CONTENTS

1. Prepare a dataset of customer having the features date, price, product_id, quantity_purchased, serial_no, user_id,user_type, user_class, purchase_week.

```
import pandas as pd
import random
import datetime

# Define the number of records in the dataset
num_records = 30

# Generate random data for each feature
dates = pd.date_range(start='2022-01-01', end='2022-12-31',
periods=num_records)
prices = [round(random.uniform(10, 100), 2) for _ in range(num_records)]
product_ids = [random.randint(1, 100) for _ in range(num_records)]
quantities = [random.randint(1, 10) for _ in range(num_records)]
serial_nos = [f'SN-{random.randint(1000, 9999)}' for _ in
range(num_records)]
user_ids = ['U' + str(random.randint(10, 20)) for _ in
range(num_records)]
user_types = ['Retail', 'Wholesale']
user_classes = ['Class A', 'Class B', 'Class C']
purchase_weeks = [date.isocalendar()[1] for date in dates]

# Create a dictionary with the data
data = {
    'Date': dates,
    'Price': prices,
    'Product_ID': product_ids,
    'Quantity_Purchased': quantities,
    'Serial_No': serial_nos,
    'User_ID': user_ids,
    'User_Type': random.choices(user_types, k=num_records),
    'User_Class': random.choices(user_classes, k=num_records),
    'Purchase_Week': purchase_weeks
}

# Create a DataFrame from the dictionary
df = pd.DataFrame(data)

# Print the first few rows of the dataset
print(df.head())

# Save the dataset to a CSV file
df.to_csv('customer_dataset.csv', index=False)
```

Output

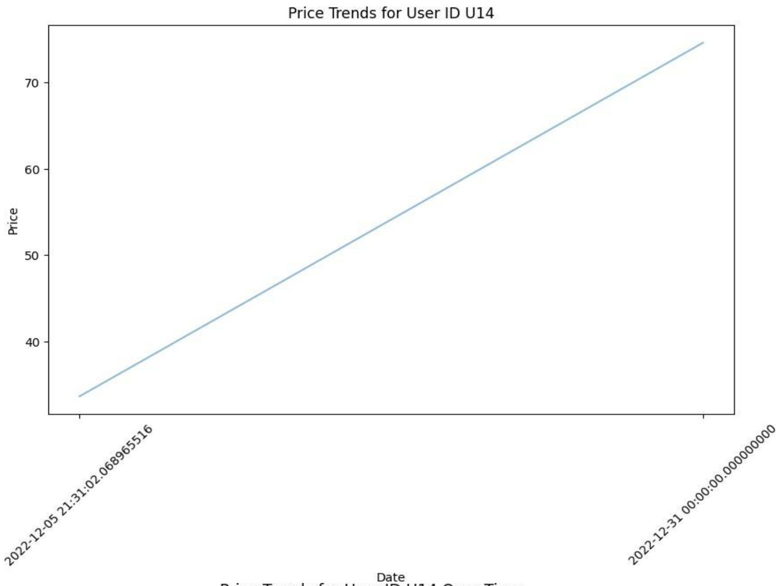
```
      Date                       Price    Product_ID   Quantity_Purchased  
\
0 2022-01-01 00:00:00.000000000  38.45          1                    3
1 2022-01-13 13:14:28.965517241  42.59         19                    3
2 2022-01-26 02:28:57.931034482  36.54         59                    3
3 2022-02-07 15:43:26.896551724  37.79         97                    1
4 2022-02-20 04:57:55.862068965  84.97         78                   10
```

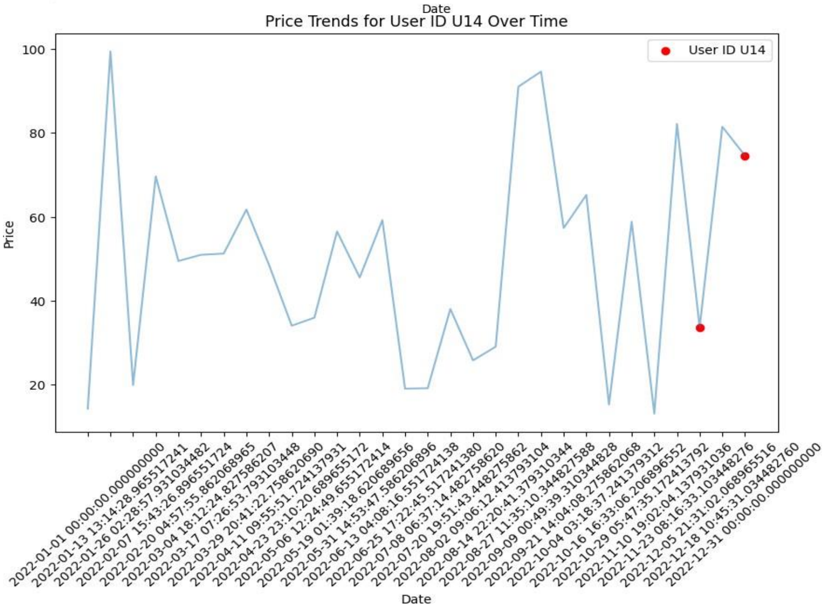
```
   Serial_No User_ID  User_Type User_Class  Purchase_Week
0   SN-7352     U15  Wholesale    Class B             52
1   SN-9381     U15  Wholesale    Class B              2
2   SN-1399     U11     Retail    Class B              4
3   SN-5305     U14  Wholesale    Class A              6
4   SN-7219     U16  Wholesale    Class B              7
```

a) Plot diagram for Price Trends for Particular User, Price Trends for Particular User Over Time using above dataset.

```
import pandas as pd
import matplotlib.pyplot as plt

# Read the dataset from the CSV file
df = pd.read_csv('customer_dataset.csv')

# Define the user ID for analysis
user_id = 'U14'

# Filter the dataset for the particular user
user_data = df[df['User_ID'] == user_id]

# Plot the price trends for the particular user
plt.figure(figsize=(10, 6))
plt.plot(user_data['Date'], user_data['Price'], alpha=0.5)
plt.xlabel('Date')
plt.ylabel('Price')
plt.title(f'Price Trends for User ID {user_id}')
plt.xticks(rotation=45)
plt.show()

# Plot the price trends for the particular user over time
plt.figure(figsize=(10, 6))
plt.plot(df['Date'], df['Price'], alpha=0.5)
plt.scatter(user_data['Date'], user_data['Price'], color='red',
label=f'User ID {user_id}')
plt.xlabel('Date')
plt.ylabel('Price')
plt.title(f'Price Trends for User ID {user_id} Over Time')
plt.xticks(rotation=45)
plt.legend()
plt.show()
```

Output

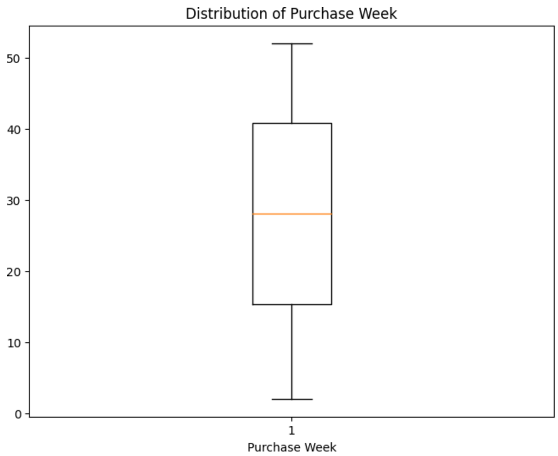

Price Trends for User ID U14
Price
70
60
50
40
2022-12-05 21:31:02.068965516
2022-12-31 00:00:00.000000000
Date


Price Trends for User ID U14 Over Time
User ID U14
Price
100
80
60
40
20
2022-01-01 00:00:00.000000000
2022-01-13 13:14:28.965517241
2022-01-26 02:28:57.931034482
2022-02-07 15:43:26.896551724
2022-02-20 04:57:55.862068965
2022-03-04 18:12:24.827586207
2022-03-17 07:26:53.793103448
2022-03-29 20:41:22.758620690
2022-04-11 09:55:51.724137931
2022-04-23 23:10:20.689655172
2022-05-06 12:24:49.655172414
2022-05-19 01:39:18.620689656
2022-05-31 14:53:47.586206896
2022-06-13 04:08:16.551724138
2022-06-25 17:22:45.517241380
2022-07-08 06:37:14.482758620
2022-07-20 19:51:43.448275862
2022-08-02 09:06:12.413793104
2022-08-14 22:20:41.379310344
2022-08-27 11:35:10.344827588
2022-09-09 00:49:39.310344828
2022-09-21 14:04:08.275862068
2022-10-04 03:18:37.241379312
2022-10-16 16:33:06.206896552
2022-10-29 05:47:35.172413792
2022-11-10 19:02:04.137931036
2022-11-23 08:16:33.103448276
2022-12-05 21:31:02.068965516
2022-12-18 10:45:31.034482760
2022-12-31 00:00:00.000000000
Date

b) Create box plot Quantity and Week value distribution having parameters of quantity_purchased','purchase_week'.

```
import pandas as pd
import matplotlib.pyplot as plt

# Load the dataset from the CSV file
df = pd.read_csv('customer_dataset.csv')
# Create a box plot for Quantity_Purchased
plt.figure(figsize=(8, 6))
plt.boxplot(df['Quantity_Purchased'])
plt.xlabel('Quantity Purchased')
plt.title('Distribution of Quantity Purchased')
plt.show()

# Create a box plot for Purchase_Week
plt.figure(figsize=(8, 6))
plt.boxplot(df['Purchase_Week'])
plt.xlabel('Purchase Week')
plt.title('Distribution of Purchase Week')
plt.show()
```

