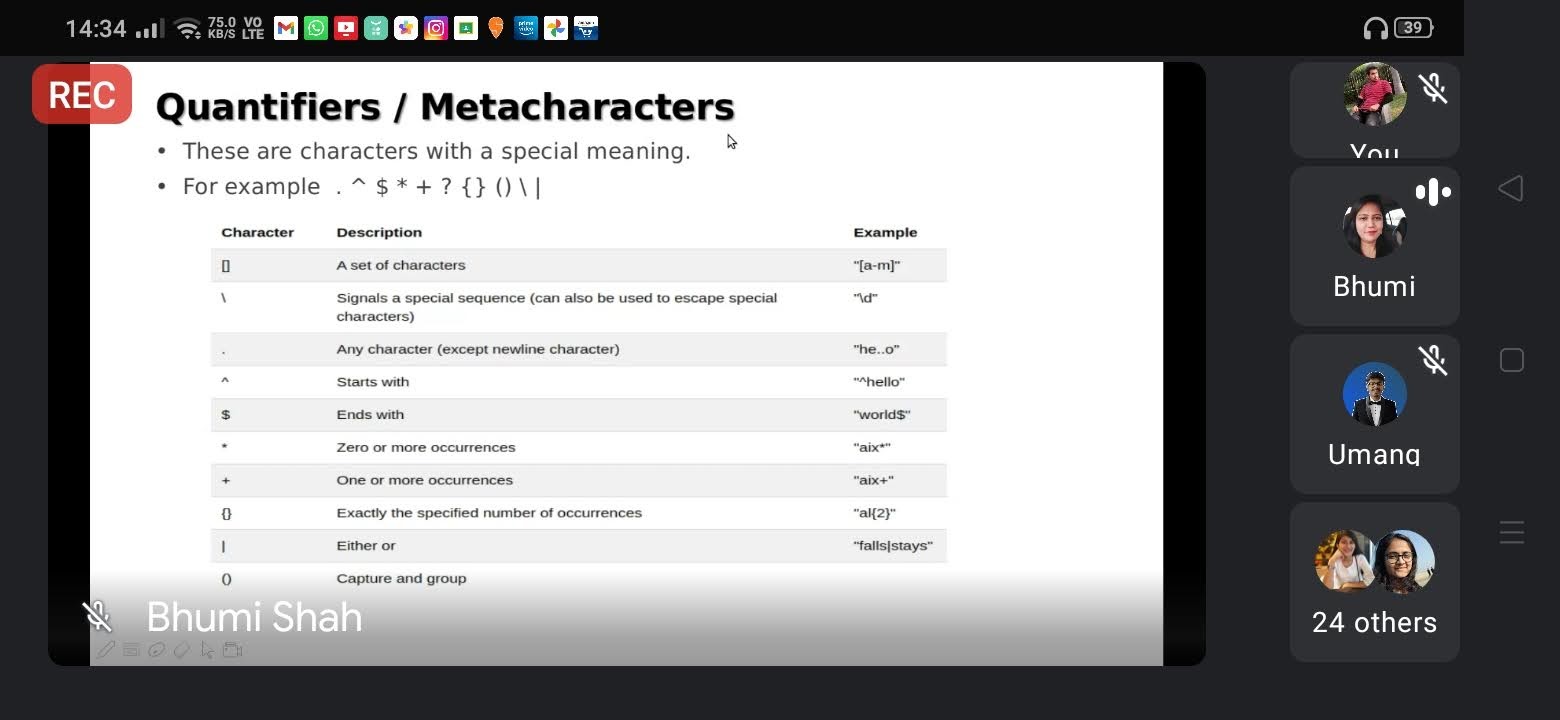
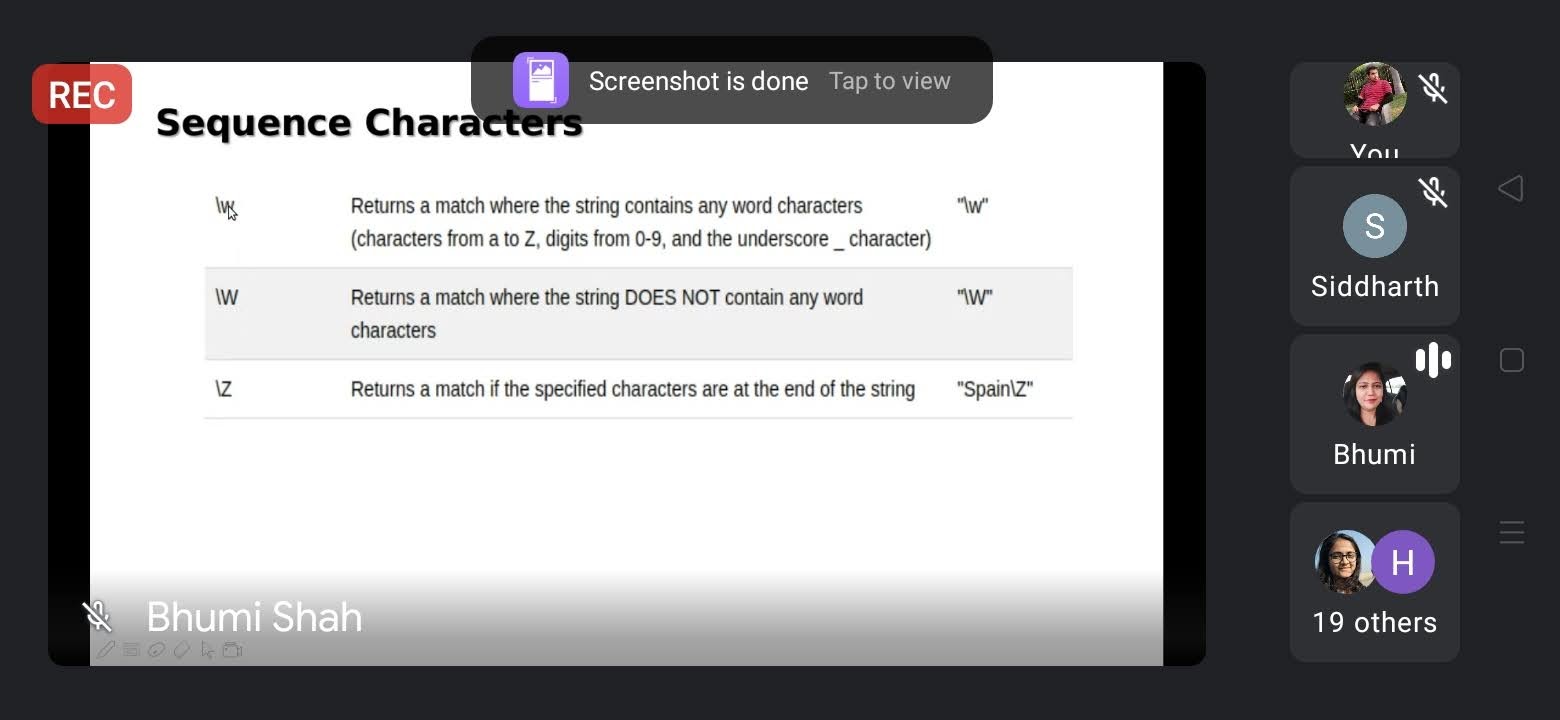
