B. K. BIRLA COLLEGE, KALYAN

(EMPOWERED AUTONOMOUS STATUS)

(DEPARTMENT OF INFORMATION TECHNOLOGY)

CERTIFICATE

This is to certify that Mr ***Aditya Arun Deo,*** *Roll No. , Exam*

Seat No. ***23258706*** *Has satisfactorily completed the Practical in*

Big Data Analytics *Of Semester* ***II*** *For the fulfilment of* ***MSc.IT***

(Cloud Computing) *For the year* ***2024 – 2025***

Date:

Place: *Kalyan*

Signature of Examiners

Professor-In-Charge

Head of Department

(Information Technology)

Sr. No.	Name of the Practical	Date	Sign
1	Exploring Big Data Characteristics with Real Datasets.		
2	Comparative Analysis of Traditional BI vs. Big Data Tools		
3	Implementing K-Means and DBSCAN Clustering		
4	Association Rule Mining using Apriori Algorithm		
5	Regression Analysis and Evaluation		
6	Building And Evaluating Classification Model		
7	Time Series Forecasting using ARIMA		
8	Text Analysis and Sentiment Detection		
9	Hadoop HDFS & MapReduce Word Count Program.		
10	Building a Spark Application in PySpark		

Practical 1: Exploring Big Data Characteristics with Real Datasets.

In [ ]:
```
#Exploring Big Data Characteristics with Real Datasets.
```

In [1]:
```
import pandas as pd
```

In [2]:
```
import numpy as np
```

In [3]:
```
df = pd.read_csv("OnlineRetail.csv")
```

In [4]:
```
df.head()
```

Out[4]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Cou
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6.0	12/1/2010 8:26	2.55	17850.0	Un Kingc
1	536365	71053	WHITE METAL LANTERN	6.0	12/1/2010 8:26	3.39	17850.0	Un Kingc
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8.0	12/1/2010 8:26	2.75	17850.0	Un Kingc
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6.0	12/1/2010 8:26	3.39	17850.0	Un Kingc
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6.0	12/1/2010 8:26	3.39	17850.0	Un Kingc

In [5]:
```
df.tail()
```

Out[5]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID
342979	566923	23032	DRAWER KNOB CRACKLE GLAZE IVORY	4.0	9/15/2011 15:05	1.65	15511.0
342980	566923	82484	WOOD BLACK BOARD ANT WHITE FINISH	1.0	9/15/2011 15:05	7.95	15511.0
342981	566923	22423	REGENCY CAKESTAND 3 TIER	2.0	9/15/2011 15:05	12.75	15511.0
342982	566923	23413	VINTAGE COFFEE GRINDER BOX	3.0	9/15/2011 15:05	4.95	15511.0
342983	566923	22061	LARGE CA	NaN	NaN	NaN	NaN

In [6]:
```
df.sample(5)
```

Out[6]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID
151733	549568	22382	LUNCH BAG SPACEBOY DESIGN	4.0	4/10/2011 15:09	1.65	15665.0
159686	550352	21324	HANGING MEDINA LANTERN SMALL	1.0	4/18/2011 10:21	2.95	16719.0
59675	541371	22088	PAPER BUNTING COLOURED LACE	3.0	1/17/2011 14:51	2.95	13983.0
47175	540395	22844	VINTAGE CREAM DOG FOOD CONTAINER	2.0	1/6/2011 18:24	8.50	14702.0
195928	553765	22585	PACK OF 6 BIRDY GIFT TAGS	12.0	5/19/2011 10:43	1.25	17245.0

In [7]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 342984 entries, 0 to 342983
Data columns (total 8 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   InvoiceNo    342984 non-null  object
 1   StockCode    342984 non-null  object
 2   Description  341788 non-null  object
 3   Quantity     342983 non-null  float64
 4   InvoiceDate  342983 non-null  object
 5   UnitPrice    342983 non-null  float64
 6   CustomerID   249983 non-null  float64
 7   Country      342983 non-null  object
dtypes: float64(3), object(5)
memory usage: 14.4+ MB
```

In [8]:
```
df.describe()
```

Out[8]:

	Quantity	UnitPrice	CustomerID
count	342983.000000	342983.000000	249983.000000
mean	9.703449	4.855598	15279.136837
std	188.246470	110.841420	1725.944155
min	-74215.000000	-11062.060000	12346.000000
25%	1.000000	1.250000	13862.000000
50%	3.000000	2.100000	15150.000000
75%	10.000000	4.130000	16813.000000
max	74215.000000	38970.000000	18287.000000

In [10]:
```
#Volume
print("Shape:", df.shape)
print("Memory usage (MB):", df.memory_usage(deep=True).sum() / 1024**2)
```

```
Shape: (342984, 8)
Memory usage (MB): 70.17771339416504
```

In [11]:
```
#Velocity
df['InvoiceDate'] = pd.to_datetime(df['InvoiceDate'], dayfirst=True, errors='coerce
```

In [12]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 342984 entries, 0 to 342983
Data columns (total 8 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   InvoiceNo    342984 non-null  object
 1   StockCode    342984 non-null  object
 2   Description  341788 non-null  object
 3   Quantity     342983 non-null  float64
 4   InvoiceDate  154132 non-null  datetime64[ns]
 5   UnitPrice    342983 non-null  float64
 6   CustomerID   249983 non-null  float64
 7   Country      342983 non-null  object
dtypes: datetime64[ns](1), float64(3), object(4)
memory usage: 15.7+ MB
```

In [13]:
```
#Varity
print("Data Types:\n", df.dtypes)
print("Variety Count:\n", df.dtypes.value_counts())
```

```
Data Types:
 InvoiceNo              object
StockCode              object
Description            object
Quantity              float64
InvoiceDate    datetime64[ns]
UnitPrice             float64
CustomerID            float64
Country                object
dtype: object
Variety Count:
 object            4
float64           3
datetime64[ns]    1
Name: count, dtype: int64
```

In [18]:
```
#Veracity
# 1) Missing Values
print("Missing Values:\n", df.isnull().sum())
```

```
Missing Values:
 InvoiceNo          0
StockCode           0
Description      1196
Quantity            1
InvoiceDate    188852
UnitPrice           1
CustomerID      93001
Country             1
dtype: int64
```

In [15]:
```
# 2) Check for Duplicates
print("Duplicate Records:", df.duplicated().sum())
```

```
Duplicate Records: 2667
```

In [19]:
```
# 3) Drop Values
df_clean = df.dropna(subset=['CustomerID'])
print("After Cleaning:", df_clean.shape)
```

```
After Cleaning: (249983, 8)
```

In [22]:
```
#Checking CustomerID for Changes
print("Missing Values:\n", df_clean.isnull().sum())

#Note: name changed to "df_clean" for Dropping the Values
```

```
Missing Values:
 InvoiceNo          0
StockCode           0
Description         0
Quantity            0
InvoiceDate    138298
UnitPrice           0
CustomerID          0
Country             0
dtype: int64
```

In [24]:
```
df_clean1 = df.dropna(subset=['InvoiceDate'])
print("After Cleaning:", df_clean1.shape)
```

```
After Cleaning: (154132, 8)
```

In [25]:
```
print("Missing Values:\n", df_clean1.isnull().sum())
```

```
Missing Values:
 InvoiceNo          0
StockCode           0
Description       571
Quantity            0
InvoiceDate         0
UnitPrice           0
CustomerID      42447
Country             0
dtype: int64
```

In [26]:
```
#Value
df['TotalPrice'] = df['Quantity'] *df['UnitPrice']
print("Top Products:\n", df['Description'].value_counts().head())
print("Top Countries by Revenue:\n", df.groupby('Country')['TotalPrice'].sum().sort
```

```
Top Products:
 Description
WHITE HANGING HEART T-LIGHT HOLDER    1761
REGENCY CAKESTAND 3 TIER              1651
JUMBO BAG RED RETROSPOT               1517
PARTY BUNTING                         1433
LUNCH BAG RED RETROSPOT               1221
Name: count, dtype: int64
Top Countries by Revenue:
 Country
United Kingdom    5116798.392
Netherlands        189845.960
EIRE               184271.280
Germany            146521.550
France             123816.990
Name: TotalPrice, dtype: float64
```

