# Slip 1

## Create ‘Position\_Salaries’ Data set. Build a linear regression model by identifying independent and target variable. Split the variables into training and testing sets. then divide the training and testing sets into a 7:3 ratio, respectively and print them. Build a simple linear regression model.

import numpy as np import pandas as pd

import matplotlib.pyplot as plt dataset = pd.read\_csv("Salary.csv")

X = dataset.iloc[:, :-1].values y = dataset.iloc[:,1].values

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=1/3, random\_state=0)

from sklearn.linear\_model import LinearRegression regressor = LinearRegression() regressor.fit(X\_train, y\_train)

y\_pred = regressor.predict(X\_test) print(y\_pred)

import matplotlib.pyplot as plt plt.scatter(X\_test, y\_train, color = 'red')

plt.plot(X\_train, regressor.predict(X\_train), color='blue') plt.title('Salary vs Experience (Test set)') plt.xlabel('Years of Experience')

plt.ylabel('Salary') plt.show()

# Slip 2

## Create ‘Salary’ Data set . Build a linear regression model by identifying independent and target variable. Split the variables into training and testing sets and print them. Build a simple linear regression model for predicting purchases.

import numpy as np import pandas as pd

import matplotlib.pyplot as plt dataset = pd.read\_csv("Salary.csv")

X = dataset.iloc[:, :-1].values y = dataset.iloc[:,1].values

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=1/3, random\_state=0)

from sklearn.linear\_model import LinearRegression regressor = LinearRegression() regressor.fit(X\_train, y\_train)

y\_pred = regressor.predict(X\_test) print(y\_pred)

import matplotlib.pyplot as plt plt.scatter(X\_test , y\_train , color = 'red')

plt.plot(X\_train , regressor.predict(X\_train) , color = 'green') plt.title("Salary vs Purchases")

plt.xlabel('Purchases') plt.ylabel('Salary') plt.show()

# Slip 3

## Create ‘User’ Data set having 5 columns namely: User ID, Gender, Age, Estimated Salary and Purchased. Build a logistic regression model that can predict whether on the given parameter a person will buy a car or not.

import numpy as np import pandas as pd

import matplotlib.pyplot as plt import seaborn as sns

from sklearn.linear\_model import LogisticRegression from sklearn.model\_selection import train\_test\_split from sklearn.preprocessing import StandardScaler from sklearn.metrics import classification\_report from sklearn.metrics import confusion\_matrix

from sklearn.metrics import accuracy\_score data = pd.read\_csv('suv\_data.csv') print(data.head(10))

data.info

print("Number of Customers " , len(data))

Gender = pd.get\_dummies(data['Gender'] , drop\_first = True) print(Gender.head(5))

data = pd.concat([data , Gender] , axis = 1) print(data.head(5))

#Dropping User ID and gender column

data.drop(['User ID' , 'Gender' ] ,axis = 1 , inplace = True) print(data.head(5))

X = data.drop('Purchased' , axis = 1) y = data['Purchased']

#Train and Test Data

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.2, random\_state = 1)

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train) X\_test = sc.transform(X\_test)

model = LogisticRegression(solver = 'liblinear')

model.fit(X\_train,y\_train)

LogisticRegression(C=1.0, class\_weight=None, dual=False, fit\_intercept=True, intercept\_scaling=1, max\_iter=100, multi\_class='warn',n\_jobs=None, penalty='l2', random\_state=None, solver='liblinear',tol=0.0001, verbose=0, warm\_start=False) predictions = model.predict(X\_test)

print(predictions) print(classification\_report(y\_test, predictions))

print("Confusion Matrix: \n",confusion\_matrix(y\_test, predictions)) print("Accuracy: ",accuracy\_score(y\_test, predictions)) plt.figure(figsize = (5,5)) sns.distplot(data[data['Purchased']==1]['Age']) data['EstimatedSalary'].plot.hist()

data['Age'].plot.hist() sns.countplot(x='Purchased', data = data) plt.figure(figsize = (20,10)) sns.barplot(x=data['Age'],y=data['Purchased'])

# Slip 4

## Build a simple linear regression model for Fish Species Weight Prediction

import numpy as np import pandas as pd

import matplotlib.pyplot as plt import seaborn as sns

from sklearn import linear\_model

from sklearn.model\_selection import train\_test\_split data = pd.read\_csv('Fish.csv')

print(data.head()) data.isnull().sum()

data.rename(columns={'Length1':'VerticalLen','Length2':'DiagonalLen','Length3':'C rossLen'},inplace = True)

data.sample(5) data.shape data.info()

data.Species.value\_counts()

data\_sp = data.Species.value\_counts() data\_sp = pd.DataFrame(data\_sp) data\_sp.T

new\_data = data.drop([40])

print("New dimension of the dataset is :-" , new\_data.shape) print(new\_data.head())

new\_data2 = new\_data.drop(['VerticalLen', 'DiagonalLen', 'CrossLen'], axis =1) # Can also use axis = 'columns'

print('New dimension of dataset is= ', new\_data2.shape) new\_data2.head()

sns.boxplot(x = new\_data2['Weight']) plt.title("Outlier Detection based on weight") def outlier\_detection(dataframe):

Q1 = dataframe.quantile(0.25) Q3 = dataframe.quantile(0.75)

IQR = Q3-Q1

upper\_end = Q3 + 1.5 \* IQR

lower\_end = Q1 - 1.5 \* IQR

Outlier = dataframe[(dataframe>upper\_end)| (dataframe<lower\_end)] return Outlier

outlier\_detection(new\_data2['Weight']) sns.boxplot(x = new\_data2['Height']) plt.title("Outlier detection based on height ") sns.boxplot(x = new\_data['Width']) plt.title("Outlier Detection based on Width") data3 = new\_data2.drop([142 , 143 , 144]) data3.shape

data3.describe().T

X = data3[['Height', 'Width']] X.head()

y = data3['Weight'] y.head()

X\_train,X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size =0.2, random\_state

= 42)

print('X\_train dimension= ', X\_train.shape) print('X\_test dimension= ', X\_test.shape) print('y\_train dimension= ', y\_train.shape) print('y\_train dimension= ', y\_test.shape) model = linear\_model.LinearRegression() model.fit(X\_train,y\_train)

print('coef:' , model.coef\_) print('Intercept:' ,model.intercept\_)

print('Score is :' , model.score(X\_test , y\_test)

predictedWeight = pd.DataFrame(model.predict(X\_test), columns=['Predicted Weight'])

actualWeight = pd.DataFrame(y\_test)

actualWeight = actualWeight.reset\_index(drop=True) # Drop the index so that we can concat it, to create new dataframe

df\_actual\_vs\_predicted = pd.concat([actualWeight,predictedWeight],axis =1) df\_actual\_vs\_predicted.T

def Visualize():

plt.scatter(X\_test['Width'], y\_test , color= 'red' , label = 'Actual Weight')

plt.scatter(X\_test['Width'] , model.predict(X\_test) , color = 'green' , label = 'Predicted Weight' )

plt.xlabel('Width') plt.ylabel('Weight') plt.rcParams["figure.figsize"] = (10,6)

plt.title('Actual vs Predicted weight for Test Data') plt.legend()

plt.show() Visualize()

sns.distplot((y\_test-model.predict(X\_test))) plt.rcParams["figure.figsize"] = (10,6) # Custom figure size in inches plt.title("Histogram of Residuals")

