PRACTICAL LAB BOOK

MCA 2025-2026 SEM 1

SUBJECT: AI/ML

Submitted By:

Name : Amit Pawar

SRN : 31242510

Course : Master of Computer Application

Div : A

Department: Commerce and Management

Guided By

Dr. Supriya Bhosale

Signature

Index Table

Sr. No.	Problem Statement
1	Python_Diwali_Sales_Analysis – EDA
2	Training–Testing Data Split
3	Linear Regression Algorithm
4	Simple SVM Classification with Visualization
5	Python Implementation of the K-Nearest Neighbors (KNN) Algorithm
6	k-Means Clustering Algorithm
7	Hierarchical Clustering
8	PCA (Principal Component Analysis)
9	Confusion Matrix
10	Accuracy, Precision, Recall, F1 Score Calculations

Practical No 1 : Python_Diwali_Sales_Analysis – EDA

Code:

```
# import python libraries


import numpy as np
import pandas as pd
import matplotlib.pyplot as plt # visualizing data
%matplotlib inline
import seaborn as sns
# import csv file
df = pd.read_csv('Diwali Sales Data.csv', encoding= 'unicode_escape')
df.shape
```

```
(11251, 15)
```

```
df.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11251 entries, 0 to 11250
Data columns (total 15 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   User_ID           11251 non-null  int64  
 1   Cust_name         11251 non-null  object 
 2   Product_ID        11251 non-null  object 
 3   Gender            11251 non-null  object 
 4   Age Group         11251 non-null  object 
 5   Age               11251 non-null  int64  
 6   Marital_Status    11251 non-null  int64  
 7   State             11251 non-null  object 
 8   Zone              11251 non-null  object 
 9   Occupation        11251 non-null  object 
 10  Product_Category  11251 non-null  object 
 11  Orders            11251 non-null  int64  
 12  Amount            11239 non-null  float64
 13  Status            0 non-null      float64
 14  unnamed1          0 non-null      float64
dtypes: float64(3), int64(4), object(8)
memory usage: 1.3+ MB
```

```
#drop unrelated/blank columns
df.drop(['Status', 'unnamed1'], axis=1, inplace=True)
```

#check for null values

pd.isnull(df).sum()

```
User_ID             0
Cust_name           0
Product_ID          0
Gender              0
Age Group           0
Age                 0
Marital_Status      0
State               0
Zone                0
Occupation          0
Product_Category    0
Orders              0
Amount             12
dtype: int64
```

drop null values

df.dropna(inplace=True)

change data type

df['Amount'] = df['Amount'].astype('int')

df['Amount'].dtypes

```
dtype('int32')
```

df.columns

```
Index(['User_ID', 'Cust_name', 'Product_ID', 'Gender', 'Age Group', 'Age',
       'Marital_Status', 'State', 'Zone', 'Occupation', 'Product_Category',
       'Orders', 'Amount'],
      dtype='object')
```

#rename column

df.rename(columns= {'Marital_Status':'Shaadi'})

	User_ID	Cust_name	Product_ID	Gender	Age Group	Age	Shaadi	State	Zone	Occupation	Product_Category	Orders	Amount
0	1002903	Sanskriti	P00125942	F	26-35	28	0	Maharashtra	Western	Healthcare	Auto	1	23952
1	1000732	Kartik	P00110942	F	26-35	35	1	Andhra Pradesh	Southern	Govt	Auto	3	23934
2	1001990	Bindu	P00118542	F	26-35	35	1	Uttar Pradesh	Central	Automobile	Auto	3	23924
3	1001425	Sudevi	P00237842	M	0-17	16	0	Karnataka	Southern	Construction	Auto	2	23912
4	1000588	Joni	P00057942	M	26-35	28	1	Gujarat	Western	Food Processing	Auto	2	23877
...	...	...	...	...	...	...	...	...	...	...	...	...	...
11246	1000695	Manning	P00296942	M	18-25	19	1	Maharashtra	Western	Chemical	Office	4	370
11247	1004089	Reichenbach	P00171342	M	26-35	33	0	Haryana	Northern	Healthcare	Veterinary	3	367
11248	1001209	Oshin	P00201342	F	36-45	40	0	Madhya Pradesh	Central	Textile	Office	4	213
11249	1004023	Noonan	P00059442	M	36-45	37	0	Karnataka	Southern	Agriculture	Office	3	206
11250	1002744	Brumley	P00281742	F	18-25	19	0	Maharashtra	Western	Healthcare	Office	3	188

11239 rows × 13 columns

describe() method returns description of the data in the DataFrame (i.e. count, mean, std, etc)

df.describe()

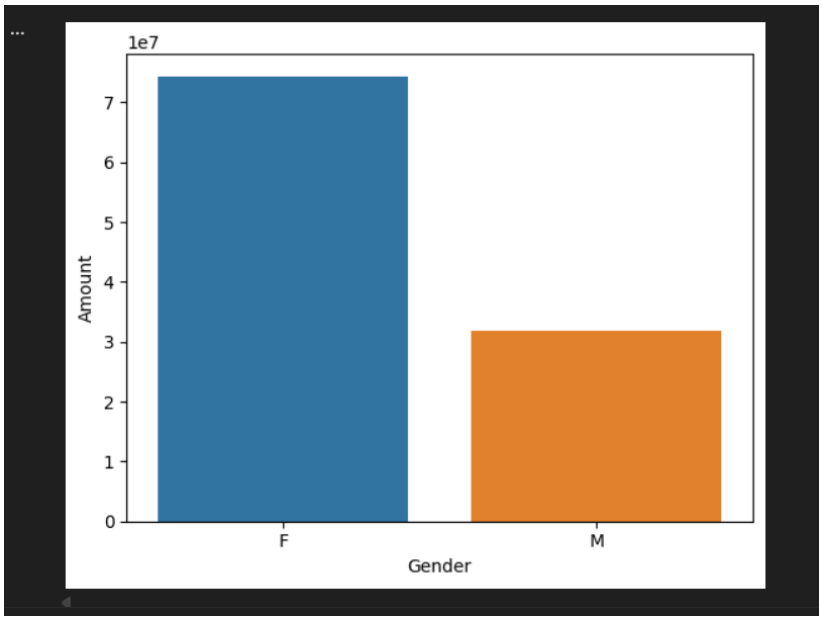
	User_ID	Age	Marital_Status	Orders	Amount
count	1.123900e+04	11239.000000	11239.000000	11239.000000	11239.000000
mean	1.003004e+06	35.410357	0.420055	2.489634	9453.610553
std	1.716039e+03	12.753866	0.493589	1.114967	5222.355168
min	1.000001e+06	12.000000	0.000000	1.000000	188.000000
25%	1.001492e+06	27.000000	0.000000	2.000000	5443.000000
50%	1.003064e+06	33.000000	0.000000	2.000000	8109.000000
75%	1.004426e+06	43.000000	1.000000	3.000000	12675.000000
max	1.006040e+06	92.000000	1.000000	4.000000	23952.000000