In [ ]:

Practical 2: Comparative Analysis of Traditional BI vs. Big Data Tools

In [2]:
```
#COMPARATIVE ANALYSIS ( PRACTICAL 2 )
import pandas as pd
df = pd.read_csv("OnlineRetail.csv")
df.head()
```

Out[2]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Cou
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6.0	12/1/2010 8:26	2.55	17850.0	Un Kingc
1	536365	71053	WHITE METAL LANTERN	6.0	12/1/2010 8:26	3.39	17850.0	Un Kingc
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8.0	12/1/2010 8:26	2.75	17850.0	Un Kingc
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6.0	12/1/2010 8:26	3.39	17850.0	Un Kingc
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6.0	12/1/2010 8:26	3.39	17850.0	Un Kingc

In [9]:
```
#Convert InvoiceDate to Datetime format:
df['InvoiceDate'] = pd.to_datetime(df['InvoiceDate'], dayfirst=True, errors='coerce
```

In [10]:
```
# Step 1: Create a new column for Total Price
df['TotalPrice'] = df['Quantity'] *df['UnitPrice']

# Step 2: Group by month and aggregate
monthly_sales = df.groupby(df['InvoiceDate'].dt.to_period('M')).agg({
    'Quantity': 'sum',
    'TotalPrice': 'sum'
}).reset_index()

# Step 3: Print Results
print(monthly_sales.head())
```

```
  InvoiceDate  Quantity  TotalPrice
0     2010-01   26814.0    58635.56
1     2010-02   21023.0    46207.28
2     2010-03   14830.0    45620.46
3     2010-05   16395.0    31383.95
4     2010-06   21419.0    53860.18
```

In [11]:
```python
import matplotlib.pyplot as plt

# CONVERT 'InvoiceDate' back to datetime for plotting
monthly_sales['InvoiceDate'] = monthly_sales['InvoiceDate'].dt.to_timestamp()
plt.figure(figsize=(14, 6))

#PLOT TOTAL SALES
plt.plot(monthly_sales['InvoiceDate'], monthly_sales['TotalPrice'], marker='o', lab

#PLOT QUANTITY
plt.plot(monthly_sales['InvoiceDate'], monthly_sales['Quantity'], marker='s', label

plt.title('Monthly Sales & Quantity')
plt.xlabel('Month')
plt.ylabel('Value')
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
```

In [ ]:

Practical 3: Implementing K-Means and DBSCAN Clustering

In [8]:
```
%pip install seaborn
```

In [9]:
```
#PRACTICAL 3

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans, DBSCAN
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import silhouette_score
```

In [11]:
```
#Step 2: Load Dataset
df = pd.read_csv("Mall_Customers.csv")
print(df.head())
```

```
   CustomerID   Genre  Age  Annual Income (k$)  Spending Score (1-100)
0           1    Male   19                  15                      39
1           2    Male   21                  15                      81
2           3  Female   20                  16                       6
3           4  Female   23                  16                      77
4           5  Female   31                  17                      40
```

In [12]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 5 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   CustomerID              200 non-null    int64
 1   Genre                   200 non-null    object
 2   Age                     200 non-null    int64
 3   Annual Income (k$)      200 non-null    int64
 4   Spending Score (1-100)  200 non-null    int64
dtypes: int64(4), object(1)
memory usage: 7.1+ KB
```

In [13]:
```
df.describe()
```

Out[13]:

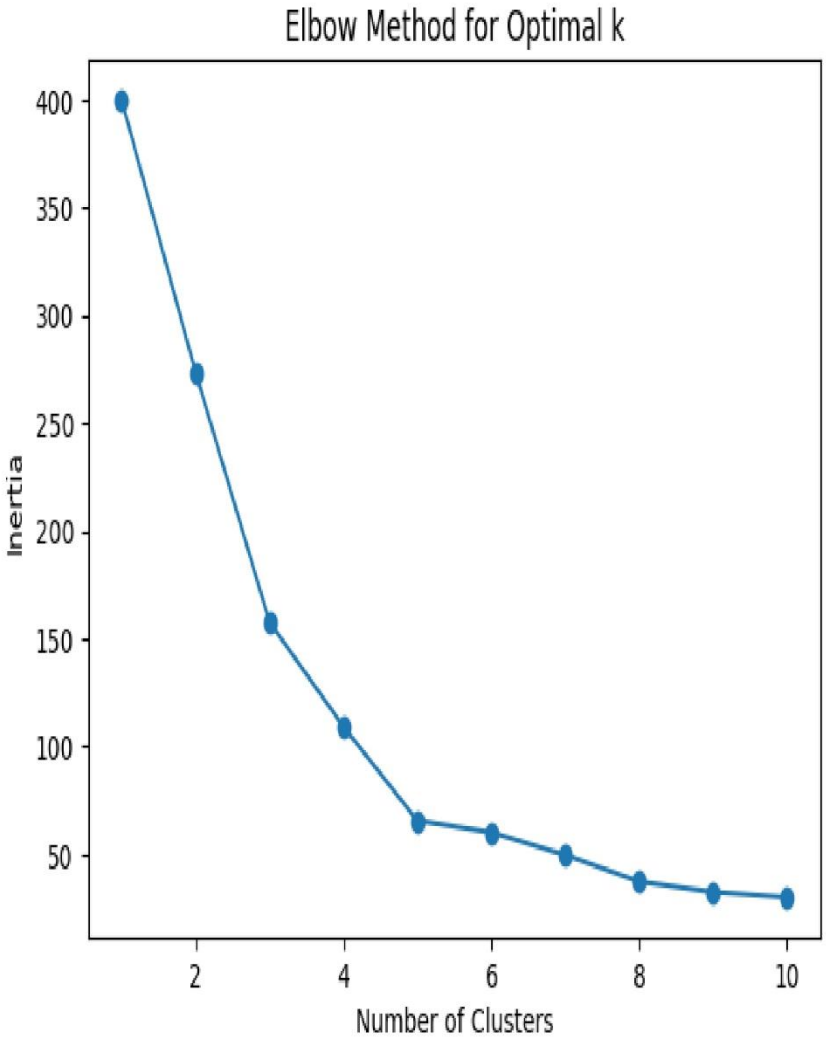
	CustomerID	Age	Annual Income (k$)	Spending Score (1-100)
count	200.000000	200.000000	200.000000	200.000000
mean	100.500000	38.850000	60.560000	50.200000
std	57.879185	13.969007	26.264721	25.823522
min	1.000000	18.000000	15.000000	1.000000
25%	50.750000	28.750000	41.500000	34.750000
50%	100.500000	36.000000	61.500000	50.000000
75%	150.250000	49.000000	78.000000	73.000000
max	200.000000	70.000000	137.000000	99.000000

In [14]:
```
df.isnull().sum()
```

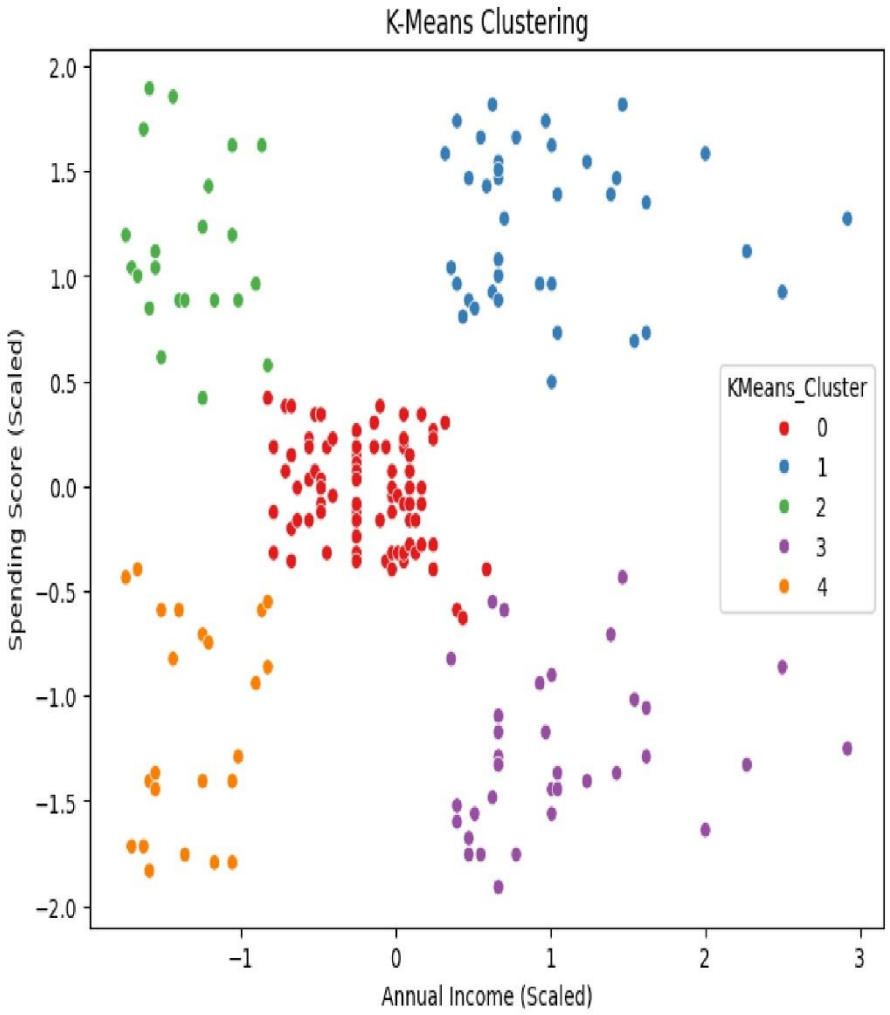
Out[14]:
```
CustomerID              0
Genre                   0
Age                     0
Annual Income (k$)      0
Spending Score (1-100)  0
dtype: int64
```

In [18]:
```
#Step 3 DATA PREPROCESSING

