//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 mode

ANS:

```
#*******************slip 1************************
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
data={'position':['junior','senior','manager','director','ceo'],'level':[1,2,3,4,5],'salary':[50000,80000,12000,18000,25000]}
df=pd.DataFrame(data)
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)
print("training set")
print(x_train)
print(y_train)
print("\n Testing set")
print(x_test)
print(y_test)

model=LinearRegression()

model.fit(x_train,y_train)
```

//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

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression

from sklearn.metrics import r2_score, mean_squared_error


# Create 'Salary' dataset

np.random.seed(0)

n_samples = 1000

age = np.random.randint(20, 65, size=n_samples)

experience = np.random.randint(1, 40, size=n_samples)

education_level = np.random.randint(0, 5, size=n_samples)  # Assuming 5 levels of education

salary = 30000 + (age * 1000) + (experience * 500) + (education_level * 2000) + np.random.randint(-5000, 5000, size=n_samples)
```

```
data = {'Age': age, 'Experience': experience, 'Education_Level': education_level, 'Salary': salary}
salary_data = pd.DataFrame(data)

# Identify independent and target variables
X = salary_data[['Age', 'Experience', 'Education_Level']]  # Independent variables
y = salary_data['Salary']                                   # Target variable

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)

# 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
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

```
# Print intercept and coefficients

print('Regressor intercept:', regressor.intercept_)

print('Regressor coefficients:', regressor.coef_)


# Predictions

y_pred = regressor.predict(X_test)


# Evaluate the model

print('R2 score:', r2_score(y_test, y_pred))

print('Mean Squared Error:', mean_squared_error(y_test, y_pred))
```

//SLIP 4:

//Build a simple linear regression model for Fish Species Weight Prediction.

```
#****************************slip 4************************

import pandas as pd

df=pd.read_csv(r'Fish.csv')

print(df)

X=df.drop(['Weight','Species'],axis=1)

y=df['Weight']

from sklearn.model_selection import train_test_split

X_train,X_test,y_train,ytest=train_test_split(X,y,test_size=0.33,random_state=
101)

from sklearn.linear_model import LinearRegression

lr=LinearRegression()
```

```
lr.fit(X_train,y_train)

LinearRegression(copy_X=True,fit_intercept=True,n_jobs=None)

lr.score(X_test,ytest)

df_t=df.copy()

df_t['predicted weight']=lr.predict(df_t.drop(['Weight','Species'],axis=1))

df_t['Difference']=df_t['Weight']-df_t['predicted weight']

df_t[['Weight','predicted weight','Difference']].head(20)
```

//SLIP 5

//2)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

```
#slip 5

import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import confusion_matrix, accuracy_score


# Load the Iris dataset

iris = pd.read_csv('iris.csv')  # Replace '/path/to/iris.csv' with the actual file path


# Prepare the input features and output variable
```

```
X = iris[['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm']]

y = iris['Species']


# Split the dataset into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=0)


# Initialize and train the logistic regression model

classifier = LogisticRegression(random_state=0, solver='lbfgs', multi_class='auto')

classifier.fit(X_train, y_train)


# Make predictions

y_pred = classifier.predict(X_test)


# Calculate accuracy

accuracy = accuracy_score(y_test, y_pred)

print("Accuracy of the logistic regression model:", accuracy)


# Generate confusion matrix

conf_matrix = confusion_matrix(y_test, y_pred)


# Print confusion matrix

print("\nConfusion Matrix:")
print(conf_matrix)
```

//SLIP 6

//2)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

```
#***************************slip 6************************

import pandas as pd

from mlxtend.frequent_patterns import apriori, association_rules

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

['bread', 'diaper', 'beer', 'eggs'],

['milk', 'diaper', 'beer', 'coke'],

['bread', 'milk', 'diaper', 'beer'],

['bread', 'milk', 'diaper', 'coke']]

from mlxtend.preprocessing import TransactionEncoder

te=TransactionEncoder()

te_array=te.fit(transactions).transform(transactions)

df=pd.DataFrame(te_array, columns=te.columns_)

df

freq_items = apriori(df, min_support = 0.5, use_colnames = True)

print(freq_items)

rules = association_rules(freq_items, metric ='support', min_threshold=0.05)

rules = rules.sort_values(['support', 'confidence'], ascending =[False,False])

print(rules)
```

