In [1]:
```python
#Importing Libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

In [2]:
```python
#For getting the data from csv file
df=pd.read_csv("Retail_Customer_Insights.csv")
```

In [3]:
```python
#To check number of rows and columns
df.shape
```

Out[3]:
```
(100000, 15)
```

In [4]:
```python
#To check column names present in data set
df.columns
```

Out[4]:
```
Index(['Customer_ID', 'Age', 'Annual_Income', 'Gender', 'Purchase_History',
       'Product_Category', 'Customer_Satisfaction', 'Loyalty_Points',
       'Marital_Status', 'Number_of_Children', 'Employment_Status',
       'Credit_Score', 'Owns_House', 'Monthly_Expenditure',
       'Internet_Usage_Hours_per_Week'],
      dtype='object')
```

In [5]:
```python
#To check data types and non
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 15 columns):
 #   Column                         Non-Null Count   Dtype  
---  ------                         --------------   -----  
 0   Customer_ID                    100000 non-null  object 
 1   Age                            100000 non-null  int64  
 2   Annual_Income                  95000 non-null   float64
 3   Gender                         100000 non-null  object 
 4   Purchase_History               100000 non-null  int64  
 5   Product_Category               100000 non-null  object 
 6   Customer_Satisfaction          97000 non-null   float64
 7   Loyalty_Points                 98000 non-null   float64
 8   Marital_Status                 100000 non-null  object 
 9   Number_of_Children             100000 non-null  int64  
 10  Employment_Status              100000 non-null  object 
 11  Credit_Score                   100000 non-null  int64  
 12  Owns_House                     100000 non-null  bool   
 13  Monthly_Expenditure            95000 non-null   float64
 14  Internet_Usage_Hours_per_Week  100000 non-null  int64  
dtypes: bool(1), float64(4), int64(5), object(5)
memory usage: 10.8+ MB
```

In [6]:
```python
#For finding total number of nan values present in every column
df.isna().sum()
```

Out[6]:
```
Customer_ID                         0
Age                                 0
Annual_Income                    5000
Gender                              0
Purchase_History                    0
Product_Category                    0
Customer_Satisfaction            3000
Loyalty_Points                   2000
Marital_Status                      0
Number_of_Children                  0
```

```
Employment_Status                  0
Credit_Score                       0
Owns_House                         0
Monthly_Expenditure             5000
Internet_Usage_Hours_per_Week      0
dtype: int64
```

In [7]:

```
#For getting number of unique values present in every column
def number_unique(a):
    l1=[i for i in a]
    l2=[df[i].nunique() for i in l1]
    nu={'name':l1,'no_unique_value':l2}
    nu=pd.DataFrame(nu)
    return nu

number_unique(df.columns)
```

Out[7]:

	name	no_unique_value
0	Customer_ID	94659
1	Age	100
2	Annual_Income	94187
3	Gender	4
4	Purchase_History	2
5	Product_Category	5
6	Customer_Satisfaction	10
7	Loyalty_Points	37093
8	Marital_Status	4
9	Number_of_Children	6
10	Employment_Status	4
11	Credit_Score	551
12	Owns_House	2
13	Monthly_Expenditure	74024
14	Internet_Usage_Hours_per_Week	56

In [8]:

```
#5 POint Summary of data set
df.describe()
```

Out[8]:

	Age	Annual_Income	Purchase_History	Customer_Satisfaction	Loyalty_Points	Number_of_Chil
count	1.000000e+05	95000.000000	100000.000000	97000.000000	98000.000000	1.000000
mean	-4.611686e+17	61167.659783	0.400300	5.542825	499.552494	-2.767012
std	2.010197e+18	19460.767682	0.489962	2.895265	100.106069	1.573397
min	-9.223372e+18	-7091.710000	0.000000	1.000000	37.180000	-9.223372
25%	3.000000e+01	50077.737500	0.000000	3.000000	432.140000	1.000000
50%	3.900000e+01	60161.030000	0.000000	6.000000	499.255000	2.000000
75%	4.700000e+01	70461.167500	1.000000	8.000000	566.742500	3.000000
max	9.300000e+01	319745.700000	1.000000	10.000000	932.410000	4.000000

In [9]:

```
#Checking null values
df.isna().sum()
```

Out[9]:
```
Customer_ID                        0
Age                                0
Annual_Income                   5000
Gender                             0
Purchase_History                   0
Product_Category                   0
Customer_Satisfaction           3000
Loyalty_Points                  2000
Marital_Status                     0
Number_of_Children                 0
Employment_Status                  0
Credit_Score                       0
Owns_House                         0
Monthly_Expenditure             5000
Internet_Usage_Hours_per_Week      0
dtype: int64
```

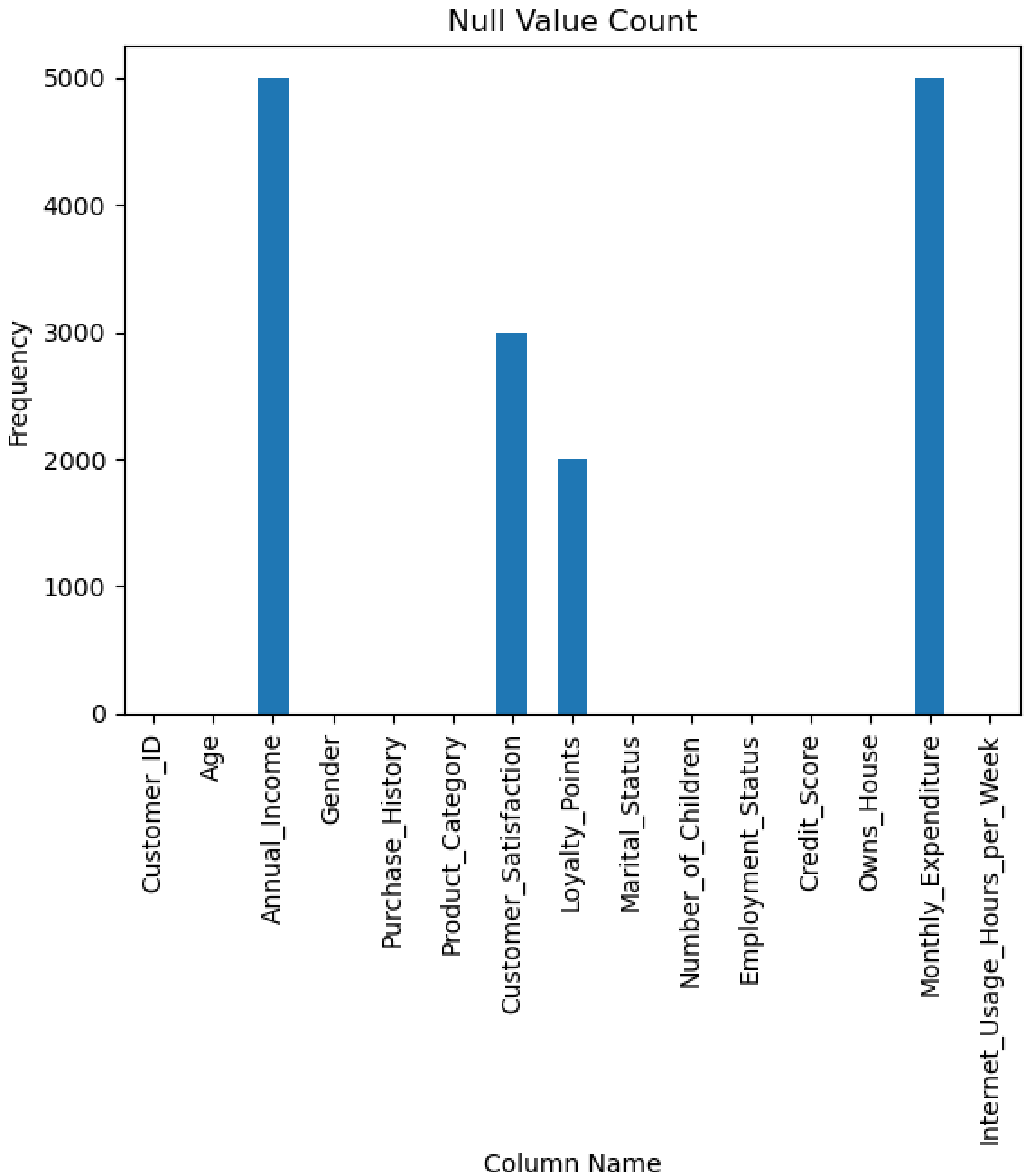
In [10]:
```
#Checking duplicate values
df.duplicated().sum()
```

Out[10]:
```
0
```

In [11]:
```
df.isna().sum()
```

Out[11]:
```
Customer_ID                        0
Age                                0
Annual_Income                   5000
Gender                             0
Purchase_History                   0
Product_Category                   0
Customer_Satisfaction           3000
Loyalty_Points                  2000
Marital_Status                     0
Number_of_Children                 0
Employment_Status                  0
Credit_Score                       0
Owns_House                         0
Monthly_Expenditure             5000
Internet_Usage_Hours_per_Week      0
dtype: int64
```

In [12]:
```
df.isnull().sum().plot(kind='bar')
plt.xlabel('Column Name')
plt.ylabel('Frequency')
plt.title('Null Value Count')
plt.show()
```



Missing Values and Negative Values

In [13]:

```
#Age Column checking negative and zero values
print("negative and zero values",(df['Age']<=0).sum(),end="\n\n")
print(df['Age'][df['Age']<=0].value_counts())
```

```
negative and zero values 5049

Age
-9223372036854775808    5000
 0                        21
-1                        10
-4                         4
-3                         4
-6                         2
-7                         2
-13                        2
-2                         1
-11                        1
-9                         1
```

```
-5                            1
Name: count, dtype: int64
```

In [14]:
```
#Age Column replacing negative and zero value with median
df['Age']=df['Age'].apply(lambda x: df['Age'].median() if x<=0 else x)
```

In [15]:
```
df['Age'][df['Age']<=0].value_counts()
```

Out[15]:
```
Series([], Name: count, dtype: int64)
```

In [16]:
```
#Age Column checking negative and zero values
print("negative and zero values",(df['Annual_Income']<=0).sum(),end="\n\n")
print(df['Annual_Income'][df['Annual_Income']<=0].value_counts())
```

```
negative and zero values 6

Annual_Income
-1212.50    1
-1958.11    1
-7091.71    1
-5826.80    1
-1435.99    1
-969.36     1
Name: count, dtype: int64
```

In [17]:
```
print("df['Annual_Income'].mean() is :",df['Annual_Income'].mean())
print("df['Annual_Income'].median() is :",df['Annual_Income'].median())
```

