Exp_1 Basic Operation:

```
def string_operation():
    string="example"
    print("Actual String :",string)
    print("Upper case",string.upper())
    print("Lower Case",string.lower())
    print("slice",string[2])
    print("Reverse",string[::-1])
string_operation()

def list_operation():
    list1=[1,2,3,4,5]
    list2=[6,7,8,9,0]
    print("list1",list1)
    list1.extend(list2)
    print("Extend",list1)
    list1.append(35)
    print("Adding",list1)
    list2.remove(9)
    print("removing",list2)

list_operation()

def set_operation():
    set1={1,2,3,4,5,6}
    set2={7,8,9,0,11,23}
    print("Intersection",set1|set2)
    print("Union",set1&set2)
    set1.add(25)
    set2.remove(11)
    print("Adding",set1)
    print("Remove",set2)

set_operation()

def tuples_operation():
    tuples=(10,20,30,40,50)
    print("Tuples",tuples)
    print("Access Second Element :",tuples[1])
    print("Length",len(tuples))
    print("Last Element",tuples[-1])

tuples_operation()
```

Exp_2:Dict to check key

```
my_dict={'Name':'John','age':56}

def check_dict_key():
    key=input("Enter Te Key:")
    value=input(f"Ener Value for {key}")

    if key not in my_dict:
        my_dict[key]=value
        print(f"\nThe {key} is add to the {value}")
    else:
        print(f"\nthe {key} is alredy Present on Dictionary")

check_dict_key()
while True:
    another_key = input("\nWant to Add Key? (yes/no): ").lower()

    if another_key == "yes":
        check_dict_key()
    else:
        print("Exist")
        break
print("\nUpdated Dictionary",my_dict)
```

Exp:3 Text Preprocessing

```
import nltk
import string
from sklearn.feature_extraction.text import TfidfVectorizer
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
import warnings
warnings.simplefilter('ignore')
nltk.download('punkt')
nltk.download('punkt_tab')
nltk.download('stopwords')
nltk.download('wordnet')
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))
def preprocess_text(text):
    text = text.lower()
    text = text.translate(str.maketrans('', '', string.punctuation))
    tokens = word_tokenize(text)
    processed_tokens = [
        lemmatizer.lemmatize(word) for word in tokens if word not in stop_words and word.isalpha()
    ]
    return ' '.join(processed_tokens)
documents = [
    "The quick brown fox jumped over the lazy dog.",
    "I love programming in Python, especially for data analysis.",
    "Natural language processing is fascinating!"
]
preprocessed_docs = [preprocess_text(doc) for doc in documents]
vectorizer = TfidfVectorizer()
tfidf_matrix = vectorizer.fit_transform(preprocessed_docs)
tfidf_array = tfidf_matrix.toarray()
feature_names = vectorizer.get_feature_names_out()
print("TF-IDF Matrix:")
for i, doc in enumerate(tfidf_array):
    print(f"\nDocument {i + 1}:")
    for word, score in zip(feature_names, doc):
        if score > 0:
            print(f"{word}: {score}")
```

Exp:4 Linear,Logistic Regression

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.preprocessing import LabelEncoder
data = pd.read_csv('Admission_Predict.csv')
X_linear = data[['GRE Score']].values
y_linear = data['Chance of Admit ']].values
X_logistic = data[['GRE Score', 'CGPA']].values
y_logistic = (data['Chance of Admit '] > 0.75).astype(int)
def linear_regression():
    X_train, X_test, y_train, y_test = train_test_split(X_linear, y_linear, test_size=0.2,
random_state=42)
    model = LinearRegression()
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    plt.scatter(X_test, y_test, color='blue', label="Actual data")
    plt.plot(X_test, y_pred, color='red', linewidth=2, label="Regression line")
    plt.xlabel("GRE Score")
    plt.ylabel("Chance of Admit")
    plt.title("Linear Regression: GRE Score vs Chance of Admit")
    plt.legend()
    plt.show()

def logistic_regression():
    X_train, X_test, y_train, y_test = train_test_split(X_logistic, y_logistic, test_size=0.2,
random_state=42)
    model = LogisticRegression()
    model.fit(X_train, y_train)
    x_min, x_max = X_logistic[:, 0].min() - 5, X_logistic[:, 0].max() + 5
    y_min, y_max = X_logistic[:, 1].min() - 0.5, X_logistic[:, 1].max() + 0.5
    xx, yy = np.meshgrid(np.linspace(x_min, x_max, 100), np.linspace(y_min, y_max, 100))
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    plt.contourf(xx, yy, Z, alpha=0.3)
    plt.scatter(X_test[:, 0], X_test[:, 1], c=y_test, edgecolors='k', marker='o', label="Test data")
    plt.xlabel("GRE Score")
    plt.ylabel("CGPA")
    plt.title("Logistic Regression Decision Boundary")
    plt.legend()
    plt.show()
```

Exp:5 Decision Tree Classification

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.tree import plot_tree
iris = datasets.load_iris()
X = iris.data
y = iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
model = DecisionTreeClassifier(random_state=42)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of the Decision Tree classifier: {accuracy:.2f}")
plt.figure(figsize=(12,8))
plot_tree(model, filled=True, feature_names=iris.feature_names, class_names=iris.target_names)
plt.show()
```

Exp:6 Naïve Bayes Classification

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

iris = datasets.load_iris()
X = iris.data
y = iris.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = GaussianNB()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of the Naive Bayes classifier: {accuracy:.2f}")
plt.figure(figsize=(8, 6))
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap=plt.cm.Set1, marker='o', label='True Labels',
alpha=0.6)
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_pred, cmap=plt.cm.Set2, marker='x', label='Predicted Labels',
alpha=0.6)
plt.title("Naive Bayes Classification")
plt.xlabel("Sepal Length")
plt.ylabel("Sepal Width")
plt.legend()
plt.colorbar(label='Species')
plt.grid(True)
plt.show()
print("\nSample Predictions:")
for _ in range(5):
   print(f"Predicted: {iris.target_names[y_pred[i]]}, Actual: {iris.target_names[y_test[i]]}")
```

Exp:7 PCA

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler

def pca(X, num_components):
    scaler = StandardScaler()
    X_std = scaler.fit_transform(X)
    covariance_matrix = np.cov(X_std, rowvar=False)
    eigenvalues, eigenvectors = np.linalg.eigh(covariance_matrix)
    sorted_indices = np.argsort(eigenvalues)[::-1]
    eigenvalues = eigenvalues[sorted_indices]
    eigenvectors = eigenvectors[:, sorted_indices]
    principal_components = eigenvectors[:, :num_components]
    X_pca = np.dot(X_std, principal_components)
    return X_pca, eigenvalues[:num_components], principal_components

def main():
    np.random.seed(25)
    X = np.random.rand(35, 2)
    num_components = 2
    X_pca, eigenvalues, principal_components = pca(X, num_components)
    plt.scatter(X_pca[:, 0], X_pca[:, 1], alpha=0.7, edgecolors='k')
    plt.xlabel('Principal Component 1')
    plt.ylabel('Principal Component 2')
    plt.title('PCA Projection')
    plt.show()
    print("Eigenvalues:", eigenvalues)
    print("Principal Components:\n", principal_components)
if __name__ == "__main__":
    main()
```

Exp:8 K-Means Clustering

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler

iris = load_iris()
X = iris.data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
k = 3
kmeans = KMeans(n_clusters=k, random_state=100, n_init=10)
y_kmeans = kmeans.fit_predict(X_scaled)
plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=y_kmeans, cmap='viridis', edgecolors='k', alpha=0.9)
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s=50, c='red', marker='X',
label='Centroids')
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.title("K-Means Clustering on Iris Dataset")
plt.legend()
plt.show()
```

Exp:9 Matplotlib,Seaborn

```
import matplotlib.pyplot as plt
x = [1, 2, 3, 4, 5]
y = [10, 20, 25, 30, 40]
plt.plot(x, y, marker='X', linestyle='-', color='r', label="Line Plot")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.title("Simple Line Plot")
plt.legend()
plt.show()
```

```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
data = pd.DataFrame({
   'Name': ['Virat Kohli', 'Klassen', 'Rohit Sharma', 'Dhoni', 'Sam Curren'],
   'Total_Score': [8000, 1027, 6628, 5243, 1774]
})
sns.barplot(x='Name', y='Total_Score', data=data, palette='Reds')
plt.title("Seaborn Bar Plot")
plt.show()
```

Exp:10 : Getting & Setting Values in Graph using python

```
import networkx as nx
import matplotlib.pyplot as plt
G=nx.Graph()
G.add_node(1, label='A')
G.add_node(2, label='B')
G.add_node(3, label='C')

G.add_edge(1, 2, weight=5)
G.add_edge(2, 3, weight=3)
G.add_edge(1, 3, weight=8)
print("Label of node 1:", G.nodes[1]['label'])
print("Weight between 1 and 2:", G[1][2]['weight'])
G.nodes[1]['label'] = 'X'
G[1][2]['weight'] = 10
pos = nx.spring_layout(G)
nx.draw_networkx_nodes(G, pos, node_color='skyblue', node_size=1000)
nx.draw_networkx_edges(G, pos, width=2)
labels = nx.get_node_attributes(G, 'label')
nx.draw_networkx_labels(G, pos, labels, font_size=14)
edge_labels = nx.get_edge_attributes(G, 'weight')
nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels)
plt.title("Graph with Node Labels and Edge Weights")
plt.axis('off')
plt.show()
```