IT18712 - Big Data Mining and Analytics Laboratory CAT-1 Assignment

Subash V

2127210801101

IT B

EXERCISE 2 : DATA VISUALIZATION AND ANALYSIS

AIM :

To load the heart disease dataset and visualize in different dimension. Attributes available in heart disease dataset : age , sex , cp , trestbps , chol , fbs , thalach ,restecg ,exang ,oldpeak , slope ,ca , thal.

PROCEDURE :

Step 1 : Import necessary packages :

- Pandas - import pandas as pd
- Numpy - import numpy as np
- Matplotlib.pyplot - import matplotlib.pyplot as plt
- Seaborn - import seaborn as sns

Step 2 : Read the csv file and display first 10 records.

```
import pandas as pd
df=pd.read_csv('heart.csv')
df.head(10)
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	52	1	0	125	212	0	1	168	0	1.0	2	2	3	0
1	53	1	0	140	203	1	0	155	1	3.1	0	0	3	0
2	70	1	0	145	174	0	1	125	1	2.6	0	0	3	0
3	61	1	0	148	203	0	1	161	0	0.0	2	1	3	0
4	62	0	0	138	294	1	1	106	0	1.9	1	3	2	0
5	58	0	0	100	248	0	0	122	0	1.0	1	0	2	1
6	58	1	0	114	318	0	2	140	0	4.4	0	3	1	0
7	55	1	0	160	289	0	0	145	1	0.8	1	1	3	0
8	46	1	0	120	249	0	0	144	0	0.8	2	0	3	0
9	54	1	0	122	286	0	0	116	1	3.2	1	2	2	0

```
# Display the columns of the data
print("Columns of the dataset:")
print(df.columns)
```

```
Columns of the dataset:
Index(['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach',
       'exang', 'oldpeak', 'slope', 'ca', 'thal', 'target'],
      dtype='object')
```

Step 3 : Display the attributes and check for null values.

```
print("Null Values in the dataset :")
print(df.isnull().sum())
```

```
Null Values in the dataset :
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64
```

Step 4 : Display the statistical information.

```
[ ] df.describe()
```

	age	sex	cp	trestbps	chol	fbs	restecg
count	1025.000000	1025.000000	1025.000000	1025.000000	1025.00000	1025.000000	1025.000000
mean	54.434146	0.695610	0.942439	131.611707	246.00000	0.149268	0.529756
std	9.072290	0.460373	1.029641	17.516718	51.59251	0.356527	0.527878
min	29.000000	0.000000	0.000000	94.000000	126.00000	0.000000	0.000000
25%	48.000000	0.000000	0.000000	120.000000	211.00000	0.000000	0.000000
50%	56.000000	1.000000	1.000000	130.000000	240.00000	0.000000	1.000000
75%	61.000000	1.000000	2.000000	140.000000	275.00000	0.000000	1.000000
max	77.000000	1.000000	3.000000	200.000000	564.00000	1.000000	2.000000

	thalach	exang	oldpeak	slope	ca	thal	targ
count	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000	1025.0000
mean	149.114146	0.336585	1.071512	1.385366	0.754146	2.323902	0.5131
std	23.005724	0.472772	1.175053	0.617755	1.030798	0.620660	0.5000
min	71.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0000
25%	132.000000	0.000000	0.000000	1.000000	0.000000	2.000000	0.0000
50%	152.000000	0.000000	0.800000	1.000000	0.000000	2.000000	1.0000
75%	166.000000	1.000000	1.800000	2.000000	1.000000	3.000000	1.0000
max	202.000000	1.000000	6.200000	2.000000	4.000000	3.000000	1.0000

Step 5 : Identify the numerical features from the dataset.

```
numerical_columns = df.select_dtypes(include=['int64', 'float64']).columns.tolist()
#Step 4: Print numerical columns
print("\nNumerical Columns:")
print(numerical_columns)
```

```
Numerical Columns:
['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach', 'exang', 'oldpeak', 'slope', 'ca', 'thal', 'target']
```

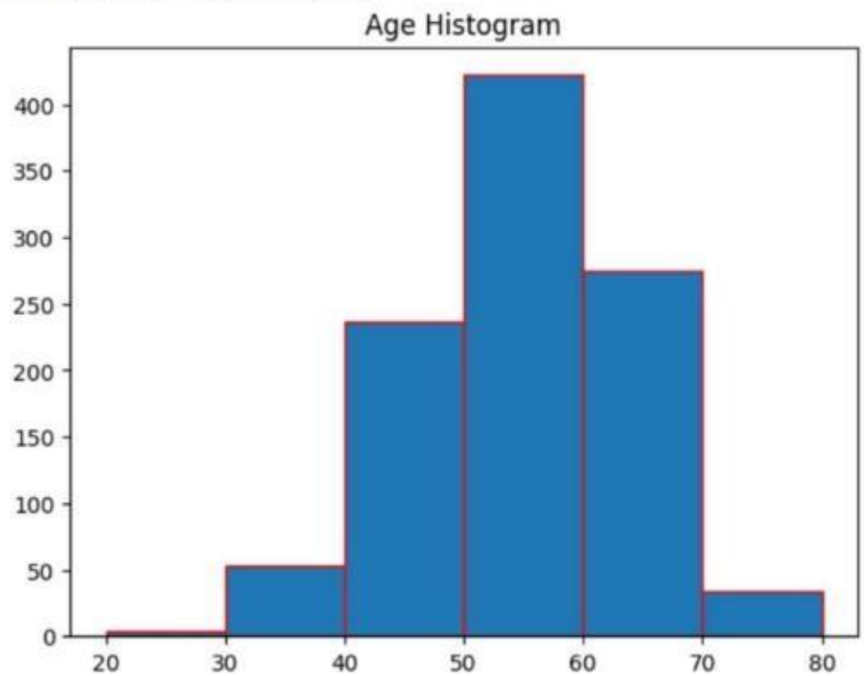
Step 6 : Generate Histogram and scatter plot for all numerical attributes with respect to target.

HISTOGRAM :

```
import matplotlib.pyplot as plt
```

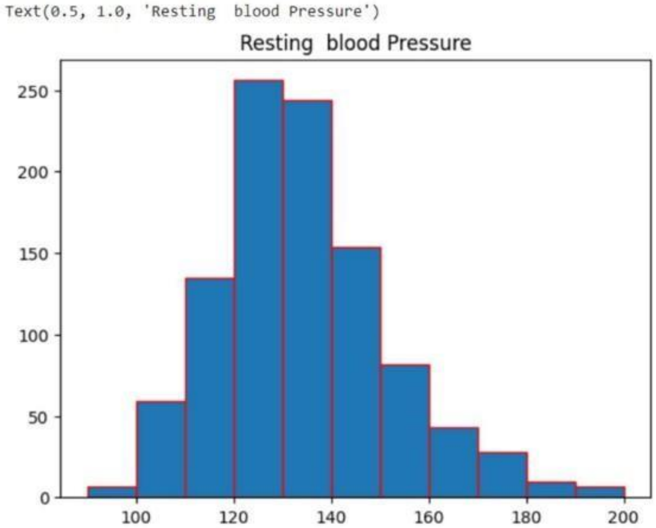
```
plt.hist(df['age'],bins=[20,30,40,50,60,70,80],edgecolor='red')
plt.title('Age Histogram')
```