```
df['Annual_Income'].mean() is : 61167.65978284211
df['Annual_Income'].median() is : 60161.03
```

In [18]:
```
#Annual_Income Column replacing negative and zero value with mean
df['Annual_Income']=df['Annual_Income'].map(lambda x: df['Annual_Income'].mean() if x <=
print("negative and zero values",(df['Annual_Income']<=0).sum(),end="\n\n")
```

```
negative and zero values 0
```

In [19]:
```
#Annual_Income Column replacing negative and zero value with mean
df['Annual_Income'].fillna(df['Annual_Income'].mean(),inplace=True)
```

In [20]:
```
print("negative values",(df['Number_of_Children']<0).sum(),end="\n\n")
print(df['Number_of_Children'][df['Number_of_Children']<=0].value_counts())
```

```
negative values 3000

Number_of_Children
 0                       19559
-9223372036854775808      3000
Name: count, dtype: int64
```

In [21]:
```
#Annual_Income Column replacing negative value with median()
df['Number_of_Children']=df['Number_of_Children'].map(lambda x: df['Number_of_Children']
```

In [22]:
```
print("negative values",(df['Number_of_Children']<0).sum(),end="\n\n")
print("null values",df['Number_of_Children'].isna().sum())
```

```
negative values 0

null values 0
```

In [ ]:
```

```

In [23]:
```
df["Customer_Satisfaction"].value_counts()
```

Out[23]:
```
Customer_Satisfaction
10.0    10651
8.0      9736
2.0      9721
6.0      9668
4.0      9626
5.0      9585
1.0      9582
3.0      9554
9.0      9497
7.0      9380
Name: count, dtype: int64
```

In [24]:
```
df["Customer_Satisfaction"].isna().sum()
```

Out[24]:
```
3000
```

In [25]:
```
#Customer_Satisfaction Column replacing null value with median
df["Customer_Satisfaction"].fillna(df["Customer_Satisfaction"].median(),inplace=True)
```

In [26]:
```
#Loyalty_Points Column replacing null value with mean
print("Null values in Loyalty_Points :",df["Loyalty_Points"].isna().sum())
df["Loyalty_Points"].fillna(df["Loyalty_Points"].mean(),inplace=True)
```

```
Null values in Loyalty_Points : 2000
```

In [27]:
```
print("Null values in Loyalty_Points :",df["Loyalty_Points"].isna().sum())
```

```
Null values in Loyalty_Points : 0
```

In [28]:
```
df.columns
```

Out[28]:
```
Index(['Customer_ID', 'Age', 'Annual_Income', 'Gender', 'Purchase_History',
       'Product_Category', 'Customer_Satisfaction', 'Loyalty_Points',
       'Marital_Status', 'Number_of_Children', 'Employment_Status',
       'Credit_Score', 'Owns_House', 'Monthly_Expenditure',
       'Internet_Usage_Hours_per_Week'],
      dtype='object')
```

In [29]:
```
#Credit_Score Column replacing null value with mean
```

In [30]:
```
print("Null values in Product_Category :",df["Credit_Score"].isna().sum())
print("Negative Score or less than 300 values in Product_Category :",(df["Credit_Score"]
df["Credit_Score"]=df["Credit_Score"].apply(lambda x: df["Credit_Score"].median() if x<3
print("Negative Score or less than 300 values in Product_Category :",df["Credit_Score"].
```

```
Null values in Product_Category : 0
Negative Score or less than 300 values in Product_Category : 4000
Negative Score or less than 300 values in Product_Category : 0
```

In [31]:
```
#Monthly_Expenditure removing null values and handling negative values
print(df['Monthly_Expenditure'].mean())
print(df['Monthly_Expenditure'].median())
print(df['Monthly_Expenditure'].isna().sum())
df['Monthly_Expenditure'].fillna(df['Monthly_Expenditure'].median(),inplace=True)
print((df['Monthly_Expenditure']<0).sum())
print(df['Monthly_Expenditure'].mean())
print(df['Monthly_Expenditure'].median())
```

```
1997.9435483157893
1997.545
5000
2
1997.9236209000003
1997.545
```

In [32]:
```
df['Monthly_Expenditure']=df['Monthly_Expenditure'].map(lambda x: df['Monthly_Expenditur
```

In [33]:
```
print(df['Internet_Usage_Hours_per_Week'].mean())
print(df['Internet_Usage_Hours_per_Week'].median())
df['Internet_Usage_Hours_per_Week']=df['Internet_Usage_Hours_per_Week'].apply(lambda x:
```

```
-3.6893488147419104e+17
31.0
```

In [34]:
```
print(df['Internet_Usage_Hours_per_Week'].mean())
print(df['Internet_Usage_Hours_per_Week'].median())
```

```
31.9949
31.0
```

In [35]:
```
#Re-Checking null values
df.isna().sum()
```

Out[35]:
```
Customer_ID                     0
Age                             0
Annual_Income                   0
Gender                          0
Purchase_History                0
Product_Category                0
Customer_Satisfaction           0
Loyalty_Points                  0
Marital_Status                  0
Number_of_Children              0
Employment_Status               0
Credit_Score                    0
Owns_House                      0
Monthly_Expenditure             0
Internet_Usage_Hours_per_Week   0
dtype: int64
```

In [36]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 15 columns):
 #   Column                         Non-Null Count   Dtype
---  ------                         --------------   -----
 0   Customer_ID                    100000 non-null  object
 1   Age                            100000 non-null  float64
 2   Annual_Income                  100000 non-null  float64
 3   Gender                         100000 non-null  object
 4   Purchase_History               100000 non-null  int64
 5   Product_Category               100000 non-null  object
 6   Customer_Satisfaction          100000 non-null  float64
 7   Loyalty_Points                 100000 non-null  float64
 8   Marital_Status                 100000 non-null  object
 9   Number_of_Children             100000 non-null  float64
 10  Employment_Status              100000 non-null  object
 11  Credit_Score                   100000 non-null  float64
 12  Owns_House                     100000 non-null  bool
 13  Monthly_Expenditure            100000 non-null  float64
 14  Internet_Usage_Hours_per_Week  100000 non-null  float64
dtypes: bool(1), float64(8), int64(1), object(5)
memory usage: 10.8+ MB
```

In [37]:
```
df.select_dtypes(include=['float64']).columns
```

Out[37]:
```
Index(['Age', 'Annual_Income', 'Customer_Satisfaction', 'Loyalty_Points',
       'Number_of_Children', 'Credit_Score', 'Monthly_Expenditure',
       'Internet_Usage_Hours_per_Week'],
      dtype='object')
```

In [38]:
```
for i in df.select_dtypes(include=['float64']).columns:
        print(i," has negative values:",(df[i]<0).sum())
```

```
Age  has negative values: 0
Annual_Income  has negative values: 0
Customer_Satisfaction  has negative values: 0
Loyalty_Points  has negative values: 0
Number_of_Children  has negative values: 0
Credit_Score  has negative values: 0
Monthly_Expenditure  has negative values: 0
Internet_Usage_Hours_per_Week  has negative values: 0
```

In [39]:
```
for i in df.select_dtypes(exclude=['float64']):
    if i!='Customer_ID':
        print(df[i].value_counts(),end="\n\n")
```

```
Gender
Female               25047
Male                 25004
Prefer not to say    25002
Non-binary           24947
Name: count, dtype: int64

Purchase_History
0    59970
1    40030
Name: count, dtype: int64

Product_Category
Grocery        20129
Electronics    20031
Clothing       19995
Furniture      19952
Books          19893
Name: count, dtype: int64

Marital_Status
Divorced    25150
Widowed     25052
Married     24999
Single      24799
Name: count, dtype: int64

Employment_Status
Unemployed    25129
Retired       25036
Employed      24969
Student       24866
Name: count, dtype: int64

Owns_House
True     70229
False    29771
Name: count, dtype: int64
```

Outlier Detection and Handling Using IQR

In [40]:
```
# Outlier detection and handling function using IQR Method:
def outlier_iqr_score(col):
    q1 = np.percentile(df[col], 25)
    q3 = np.percentile(df[col], 75)
    iqr=q3-q1
    lower_fence=q1-1.5*iqr
```

```
    upper_fence=q3+1.5*iqr
    print("__________",col,"__________")
    print("mean is :",df[col].mean())
    print("Median is :",df[col].median())
    print(f"q1 is {q1}, q3 is {q3}, iqr is {iqr}\nlower_fence is {lower_fence}, upper_fe
    count=0
    for i in df[col]:
        if i<lower_fence or i>upper_fence:
            count+=1
    print("outliers BEFORE",count,end="\n\n")
    m=df[col].median()
    df[col] = df[col].apply(lambda x: m if (x < lower_fence or x > upper_fence) else x)
    dount=0
    for ii in df[col]:
        if ii<lower_fence or ii>upper_fence:
            dount+=1
    print("outliers AFTER",dount)
    print("*****************************")
    print("\n\n")
```

In [41]:
```
for i in df.select_dtypes(include='float64'):
    outlier_iqr_score(i)
```

```
__________ Age __________
mean is : 39.51787
Median is : 39.0
q1 is 32.0, q3 is 47.0, iqr is 15.0
lower_fence is 9.5, upper_fence is 69.5

outliers BEFORE 1133

outliers AFTER 0
*****************************




__________ Annual_Income __________
mean is : 61171.717682091534
Median is : 61149.165
q1 is 50653.645, q3 is 69848.6125, iqr is 19194.967500000006
lower_fence is 21861.193749999988, upper_fence is 98641.06375000002

outliers BEFORE 1897

outliers AFTER 0
*****************************




__________ Customer_Satisfaction __________
mean is : 5.55654
Median is : 6.0
q1 is 3.0, q3 is 8.0, iqr is 5.0
lower_fence is -4.5, upper_fence is 15.5

outliers BEFORE 0

outliers AFTER 0
*****************************




__________ Loyalty_Points __________
mean is : 499.5524942857143
Median is : 499.5524942857143
```

```
q1 is 433.8, q3 is 565.28, iqr is 131.47999999999996
lower_fence is 236.58000000000007, upper_fence is 762.4999999999999

outliers BEFORE 870

outliers AFTER 0
*****************************




__________ Number_of_Children __________
mean is : 1.99694
Median is : 2.0
q1 is 1.0, q3 is 3.0, iqr is 2.0
lower_fence is -2.0, upper_fence is 6.0

outliers BEFORE 0

outliers AFTER 0
*****************************




__________ Credit_Score __________
mean is : 573.8586
Median is : 563.0
q1 is 443.0, q3 is 706.0, iqr is 263.0
lower_fence is 48.5, upper_fence is 1100.5

outliers BEFORE 0

outliers AFTER 0
*****************************




__________ Monthly_Expenditure __________
mean is : 1997.967156172418
Median is : 1997.545
q1 is 1684.41, q3 is 2315.5025, iqr is 631.0925
lower_fence is 737.7712500000001, upper_fence is 3262.14125

outliers BEFORE 1090

outliers AFTER 0
*****************************




__________ Internet_Usage_Hours_per_Week __________
mean is : 31.9949
Median is : 31.0
q1 is 19.0, q3 is 45.0, iqr is 26.0
lower_fence is -20.0, upper_fence is 84.0

outliers BEFORE 0

outliers AFTER 0
*****************************
```