# Slip 5

## Use the iris dataset. Write a Python program to view some basic statistical details like percentile, mean, std etc. of the species of 'Iris-setosa', 'Iris- versicolor' an'Iris-virginica'. Apply logistic regression on the dataset to identify different species (setosa, versicolor, verginica) of Iris flowers given just 4 features: sepal and petal lengths and widths.. Find the accuracy of the model.

import numpy as np import pandas as pd

import matplotlib.pyplot as plt import seaborn as sns

from sklearn.linear\_model import LogisticRegression from sklearn.model\_selection import train\_test\_split from sklearn.preprocessing import StandardScaler from sklearn.metrics import classification\_report from sklearn.metrics import confusion\_matrix

from sklearn.metrics import accuracy\_score from sklearn.datasets import load\_iris

data = sns.load\_dataset("iris") print(data.head())

X = data.iloc[: , :-1]

y = data.iloc[:, -1]

#Split the data 80% on training data and 20% on test data

X\_train , X\_test , y\_train ,y\_test = train\_test\_split(X , y , test\_size = 0.2 , random\_state= 42)

model = LogisticRegression() model.fit(X\_train , y\_train) prediction = model.predict(X\_test) print(prediction)

print()

print(classification\_report(y\_test , prediction)) print(accuracy\_score(y\_test , prediction))

def Visualize\_iris\_dataset(): plt.xlabel('Features') plt.ylabel('Species')

pltX = data.loc[: , 'sepal\_length'] pltY = data.loc[: , 'species']

plt.scatter(pltX , pltY , color = 'blue' , label = 'sepal\_length') pltX = data.loc[:,'sepal\_width']

pltY = data.loc[: , 'species']

plt.scatter(pltX , pltY ,color = 'green' ,label='sepal\_width') pltX = data.loc[:,'petal\_length']

pltY = data.loc[:, 'species']

plt.scatter(pltX , pltY ,color = 'red' , label='petal\_length') pltX= data.loc[: , 'petal\_width']

pltY = data.loc[: , 'species']

plt.scatter(pltX , pltY ,color ='black' , label = 'petal\_width') plt.legend(loc = 4 , prop={'size':8})

plt.show() Visualize\_iris\_dataset()

# Slip 6

## Create the following dataset in python & Convert the categorical values into numeric format.Apply the apriori algorithm on the above dataset to generate the frequent itemsets and association rules. Repeat the process with different min\_sup value.

import pandas as pd

from mlxtend.preprocessing import TransactionEncoder from mlxtend.frequent\_patterns import apriori

from mlxtend.frequent\_patterns import association\_rules dataset = [

["Bread" , "Milk"] ,

["Bread" , "Diaper" ,"Beer" ,"Eggs"] ,

["Milk" ,"Diaper" ,"Bread" ,"Coke"] ,

["Bread" ,"Milk" , "Diaper" ,"Beer"],

["Bread" , "Milk" ,"Diaper" , "Coke"],

]

te = TransactionEncoder()

te\_array = te.fit(dataset).transform(dataset)

df = pd.DataFrame(te\_array , columns = te.columns\_) #Result after Preprocessing

print("Result after Preprocessing") print(df)

frequent\_itemsets\_ap = apriori(df ,min\_support=0.01 ,use\_colnames=True) print("\n Results after Applying apriori Alogorithm") print(frequent\_itemsets\_ap)

rules\_ap = association\_rules(frequent\_itemsets\_ap , metric="confidence" , min\_threshold=0.8)

frequent\_itemsets\_ap['length'] = frequent\_itemsets\_ap['itemsets'].apply(lambda x: len(x))

print("\n Frequent 2 Item Sets")

print(frequent\_itemsets\_ap[frequent\_itemsets\_ap['length']>=2]) print("\n Frequent 3 Item sets") print(frequent\_itemsets\_ap[frequent\_itemsets\_ap['length']>=3]) print("\n Frequent 4 Item sets") print(frequent\_itemsets\_ap[frequent\_itemsets\_ap['length']>=4])

# Slip 7

## Download the Market basket dataset. Write a python program to read the dataset and display its information. Preprocess the data (drop null values etc.) Convert the categorical values into numeric format. Apply the apriori algorithm on the above dataset to generate the frequent itemsets and association rules.

import numpy as np import pandas as pd

from matplotlib import pyplot as plt import seaborn as sns

from csv import reader

from mlxtend.preprocessing import TransactionEncoder

from mlxtend.frequent\_patterns import apriori, association\_rules groceries = []

with open('groceries.csv', 'r') as read\_obj: csv\_reader = reader(read\_obj)

for row in csv\_reader: groceries.append(row)

items = set(sum(groceries, []))

df = pd.DataFrame(columns=items) print(df)

# fitting the list and converting the transactions to true and false encoder = TransactionEncoder()

transactions = encoder.fit(groceries).transform(groceries) transactions = transactions.astype('int')

df = pd.DataFrame(transactions, columns=encoder.columns\_) df.head()

df.shape

frequent\_itemsets = apriori(df, min\_support=0.02, use\_colnames=True) frequent\_itemsets['length'] = frequent\_itemsets['itemsets'].apply(lambda x: len(x))

frequent\_itemsets

frequent\_itemsets = frequent\_itemsets.sort\_values(by='support', ascending=False)

print(frequent\_itemsets)

# finding top 5 items with minimum support of 2% frequent\_itemsets[ (frequent\_itemsets['length'] == 1) &

(frequent\_itemsets['support'] >= 0.02)][0:5]

# finding itemsets having length 2 and minimum support of 2% frequent\_itemsets[(frequent\_itemsets['length'] == 2) &

(frequent\_itemsets['support'] >= 0.02)]

rules = association\_rules(frequent\_itemsets, metric='support', min\_threshold=0.02)

rules

rules[(rules['support'] >= 0.02) & (rules['lift'] > 1.0)]

# Slip 8

## Download the groceries dataset. Write a python program to read the dataset and display its information. Preprocess the data (drop null values etc.) Convert the categorical values into numeric format. Apply the apriori algorithm on the above dataset to generate the frequent itemsets and association rules.

