JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY HYDERABAD

UNIVERSITY COLLEGE OF ENGINEERING, SCIENCE AND TECHNOLOGY

KUKATPALLY, HYDERABAD – 500 085

CERTIFICATE

This is to certify that NISHITHA KOMMULA of B. Tech III year II Semester bearing the Hall-Ticket number 22011A6620 has fulfilled DATA ANALYTICS LAB record for the academic year 2024-25.

Signature of the Head of the Department **Signature of the staff member**

Date of Examination:_____

Internal Examiner **External Examiner**

TABLE OF CONTENTS

1. Data Preprocessing

a. Handling missing values

Code:

```
import pandas as pd
df=pd.read_csv('titanic.csv')
print(“Before removing missing values”)
df.info()
## print(df.isnull().sum())
df.isnull()
dropped_rows=df.dropna()
## print(dropped_rows.info())
dropped_cols=df.dropna(axis=1)
## print(dropped_cols.info())
cleaned_data=df.dropna(subset=['Age','Embarked'])
## print(cleaned_data.info())
cleaned_data=cleaned_data.drop(columns=['Cabin'])
print(“After removing missing values”)
print(cleaned_data.info())
```

Output:

```
Before removing misssing values:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
After removing misssing values:
<class 'pandas.core.frame.DataFrame'>
Index: 712 entries, 0 to 890
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  712 non-null    int64
 1   Survived     712 non-null    int64
 2   Pclass       712 non-null    int64
 3   Name         712 non-null    object
 4   Sex          712 non-null    object
 5   Age          712 non-null    float64
 6   SibSp        712 non-null    int64
 7   Parch        712 non-null    int64
 8   Ticket       712 non-null    object
 9   Fare         712 non-null    float64
 10  Embarked     712 non-null    object
dtypes: float64(2), int64(5), object(4)
memory usage: 66.8+ KB
None
```

b. Noise detection removal

Code:

```
Import pandas as pd

df=pd.read_csv('titanic.csv')

print(df.shape)

df=df[df['Age']>=0 ]

df=df[df['Age']<=100]

print(df.shape)

q1=df['Fare'].quantile(0.25)

q3=df['Fare'].quantile(0.75)

iqr=q3-q1

lower=q1-1.5*iqr

upper=q3+1.5*iqr

df=df[(df['Fare']>=lower) & (df['Fare']<=upper)]
```

print(df.shape)

Output:

```
(891, 12)
(714, 12)
(620, 12)
```

c. Identifying data redundancy and elimination

code:

```
df=pd.read_csv('titanic.csv')
duplicates=df[df.duplicated()]
print( f"duplicate rows:\n {duplicates}")
df=df.drop_duplicates()
print(df.shape)
```

Output:

```
duplicate rows:
 Empty DataFrame
Columns: [PassengerId, Survived, Pclass, Name, Sex, Age, SibSp, Parch, Ticket, Fare, Cabin, Embarked]
Index: []
(891, 12)
```

---

2. Implement any one imputation model

Code:

```
import pandas as pd
import numpy as np
from sklearn.impute import SimpleImputer
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eva'],
    'Age': [25, np.nan, 30, np.nan, 22],
    'Salary': [50000, 60000, 55000, 52000, 58000]
}
```

```
df = pd.DataFrame(data)

print("Before imputation:")

print(df)

imputer = SimpleImputer(strategy='mean')

df['Age'] = imputer.fit_transform(df[['Age']])

print("\nAfter mean imputation:")

print(df)
```

Output:

```
Before imputation:
      Name   Age  Salary
0    Alice  25.0   50000
1      Bob   NaN   60000
2  Charlie  30.0   55000
3    David   NaN   52000
4      Eva  22.0   58000

After mean imputation:
      Name        Age  Salary
0    Alice  25.000000   50000
1      Bob  25.666667   60000
2  Charlie  30.000000   55000
3    David  25.666667   52000
4      Eva  22.000000   58000
```