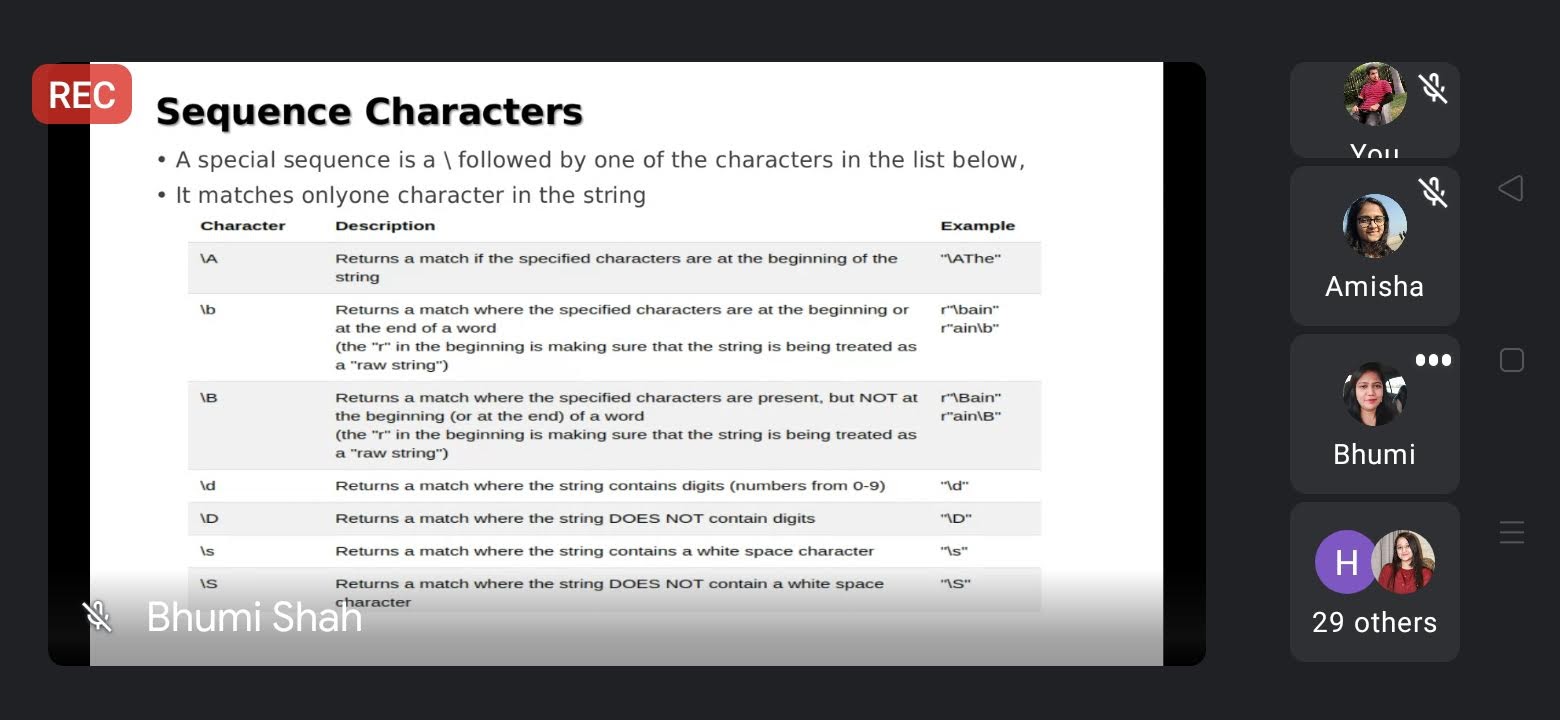
* tokens = nltk.word\_tokenize(text)

