AIML LAB FILE

Submitted By:

Name: Anmol Thapliyal
SAP ID:590011794
Batch-36
Branch: B.Tech CSE

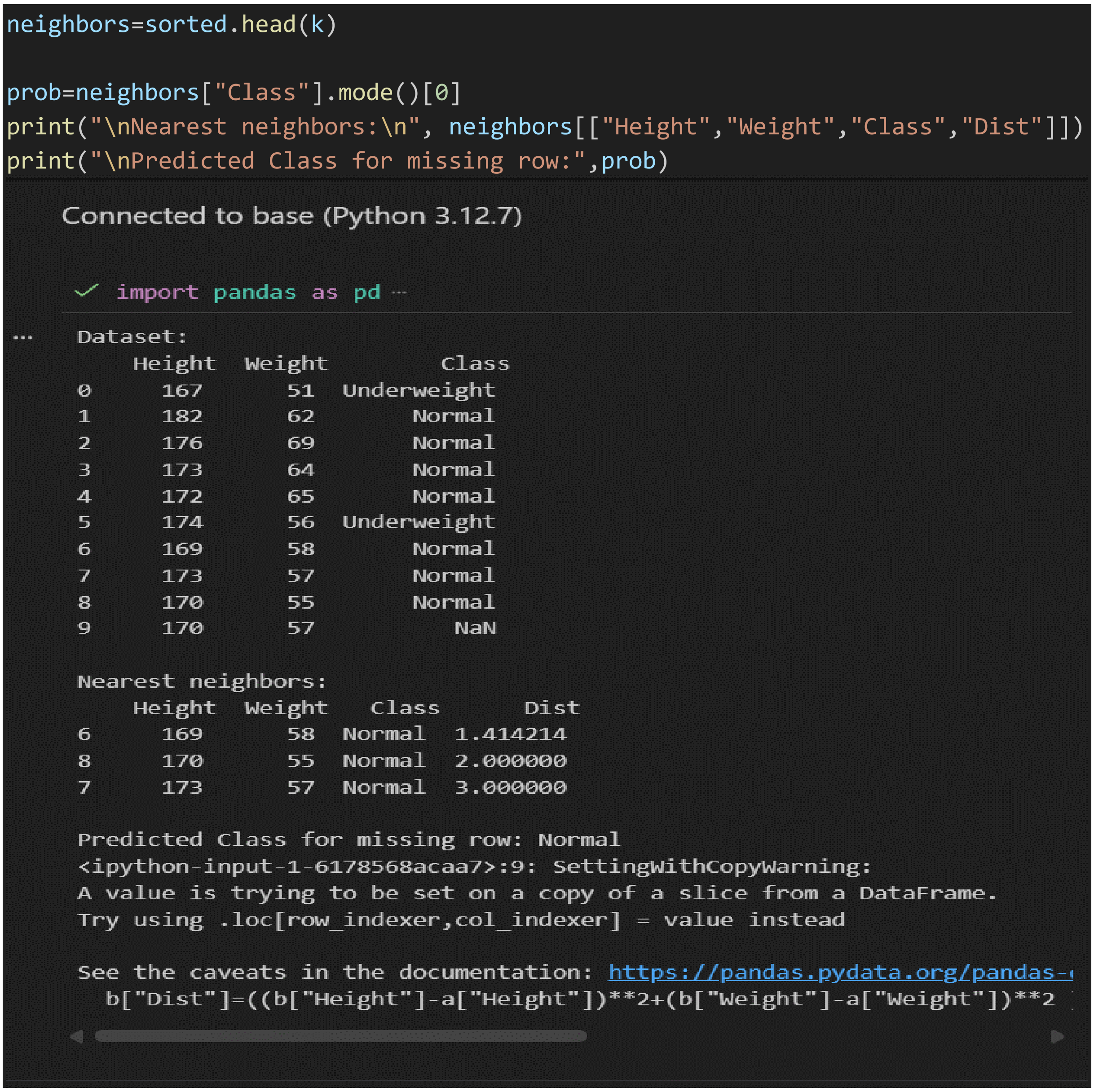
Submitted To:

Faculty Name: Prof. Sumit Singh
Department: Computer Science & Engineering

INDEX:

EXP:1- KNN Missing Value Prediction

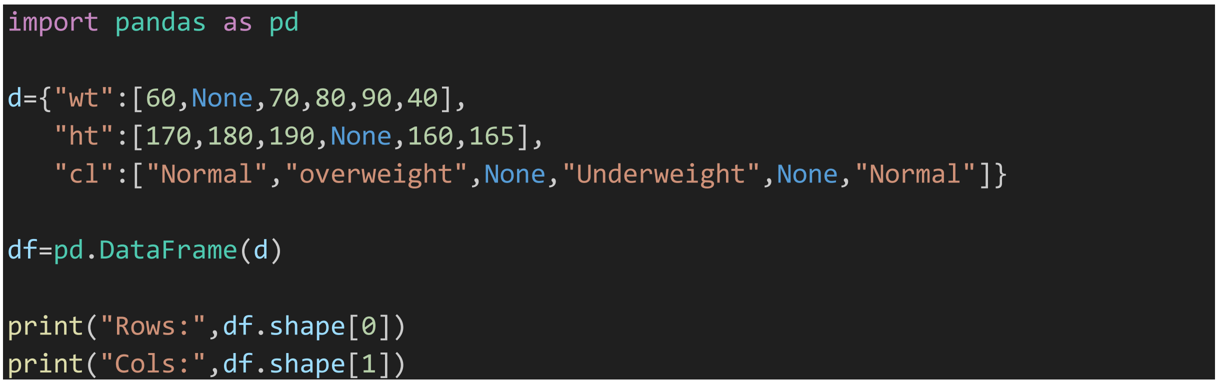
```
import pandas as pd

df=pd.read_excel("e:/sem 3/aiml lab/data.xlsx")
print("Dataset:\n",df)

a=df[df["Class"].isnull()].iloc[0]
b=df[df["Class"].notnull()]

b["Dist"]=((b["Height"]-a["Height"])**2+(b["Weight"]-a["Weight"])**2 )**0.5

sorted=b.sort_values("Dist")

k=3
```



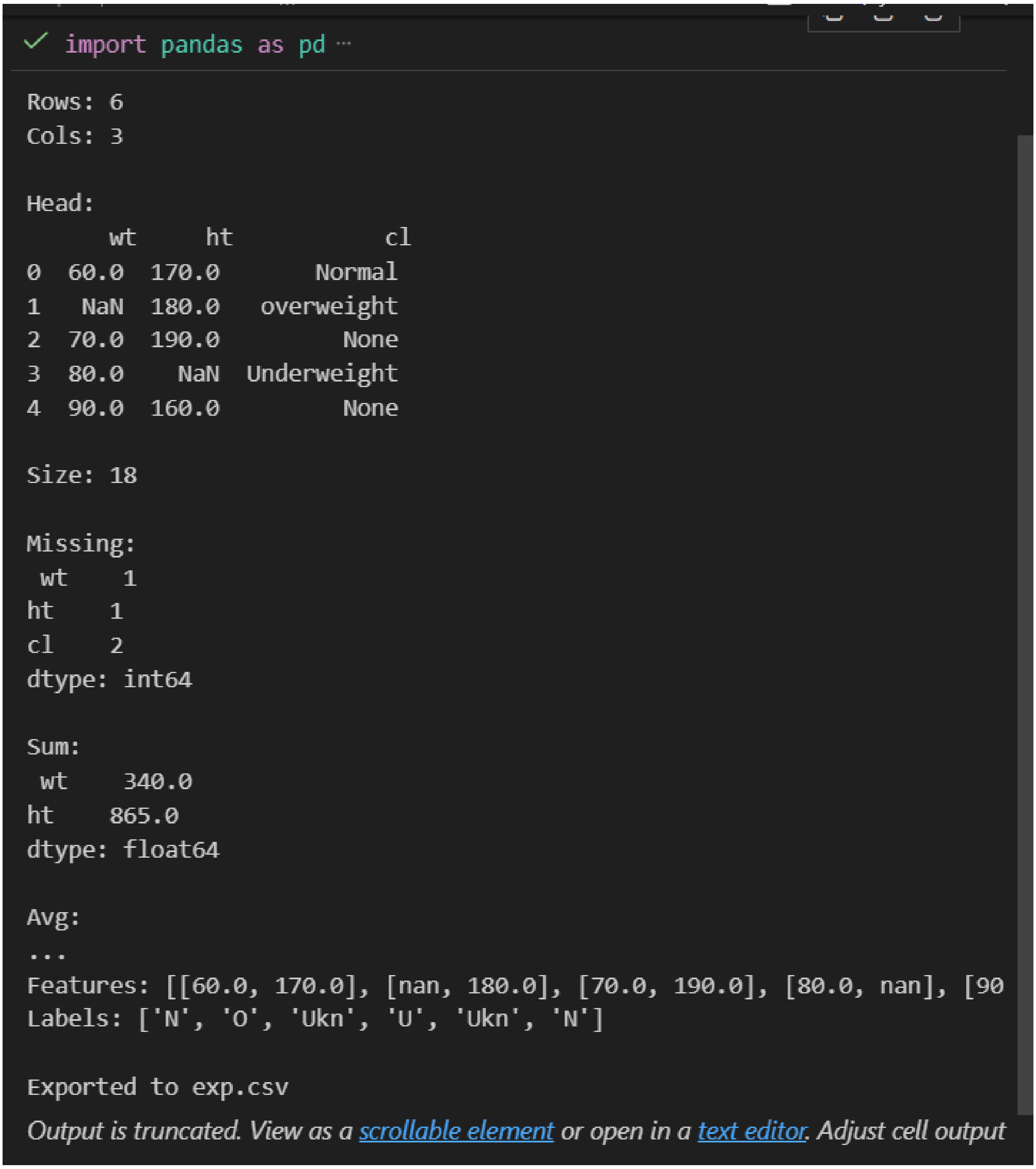
```
neighbors=sorted.head(k)

prob=neighbors["Class"].mode()[0]
print("\nNearest neighbors:\n", neighbors[["Height","Weight","Class","Dist"]])
print("\nPredicted Class for missing row:",prob)
```

```
Connected to base (Python 3.12.7)

✓ import pandas as pd ···

Dataset:
   Height  Weight        Class
0     167      51  Underweight
1     182      62       Normal
2     176      69       Normal
3     173      64       Normal
4     172      65       Normal
5     174      56  Underweight
6     169      58       Normal
7     173      57       Normal
8     170      55       Normal
9     170      57          NaN

Nearest neighbors:
   Height  Weight   Class      Dist
6     169      58  Normal  1.414214
8     170      55  Normal  2.000000
7     173      57  Normal  3.000000

Predicted Class for missing row: Normal
<ipython-input-1-6178568acaa7>:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
  b["Dist"]=((b["Height"]-a["Height"])**2+(b["Weight"]-a["Weight"])**2
```

EXP:2- Basic Data Analysis & Encoding



```
import pandas as pd

d={"wt":[60,None,70,80,90,40],
   "ht":[170,180,190,None,160,165],
   "cl":["Normal","overweight",None,"Underweight",None,"Normal"]}

df=pd.DataFrame(d)

print("Rows:",df.shape[0])
print("Cols:",df.shape[1])
print("\nHead:\n",df.head())
print("\nSize:",df.size)
print("\nMissing:\n",df.isnull().sum())

n=df.select_dtypes(include='number')
print("\nSum:\n",n.sum())
print("\nAvg:\n",n.mean())
print("\nMin:\n",n.min())
print("\nMax:\n",n.max())

m={"Normal":"N","overweight":"O","Underweight":"U"}
df['cl'] = df['cl'].map(m).fillna("Ukn")

feat=df[['wt','ht']].values.tolist()
label=df['cl'].tolist()

print("\nFeatures:",feat)
print("Labels:",label)

df.to_csv("exp.csv")
print("\nExported to exp.csv")
```

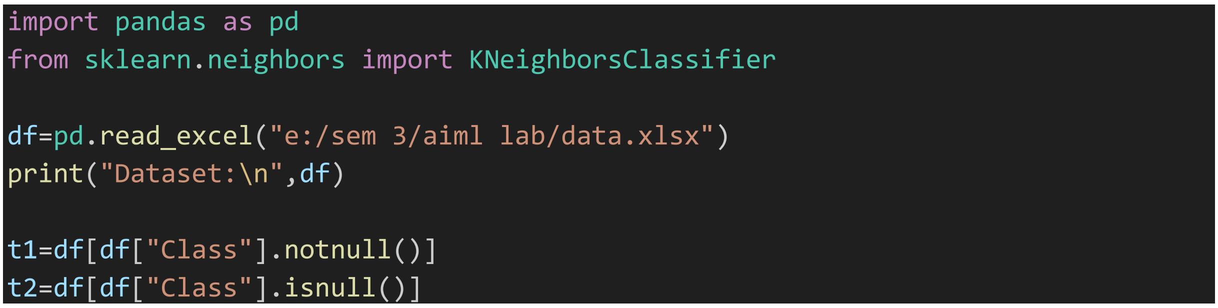


```
✓ import pandas as pd ···
```

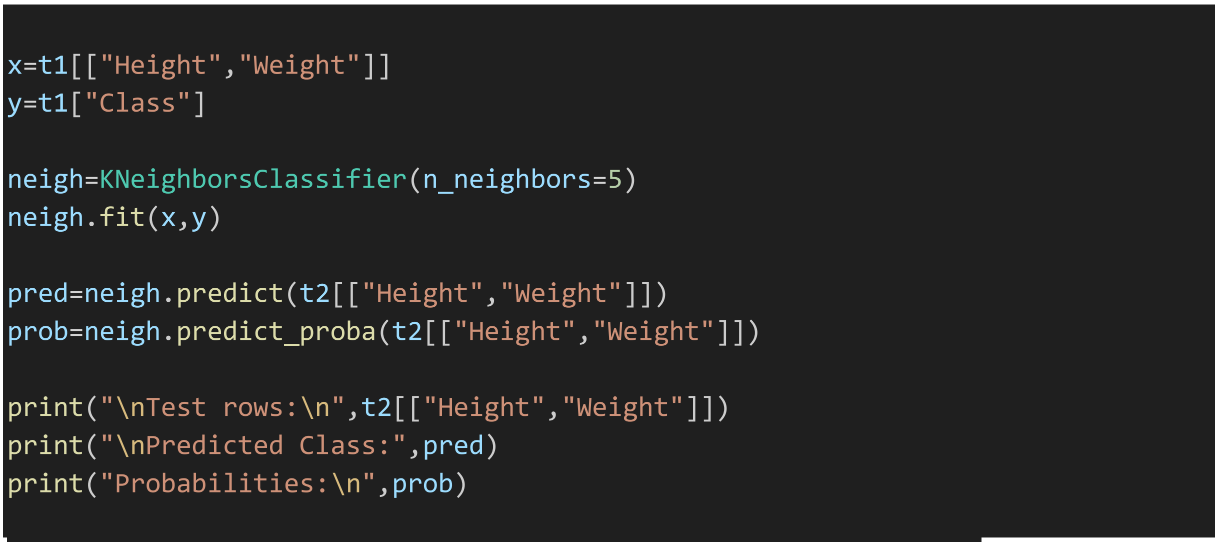
```
Rows: 6
Cols: 3

Head:
      wt     ht            cl
0  60.0  170.0        Normal
1   NaN  180.0    overweight
2  70.0  190.0          None
3  80.0    NaN   Underweight
4  90.0  160.0          None

Size: 18

Missing:
 wt    1
ht    1
cl    2
dtype: int64

Sum:
 wt    340.0
ht    865.0
dtype: float64

Avg:
...
Features: [[60.0, 170.0], [nan, 180.0], [70.0, 190.0], [80.0, nan], [90
Labels: ['N', 'O', 'Ukn', 'U', 'Ukn', 'N']

Exported to exp.csv
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output

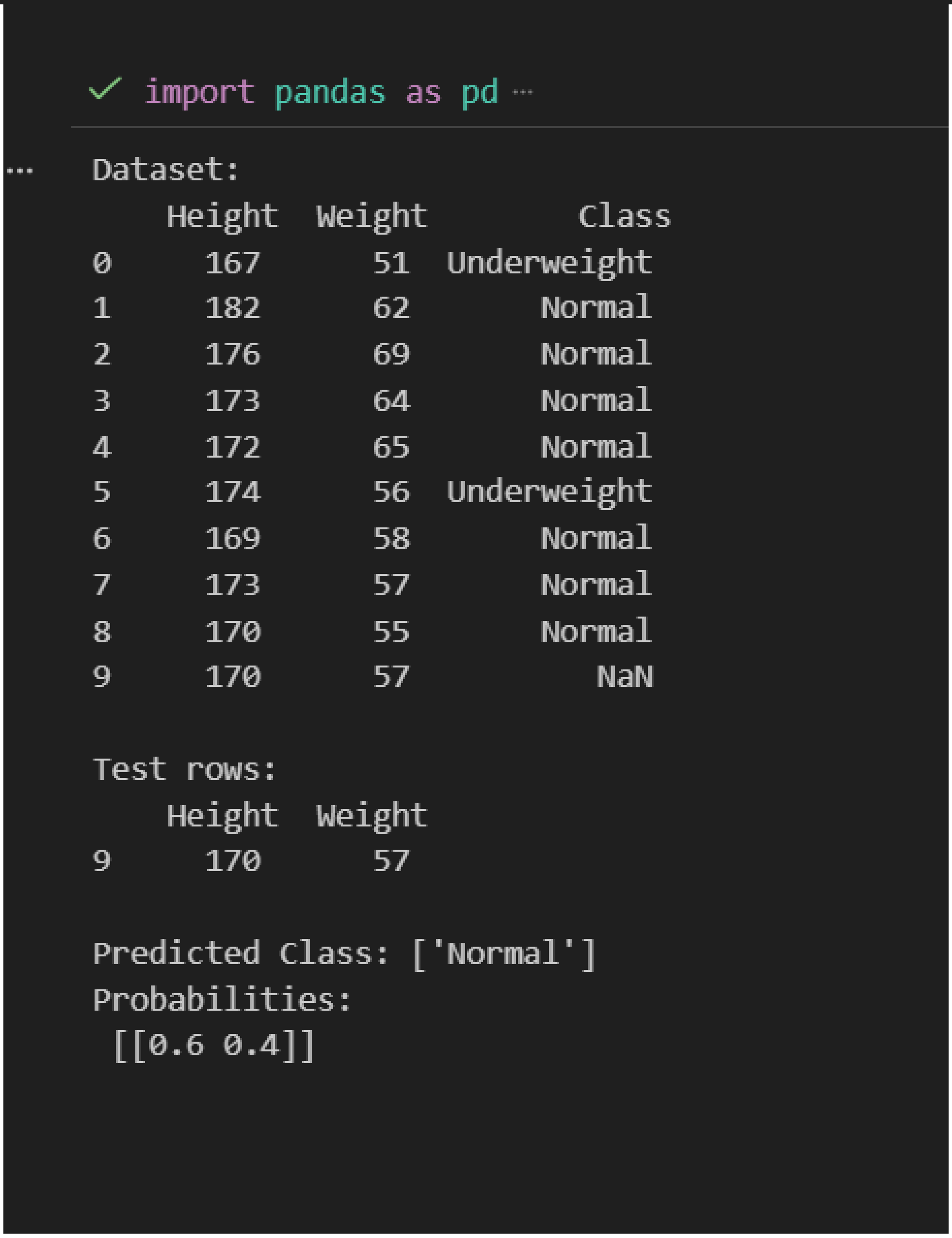
EXP:3- KNN Classification (sklearn)