Output



2. Write a program to Transforming Nominal Features, Transforming Ordinal Features and Encoding Categorical Features using one-hot Encoding Scheme.

```
import pandas as pd

# Load the dataset from the CSV file
df = pd.read_csv('/home/shyma/Desktop/Machinelearning/pokemon.csv',
encoding='utf-8')

# Transforming nominal features
nominal_features = ['Generation']
df_nominal = pd.get_dummies(df[nominal_features])

# Transforming ordinal features
gen_ord_map = {'Gen1': 1, 'Gen2': 2, 'Gen3': 3,
               'Gen4': 4, 'Gen5': 5, 'Gen6': 6}

df['GenerationLabel'] = df['Generation'].map(gen_ord_map)

# Encoding categorical features using one-hot encoding scheme
categorical_features = ['Generation', 'Legendary']
df_encoded = pd.get_dummies(df[categorical_features])

# Combine the transformed and encoded features with the original dataset
df_transformed = pd.concat([df, df_nominal, df_encoded], axis=1)

# Print the transformed dataset
print(df_transformed.head())
```

Output

```
   #                  Name Type 1  Type 2  Total  HP  Attack  Defense  \
0  1             Bulbasaur  Grass  Poison    318  45      49       49
1  2               Ivysaur  Grass  Poison    405  60      62       63
2  3              Venusaur  Grass  Poison    525  80      82       83
3  3  VenusaurMega Venusaur  Grass  Poison    625  80     100      123
4  4            Charmander   Fire     NaN    309  39      52       43

   Sp. Atk  Sp. Def  Speed  Generation  Legendary  GenerationLabel  \
0       65       65     45           1      False              NaN
1       80       80     60           1      False              NaN
2      100      100     80           1      False              NaN
3      122      120     80           1      False              NaN
4       60       50     65           1      False              NaN

   Generation  Generation  Legendary
0           1           1      False
1           1           1      False
2           1           1      False
3           1           1      False
4           1           1      False
```

3. Write a program to implement Raw Measures such as Values, count, Binarization, Rounding, Interactions, Binning, Fixed-width binning,Quantile based binning and Mathematical Transformations such as Log transform, Box-Cox transform.

```
import pandas as pd
import numpy as np
from scipy import stats

# Load the dataset from the CSV file

df = pd.read_csv("/home/shyma/Desktop/Machinelearning/ML Data/bank-
data.csv")

# Raw Measures - Values and Count
print("Raw Measures - Values and Count:")
print(df['married'].values)
print(df['married'].count())
print()

# Binarization
threshold = 5
df['married'] = np.where(df['married'] > threshold, 1, 0)
print("Binarization:")
print(df['married'].values)
print()

# Rounding
df['income_Rounded'] = df['income'].round(decimals=2)
print("Rounding:")
print(df['income_Rounded'].values)
print()

# Interactions
df['Interaction'] = df['save_act'] * df['income']
print("Interactions:")
print(df['Interaction'].values)
print()

# Binning - Fixed-width binning
bin_width = 10
df['Income_Bin'] = pd.cut(df['income'], bins=np.arange(0,
df['income'].max() + bin_width, bin_width), right=False)
print("Binning - Fixed-width binning:")
print(df['Income_Bin'].values)
print()

# Binning - Quantile based binning
num_bins = 3
df['Income_Quantile_Bin'] = pd.qcut(df['income'], q=num_bins,
labels=False)
print("Binning - Quantile based binning:")
print(df['Income_Quantile_Bin'].values)
print()

# Mathematical Transformations - Log transform
df['Income_Log_Transformed'] = np.log(df['income'])
```

```python
print("Mathematical Transformations - Log transform:")
print(df['Income_Log_Transformed'].values)
print()

# Mathematical Transformations - Box-Cox transform
df['Income_BoxCox_Transformed'], _ = stats.boxcox(df['income'])
print("Mathematical Transformations - Box-Cox transform:")
print(df['Income_BoxCox_Transformed'].values)
print()
```

Output

```
Raw Measures - Values and Count:
[0 1 1 1 1 1 0 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 0 1 1 1 1 0 0 1 1
 0 1 0 1 1 0 1 0 1 0 0 0 1 0 1 1 0 1 0 0 1 1 0 1 0 1 1 0 1 0 0 0 1 1 0 1 1
 0 0 0 1 1 1 0 0 1 1 1 1 1 0 1 1 1 0 0 1 1 1 1 1 0 1 1 1 1 0 1 1 1 1 0 1 0
 1 1 0 1 1 1 1 0 1 0 1 1 0 0 1 1 1 1 0 0 1 0 1 1 1 1 0 1 0 1 1 1 1 1 1 1 0
 1 1 1 0 1 1 0 1 1 1 1 1 1 1 0 0 1 1 0 0 0 1 1 1 1 1 1 0 1 1 1 1 1 0 0 1 0
 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 0 1 0 0 1 1 0 0 1 1 0 1 1 1 1 0 1 0 1
 1 1 1 0 1 0 0 0 1 0 1 1 0 0 0 1 1 1 1 0 1 1 1 0 0 1 1 1 0 1 1 1 0 0 0 1 0
 0 1 1 1 1 0 1 0 0 1 0 0 0 1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1
 0 1 0 1 1 1 1 1 0 1 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 1 1 0 1 0 1 1 0 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0 0 1 1 1 0 0 1 0 1 1 1 1 1 0 1 1 1
 1 0 0 1 1 1 0 0 1 1 0 1 1 1 1 0 0 0 0 1 1 0 1 1 0 1 1 1 0 1 0 0 1 0 1 1 0
 0 1 1 0 0 0 1 1 1 0 1 1 1 1 0 1 1 0 1 1 1 1 0 1 1 0 1 1 0 1 0 1 1 1 1 1 0
 1 1 1 1 1 0 1 0 0 0 0 1 1 1 0 1 1 1 1 1 0 1 1 0 1 0 1 1 1 0 0 1 0 1 1 0 1
 0 0 1 1 1 1 1 1 1 1 0 1 0 1 1 0 0 0 1 1 0 1 0 1 1 1 0 0 1 1 1 1 0 0 1 0 1
 0 1 1 1 0 1 1 1 0 0 1 1 1 0 0 1 0 1 0 0 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1
 1 1 1 1 1 0 0 1 0 1 1 1 1 1 0 0 1 0 1 1 1 1 1 0 1 1 1 0 0 1 0 1 0 1 1 1 0
 1 1 1 0 1 1 1 0]
600

Binarization:
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0]

Rounding:
[17546.   30085.1  16575.4  20375.4  50576.3  37869.6   8877.07 24946.6
 25304.3  24212.1  59803.9  26658.8  15735.8  55204.7  19474.6  22342.1
 17729.8  41016.   26909.2  22522.8  57880.7  16497.3  38446.6  15538.8
 12640.3  41034.   20809.7  20114.   29359.1  24270.1  22942.9  16325.8
 23443.2  29921.3  37521.9  19868.   10953.   13381.   18504.3  25391.5
 26774.2  26952.6  55716.5  27571.5  13740.   52670.6  13283.9  13106.6
 39547.8  17867.3  14309.7  23894.8  16259.7  29794.1  56842.5  47835.8
```