Exploratory Data Analysis

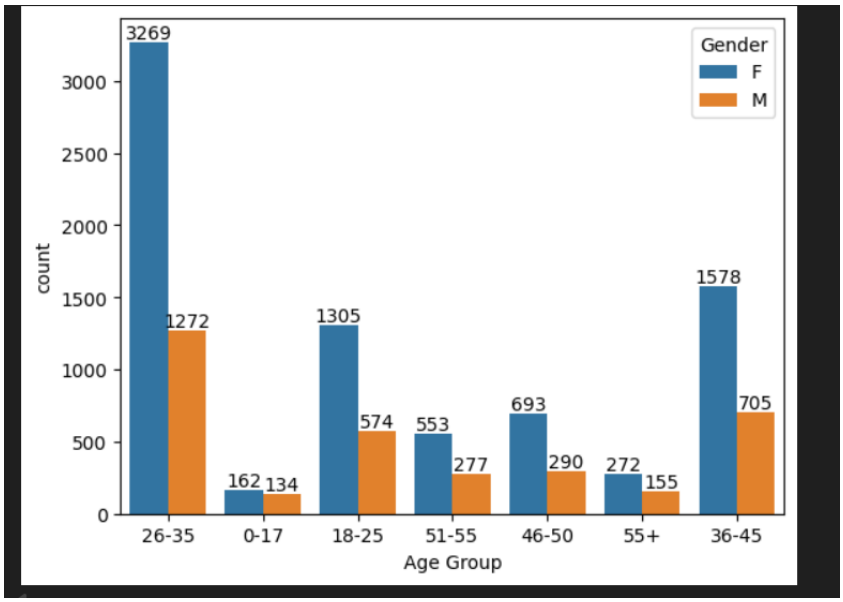
plotting a bar chart for gender vs total amount

```
sales_gen = df.groupby(['Gender'],
as_index=False)['Amount'].sum().sort_values(by='Amount', ascending=False)
```

```
sns.barplot(x = 'Gender',y= 'Amount' ,data = sales_gen)
```

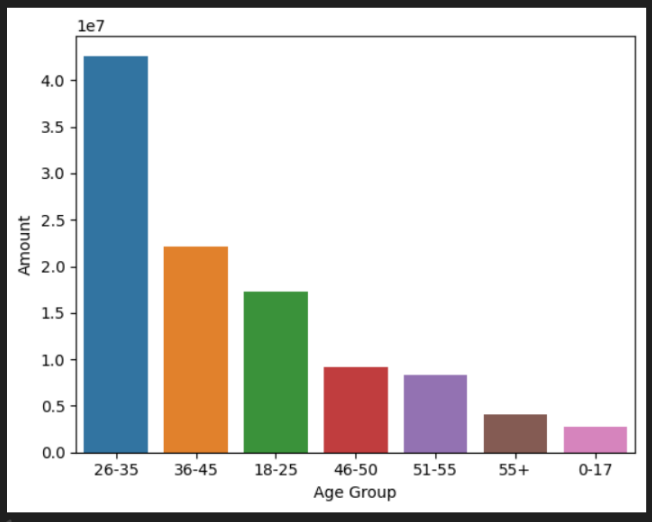


```
ax = sns.countplot(data = df, x = 'Age Group', hue = 'Gender')

for bars in ax.containers:
    ax.bar_label(bars)
```



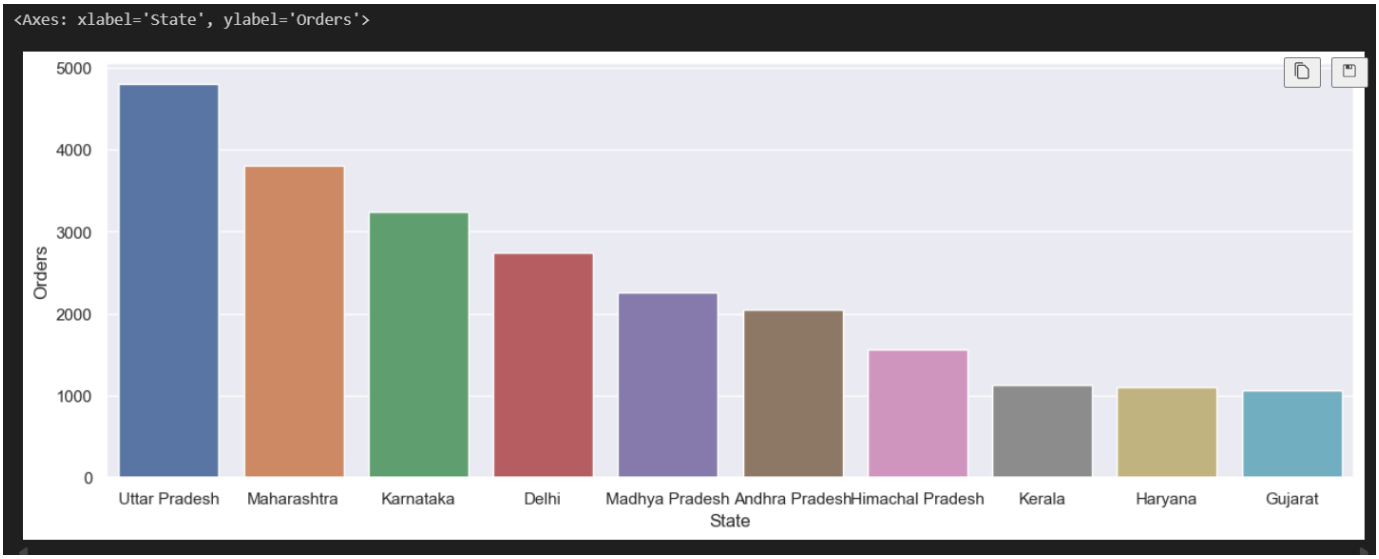

```
# Total Amount vs Age Group
sales_age = df.groupby(['Age Group'], as_index=False)['Amount'].sum().sort_values(by='Amount', ascending=False)

sns.barplot(x = 'Age Group',y= 'Amount' ,data = sales_age)
```

```
<Axes: xlabel='Age Group', ylabel='Amount'>
```



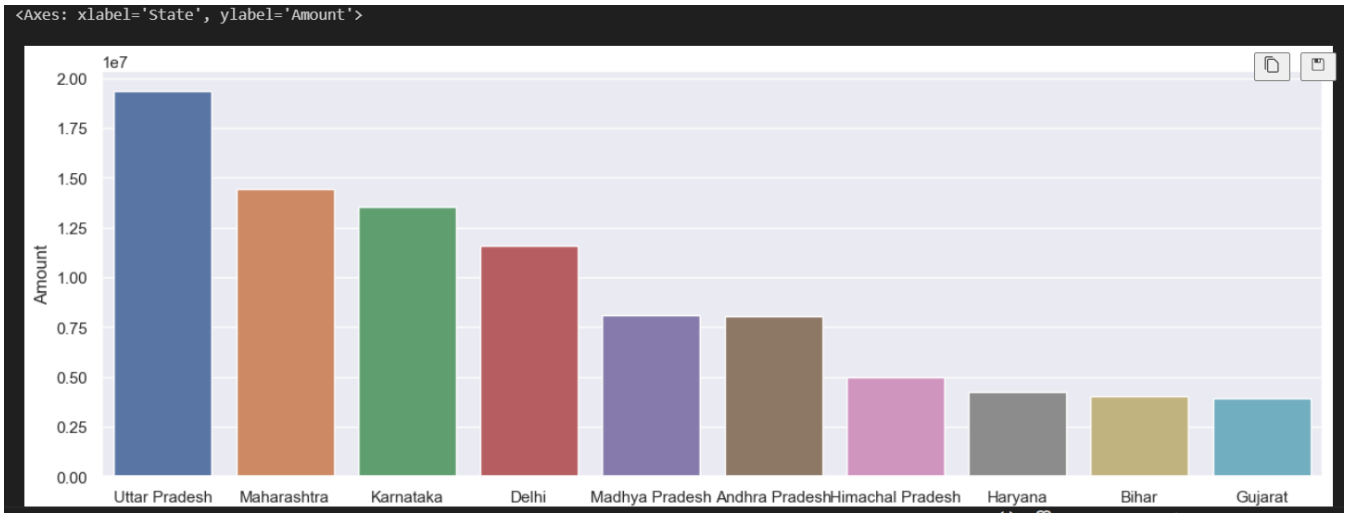

```
# total number of orders from top 10 states

sales_state = df.groupby(['State'], as_index=False)['Orders'].sum().sort_values(by='Orders', ascending=False).head(10)

sns.set(rc={'figure.figsize':(15,5)})
sns.barplot(data = sales_state, x = 'State',y= 'Orders')
```

```
<Axes: xlabel='State', ylabel='Orders'>
```

