Practical 1

Python Code:

```
import pandas as pd
import numpy as np
df=pd.read_csv("Iris.csv")
print(df.head())
df.head()
df.tail()
df.describe()
df.info()
df.shape
df.dtypes
df.columns
missing_values=df.isnull().sum()
print("\nMissing values in csv data:")
print("Missing_values")
df = df.drop(columns=['Species'])
fill= df.fillna(df.mean())
print("\nFilled Missing Values with Mean (CSV Data):")
print(df.head())
z_scores = np.abs((fill - fill.mean()) / fill.std())
z_scores
do = fill[(z_scores < 3).all(axis=1)]
print("\nData After Removing Outliers (CSV Data):")
print(do.head())
threshold_value = 3
filter = do[do['SepalLengthCm'] > threshold_value]
print(f"\nFiltered Data (Rows with column_name > {threshold_value}):")
print(filter)
df_sorted = filter.sort_values(by='SepalWidthCm', ascending=False)
print(df_sorted.head())
df_grouped = df_sorted.groupby('SepalLengthCm').agg({
'PetalLengthCm': 'mean',
'PetalWidthCm': 'sum'
}).reset_index()
print("\nGrouped Data (Mean and Sum for Each Group):")
print(df_grouped.head())
```

Iris.csv:

```
Id,SepalLengthCm,SepalWidthCm,PetalLengthCm,PetalWidthCm,Species
1,5.1,3.5,1.4,0.2,Iris-setosa
2,4.9,3.0,1.4,0.2,Iris-setosa
3,4.7,3.2,1.3,0.2,Iris-setosa
4,4.6,3.1,1.5,0.2,Iris-setosa
5,5.0,3.6,1.4,0.2,Iris-setosa
6,5.4,3.9,1.7,0.4,Iris-setosa
7,4.6,3.4,1.4,0.3,Iris-setosa
```

8,5.0,3.4,1.5,0.2,Iris-setosa
9,4.4,2.9,1.4,0.2,Iris-setosa
10,4.9,3.1,1.5,0.1,Iris-setosa
11,5.4,3.0,4.5,1.5,Iris-versicolor
12,6.0,3.4,4.5,1.6,Iris-versicolor
13,6.7,3.1,4.7,1.5,Iris-versicolor
14,6.3,2.3,4.4,1.3,Iris-versicolor
15,5.6,3.0,4.1,1.3,Iris-versicolor
16,5.5,2.5,4.0,1.3,Iris-versicolor
17,5.5,2.6,4.4,1.2,Iris-versicolor
18,6.1,3.0,4.6,1.4,Iris-versicolor
19,5.8,2.6,4.0,1.2,Iris-versicolor
20,5.0,2.3,3.3,1.0,Iris-versicolor
21,6.3,3.3,6.0,2.5,Iris-virginica
22,5.8,2.7,5.1,1.9,Iris-virginica
23,7.1,3.0,5.9,2.1,Iris-virginica
24,6.3,2.9,5.6,1.8,Iris-virginica
25,6.5,3.0,5.2,2.0,Iris-virginica
26,7.6,3.0,6.6,2.1,Iris-virginica
27,4.9,2.5,4.5,1.7,Iris-virginica
28,7.3,2.9,6.3,1.8,Iris-virginica
29,6.7,2.5,5.8,1.8,Iris-virginica
30,7.2,3.6,6.1,2.5,Iris-virginica

OUTPUT:

```
   Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm      Species
0   1            5.1           3.5            1.4           0.2  Iris-setosa
1   2            4.9           3.0            1.4           0.2  Iris-setosa
2   3            4.7           3.2            1.3           0.2  Iris-setosa
3   4            4.6           3.1            1.5           0.2  Iris-setosa
4   5            5.0           3.6            1.4           0.2  Iris-setosa
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30 entries, 0 to 29
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             30 non-null     int64
 1   SepalLengthCm  30 non-null     float64
 2   SepalWidthCm   30 non-null     float64
 3   PetalLengthCm  30 non-null     float64
 4   PetalWidthCm   30 non-null     float64
 5   Species        30 non-null     object
dtypes: float64(4), int64(1), object(1)
memory usage: 1.5+ KB

Missing values in csv data:
Missing_values
```