24977.5	23124.9	15143.8	25334.3	24763.3	36589.	27022.6	11700.4
5014.21	17390.1	10861.	34892.9	19403.1	10441.9	14064.9	8062.73
31982.	23197.5	52674.	35610.5	26948.	49456.7	14724.5	34524.9
22052.1	27808.1	12591.4	16394.4	24026.1	31683.1	15525.	22562.2
15848.7	31095.6	24814.5	25429.3	34866.5	42579.1	41127.4	9990.11
7948.62	30870.8	12125.8	15348.9	26707.9	11604.4	15499.9	33088.5
34513.6	32395.5	46633.	13039.9	12681.9	24031.5	37330.5	25333.2
37094.2	33630.6	43228.2	47796.8	21730.3	10044.1	17270.1	45765.
29525.5	54863.8	20799.	33028.3	45031.9	39010.8	25257.7	42603.9
14092.7	21350.3	23246.4	41609.5	16716.1	36436.4	59503.8	31334.8
14048.9	39205.3	42173.9	55263.	37095.2	22791.4	17240.6	48974.8
18923.	51204.2	20236.2	18860.3	25732.5	28240.4	28193.6	36432.8
54618.8	24760.8	23356.1	8143.75	26462.5	20467.3	21506.2	15315.3
18875.7	12977.2	20708.5	7549.38	24904.	24071.8	9589.91	8562.86
26707.5	34020.5	49175.7	19726.3	24346.6	26999.4	41558.1	56340.3
37558.5	30099.3	15254.8	36086.1	17655.	56658.9	37706.5	18516.
29622.	32669.9	18275.5	34410.	34866.9	21796.6	63130.1	14996.4
49024.9	16249.8	36192.1	17839.9	18802.4	48720.3	14585.9	20819.
26077.8	41627.1	16977.3	19012.8	12764.8	14388.6	59409.1	14960.2
39666.6	20771.9	24474.1	33123.7	14433.4	13175.5	9824.37	17610.3
15156.2	31774.1	31693.5	28598.7	26261.7	42124.1	39308.7	43530.
49874.4	27434.8	50474.6	24888.2	28021.6	12279.5	30189.4	28969.4
14058.5	30404.3	41438.2	16711.3	52255.9	17866.9	18067.5	12823.7
11299.3	56031.1	35263.5	19968.1	27825.5	37773.9	7606.25	21384.4
20347.	21332.3	57671.7	36057.8	14290.5	17882.9	10629.1	24262.8
26097.9	23371.	21495.6	12166.9	17180.2	28882.3	21612.2	46358.4
19166.	17921.8	33229.	30396.1	34625.2	16672.8	60747.5	56394.3
13236.4	28409.4	27056.5	9362.58	28702.7	22366.1	24477.5	36972.4
22327.8	15610.2	54314.5	39175.8	13739.	9485.84	24675.7	28253.6
14136.5	37162.1	13519.2	39253.6	46323.8	20950.7	22495.7	32548.9
24583.4	8639.24	17139.5	13667.7	8162.42	15349.6	29231.4	41462.3
57398.1	11520.8	52117.3	26281.4	25683.4	11920.7	30658.7	36646.4
30760.4	16109.9	18036.7	42628.3	22110.1	37689.1	23171.8	21951.3
38103.4	22882.9	11043.7	24027.6	28495.1	9465.21	34852.3	21268.4
50849.2	18555.9	52769.9	11601.4	29541.7	17861.	21042.	26688.1
26900.6	38080.9	37554.1	18184.6	28864.9	48346.1	53104.3	19416.8
23638.1	42378.2	39745.3	45189.8	37930.9	24042.	31207.1	24424.3
24607.8	43057.	30198.5	50186.1	22916.1	9592.73	34253.6	22792.3
51620.8	19918.9	29625.1	12549.	51299.3	17364.8	29866.9	47750.2
11281.5	34073.8	46870.4	38453.7	7756.36	28413.8	47198.6	20866.3
33204.3	24823.5	17986.8	9909.82	26542.8	32583.5	14606.6	34836.8
26920.8	38248.3	15689.1	30157.7	14642.2	15933.3	44288.3	22197.1
38248.3	22053.2	25468.5	23485.9	25768.6	34182.2	57444.5	38059.8
19481.3	19563.8	38598.4	20754.3	13864.6	36599.	45856.1	22362.3
21984.	11073.	18158.5	7304.2	58092.	16518.6	46461.5	20058.7
12533.2	22848.5	25699.4	21612.6	48950.9	41438.	11411.	43940.6
17239.5	30488.7	29866.3	32184.4	17308.7	27863.9	28920.6	58367.3
16849.3	28138.5	23038.2	11736.9	16479.5	31415.7	12117.3	15417.1
29414.6	44682.1	36281.	33302.8	15797.1	31864.8	43719.5	30799.5
48971.6	34061.4	28938.6	38540.	27045.1	51284.3	16352.2	11866.4
13267.6	61554.6	13700.2	46963.9	23475.6	24554.1	18050.	15237.6
20555.	28421.7	21876.5	12810.2	15109.4	37414.7	41521.6	25372.8
21139.8	27757.6	22678.1	12178.5	26106.7	27417.6	23337.2	43395.5
11536.2	44658.6	32762.5	16403.8	21184.7	49917.3	21623.8	16625.9
14014.5	20409.3	31671.3	17149.2	27756.3	40949.9	43743.2	38459.9
40972.9	46587.9	43799.6	18912.2	27765.8	33007.3	26325.3	15308.2
59805.6	28658.3	23175.	11595.4	50409.9	11215.3	13327.8	16088.8
43943.	14505.3	33886.4	16662.5	20262.6	33615.4	22007.1	28981.1
12163.9	17247.7	12683.6	16291.	18707.3	19326.9	14511.8	10672.
25830.5	43499.5	59175.1	27642.9	30067.5	29714.4	13950.4	10072.6
37850.6	57176.4	38784.	10191.8	21821.4	37389.	14627.9	48770.5

```
 21096.2  36256.9  15281.8   9316.98 20736.2  52662.5   8020.19 32245.4
 41107.2  39358.3  36095.9   7723.93 18565.8  25132.9  31290.6  24858.4
 16398.8  23287.9  50897.6  22446.5  23092.1  24867.6  22234.7  17371.1
 29574.   17944.2  33665.5  36166.2  27712.9  22400.7  28469.9  30488.
 19160.3  45342.5   6294.21 25127.7  51879.3  12644.9  21984.4  29093.1
 23528.4   9516.91 18364.9  31273.8  49673.6  12623.4  23818.6  31473.9
 20268.   51417.   30971.8  47025.    9672.25 15976.3  14711.8  26671.6 ]

Interactions:
[    0.       0.   16575.4      0.   50576.3  37869.6      0.   24946.6
     0.   24212.1  59803.9  26658.8  15735.8  55204.7  19474.6  22342.1
     0.   41016.   26909.2  22522.8  57880.7  16497.3  38446.6  15538.8
 12640.3  41034.   20809.7      0.       0.       0.   22942.9  16325.8
 23443.2  29921.3  37521.9  19868.   10953.       0.   18504.3      0.
     0.       0.   55716.5      0.   13740.   52670.6  13283.9  13106.6
     0.   17867.3  14309.7      0.   16259.7  29794.1  56842.5  47835.8
     0.   23124.9      0.   25334.3  24763.3      0.   27022.6  11700.4
  5014.21 17390.1  10861.   34892.9  19403.1      0.   14064.9      0.
 31982.   23197.5  52674.       0.   26948.   49456.7      0.   34524.9
 22052.1      0.   12591.4  16394.4      0.   31683.1  15525.   22562.2
 15848.7      0.       0.   25429.3      0.   42579.1  41127.4   9990.11
     0.   30870.8  12125.8      0.   26707.9  11604.4      0.   33088.5
 34513.6  32395.5  46633.       0.   12681.9  24031.5  37330.5      0.
     0.   33630.6  43228.2  47796.8  21730.3  10044.1      0.   45765.
 29525.5  54863.8  20799.   33028.3  45031.9      0.   25257.7  42603.9
 14092.7  21350.3      0.   41609.5  16716.1      0.   59503.8  31334.8
     0.   39205.3  42173.9  55263.   37095.2      0.       0.   48974.8
 18923.   51204.2      0.   18860.3  25732.5  28240.4  28193.6      0.
 54618.8  24760.8      0.    8143.75 26462.5  20467.3      0.   15315.3
 18875.7      0.       0.       0.       0.       0.    9589.91     0.
 26707.5      0.   49175.7      0.   24346.6  26999.4  41558.1  56340.3
 37558.5  30099.3      0.       0.   17655.   56658.9  37706.5  18516.
 29622.   32669.9      0.       0.       0.       0.   63130.1  14996.4
 49024.9      0.       0.   17839.9      0.   48720.3      0.   20819.
 26077.8  41627.1      0.       0.   12764.8      0.   59409.1  14960.2
 39666.6  20771.9      0.   33123.7  14433.4      0.    9824.37 17610.3
     0.   31774.1  31693.5  28598.7      0.   42124.1  39308.7  43530.
 49874.4  27434.8  50474.6  24888.2      0.       0.       0.       0.
     0.   30404.3  41438.2  16711.3  52255.9  17866.9  18067.5  12823.7
     0.   56031.1  35263.5  19968.1  27825.5  37773.9      0.       0.
 20347.       0.   57671.7  36057.8      0.       0.   10629.1      0.
 26097.9      0.       0.       0.   17180.2  28882.3      0.   46358.4
 19166.       0.   33229.   30396.1      0.       0.   60747.5  56394.3
 13236.4      0.   27056.5      0.   28702.7  22366.1  24477.5      0.
     0.   15610.2  54314.5  39175.8  13739.       0.   24675.7      0.
 14136.5  37162.1  13519.2  39253.6  46323.8  20950.7  22495.7  32548.9
 24583.4      0.       0.   13667.7   8162.42 15349.6  29231.4  41462.3
 57398.1      0.   52117.3  26281.4  25683.4  11920.7  30658.7  36646.4
     0.   16109.9  18036.7  42628.3  22110.1  37689.1  23171.8  21951.3
 38103.4      0.   11043.7  24027.6  28495.1      0.   34852.3      0.
 50849.2      0.   52769.9  11601.4  29541.7  17861.       0.   26688.1
     0.   38080.9  37554.1      0.   28864.9  48346.1  53104.3  19416.8
 23638.1  42378.2  39745.3  45189.8  37930.9      0.       0.   24424.3
 24607.8  43057.       0.   50186.1      0.    9592.73     0.   22792.3
 51620.8  19918.9      0.   12549.   51299.3  17364.8      0.   47750.2
 11281.5  34073.8  46870.4  38453.7      0.       0.   47198.6  20866.3
     0.       0.       0.       0.   26542.8  32583.5      0.   34836.8
     0.       0.       0.       0.       0.   15933.3  44288.3      0.
 38248.3  22053.2      0.       0.   25768.6  34182.2  57444.5  38059.8
     0.       0.       0.       0.   13864.6  36599.   45856.1  22362.3
     0.   11073.       0.    7304.2  58092.       0.   46461.5  20058.7
```

```
 12533.2      0.   25699.4  21612.6  48950.9  41438.       0.   43940.6
     0.   30488.7  29866.3      0.       0.   27863.9  28920.6  58367.3
 16849.3      0.   23038.2  11736.9      0.   31415.7      0.       0.
 29414.6  44682.1  36281.       0.   15797.1  31864.8  43719.5      0.
 48971.6  34061.4      0.   38540.       0.   51284.3  16352.2  11866.4
 13267.6  61554.6  13700.2  46963.9      0.   24554.1      0.       0.
 20555.   28421.7      0.   12810.2      0.   37414.7  41521.6  25372.8
 21139.8  27757.6  22678.1  12178.5      0.   27417.6  23337.2  43395.5
     0.   44658.6  32762.5  16403.8  21184.7  49917.3  21623.8      0.
 14014.5  20409.3  31671.3  17149.2  27756.3  40949.9  43743.2      0.
 40972.9  46587.9  43799.6      0.   27765.8      0.       0.   15308.2
 59805.6  28658.3      0.   11595.4  50409.9  11215.3  13327.8  16088.8
 43943.   14505.3  33886.4  16662.5  20262.6  33615.4      0.   28981.1
     0.   17247.7  12683.6  16291.   18707.3      0.   14511.8  10672.
     0.   43499.5  59175.1      0.   30067.5      0.   13950.4      0.
 37850.6  57176.4  38784.   10191.8  21821.4  37389.   14627.9  48770.5
 21096.2      0.       0.    9316.98 20736.2  52662.5      0.       0.
 41107.2      0.   36095.9   7723.93 18565.8      0.   31290.6      0.
 16398.8      0.   50897.6      0.       0.   24867.6      0.   17371.1
 29574.   17944.2      0.       0.   27712.9  22400.7  28469.9  30488.
 19160.3  45342.5   6294.21     0.   51879.3      0.   21984.4  29093.1
 23528.4   9516.91     0.   31273.8  49673.6  12623.4      0.   31473.9
 20268.   51417.   30971.8  47025.    9672.25 15976.3  14711.8      0.  ]

Binning - Fixed-width binning:
[[17540.0, 17550.0), [30080.0, 30090.0), [16570.0, 16580.0), [20370.0, 2038
0.0), [50570.0, 50580.0), ..., [47020.0, 47030.0), [9670.0, 9680.0), [15970
.0, 15980.0), [14710.0, 14720.0), [26670.0, 26680.0)]
Length: 600
Categories (6314, interval[float64, left]): [[0.0, 10.0) < [10.0, 20.0) < [
20.0, 30.0) < [30.0, 40.0) ... [63100.0, 63110.0) < [63110.0, 63120.0) < [6
3120.0, 63130.0) < [63130.0, 63140.0)]

Binning - Quantile based binning:
[0 1 0 1 2 2 0 1 1 1 2 1 0 2 0 1 0 2 1 1 2 0 2 0 0 2 1 0 1 1 1 0 1 1 2 0 0
 0 0 1 1 1 2 1 0 2 0 0 2 0 0 1 0 1 2 2 1 1 0 1 1 2 1 0 0 0 0 2 0 0 0 0 2 1
 2 2 1 2 0 2 1 1 0 0 1 2 0 1 0 1 1 1 2 2 2 0 0 1 0 0 1 0 0 2 2 2 2 0 0 1 2
 1 2 2 2 2 1 0 0 2 1 2 1 2 2 2 1 2 0 1 1 2 0 2 2 2 0 2 2 2 2 1 0 2 0 2 0 0
 1 1 1 2 2 1 1 0 1 1 1 0 0 0 1 0 1 1 0 0 1 2 2 0 1 1 2 2 2 1 0 2 0 2 2 0 1
 2 0 2 2 1 2 0 2 0 2 0 0 2 0 1 1 2 0 0 0 0 2 0 2 1 1 2 0 0 0 0 0 2 2 1 1 2
 2 2 2 1 2 1 1 0 1 1 0 1 2 0 2 0 0 0 0 2 2 0 1 2 0 1 1 1 2 2 0 0 0 1 1 1 1
 0 0 1 1 2 0 0 2 1 2 0 2 2 0 1 1 0 1 1 1 2 1 0 2 2 0 0 1 1 0 2 0 2 2 1 1 2
 1 0 0 0 0 0 1 2 2 0 2 1 1 0 1 2 1 0 0 2 1 2 1 1 2 1 0 1 1 0 2 1 2 0 2 0 1
 0 1 1 1 2 2 0 1 2 2 0 1 2 2 2 2 1 2 1 1 2 1 2 1 0 2 1 2 0 1 0 2 0 1 2 0 2
 2 2 0 1 2 1 2 1 0 0 1 2 0 2 1 2 0 1 0 0 2 1 2 1 1 1 1 2 2 2 0 0 2 1 0 2 2
 1 1 0 0 0 2 0 2 0 0 1 1 1 2 2 0 2 0 1 1 2 0 1 1 2 0 1 1 0 0 2 0 0 1 2 2 2
 0 2 2 1 2 2 1 2 1 2 0 0 0 2 0 2 1 1 0 0 1 1 1 0 0 2 2 1 1 1 1 0 1 1 1 2 0
 2 2 0 1 2 1 0 0 1 2 0 1 2 2 2 2 2 2 0 1 2 1 0 2 1 1 0 2 0 0 0 2 0 2 0 1 2
 1 1 0 0 0 0 0 0 0 0 1 2 2 1 1 1 0 0 2 2 2 0 1 2 0 2 1 2 0 0 1 2 0 2 2 2 2
 0 0 1 2 1 0 1 2 1 1 1 1 0 1 0 2 2 1 1 1 1 0 2 0 1 2 0 1 1 1 0 0 2 2 0 1 2
 1 2 1 2 0 0 0 1]

Mathematical Transformations - Log transform:
[ 9.77258128 10.31178531  9.71567495  9.92208357 10.83123837 10.54190396
  9.09122683 10.12449282 10.13872962 10.09460779 10.99882616 10.19087458
  9.66369365 10.91880337  9.87686633 10.01422807  9.78300212 10.62171751
 10.20022351 10.02228341 10.96613927  9.71095201 10.55702554  9.6510954
  9.4446454  10.62215627  9.9431745   9.90917137 10.28735783 10.09700042
 10.0407638   9.70050196 10.06233575 10.30632588 10.53268004  9.89686568
  9.30136867  9.50159107  9.82575842 10.14216975 10.19519402 10.20183505
 10.92803161 10.22453791  9.52806657 10.8718127   9.49430805  9.4808712
```

