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 pandas as pd

import numpy as np

import matplotlib.pyplot as plt

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

from sklearn.linear\_model import LinearRegression

dataset = pd.read\_csv('Downloads/archive/Position\_Salaries.csv')

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

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=0)

regressor = LinearRegression()

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

print("Training set:")

print("X\_train:", X\_train)

print("y\_train:", y\_train)

print("Testing set:")

print("X\_test:", X\_test)

print("y\_test:", y\_test)

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 pandas as pd

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

from sklearn.linear\_model import LinearRegression

df = pd.DataFrame({

'Position': ['Software Engineer', 'Product Manager', 'Data Scientist', 'Sales Executive', 'Marketing Manager', 'Intern'],

'Level': [1, 2, 3, 4, 5, 6],

'Salary': [5000, 8000, 11000, 15000, 20000, 25000]

})

X = df[['Level']]

y = df['Salary']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

lr = LinearRegression()

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

print('Training set:')

print('X\_train:', X\_train)

print('y\_train:', y\_train)

print('Testing set:')

print('X\_test:', X\_test)

print('y\_test:', y\_test)

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 pandas as pd

import numpy as np

user\_data = {'User ID': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],

'Age': [19, 35, 26, 27, 19, 32, 25, 29, 34, 25],

'Estimated Salary': [19000, 20000, 43000, 57000, 76000, 58000, 84000, 15000, 43000, 22000],

'Purchased': [0, 0, 0, 0, 0, 1, 1, 0, 1, 0]}

user\_df = pd.DataFrame(user\_data)

X = user\_df.iloc[:, 1:-1].values

y = user\_df.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=0.3, random\_state=0)

from sklearn.linear\_model import LogisticRegression

logistic\_model = LogisticRegression()

logistic\_model.fit(X\_train, y\_train)

y\_pred = logistic\_model.predict(X\_test)

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score

print("Accuracy:", accuracy\_score(y\_test, y\_pred))

print("Precision:", precision\_score(y\_test, y\_pred))

print("Recall:", recall\_score(y\_test, y\_pred))

print("F1-score:", f1\_score(y\_test, y\_pred))

slip 4

Build a simple linear regression model for import pandas as pd

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

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score

data = pd.read\_csv('Downloads/archive (2)/Fish.csv')

X\_train, X\_test, y\_train, y\_test = train\_test\_split(data.drop('Weight', axis=1), data['Weight'], test\_size=0.2, random\_state=42)

model = LinearRegression()

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

y\_pred = model.predict(X\_test)

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

print('Mean Squared Error:', mse)

print('R-squared:', r2)

new\_data = pd.DataFrame({'length': [25.4, 20.5, 30.0]})

predictions = model.predict(new\_data)

print('Predictions:', predictions)

slip 5

import pandas as pd

from sklearn.datasets import load\_iris

iris = load\_iris()

iris\_df = pd.DataFrame(data=iris.data, columns=iris.feature\_names)

iris\_df['target'] = iris.target

iris\_df['target'] = iris\_df['target'].apply(lambda x: iris.target\_names[x])

print(iris\_df[iris\_df['target'] == 'setosa'].describe())

print(iris\_df[iris\_df['target'] == 'versicolor'].describe())

print(iris\_df[iris\_df['target'] == 'virginica'].describe())

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

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score

iris = load\_iris()

X = iris.data

y = iris.target

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

model = LogisticRegression()

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

y\_pred = model.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

print('Accuracy:', accuracy)

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 values.

import pandas as pd

from mlxtend.preprocessing import TransactionEncoder

from mlxtend.frequent\_patterns import apriori, association\_rules

data = [['Milk', 'Egg', 'Bread'],

['Milk', 'Bread'],

['Milk', 'Egg', 'Bread', 'Cheese'],

['Milk', 'Egg'],

['Bread', 'Cheese']]

te = TransactionEncoder()

te\_ary = te.fit\_transform(data)

tid = pd.DataFrame(te\_ary, columns=te.columns\_)

items = tid.astype('int')

items = items.replace({True: 1, False: 0})

min\_sup\_values = [0.4, 0.6, 0.8]

for min\_sup in min\_sup\_values:

frequent\_itemsets = apriori(items, min\_support=min\_sup, use\_colnames=True)

rules = association\_rules(frequent\_itemsets, metric="confidence", min\_threshold=0.7)

print('Min support:', min\_sup)

print('Frequent itemsets:')

print(frequent\_itemsets)

print('Association rules:')

print(rules)

print()

