Guru Gobind Singh Indraprastha University

University School of Automation & Robotics

Data Analytics Lab (ARI351)

Practical File

Name	Sujal Singh
Enrollment Number	04119051723
Batch	IIOT–B1–23
Submitted To	Prof. Anuj Kumar

Index

S. No.	Date	Name of Experiment	Signature
1.		Numpy Basics.	
2.		Data Visualization using Matplotlib.	
3.		Pandas Data Handling.	
4.		Logistic Regression (Classification).	
5.		Data Preprocessing Techniques.	
6.		Classification Algorithm Comparison.	
7.		Regression.	
8.		Clustering.	

Experiment–1

Aim:

To understand and apply various data pre-processing techniques.

Code:

```
import numpy as np
print("NumPy Version:", np.version )
print("="*50)

# 1. Creating Arrays
print("1. Creating Arrays")
print("-"*30)

# From Python list
arr1 = np.array([1, 2, 3, 4, 5])
print("From list:", arr1)
print("Type:", type(arr1))
print("Data type:", arr1.dtype)

# 2D Array (Matrix)
arr2d = np.array([[1, 2, 3], [4, 5, 6]])
print("\n2D Array:")
print(arr2d)

# Zeros, Ones, Empty
zeros = np.zeros(5)
ones = np.ones((3, 4))
empty = np.empty(4) # uninitialized values
print("\nZeros:", zeros)
print("Ones (3x4) :\n", ones)

# Range of numbers
range_arr = np.arange(0, 20, 2) # start, stop, step
print("arange(0,20,2) :", range_arr)
# Evenly spaced numbers
linspace_arr = np.linspace(0, 1, 5) # 5 numbers from 0 to 1
print("linspace(0,1,5) :", linspace_arr)

# Random numbers
print("\nRandom Arrays:")
print("Random (0-1) :", np.random.random(5))
print("Random integers :", np.random.randint(1, 100, size=6))
print("="*50)

# 2. Array Attributes
print("2. Array Attributes")
```

```
print("-"*30)
print("Shape:", arr2d.shape)
print("Size (elements):", arr2d.size)
print("Dimensions:", arr2d.ndim)
print("Item size (bytes):", arr2d.itemsize)
print("Data type:", arr2d.dtype)
print("="*50)

# 3. Basic Operations
print("3. Basic Operations (Element-wise)")
print("-"*30)
a = np.array([10, 20, 30, 40])
b = np.array([1, 2, 3, 4])
print("a:", a)
print("b:", b)
print("a + b:", a + b)
print("a - b:", a - b)
print("a * b:", a * b)
print("a / b:", a / b)
print("a ** 2:", a ** 2)
print("sin(a):", np.sin(a))
print("sqrt(a):", np.sqrt(a))
print("="*50)

# 4. Indexing & Slicing
print("4. Indexing and Slicing")
print("-"*30)
arr = np.array([10, 20, 30, 40, 50, 60])
print("arr:", arr)
print("First element :", arr[0])
print("Last element :", arr[-1])
print("Slice [1:4] :", arr[1:4])
print("Every 2nd:", arr[::2])
print("Reverse:", arr[::-1])

# 2D Indexing
matrix = np.array([[1, 2, 3],
[4, 5, 6],
[7, 8, 9]])
print("\n2D Matrix:\n", matrix)
print("Element [1,2] :", matrix[1, 2]) # row 1, col 2 → 6
print("First row:", matrix[0])
print("First column:", matrix[:, 0])
print("Submatrix 2x2:\n", matrix[:2, :2])
print("="*50)

# 5. Reshaping Arrays
print("5. Reshaping")
```

```
print("-"*30)
arr = np.arange(1, 13)
print("Original:", arr)
reshaped = arr.reshape(3, 4) # 3 rows, 4 columns
print("Reshaped 3x4: \n", reshaped)

# Flatten back
flattened = reshaped.flatten()
print("Flattened:", flattened)
print("="*50)

# 6. Mathematical & Statistical Functions
print("6. Math & Stats Functions")
print("-"*30)
data = np.array([12, 15, 18, 22, 30, 35])
print("Data:", data)
print("Mean:", np.mean(data))
print("Median:", np.median(data))
print("Standard Dev:", np.std(data))
print("Variance:", np.var(data))
print("Min:", np.min(data))
print("Max:", np.max(data))
print("Sum:", np.sum(data))
print("Cumulative Sum :", np.cumsum(data))
print("="*50)
```

Output:

```
NumPy Version: <module 'numpy.version' from
 ↪  '/tmp/.venv/lib64/python3.14/site-packages/numpy/version.py'>
==================================================
1. Creating Arrays
------------------------------
From list: [1 2 3 4 5]
Type: <class 'numpy.ndarray'>
Data type: int64

2D Array:
[[1 2 3]
 [4 5 6]]

Zeros: [0. 0. 0. 0. 0.]
Ones (3x4) :
 [[1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]]
arange(0,20,2) : [ 0  2  4  6  8 10 12 14 16 18]
linspace(0,1,5) : [0.   0.25 0.5  0.75 1.  ]

Random Arrays:
```

```
Random (0-1) : [0.50083525 0.09400569 0.44989897 0.74831388 0.57466536]
Random integers : [10 69  1 65 17  9]
==================================================
2. Array Attributes
------------------------------
Shape: (2, 3)
Size (elements): 6
Dimensions: 2
Item size (bytes): 8
Data type: int64
==================================================
3. Basic Operations (Element-wise)
------------------------------
a: [10 20 30 40]
b: [1 2 3 4]
a + b: [11 22 33 44]
a - b: [ 9 18 27 36]
a * b: [ 10  40  90 160]
a / b: [10. 10. 10. 10.]
a ** 2: [ 100  400  900 1600]
sin(a): [-0.54402111  0.91294525 -0.98803162  0.74511316]
sqrt(a): [3.16227766 4.47213595 5.47722558 6.32455532]
==================================================
4. Indexing and Slicing
------------------------------
arr: [10 20 30 40 50 60]
First element : 10
Last element : 60
Slice [1:4] : [20 30 40]
Every 2nd: [10 30 50]
Reverse: [60 50 40 30 20 10]

2D Matrix:
 [[1 2 3]
 [4 5 6]
 [7 8 9]]
Element [1,2] : 6
First row: [1 2 3]
First column: [1 4 7]
Submatrix 2x2:
 [[1 2]
 [4 5]]
==================================================
5. Reshaping
------------------------------
Original: [ 1  2  3  4  5  6  7  8  9 10 11 12]
Reshaped 3x4:
 [[ 1  2  3  4]
```

```
 [ 5  6  7  8]
 [ 9 10 11 12]]
Flattened: [ 1  2  3  4  5  6  7  8  9 10 11 12]
==================================================
6. Math & Stats Functions
------------------------------
Data: [12 15 18 22 30 35]
Mean: 22.0
Median: 20.0
Standard Dev: 8.144527815247077
Variance: 66.33333333333333
Min: 12
Max: 35
Sum: 132
Cumulative Sum : [ 12  27  45  67  97 132]
==================================================
```

Experiment–2

Aim:

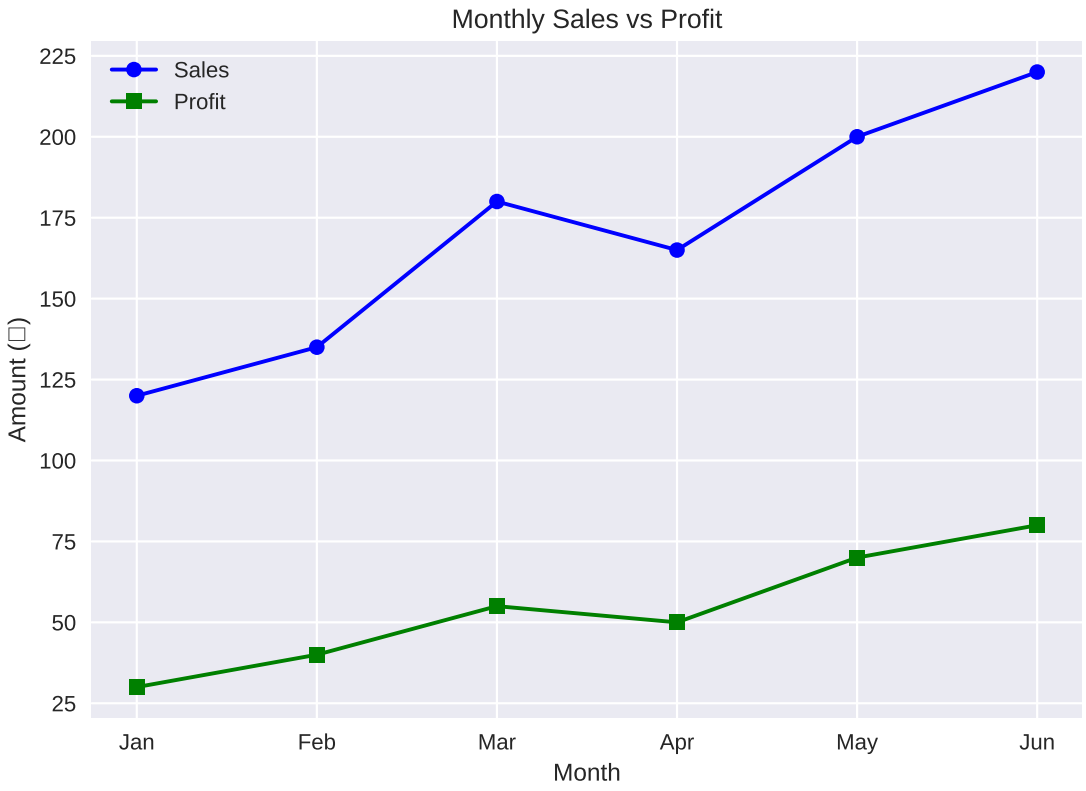
To understand basic matplotlib for plotting bar, line, pie, histogram etc.

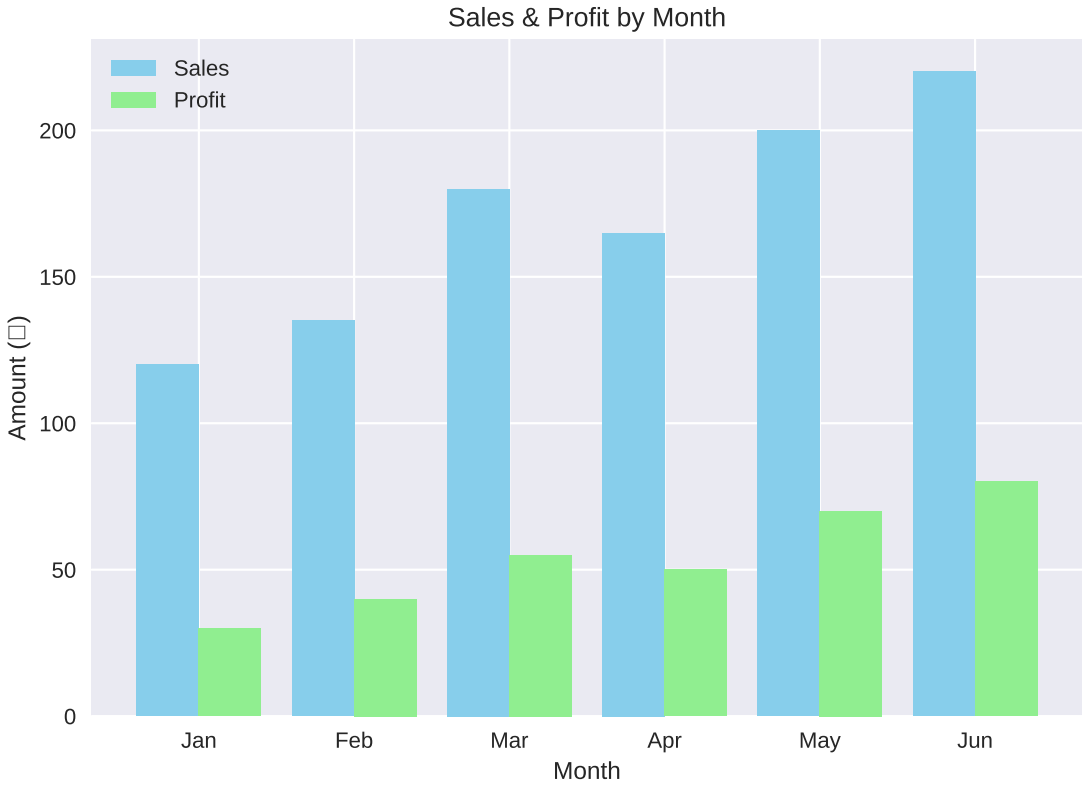
Code:

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

plt.style.use('seaborn-v0_8')

months = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun']
sales = [120, 135, 180, 165, 200, 220]
profit = [30, 40, 55, 50, 70, 80]
categories = ['Rent', 'Food', 'Transport', 'Entertainment', 'Savings']
expenses = [5000, 3000, 1500, 2000, 4000]
ages = np.random.normal(35, 10, 1000)

# Line Plot plt.figure(figsize=(10,6))
plt.plot(months, sales, 'o-b', label='Sales')
plt.plot(months, profit, 's-g', label='Profit')
plt.title('Monthly Sales vs Profit')
plt.xlabel('Month')
plt.ylabel('Amount ()')
plt.legend()
plt.grid(True)
plt.show()

# Bar Plot plt.figure(figsize=(10,6))
x = np.arange(len(months))
plt.bar(x-0.2, sales, 0.4, label='Sales', color='skyblue')
plt.bar(x+0.2, profit, 0.4, label='Profit', color='lightgreen')
plt.title('Sales & Profit by Month')
plt.xlabel('Month')
plt.ylabel('Amount ()')
plt.xticks(x, months)
plt.legend()
plt.show()

# Horizontal Bar plt.figure(figsize=(10,6))
plt.barh(categories, expenses, color='coral')
plt.title('Monthly Expenses')
plt.xlabel('Amount ()')
plt.show()

# Pie Chart plt.figure(figsize=(8,8))
plt.pie(expenses, labels=categories, autopct='%1.1f%%', startangle=90)
```

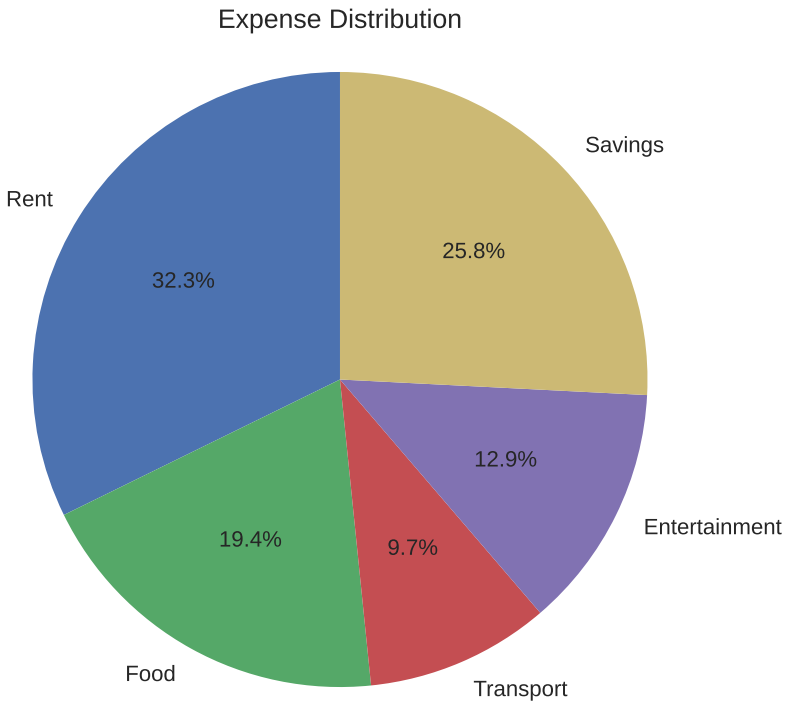
```
plt.title('Expense Distribution')
plt.axis('equal')
plt.show()

# Histogram plt.figure(figsize=(10,6))
plt.hist(ages, bins=30, color='purple', edgecolor='black', alpha=0.7)
plt.title('Age Distribution')
plt.xlabel('Age')
plt.ylabel('Count')
plt.show()
# Scatter Plot plt.figure(figsize=(10,6))
plt.scatter(np.random.rand(100), np.random.rand(100), c=np.random.rand(100),
    ↪ s=200, alpha=0.6, cmap='viridis')
plt.title('Scatter Plot')
plt.show()

# Subplots
fig, axes = plt.subplots(2, 2, figsize=(12,10))
axes[0,0].plot(months, sales, 'o-r')
axes[0,0].set_title('Line Plot')

axes[0,1].bar(categories, expenses, color='orange')
axes[0,1].set_title('Bar Plot')

axes[1,0].pie(expenses, labels=categories, autopct='%1.0f%%')
axes[1,0].set_title('Pie Chart')
axes[1,1].hist(ages, bins=25, color='teal')
axes[1,1].set_title('Histogram')

plt.tight_layout()
plt.show()
```

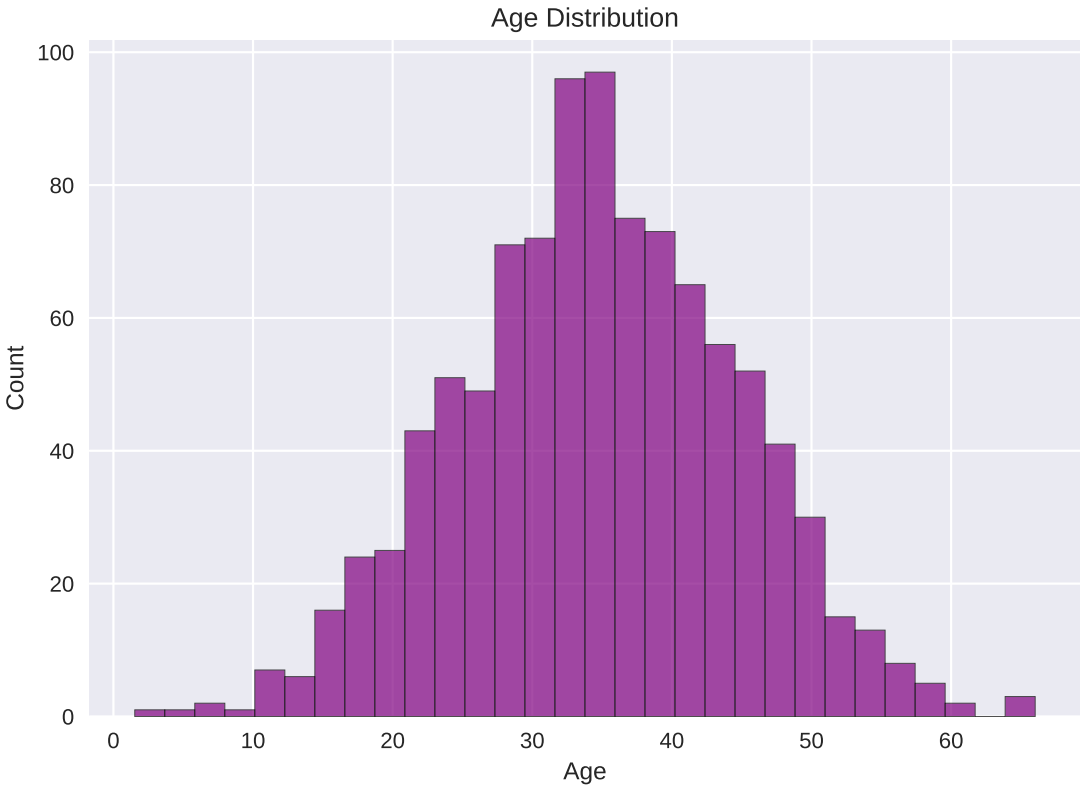
Plots:

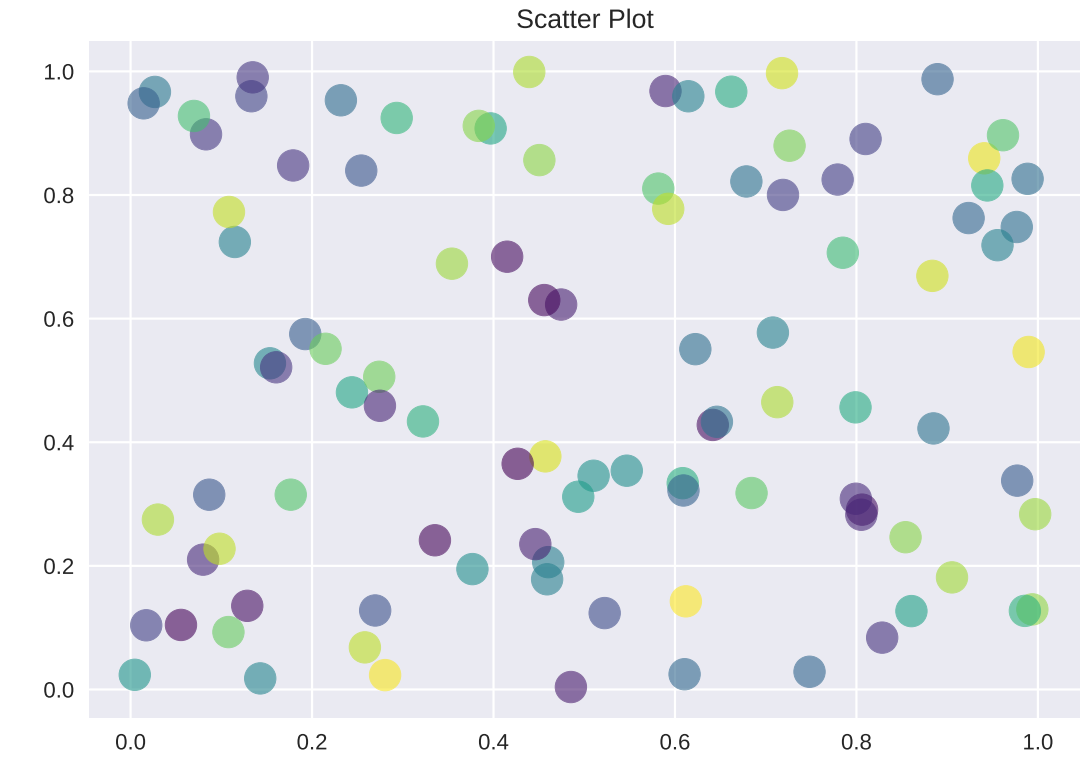

Monthly Sales vs Profit
Sales
Profit
225
200
175
150
125
100
75
50
25
Amount (□)
Jan
Feb
Mar
Apr
May
Jun
Month

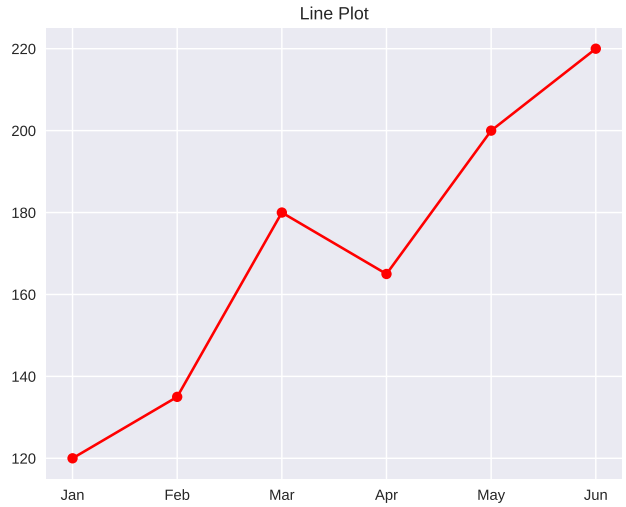

Sales & Profit by Month
Sales
Profit
200
150
100
50
0
Amount (□)
Jan
Feb
Mar
Apr
May
Jun
Month

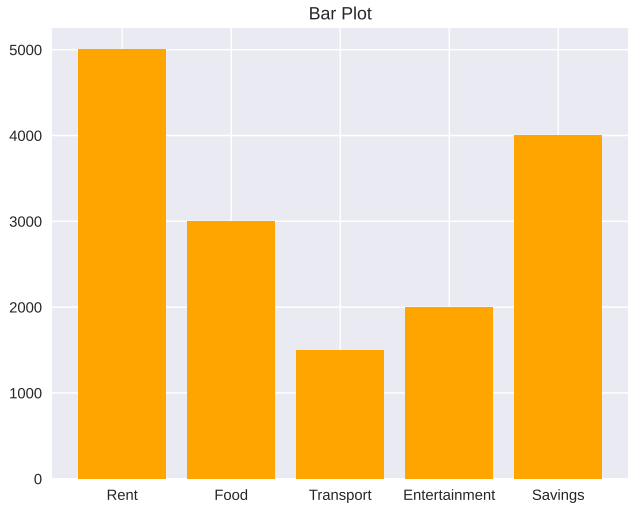

Monthly Expenses
Savings
Entertainment
Transport
Food
Rent
0
1000
2000
3000
4000
5000
Amount (□)

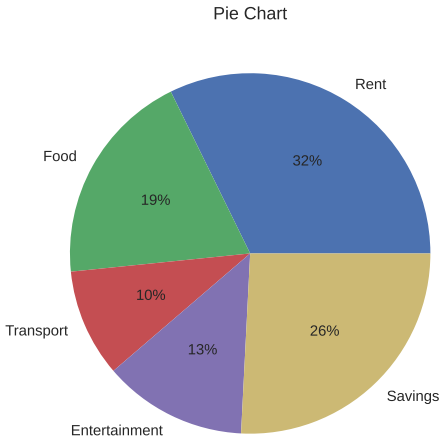

Expense Distribution
Savings
25.8%
Rent
32.3%
12.9%
Entertainment
19.4%
9.7%
Food
Transport

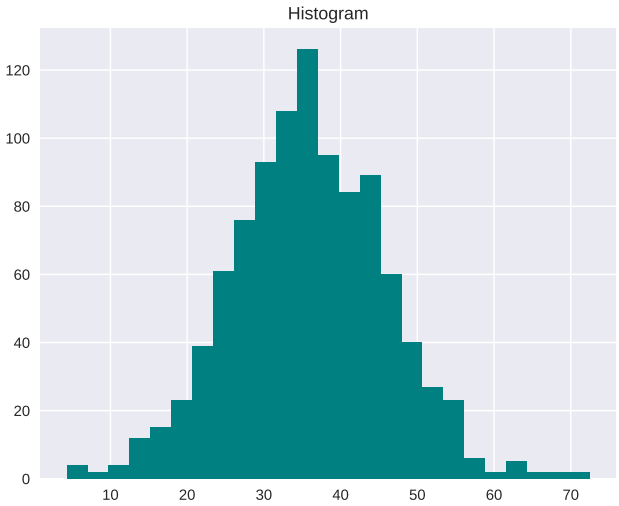

Age Distribution
Count
Age
0
20
40
60
80
100
0
10
20
30
40
50
60


Scatter Plot
0.0
0.2
0.4
0.6
0.8
1.0


Line Plot
220
200
180
160
140
120
Jan
Feb
Mar
Apr
May
Jun


Bar Plot
5000
4000
3000
2000
1000
0
Rent
Food
Transport
Entertainment
Savings


Pie Chart
Rent
32%
Food
19%
10%
Transport
13%
26%
Savings
Entertainment


Histogram
120
100
80
60
40
20
0
10
20
30
40
50
60
70

Experiment–3

Aim:

Understanding data preprocessing using Pandas.

Code:

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

df = pd.read_csv("./iris.csv")
print("Shape (rows, columns) :", df.shape)
print("Total samples:", df.shape[0])
print("Total features:", df.shape[1])
print("\nColumns:")
print(df.columns.tolist())

print("\nFirst 5 rows:")
print(df.head())

print("\nData types & missing values:")
print(df.info())

print("\nMissing values per column:")
print(df.isnull().sum())

target_col = df.columns[-1] # assumes last column is target
print(f"\nTarget column: '{target_col}'")
print("Number of classes :", df[target_col].nunique())
print("\nClass distribution:")
print(df[target_col].value_counts())

# 1. Countplot
plt.figure(figsize=(10,5))
sns.countplot(data=df, x=target_col, palette="viridis")
plt.title(f"Countplot of {target_col}")
plt.xticks(rotation=45)
plt.show()

# 2. Pairplot (only if dataset is small)
if df.shape[1] <= 12:
    sns.pairplot(df, hue=target_col, diag_kind='kde', corner=True)
    plt.show()

# 3. Correlation Heatmap (numeric columns only)
numeric_df = df.select_dtypes(include=['float64', 'int64'])
plt.figure(figsize=(10,8))
```