```
Text(0.5, 1.0, 'Age Histogram')
```



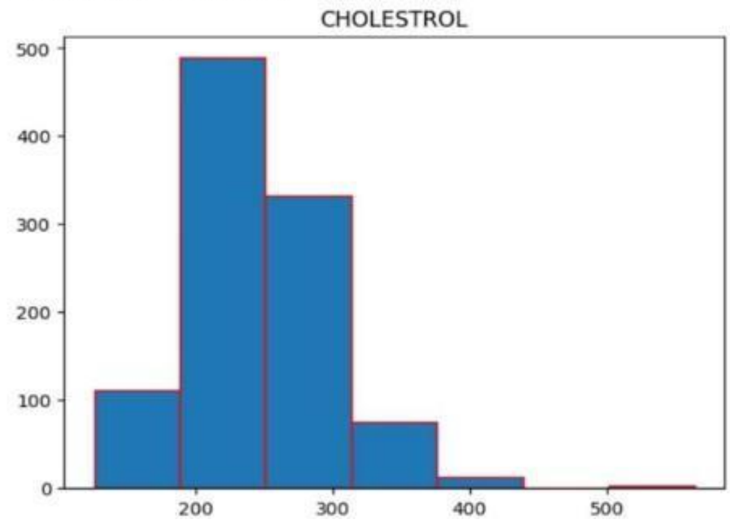

```
plt.hist(df['trestbps'],bins=[90, 100, 110, 120, 130, 140, 150, 160, 170, 180, 190, 200],edgecolor='red')
plt.title('Resting  blood Pressure')
```

```
Text(0.5, 1.0, 'Resting  blood Pressure')
```



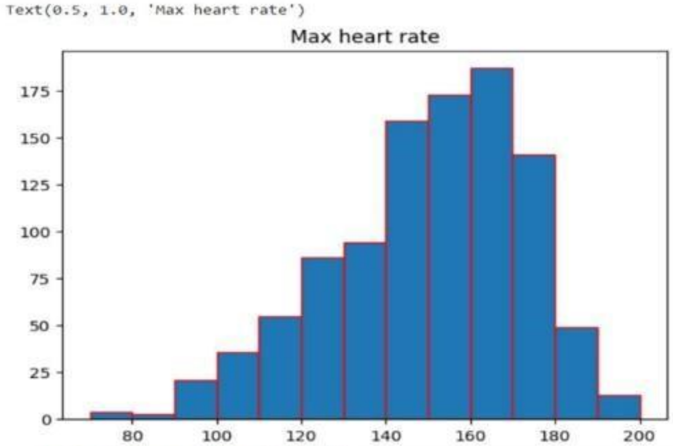
```
plt.hist(df['chol'],bins=7,edgecolor='red')
plt.title('CHOLESTROL')
```

```
Text(0.5, 1.0, 'CHOLESTROL')
```



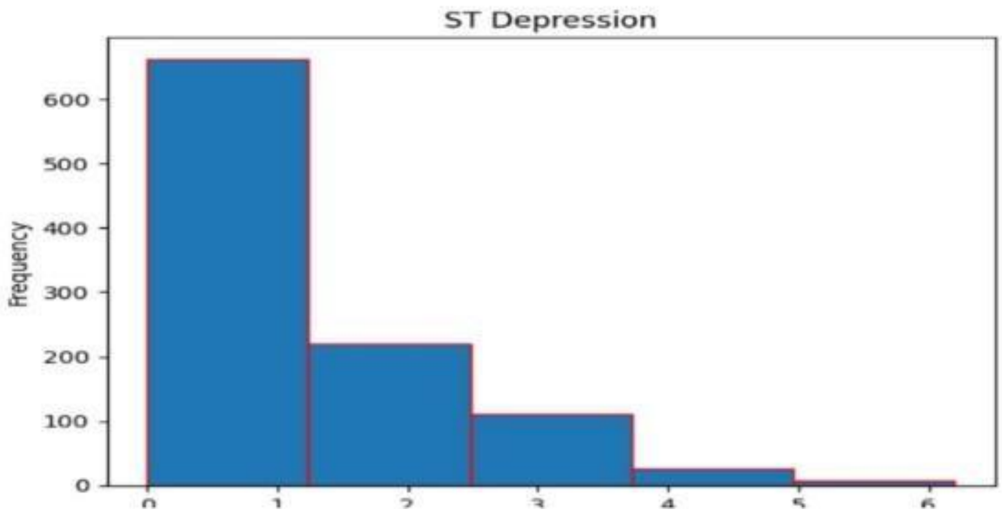

```
plt.hist(df['thalach'],bins=[70,80,90, 100, 110, 120, 130, 140, 150, 160, 170, 180, 190, 200],edgecolor='red')
plt.title('Max heart rate')
```

```
Text(0.5, 1.0, 'Max heart rate')
```




```
# Plot histogram for 'oldpeak'
plt.hist(df['oldpeak'], bins=5, edgecolor='red')
plt.title('ST Depression')
plt.xlabel('Oldpeak')
plt.ylabel('Frequency')
plt.show()
```

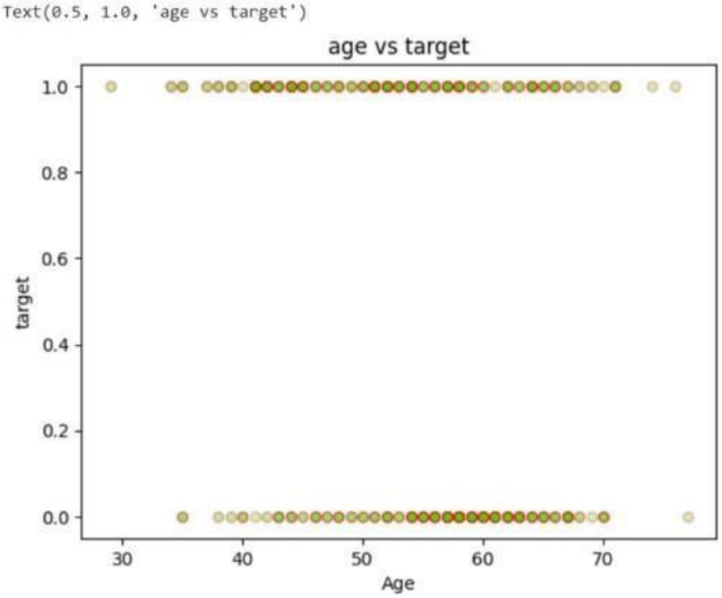
```
# Plot histogram for 'oldpeak'
plt.hist(df['oldpeak'], bins=5, edgecolor='red')
plt.title('ST Depression')
plt.xlabel('Oldpeak')
plt.ylabel('Frequency')
plt.show()
```



SCATTER PLOT :

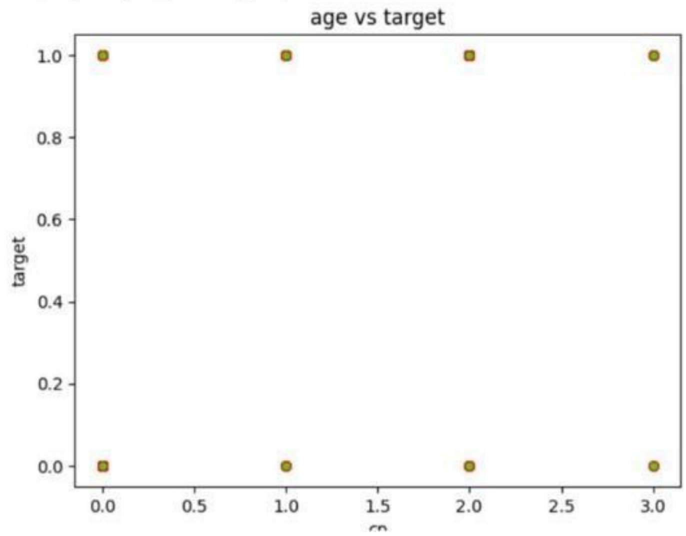
```
plt.scatter(df['age'],df['target'],s=30,c='#55cc10',edgecolor='red',linewidth=1,alpha=0.1)
plt.xlabel('Age')
plt.ylabel('target')
plt.title('age vs target')
```