Filled Missing Values with Mean (CSV Data):

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
0	1	5.1	3.5	1.4	0.2
1	2	4.9	3.0	1.4	0.2
2	3	4.7	3.2	1.3	0.2
3	4	4.6	3.1	1.5	0.2
4	5	5.0	3.6	1.4	0.2

Data After Removing Outliers (CSV Data):

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
0	1	5.1	3.5	1.4	0.2
1	2	4.9	3.0	1.4	0.2
2	3	4.7	3.2	1.3	0.2
3	4	4.6	3.1	1.5	0.2
4	5	5.0	3.6	1.4	0.2

Filtered Data (Rows with column_name > 3):

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
0	1	5.1	3.5	1.4	0.2
1	2	4.9	3.0	1.4	0.2
2	3	4.7	3.2	1.3	0.2
3	4	4.6	3.1	1.5	0.2
4	5	5.0	3.6	1.4	0.2
5	6	5.4	3.9	1.7	0.4
6	7	4.6	3.4	1.4	0.3
7	8	5.0	3.4	1.5	0.2
8	9	4.4	2.9	1.4	0.2
9	10	4.9	3.1	1.5	0.1
10	11	5.4	3.0	4.5	1.5
11	12	6.0	3.4	4.5	1.6
12	13	6.7	3.1	4.7	1.5
13	14	6.3	2.3	4.4	1.3
14	15	5.6	3.0	4.1	1.3
15	16	5.5	2.5	4.0	1.3
16	17	5.5	2.6	4.4	1.2
17	18	6.1	3.0	4.6	1.4
18	19	5.8	2.6	4.0	1.2
19	20	5.0	2.3	3.3	1.0
20	21	6.3	3.3	6.0	2.5
21	22	5.8	2.7	5.1	1.9
22	23	7.1	3.0	5.9	2.1
23	24	6.3	2.9	5.6	1.8
24	25	6.5	3.0	5.2	2.0
25	26	7.6	3.0	6.6	2.1
26	27	4.9	2.5	4.5	1.7
27	28	7.3	2.9	6.3	1.8
28	29	6.7	2.5	5.8	1.8
29	30	7.2	3.6	6.1	2.5

```
   Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
5   6            5.4           3.9            1.7           0.4
4   5            5.0           3.6            1.4           0.2
29  30           7.2           3.6            6.1           2.5
0   1            5.1           3.5            1.4           0.2
7   8            5.0           3.4            1.5           0.2
Grouped Data (Mean and Sum for Each Group):
   SepalLengthCm  PetalLengthCm  PetalWidthCm
0            4.4       1.400000           0.2
1            4.6       1.450000           0.5
2            4.7       1.300000           0.2
3            4.9       2.466667           2.0
4            5.0       2.066667           1.4
```

Practical 2

Python Code:

```
#part1:feature scaling
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler, StandardScaler, LabelEncoder
df = pd.read_csv("wine.csv")
print("\nFull CSV Loaded:")
print(df.head())
df1 = pd.read_csv("wine.csv", usecols=[0, 1, 2], skiprows=1)
df1.columns = ['classlabel', 'Alcohol', 'Malic Acid']
print("\nOriginal DataFrame (First 5 Rows):")
print(df1.head())
scaling = MinMaxScaler()
scaled_value = scaling.fit_transform(df1[['Alcohol', 'Malic Acid']])
df1[['Alcohol', 'Malic Acid']] = scaled_value
print("\nDataframe After Min-Max Scaling:")
print(df1.head())
scaling = StandardScaler()
scaled_standardvalue = scaling.fit_transform(df1[['Alcohol', 'Malic Acid']])
df1[['Alcohol', 'Malic Acid']] = scaled_standardvalue
print("\nDataframe After Standard Scaling:")
print(df1.head())
plt.scatter(df1['Alcohol'], df1['Malic Acid'])
plt.title("Standard Scaled Alcohol vs Malic Acid")
plt.xlabel("Alcohol")
plt.ylabel("Malic Acid")
plt.show()
#part2:feature dummification
iris=pd.read_csv("Iris.csv")
iris
le=LabelEncoder()
iris['code']=le.fit_transform(iris.Species)
iris
```