10.58526535	9.79072751	9.56869291	10.08141614	9.69644493	10.30206567
10.94803956	10.77552959	10.1257307	10.04866524	9.62534649	10.13991449
10.117118	10.50750293	10.20442883	9.36737831	8.52003116	9.76365636
9.29293367	10.46003865	9.87318813	9.25358184	9.55143761	8.99500749
10.37292852	10.05179979	10.87187725	10.48039582	10.20166436	10.80885282
9.59726805	10.44943608	10.00116311	10.23308262	9.44076932	9.70469509
10.08689602	10.36353869	9.65020691	10.02403122	9.67084276	10.34482161
10.11918344	10.14365733	10.45928176	10.6591188	10.62442985	9.20935088
8.98075361	10.33756603	9.40309069	9.63879909	10.19271468	9.35913962
9.64858885	10.40694107	10.44910873	10.3857748	10.75006372	9.47576917
9.44793106	10.08712075	10.52756597	10.13987107	10.5212159	10.42319165
10.67424834	10.77471397	9.98646288	9.21474068	9.75673196	10.73127489
10.29300958	10.91260903	9.94266019	10.40512005	10.71512641	10.57159381
10.13688634	10.65970108	9.55341221	9.96882107	10.05390556	10.63608379
9.72412761	10.50332355	10.99379546	10.35248458	9.55029938	10.57656722
10.64955683	10.91985889	10.52124286	10.03413855	9.75502235	10.79906116
9.84813339	10.84357684	9.91522836	9.84481446	10.15551006	10.24850886
10.24685028	10.50322475	10.90813342	10.11701704	10.05861347	9.00500604
10.18348392	9.92658377	9.97609654	9.63660761	9.84563066	9.47094925
9.93829952	8.92922072	10.12278371	10.08879631	9.16846678	9.05518953
10.1926997	10.43471856	10.80315488	9.88970805	10.10014749	10.20356992
10.63484773	10.93916537	10.533655	10.31225719	9.63264949	10.49366303
9.77877431	10.94480436	10.53758777	9.8263905	10.29627261	10.39420944
9.81331664	10.4461025	10.45929323	9.98950927	11.05295296	9.61556545
10.80008361	9.69583588	10.49659614	9.7891928	9.8417398	10.79385106
9.58781059	9.94362131	10.16883966	10.63650668	9.73963244	9.85286772
9.45444666	9.57419151	10.99220269	9.61314862	10.5882648	9.94135639
10.10537069	10.40800432	9.57730024	9.48611432	9.19262131	9.77623924
9.62616497	10.36640677	10.36386689	10.26111654	10.17586688	10.6483753
10.57920115	10.68120563	10.81726312	10.21956756	10.82922552	10.12214907
10.24073092	9.41568648	10.31524615	10.27399538	9.55098247	10.32233932
10.63195844	9.72384042	10.86390808	9.79070512	9.80187002	9.4590503
9.33249606	10.93366217	10.47060371	9.90189128	10.23370815	10.53937367
8.93672556	9.97041696	9.92068876	9.96797764	10.96252186	10.49287849
9.56735026	9.79160023	9.2713508	10.09669959	10.16961013	10.05925122
9.97560354	9.40647443	9.75151284	10.27098423	9.98101325	10.74415778
9.86089315	9.79377313	10.41117827	10.32206959	10.45233702	9.72153393
11.01448121	10.94012337	9.49072589	10.25447536	10.20568255	9.14447617
10.26474647	10.0153017	10.10550961	10.51792697	10.01358782	9.65567983
10.90254651	10.57581449	9.52799378	9.15755544	10.11357423	10.24897616
9.55651538	10.5230447	9.51186618	10.57779844	10.74341115	9.94992734
10.02107946	10.39049885	10.1098267	9.06406989	9.74914102	9.52279066
9.00729597	9.63884469	10.28299875	10.63253986	10.95776648	9.35190938
10.86125223	10.17661674	10.15360015	9.38603166	10.33067175	10.50907048
10.33398343	9.68718927	9.80016385	10.66027363	10.0037898	10.53712621
10.0506913	9.99658164	10.5480588	10.03814519	9.30961541	10.08695845
10.25748742	9.15537825	10.45887441	9.96497768	10.83661967	9.82854308
10.87369623	9.35888106	10.2935581	9.79037484	9.95427572	10.19197305
10.19990387	10.54746812	10.53353784	9.80833036	10.2703816	10.78614084
10.88001318	9.87389395	10.0706151	10.65438936	10.59024687	10.71862668
10.54352136	10.08755758	10.34840091	10.10333382	10.11081874	10.6702801
10.31554753	10.82349337	10.039595	9.1687608	10.44154695	10.03417804
10.85167997	9.89942431	10.29637725	9.43739626	10.84543239	9.76220045
10.30450612	10.77373853	9.33091949	10.43628404	10.75514163	10.5572102
8.95626843	10.25463022	10.76211951	9.9458907	10.41043466	10.11954606
9.79739343	9.20128146	10.1865138	10.3915613	9.58922876	10.45842958
10.2006545	10.55185439	9.66072148	10.31419556	9.59166305	9.67616654
10.69847581	10.00771693	10.55185439	10.001213	10.14519767	10.06415552
10.15691198	10.43946032	10.95857454	10.54691389	9.87721031	9.8814362
10.5609661	9.94050873	9.53709411	10.5077762	10.73326351	10.01513178
9.9980702	9.31226499	9.80689405	8.8962048	10.96978324	9.7122423

```
 10.74637929  9.90641825  9.4361364  10.03664075 10.15422292  9.98103176
 10.79857303 10.63195361  9.34233308 10.690594    9.75495854 10.3251114
 10.30448603 10.37923714  9.75896454 10.23508722 10.27230942 10.97451108
  9.73206439 10.24489402 10.04490899  9.370493    9.70987246 10.35506305
  9.40238946  9.64323256 10.28924643 10.70732825 10.49904947 10.41339676
  9.66758166 10.36925723 10.68554951 10.33525374 10.79899582 10.43592006
 10.27293162 10.55945194 10.20526112 10.84513994  9.70211772  9.38146616
  9.49308025 11.02767986  9.52516571 10.7571345  10.06371686 10.10863413
  9.80090096  9.63152134  9.9308595  10.25490822  9.99316828  9.45799701
  9.62307235 10.52981895 10.63396905 10.14143301  9.9589128  10.23126496
 10.02915498  9.40742738 10.16994727 10.21894042 10.05780393 10.67811103
  9.3532452  10.70680218 10.39703985  9.70526829  9.9610345  10.81812292
  9.98154984  9.718717    9.54784779  9.92374596 10.36316619  9.7497068
 10.23121812 10.62010465 10.68609145 10.55737142 10.62066615 10.74909613
 10.68737996  9.8475625  10.23156033 10.40448403 10.17828573  9.63614391
 10.99885458 10.26319838 10.05082939  9.35836375 10.82794286  9.3250342
  9.49760736  9.68587866 10.69064862  9.58226938 10.43076903  9.72091596
  9.9165321  10.42273957  9.99912041 10.27439917  9.40622783  9.75543408
  9.4480651   9.69836809  9.8366691   9.86925319  9.58271739  9.27537877
 10.15931124 10.68050472 10.98825612 10.22712419 10.31120013 10.29938706
  9.54326346  9.21757415 10.54140211 10.9538965  10.56576307  9.22933875
  9.99064642 10.52913182  9.59068594 10.7948809   9.95684821 10.49838499
  9.63441786  9.13959382  9.93963624 10.87165891  8.98971739 10.38113068
 10.62393857 10.58046216 10.49393456  8.95207858  9.82907646 10.13193302
 10.35107301 10.120951    9.70496344 10.05568919 10.83757105 10.01888998
 10.04724585 10.12132103 10.00940941  9.76256318 10.29465088  9.79502222
 10.42422885 10.49588026 10.22965329 10.01684749 10.25660267 10.32508844
  9.86059571 10.72200006  8.74738544 10.1317261  10.85667515  9.44500925
  9.99808839 10.27825631 10.06596348  9.1608255   9.81819651 10.35053597
 10.81322888  9.44330751 10.07822207 10.35691391  9.91679857 10.84772414
 10.34083239 10.75843465  9.17701624  9.67886165  9.59640517 10.19135461]

Mathematical Transformations - Box-Cox transform:
[37.9708847  43.66863677 37.41055727 39.47911757 49.88352826 46.32860271
 31.72941358 41.60728463 41.76081319 41.28667072 52.05575806 42.32754155
 36.9052238  51.00786144 39.01735411 40.43541376 38.07430654 47.28524839
 42.4298841  40.5200011  51.62534487 37.36438841 46.50847674 36.783676
 34.8422105  47.29055702 39.69618233 39.34675644 43.39462004 41.31225672
 40.71466504 37.26241612 40.94296474 43.60725888 46.21919561 39.22098533
 33.54907644 35.36836862 38.50130059 41.79798831 42.37479839 42.44754821
 51.12770344 42.69710716 35.61539822 50.40162977 35.30068325 35.17610985
 46.84610856 38.15114077 35.99745999 41.14586141 37.22289476 43.55941785
 51.38842882 49.18016269 41.62061347 40.7981532  36.53636589 41.77361384
 41.52795661 45.92176664 42.47599289 34.13944193 27.21375341 37.88250995
 33.47429391 45.36580679 38.98000697 33.1273607  35.83474105 30.92571843
 44.36144192 40.83131644 50.40245796 45.60349655 42.44567701 49.59978794
 36.26837966 45.24246055 40.29856112 42.79137906 34.80665184 37.30330296
 41.20430139 44.25439981 36.77511737 40.5383756  36.97435834 44.04173143
 41.5501599  41.81407287 45.35699127 47.73974461 47.31807432 32.74121822
 30.80819555 43.95954265 34.46267364 36.66538738 42.34766733 34.06525761
 36.75953567 44.75115889 45.23865713 44.50826954 48.86168704 35.1289116
 34.87237802 41.20669962 46.15863832 41.77314469 46.08354642 44.9384598
 47.92473347 49.16993301 40.14508018 32.78805767 37.81407189 48.62793156
 43.45787922 50.92756441 39.69087628 44.73021465 48.42784739 46.68237377
 41.74090645 47.74685195 35.85332833 39.96158488 40.85360893 47.45935392
 37.49331372 45.87256363 51.98929979 44.12868715 35.82403058 46.74187617
 47.62316865 51.02155565 46.08386498 40.6447806  37.79719182 49.47614503
 38.72646648 50.04055883 39.40879583 38.69299238 41.94243212 42.9620527
 42.9436728  45.87140098 50.86961994 41.52687156 40.90348862 31.00839121
 42.24679489 39.52534282 40.03716622 36.64434149 38.70122197 35.08437506
 39.64591423 30.38657356 41.58888822 41.22458454 32.38784203 31.42627538
```