```
Text(0.5, 1.0, 'age vs target')
```



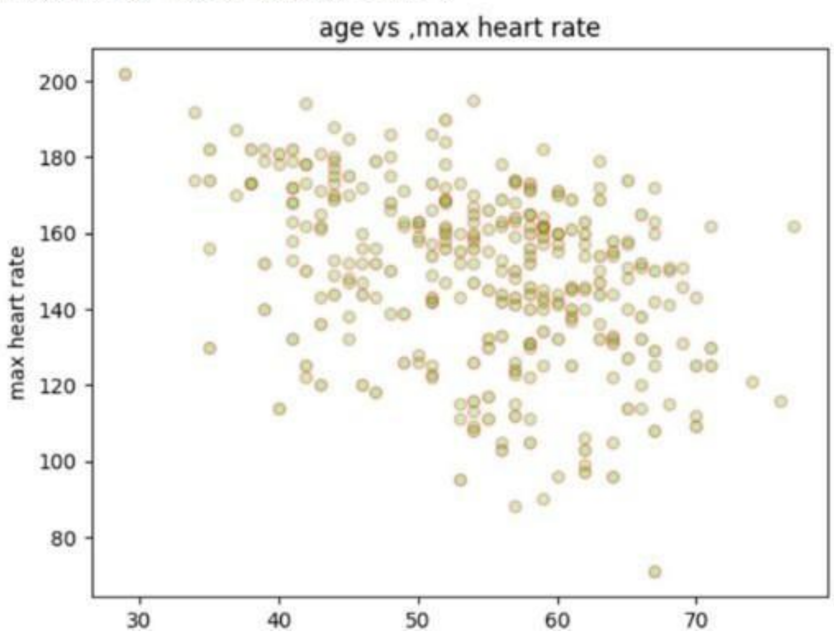
```
plt.scatter(df['cp'],df['target'],s=30,c='#55cc10',edgecolor='red',linewidth=1,alpha=0.1)
plt.xlabel('cp')
plt.ylabel('target')
plt.title('age vs target')
```

```
Text(0.5, 1.0, 'age vs target')
```



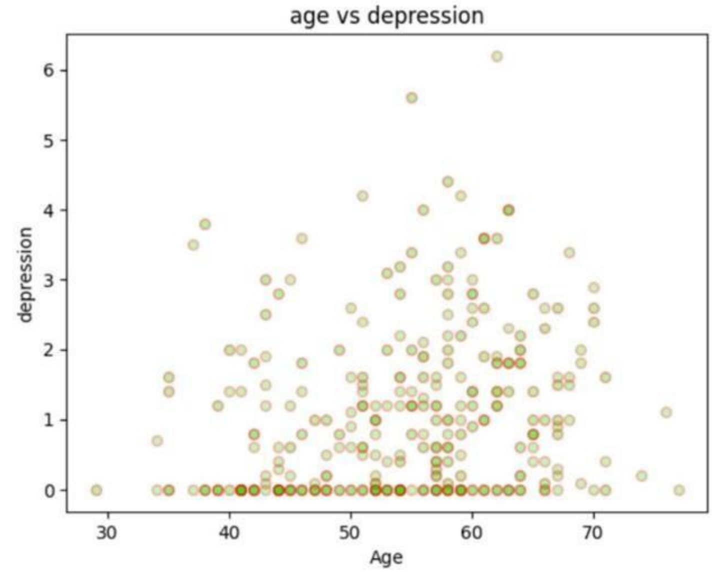

```
plt.scatter(df['age'],df['thalach'],s=30,c='#55cc10',edgecolor='red',linewidth=1,alpha=0.1)
plt.xlabel('Age')
plt.ylabel('max heart rate')
plt.title('age vs ,max heart rate')
```

```
Text(0.5, 1.0, 'age vs ,max heart rate')
```



```
plt.scatter(df['age'],df['oldpeak'],s=30,c='#55cc10',edgecolor='red',linewidth=1,alpha=0.1)
plt.xlabel('Age')
plt.ylabel('depression')
plt.title('age vs depression')
```

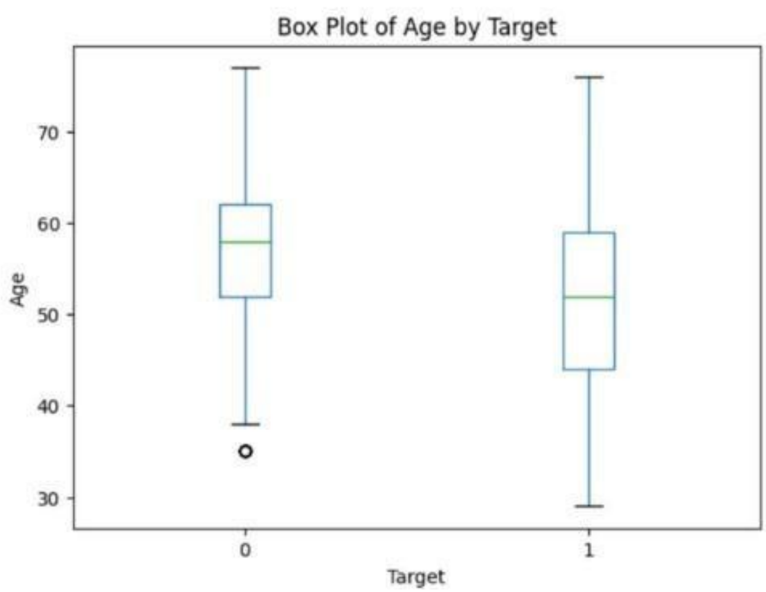
```
Text(0.5, 1.0, 'age vs depression')
```



Step 7 : Analyze the attributes with respect to target using the following chart ●

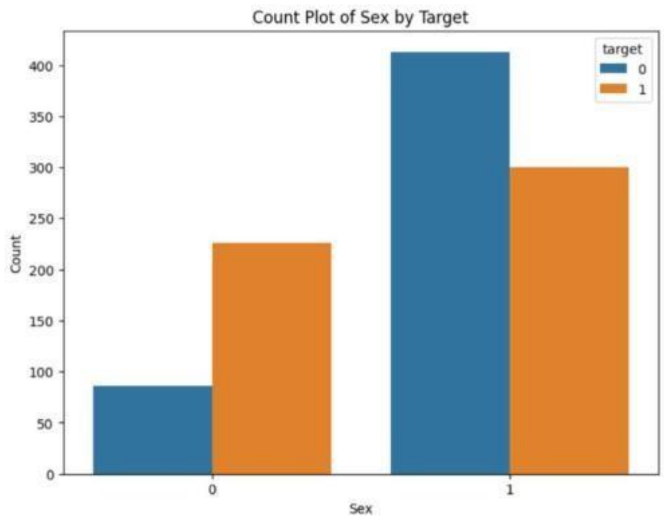
Box plot (Whiskar plot)