X = df[['Annual Income (k$)', 'Spending Score (1-100)']]
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

In [20]:
```
# Step 4.1 Elbow Method to find optimal clusters
inertia = []
for k in range(1,11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_scaled)
    inertia.append(kmeans.inertia_)

plt.plot(range(1,11), inertia, marker='o')
plt.xlabel('Number of Clusters')
plt.ylabel('Inertia')
plt.title('Elbow Method for Optimal k')
plt.show()
```

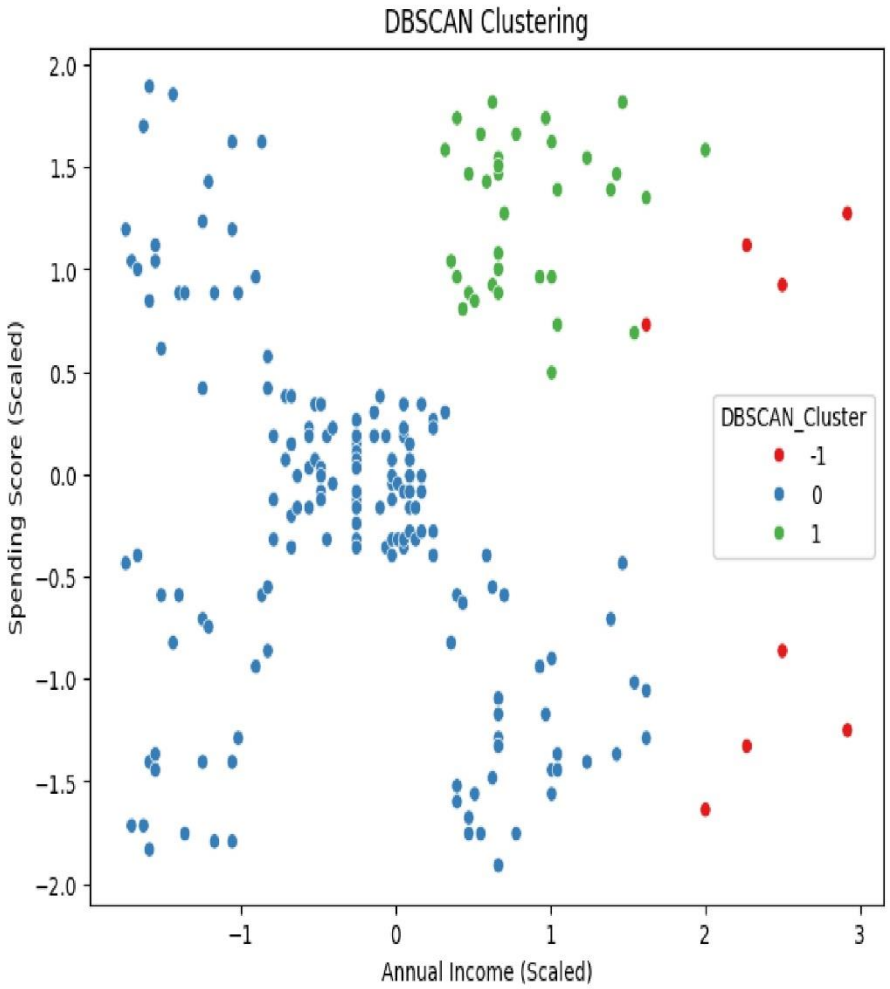


In [21]:
```
# 4.2 FIT Kmeans method with k=5 (example)
kmeans = KMeans(n_clusters=5, random_state=42)
df['KMeans_Cluster'] = kmeans.fit_predict(X_scaled)

#VISUALIZE KMEANS CLUSTER
plt.figure(figsize=(8,6))
sns.scatterplot(x=X_scaled[:,0], y=X_scaled[:,1], hue=df['KMeans_Cluster'], palette
plt.title('K-Means Clustering')
plt.xlabel('Annual Income (Scaled)')
plt.ylabel('Spending Score (Scaled)')
plt.show()
```



In [22]:
```
# STEP 5 : DBSCAN CLUSTER
dbscan = DBSCAN(eps=0.5, min_samples=5)
df['DBSCAN_Cluster'] = dbscan.fit_predict(X_scaled)
```

In [23]:
```
# STEP 6 : VISUALIZE THE DBSCAN CLUSTER
plt.figure(figsize=(8,6))
sns.scatterplot(x=X_scaled[:,0], y=X_scaled[:,1], hue=df['DBSCAN_Cluster'], palette
plt.title('DBSCAN Clustering')
plt.xlabel('Annual Income (Scaled)')
plt.ylabel('Spending Score (Scaled)')
plt.show()
```


DBSCAN Clustering
Spending Score (Scaled)
Annual Income (Scaled)
DBSCAN_Cluster
-1
0
1
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−1
0
1
2
3

In [ ]:

Practical 4: Association Rule Mining using Apriori Algorithm

In [1]:
```
#Association Rule Mining Using Apriori Algorithm
```

```
---------------------------------------------------------------------------
OSError                                   Traceback (most recent call last)
Cell In[1], line 3
      1 #Association Rule Mining Using Apriori Algorithm
----> 3 get_ipython().system('pip install mlxtend')

File /lib/python3.12/site-packages/IPython/core/interactiveshell.py:2653, in InteractiveShell.system_piped(self, cmd)
   2648     raise OSError("Background processes not supported.")
   2650 # we explicitly do NOT return the subprocess status code, because
   2651 # a non-None value would trigger :func:`sys.displayhook` calls.
   2652 # Instead, we store the exit_code in user_ns.
-> 2653 self.user_ns['_exit_code'] = system(self.var_expand(cmd, depth=1))

File /lib/python3.12/site-packages/IPython/utils/_process_emscripten.py:11, in system(cmd)
     10 def system(cmd):
---> 11     raise OSError("Not available")

OSError: Not available
```

In [2]:
```
%pip install mlxtend
```

In [3]:
```
#Step 1 : Import Librarires
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori, association_rules
```