wine.csv

```
classlabel,Alcohol,Malic Acid
1,14.23,1.71
1,13.20,1.78
1,13.16,2.36
1,14.37,1.95
1,13.24,2.59
2,12.37,1.13
2,12.33,1.10
2,13.30,1.40
2,12.71,1.45
2,12.00,1.38
3,13.75,1.75
3,14.10,2.20
3,14.00,1.90
3,13.90,2.30
3,13.50,1.60
```

Output

Full CSV Loaded:

	classlabel	Alcohol	Malic Acid
0	1	14.23	1.71
1	1	13.20	1.78
2	1	13.16	2.36
3	1	14.37	1.95
4	1	13.24	2.59

Original DataFrame (First 5 Rows):

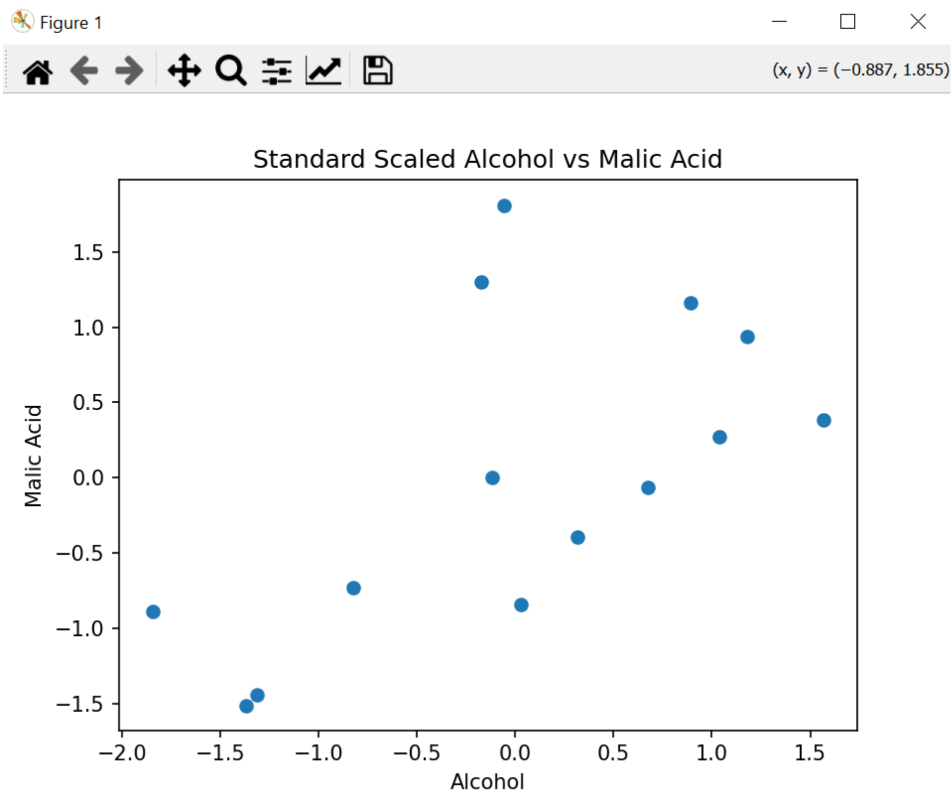
	classlabel	Alcohol	Malic Acid
0	1	13.20	1.78
1	1	13.16	2.36
2	1	14.37	1.95
3	1	13.24	2.59
4	2	12.37	1.13

Dataframe After Min-Max Scaling:

	classlabel	Alcohol	Malic Acid
0	1	0.506329	0.456376
1	1	0.489451	0.845638
2	1	1.000000	0.570470
3	1	0.523207	1.000000
4	2	0.156118	0.020134

Dataframe After Standard Scaling:

	classlabel	Alcohol	Malic Acid
0	1	-0.116248	0.004780
1	1	-0.173858	1.298461
2	1	1.568833	0.383962
3	1	-0.058638	1.811472
4	2	-1.311647	-1.445035