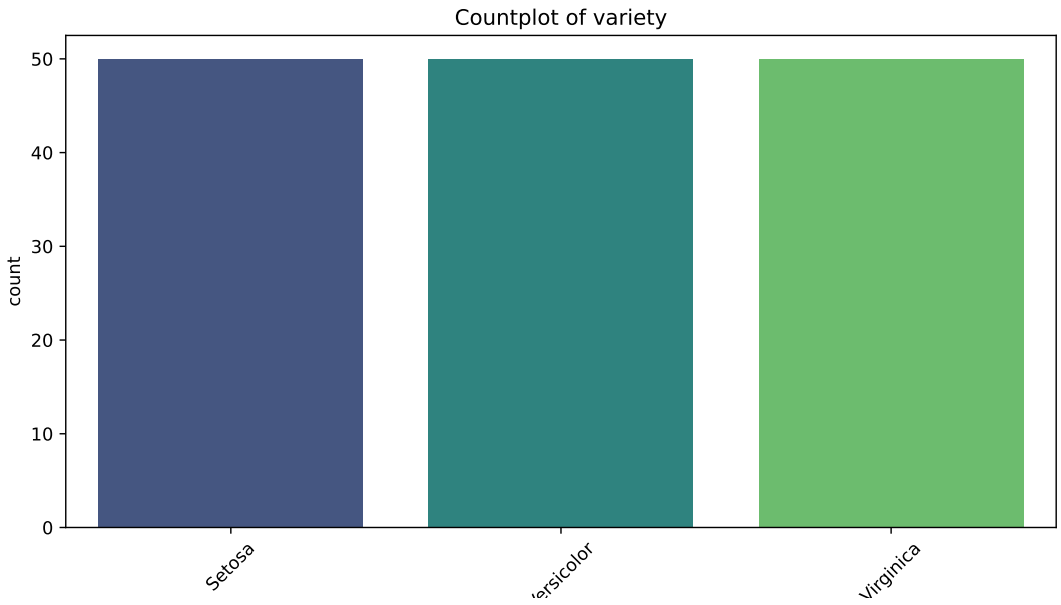
```
sns.heatmap(numeric_df.corr(), annot=True, cmap='coolwarm', center=0,
    ↪ linewidths=0.5)
plt.title("Correlation Heatmap")
plt.show()

# 4. Statistical Summary
print("\nStatistical Summary (Numeric):")
print(df.describe())
print("\nAll done! Data fully explored.")
```

Output:

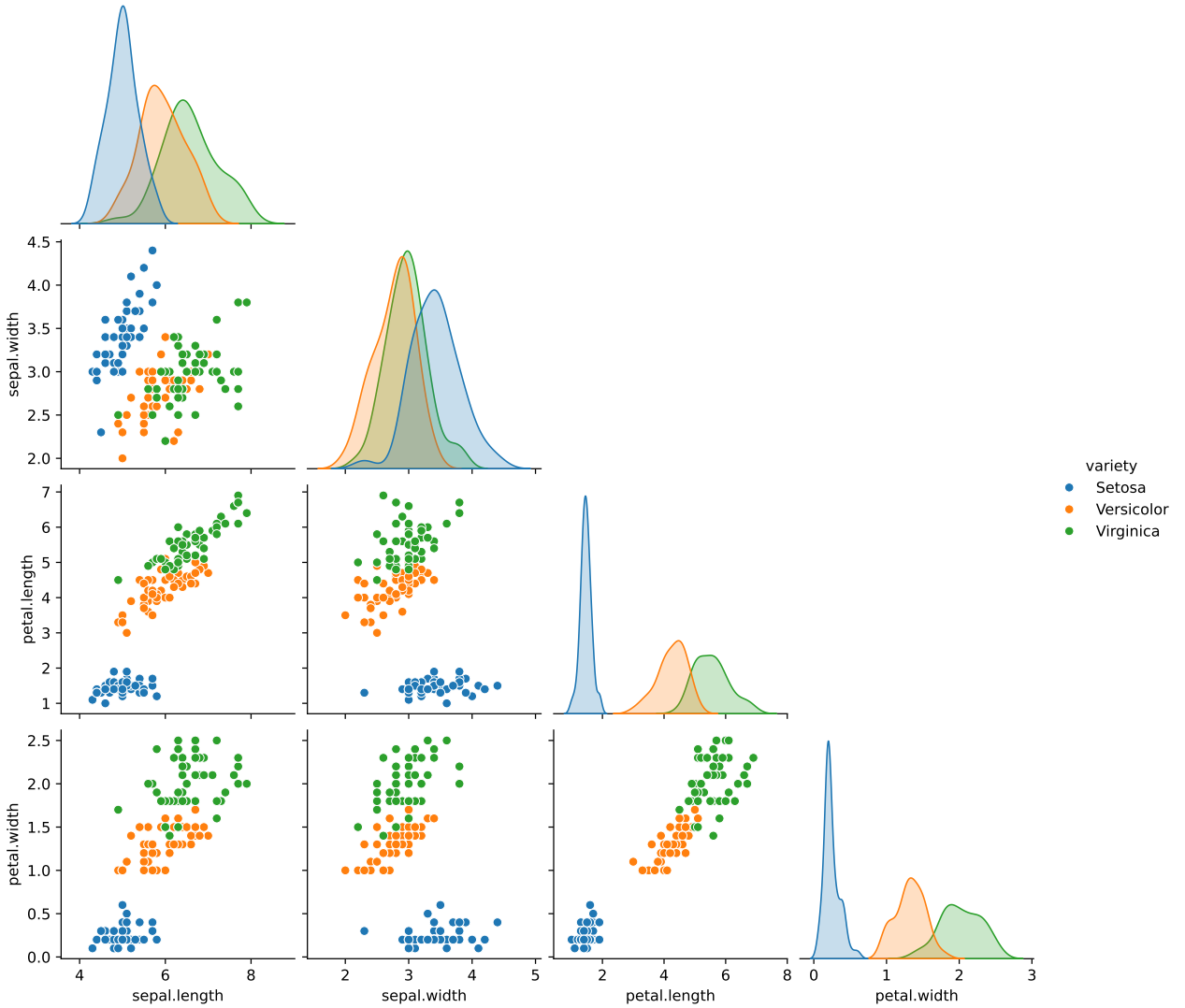
```
Shape (rows, columns) : (150, 5)
Total samples: 150
Total features: 5

Columns:
['sepal.length', 'sepal.width', 'petal.length', 'petal.width', 'variety']

First 5 rows:
   sepal.length  sepal.width  petal.length  petal.width variety
0           5.1          3.5           1.4          0.2  Setosa
1           4.9          3.0           1.4          0.2  Setosa
2           4.7          3.2           1.3          0.2  Setosa
3           4.6          3.1           1.5          0.2  Setosa
4           5.0          3.6           1.4          0.2  Setosa

Data types & missing values:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   sepal.length  150 non-null    float64
 1   sepal.width   150 non-null    float64
 2   petal.length  150 non-null    float64
 3   petal.width   150 non-null    float64
 4   variety       150 non-null    object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
None

Missing values per column:
sepal.length    0
sepal.width     0
petal.length    0
petal.width     0
variety         0
dtype: int64

```

```
Target column: 'variety'
Number of classes : 3

Class distribution:
variety
Setosa        50
Versicolor    50
Virginica     50
Name: count, dtype: int64
```

Plots:



Plots:


sepal.width
petal.length
petal.width
sepal.length
sepal.width
petal.length
petal.width
variety
Setosa
Versicolor
Virginica

Experiment–4 (a)

Aim:

Understand and apply regression algorithm.