```
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier

df=pd.read_excel("e:/sem 3/aiml lab/data.xlsx")
print("Dataset:\n",df)

t1=df[df["Class"].notnull()]
t2=df[df["Class"].isnull()]
```



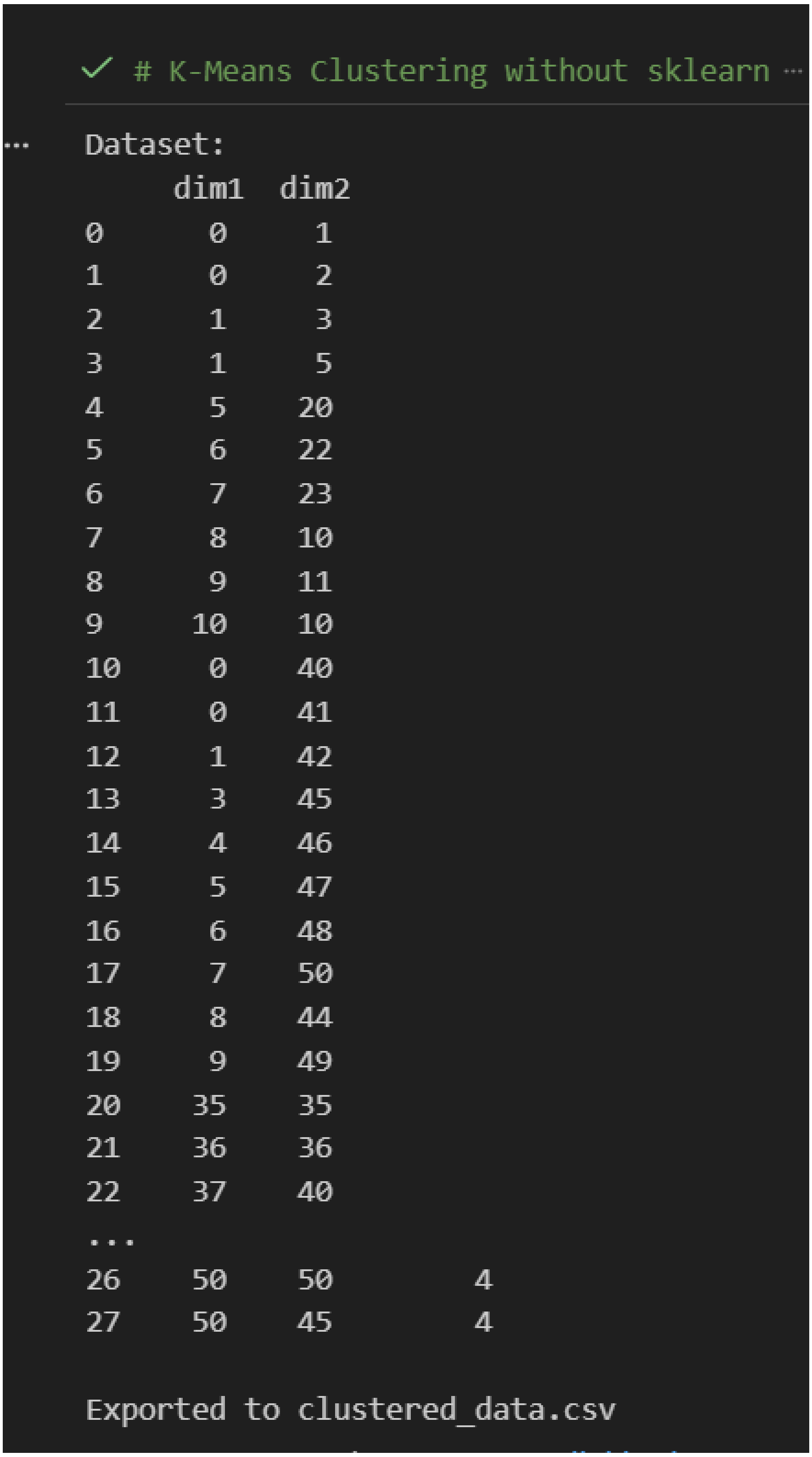
```
x=t1[["Height","Weight"]]
y=t1["Class"]

neigh=KNeighborsClassifier(n_neighbors=5)
neigh.fit(x,y)

pred=neigh.predict(t2[["Height","Weight"]])
prob=neigh.predict_proba(t2[["Height","Weight"]])

print("\nTest rows:\n",t2[["Height","Weight"]])
print("\nPredicted Class:",pred)
print("Probabilities:\n",prob)
```



```
✓ import pandas as pd ···

··· Dataset:
       Height  Weight        Class
    0     167      51  Underweight
    1     182      62       Normal
    2     176      69       Normal
    3     173      64       Normal
    4     172      65       Normal
    5     174      56  Underweight
    6     169      58       Normal
    7     173      57       Normal
    8     170      55       Normal
    9     170      57          NaN

    Test rows:
       Height  Weight
    9     170      57

    Predicted Class: ['Normal']
    Probabilities:
     [[0.6 0.4]]
```

EXP:4- K-Means Clustering (Manual)

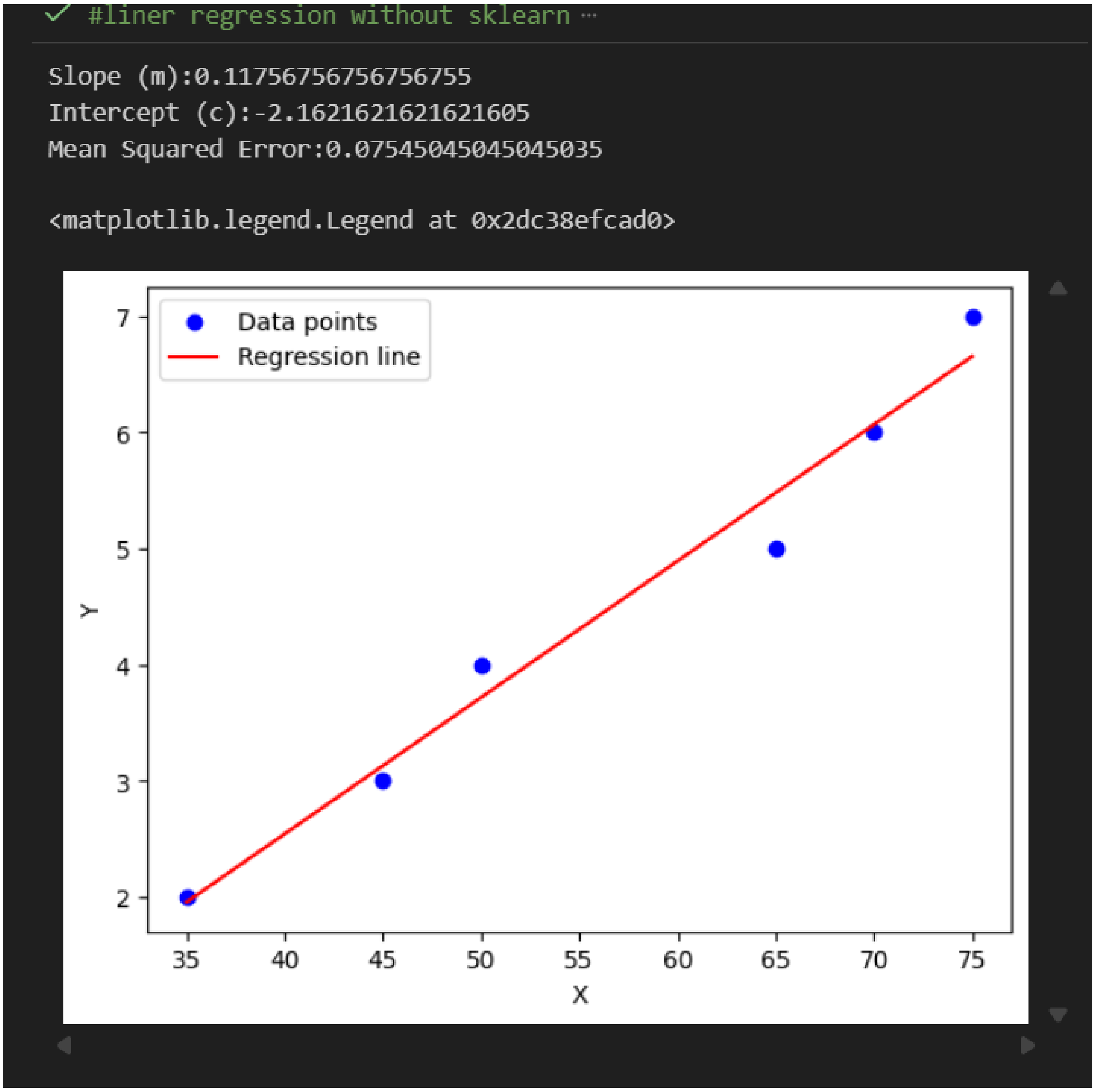
```python
# K-Means Clustering without sklearn
import pandas as pd
import numpy as np
import random
random.seed(69)
df=pd.read_excel("e:/sem 3/aiml lab/cluster_data.xlsx")
print("Dataset:\n",df)
x=df[["dim1","dim2"]].values
def kmeans(x,k,i=100):
    c=x[random.sample(range(len(x)),k)]
    for _ in range(i):
        l=[]
        for p in x:
            dists=[]
            for ci in c:
                d=((p[0]-ci[0])**2+(p[1]-ci[1])**2)**0.5
                dists.append(d)
            min_idx=0
            for j in range(1,len(dists)):
                if dists[j] < dists[min_idx]:
                    min_idx=j
            l.append(min_idx)
        nc=[]
        for j in range(k):
            pts=[x[m] for m in range(len(x)) if l[m]==j]
            if pts:
                nc.append(np.mean(pts,axis=0))
            else:
                nc.append(c[j])
        if np.allclose(c,nc):
            break
        c=nc
    return c,l
k=5
centers,labels=kmeans(x,k)
df["Cluster"]=[l+1 for l in labels]
print("\nCluster Centers:\n",centers)
print("\nClustered Dataset:\n",df)
df.to_csv("clustered_data.csv")
print("\nExported to clustered_data.csv")
```



```
✓ # K-Means Clustering without sklearn ···

Dataset:
      dim1  dim2
0        0     1
1        0     2
2        1     3
3        1     5
4        5    20
5        6    22
6        7    23
7        8    10
8        9    11
9       10    10
10       0    40
11       0    41
12       1    42
13       3    45
14       4    46
15       5    47
16       6    48
17       7    50
18       8    44
19       9    49
20      35    35
21      36    36
22      37    40
...
26      50    50         4
27      50    45         4

Exported to clustered_data.csv
```

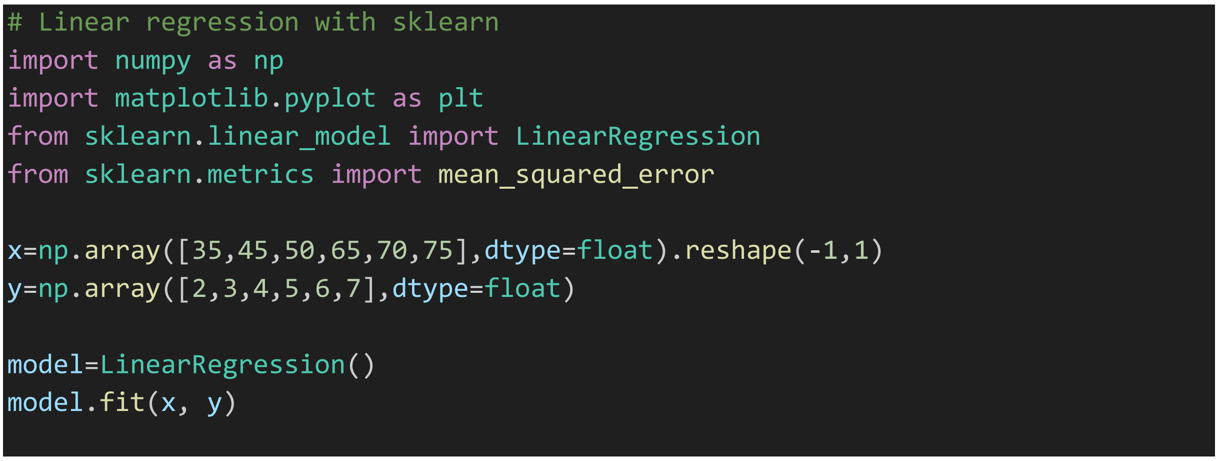
EXP:5- Linear Regression Without Sklearn

```
#liner regression without sklearn
import numpy as np
import matplotlib.pyplot as plt

x=np.array([35,45,50,65,70,75],dtype=float)
y=np.array([2,3,4,5,6,7],dtype=float)

xm=np.mean(x)
ym=np.mean(y)

num=np.sum((x-xm)*(y-ym))
den=np.sum((x-xm)**2)
m=num/den
c=ym-m*xm

print(f"Slope (m):{m}")
print(f"Intercept (c):{c}")

yp=m*x+c
mse=np.mean((y-yp)**2)
print(f"Mean Squared Error:{mse}")
plt.scatter(x,y,color='blue',label='Data points')
plt.plot(x,yp,color='red',label='Regression line')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
```



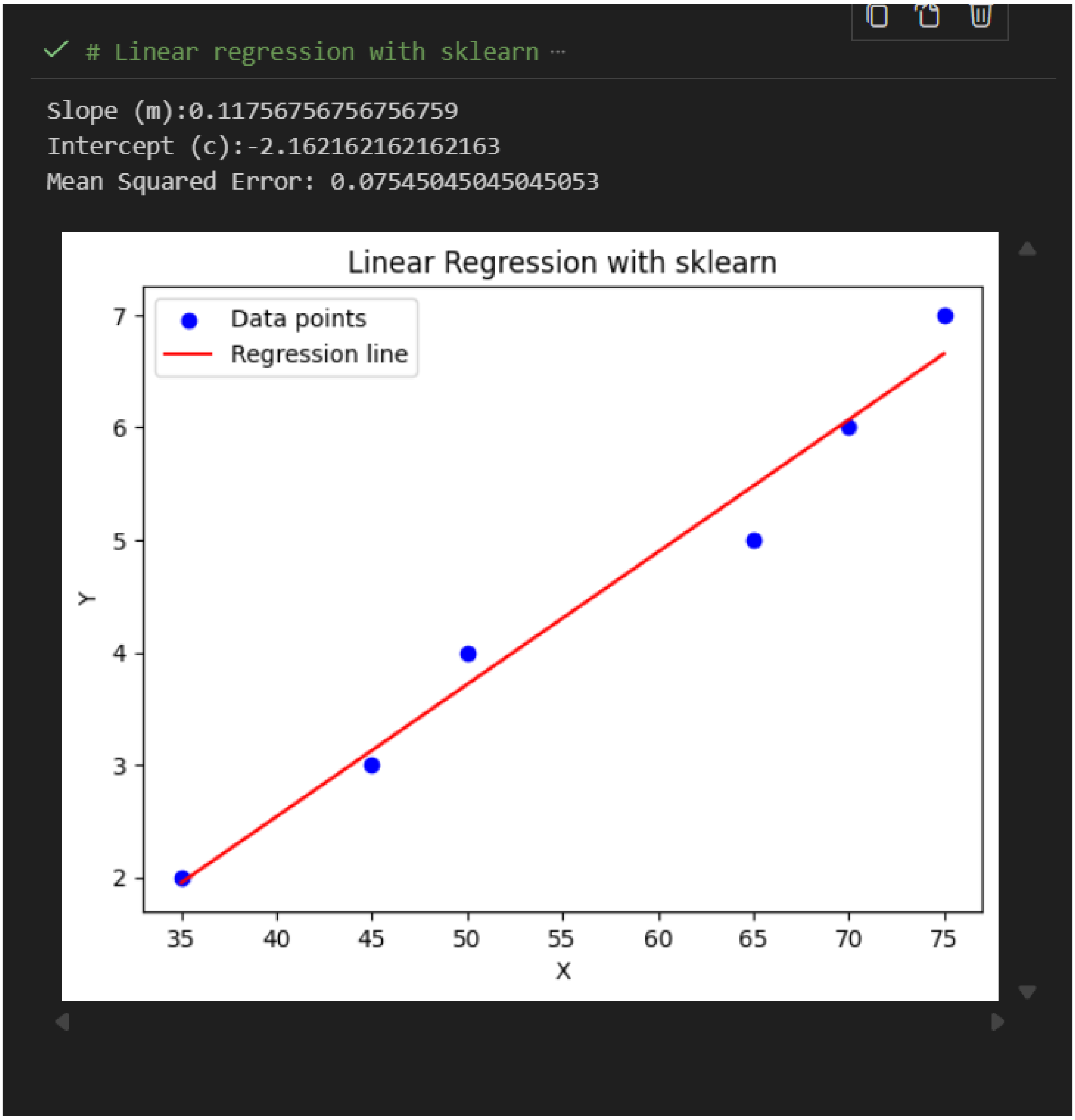
```
✓ #liner regression without sklearn ···

Slope (m):0.11756756756756755
Intercept (c):-2.1621621621621605
Mean Squared Error:0.07545045045045035

<matplotlib.legend.Legend at 0x2dc38efcad0>
```

EXP:6- Linear Regression With Sklearn