42.34750348	45.07175052	49.52780337	39.14799727	41.34593231	42.46657177
47.44435098	51.27263725	46.2307486	43.67394563	36.6063571	45.75901657
38.03231704	51.34618755	46.2773779	38.50764523	43.49444009	44.60491501
38.37660606	45.2037423	45.35712488	40.17684323	52.77578879	36.44281319
49.48904263	37.21696495	45.79346434	38.13586599	38.66200518	49.41047092
36.17851261	39.70079244	42.08721232	47.46448786	37.6455425	38.77426111
34.93227041	36.04945086	51.96827488	36.41973	46.88210137	39.67742824
41.40187878	44.76339185	36.07887462	35.22467224	32.59620842	38.00715935
36.54420417	44.28707027	44.25813721	43.10200201	42.16372151	47.60878213
46.77341689	48.01002108	49.70621508	42.64235732	49.85795406	41.58205901
42.87592148	34.57732564	43.70758613	43.24538943	35.83045798	43.78751394
47.40929856	37.49049927	50.30030677	38.15091791	38.26220637	34.97464299
33.82632864	51.20095201	45.48902244	39.27230594	42.79828772	46.29856674
30.4476584	39.97815278	39.46480024	39.9528312	51.57791429	45.74980657
35.98477505	38.15982935	33.28361852	41.30903899	42.09559472	40.91024969
40.03204058	34.49344049	37.76256154	43.21182576	40.08831658	48.78809901
38.85540213	38.18146987	44.79992716	43.78447206	45.27617846	37.46790264
52.26307258	51.28512577	35.2674345	43.02823112	42.48974799	32.18205571
43.14237225	40.44667846	41.40336763	46.0446975	40.42869747	36.8278649
50.79737294	46.73286594	35.61471701	32.29410318	41.48988656	42.96723257
35.88255647	46.10516122	35.46405827	46.75661732	48.77880316	39.76590891
40.50734853	44.56237489	41.44966187	31.50073728	37.73917365	35.56604887
31.02735277	36.66582546	43.34588684	47.41635035	51.51562412	34.00027188
50.26630565	42.1718931	41.92172453	34.30793486	43.88157608	45.94023356
43.91901167	37.1328718	38.24518086	47.75384156	40.32604137	46.27190308
40.81958593	40.25067047	46.40173859	40.68703039	33.62233316	41.2049676
43.06167517	32.27542777	45.35224748	39.92170988	49.95195967	38.52925905
50.42580094	34.0629318	43.46402328	38.14763027	39.81086678	42.33955484
42.42638125	46.39471524	46.22936017	38.32673442	43.20511145	49.31342958
50.50694399	38.98717115	41.03089387	47.68205234	46.90589969	48.47115236
46.34781155	41.21136167	44.08232818	41.38005335	41.46030671	47.87615022
43.71097951	49.78519057	40.70232841	32.39037123	45.15088034	40.64519681
50.1439343	39.24710609	43.49561307	34.77573416	50.06421356	37.86811106
43.58681765	49.15770147	33.81223743	45.08988063	48.92503932	46.51067719
30.60723143	43.02995008	49.0122192	39.72421517	44.79136511	41.55405919
38.21755003	32.67120242	42.27988096	44.5745516	36.19197571	45.34706771
42.43460752	46.44689311	36.87651595	43.69575912	36.21509553	37.02591671
48.22233447	40.36715848	46.44689311	40.29908281	41.83073383	40.96227657
41.95763777	45.12668627	51.52620393	46.38812601	39.0208484	39.0637998
46.55545537	39.66868726	35.69998126	45.92498547	48.65262367	40.4448955
40.26622483	33.64589983	38.3123796	30.11911823	51.67316437	37.37699648
48.81576703	39.3185865	34.76419249	40.67116151	41.92847575	40.08850923
49.46998873	47.40924002	33.91436927	48.12533153	37.79656197	43.81878663
43.58659204	44.43349148	37.83612573	42.81352274	43.22659405	51.73526848
37.57116908	42.92200326	40.7584446	34.16752522	37.35384253	44.15798155
34.45630071	36.70799735	43.41574954	48.33149729	45.82229548	44.82548034
36.9428079	44.31956185	48.06334209	43.9333791	49.47532089	45.08566468
43.2335296	46.53739867	42.48512385	50.06048476	37.27816653	34.26662712
35.28928395	52.43845071	35.58825684	48.94992322	40.95762067	41.43686866
38.25253551	36.59553719	39.56930762	43.03303588	40.21502367	34.96494429
36.51459524	46.18530767	47.43368851	41.79002436	39.85886007	42.7713093
40.59228448	34.50210968	42.09926307	42.63545376	40.89490775	47.97206794
34.01226999	48.3250037	44.63738914	37.30889532	39.88083691	49.71710704
40.0939025	37.44032182	35.80097143	39.49618764	44.25015816	37.74475152
42.77079228	47.26573862	48.06999829	46.51259845	47.27252993	48.84962382
48.08582725	38.72070666	42.77457005	44.72290165	42.19008604	36.63988977
52.05613381	43.12515098	40.82104708	34.05827878	49.84166361	33.75968101
35.3313313	37.12014025	48.1260031	36.12595135	45.02603978	37.46185053
39.4221611	44.93323965	40.27720211	43.2498921	34.49119742	37.8012565
34.87360922	37.24162441	38.61095026	38.94008873	36.13019844	33.3191306
41.98367271	48.00142248	51.91621295	42.72562108	43.66205412	43.52936227

```
35.75788784 32.81270542 46.32264408 51.46498309 46.61270259 32.91521878
40.18870542 46.17717237 36.20581381 49.42344586 39.83748533 45.81448496
36.62332303 32.14031766 39.65969204 50.39965656 30.88205583 44.455138
47.3121271  46.78852414 45.76220461 30.57295846 38.53461637 41.68745522
44.1126576  41.56916958 37.30592096 40.87249978 49.96406692 40.48434795
40.78314422 41.57315013 40.38489047 37.87169805 43.47626579 38.19391494
44.95043872 45.78505452 42.75352125 40.46290214 43.05184898 43.81852753
38.85239208 48.5129212  28.93895911 41.68522368 50.20775812 34.84555006
40.26641499 43.29292432 40.98147128 32.32217089 38.42546992 44.10656038
49.65513808 34.82993323 41.11183297 44.17902044 39.42489326 50.09344329
43.99652557 48.96616378 32.46145795 37.05204223 36.2601721  42.33279092]
```