```
# Box plot for 'age' with respect to 'target'
df.boxplot(by='target', column=['age'], grid=False)
plt.title('Box Plot of Age by Target')
plt.suptitle('')
plt.xlabel('Target')
plt.ylabel('Age')
plt.show()
```



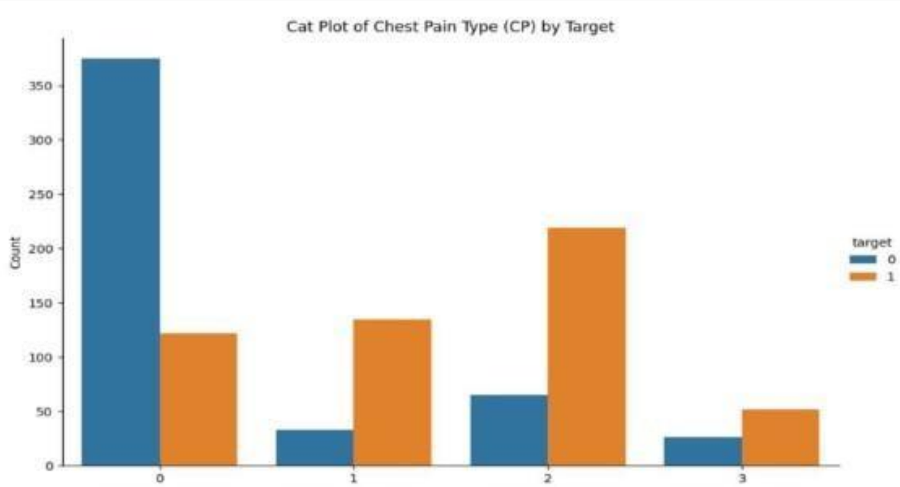
- **Count plot :**

```python
# Count plot for 'sex' with respect to 'target'
plt.figure(figsize=(8, 6))
sns.countplot(data=df, x='sex', hue='target')
plt.title('Count Plot of Sex by Target')
plt.xlabel('Sex')
plt.ylabel('Count')
plt.show()
```



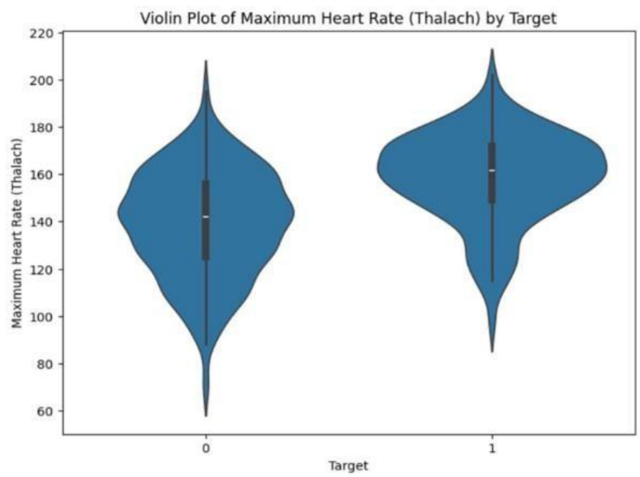
- **Cat plot :**

```python
# Categorical plot for 'cp' with respect to 'target'
sns.catplot(data=df, x='cp', hue='target', kind='count', height=6, aspect=1.5)

# Display the plot
plt.title('Cat Plot of Chest Pain Type (CP) by Target')
plt.xlabel('Chest Pain Type (CP)')
plt.ylabel('Count')
plt.show()
```



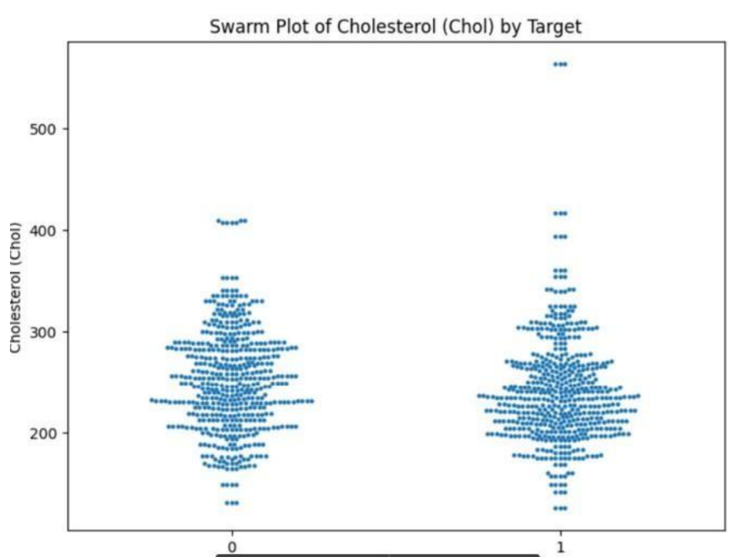
- **Violin plot :**

```python
# Violin plot for 'thalach' with respect to 'target'
plt.figure(figsize=(8, 6))
sns.violinplot(data=df, x='target', y='thalach')

# Display the plot
plt.title('Violin Plot of Maximum Heart Rate (Thalach) by Target')
plt.xlabel('Target')
plt.ylabel('Maximum Heart Rate (Thalach)')
plt.show()
```



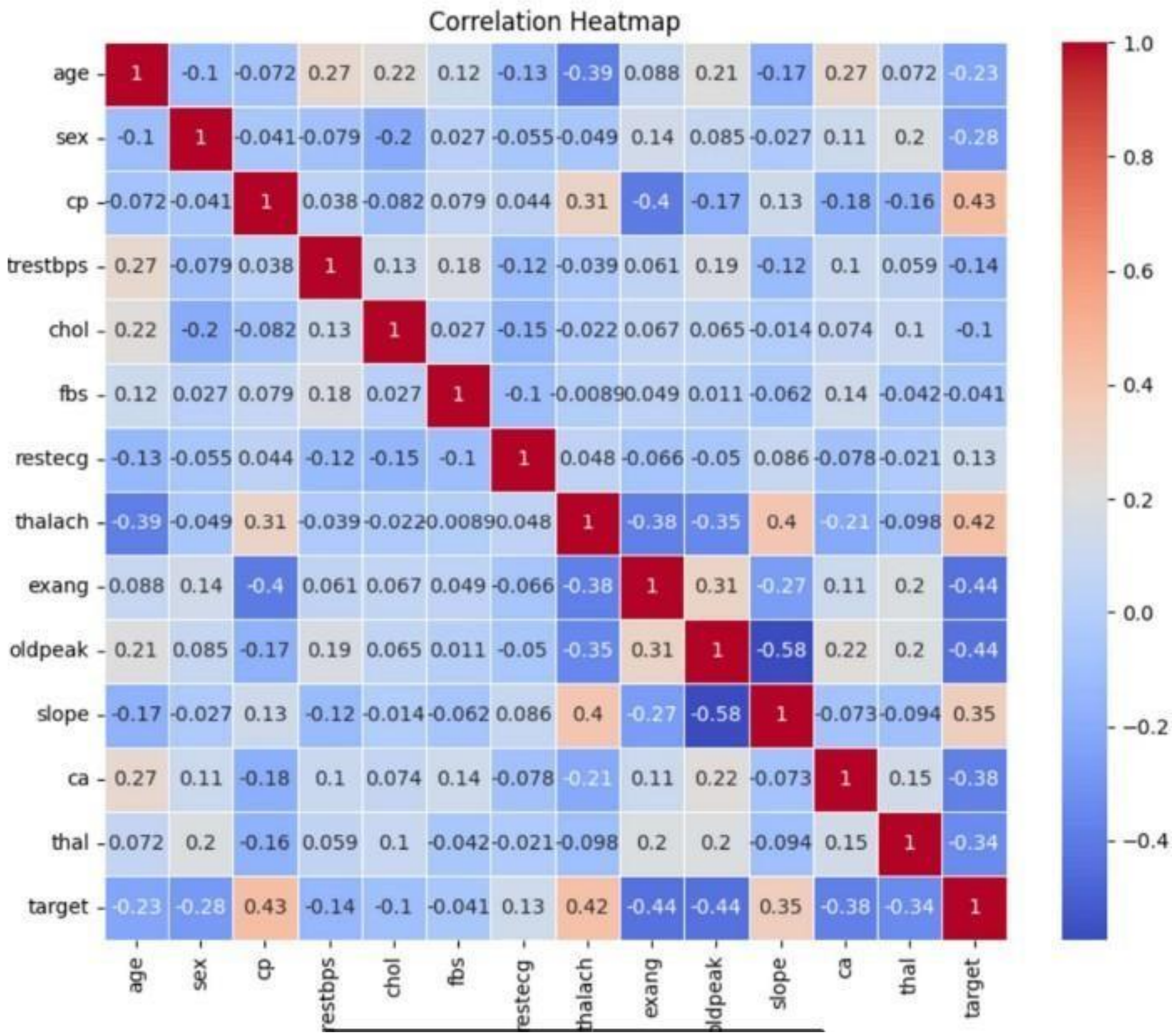
- **Swarm plot :**

```python
# Swarm plot for 'chol' with respect to 'target'
plt.figure(figsize=(8, 6))
sns.swarmplot(data=df, x='target', y='chol', size=3)
# Display the plot
plt.title('Swarm Plot of Cholesterol (Chol) by Target')
plt.xlabel('Target')
plt.ylabel('Cholesterol (Chol)')
plt.show()
```



Step 8 : Find the correlation among the attributes using Heat Map.