output = list(nltk.trigrams(tokens))

print(output)



1. **List out meta-characters in regular expressions. Validate email and mobile number using meta-characters of regex.**

****

* import re

mail = input("Enter mail: ")

pattern = "\w+@.\w+"

r = re.match(pattern, mail)

if r:

print(“valid email”)

else:

print(invalid email)



* country\_code = input("Enter county code: ")

mobile = input("Enter mobile no: ")

pattern1 = "\+91?"

pattern2 = "\d{10}"

r1 = re.match(pattern1, country\_code)

r2 = re.match(pattern2, mobile)

if r1:

print("Yes, it is valid for India.")

else:

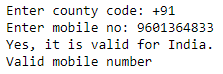
print("No, it is not valid for India.")

if r2:

print("Valid mobile number")

else:

print("Invalid mobile number")



**Task 8**

1. **Explain TF-IDF with example**

* **TF:** The number of times the word has been represented in a particular document.

TF = Number of times term appears in a document / Total number of terms in the document.

* **IDF:** Effect of word occurring in all the documents.

IDF = log\_e(Total number of documents / number of documents with term in it)

* TF-IDF is used to weigh a keyword in any content and assign importance to that keyword based on the number of times it appears in the document.
* For a term **t** in document **d**, the weight **Wt, d** of term t in document d is given by:

Wt,d = Tft,d log(N/DFt)

Where, Tft, d is the number of occurences of t in document.

Dft is is the number of documents containing the term t.

N is the total number of documents in the corpus

* from sklearn.feature\_extraction.text import TfidfVectorizer

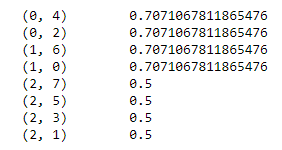
sentences = ["Sahil Parmar", "BrainyBeam Technologies", "Python with Data Science"]

vector = TfidfVectorizer()

vector.fit(sentences)

transform = vector.transform(sentences)

print(transform)



print(vector.vocabulary\_)



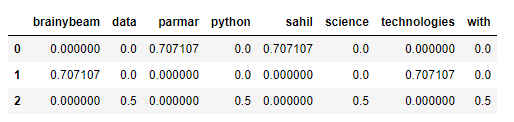
transform.shape



import pandas as pd

df=pd.DataFrame(vector.fit\_transform(sentences).toarray(),columns=vector.get\_feature\_names())

df



**Task 9**

**Explain three techniques of stemming**

* **Stemming:** Stemming is a technique used to extract the base from the words by removing affixes from them
* e.g. eating, eats, eaten is eat
  + **PorterStemmer**
* from nltk.stem import PorterStemmer

word = PorterStemmer()

word.stem("writing")



* + **Lancaster Stemming**
* from nltk.stem import LancasterStemmer

lancaster = LancasterStemmer()

print(lancaster.stem("reading"))



* + **SnoeballStemmer**
* from nltk.stem import SnowballStemmer

snow = SnowballStemmer("english")

print(snow.stem("editing"))



**Task 10**

1. **Explain collaborative and content based filtering with example.**

* **Collaborative Filtering:** Collaborative filtering encompasses techniques for matching people with similar interests and making recommendations on this basis.
* It doesn’t need of features of the item to be given. Every user and item is descried by a feature vector of embedding.
* It considers other user’s reaction while recommending a particular user. It notes which items a particular like and also the items that the users with behaviour and likings like him/her likes, so recommend items to that user.
* It collects user feedback on different items and uses them for recommendation.
* **Memory based collaborative filtering:** Done mainly remembering the user item interaction matrix, and how a user reacts to it, i.e. the rating that a user gives to an item. There is no dimensionally reduction or model fitting as such.
* It has two types: (1) User-User Filtering (2) Item-Item Filtering
* User-based, which measures the similarity between target users and other users.
* Item-based, which measures the similarity between the items that target users rate or interact with and other items.
* For example, likes and dislikes done by user.
* **Content based Filtering:** It tries to guess the features or behavior of a user given the item’s features, he/she reacts positively to.
* Friend’s recommendation on Social Networking Sites

1. **Explain cosine similarity with equation.**

* It is a metric, of helpful in determining, how similar data objects are irrespective of their size. We can measure the similarity between two sentences using cosine similarity.
* In cosine similarity, data objects in a dataset are treated as a vector.
* Cosine similarity = x . y/ ||x|| \* ||y||
* Where, x . y is dot product of x and y

||x|| and ||y|| is length x and y

||x||\*||y|| is cross product of the two vectors x and y

* x = {3, 2, 0, 5} y = {1, 0, 0, 0}

x . y = 3

||x|| = square root of ((x1)2 + (x2)2  + (x3)2 + (x4)2) = 6.16

||y|| = square root of ((y1)2 + (y2)2  + (y3)2 + (y4)2) = 1

* ||x|| \* ||y|| = 6.16 \*1 = 6.16
* cos(x, y) = 3/6.16 = 0.49

1. **Explain rmse and mse with mathematical equation.**

* **RMSE (Root Mean Squared Error):** It is square root of the mean of the square of all of the error.
* RMSE = square root((sum of (prediction - observation)2) / total number of observation)
* **MSE (Mean Squared Error):** It is sum of square of measured and predicted values. It finds average error.
* sum((predicted – actual value)2) / total values

1. **Make list of parameters used in KNN**

* **n\_neighbours:** It is use for kneighbor queries.
* It can be any integer value.
* It’s default value is 5.

