Ex.no:	Data Pre-processing using Pandas
Date:	

AIM:

To Load Real Time data Set and Python Libraries, Installing Libraries through Anaconda Prompt, Perform data pre-processing through Pandas Library.

ALGORITHM:

Step 1: Start the code
Step 2: Open a Jupyter Notebook
Step 3: Install the needed python packages using Pip install package name.
Step 4: Import the needed python packages using Import package name.
import pandas as pd, import numpy as np, import matplotlib.pyplot as plt.
Step 5: Import the data set using pandas package.

```
df = pd.read_csv("D:\Jeyashri\IBM\Datasets\Social_Network_Ads.csv")
df
```

Step 6: Drop the null values by using pandas method.

```
df = df.dropna()
df
```

Step 7: Clear the Duplicate data by using pandas method.

```
df = df.drop_duplicates()
df
```

Step 8: Create new dummies values to process the data based on column.

```
df = pd.get_dummies(df, columns=['Gender'])
df
```

Step 10: Stop the process.

PROGRAM:

```
import pandas as pd
#Uploading data into dataframe
df = pd.read_csv("D:\Jeyashri\IBM\Datasets\Social_Network_Ads.csv")
df
# Fill missing values with a specific value
df = df.fillna(values)
df
# Drop rows with missing values
df = df.dropna()
df
# Interpolate missing values
df = df.interpolate()
df
# Remove duplicate rows
df = df.drop_duplicates()
df
# One-hot encoding categorical columns
df = pd.get_dummies(df, columns=['Gender'])
df
```

OUTPUT:

Out[7]:

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19.0	19000	NaN
1	15810944	Male	35.0	20000	0.0
2	15668575	Female	26.0	43000	0.0
3	15603246	Female	27.0	57000	0.0
4	15804002	Male	19.0	76000	0.0
...	...	...	...	...	...
395	15691863	Female	46.0	41000	1.0
396	15706071	Male	51.0	23000	1.0
397	15654296	Female	50.0	20000	1.0
398	15755018	Male	36.0	33000	0.0
399	15594041	Female	49.0	36000	1.0

400 rows × 5 columns

Out[8]:

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19.0	19000	2.0
1	15810944	Male	35.0	20000	0.0
2	15668575	Female	26.0	43000	0.0
3	15603246	Female	27.0	57000	0.0
4	15804002	Male	19.0	76000	0.0
...	...	...	...	...	...
395	15691863	Female	46.0	41000	1.0
396	15706071	Male	51.0	23000	1.0
397	15654296	Female	50.0	20000	1.0
398	15755018	Male	36.0	33000	0.0
399	15594041	Female	49.0	36000	1.0

400 rows × 5 columns

Out[9]:

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19.0	19000	2.0
1	15810944	Male	35.0	20000	0.0
2	15668575	Female	26.0	43000	0.0
3	15603246	Female	27.0	57000	0.0
4	15804002	Male	19.0	76000	0.0
...	...	...	...	...	...
395	15691863	Female	46.0	41000	1.0
396	15706071	Male	51.0	23000	1.0
397	15654296	Female	50.0	20000	1.0
398	15755018	Male	36.0	33000	0.0
399	15594041	Female	49.0	36000	1.0

400 rows × 5 columns

Out[10]:

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19.0	19000	2.0
1	15810944	Male	35.0	20000	0.0
2	15668575	Female	26.0	43000	0.0
3	15603246	Female	27.0	57000	0.0
4	15804002	Male	19.0	76000	0.0
...	...	...	...	...	...
395	15691863	Female	46.0	41000	1.0
396	15706071	Male	51.0	23000	1.0
397	15654296	Female	50.0	20000	1.0
398	15755018	Male	36.0	33000	0.0
399	15594041	Female	49.0	36000	1.0

400 rows × 5 columns

Out[11]:

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19.0	19000	2.0
1	15810944	Male	35.0	20000	0.0
2	15668575	Female	26.0	43000	0.0
3	15603246	Female	27.0	57000	0.0
4	15804002	Male	19.0	76000	0.0
...	...	...	...	...	...
395	15691863	Female	46.0	41000	1.0
396	15706071	Male	51.0	23000	1.0
397	15654296	Female	50.0	20000	1.0
398	15755018	Male	36.0	33000	0.0
399	15594041	Female	49.0	36000	1.0

400 rows × 5 columns

Out[12]:

	User ID	Age	EstimatedSalary	Purchased	Gender_2	Gender_Female	Gender_Male
0	15624510	19.0	19000	2.0	0	0	1
1	15810944	35.0	20000	0.0	0	0	1
2	15668575	26.0	43000	0.0	0	1	0
3	15603246	27.0	57000	0.0	0	1	0
4	15804002	19.0	76000	0.0	0	0	1
...	...	...	...	...	...	...	...
395	15691863	46.0	41000	1.0	0	1	0
396	15706071	51.0	23000	1.0	0	0	1
397	15654296	50.0	20000	1.0	0	1	0
398	15755018	36.0	33000	0.0	0	0	1
399	15594041	49.0	36000	1.0	0	1	0

400 rows × 7 columns

RESULT:

Hence the data pre-process successfully.

Ex.no:	**Bayesian network**
Date:	

AIM:

To Construct a Bayesian network considering student data.

ALGORITHM:

Step 1: Start the code
Step 2: Open a Jupyter Notebook
Step 3: Install the needed python packages using Pip install package name.
Step 4: Import the needed python packages using Import package name.
import pandas as pd, import numpy as np, import matplotlib.pyplot as plt.
from sklearn.naive_bayes import GaussianNB
Step 5: Import a data set by using pandas
dataset = pd.read_csv("results.csv")
Step 6: Select the x and y data points with the help of loc() methods.
X = data.loc[:, ["Hours", "StudentId"]] # Features
y = data["Result"] # Target
new=pd.DataFrame(X,y)
Step 7: Split the dataset into train and test for training and testing the machine.
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 0)
Step 8: Use GaussianNB Algorithm to fit and predict the model
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(X_train, y_train)
Step 9: Finally Evaluate the result using metrics ()
from sklearn.metrics import confusion_matrix,accuracy_score
cm = confusion_matrix(y_test, y_pred)
ac = accuracy_score(y_test,y_pred)
cr= classification_report(y_test, y_pred))
print(cm) print(ac) print(cr)
Step 10: Visualise the confusion-matrix by using Seaborn package
Step 11: Stop the Implementation