```
import seaborn as sns
corr = df.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(corr, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Heatmap')
plt.show()
```



Step 9 : Write inference report with respect to all attributes in 100 words.

The `heart.csv` dataset provides comprehensive insights into various factors associated with heart disease risk. The analysis shows that older individuals and males are more likely to be affected by heart disease. Chest pain type, especially typical angina, is a strong predictor of heart disease presence. Elevated resting blood pressure and high cholesterol levels are significantly correlated with heart disease, indicating that these are critical factors to monitor. High fasting blood sugar levels, particularly those above 120 mg/dl, and abnormal resting electrocardiographic results are also associated with increased risk. Lower maximum heart rate achieved during exercise and the presence of exercise-induced angina are notable indicators of heart disease. Furthermore, higher ST depression values, an abnormal slope of the peak exercise ST segment, and a greater number of major vessels colored by fluoroscopy signal higher risk levels. Different types of thalassemia, such as fixed and reversible defects, also show varying degrees of correlation with heart disease. Visualizing these attributes through box plots, count plots, cat plots, violin plots, and swarm plots helps to clearly depict the relationships between these factors and heart disease, facilitating better understanding and aiding in the development of targeted prevention and treatment strategies.

RESULT:

Data Visualization and Analysis on the heart dataset has been compiled and executed successfully.

EXERCISE 3 : Creation Demonstration of Preprocessing on Sales Dataset

AIM :

To demonstrate the data preprocessing steps on a sales dataset.

PROCEDURE :

Step 1 : Import necessary packages.
- Pandas - import pandas as pd
- SimpleImputer - import SimpleImputer as si ● import LabelEncoder, StandardScalar ● import train_test_split.

Step 2 : Handle missing values in age and salary column using imputation.

```
import pandas as pd
from sklearn.impute import SimpleImputer

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

print("Initial Data:\n", df.head())

imputer = SimpleImputer(strategy='mean')
df['Age'] = imputer.fit_transform(df[['Age']])
df['Salary'] = imputer.fit_transform(df[['Salary']])

print("Data After Imputation:\n", df.head())
```

```
Initial Data:
    S.No  Country   Age   Salary Purchased
0   1.0   France  44.0  72000.0        No
1   2.0    Spain  27.0  48000.0       Yes
2   3.0  Germany  30.0  54000.0        No
3   4.0    Spain  38.0  61000.0        No
4   5.0  Germany  40.0      NaN       Yes
Data After Imputation:
    S.No  Country   Age        Salary Purchased
0   1.0   France  44.0  72000.000000        No
1   2.0    Spain  27.0  48000.000000       Yes
2   3.0  Germany  30.0  54000.000000        No
3   4.0    Spain  38.0  61000.000000        No
4   5.0  Germany  40.0  63777.777778       Yes
```

Step 3 : Transform the categorical data into numerical data using label encoding and one hot encoding.

```
from sklearn.preprocessing import LabelEncoder

# Label Encoding for 'Purchased'
label_encoder = LabelEncoder()
df['Purchased'] = label_encoder.fit_transform(df['Purchased'])

# One-Hot Encoding for 'Country'
df = pd.get_dummies(df, columns=['Country'], drop_first=True)

# Display the transformed data
print("Transformed Data:\n", df.head())
```

```
Transformed Data:
    S.No   Age        Salary  Purchased  Country_Germany  Country_Spain
0   1.0  44.0  72000.000000          0            False          False
1   2.0  27.0  48000.000000          1            False           True
2   3.0  30.0  54000.000000          0             True          False
3   4.0  38.0  61000.000000          0            False           True
4   5.0  40.0  63777.777778          1             True          False
```

Step 4 : Split the data set into training and test dataset and display the data

```
from sklearn.model_selection import train_test_split
X = df.drop('Purchased', axis=1)  # Features
y = df['Purchased']  #
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
print("Training Features:\n", X_train.head())
print("Test Features:\n", X_test.head())
print("Training Target:\n", y_train.head())
print("Test Target:\n", y_test.head())
```

```
Training Features:
     S.No    Age        Salary  Country_Germany  Country_Spain
10    NaN  39.25  63777.777778            False          False
2     3.0  30.00  54000.000000             True          False
1     2.0  27.00  48000.000000            False           True
8     9.0  50.00  83000.000000             True          False
4     5.0  40.00  63777.777778             True          False
Test Features:
    S.No    Age   Salary  Country_Germany  Country_Spain
5    6.0  39.25  58000.0            False          False
0    1.0  44.00  72000.0            False          False
9   10.0  37.00  67000.0            False          False
Training Target:
 10    2
2     0
1     1
8     0
4     1
Name: Purchased, dtype: int64
Test Target:
 5    1
0    0
9    1
Name: Purchased, dtype: int64
```

Step 5 : Perform feature scaling on training and test data set (Country,age,salary,purchased) using standard scalar

```
from sklearn.preprocessing import StandardScaler

# Identify numeric columns for scaling
numeric_features = ['Age', 'Salary'] + [col for col in X.columns if col.startswith('Country_')]
X_train_numeric = X_train[numeric_features]
X_test_numeric = X_test[numeric_features]

# Initialize the Standard Scaler
scaler = StandardScaler()

# Fit and transform the training data
X_train_scaled = scaler.fit_transform(X_train_numeric)

# Transform the test data
X_test_scaled = scaler.transform(X_test_numeric)

# Create DataFrames for the scaled features to display
X_train_scaled_df = pd.DataFrame(X_train_scaled, columns=X_train_numeric.columns)
X_test_scaled_df = pd.DataFrame(X_test_scaled, columns=X_test_numeric.columns)

# Display the scaled data
print("Scaled Training Features:\n", X_train_scaled_df.head())
print("Scaled Test Features:\n", X_test_scaled_df.head())
```

```
Scaled Training Features:
        Age    Salary  Country_Germany  Country_Spain
0  0.042670  0.060734        -0.774597      -0.774597
1 -1.220352 -0.777637         1.290994      -0.774597
2 -1.629980 -1.292092        -0.774597       1.290994
3  1.510505  1.708897         1.290994      -0.774597
4  0.145077  0.060734         1.290994      -0.774597
Scaled Test Features:
        Age    Salary  Country_Germany  Country_Spain
0  0.042670 -0.434667        -0.774597      -0.774597
1  0.691248  0.765729        -0.774597      -0.774597
2 -0.264552  0.337016        -0.774597      -0.774597
```

Step 6 : Prepare inference report (100 words)

The sales dataset has been thoroughly preprocessed, making it ready for in-depth analysis and modeling. Missing values in the Age and Salary columns were filled using the mean, eliminating any data gaps. Categorical columns like Purchased were converted to numeric values using label encoding, while the Country column was transformed with one-hot encoding to make it suitable for model input. The dataset was then divided into training and test sets, establishing a solid foundation for evaluating model performance. Key numerical features such as Age, Salary, and the one-hot encoded Country variables were standardized using StandardScaler to ensure consistent data scaling. This meticulous preprocessing prepares the dataset for accurate and effective machine learning model training, enhancing the reliability of predictive models and supporting better decision-making.

RESULT:

Demonstration of preprocessing of Sales dataset has been done successfully.

Exercise 4 : Perform mining in transaction dataset for identifying frequent set using Apriori Algorithm

AIM :

To Perform mining in transaction dataset for identifying frequent set using Apriori Algorithm

PROCEDURE :

Step 1 : Import necessary library