* **weights:** This is the function used in prediction of values.
* Possible values for weights are “uniform”, “distance” or “callable”
* It’s default value is uniform.
* In **uniform** weights, all points in each neighborhood are weighted equally.
* In **distance** weights, points is count by inverse of their distance. In this case, closer neighbors of a query point will have a greater influence than neighbors which are further away.
* **[callable]** is user defined function which accepts an array of distances and returns an array of the same shape containing the weights.
* **algorithm:** Algorithm is used to compute the nearest neighbor.
* Possible values of algorithms are: “ball\_tree”, “kd\_tree”, “brute”
* It’s default value is “auto”.
* **metric:** The distance metric to use for the tree.
* It can be string or callable method. It’s default value id “minkowski”, and with p = 2 is equivalent to the standard Euclidean metric.
* **leaf\_size:** Leaf size passed to BallTree or KDTree. This can affect the speed of the construction and query, as well as the memory required to store the tree. The optimal value depends on the nature of the problem.

**Task 11**

**Perform Recommendation based on rating on different dataset.**

* # Import necessary libraries

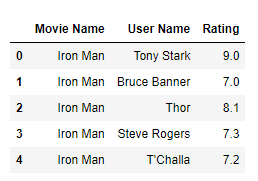
import pandas as pd

import numpy as np

# load datset

mcu = pd.read\_csv(r"C:\Users\hp\Documents\Marvel Cinematic Universe.csv")

mcu.head()

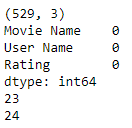


print(mcu.shape) # shape of data

print(mcu.isnull().sum()) # total null values inside data

print(mcu["Movie Name"].nunique()) # number of unique movie name

print(mcu["User Name"].nunique()) # number of unique user name

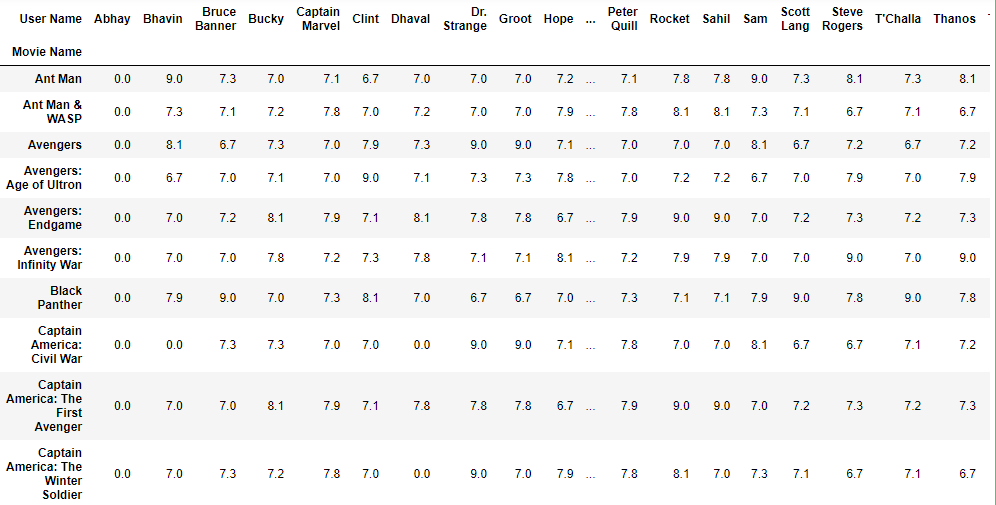


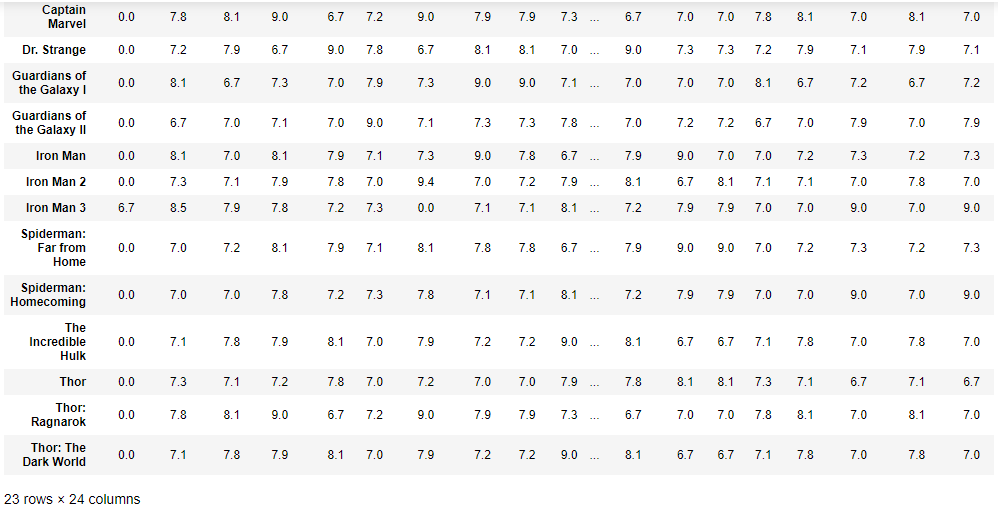
# pivot table of data

# fill NaN values by 0

mcu\_pivot = pd.pivot\_table(mcu, index = "Movie Name", columns = "User Name", values = "Rating").fillna(0)

mcu\_pivot

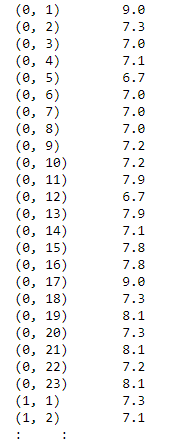


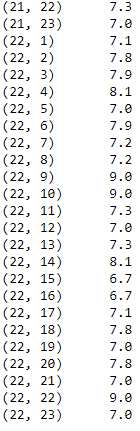


from scipy.sparse import csr\_matrix # import csr matrix

mcu\_pivot\_matrix = csr\_matrix(mcu\_pivot.values)

print(mcu\_pivot\_matrix) # printing csr matrix first value is movie index, second value is user index, third value is rating given by user