PROGRAM:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
# Load data from CSV file
data = pd.read_csv("results.csv")
print(data)
# Assuming the last column is the target column and rest are features
X = data.loc[:, ["Hours", "StudentId"]]  # Features
y = data["Result"]  # Target
new=pd.DataFrame(X,y)
print(new)
# Split data into train and test sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Initialize Naive Bayes classifier
classifier = GaussianNB()
classifier.fit(X_train, y_train)
# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
# Classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))
# Calculate false percentage
false_percentage = (1 - accuracy) * 100
print("False Percentage:", false_percentage)
new=pd.DataFrame(X_test,y_pred)
print(new)
conf_matrix = confusion_matrix(y_test, y_pred)
conf_matrix
# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", cbar=False)
plt.title("Confusion Matrix")
plt.xlabel("Predicted Labels")
```

plt.ylabel("True Labels")
plt.show()

OUTPUT:

```
    Hours  StudentId  Result
0       1         10       0
1       1         15       0
2       1         21       0
3       1         16       0
4       2         14       0
5       2          5       1
6       2          7       0
7       2          2       1
8       2         17       0
9       3         18       1
10      3          6       1
11      3         20       1
```

```
        Hours  StudentId
Result
0           1         10
0           1         10
0           1         10
0           1         10
0           1         10
1           1         15
0           1         10
1           1         15
0           1         10
1           1         15
1           1         15
1           1         15
```

```
Out[6]: GaussianNB()
```

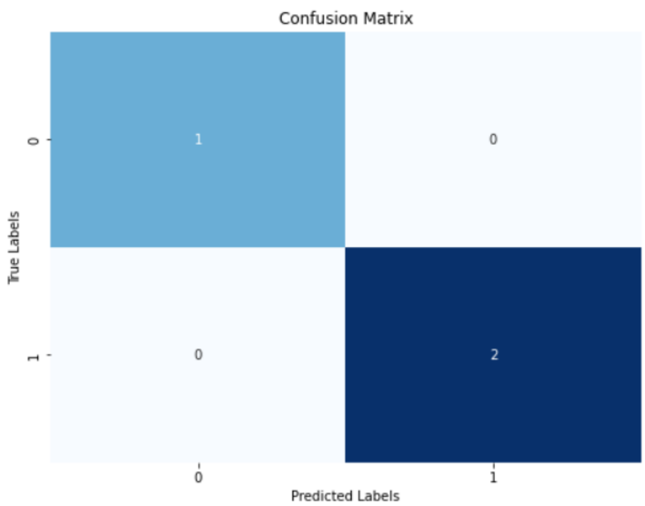
```
[1 1 0]
```

```
Accuracy: 1.0
```

```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         1
           1       1.00      1.00      1.00         2

    accuracy                           1.00         3
   macro avg       1.00      1.00      1.00         3
weighted avg       1.00      1.00      1.00         3
```

```
   Hours  StudentId
1    NaN        NaN
1    NaN        NaN
0    1.0       10.0
```

```
Out[61]: array([[1, 0],
                [0, 2]], dtype=int64)
```



RESULT:

Hence the Student dataset evaluated successfully with help of Bayesian network

Ex.no:	**K-Means Clustering**
Date:	

AIM:

To Implement a K-Means Clustering.

ALGORITHM:

Step 1: Start the code
Step 2: Open a Jupyter Notebook
Step 3: Install the needed python packages using Pip install package name.
Step 4: Import the needed python packages using Import package name.
import pandas as pd, import numpy as np, import matplotlib.pyplot as plt.
from sklearn.cluster import KMeans
Step 5: Import a data set by using pandas
data = pd.read_csv("D:\\Jeyashri\\IBM\\Datasets\\Country clusters.csv")
data
Step 6: Scatter the data point to find out the cluster formation by using matplot.pyplot package.
Step 7: Select the x data points with the help of iloc() methods
x = data.iloc[:,1:3]
x
Step 8: Select the model and define number of cluster to be form.
kmeans = KMeans(3)
kmeans.fit(x)
Step 9: Use the fit_predict() method to perform both the fit and predict in a single line. And store the X value in Identified_clusters variable.
identified_clusters = kmeans.fit_predict(x)
identified_clusters
Step 10: Make a copy on the data set using copy () method and for a cluster.
Step 11: Stop the code

PROGRAM:

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

data = pd.read_csv("D:\\Jeyashri\\IBM\\Datasets\\Country clusters.csv")

data

plt.scatter(data['Longitude'],data['Latitude'])

plt.xlim(-180,180)

plt.ylim(-90,90)

plt.xlabel("Longitude")

plt.ylabel("Latitude")

plt.show()

x = data.iloc[:,1:3]

x

kmeans = KMeans(3)

kmeans.fit(x)

identified_clusters = kmeans.fit_predict(x)

identified_clusters

data_with_clusters = data.copy()

data_with_clusters['Clusters'] = identified_clusters

plt.scatter(data_with_clusters['Longitude'],data_with_clusters['Latitude'],c=data
_with_clusters['Clusters'],cmap='rainbow')
```

OUTPUT:

	Date	Symbol	Series	Prev Close	Open	High	Low	Last	Close	VWAP	Volume	Turnover	Trades	Deliverable Volume	%Deliverble
0	2000-01-03	HDFCBANK	EQ	157.40	166.00	170.00	166.00	170.00	170.00	169.52	33259	5.638122e+11	NaN	NaN	NaN
1	2000-01-04	HDFCBANK	EQ	170.00	182.00	183.45	171.00	174.00	173.80	174.99	168710	2.952261e+12	NaN	NaN	NaN
2	2000-01-05	HDFCBANK	EQ	173.80	170.00	173.90	165.00	168.00	166.95	169.20	159820	2.704094e+12	NaN	NaN	NaN
3	2000-01-06	HDFCBANK	EQ	166.95	168.00	170.00	165.30	168.95	168.30	168.44	85026	1.432166e+12	NaN	NaN	NaN
4	2000-01-07	HDFCBANK	EQ	168.30	162.15	171.00	162.15	170.75	168.35	166.79	85144	1.420158e+12	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
5301	2021-04-26	HDFCBANK	EQ	1414.15	1413.00	1429.00	1402.75	1407.55	1404.80	1413.19	15085476	2.131861e+15	291268.0	9791881.0	0.6491
5302	2021-04-27	HDFCBANK	EQ	1404.80	1407.25	1442.00	1404.80	1435.05	1438.70	1430.40	10296453	1.472810e+15	233200.0	5650216.0	0.5488
5303	2021-04-28	HDFCBANK	EQ	1438.70	1436.25	1479.00	1431.00	1475.00	1476.80	1463.19	12051970	1.763438e+15	197146.0	7196647.0	0.5971
5304	2021-04-29	HDFCBANK	EQ	1476.80	1486.20	1503.65	1461.00	1471.65	1472.50	1481.15	12039276	1.783196e+15	252296.0	4818551.0	0.4002
5305	2021-04-30	HDFCBANK	EQ	1472.50	1445.00	1453.80	1407.50	1412.90	1412.30	1421.13	17616451	2.503529e+15	447876.0	8982938.0	0.5099

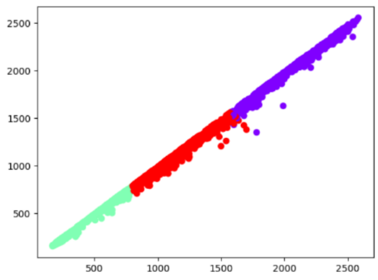
5306 rows × 15 columns

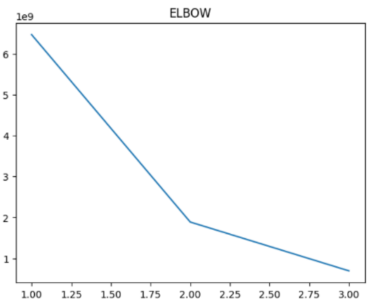
	Prev Close	Open	High
0	157.40	166.00	170.00
1	170.00	182.00	183.45
2	173.80	170.00	173.90
3	166.95	168.00	170.00
4	168.30	162.15	171.00
...	...	...	...
5301	1414.15	1413.00	1429.00
5302	1404.80	1407.25	1442.00
5303	1438.70	1436.25	1479.00
5304	1476.80	1486.20	1503.65
5305	1472.50	1445.00	1453.80

5306 rows × 3 columns

```
KMeans
KMeans(n_clusters=3)
```

```
array([1, 1, 1, ..., 2, 2, 2], dtype=int32)
```





RESULT:

Hence the Clusters are formed successfully.

Ex.no:	ID3 Algorithm
Date:	

AIM:

To Implement ID3 algorithm with the help of use define functions

ALGORITHM:

Step 1: Start the code
Step 2: Open a Jupyter Notebook
Step 3: Install the needed python packages using Pip install package name.
Step 4: Import the needed python packages using Import package name.
import math, import pandas as pd, import numpy as np, import matplotlib.pyplot as plt and from operator import itemgetter
Step 5: Create a decision tree class to perform user define function using ID3 algorithm

Step 6: Create a Contractor to init the process by set the target, positive, parent_val, parent

def __init__(self, df, target, positive, parent_val, parent):

Step 7: Create user define function like entropy, Gain, splitter, etc to process the descision tree

Step 8: Import a data set by using pandas.

df = pd.read_excel('exa.xlsx')
df

Step 8: Updated note values by using update node method.

Step 9: Print the values Parent values using print_tree(dt)

Step 10: Save and stop the process.

PROGRAM:

```
import math
import pandas as pd
class DecisionTree:
    def __init__(self, df, target, positive, parent_val, parent):
        self.data = df
        self.target = target
        self.positive = positive
        self.parent_val = parent_val
        self.parent = parent
        self.childs = []
        self.decision = ''
    def _get_entropy(self, data):
        p = sum(data[self.target] == self.positive)
        n = data.shape[0] - p
        p_ratio = p / (p + n)
        n_ratio = 1 - p_ratio
        entropy_p = -p_ratio * math.log2(p_ratio) if p_ratio != 0 else 0
        entropy_n = -n_ratio * math.log2(n_ratio) if n_ratio != 0 else 0
        return entropy_p + entropy_n
    def _get_gain(self, feat):
        avg_info = sum(self._get_entropy(self.data[self.data[feat] == val]) *
                   sum(self.data[feat] == val) / self.data.shape[0]
                   for val in self.data[feat].unique())
        return self._get_entropy(self.data) - avg_info
    def _get_splitter(self):
```

```
        self.splitter = max(self.gains, key=lambda x: x[1])[0]
    def update_nodes(self):
        self.features = [col for col in self.data.columns if col != self.target]
        self.entropy = self._get_entropy(self.data)
        if self.entropy != 0:
            self.gains = [(feat, self._get_gain(feat)) for feat in self.features]
            self._get_splitter()
            residual_columns = [k for  k in self.data.columns if k != self.splitter]
            for val in self.data[self.splitter].unique():
                df_tmp = self.data[self.data[self.splitter] == val][residual_columns]
                tmp_node = DecisionTree(df_tmp, self.target, self.positive, val, 
self.splitter)
                tmp_node.update_nodes()
                self.childs.append(tmp_node)
def print_tree(node, depth=0):
    if node:
        print(f"{'  ' * depth}Parent: {node.parent} | Parent Value: 
{node.parent_val}")
        for child in node.childs:
            print_tree(child, depth + 1)


df = pd.read_excel('exa.xlsx')
dt = DecisionTree(df, 'Play', 'Yes', '', '')
dt.update_nodes()
print_tree(dt)
```

OUTPUT:

Out[5]:

click to expand output; double click to hide output

	Outlook	Temperature	Humidity	WindSpeed	Play
0	Sunny	Hot	High	Weak	No
1	Sunny	Hot	High	Strong	No
2	Overcast	Hot	High	Weak	Yes
3	Rainy	Mild	High	Weak	Yes
4	Rainy	Cool	Normal	Weak	Yes
5	Rainy	Cool	Normal	Strong	No
6	Overcast	Cool	Normal	Strong	Yes
7	Sunny	Mild	High	Weak	No
8	Sunny	Cool	Normal	Weak	Yes
9	Rainy	Mild	Normal	Weak	Yes
10	Sunny	Mild	Normal	Strong	Yes
11	Overcast	Mild	High	Strong	Yes
12	Overcast	Hot	Normal	Weak	Yes
13	Rainy	Mild	High	Strong	No
14	Overcast	Mild	High	Strong	Yes
15	Overcast	Hot	Normal	Weak	Yes
16	Rainy	Mild	High	Strong	No

```
Parent:  | Parent Value:
  Parent: Outlook | Parent Value: Sunny
    Parent: Humidity | Parent Value: High
    Parent: Humidity | Parent Value: Normal
  Parent: Outlook | Parent Value: Overcast
  Parent: Outlook | Parent Value: Rainy
    Parent: WindSpeed | Parent Value: Weak
    Parent: WindSpeed | Parent Value: Strong
```

RESULT:

Hence the new nodes have been creating using ID3 Algorithm successfully.

