Q SLIP 1

```
# 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
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

# Creating Position_Salaries dataset
positions = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]).reshape(-1, 1)
salaries = np.array([45000, 50000, 60000, 80000, 110000, 150000, 200000, 300000, 500000, 1000000])

# Combine positions and salaries into a DataFrame
data = pd.DataFrame(data=np.concatenate((positions, salaries.reshape(-1,1)), axis=1), columns=['Position', 'Salary'])

# Identify independent and target variables
X = data[['Position']]  # Independent variable
y = data['Salary']      # Target variable

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Print the training and testing sets
print("Training Set:")
print(X_train)
print(y_train)
print("\nTesting Set:")
print(X_test)
print(y_test)

# Build a simple linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Print the coefficients of the linear regression model
print("Coefficients:", model.coef_)
print("Intercept:", model.intercept_)
```

```
# Slip 2
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

# Generate synthetic data
years_of_experience = np.array([1, 2, 3, 4, 5]).reshape(-1, 1)  # Independent variable
salary = np.array([30000, 35000, 40000, 45000, 50000])  # Target variable

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(years_of_experience, salary, test_size=0.2,
random_state=42)

# Print the shapes of training and testing sets
print("Training set shapes:")
print("X_train:", X_train.shape)
print("y_train:", y_train.shape)
print("\nTesting set shapes:")
print("X_test:", X_test.shape)
print("y_test:", y_test.shape)

# Build and train the linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Predictions on the testing set
y_pred = model.predict(X_test)

# Print the coefficients
print("\nCoefficients:", model.coef_)
print("Intercept:", model.intercept_)
```

QSlip 3

```
# 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
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Creating a User dataset
data = {
    'User ID': [1, 2, 3, 4, 5],
    'Gender': ['Male', 'Female', 'Male', 'Female', 'Male'],
    'Age': [25, 30, 35, 40, 45],
    'Estimated Salary': [50000, 60000, 70000, 80000, 90000],
    'Purchased': [0, 1, 0, 1, 0]  # 0 for not purchased, 1 for purchased
}

df = pd.DataFrame(data)

# Convert Gender column to numeric using one-hot encoding
df = pd.get_dummies(df, columns=['Gender'])

# Splitting dataset into features and target variable
X = df.drop(['User ID', 'Purchased'], axis=1)
y = df['Purchased']

# Splitting dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Feature scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Building logistic regression model
model = LogisticRegression()
model.fit(X_train_scaled, y_train)

# Predictions
y_pred = model.predict(X_test_scaled)

# Evaluating the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
```

```
class_report = classification_report(y_test, y_pred)

print("Accuracy:", accuracy)
print("\nConfusion Matrix:")
print(conf_matrix)
print("\nClassification Report:")
print(class_report)
```

QSlip 4

```
# Build a simple linear regression model for Fish Species Weight Prediction.
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt

# Load the dataset
fish_data = pd.read_csv('fish.csv')

# Let's consider only the 'Length1' column as our independent variable and the 'Weight'
column as our target variable
X = fish_data[['Length']].values
y = fish_data['Weight'].values

# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Building the linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Making predictions
y_pred = model.predict(X_test)

# Plotting the results
plt.scatter(X_test, y_test, color='blue')
plt.plot(X_test, y_pred, color='red')
plt.title('Fish Species Weight Prediction')
plt.xlabel('Length')
plt.ylabel('Weight')
plt.show()
```

Q 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' and '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 pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Load the Iris dataset
iris_data = pd.read_csv('iris.csv')

# Display basic statistical details for each species
species_data = iris_data.groupby('variety')
print(species_data.describe())

# Splitting the dataset into features and target
X = iris_data.drop('variety', axis=1)
y = iris_data['variety']

# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Creating and training the logistic regression model
model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)

# Predicting the species using the test data
y_pred = model.predict(X_test)

# Calculating the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print("\nAccuracy of the logistic regression model:", accuracy)
```

QSlip 6

```
# 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.frequent_patterns import apriori, association_rules

# Create the dataset
data = {
    'Tid': [1, 2, 3, 4, 5],
    'Items': [['Bread', 'Milk'],
              ['Bread', 'Diaper', 'Beer', 'Eggs'],
              ['Milk', 'Diaper', 'Beer', 'Coke'],
              ['Bread', 'Milk', 'Diaper', 'Beer'],
              ['Bread', 'Milk', 'Diaper', 'Coke']]
}