3. Implement Linear Regression

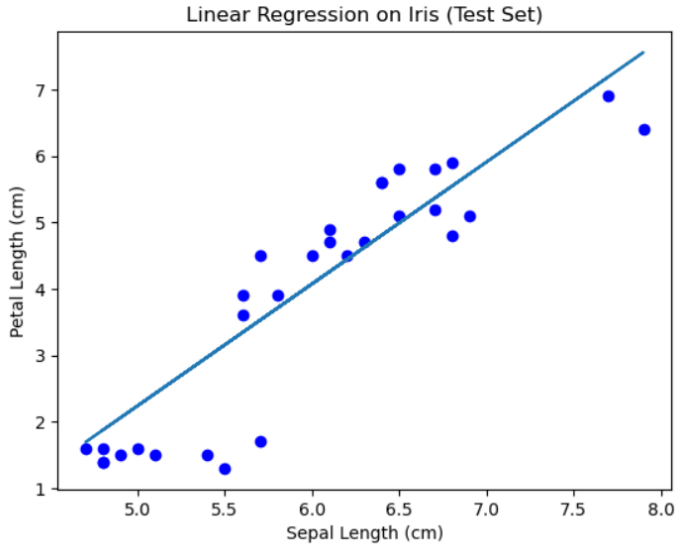
Code:

```
import pandas as pd

import matplotlib.pyplot as plt

from sklearn.datasets import load_iris

from sklearn.linear_model import LinearRegression

from sklearn.model_selection import train_test_split

from sklearn.metrics import mean_squared_error, r2_score
```

```
iris = load_iris()

df = pd.DataFrame(data=iris.data, columns=iris.feature_names)

X = df[['sepal length (cm)']]

y = df['petal length (cm)']

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(f"Mean Squared Error: {mean_squared_error(y_test, y_pred):.2f}")

print(f"R² Score: {r2_score(y_test, y_pred):.2f}")

plt.scatter(X_test, y_test, color='blue', label='Actual')

plt.plot(X_test, y_pred)

plt.xlabel('Sepal Length (cm)')

plt.ylabel('Petal Length (cm)')

plt.title('Linear Regression on Iris (Test Set)')
```

Output:

```
Mean Squared Error: 0.60
R² Score: 0.82
Text(0.5, 1.0, 'Linear Regression on Iris (Test Set)')
```



4.Implement Logistic Regression

Code:

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
iris = load_iris()
X = iris.data
y = iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = LogisticRegression(max_iter=200)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

Output:

```
Accuracy: 1.0

Confusion Matrix:
 [[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

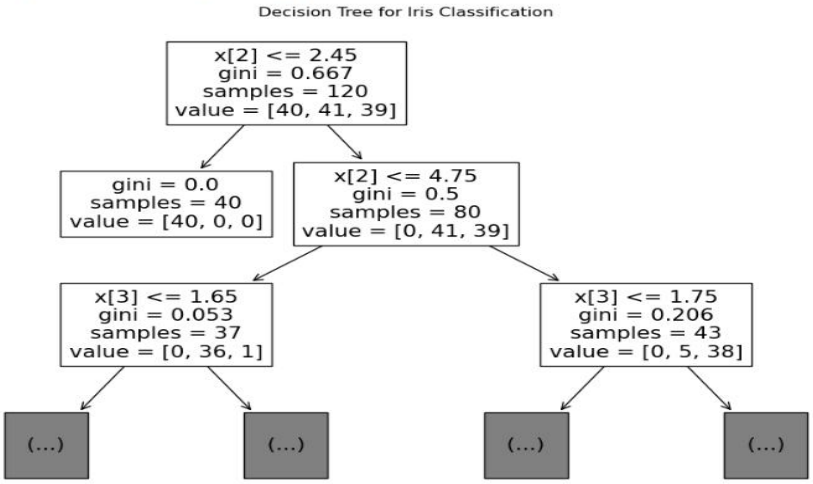
5.Implement Decision Tree Induction for classification

Code:

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
import matplotlib.pyplot as plt
iris = load_iris()
X = iris.data
y = iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
dtree = DecisionTreeClassifier(random_state=42)
dtree.fit(X_train, y_train)
y_pred = dtree.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
plt.figure(figsize=(12,8))
plot_tree(dtree,max_depth=3)
plt.title("Decision Tree for Iris Classification")
```

Output:

```
Accuracy: 1.0

Confusion Matrix:
 [[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30

]: Text(0.5, 1.0, 'Decision Tree for Iris Classification')
```