import numpy as np import pandas as pd

from matplotlib import pyplot as plt import seaborn as sns

from csv import reader

from mlxtend.preprocessing import TransactionEncoder

from mlxtend.frequent\_patterns import apriori, association\_rules # reading the dataset

groceries = []

with open('groceries.csv', 'r') as read\_obj: csv\_reader = reader(read\_obj)

for row in csv\_reader: groceries.append(row)

items = set(sum(groceries, []))

df = pd.DataFrame(columns=items) print(df)

# fitting the list and converting the transactions to true and false encoder = TransactionEncoder()

transactions = encoder.fit(groceries).transform(groceries) # converting the true and false to 1 and 0

transactions = transactions.astype('int')

df = pd.DataFrame(transactions, columns=encoder.columns\_) # viewing the first few rows of the dataframe

df.head() df.shape

# applying the apriori algorithm

frequent\_itemsets = apriori(df, min\_support=0.02, use\_colnames=True) frequent\_itemsets['length'] = frequent\_itemsets['itemsets'].apply(lambda x: len(x))

frequent\_itemsets

frequent\_itemsets = frequent\_itemsets.sort\_values(by='support', ascending=False)

print(frequent\_itemsets)

# finding top 5 items with minimum support of 2% frequent\_itemsets[ (frequent\_itemsets['length'] == 1) &

(frequent\_itemsets['support'] >= 0.02)][0:5]

# finding itemsets having length 2 and minimum support of 2% frequent\_itemsets[(frequent\_itemsets['length'] == 2) &

(frequent\_itemsets['support'] >= 0.02)]

# finding top 10 association rules with minimum support of 2% rules = association\_rules(frequent\_itemsets, metric='support', min\_threshold=0.02)

rules

rules[(rules['support'] >= 0.02) & (rules['lift'] > 1.0)]

# Slip 9

## Create your own transactions dataset and apply the above process on your dataset.

import numpy as np import pandas as pd

from matplotlib import pyplot as plt import seaborn as sns

from csv import reader

from mlxtend.preprocessing import TransactionEncoder

from mlxtend.frequent\_patterns import apriori, association\_rules groceries = []

with open('groceries.csv', 'r') as read\_obj: csv\_reader = reader(read\_obj)

for row in csv\_reader: groceries.append(row)

items = set(sum(groceries, []))

df = pd.DataFrame(columns=items) print(df)

encoder = TransactionEncoder()

transactions = encoder.fit(groceries).transform(groceries) transactions = transactions.astype('int')

df = pd.DataFrame(transactions, columns=encoder.columns\_) df.head()

df.shape

frequent\_itemsets = apriori(df, min\_support=0.02, use\_colnames=True) frequent\_itemsets['length'] = frequent\_itemsets['itemsets'].apply(lambda x: len(x))

frequent\_itemsets

frequent\_itemsets = frequent\_itemsets.sort\_values(by='support', ascending=False)

print(frequent\_itemsets)

frequent\_itemsets[ (frequent\_itemsets['length'] == 1) & (frequent\_itemsets['support'] >= 0.02) ][0:5] frequent\_itemsets[(frequent\_itemsets['length'] == 2) &

(frequent\_itemsets['support'] >= 0.02)]

rules = association\_rules(frequent\_itemsets, metric='support', min\_threshold=0.02)

rules

rules[(rules['support'] >= 0.02) & (rules['lift'] > 1.0)]

# Slip 10

## Create the following dataset in python & Convert the categorical values into numeric format.Apply the apriori algorithm on the above dataset to generate the frequent itemsets and association rules. Repeat the process with different min\_sup value

import pandas as pd

from mlxtend.preprocessing import TransactionEncoder from mlxtend.frequent\_patterns import apriori

from mlxtend.frequent\_patterns import association\_rules dataset = [

["Eggs", "Milk", "Bread"]

["Eggs", "Apple"],

["Milk", "Bread",],

["Apple", "Milk"],

["Milk", "Apple", "Bread"],

]

te = TransactionEncoder()

te\_array = te.fit(dataset).transform(dataset)

df = pd.DataFrame(te\_array , columns = te.columns\_) print("Result after Preprocessing")

print(df)

frequent\_itemsets\_ap = apriori(df ,min\_support=0.01 ,use\_colnames=True) print("\n Results after Applying apriori Alogorithm") print(frequent\_itemsets\_ap)

rules\_ap = association\_rules(frequent\_itemsets\_ap , metric="confidence" , min\_threshold=0.8)

frequent\_itemsets\_ap['length'] = frequent\_itemsets\_ap['itemsets'].apply(lambda x: len(x))

print("\n Frequent 2 Item Sets") print(frequent\_itemsets\_ap[frequent\_itemsets\_ap['length']>=2]) print("\n Frequent 3 Item sets") print(frequent\_itemsets\_ap[frequent\_itemsets\_ap['length']>=3])

print("\n Frequent 4 Item sets") print(frequent\_itemsets\_ap[frequent\_itemsets\_ap['length']>=4])

# Slip 11

## Create the following dataset in python & Convert the categorical values into numeric format.Apply the apriori algorithm on the above dataset to generate the frequent itemsets and associationrules. Repeat the process with different min\_sup values

import pandas as pd

from mlxtend.preprocessing import TransactionEncoder from mlxtend.frequent\_patterns import apriori

from mlxtend.frequent\_patterns import association\_rules dataset = [

["butter", "bread", "Milk"],

["butter", "flour", "Milk", "Sagar"],

["butter", "eggs", "milk", "salt"], ["eggs"],

["butter", "flour", "milk", "Salt"],

]

te = TransactionEncoder()

te\_array = te.fit(dataset).transform(dataset)

df = pd.DataFrame(te\_array , columns = te.columns\_)

print("Result after Preprocessing") print(df)

frequent\_itemsets\_ap = apriori(df ,min\_support=0.01 ,use\_colnames=True) print("\n Results after Applying apriori Alogorithm") print(frequent\_itemsets\_ap)

rules\_ap = association\_rules(frequent\_itemsets\_ap , metric="confidence" , min\_threshold=0.8)

frequent\_itemsets\_ap['length'] = frequent\_itemsets\_ap['itemsets'].apply(lambda x: len(x))

print("\n Frequent 2 Item Sets") print(frequent\_itemsets\_ap[frequent\_itemsets\_ap['length']>=2]) print("\n Frequent 3 Item sets")

print(frequent\_itemsets\_ap[frequent\_itemsets\_ap['length']>=3]) print("\n Frequent 4 Item sets") print(frequent\_itemsets\_ap[frequent\_itemsets\_ap['length']>=4])