In [392…

```
df.duplicated().sum()
```

```
0
```

Out[392]:

In [393…
```
df.columns
```

Out[393]:
```
Index(['Age', 'Annual_Income', 'Gender', 'Purchase_History',
       'Product_Category', 'Customer_Satisfaction', 'Loyalty_Points',
       'Marital_Status', 'Number_of_Children', 'Employment_Status',
       'Credit_Score', 'Owns_House', 'Monthly_Expenditure',
       'Internet_Usage_Hours_per_Week'],
      dtype='object')
```

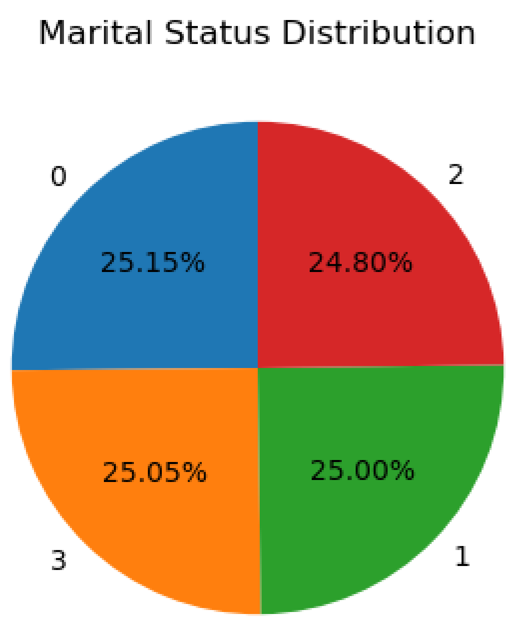
In [44]:
```
#Observations
print("OBSERVATIONS",end="\n\n")
print("1.] People whos Monthly Income is LESS than Monthly Expenditure =",((df['Annual_I
print("2.] Children whos age is less than 18 and NOT Single =",((df['Age']<18) & (df['Ma
print("3.] Customer who are repeated = ",df['Customer_ID'].duplicated().sum())
```

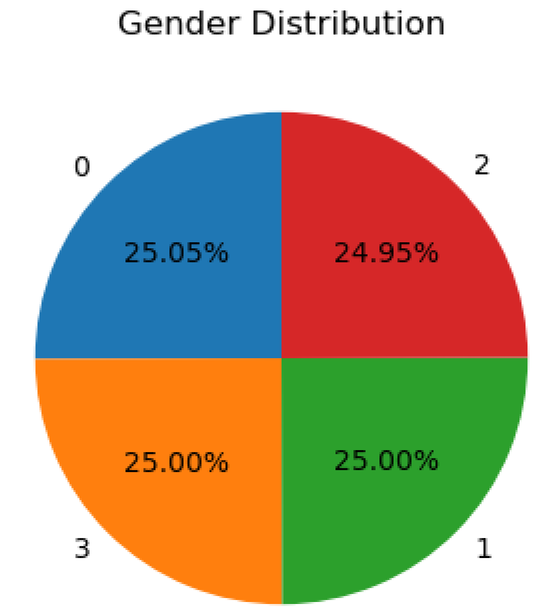
```
OBSERVATIONS

1.] People whos Monthly Income is LESS than Monthly Expenditure = 692
2.] Children whos age is less than 18 and NOT Single = 1948
3.] Customer who are repeated =  5341
```

In [45]:
```
median=df['Age'][( (df['Age']<18) & (df['Marital_Status']!='Single') )==True].mean()
```

In [46]:
```
#Replacing age with median
df['Age']=df.apply(lambda x: median if (x['Age']<18 and x['Marital_Status']!='Single')==
```

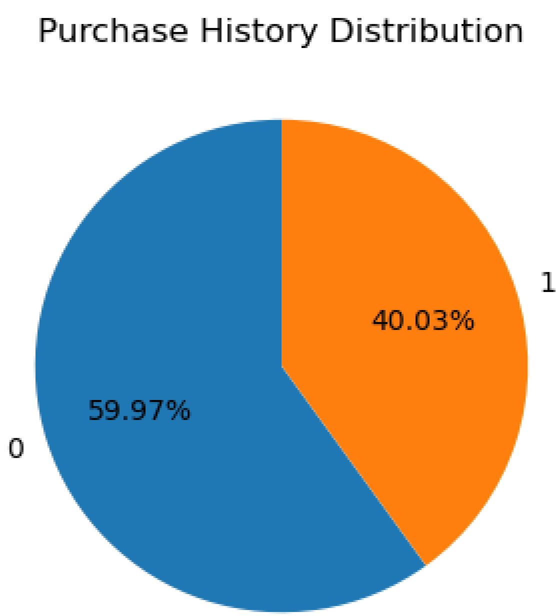
In [197…
```
marital_counts = df['Marital_Status'].value_counts()
plt.figure(figsize=(4,4))
plt.pie(marital_counts, labels=marital_counts.index, autopct='%1.2f%%', startangle=90)
plt.title('Marital Status Distribution')
plt.show()

gender_counts = df['Gender'].value_counts()
plt.figure(figsize=(4,4))
plt.pie(gender_counts, labels=marital_counts.index, autopct='%1.2f%%', startangle=90)
plt.title('Gender Distribution')
plt.show()
```





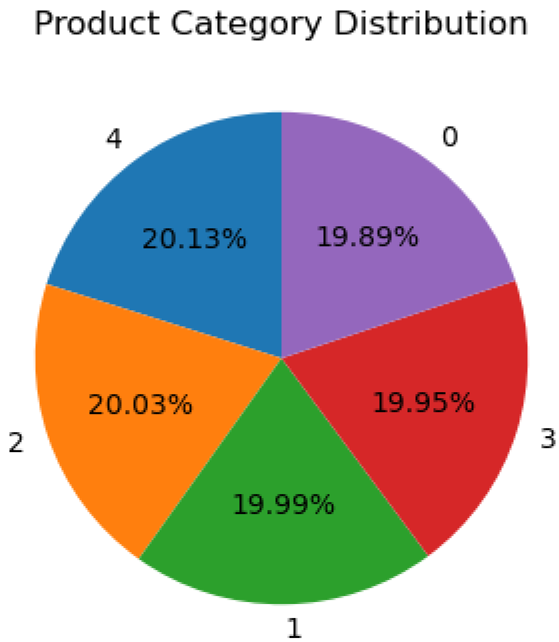
In [196…

```
purchase_counts = df['Purchase_History'].value_counts()
plt.figure(figsize=(4,4))
plt.pie(purchase_counts, labels=purchase_counts.index, autopct='%1.2f%%', startangle=90)
plt.title('Purchase History Distribution')
plt.show()
```



In [237…

```
product_counts = df['Product_Category'].value_counts()
plt.figure(figsize=(4,4))
plt.pie(product_counts, labels=product_counts.index, autopct='%1.2f%%', startangle=90)
plt.title('Product Category Distribution')
plt.show()
```



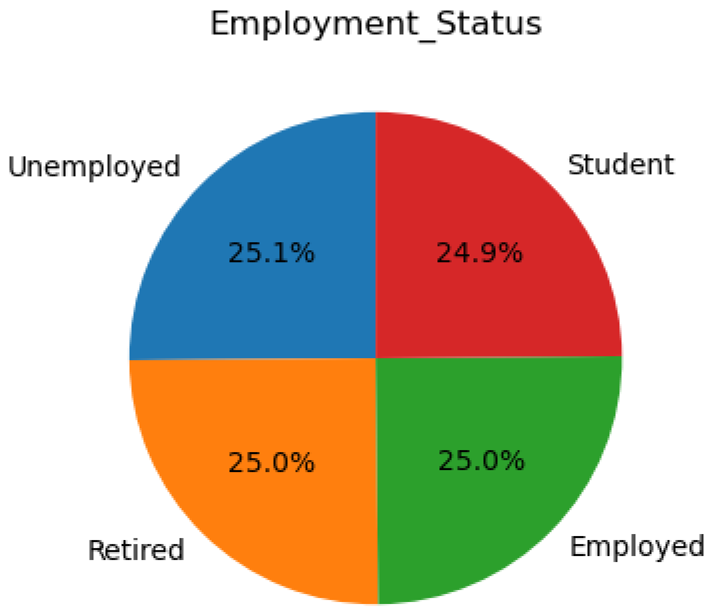
In [50]:
```
purchase_counts
```

Out[50]:
```
Purchase_History
0    59970
1    40030
Name: count, dtype: int64
```

In [51]:
```
df.columns
```

Out[51]:
```
Index(['Customer_ID', 'Age', 'Annual_Income', 'Gender', 'Purchase_History',
       'Product_Category', 'Customer_Satisfaction', 'Loyalty_Points',
       'Marital_Status', 'Number_of_Children', 'Employment_Status',
       'Credit_Score', 'Owns_House', 'Monthly_Expenditure',
       'Internet_Usage_Hours_per_Week'],
      dtype='object')
```

In [52]:
```
employee_counts = df['Employment_Status'].value_counts()
plt.figure(figsize=(4,4))
plt.pie(employee_counts, labels=employee_counts.index, autopct='%1.1f%%', startangle=90)
plt.title('Employment_Status')
plt.show()
```