//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

```
#*********************slip 7*************************
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from apyori import apriori
store_data = pd.read_csv(r"C:\Users\lenovo\Documents\Assignment 2/Market_Basket_Optimisation.csv",header=None)
display(store_data.head())
print(store_data.shape)
store_data.fillna(0,inplace=True)
store_data.head()
records = []
for i in range(1, 7501):
   records.append([str(store_data.values[i, j]) for j in range(0, 20)])
for i in range(0, len(association_results)):
   print(association_results[i][0])
for item in association_results:
   pair = item[0]
   items = [x for x in pair]
```

```
    print("Rule: " + items[0] + " -> " + items[1])

    print("Support: " + str(item[1]))

    print("Confidence: " + str(item[2][0][2]))

    print("Lift: " + str(item[2][0][3]))

    print("========================================")
```

//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.

```
#**********************************slip 8*****************************

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from mlxtend.frequent_patterns import apriori

from mlxtend.frequent_patterns import association_rules

from mlxtend.preprocessing import TransactionEncoder

df=pd.read_csv(r"C:\Users\lenovo\Documents/Groceries_dataset.csv")

print("\n -----Dataset-------\n",df.head(5))

df=df.dropna()

df.info()

df.itemDescription=df.itemDescription.transform(lambda x:[x])
```

```
df=df.groupby(['Member_number','Date']).sum()['itemDescription'].reset_index(drop=True)
encoder=TransactionEncoder()
transactions=pd.DataFrame(encoder.fit(df).transform(df),columns=encoder.columns_)
print("\n--------------Transaction Data--------------\n",transactions.head(5))
frequent_itemsets=apriori(transactions,min_support=6/len(df),use_colnames=True,max_len=2)
rules=association_rules(frequent_itemsets,metric="lift",min_threshold=1.5)
print("\n------Frequent Itemsets-----------\n",frequent_itemsets)
print("\n-------------Association Rules-----------\n",rules.head(5))
print("Rules identified:",len(rules))
```

//SLIP 9

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

```
#*********************slip 9*****************************
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules
transactions = [['Apple', 'Beer', 'Chicken', 'Rice'],
        ['Apple', 'Beer', 'Rice'],
        ['Apple', 'Beer'],
        ['Apple', 'Banana'],
        ['Beer', 'Chicken', 'Milk', 'Rice'],
        ['Beer', 'Milk', 'Rice'],
        ['Apple', 'Banana']]
```

```
from mlxtend.preprocessing import TransactionEncoder
te=TransactionEncoder()
te_array=te.fit(transactions).transform(transactions)
df=pd.DataFrame(te_array, columns=te.columns_)
df
freq_items = apriori(df, min_support = 0.5, use_colnames = True)
print(freq_items)
rules = association_rules(freq_items, metric ='support', min_threshold=0.05)
rules = rules.sort_values(['support', 'confidence'], ascending =[False,False])
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.

```
#*****************slip 10**********************
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules
transactions = [['eggs', 'milk','bread'],
['eggs',  'apple'],
['milk', 'bread'],
['apple', 'milk'],
['milk', 'apple', 'bread']]
from mlxtend.preprocessing import TransactionEncoder
```

```
te=TransactionEncoder()
te_array=te.fit(transactions).transform(transactions)
df=pd.DataFrame(te_array, columns=te.columns_)
df
freq_items = apriori(df, min_support = 0.5, use_colnames = True)
print(freq_items)
rules = association_rules(freq_items, metric ='support', min_threshold=0.05)
rules = rules.sort_values(['support', 'confidence'], ascending =[False,False])
print(rules)
```

//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 pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

# Load the dataset
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/nursery/nursery.data"
names = ['parents', 'has_nurs', 'form', 'children', 'housing', 'finance', 'social', 'health', 'class']
dataset = pd.read_csv(url, names=names)
```

```
# Convert categorical variables into numerical variables using one-hot encoding

dataset_encoded = pd.get_dummies(dataset)


# Print the column names to identify the target variable

print(dataset_encoded.columns)


# Identify the correct column name for the target variable

# Based on the column names provided, it seems that the target variable is 'class_recommend'

# Then, specify it as the target variable

X = dataset_encoded.drop('class_recommend', axis=1)  # Independent variables

y = dataset_encoded['class_recommend']  # Target variable


# Split 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)


# Build a simple linear regression model

model = LinearRegression()


# Train the model

model.fit(X_train, y_train)
```

```
# Evaluate the model

train_score = model.score(X_train, y_train)

test_score = model.score(X_test, y_test)


print("Train Score:", train_score)

print("Test Score:", test_score)


# Print the coefficients and intercept

print("Coefficients:", model.coef_)

print("Intercept:", model.intercept_)
```