Code:

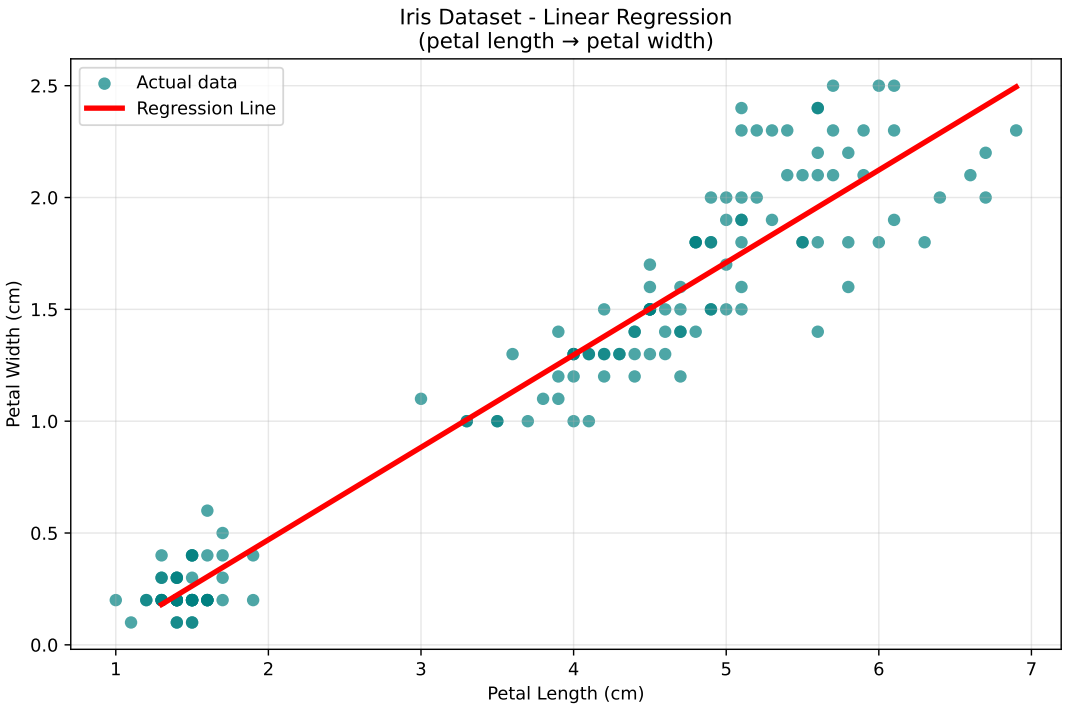
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score,
 ↪  mean_absolute_error

# Load Iris dataset correctly
iris = load_iris(as_frame=True)
df = iris.frame

# Check actual column names
print("Columns in dataset:", df.columns.tolist())
print("\nFirst 5 rows:")
print(df.head())

# Correct column names
X = df[['petal length (cm)']]
y = df['petal width (cm)']

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
 ↪  random_state=42)

# Train model
model = LinearRegression()
model.fit(X_train, y_train)

# Predict
y_pred = model.predict(X_test)

# Results
print("\n" + "="*60)
print("LINEAR REGRESSION: petal length → petal width")
print("="*60)
print(f"Slope: {model.coef_[0]:.4f}")
print(f"Intercept: {model.intercept_:.4f}")
print(f"R² Score: {r2_score(y_test, y_pred):.4f}")

```

```
# Plot
plt.figure(figsize=(10,6))
plt.scatter(X, y, color='teal', alpha=0.7, label='Actual data')
plt.plot(
    X_test.sort_values(by='petal length (cm)'),
    model.predict(X_test.sort_values(by='petal length (cm)')),
    color='red',
    linewidth=3,
    label='Regression Line'
)
plt.xlabel('Petal Length (cm)')
plt.ylabel('Petal Width (cm)')
plt.title('Iris Dataset - Linear Regression\n(petal length → petal width)')
plt.legend()
plt.grid(True, alpha=0.3)
plt.show()

# Predict new values
new = [[1.4], [4.0], [5.5]]
for length in new:
    pred = model.predict([length])
    print(f"Petal length {length[0]} cm → Predicted width: {pred[0]:.3f} cm")
```

Output:

```
Columns in dataset: ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)', 'target']

First 5 rows:
   sepal length (cm)  sepal width (cm)  ...  petal width (cm)  target
0                5.1               3.5  ...               0.2       0
1                4.9               3.0  ...               0.2       0
2                4.7               3.2  ...               0.2       0
3                4.6               3.1  ...               0.2       0
4                5.0               3.6  ...               0.2       0

[5 rows x 5 columns]

============================================================
LINEAR REGRESSION: petal length → petal width
============================================================
Slope: 0.4132
Intercept: -0.3567
R² Score: 0.9283
Petal length 1.4 cm → Predicted width: 0.222 cm
Petal length 4.0 cm → Predicted width: 1.296 cm
Petal length 5.5 cm → Predicted width: 1.916 cm
```

Plots:


Iris Dataset - Linear Regression
(petal length → petal width)
Actual data
Regression Line
Petal Width (cm)
Petal Length (cm)
0.0
0.5
1.0
1.5
2.0
2.5
1
2
3
4
5
6
7

Experiment–4 (b)

Aim:

Apply regression algorithm and handle missing value and categorical data.

Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_absolute_error

# Load Iris
iris = load_iris(as_frame=True)
df = iris.frame

# Add species name as categorical column
df['species'] = pd.Categorical.from_codes(iris.target, iris.target_names)

# Intentionally create missing values (for learning)
np.random.seed(42)
df.loc[np.random.choice(df.index, 20), 'sepal length (cm)'] = np.nan
df.loc[np.random.choice(df.index, 15), 'petal length (cm)'] = np.nan
print("Missing values created:")
print(df.isnull().sum())

# Target: predict petal_width
X = df.drop(['petal width (cm)', 'target'], axis=1)
y = df['petal width (cm)']

# Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
    ↪ random_state=42)

# Preprocessing
numeric_features = ['sepal length (cm)', 'sepal width (cm)', 'petal length
    ↪ (cm)']
categorical_features = ['species']
numeric_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='mean')),
```

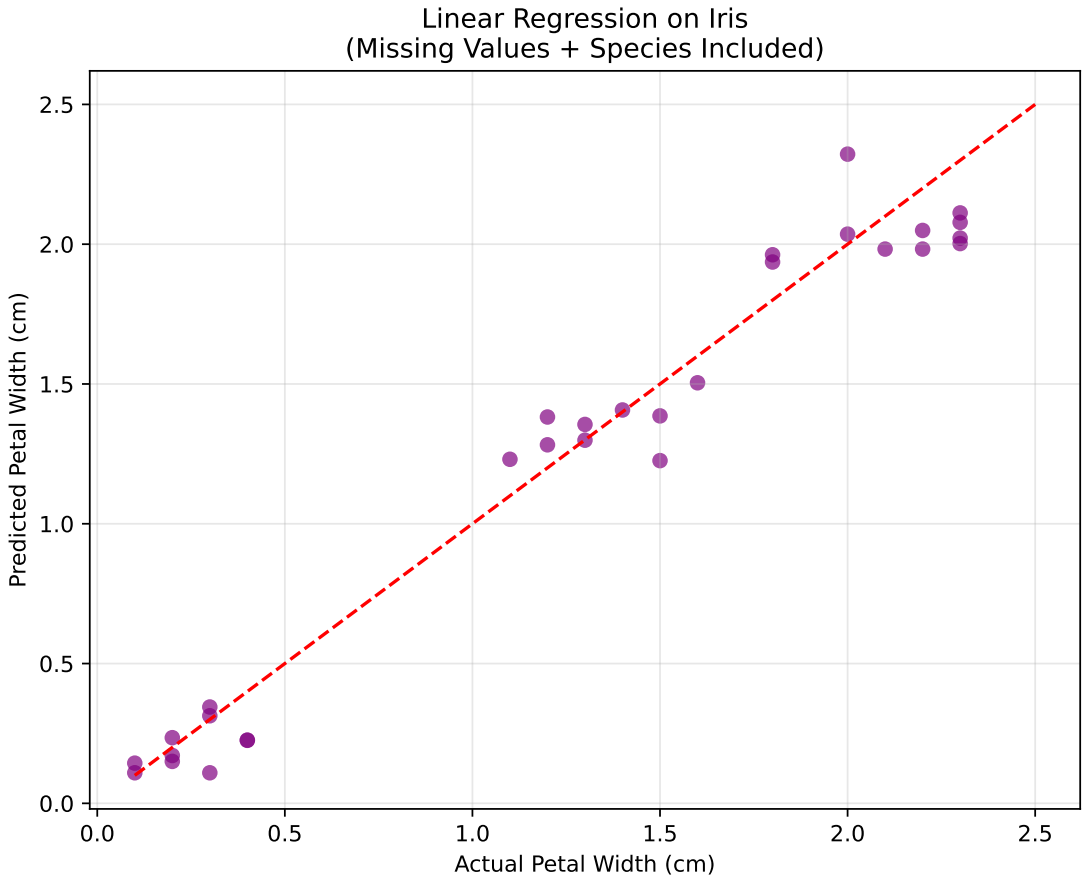
```
    ('scaler', StandardScaler())
])
categorical_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(drop='first'))
])
preprocessor = ColumnTransformer([
    ('num', numeric_pipeline, numeric_features),
    ('cat', categorical_pipeline, categorical_features)
])

# Full model
model = Pipeline([
    ('preprocessor', preprocessor),
    ('regressor', LinearRegression())
])

# Train
model.fit(X_train,y_train)

# Predict
y_pred = model.predict(X_test)

# Results
print("\n" + "="*50)
print("REGRESSION ON IRIS (with missing + categorical)")
print("="*50)
print(f"R² Score: {r2_score(y_test, y_pred):.4f}")
print(f"MAE: {mean_absolute_error(y_test, y_pred):.4f}")

# Plot
plt.figure(figsize=(8,6))
plt.scatter(y_test, y_pred, color='purple', alpha=0.7)
plt.plot([y.min(), y.max()], [y.min(), y.max()], 'r--')
plt.xlabel('Actual Petal Width (cm)')
plt.ylabel('Predicted Petal Width (cm)')
plt.title('Linear Regression on Iris\n(Missing Values + Species
 ↪  Included)')
plt.grid(True, alpha=0.3)
plt.show()

# Predict on new flower
new_flower = pd.DataFrame({
    'sepal length (cm)': [5.9],
    'sepal width (cm)': [3.0],
    'petal length (cm)': [np.nan],
    'species':
     ['virginica']
```

```
})

# missing
pred = model.predict(new_flower)[0]
print(f"\nPredicted petal width = {pred:.3f} cm")

sns.pairplot(
    df[['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)',
    ↪  'petal width (cm)', 'species']],
    diag_kind='kde'
)
plt.show()
```

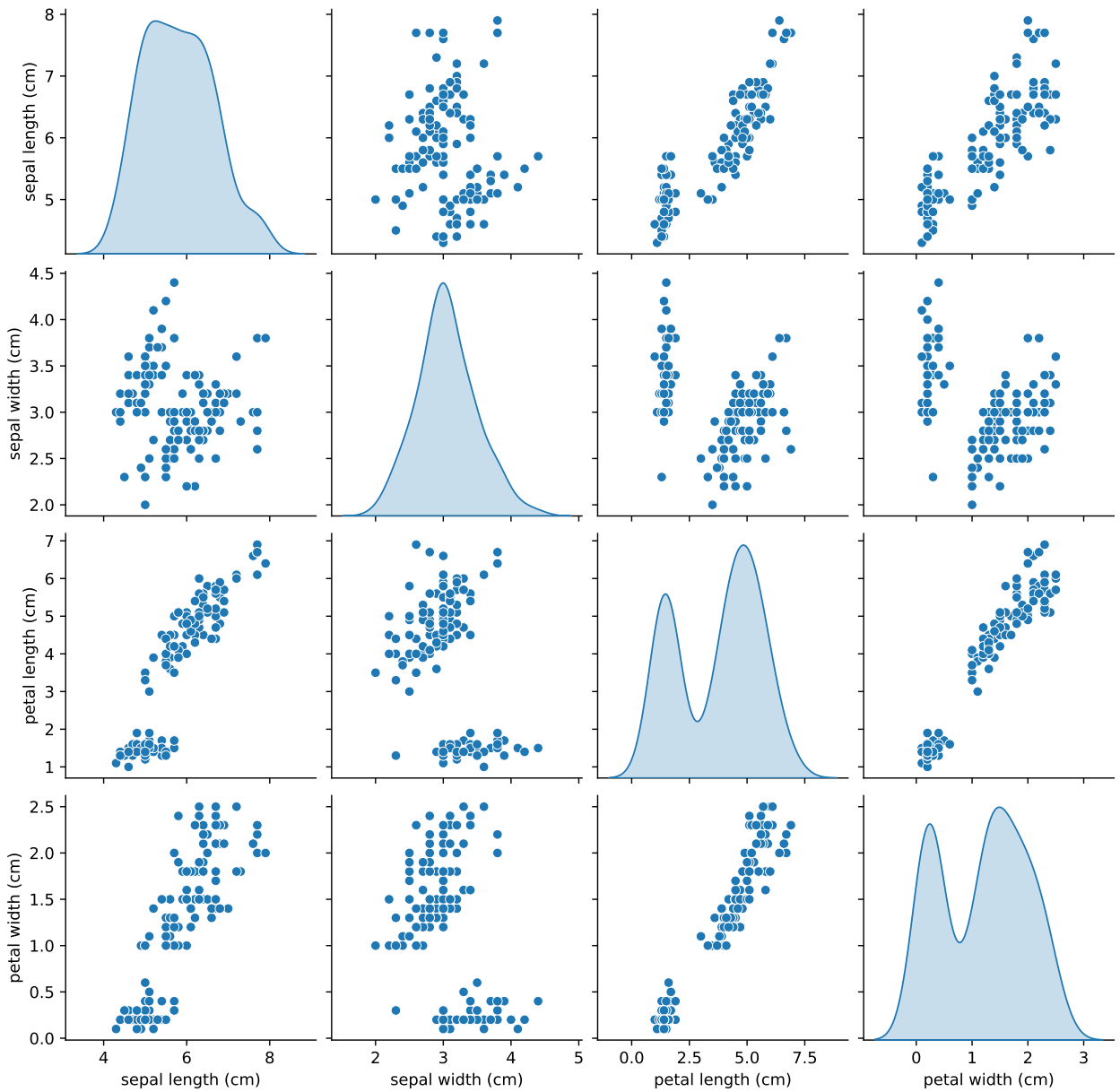
Output:

```
Missing values created:
sepal length (cm)    18
sepal width (cm)      0
petal length (cm)    13
petal width (cm)      0
target                0
species               0
dtype: int64

==================================================
REGRESSION ON IRIS (with missing + categorical)
==================================================
R² Score: 0.9608
MAE: 0.1278

Predicted petal width = 1.855 cm
```