4. Write a classification program for implementing logistic regression using wine dataset

```
import pandas as mypd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
wine_data=mypd.read_csv("/home/shyma/Desktop/Machinelearning/ML
Data/Wine.csv")
for col in wine_data.columns:
    if wine_data[col].isnull().sum() > 0:
        wine_data[col] = wine_data[col].fillna(wine_data[col].mean())

wine_data.isnull().sum().sum()

wine_data.replace({'white': 1, 'red': 0}, inplace=True)
X = wine_data.drop('quality', axis=1)
y = wine_data['quality']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Create a logistic regression model
logreg = LogisticRegression()

# Train the model on the training data
logreg.fit(X_train, y_train)

# Make predictions on the testing data
y_pred = logreg.predict(X_test)

# Evaluate the model's accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")

# Display the classification report
class_report = classification_report(y_test, y_pred)
print("Classification Report:")
print(class_report)
```

Output

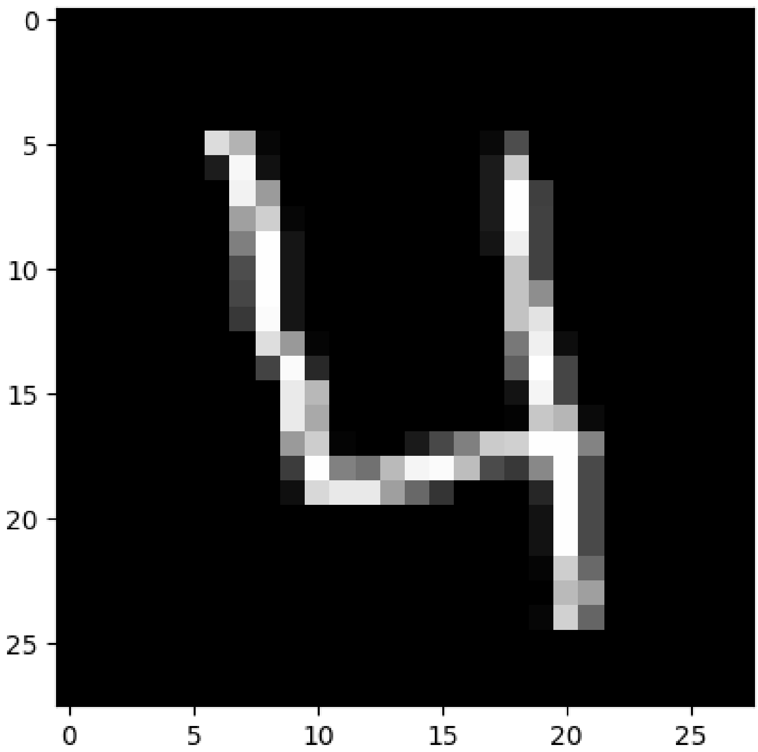
```
Accuracy: 0.47615384615384615
Classification Report:
              precision    recall  f1-score   support

           3       0.00      0.00      0.00         2
           4       0.00      0.00      0.00        46
           5       0.50      0.41      0.45       420
           6       0.47      0.77      0.58       579
           7       0.33      0.00      0.01       221
           8       0.00      0.00      0.00        32

    accuracy                           0.48      1300
   macro avg       0.22      0.20      0.17      1300
weighted avg       0.43      0.48      0.41      1300
```

5. Write a classification program for implementing SVM using MNIST dataset.

```
import numpy as np
import pandas as pd
from sklearn import svm
from sklearn import metrics
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import linear_model
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import scale
import gc
import cv2
digits = pd.read_csv("/home/shyma/Desktop/Machinelearning/ML Data/train.csv")
four = digits.iloc[3, 1:]
four = four.values.reshape(28, 28)
plt.imshow(four, cmap='gray')
X = digits.iloc[:, 1:]
Y = digits.iloc[:, 0]

X = scale(X)

# train test split with train_size=10% and test size=90%
x_train, x_test, y_train, y_test = train_test_split(X, Y, train_size=0.10, random_state=101)
# an initial SVM model with linear kernel
svm_linear = svm.SVC(kernel='linear')

# fit
svm_linear.fit(x_train, y_train)
predictions = svm_linear.predict(x_test)
accuracy=metrics.accuracy_score(y_true=y_test, y_pred=predictions)
print(f"Accuracy: {accuracy}")
class_report=metrics.classification_report(y_true=y_test, y_pred=predictions)
print("Classification Report:")
print(class_report)
```