# Convert to DataFrame
df = pd.DataFrame(data)

# One-hot encode the items
df_encoded = df['Items'].str.join('|').str.get_dummies()

# Add Tid column
df_encoded['Tid'] = df['Tid']

# Function to apply Apriori algorithm and display results
def apply_apriori_and_display(min_sup):
    frequent_itemsets = apriori(df_encoded.drop('Tid', axis=1), min_support=min_sup,
use_colnames=True)
    association_rules_df = association_rules(frequent_itemsets, metric='confidence',
min_threshold=0.7)
    print("\nFrequent Itemsets with min_support =", min_sup)
    print(frequent_itemsets)
    print("\nAssociation Rules with min_support =", min_sup)
    print(association_rules_df)

# Test with different min_sup values
apply_apriori_and_display(0.2)
apply_apriori_and_display(0.4)
```

QSlip 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.frequent_patterns import apriori, association_rules

# Read the dataset
dataset = pd.read_csv('Market_Basket_Optimisation.csv')

# Display dataset information
print("Dataset Information:")
print(dataset.info())

# Preprocess the data (drop null values)
dataset.dropna(inplace=True)

# Convert categorical values into numeric format
# Assuming dataset is already in a transaction format, so we just need to one-hot encode
encoded_dataset = pd.get_dummies(dataset)

# Adjust minimum support value
min_support = 0.01  # Adjust as needed

# Apply Apriori algorithm to generate frequent itemsets
frequent_itemsets = apriori(encoded_dataset, min_support=min_support,
use_colnames=True)

# Check if frequent itemsets are empty
if frequent_itemsets.empty:
   print("Error: No frequent itemsets found with the given minimum support.")
else:
   # Generate association rules
   association_rules_df = association_rules(frequent_itemsets, metric='confidence',
min_threshold=0.9)

   # Display frequent itemsets
   print("\nFrequent Itemsets:")
   print(frequent_itemsets)

   # Display association rules
   print("\nAssociation Rules:")
   print(association_rules_df)
```

Q Slip 8

```
# 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.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules

# Read the dataset
df = pd.read_csv('groceries.csv',on_bad_lines='skip')

# Display dataset information
print("Dataset information:")
print(df.info())

# Preprocessing
# Drop null values
df.dropna(inplace=True)

# Convert categorical values into numeric format using one-hot encoding
df = pd.get_dummies(df)

# Apriori algorithm
frequent_itemsets = apriori(df, min_support=0.01, use_colnames=True)

# Generate association rules
rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1)

# Display frequent itemsets
print("\nFrequent Itemsets:")
print(frequent_itemsets)

# Display association rules
print("\nAssociation Rules:")
print(rules)
```

Q slip 9

```
# 9)Create your own transactions dataset and apply the above process on your dataset
import pandas as pd
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules

# Create transactions dataset
transactions = [
    ['bread', 'milk', 'eggs'],
    ['bread', 'butter', 'eggs', 'jam'],
    ['bread', 'milk', 'butter'],
    ['milk', 'butter', 'jam'],
    ['bread', 'milk', 'eggs', 'butter']
]

# Convert transactions to DataFrame
df = pd.DataFrame(transactions)

# Display dataset information
print("Dataset information:")
print(df.info())

# Preprocessing
# Convert categorical values into numeric format using one-hot encoding
df_encoded = pd.get_dummies(df)

# Apriori algorithm
frequent_itemsets = apriori(df_encoded, min_support=0.2, use_colnames=True)

# Generate association rules
rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1)

# Display frequent itemsets
print("\nFrequent Itemsets:")
print(frequent_itemsets)

# Display association rules
print("\nAssociation Rules:")
print(rules)
```

Q slip 10

```
# 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 associationrules. Repeat
# the process with different min_sup values.
import pandas as pd
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules

# Create the dataset
data = {
    'TID': [1, 2, 3, 4, 5],
    'Items': [
        'butter, bread, milk',
        'butter, flour, milk, sugar',
        'butter, eggs, milk, salt',
        'eggs',
        'butter, flour, milk, salt'
    ]
}

# Convert the dictionary to a DataFrame
df = pd.DataFrame(data)

# Convert categorical values into numeric format using one-hot encoding
df_encoded = df['Items'].str.get_dummies(', ')

# Apriori algorithm with different min_sup values
min_sup_values = [0.2, 0.3, 0.4]

for min_sup in min_sup_values:
    print(f"\nMinimum Support: {min_sup}\n")

    # Apriori algorithm
    frequent_itemsets = apriori(df_encoded, min_support=min_sup, use_colnames=True)

    # Generate association rules
    rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1)

    # Display frequent itemsets
    print("Frequent Itemsets:")
    print(frequent_itemsets)

    # Display association rules
    print("\nAssociation Rules:")
    print(rules)
```