Ex.no:	Non- Parametric Locally Weighted Regression
Date:	

AIM:

To Implement Non- Parametric Locally Weighted Regression and Visualize the value

ALGORITHM:

Step 1: Start the code
Step 2: Open a Jupyter Notebook
Step 3: Install the needed python packages using Pip install package name.
Step 4: Import the needed python packages using Import package name.
import math, import pandas as pd, import numpy as np, import matplotlib.pyplot as plt and from operator import itemgetter
Step 5: Create a pre-define function calculating the weight of regression

def locally_weighted_regression(test_point, X, y, tau):

test_point: The point at which prediction is to be made.

X: Feature matrix.

y: Target values.

tau: Bandwidth parameter for weighting.

Step 6: Calculate weights for each training point based on their distance from the test point using a Gaussian kernel.

Step 7: Call the plot_lwr function with the generated dataset X and y, along with the specified tau_values

Step 8: Generate a random dataset X of 100 points between 0 and 5. Calculate y values by taking the sine of each X value.

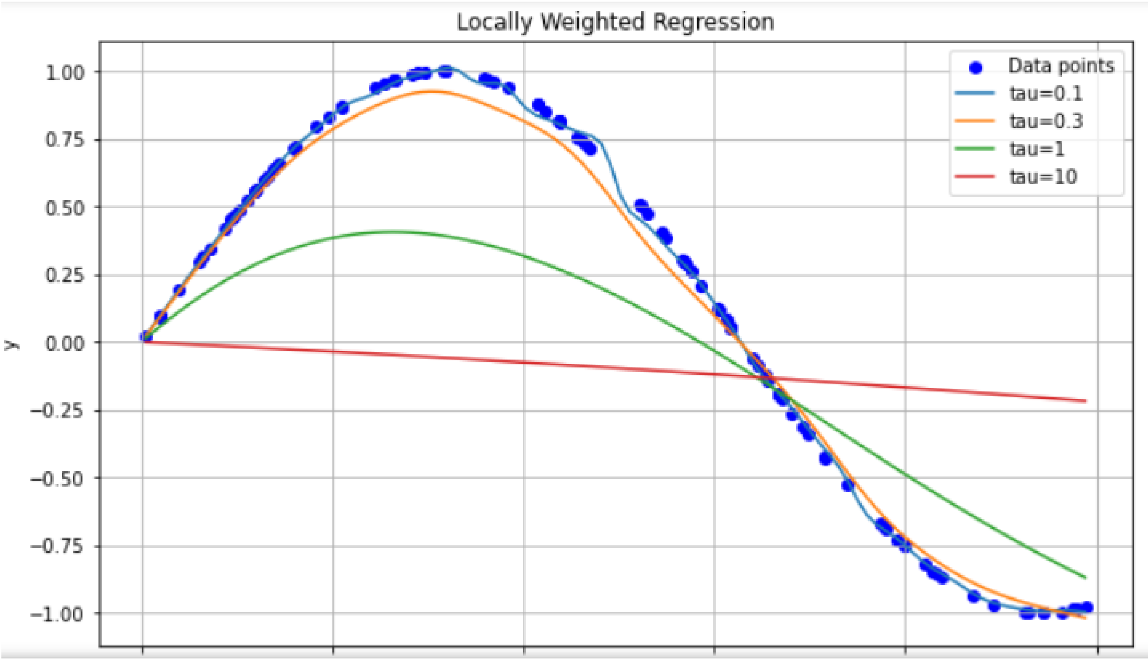
Step 9: Visualize the data using plot method.

Step 10: Stop the process.

PROGRAM:

```
import numpy as np
import matplotlib.pyplot as plt
def locally_weighted_regression(test_point, X, y, tau):
    # Locally Weighted Regression (LWR) function
    m = X.shape[0]
    weights = np.exp(-np.sum((X - test_point)**2, axis=1) / (2 * tau**2))
    W = np.diag(weights)
    theta = np.linalg.inv(X.T @ W @ X) @ (X.T @ W @ y)
    prediction = test_point @ theta
    return prediction
def plot_lwr(X, y, tau_values):
    #Plotting the Locally Weighted Regression predictions for different tau values
    X_test = np.linspace(np.min(X), np.max(X), 100).reshape(-1, 1)
    plt.figure(figsize=(10, 6))
    plt.scatter(X, y, color='blue', label='Data points')
    for tau in tau_values:
        predictions = [locally_weighted_regression(np.array([x]), X, y, tau) for x in X_test]
        plt.plot(X_test, predictions, label=f'tau={tau}')
    plt.xlabel('X')
    plt.ylabel('y')
    plt.title('Locally Weighted Regression')
    plt.legend()
    plt.grid(True)
    plt.show()
# Generate sample dataset
np.random.seed(0)
X = np.sort(5 * np.random.rand(100, 1), axis=0)
y = np.sin(X).ravel()
# Define tau values
tau_values = [0.1, 0.3, 1, 10]
# Plot Locally Weighted Regression for different tau values
plot_lwr(X, y, tau_values)
```

OUTPUT:



RESULT:

Hence the above Non- Parametric Locally Weighted Regression executed successful

Ex.no:	**k-Nearest Neighbour**
Date:	

AIM:

To Implement k-Nearest Neighbour algorithm to classify the iris data set.

