```
import numpy as np
arr = np.array([1, 2, 3])
print("Array with Rank 1: \n",arr)
arr = np.array([[1, 2, 3],
                [4, 5, 6]])
print("Array with Rank 2: \n", arr)
arr = np.array((1, 3, 2))
print("\nArray created using "
      "passed tuple:\n", arr)
```

```
Array with Rank 1:
 [1 2 3]
Array with Rank 2:
 [[1 2 3]
 [4 5 6]]

Array created using passed tuple:
 [1 3 2]
```

Double-click (or enter) to edit

UNIT-I

Double-click (or enter) to edit

```
Start coding or generate with AI.
```

UNIT _ I 1.Reading Structured Data from CSV file:

```
import pandas  as pd
a=pd.read_csv('/content/sample_data/mnist_test.csv')

print(a.tail())
print(a.info())
```

```
      7  0  0.1  0.2  0.3  0.4  0.5  0.6  0.7  0.8  ...  0.658  0.659  0.660  \
9994  2  0    0    0    0    0    0    0    0    0  ...      0      0      0
9995  3  0    0    0    0    0    0    0    0    0  ...      0      0      0
9996  4  0    0    0    0    0    0    0    0    0  ...      0      0      0
9997  5  0    0    0    0    0    0    0    0    0  ...      0      0      0
9998  6  0    0    0    0    0    0    0    0    0  ...      0      0      0

      0.661  0.662  0.663  0.664  0.665  0.666  0.667
9994      0      0      0      0      0      0      0
9995      0      0      0      0      0      0      0
9996      0      0      0      0      0      0      0
9997      0      0      0      0      0      0      0
9998      0      0      0      0      0      0      0

[5 rows x 785 columns]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9999 entries, 0 to 9998
Columns: 785 entries, 7 to 0.667
dtypes: int64(785)
memory usage: 59.9 MB
None
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
---------------------------------------------------------------------------
MessageError                              Traceback (most recent call last)
/tmp/ipython-input-2-1408506528.py in <cell line: 0>()
      1 from google.colab import drive
----> 2 drive.mount('/content/drive')

3 frames

/usr/local/lib/python3.11/dist-packages/google/colab/_message.py in read_reply_from_input(message_id, timeout_sec)
    101     ):
    102       if 'error' in reply:
--> 103         raise MessageError(reply['error'])
    104       return reply.get('data', None)
    105

MessageError: Error: credential propagation was unsuccessful
```

```
import pandas as pd

df = pd.read_csv('/content/diabetes.csv')
print(df)
print(df.head(2))
df.describe()
```

```
import pandas as pd
df=pd.read_csv('/content/diabetes.csv')
print(df.info())
df.describe()
```

Show code

```
Start coding or generate with AI.
```

New Section

2.Reading Unstructured Data (Text) from a File:

```
# Read a text file (unstructured data)
with open('/content/abc.txt', 'r') as file:
    data = file.read()
    print(data)
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
/tmp/ipython-input-2-3180383937.py in <cell line: 0>()
      1 # Read a text file (unstructured data)
----> 2 with open('/content/abc.txt', 'r') as file:
      3     data = file.read()
      4     print(data)

FileNotFoundError: [Errno 2] No such file or directory: '/content/abc.txt'
```

```
Start coding or generate with AI.
```

3.Exploring Quantitative Data Analysis:

```
Start coding or generate with AI.
```

```
import numpy as np
from scipy import stats
# Generate some quantitative data
data = np.random.normal(loc=0, scale=20, size=50)
print(data)
x=[3,4,3,4,5]
print(x)
print("Mean:", np.mean(x))
print("median:", np.median(x))
print("mode:", stats.mode(x))

print("Standard Deviation:", np.std(data))
```

4.Handling Categorical Data with Pandas:

```
import pandas as pd
# Create a DataFrame with categorical data
data = {'Category': ['A', 'B', 'C', 'A', 'B'],"QUNT":[2,5,4,7,8]}
print(data)
df = pd.DataFrame(data)
print(df)
# Convert categorical data to numerical
print(df['Category'].value_counts())
```

```
{'Category': ['A', 'B', 'C', 'A', 'B'], 'QUNT': [2, 5, 4, 7, 8]}
  Category  QUNT
0        A     2
```

```
1        B     5
2        C     4
3        A     7
4        B     8
Category
A    2
B    2
C    1
Name: count, dtype: int64
```

5.Processing Big Data with Dask (Parallel Computing):