6. Implement Random Forest Classifier

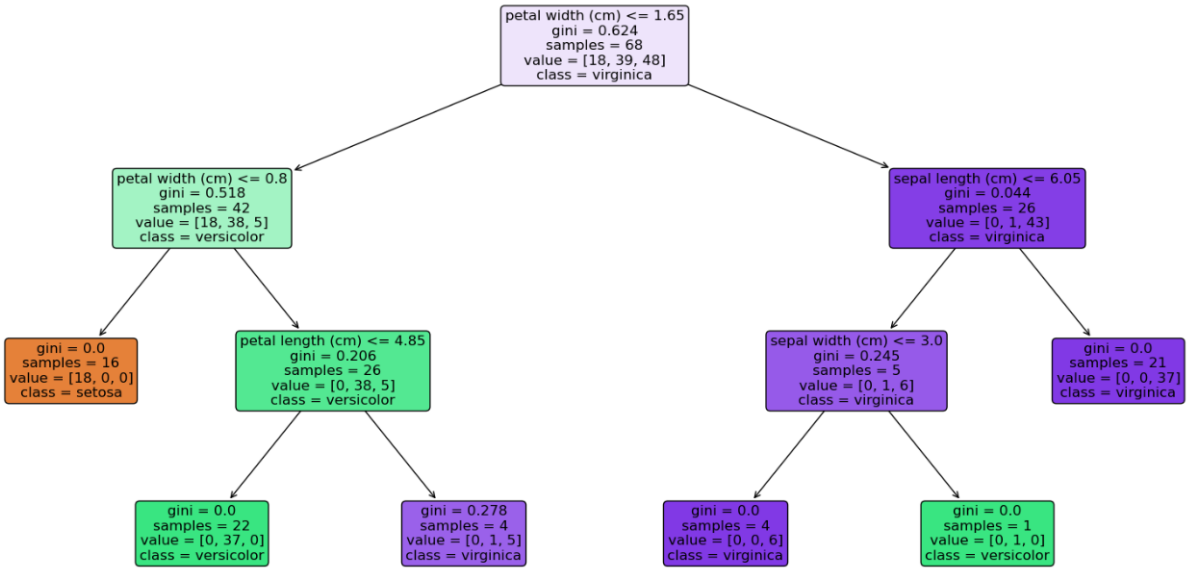
Code:

```
from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy_score, classification_report

from sklearn.tree import plot_tree

import matplotlib.pyplot as plt

iris = load_iris()

X = iris.data

y = iris.target

X_train, X_test, y_train, y_test = train_test_split(
```

```
    X, y, test_size=0.3, random_state=42
)
rf_model = RandomForestClassifier(n_estimators=100, max_depth=3, random_state=42)
rf_model.fit(X_train, y_train)
y_pred = rf_model.predict(X_test)
print("Random Forest Classifier Results (max_depth=3)")
print("-----------------------------------------------")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred, target_names=iris.target_names))
plt.figure(figsize=(20, 10))
plot_tree(
    rf_model.estimators_[0],
    feature_names=iris.feature_names,
    class_names=iris.target_names,
    filled=True,
    rounded=True,
    fontsize=12
)
plt.title("First Decision Tree in the Random Forest (max_depth=3)")
plt.show()
```

Output:

```
Random Forest Classifier Results (max_depth=3)
------------------------------------------------
Accuracy: 1.0

Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        19
  versicolor       1.00      1.00      1.00        13
   virginica       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
```



7.Implement ARIMA on Time Series data

Code:

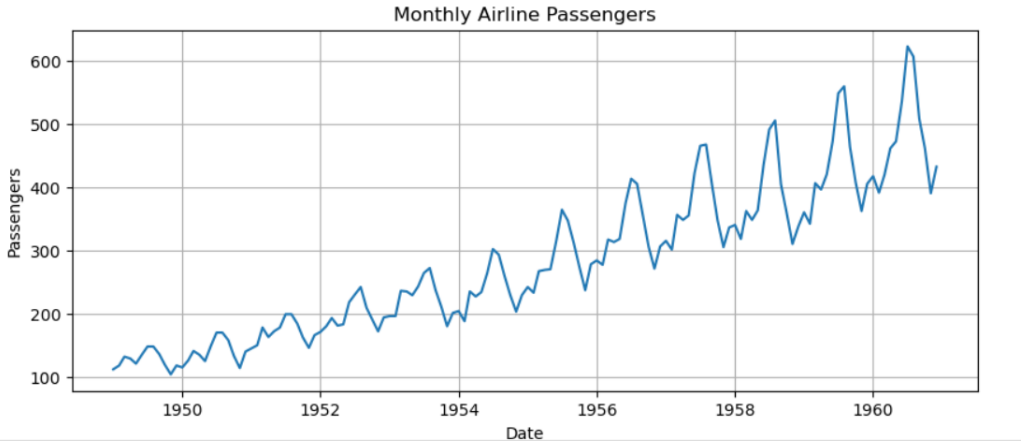
```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from statsmodels.tsa.arima.model import ARIMA

from statsmodels.tsa.stattools import adfuller

from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

from sklearn.metrics import mean_squared_error, mean_absolute_error,
mean_absolute_percentage_error

import warnings

warnings.filterwarnings("ignore")

url = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/airline-passengers.csv'

data = pd.read_csv(url, index_col='Month', parse_dates=True)

data = data['Passengers']

print(data)

plt.figure(figsize=(10, 4))

plt.plot(data)

plt.title('Monthly Airline Passengers')

plt.xlabel('Date')

plt.ylabel('Passengers')

plt.grid(True)

plt.show()

result = adfuller(data)

print("ADF Statistic:", result[0])

print("p-value:", result[1])
```

```
data_diff = data.diff().dropna()
plt.figure(figsize=(10, 4))
plt.plot(data_diff)
plt.title('First Order Differenced Data')
plt.grid(True)
plt.show()
result = adfuller(data_diff)
print("ADF Statistic after first differencing:", result[0])
print("p-value:", result[1])
data_diff = data.diff().diff().dropna()
plt.figure(figsize=(10, 4))
plt.plot(data_diff)
plt.title('Second Order Differenced Data')
plt.grid(True)
plt.show()
result = adfuller(data_diff)
print("ADF Statistic after second differencing:", result[0])
print("p-value:", result[1])
plot_acf(data_diff, lags=20)
plot_pacf(data_diff, lags=20)
plt.show()
train_size = int(len(data) * 0.8)
train = data[:train_size]
test = data[train_size:]
print(f"Train size: {len(train)}, Test size: {len(test)}")
model = ARIMA(train, order=(11, 2, 13))
```