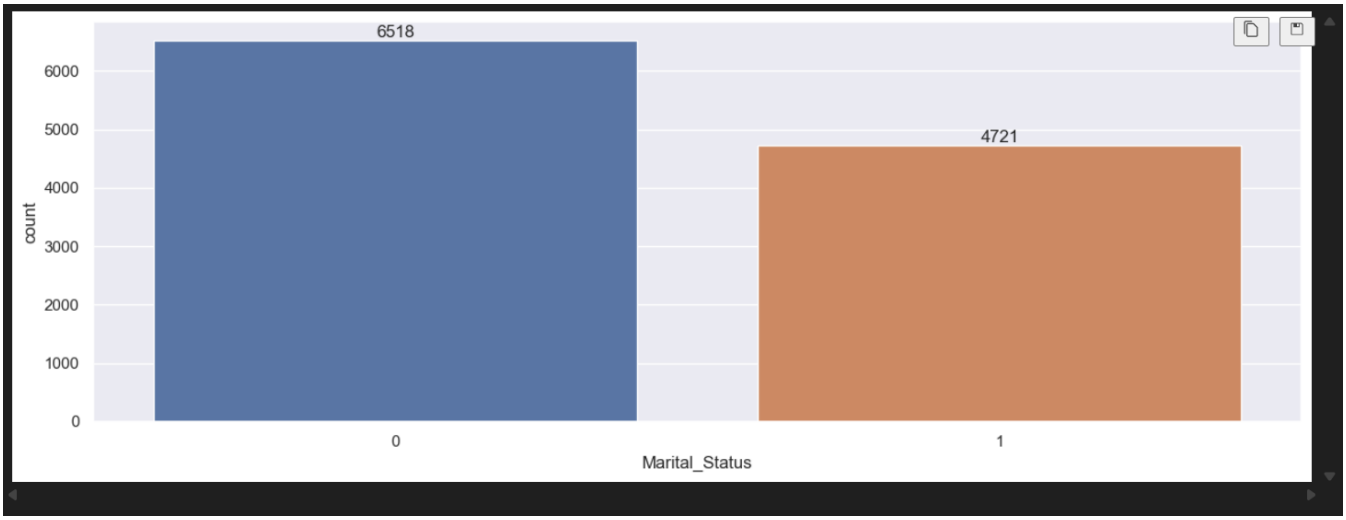



```
# total amount/sales from top 10 states

sales_state = df.groupby(['State'], as_index=False)['Amount'].sum().sort_values(by='Amount', ascending=False).head(10)

sns.set(rc={'figure.figsize':(15,5)})
sns.barplot(data = sales_state, x = 'State',y= 'Amount')
```

```
<Axes: xlabel='State', ylabel='Amount'>
```

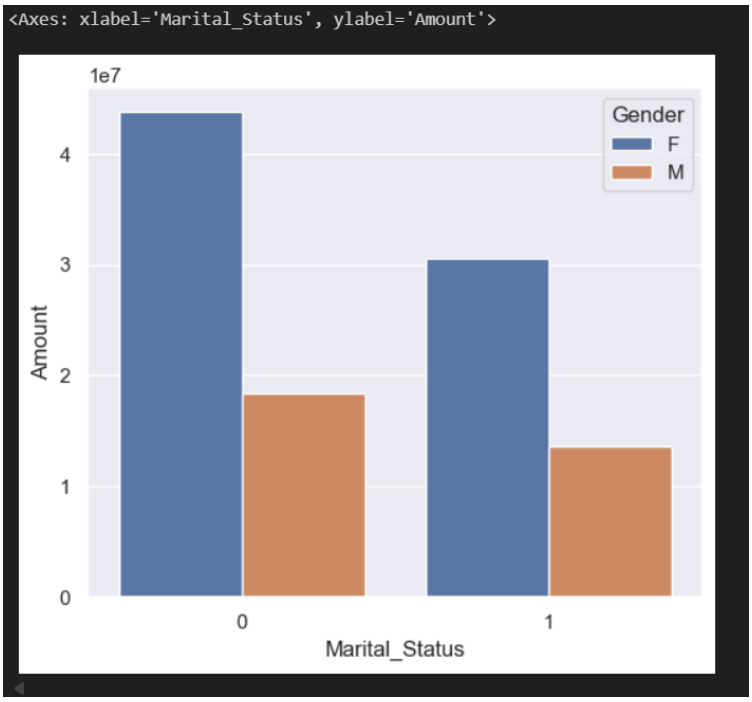



```
ax = sns.countplot(data = df, x = 'Marital_Status')

sns.set(rc={'figure.figsize':(7,5)})
for bars in ax.containers:
    ax.bar_label(bars)
```

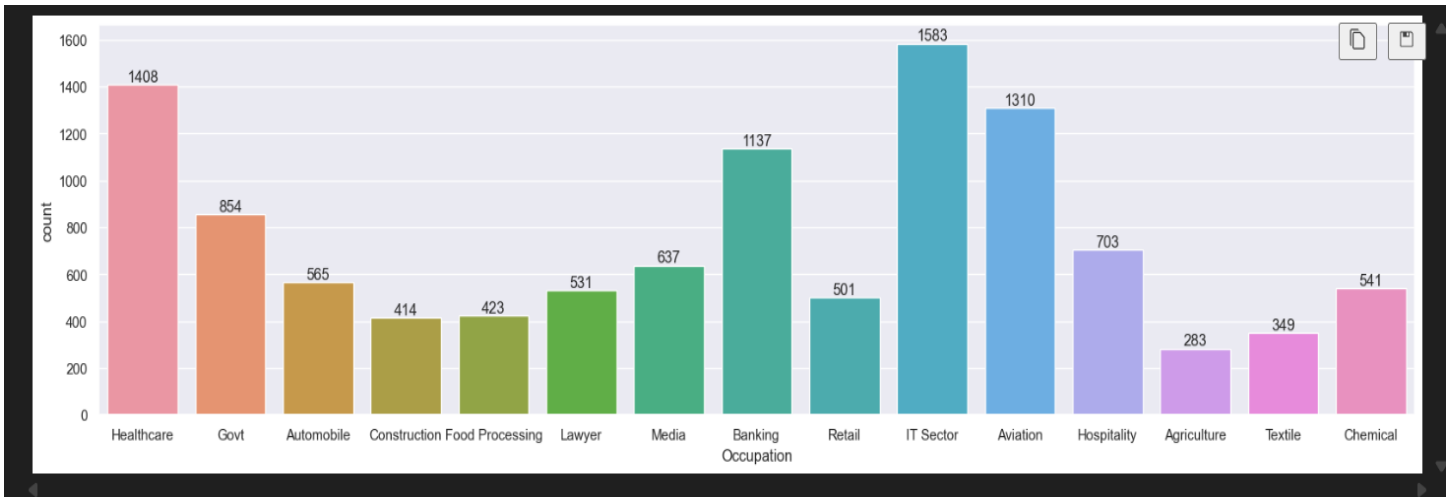



```
sales_state = df.groupby(['Marital_Status', 'Gender'],
as_index=False)['Amount'].sum().sort_values(by='Amount', ascending=False)

sns.set(rc={'figure.figsize':(6,5)})
sns.barplot(data = sales_state, x = 'Marital_Status',y= 'Amount', hue='Gender')
```

```
<Axes: xlabel='Marital_Status', ylabel='Amount'>
```