Q Slip 11

```
# 11)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 pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

# Create a simple dataset using dictionaries
data = {
    'heights': [150, 155, 160, 165, 170, 175, 180, 185, 190, 195],
    'weights': [55, 58, 62, 65, 70, 75, 80, 85, 90, 95]
}

# Convert the dictionary to a DataFrame
heights_and_weights = pd.DataFrame(data)

# Splitting the dataset into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(heights_and_weights[['heights']],
heights_and_weights['weights'], test_size=0.2, random_state=42)

# Printing the training and testing sets
print("Training Set - Independent Variable (heights):")
print(X_train)
print("\nTraining Set - Target Variable (weights):")
print(y_train)
print("\nTesting Set - Independent Variable (heights):")
print(X_test)
print("\nTesting Set - Target Variable (weights):")
print(y_test)

# Building a simple linear regression model
linear_reg_model = LinearRegression()

# Fitting the model
linear_reg_model.fit(X_train, y_train)

# Printing the coefficients
print("\nModel Coefficients:")
print("Intercept:", linear_reg_model.intercept_)
print("Coefficient:", linear_reg_model.coef_[0])
```

Q Slip 13
13)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. [Marks 15]

```
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori, association_rules

# Create the dataset
data = {
    'TID': [1, 2, 3, 4],
    'Items': [
        {'Apple', 'Mango', 'Banana'},
        {'Mango', 'Banana', 'Cabbage', 'Carrots'},
        {'Mango', 'Banana', 'Carrots'},
        {'Mango', 'Carrots'}
    ]
}

# Convert the dictionary to a DataFrame
df = pd.DataFrame(data)

# Convert categorical values into numeric format using one-hot encoding
te = TransactionEncoder()
encoded_data = te.fit_transform(df['Items'])
df_encoded = pd.DataFrame(encoded_data, columns=te.columns_)

# Apply the Apriori algorithm with different min_sup values
min_sup_values = [0.2, 0.3]

for min_sup in min_sup_values:
    print(f"\nMinimum Support: {min_sup}\n")

    # Apriori algorithm
    frequent_itemsets = apriori(df_encoded, min_support=min_sup, use_colnames=True)

    # Generate association rules
    rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1)

    # Display frequent itemsets
    print("Frequent Itemsets:")
    print(frequent_itemsets)

    # Display association rules
    print("\nAssociation Rules:")
    print(rules)
```

Q slip 14

```
# 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, association_rules

# Create the dataset
data = {
    'Company': ['Tata', 'MG', 'Kia', 'Hyundai'],
    'Model': ['Nexon', 'Astor', 'Seltos', 'Creta'],
    'Year': [2017, 2021, 2019, 2015]
}

# Convert the dictionary to a DataFrame
df = pd.DataFrame(data)

# Convert categorical values into numeric format using one-hot encoding
te = TransactionEncoder()
encoded_data = te.fit_transform(df.apply(lambda x: x.astype(str), axis=1))
df_encoded = pd.DataFrame(encoded_data, columns=te.columns_)

# Apply the Apriori algorithm with different min_sup values
min_sup_values = [0.2, 0.3]

for min_sup in min_sup_values:
    print(f"\nMinimum Support: {min_sup}\n")

    # Apriori algorithm
    frequent_itemsets = apriori(df_encoded, min_support=min_sup, use_colnames=True)

    # Generate association rules
    rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1)

    # Display frequent itemsets
    print("Frequent Itemsets:")
    print(frequent_itemsets)

    # Display association rules
    print("\nAssociation Rules:")
    print(rules)
```