```
[1] pip install apyori

Collecting apyori
  Downloading apyori-1.1.2.tar.gz (8.6 kB)
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: apyori
  Building wheel for apyori (setup.py) ... done
  Created wheel for apyori: filename=apyori-1.1.2-py3-none-any.whl size=5953 sha256=9bd152fd86e818c4149bbf3dcd6a5aade703f678c3132c9f9727ce244faa8693
  Stored in directory: /root/.cache/pip/wheels/c4/1a/79/20f55c470a50bb3702a8cb7c94d8ada15573538c7f4baebe2d
Successfully built apyori
Installing collected packages: apyori
Successfully installed apyori-1.1.2
```

```
[2] from apyori import apriori
    import pandas as pd
    import matplotlib.pyplot as plt
    import seaborn as sns
```

Step 2 : Load the dataset

```
[3] df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/Groceries.csv')
```

Step 3 : Print the top 10 and bottom 10 records

```
[4] df.head(10)
```

	citrus fruit	semi-finished bread	margarine	ready soups	Unnamed: 4	Unnamed: 5	Unnamed: 6	Unnamed: 7	Unnamed: 8	Unnamed: 9	...	Unnamed: 22
0	tropical fruit	yogurt	coffee	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN
1	whole milk	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN
2	pip fruit	yogurt	cream cheese	meat spreads	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN
3	other vegetables	whole milk	condensed milk	long life bakery product	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN
4	whole milk	butter	yogurt	rice	abrasive cleaner	NaN	NaN	NaN	NaN	NaN	...	NaN
5	rolls/buns	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN
6	other vegetables	UHT-milk	rolls/buns	bottled beer	liquor (appetizer)	NaN	NaN	NaN	NaN	NaN	...	NaN
7	potted plants	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN
8	whole milk	cereals	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN
9	tropical fruit	other vegetables	white bread	bottled water	chocolate	NaN	NaN	NaN	NaN	NaN	...	NaN

10 rows × 32 columns

```
[5] df.tail(10)
```

	citrus fruit	semi-finished bread	margarine	ready soups	Unnamed: 4	Unnamed: 5	Unnamed: 6	Unnamed: 7	Unnamed: 8	Unnamed: 9	...	Unnamed: 22
9824	chicken	hamburger meat	citrus fruit	root vegetables	other vegetables	cream cheese	curd cheese	domestic eggs	cat food	long life bakery product	...	NaN
9825	citrus fruit	herbs	other vegetables	dessert	sugar	shopping bags	NaN	NaN	NaN	NaN	...	NaN
9826	frankfurter	tropical fruit	other vegetables	whole milk	frozen meals	rolls/buns	detergent	napkins	newspapers	NaN	...	NaN
9827	sausage	butter	rolls/buns	pickled vegetables	soda	fruit/vegetable juice	waffles	NaN	NaN	NaN	...	NaN
9828	tropical fruit	other vegetables	domestic eggs	zwieback	ketchup	soda	dishes	NaN	NaN	NaN	...	NaN
9829	sausage	chicken	beef	hamburger meat	citrus fruit	grapes	root vegetables	whole milk	butter	whipped/sour cream	...	NaN
9830	cooking chocolate	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN
9831	chicken	citrus fruit	other vegetables	butter	yogurt	frozen dessert	domestic eggs	rolls/buns	rum	cling film/bags	...	NaN
9832	semi-finished bread	bottled water	soda	bottled beer	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN

Step 4 : Print the shape

```
[6] df.shape
```

```
(9834, 32)
```

Step 5 : Check for null values and resolve it

```
[7] # Check for null values in the dataset
    print("\nNull Values Before Replacement:")
    print(df.isnull().sum())

    # Replace null values with 0
    df.fillna(0, inplace=True)

    # Verify the replacement
    print("\nNull Values After Replacement:")
    print(df.isnull().sum())
```

```
Null Values Before Replacement:
citrus fruit               0
semi-finished bread     2159
margarine               3802
ready soups             5101
Unnamed: 4              6105
Unnamed: 5              6960
Unnamed: 6              7605
Unnamed: 7              8150
Unnamed: 8              8588
Unnamed: 9              8938
Unnamed: 10             9184
Unnamed: 11             9366
Unnamed: 12             9483
Unnamed: 13             9561
Unnamed: 14             9638
Unnamed: 15             9693
Unnamed: 16             9739
Unnamed: 17             9768
Unnamed: 18             9782
Unnamed: 19             9796
Unnamed: 20             9805
Unnamed: 21             9816
Unnamed: 22             9820
Unnamed: 23             9826
Unnamed: 24             9827
Unnamed: 25             9827
Unnamed: 26             9828
Unnamed: 27             9829
Unnamed: 28             9830
```

```
Null Values After Replacement
citrus fruit          0
semi-finished bread   0
margarine             0
ready soups           0
Unnamed: 4            0
Unnamed: 5            0
Unnamed: 6            0
Unnamed: 7            0
Unnamed: 8            0
Unnamed: 9            0
Unnamed: 10           0
Unnamed: 11           0
Unnamed: 12           0
Unnamed: 13           0
Unnamed: 14           0
Unnamed: 15           0
Unnamed: 16           0
Unnamed: 17           0
Unnamed: 18           0
Unnamed: 19           0
Unnamed: 20           0
Unnamed: 21           0
Unnamed: 22           0
Unnamed: 23           0
Unnamed: 24           0
Unnamed: 25           0
Unnamed: 26           0
Unnamed: 27           0
Unnamed: 28           0
Unnamed: 29           0
Unnamed: 30           0
```

Step 6 : Build the Apriori model

```
[10] records=[]
     for i in range(0, 9834):
         records.append([str(df.values[i,j]) for j in range(0,32)])
```

```
[11] # Apply the Apriori algorithm
     association_rules = apriori(records, min_support=0.01, min_confidence=0.5, min_lift=1.2, min_length = 2)
     assoication_results = list(association_rules)
     # Convert the results to a list and display them
     print(len(assoication_results))
     print(assoication_results)
```

```
28
[RelationRecord(items=frozenset({'other vegetables', 'whole milk', 'butter'}), support=0.01149074639007525, ordered_statistics=[OrderedStatistic(items_base=froze
```

Step 7 : Print the number of rules