ALGORITHM:

Step 1: Start the code
Step 2: Open a Jupyter Notebook
Step 3: Install the needed python packages using Pip install package name.
Step 4: Import the needed python packages using Import package name.
    import pandas as pd, import numpy as np, import matplotlib.pyplot as plt.
    from sklearn.neighbors import KNeighborsClassifier
Step 5: Import a data set by using pandas
    data= pd.read_csv(head.csv')
Step 6: Convert the dataset into a pandas data frame.
    dataset= pd.DataFrame(data)
Step 7: Select the x and y data points with the help of iloc() methods.
    X = dataset.iloc[:, [1, 2]].values
    y = dataset.iloc[:, -1].values
Step 8: Split the dataset into train and test for training and testing the machine.
    from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
Step 9: Train and test the machine by using fit () and predit() method.
    classifier.fit(X_train, y_train)
    y_pred = classifier.predict(X_test)
Step 10: Finally Evaluate the result using metrics ()
    from sklearn.metrics import accuracy_score
Step 11: Create new two variable to store the count of correct-predictions and Wrong-predictions values.
Step 12: Calculate the value using length of y-test and y-pred data
Step 13: Stop the Implementation.

PROGRAM:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
# Load dataset
dataset = pd.read_csv('head.csv')
# Extracting features and target variable
X = dataset.iloc[:, [1, 2]].values
y = dataset.iloc[:, -1].values
# Split dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=0)
# Initialize k-NN classifier
k = 3  # You can adjust the value of k
knn = KNeighborsClassifier(n_neighbors=k)
# Train the classifier
knn.fit(X_train, y_train)
# Predict classes for the test set
y_pred = knn.predict(X_test)
# Compute accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
```

```
# Print correct and wrong predictions
correct_predictions = 0
wrong_predictions = 0
for i in range(len(y_test)):
    if y_test[i] == y_pred[i]:
        print(f"Correct prediction: Predicted {y_pred[i]}, Actual {y_test[i]}")
        correct_predictions += 1
    else:
        print(f"Wrong prediction: Predicted {y_pred[i]}, Actual {y_test[i]}")
        wrong_predictions += 1
print(f"\nTotal correct predictions: {correct_predictions}")
print(f"Total wrong predictions: {wrong_predictions}")
```

OUTPUT:

```
▾ KNeighborsClassifier
KNeighborsClassifier(n_neighbors=3)
```

```
Wrong prediction: Predicted Iris-versicolor, Actual Iris-virginica
Correct prediction: Predicted Iris-versicolor, Actual Iris-versicolor
Correct prediction: Predicted Iris-setosa, Actual Iris-setosa
Correct prediction: Predicted Iris-virginica, Actual Iris-virginica
Correct prediction: Predicted Iris-setosa, Actual Iris-setosa
Correct prediction: Predicted Iris-virginica, Actual Iris-virginica
Correct prediction: Predicted Iris-setosa, Actual Iris-setosa
Wrong prediction: Predicted Iris-virginica, Actual Iris-versicolor
Wrong prediction: Predicted Iris-virginica, Actual Iris-versicolor
Correct prediction: Predicted Iris-versicolor, Actual Iris-versicolor
Correct prediction: Predicted Iris-virginica, Actual Iris-virginica
Wrong prediction: Predicted Iris-virginica, Actual Iris-versicolor
Correct prediction: Predicted Iris-versicolor, Actual Iris-versicolor
Wrong prediction: Predicted Iris-virginica, Actual Iris-versicolor
Wrong prediction: Predicted Iris-virginica, Actual Iris-versicolor
Correct prediction: Predicted Iris-setosa, Actual Iris-setosa
Correct prediction: Predicted Iris-versicolor, Actual Iris-versicolor
Correct prediction: Predicted Iris-versicolor, Actual Iris-versicolor
Correct prediction: Predicted Iris-setosa, Actual Iris-setosa
Correct prediction: Predicted Iris-setosa, Actual Iris-setosa
Wrong prediction: Predicted Iris-versicolor, Actual Iris-virginica
Correct prediction: Predicted Iris-versicolor, Actual Iris-versicolor
Correct prediction: Predicted Iris-setosa, Actual Iris-setosa
Correct prediction: Predicted Iris-setosa, Actual Iris-setosa
Correct prediction: Predicted Iris-virginica, Actual Iris-virginica
Correct prediction: Predicted Iris-setosa, Actual Iris-setosa
Correct prediction: Predicted Iris-setosa, Actual Iris-setosa
Wrong prediction: Predicted Iris-virginica, Actual Iris-versicolor
Correct prediction: Predicted Iris-versicolor, Actual Iris-versicolor
Correct prediction: Predicted Iris-setosa, Actual Iris-setosa
```

```
Total correct predictions: 22
Total wrong predictions: 8
```

RESULT:

This the above dataset has been evaluated successfully and find out the count of correct and wrong prediction.

Ex.no:	Semi Supervised Classifier
Date:	

AIM:

To Assuming a set of documents that need to be classified, use the Semi Supervised Classifier model to perform this task. Built-in Java classes/API can be used to write the program. Calculate the accuracy, precision, and recall for your data set.

ALGORITHM:

Step 1: Start the code

Step 2: Open a Jupyter Notebook

Step 3: Install the needed python packages using Pip install package name. Step 4: Import the needed python packages using Import package name.

Step 5: Prepare a labeled and unlabeled data for implementation

Step 6: Combine data for extraction feature from the input data.

Step 7: Create label distributions matrix

Step 8: Split labeled data into training set and test set

Step 9: Using predict method predict the new value

Step 10: Evaluate the output using metric method

Step 11: Stop the process

PROGRAM:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.semi_supervised import LabelPropagation
from sklearn.metrics import accuracy_score, precision_score, recall_score
from sklearn.model_selection import train_test_split
import numpy as np
# Sample data (replace with your actual data)
labeled_data = [("This is a document about sports", "Sports"),
               ("This is a news article", "News"),
               ("Another document about sports", "Sports"),
               ("A text sample about politics", "Politics"),
               ("A document discussing music", "Music")]
unlabeled_data = ["This document discusses machine learning",
                 "Another document about music",
                 "A short text sample"]
# Combine data for feature extraction
all_data = [text for text, _ in labeled_data] + unlabeled_data
# Extract labels from labeled data
texts, labels = zip(*labeled_data)
# Feature Extraction (TF-IDF)
vectorizer = TfidfVectorizer(max_features=500)
features = vectorizer.fit_transform(all_data)
# Convert features to dense numpy array
features_dense = features.toarray()
# Get all unique labels
all_labels = sorted(set(labels))
```

```
# Create label distributions matrix

label_distributions = np.zeros((len(texts), len(all_labels)))

for i, label in enumerate(labels):

    label_distributions[i, all_labels.index(label)] = 1

# Split labeled data into training set and test set

X_train, X_test, y_train, y_test = train_test_split(features_dense[:len(texts)],
labels, test_size=0.2, random_state=42)

# Convert y_train to indices of true classes

y_train_indices = np.array([all_labels.index(label) for label in y_train])

# Train the semi-supervised classifier

semi_clf = LabelPropagation()

semi_clf.fit(X_train, y_train_indices)  # Ensure y_train_indices is passed as is

# Predict labels for test set

predictions = semi_clf.predict(X_test)

# Calculate evaluation metrics

accuracy = accuracy_score(np.array([all_labels.index(label) for label in y_test]),
predictions)

precision = precision_score(np.array([all_labels.index(label) for label in
y_test]), predictions, average='weighted', labels=np.unique(predictions))

recall = recall_score(np.array([all_labels.index(label) for label in y_test]),
predictions, average='weighted', labels=np.unique(predictions))

print(f"Accuracy: {accuracy:.4f}")

print(f"Precision: {precision:.4f}")

print(f"Recall: {recall:.4f}")
```

