TASK – 1

```
# Importing necessary libraries
import pandas as pd      # Data manipulation
import numpy as np       # Numerical computations
import matplotlib.pyplot as plt  # Basic plotting
import seaborn as sns    # Advanced visualization

print("Libraries imported successfully.")

import numpy as np

# Creating Matrices
arr_zeros = np.zeros((3, 3))
print("Matrix of Zeros:\n", arr_zeros)
arr_ones = np.ones((2, 2))
print("Matrix of Ones:\n", arr_ones)
arr_full = np.full((1, 6), 5)
print("Full Matrix:\n", arr_full)
arr_eye = np.eye(4)
print("Identity Matrix:\n", arr_eye)
arr_range = np.arange(0, 14, 3)
print("Matrix Range:\n", arr_range)
arr_l = np.linspace(0, 6, 4)
print("Evenly distributed arrays:\n", arr_l)
print()

# Arithmetic operations
arr = np.array([[1, 2, 3], [4, 5, 6]])
print("Addition:\n", arr + 2)
print("Multiplication:\n", arr * 5)
```

```
# Universal functions
print("Square root:\n", np.sqrt(arr))

# Shape and Reshape
print("Shape:", arr.shape)
print("Reshaped:\n", arr.reshape(1, 6))

# Indexing and Slicing
print("Specific element:", arr[1, 2])
print("Conditional Selection:\n", arr[arr > 3])
```

```
Matrix of Zeros:
 [[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
Matrix of Ones:
 [[1. 1.]
 [1. 1.]]
Full Matrix:
 [[5 5 5 5 5 5]]
Identity Matrix:
 [[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]
Matrix Range:
 [ 0  3  6  9 12]
Evenly distributed arrays:
 [0.   2.  4.  6. ]

Addition:
```

```
 [[3 4 5]
 [6 7 8]]
Multiplication:
 [[ 5 10 15]
 [20 25 30]]
Square root:
 [[1.         1.41421356 1.73205081]
 [2.         2.23606798 2.44948974]]
Shape: (2, 3)
Reshaped:
 [[1 2 3 4 5 6]]
Specific element: 6
Conditional Selection:
 [4 5 6]
```

```
import pandas as pd

# Creating Series
data = [1, 2, 3, 4, 5]
serie = pd.Series(data)

# Custom index
index = ['a', 'b', 'c', 'd', 'e']
serie = pd.Series(data, index=index)

# DataFrame Creation
data = {
    "Name": ['Kitty', 'Bob', 'Alan'],
    "Age": [21, 20, 23],
    "City": ['LA', 'NY', 'CH']
}
```

```
df = pd.DataFrame(data)
print(df.head(1))
print(df.describe())
print(df.info())

import matplotlib.pyplot as plt
import seaborn as sns

# Simple Plot
x = [1, 2, 3, 4, 5]
y = [10, 20, 30, 40, 50]
plt.plot(x, y, marker='o')
plt.title("Sample Plot")
plt.show()

import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Plotting the box plot
sns.boxplot(x="day", y="total_bill", data=tips)
plt.title("Box Plot of Total Bill by Day")
plt.show()

from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
```

```
# Loading the dataset
df = pd.read_csv('Iris.csv')
print(df.head())

# Label Encoding
encoder = LabelEncoder()
df['Species'] = encoder.fit_transform(df['Species'])

# Standard Scaling
scaler = StandardScaler()
df_scaled = pd.DataFrame(scaler.fit_transform(df.drop(columns=['Id', 'Species'])))

# Histogram
plt.hist(df['SepalLengthCm'], bins=20, edgecolor='black')
plt.title('Sepal Length Distribution')
plt.xlabel('Sepal Length')
plt.ylabel('Frequency')
plt.show()

# Scatter Plot
sns.scatterplot(x='SepalLengthCm', y='SepalWidthCm', hue='Species', data=df)
plt.show()

# Heatmap for Correlation
corr_matrix = df.drop(columns=['Id']).corr()
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```

TASK – 2

```
# Import necessary libraries
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

# Step 1: Load the dataset
df = pd.read_csv('Iris.csv')

# Step 2: Display first few rows and dataset info
print("DataFrame Head:\n", df.head())
print("DataFrame Info:\n", df.info())