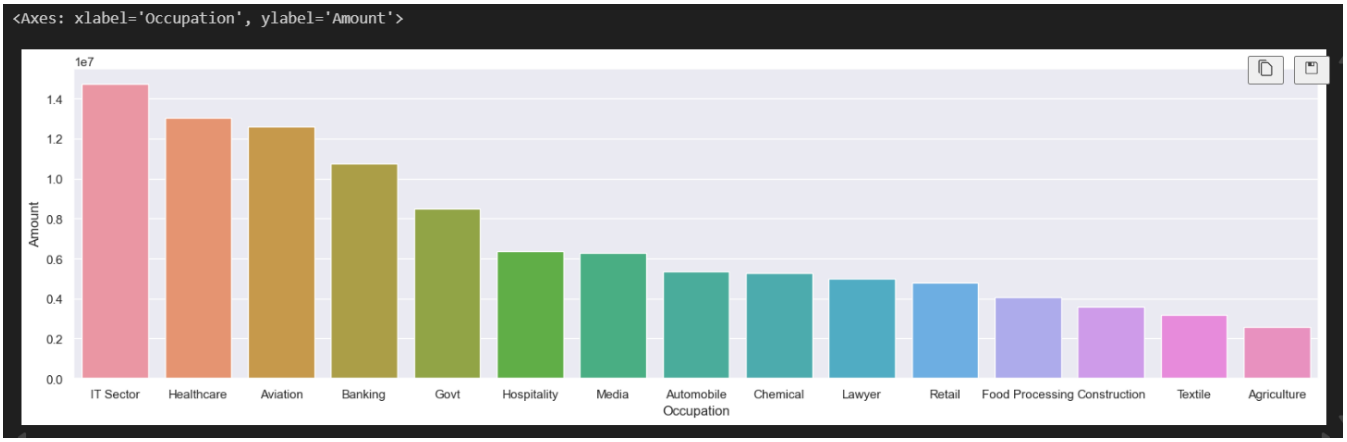



```
sns.set(rc={'figure.figsize':(20,5)})
ax = sns.countplot(data = df, x = 'Occupation')

for bars in ax.containers:
    ax.bar_label(bars)
```

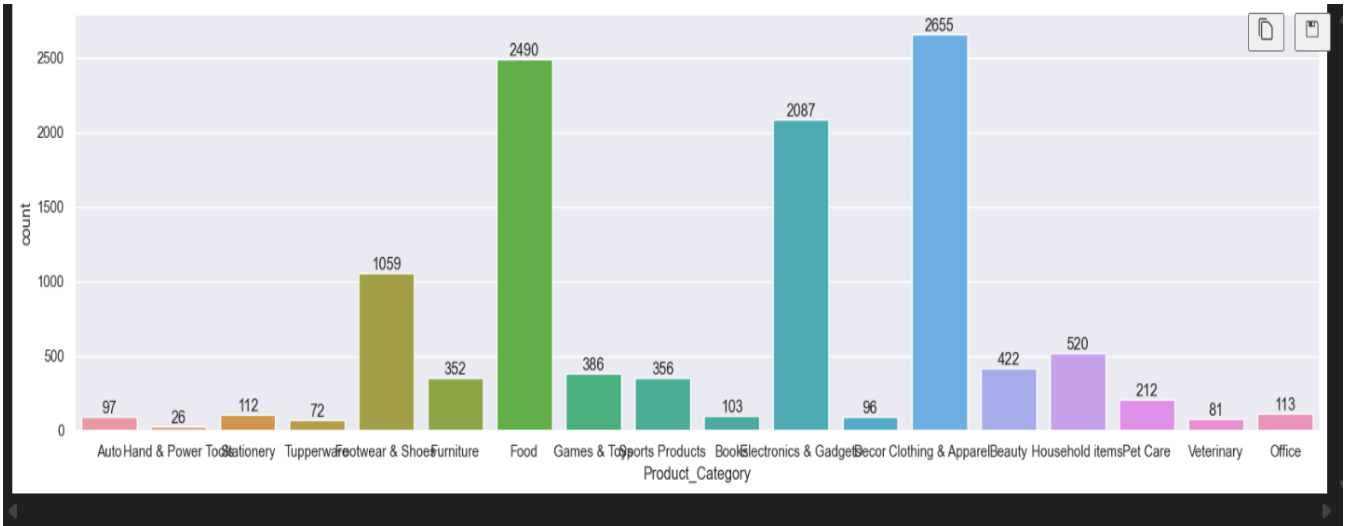



```
sales_state = df.groupby(['Occupation'], as_index=False)['Amount'].sum().sort_values(by='Amount', ascending=False)

sns.set(rc={'figure.figsize':(20,5)})
sns.barplot(data = sales_state, x = 'Occupation',y= 'Amount')
```

```
<Axes: xlabel='Occupation', ylabel='Amount'>
```

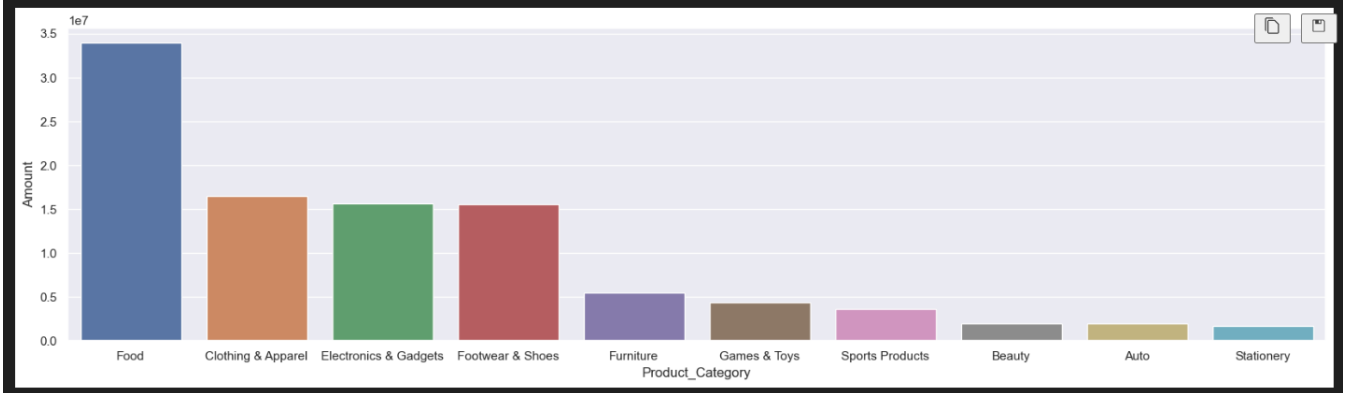



```
sns.set(rc={'figure.figsize':(20,5)})
ax = sns.countplot(data = df, x = 'Product_Category')

for bars in ax.containers:
    ax.bar_label(bars)
```



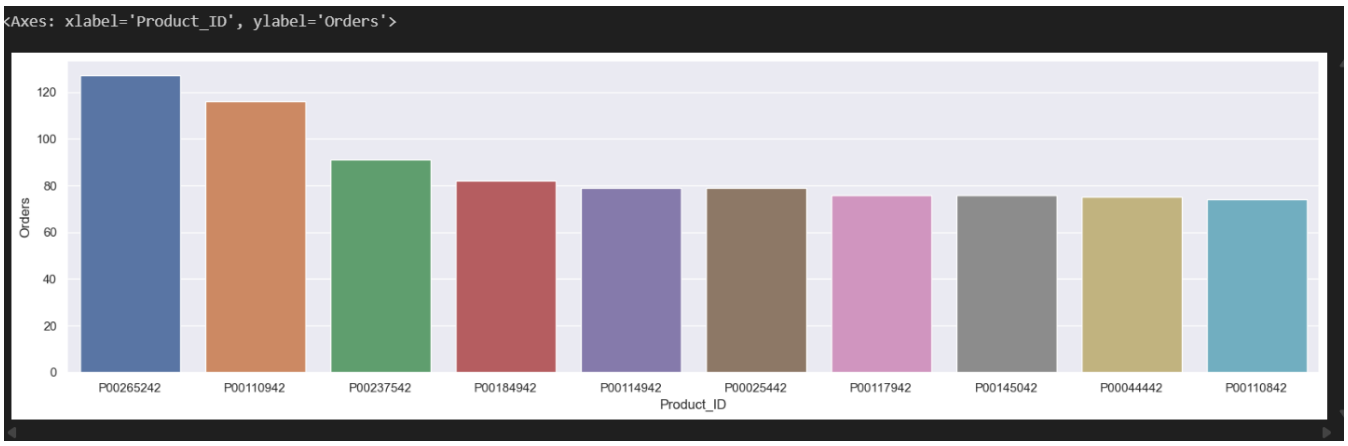
```
sales  state = df.groupby(['Product  Category'], as  index=False)['Amount'].sum().sort  values(by='Amount',
ascending=False).head(10)

sns.set(rc={'figure.figsize':(20,5)})
sns.barplot(data = sales_state, x = 'Product_Category',y= 'Amount')
```

```
<Axes: xlabel='Product_Category', ylabel='Amount'>
```