//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.

```
#******************************slip 14***************************

import pandas as pd

from mlxtend.frequent_patterns import apriori, association_rules

transactions = [['apple', 'mango','banana'],

['mango',  'banana','cabbage','carrot'],

['mango', 'banana','carrot'],


['mango', 'carrot']]
```

```
from mlxtend.preprocessing import TransactionEncoder

te=TransactionEncoder()

te_array=te.fit(transactions).transform(transactions)

df=pd.DataFrame(te_array, columns=te.columns_)

df

freq_items = apriori(df, min_support = 0.3, use_colnames = True)

print(freq_items)

rules = association_rules(freq_items, metric ='support', min_threshold=0.05)

rules = rules.sort_values(['support', 'confidence'], ascending =[False,False])

print(rules)
```

//SLIP 16

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

```
#*******************************slip 16************************

import nltk

import re


text = """Hello all Students,Welcome to [Today's] Practical. Lets learn how to
perform text and social media analysis.there are total 3 sets and 10
questions."""

text =re.sub(r'[[0-9]{}*]',' ',text)

formatted_text = re.sub('[^a-zA-Z]',' ',text)

print("\nText after removing digits and special characters\n",formatted_text)
```

```
from nltk.corpus import stopwords

from nltk.tokenize import word_tokenize,sent_tokenize

stopWords = set(stopwords.words("english"))

print("\n\n\nStopWords are :\n",stopWords)

words = word_tokenize(formatted_text)

print("\n\n\n-------------------------Extractive Summarization --------------------")

print("\n\n\n------------------- Words in Text -------------\n",words)


wordfreq = {}

for word in words:

   if word in stopWords:

      continue

   if word in wordfreq:

      wordfreq[word] += 1

   else:

      wordfreq[word] = 1


maximum_frequency = max(wordfreq.values())


for word in wordfreq.keys():

   wordfreq[word] = (wordfreq[word]/maximum_frequency)


print("\n\n\n---------------- Word Frequencies -------------\n",wordfreq)
```

```
sentences = sent_tokenize(text)

sentenceValue = {}


for sentence in sentences:

    for word , freq in wordfreq.items():

        if word in sentence.lower():

            if sentence in sentenceValue:

                sentenceValue[sentence] += freq

            else:

                sentenceValue[sentence] = freq


import heapq

summaray = ''

summaray_sentences = heapq.nlargest(4, sentenceValue ,
key=sentenceValue.get)

summaray = ' '.join(summaray_sentences)

print("\n\n\n------------------ Summary Text ------------------\n",summaray)
```

//SLIP 17

//2)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.

```
#**************************slip 17**************************

import nltk

import re


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."""

text =re.sub(r'[[0-9]{}*]',' ',text)

formatted_text = re.sub('[^a-zA-Z]',' ',text)

print("\nText after removing digits and special characters\n",formatted_text)


from nltk.corpus import stopwords

from nltk.tokenize import word_tokenize,sent_tokenize

stopWords = set(stopwords.words("english"))

print("\n\n\nStopWords are :\n",stopWords)

words = word_tokenize(formatted_text)

print("\n\n\n-------------------------Extractive Summarization --------------------")

print("\n\n\n------------------- Words in Text --------------\n",words)


wordfreq = {}

for word in words:

   if word in stopWords:

      continue
```

```
    if word in wordfreq:
        wordfreq[word] += 1
    else:
        wordfreq[word] = 1

maximum_frequency = max(wordfreq.values())

for word in wordfreq.keys():
    wordfreq[word] = (wordfreq[word]/maximum_frequency)

print("\n\n\n---------------- Word Frequencies --------------\n",wordfreq)

sentences = sent_tokenize(text)
sentenceValue = {}

for sentence in sentences:
    for word , freq in wordfreq.items():
        if word in sentence.lower():
            if sentence in sentenceValue:
                sentenceValue[sentence] += freq
            else:
                sentenceValue[sentence] = freq
```

```
import heapq

summaray = ''

summaray_sentences = heapq.nlargest(4, sentenceValue , key=sentenceValue.get)

summaray = ' '.join(summaray_sentences)

print("\n\n\n------------------ Summary Text ------------------\n",summaray)
```