Practical 3

Python Code:

```
#part1
import pandas as pd
import numpy as np
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
df = pd.read_csv("FCH.csv")
print("\nCSV Data Loaded:")
print(df.head())
print(df.tail())
print("Shape:", df.shape)
print("Size:", df.size)
print(df.describe())
print(df.info())
print("\nData Types:\n", df.dtypes)
housing = fetch_california_housing()
housing_df = pd.DataFrame(housing.data, columns=housing.feature_names)
housing_df["PRICE"] = housing.target
print("\nCalifornia Housing Dataset:")
print(housing_df.head())
#part2
X = housing_df[['AveRooms']]
y = housing_df[['PRICE']]
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
model = LinearRegression()
model.fit(X_train, y_train)
mse = mean_squared_error(y_test, model.predict(X_test))
r2 = r2_score(y_test, model.predict(X_test))
print("\n===== SIMPLE LINEAR REGRESSION =====")
print("Mean Squared Error:", mse)
print("R-squared Value:", r2)
print("Intercept:", model.intercept_)
print("Coefficient:", model.coef_)
#part2:
X = housing_df.drop("PRICE", axis=1)
y = housing_df["PRICE"]
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
model = LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
```

```
print("\n===== MULTIPLE LINEAR REGRESSION =====")
print("Mean Squared Error:", mse)
print("R-squared Value:", r2)
print("Intercept:", model.intercept_)
print("Coefficients:", model.coef_)
```

FCH.csv:

```
MedInc,HouseAge,AveRooms,AveBedrms,Population,AveOccup,Latitude,Longitude,PRICE
8.3252,41,6.9841,1.0238,322,2.5556,37.88,-122.23,4.526
8.3014,21,6.2381,0.9719,2401,2.1098,37.86,-122.22,3.585
7.2574,52,8.2881,1.0734,496,2.8023,37.85,-122.24,3.521
5.6431,52,5.8174,1.0734,558,2.5479,37.84,-122.25,3.413
3.8462,52,6.2810,1.0811,565,2.1815,37.83,-122.26,3.422
4.0368,42,6.0836,1.0542,413,2.2426,37.85,-122.28,2.697
3.6591,52,6.0080,1.0234,1094,2.1270,37.85,-122.29,1.854
3.1200,52,5.0000,1.0200,850,2.1100,37.85,-122.30,2.234
2.8450,20,4.5000,0.9800,700,2.2000,37.80,-122.32,1.965
3.5000,25,6.2000,1.0500,900,2.3000,37.78,-122.33,2.540
```