In [5]:
```
#Step 2: Sample Supermarket transaction Dataset
transactions = [
    ['Milk','Bread','Butter'],
    ['Bread','Butter'],
    ['Milk','Diaper','Beer','Eggs'],
    ['Bread','Butter','Diaper'],
    ['Milk','Bread','Butter','Diaper'],
    ['Beer','Diaper'],
    ['Milk','Bread','Butter','Eggs'],
]
```

```
/lib/python3.12/site-packages/pyodide_kernel/kernel.py:100: DeprecationWarning: `should_run_async` will not call `transform_cell` automatically in the future. Please pass the result to `transformed_cell` argument and any exception that happen during thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  if self.interpreter.should_run_async(code):
```

In [8]:
```
#Step 3: Convert Transactions to one-hot encoded DataFrame

te = TransactionEncoder()
te_ary = te.fit(transactions).transform(transactions)
df_onehot = pd.DataFrame(te_ary, columns=te.columns_)
```

```
print("One-hot encoded dataset:")
display(df_onehot)
```

```
/lib/python3.12/site-packages/pyodide_kernel/kernel.py:100: DeprecationWarning: `sho
uld_run_async` will not call `transform_cell` automatically in the future. Please pa
ss the result to `transformed_cell` argument and any exception that happen during th
etransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  if self.interpreter.should_run_async(code):
```

One-hot encoded dataset:

	Beer	Bread	Butter	Diaper	Eggs	Milk
0	False	True	True	False	False	True
1	False	True	True	False	False	False
2	True	False	False	True	True	True
3	False	True	True	True	False	False
4	False	True	True	True	False	True
5	True	False	False	True	False	False
6	False	True	True	False	True	True

In [10]:
```
#Step 4: Generate frequency itemsets with minimum support of 0.4
frequent_itemsets = apriori(df_onehot, min_support=0.4, use_colnames=True)
print("Frequent Itemsets:")
display(frequent_itemsets)

#Step 5: Generate Association rules with minimum confidence of 0.7
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.7

print("Association Rules:")
display(rules[['antecedents','consequents','support','confidence','lift']])
```

```
/lib/python3.12/site-packages/pyodide_kernel/kernel.py:100: DeprecationWarning: `sho
uld_run_async` will not call `transform_cell` automatically in the future. Please pa
ss the result to `transformed_cell` argument and any exception that happen during th
etransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  if self.interpreter.should_run_async(code):
```

Frequent Itemsets:

	support	itemsets
0	0.714286	(Bread)
1	0.714286	(Butter)
2	0.571429	(Diaper)
3	0.571429	(Milk)
4	0.714286	(Bread, Butter)
5	0.428571	(Bread, Milk)
6	0.428571	(Butter, Milk)
7	0.428571	(Bread, Butter, Milk)

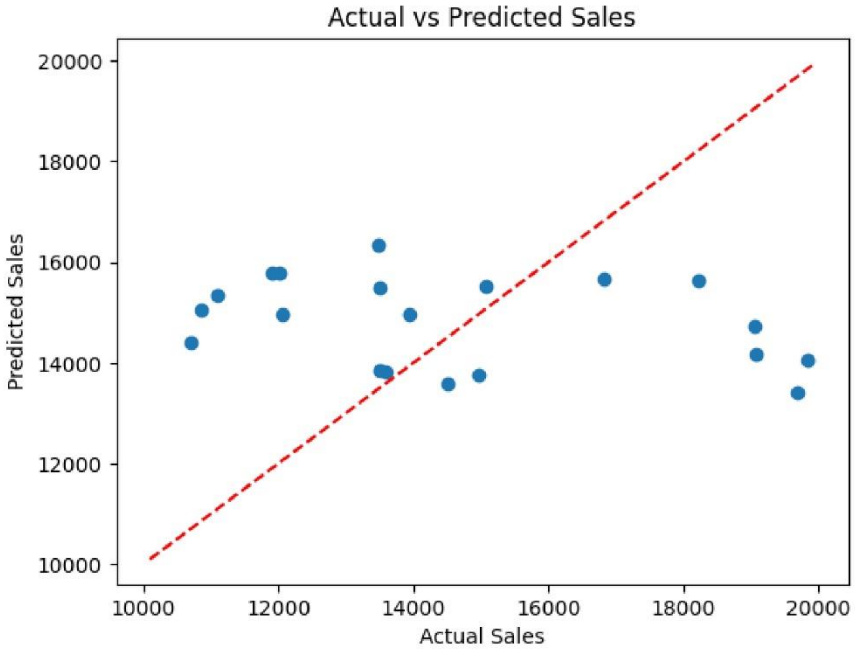
```
Association Rules:
```

	antecedents	consequents	support	confidence	lift
0	(Bread)	(Butter)	0.714286	1.00	1.40
1	(Butter)	(Bread)	0.714286	1.00	1.40
2	(Milk)	(Bread)	0.428571	0.75	1.05
3	(Milk)	(Butter)	0.428571	0.75	1.05
4	(Bread, Milk)	(Butter)	0.428571	1.00	1.40
5	(Butter, Milk)	(Bread)	0.428571	1.00	1.40
6	(Milk)	(Bread, Butter)	0.428571	0.75	1.05

```
In [ ]:
```

Practical 5: Regression Analysis and Evaluation

In [9]:
```
#Step 8: Plot the Actual vs Predicted
plt.scatter(y_test, y_pred)
plt.xlabel("Actual Sales")
plt.ylabel("Predicted Sales")
plt.title("Actual vs Predicted Sales")
plt.plot([y.min(), y.max()], [y.min(), y.max()], 'r--')
plt.show()
```



In [12]:
```
#Part B: Logistic Regression - Predicting Customer Churn

#Step 1: Import Libraries
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report


#Step 2: Create Synthetic Churn Dataset
churn_data = pd.DataFrame({
    'Monthly_Charges' : np.random.uniform(20, 120, 100),
    'Tenure': np.random.randient(1, 72, 100),
    'Churn': np.random.choice([0,1], 100) # 0 = No Churn, 1 = Churn
})


#Step 3: Prepare Features and Target Variable
X = churn_data[['Monthly_Charges','Tenure']]
y= churn_data['Churn']

#Step 4: Split Data into Train & Test sets
```

```
  warnings.warn(
Linear Regression - RMSE: 3365.34
Linear Regression - R^2 Score: -0.27
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

#Step 5: Train Logistic Regression Model
log_model = LogisticRegression()
log_model.fit(X_train, y_train)

#Step 6: Predict on Test Data
y_pred = log_model.predict(X_test)

#Step 7: Evaluate the Model Performance
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

print(f"Logistic Regression - Accuracy: {accuracy:.2f}")
print("Confusion Matrix:")
print(conf_matrix)
print("\nClassification Report:")
print(classification_report(y_test, y_pred)
```

```
  Cell In[12], line 40

    ^
SyntaxError: incomplete input
```

In [13]:
```
#Step 1: Import Libraries
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

In [16]:
```
#Step 2: Create Synthetic Churn Dataset
churn_data = pd.DataFrame({
    'Monthly_Charges' : np.random.uniform(20, 120, 100),
    'Tenure': np.random.randint(1, 72, 100),
    'Churn': np.random.choice([0,1], 100) # 0 = No Churn, 1 = Churn
})
```

In [17]:
```
#Step 3: Prepare Features and Target Variable
X = churn_data[['Monthly_Charges','Tenure']]
y= churn_data['Churn']
```

In [20]:
```
#Step 4: Split Data into Train & Test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

#Step 5: Train Logistic Regression Model
log_model = LogisticRegression()
log_model.fit(X_train, y_train)

#Step 6: Predict on Test Data
y_pred = log_model.predict(X_test)

#Step 7: Evaluate the Model Performance
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

print(f"Logistic Regression - Accuracy: {accuracy:.2f}")
```

```
print("Confusion Matrix:")
print(conf_matrix)
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
Logistic Regression - Accuracy: 0.50
Confusion Matrix:
[[9 1]
 [9 1]]