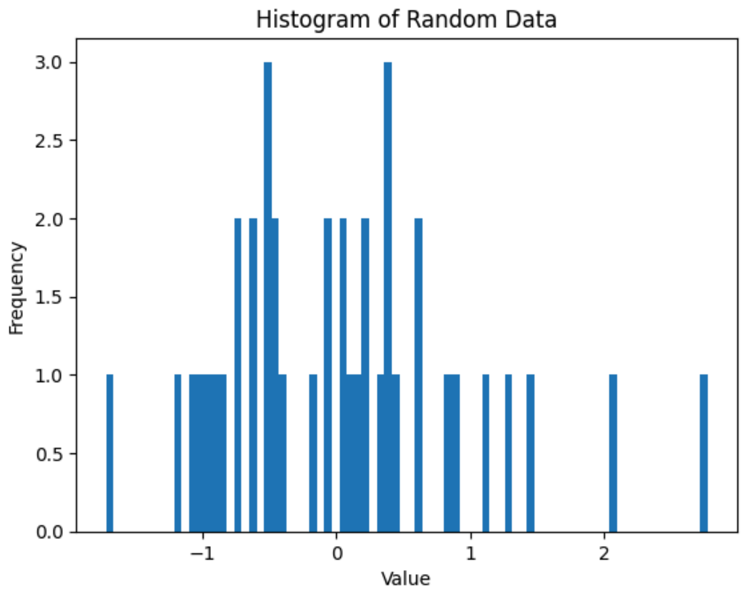
```
import dask.dataframe as dd
# Read and process large CSV file with Dask
df = dd.read_csv('/content/sample_data/mnist_test.csv')
print(df.tail())
print(df.info())
```

```
      7  0  0.1  0.2  0.3  0.4  0.5  0.6  0.7  0.8  ...  0.658  0.659  0.660  \
9994  2  0    0    0    0    0    0    0    0    0  ...      0      0      0
9995  3  0    0    0    0    0    0    0    0    0  ...      0      0      0
9996  4  0    0    0    0    0    0    0    0    0  ...      0      0      0
9997  5  0    0    0    0    0    0    0    0    0  ...      0      0      0
9998  6  0    0    0    0    0    0    0    0    0  ...      0      0      0

      0.661  0.662  0.663  0.664  0.665  0.666  0.667
9994      0      0      0      0      0      0      0
9995      0      0      0      0      0      0      0
9996      0      0      0      0      0      0      0
9997      0      0      0      0      0      0      0
9998      0      0      0      0      0      0      0

[5 rows x 785 columns]
<class 'dask.dataframe.dask_expr.DataFrame'>
Columns: 785 entries, 7 to 0.667
dtypes: int64(785)None
```

6.Data Visualization with Matplotlib:

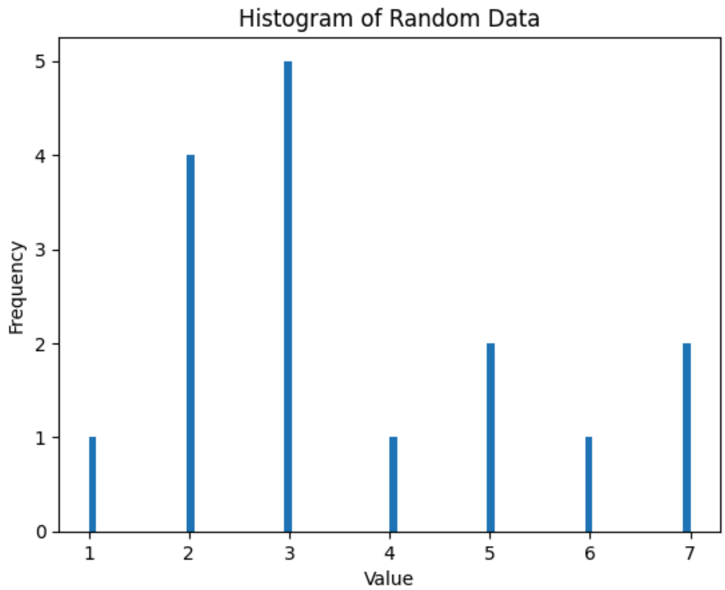
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
# Plotting quantitative data
data = np.random.randn(40)
print(data)
plt.hist(data, bins=80)
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.title('Histogram of Random Data')
plt.show()
```

```
[ 0.23356632 -0.63541803 -0.82609607 -0.88968225 -0.50833056 -0.17313629
 -0.52001757 -0.03743259  0.46841914 -0.71051762  0.34663698  0.1404032
  0.1018992   0.37572374  0.06840139  1.1108578   1.43766931 -0.08445489
  0.03122318 -0.44922931 -0.42557682 -0.93970975 -0.39010039  0.62503851
  0.37126321  1.26312157 -0.60700966 -1.01079815  2.08051647 -1.06347532
 -1.71474014  0.41685972  0.87803763 -1.17259437 -0.51971501  0.86500189
  0.19748307  0.62746142  2.7741298  -0.70518417]
```