OUTPUT:

```
Accuracy: 1.0000
Precision: 1.0000
Recall: 1.0000
```

RESULT:

This the above dataset has been executed successfully

Ex.no:	**Implementing Q Learning with Linear Function**
Date:	

AIM:

To Implementing Q Learning with Linear Function.

ALGORITHM:

Step 1: Start the code
Step 2: Open a Jupyter Notebook
Step 3: Install the needed python packages using Pip install package name.
Step 4: Import the needed python packages using Import package name.
import numpy as np.
Step 5: Create a new class called QlearningLinearFA to Implement Qlearning Algorithm.

Step 6: Initialize Q-learning with linear function approximation

Step 7: Select action using Q-learning policy for Update Q-function approximation.

Step 8: Simulate new environment, get reward and next state

Step 10: Stop the process

PROGRAM:

```
import numpy as np

class QLearningLinearFA:

    def __init__(self, num_features, num_actions, learning_rate=0.1, discount_factor=0.9, epsilon=0.1):

        self.num_features = num_features
```

```
        self.num_actions = num_actions
        self.learning_rate = learning_rate
        self.discount_factor = discount_factor
        self.epsilon = epsilon
        self.weights = np.zeros((num_actions, num_features))
    def select_action(self, state):
        if np.random.rand() < self.epsilon:
            return np.random.choice(self.num_actions)  # Random action
        else:
            return np.argmax(np.dot(self.weights, state))  # Greedy action
    def update_weights(self, state, action, reward, next_state):
        target = reward + self.discount_factor * np.max(np.dot(self.weights,
next_state))
        predicted = np.dot(self.weights[action], state)
        error = target - predicted
        self.weights[action] += self.learning_rate * error * state
num_features = 4  # Number of features representing the state
num_actions = 3  # Number of possible actions
# Initialize Q-learning with linear function approximation
ql = QLearningLinearFA(num_features, num_actions)

# Example training loop
num_episodes = 1000
for episode in range(num_episodes):
    # Simulate environment, get initial state
    state = np.random.rand(num_features)
    done = False
```

```
    total_reward = 0
    while not done:
        # Select action using Q-learning policy
        action = ql.select_action(state)
        # Simulate environment, get reward and next state
        next_state = np.random.rand(num_features)
        reward = np.random.randn()  # Replace with actual reward from environment
        done = np.random.rand() < 0.1  # Example termination condition
        # Update Q-function approximation
        ql.update_weights(state, action, reward, next_state)
        # Update current state
        state = next_state
        # Accumulate total reward
        total_reward += reward
    print("Episode:", episode, "Total Reward:", total_reward)
```

OUTPUT:

```
Episode: 0 Total Reward: 0.23164191048727734
Episode: 1 Total Reward: 1.216631791372024
Episode: 2 Total Reward: 1.5481514225924031
Episode: 3 Total Reward: -0.5850989635363393
Episode: 4 Total Reward: 0.27616117661186623
Episode: 5 Total Reward: 3.39481332461527
Episode: 6 Total Reward: -0.318975381670183
Episode: 7 Total Reward: 2.1505603930930777
Episode: 8 Total Reward: -0.5352818390031747
Episode: 9 Total Reward: -5.459316766220038
Episode: 10 Total Reward: -3.822035062104668
Episode: 11 Total Reward: -0.7199283886642969
Episode: 12 Total Reward: -1.118032836706928
Episode: 13 Total Reward: -3.1250452718359734
Episode: 14 Total Reward: -4.039905704341215
Episode: 15 Total Reward: 0.33523250345472044
Episode: 16 Total Reward: 3.1549129465787074
Episode: 17 Total Reward: -0.857804181145866
Episode: 18 Total Reward: 2.1867597297500656
Episode: 19 Total Reward: 2.4359862236893033
Episode: 20 Total Reward: 0.4134972584309905
Episode: 21 Total Reward: -5.850855641023605
Episode: 22 Total Reward: -0.12438232391601778
Episode: 23 Total Reward: -0.13977500190133935
Episode: 24 Total Reward: 2.432107374637901
Episode: 25 Total Reward: -1.987578641003282
Episode: 26 Total Reward: 0.6699654801850995
Episode: 27 Total Reward: -1.4879187942317895
Episode: 28 Total Reward: -6.210664969248061
Episode: 29 Total Reward: -4.660790023896409
Episode: 30 Total Reward: 4.674659304061066
```

```
Episode: 70 Total Reward: -2.3655588325917023
click to scroll output; double click to hide 95940338173836
Episode: 72 Total Reward: -0.7032910108997836
Episode: 73 Total Reward: -10.239050383766754
Episode: 74 Total Reward: -1.4603398186442806
Episode: 75 Total Reward: 0.9621400004671353
Episode: 76 Total Reward: -1.0556395126893692
Episode: 77 Total Reward: -1.0181193882847674
Episode: 78 Total Reward: -2.5701086836988765
Episode: 79 Total Reward: -0.6134723885325147
Episode: 80 Total Reward: -1.3047732984309528
Episode: 81 Total Reward: 2.7919199823116987
Episode: 82 Total Reward: -1.2268426429825725
Episode: 83 Total Reward: 0.11379085206825446
Episode: 84 Total Reward: -3.8245161960740566
Episode: 85 Total Reward: -1.2056440166488265
Episode: 86 Total Reward: -0.04378846811938364
Episode: 87 Total Reward: 1.6905037346427647
Episode: 88 Total Reward: -0.8056048961419616
Episode: 89 Total Reward: 1.86346839566992
Episode: 90 Total Reward: -5.30074278397618
Episode: 91 Total Reward: -1.0818440249063617
Episode: 92 Total Reward: -3.1506061365900795
Episode: 93 Total Reward: -1.920603602231221
Episode: 94 Total Reward: -4.195245854756887
Episode: 95 Total Reward: -3.461766236516185
Episode: 96 Total Reward: 2.472250141749173
Episode: 97 Total Reward: -1.9522792255338572
Episode: 98 Total Reward: 0.3758726988829073
Episode: 99 Total Reward: -4.770064041231527
```