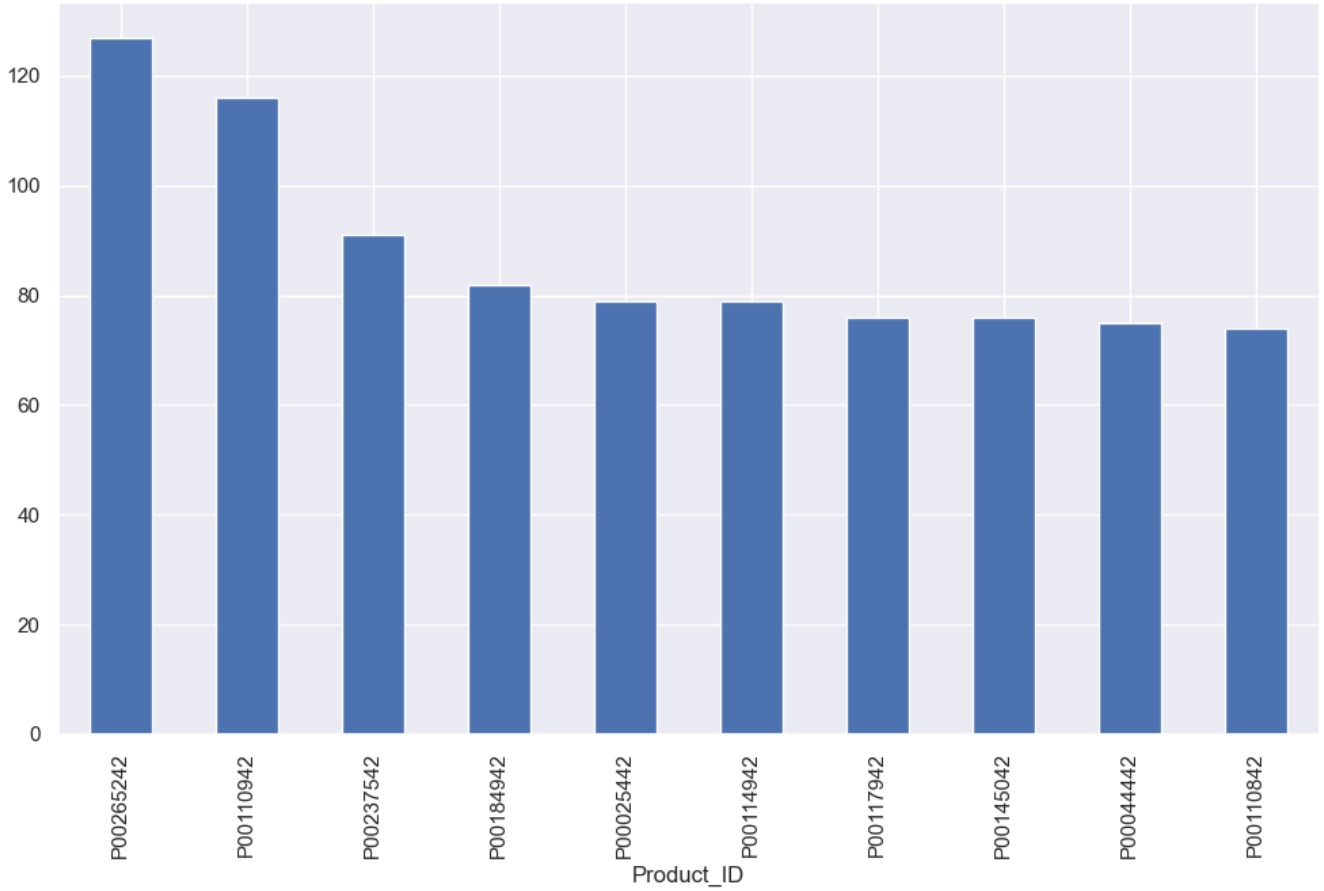



```
sales_state = df.groupby(['Product_ID'], as_index=False)['Orders'].sum().sort_values(by='Orders',
ascending=False).head(10)

sns.set(rc={'figure.figsize':(20,5)})
sns.barplot(data = sales_state, x = 'Product_ID',y= 'Orders')
```

```
<Axes: xlabel='Product_ID', ylabel='Orders'>
```




```
# top 10 most sold products (same thing as above)

fig1, ax1 = plt.subplots(figsize=(12,7))
df.groupby('Product_ID')['Orders'].sum().nlargest(10).sort_values(ascending=False).plot(kind='bar')
```


120
100
80
60
40
20
0
P00265242
P00110942
P00237542
P00184942
P00025442
P00114942
P00117942
P00145042
P00044442
P00110842
Product_ID

Practical No 2

Training and Testing Data Split

We have a dataset containing prices of used BMW cars. We are going to analyze this dataset and build a prediction function that can predict a price by taking mileage and age of the car as input. We will use sklearn train_test_split method to split training and testing dataset

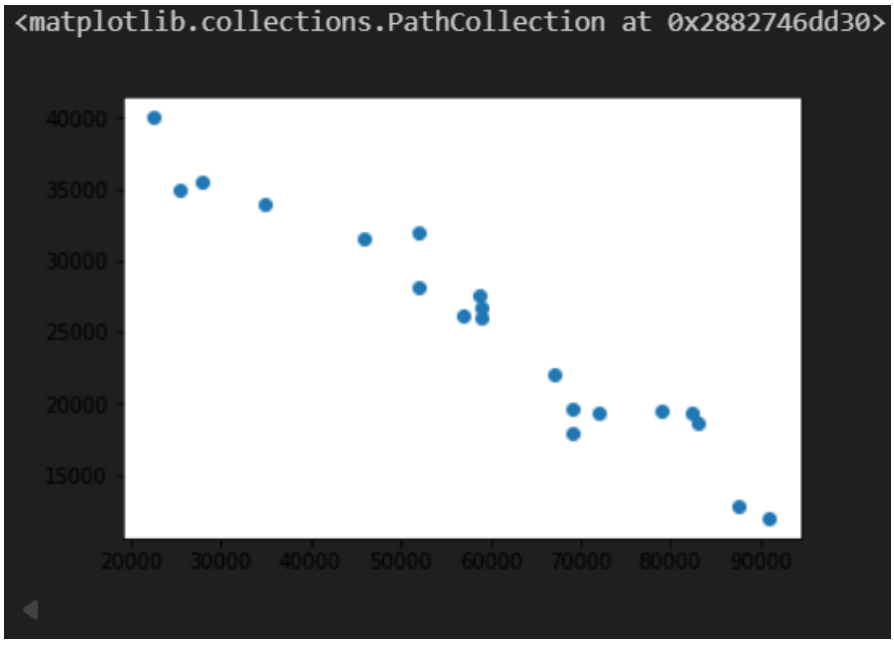
```
import pandas as pd
df = pd.read_csv("carprices.csv")
df.head()
```

	Mileage	Age(yrs)	Sell Price($)
0	69000	6	18000
1	35000	3	34000
2	57000	5	26100
3	22500	2	40000
4	46000	4	31500

```
import matplotlib.pyplot as plt
%matplotlib inline
```