# create a model using knn

from sklearn.neighbors import NearestNeighbors

model\_knn = NearestNeighbors(metric = "cosine", n\_neighbors = 20, radius = 1)

model\_knn.fit(mcu\_pivot\_matrix)

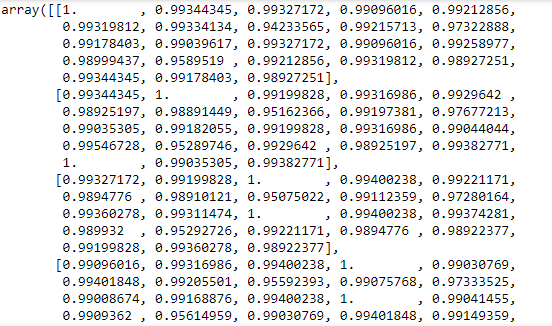


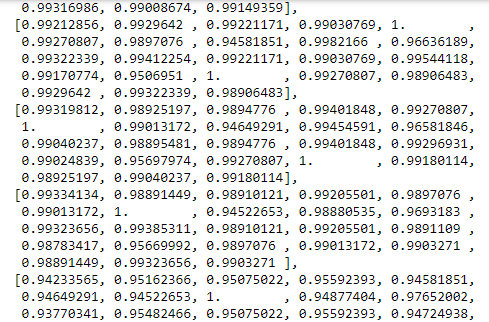
# apply cosine similarity for determining distance between one movie rating to other movies

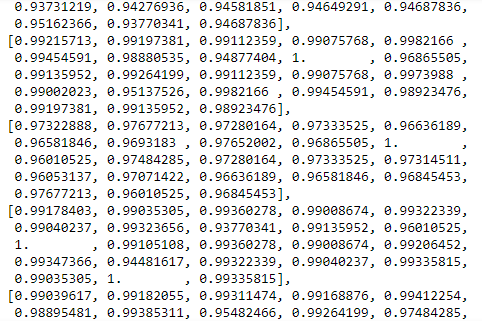
from sklearn.metrics.pairwise import cosine\_similarity

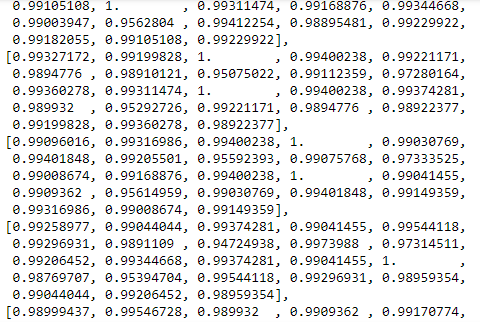
similarity\_matrix = cosine\_similarity(mcu\_pivot)

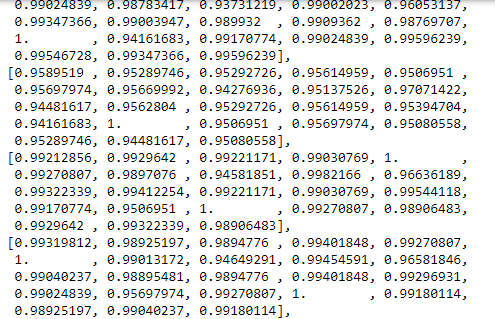
similarity\_matrix

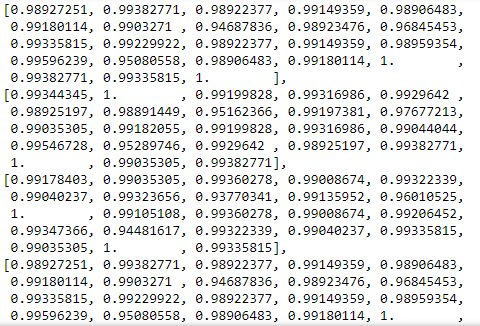










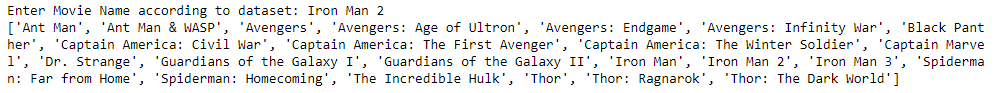




movie\_name = input("Enter Movie Name according to dataset: ") # enter a movie name inside dataset

data = list(mcu\_pivot.index) # According to index most similar movie will suggest by model based on ratings

print(data) # print recommended movies



query\_index = data.index(movie\_name) # index of movie name entered by user

print(query\_index)



similarity, indices = model\_knn.kneighbors(mcu\_pivot.iloc[query\_index].values.reshape(1, -1), n\_neighbors = 9) #Distance with n\_nieghbors of movie entered by a user

print(similarity)

print(indices)



# print recommended movie name by model with similarity distnace

data\_dict = {}

for i in range(0, len(similarity.flatten())):

if i == 0:

print("Recommendation for {} is: \n".format(mcu\_pivot.index[query\_index]))