```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
# Plotting quantitative data
#data = np.random.randn(40)
data = pd.DataFrame({'col1': [1,2,3,4,3,7,5,2,2,3,3,2,5,6,3,7], 'col2': [4,5,4,3,4,8,5,8,5,5,2,4,6,2,2,3]})
print(data)
plt.hist(data['col1'], bins=80,) # Changed to use a column from the DataFrame
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.title('Histogram of Random Data')
plt.show()
```

```
    col1  col2
0      1     4
1      2     5
2      3     4
3      4     3
4      3     4
5      7     8
6      5     5
7      2     8
8      2     5
9      3     5
10     3     2
11     2     4
12     5     6
13     6     2
14     3     2
15     7     3
```




```
!pip install pypdf
```

```
#Working with pdf
# importing required classes
from pypdf import PdfReader

# creating a pdf reader object
reader = PdfReader('/content/xy.pdf')

# printing number of pages in pdf file
print(len(reader.pages))

# creating a page object
page = reader.pages[0]

# extracting text from page
print(page.extract_text())
```

```
import pandas as pd
df = pd.read_excel('/content/xy.xlsx')
print(df.head())
```

```
pip install python-docx
```

```
#Working With Microsoft document
import docx
doc = docx.Document("/content/xyz.docx")
all_paras = doc.paragraphs
len(all_paras)
print(all_paras)
```

```
for para in all_paras:
    print(para.text)
    print("-------")
```

Example Extra

```
import matplotlib.pyplot as plt
import numpy as np
x=[12,34,56]
y=[23,46,60]
plt.plot(x,y,'red')
plt.xlabel('X.Axis')
plt.ylabel('Y.Axis')
plt.title('line chart')
plt.show()
```

7.Data Cleaning and Preprocessing with Pandas:

```
import pandas as pd
# Data cleaning example
df = pd.DataFrame({'A': [3, 2, None, 4], 'B': ['X', 'Y', 'Z', 'W']})
print(df)
df_cleaned = df.dropna()
print(df_cleaned)
```

Working With Video

```
input_file = '/content/026c7465-309f6d33.mov'
subprocess.run(['ffmpeg',
                '-i',
                input_file,
                '-qscale',
                '0',
                '026c7465-309f6d33.mov',
                '-loglevel',
                'quiet']
              )
```

```
!pip install cv
```

```
!pip install glob
```

```
import pandas as pd
import numpy as np
import cv2
import matplotlib.pyplot as plt

from glob import glob

import IPython.display as ipd
from tqdm import tqdm

import subprocess

plt.style.use('ggplot')
```

```
!ls -GFlash --color
```

```
ipd.Video('/content/026c7465-309f6d33.mov', width=700)
```

```
# Load in video capture
cap = cv2.VideoCapture('026c7465-309f6d33.mov')
```

```
# Load in video capture
cap = cv2.VideoCapture('026c7465-309f6d33.mov')
```

*Unit-II Data Preprocessing and Warehouse *

1.Handling Missing Values with Pandas:

```
import pandas as pd
import numpy as np

# Create a DataFrame with missing values
data = {'A': [1, 2, np.nan, 4], 'B': ['X', 'Y', np.nan, 'Z']}
df = pd.DataFrame(data)
print(df)
# Handling missing values
df_filled = df.fillna(0)  # Fill missing values with 0
print(df_filled)
```

```
     A    B
0  1.0    X
1  2.0    Y
2  NaN  NaN
3  4.0    Z
     A  B
0  1.0  X
1  2.0  Y
2  0.0  0
3  4.0  Z
```

```
import pandas as pd
# Create a sample DataFrame with some null values
data = {
    'Name': ['Alice', 'Bob', None, 'David'],
    'Age': [25, None, 30, 22],
    'City': ['New York', 'Los Angeles', 'Chicago', None]
}
df = pd.DataFrame(data)
print("Original DataFrame:")
print(df)
# Remove rows with any null values
df_clean = df.dropna()

print("\nDataFrame after removing rows with null values:")
print(df_clean)
```

```
Original DataFrame:
    Name   Age         City
0  Alice  25.0     New York
1    Bob   NaN  Los Angeles
2   None  30.0      Chicago
3  David  22.0         None

DataFrame after removing rows with null values:
    Name   Age      City
0  Alice  25.0  New York
```

2.Removing Duplicate Rows with Pandas:

```
import pandas as pd

# Create a DataFrame with duplicate rows
data = {'A': [1, 2, 2, 3, 3], 'B': ['X', 'Y', 'Y', 'Z', 'X']}
df = pd.DataFrame(data)
print("Original DataFrame:")
print(df)
# Remove duplicate rows
df_unique = df.drop_duplicates()
print("\nDataFrame after removing duplicates:")
print(df_unique)
```

```
Original DataFrame:
   A  B
0  1  X
1  2  Y
2  2  Y
```

```
3  3  Z
4  3  X

DataFrame after removing duplicates:
   A  B
0  1  X
1  2  Y
3  3  Z
4  3  X
```

3.Handling Categorical Data with One-Hot Encoding:

```
import pandas as pd

# Create a DataFrame with categorical data
data = {'Category': ['A', 'B', 'A', 'C']}
df = pd.DataFrame(data)

# Perform one-hot encoding
df_encoded = pd.get_dummies(df['Category'])
print(df_encoded)
```

```
       A      B      C
0   True  False  False
1  False   True  False
2   True  False  False
3  False  False   True
```

```
from sklearn.preprocessing import MinMaxScaler
import numpy as np

data = np.array([[20], [40], [60], [80], [100]])
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(data)

print(scaled_data)
```

```
[[0.  ]
 [0.25]
 [0.5 ]
 [0.75]
 [1.  ]]
```

4.Scaling Numerical Data with Scikit-Learn:

```
from sklearn.preprocessing import MinMaxScaler
import pandas as pd
# Create a DataFrame with numerical data
data = {'Value': [10, 20, 30, 40]}
df = pd.DataFrame(data)

# Scale numerical data
scaler = MinMaxScaler()
df_scaled = scaler.fit_transform(df)
print(df_scaled)
```

```
[[0.        ]
 [0.33333333]
 [0.66666667]
 [1.        ]]
```

5.Handling Outliers with Z-Score:

```
import pandas as pd
from scipy import stats
import numpy as np
# Create a DataFrame with numerical data including outliers
data = {'Value': [1000,10, 20, 100, 30, 40]}
df = pd.DataFrame(data)
# Calculate Z-score
z_scores = np.abs(stats.zscore(df))
print(z_scores)
df_no_outliers = df[(z_scores < 4).all(axis=1)]
print(df_no_outliers)
#correction  needed
```

```
[[2.22882446]
 [0.52934581]
 [0.5014855 ]
 [0.27860306]
 [0.4736252 ]
 [0.44576489]]
   Value
0   1000
1     10
2     20
3    100
4     30
5     40
```

```
import pandas as pd
# Sample data
df = pd.DataFrame({'Score': [10, 12, 11, 13, 95]})
# IQR method
Q1 = df['Score'].quantile(0.25)
Q3 = df['Score'].quantile(0.75)
IQR = Q3 - Q1
# Define bounds
lower = Q1 - 1.5 * IQR
upper = Q3 + 1.5 * IQR
# Filter data
filtered_df = df[(df['Score'] >= lower) & (df['Score'] <= upper)]
print(filtered_df)
```

```
   Score
0     10
1     12
2     11
3     13
```

```
import pandas as pd

# Create a DataFrame with numerical data including outliers
data = {'Value': [10, 20, 100, 30, 40]}
df = pd.DataFrame(data)
print(data)
# Calculate the first quartile (Q1) and third quartile (Q3)
Q1 = df['Value'].quantile(0.25)
Q3 = df['Value'].quantile(0.75)

# Calculate the interquartile range (IQR)
IQR = Q3 - Q1

# Define the lower and upper bounds to filter outliers
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Filter the DataFrame to remove outliers
df_no_outliers = df[(df['Value'] >= lower_bound) & (df['Value'] <= upper_bound)]

print(df_no_outliers)
```

```
{'Value': [10, 20, 100, 30, 40]}
   Value
0     10
1     20
3     30
4     40
```

```
import pandas as pd
df = pd.read_csv('/content/diabetes.csv')
print(df)
l = ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin']
for i in l:
  df[i]=df[i].replace(0,np.nan)
  print(df)
```

```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0              6      148             72             35        0  33.6
1              1       85             66             29        0  26.6
2              8      183             64              0        0  23.3
3              1       89             66             23       94  28.1
4              0      137             40             35      168  43.1
..           ...      ...            ...            ...      ...   ...
763           10      101             76             48      180  32.9
764            2      122             70             27        0  36.8
765            5      121             72             23      112  26.2
```

```
766           1      126            60             0       0  30.1
767           1       93            70            31       0  30.4