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 pandas as pd

from mlxtend.preprocessing import TransactionEncoder

from mlxtend.frequent\_patterns import apriori, association\_rules

import urllib.request

url = "http://archive.ics.uci.edu/ml/machine-learning-databases/00352/Online%20Retail.xlsx"

filename = "Online Retail.xlsx"

urllib.request.urlretrieve(url, filename)

df = pd.read\_excel(filename)

print("Dataset information:")

print(df.info())

df = df.dropna()

df = df[df['Quantity'] > 0]

df['StockCode'] = pd.to\_numeric(df['StockCode'], errors='coerce')

transactions = df.groupby(['InvoiceNo'])['StockCode'].apply(list).values.tolist()

te = TransactionEncoder()

te\_ary = te.fit\_transform(transactions)

tid = pd.DataFrame(te\_ary, columns=te.columns\_)

items = tid.astype('int')

min\_sup = 0.03

frequent\_itemsets = apriori(items, min\_support=min\_sup, use\_colnames=True)

rules = association\_rules(frequent\_itemsets, metric="confidence", min\_threshold=0.7)

print('Min support:', min\_sup)

print('Frequent itemsets:')

print(frequent\_itemsets)

print('Association rules:')

print(rules)

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 pandas as pd

from mlxtend.preprocessing import TransactionEncoder

from mlxtend.frequent\_patterns import apriori, association\_rules

df = pd.read\_csv('Downloads/archive (3)/Groceries\_dataset.csv')

print(df.info())

df.dropna(inplace=True) # Drop rows with missing values

df = df.apply(lambda x: pd.factorize(x)[0]) # Convert categorical values to numeric format

te = TransactionEncoder()

te\_ary = te.fit\_transform(df.values)

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

frequent\_itemsets = apriori(df, min\_support=0.01, use\_colnames=True)

association\_rules = association\_rules(frequent\_itemsets, metric="confidence", min\_threshold=0.5)

print("Frequent Itemsets:")

print(frequent\_itemsets)

print("Association Rules:")

print(association\_rules)

slip 9

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

import pandas as pd

from mlxtend.preprocessing import TransactionEncoder

from mlxtend.frequent\_patterns import apriori, association\_rules

dataset = [['beer', 'chips', 'soda'],

['beer', 'soda', 'pizza', 'wings'],

['soda', 'pizza'],

['beer', 'chips', 'soda', 'pizza'],

['beer', 'chips', 'wings'],

['chips', 'soda', 'pizza']]

df = pd.DataFrame(dataset)

print("Dataset information:")

print(df.info())

te = TransactionEncoder()

te\_ary = te.fit\_transform(df.values)

tid = pd.DataFrame(te\_ary, columns=te.columns\_)

items = tid.astype('int')

min\_sup = 0.5

frequent\_itemsets = apriori(items, min\_support=min\_sup, use\_colnames=True)

rules = association\_rules(frequent\_itemsets, metric="confidence", min\_threshold=0.5)

print('Min support:', min\_sup)

print('Frequent itemsets:')

print(frequent\_itemsets)

print('Association rules:')

print(rules)

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 values.

import pandas as pd

from mlxtend.preprocessing import TransactionEncoder

from mlxtend.frequent\_patterns import apriori, association\_rules

data = [['Milk', 'Egg', 'Bread'],

['Milk', 'Bread'],

['Milk', 'Egg', 'Bread', 'Cheese'],

['Milk', 'Egg'],

['Bread', 'Cheese']]

te = TransactionEncoder()

te\_ary = te.fit\_transform(data)

tid = pd.DataFrame(te\_ary, columns=te.columns\_)

items = tid.astype('int')

items = items.replace({True: 1, False: 0})

min\_sup\_values = [0.4, 0.6, 0.8]

for min\_sup in min\_sup\_values:

frequent\_itemsets = apriori(items, min\_support=min\_sup, use\_colnames=True)

rules = association\_rules(frequent\_itemsets, metric="confidence", min\_threshold=0.7)

print('Min support:', min\_sup)

print('Frequent itemsets:')

print(frequent\_itemsets)

print('Association rules:')

print(rules)

print()