Classification Report:
              precision    recall  f1-score   support

           0       0.50      0.90      0.64        10
           1       0.50      0.10      0.17        10

    accuracy                           0.50        20
   macro avg       0.50      0.50      0.40        20
weighted avg       0.50      0.50      0.40        20
```

In [ ]:

Practical 6: Building and Evaluating Classification Model

In [1]:
```
#Building And Evaluating Classification Model

# Step1: Import necessary libraries
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix

# Step 2: Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target
```

In [2]:
```
#Step 3:
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_sta

#Step 4: Initialize and Train Models
# Initialize models
dt_model = DecisionTreeClassifier(random_state=42)
nb_model = GaussianNB()
svm_model = SVC(random_state=42)
knn_model = KNeighborsClassifier()

# Train models
dt_model.fit(X_train, y_train)
nb_model.fit(X_train, y_train)
svm_model.fit(X_train, y_train)
knn_model.fit(X_train, y_train)

#Step 5: Make Predictions
dt_pred = dt_model.predict(X_test)
nb_pred = nb_model.predict(X_test)
svm_pred = svm_model.predict(X_test)
knn_pred = knn_model.predict(X_test)

#Step 6: Evaluate Models
#Use classification_report to get precision, recall, F1-score:

print("Decision Tree:\n", classification_report(y_test, dt_pred))
print("Naive Bayes:\n", classification_report(y_test, nb_pred))
print("SVM:\n", classification_report(y_test, svm_pred))
print("KNN:\n", classification_report(y_test, knn_pred))
print("Decision Tree Confusion Matrix:\n", confusion_matrix(y_test, dt_pred))
print("Naive Bayes Confusion Matrix:\n", confusion_matrix(y_test, nb_pred))
print("SVM Confusion Matrix:\n", confusion_matrix(y_test, svm_pred))
print("KNN Confusion Matrix:\n", confusion_matrix(y_test, knn_pred))
```

```
Decision Tree:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      1.00      1.00        13
           2       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45

Naive Bayes:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      0.92      0.96        13
           2       0.93      1.00      0.96        13

    accuracy                           0.98        45
   macro avg       0.98      0.97      0.97        45
weighted avg       0.98      0.98      0.98        45

SVM:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      1.00      1.00        13
           2       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45

KNN:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      1.00      1.00        13
           2       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45

Decision Tree Confusion Matrix:
 [[19  0  0]
 [ 0 13  0]
 [ 0  0 13]]
Naive Bayes Confusion Matrix:
 [[19  0  0]
 [ 0 12  1]
 [ 0  0 13]]
SVM Confusion Matrix:
 [[19  0  0]
 [ 0 13  0]
 [ 0  0 13]]
```

```
KNN Confusion Matrix:
 [[19  0  0]
 [ 0 13  0]
 [ 0  0 13]]
```

In [ ]:

Practical 7: Time Series Forecasting using ARIMA

In [1]:
```
#PRACTICAL 7 : TIME SERIES FORCASTING USING ARIMA MODEL
#Step 1: Import Required Libraries
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from sklearn.metrics import mean_squared_error
import numpy as np
```

```
Matplotlib is building the font cache; this may take a moment.
```

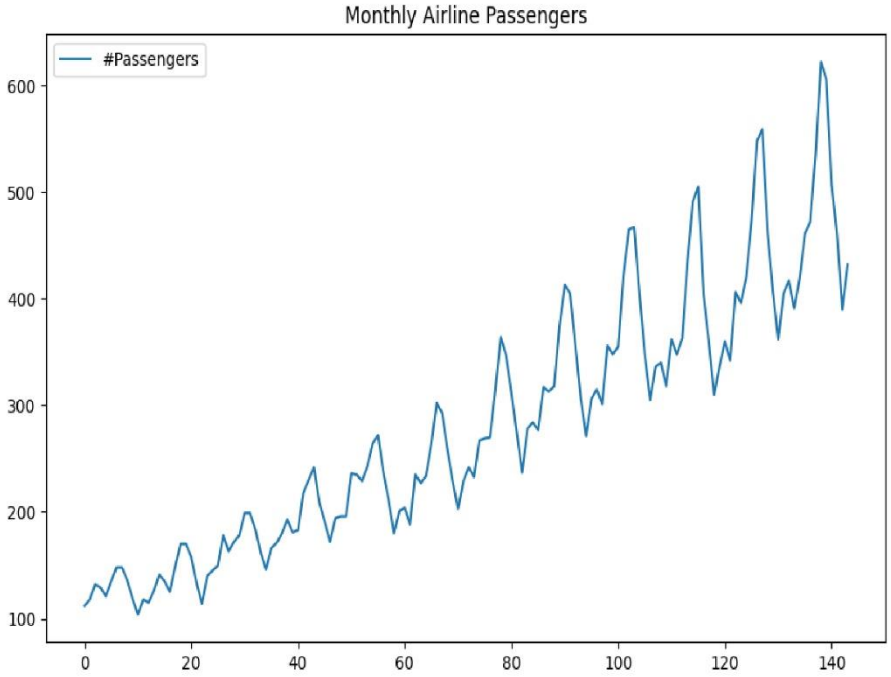
In [2]:
```
#Step 2: Load Time Series Dataset
 #For demonstration, let's use a simple time series dataset – for example, monthly
 # Load sample data (Monthly Airline Passengers dataset)
data = pd.read_csv('AirPassengers.csv')
print(data.columns)
# Display the first few rows
data.head()
```

```
Index(['Month', '#Passengers'], dtype='object')
```

Out[2]:

	Month	#Passengers
0	1949-01	112
1	1949-02	118
2	1949-03	132
3	1949-04	129
4	1949-05	121

In [3]:
```
#Step 3: Visualize the Time Series
data.plot(figsize=(10,6))
plt.title('Monthly Airline Passengers')
plt.show()
```



In [4]:
```
print(data.columns)
```