# Slip 12

## Create ‘heights-and-weights’ Data set . Build a linear regression model by identifying independent and target variable. Split the variables into training and testing sets and print them. Build a simple linear regression model for predicting purchases.

import numpy as np import pandas as pd

import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression from sklearn.model\_selection import train\_test\_split data = pd.read\_csv("HeightWeight.csv") print(data.head())

data.describe() data.info()

height\_values = data["Height(Inches)"].values weight\_values = data["Weight(Pounds)"].values plt.scatter(weight\_values, height\_values) weight\_vector = weight\_values.reshape(-1,1)

x\_train, x\_test, y\_train, y\_test = train\_test\_split(weight\_vector, height\_values, train\_size=.8, test\_size=.2)

lm = LinearRegression() lm.fit(x\_train, y\_train) y\_predict = lm.predict(x\_test)

print(f"Train accuracy {round(lm.score(x\_train,y\_train)\*100,2)} %") print(f"Test accuracy {round(lm.score(x\_test,y\_test)\*100,2)} %") plt.scatter(x\_train,y\_train,color='red')

plt.plot(x\_test,y\_predict) plt.xlabel("Weight (Pounds)") plt.ylabel("Height (Inches)") plt.title("Trained Height Weight Data") plt.plot

# Slip 13

## Download nursery dataset from UCI. Build a linear regression model by identifying independent and target variable. Split the variables into training and testing sets and print them. Build a simple linear regression model for predicting purchases

import numpy as np import pandas as pd import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression from sklearn.model\_selection import train\_test\_split from sklearn.metrics import classification\_report

df = pd.read\_csv('nursery\_dataset.csv')

df = df.rename(columns={'final evaluation': 'final'}) L = len(df.index)

def print\_counts(df): for x in df.columns:

for i, y in zip(df[x].value\_counts().index, df[x].value\_counts()): i = str(i)

s = f'{i:14} {y/L:4.2f} {y:4d}'

print(s) print('\n')

print\_counts(df)

for x in df.drop(['health', 'final'], axis=1).columns: lst = list(df[x].value\_counts().index)

dic = {k:i+1 for i, k in enumerate(lst)} df[x].replace(dic, inplace=True)

print\_counts(df)

dic1 = {'recommended': 2, 'priority': 3,

'not\_recom': 1} dic2 = {'not\_recom':1,

'priority':4, 'spec\_prior':5,

'very\_recom':3, 'recommend':2}

df['health'].replace(dic1, inplace=True) df['final'].replace(dic2, inplace=True) print\_counts(df)

ind = (df.loc[:, 'final'] == 2) | (df.loc[:, 'final'] == 3) df\_23 = df[ind].reset\_index(drop=True)

df = df[~ind].reset\_index(drop=True) dic3 = {4: 2, 5: 3}

df['final'].replace(dic3, inplace=True) X = df.iloc[:, :-1]

y = df.iloc[:, -1]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.33, stratify=y) model = LinearRegression()

model.fit(X\_train, y\_train) yp\_train = model.predict(X\_train) yp\_test = model.predict(X\_test) print((y\_train, yp\_train))

X\_23, y\_23 = df\_23.iloc[:,:-1], df\_23.iloc[:,-1] y\_pred\_23 = model.predict(X\_23) print(y\_pred\_23)

plt.figure(figsize=(12, 6)) sns.heatmap(df.corr(), annot=True, fmt='.2f');

# Slip 14

## Create the following dataset in python & Convert the categorical values into numeric format.Apply the apriori algorithm on the above dataset to generate the frequent itemsets and association rules. Repeat the process with different min\_sup values

import pandas as pd

from mlxtend.preprocessing import TransactionEncoder from mlxtend.frequent\_patterns import apriori

from mlxtend.frequent\_patterns import association\_rules #Create Dataset

dataset = [

["Apple" , "Mango" , "Banana"] ,

["Mango" , "Banana" , "Cabbage" , "Carrots"] ,

["Mango" ,"banana" , "Carrots" ] ,

["Mango" , "Carrots"],

]

#Convert the list to dataframe with boolean Vlaues te = TransactionEncoder()

te\_array = te.fit(dataset).transform(dataset)

df = pd.DataFrame(te\_array , columns = te.columns\_) #Result after Preprocessing

print("Result after Preprocessing") print(df)

#Find the frequently occuring itemsets using Apriori Algorithm:- frequent\_itemsets\_ap = apriori(df ,min\_support=0.01 ,use\_colnames=True) print("\n Results after Applying apriori Alogorithm") print(frequent\_itemsets\_ap)

rules\_ap = association\_rules(frequent\_itemsets\_ap , metric="confidence" , min\_threshold=0.8)

frequent\_itemsets\_ap['length'] = frequent\_itemsets\_ap['itemsets'].apply(lambda x: len(x))

print("\n Frequent 2 Item Sets") print(frequent\_itemsets\_ap[frequent\_itemsets\_ap['length']>=2]) print("\n Frequent 3 Item sets") print(frequent\_itemsets\_ap[frequent\_itemsets\_ap['length']>=3]) print("\n Frequent 4 Item sets") print(frequent\_itemsets\_ap[frequent\_itemsets\_ap['length']>=4])

# Slip 15

## Create the following dataset in python & Convert the categorical values into numeric format.Apply the apriori algorithm on the above dataset to generate the frequent itemsets and association rules. Repeat the process with different min\_sup values

import pandas as pd

from mlxtend.preprocessing import TransactionEncoder from mlxtend.frequent\_patterns import apriori

from mlxtend.frequent\_patterns import association\_rules dataset = {

'Company':['Tata' , 'MG', 'KIA' , 'Hyundai'] ,

'Model': ['Nexon' , 'Altos' ,'Seltos' , 'Creta'], 'Year' : [2017 , 2021 , 2019 , 2015]

}

df = pd.DataFrame(dataset, index=['0', '1',

'2',

'3'])

print(df)

te = TransactionEncoder() te\_array = te.fit(df).transform(df)

df = pd.DataFrame(te\_array , columns = te.columns\_) print("Result after Preprocessing")

print(df)

frequent\_itemsets\_ap = apriori(df ,min\_support=0.01 ,use\_colnames=True) print("\n Results after Applying apriori Alogorithm") print(frequent\_itemsets\_ap)

rules\_ap = association\_rules(frequent\_itemsets\_ap , metric="confidence" , min\_threshold=0.8)

frequent\_itemsets\_ap['length'] = frequent\_itemsets\_ap['itemsets'].apply(lambda x: len(x))

print("\n Frequent 2 Item Sets")

print(frequent\_itemsets\_ap[frequent\_itemsets\_ap['length']>=2]) print("\n Frequent 3 Item sets") print(frequent\_itemsets\_ap[frequent\_itemsets\_ap['length']>=3]) print("\n Frequent 4 Item sets") print(frequent\_itemsets\_ap[frequent\_itemsets\_ap['length']>=4])