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, association\_rules

dataset = [['tata', 'nexon', '2017'],

['MG', 'astor', '2021'],

['KIA', 'seltos', '2019'],

['hyundai', 'creta', '2015']

]

te = TransactionEncoder()

te\_ary = te.fit\_transform(dataset)

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

for min\_sup in [0.2, 0.4]:

frequent\_itemsets = apriori(df, min\_support=min\_sup, use\_colnames=True)

association\_rules = association\_rules(frequent\_itemsets, metric="confidence", min\_threshold=0.5)

# Display the results

print(f"Frequent Itemsets with minimum support of {min\_sup}:")

print(frequent\_itemsets)

print(f"Association Rules with minimum support of {min\_sup}:")

print(association\_rules)

slip 16

import re

import nltk

from nltk.corpus import stopwords

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

from heapq import nlargest

text = '''Extractive summarization is a process that involves automatically selecting sentences from a text

document to create a shorter version that conveys the most important information. It is commonly used in news articles,

scientific papers, and other types of text documents. The main advantage of extractive summarization is that it preserves

the original wording and context of the text, making it easier for readers to understand the main points. In this program,

we will demonstrate how to perform extractive summarization on a text paragraph using Python and NLTK.'''

text = re.sub(r'\d+', '', text)

text = re.sub(r'[^\w\s]', '', text)

sentences = sent\_tokenize(text)

words = word\_tokenize(text.lower())

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

word\_frequencies = {}

for word in words:

if word not in stop\_words:

if word not in word\_frequencies:

word\_frequencies[word] = 1

else:

word\_frequencies[word] += 1

sentence\_scores = {}

for sentence in sentences:

for word in word\_tokenize(sentence.lower()):

if word in word\_frequencies:

if len(sentence.split(' ')) < 30: # Exclude long sentences

if sentence not in sentence\_scores:

sentence\_scores[sentence] = word\_frequencies[word]

else:

sentence\_scores[sentence] += word\_frequencies[word]

summary\_sentences = nlargest(2, sentence\_scores, key=sentence\_scores.get)

summary = ' '.join(summary\_sentences)

print("Original text:\n", text)

print("\nSummary:\n", 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 re

import nltk

from nltk.corpus import stopwords

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

from heapq import nlargest

# Sample text paragraph

text = '''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

text = re.sub(r'\d+', '', text) # Remove digits

text = re.sub(r'[^\w\s]', '', text) # Remove special characters

sentences = sent\_tokenize(text) # Tokenize into sentences

words = word\_tokenize(text.lower()) # Tokenize into words and convert to lowercase

stop\_words = set(stopwords.words('english')) # Load English stop words

word\_frequencies = {} # Calculate word frequencies

for word in words:

if word not in stop\_words:

if word not in word\_frequencies:

word\_frequencies[word] = 1

else:

word\_frequencies[word] += 1

# Calculate sentence scores

sentence\_scores = {}

for sentence in sentences:

for word in word\_tokenize(sentence.lower()):

if word in word\_frequencies:

if len(sentence.split(' ')) < 30: # Exclude long sentences

if sentence not in sentence\_scores:

sentence\_scores[sentence] = word\_frequencies[word]

else:

sentence\_scores[sentence] += word\_frequencies[word]

# Generate summary

summary\_sentences = nlargest(2, sentence\_scores, key=sentence\_scores.get)

summary = ' '.join(summary\_sentences)

# Display the results

print("Original text:\n", text)

print("\nSummary:\n", summary)

slip 24

consider the following dataset : https://www.kaggle.com/datasets/datasnaek/youtubenew?select=INvideos.csv

Write a Python script for the following :

i. Read the dataset and perform data cleaning operations on it.

ii. Find the total views, total likes, total dislikes and comment count.

import pandas as pd

# Read the dataset

df = pd.read\_csv('INvideos.csv')