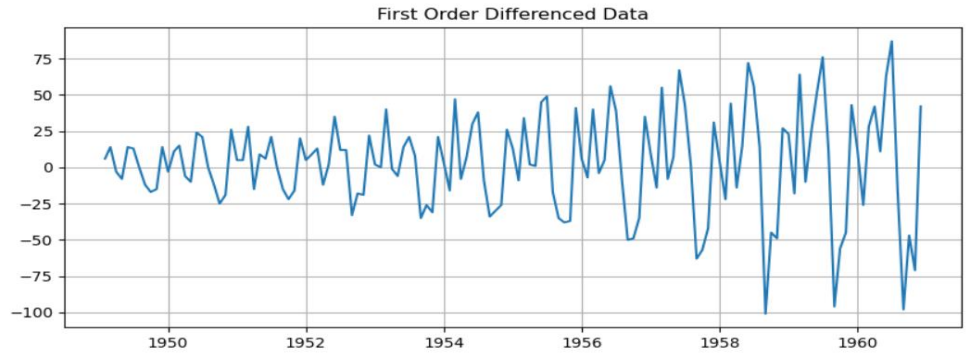
```
model_fit = model.fit()
print(model_fit.summary())
forecast = model_fit.forecast(steps=len(test))
print("Forecasted Values:\n", forecast)
plt.figure(figsize=(10, 4))
plt.plot(train, label='Train')
plt.plot(test, label='Test')
plt.plot(test.index, forecast, label='Forecast', color='red')
plt.title('ARIMA Forecast vs Actual')
plt.xlabel('Date')
plt.ylabel('Passengers')
plt.legend()
plt.grid(True)
plt.show()
mse = mean_squared_error(test, forecast)
rmse = np.sqrt(mse)
mae = mean_absolute_error(test, forecast)
mape = mean_absolute_percentage_error(test, forecast) * 100
print("\nAccuracy Metrics:")
print(f"MSE  (Mean Squared Error):      {mse:.2f}")
print(f"RMSE (Root Mean Squared Error): {rmse:.2f}")
print(f"MAE  (Mean Absolute Error):     {mae:.2f}")
print(f"MAPE (Mean Absolute % Error):   {mape:.2f}%")
```

Output:

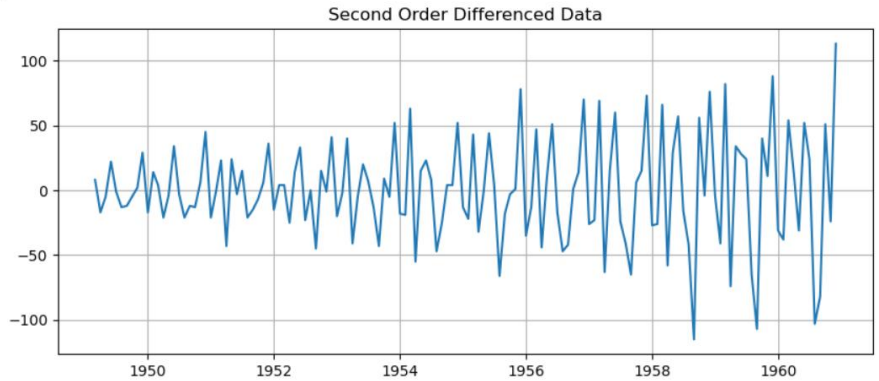
```
Month
1949-01-01    112
1949-02-01    118
1949-03-01    132
1949-04-01    129
1949-05-01    121
              ...
1960-08-01    606
1960-09-01    508
1960-10-01    461
1960-11-01    390
1960-12-01    432
Name: Passengers, Length: 144, dtype: int64
```