# Slip 16

## Consider any text paragraph. Preprocess the text to remove any special characters and digits. Generate the summary using extractive summarization process

import bs4 as bs import urllib.request import re

import nltk nltk.download('punkt') nltk.download('stopwords') scraped\_data =

urllib.request.urlopen('https://en.wikipedia.org/wiki/Severe\_acute\_respiratory\_s yndrome\_coronavirus\_2')

article = scraped\_data.read()

parsed\_article = bs.BeautifulSoup(article,'lxml') paragraphs = parsed\_article.find\_all('p') article\_text = ""

for p in paragraphs: article\_text += p.text

# Removing Square Brackets and Extra Spaces article\_text = re.sub(r'\[[0-9]\*\]', ' ', article\_text) article\_text = re.sub(r'\s+', ' ', article\_text)

# Removing special characters and digits formatted\_article\_text = re.sub('[^a-zA-Z]', ' ', article\_text )

formatted\_article\_text = re.sub(r'\s+', ' ', formatted\_article\_text) sentence\_list = nltk.sent\_tokenize(article\_text)

stopwords = nltk.corpus.stopwords.words('english') word\_frequencies = {}

for word in nltk.word\_tokenize(formatted\_article\_text): if word not in stopwords:

if word not in word\_frequencies.keys(): word\_frequencies[word] = 1

else:

word\_frequencies[word] += 1

maximum\_frequncy = max(word\_frequencies.values()) for word in word\_frequencies.keys():

word\_frequencies[word] = (word\_frequencies[word]/maximum\_frequncy) sentence\_scores = {}

for sent in sentence\_list:

for word in nltk.word\_tokenize(sent.lower()): if word in word\_frequencies.keys():

if len(sent.split(' ')) < 30:

if sent not in sentence\_scores.keys(): sentence\_scores[sent] = word\_frequencies[word]

else:

sentence\_scores[sent] += word\_frequencies[word] import heapq

summary\_sentences = heapq.nlargest(7, sentence\_scores, key=sentence\_scores.get)

summary = ' '.join(summary\_sentences) print(summary)

# Slip 17

**Consider text paragraph.*So, keep working. Keep striving. Never give up. Fall down seven times, get up eight. Ease is a greater threat to progress than hardship. Ease is a greater threat to progress than hardship. So, keep moving, keep growing, keep learning. See you at work.*Preprocess the text to remove any special characters and digits. Generate the summary using extractive summarization process.**

import nltk

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize, sent\_tokenize

text = """ keep working. Keep striving. Never give up. Fall down seven times, get up eight. Ease is a greater threat to progress than hardship. Ease is a greater threat to progress than

hardship. So, keep moving, keep growing, keep learning. See you at work""" stopWords = set(stopwords.words("english"))

words = word\_tokenize(text) freqTable = dict()

for word in words: word = word.lower() if word in stopWords:

continue

if word in freqTable: freqTable[word] += 1

else:

freqTable[word] = 1 sentences = sent\_tokenize(text) sentenceValue = dict()

for sentence in sentences:

for word, freq in freqTable.items(): if word in sentence.lower():

if sentence in sentenceValue: sentenceValue[sentence] += freq

else:

sentenceValue[sentence] = freq

sumValues = 0

for sentence in sentenceValue: sumValues += sentenceValue[sentence]

average = int(sumValues / len(sentenceValue)) summary = ''

for sentence in sentences:

if (sentence in sentenceValue) and (sentenceValue[sentence] > (1.2 \* average)): summary += " " + sentence

print(summary)

# Slip 18

## Consider any text paragraph. Remove the stopwords. Tokenize the paragraph to extract words and sentences. Calculate the word frequency distribution and plot the frequencies. Plot the wordcloud of the text.

import warnings warnings.filterwarnings('ignore') import numpy as np

import pandas as pd import nltk

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize, sent\_tokenize import string

import collections

from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator import matplotlib.cm as cm

import matplotlib.pyplot as plt

text = """ keep working. Keep striving. Never give up. Fall down seven times, get up eight. Ease is a greater threat to progress than hardship. Ease is a greater threat to progress than

hardship. So, keep moving, keep growing, keep learning. See you at work""" stopWords = set(stopwords.words("english"))

words = word\_tokenize(text) freqTable = dict()

for word in words: word = word.lower() if word in stopWords:

continue

if word in freqTable: freqTable[word] += 1

else:

freqTable[word] = 1 sentences = sent\_tokenize(text) sentenceValue = dict()

for sentence in sentences:

for word, freq in freqTable.items(): if word in sentence.lower():

if sentence in sentenceValue: sentenceValue[sentence] += freq

else:

sentenceValue[sentence] = freq sumValues = 0

for sentence in sentenceValue: sumValues += sentenceValue[sentence]

wordcloud\_spam = WordCloud(background\_color="white").generate(text) plt.figure(figsize = (20,20))

plt.imshow(wordcloud\_spam, interpolation='bilinear') plt.axis("off")

plt.show()

# Slip 19

## Download the movie\_review.csv dataset from Kaggle by using the following link https://[www.kaggle.com/nltkdata/movie](http://www.kaggle.com/nltkdata/movie) review/version/3?select=movie\_review.csv to perform sentiment analysis on above dataset and create a word cloud

import numpy as np import pandas as pd import re

import string import nltk

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize

from nltk.stem.snowball import SnowballStemmer

from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator import matplotlib.cm as cm

import matplotlib.pyplot as plt

data = pd.read\_csv('movie\_review.csv') print(data.head())

data.isnull() data.dtypes data.shape

stemmer = SnowballStemmer(language='english') def preprocessing(phrase):

lower = [phrase.lower() for phrase in phrase]

no\_punct = [text.translate(str.maketrans('','',string.punctuation)) for text in lower]

stem = [stemmer.stem(i) for i in no\_punct] join = ["".join(text) for text in stem]

return join label = data['text']

data1 = preprocessing(data["tag"]) print(data1)

from sklearn.feature\_extraction.text import TfidfVectorizer vectorizer= TfidfVectorizer(stop\_words = 'english')

data2 = vectorizer.fit\_transform(data1) #tf\_x\_test = vectorizer.transform(test\_final) print(data2)

from sklearn.svm import LinearSVC clf = LinearSVC(random\_state=0) clf.fit(data2,label)

print(clf) y\_pred=clf.predict(data2) print(y\_pred)

text = """

films adapted from comic books have had plenty of success , whether they're about superheroes ( batman , superman , spawn ) , or geared toward kids ( casper