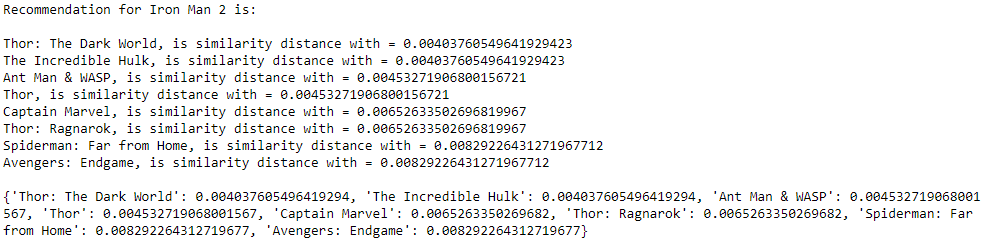
else:

data\_dict[str(mcu\_pivot.index[indices.flatten()[i]])] = float(similarity.flatten()[i])

print(f"{mcu\_pivot.index[indices.flatten()[i]]}, is similarity distance with = {similarity.flatten()[i]:.20f}")

print()

print(data\_dict)



**Task 12**

**Make a dictionary and keep values in range from -5 to 5. Enter the input from user. After entering the number program shows the key from that dictionary.**

* class\_dict = {"1": [-5.0, -4.75, -4.5, 4.25, -4, -3.75, -3.5, -3.25],

"2": [-3, -2.75, -2.5, -2.25, -2, -1.75, -1.5, -1.25],

"3": [-1, -0.75, -0.5, -0.25, 0, 0.25, 0.5, 0.75],

"4":[1, 1.25, 1.5, 1.75, 2, 2.25, 2.5, 2.75],

"5": [3, 3.25, 3.5, 3.75, 4, 4.25, 4.5, 4.75, 5]}

def fun(val):

for i in class\_dict:

for j in class\_dict[i]:

if val == j:

return i

if val <= j:

return i

fun(-5)



fun(-4.76)



**Task 13**

**Apply above logic in your working project file.**

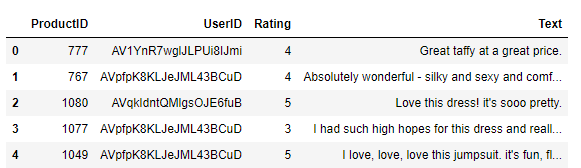
* #import necessary libraries and load the dataset

import pandas as pd

import numpy as np

rating = pd.read\_csv(r"C:\Users\hp\Downloads\cloths-rating.csv")

rating.head()

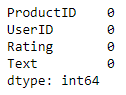


print(rating["UserID"].nunique())

print(rating["ProductID"].nunique())



rating.isnull().sum()



# give sentiment score to text based on text and create new column for sentiment score in dataset

from textblob import TextBlob

def sentiment\_score(text):

try:

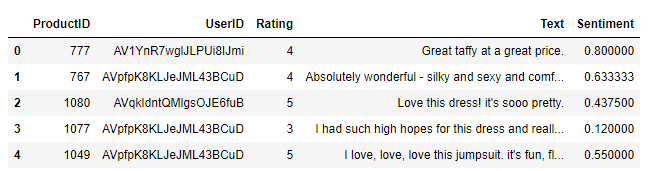
return TextBlob(str(text)).sentiment.polarity

except:

return None

rating["Sentiment"] = rating["Text"].apply(sentiment\_score)

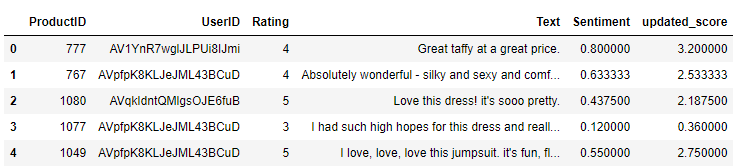
rating.head()



# apply multiplication between ratings and sentiment

rating["updated\_score"] = rating["Rating"]\*rating["Sentiment"]

rating.head()



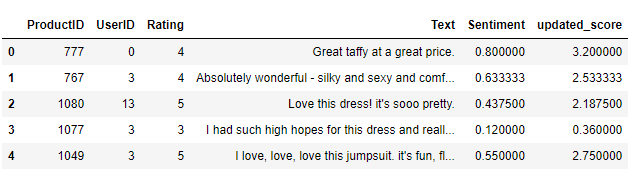
# make userid into normal form

from sklearn import preprocessing

le = preprocessing.LabelEncoder()

rating["UserID"] = le.fit\_transform(rating["UserID"])

rating.head()



# make function to classify updated score

class\_dict = {"1": [-5.0, -4.75, -4.5, 4.25, -4, -3.75, -3.5, -3.25],

"2": [-3, -2.75, -2.5, -2.25, -2, -1.75, -1.5, -1.25],

"3": [-1, -0.75, -0.5, -0.25, 0, 0.25, 0.5, 0.75],

"4":[1, 1.25, 1.5, 1.75, 2, 2.25, 2.5, 2.75],

"5": [3, 3.25, 3.5, 3.75, 4, 4.25, 4.5, 4.75, 5]}

def fun(val):

for i in class\_dict:

if val >= class\_dict[i][0] and val <= class\_dict[i][-1]:

return int(i)

print(fun(4))

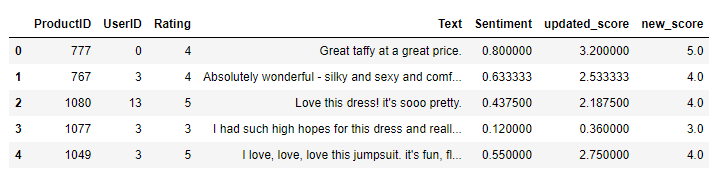


# apply function to updated score and create new column new score

rating["new\_score"] = rating["updated\_score"].apply(fun)

rating["new\_score"] = pd.to\_numeric(rating["new\_score"])