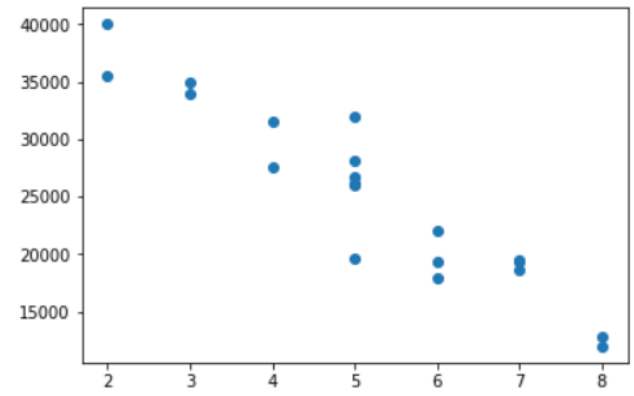
```
plt.scatter(df['Mileage'],df['Sell Price($)'])
```

```
<matplotlib.collections.PathCollection at 0x2882746dd30>
```



```
plt.scatter(df['Age(yrs)'],df['Sell Price($)'])
```

```
<matplotlib.collections.PathCollection at 0x28826e06240>
```




```
X = df[['Mileage','Age(yrs)']]
```

```
y = df['Sell Price($)']
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3)
```

```
X_train
```

	Mileage	Age(yrs)
11	79000	7
17	69000	5
10	83000	7
1	35000	3
0	69000	6
8	91000	8
7	72000	6
16	28000	2
6	52000	5
4	46000	4
19	52000	5
2	57000	5
5	59000	5
15	25400	3

X_test

	Mileage	Age(yrs)
3	22500	2
12	59000	5
14	82450	7
13	58780	4
9	67000	6
18	87600	8

y_train

```
11    19500
17    19700
10    18700
1     34000
0     18000
8     12000
7     19300
16    35500
6     32000
4     31500
19    28200
2     26100
5     26750
15    35000
Name: Sell Price($), dtype: int64
```

y_test

```
3     40000
12    26000
14    19400
13    27500
9     22000
18    12800
Name: Sell Price($), dtype: int64
```

```
from sklearn.linear_model import LinearRegression
clf = LinearRegression()
clf.fit(X_train, y_train)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

```
clf.predict(X_test)
```

```
array([ 38166.23426912,  25092.95646646,  16773.29470749,  24096.93956163,
        22602.44614295,  15559.98266172])
```

```
clf.score(X_test, y_test)
```

```
0.92713129118963111
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3,random_state=10)
X_test
```

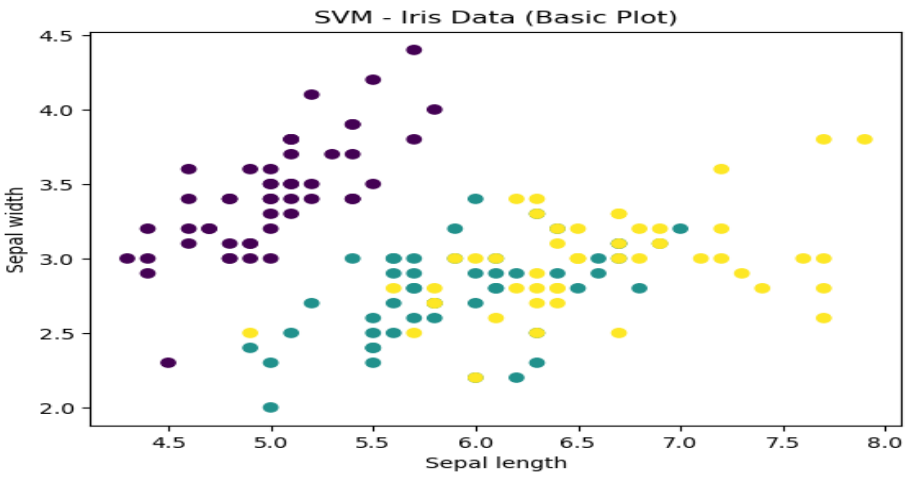
	Mileage	Age(yrs)
7	72000	6
10	83000	7
5	59000	5
6	52000	5
3	22500	2
18	87600	8

Practical No 3

Simple SVM Classification with Visualization

```
from sklearn import datasets
from sklearn.svm import SVC
import matplotlib.pyplot as plt

# Load dataset (first 2 features for easy plotting)
iris = datasets.load_iris()
X = iris.data[:, :2]
y = iris.target

# Train SVM
model = SVC(kernel='linear')
model.fit(X, y)

# Basic plot
plt.scatter(X[:, 0], X[:, 1], c=y)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.title('SVM - Iris Data (Basic Plot)')
plt.show()
```

Output:



Practical No 4

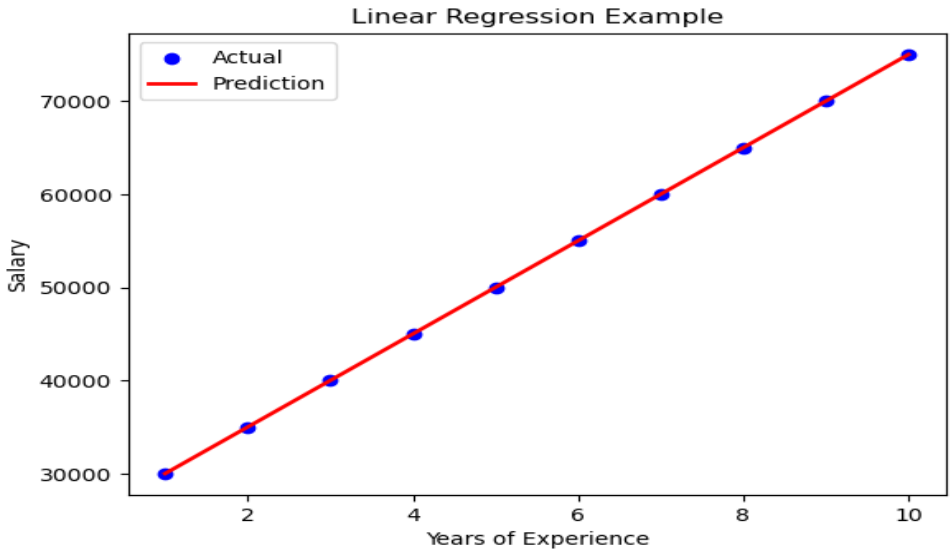
Linear Regression Algorithm:

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

(Using sample data for simplicity)
# Sample dataset
data = {
    'Experience': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
    'Salary': [30000, 35000, 40000, 45000, 50000, 55000, 60000, 65000, 70000, 75000]
}
df = pd.DataFrame(data)
print(df.head())

X = df[['Experience']]     # Independent variable
y = df['Salary']           # Dependent variable

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)

print("Mean Squared Error:", mean_squared_error(y_test, y_pred))
print("R^2 Score:", r2_score(y_test, y_pred))

plt.scatter(X, y, color='blue', label='Actual')
plt.plot(X, model.predict(X), color='red', linewidth=2, label='Prediction')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.title('Linear Regression Example')
plt.legend()
```

plt.show()

Output:

```
Experience  Salary
0           1   30000
1           2   35000
2           3   40000
3           4   45000
4           5   50000
Mean Squared Error: 0.0
R^2 Score: 1.0
```



Practical No 5

Python implementation of the K-Nearest Neighbors (KNN) algorithm

```
# Import required libraries

from sklearn.datasets import load_iris

from sklearn.neighbors import KNeighborsClassifier

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score


# Load the Iris dataset

iris = load_iris()

X = iris.data[:, :2]  # Only use first 2 features for simplicity

y = iris.target


# Split into training and testing data (70% train, 30% test)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)


# Create KNN classifier with k=3

knn = KNeighborsClassifier(n_neighbors=3)


# Train the model

knn.fit(X_train, y_train)


# Predict

y_pred = knn.predict(X_test)


# Accuracy

print("Accuracy:", accuracy_score(y_test, y_pred))

print("Predicted:", y_pred)

print("Actual:   ", y_test)
```