) or the arthouse crowd ( ghost world ) , but there's never really been a comic book like from hell before .for starters , it was created by alan moore ( and eddie campbell ) , who brought the medium to a whole new level in the mid '80s with a 12-part series called the watchmen .to say moore and campbell thoroughly researched the subject of jack the ripper would be like saying michael jackson is starting to look a little odd .the book ( or " graphic novel , " if you will ) is over 500 pages long and includes nearly 30 more that consist of nothing but footnotes .in other words , don't dismiss this film because of its source.if you can get past the whole comic book thing , you might find another stumbling block in from hell's directors , albert and allen hughes .getting the hughes brothers to direct this seems almost as ludicrous as casting carrot top in , well , anything , but riddle me this : who better to direct a film that's set in the ghetto and features really violent street crime than the mad geniuses behind menace ii society ?

the ghetto in question is , of course , whitechapel in 1888 london's east end . """

wordcloud\_spam = WordCloud(background\_color="white").generate(text)

# Lines 2 - 5

plt.figure(figsize = (20,20)) plt.imshow(wordcloud\_spam, interpolation='bilinear') plt.axis("off")

plt.show()

# Slip 20

**Consider text paragraph.*"""Hello all, Welcome to Python Programming Academy. Python Programming Academy is a nice platform to learn new programming skills. It is difficult to get enrolled in this Academy."""*Remove the stopwords**.

import warnings warnings.filterwarnings('ignore') #loading all necessary libraries import numpy as np

import pandas as pd import nltk

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize, sent\_tokenize import string

import collections

text = """ Hello all, Welcome to Python Programming Academy.

Python Programming Academy is a nice platform to learn new programming skills. It is difficult to get enrolled in this Academy."""

# Tokenizing the text

stopWords = set(stopwords.words("english")) words = word\_tokenize(text)

print(words)

# Slip 21

## Build a simple linear regression model for User Data.

import matplotlib.pyplot as plt import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split from sklearn.linear\_model import LinearRegression

x= np.array([2.4,5.0,1.5,3.8,8.7,3.6,1.2,8.1,2.5,5,1.6,1.6,2.4,3.9,5.4])

y = np.array([2.1,4.7,1.7,3.6,8.7,3.2,1.0,8.0,2.4,6,1.1,1.3,2.4,3.9,4.8])

n = np.size(x)

X\_train , x\_test , Y\_train , y\_test = train\_test\_split(x , y , test\_size = 0.2 , random\_state = 1)

print(X\_train.shape) print(Y\_train.shape) print(x\_test.shape) print(y\_test.shape)

regressor = LinearRegression() X\_train = X\_train.reshape(-1,1) regressor.fit(X\_train , Y\_train) print(regressor.intercept\_) print(regressor.coef\_)

x\_test = x\_test.reshape(-1,1) y\_pred = regressor.predict(x\_test) print(y\_pred)

# Slip 22

## Consider any text paragraph. Remove the stopwords.

import warnings warnings.filterwarnings('ignore')

import numpy as np import pandas as pd import nltk

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize , sent\_tokenize text = """

Hello , Hi everyone It's John your lovely data scientists If you want any help , feel free

to reach me . Okaayyy Thank you

"""

StopWords = set(stopwords.words("english")) words = word\_tokenize(text)

print(words) print(StopWords)

# Slip 23

## Consider any text paragraph. Preprocess the text to remove any special characters and digits.

import nltk

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize, sent\_tokenize

text =""" keep working. Keep striving. Never give up. Fall down seven times, get up eight. Ease is a greater threat to progress than hardship. Ease is a greater threat to progress than hardship. So, keep moving, keep growing, keep learning. See you at work"""

stopWords = set(stopwords.words("english")) words = word\_tokenize(text)

freqTable = dict() for word in words:

word = word.lower() if word in stopWords:

continue

if word in freqTable: freqTable[word] += 1

else:

freqTable[word] = 1

sentences = sent\_tokenize(text) sentenceValue = dict()

for sentence in sentences:

for word, freq in freqTable.items(): if word in sentence.lower():

if sentence in sentenceValue: sentenceValue[sentence] += freq

else:

sentenceValue[sentence] = freq sumValues = 0

for sentence in sentenceValue: sumValues += sentenceValue[sentence]

average = int(sumValues / len(sentenceValue)) summary = ''

for sentence in sentences:

if (sentence in sentenceValue) and (sentenceValue[sentence] > (1.2 \* average)): summary += " " + sentence

print(summary)

# Slip 24

**Consider the following dataset : https://**[**www.kaggle.com/datasets/datasnaek/youtube-**](http://www.kaggle.com/datasets/datasnaek/youtube-) **new?select=INvideos.csv**

**Write a Python script for the following :**

1. **Read the dataset and perform data cleaning operations on it.**
2. **Find the total views, total likes, total dislikes and comment count**.

import numpy as np import pandas as pd

import matplotlib.pyplot as plt import seaborn as sns

import re

import plotly.express as px import plotly.graph\_objects as go

df\_videos = pd.read\_csv('INvideos.csv', error\_bad\_lines=False) df\_videos.head()

df\_videos.shape df\_videos.columns df\_videos.isnull().sum() new\_df =

df\_videos['video\_id'].value\_counts().rename\_axis('video\_id').head(10).reset\_inde x(name='counts')

fig = px.bar(new\_df, x="video\_id", y="counts") fig.show()

df1 =pd.DataFrame(df\_videos.channel\_title.value\_counts()) df1.columns=['times channel got trenidng']

df1.head(6) df\_channel

=pd.DataFrame(df\_videos.groupby(by=['channel\_title'])['views'].mean()).sort\_val ues(by='views',ascending=False)

df\_channel.head(10).plot(kind='bar'); plt.title('Most viewed channels'); sns.regplot(data=df\_videos,x='views',y='likes') plt.title('Regression plot for views & likes') sns.regplot(data=df\_videos,x='views',y='dislikes') plt.title('Regression plot for views & dislikes')

# Slip 25

**Consider the following dataset : https://**[**www.kaggle.com/datasets/seungguini/youtube-**](http://www.kaggle.com/datasets/seungguini/youtube-) **comments-for-covid19-relatedvideos?select=covid\_2021\_1.csv**

**Write a Python script for the following :**

1. **Read the dataset and perform data cleaning operations on it.**
2. **ii. Tokenize the comments in words. iii. Perform sentiment analysis and find the percentage of positive, negative and neutral comments**..

from mpl\_toolkits.mplot3d import Axes3D

from sklearn.preprocessing import StandardScaler import matplotlib.pyplot as plt

import numpy as np import os

import pandas as pd import nltk

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize, sent\_tokenize nRowsRead = 1000

df1 = pd.read\_csv('youtube\_comments\_coronavirus.csv') nRow, nCol = df1.shape

print(f'There are {nRow} rows and {nCol} columns') df1.head(5)

df1.isnull().sum() df1.shape

to\_drop = ['url' , 'title' ,'views' , 'likes', 'dislikes'] df1.drop(to\_drop, inplace=True, axis=1) print(df1.head())

data = df1['comment\_text'].map(lambda x: x.lstrip('+-').rstrip('aAbBcC')) print(data.head())

text = """

Everyone who reads this message may your mother lives a 100 years.