Output

```
Accuracy:  0.9042592592592592
Classification  Report:
              precision    recall  f1-score   support

           0       0.94      0.97      0.95      3715
           1       0.94      0.98      0.96      4185
           2       0.89      0.89      0.89      3790
           3       0.88      0.87      0.87      3900
           4       0.88      0.92      0.90      3702
           5       0.87      0.85      0.86      3418
           6       0.94      0.94      0.94      3693
           7       0.90      0.92      0.91      3954
           8       0.91      0.84      0.88      3665
           9       0.88      0.85      0.87      3778

    accuracy                           0.90     37800
   macro avg       0.90      0.90      0.90     37800
    weighted       0.90      0.90      0.90     37800
         avg
```



6. Write a classification program for implementing Naïve Bayes algorithm using iris dataset

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import pandas as mypd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.naive_bayes import GaussianNB
from sklearn import metrics
mydata = mypd.read_csv("/home/shyma/Desktop/Machinelearning/ML
Data/Iris_data.csv")
X = mydata.iloc[:,:4].values
y = mydata['Species'].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
classifier = GaussianNB()
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
print ("Accuracy : ", accuracy_score(y_test, y_pred))
class_report=metrics.classification_report(y_test, y_pred )
print("Classification Report:")
print(class_report)
```

Output

```
Accuracy :  0.9523809523809523
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         6
           1       0.86      1.00      0.92         6
           2       1.00      0.89      0.94         9

    accuracy                           0.95        21
   macro avg       0.95      0.96      0.95        21
weighted avg       0.96      0.95      0.95        21
```

7. Write a classification program for implementing decision tree using pima-indians-diabetes dataset.

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import pandas as mypd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn import metrics
import numpy as np
from sklearn.tree import DecisionTreeClassifier
pima =  pd.read_csv("/home/shyma/Desktop/Machinelearning/ML
Data/diabetes.csv")
feature_cols = ['Pregnancies', 'Insulin', 'BMI', 'Age', 'Glucose',
'BloodPressure', 'DiabetesPedigreeFunction']
x = pima[feature_cols]
y = pima.Outcome
X_train, X_test, Y_train, Y_test = train_test_split(x, y, test_size = 0.3,
random_state=1)
classifier = DecisionTreeClassifier()
classifier = classifier.fit(X_train, Y_train)
y_pred = classifier.predict(X_test)
print("Accuracy:", metrics.accuracy_score(Y_test,y_pred))
class_report=metrics.classification_report(Y_test, y_pred )
print("Classification Report:")
print(class_report)
```

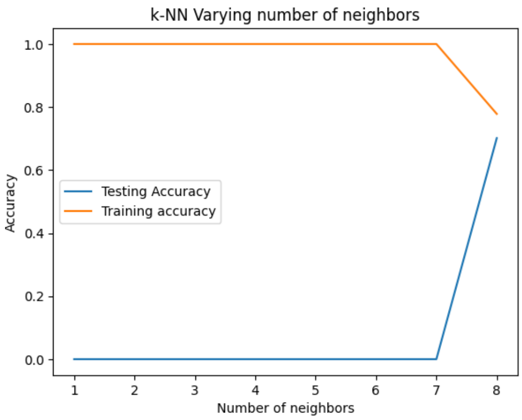
Output

```
Accuracy: 0.658008658008658
Classification Report:
              precision    recall  f1-score   support

           0       0.72      0.75      0.73       146
           1       0.54      0.51      0.52        85

    accuracy                           0.66       231
   macro avg       0.63      0.63      0.63       231
weighted avg       0.65      0.66      0.66       231
```

8. Write a classification program for implementing KNN.

```
import numpy as np
import matplotlib.pyplot as myplot
import matplotlib.image as mpimg
import pandas as mypd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn import metrics
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
pima =  pd.read_csv("/home/shyma/Desktop/Machinelearning/ML
Data/diabetes.csv")
X = pima.drop('Outcome',axis=1).values
y = pima['Outcome'].values
X_train,X_test,y_train,y_test =
train_test_split(X,y,test_size=0.4,random_state=42, stratify=y)
#Setup arrays to store training and test accuracies
neighbors = np.arange(1,9)
train_accuracy =np.empty(len(neighbors))
test_accuracy = np.empty(len(neighbors))

for i,k in enumerate(neighbors):
    #Setup a knn classifier with k neighbors
    knn = KNeighborsClassifier(n_neighbors=k)

    #Fit the model
    knn.fit(X_train, y_train)
train_accuracy[i] = knn.score(X_train, y_train)

    #Compute accuracy on the test set
test_accuracy[i] = knn.score(X_test, y_test)
myplot.title('k-NN Varying number of neighbors')
myplot.plot(neighbors, test_accuracy, label='Testing Accuracy')
myplot.plot(neighbors, train_accuracy, label='Training accuracy')
myplot.legend()
myplot.xlabel('Number of neighbors')
myplot.ylabel('Accuracy')
myplot.show()
y_pred = knn.predict(X_test)
print("Accuracy:", metrics.accuracy_score(y_test,y_pred))
class_report=metrics.classification_report(y_test, y_pred )
print("Classification Report:")
print(class_report)
```

Output




```
Accuracy:  0.7012987012987013
Classification  Report:
```

	precision	Recall	f1-score	support
0	0.74	0.84	0.79	201
1	0.59	0.45	0.51	107
accuracy			0.70	308
macro avg	0.67	0.64	0.65	308
weighted avg	0.69	0.70	0.69	308

9. Write a clustering program for implementing k Means , k-medoids and Hierarchical Clustering using Wisconsin Breast Cancer Dataset

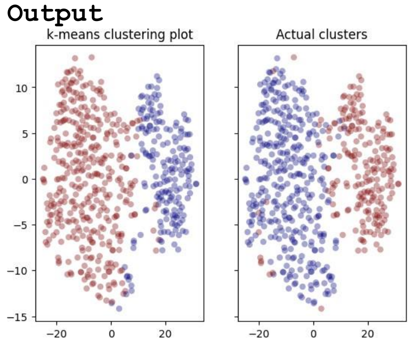
```
import numpy as np
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
import matplotlib.pyplot as plt2
import matplotlib.cm as cm
from sklearn import preprocessing
from sklearn.manifold import TSNE
from sklearn.cluster import KMeans
from sklearn_extra.cluster import KMedoids
from sklearn.cluster import AgglomerativeClustering
df= pd.read_csv("/home/shyma/Desktop/Machinelearning/ML Data/data.csv")
df = df.drop('id',axis=1)
df = df.drop('Unnamed: 32',axis=1)
# Mapping Benign to 0 and Malignant to 1
df['diagnosis'] = df['diagnosis'].map({'M':1,'B':0})
# Scaling the dataset
datas = pd.DataFrame(preprocessing.scale(df.iloc[:,1:32]))
datas.columns = list(df.iloc[:,1:32].columns)
datas['diagnosis'] = df['diagnosis']
# Creating the high dimensional feature space X
data_drop = datas.drop('diagnosis',axis=1)
X = data_drop.values

#Creating a 2D visualization to visualize the clusters

tsne = TSNE(verbose=1, perplexity=40, n_iter= 4000)
Y = tsne.fit_transform(X)

kmns = KMeans(n_clusters=2, init='k-means++', n_init=50, max_iter=300,
tol=0.0001)
kY = kmns.fit_predict(X)

f, (ax1, ax2) = plt.subplots(1, 2, sharey=True)

ax1.scatter(Y[:,0],Y[:,1],  c=kY, cmap = "jet", edgecolor = "None",
alpha=0.35)
ax1.set_title('k-means clustering plot')

ax2.scatter(Y[:,0],Y[:,1],  c = datas['diagnosis'], cmap = "jet", edgecolor
= "None", alpha=0.35)
ax2.set_title('Actual clusters')
kmedoids = KMedoids(n_clusters=2, random_state=0)
kY = kmedoids.fit_predict(X)
f, (ax1, ax2) = plt.subplots(1, 2, sharey=True)

ax1.scatter(Y[:,0],Y[:,1],  c=kY, cmap = "jet", edgecolor = "None",
alpha=0.35)
ax1.set_title('k-medoids clustering plot')

ax2.scatter(Y[:,0],Y[:,1],  c = datas['diagnosis'], cmap = "jet", edgecolor
= "None", alpha=0.35)
ax2.set_title('Actual clusters')
```

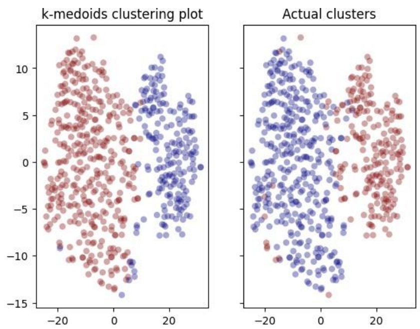
```
aggC = AgglomerativeClustering(n_clusters=2, linkage='ward')
kY = aggC.fit_predict(X)


f, (ax1, ax2) = plt.subplots(1, 2, sharey=True)


ax1.scatter(Y[:,0],Y[:,1],  c=kY, cmap = "jet", edgecolor = "None",
alpha=0.35)
ax1.set_title('Hierarchical clustering plot')

ax2.scatter(Y[:,0],Y[:,1],  c = datas['diagnosis'], cmap = "jet", edgecolor
= "None", alpha=0.35)
ax2.set_title('Actual clusters')
```