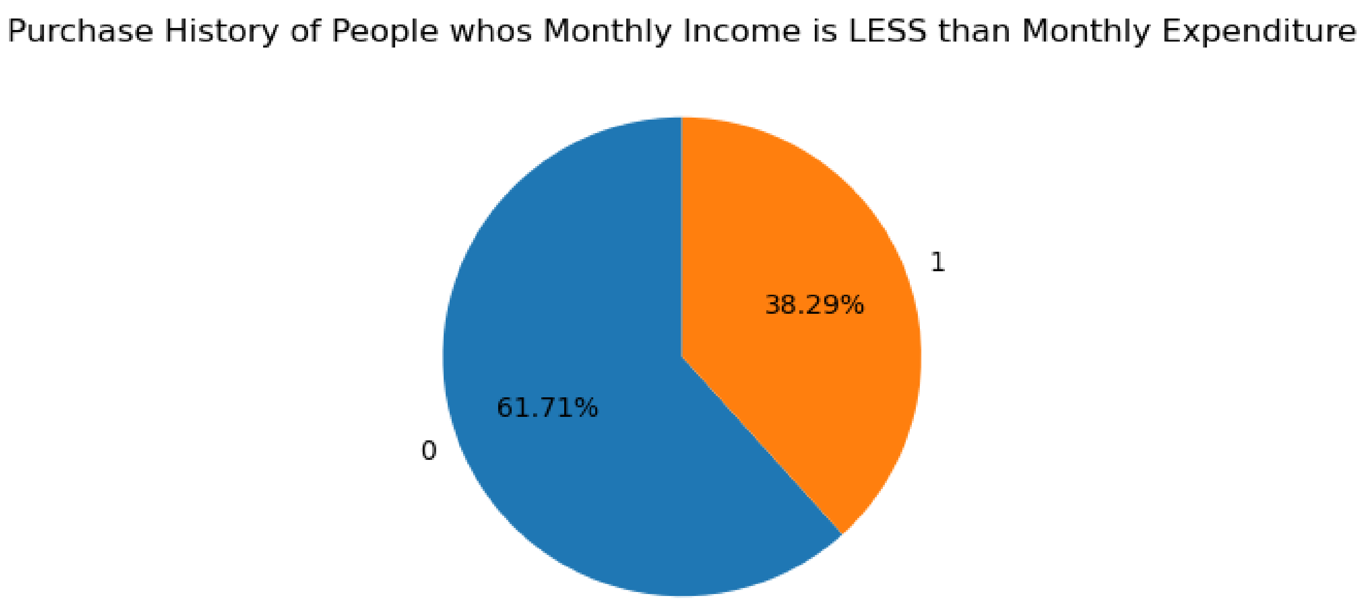


In [ ]:

```
```

In [205…

```
obs1_counts = df["Purchase_History"][(df['Annual_Income']/12)<df['Monthly_Expenditure']]
plt.figure(figsize=(4,4))
plt.pie(obs1_counts, labels=obs1_counts.index, autopct='%1.2f%%', startangle=90)
plt.title('Purchase History of People whos Monthly Income is LESS than Monthly Expenditu
plt.show()
```



In [54]:

```
df2=df.copy()
```

In [55]:

```
df_backup=df.copy() #Created back up copy
```

In [56]:

```
df.drop('Customer_ID',axis=1,inplace=True)
```

In [57]:

```
df.head(1)
```

Out[57]:

	Age	Annual_Income	Gender	Purchase_History	Product_Category	Customer_Satisfaction	Loyalty_Points	I
0	45.0	72633.53	Non-	0	Electronics	9.0	541.11	

In [59]:
```
#To split dataframe into x and y
from sklearn.model_selection import train_test_split
```

In [60]:
```
#To bring independent variable to same scale
from sklearn.preprocessing import StandardScaler,LabelEncoder
```

In [61]:
```
#To import Logistic Regression Model
from sklearn.linear_model import LogisticRegression
```

In [62]:
```
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report,roc_cu
```

In [65]:
```
le=LabelEncoder()
```

In [66]:
```
for i in df.select_dtypes(exclude='float64').columns:
    df[i]=le.fit_transform(df[i])
```

In [70]:
```
scaler=StandardScaler()
```

In [82]:
```
#Spliting data into x and y
y=df['Purchase_History']
x = df.drop('Purchase_History', axis=1)
```

In [84]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2, random_state=32)
```

In [88]:
```
x_train=scaler.fit_transform(x_train)
```

In [90]:
```
x_test=scaler.fit_transform(x_test)
```

Logistic Regression

In [173…
```
model=LogisticRegression()s
model2 = LogisticRegression(class_weight='balanced')
```

In [174…
```
#Training the model=LogisticRegression() using x-train and y-train
model.fit(x_train,y_train)
```

Out[174]:
```
LogisticRegression
LogisticRegression()
```

In [171…
```
#Prediction on trained data
y_train_predict=model.predict(x_train)
```

In [172…
```
#Checking the accuracy
train_accuracy=accuracy_score(y_train,y_train_predict)
train_accuracy
```

Out[172]:
```
0.601025
```

In [153…
```
train_cm=confusion_matrix(y_train,y_train_predict)
```

In [154…
```
#Checking TP,FP,FN,TN values
train_tn,train_fp,train_fn,train_tp=confusion_matrix(y_train,y_train_predict).ravel()
```

```
print("train_tn",train_tn)
print("train_fp",train_fp)
print("train_fn",train_fn)
print("train_tp",train_tp)
```

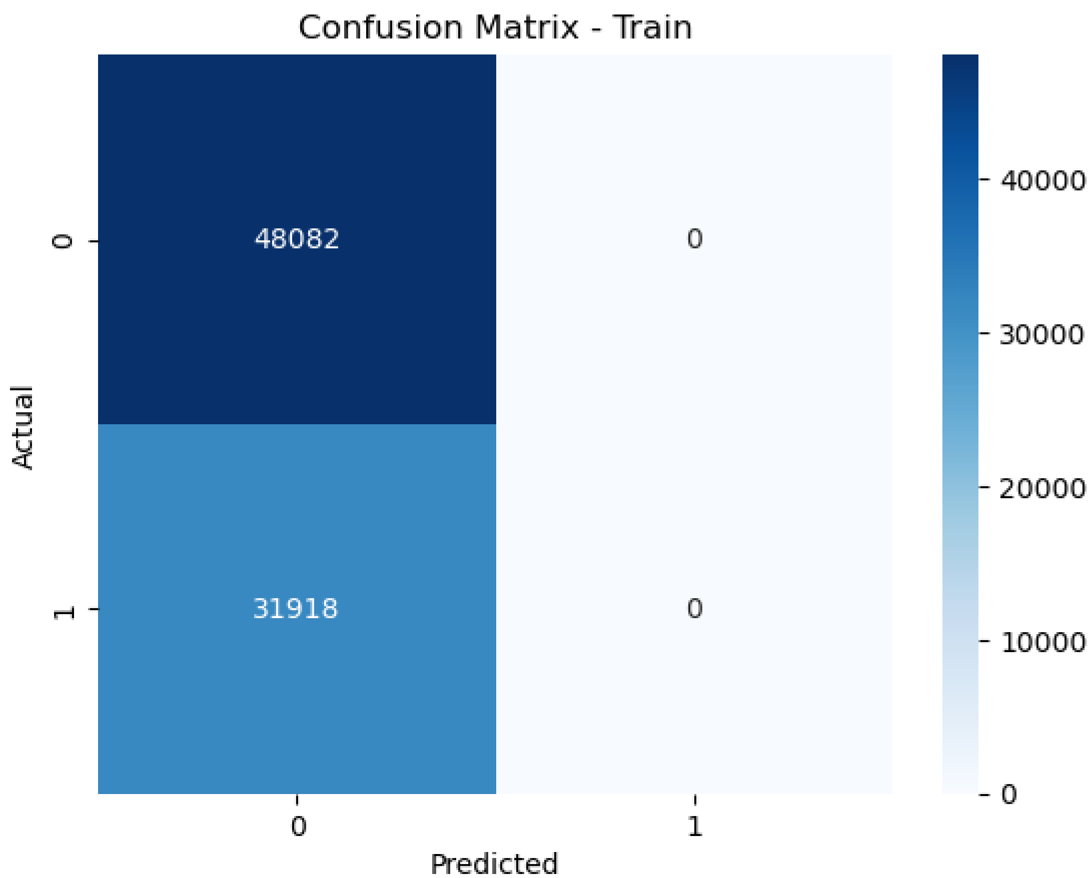
```
train_tn 48082
train_fp 0
train_fn 31918
train_tp 0
```

In [155…

```
#Plotting heat map
sns.heatmap(train_cm,fmt='d',annot=True,cmap='Blues')
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix - Train")
plt.show
```

Out[155]:

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



In [161…

```
#Prediction on test data
y_test_predict=model.predict(x_test)
```

In [175…

```
#Checking TP,FP,FN,TN values
test_tn,test_fp,test_fn,test_tp=confusion_matrix(y_test,y_test_predict).ravel()
print("test_tn",test_tn)
print("test_fp",test_fp)
print("test_fn",test_fn)
print("test_tp",test_tp)
```

```
test_tn 11888
test_fp 0
test_fn 8112
test_tp 0
```

In [180…

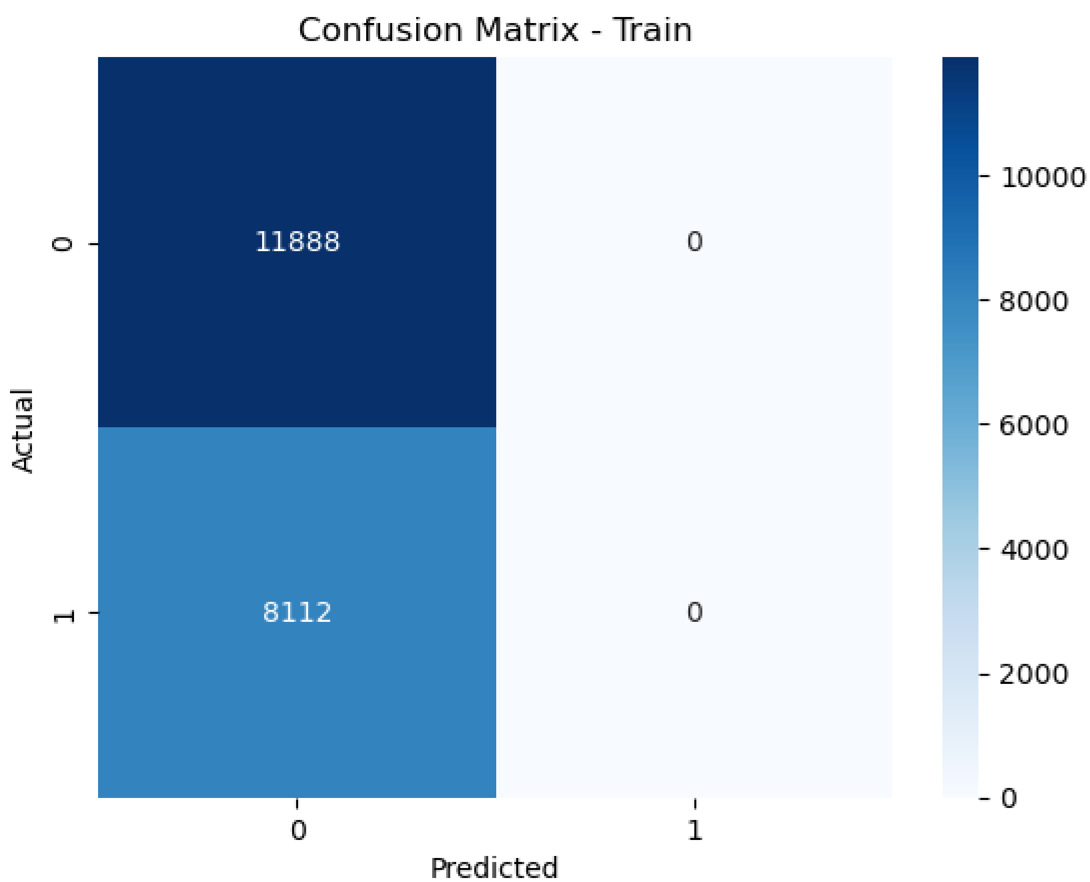
```
#Checking test accuracy score
test_accuracy=accuracy_score(y_test,y_test_predict)
test_accuracy
```

Out[180]: 0.5944

In [177…
```
test_cm=confusion_matrix(y_test,y_test_predict)
```

In [178…
```
sns.heatmap(test_cm,fmt='d',annot=True,cmap='Blues')
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix - Train")
plt.show
```

Out[178]: <function matplotlib.pyplot.show(close=None, block=None)>



LogisticRegression Model Observations

01-We can see that for test and train TP and FP is 0.

02-Train accuracy is 60.1% and test accuracy is 59.4%

03-There can be class imbalance inbalance in data which needs to bed fixed

We can try using model2=LogisticRegression(class_weight='balanced')

In [184…
```
#Training the model2=LogisticRegression(class_weight='balanced') using x-train and y-tra
model2.fit(x_train,y_train)
#Prediction on trained data
y_train_predict=model2.predict(x_train)
#Checking the accuracy
train_accuracy=accuracy_score(y_train,y_train_predict)
train_accuracy
```

Out[184]: 0.5037375

In [186…
```
#Checking TP,FP,FN,TN values
```