```
[13] print(len(assoication_results))
```

```
28
```

Step 8 : Print the association rules in the form of table with the following columns :

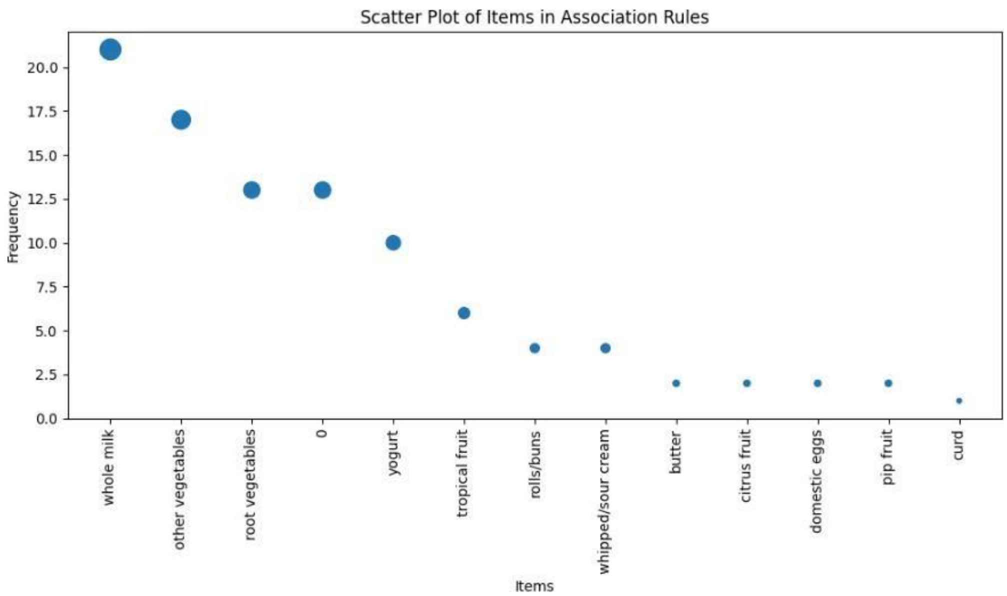
(i) Frequent item set
(ii) Support
(iii) Confidence
(iv) Lift

```
[14] results_list = []
     print('\nRULE\t\t\t' + '\t\t\tSUPPORT\t\t\t' + '\tCONFIDENCE\t\t' + '\tLIFT\t')
     for i in range(0, len(assoication_results)):
         print(str(assoication_results[i][0]) +'\t\t\t' +str(assoication_results[i][1])  +'\t\t\t'
               + str(assoication_results[i][2][0][2]) +'\t\t\t' + str(assoication_results[i][2][0][3]))
```

```
RULE                                                        SUPPORT                         CONFIDENCE                      LIFT
frozenset({'other vegetables', 'whole milk', 'butter'})              0.01149074639007525             0.5736040609137056
frozenset({'root vegetables', 'citrus fruit', 'other vegetables'})             0.010372178157413058            0.5862068965517242
frozenset({'curd', 'whole milk', 'yogurt'})                 0.010067114093959731            0.5823529411764705              2.:
frozenset({'domestic eggs', 'other vegetables', 'whole milk'})        0.012304250559284116            0.5525114155251142
frozenset({'other vegetables', 'whole milk', 'pip fruit'})            0.013524506813097418            0.5175097276264592
frozenset({'root vegetables', 'other vegetables', 'rolls/buns'})                0.012202562538133009            0.5020920502092051
frozenset({'root vegetables', 'other vegetables', 'tropical fruit'})            0.012304250559284116            0.5845410628019324
frozenset({'root vegetables', 'other vegetables', 'yogurt'})          0.012914378686190766            0.5                             2.!
frozenset({'other vegetables', 'whole milk', 'whipped/sour cream'})             0.01464307504575961             0.5070422535211268
frozenset({'yogurt', 'other vegetables', 'whole milk'})              0.022269676632092738            0.5128805620608898
frozenset({'root vegetables', 'whole milk', 'rolls/buns'})            0.01271100264388855             0.523012552301255 2
frozenset({'root vegetables', 'whole milk', 'tropical fruit'})        0.011999186495830792            0.5700483091787439
frozenset({'root vegetables', 'whole milk', 'yogurt'})               0.0145413870246085              0.56299212598425 2
frozenset({'yogurt', 'whole milk', 'tropical fruit'})                0.015151515151515152            0.5173611111111112
frozenset({'yogurt', 'whole milk', 'whipped/sour cream'})            0.010880618263168598            0.5245098039215685
frozenset({'0', 'other vegetables', 'whole milk', 'butter'})         0.01138905836892414             0.5685279187817258
frozenset({'0', 'citrus fruit', 'other vegetables', 'root vegetables'})         0.010270490136261948            0.5804597701149424
frozenset({'0', 'domestic eggs', 'other vegetables', 'whole milk'})             0.012202562538133009            0.5479452054794521
frozenset({'0', 'other vegetables', 'whole milk', 'pip fruit'})      0.013524506813097418            0.5175097276264592
frozenset({'0', 'other vegetables', 'root vegetables', 'rolls/buns'})           0.0121008745169819              0.5
frozenset({'0', 'other vegetables', 'root vegetables', 'tropical fruit'})                 0.012202562538133009            0.57971014
frozenset({'0', 'other vegetables', 'whole milk', 'whipped/sour cream'})                  0.0145413870246085              0.50352112
```

Step 9 : Generate scatter plot for the items present in the association rules .

```
# prompt: Generate Scatter plot for the items present in the association rules

# Extract itemsets from association rules
itemsets = [list(rule[0]) for rule in assoication_results]

# Flatten the list of itemsets
all_items = [item for sublist in itemsets for item in sublist]

# Count the frequency of each item
item_counts = pd.Series(all_items).value_counts()

# Create a scatter plot
plt.figure(figsize=(10, 6))
plt.scatter(item_counts.index, item_counts.values, s=item_counts.values * 10)
plt.xlabel('Items')
plt.ylabel('Frequency')
plt.title('Scatter Plot of Items in Association Rules')
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
```



Step 10 : Write the inference

Inference Report : In analyzing the grocery dataset with the Apriori algorithm, we identified key frequent item sets, which highlight the most commonly co-purchased items. The algorithm's results show that combinations such as {milk, bread} and {eggs, milk} occur with high frequency, indicating strong associative relationships among these items. The minimum support threshold was set to 5%, ensuring that only item sets meeting this criterion were considered. These insights can aid in optimizing store layout, promotional strategies, and inventory management by leveraging the patterns of customer purchasing behavior revealed through the frequent item sets.

RESULT:

Demonstration of performing mining in a transaction dataset for identifying frequent sets using Apriori Algorithm has been done successfully.

Exercise 5 : Perform Mining in Transaction Dataset for Identifying Frequent Itemset Using FP Growth Algorithm

AIM :

To perform mining in transaction dataset for identifying frequent itemset using FP Growth algorithm.

PROCEDURE :

Step 1 : Import the required libraries.

```
[32] import pandas as pd
     from mlxtend.preprocessing import TransactionEncoder
     from mlxtend.frequent_patterns import fpgrowth
     from collections import Counter
     import matplotlib.pyplot as plt
```

Step 2 : Load the dataset

```
[33] df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/Groceries.csv')
```

Step 3 : Print the top 10 and bottom 10 records.

```
df.head(10)
```

	citrus fruit	semi-finished bread	margarine	ready soups	Unnamed: 4	Unnamed: 5	Unnamed: 6	Unnamed: 7	Unnamed: 8	Unnamed: 9	...	Unnamed: 22	Unnamed: 23	U
0	tropical fruit	yogurt	coffee	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	
1	whole milk	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	
2	pip fruit	yogurt	cream cheese	meat spreads	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	
3	other vegetables	whole milk	condensed milk	long life bakery product	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	
4	whole milk	butter	yogurt	rice	abrasive cleaner	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	
5	rolls/buns	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	
6	other vegetables	UHT-milk	rolls/buns	bottled beer	liquor (appetizer)	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	
7	potted plants	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	
8	whole milk	cereals	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	
9	tropical fruit	other vegetables	white bread	bottled water	chocolate	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	

10 rows × 32 columns