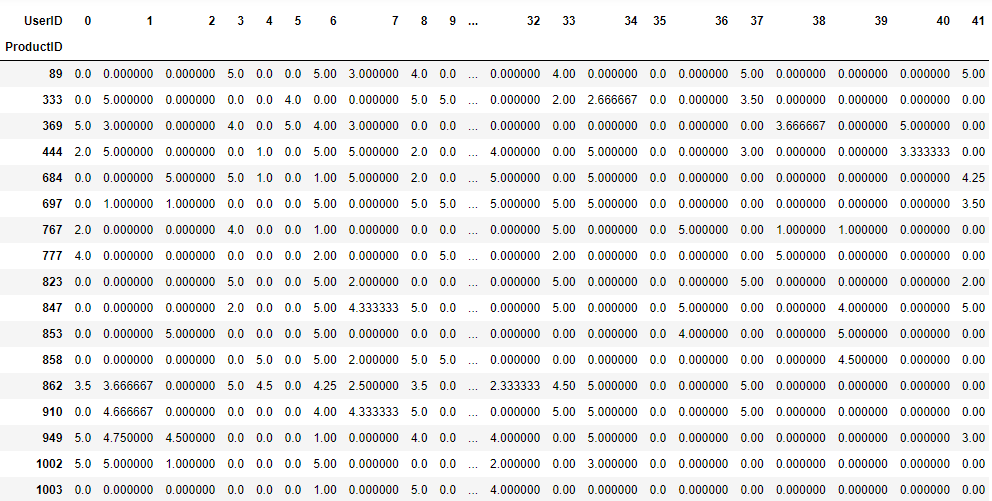
rating.head()



# Pivot table of ProductID, UserID and New\_score

rating\_pivot = pd.pivot\_table(rating, index = "ProductID", columns = "UserID", values = "Rating").fillna(0)

rating\_pivot



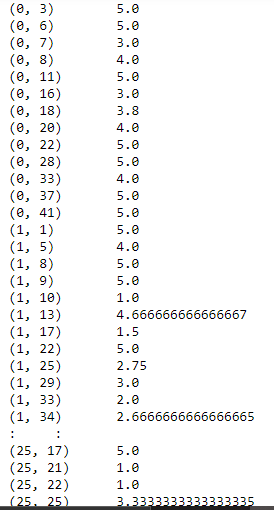


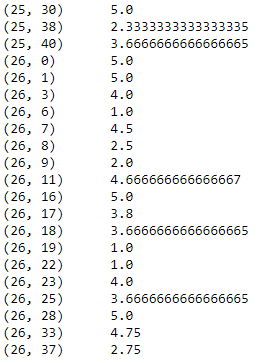
# sparse and csr matrix

from scipy.sparse import csr\_matrix

rating\_pivot\_matrix = csr\_matrix(rating\_pivot.values)

print(rating\_pivot\_matrix)





# fitting data into NearestNeighbor model

from sklearn.neighbors import NearestNeighbors

model\_knn = NearestNeighbors(metric = "cosine", n\_neighbors = 20, radius = 1)

model\_knn.fit(rating\_pivot\_matrix)

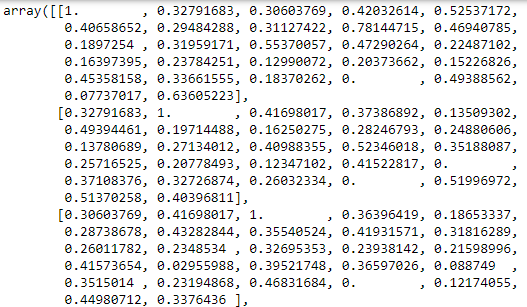


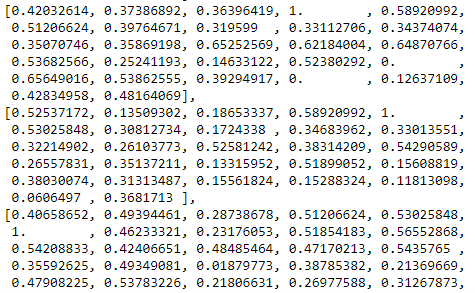
# matrix of cosine similarity

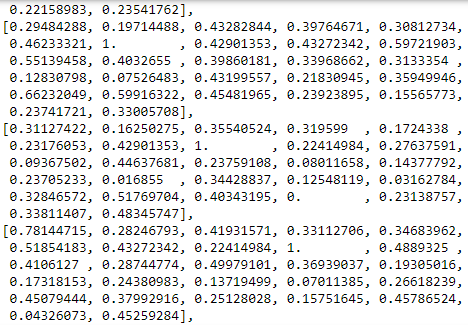
from sklearn.metrics.pairwise import cosine\_similarity

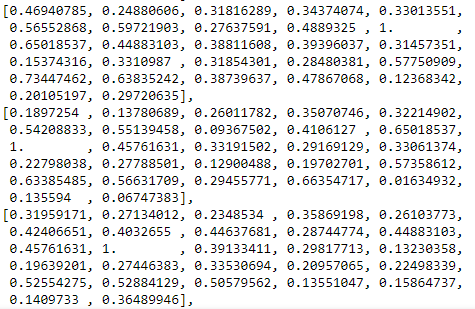
similarity\_matrix = cosine\_similarity(rating\_pivot)