Plot:


Linear Regression on Iris
(Missing Values + Species Included)
Predicted Petal Width (cm)
Actual Petal Width (cm)
0.0
0.5
1.0
1.5
2.0
2.5

Plot:


sepal length (cm)
sepal width (cm)
petal length (cm)
petal width (cm)

Experiment–5

Aim:

Program to implement feature scaling, feature extraction and selection.

Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.decomposition import PCA
from sklearn.feature_selection import SelectKBest, f_classif, RFE
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Load Iris
iris = load_iris(as_frame=True)
X = iris.data
y = iris.target
feature_names = iris.feature_names
print("Original Features:", feature_names)
print("Shape:", X.shape)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
    ↪ random_state=42)

# 1. No Scaling (Baseline)
model = LogisticRegression(max_iter=200)
model.fit(X_train, y_train)
acc0 = accuracy_score(y_test, model.predict(X_test))
print(f"\n1. No Scaling → Accuracy: {acc0:.4f}")

# 2. Standard Scaling (Zero mean, unit variance)
scaler_std = StandardScaler()
X_train_std = scaler_std.fit_transform(X_train)
X_test_std = scaler_std.transform(X_test)
model.fit(X_train_std, y_train)
acc1 = accuracy_score(y_test, model.predict(X_test_std))
print(f"2. StandardScaler → Accuracy: {acc1:.4f}")

# 3. MinMax Scaling (0 to 1)
scaler_mm = MinMaxScaler()
X_train_mm = scaler_mm.fit_transform(X_train)
X_test_mm = scaler_mm.transform(X_test)
model.fit(X_train_mm, y_train)
```

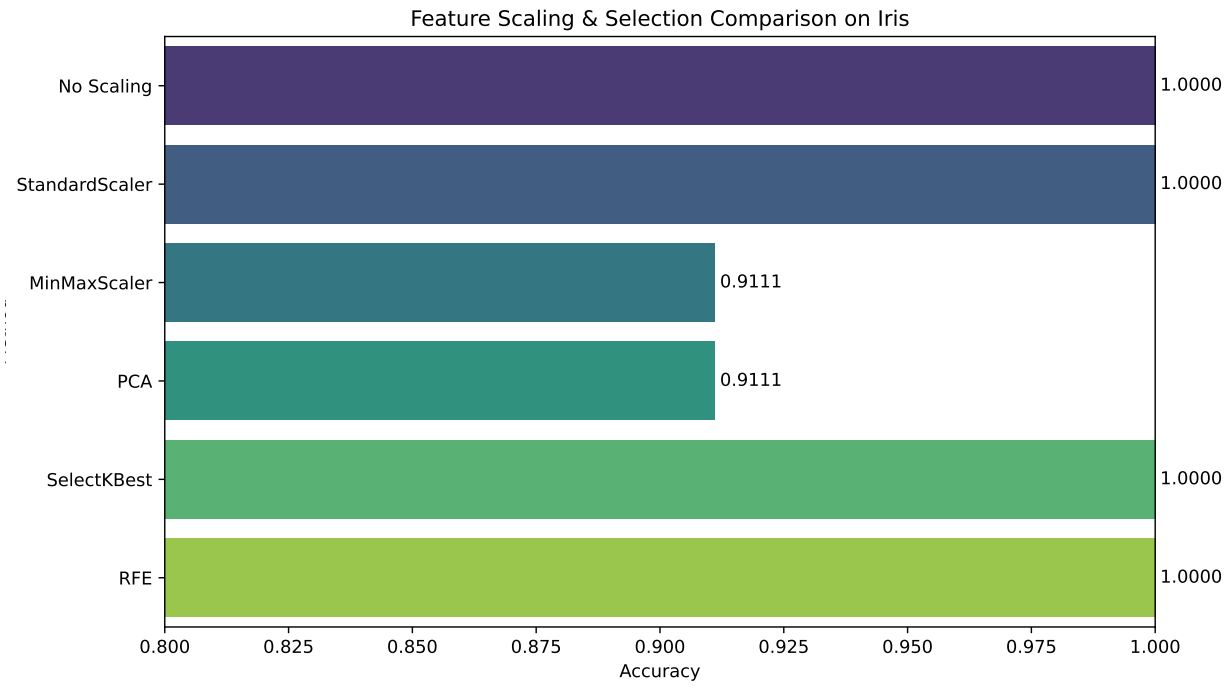
```
acc2 = accuracy_score(y_test, model.predict(X_test_mm))
print(f"3. MinMaxScaler → Accuracy: {acc2:.4f}")

# 4. Feature Extraction using PCA (reduce to 2 components)
pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train_std)
X_test_pca = pca.transform(X_test_std)

model.fit(X_train_pca, y_train)
acc3 = accuracy_score(y_test, model.predict(X_test_pca))
print(f"4. PCA (2 components) → Accuracy: {acc3:.4f}")
print(f" Explained Variance: {pca.explained_variance_ratio_.sum():.4f}")

# 5. Feature Selection - SelectKBest (top 2 features)
selector_k = SelectKBest(score_func=f_classif, k=2)
X_train_k = selector_k.fit_transform(X_train_std, y_train)
X_test_k = selector_k.transform(X_test_std)

model.fit(X_train_k, y_train)
acc4 = accuracy_score(y_test, model.predict(X_test_k))
print(f"5. SelectKBest (k=2) → Accuracy: {acc4:.4f}")
print(" Selected features:", [feature_names[i] for i in
    ↪ selector_k.get_support(indices=True)])

# 6. Recursive Feature Elimination (RFE)
rfe = RFE(estimator=LogisticRegression(max_iter=200),
    ↪ n_features_to_select=2)
X_train_rfe = rfe.fit_transform(X_train_std, y_train)
X_test_rfe = rfe.transform(X_test_std)

model.fit(X_train_rfe, y_train)
acc5 = accuracy_score(y_test, model.predict(X_test_rfe))
print(f"6. RFE (2 features) → Accuracy: {acc5:.4f}")
print(" Selected features:", [feature_names[i] for i in
    ↪ range(len(rfe.support_)) if rfe.support_[i]])

# Final Comparison Plot
results = pd.DataFrame({
'Method': ['No Scaling', 'StandardScaler', 'MinMaxScaler', 'PCA',
    ↪ 'SelectKBest', 'RFE'],
'Accuracy': [acc0, acc1, acc2, acc3, acc4, acc5]
})

plt.figure(figsize=(10,6))
sns.barplot(x='Accuracy', y='Method', data=results, palette='viridis')
plt.title('Feature Scaling & Selection Comparison on Iris')
plt.xlim(0.8, 1.0)

```

```
for i, v in enumerate(results['Accuracy']):
    plt.text(v + 0.001, i, f"{v:.4f}", va='center')
plt.show()
```

Output:

```
Original Features: ['sepal length (cm)', 'sepal width (cm)', 'petal length
 ↪  (cm)', 'petal width (cm)']
Shape: (150, 4)

1. No Scaling → Accuracy: 1.0000
2. StandardScaler → Accuracy: 1.0000
3. MinMaxScaler → Accuracy: 0.9111
4. PCA (2 components) → Accuracy: 0.9111
 Explained Variance: 0.9521
5. SelectKBest (k=2) → Accuracy: 1.0000
 Selected features: ['petal length (cm)', 'petal width (cm)']
6. RFE (2 features) → Accuracy: 1.0000
 Selected features: ['petal length (cm)', 'petal width (cm)']
```

Plots:



Experiment–6

Aim:

Program to implement simple and multiple linear regression.

Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_absolute_error,
    mean_squared_error

# Load Iris
iris = load_iris(as_frame=True)
df = iris.frame
df.columns = ['sepal_length', 'sepal_width', 'petal_length',
    'petal_width', 'species']

print("Dataset Shape:", df.shape)
print(df.head())

# Target: petal_width
y = df['petal_width']

# 1. SIMPLE LINEAR REGRESSION (Best single feature: petal_length)
X_simple = df[['petal_length']]

X_train_s, X_test_s, y_train_s, y_test_s = train_test_split(X_simple, y,
    test_size=0.2, random_state=42)

model_simple = LinearRegression()
model_simple.fit(X_train_s, y_train_s)
y_pred_s = model_simple.predict(X_test_s)

print("\n" + "="*60)
print("1. SIMPLE LINEAR REGRESSION (petal_length → petal_width)")
print("="*60)
print(f"Slope: {model_simple.coef_[0]:.4f}")
print(f"Intercept: {model_simple.intercept_:.4f}")
print(f"R² Score: {r2_score(y_test_s, y_pred_s):.4f}")
print(f"RMSE: {np.sqrt(mean_squared_error(y_test_s, y_pred_s)):.4f}")

# Plot Simple Regression
```