```
Index(['Month', '#Passengers'], dtype='object')
```

In [5]:
```
# Rename '#Passengers' to 'Passengers' once
data.rename(columns={'#Passengers': 'Passengers'}, inplace=True)
```

In [7]:
```
print(data.dtypes)
```

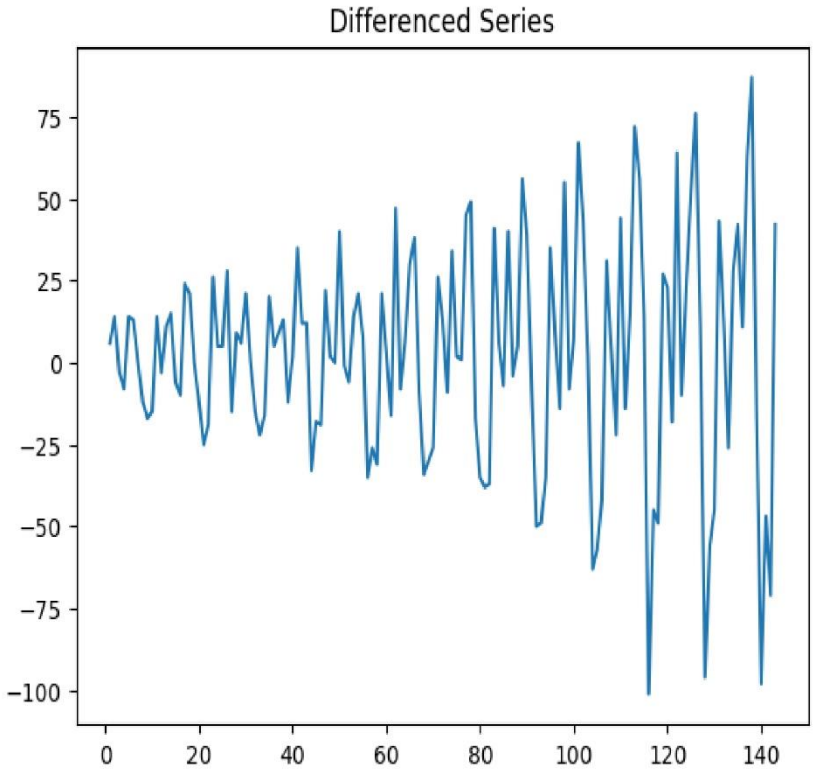
```
Month         object
Passengers     int64
dtype: object
```

In [8]:
```
data['Passengers'] = pd.to_numeric(data['Passengers'], errors='coerce')
```

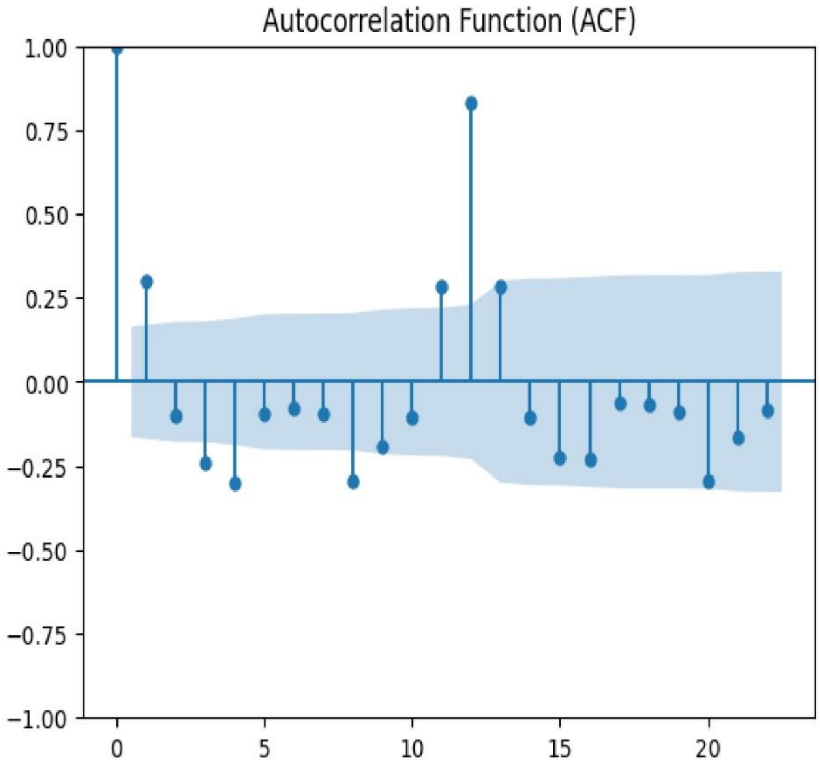
In [14]:
```
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.stattools import adfuller

# Convert to numeric
data['Passengers'] = pd.to_numeric(data['Passengers'], errors='coerce')

# Drop rows where conversion failed
data = data.dropna(subset=['Passengers'])

# Check stationarity
result = adfuller(data['Passengers'])
print(f'ADF Statistic: {result[0]}')
print(f'p-value: {result[1]}')

if result[1] > 0.05:
    print("Non-stationary, applying differencing...")
```

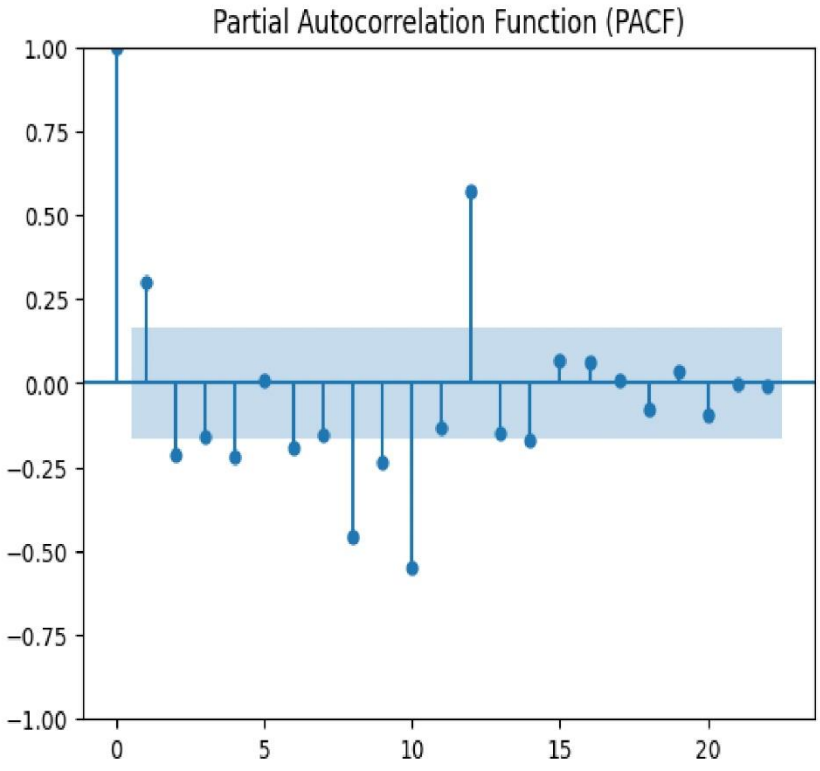
```
    data_diff = data['Passengers'].diff().dropna()
    data_diff.plot()
    plt.title('Differenced Series')
    plt.show()
else:
    print("Series is stationary, no differencing needed.")
```

```
ADF Statistic: 0.8153688792060613
p-value: 0.9918802434376413
Non-stationary, applying differencing...
```



In [15]:
```
import matplotlib.pyplot as plt
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

# Assuming data_diff is your differenced series (stationary time series)

# Plot ACF (for q parameter)
plot_acf(data_diff)
plt.title('Autocorrelation Function (ACF)')
plt.show()

# Plot PACF (for p parameter)
plot_pacf(data_diff, method='ywm')  # ywm method is recommended for stability
plt.title('Partial Autocorrelation Function (PACF)')
plt.show()
```





In [17]:
```
from statsmodels.tsa.arima.model import ARIMA

# Use original series (not differenced), just the column as Series
```

```
series = data['Passengers']

# Fit ARIMA model with order p=2, d=1, q=2
model = ARIMA(series, order=(2,1,2))
model_fit = model.fit()

print(model_fit.summary())
```

```
/lib/python3.12/site-packages/statsmodels/base/model.py:607: ConvergenceWarning: Max
imum Likelihood optimization failed to converge. Check mle_retvals
  warnings.warn("Maximum Likelihood optimization failed to "
```

SARIMAX Results

Dep. Variable:	Passengers	No. Observations:	144
Model:	ARIMA(2, 1, 2)	Log Likelihood	-671.673
Date:	Sat, 24 May 2025	AIC	1353.347
Time:	13:01:58	BIC	1368.161
Sample:	0	HQIC	1359.366
	- 144		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	1.6850	0.020	83.057	0.000	1.645	1.725
ar.L2	-0.9548	0.017	-55.418	0.000	-0.989	-0.921
ma.L1	-1.8432	0.125	-14.752	0.000	-2.088	-1.598
ma.L2	0.9953	0.135	7.352	0.000	0.730	1.261
sigma2	665.9521	114.318	5.825	0.000	441.892	890.012

Ljung-Box (L1) (Q):	0.30	Jarque-Bera (JB):	1.84
Prob(Q):	0.59	Prob(JB):	0.40
Heteroskedasticity (H):	7.38	Skew:	0.27
Prob(H) (two-sided):	0.00	Kurtosis:	3.14

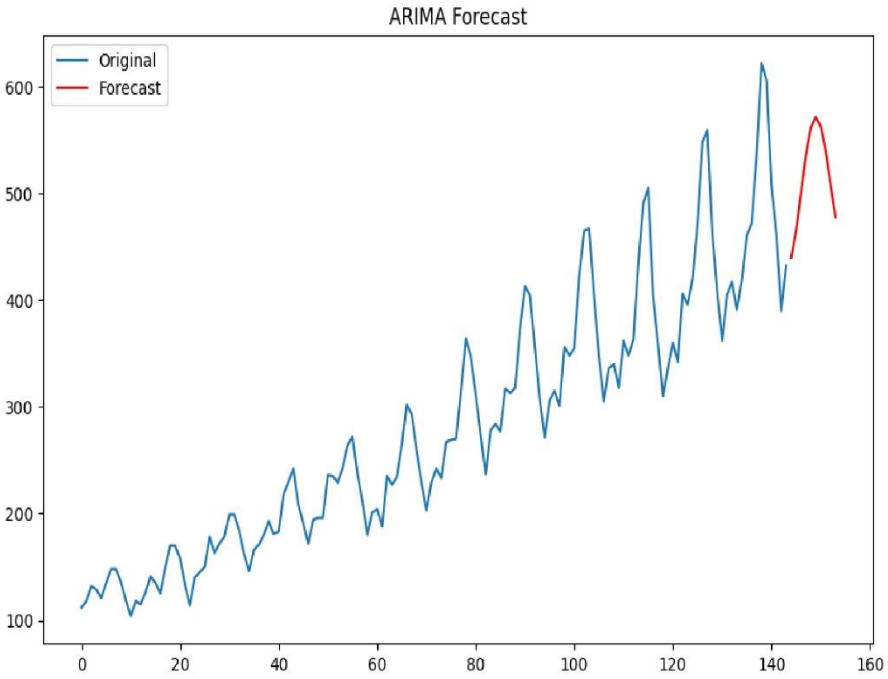
Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-ste
p).

In [20]:
```
#Step 7: Forecast Future Values
#Forecast the next 12 months:
forecast = model_fit.forecast(steps=12)
print(forecast)
```