Just imagine, years from now kids are gonna watch this in class and take a test on this pandemic

Who else started manually breathing while watching the part of the video talking and showing lungs See you in 10 years when it gets recommended to you!!! When I get the vaccine I hope I would get the War Hammer titan.

Canâ€™t wait for this to be all over Jet lock airplane travellerscoronavirus without air plane hospital becoming airplanes to spread corona virus

I have had covid for 12 days and my dad has had it for 17 days I hope we get well soon and I hope all of you guys that are struggling yk with the virus get well soon This virus has really taught me to appreciate what you and the people you have because you never know when youâ€™ll see them againhere we are 2021 less then a year latter! HUMANITY is extraordinaryMy grandma suffers from corona and is ICU, I would ask everyone to pray for her fast recovery

ðŸ˜¢Thestart Everyone be like: COVER YOUR SNEEZE

The festive dime phylogenetically flower because arch longitudinally seal absent a cold playroom. sweet, sick spleen

Corona" is the Spanish, Catalan, Hungarian, and Italian word for "crown".prayers to everyone who got covid-19

Who ever reading this - I pray for you and your family's good health.ÐŸÐ¾Ð»Ñ‚Ð¾Ñ€Ñ‹ Ð½ÐµÐ´ÐµÐ»Ð¸ Ñ‚ÐµÐ¼Ð¿ÐµÑ€Ð°Ñ‚ÑƒÑ€Ð°, Ð½ÐµÐ´ÐµÐ»ÑŽ Ð´Ð¸Ð°Ñ€ÐµÑ , Ñ‚Ñ€Ð¸ Ð´Ð½Ñ Ð´Ð°Ð²Ð»ÐµÐ½Ð¸Ðµ Ð¸ ÐºÑ€Ð¾Ð²ÑŒ Ð¸Ð· Ð´ÐµÑ ÐµÐ½, Ð¿Ð¾Ñ‚Ð¾Ð¼ Ð½ÐµÐ´ÐµÐ»Ñ ÐºÐ°ÑˆÐ»Ñ Ð¸ Ñ Ð»Ð°Ð±Ð¾Ñ Ñ‚Ð¸.

My mother is experiencing time to time fever,extreme weakness,headaches,mild cough,chills and back pain... I'm confused n scared about her plz pray for her recovery and healthâ ¤ï¸

That's life and you never know what tomorrow brings !!

Every vacation we swolling corona virus germs in flight zuzubi matter 6 months problem

Dear 2019, Iâ€™m sorry Iâ€™ve ever complained about youðŸ˜ðŸ˜‚ 99.5% live after infection of C19 you end up living

It was predicted that coronavirus would end by the end of this year so buckle up Some people just die from this so sad found out wow god bless them and those people going through losing there fam man wow.

I hate having to isolate for this long. I got it then 2 weeks later my kids got it. I have been inside for 3 weeks. It's really affecting my mental health.

The lonely objective ipsilaterally mine because invention eventually dream amongst a abortive pants. hurried, fine zinc

Ã“timo vÃdeo, Ã³tima explicaÃ§Ã£o Simplesmente,ameiiiâ ¤ï¸

This channel is PERFECT ABSOLUTELY PERFECT, other than the graphics and amazing animations, this thing has been helpful to know! Thanks to you... no all of

nucleus medical media i found out the perfect video to share to my classmates at home. Keep up thew good work, stay safe and stay strong!

Thanks So Much Dr Otor for your wonderful change in my life with your Herbal Medicine curing my Herpes illness I love you so much Dr Otor

RIP to all that have died from Covid 19. We will never forget this terrible tragedy. ã‚³ãƒãƒŠäººé¡žæ»…ã ³ã‚‹ã€€ã€€ç—…åŽŸå¾®ç”Ÿç‰©å·»ã è¿”ã —

how could you say that is sars cov2? when it hasn't been isolated or photographed? are you parroting the theories?

My body does more work then me, itâ€™s fighting wars while I sit on my couch """

stopWords = set(stopwords.words("english")) words = word\_tokenize(text)

print(words)

# Slip 26

## Consider text paragraph. """Hello all, Welcome to Python Programming Academy. Python Programming Academy is a nice platform to learn new programming skills. It is difficult to get enrolled in this Academy.""" Preprocess the text to remove any special characters and digits. Generate the summary using extractive summarization process.

import nltk

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize, sent\_tokenize

text = """ Hello all, Welcome to Python Programming Academy. Python Programming Academy

is a nice platform to learn new programming skills. It is difficult to get enrolled in this Academy."""

StopWords = set(stopwords.words("english")) words = word\_tokenize(text)

print(words) freqtable = dict() for word in words:

word = word.lower() if word in StopWords:

continue

if word in freqtable: freqtable[word]+=1

else:

freqtable[word] = 1 sentences = sent\_tokenize(text) sentenceValue = dict()

for sentence in sentences:

for word, freq in freqtable.items(): if word in sentence.lower():

if sentence in sentenceValue: sentenceValue[sentence] += freq

else:

sentenceValue[sentence] = freq sumValues = 0

for sentence in sentenceValue: sumValues += sentenceValue[sentence]

print(sentences)

average = int(sumValues / len(sentenceValue)) summary = ''

for sentence in sentences:

if(sentence in sentenceValue) and (sentenceValue[sentence]>(1.2\*average)): summary +=" "+sentence

print(summary)