     DiabetesPedigreeFunction  Age  Outcome
0                       0.627   50        1
1                       0.351   31        0
2                       0.672   32        1
3                       0.167   21        0
4                       2.288   33        1
..                        ...  ...      ...
763                     0.171   63        0
764                     0.340   27        0
765                     0.245   30        0
766                     0.349   47        1
767                     0.315   23        0

[768 rows x 9 columns]
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0              6    148.0             72             35        0  33.6
1              1     85.0             66             29        0  26.6
2              8    183.0             64              0        0  23.3
3              1     89.0             66             23       94  28.1
4              0    137.0             40             35      168  43.1
..           ...      ...            ...            ...      ...   ...
763           10    101.0             76             48      180  32.9
764            2    122.0             70             27        0  36.8
765            5    121.0             72             23      112  26.2
766            1    126.0             60              0        0  30.1
767            1     93.0             70             31        0  30.4

     DiabetesPedigreeFunction  Age  Outcome
0                       0.627   50        1
1                       0.351   31        0
2                       0.672   32        1
3                       0.167   21        0
4                       2.288   33        1
..                        ...  ...      ...
763                     0.171   63        0
764                     0.340   27        0
765                     0.245   30        0
766                     0.349   47        1
767                     0.315   23        0

[768 rows x 9 columns]
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0              6    148.0           72.0             35        0  33.6
1              1     85.0           66.0             29        0  26.6
2              8    183.0           64.0              0        0  23.3
```

```
l = ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin']
for i in l:
  mean=int(df[i].mean(skipna=True))
  df[i]=df[i].replace(np.nan,mean)
  print(df.head(20))
```

Unit-III Classification

1.Nearest Neighbor Classification with Scikit-Learn

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Load the Iris dataset
iris = load_iris()
print(iris)
X, y = iris.data, iris.target

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

# Create and train a k-Nearest Neighbor classifier
clf = KNeighborsClassifier(n_neighbors=3)
clf.fit(X_train, y_train)

# Predict on test data
y_pred = clf.predict(X_test)

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

KNN_Classification

```
import pandas as pd
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
```

```
data = pd.read_csv('/content/KNNAlgorithmDataset1.csv')
x = data.drop('diagnosis',axis=1).values
y = data['diagnosis'].values
imputer = SimpleImputer(strategy='mean')
X = imputer.fit_transform(x)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state = 42, shuffle = True)
```

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
knn = KNeighborsClassifier(n_neighbors=10)
# Train the classifier
knn.fit(X_train, y_train)
# Make predictions on the test set
y_pred = knn.predict(X_test)
```

Start coding or generate with AI.

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

```
import numpy as np
import matplotlib.pyplot as plt

# Map category labels to colors
colors = {'B': 'blue', 'M': 'red'}

# Varying the number of neighbors from 1 to 20
neighbors = np.arange(1, 21)

# Initialize lists to store accuracies and predictions
accuracies = []
y_preds_list = []

# Loop through different values of n_neighbors
for n in neighbors:
    # Initialize the KNN classifier
    knn = KNeighborsClassifier(n_neighbors=10)
```

```
        # Train the classifier
        knn.fit(X_train, y_train)

        # Make predictions on the test set
        y_pred = knn.predict(X_test)

        # Calculate accuracy
        accuracy = accuracy_score(y_test, y_pred)
        accuracies.append(accuracy)

        # Store predictions
        y_preds_list.append(y_pred)

    # Plot the scatter plot
    plt.figure(figsize=(12, 8))

    # Plot category points (training set)
    plt.scatter(X_train[:, 0], X_train[:, 1], c=[colors[label] for label in y_train], label='Category Points (Training Set)', alpha=0.6)

    # Plot new data points (test set) with predicted labels
    #for i, n in enumerate(neighbors):
       # plt.scatter(X_test[:, 0], X_test[:, 1], c=[colors[label] for label in y_preds_list[i]], marker='${}$'.format(n), label='k={}'.forma

    plt.title('KNN Classification Results with Varying Number of Neighbors')
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.legend()
    plt.grid(True)
    plt.show()
```

```
from sklearn.metrics import confusion_matrix
import seaborn as sns


# Initialize the KNN classifier with the chosen value of n_neighbors
knn = KNeighborsClassifier(n_neighbors=10)

# Train the classifier
knn.fit(X_train, y_train)

# Make predictions on the test set
y_pred = knn.predict(X_test)

# Compute the confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot the confusion matrix as a heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, cmap='Blues', fmt='g')
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```

```
from sklearn.metrics import classification_report

knn = KNeighborsClassifier(n_neighbors=3)

knn.fit(X_train, y_train)

y_pred = knn.predict(X_test)

y_test_labels = y_test
y_pred_labels = y_pred

# Create classification report
report = classification_report(y_test_labels, y_pred_labels)

print(report)
```

2.Naïve Bayes Classification with Scikit-Learn

```
from sklearn.naive_bayes import GaussianNB
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Load the Iris dataset
```