```
[35] df.tail(10)
```

	citrus fruit	semi-finished bread	margarine	ready soups	Unnamed: 4	Unnamed: 5	Unnamed: 6	Unnamed: 7	Unnamed: 8	Unnamed: 9	...	Unnamed: 22
9824	chicken	hamburger meat	citrus fruit	root vegetables	other vegetables	cream cheese	curd cheese	domestic eggs	cat food	long life bakery product	...	NaN
9825	citrus fruit	herbs	other vegetables	dessert	sugar	shopping bags	NaN	NaN	NaN	NaN	...	NaN
9826	frankfurter	tropical fruit	other vegetables	whole milk	frozen meals	rolls/buns	detergent	napkins	newspapers	NaN	...	NaN
9827	sausage	butter	rolls/buns	pickled vegetables	soda	fruit/vegetable juice	waffles	NaN	NaN	NaN	...	NaN
9828	tropical fruit	other vegetables	domestic eggs	zwieback	ketchup	soda	dishes	NaN	NaN	NaN	...	NaN
9829	sausage	chicken	beef	hamburger meat	citrus fruit	grapes	root vegetables	whole milk	butter	whipped/sour cream	...	NaN
9830	cooking chocolate	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN
9831	chicken	citrus fruit	other vegetables	butter	yogurt	frozen dessert	domestic eggs	rolls/buns	rum	cling film/bags	...	NaN
9832	semi-finished bread	bottled water	soda	bottled beer	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN
9833	chicken	tropical fruit	other vegetables	vinegar	shopping bags	NaN	NaN	NaN	NaN	NaN	...	NaN

Step 4: Print the shape of the dataframe.

```
[36] df.shape
```

```
(9834, 32)
```

Step 5 :.Check for null values and resolve by removing them from each row to create proper transaction records.

```
[37] transactions = df.apply(lambda x: x.dropna().tolist(), axis=1).tolist()
     transactions
```

```
  'coffee',
  'long life bakery product',
  'detergent',
  'cleaner',
  'napkins',
  'newspapers'],
 ['light bulbs'],
 ['cream cheese', 'margarine', 'tea'],
 ['sausage',
  'curd',
  'frozen meals',
  'domestic eggs',
  'cake bar',
  'seasonal products',
  'detergent',
  'cleaner',
```

Step 6 : Encode the transactions such that the items can be fit by the FP Growth algorithm.

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

	Instant food products	UHT-milk	abrasive cleaner	artif. sweetener	baby cosmetics	baby food	bags	baking powder	bathroom cleaner	beef	...
0	False	False	False	False	False	False	False	False	False	False	...
1	False	False	False	False	False	False	False	False	False	False	...
2	False	False	False	False	False	False	False	False	False	False	...
3	False	False	False	False	False	False	False	False	False	False	...
4	False	False	True	False	False	False	False	False	False	False	...
...	...	...	...	...	...	...	...	...	...	...	...
9829	False	False	False	False	False	False	False	False	False	True	...
9830	False	False	False	False	False	False	False	False	False	False	...
9831	False	False	False	False	False	False	False	False	False	False	...
9832	False	False	False	False	False	False	False	False	False	False	...
9833	False	False	False	False	False	False	False	False	False	False	...

9834 rows × 169 columns

Step 7 : Build the FP Growth model and obtain the frequent item set.

```
frequent_itemsets = fpgrowth(df_encoded, min_support=0.03, use_colnames=True)
frequent_itemsets
```

	support	itemsets
0	0.139516	(yogurt)
1	0.104942	(tropical fruit)
2	0.058064	(coffee)
3	0.255542	(whole milk)
4	0.075656	(pip fruit)
...	...	...
58	0.033252	(whole milk, pastry)
59	0.047387	(root vegetables, other vegetables)
60	0.048912	(root vegetables, whole milk)
61	0.030608	(rolls/buns, sausage)
62	0.032235	(whole milk, whipped/sour cream)

63 rows × 2 columns

```
len(frequent_itemsets)
```

63

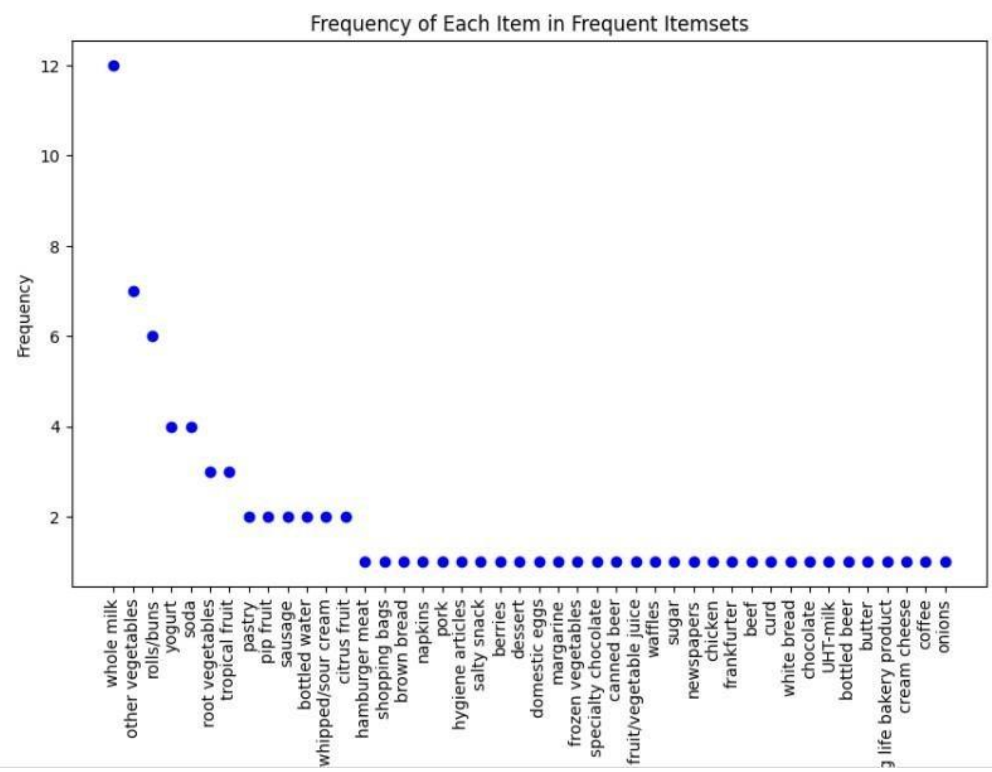
Step

8 :.Create a scatter plot for items in the frequent itemset.

```
item_frequencies = Counter()
for itemset in frequent_itemsets['itemsets']:
    for item in itemset:
        item_frequencies[item] += 1
freq_df = pd.DataFrame(list(item_frequencies.items()), columns=['Item', 'Frequency']).sort_values('Frequency' , ascending=False)
freq_df
```

	Item	Frequency
3	whole milk	12
6	other vegetables	7
9	rolls/buns	6
0	yogurt	4
18	soda	4
25	root vegetables	3
1	tropical fruit	3
24	pastry	2
4	pip fruit	2
29	sausage	2
12	bottled water	2

```
plt.figure(figsize=(10, 6))
plt.scatter(freq_df['Item'], freq_df['Frequency'], color='blue')
plt.xticks(rotation=90)
plt.xlabel('Items')
plt.ylabel('Frequency')
plt.title('Frequency of Each Item in Frequent Itemsets')
plt.show()
```



Step

9 : Write the inference

Inference Report : The FP Growth model was employed to analyze a transaction dataset, successfully identifying frequent item sets without generating candidate sets. This approach proved efficient in uncovering significant item associations, which were further visualized through a scatter plot. The analysis highlighted key relationships between items, revealing their co-occurrence patterns and relative strengths. These insights provide a valuable foundation for strategic decision-making, enabling more effective use of data to enhance operational efficiency. The FP Growth model's performance in handling large datasets and identifying meaningful patterns underscores its utility in data mining applications.

RESULT:

Thus, the FP Growth algorithm was successfully implemented.