//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.

```
#***********************slip 18********************

import nltk

from nltk.tokenize import word_tokenize

from nltk.tokenize import sent_tokenize

from nltk.corpus import stopwords

from nltk.probability import FreqDist

from wordcloud import WordCloud, get_single_color_func


paragraph_text = """Hello all Students,Welcome to [Today's] Practical. Lets learn how to perform text and social media analysis. there are total 3 sets and 10 questions."""

tokenized_words = word_tokenize(paragraph_text)

sentences = sent_tokenize(paragraph_text)

print("\n---------- Sentences in paragraph -----------\n",sentences)
```

```
stop_words_data = set(stopwords.words("english"))


filtered_words_list = []

for words in tokenized_words:

   if words not in stop_words_data:

      filtered_words_list.append(words)


print("\n\n\n --------------------- Tokenized Words ---------------------
\n",tokenized_words)

print("\n\n\n----------------------Filtered Words After removing StopWords -------
-----\n",filtered_words_list)

print("\n\n\n")

frequency_distribution = FreqDist(tokenized_words)


import matplotlib.pyplot as plt

frequency_distribution.plot(32,cumulative=False)

plt.show()


print("\n\n")

print("WordCloud\n\n")

word_cloud =
WordCloud(collocations=False,background_color='black').generate(paragraph_
text)

plt.figure()

plt.imshow(word_cloud,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-review/version/3?select=movie_review.csv to perform sentiment analysis on above dataset and create a wordcloud.

```
#********************slip 19***********************

import pandas as pd

import nltk

from nltk.tokenize import word_tokenize

from nltk.corpus import stopwords

from nltk.sentiment import SentimentIntensityAnalyzer

from wordcloud import WordCloud

import matplotlib.pyplot as plt

#nltk.download('punkt')

#nltk.download('stopwords')

#nltk.download('vader_lexicon')

df=pd.read_csv(r"C:\Users\lenovo\Documents/movie_review.csv")

print(df.head())

sia=SentimentIntensityAnalyzer()

def get_sentiment_score(text):

   return sia.polarity_scores(text)

['compound']
```

```
df['Sentiment Score']=df['text'].apply(get_sentiment_score)

text=''.join(df['text'])

wordcloud=WordCloud(width=800,height=400,background_color='white',stopwords=set(stopwords.words('english'))).generate(text)

plt.figure(figsize=(10,5))

plt.imshow(wordcloud,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.

```
#**************************slip 20***********************

import nltk

from nltk.tokenize import word_tokenize, sent_tokenize

from nltk.corpus import stopwords

from nltk.probability import FreqDist

import matplotlib.pyplot as plt

from wordcloud import WordCloud, get_single_color_func


#nltk.download('punkt')

#nltk.download('stopwords')


# Textual data to remove stopwords
```

```
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."""


# Word Tokenization

tokenized_words = word_tokenize(paragraph_text)


# Sentence Tokenization

sentences = sent_tokenize(paragraph_text)

print("\n--- Sentences in Paragraph ---\n", sentences)


# Remove stopwords

stop_words_data = set(stopwords.words("english"))

filtered_words_list = [word for word in tokenized_words if word.lower() not in
stop_words_data]
```