```
iris = load_iris()
print(iris)
X, y = iris.data, iris.target

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

# Create and train a Naïve Bayes classifier
clf = GaussianNB()
clf.fit(X_train, y_train)

# Predict on test data
y_pred = clf.predict(X_test)

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

```
# Import necessary libraries
from sklearn.naive_bayes import GaussianNB
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

# Load the Iris dataset
iris = load_iris()
X, y = iris.data, iris.target

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

# Create and train a Naïve Bayes classifier
clf = GaussianNB()
clf.fit(X_train, y_train)

# Predict on test data
y_pred = clf.predict(X_test)

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

# Create and display the confusion matrix
# confusion_matrix expects numeric labels
cm = confusion_matrix(y_test, y_pred)

# Create the ConfusionMatrixDisplay object
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=iris.target_names)

# Plot the confusion matrix
disp.plot(cmap='Blues', values_format='d')
plt.show()
```

3.Decision Tree Classification with Scikit-Learn

Double-click (or enter) to edit

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Load the Iris dataset
iris = load_iris()
X, y = iris.data, iris.target

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

# Create and train a Decision Tree classifier
clf = DecisionTreeClassifier(max_depth=3)
clf.fit(X_train, y_train)

# Predict on test data
y_pred = clf.predict(X_test)
```

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

4.Regression Methods for Forecasting Numeric Data: Linear Regression for Numeric Forecasting

```
from sklearn.linear_model import LinearRegression
import numpy as np

# Generate synthetic data
np.random.seed(0)
X = np.random.rand(100, 1) * 10
y = 2.5 * X.squeeze() + np.random.randn(100) * 2

# Create and train a Linear Regression model
model = LinearRegression()
model.fit(X, y)

# Make predictions
X_new = np.array([[5.0]])  # Example prediction for input 5.0
y_pred = model.predict(X_new)

print("Predicted value:", y_pred[0])
```

```
Start coding or generate with AI.
```

5.Evaluating Model Performance: Measuring Performance for Classification (Accuracy, Precision, Recall, F1-score):

```
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

# Load the Iris dataset
iris = load_iris()
X, y = iris.data, iris.target

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

# Create and train a Random Forest classifier
clf = RandomForestClassifier(n_estimators=100)
clf.fit(X_train, y_train)

# Predict on test data
y_pred = clf.predict(X_test)

# Evaluate performance
print("Classification Report:")
print(classification_report(y_test, y_pred))

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

Unit IV Association Rule mining

```
Start coding or generate with AI.
```

**1.Apriori Algorithm: **

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

Now we have to proceed by reading the dataset we have, that is in a csv format. We do that using pandas module's read_csv function

```
dataset = pd.read_csv("/content/Market_Basket_Optimisation.csv", header = None)
transactions = []
for i in range(0, 7501):
    transactions.append([str(dataset.values[i,j]) for j in range(0,20)])
```

Take a glance at the records

```
dataset
```

Look at the shape

```
dataset.shape
```

Convert Pandas DataFrame into a list of lists

```
for i in range(0, 7501):
    transactions.append([str(dataset.values[i,j]) for j in range(0,20)])
```

```
!pip install apyori
```

```
Start coding or generate with AI.
```

Build the Apriori model

```
from apyori import apriori
rules = apriori(transactions = transactions, min_support = 0.003, min_cinfidence = 0.2, min_lift = 3, min_length = 2, max_length = 2)
```

```
results = list(rules)
```

```
results
```

Visualizing the results

Double-click (or enter) to edit

```
Start coding or generate with AI.
```

```
resultsinDataFrame
```

2.FP Growth

```
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
```

```
dataset = [['Milk', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', 'Yogurt'],
           ['Dill', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', 'Yogurt'],
           ['Milk', 'Apple', 'Kidney Beans', 'Eggs'],
           ['Milk', 'Unicorn', 'Corn', 'Kidney Beans', 'Yogurt'],
           ['Corn', 'Onion', 'Onion', 'Kidney Beans', 'Ice cream', 'Eggs']]
```

```
te = TransactionEncoder()
print(te)
te_ary = te.fit(dataset).transform(dataset)
print(te_ary)
df = pd.DataFrame(te_ary, columns=te.columns_)
df
```

```
from mlxtend.frequent_patterns import fpgrowth
```

```
fpgrowth(df, min_support=0.6)
```