Q Slip 15

```
# 15)Consider any text paragraph. Preprocess the text to remove any special characters and digits.
# Generate the summary using extractive summarization process
from sumy.parsers.plaintext import PlaintextParser
from sumy.nlp.tokenizers import Tokenizer
from sumy.summarizers.lsa import LsaSummarizer

# Sample text paragraph
text = """
Natural language processing (NLP) is a subfield of linguistics, computer science, and
artificial intelligence concerned with the interactions between computers
and human language, in particular how to program computers to process and analyze large
amounts of natural language data. The result is a computer capable
of "understanding" the contents of documents, including the contextual nuances of the
language within them.The history of natural language processing
generally started in the 1950s, although work can be found from earlier periods. In 1950,
Alan Turing published an article titled "Computing Machinery and Intelligence"
which proposed what is now called the Turing test as a criterion of intelligence.
The general methodology of NLP divides into two steps: understanding and generating
human language. Understanding includes understanding the semantics,
i.e., meaning, of the text, and includes tasks such as sentiment analysis, named entity
recognition, and topic modeling. Generating human language includes
tasks such as machine translation, text summarization, and question answering.
"""

# Initialize the summarizer
parser = PlaintextParser.from_string(text, Tokenizer('english'))
summarizer = LsaSummarizer()

# Generate the summary
summary = summarizer(parser.document, sentences_count=3)  # Adjust sentences_count as needed for summary length

# Print the summary
for sentence in summary:
    print(sentence)
```

Q Slip 16
16)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.

```
from sumy.parsers.plaintext import PlaintextParser
from sumy.nlp.tokenizers import Tokenizer
from sumy.summarizers.lsa import LsaSummarizer

# 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 to remove special characters and digits
preprocessed_text = ''.join(char for char in text if char.isalnum() or char in [' ', '\n'])

# Initialize the summarizer
parser = PlaintextParser.from_string(preprocessed_text, Tokenizer('english'))
summarizer = LsaSummarizer()

# Generate the summary
summary = summarizer(parser.document, sentences_count=2)  # Adjust sentences_count
as needed for summary length

# Print the summary
for sentence in summary:
    print(sentence)
```

Q Slip 17
17)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 matplotlib.pyplot as plt
from wordcloud import WordCloud
from nltk.tokenize import word_tokenize, sent_tokenize
from nltk.corpus import stopwords
from nltk.probability import FreqDist

# Sample text paragraph
text = """
Natural language processing (NLP) is a subfield of linguistics, computer science, and
artificial intelligence concerned with the interactions between computers and human
language, in particular how to program computers to process and analyze large amounts of
natural language data. The result is a computer capable of "understanding" the contents of
documents, including the contextual nuances of the language within them.

The history of natural language processing generally started in the 1950s, although work can
be found from earlier periods. In 1950, Alan Turing published an article titled "Computing
Machinery and Intelligence" which proposed what is now called the Turing test as a criterion
of intelligence.

The general methodology of NLP divides into two steps: understanding and generating
human language. Understanding includes understanding the semantics, i.e., meaning, of the
text, and includes tasks such as sentiment analysis, named entity recognition, and topic
modeling. Generating human language includes tasks such as machine translation, text
summarization, and question answering.
"""

# Remove stopwords
stop_words = set(stopwords.words('english'))
words = word_tokenize(text)
filtered_words = [word for word in words if word.lower() not in stop_words]

# Tokenize sentences
sentences = sent_tokenize(text)

# Calculate word frequency distribution
fdist = FreqDist(filtered_words)

# Plot word frequency distribution
plt.figure(figsize=(10, 5))
fdist.plot(20, cumulative=False)
plt.title('Word Frequency Distribution')
plt.xlabel('Words')
```

```
plt.ylabel('Frequency')
plt.show()

# Generate and plot word cloud
wordcloud = WordCloud(width=800, height=400, background_color='white').generate('
'.join(filtered_words))

plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.title('Word Cloud')
plt.axis('off')
plt.show()
```

Q Slip 18

```
# Q.18)Download the movie_review.csv dataset from Kaggle by using the following link
# :https://www.kaggle.com/nltkdata/movie-review/version/3?select=movie_review.csv to
perform
# sentiment analysis on above dataset and create a wordcloud.
import pandas as pd
import matplotlib.pyplot as plt
from wordcloud import WordCloud
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

# Load the IMDB dataset
df = pd.read_csv('IMDB_Dataset.csv')

# Preprocess text
def preprocess_text(text):
    stop_words = set(stopwords.words('english'))
    words = word_tokenize(text)
    filtered_words = [word.lower() for word in words if word.isalpha() and word.lower() not in
stop_words]
    return ' '.join(filtered_words)

# Apply preprocessing to the entire dataset
df['review'] = df['review'].apply(preprocess_text)

# Join all reviews
all_reviews = ' '.join(df['review'])

# Generate word cloud
wordcloud = WordCloud(width=800, height=400,
background_color='white').generate(all_reviews)

# Plot word cloud
plt.figure(figsize=(10, 6))
```

```
plt.imshow(wordcloud, interpolation='bilinear')
plt.title('Word Cloud of Movie Reviews')
plt.axis('off')
plt.show()
```

Q Slip 19