# Slip 27

## Create your own transactions dataset and apply the above process on your dataset

import numpy as np import pandas as pd

from matplotlib import pyplot as plt import seaborn as sns

from csv import reader

from mlxtend.preprocessing import TransactionEncoder

from mlxtend.frequent\_patterns import apriori, association\_rules groceries = []

with open('groceries.csv', 'r') as read\_obj: csv\_reader = reader(read\_obj)

for row in csv\_reader: groceries.append(row)

items = set(sum(groceries, []))

df = pd.DataFrame(columns=items) print(df)

encoder = TransactionEncoder()

transactions = encoder.fit(groceries).transform(groceries) transactions = transactions.astype('int')

df = pd.DataFrame(transactions, columns=encoder.columns\_) df.head()

df.shape

frequent\_itemsets = apriori(df, min\_support=0.02, use\_colnames=True) frequent\_itemsets['length'] = frequent\_itemsets['itemsets'].apply(lambda x: len(x))

frequent\_itemsets

frequent\_itemsets = frequent\_itemsets.sort\_values(by='support', ascending=False)

print(frequent\_itemsets)

frequent\_itemsets[ (frequent\_itemsets['length'] == 1) & (frequent\_itemsets['support'] >= 0.02) ][0:5]

frequent\_itemsets[(frequent\_itemsets['length'] == 2) & (frequent\_itemsets['support'] >= 0.02)]

rules = association\_rules(frequent\_itemsets, metric='support', min\_threshold=0.02)

rules

rules[(rules['support'] >= 0.02) & (rules['lift'] > 1.0)]

# Slip 28

## Build a simple linear regression model for Car Dataset.

import warnings warnings.filterwarnings('ignore') import numpy as np

import pandas as pd

import matplotlib.pyplot as plt import seaborn as sns

from sklearn.preprocessing import StandardScaler Scaler = StandardScaler()

cars = pd.read\_csv('CarPrice\_Assignment.csv') cars.head()

cars.shape cars.describe() cars.info()

cars.loc[cars.duplicated()] cars.columns plt.figure(figsize=(20,8)) plt.subplot(1,2,1)

plt.title('Car Price Distribution Plot') sns.distplot(cars.price) plt.subplot(1,2,2)

plt.title('Car Price Spread') sns.boxplot(y=cars.price) plt.show() plt.figure(figsize=(25, 6)) df =

pd.DataFrame(cars.groupby(['horsepower'])['price'].mean().sort\_values(ascendin g = False))

df.plot.bar()

plt.title('HorsePower vs Average Price') plt.show()

df = pd.DataFrame(cars.groupby(['stroke'])['price'].mean().sort\_values(ascending

= False)) df.plot.bar()

plt.title('Stroke vs Average Price') plt.show()

df = pd.DataFrame(cars.groupby(['enginesize'])['price'].mean().sort\_values(ascending

= False)) df.plot.bar()

plt.title('Engine\_size vs Average Price') plt.show()

cars = pd.read\_csv('CarPrice\_Assignment.csv') cars.head()

X = cars.iloc[:, :-1].values y = cars.iloc[:,1].values

x =np.array(cars['price'])

y =np.array(cars['horsepower']) print(X)

print(y) print(x) print(y)

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=1/3, random\_state=0)

X\_train.shape y\_train.shape

X\_train= X\_train.reshape(-1 , 1) y\_train = y\_train.reshape(-1 , 1) X\_test = X\_test.reshape(-1 , 1)

from sklearn.linear\_model import LinearRegression regressor = LinearRegression() regressor.fit(X\_train, y\_train)

y\_pred = regressor.predict(X\_test) print(y\_pred)

# Slip 29

## Build a logistic regression model for Student Score Dataset.

import numpy as np import pandas as pd

import matplotlib.pyplot as plt import seaborn as sns

from sklearn.linear\_model import LogisticRegression from sklearn.model\_selection import train\_test\_split from sklearn.preprocessing import StandardScaler from sklearn.preprocessing import LabelEncoder

lc = LabelEncoder()

from sklearn.metrics import classification\_report from sklearn.metrics import confusion\_matrix from sklearn.metrics import accuracy\_score

df = pd.read\_csv('StudentsPerformance.csv') print(df.head())

df.shape df.describe()

df["mean\_Score"] = ((df["math score"]+df["reading score"]+df["writing score"])/3).round()

df.head() df['gender'].value\_counts()

df['gender'] = lc.fit\_transform(df['gender']) df['race/ethnicity'] = lc.fit\_transform(df['race/ethnicity'])

df['parental level of education'] = lc.fit\_transform(df['parental level of education'])

df['lunch'] = lc.fit\_transform(df['lunch'])

df['test preparation course'] = lc.fit\_transform(df['test preparation course']) df.head()

df['test preparation course'].value\_counts()

df = df.drop(['math score', 'writing score', 'reading score'],axis = 1) df.head()

y = df['mean\_Score']

x = df.drop(['mean\_Score'], axis = 1)

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x,y,test\_size = 0.2, random\_state

= 0)

model = LogisticRegression(solver='liblinear', random\_state=0) print(model)

model.fit(x\_train, y\_train) predictions = model.predict(x\_test) print(predictions)

difference = abs(predictions - y\_test) print(difference)

difference.mean()

labels = ['None', 'Completed'] colors = ['blue', 'gold']

plt.pie(df['test preparation course'].value\_counts() , labels = labels, colors = colors)

sns.countplot(x = df['gender'], hue = df['race/ethnicity']) plt.figure(figsize = (12,6))

sns.pairplot(df) plt.show() plt.figure(figsize = (12,6)) sns.heatmap(df.corr()) plt.show()

# Slip 30

**Create the dataset . transactions = [['eggs', 'milk','bread'], ['eggs', 'apple'], ['milk', 'bread'], ['apple', 'milk'], ['milk', 'apple', 'bread']] .**

**Convert the categorical values into numeric format.Apply the apriori algorithm on the above dataset to generate the frequent itemsets and association rules**.

import pandas as pd

from mlxtend.preprocessing import TransactionEncoder from mlxtend.frequent\_patterns import apriori

from mlxtend.frequent\_patterns import association\_rules dataset = [

["eggs" , "milk" , "bread"] ,

["eggs" , "apple"] ,

["milk" ,"bread" ] ,

["Apple" ,"milk"],

["Milk" , "apple" ,"bread"],

]

te = TransactionEncoder()

te\_array = te.fit(dataset).transform(dataset)

df = pd.DataFrame(te\_array , columns = te.columns\_) print("Result after Preprocessing")

print(df)

frequent\_itemsets\_ap = apriori(df ,min\_support=0.01 ,use\_colnames=True) print("\n Results after Applying apriori Alogorithm") print(frequent\_itemsets\_ap)

rules\_ap = association\_rules(frequent\_itemsets\_ap , metric="confidence" , min\_threshold=0.8)

frequent\_itemsets\_ap['length'] = frequent\_itemsets\_ap['itemsets'].apply(lambda x: len(x))

print("\n Frequent 2 Item Sets") print(frequent\_itemsets\_ap[frequent\_itemsets\_ap['length']>=2]) print("\n Frequent 3 Item sets") print(frequent\_itemsets\_ap[frequent\_itemsets\_ap['length']>=3]) print("\n Frequent 4 Item sets") print(frequent\_itemsets\_ap[frequent\_itemsets\_ap['length']>=4])