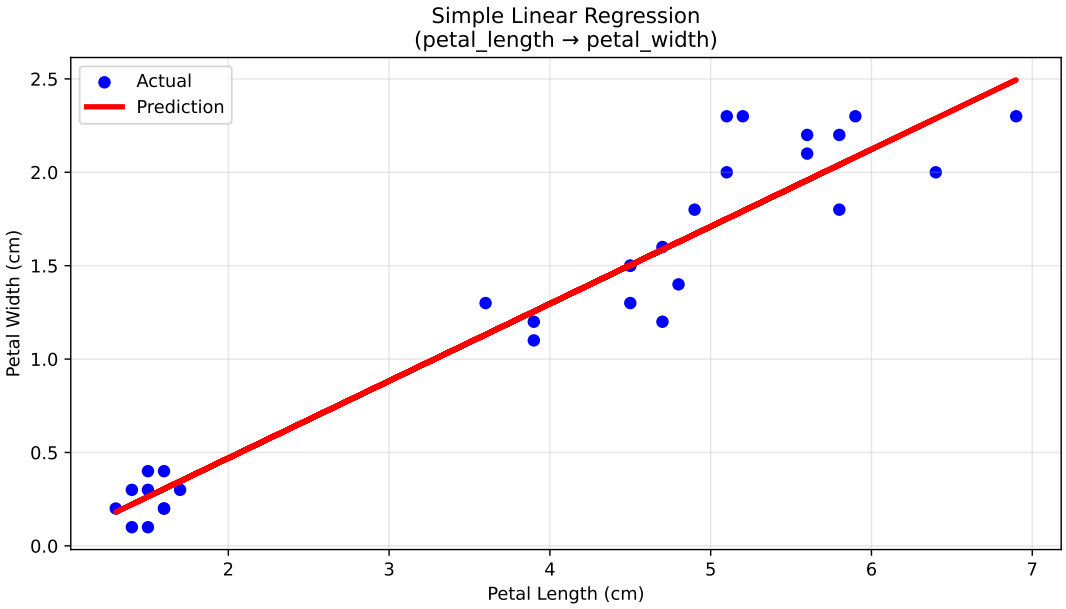
```
plt.figure(figsize=(10,5))
plt.scatter(X_test_s, y_test_s, color='blue', label='Actual')
plt.plot(X_test_s, y_pred_s, color='red', linewidth=3, label='Prediction')
plt.title('Simple Linear Regression\n(petal_length → petal_width)')
plt.xlabel('Petal Length (cm)')
plt.ylabel('Petal Width (cm)')
plt.legend()
plt.grid(True, alpha=0.3)
plt.show()

# 2. MULTIPLE LINEAR REGRESSION (All 3 numeric features)
X_multiple = df[['sepal_length', 'sepal_width', 'petal_length']]

X_train_m, X_test_m, y_train_m, y_test_m = train_test_split(X_multiple, y,
    ↪  test_size=0.2, random_state=42)

model_multiple = LinearRegression()
model_multiple.fit(X_train_m, y_train_m)
y_pred_m = model_multiple.predict(X_test_m)

print("\n" + "="*60)
print("2. MULTIPLE LINEAR REGRESSION (3 features → petal_width)")
print("="*60)
print(f"Coefficients : {model_multiple.coef_.round(4)}")
print(f"Intercept: {model_multiple.intercept_:.4f}")
print(f"R² Score: {r2_score(y_test_m, y_pred_m):.4f}")

# Feature importance (coefficient magnitude)
features = X_multiple.columns
coefs = np.abs(model_multiple.coef_)
plt.figure(figsize=(8,5))
sns.barplot(x=coefs, y=features, palette='magma')
plt.title('Feature Importance in Multiple Regression')
plt.xlabel('Absolute Coefficient Value')
plt.show()

# Actual vs Predicted (Multiple)
plt.figure(figsize=(8,6))
plt.scatter(y_test_m, y_pred_m, color='green', alpha=0.7)
plt.plot([y.min(), y.max()], [y.min(), y.max()], 'r--')
plt.xlabel('Actual Petal Width')
plt.ylabel('Predicted Petal Width')
plt.title('Multiple Linear Regression: Actual vs Predicted')
plt.grid(True, alpha=0.3)
plt.show()

# FINAL COMPARISON
print("\n" + "="*60)
```

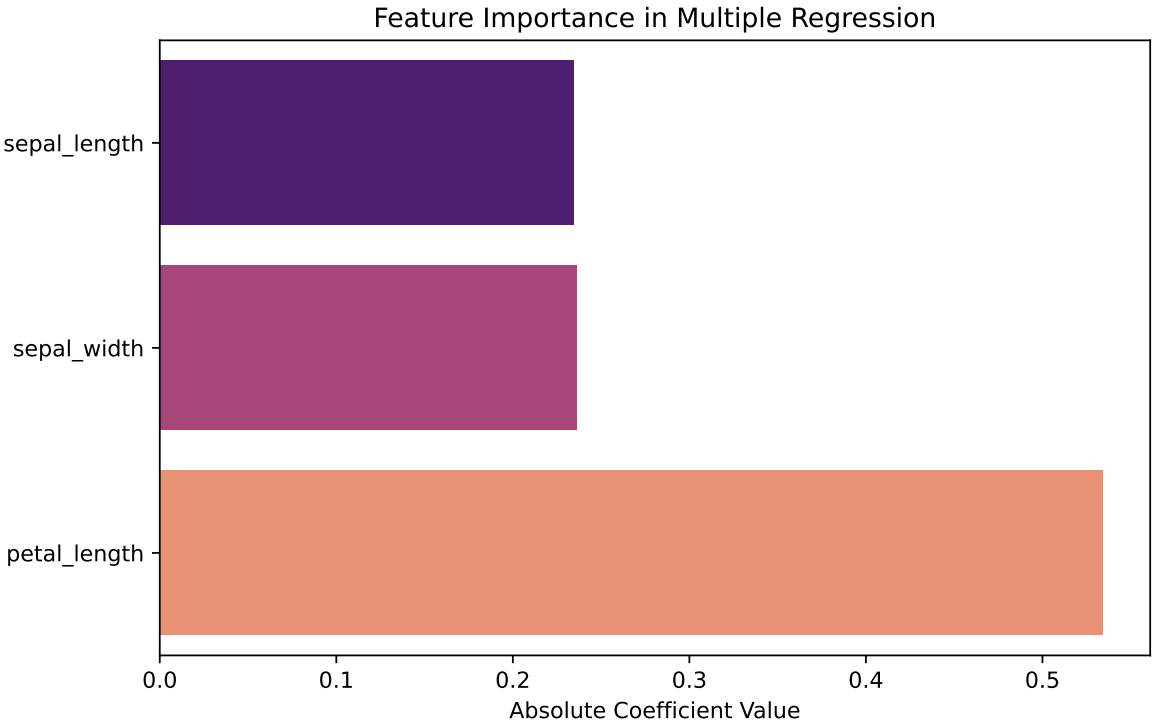
```
print("COMPARISON")
print("="*60)
print(f"Simple Linear Regression R² = {r2_score(y_test_s, y_pred_s):.4f}")
print(f"Multiple Linear Regression R² = {r2_score(y_test_m, y_pred_m):.4f}")
```

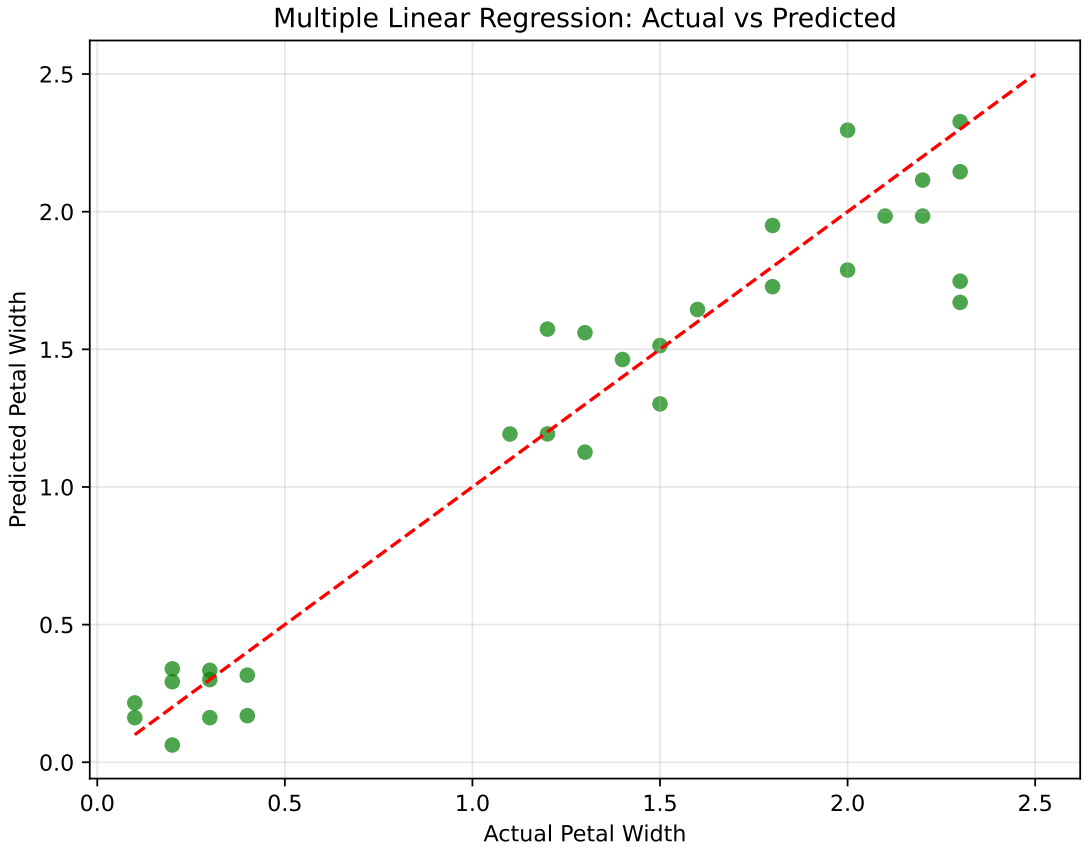
Output:

```
Dataset Shape: (150, 5)
   sepal_length  sepal_width  petal_length  petal_width  species
0           5.1          3.5           1.4          0.2        0
1           4.9          3.0           1.4          0.2        0
2           4.7          3.2           1.3          0.2        0
3           4.6          3.1           1.5          0.2        0
4           5.0          3.6           1.4          0.2        0

============================================================
1. SIMPLE LINEAR REGRESSION (petal_length → petal_width)
============================================================
Slope: 0.4132
Intercept: -0.3567
R² Score: 0.9283
RMSE: 0.2136

============================================================
2. MULTIPLE LINEAR REGRESSION (3 features → petal_width)
============================================================
Coefficients : [-0.2343  0.2359  0.5343]
Intercept: -0.1693
R² Score: 0.9271

============================================================
COMPARISON
============================================================
Simple Linear Regression R² = 0.9283
Multiple Linear Regression R² = 0.9271
```

Plots:


Simple Linear Regression
(petal_length → petal_width)
Actual
Prediction
Petal Width (cm)
Petal Length (cm)
0.0
0.5
1.0
1.5
2.0
2.5
2
3
4
5
6
7


Feature Importance in Multiple Regression
sepal_length
sepal_width
petal_length
0.0
0.1
0.2
0.3
0.4
0.5
Absolute Coefficient Value


Multiple Linear Regression: Actual vs Predicted
Predicted Petal Width
Actual Petal Width
0.0
0.5
1.0
1.5
2.0
2.5

Experiment–7

Aim:

Program to implement regularized and non linear regression.

Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline
from sklearn.metrics import r2_score, mean_squared_error

# Load Iris
iris = load_iris(as_frame=True)
df = iris.frame
df.columns = ['sepal_length', 'sepal_width', 'petal_length',
    ↪  'petal_width', 'species']

X = df[['petal_length']] # Feature
y = df['petal_width'] # Target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
    ↪  random_state=42)

results = []

# 1. Ordinary Linear Regression
lr = LinearRegression()
lr.fit(X_train, y_train)
y_pred_lr = lr.predict(X_test)
r2_lr = r2_score(y_test, y_pred_lr)
results.append(['Linear', r2_lr])

# 2. Ridge Regression (L2)
ridge = Ridge(alpha=1.0)
ridge.fit(X_train, y_train)
y_pred_ridge = ridge.predict(X_test)
r2_ridge = r2_score(y_test, y_pred_ridge)
results.append(['Ridge (L2)', r2_ridge])

# 3. Lasso Regression (L1)
lasso = Lasso(alpha=0.01, max_iter=10000)
```

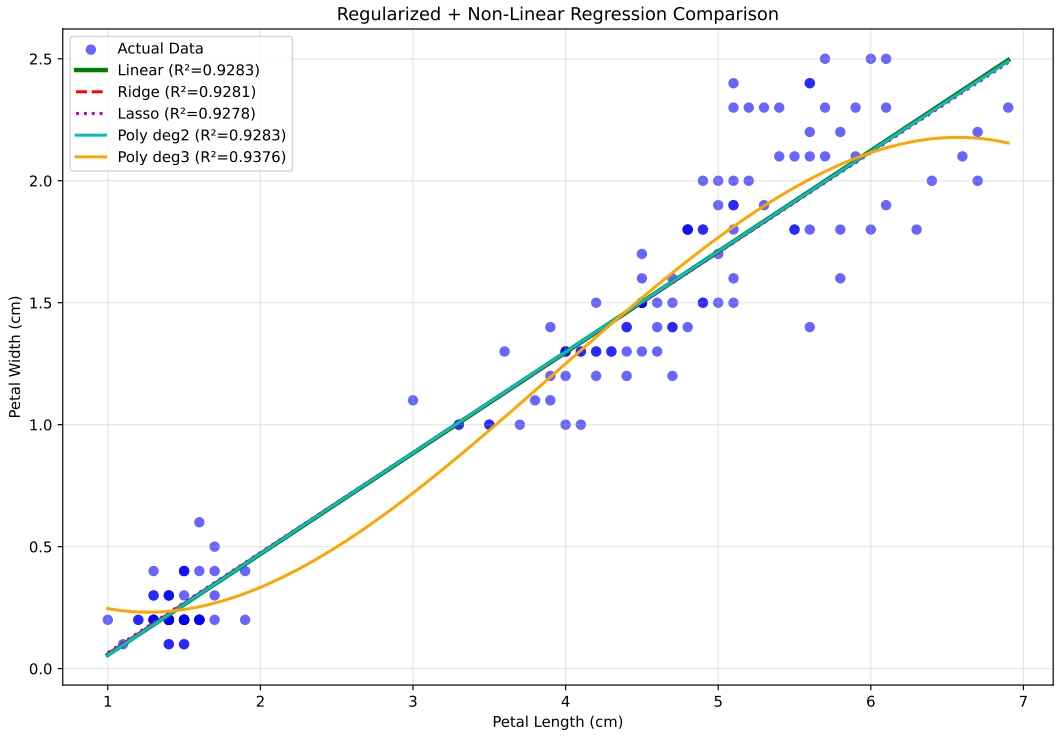
```
lasso.fit(X_train, y_train)
y_pred_lasso = lasso.predict(X_test)
r2_lasso = r2_score(y_test, y_pred_lasso)
results.append(['Lasso (L1)', r2_lasso])