```
# Q.19)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 the text
words = word_tokenize(text)

# Remove stopwords
stop_words = set(stopwords.words('english'))
filtered_words = [word for word in words if word.lower() not in stop_words]

# Join the filtered words to form a sentence
filtered_sentence = ' '.join(filtered_words)

# Print the filtered text
print("Text after removing stopwords:")
print(filtered_sentence)

# Troubleshooting prints
print("\nNumber of words before removing stopwords:", len(words))
print("Number of words after removing stopwords:", len(filtered_words))
```

Q Slip 20

```
# 20)Build a simple linear regression model for User Data.
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

# Load the dataset
df = pd.read_csv('User_Data.csv')

# Split the dataset into features (X) and target variable (y)
X = df[['Age']]  # Feature: Age
y = df['EstimatedSalary']  # Target variable: Income

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the linear regression model
model = LinearRegression()

# Fit the model on the training data
model.fit(X_train, y_train)

# Predict on the testing data
y_pred = model.predict(X_test)

# Calculate the mean squared error
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)

# Print the coefficients
print("Intercept:", model.intercept_)
print("Coefficient:", model.coef_)
```

Q Slip 21

```
# 21)Consider any text paragraph. 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 the text
words = word_tokenize(text)
```

```
# Remove stopwords
stop_words = set(stopwords.words('english'))
filtered_words = [word for word in words if word.lower() not in stop_words]

# Join the filtered words to form a sentence
filtered_sentence = ' '.join(filtered_words)

# Print the filtered text
print("Text after removing stopwords:")
print(filtered_sentence)

# Troubleshooting prints
print("\nNumber of words before removing stopwords:", len(words))
print("Number of words after removing stopwords:", len(filtered_words))
```

Q Slip 22

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

from sumy.parsers.plaintext import PlaintextParser
from sumy.nlp.tokenizers import Tokenizer
from sumy.summarizers.lsa import LsaSummarizer

# Sample text paragraph
text = """
Natural language processing (NLP) is a subfield of linguistics, computer science, and
artificial intelligence concerned with the interactions between computers
and human language, in particular how to program computers to process and analyze large
amounts of natural language data. The result is a computer capable
of "understanding" the contents of documents, including the contextual nuances of the
language within them.The history of natural language processing
generally started in the 1950s, although work can be found from earlier periods. In 1950,
Alan Turing published an article titled "Computing Machinery and Intelligence"
which proposed what is now called the Turing test as a criterion of intelligence.
The general methodology of NLP divides into two steps: understanding and generating
human language. Understanding includes understanding the semantics,
i.e., meaning, of the text, and includes tasks such as sentiment analysis, named entity
recognition, and topic modeling. Generating human language includes
tasks such as machine translation, text summarization, and question answering.
"""

# Initialize the summarizer
parser = PlaintextParser.from_string(text, Tokenizer('english'))
summarizer = LsaSummarizer()

# Generate the summary
```

```
summary = summarizer(parser.document, sentences_count=3)  # Adjust sentences_count
as needed for summary length

# Print the summary
for sentence in summary:
    print(sentence)
```

Q Slip 23

```
# Q.23)Consider the following dataset : https://www.kaggle.com/datasets/datasnaek/youtube
# new?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

# i. Read the dataset and perform data cleaning operations
# Load the dataset
df = pd.read_csv('USvideos.csv', on_bad_lines = 'skip')

# Display basic information about the dataset
print("Dataset information:")
print(df.info())

# Check for missing values
print("\nMissing values:")
print(df.isnull().sum())

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

# Check for duplicates
print("\nDuplicate rows:")
print(df.duplicated().sum())

# Remove duplicate rows
df.drop_duplicates(inplace=True)

# ii. Find the total views, total likes, total dislikes, and comment count
total_views = df['views'].sum()
total_likes = df['likes'].sum()
total_dislikes = df['dislikes'].sum()
total_comments = df['comment_total'].sum()

print("\nTotal views:", total_views)
print("Total likes:", total_likes)
print("Total dislikes:", total_dislikes)
print("Total comment count:", total_comments)
```