```
144    439.856218
145    465.298515
146    500.667519
147    535.971291
148    561.686794
149    571.308307
150    562.966575
151    539.723776
152    508.524522
153    478.146642
154    456.749871
155    449.702036
Name: predicted_mean, dtype: float64
```

In [23]:

```python
import matplotlib.pyplot as plt

# Forecast next 10 steps (adjust as needed)
forecast_steps = 10
forecast = model_fit.forecast(steps=forecast_steps)

# If your original data is indexed by time, create new index for forecast
forecast_index = range(len(data), len(data) + forecast_steps)

plt.figure(figsize=(10,6))

# Plot original series
plt.plot(data['Passengers'], label='Original')

# Plot forecast (make sure index aligns)
plt.plot(forecast_index, forecast, label='Forecast', color='red')

plt.title('ARIMA Forecast')
plt.legend()
plt.show()
```



In [25]:
```
import numpy as np
from sklearn.metrics import mean_squared_error
from statsmodels.tsa.arima.model import ARIMA

# Split data into train and test
train = data['Passengers'].iloc[:-12]
test = data['Passengers'].iloc[-12:]

# Fit model on training data
model = ARIMA(train, order=(2,1,2))
model_fit = model.fit()

# Forecast for test period
pred = model_fit.forecast(steps=12)

# Calculate RMSE between test and predictions
rmse = np.sqrt(mean_squared_error(test, pred))
print(f'RMSE: {rmse}')
```

```
RMSE: 55.22283859969251
```

In [ ]:

Practical 8: Text Analysis and Sentiment Detection

```
[15] #Step 1: Import Libraries
     import pandas as pd
     import nltk
     from nltk.corpus import stopwords
     from nltk.tokenize import word_tokenize
     from sklearn.feature_extraction.text import TfidfVectorizer
     from textblob import TextBlob
     from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
```

```
!pip install vaderSentiment
```

```
Collecting vaderSentiment
  Downloading vaderSentiment-3.3.2-py2.py3-none-any.whl.metadata (572 bytes)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from vaderSentiment) (2.32.3)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests->vaderSentiment) (3.4.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests->vaderSentiment) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests->vaderSentiment) (2.4.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests->vaderSentiment) (2025.4.26)
Downloading vaderSentiment-3.3.2-py2.py3-none-any.whl (125 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 126.0/126.0 kB 2.5 MB/s eta 0:00:00
Installing collected packages: vaderSentiment
Successfully installed vaderSentiment-3.3.2
```

```
[16] #Step 1: Import Libraries
     import pandas as pd
     import nltk
     from nltk.corpus import stopwords
     from nltk.tokenize import word_tokenize
     from sklearn.feature_extraction.text import TfidfVectorizer
     from textblob import TextBlob
     from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
```

```
[23] # Download NLTK data (only first time)
```

```
[23] # Download NLTK data (only first time)

     nltk.download('punkt')        # Tokenizer
     nltk.download('stopwords')    # Stopwords
     import nltk
     nltk.download('punkt_tab')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
True
```

```
#Step 2: Load Sample Text Data
#For demonstration, you can use a sample dataset of reviews, for example:
data = {'review': [
"I loved the movie! It was fantastic and thrilling.",
"The product is terrible, it broke after one use.",
"Not bad, but could be better.",
"Absolutely amazing! Highly recommend it.",
"Worst experience ever, will not buy again."
]}
df = pd.DataFrame(data)
```

```
[25] #Step 3: Text Preprocessing
     #Convert to lowercase
     #Tokenize text
     #Remove stopwords
     #(Optional) Stemming or Lemmatization
     stop_words = set(stopwords.words('english'))
     def preprocess_text(text):
       text = text.lower()
       words = word_tokenize(text)
       words = [w for w in words if w.isalpha()]  # Remove punctuation/numbers
       words = [w for w in words if w not in stop_words]
       return ' '.join(words)
     df['cleaned_review'] = df['review'].apply(preprocess_text)
     print(df[['review', 'cleaned_review']])
```

```
                                                   review  \
0  I loved the movie! It was fantastic and thrill...
1   The product is terrible, it broke after one use.
2                       Not bad, but could be better.
3             Absolutely amazing! Highly recommend it.
4           Worst experience ever, will not buy again.

                          cleaned_review
0       loved movie fantastic thrilling
1        product terrible broke one use
2                    bad could better
3  absolutely amazing highly recommend
4              worst experience ever buy
```

```
[26] #Step 4: Compute TF-IDF Features
tfidf = TfidfVectorizer()
tfidf_matrix = tfidf.fit_transform(df['cleaned_review'])
print("TF-IDF feature names:\n", tfidf.get_feature_names_out())
print("TF-IDF matrix shape:", tfidf_matrix.shape)
```

```
TF-IDF feature names:
 ['absolutely' 'amazing' 'bad' 'better' 'broke' 'buy' 'could' 'ever'
 'experience' 'fantastic' 'highly' 'loved' 'movie' 'one' 'product'
 'recommend' 'terrible' 'thrilling' 'use' 'worst']
TF-IDF matrix shape: (5, 20)
```

```
#Step 5: Perform Sentiment Analysis Using TextBlob:
def get_textblob_sentiment(text):
    analysis = TextBlob(text)
    polarity = analysis.sentiment.polarity
    if polarity > 0:
        return 'Positive'
    elif polarity == 0:
        return 'Neutral'
    else:
        return 'Negative'

# VADER Sentiment Function
analyzer = SentimentIntensityAnalyzer()

def get_vader_sentiment(text):
    score = analyzer.polarity_scores(text)
```

```
        return 'Neutral'
    else:
        return 'Negative'

# VADER Sentiment Function
analyzer = SentimentIntensityAnalyzer()

def get_vader_sentiment(text):
    score = analyzer.polarity_scores(text)
    compound = score['compound']
    if compound >= 0.05:
        return 'Positive'
    elif compound <= -0.05:
        return 'Negative'
    else:
        return 'Neutral'

# Apply both sentiment functions
df['sentiment_textblob'] = df['review'].apply(get_textblob_sentiment)
df['sentiment_vader'] = df['review'].apply(get_vader_sentiment)