```
# Linear regression with sklearn
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

x=np.array([35,45,50,65,70,75],dtype=float).reshape(-1,1)
y=np.array([2,3,4,5,6,7],dtype=float)

model=LinearRegression()
model.fit(x, y)
```

```
m=model.coef_[0]
c=model.intercept_

print(f"Slope (m):{m}")
print(f"Intercept (c):{c}")

yp=model.predict(x)

mse=mean_squared_error(y, yp)
print(f"Mean Squared Error: {mse}")

plt.scatter(x, y, color='blue', label='Data points')
plt.plot(x, yp, color='red', label='Regression line')
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Linear Regression with sklearn')
plt.legend()
plt.show()
```

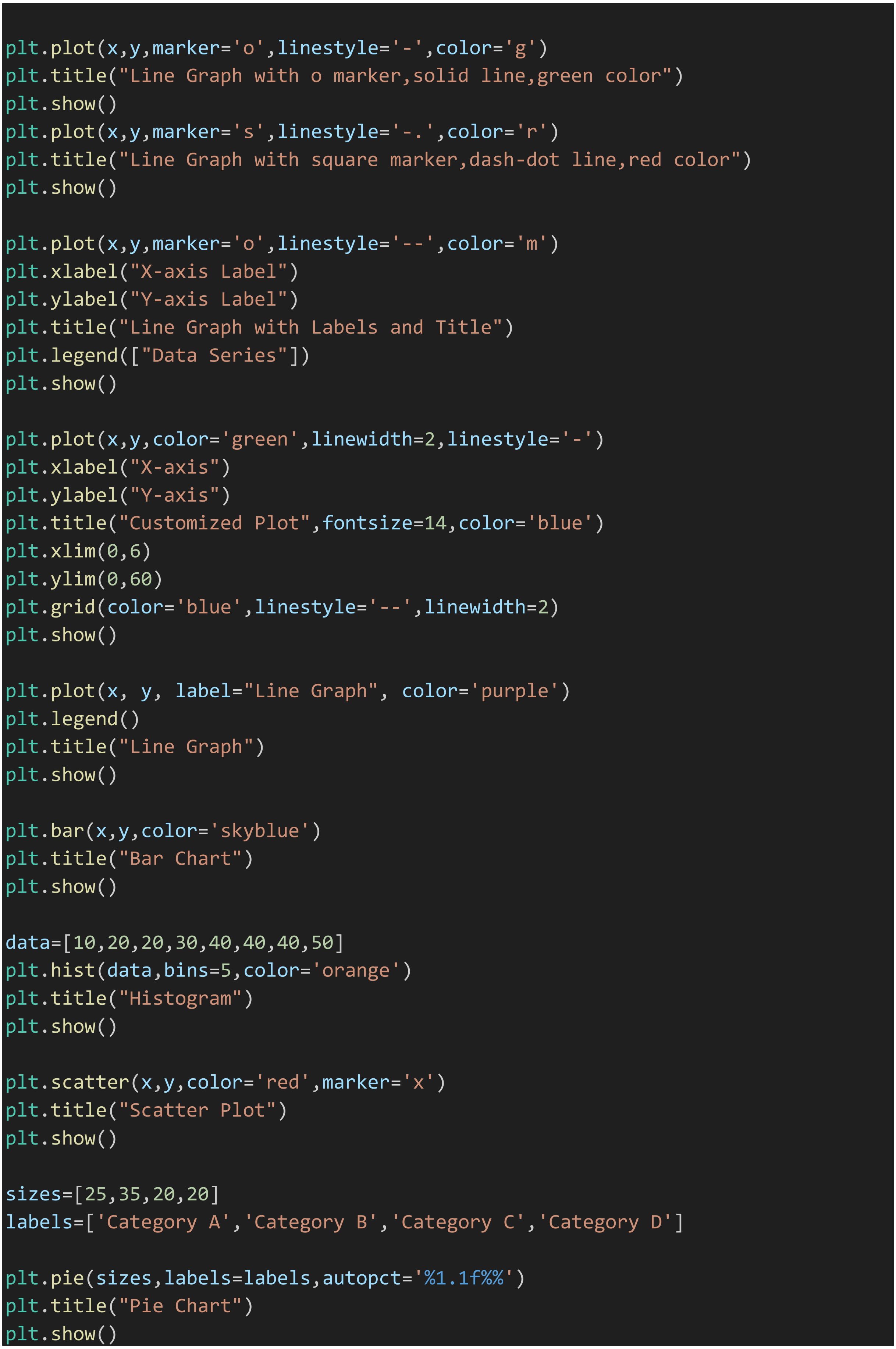


```
# Linear regression with sklearn ...
```

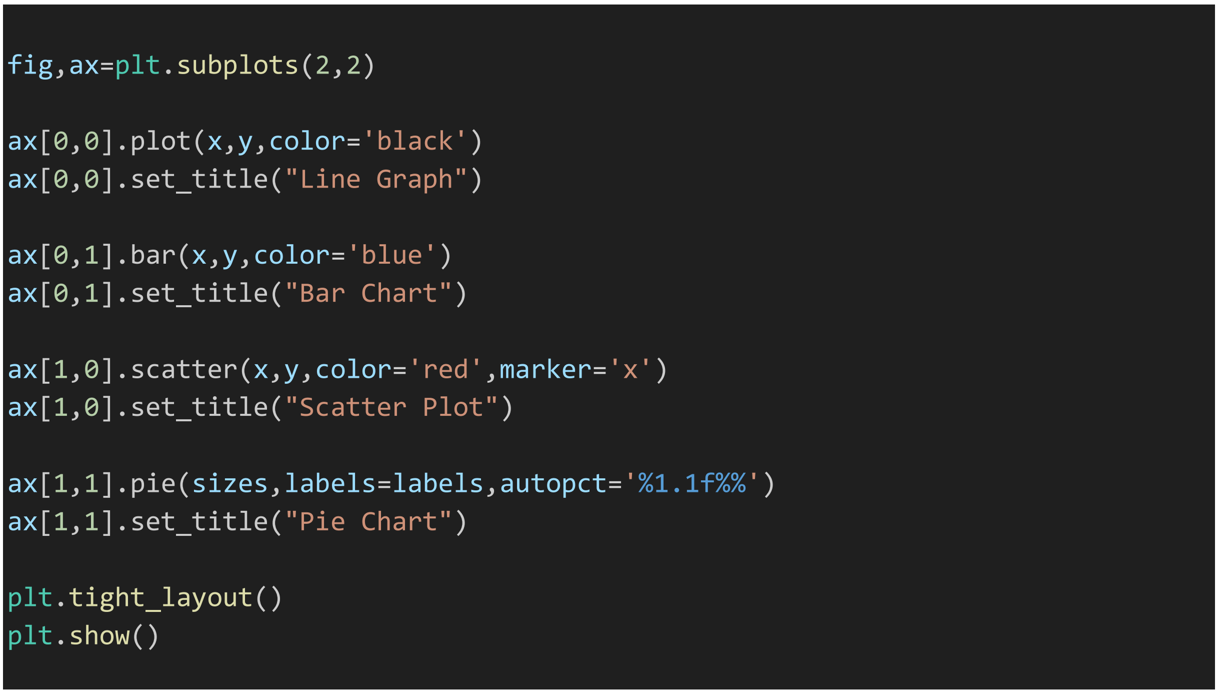
```
Slope (m):0.11756756756756759
Intercept (c):-2.162162162162163
Mean Squared Error: 0.07545045045045053
```

EXP:7- Matplotlib Plotting Techniques

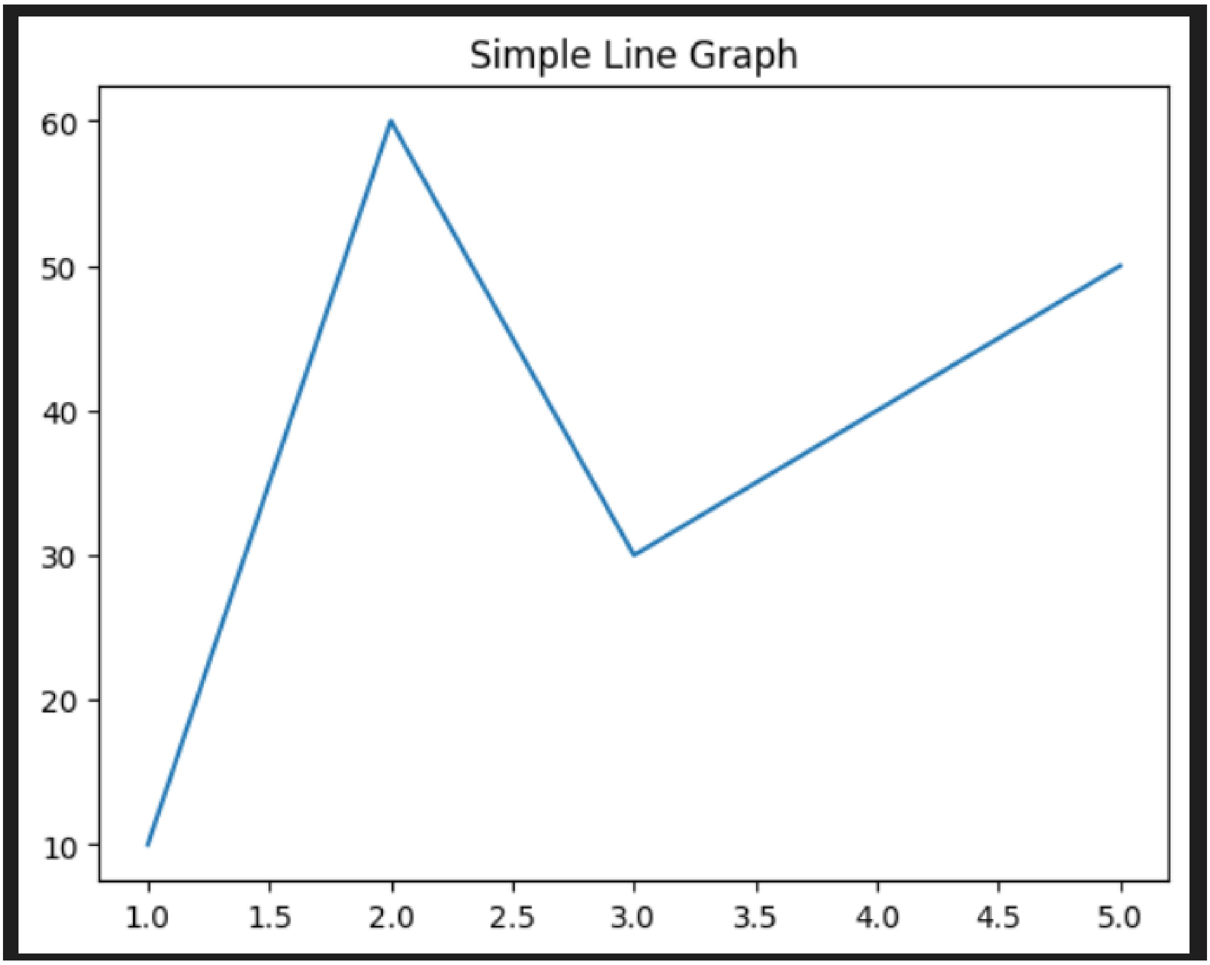
```
import matplotlib.pyplot as plt

x=[1,2,3,4,5]
y=[10,60,30,40,50]
plt.plot(x,y)
plt.title("Simple Line Graph")
plt.show()

plt.plot(x,y,marker='+',linestyle='--',color='b')
plt.title("Line Graph with + marker,dashed line,blue color")
plt.show()
```

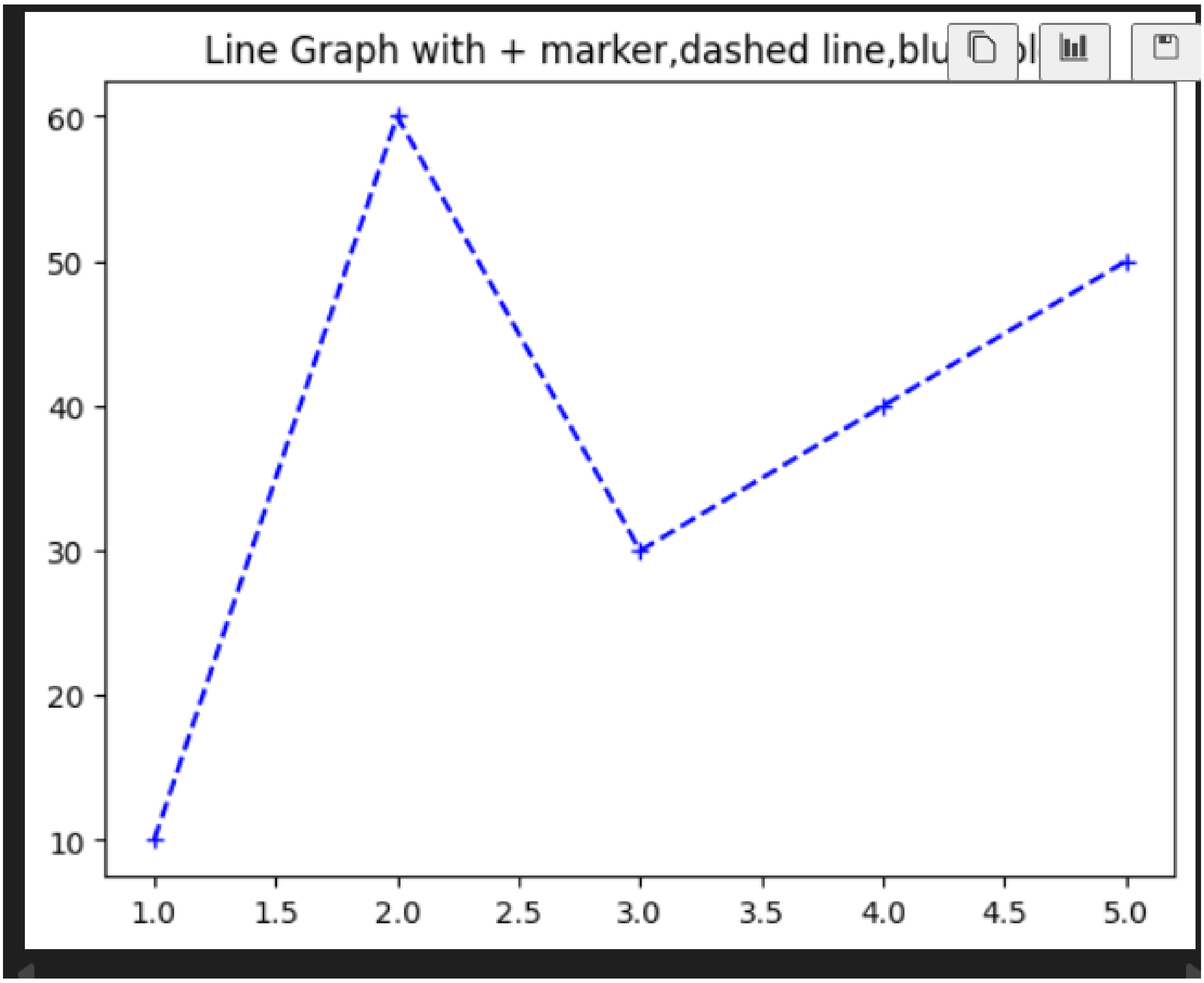


```
plt.plot(x,y,marker='o',linestyle='-',color='g')
plt.title("Line Graph with o marker,solid line,green color")
plt.show()
plt.plot(x,y,marker='s',linestyle='-.',color='r')
plt.title("Line Graph with square marker,dash-dot line,red color")
plt.show()

plt.plot(x,y,marker='o',linestyle='--',color='m')
plt.xlabel("X-axis Label")
plt.ylabel("Y-axis Label")
plt.title("Line Graph with Labels and Title")
plt.legend(["Data Series"])
plt.show()

plt.plot(x,y,color='green',linewidth=2,linestyle='-')
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.title("Customized Plot",fontsize=14,color='blue')
plt.xlim(0,6)
plt.ylim(0,60)
plt.grid(color='blue',linestyle='--',linewidth=2)
plt.show()

plt.plot(x, y, label="Line Graph", color='purple')
plt.legend()
plt.title("Line Graph")
plt.show()

plt.bar(x,y,color='skyblue')
plt.title("Bar Chart")
plt.show()

data=[10,20,20,30,40,40,40,50]
plt.hist(data,bins=5,color='orange')
plt.title("Histogram")
plt.show()

plt.scatter(x,y,color='red',marker='x')
plt.title("Scatter Plot")
plt.show()

sizes=[25,35,20,20]
labels=['Category A','Category B','Category C','Category D']