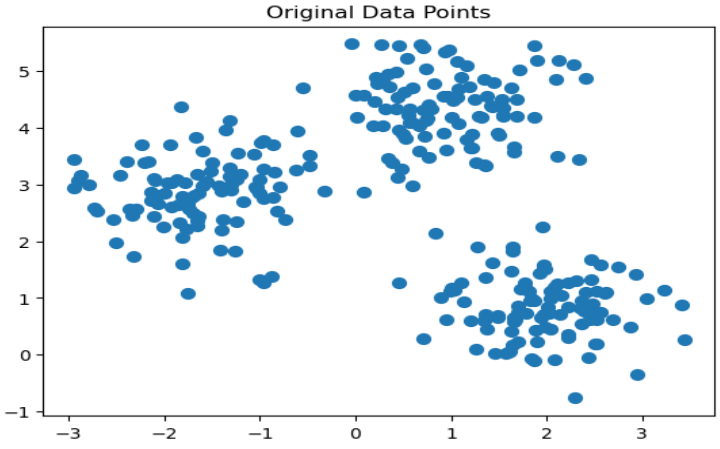
Output:

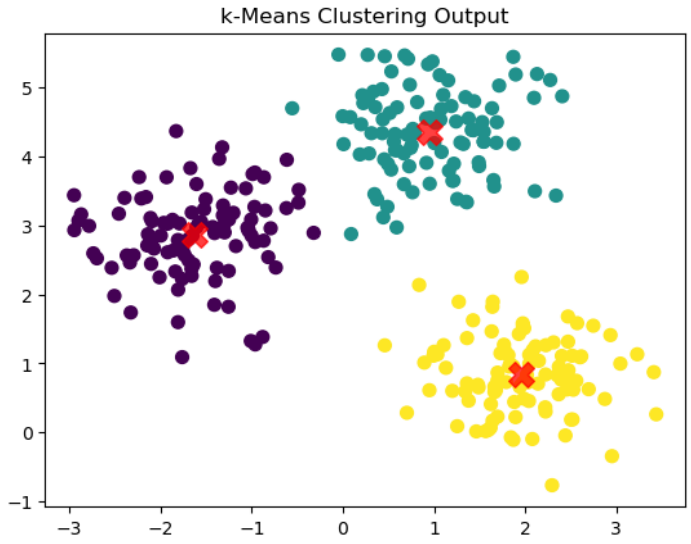
```
Accuracy: 0.7333333333333333
Predicted: [1 1 0 2 0 2 0 1 1 2 2 2 2 2 2 0 2 1 0 0 1 1 0 0 2 0 0 2 1 0 2 1 0 2 2
0 0
 1 2 1 1 0 2 0 0]
Actual:    [2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0 0 1 1 0 2 1 0 2 2
1 0
 1 1 1 2 0 2 0 0]
```

Practical No 6
k-Means Clustering Algorithm:

```
import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

from sklearn.datasets import make_blobs


X, y_true = make_blobs(n_samples=300, centers=3, cluster_std=0.60, random_state=0)


plt.scatter(X[:, 0], X[:, 1], s=50)

plt.title("Original Data Points")

plt.show()


kmeans = KMeans(n_clusters=3)

kmeans.fit(X)

y_kmeans = kmeans.predict(X)



plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, s=50)


centers = kmeans.cluster_centers_

plt.scatter(centers[:, 0], centers[:, 1], c='red', s=50, marker='X')

plt.title("k-Means Clustering Output")

plt.show()
```

Output:

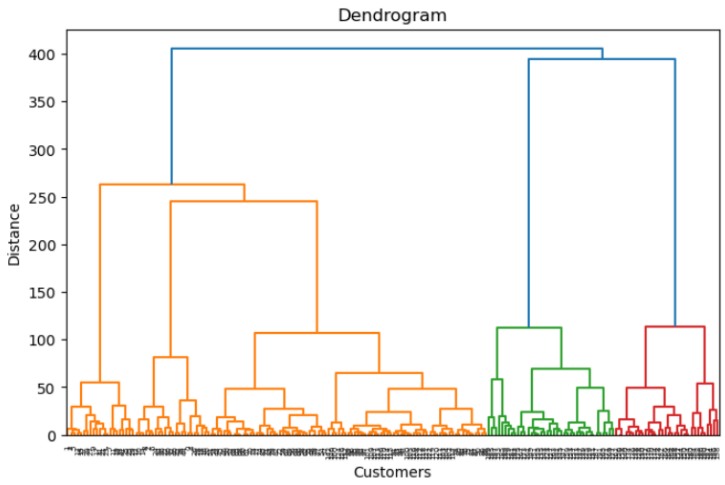


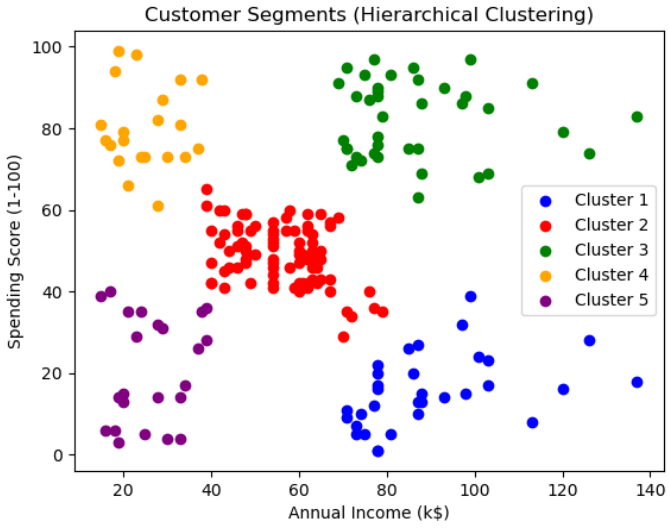


Practical No 7

Write a program for Hierarchical Clustering on Customers Dataset.

```
import pandas as pd
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram, linkage
from sklearn.cluster import AgglomerativeClustering

# Load dataset
df = pd.read_csv("C:/Mall_Customers.csv")

# Select Annual Income & Spending Score
X = df.iloc[:, [3, 4]].values

# Step 1: Draw dendrogram
plt.figure(figsize=(8,5))
dendrogram(linkage(X, method='ward'))
plt.title("Dendrogram")
plt.xlabel("Customers")
plt.ylabel("Distance")
plt.show()

#  Fit hierarchical clustering with 5 clusters
hc = AgglomerativeClustering(n_clusters=5, metric='euclidean', linkage='ward')
y_hc = hc.fit_predict(X)

#  Visualize clusters
plt.scatter(X[y_hc == 0, 0], X[y_hc == 0, 1], c='blue', label='Cluster 1')
plt.scatter(X[y_hc == 1, 0], X[y_hc == 1, 1], c='red', label='Cluster 2')
plt.scatter(X[y_hc == 2, 0], X[y_hc == 2, 1], c='green', label='Cluster 3')
plt.scatter(X[y_hc == 3, 0], X[y_hc == 3, 1], c='orange', label='Cluster 4')
plt.scatter(X[y_hc == 4, 0], X[y_hc == 4, 1], c='purple', label='Cluster 5')

plt.title("Customer Segments (Hierarchical Clustering)")
plt.xlabel("Annual Income (k$)")
plt.ylabel("Spending Score (1-100)")
plt.legend()
plt.show()
```





Practical No 8

Write a program for using PCA on MNIST Dataset.