```
train_tn,train_fp,train_fn,train_tp=confusion_matrix(y_train,y_train_predict).ravel()
print("train_tn",train_tn)
print("train_fp",train_fp)
print("train_fn",train_fn)
print("train_tp",train_tp)
```

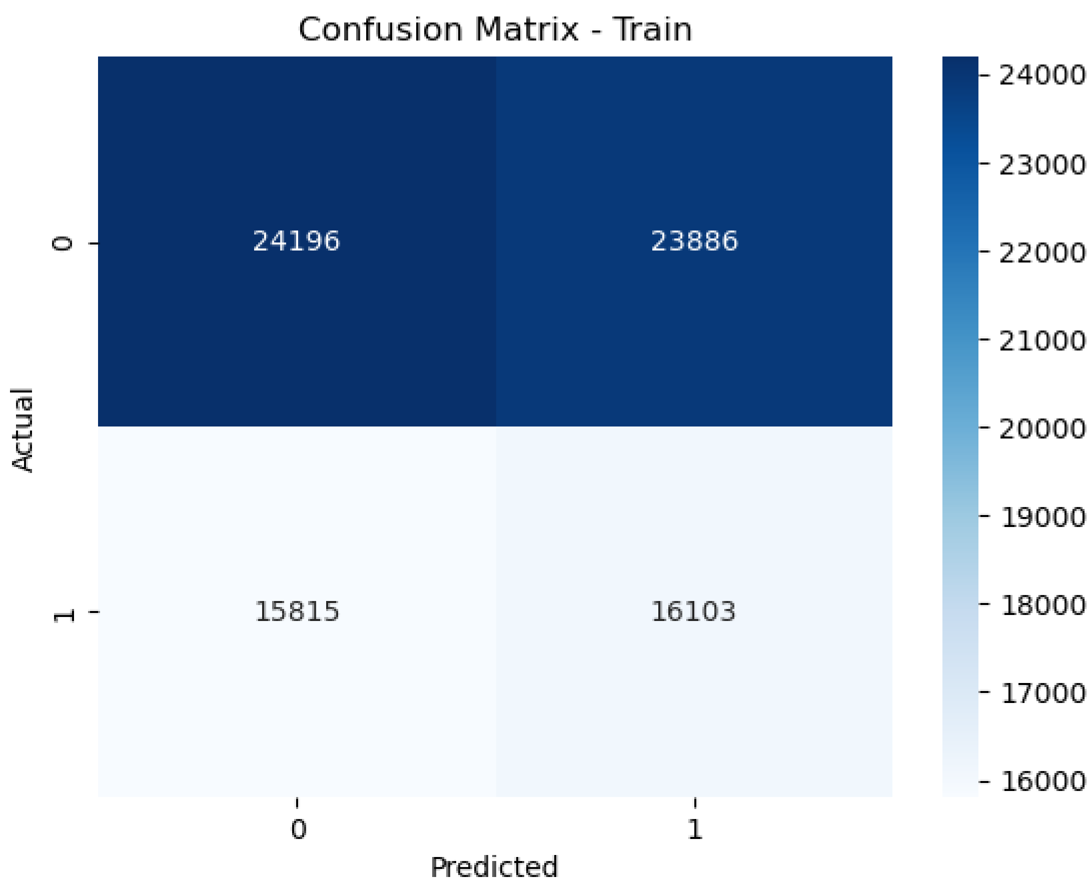
```
train_tn 24196
train_fp 23886
train_fn 15815
train_tp 16103
```

In [188…

```
#Plotting heat map
train_cm=confusion_matrix(y_train,y_train_predict)
sns.heatmap(train_cm,fmt='d',annot=True,cmap='Blues')
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix - Train")
plt.show
```

Out[188]:

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



In [189…

```
#Prediction on test data
y_test_predict=model2.predict(x_test)
#Checking the accuracy
test_accuracy=accuracy_score(y_test,y_test_predict)
test_accuracy
```

Out[189]:

```
0.49885
```

LogisticRegression(class_weight='balanced') Observations

1-We can see that for test and train TP and FP is NO LONGER 0.

2-However Accuracy has reduced Train accuracy is 50.3% and test accuracy is49.8%

We can try using SMOTE and also try Random Forest

In [214…
```
df.groupby(['Purchase_History'])['Loyalty_Points'].describe()
```

Out[214]:

Purchase_History	count	mean	std	min	25%	50%	75%	max
0	59970.0	499.444313	95.240596	236.99	435.13	499.552494	563.92	762.43
1	40030.0	499.410016	95.122746	236.72	435.38	499.552494	563.50	762.47

In [ ]:
```
##Loyalty_Points is having same 5 point summary for
```

In [218…
```
df.groupby(['Purchase_History'])['Age'].describe()
```

Out[218]:

Purchase_History	count	mean	std	min	25%	50%	75%	max
0	59970.0	39.549486	11.106742	10.0	32.0	39.0	47.0	69.0
1	40030.0	39.419943	11.097934	10.0	32.0	39.0	47.0	69.0

In [219…
```
df.groupby(['Purchase_History'])['Annual_Income'].describe()
```

Out[219]:

Purchase_History	count	mean	std	min	25%	50%	75%	max
0	59970.0	60149.568464	13921.098557	21891.77	50967.945	61149.165	69175.2200	98599.80
1	40030.0	60077.765374	13876.405985	21907.95	50818.685	61149.165	69165.9325	98633.17

In [216…
```
df.columns
```

Out[216]:
```
Index(['Age', 'Annual_Income', 'Gender', 'Purchase_History',
       'Product_Category', 'Customer_Satisfaction', 'Loyalty_Points',
       'Marital_Status', 'Number_of_Children', 'Employment_Status',
       'Credit_Score', 'Owns_House', 'Monthly_Expenditure',
       'Internet_Usage_Hours_per_Week'],
      dtype='object')
```

SMOTE

In [261…
```
from imblearn.over_sampling import SMOTE
smote=SMOTE()
x_train_smote,y_train_smote=smote.fit_resample(x_train,y_train)
x_test_smote,y_test_smote=smote.fit_resample(x_test,y_test)
```

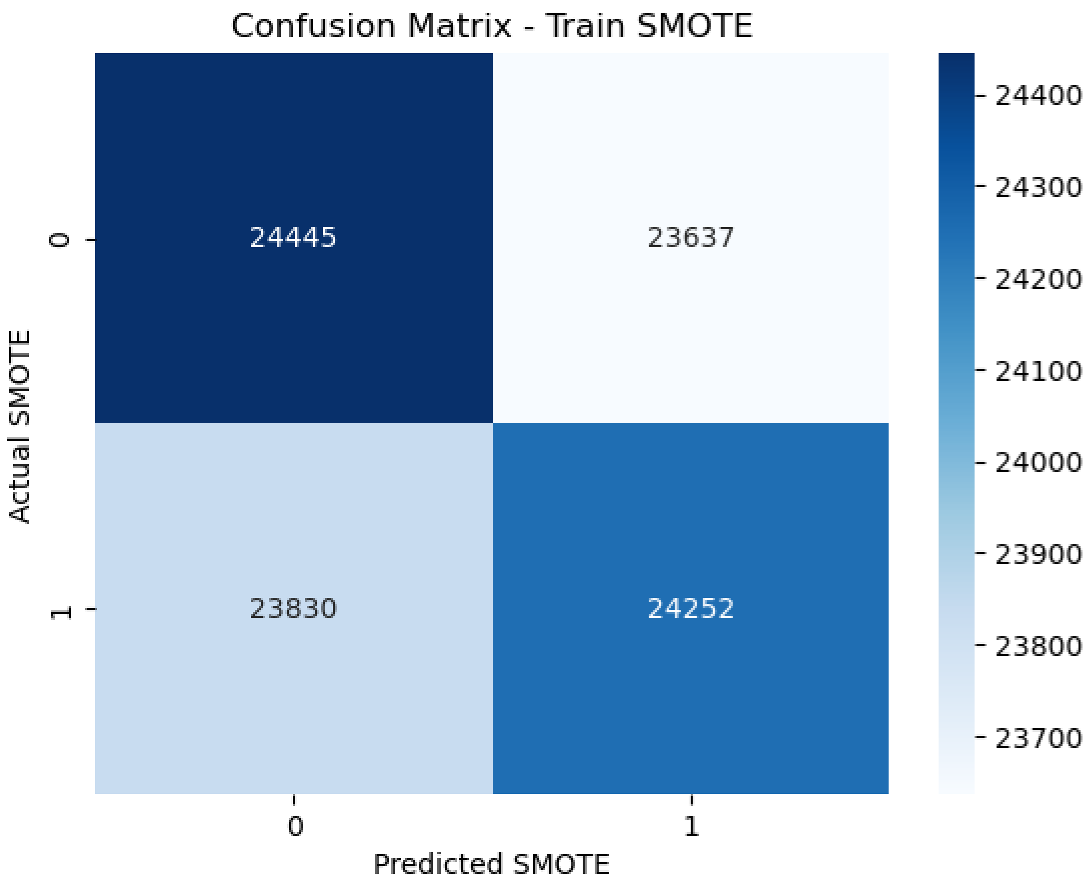
In [260…
```
#Training the model=LogisticRegression() using SMOTE x-train and y-train
model.fit(x_train_smote,y_train_smote)
y_train_smote_predict=model.predict(x_train_smote)

#Checking the accuracy
train_smote_accuracy=accuracy_score(y_train_smote,y_train_smote_predict)
print("train_smote_accuracy = ",train_smote_accuracy,end="\n\n")