plt.pie(sizes,labels=labels,autopct='%1.1f%%')
plt.title("Pie Chart")
plt.show()
```

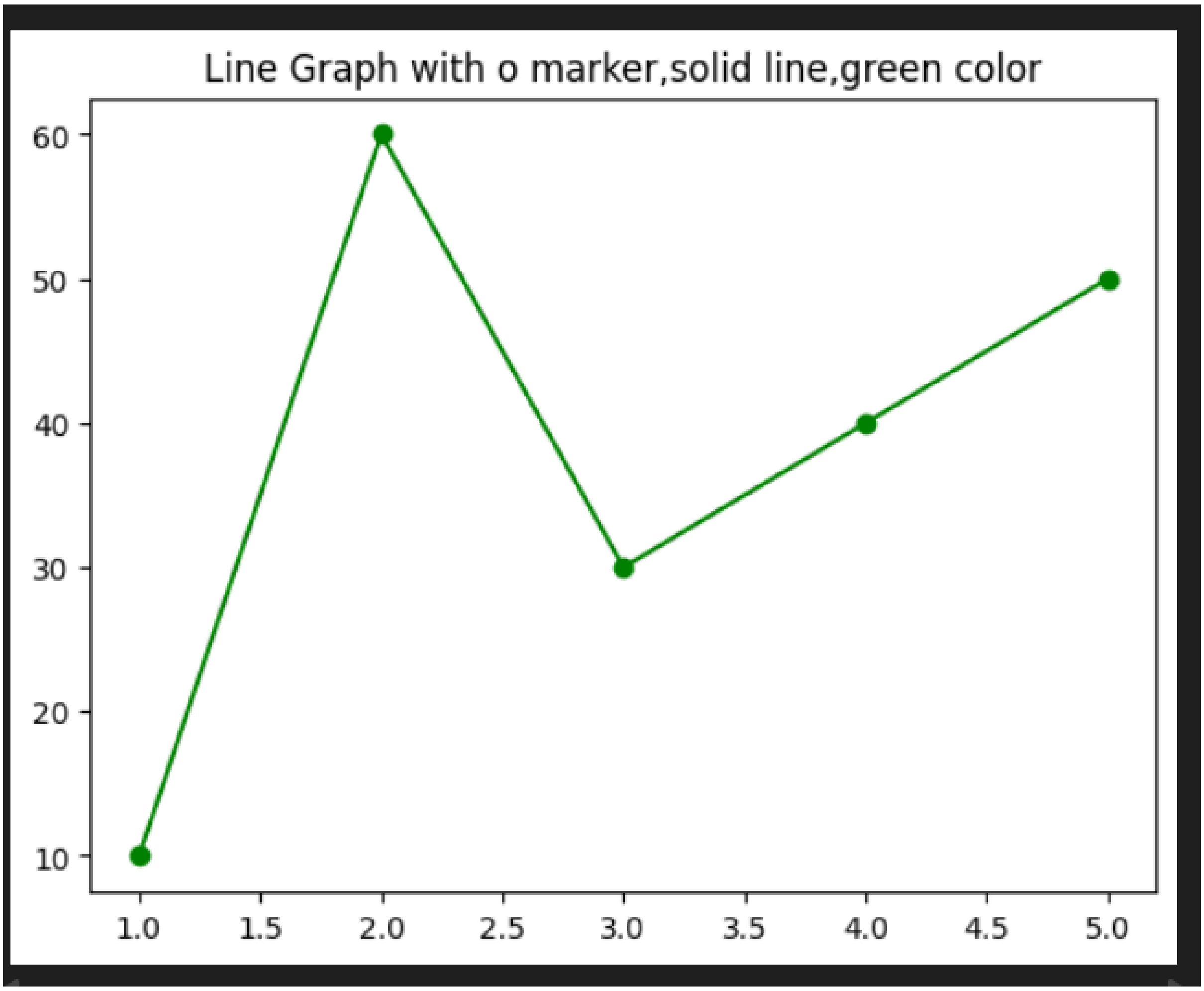


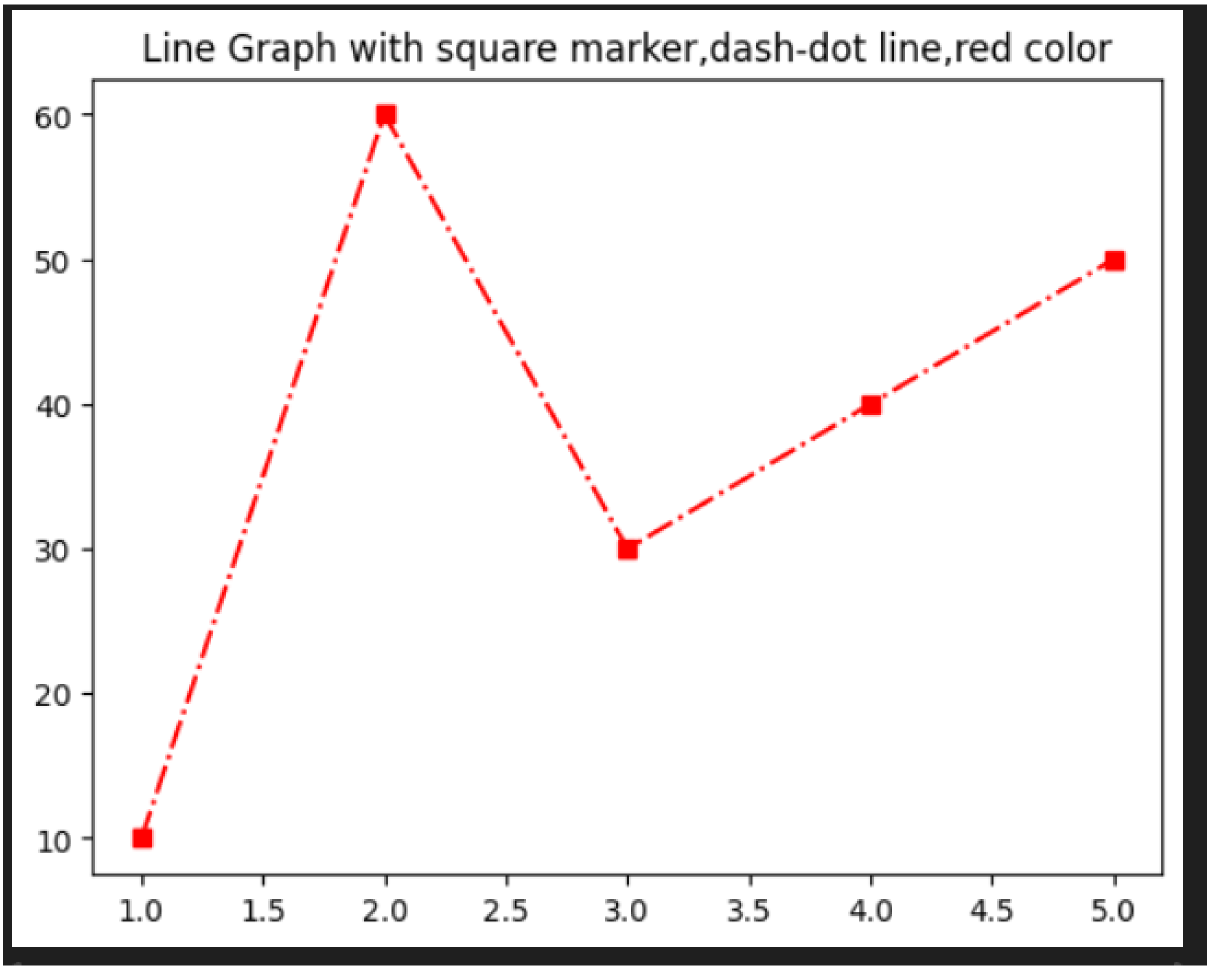
```
fig,ax=plt.subplots(2,2)

ax[0,0].plot(x,y,color='black')
ax[0,0].set_title("Line Graph")

ax[0,1].bar(x,y,color='blue')
ax[0,1].set_title("Bar Chart")

ax[1,0].scatter(x,y,color='red',marker='x')
ax[1,0].set_title("Scatter Plot")

ax[1,1].pie(sizes,labels=labels,autopct='%1.1f%%')
ax[1,1].set_title("Pie Chart")

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

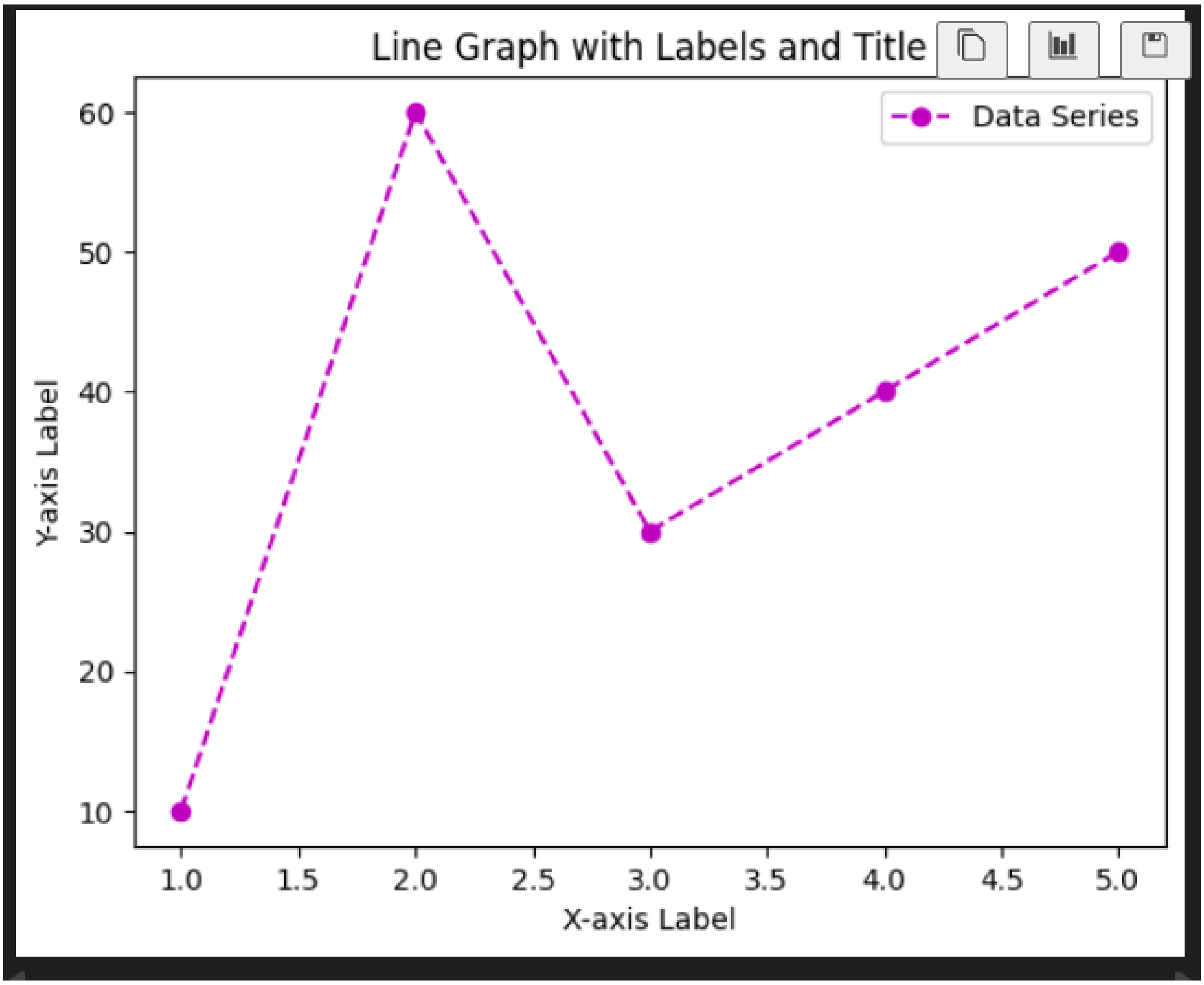


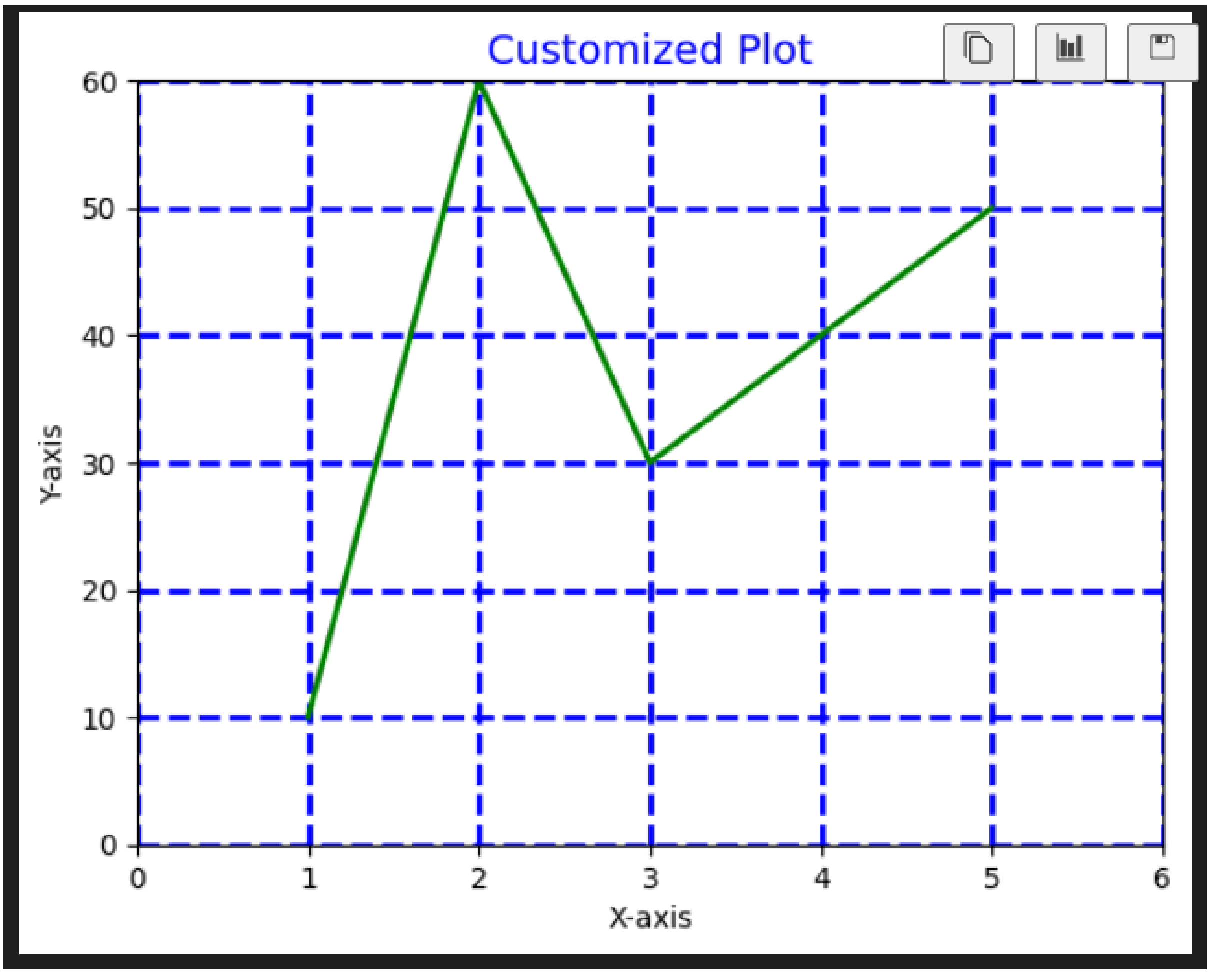


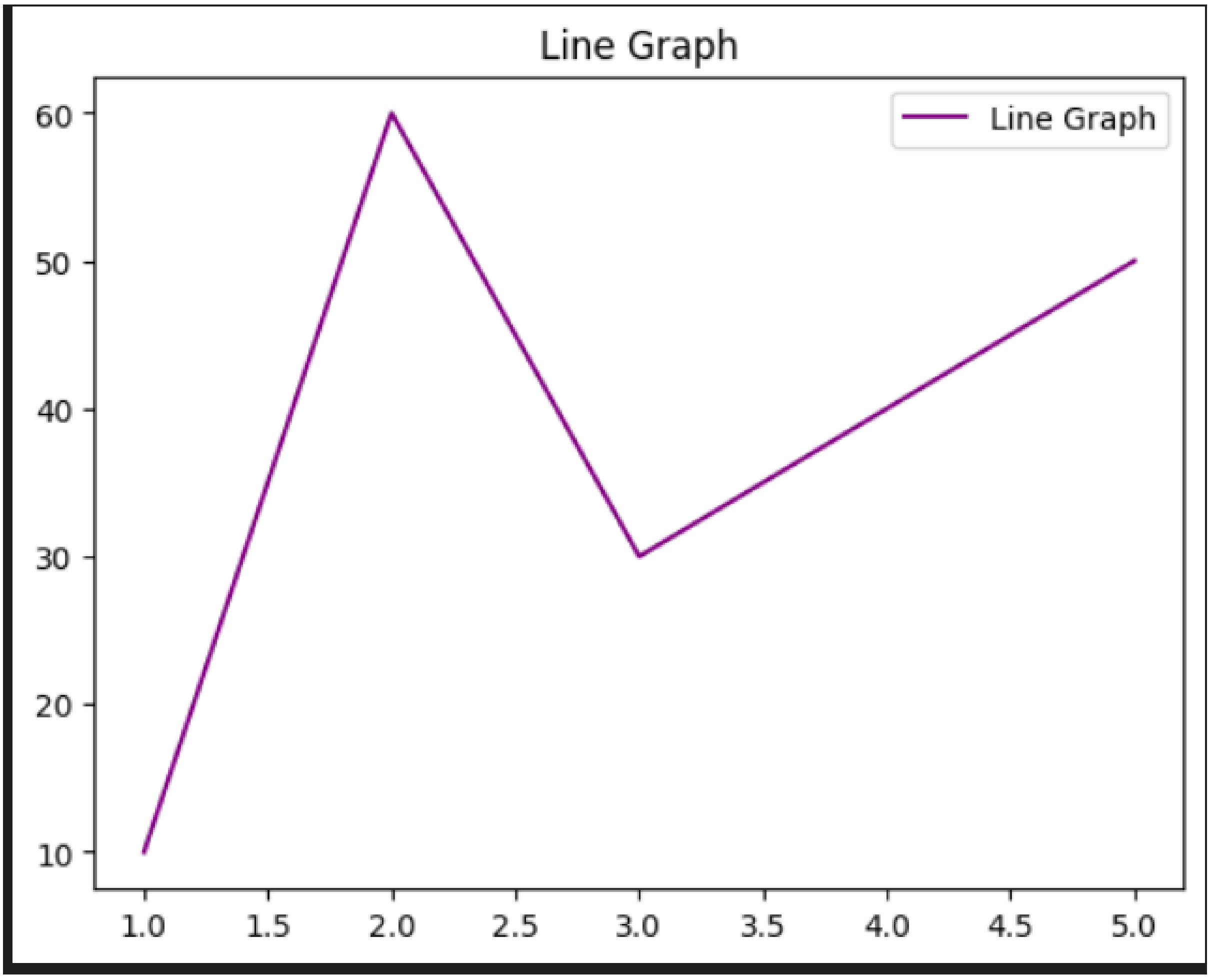
Line Graph with + marker,dashed line,blu
l
60
50
40
30
20
10
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0

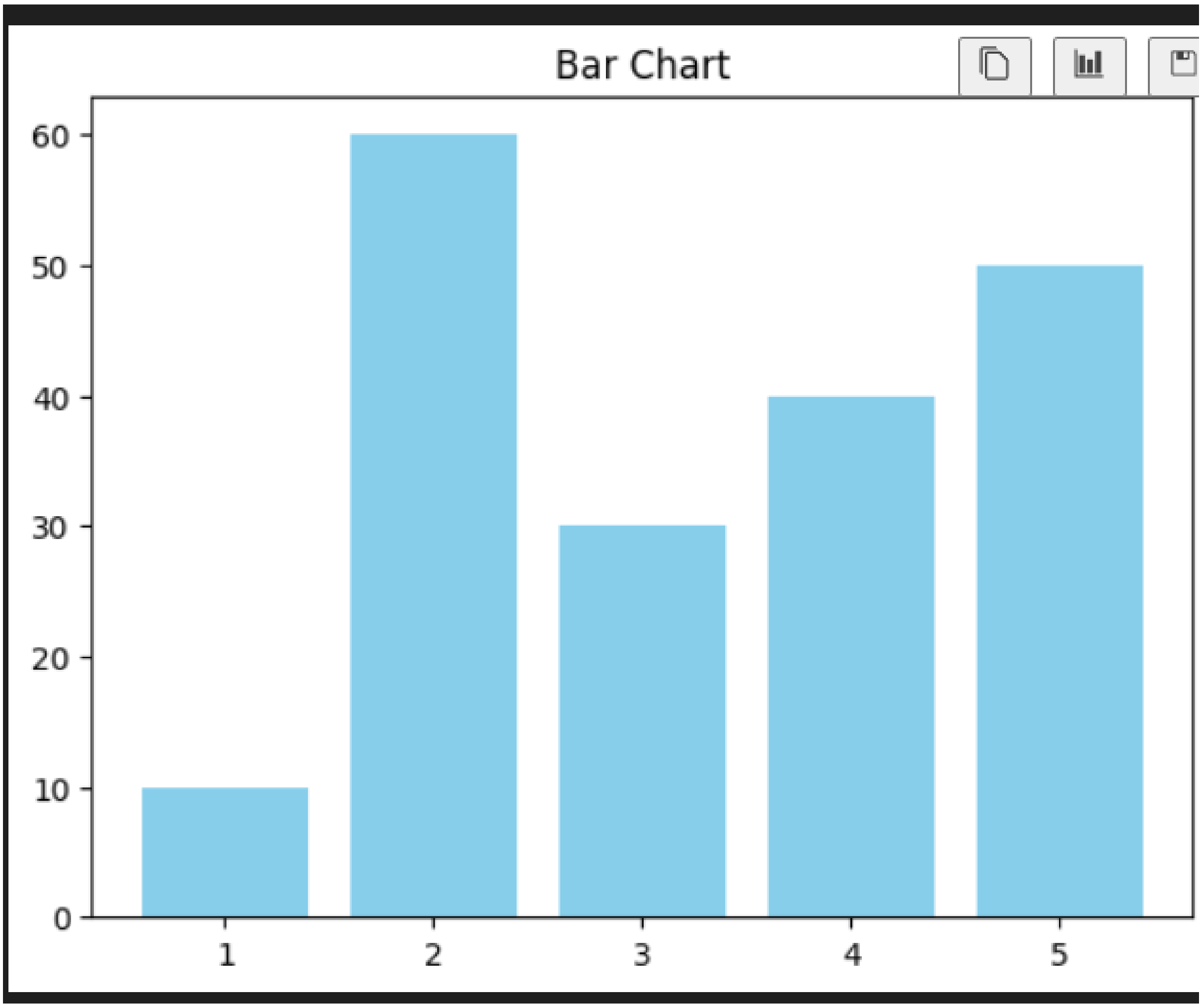

Line Graph with o marker,solid line,green color
60
50
40
30
20
10
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0