RESULT:

Hence the above program executed successfully.

Ex.no:	**Implement the Policy Gradient**
Date:	

AIM:

To Implement the Policy Gradient

ALGORITHM:

Step 1: Start the code
Step 2: Open a Jupyter Notebook
Step 3: Install the needed python packages using Pip install package name.
Step 4: Import the needed python packages using Import package name.
import numpy as np.
Step 5: Create class called REINFORCEBaselineAgent

Step 6: Create a constructer for processing Time series

Step 7: Create a loop for tarin the data to the model

num_states = 5

num_actions = 2

num_episodes = 1000

Step 8: Create class called ActorCriticBaselineAgent for basing the base line

Step 9: print the Episode values

print("Actor-Critic Episode {}: Total Reward = {}".format(episode + 1, episode_rewards))

Step 10: Stop the process

PROGRAM:

```
import numpy as np #package


class REINFORCEAgent:# class
    def __init__(self, num_actions, num_states, gamma=0.99,
learning_rate=0.01):
         # gamma is discount factor for finding future reward
        self.num_actions = num_actions
        self.num_states = num_states
        self.gamma = gamma
        self.learning_rate = learning_rate
        self.policy = np.zeros((num_states, num_actions))


    def get_action(self, state): #return action for current policy
        action_probs = self._softmax(self.policy[state])
        #Accesses the policy for the current state
        #softmax function that takes as input a vector of real numbers and returns a
vector of probabilities
        return np.random.choice(self.num_actions, p=action_probs)


    def train(self, episode):
        states, actions, rewards = zip(*episode)
        returns = self._calculate_returns(rewards)
        for t, (state, action) in enumerate(zip(states, actions)):
            delta = returns[t] - self.policy[state, action]
            self.policy[state, action] += self.learning_rate * delta
```

```
        print("Episode training complete.")

            #enumerate built-in function that allows you to loop over an iterable
(such as a list, tuple, or string)

    def _calculate_returns(self, rewards):
        G = 0
        returns = []
        for r in reversed(rewards):
            G = r + self.gamma * G
            returns.insert(0, G)
        return returns

    def _softmax(self, x):
        exp_values = np.exp(x - np.max(x)) #exponential
        return exp_values / np.sum(exp_values)

# Simple environment
class SimpleEnvironment:# class
    def __init__(self, num_states, num_actions):
        self.num_states = num_states
        self.num_actions = num_actions

    def reset(self):
        return 0

    def step(self, state, action):
        new_state = max(0, min(self.num_states - 1, state + (action * 2 - 1)))
        reward = 1 if new_state == self.num_states - 1 else 0
```

```
        return new_state, reward


# Training loop
num_states = 5
num_actions = 2
num_episodes = 1000
env = SimpleEnvironment(num_states, num_actions)
agent = REINFORCEAgent(num_actions, num_states)


for episode_num in range(num_episodes):
    state = env.reset()
    episode = []
    total_reward = 0  # Track total reward for the episode
    done = False
    while not done:
        action = agent.get_action(state)
        next_state, reward = env.step(state, action)
        total_reward += reward  # Accumulate reward for the episode
        episode.append((state, action, reward))
        state = next_state
        done = next_state == num_states - 1
    agent.train(episode)
    print("Episode:", episode_num + 1, "Total Reward:", total_reward)  # Print total reward for the episode
```

OUTPUT:

```
Episode training complete.
Episode: 1 Total Reward: 1
Episode training complete.
Episode: 2 Total Reward: 1
Episode training complete.
Episode: 3 Total Reward: 1
Episode training complete.
Episode: 4 Total Reward: 1
Episode training complete.
Episode: 5 Total Reward: 1
Episode training complete.
Episode: 6 Total Reward: 1
Episode training complete.
Episode: 7 Total Reward: 1
Episode training complete.
Episode: 8 Total Reward: 1
Episode training complete.
Episode: 9 Total Reward: 1
Episode training complete.
Episode: 10 Total Reward: 1
Episode training complete.
Episode: 11 Total Reward: 1
Episode training complete.
Episode: 12 Total Reward: 1
Episode training complete.
Episode: 13 Total Reward: 1
Episode training complete.
Episode: 14 Total Reward: 1
Episode training complete.
Episode: 15 Total Reward: 1
```

```
Episode training complete.
Episode: 86 Total Reward: 1
Episode training complete.
Episode: 87 Total Reward: 1
Episode training complete.
Episode: 88 Total Reward: 1
Episode training complete.
Episode: 89 Total Reward: 1
Episode training complete.
Episode: 90 Total Reward: 1
Episode training complete.
Episode: 91 Total Reward: 1
Episode training complete.
Episode: 92 Total Reward: 1
Episode training complete.
Episode: 93 Total Reward: 1
Episode training complete.
Episode: 94 Total Reward: 1
Episode training complete.
Episode: 95 Total Reward: 1
Episode training complete.
Episode: 96 Total Reward: 1
Episode training complete.
Episode: 97 Total Reward: 1
Episode training complete.
Episode: 98 Total Reward: 1
Episode training complete.
Episode: 99 Total Reward: 1
Episode training complete.
Episode: 100 Total Reward: 1
```

RESULT:

Hence the above program executed successfully.