# Step 3: One-Hot Encoding for categorical columns
df_encoded = pd.get_dummies(df, columns=['Species'], drop_first=True)
print("Encoded DataFrame:\n", df_encoded.head())

# Step 4: Standardize the features
scaler = StandardScaler()
x = df_encoded.drop(columns=['Id'])
x_scaled = scaler.fit_transform(x)

# Step 5: Apply PCA for dimensionality reduction
pca = PCA(n_components=2)
x_pca = pca.fit_transform(x_scaled)

# Step 6: Create PCA DataFrame and add 'Species' column
df_pca = pd.DataFrame(x_pca, columns=['PC1', 'PC2'])
df_pca['Species'] = df['Species']
```

```
# Step 7: Visualize original data
sns.scatterplot(x='SepalLengthCm', y='SepalWidthCm', hue='Species', data=df)
plt.title('Original Data: Sepal Length vs Sepal Width')
plt.xlabel('Sepal Length')
plt.ylabel('Sepal Width')
plt.show()

# Step 8: Visualize PCA results
sns.scatterplot(x='PC1', y='PC2', hue='Species', data=df_pca)
plt.title('PCA: Principal Component 1 vs Principal Component 2')
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.show()
```

**RFE**

```
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.feature_selection import RFE
from sklearn.metrics import accuracy_score

# Load dataset
df = pd.read_csv('Iris.csv')
print(df.head())
print(df.info())

# Preprocessing
X = df.drop(columns=['Species'])
```

```
y = df['Species']

# Standardize and split data
X_scaled = StandardScaler().fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=40)

# Logistic Regression model
model = LogisticRegression(max_iter=10000)

# Evaluate accuracy before RFE
model.fit(X_train, y_train)
xpred=model.predict(X_test)
acc_b = accuracy_score(y_test, xpred)
print("Accuracy before RFE:", acc_b)

# Feature selection with RFE
rfe = RFE(model, n_features_to_select=2)
X_train_rfe = rfe.fit_transform(X_train, y_train)
X_test_rfe = rfe.transform(X_test)

# Print number of selected features
print("Number of selected features:", sum(rfe.support_))

# Evaluate accuracy after RFE
model.fit(X_train_rfe, y_train)
xpred=model.predict(X_test_rfe)
acc_a = accuracy_score(y_test, xpred)
print("Accuracy after RFE:", acc_a)

# Print selected features
print("Selected features:", X.columns[rfe.support_])
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Load and preprocess data
df = pd.read_csv('Pune_rent.csv')
df['price'] = pd.to_numeric(df['price'].str.replace(",", ''))
df.fillna(df.mode().iloc[0], inplace=True)

# Encoding categorical features
label_encoder = LabelEncoder()
categorical_cols = df.select_dtypes(include=['object']).columns
for col in categorical_cols:
    df[col] = label_encoder.fit_transform(df[col])

# Feature selection
X = df[['bathroom', 'bedroom', 'area', 'locality', 'furnish_type']]
y = df['price']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Scaling features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train model
model = LinearRegression()
```

```
model.fit(X_train_scaled, y_train)

# Predictions and evaluation
y_pred = model.predict(X_test_scaled)
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Display metrics
print(f'Mean Squared Error:{mse:.2f}')
print(f'Mean Absolute Error:{mae:.2f}')
print(f'Root Mean Squared Error:{np.sqrt(mse):.2f}')
print(f'R-Squared:{r2:.2f}')
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Step 1: Load the dataset
df = pd.read_csv('User_Data.csv')

# Step 2: Data Preprocessing
# Convert 'Gender' column to numerical using Label Encoding
label_encoder = LabelEncoder()
df['Gender'] = label_encoder.fit_transform(df['Gender'])

# Fill missing values if any (Optional based on dataset)
df.fillna(df.mode().iloc[0], inplace=True)

# Step 3: Feature selection and target variable
X = df[['Gender', 'EstimatedSalary', 'Purchased']]  # Independent variables
y = df['Age']  # Target variable

# Step 4: Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 5: Feature Scaling (Normalize the features)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Step 6: Train the Linear Regression model
model = LinearRegression()
```

```
model.fit(X_train_scaled, y_train)