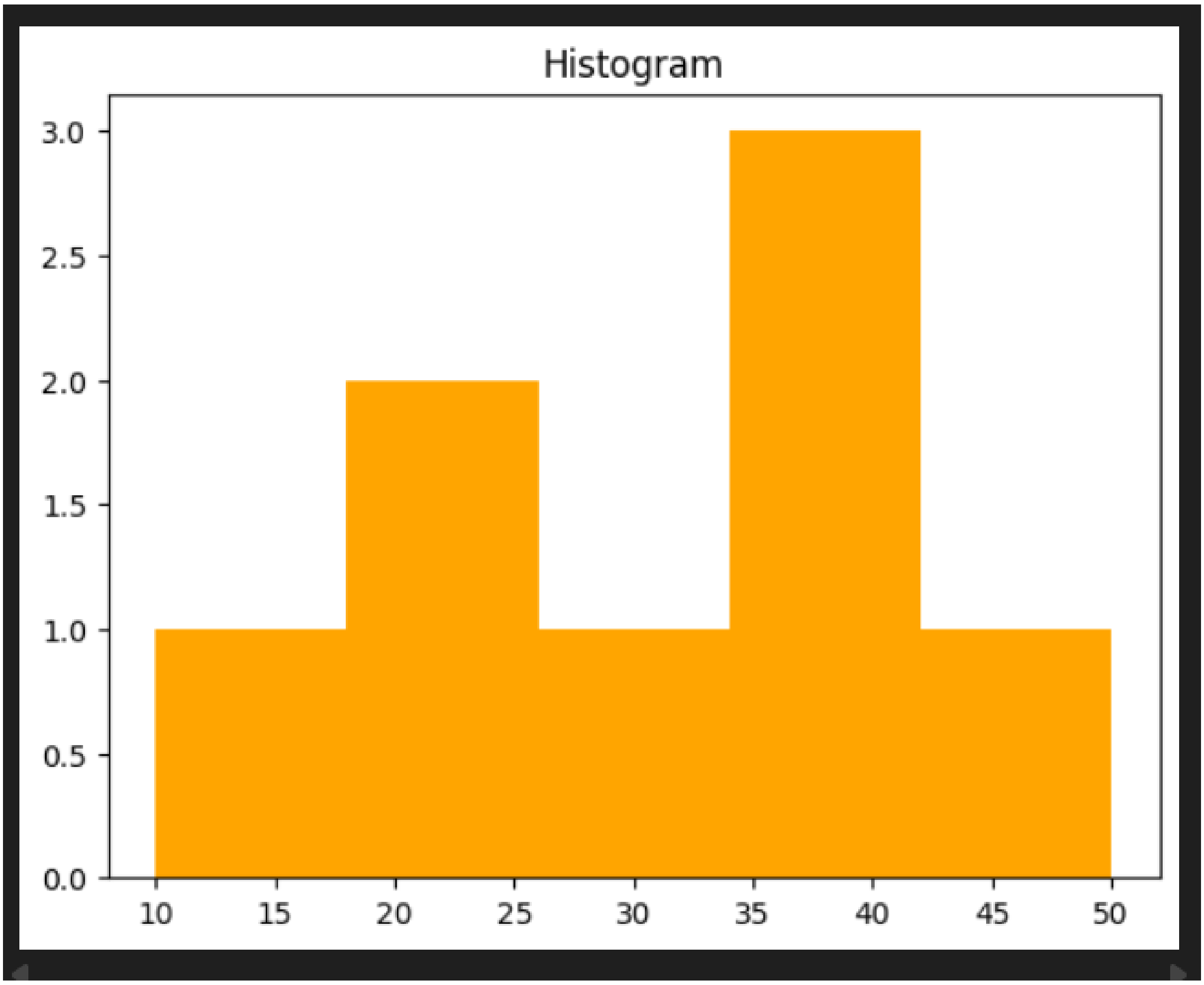

Line Graph with square marker,dash-dot line,red color
60
50
40
30
20
10
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0

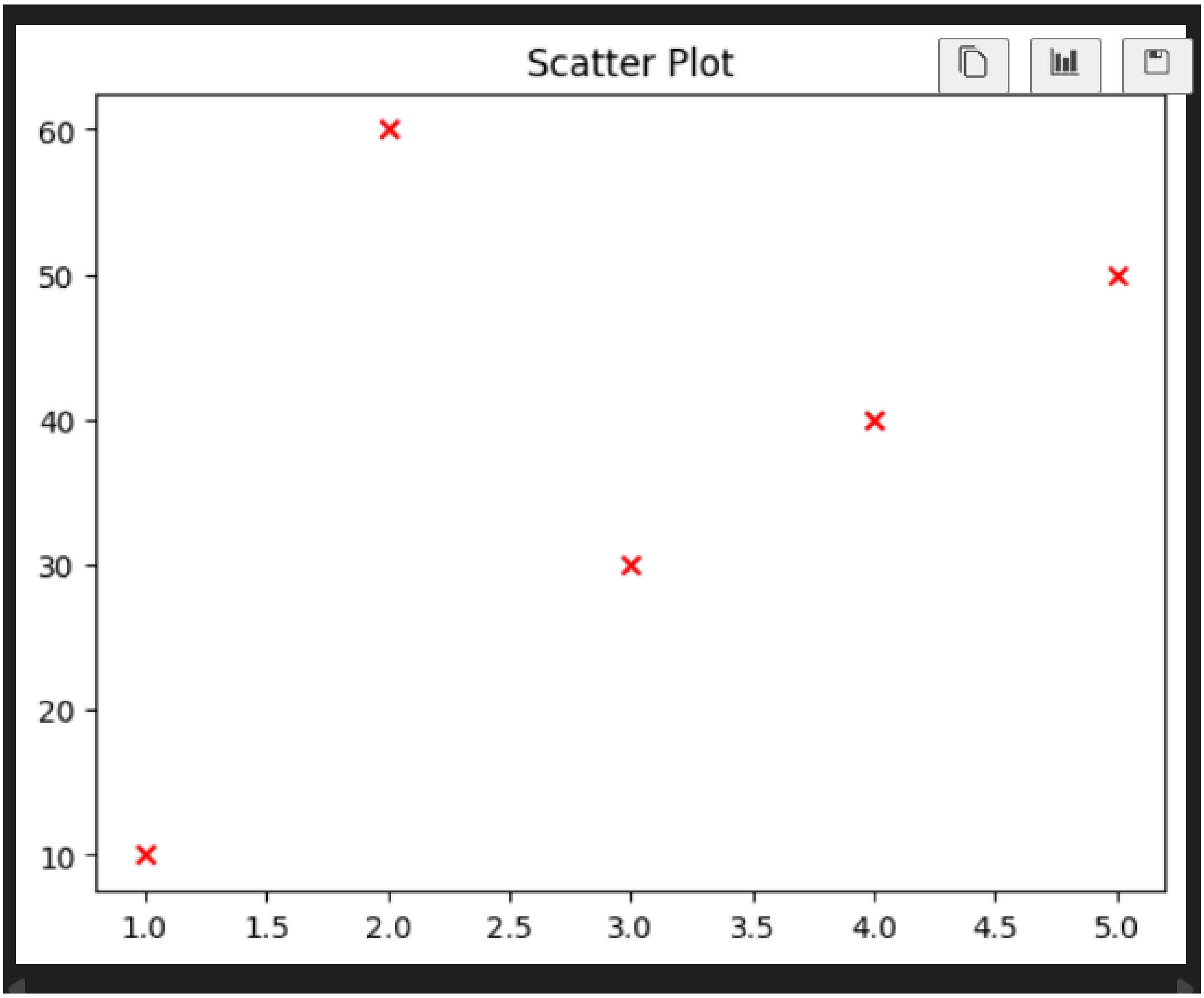
Line Graph with Labels and Title
Data Series
60
50
40
30
20
10
Y-axis Label
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
X-axis Label

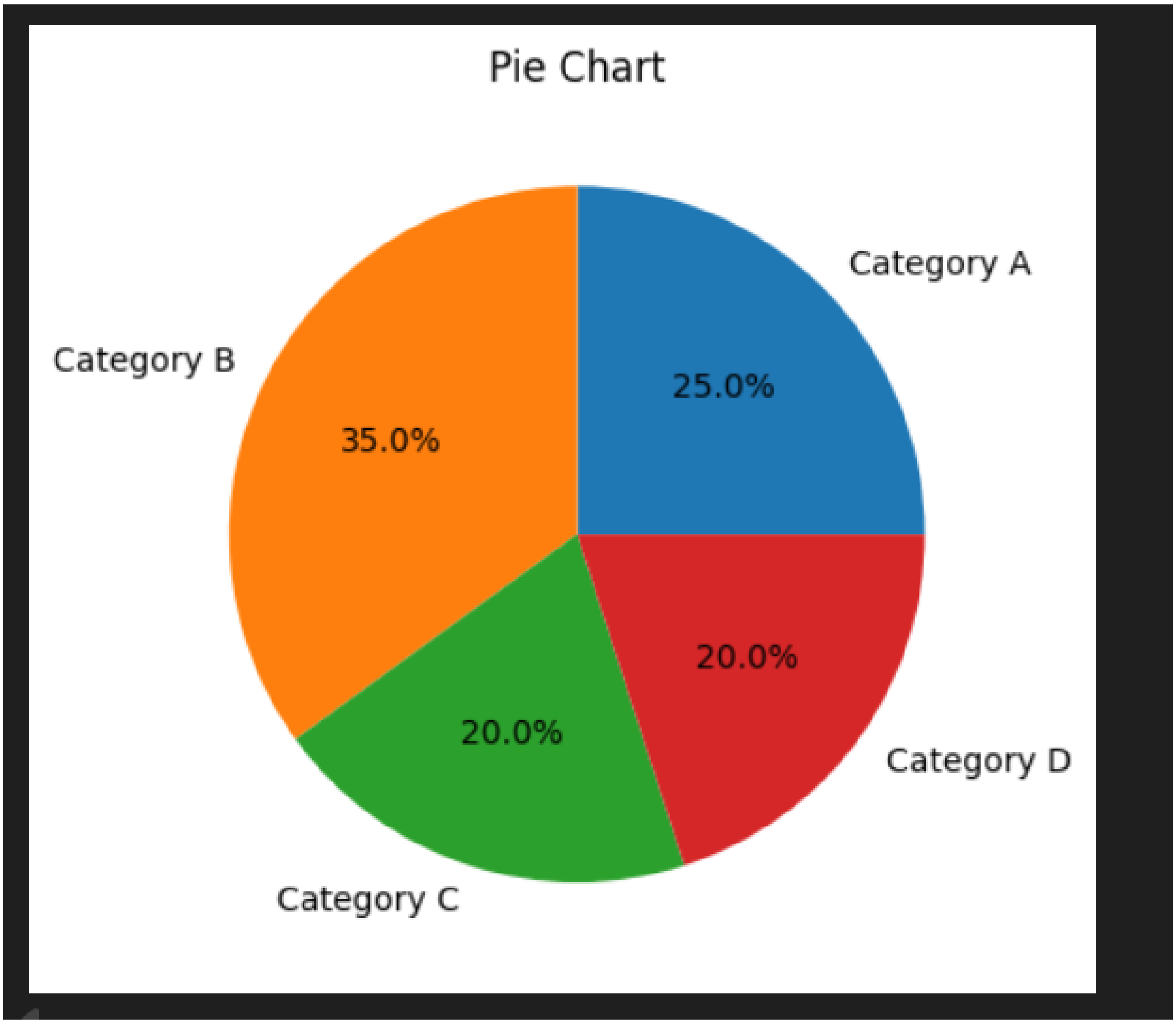

Customized Plot
60
50
40
30
20
10
0
Y-axis
0
1
2
3
4
5
6
X-axis

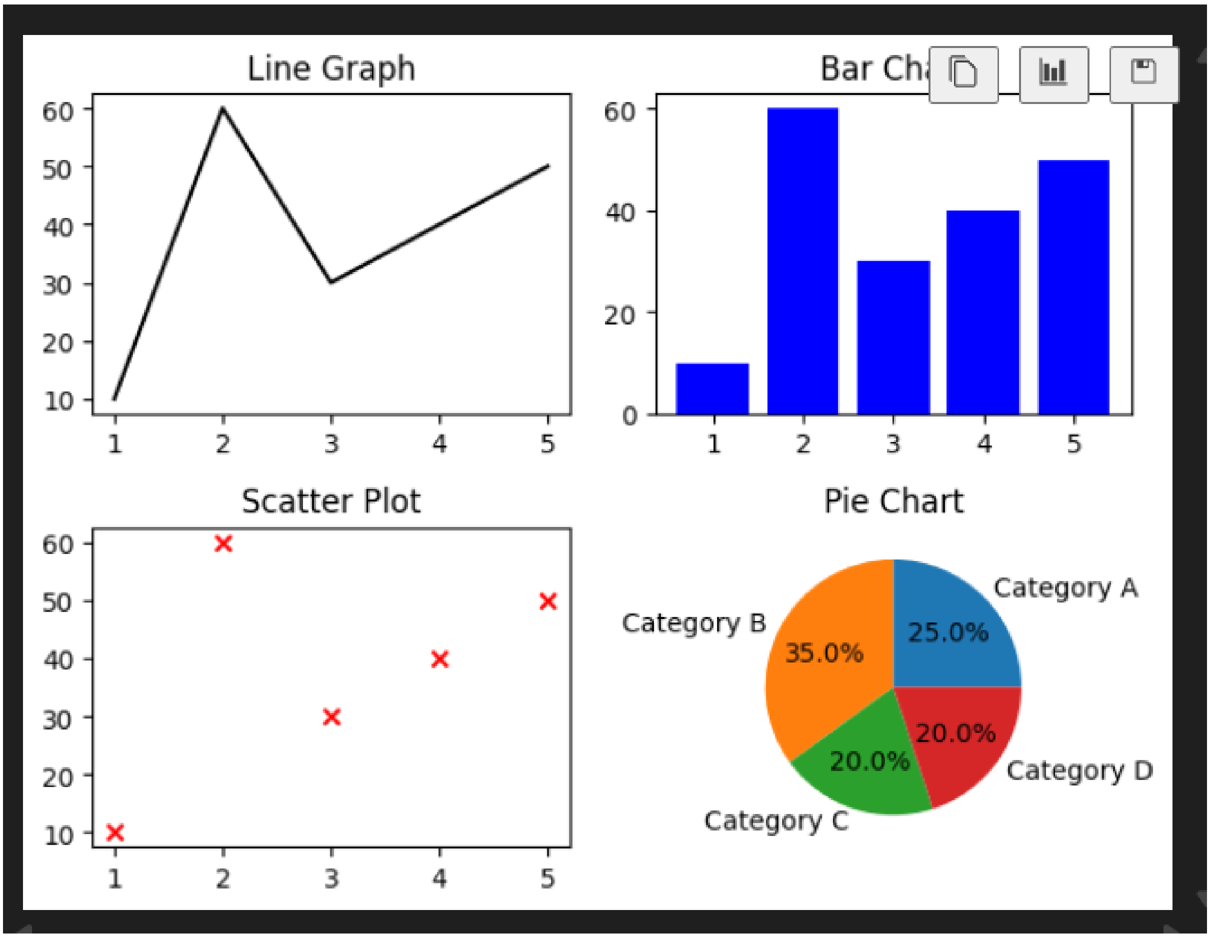

Line Graph
Line Graph
60
50
40
30
20
10
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0


Bar Chart
60
50
40
30
20
10
0
1
2
3
4
5

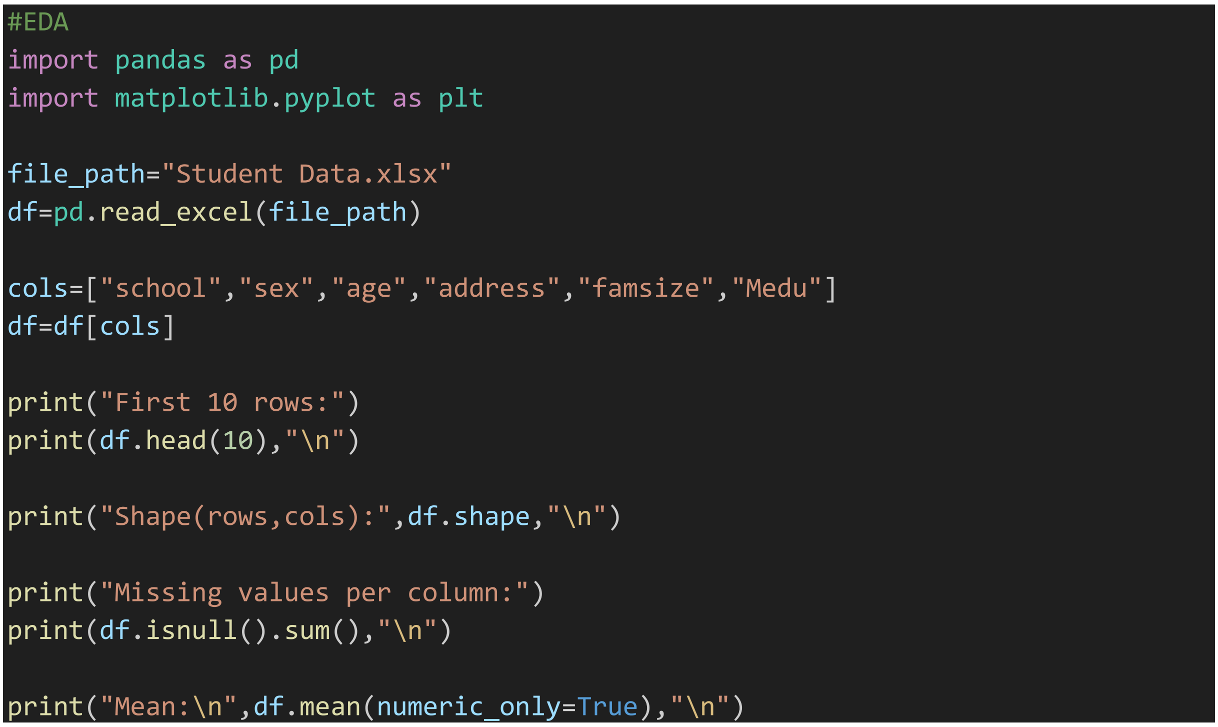

Histogram
3.0
2.5
2.0
1.5
1.0
0.5
0.0
10
15
20
25
30
35
40
45
50


Scatter Plot
60
50
40
30
20
10
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0