OUTPUT:

```
CSV Data Loaded:
   MedInc  HouseAge  AveRooms  AveBedrms  ...  AveOccup  Latitude  Longitude  PRICE
0  8.3252        41    6.9841     1.0238  ...    2.5556     37.88    -122.23  4.526
1  8.3014        21    6.2381     0.9719  ...    2.1098     37.86    -122.22  3.585
2  7.2574        52    8.2881     1.0734  ...    2.8023     37.85    -122.24  3.521
3  5.6431        52    5.8174     1.0734  ...    2.5479     37.84    -122.25  3.413
4  3.8462        52    6.2810     1.0811  ...    2.1815     37.83    -122.26  3.422
[5 rows x 9 columns]
   MedInc  HouseAge  AveRooms  AveBedrms  ...  AveOccup  Latitude  Longitude  PRICE
5  4.0368        42    6.0836     1.0542  ...    2.2426     37.85    -122.28  2.697
6  3.6591        52    6.0080     1.0234  ...    2.1270     37.85    -122.29  1.854
7  3.1200        52    5.0000     1.0200  ...    2.1100     37.85    -122.30  2.234
8  2.8450        20    4.5000     0.9800  ...    2.2000     37.80    -122.32  1.965
9  3.5000        25    6.2000     1.0500  ...    2.3000     37.78    -122.33  2.540
[5 rows x 9 columns]
Shape: (10, 9)
Size: 90
          MedInc   HouseAge   AveRooms  ...   Latitude   Longitude      PRICE
count  10.000000  10.000000  10.000000  ...  10.000000   10.000000  10.000000
mean    5.053420  40.900000   6.140030  ...  37.839000 -122.272000   2.975700
std     2.158254  13.755403   1.025986  ...   0.029231    0.037947   0.853737
min     2.845000  20.000000   4.500000  ...  37.780000 -122.330000   1.854000
25%     3.539775  29.000000   5.865050  ...  37.832500 -122.297500   2.310500
50%     3.941500  47.000000   6.141800  ...  37.850000 -122.270000   3.055000
75%     6.853825  52.000000   6.270275  ...  37.850000 -122.242500   3.496250
max     8.325200  52.000000   8.288100  ...  37.880000 -122.220000   4.526000
[8 rows x 9 columns]
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 10 entries, 0 to 9
Data columns (total 9 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   MedInc      10 non-null     float64
 1   HouseAge    10 non-null     int64
 2   AveRooms    10 non-null     float64
 3   AveBedrms   10 non-null     float64
 4   Population  10 non-null     int64
 5   AveOccup    10 non-null     float64
 6   Latitude    10 non-null     float64
 7   Longitude   10 non-null     float64
 8   PRICE       10 non-null     float64
dtypes: float64(7), int64(2)
memory usage: 852.0 bytes
None
Data Types:
 MedInc        float64
HouseAge        int64
AveRooms      float64
AveBedrms     float64
Population      int64
AveOccup      float64
Latitude      float64
Longitude     float64
PRICE         float64
dtype: object
California Housing Dataset:
   MedInc  HouseAge  AveRooms  AveBedrms  ...  AveOccup  Latitude  Longitude  PRICE
0  8.3252      41.0  6.984127  1.023810  ...  2.555556     37.88    -122.23  4.526
1  8.3014      21.0  6.238137  0.971880  ...  2.109842     37.86    -122.22  3.585
2  7.2574      52.0  8.288136  1.073446  ...  2.802260     37.85    -122.24  3.521
3  5.6431      52.0  5.817352  1.073059  ...  2.547945     37.85    -122.25  3.413
4  3.8462      52.0  6.281853  1.081081  ...  2.181467     37.85    -122.25  3.422
[5 rows x 9 columns]
===== SIMPLE LINEAR REGRESSION =====
Mean Squared Error: 1.2923314440807299
R-squared Value: 0.013795337532284901
Intercept: [1.65476227]
Coefficient: [[0.07675559]]
===== MULTIPLE LINEAR REGRESSION =====
Mean Squared Error: 0.555891598695244
R-squared Value: 0.5757877060324511
Intercept: -37.023277706064064
Coefficients: [ 4.48674910e-01  9.72425752e-03 -1.23323343e-01  7.83144907e-01
 -2.02962058e-06 -3.52631849e-03 -4.19792487e-01 -4.33708065e-01]
```

Practical 4

Python Code:

```
#part1
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
df = pd.read_csv("Iris.csv")
print("\nOriginal Dataset:")
print(df.head())
df = df[df['Species'] != 'Iris-virginica']
print("\nDataset After Removing Class 2:")
print(df.head())
X = df.drop('Species', axis=1)
y = df['Species']
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
logistic_model = LogisticRegression()
logistic_model.fit(X_train, y_train)
y_pred_logistic = logistic_model.predict(X_test)
print("\n===== Logistic Regression Results =====")
print("Accuracy:", accuracy_score(y_test, y_pred_logistic))
print("\nClassification Report:\n", classification_report(y_test, y_pred_logistic))
#part2
from sklearn.tree import DecisionTreeClassifier
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
model = DecisionTreeClassifier()
model.fit(X_train, y_train)
y_pred_tree = model.predict(X_test)
print("\n===== Decision Tree Classifier Results =====")
print("Accuracy:", accuracy_score(y_test, y_pred_tree))
print("\nClassification Report:\n", classification_report(y_test, y_pred_tree))
```