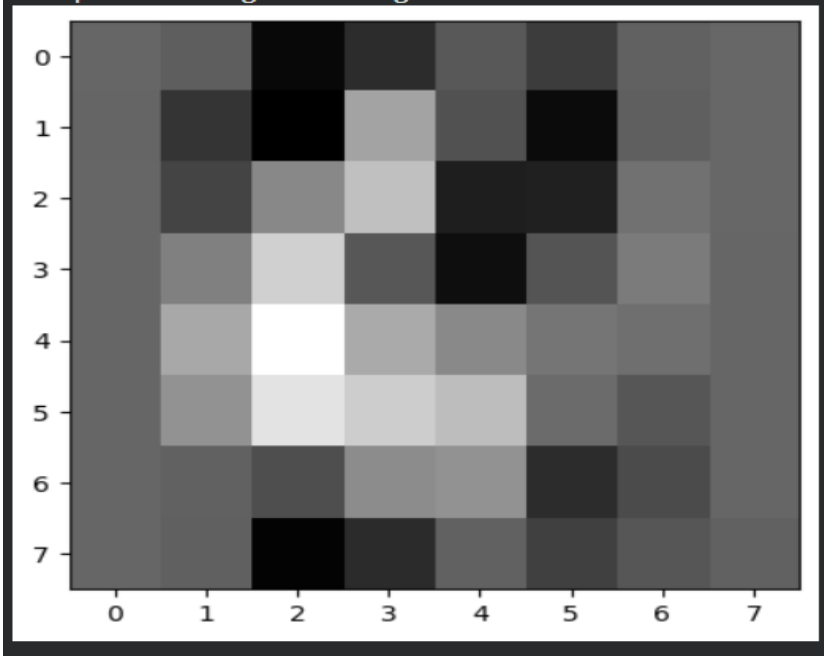
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.datasets import load_digits
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score,classification_report
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
import warnings
warnings.filterwarnings('ignore')
data = load_digits()
digits = pd.DataFrame(data.data)
target = pd.DataFrame(data.target)
digits.head()
print(digits.shape)
plt.imshow(digits.iloc[1].values.reshape(8,8),cmap='gray')
x_train,x_test,y_train,y_test = train_test_split(digits,target,test_size=0.2,random_state = 42)
random = RandomForestClassifier(n_estimators = 100,max_depth=10,min_samples_split=0.2)
random.fit(x_train,y_train)
predict = random.predict(x_test)
print('Accuracy Score for Random Forest Classification without PCA is
{}'.format(accuracy_score(y_test,predict)))
print(classification_report(y_test,predict))
pca = PCA(0.95)  #PCA(5)
x = pca.fit_transform(digits)
x.shape
x_train,x_test,y_train,y_test = train_test_split(x,target,test_size = 0.2,random_state = 42)
random = RandomForestClassifier(n_estimators = 100,max_depth=10,min_samples_split=0.2)
random.fit(x_train,y_train)
predict = random.predict(x_test)
print('Accuracy Score for Random Forest Classification after PCA is
{}'.format(accuracy_score(y_test,predict)))
print(classification_report(y_test,predict))
```

```
pca.components_[0]

len(pca.components_)

pca.explained_variance_ratio_

plt.imshow(pca.components_[0].reshape(8,8),cmap='gray')
```

Output

```
(1797, 64)
Accuracy Score for Random Forest Classification without PCA is 0.8138888888888889
              precision    recall  f1-score   support

           0       0.71      0.97      0.82        33
           1       0.81      0.75      0.78        28
           2       0.82      0.94      0.87        33
           3       0.79      0.91      0.85        34
           4       0.93      0.83      0.87        46
           5       0.94      0.64      0.76        47
           6       0.89      0.97      0.93        35
           7       0.63      0.94      0.75        34
           8       0.95      0.60      0.73        30
           9       0.84      0.65      0.73        40

    accuracy                           0.81       360
   macro avg       0.83      0.82      0.81       360
weighted avg       0.84      0.81      0.81       360

Accuracy Score for Random Forest Classification after PCA is 0.8388888888888889
              precision    recall  f1-score   support

           0       0.78      0.88      0.83        33
           1       0.74      0.71      0.73        28
           2       0.91      0.88      0.89        33
           3       0.86      0.91      0.89        34
           4       0.98      0.96      0.97        46
           5       0.85      0.87      0.86        47
           6       0.88      0.80      0.84        35
           7       0.74      1.00      0.85        34
           8       0.79      0.73      0.76        30
           9       0.83      0.60      0.70        40

    accuracy                           0.84       360
   macro avg       0.84      0.83      0.83       360
weighted avg       0.84      0.84      0.84       360
```

```
<matplotlib.image.AxesImage at 0x7f2e1ec0f950>
```

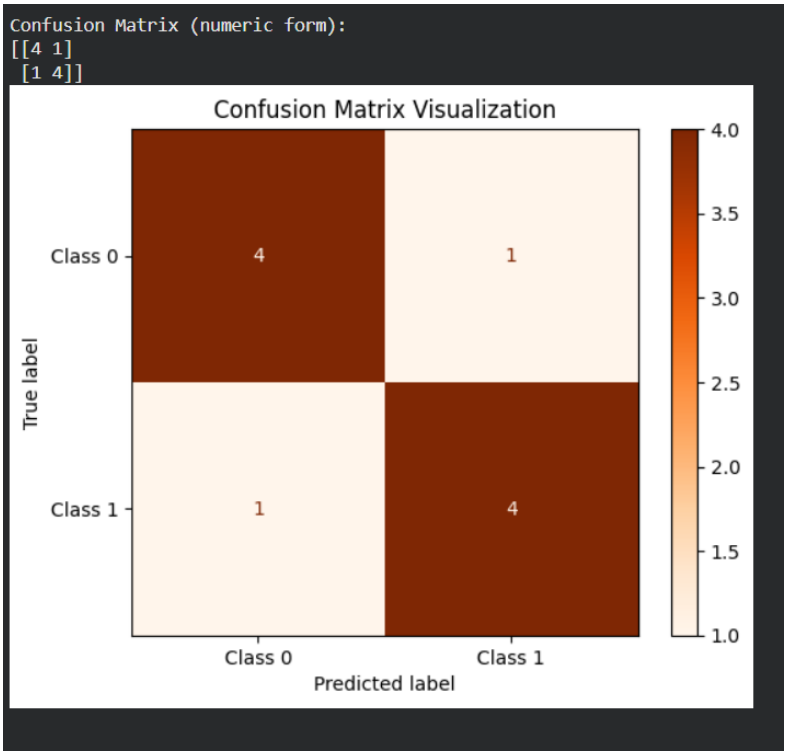


Practical No 9

Confusion Matrix

```
# Import required libraries
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

# Step 1: Define actual (true) and predicted labels
y_true = [0, 1, 1, 1, 0, 1, 0, 0, 1, 0]   # Actual values
y_pred = [0, 1, 0, 1, 0, 1, 0, 1, 1, 0]   # Predicted values

# Step 2: Compute the confusion matrix
cm = confusion_matrix(y_true, y_pred)

# Step 3: Print numeric confusion matrix
print("Confusion Matrix (numeric form):")
print(cm)

# Step 4: Plot a labeled confusion matrix (diagram)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                              display_labels=["Class 0", "Class 1"])
disp.plot(cmap=plt.cm.Oranges)  # You can try Blues, Greens, Purples, etc.
plt.title("Confusion Matrix Visualization")
plt.show()
```

```
Confusion Matrix (numeric form):
[[4 1]
 [1 4]]
```



Practical No 10

Accuracy, Precision, Recall, F1 Score calculations:

```
# Import necessary libraries
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
# Sample ground truth (actual values)
y_true = [0, 1, 1, 1, 0, 1, 0, 0, 1, 0]
# Sample model predictions
y_pred = [0, 1, 0, 1, 0, 1, 0, 1, 1, 0]
# Accuracy
accuracy = accuracy_score(y_true, y_pred)
# Precision
precision = precision_score(y_true, y_pred)
# Recall
recall = recall_score(y_true, y_pred)
# F1 Score
f1 = f1_score(y_true, y_pred)
# Display the results
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
```