# View Results
print(df[['review', 'sentiment_textblob', 'sentiment_vader']])
```

```
                                              review sentiment_textblob  \
0  I loved the movie! It was fantastic and thrill...           Positive
1   The product is terrible, it broke after one use.           Negative
2                       Not bad, but could be better.           Positive
3             Absolutely amazing! Highly recommend it.           Positive
4           Worst experience ever, will not buy again.           Negative

  sentiment_vader
0        Positive
1        Negative
2        Positive
3        Positive
4        Negative
```

Practical 9: Hadoop HDFS & MapReduce Word Count Program.

In [0]:
```
spark
```

SparkSession - hive

SparkContext

Spark UI

Version	v3.3.2
Master	local[8]
AppName	Databricks Shell

In [0]:
```
from pyspark.sql import SparkSession
```

In [0]:
```
#RDD ( READING THE RDD CONTENT )
from pyspark import SparkContext
spark = SparkSession.builder.getOrCreate()
sc = spark.sparkContext
data=[1,2,3,4,5,6,7,8,9]
rdd=sc.parallelize(data,4)
rdd.collect()
rdd.glom().collect()
```

```
Out[8]: [[1, 2], [3, 4], [5, 6], [7, 8, 9]]
```

In [0]:
```
#RDD TRANSFORMATIONS
#MAP Transformation

rdd = sc.parallelize([1,2,3,4])
result = rdd.map(lambda x:x*2)
print(result.collect())
```

```
[2, 4, 6, 8]
```

In [0]:
```
#Filter Transformation
rdd2 = sc.parallelize([1,2,3,4,5,6,])
result = rdd2.filter(lambda x:x%2 == 0)
print(result.collect())
```

```
[2, 4, 6]
```

In [0]:
```
#FLAT MAP TRANSFORMATION

rdd2 = sc.parallelize([1,2,3,])
result = rdd2.flatMap(lambda x:(x,x*2))
print(result.collect())
```

```
[1, 2, 2, 4, 3, 6]
```

In [0]:
```
#REDUCE BY TRANSFORMATION

rdd3 = sc.parallelize([('a',1),('b',2),('a',3)])
result = rdd3.reduceByKey(lambda x,y:x+y)
print(result.collect())
```

```
[('a', 4), ('b', 2)]
```

In [0]:
```
rdd4 = sc.parallelize([('a',1),('b',2),('a',3),('b',4)])
result = rdd4.reduceByKey(lambda x,y:x+y)
print(result.collect())
```

```
[('a', 4), ('b', 6)]
```

In [0]:
```
#GroupByKey Transformation

rdd4 = sc.parallelize([('a',1),('b',2),('a',3)])
result = rdd4.groupByKey().mapValues(list)
print(result.collect())
```

```
[('a', [1, 3]), ('b', [2])]
```

In [0]:
```
#Join Transformation

rdd5 = sc.parallelize([('a',1),('b',2)])
rdd6 = sc.parallelize([('a',3),('b',4)])
result = rdd5.join(rdd6)
print(result.collect())
```

```
[('b', (2, 4)), ('a', (1, 3))]
```

In [0]:
```
#Distinct Transformation (Unique elements , Not Repeated)

rdd7 = sc.parallelize([(1,2,3,2,1)])
result = rdd7.distinct()
print(result.collect())
```

```
[(1, 2, 3, 2, 1)]
```

In [0]:
```
#RDD ACTIONS
#Collect
rdd8 = sc.parallelize([1,2,3,4])
result = rdd.collect()
print(result)
```

```
[1, 2, 3, 4]
```

In [0]:
```
#Count

result = rdd8.count()
print(result)
```

```
4
```

In [0]:
```
#Reduce
result = rdd8.reduce(lambda x,y:x+y)
print(result)
```

```
10
```

In [0]:
```
#First
result = rdd8.first()
print(result)
```

```
1
```

In [0]:
```
#take(n)

result = rdd8.take(3)
print(result)
```

```
[1, 2, 3]
```

Practical 10: Building a Spark Application in PySpark

In [0]:
```
#PRACTICAL 10

#Create a DataFrame
data=[
    ("Alice",29,"Engineering"),
    ("Bob",35,"Sales"),
    ("Charlie",40,"HR"),
    ("David",30,"Engineering"),
    ("Eva",25,"Sales")
    ]
columns = ["name","age","department"]

df=spark.createDataFrame(data,columns)
df.display()
```

name	age	department
Alice	29	Engineering
Bob	35	Sales
Charlie	40	HR
David	30	Engineering
Eva	25	Sales

In [0]:
```
df.printSchema()
```

```
root
 |-- name: string (nullable = true)
 |-- age: long (nullable = true)
 |-- department: string (nullable = true)
```

In [0]:
```
df.show()
```

```
+-------+---+-----------+
|   name|age| department|
+-------+---+-----------+
|  Alice| 29|Engineering|
|    Bob| 35|      Sales|
|Charlie| 40|         HR|
|  David| 30|Engineering|
|    Eva| 25|      Sales|
+-------+---+-----------+
```

In [0]:
```
#Select Transformation
df.select("name","age").show()
```

```
+-------+---+
|   name|age|
+-------+---+
|  Alice| 29|
|    Bob| 35|
|Charlie| 40|
|  David| 30|
|    Eva| 25|
+-------+---+
```

In [0]:
```
#Filter Transformation
df.filter(df.age>30).show()
```

```
+-------+---+----------+
|   name|age|department|
+-------+---+----------+
|    Bob| 35|     Sales|
|Charlie| 40|        HR|
+-------+---+----------+
```

In [0]:
```
#Add a new Column
from pyspark.sql.functions import col
df.withColumn("age_plus_5",col("age")+5).show()
```

```
+-------+---+-----------+----------+
|   name|age| department|age_plus_5|
+-------+---+-----------+----------+
|  Alice| 29|Engineering|        34|
|    Bob| 35|      Sales|        40|
|Charlie| 40|         HR|        45|
|  David| 30|Engineering|        35|
|    Eva| 25|      Sales|        30|
+-------+---+-----------+----------+
```

In [0]:
```
#GroupBy & Aggregate

df.groupBy("department").avg("age").show()
```

```
+-----------+--------+
| department|avg(age)|
+-----------+--------+
|Engineering|    29.5|
|      Sales|    30.0|
|         HR|    40.0|
+-----------+--------+
```

In [0]:
```
#Sort Rows using orderBy

df.orderBy("age",ascending=False).show()
```

```
+-------+---+-----------+
|   name|age| department|
+-------+---+-----------+
|Charlie| 40|         HR|
|    Bob| 35|      Sales|
|  David| 30|Engineering|
|  Alice| 29|Engineering|
|    Eva| 25|      Sales|
+-------+---+-----------+
```

In [0]:
```
#Rename A Column
df.withColumnRenamed("age","employee_age").show()
```

```
+-------+------------+-----------+
|   name|employee_age| department|
+-------+------------+-----------+
|  Alice|          29|Engineering|
|    Bob|          35|      Sales|
|Charlie|          40|         HR|
|  David|          30|Engineering|
|    Eva|          25|      Sales|
+-------+------------+-----------+
```

In [0]:
```
#Drop a Column
df.drop("department").show()
```

```
+-------+---+
|   name|age|
+-------+---+
|  Alice| 29|
|    Bob| 35|
|Charlie| 40|
|  David| 30|
|    Eva| 25|
+-------+---+
```

In [0]:
```
#Changed Data Types
df.withColumn("age",col("age").cast("string")).printSchema()
```

```
root
 |-- name: string (nullable = true)
 |-- age: string (nullable = true)
 |-- department: string (nullable = true)
```

In [0]:
```
#Creating New Column based on the Condition
from pyspark.sql.functions import when
df.withColumn("senior",when(col("age")>30,"Yes").otherwise("No")).show()
```

```
+-------+---+-----------+------+
|   name|age| department|senior|
+-------+---+-----------+------+
|  Alice| 29|Engineering|    No|
|    Bob| 35|      Sales|   Yes|
|Charlie| 40|         HR|   Yes|
|  David| 30|Engineering|    No|
|    Eva| 25|      Sales|    No|
+-------+---+-----------+------+
```

In [0]:
```
#Distinct Values
df.select("department").distinct().show()
```

```
+-----------+
| department|
+-----------+
|Engineering|
|      Sales|
|         HR|
+-----------+
```

In [0]:
```

```