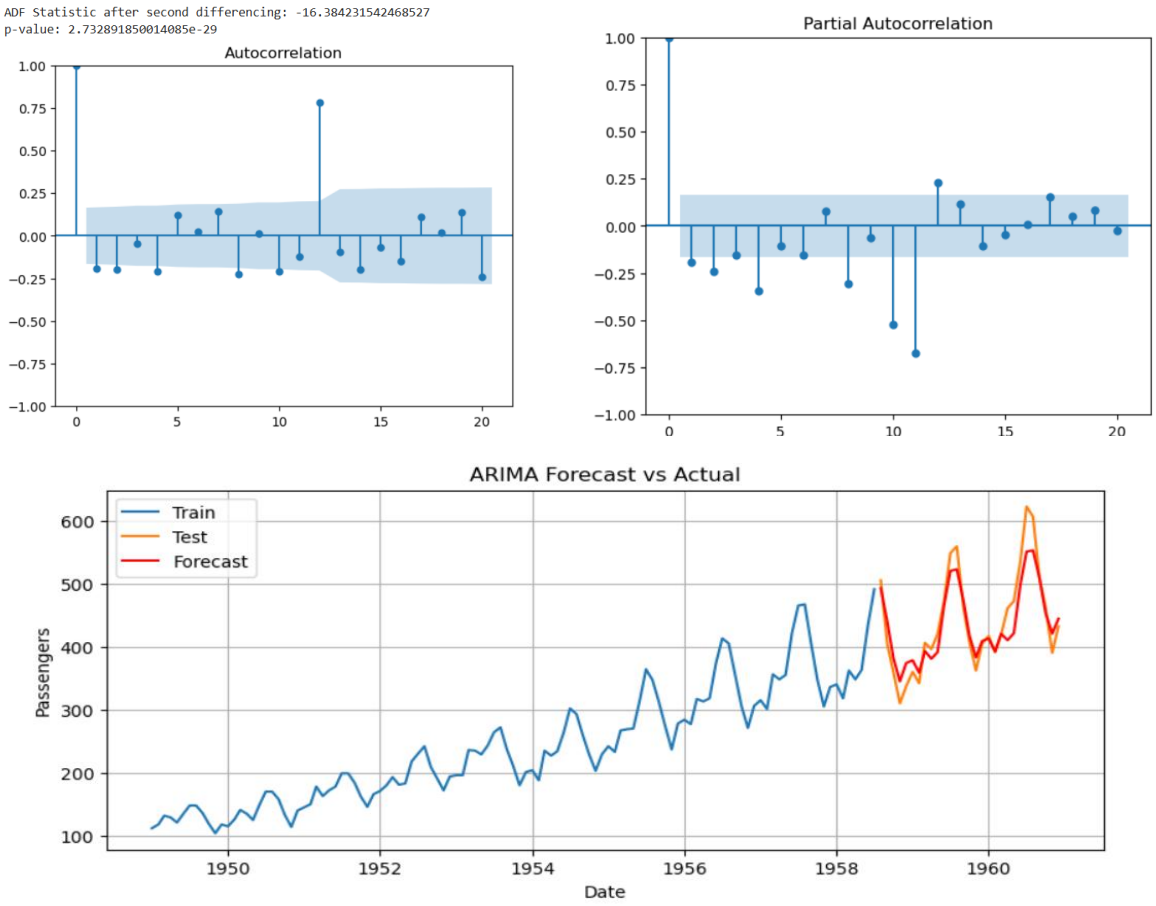



```
ADF Statistic: 0.8153688792060597
p-value: 0.9918802434376411
```




```
ADF Statistic after first differencing: -2.829266824169992
p-value: 0.0542132902838265
```






```
Accuracy Metrics:
MSE  (Mean Squared Error):       857.66
RMSE (Root Mean Squared Error): 29.29
MAE  (Mean Absolute Error):     23.23
MAPE (Mean Absolute % Error):   5.22%
```

8. Object segmentation using hierarchical based methods

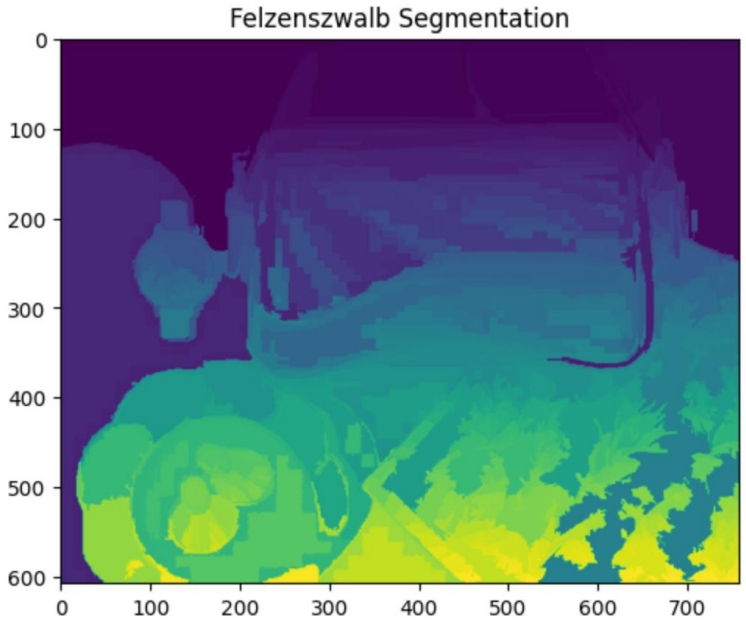
Code:

```
from skimage.segmentation import felzenszwalb
from skimage.io import imread
from skimage.color import rgb2gray
import matplotlib.pyplot as plt
image = imread('bag.jpg')
segments = felzenszwalb(image, scale=100, sigma=0.5, min_size=50)
plt.imshow(segments)
```

plt.title("Felzenszwalb Segmentation")

plt.show()