Final Result

```
fpgrowth(df, min_support=0.6, use_colnames=True)
```

3.Eclat Algorithm - Association Rule Learning

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from mlxtend.frequent_patterns import apriori,association_rules
from mlxtend.preprocessing import TransactionEncoder
df=pd.read_csv("/content/basket_analysis.csv",header=None)
#shape
df.shape
#head of data
df.head()
```

```
Start coding or generate with AI.
```

```
pip install pyECLAT
```

```
from pyECLAT import ECLAT
eclat_instance = ECLAT(data=df, verbose=True)
```

```
eclat_instance.df_bin   #generate a binary dataframe, that can be used for other analyzes.
eclat_instance.uniq_    #a list with all the names of the different items
```

```
get_ECLAT_indexes, get_ECLAT_supports = eclat_instance.fit(min_support=0.08,min_combination=1,max_combination=3,separator=' & ',verbose=
```

```
get_ECLAT_indexes
```

Double-click (or enter) to edit

```
get_ECLAT_supports
```

```
help(eclat_instance.fit)
```

```
help(eclat_instance.fit_all)
help(eclat_instance.support)
```

```
from mlxtend.frequent_patterns import association_rules
import pandas as pd

# Sample frequent itemsets (DataFrame)
frequent_itemsets = pd.DataFrame({
    'itemsets': [['Milk'], ['Bread'], ['Milk', 'Bread']],
    'support': [0.6, 0.8, 0.4]
})

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

print("Association Rules:")
print(rules[['antecedents', 'consequents', 'support', 'confidence', 'lift']])
```

Unit V Clustering

1.K-means Clustering

```
from sklearn.cluster import KMeans
import numpy as np

# Sample data
X = np.array([[1, 2], [5, 8], [1.5, 1.8], [8, 8], [1, 0.6], [9, 11]])

# Initialize KMeans with 2 clusters
kmeans = KMeans(n_clusters=2)
kmeans.fit(X)
```

```
# Print cluster centers and labels
print("Cluster centers:")
print(kmeans.cluster_centers_)
print("Labels:")
print(kmeans.labels_)
```

2Hierarchical Clustering

```
from scipy.cluster.hierarchy import linkage, dendrogram
import matplotlib.pyplot as plt
import numpy as np

# Sample data
X = np.array([[1, 2], [5, 8], [1.5, 1.8], [8, 8], [1, 0.6]])

# Perform hierarchical clustering
linked = linkage(X, 'single')

# Plot dendrogram
plt.figure(figsize=(10, 7))
dendrogram(linked, orientation='top', distance_sort='descending')
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('Sample Index')
plt.ylabel('Distance')
plt.show()
```

3.DBSCAN (Density-Based Clustering)

```
from sklearn.cluster import DBSCAN
import numpy as np

# Sample data
X = np.array([[1, 2], [2, 2], [2, 3], [8, 7], [8, 8], [25, 80]])

# Apply DBSCAN
dbscan = DBSCAN(eps=5, min_samples=3)
dbscan.fit(X)

# Print cluster labels
print("Cluster labels:")
print(dbscan.labels_)
```

4.Evaluation Metrics: Silhouette Score

```
from sklearn.metrics import silhouette_score
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans

# Generate sample data
X, _ = make_blobs(n_samples=100, centers=4, cluster_std=1, random_state=42)

# Fit KMeans clustering
kmeans = KMeans(n_clusters=4, random_state=42)
kmeans.fit(X)

# Calculate silhouette score
silhouette_avg = silhouette_score(X, kmeans.labels_)
print(f"Silhouette Score: {silhouette_avg}")
```

5.Clustering Visualization

```
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans

# Generate sample data
X, _ = make_blobs(n_samples=300, centers=4, cluster_std=0.60, random_state=0)

# Fit KMeans clustering
kmeans = KMeans(n_clusters=4)
kmeans.fit(X)
```

```
# Visualize clusters
plt.scatter(X[:, 0], X[:, 1], c=kmeans.labels_, cmap='viridis')
centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], marker='*', s=200, edgecolors='k', c='red')
plt.title('KMeans Clustering')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
```

Image Segmentation using KMeans

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from skimage import io

# Load image
image = io.imread('/content/sp.jpg')
image = np.array(image, dtype=np.float64) / 255

# Reshape the image to 2D array of pixels
w, h, d = tuple(image.shape)
image_array = np.reshape(image, (w * h, d))

# Apply KMeans clustering
kmeans = KMeans(n_clusters=5, random_state=42)
kmeans.fit(image_array)

# Reshape labels back to original image shape
labels = np.reshape(kmeans.labels_, (w, h))

# Display segmented image
plt.imshow(labels, cmap='viridis')
plt.title('Image Segmentation using KMeans')
plt.axis('off')
plt.show()
```

Evaluation of Clustering Performance

```
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

# Generate sample data
X, _ = make_blobs(n_samples=300, centers=4, cluster_std=0.60, random_state=0)

# Fit KMeans clustering
kmeans = KMeans(n_clusters=4)
kmeans.fit(X)

# Evaluate clustering performance using Silhouette Score
silhouette_avg = silhouette_score(X, kmeans.labels_)
print(f"Silhouette Score: {silhouette_avg}")
```

DBScan Algorithm

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler
data = pd.read_csv("/content/Mall_Customers.csv", index_col=0)
data.head()
```

Annual Income (k$) vs Spending Score (1-100)

```
# @title Annual Income (k$) vs Spending Score (1-100)

from matplotlib import pyplot as plt
data.plot(kind='scatter', x='Annual Income (k$)', y='Spending Score (1-100)', s=32, alpha=.8)
plt.gca().spines[['top', 'right',]].set_visible(False)
```

```
#Taking the full fraction of data -> shuffles the data
data = data.sample(frac=1)
data.head()
```

Unit VI Data visualization

Bar Graph

```
import matplotlib.pyplot as plt
categories = ['Apples', 'Bananas', 'Grapes']
values = [30, 40, 25]
plt.bar(categories, values)
plt.xlabel('Categories')
plt.ylabel('Values')
plt.title('Bar Graph Example')
plt.show()
```

Stacked Bar Chart

```
import matplotlib.pyplot as plt
categories = ['A', 'B', 'C']
values1 = [20, 35, 30]
values2 = [25, 32, 34]
plt.bar(categories, values1, label='Group 1')
plt.bar(categories, values2, bottom=values1, label='Group 2')
plt.xlabel('Categories')
plt.ylabel('Values')
plt.title('Stacked Bar Chart Example')
plt.legend()
plt.show()
```

Pie Chart

```
import matplotlib.pyplot as plt
sizes = [25, 30, 20, 25]
labels = ['A', 'B', 'C', 'D']
plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=140)
plt.axis('equal')
plt.title('Pie Chart Example')
plt.show()
```

Doughnut Chart

```
import matplotlib.pyplot as plt

sizes = [30, 20, 25, 25]
labels = ['A', 'B', 'C', 'D']

plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=140)
centre_circle = plt.Circle((0,0),0.30,fc='white')
fig = plt.gcf()
fig.gca().add_artist(centre_circle)
plt.axis('equal')
plt.title('Doughnut Chart Example')
plt.show()
```

Line Chart

```
import matplotlib.pyplot as plt

x = [1, 2, 3, 4, 5]
y = [10, 15, 7, 10, 5]

plt.plot(x, y, marker='o')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Line Chart Example')
plt.grid(True)
```

```
plt.show()
```

Area Chart

```
import matplotlib.pyplot as plt

x = [1, 2, 3, 4, 5]
y1 = [10, 15, 7, 10, 5]
y2 = [5, 8, 12, 6, 15]

plt.fill_between(x, y1, y2, alpha=0.2)
plt.plot(x, y1, label='Y1', marker='o')
plt.plot(x, y2, label='Y2', marker='o')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Area Chart Example')
plt.legend()
plt.grid(True)
plt.show()
```

Treemap Chart

```
import matplotlib.pyplot as plt
!pip install squarify
import squarify

sizes = [25, 40, 15, 20]
labels = ['A', 'B', 'C', 'D']

squarify.plot(sizes=sizes, label=labels, alpha=0.7)
plt.axis('off')
plt.title('Treemap Chart Example')
plt.show()
```

Heatmap

```
import matplotlib.pyplot as plt
import numpy as np

data = np.random.rand(10, 12)

plt.imshow(data, cmap='hot', interpolation='nearest')
plt.colorbar()
plt.title('Heatmap Example')
plt.show()
```

Waterfall Chart

```
import matplotlib.pyplot as plt

categories = ['Start', 'Step 1', 'Step 2', 'Step 3', 'End']
values = [0, 5, 8, -3, 10]

plt.bar(categories, values, color='b')
plt.plot(categories, values, marker='o', color='g')
plt.xlabel('Categories')
plt.ylabel('Values')
plt.title('Waterfall Chart Example')
plt.show()
```

Scatter Plot

```
import matplotlib.pyplot as plt
import numpy as np

x = np.random.rand(50)
y = np.random.rand(50)
sizes = np.random.rand(50) * 100

plt.scatter(x, y, s=sizes, alpha=0.5)
```

```
    plt.xlabel('X-axis')
    plt.ylabel('Y-axis')
```