#Checking TP,FP,FN,TN values
train_smote_tn,train_smote_fp,train_smote_fn,train_smote_tp=confusion_matrix(y_train_smo
print("train_smote_tn =",train_smote_tn)
print("train_smote_fp =",train_smote_fp)
print("train_smote_fn =",train_smote_fn)
```

```
print("train_smote_tp =",train_smote_tp)
print("\n")

train_smote_cm=confusion_matrix(y_train_smote,y_train_smote_predict)

#Plotting heat map
sns.heatmap(train_smote_cm,fmt='d',annot=True,cmap='Blues')
plt.xlabel("Predicted SMOTE")
plt.ylabel("Actual SMOTE")
plt.title("Confusion Matrix - Train SMOTE")
plt.show
```

```
train_smote_accuracy =  0.5063953246537166

train_smote_tn = 24445
train_smote_fp = 23637
train_smote_fn = 23830
train_smote_tp = 24252
```

Out[260]:

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



In [262…

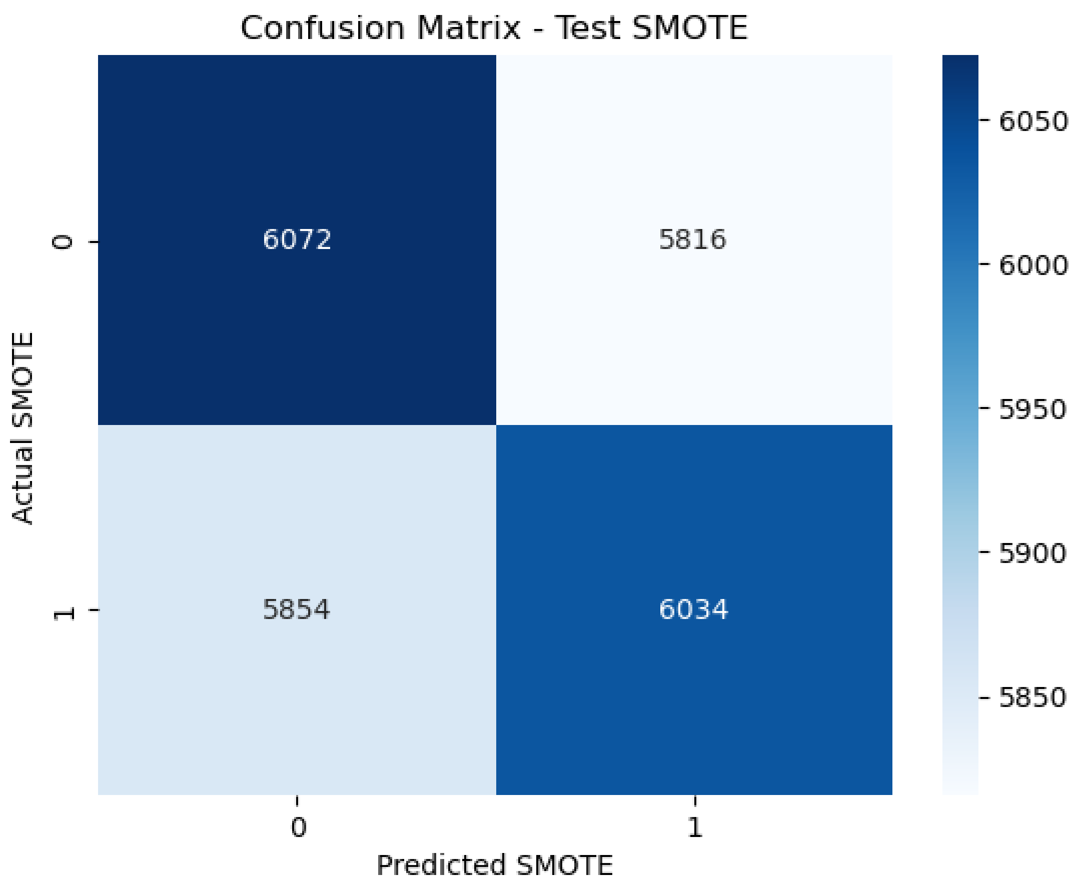
```
#Testing the model=LogisticRegression() using SMOTE x-test and y-test
model.fit(x_test_smote,y_test_smote)
y_test_smote_predict=model.predict(x_test_smote)

#Checking the accuracy
test_smote_accuracy=accuracy_score(y_test_smote,y_test_smote_predict)
print("test_smote_accuracy = ",test_smote_accuracy,end="\n\n")

#Checking TP,FP,FN,TN values
test_smote_tn,test_smote_fp,test_smote_fn,test_smote_tp=confusion_matrix(y_test_smote,y_
print("test_smote_tn =",test_smote_tn)
print("test_smote_fp =",test_smote_fp)
print("test_smote_fn =",test_smote_fn)
print("test_smote_tp =",test_smote_tp)
```

```
print("\n")

test_smote_cm=confusion_matrix(y_test_smote,y_test_smote_predict)

#Plotting heat map
sns.heatmap(test_smote_cm,fmt='d',annot=True,cmap='Blues')
plt.xlabel("Predicted SMOTE")
plt.ylabel("Actual SMOTE")
plt.title("Confusion Matrix - Test SMOTE")
plt.show
```

```
test_smote_accuracy =  0.5091689098250336

test_smote_tn = 6072
test_smote_fp = 5816
test_smote_fn = 5854
test_smote_tp = 6034
```

Out[262]:

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



DecisionTreeClassifier

In [320…

```
from sklearn.tree import DecisionTreeClassifier
```

In [321…

```
list_train_accuracy_dt=[]
list_test_accuracy_dt=[]
for i in range(1,31):
    model_dt=DecisionTreeClassifier(max_depth=i)
    model_dt.fit(x_train,y_train)
    y_train_predict_dt=model_dt.predict(x_train)
    y_test_predict_dt=model_dt.predict(x_test)
    train_accuracy_dt=accuracy_score(y_train,y_train_predict_dt)
    test_accuracy_dt=accuracy_score(y_test,y_test_predict_dt)
    list_train_accuracy_dt.append(train_accuracy_dt)
```

```
    list_test_accuracy_dt.append(test_accuracy_dt)
    print("max_depth =",i,"train_accuracy_dt =",train_accuracy_dt,"test_accuracy_dt =",t
```

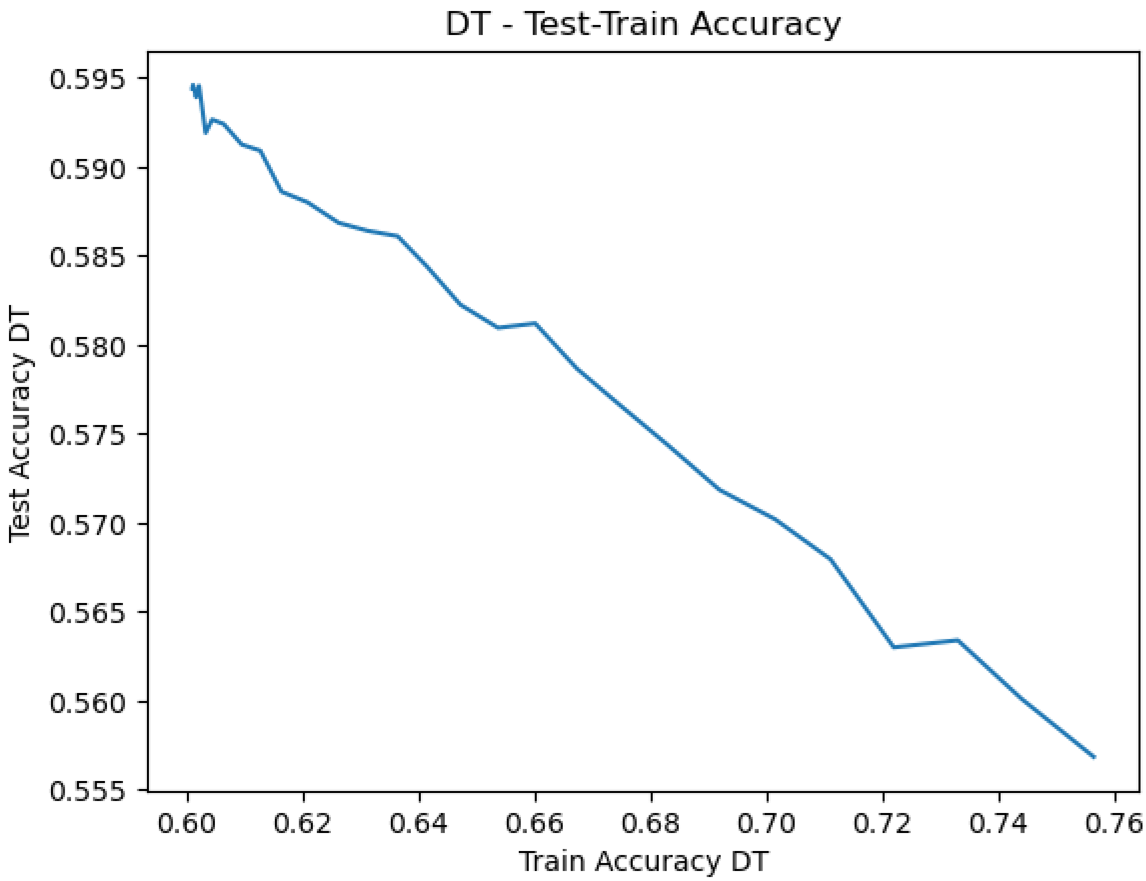
```
max_depth = 1 train_accuracy_dt = 0.601025 test_accuracy_dt = 0.5944
max_depth = 2 train_accuracy_dt = 0.60115 test_accuracy_dt = 0.5946
max_depth = 3 train_accuracy_dt = 0.60115 test_accuracy_dt = 0.5946
max_depth = 4 train_accuracy_dt = 0.6014 test_accuracy_dt = 0.5943
max_depth = 5 train_accuracy_dt = 0.6017 test_accuracy_dt = 0.5939
max_depth = 6 train_accuracy_dt = 0.6021875 test_accuracy_dt = 0.59455
max_depth = 7 train_accuracy_dt = 0.603325 test_accuracy_dt = 0.5919
max_depth = 8 train_accuracy_dt = 0.6044375 test_accuracy_dt = 0.59265
max_depth = 9 train_accuracy_dt = 0.6064375 test_accuracy_dt = 0.5924
max_depth = 10 train_accuracy_dt = 0.6095375 test_accuracy_dt = 0.59125
max_depth = 11 train_accuracy_dt = 0.6127625 test_accuracy_dt = 0.5909
max_depth = 12 train_accuracy_dt = 0.6164 test_accuracy_dt = 0.5886
max_depth = 13 train_accuracy_dt = 0.6209 test_accuracy_dt = 0.588
max_depth = 14 train_accuracy_dt = 0.62625 test_accuracy_dt = 0.58685
max_depth = 15 train_accuracy_dt = 0.631275 test_accuracy_dt = 0.5864
max_depth = 16 train_accuracy_dt = 0.636425 test_accuracy_dt = 0.5861
max_depth = 17 train_accuracy_dt = 0.641475 test_accuracy_dt = 0.5844
max_depth = 18 train_accuracy_dt = 0.64725 test_accuracy_dt = 0.58225
max_depth = 19 train_accuracy_dt = 0.65375 test_accuracy_dt = 0.58095
max_depth = 20 train_accuracy_dt = 0.66015 test_accuracy_dt = 0.5812
max_depth = 21 train_accuracy_dt = 0.6673625 test_accuracy_dt = 0.57865
max_depth = 22 train_accuracy_dt = 0.6747875 test_accuracy_dt = 0.5766
max_depth = 23 train_accuracy_dt = 0.6830625 test_accuracy_dt = 0.57435
max_depth = 24 train_accuracy_dt = 0.6919125 test_accuracy_dt = 0.57185
max_depth = 25 train_accuracy_dt = 0.7014875 test_accuracy_dt = 0.5702
max_depth = 26 train_accuracy_dt = 0.7110625 test_accuracy_dt = 0.56795
max_depth = 27 train_accuracy_dt = 0.7219625 test_accuracy_dt = 0.563
max_depth = 28 train_accuracy_dt = 0.733 test_accuracy_dt = 0.5634
max_depth = 29 train_accuracy_dt = 0.744025 test_accuracy_dt = 0.5601
max_depth = 30 train_accuracy_dt = 0.7564375 test_accuracy_dt = 0.55685
```

In [322…