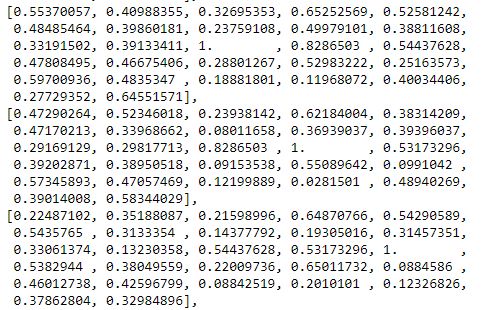
similarity\_matrix

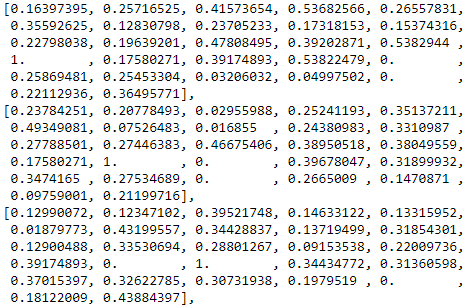


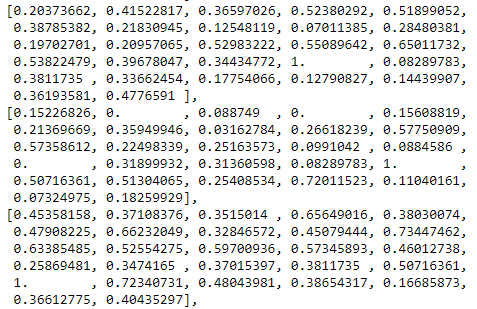


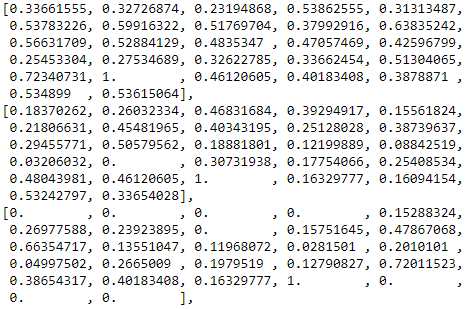


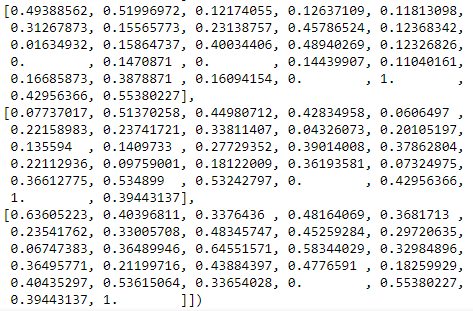












product\_Id = int(input("Enter product ID according to dataset: "))

data = list(rating\_pivot.index) #shows the productId list inside the data

print(data)



query\_index = data.index(product\_Id) # shows index of product id given by user.

print(query\_index)

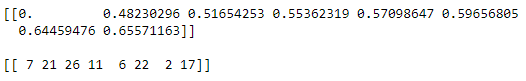


similarity, indices = model\_knn.kneighbors(rating\_pivot.iloc[query\_index].values.reshape(1, -1), n\_neighbors = 8)

print(similarity) # calculate distance through other productid to the id given by user

print()

print(indices) # print the index of product id given by user



# create dictionary and store the recommendation given by model to the user

data\_dict = {}

for i in range(0, len(similarity.flatten())):

if i == 0:

print("Recommendation for {} is: \n".format(rating\_pivot.index[query\_index]))

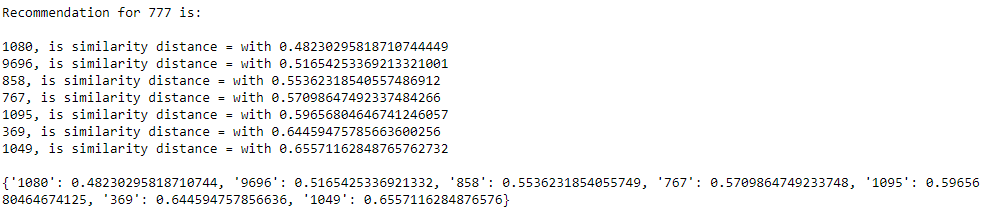
else:

data\_dict[str(rating\_pivot.index[indices.flatten()[i]])] = float(similarity.flatten()[i])

print(f"{rating\_pivot.index[indices.flatten()[i]]}, is similarity distance = with {similarity.flatten()[i]:.20f}")

print()

print(data\_dict)



**Task 14**

**Explain surprise package and its working.**

* Surprise is a scikit learn package for building and analyzing recommender systems that deal with explicit learning data.
* Surprise is stands for Simple Python Recommendation System Engine.
* **Purpose of surprise:**
* Give users perfect control over their experiments.
* User can use both inbuilt as well as custom datasets.
* Provide various ready to use prediction algorithm such as baseline algorithms, neighborhood methods, matrix factorization-based, and many others.
* Also, various similarity measures are built-in. Make it easy to implement new algorithm ideas.
* Provide tools to evaluate, analyse and compare the algorithm’s performance. Cross validation procedures can be run very easily using powerful CV iterators.

**Google BERT**

* BERT is launched by Google in 2019.
* In its starting time, it is only used in USA for 10% query only.
* Now, BERT is used almost in every country.
* Mainly 7 languages are supported in BERT, which are English, Spanish, Portuguese, Hindi, Arabic, German, Amharic
* BERT is stands for **B**idirectional **E**ncoder **R**epresentations from **T**ransformers.
* It helps to understand the meaning of particular word in context of emotion and according to culture.
* It is bidirectional because it starts reading the sentences from both the directions so that it can understand the meaning of sentence.
* Transformer is the mechanism which understands the interrelation between the words inside the sentence.
* Transformer has two parts Encoder and Decoder. Encoder understands the meaning and Decoder makes the meaningful sentence based on Encoder.
* BERT can understands the meaning of sentences which is sent by transformer and read by bidirectional, but can’t generate new sentence by itself.