# Drop unnecessary columns

df.drop(['video\_id', 'trending\_date', 'title', 'channel\_title', 'category\_id', 'publish\_time', 'tags', 'thumbnail\_link', 'comments\_disabled', 'ratings\_disabled', 'video\_error\_or\_removed'], axis=1, inplace=True)

# Drop duplicate rows

df.drop\_duplicates(inplace=True)

# Drop rows with missing values

df.dropna(inplace=True)

# Convert the data types of relevant columns

df['views'] = pd.to\_numeric(df['views'])

df['likes'] = pd.to\_numeric(df['likes'])

df['dislikes'] = pd.to\_numeric(df['dislikes'])

df['comment\_count'] = pd.to\_numeric(df['comment\_count'])

# Calculate total views, likes, dislikes, and comment count

total\_views = df['views'].sum()

total\_likes = df['likes'].sum()

total\_dislikes = df['dislikes'].sum()

total\_comment\_count = df['comment\_count'].sum()

# Display the results

print('Total views:', total\_views)

print('Total likes:', total\_likes)

print('Total dislikes:', total\_dislikes)

print('Total comment count:', total\_comment\_count)

slip 25

consider the following dataset : https://www.kaggle.com/datasets/seungguini/youtube-commentsfor-covid19-relatedvideos?select=covid\_2021\_1.csv

Write a Python script for the following :

i. Read the dataset and perform data cleaning operations on it.

ii. Tokenize the comments in words. iii. Perform sentiment analysis and find the percentage of

positive, negative and neutral comments..

import pandas as pd

import re

import nltk

from nltk.sentiment import SentimentIntensityAnalyzer

# Read the dataset

df = pd.read\_csv('Downloads/archive (4)/covid\_2021\_1.csv')

# Drop unnecessary columns

df.drop(['comment\_id', 'author', 'comment\_date'], axis=1, inplace=True)

# Drop duplicate rows

df.drop\_duplicates(inplace=True)

# Drop rows with missing values

df.dropna(inplace=True)

# Tokenize comments into words

df['comment'] = df['comment'].apply(lambda x: re.findall(r'\b\w+\b', x.lower()))

# Perform sentiment analysis

nltk.download('vader\_lexicon')

sia = SentimentIntensityAnalyzer()

df['sentiment\_score'] = df['comment'].apply(lambda x: sia.polarity\_scores(' '.join(x))['compound'])

# Classify comments as positive, negative, or neutral

df['sentiment'] = df['sentiment\_score'].apply(lambda x: 'positive' if x > 0 else 'negative' if x < 0 else 'neutral')

# Calculate percentage of positive, negative, and neutral comments

positive\_percent = (df['sentiment'] == 'positive').sum() / len(df) \* 100

negative\_percent = (df['sentiment'] == 'negative').sum() / len(df) \* 100

neutral\_percent = (df['sentiment'] == 'neutral').sum() / len(df) \* 100

# Display the results

print('Percentage of positive comments:', positive\_percent)

print('Percentage of negative comments:', negative\_percent)

print('Percentage of neutral comments:', neutral\_percent)

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.

from mlxtend.preprocessing import TransactionEncoder

from mlxtend.frequent\_patterns import apriori, association\_rules

# create dataset

transactions = [['eggs', 'milk', 'bread'],

['eggs', 'apple'],

['milk', 'bread'],

['apple', 'milk'],

['milk', 'apple', 'bread']]

# convert categorical values to numeric format

te = TransactionEncoder()

te\_ary = te.fit(transactions).transform(transactions)

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

# apply apriori algorithm to generate frequent itemsets

frequent\_itemsets = apriori(df, min\_support=0.4, use\_colnames=True)

# generate association rules

rules = association\_rules(frequent\_itemsets, metric='lift', min\_threshold=1)

# print results

print("Frequent Itemsets:\n", frequent\_itemsets)

print("\nAssociation Rules:\n", rules)

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 nltk

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize

# Sample text paragraph

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."""

# Tokenize into words

words = word\_tokenize(text.lower())

# Load English stop words

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

# Remove stop words

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

# Join the filtered words into a string

filtered\_text = ' '.join(filtered\_words)

# Display the results

print("Original text:\n", text)

print("\nText after removing stopwords:\n", filtered\_text)