# 4. Polynomial Regression (Degree 2 & 3)
poly2 = make_pipeline(PolynomialFeatures(degree=2), LinearRegression())
poly2.fit(X_train, y_train)
y_pred_p2 = poly2.predict(X_test)
r2_p2 = r2_score(y_test, y_pred_p2)
results.append(['Polynomial (deg=2)', r2_p2])

poly3 = make_pipeline(PolynomialFeatures(degree=3), LinearRegression())
poly3.fit(X_train, y_train)
y_pred_p3 = poly3.predict(X_test)
r2_p3 = r2_score(y_test, y_pred_p3)
results.append(['Polynomial (deg=3)', r2_p3])

# Print Results
print("="*65)
print("REGRESSION COMPARISON ON IRIS (petal_length → petal_width)")
print("="*65)
for name, score in results:
    print(f"{name:25} → R² = {score:.5f}")
print("="*65)

# Plot all models
plt.figure(figsize=(12,8))
plt.scatter(X, y, color='blue', alpha=0.6, label='Actual Data')

# Sort for smooth lines
X_plot = np.linspace(X.min(), X.max(), 100).reshape(-1,1)

plt.plot(X_plot, lr.predict(X_plot), 'g-', linewidth=3, label=f'Linear
    ↪ (R²={r2_lr:.4f})')
plt.plot(X_plot, ridge.predict(X_plot), 'r--', linewidth=2, label=f'Ridge
    ↪ (R²={r2_ridge:.4f})')
plt.plot(X_plot, lasso.predict(X_plot), 'm:', linewidth=2, label=f'Lasso
    ↪ (R²={r2_lasso:.4f})')
plt.plot(X_plot, poly2.predict(X_plot), 'c-', linewidth=2, label=f'Poly
    ↪ deg2 (R²={r2_p2:.4f})')
plt.plot(X_plot, poly3.predict(X_plot), 'orange', linewidth=2, label=f'Poly
    ↪ deg3 (R²={r2_p3:.4f})')

plt.xlabel('Petal Length (cm)')
plt.ylabel('Petal Width (cm)')
plt.title('Regularized + Non-Linear Regression Comparison')
plt.legend()
```

```
plt.grid(True, alpha=0.3)
plt.show()

# Coefficient comparison
print(f"\nCoefficients:")
print(f"Linear : {lr.coef_[0]:.4f}")
print(f"Ridge : {ridge.coef_[0]:.4f}")
print(f"Lasso : {lasso.coef_[0]:.4f}")
```

Output:

```
======================================================================
REGRESSION COMPARISON ON IRIS (petal_length → petal_width)
======================================================================
Linear                    → R² = 0.92826
Ridge (L2)                → R² = 0.92811
Lasso (L1)                → R² = 0.92779
Polynomial (deg=2)        → R² = 0.92834
Polynomial (deg=3)        → R² = 0.93758
======================================================================

Coefficients:
Linear : 0.4132
Ridge : 0.4121
Lasso : 0.4100
```

Plots:



Experiment–8

Aim:

Program to implement K-Means clustering techniques.

Code:

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.cluster.hierarchy import dendrogram,linkage
from sklearn.datasets import make_blobs
from sklearn.cluster import AgglomerativeClustering
from sklearn.cluster import DBSCAN
from sklearn.cluster import KMeans
from sklearn.datasets import make_moons


x,y=make_blobs(n_samples=150,n_features=2,centers=3,cluster_std=0.5,

shuffle=True,random_state=0)

plt.scatter(x[:,0],x[:,1],c='white',edgecolor='black',marker='o',s=50)
plt.show()

km=KMeans(n_clusters=3,init='random',max_iter=100,n_init=10)
y_km=km.fit_predict(x)
print('Distortion=',km.inertia_)
print(km.cluster_centers_)

plt.figure()
plt.scatter(
    x[y_km==0,0], x[y_km==0,1], s=50,
    c='lightgreen', marker='d', edgecolor='black',
    label='cluster-1'
)
plt.scatter(
    x[y_km==1,0], x[y_km==1,1], s=50,
    c='orange', marker='o', edgecolor='black',
    label='cluster-2'
)
plt.scatter(
    x[y_km==2,0], x[y_km==2,1], s=50,
    c='blue', marker='>', edgecolor='black',
    label='cluster-3'
)
plt.scatter(
    km.cluster_centers_[:,0], km.cluster_centers_[:,1], s=100,
    c='red', marker='*', edgecolor='black',
```

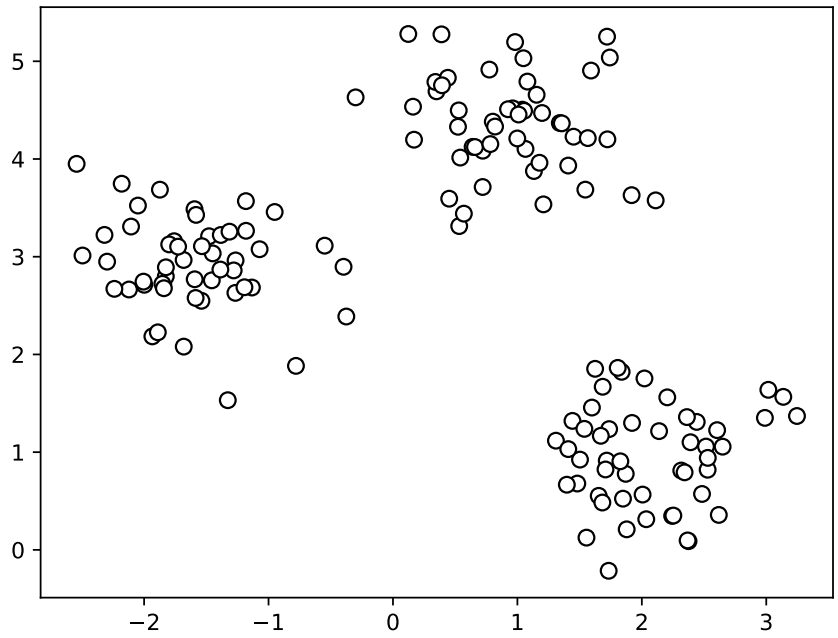
```
    label='centroids'
)
plt.legend()
plt.show()

dist=[]
for i in range(1,11):
    km=KMeans(n_clusters=i,init='k-means++',n_init=10,max_iter=100)
    km.fit(x)
    dist.append(km.inertia_)
plt.plot(range(1,11),dist,marker='o')
plt.xlabel('Number of clusters (K)-->')
plt.ylabel('Distortions')
plt.show()

# Agglomerative
x=np.array([[1,2],[1,4],[1,0],[4,2],[4,4],[4,0]])
plt.scatter(x[:,0],x[:,1])
plt.show()
ac=AgglomerativeClustering(n_clusters=2,metric='euclidean',linkage='single')
 ↪  # 'complete'
labels=ac.fit_predict(x)
print(labels)

x=np.array([[1,2],[1,4],[1,0],[4,2],[4,4],[4,0]])
z=linkage(x,'single') # 'complete'
dendrogram(z)
plt.title('Hierarchical clustering')
plt.xlabel('Data Points')
plt.ylabel('Distance')
plt.show()

x,y=make_moons(n_samples=200,noise=0.05,random_state=0)
plt.scatter(x[:,0],x[:,1])
plt.show()

km=KMeans(n_clusters=2,random_state=0)
y_km=km.fit_predict(x)
plt.scatter(x[y_km==0,0],x[y_km==0,1],c='blue',s=50,marker='o')
plt.scatter(x[y_km==1,0],x[y_km==1,1],c='red',s=50,marker='d')
plt.scatter(
    km.cluster_centers_[:,0],
    km.cluster_centers_[:,1],
    c='black',s=50,marker='*'
)
plt.show()

km=AgglomerativeClustering(n_clusters=2,linkage='single') #'compete
```

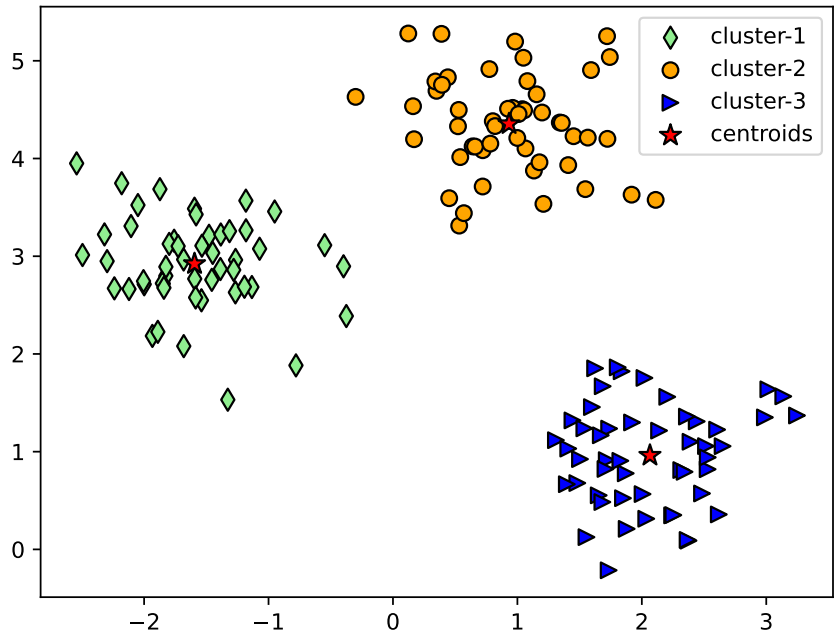
```
y_km=km.fit_predict(x)
plt.scatter(x[y_km==0,0],x[y_km==0,1],c='blue',s=50,marker='o')
plt.scatter(x[y_km==1,0],x[y_km==1,1],c='red',s=50,marker='d')
plt.show()

km=DBSCAN(eps=0.2,min_samples=5,metric='euclidean')
y_km=km.fit_predict(x)
plt.scatter(x[y_km==0,0],x[y_km==0,1],c='blue',s=50,marker='o')
plt.scatter(x[y_km==1,0],x[y_km==1,1],c='red',s=50,marker='d')
plt.show()
```

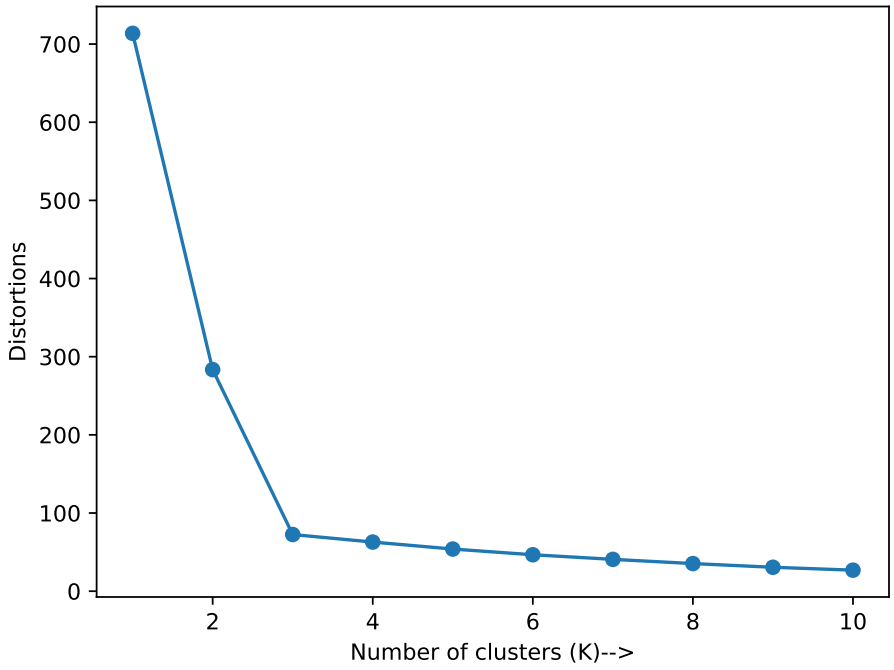
Output:

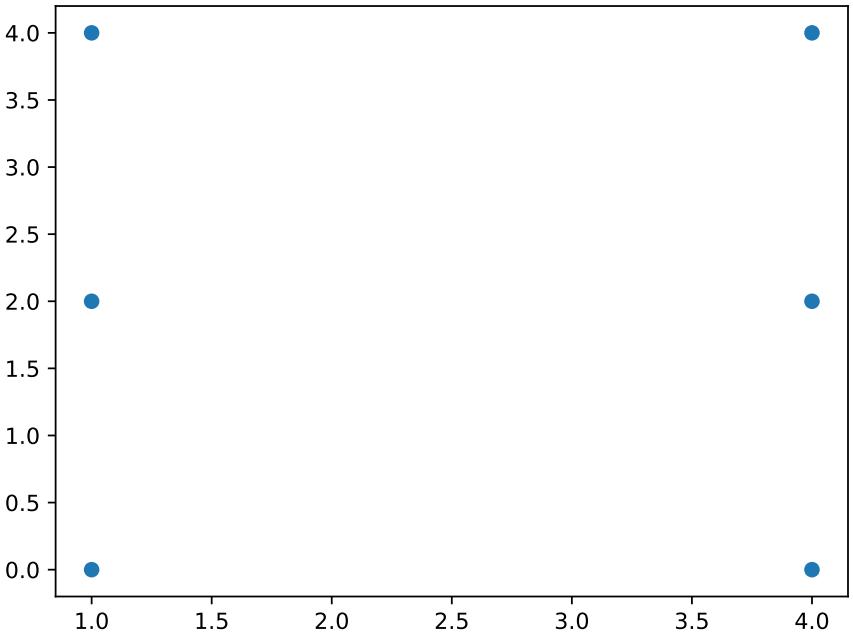
```
Distortion= 72.47601670996698
[[-1.5947298   2.92236966]
 [ 2.06521743  0.96137409]
 [ 0.9329651   4.35420712]]
[1 1 1 0 0 0]
```

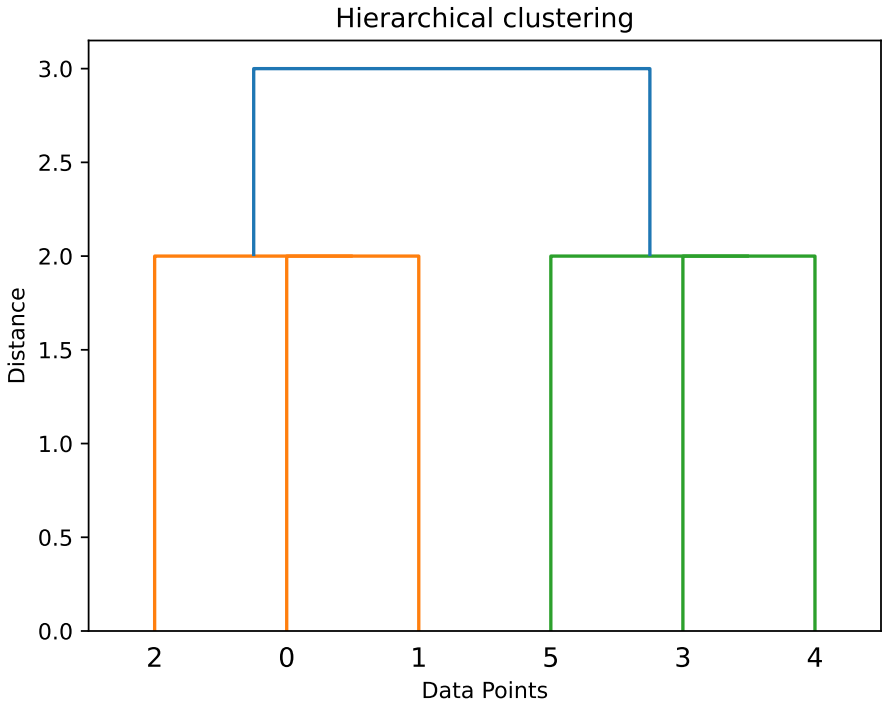
Plots:

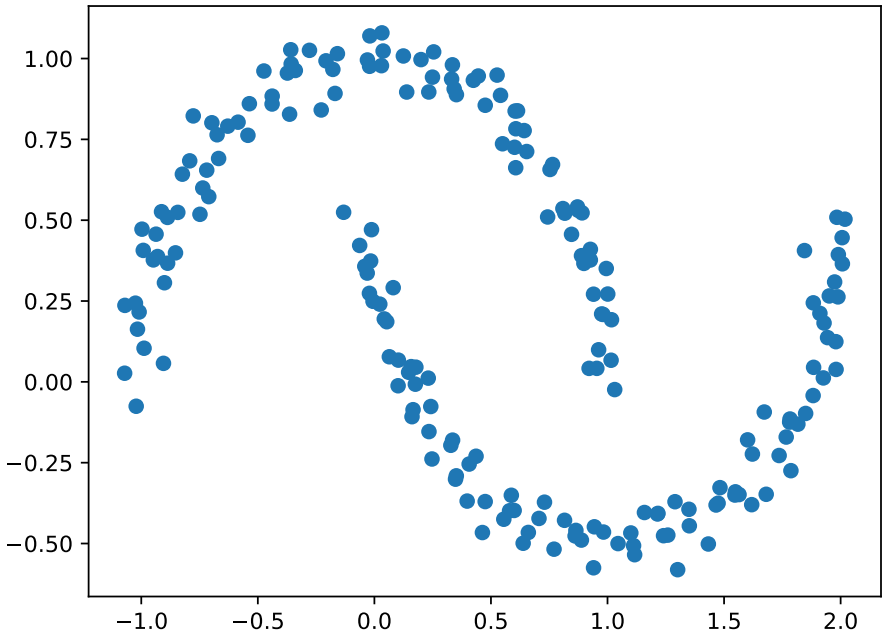

5
4
3
2
1
0
−2
−1
0
1
2
3

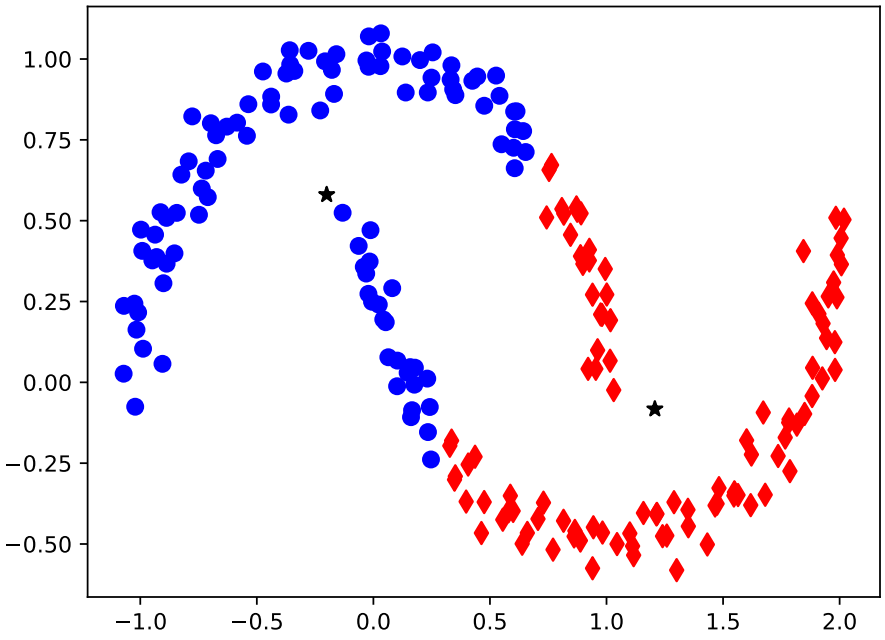

cluster-1
cluster-2
cluster-3
centroids
5
4
3
2
1
0
−2
−1
0
1
2
3

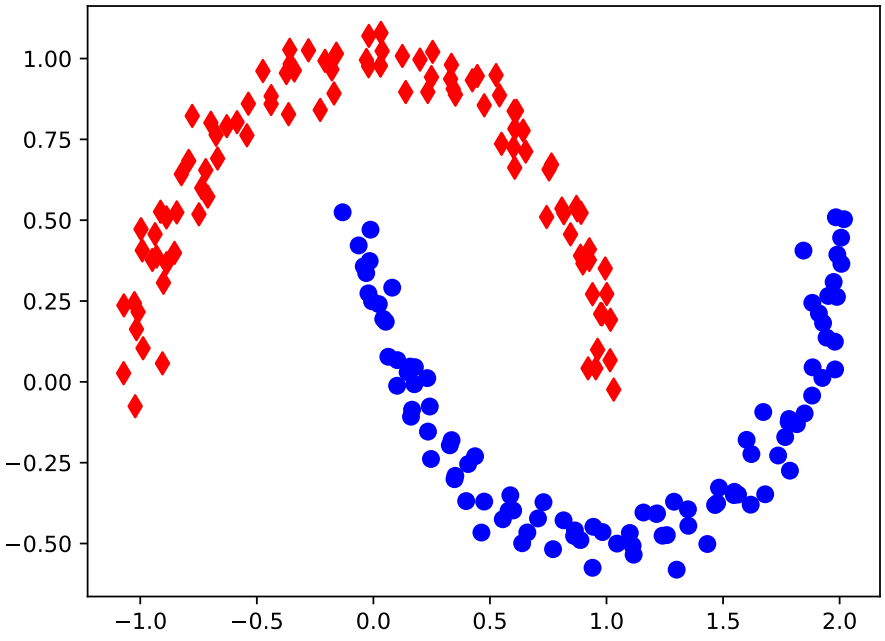

Distortions
700
600
500
400
300
200
100
0
2
4
6
8
10
Number of clusters (K)-->

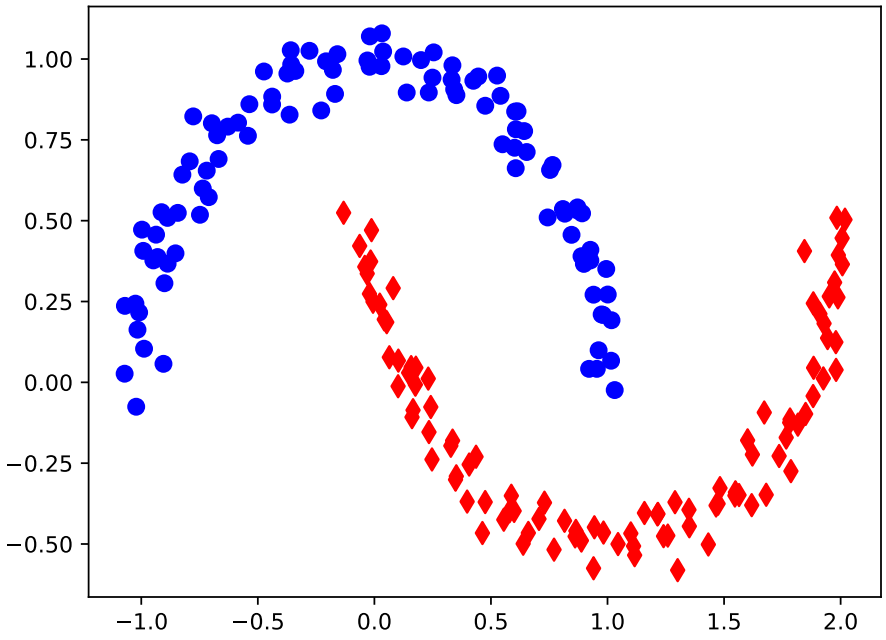

4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
1.0
1.5
2.0
2.5
3.0
3.5
4.0


Hierarchical clustering
Distance
3.0
2.5
2.0
1.5
1.0
0.5
0.0
2
0
1
5
3
4
Data Points


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0