Output:



9.Perform Visualization techniques (types of maps - Bar, Colum, Line, Scatter, 3D Cubes etc)

Code:

1.Bar chart

import matplotlib.pyplot as plt

x = ['A', 'B', 'C', 'D', 'E']

y = [23, 45, 56, 78, 12]

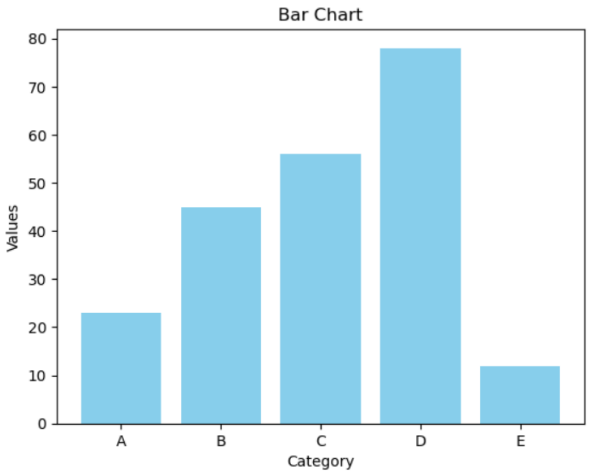
plt.bar(x, y, color='skyblue')

plt.title('Bar Chart')

plt.xlabel('Category')

plt.ylabel('Values')

Output:



2. Column Chart

plt.barh(x, y)

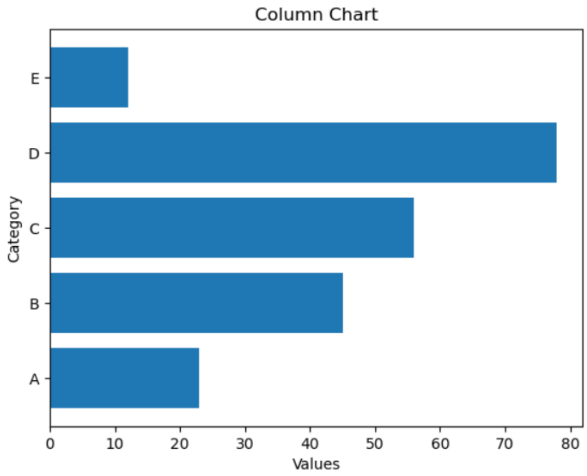
plt.title('Column Chart')

plt.xlabel('Values')

plt.ylabel('Category')

plt.show()

Output:



3.Line Chart

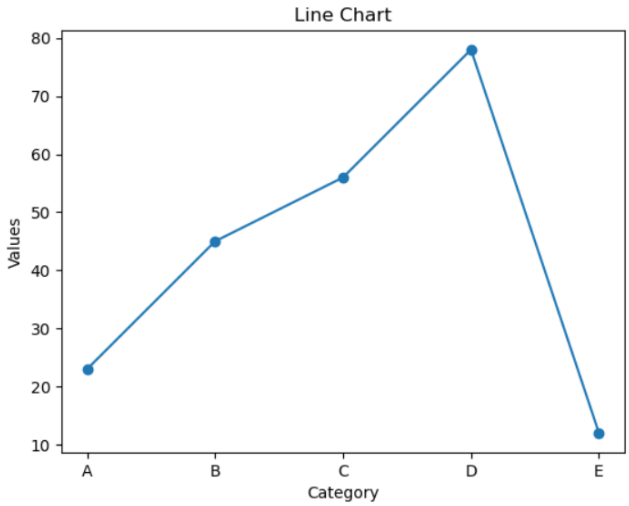
plt.plot(x, y, marker='o')

plt.title('Line Chart')

plt.xlabel('Category')

plt.ylabel('Values')

Output:



4. Scatter Plot

Code:

plt.scatter(x, y)