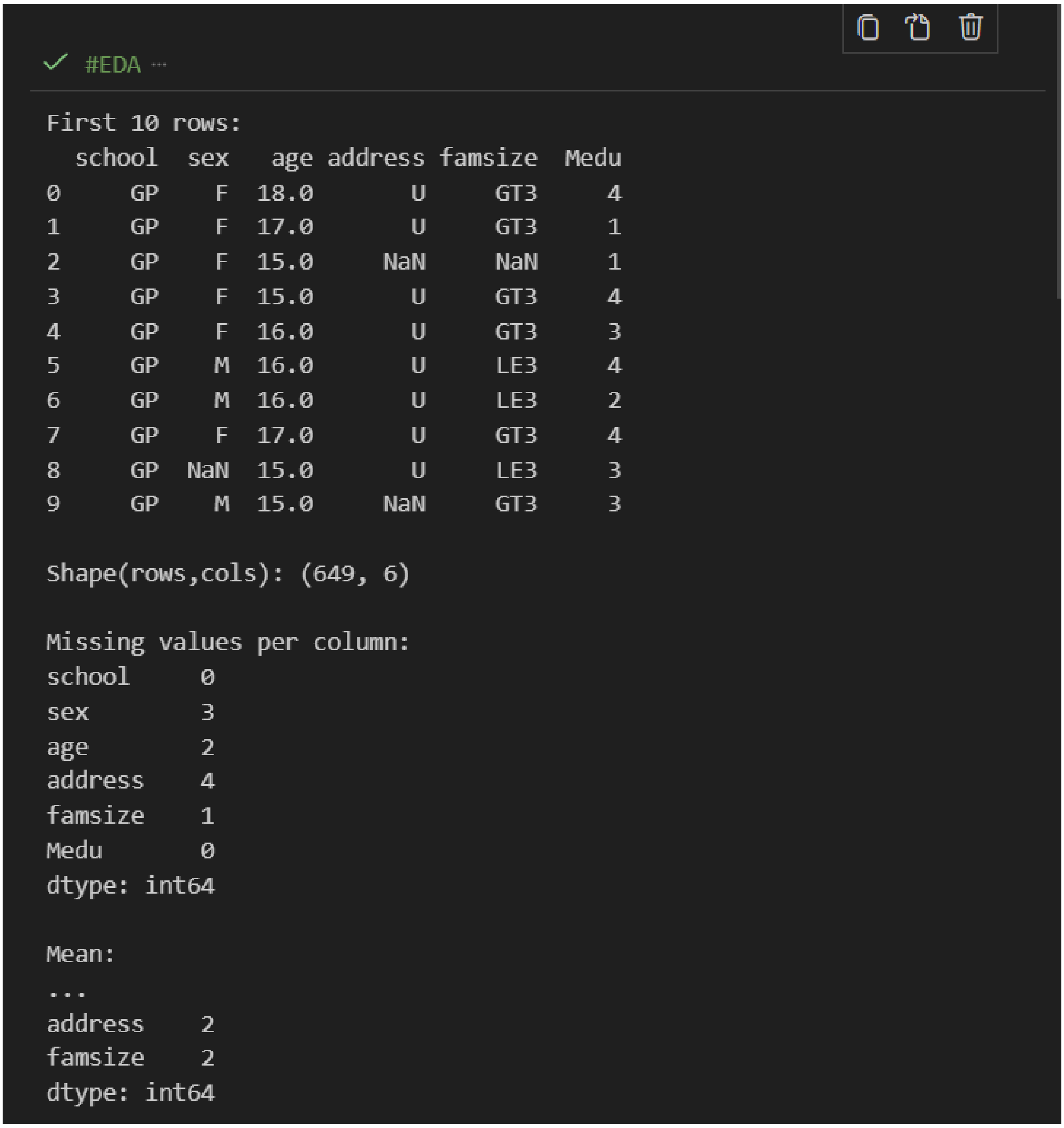

Pie Chart
Category A
25.0%
Category B
35.0%
20.0%
Category C
20.0%
Category D



EXP:8- Student Dataset EDA

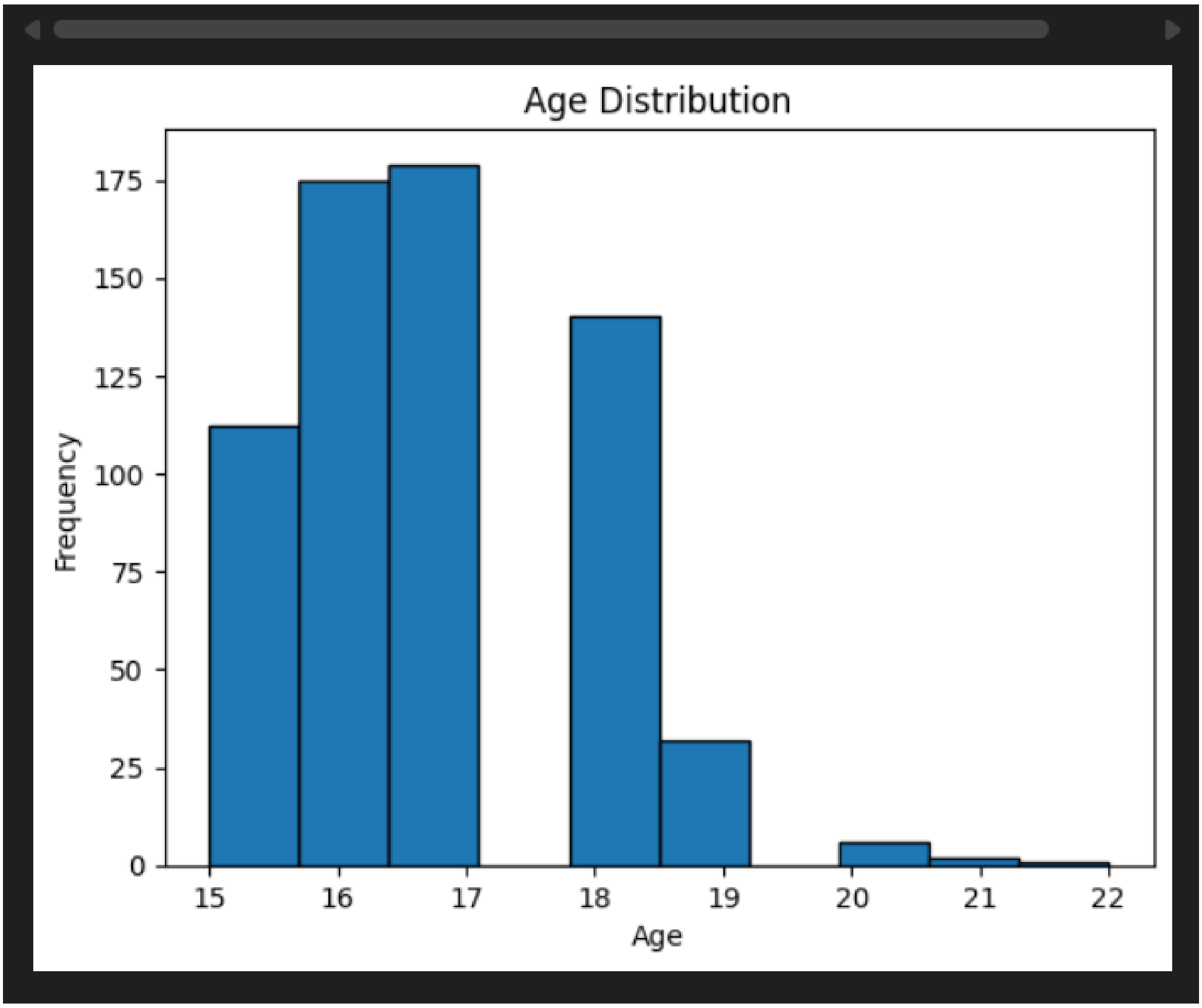


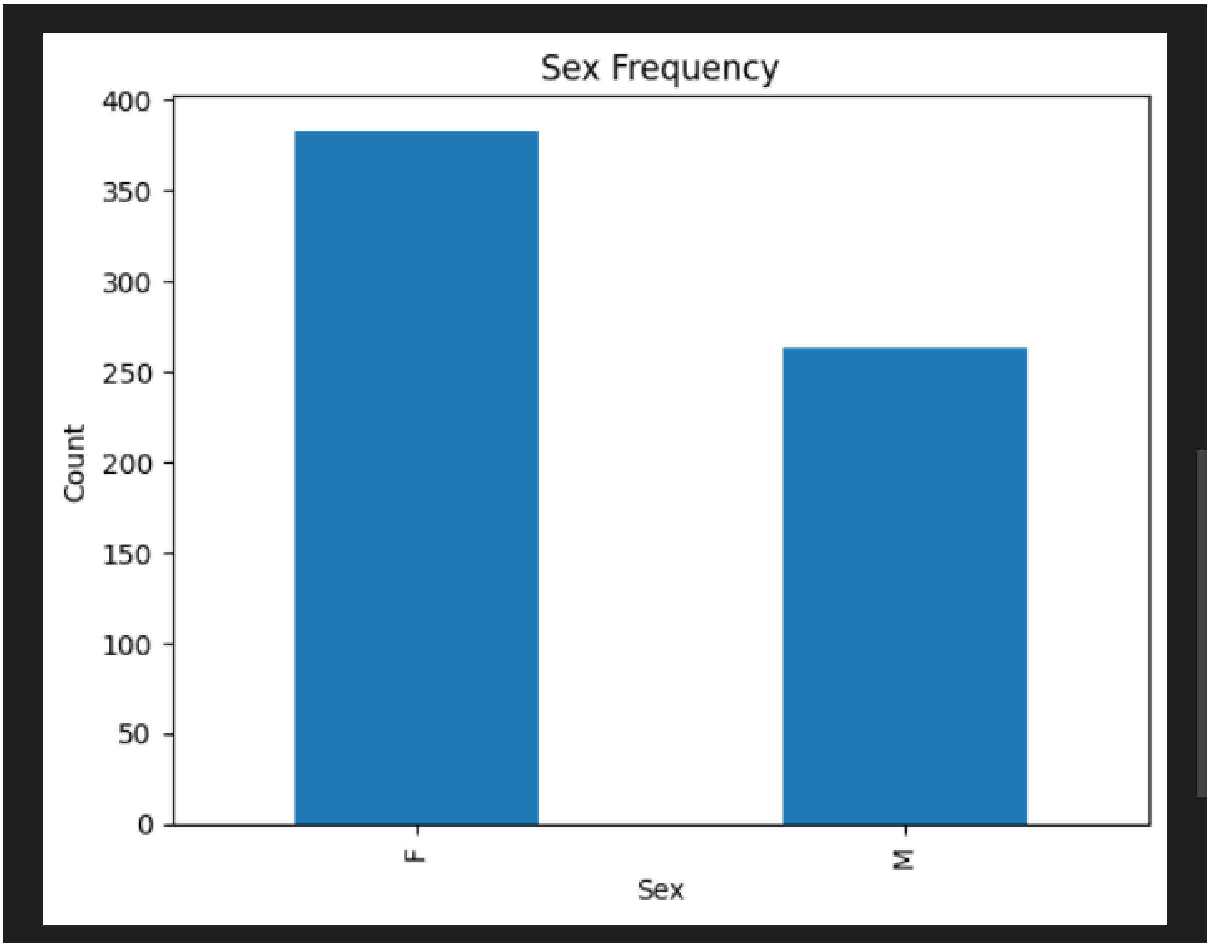
```
#EDA
import pandas as pd
import matplotlib.pyplot as plt

file_path="Student Data.xlsx"
df=pd.read_excel(file_path)

cols=["school","sex","age","address","famsize","Medu"]
df=df[cols]

print("First 10 rows:")
print(df.head(10),"\n")

print("Shape(rows,cols):",df.shape,"\n")

print("Missing values per column:")
print(df.isnull().sum(),"\n")

print("Mean:\n",df.mean(numeric_only=True),"\n")
print("Median:\n",df.median(numeric_only=True),"\n")
print("Mode:\n",df.mode().iloc[0],"\n")
print("Standard Deviation:\n",df.std(numeric_only=True),"\n")

print("Max values:\n",df.max(numeric_only=True),"\n")
print("Min values:\n",df.min(numeric_only=True),"\n")

print("Unique values per categorical column:")
print(df.select_dtypes(include=['object']).nunique(),"\n")

df['age'].plot(kind='hist',bins=10,edgecolor='black',title="Age Distribution")
plt.xlabel("Age")
plt.show()

df['sex'].value_counts().plot(kind='bar',title="Sex Frequency")
plt.xlabel("Sex")
plt.ylabel("Count")
plt.show()

plt.scatter(df['age'],df['Medu'])
plt.xlabel("Age")
plt.ylabel("Medu (Mother's Education)")
plt.title("Scatter plot: Age vs Medu")
plt.show()

df['age'].plot(kind='box',title="Boxplot of Age")
plt.show()
```

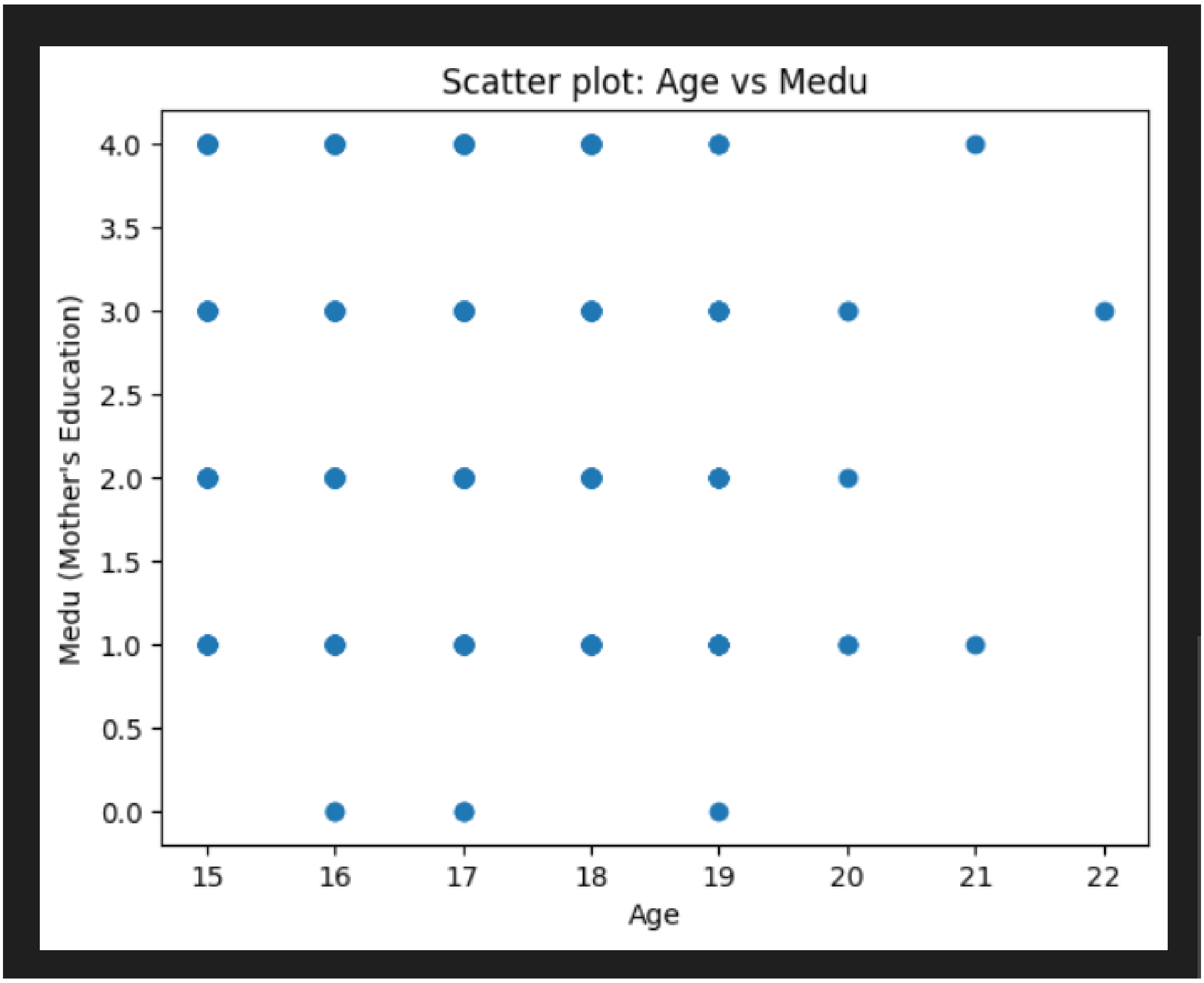


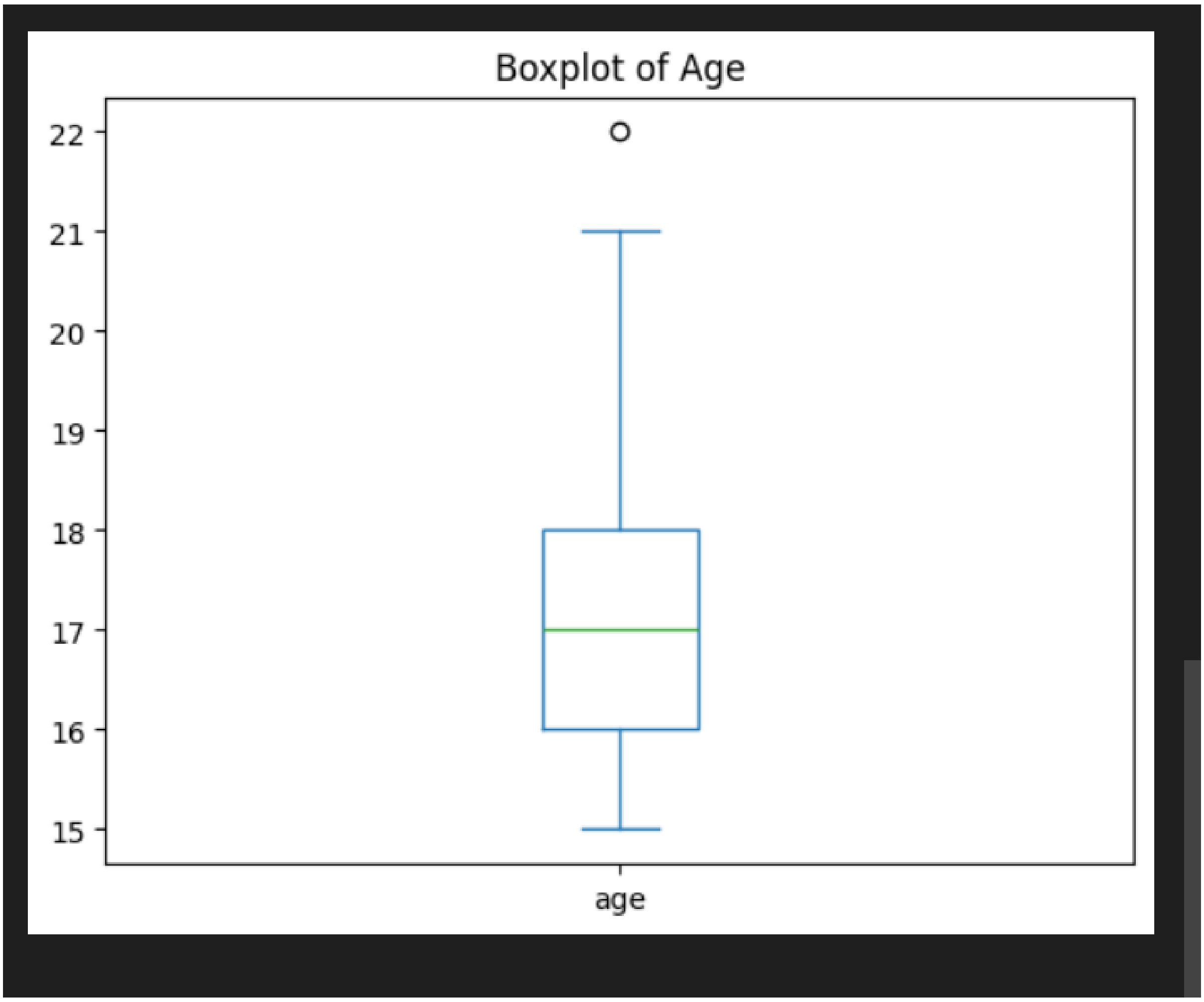
✓ #EDA ···

```
First 10 rows:
  school  sex   age address famsize  Medu
0     GP    F  18.0       U     GT3     4
1     GP    F  17.0       U     GT3     1
2     GP    F  15.0     NaN     NaN     1
3     GP    F  15.0       U     GT3     4
4     GP    F  16.0       U     GT3     3
5     GP    M  16.0       U     LE3     4
6     GP    M  16.0       U     LE3     2
7     GP    F  17.0       U     GT3     4
8     GP  NaN  15.0       U     LE3     3
9     GP    M  15.0     NaN     GT3     3

Shape(rows,cols): (649, 6)

Missing values per column:
school     0
sex        3
age        2
address    4
famsize    1
Medu       0
dtype: int64

Mean:
...
address    2
famsize    2
dtype: int64
```