```
dt={"Train Accuracy DT":list_train_accuracy_dt,"Test Accuracy DT":list_test_accuracy_dt}
dt=pd.DataFrame(dt)
```

```
sns.lineplot(x='Train Accuracy DT', y='Test Accuracy DT', data=dt)
plt.title('DT - Test-Train Accuracy')
```

Out[322]: Text(0.5, 1.0, 'DT - Test-Train Accuracy')



In [323…

```
#Checking TP,FP,FN,TN values maxdepth =30
train_smote_tn_dt,train_smote_fp_dt,train_smote_fn_dt,train_smote_tp_dt=confusion_matrix
print("train_smote_tn_dt =",train_smote_tn_dt)
print("train_smote_fp_dt =",train_smote_fp_dt)
print("train_smote_fn_dt =",train_smote_fn_dt)
print("train_smote_tp_dt =",train_smote_tp_dt)
print("\n")
```

```
train_smote_tn_dt = 46537
train_smote_fp_dt = 1545
train_smote_fn_dt = 17940
train_smote_tp_dt = 13978
```

Observation:

1.]In Decision Tree As max depth is increasing, train accuracy is increasing TP and FP values are no longer zero

2.]However test accuracy is going down, as max depth is increasing

DT using Entropy

In [332…

```
list_train_accuracy_dt_entropy=[]
list_test_accuracy_dt_entropy=[]
for i in range(1,30):
    model_dt_entropy=DecisionTreeClassifier(criterion='entropy',max_depth=i)
    model_dt_entropy.fit(x_train,y_train)
    y_train_predict_dt_entropy=model_dt_entropy.predict(x_train)
```

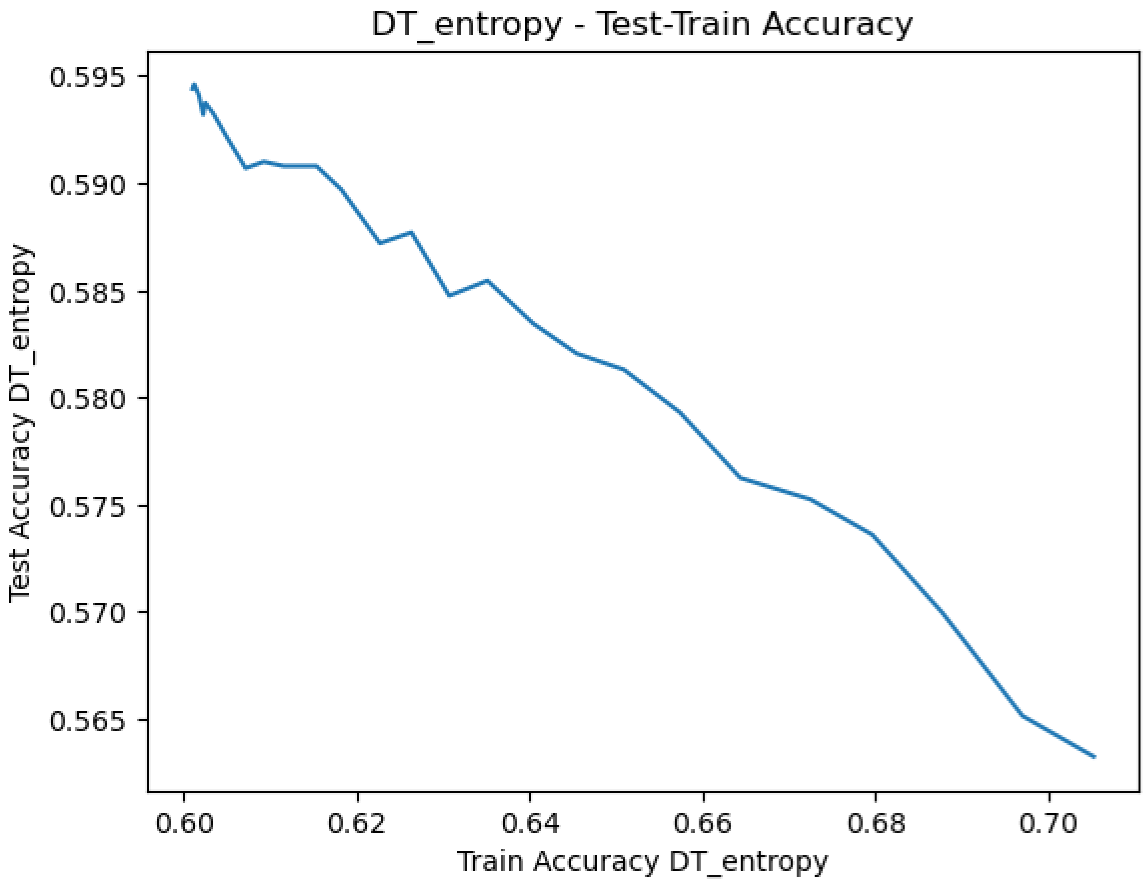
```
    y_test_predict_dt_entropy=model_dt_entropy.predict(x_test)
    train_accuracy_dt_entropy=accuracy_score(y_train,y_train_predict_dt_entropy)
    test_accuracy_dt_entropy=accuracy_score(y_test,y_test_predict_dt_entropy)
    list_train_accuracy_dt_entropy.append(train_accuracy_dt_entropy)
    list_test_accuracy_dt_entropy.append(test_accuracy_dt_entropy)
    print("max_depth =",i,"train_accuracy_dt_entropy =",train_accuracy_dt_entropy,"test_
```

```
max_depth = 1 train_accuracy_dt_entropy = 0.601025 test_accuracy_dt_entropy = 0.5944

max_depth = 2 train_accuracy_dt_entropy = 0.601025 test_accuracy_dt_entropy = 0.5944

max_depth = 3 train_accuracy_dt_entropy = 0.6012125 test_accuracy_dt_entropy = 0.5946

max_depth = 4 train_accuracy_dt_entropy = 0.6012625 test_accuracy_dt_entropy = 0.5946

max_depth = 5 train_accuracy_dt_entropy = 0.601425 test_accuracy_dt_entropy = 0.59445

max_depth = 6 train_accuracy_dt_entropy = 0.601775 test_accuracy_dt_entropy = 0.5941

max_depth = 7 train_accuracy_dt_entropy = 0.602275 test_accuracy_dt_entropy = 0.5932

max_depth = 8 train_accuracy_dt_entropy = 0.602525 test_accuracy_dt_entropy = 0.59375

max_depth = 9 train_accuracy_dt_entropy = 0.60345 test_accuracy_dt_entropy = 0.59325

max_depth = 10 train_accuracy_dt_entropy = 0.604925 test_accuracy_dt_entropy = 0.5922

max_depth = 11 train_accuracy_dt_entropy = 0.607175 test_accuracy_dt_entropy = 0.5907

max_depth = 12 train_accuracy_dt_entropy = 0.609275 test_accuracy_dt_entropy = 0.591

max_depth = 13 train_accuracy_dt_entropy = 0.6115875 test_accuracy_dt_entropy = 0.5908

max_depth = 14 train_accuracy_dt_entropy = 0.6153625 test_accuracy_dt_entropy = 0.5908

max_depth = 15 train_accuracy_dt_entropy = 0.61825 test_accuracy_dt_entropy = 0.5897

max_depth = 16 train_accuracy_dt_entropy = 0.6227 test_accuracy_dt_entropy = 0.5872

max_depth = 17 train_accuracy_dt_entropy = 0.62635 test_accuracy_dt_entropy = 0.5877

max_depth = 18 train_accuracy_dt_entropy = 0.6307125 test_accuracy_dt_entropy = 0.58475

max_depth = 19 train_accuracy_dt_entropy = 0.6351375 test_accuracy_dt_entropy = 0.58545

max_depth = 20 train_accuracy_dt_entropy = 0.640425 test_accuracy_dt_entropy = 0.58345

max_depth = 21 train_accuracy_dt_entropy = 0.64545 test_accuracy_dt_entropy = 0.58205

max_depth = 22 train_accuracy_dt_entropy = 0.6509125 test_accuracy_dt_entropy = 0.5813

max_depth = 23 train_accuracy_dt_entropy = 0.6574125 test_accuracy_dt_entropy = 0.5793

max_depth = 24 train_accuracy_dt_entropy = 0.664375 test_accuracy_dt_entropy = 0.57625

max_depth = 25 train_accuracy_dt_entropy = 0.6724875 test_accuracy_dt_entropy = 0.57525

max_depth = 26 train_accuracy_dt_entropy = 0.67965 test_accuracy_dt_entropy = 0.5736

max_depth = 27 train_accuracy_dt_entropy = 0.687775 test_accuracy_dt_entropy = 0.56995

max_depth = 28 train_accuracy_dt_entropy = 0.6970125 test_accuracy_dt_entropy = 0.56515

max_depth = 29 train_accuracy_dt_entropy = 0.7052625 test_accuracy_dt_entropy = 0.56325
```

In [400…
```
train_dt_entropy_tn,train_dt_entropy_fp,train_dt_entropy_fn,train_dt_entropy_tp=confusio
print(f"train_dt_entropy_tn = {train_dt_entropy_tn},train_dt_entropy_fp= {train_dt_entro
dt={"Train Accuracy DT_entropy":list_train_accuracy_dt_entropy,"Test Accuracy DT_entropy
dt=pd.DataFrame(dt)
sns.lineplot(x='Train Accuracy DT_entropy', y='Test Accuracy DT_entropy', data=dt)
plt.title('DT_entropy - Test-Train Accuracy')
```