Q Slip 24

```
# Q.24)Consider the following dataset :
https://www.kaggle.com/datasets/seungguini/youtube-comments
# for-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. ii. Tokenize the comments in words. iii. Perform sentiment analysis and find the
percentage of
# positive, negative and neutral comments..
import pandas as pd
from textblob import TextBlob

# Read the dataset
df = pd.read_csv('covid_2021_1.csv')

# Display basic information about the dataset
print("Dataset information:")
print(df.info())

# Check for missing values
print("\nMissing values:")
print(df.isnull().sum())

# Clean the dataset (remove missing values)
df.dropna(inplace=True)

# Tokenize comments into words
df['tokenized_comments'] = df['comment_text'].apply(lambda x: x.split())

# Perform sentiment analysis and find percentages
positive_comments = sum(TextBlob(comment).sentiment.polarity > 0 for comment in
df['comment_text'])
negative_comments = sum(TextBlob(comment).sentiment.polarity < 0 for comment in
df['comment_text'])
neutral_comments = sum(TextBlob(comment).sentiment.polarity == 0 for comment in
df['comment_text'])

total_comments = len(df)
positive_percentage = (positive_comments / total_comments) * 100
negative_percentage = (negative_comments / total_comments) * 100
neutral_percentage = (neutral_comments / total_comments) * 100

# Print results
print("\nSentiment Analysis Results:")
print("Total comments:", total_comments)
print("Positive comments:", positive_comments, f"({positive_percentage:.2f}%)")
print("Negative comments:", negative_comments, f"({negative_percentage:.2f}%)")
print("Neutral comments:", neutral_comments, f"({neutral_percentage:.2f}%)")
```

Q Slip 25

```
# Q.25)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 re
from nltk.tokenize import sent_tokenize
from sklearn.feature_extraction.text import TfidfVectorizer

def preprocess_text(text):
   # Remove special characters and digits
   text = re.sub(r'[^a-zA-Z\s]', '', text)
   return text

def generate_summary(text, num_sentences=2):
   # Tokenize the text into sentences
   sentences = sent_tokenize(text)

   # Preprocess each sentence
   preprocessed_sentences = [preprocess_text(sentence) for sentence in sentences]

   # Vectorize the sentences using TF-IDF
   vectorizer = TfidfVectorizer().fit(preprocessed_sentences)
   vectors = vectorizer.transform(preprocessed_sentences)

   # Calculate pairwise cosine similarity
   similarity_matrix = (vectors * vectors.T).A

   # Rank sentences based on similarity
   ranking = similarity_matrix.sum(axis=1)
   ranked_sentences = [sentence for _, sentence in sorted(zip(ranking, sentences),
reverse=True)]

   # Select top sentences for summary
   summary = ranked_sentences[:num_sentences]

   return ' '.join(summary)

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."""

preprocessed_text = preprocess_text(text)
summary = generate_summary(preprocessed_text)
```

```
print("Original Text:")
print(text)
print("\nPreprocessed Text:")
print(preprocessed_text)
print("\nSummary:")
print(summary)
```

Q Slip 27

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

# Load the dataset
car_data = pd.read_csv("cars.csv")

# Let's assume we have columns 'horsepower' as independent variable and 'price' as
dependent variable
X = car_data[['hp']]  # Independent variable
y = car_data['disp']          # Dependent variable

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the linear regression model
model = LinearRegression()

# Train the model
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Calculate Mean Squared Error
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)

# Coefficients and Intercept
print("Coefficients:", model.coef_)
print("Intercept:", model.intercept_)
```

Q Slip 28

```
# Build a logistic regression model for Student Score Dataset.
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Load the dataset
student_data = pd.read_csv("Hours and Scores.csv")

# Assuming 'hours' and 'pass' are the features
X = student_data[['Hours']]  # Independent variable
y = student_data['Scores']     # Dependent variable

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the logistic regression model
model = LogisticRegression()

# Train the model
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)

# Classification report
class_report = classification_report(y_test, y_pred)
print("Classification Report:")
print(class_report)
```

Q Slip 29

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

# Define the dataset
transactions = [['eggs', 'milk', 'bread'],
          ['eggs', 'apple'],
          ['milk', 'bread'],
          ['apple', 'milk'],
          ['milk', 'apple', 'bread']]

# Initialize TransactionEncoder
encoder = TransactionEncoder()

# One-hot encode the dataset
onehot = encoder.fit(transactions).transform(transactions)

# Convert one-hot encoded data to DataFrame
df = pd.DataFrame(onehot, columns=encoder.columns_)

# Apply Apriori algorithm to find frequent itemsets
frequent_itemsets = apriori(df, min_support=0.5, use_colnames=True)

# Generate association rules
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.7)

# Print frequent itemsets
print("Frequent Itemsets:")
print(frequent_itemsets)

# Print association rules
print("\nAssociation Rules:")
print(rules)
```