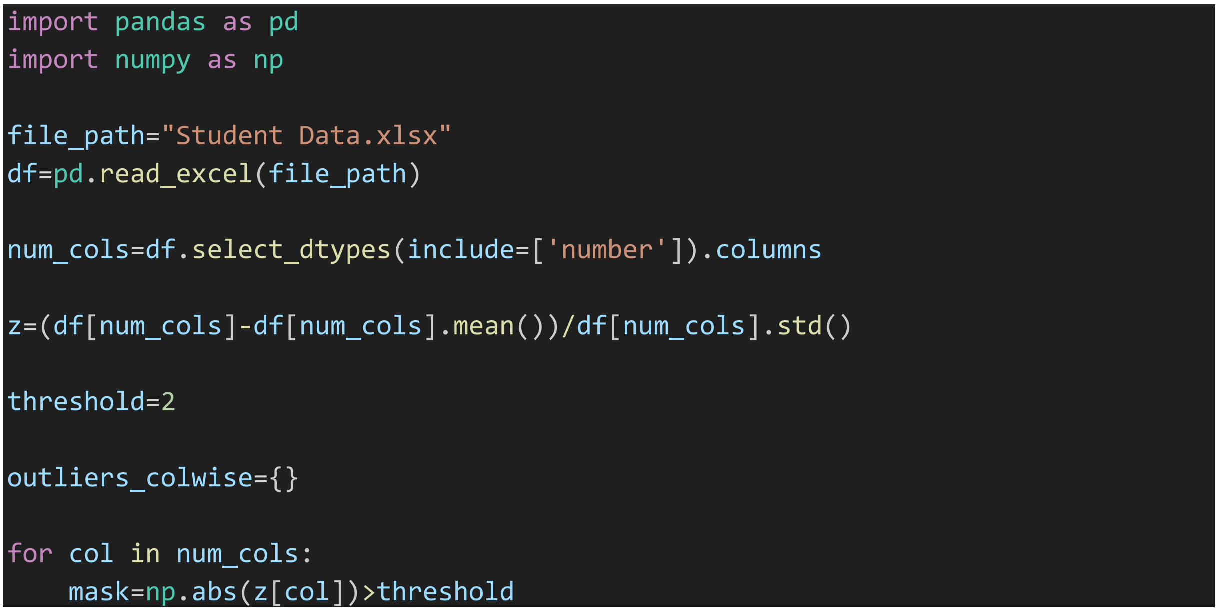

Age Distribution
Frequency
175
150
125
100
75
50
25
0
15
16
17
18
19
20
21
22
Age


Sex Frequency
400
350
300
250
200
150
100
50
0
Count
F
M
Sex


Scatter plot: Age vs Medu
Medu (Mother's Education)
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
15
16
17
18
19
20
21
22
Age



EXP:9- Z-Score Outlier Detection



```
import pandas as pd
import numpy as np

file_path="Student Data.xlsx"
df=pd.read_excel(file_path)

num_cols=df.select_dtypes(include=['number']).columns

z=(df[num_cols]-df[num_cols].mean())/df[num_cols].std()

threshold=2

outliers_colwise={}

for col in num_cols:
    mask=np.abs(z[col])>threshold
    outliers_colwise[col]=df.loc[mask,[col]]

for col,vals in outliers_colwise.items():
    print(f"\n{col}:{len(vals)} outliers")
    if len(vals)>0:
        print(vals.to_string(index=False))
    else:
        print("No outliers detected.")
```

```
age:9 outliers
 age
22.0
20.0
20.0
21.0
21.0
20.0
20.0
20.0
20.0

Medu:6 outliers
 Medu
   0
   0
   0
   0
   0
   0
```

Fedu:7 outliers

```
 Fedu
   0
   0
   0
   0
   0
   0
   0
```

traveltime:16 outliers

```
 traveltime
         4
         4
         4
         4
         4
         4
         4
         4
         4
         4
         4
         4
```

4

4

4

4

studytime:35 outliers

studytime

4

4

4

4

4

4

4

4

4

4

4

4

4

4

4

4

4

4

4

4

4

4

4

4

4

4

4

4

4

4

4

4

4

4

4

failures:30 outliers

failures

3

3

2

3

2

2

3

3

2

2

3

3

3

2

2

2

2

2

3

2

3

2

3

3

2

3

2

2

2

3

famrel:51 outliers

famrel
1
2
2
2
2
1
2
1
2
1
2
2
2
2
1
2
1
2
2
2
1
2

2

1

1

2

1

1

1

1

1

2

1

2

1

2

2

2

1

1

2

2

2

1

2

2

1

```
      1
      2
      1
      2
```

freetime:45 outliers

```
 freetime
        1
        1
        1
        1
        1
        1
        1
        1
        1
        1
        1
        1
        1
        1
        1
        1
        1
        1
```

1

1

1

1

1

1

1

1

1

1

1

1

1

1

1

1

1

1

1

1

1

1

1

1

1

```
    1
    1

goout:0 outliers
No outliers detected.

Dalc:34 outliers
 Dalc
  5
  4
  5
  5
  5
  4
  5
  4
  4
  4
  4
  5
  4
  5
  5
  4
  4
```

4

5

5

4

4

4

5

5

4

4

5

5

5

5

5

4

4

Walc:45 outliers

Walc

5

5

5

5

5

5

5

5

5

5

5

5

5

5

5

5

5

5

5

5

5

5

5

5

5

5

5

5

5

5

```
  5
  5
  5
  5
  5
  5
  5
  5
  5
  5
  5
  5
  5
  5
  5

health:0 outliers
No outliers detected.

absences:32 outliers
 absences
     16
     14
     14
     16
```

14

24

22

16

14

32

16

16

30

21

14

15

16

18

16

14

26

14

16

16

15

22

18

14

18

16

21

13

G1:32 outliers

G1

0

17

17

17

18

17

17

17

17

18

17

17

18

18

18

17

17

17

5

17

4

4

5

18

17

17

5

5

18

19

17

5

G2:25 outliers

G2

18

18

18

18

19

18

18

18

18

18

0

5

18

0

0

5

18

18

0

0

0

18

0

5

18

G3:19 outliers

G3

0

1

5

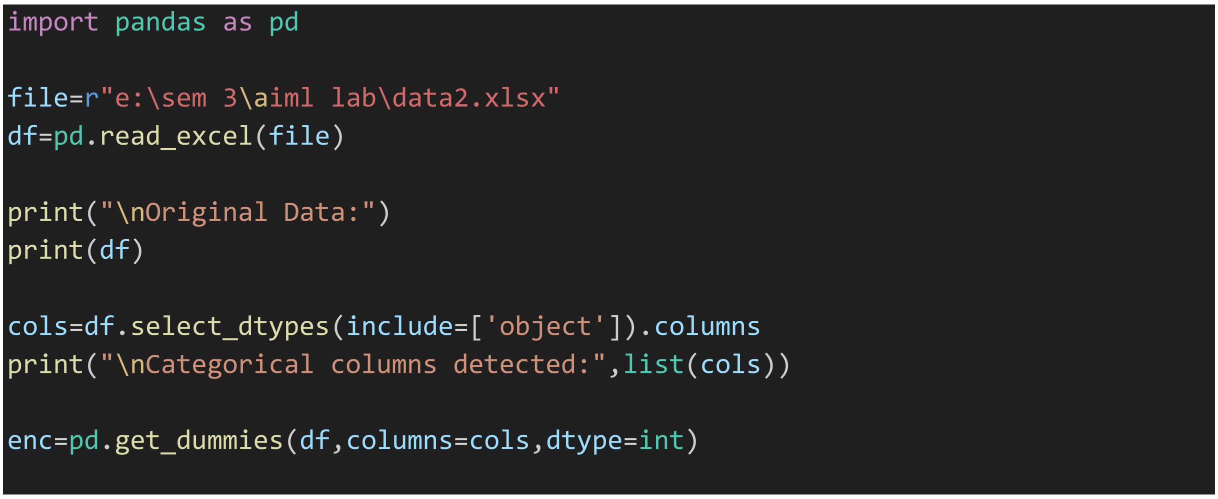
19

0

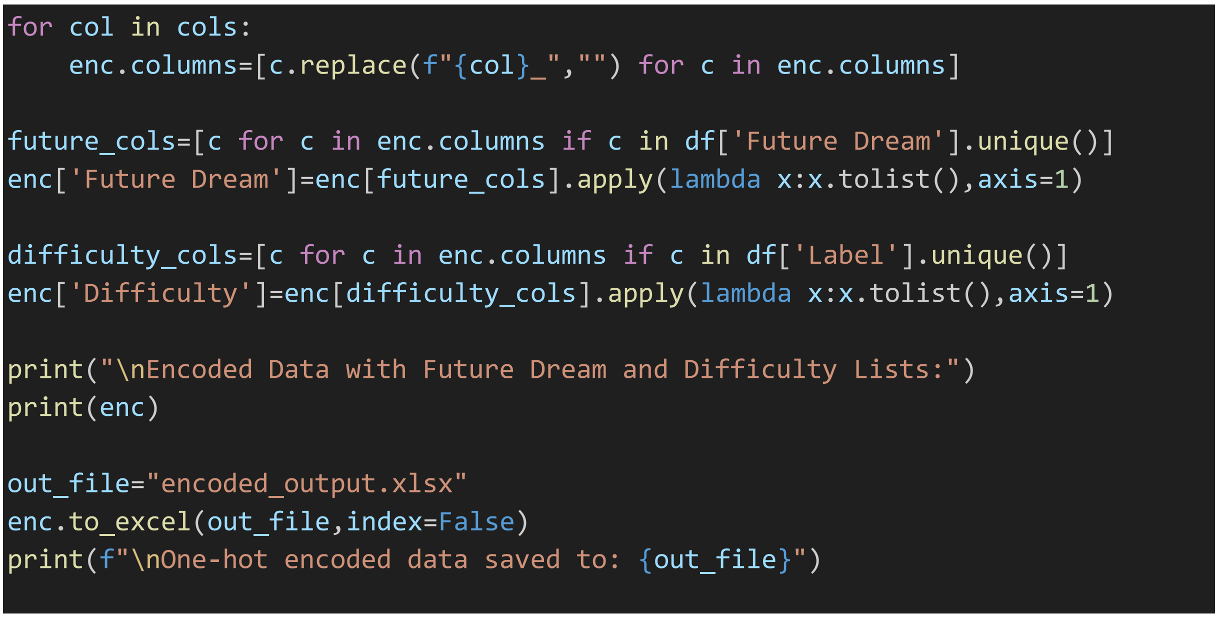
0

```
0
0
0
0
0
0
0
0
0
19
0
0
0
```

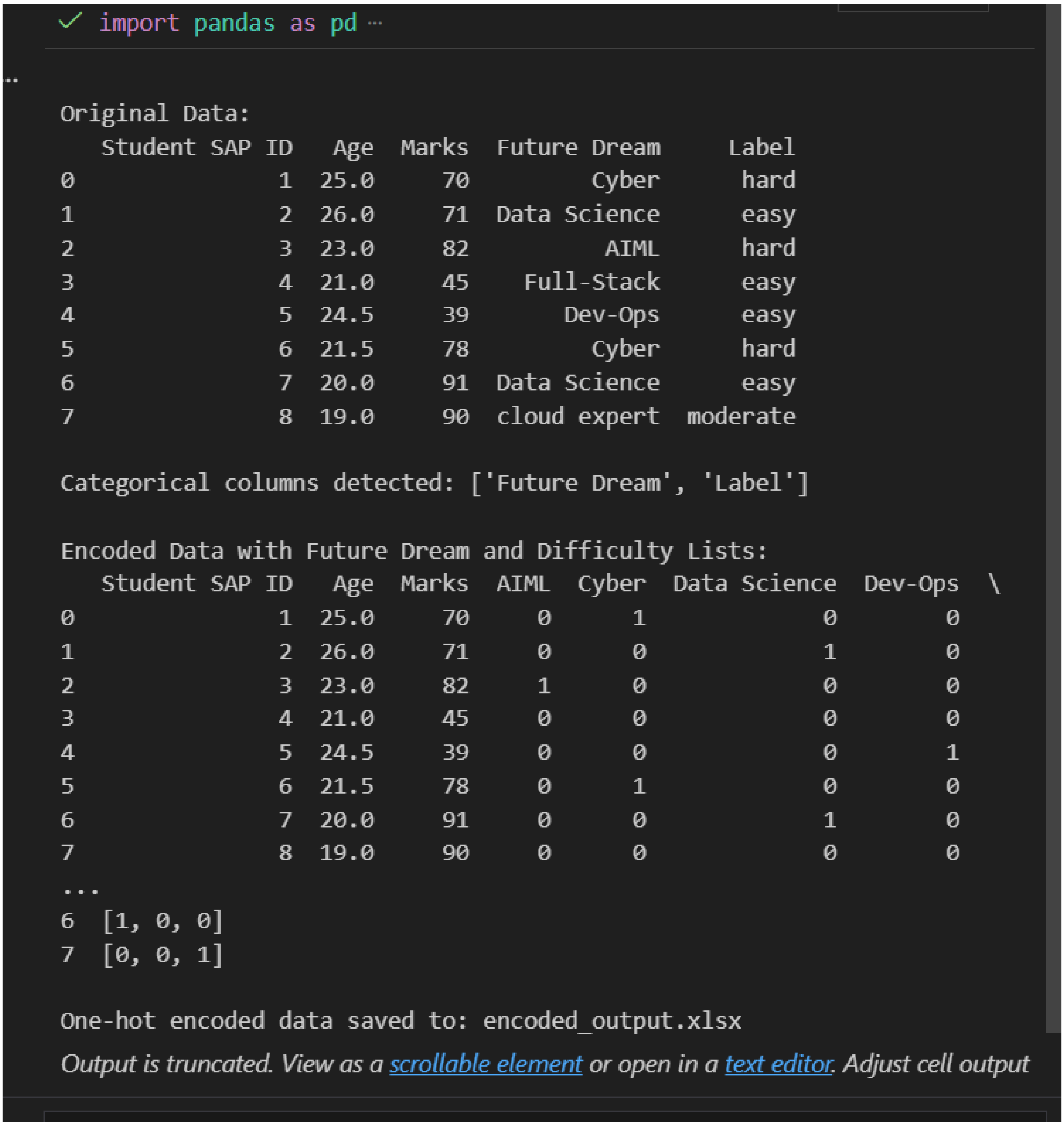
EXP:10- One-Hot Encoding & List Merge



```python
import pandas as pd

file=r"e:\sem 3\aiml lab\data2.xlsx"
df=pd.read_excel(file)

print("\nOriginal Data:")
print(df)

cols=df.select_dtypes(include=['object']).columns
print("\nCategorical columns detected:",list(cols))

enc=pd.get_dummies(df,columns=cols,dtype=int)
```



```
for col in cols:
    enc.columns=[c.replace(f"{col}_","") for c in enc.columns]

future_cols=[c for c in enc.columns if c in df['Future Dream'].unique()]
enc['Future Dream']=enc[future_cols].apply(lambda x:x.tolist(),axis=1)

difficulty_cols=[c for c in enc.columns if c in df['Label'].unique()]
enc['Difficulty']=enc[difficulty_cols].apply(lambda x:x.tolist(),axis=1)

print("\nEncoded Data with Future Dream and Difficulty Lists:")
print(enc)

out_file="encoded_output.xlsx"
enc.to_excel(out_file,index=False)
print(f"\nOne-hot encoded data saved to: {out_file}")
```