```
train_dt_entropy_tn = 46294,train_dt_entropy_fp= 1788,
train_dt_entropy_fn = 21791,train_dt_entropy_tp = 10127
```

Out[400]:
```
Text(0.5, 1.0, 'DT_entropy - Test-Train Accuracy')
```



Observation:

1.]In Decision Tree using Entropy As max depth is increasing, train accuracy is increasing

2.]However test accuracy is going down, as max depth is increasing

Adaboost

In [336…
```
from sklearn.ensemble import AdaBoostClassifier
```

In [383…
```
base_estimator = DecisionTreeClassifier(max_depth=1)
model_adaboost=AdaBoostClassifier(base_estimator=base_estimator,n_estimators=10,learning
```

In [388…
```
model_adaboost.fit(x_train_smote,y_train_smote)
y_train_predict_adaboost=model_adaboost.predict(x_train_smote)
```

```
C:\Users\venky\anaconda3\Lib\site-packages\sklearn\ensemble\_base.py:166: FutureWarning:
`base_estimator` was renamed to `estimator` in version 1.2 and will be removed in 1.4.
  warnings.warn(
```

In [390…
```
train_accuracy_adaboost=accuracy_score(y_train_smote,y_train_predict_adaboost)
train_accuracy_adaboost
```

Out[390]: 0.5025997254689905

In [402…
```
y_test_predict_adaboost=model_adaboost.predict(x_test_smote)
test_accuracy_adaboost=accuracy_score(y_test_smote,y_test_predict_adaboost)
test_accuracy_adaboost
```

Out[402]: 0.4874242934051144

In [406…
```
#Checking TP,FP,FN,TN values maxdepth =30
test_smote_tn_dt,test_smote_fp_dt,test_smote_fn_dt,test_smote_tp_dt=confusion_matrix(y_t
print("test_smote_tn_dt =",test_smote_tn_dt)
print("test_smote_fp_dt =",test_smote_fp_dt)
print("test_smote_fn_dt =",test_smote_fn_dt)
print("test_smote_tp_dt =",test_smote_tp_dt)
print("\n")
```

```
test_smote_tn_dt = 9446
test_smote_fp_dt = 2442
test_smote_fn_dt = 9745
test_smote_tp_dt = 2143
```

Observations

Train accuracy 0.5025997254689905 is and Test accuracy is 0.4874242934051144

RandomForestClassifier

from sklearn.ensemble import RandomForestClassifier from sklearn.metrics import classification_report

In [413…
```
for i in range(1,21):
  model_rf=RandomForestClassifier(n_estimators=i, random_state=42)
  model_rf.fit(x_train,y_train)
  y_train_predict_rf=model_rf.predict(x_train)
  y_test_predict_rf=model_rf.predict(x_test)
  train_accuracy_rf=accuracy_score(y_train,y_train_predict_rf)
  test_accuracy_rf=accuracy_score(y_test,y_test_predict_rf)
  print("n_estimators =",i,"train_accuracy_rf =",train_accuracy_rf,"test_accuracy_rf =",
```

```
n_estimators = 1 train_accuracy_rf = 0.8242375 test_accuracy_rf = 0.5212

n_estimators = 2 train_accuracy_rf = 0.8315875 test_accuracy_rf = 0.56445

n_estimators = 3 train_accuracy_rf = 0.91475 test_accuracy_rf = 0.5271

n_estimators = 4 train_accuracy_rf = 0.907975 test_accuracy_rf = 0.5609

n_estimators = 5 train_accuracy_rf = 0.953625 test_accuracy_rf = 0.53275

n_estimators = 6 train_accuracy_rf = 0.9467625 test_accuracy_rf = 0.5627

n_estimators = 7 train_accuracy_rf = 0.9740125 test_accuracy_rf = 0.53735

n_estimators = 8 train_accuracy_rf = 0.9679625 test_accuracy_rf = 0.5608

n_estimators = 9 train_accuracy_rf = 0.9848 test_accuracy_rf = 0.54305

n_estimators = 10 train_accuracy_rf = 0.980075 test_accuracy_rf = 0.56395

n_estimators = 11 train_accuracy_rf = 0.99075 test_accuracy_rf = 0.54845
```

```
n_estimators = 12 train_accuracy_rf = 0.98695 test_accuracy_rf = 0.56115

n_estimators = 13 train_accuracy_rf = 0.994 test_accuracy_rf = 0.54915

n_estimators = 14 train_accuracy_rf = 0.991825 test_accuracy_rf = 0.5642

n_estimators = 15 train_accuracy_rf = 0.9963 test_accuracy_rf = 0.5518

n_estimators = 16 train_accuracy_rf = 0.9948875 test_accuracy_rf = 0.56445

n_estimators = 17 train_accuracy_rf = 0.99755 test_accuracy_rf = 0.555

n_estimators = 18 train_accuracy_rf = 0.996525 test_accuracy_rf = 0.56585

n_estimators = 19 train_accuracy_rf = 0.998575 test_accuracy_rf = 0.55465

n_estimators = 20 train_accuracy_rf = 0.99775 test_accuracy_rf = 0.5655
```

GridSearchCV

In [416…

```
from sklearn.model_selection import GridSearchCV
```

In [421…

```
param_dict={'max_depth':[1,2,3,4,5,],
           'bootstrap':[True,False],
           'max_features':['auto','log2','sqrt'],
           'criterion':['gini','entropy']}
cv_rf=GridSearchCV(model_rf,cv=10,param_grid=param_dict,n_jobs=3)
```

In [424…

```
cv_rf.fit(x_train,y_train)
```

```
C:\Users\venky\anaconda3\Lib\site-packages\sklearn\ensemble\_forest.py:424: FutureWarnin
g: `max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep
the past behaviour, explicitly set `max_features='sqrt'` or remove this parameter as it
is also the default value for RandomForestClassifiers and ExtraTreesClassifiers.
  warn(
```

Out[424]:

```
GridSearchCV
estimator: RandomForestClassifier
RandomForestClassifier
```

In [425…

```
print("Best Parameter Using GridSearchCV: \n",cv_rf.best_params_)
```

```
Best Parameter Using GridSearchCV:
 {'bootstrap': True, 'criterion': 'gini', 'max_depth': 1, 'max_features': 'auto'}
```

In [428…

In [432…

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[432], line 4
      1 import time
      2 start=time.time()
----> 4 cv_rf2.fit(x_train,y_train)
      5 end=time.time()
      6 print('Time taken to process: %0.2f'%(end-start))

File ~\anaconda3\Lib\site-packages\sklearn\model_selection\_search.py:874, in BaseSearch
CV.fit(self, X, y, groups, **fit_params)
```

```
    868     results = self._format_results(
    869         all_candidate_params, n_splits, all_out, all_more_results
    870     )
    872     return results
--> 874 self._run_search(evaluate_candidates)
    876 # multimetric is determined here because in the case of a callable
    877 # self.scoring the return type is only known after calling
    878 first_test_score = all_out[0]["test_scores"]

File ~\anaconda3\Lib\site-packages\sklearn\model_selection\_search.py:1388, in GridSearc
hCV._run_search(self, evaluate_candidates)
   1386 def _run_search(self, evaluate_candidates):
   1387     """Search all candidates in param_grid"""
-> 1388     evaluate_candidates(ParameterGrid(self.param_grid))

File ~\anaconda3\Lib\site-packages\sklearn\model_selection\_search.py:821, in BaseSearch
CV.fit.<locals>.evaluate_candidates(candidate_params, cv, more_results)
    813 if self.verbose > 0:
    814     print(
    815         "Fitting {0} folds for each of {1} candidates,"
    816         " totalling {2} fits".format(
    817             n_splits, n_candidates, n_candidates * n_splits
    818         )
    819     )
--> 821 out = parallel(
    822     delayed(_fit_and_score)(
    823         clone(base_estimator),
    824         X,
    825         y,
    826         train=train,
    827         test=test,
    828         parameters=parameters,
    829         split_progress=(split_idx, n_splits),
    830         candidate_progress=(cand_idx, n_candidates),
    831         **fit_and_score_kwargs,
    832     )
    833     for (cand_idx, parameters), (split_idx, (train, test)) in product(
    834         enumerate(candidate_params), enumerate(cv.split(X, y, groups))
    835     )
    836 )
    838 if len(out) < 1:
    839     raise ValueError(
    840         "No fits were performed. "
    841         "Was the CV iterator empty? "
    842         "Were there no candidates?"
    843     )

File ~\anaconda3\Lib\site-packages\sklearn\utils\parallel.py:63, in Parallel.__call__(se
lf, iterable)
     58 config = get_config()
     59 iterable_with_config = (
     60     (_with_config(delayed_func, config), args, kwargs)
     61     for delayed_func, args, kwargs in iterable
     62 )
---> 63 return super().__call__(iterable_with_config)

File ~\anaconda3\Lib\site-packages\joblib\parallel.py:1098, in Parallel.__call__(self, i
terable)
   1095     self._iterating = False
   1097 with self._backend.retrieval_context():
-> 1098     self.retrieve()
   1099 # Make sure that we get a last message telling us we are done
   1100 elapsed_time = time.time() - self._start_time

File ~\anaconda3\Lib\site-packages\joblib\parallel.py:975, in Parallel.retrieve(self)
    973 try:
```

```
    974     if getattr(self._backend, 'supports_timeout', False):
--> 975         self._output.extend(job.get(timeout=self.timeout))
    976     else:
    977         self._output.extend(job.get())

File ~\anaconda3\Lib\site-packages\joblib\_parallel_backends.py:567, in LokyBackend.wrap
_future_result(future, timeout)
    564 """Wrapper for Future.result to implement the same behaviour as
    565 AsyncResults.get from multiprocessing."""
    566 try:
--> 567     return future.result(timeout=timeout)
    568 except CfTimeoutError as e:
    569     raise TimeoutError from e

File ~\anaconda3\Lib\concurrent\futures\_base.py:451, in Future.result(self, timeout)
    448 elif self._state == FINISHED:
    449     return self.__get_result()
--> 451 self._condition.wait(timeout)
    453 if self._state in [CANCELLED, CANCELLED_AND_NOTIFIED]:
    454     raise CancelledError()

File ~\anaconda3\Lib\threading.py:327, in Condition.wait(self, timeout)
    325 try:    # restore state no matter what (e.g., KeyboardInterrupt)
    326     if timeout is None:
--> 327         waiter.acquire()
    328         gotit = True
    329     else:

KeyboardInterrupt:
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

```
In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:
```