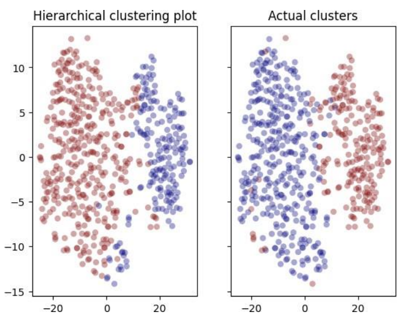
Output

```
[t-SNE] Computing 121 nearest neighbors...
[t-SNE] Indexed 569 samples in 0.002s...
[t-SNE] Computed neighbors for 569 samples in 0.828s...
[t-SNE] Computed conditional probabilities for sample 569 / 569
[t-SNE] Mean sigma: 1.522404
[t-SNE] KL divergence after 250 iterations with early exaggeration: 57.1762
31
[t-SNE] KL divergence after 1750 iterations: 0.862932
Text(0.5, 1.0, 'Actual clusters')
```

Output







10. Write a program to implement PCA

```
import numpy as np
import matplotlib.pyplot as myplot
iimport pandas as mypd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
mydata = mypd.read_csv("/home/shyma/Desktop/Machinelearning/ML
Data/Iris_data.csv")
scalar = StandardScaler()
scaled_data = mypd.DataFrame(scalar.fit_transform(mydata))
print(scaled_data)
pca = PCA(n_components = 2)
pca.fit(mydata)
data_pca = pca.transform(mydata)
data_pca = mypd.DataFrame(data_pca,columns=['PC1','PC2'])
print(data_pca)
```

Output

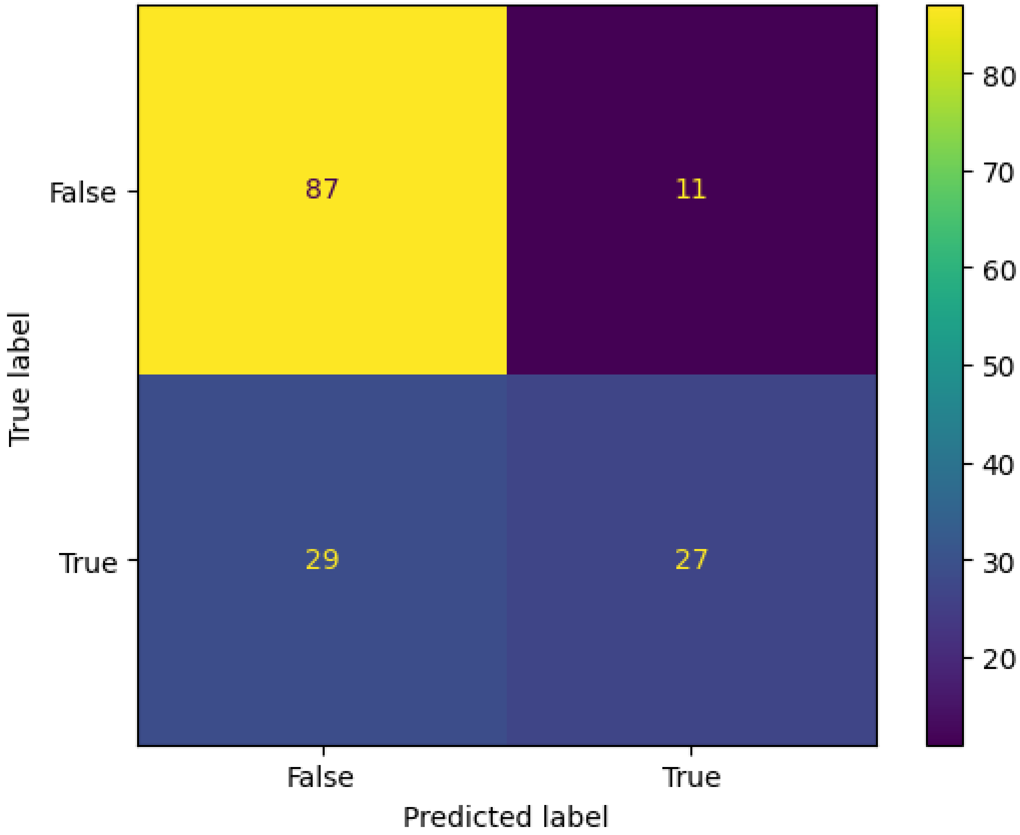
```
            0         1         2         3         4         5
0   -1.690187 -0.820478  0.975487 -1.301159 -1.299067 -1.210338
1   -1.667444 -1.073126 -0.136524 -1.301159 -1.299067 -1.210338
2   -1.644701 -1.325773  0.308280 -1.358874 -1.299067 -1.210338
3   -1.621958 -1.452097  0.085878 -1.243444 -1.299067 -1.210338
4   -1.599215 -0.946802  1.197889 -1.301159 -1.299067 -1.210338
..        ...       ...       ...       ...       ...       ...
96   1.584803  1.200700  0.530683  1.180586  1.789707  1.284440
97   1.607546  1.200700 -0.136524  0.892011  1.521118  1.284440
98   1.630289  0.695405 -1.248535  0.776581  0.983940  1.284440
99   1.675775  0.569082  0.753085  1.007441  1.521118  1.284440
100  1.698518  0.190111 -0.136524  0.834296  0.849645  1.284440

[101 rows x 6 columns]
           PC1       PC2
0   -74.365395  0.527860
1   -73.366955  0.401515
2   -72.374772  0.182620
3   -71.369647  0.259184
4   -70.370931  0.311149
..         ...       ...
96   69.736694 -0.260211
97   70.716416 -0.761859
98   71.699201 -1.255177
99   73.712492 -0.961413
100  74.691154 -1.519958

[101 rows x 2 columns]
```

11. Write a program to evaluate Classification Model using different Evaluation Metrics

```
import numpy as np
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
from sklearn.model_selection import train_test_split
pima =  pd.read_csv("/home/shyma/Desktop/Machinelearning/ML
Data/diabetes.csv")
feature_cols = ['Pregnancies', 'Insulin', 'BMI', 'Age', 'Glucose',
'BloodPressure', 'DiabetesPedigreeFunction']
x = pima[feature_cols]
y = pima.Outcome
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2,
random_state=24)
print(f"Train Data: {X_train.shape}, {y_train.shape}")
print(f"Train Data: {X_test.shape}, {y_test.shape}")
classifier = RandomForestClassifier(random_state=18)
classifier.fit(X_train, y_train)
predictions = classifier.predict(X_test)
print(f"Accuracy of the classifier is: {metrics.accuracy_score(y_test,
predictions)}")
print(metrics.confusion_matrix(y_test, predictions))
cmdisplay=metrics.ConfusionMatrixDisplay(confusion_matrix=confusion_matrix)
cmdisplay.plot()
plt.show()
print(f"Precision Score of the classifier is:
{metrics.precision_score(y_test, predictions)}")
print(f"F1 Score of the classifier is: {metrics.f1_score(y_test,
predictions)}")
class_report=metrics.classification_report(y_test, predictions )
print("Classification Report:")
print(class_report)
```

Output

```
Train Data: (614, 7), (614,)
Train Data: (154, 7), (154,)
Accuracy of the classifier is: 0.7402597402597403
[[87 11]
 [29 27]]
Precision Score of the classifier is: 0.7105263157894737
F1 Score of the classifier is: 0.5744680851063829
```




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

           0       0.75      0.89      0.81        98
           1       0.71      0.48      0.57        56

    accuracy                           0.74       154
   macro avg       0.73      0.68      0.69       154
weighted avg       0.74      0.74      0.73       154
```

12. Write a program to evaluate Classification Model using different Evaluation Metrics.

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn import preprocessing

from sklearn.manifold import TSNE

from sklearn.cluster import KMeans

from sklearn.datasets import make_blobs

from sklearn.metrics import
rand_score,adjusted_rand_score,silhouette_score

mydata=mypd.read_csv("/home/shyma/Desktop/Machinelearning/ML
Data/Iris_data.csv")

feature, target = make_blobs(n_samples=500,centers=5,random_state=42,

shuffle=False)

plt.scatter(feature[:, 0], feature[:, 1])

model = KMeans(n_clusters=4)

model.fit(feature)

plt.scatter(feature[:, 0], feature[:, 1], color="r")

plt.scatter(model.cluster_centers_[1],model.cluster_centers_[3],

color="k", marker="*")

plt.scatter(model.cluster_centers_[2],model.cluster_centers_[0],

color="k", marker="*")

RI = rand_score(target, model.labels_)

ARI = adjusted_rand_score(target, model.labels_)

print(ARI)

ris = rand_score(target, model.labels_)

print(ris)

print(silhouette_score(feature, model.labels_))
```

Output

```
0.7812362998684788

0.9198396793587175

0.7328381899726921
```