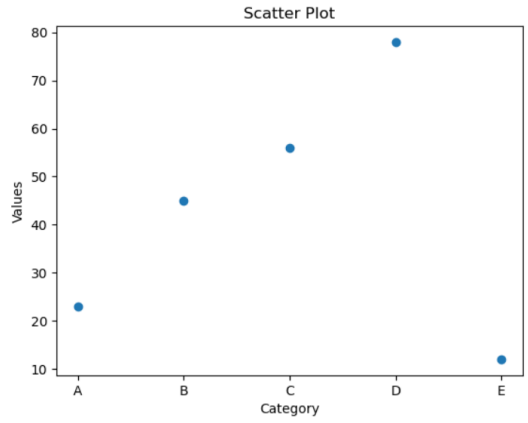
plt.title('Scatter Plot')

plt.xlabel('Category')

plt.ylabel('Values')

plt.show()

Output:

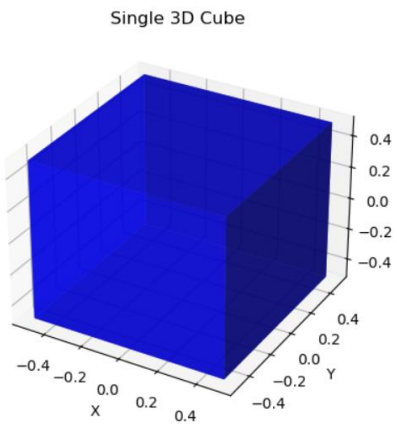


3D cubes:

Code:

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
cube_size = 1
x, y, z = 0, 0, 0
ax.bar3d(
    x - cube_size/2, y - cube_size/2, z - cube_size/2,
    cube_size, cube_size, cube_size,
    color='b', alpha=0.7
)
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
ax.set_title('Single 3D Cube')
plt.show()
```

Output:



10.perform Descriptive analytics on healthcare Data.

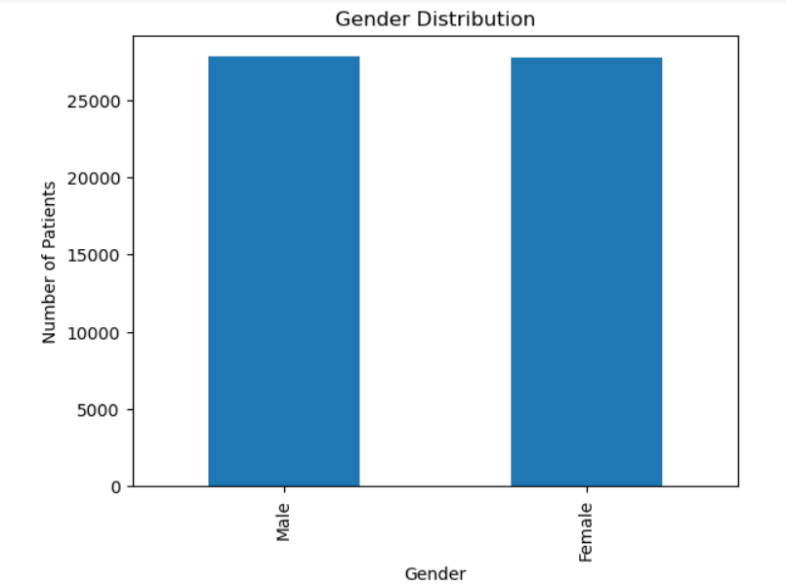
Code:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
df = pd.read_csv("C:\\Users\\user\\Downloads\\archive (4)\\healthcare_dataset.csv")
print("First 5 rows:\n", df.head())
print("\nDataset Info:\n")
print(df.info())
print("\nMissing Values:\n", df.isnull().sum())
print("\nDescriptive Statistics:\n", df.describe())
print("\nGender Distribution:\n", df['Gender'].value_counts())
print("\nMedical Condition Count:\n", df['Medical Condition'].value_counts())
print("\nAdmission Type Count:\n", df['Admission Type'].value_counts())
print("\nTest Results Count:\n", df['Test Results'].value_counts())
df['Gender'].value_counts().plot(kind='bar', title='Gender Distribution')
plt.xlabel("Gender")
plt.ylabel("Number of Patients")
plt.show()
df['Admission Type'].value_counts().plot(kind='bar', title='Admission Type Count',
color='orange')
plt.xlabel("Admission Type")
plt.ylabel("Number of Admissions")
plt.show()
avg_billing = df.groupby('Medical Condition')['Billing Amount'].mean()
print("\nAverage Billing by Condition:\n", avg_billing)
```

```
avg_billing.plot(kind='bar', title='Average Billing by Medical Condition', color='green')

plt.ylabel("Avg Billing Amount")

plt.show()

sns.countplot(x='Test Results', data=df)

plt.title("Test Results Distribution")

plt.show()
```