//SLIP 25

//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..

```
#*******************************slip
25*****************************

import pandas as pd
```

```
from nltk.tokenize import word_tokenize

from nltk.corpus import stopwords

from nltk.sentiment.vader import SentimentIntensityAnalyzer


# Reading Dataset

data = pd.read_csv(r"C:\Users\lenovo\Documents/covid_2021.csv")

df = pd.DataFrame(data)


# Data cleaning operation

df = df.dropna()  # Add assignment to keep changes


print("Displaying Dataset:\n", df.head(5))


# Tokenize comments into words

comment_text = df['comment_text'].values.astype(str)

tokenized_words = word_tokenize(" ".join(comment_text))

print("\nComments into Tokenized words are:\n", tokenized_words)


# Perform Sentiment Analysis

vader_analyzer = SentimentIntensityAnalyzer()


total = len(comment_text)

pos_cnt, neg_cnt, neu_cnt = 0, 0, 0
```

```
for comment in comment_text:
    result = vader_analyzer.polarity_scores(comment)
    if result['compound'] >= 0.05:
        pos_cnt += 1
    elif result['compound'] <= -0.05:
        neg_cnt += 1
    else:
        neu_cnt += 1

# Display percentage of positive, negative, and neutral comments
print("\nPercentage of Positive Comments: {:.2f}%".format((pos_cnt / total) * 100))
print("Percentage of Negative Comments: {:.2f}%".format((neg_cnt / total) * 100))
print("Percentage of Neutral Comments: {:.2f}%".format((neu_cnt / total) * 100))
```

//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.

```
#******************************slip 26************************
import nltk
import re
```

```
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."""

text =re.sub(r'[[0-9]{}*]',' ',text)

formatted_text = re.sub('[^a-zA-Z]',' ',text)

print("\nText after removing digits and special characters\n",formatted_text)


from nltk.corpus import stopwords

from nltk.tokenize import word_tokenize,sent_tokenize

stopWords = set(stopwords.words("english"))

print("\n\n\nStopWords are :\n",stopWords)

words = word_tokenize(formatted_text)

print("\n\n\n-------------------------Extractive Summarization ---------------------")

print("\n\n\n------------------- Words in Text --------------\n",words)


wordfreq = {}

for word in words:

   if word in stopWords:

      continue

   if word in wordfreq:

      wordfreq[word] += 1

   else:

      wordfreq[word] = 1
```

```
maximum_frequency = max(wordfreq.values())

for word in wordfreq.keys():
    wordfreq[word] = (wordfreq[word]/maximum_frequency)

print("\n\n\n---------------- Word Frequencies --------------\n",wordfreq)

sentences = sent_tokenize(text)
sentenceValue = {}

for sentence in sentences:
    for word , freq in wordfreq.items():
        if word in sentence.lower():
            if sentence in sentenceValue:
                sentenceValue[sentence] += freq
            else:
                sentenceValue[sentence] = freq

import heapq
summaray = ''
summaray_sentences = heapq.nlargest(4, sentenceValue ,
key=sentenceValue.get)
summaray = ' '.join(summaray_sentences)
```

```
print("\n\n\n------------------ Summary Text ------------------\n",summaray)
```

#slip 27

#same as slip 9

//SLIP 29

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

```
import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import confusion_matrix, accuracy_score

import matplotlib.pyplot as plt


# Generating a synthetic dataset

np.random.seed(0)

n_samples = 1000

math_score = np.random.randint(0, 101, size=n_samples)

reading_score = np.random.randint(0, 101, size=n_samples)

writing_score = np.random.randint(0, 101, size=n_samples)

pass_fail = np.random.choice([0, 1], size=n_samples)  # 0: Fail, 1: Pass


data = {

    'Math Score': math_score,

    'Reading Score': reading_score,
    'Writing Score': writing_score,

    'Pass/Fail': pass_fail

}


# Creating DataFrame

student_data = pd.DataFrame(data)


# Displaying first few rows of the dataset

print(student_data.head())


# Input features

X = student_data[['Math Score', 'Reading Score', 'Writing Score']]


# Output variable

y = student_data['Pass/Fail']


# Splitting the dataset into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
random_state=0)


# Initializing and training the logistic regression model

classifier = LogisticRegression(random_state=0, solver='lbfgs')

classifier.fit(X_train, y_train)
```

```
# Make predictions
y_pred = classifier.predict(X_test)
print('Actual Values:', y_test.values)
print('Predicted Values:', y_pred)

# Calculate confusion matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", cm)

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

# Plotting confusion matrix
plt.figure(figsize=(8, 6))
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.colorbar()
tick_marks = np.arange(2)
plt.xticks(tick_marks, ['Fail', 'Pass'])
plt.yticks(tick_marks, ['Fail', 'Pass'])

# Adding values to the plot
```

```
thresh = cm.max() / 2.
for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
        plt.text(j, i, format(cm[i, j], 'd'),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.tight_layout()
plt.show()
```