Ex.no:	Time Series Data processing
Date:	

AIM:

To process Time series using decomposition methods

ALGORITHM:

Step 1: Start the code
Step 2: Open a Jupyter Notebook
Step 3: Install the needed python packages using Pip install package name.
Step 4: Import the needed python packages using Import package name.

import pandas as pd, import matplotlib.pyplot as plt

from statsmodels.tsa.seasonal import seasonal_decompose

from statsmodels.tsa.stattools import adfuller

Step 5: Import the data set using pandas package.

df = pd.read_csv("D:\Jeyashri\IBM\Datasets\Social_Network_Ads.csv")

df

Step 6: plot the data using matplotlib.pyplot before processing it

Step 7: Decompose the time series data for seasonal period of 12 months

decomposition = seasonal_decompose(data[data.columns[0]], model='additive', period=12)

Step 8: Check for stationarity using Augmented Dickey-Fuller test

adf_result = adfuller(data[data.columns[0]])

print('ADF Statistic:', adf_result[0])

print('p-value:', adf_result[1])

print('Critical Values:', adf_result[4])

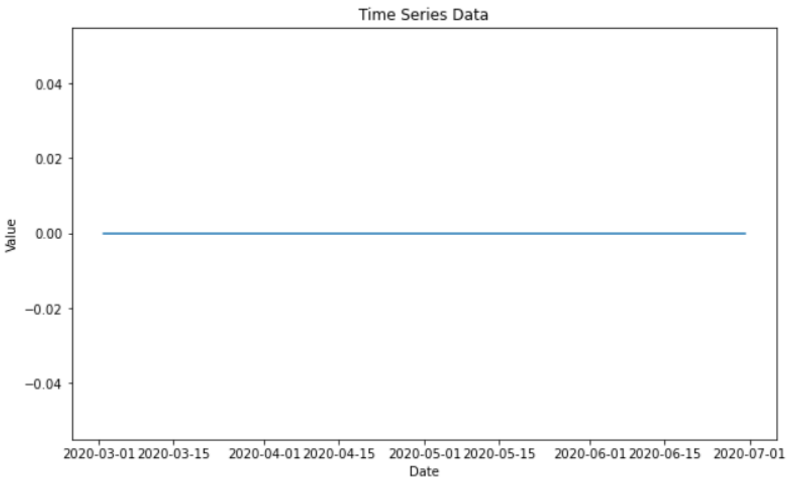
Step 9: Stop the process

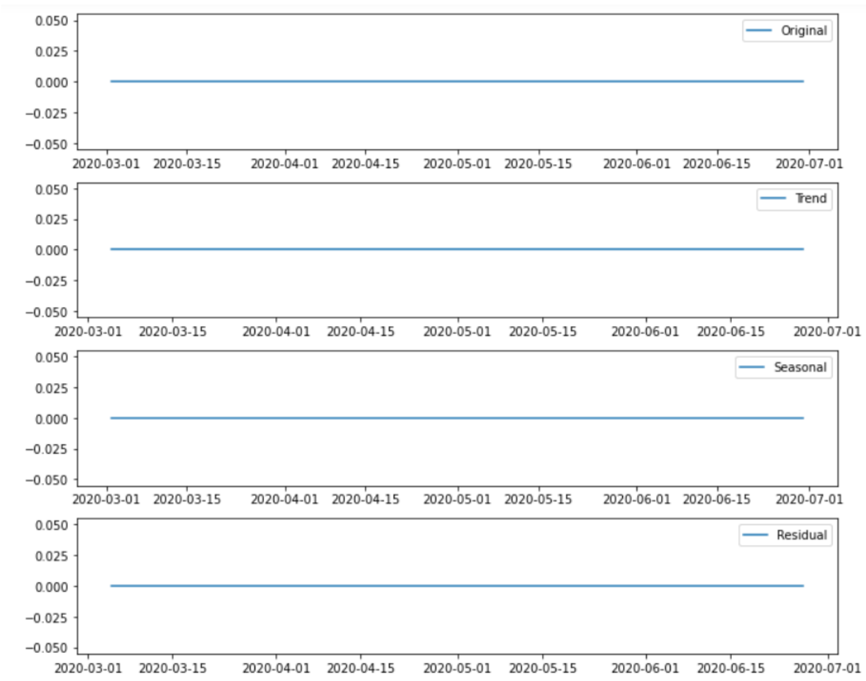
PROGRAM:

```
import pandas as pd

import matplotlib.pyplot as plt

from statsmodels.tsa.seasonal import seasonal_decompose

from statsmodels.tsa.stattools import adfuller

# Load time series data

data = pd.read_csv('TimeGender.csv', index_col=0, parse_dates=True)

# Plot the time series data

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

plt.plot(data.index, data[data.columns[0]])

plt.title('Time Series Data')

plt.xlabel('Date')

plt.ylabel('Value')

plt.show()

# Decompose the time series data

decomposition = seasonal_decompose(data[data.columns[0]], model='additive',
period=12)

# Assuming a seasonal period of 12 months

trend = decomposition.trend

seasonal = decomposition.seasonal

residual = decomposition.resid

# Plot the decomposed components

plt.figure(figsize=(10, 8))

plt.subplot(411)

plt.plot(data[data.columns[0]], label='Original')
```

```
plt.legend(loc='best')
plt.subplot(412)
plt.plot(trend, label='Trend')
plt.legend(loc='best')
plt.subplot(413)
plt.plot(seasonal, label='Seasonal')
plt.legend(loc='best')
plt.subplot(414)
plt.plot(residual, label='Residual')
plt.legend(loc='best')
plt.tight_layout()
# Check for stationarity using Augmented Dickey-Fuller test
adf_result = adfuller(data[data.columns[0]])
print('ADF Statistic:', adf_result[0])
print('p-value:', adf_result[1])
print('Critical Values:', adf_result[4])
# Identify the trend pattern
plt.show()
```

OUTPUT:





RESULT:

Hence the above program executed successfully.