# Step 7: Make Predictions
y_pred = model.predict(X_test_scaled)

# Step 8: Evaluate Model Performance
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Step 9: Print Evaluation Metrics
print(f'Mean Squared Error (MSE): {mse:.2f}')
print(f'Mean Absolute Error (MAE): {mae:.2f}')
print(f'Root Mean Squared Error (RMSE): {np.sqrt(mse):.2f}')
print(f'R-Squared (R²): {r2:.2f}')
```

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Step 1: Load the Iris dataset
iris = load_iris()
df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
df['species'] = iris.target  # Add target variable

# Step 2: Prepare the features and target
X = df.drop('species', axis=1)  # Features (input variables)
y = df['species']  # Target variable

# Step 3: 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)

# Step 4: Create and train the model
model = DecisionTreeClassifier()
model.fit(X_train, y_train)

# Step 5: Make predictions
y_pred = model.predict(X_test)

# Step 6: Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
print("Classification Report:\n", classification_report(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```python
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Step 1: Load the Iris dataset
iris = load_iris()
df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
df['species'] = iris.target  # Add target variable

# Step 2: Prepare the features and target
X = df.drop('species', axis=1)  # Features (input variables)
y = df['species']  # Target variable

# Step 3: 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)

# Step 4: Create and train the KNN model
knn = KNeighborsClassifier(n_neighbors=3)  # You can adjust the number of neighbors
knn.fit(X_train, y_train)

# Step 5: Make predictions
y_pred = knn.predict(X_test)

# Step 6: Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
print("Classification Report:\n", classification_report(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
# Step 1: Load the dataset
df = pd.read_csv('loan_data_set.csv')

# Step 2: Prepare features and target
X = df.drop(columns=['Loan_Status'])
y = df['Loan_Status']

# Step 3: Encode categorical features
label_encoder = LabelEncoder()
X_encoded = X.apply(label_encoder.fit_transform)

# Step 4: Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_encoded, y, test_size=0.2, random_state=42)

# Step 5: Create and train the Random Forest model
rf = RandomForestClassifier()
rf.fit(X_train, y_train)

# Step 6: Make predictions
y_pred = rf.predict(X_test)

# Step 7: Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Random Forest Accuracy: {accuracy:.2f}")
print("Classification Report:\n", classification_report(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import adjusted_rand_score, silhouette_score
import pandas as pd

# Load the dataset
iris = load_iris()
df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
df['species'] = iris.target

# Preprocessing: Standardize the data
scaler = StandardScaler()
x_scaled = scaler.fit_transform(df.drop(columns='species'))

# Apply K-Means Clustering
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(x_scaled)
df['cluster'] = kmeans.labels_

# Performance Measures: Adjusted Rand Index and Silhouette Score
ari = adjusted_rand_score(df['species'], df['cluster'])
silhouette = silhouette_score(x_scaled, df['cluster'])

print("Cluster Centers:\n", kmeans.cluster_centers_)
print("Adjusted Rand Index:", ari)
print("Silhouette Score:", silhouette)
```

```python
from scipy.cluster.hierarchy import dendrogram, linkage, fcluster
from sklearn.metrics import adjusted_rand_score, silhouette_score
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import load_iris

# Load the dataset
iris = load_iris()
df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
df['species'] = iris.target

# Preprocessing: Standardize the data
scaler = StandardScaler()
x_scaled = scaler.fit_transform(df.drop(columns='species'))

# Apply Hierarchical Clustering (Ward's method)
linkage_matrix = linkage(x_scaled, method='ward')

# Plot Dendrogram
dendrogram(linkage_matrix, labels=iris.target)
plt.title('Dendrogram')
plt.xlabel('Samples')
plt.ylabel('Distance')
plt.show()

# Get clusters
clusters = fcluster(linkage_matrix, t=3, criterion='maxclust')
df['cluster'] = clusters

# Performance Measures: Adjusted Rand Index and Silhouette Score
```

```
ari = adjusted_rand_score(df['species'], df['cluster'])
silhouette = silhouette_score(x_scaled, df['cluster'])

print("Cluster Distribution:\n", df.groupby(['cluster', 'species']).size())
print("Adjusted Rand Index:", ari)
print("Silhouette Score:", silhouette)
```