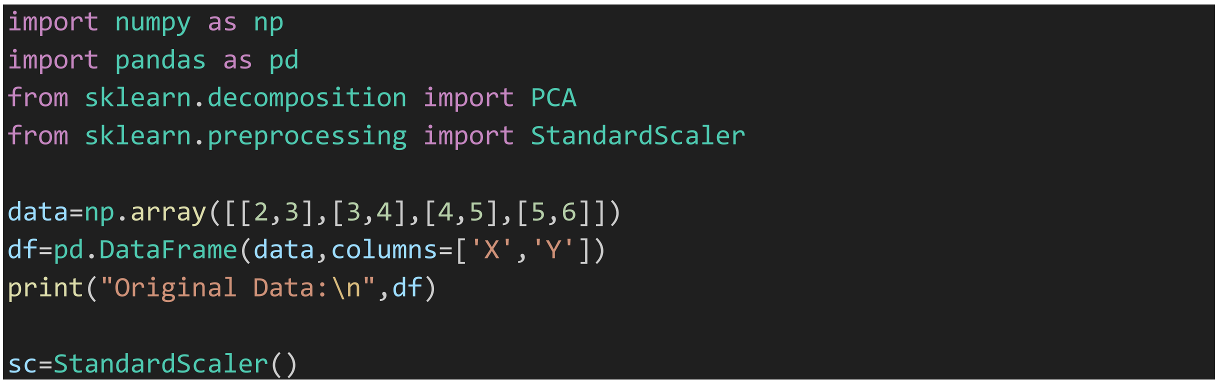


```
import pandas as pd ···
```

```
Original Data:
   Student SAP ID   Age  Marks  Future Dream     Label
0               1  25.0     70         Cyber      hard
1               2  26.0     71  Data Science      easy
2               3  23.0     82          AIML      hard
3               4  21.0     45    Full-Stack      easy
4               5  24.5     39       Dev-Ops      easy
5               6  21.5     78         Cyber      hard
6               7  20.0     91  Data Science      easy
7               8  19.0     90  cloud expert  moderate

Categorical columns detected: ['Future Dream', 'Label']

Encoded Data with Future Dream and Difficulty Lists:
   Student SAP ID   Age  Marks  AIML  Cyber  Data Science  Dev-Ops  \
0               1  25.0     70     0      1             0        0
1               2  26.0     71     0      0             1        0
2               3  23.0     82     1      0             0        0
3               4  21.0     45     0      0             0        0
4               5  24.5     39     0      0             0        1
5               6  21.5     78     0      1             0        0
6               7  20.0     91     0      0             1        0
7               8  19.0     90     0      0             0        0
...
6  [1, 0, 0]
7  [0, 0, 1]

One-hot encoded data saved to: encoded_output.xlsx
```

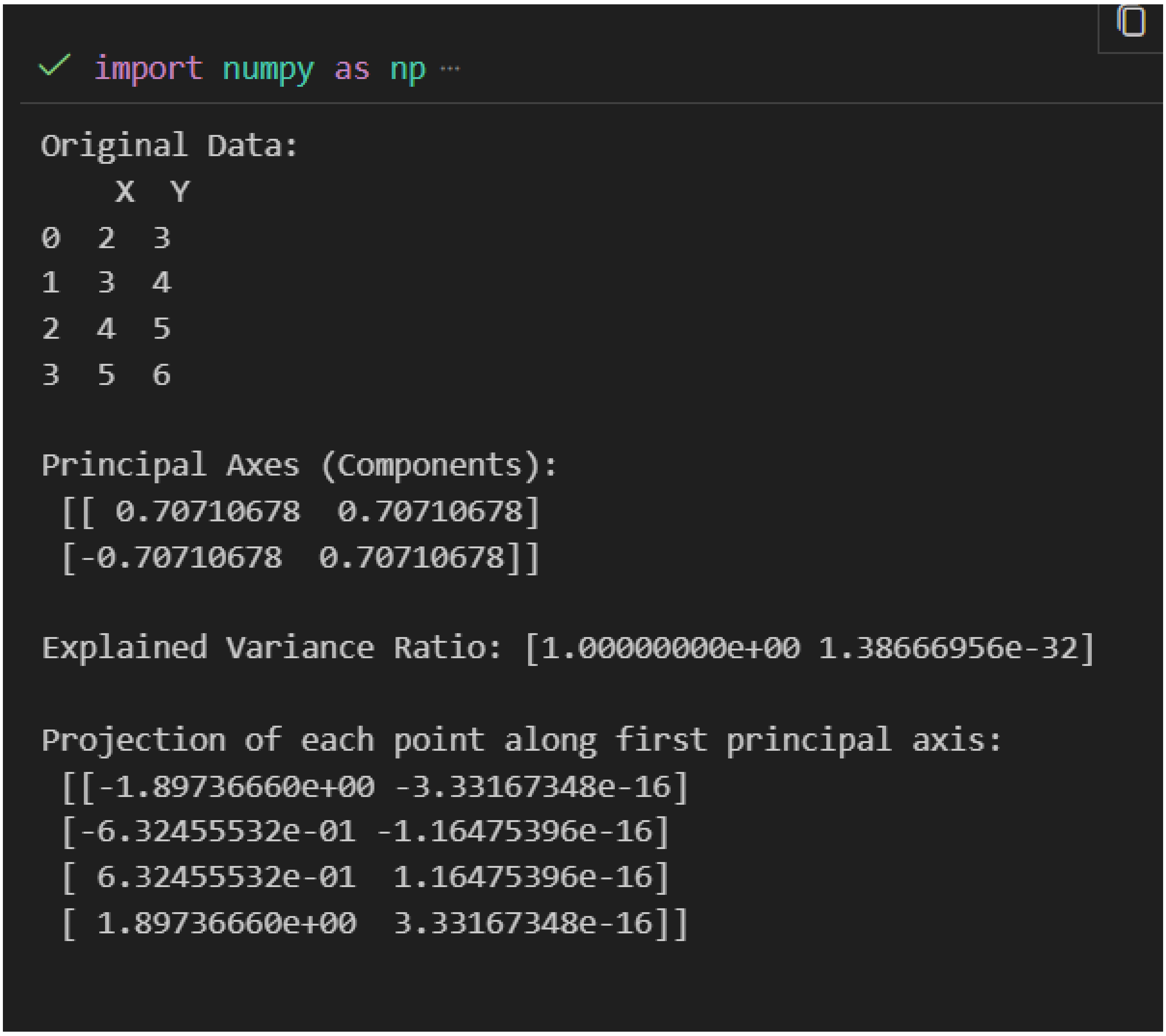
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output

EXP:11.1- PCA Using Sklearn



```
import numpy as np
import pandas as pd
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

data=np.array([[2,3],[3,4],[4,5],[5,6]])
df=pd.DataFrame(data,columns=['X','Y'])
print("Original Data:\n",df)

sc=StandardScaler()
```

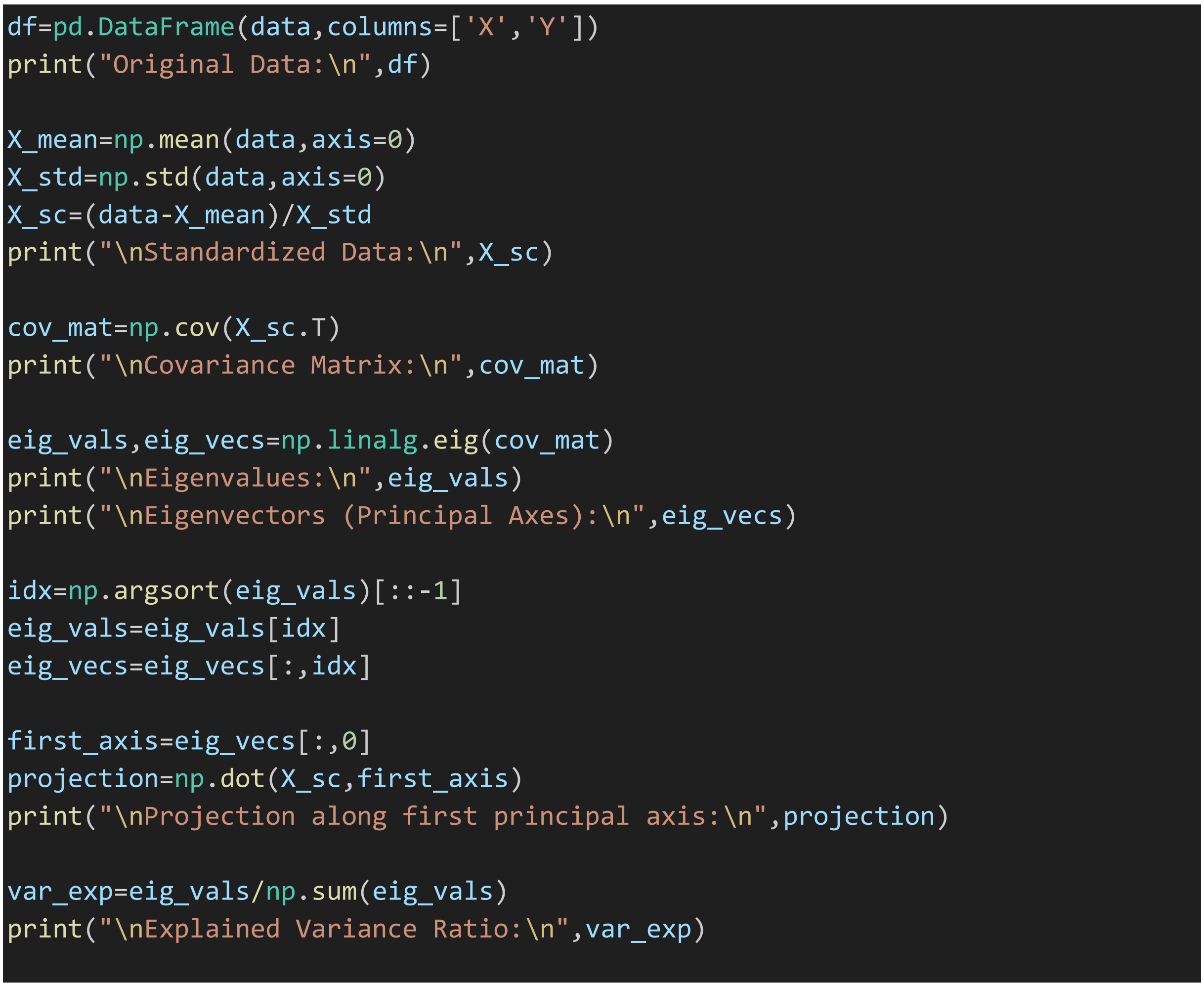
```
X_sc=sc.fit_transform(df)

pca=PCA(n_components=2)
pca.fit(X_sc)
X_pca=pca.transform(X_sc)

print("\nPrincipal Axes (Components):\n",pca.components_)
print("\nExplained Variance Ratio:", pca.explained_variance_ratio_)
print("\nProjection of each point along first principal axis:\n", X_pca)
```



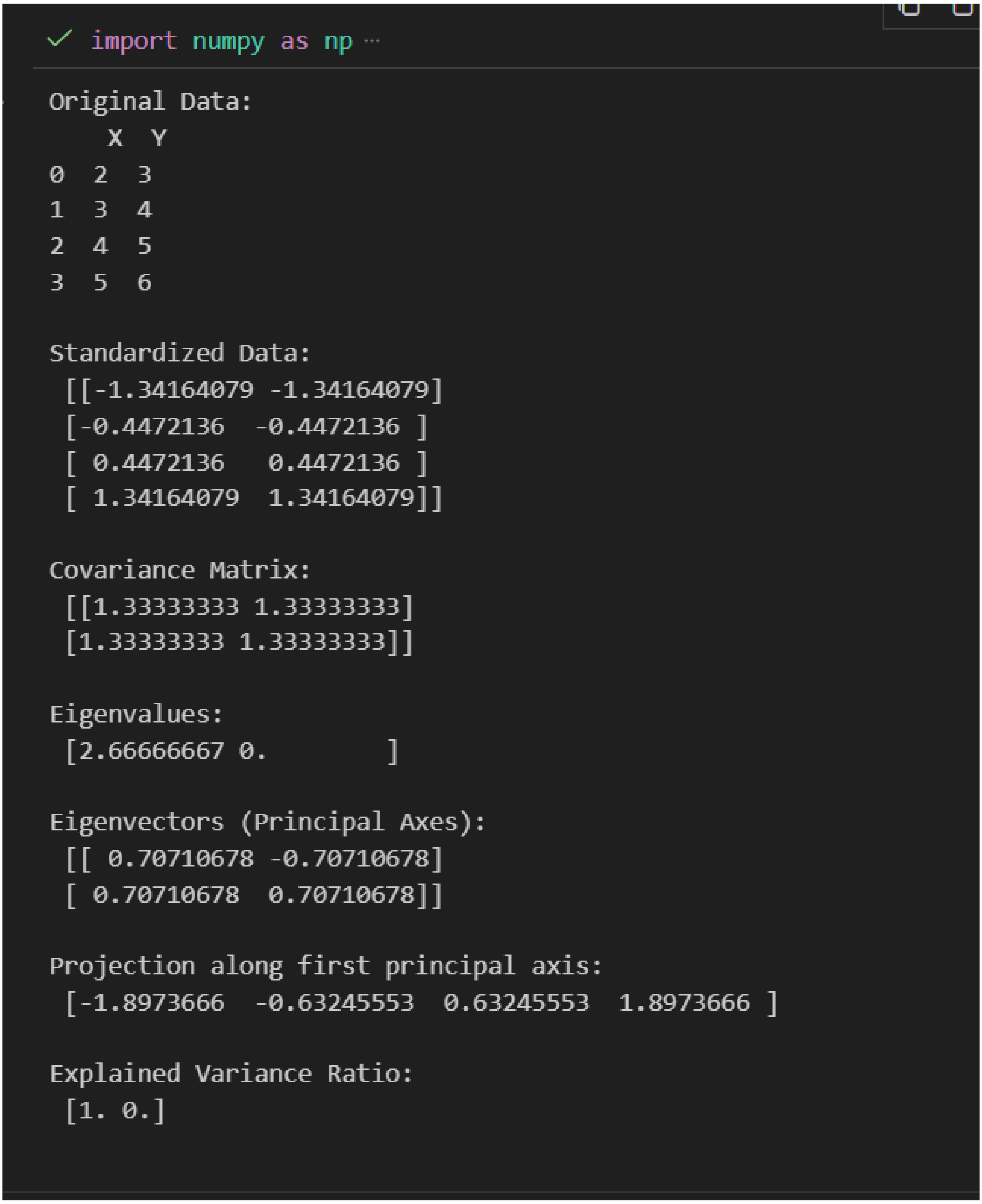
```
✓ import numpy as np ···

Original Data:
    X  Y
0  2  3
1  3  4
2  4  5
3  5  6

Principal Axes (Components):
 [[ 0.70710678  0.70710678]
 [-0.70710678  0.70710678]]

Explained Variance Ratio: [1.00000000e+00 1.38666956e-32]

Projection of each point along first principal axis:
 [[-1.89736660e+00 -3.33167348e-16]
 [-6.32455532e-01 -1.16475396e-16]
 [ 6.32455532e-01  1.16475396e-16]
 [ 1.89736660e+00  3.33167348e-16]]
```

EXP:11.2- PCA Without Sklearn

```
import numpy as np
import pandas as pd

data=np.array([[2,3],[3,4],[4,5],[5,6]])
```



```python
df=pd.DataFrame(data,columns=['X','Y'])
print("Original Data:\n",df)

X_mean=np.mean(data,axis=0)
X_std=np.std(data,axis=0)
X_sc=(data-X_mean)/X_std
print("\nStandardized Data:\n",X_sc)

cov_mat=np.cov(X_sc.T)
print("\nCovariance Matrix:\n",cov_mat)

eig_vals,eig_vecs=np.linalg.eig(cov_mat)
print("\nEigenvalues:\n",eig_vals)
print("\nEigenvectors (Principal Axes):\n",eig_vecs)

idx=np.argsort(eig_vals)[::-1]
eig_vals=eig_vals[idx]
eig_vecs=eig_vecs[:,idx]

first_axis=eig_vecs[:,0]
projection=np.dot(X_sc,first_axis)
print("\nProjection along first principal axis:\n",projection)

var_exp=eig_vals/np.sum(eig_vals)
print("\nExplained Variance Ratio:\n",var_exp)
```



```
import numpy as np ···
```

```
Original Data:
    X  Y
0  2  3
1  3  4
2  4  5
3  5  6

Standardized Data:
 [[-1.34164079 -1.34164079]
 [-0.4472136  -0.4472136 ]
 [ 0.4472136   0.4472136 ]
 [ 1.34164079  1.34164079]]

Covariance Matrix:
 [[1.33333333 1.33333333]
 [1.33333333 1.33333333]]

Eigenvalues:
 [2.66666667 0.        ]

Eigenvectors (Principal Axes):
 [[ 0.70710678 -0.70710678]
 [ 0.70710678  0.70710678]]

Projection along first principal axis:
 [-1.8973666  -0.63245553  0.63245553  1.8973666 ]

Explained Variance Ratio:
 [1. 0.]
```

EXP:12- Logistic Regression on Credit Fraud

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score
```

```python
f=r"e:\sem 3\aiml lab\creditcard.csv"
df=pd.read_csv(f)
print("Data loaded:",df.shape)

n=df['Class'].nunique()
print("No. of classes:",n)

c=df['Class'].value_counts()
print("Samples per class:\n",c)

if c.nunique()!=1:
    print("Unbalanced...balancing")
    maj=df[df['Class']==c.idxmax()]
    mino=df[df['Class']==c.idxmin()]
    mino_up=mino.sample(len(maj),replace=True,random_state=0)
    dfb=pd.concat([maj,mino_up],axis=0).sample(frac=1,random_state=0)
    print("Balanced counts:\n",dfb['Class'].value_counts())
else:
    print("Already balanced")

dfb.to_csv("balanced_creditcard.csv",index=False)
print("Saved: balanced_creditcard.csv")

x=dfb.drop('Class',axis=1)
y=dfb['Class']
xtr,xts,ytr,yts=train_test_split(x,y,test_size=0.3,random_state=0)

ep=10
r=[]
for i in range(ep):
    it=(i+1)*50
    m=LogisticRegression(max_iter=it,solver='lbfgs')
    m.fit(xtr,ytr)
    yp=m.predict(xts)
    acc=accuracy_score(yts,yp)
    pre=precision_score(yts,yp)
    rec=recall_score(yts,yp)
    f1=f1_score(yts,yp)
    r.append([i+1,it,acc,pre,rec,f1])

res=pd.DataFrame(r,columns=['Epoch','Iter','Accuracy','Precision','Recall','F1'])
print("\nResults:\n",res)

best=res.loc[res['F1'].idxmax()]
print(f"\nBest Epoch: {int(best['Epoch'])} | Acc={best['Accuracy']:.4f} | Pre={best['Precision']:.4f} | Rec={best['Recall']:.4f} | F1={best['F1']:.4f}")
```

[redacted]

```
Data loaded: (284807, 31)
No. of classes: 2
Samples per class:
 Class
0    284315
1       492
Name: count, dtype: int64
Unbalanced...balancing
Balanced counts:
 Class
1    284315
0    284315
Name: count, dtype: int64
Saved: balanced_creditcard.csv
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

```
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.


Results:
   Epoch  Iter  Accuracy  Precision    Recall       F1
0      1    50  0.917855   0.951327  0.880333  0.914454
1      2   100  0.945647   0.964577  0.924986  0.944367
2      3   150  0.944944   0.963161  0.924986  0.943688
3      4   200  0.944920   0.967000  0.920990  0.943434
4      5   250  0.949293   0.969933  0.927067  0.948016
5      6   300  0.949282   0.969909  0.927067  0.948004
6      7   350  0.948924   0.971081  0.925139  0.947553
7      8   400  0.948953   0.971141  0.925139  0.947582
8      9   450  0.948930   0.971093  0.925139  0.947559
9     10   500  0.948889   0.971009  0.925139  0.947519

Best Epoch: 5 | Acc=0.9493 | Pre=0.9699 | Rec=0.9271 | F1=0.9480
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```