OUTPUT:

```
Original Dataset:
   Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm      Species
0   1            5.1           3.5            1.4           0.2  Iris-setosa
1   2            4.9           3.0            1.4           0.2  Iris-setosa
2   3            4.7           3.2            1.3           0.2  Iris-setosa
3   4            4.6           3.1            1.5           0.2  Iris-setosa
4   5            5.0           3.6            1.4           0.2  Iris-setosa
Dataset After Removing Class 2:
   Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm      Species
0   1            5.1           3.5            1.4           0.2  Iris-setosa
1   2            4.9           3.0            1.4           0.2  Iris-setosa
```

```
2   3          4.7          3.2          1.3          0.2  Iris-setosa
3   4          4.6          3.1          1.5          0.2  Iris-setosa
4   5          5.0          3.6          1.4          0.2  Iris-setosa
===== Logistic Regression Results =====
Accuracy: 1.0
Classification Report:
                 precision    recall  f1-score   support
    Iris-setosa       1.00      1.00      1.00         2
Iris-versicolor       1.00      1.00      1.00         2
       accuracy                           1.00         4
      macro avg       1.00      1.00      1.00         4
   weighted avg       1.00      1.00      1.00         4
===== Decision Tree Classifier Results =====
Accuracy: 1.0
Classification Report:
                 precision    recall  f1-score   support
    Iris-setosa       1.00      1.00      1.00         2
Iris-versicolor       1.00      1.00      1.00         2
       accuracy                           1.00         4
      macro avg       1.00      1.00      1.00         4
   weighted avg       1.00      1.00      1.00         4
```

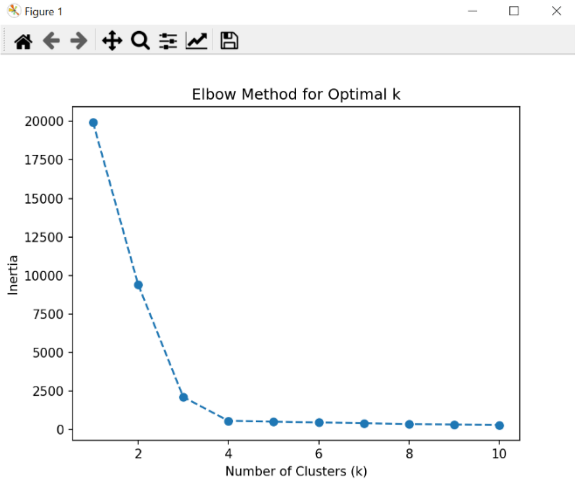
Practical 5

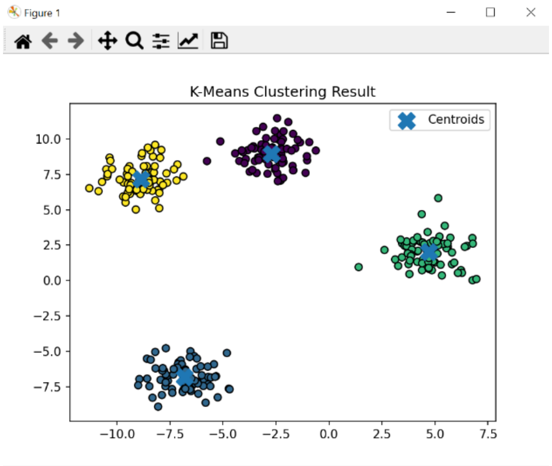
Python Code:

```
import warnings
warnings.filterwarnings("ignore")
import os
os.environ["LOKY_MAX_CPU_COUNT"] = "1"
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
X, _ = make_blobs(n_samples=300, centers=4, random_state=42)
inertia = []
K_range = range(1, 11)
for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(X)
    inertia.append(kmeans.inertia_)
plt.plot(K_range, inertia, marker='o', linestyle='--')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia')
plt.title('Elbow Method for Optimal k')
plt.show()
kmeans = KMeans(n_clusters=4, random_state=42, n_init=10)
y_kmeans = kmeans.fit_predict(X)
plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, cmap='viridis', edgecolors='k')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
```

```
        s=200, marker='X', label='Centroids')
plt.title('K-Means Clustering Result')
plt.legend()
plt.show()
```

OUTPUT:





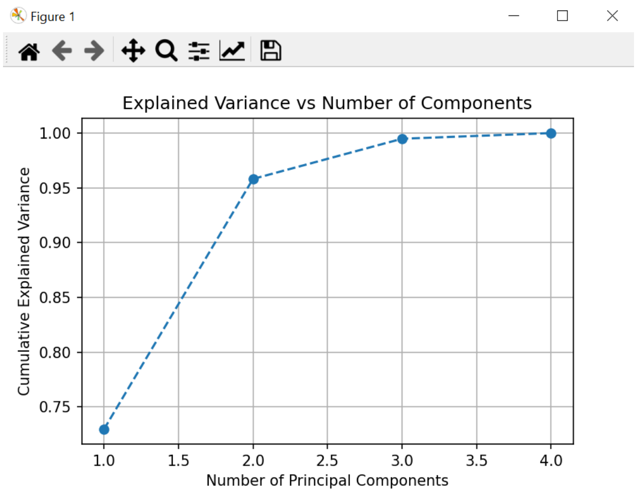
Practical 6

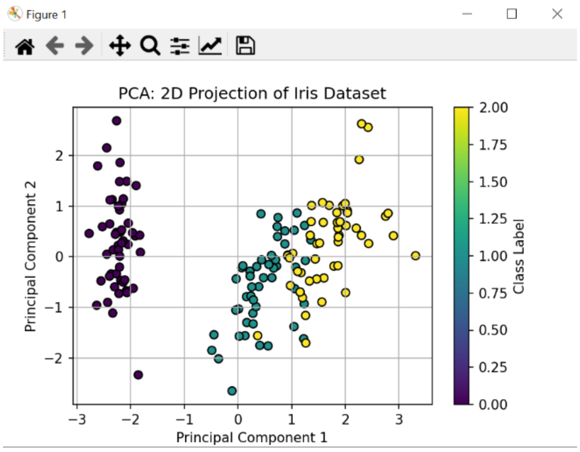
Python Code:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
data = load_iris()
X = data.data
y = data.target
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
pca = PCA()
X_pca = pca.fit_transform(X_scaled)
explained_variance = pca.explained_variance_ratio_
cumulative_variance = np.cumsum(explained_variance)
plt.figure(figsize=(6, 4))
plt.plot(range(1, len(explained_variance) + 1), cumulative_variance,
      marker='o', linestyle='--')
plt.xlabel('Number of Principal Components')
plt.ylabel('Cumulative Explained Variance')
plt.title('Explained Variance vs Number of Components')
plt.grid(True)
plt.show()
pca_2d = PCA(n_components=2)
X_pca_2d = pca_2d.fit_transform(X_scaled)
plt.figure(figsize=(6, 4))
```

```
plt.scatter(X_pca_2d[:, 0], X_pca_2d[:, 1], c=y, cmap='viridis', edgecolors='k')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('PCA: 2D Projection of Iris Dataset')
plt.colorbar(label='Class Label')
plt.grid(True)
plt.show()
```

OUTPUT:





Practical 8

Python Code:

```
import numpy as np
from scipy import stats
import matplotlib.pyplot as plt
np.random.seed(42)
sample1 = np.random.normal(loc=10, scale=2, size=30)
sample2 = np.random.normal(loc=12, scale=2, size=30)
t_statistic, p_value = stats.ttest_ind(sample1, sample2)
alpha = 0.05
print("Results of Two-Sample t-test:")
print("T-statistic:", t_statistic)
print("P-value:", p_value)
print("Degrees of Freedom:", len(sample1) + len(sample2) - 2)
plt.figure(figsize=(10, 5))
plt.hist(sample1, alpha=0.5, label='Sample 1', color='blue')
plt.hist(sample2, alpha=0.5, label='Sample 2', color='orange')
plt.axvline(np.mean(sample1), color='blue', linestyle='dashed', linewidth=2)
plt.axvline(np.mean(sample2), color='orange', linestyle='dashed', linewidth=2)
plt.title('Distributions of Sample 1 and Sample 2')
plt.xlabel('Values')
plt.ylabel('Frequency')
plt.legend()
plt.grid(True)
t_critical = stats.t.ppf(1 - alpha/2, len(sample1) + len(sample2) - 2)
```

```
sample_min, sample_max = min(sample1.min(), sample2.min()), max(sample1.max(), sample2.max())
x_vals = np.linspace(sample_min, sample_max, 100)
critical_region = np.abs(x_vals - np.mean(sample1)) > t_critical
plt.fill_between(x_vals, 0, 5, where=critical_region, alpha=0.3, label='Critical Region')
plt.text(11.5, 5, f"T = {t_statistic:.2f}", ha='center', color='black', backgroundcolor='white')
plt.show()
if p_value < alpha:
    print("\nConclusion: There is significant evidence to reject the null hypothesis.")
    print("Interpretation: The mean of Sample 1 is significantly higher than that of Sample 2.")
else:
    print("\nConclusion: Fail to reject the null hypothesis.")
    print("Interpretation: There is not enough evidence to claim a significant difference between the means.")

# chi-test
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sb
import warnings
from scipy import stats
warnings.filterwarnings('ignore')
df = sb.load_dataset('mpg')
print(df)
print(df['horsepower'].describe())
print(df['model_year'].describe())
# Binning horsepower
bins = [0, 75, 150, 240]
df['horsepower_new'] = pd.cut(df['horsepower'], bins=bins, labels=['l', 'm', 'h'])
print(df['horsepower_new'])
ybins = [69, 72, 74, 84]
label = ['1', '2', '3']
df['modelyear_new'] = pd.cut(df['model_year'], bins=ybins, labels=label)
newyear = df['modelyear_new']
print(newyear)
df_chi = pd.crosstab(df['horsepower_new'], df['modelyear_new'])
print(df_chi)
print(stats.chi2_contingency(df_chi))
```

<u>OUTPUT:</u>

```
Results of Two-Sample t-test:
T-statistic: -4.512913234547555
P-value: 3.176506547470154e-05
Degrees of Freedom: 58

Conclusion: There is significant evidence to reject the null hypothesis.
Interpretation: The mean of Sample 1 is significantly higher than that of Sample 2.
```

```
      mpg  cylinders  ...  origin                       name
0    18.0          8  ...     usa  chevrolet chevelle malibu
1    15.0          8  ...     usa          buick skylark 320
2    18.0          8  ...     usa         plymouth satellite
3    16.0          8  ...     usa              amc rebel sst
4    17.0          8  ...     usa                ford torino
..    ...        ...  ...     ...                        ...
393  27.0          4  ...     usa            ford mustang gl
394  44.0          4  ...  europe                  vw pickup
395  32.0          4  ...     usa              dodge rampage
396  28.0          4  ...     usa                ford ranger
397  31.0          4  ...     usa                 chevy s-10

[398 rows x 9 columns]
count    392.000000
mean     104.469388
std       38.491160
min       46.000000
25%       75.000000
50%       93.500000
75%      126.000000
max      230.000000
Name: horsepower, dtype: float64
count    398.000000
mean      76.010050
std        3.697627
min       70.000000
25%       73.000000
50%       76.000000
75%       79.000000
max       82.000000
Name: model_year, dtype: float64
0      m
1      h
2      m
3      m
4      m
      ..
393    m
394    l
395    m
396    m
397    m
Name: horsepower_new, Length: 398, dtype: category
Categories (3, object): ['l' < 'm' < 'h']
0      1
1      1
2      1
```

```
3      1
4      1
      ..
393    3
394    3
395    3
396    3
397    3
Name: modelyear_new, Length: 398, dtype: category
Categories (3, object): ['1' < '2' < '3']
modelyear_new    1   2    3
horsepower_new
l                9  14   76
m               49  41  158
h               26  11    8
Chi2ContingencyResult(statistic=np.float64(54.95485392447537),
pvalue=np.float64(3.320518009555984e-11), dof=4, expected_freq=array([[ 21.21428571,  16.66836735,
61.11734694],
       [ 53.14285714,  41.75510204, 153.10204082],
       [  9.64285714,   7.57653061,  27.78061224]]))
```