OUTPUT:

```
First 5 rows:
              Name  Age  Gender Blood Type Medical Condition Date of Admission  \
0  Bobby JacksOn    30    Male         B-            Cancer        2024-01-31
1   LesLie TErRy    62    Male         A+           Obesity        2019-08-20
2    DaNnY sMitH    76  Female         A-           Obesity        2022-09-22
3   andrEw waTtS    28  Female         O+          Diabetes        2020-11-18
4  adrIENNE bEll    43  Female        AB+            Cancer        2022-09-19

             Doctor                     Hospital Insurance Provider  \
0     Matthew Smith             Sons and Miller         Blue Cross
1   Samantha Davies                      Kim Inc           Medicare
2  Tiffany Mitchell                     Cook PLC              Aetna
3       Kevin Wells  Hernandez Rogers and Vang,           Medicare
4    Kathleen Hanna                 White-White              Aetna

   Billing Amount  Room Number Admission Type Discharge Date   Medication  \
0    18856.281306          328         Urgent     2024-02-02  Paracetamol
1    33643.327287          265      Emergency     2019-08-26    Ibuprofen
```



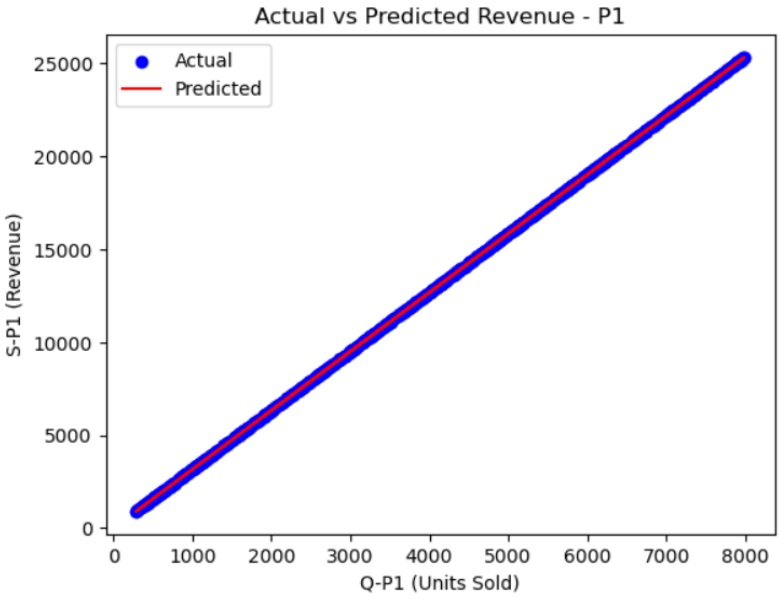
11.perform Predective analytics on Product Sales Data

Code:

```
import pandas as pd
df = pd.read_csv("statsfinal.csv")
print(df.head())
print(df.info())
print(df.describe())
print(df.isnull().sum())
df.drop(columns=['Unnamed: 0'], inplace=True)
X = df[['Q-P1']]
y = df['S-P1']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model_p1 = LinearRegression()
model_p1.fit(X_train, y_train)
y_pred = model_p1.predict(X_test)
print(" Product P1 - R²:", r2_score(y_test, y_pred))
print(" Product P1 - MSE:", mean_squared_error(y_test, y_pred))
plt.scatter(X_test, y_test, label='Actual', color='blue')
plt.plot(X_test, y_pred, label='Predicted', color='red')
plt.title("Actual vs Predicted Revenue - P1")
plt.xlabel("Q-P1 (Units Sold)")
plt.ylabel("S-P1 (Revenue)")
plt.legend()
plt.show()
```

Output:

```
Product P1 - R²: 1.0
Product P1 - MSE: 5.777395785135145e-24
```



12.Apply Predictive analytics for wheather forecasting.

Code:

```
import pandas as pd
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
data = {
    'Date': ['2025-06-01', '2025-06-02', '2025-06-03', '2025-06-04', '2025-06-05', '2025-06-06', '2025-06-07'],
    'Temperature': [32, 34, 33, 35, 36, 37, 38],
    'Humidity': [60, 55, 58, 53, 50, 48, 45],
    'WindSpeed': [12, 10, 11, 9, 8, 7, 6]
}
df = pd.DataFrame(data)
X = df[['Humidity', 'WindSpeed']]
y = df['Temperature']
```

```
model = LinearRegression()
model.fit(X, y)
df['Predicted_Temp'] = model.predict(X)
next_day_features = [[43, 5]]  # Replace with actual forecasted humidity/wind speed
next_day_prediction = model.predict(next_day_features)[0]
df.loc[len(df)] = ['2025-06-08', None, 43, 5, next_day_prediction]
print(df)
plt.plot(df['Date'][:-1], df['Temperature'][:-1], label='Actual', marker='o')  # Skip None
plt.plot(df['Date'], df['Predicted_Temp'], label='Predicted', marker='x', linestyle='--')
plt.xlabel('Date')
plt.ylabel('Temperature')
plt.title('Weather Forecast - Actual vs Predicted Temperature')
plt.legend()
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

Output: