I] Importing Dataset and Basic Check

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')
```

1) Importing Dataset

```
path=r'C:\Users\venky\Downloads\ecommerce_recommendation_dataset.csv'
df=pd.read_csv(path)
df.head()
```

```
   user_id  product_id   category   price  rating  review_count
user_age  \
0    78517        1645      Books  842.23       2           155
24
1    52887         100      Books  253.76       3           331
43
2    59395         585      Books  483.65       2           236
64
3    54739        3774  Groceries  459.37       2           227
34
4    42723        2119  Groceries  150.11       2           214
51

  user_gender user_location  purchase_history  ...
product_rating_variance  \
0       Other         Urban             False  ...
0.13
1       Other      Suburban             False  ...
0.02
2      Female         Rural              True  ...
1.55
3      Female         Urban             False  ...
1.41
4      Female         Urban              True  ...
1.29

   review_sentiment_score user_engagement_score  ad_click_rate
time_of_day  \
0                   -0.28                  0.68           0.04
Night
1                    0.28                  0.11           0.89
Morning
```

```
2                     0.23                      0.35            0.99
Evening
3                     0.93                      0.73            0.16
Afternoon
4                     0.11                      0.26            0.17
Night

  day_of_week   season payment_method  coupon_used  product_popularity

0     Thursday  Summer      Debit Card        False                0.54

1     Saturday  Summer      Debit Card        False                0.77

2      Tuesday    Fall      Debit Card        False                0.14

3      Tuesday  Spring     Credit Card        False                0.18

4    Wednesday  Spring          PayPal        False                0.66


[5 rows x 51 columns]
```

2) Checking Shape and Size of Dataset

```
print(f"shape of data set is = {df.shape}")
print(f"size of data set is = {df.size}")
```

```
shape of data set is = (60000, 51)
size of data set is = 3060000
```

3) Checking Duplicate Rows and Missing Value

```
print(f"total duplicate rows in dataset is = {df.duplicated().sum()}")
print()
print(f"total missing values rows in dataset is =
{df.isnull().sum().sum()}")
print()
print(f"missing data in columns is as below\
n{df.isnull().sum()}",sep='')
```

```
total duplicate rows in dataset is = 0

total missing values rows in dataset is = 0

missing data in columns is as below
user_id                    0
product_id                 0
category                   0
price                      0
rating                     0
```

```
review_count                 0
user_age                     0
user_gender                  0
user_location                0
purchase_history             0
time_on_page                 0
add_to_cart_count            0
search_keywords              0
discount_applied             0
user_membership              0
user_browser                 0
user_device                  0
purchase_time                0
session_duration             0
clicks_on_ads                0
page_views                   0
referral_source              0
wishlist_additions           0
cart_abandonment_rate        0
average_spent                0
user_income                  0
user_education               0
user_marital_status          0
product_availability         0
stock_status                 0
product_return_rate          0
product_color                0
product_size                 0
is_top_seller                0
discount_percentage          0
time_to_purchase             0
delivery_time                0
shipping_fee                 0
seller_rating                0
seller_response_time         0
seller_location              0
product_rating_variance      0
review_sentiment_score       0
user_engagement_score        0
ad_click_rate                0
time_of_day                  0
day_of_week                  0
season                       0
payment_method               0
coupon_used                  0
product_popularity           0
dtype: int64
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 60000 entries, 0 to 59999
Data columns (total 51 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   user_id                   60000 non-null  int64
 1   product_id                60000 non-null  int64
 2   category                  60000 non-null  object
 3   price                     60000 non-null  float64
 4   rating                    60000 non-null  int64
 5   review_count              60000 non-null  int64
 6   user_age                  60000 non-null  int64
 7   user_gender               60000 non-null  object
 8   user_location             60000 non-null  object
 9   purchase_history          60000 non-null  bool
 10  time_on_page              60000 non-null  float64
 11  add_to_cart_count         60000 non-null  int64
 12  search_keywords           60000 non-null  object
 13  discount_applied          60000 non-null  bool
 14  user_membership           60000 non-null  object
 15  user_browser              60000 non-null  object
 16  user_device               60000 non-null  object
 17  purchase_time             60000 non-null  object
 18  session_duration          60000 non-null  float64
 19  clicks_on_ads             60000 non-null  int64
 20  page_views                60000 non-null  int64
 21  referral_source           60000 non-null  object
 22  wishlist_additions        60000 non-null  int64
 23  cart_abandonment_rate     60000 non-null  float64
 24  average_spent             60000 non-null  float64
 25  user_income               60000 non-null  float64
 26  user_education            60000 non-null  object
 27  user_marital_status       60000 non-null  object
 28  product_availability      60000 non-null  object
 29  stock_status              60000 non-null  object
 30  product_return_rate       60000 non-null  float64
 31  product_color             60000 non-null  object
 32  product_size              60000 non-null  object
 33  is_top_seller             60000 non-null  bool
 34  discount_percentage       60000 non-null  float64
 35  time_to_purchase          60000 non-null  float64
 36  delivery_time             60000 non-null  float64
 37  shipping_fee              60000 non-null  float64
 38  seller_rating             60000 non-null  int64
 39  seller_response_time      60000 non-null  float64
 40  seller_location           60000 non-null  object
 41  product_rating_variance   60000 non-null  float64
 42  review_sentiment_score    60000 non-null  float64
 43  user_engagement_score     60000 non-null  float64
 44  ad_click_rate             60000 non-null  float64
```

```
 45  time_of_day              60000 non-null  object
 46  day_of_week              60000 non-null  object
 47  season                   60000 non-null  object
 48  payment_method           60000 non-null  object
 49  coupon_used              60000 non-null  bool
 50  product_popularity       60000 non-null  float64
dtypes: bool(4), float64(17), int64(10), object(20)
memory usage: 21.7+ MB
```

4) Checking DataTypes

```
df.dtypes
```

```
user_id                        int64
product_id                     int64
category                      object
price                        float64
rating                         int64
review_count                   int64
user_age                       int64
user_gender                   object
user_location                 object
purchase_history                bool
time_on_page                 float64
add_to_cart_count              int64
search_keywords               object
discount_applied                bool
user_membership               object
user_browser                  object
user_device                   object
purchase_time                 object
session_duration             float64
clicks_on_ads                  int64
page_views                     int64
referral_source               object
wishlist_additions             int64
cart_abandonment_rate        float64
average_spent                float64
user_income                  float64
user_education                object
user_marital_status           object
product_availability          object
stock_status                  object
product_return_rate          float64
product_color                 object
product_size                  object
is_top_seller                   bool
discount_percentage          float64
time_to_purchase             float64
```

```
delivery_time                float64
shipping_fee                 float64
seller_rating                  int64
seller_response_time         float64
seller_location               object
product_rating_variance      float64
review_sentiment_score       float64
user_engagement_score        float64
ad_click_rate                float64
time_of_day                   object
day_of_week                   object
season                        object
payment_method                object
coupon_used                     bool
product_popularity           float64
dtype: object
```

5) Checking Target Column - 'purchase_history'

```
df['purchase_history'].value_counts(normalize=True)*100
```

```
purchase_history
True     50.146667
False    49.853333
Name: proportion, dtype: float64
```

6) Feature Engineering of 'purchase_time'

```
print("purchase_time")
print("number of unique values :",df['purchase_time'].nunique())
print("max :",df['purchase_time'].max())
print("min :",df['purchase_time'].min())
print("datatype :",df['purchase_time'].dtype)
```

```
purchase_time
number of unique values : 8749
max : 2024-12-31 00:00:00
min : 2024-01-01 00:00:00
datatype : object
```

6.a) Converting To datetime datatype

```
df['purchase_time']=pd.to_datetime(df['purchase_time'])
print("datatype :",df['purchase_time'].dtype)
```

```
datatype : datetime64[ns]
```

6.b) Extracting values from date time

```
print("max year = ",df['purchase_time'].dt.year.max())
print("min year = ",df['purchase_time'].dt.year.min())
print()
print("max month = ",df['purchase_time'].dt.month.max())
print("min month = ",df['purchase_time'].dt.month.min())
# df['day'] = df['purchase_time'].dt.day
# df['month'] = df['purchase_time'].dt.month
# df['year'] = df['purchase_time'].dt.year
# df['hour'] = df['purchase_time'].dt.hour
# df['minute'] = df['purchase_time'].dt.minute
```

```
max year =  2024
min year =  2024

max month =  12
min month =  1
```

```
df['purchase_time_month']=df['purchase_time'].dt.month
df['purchase_time_month'].value_counts()
```

```
purchase_time_month
5     5174
12    5163
10    5152
3     5105
1     5044
7     5010
8     4985
4     4982
6     4920
11    4884
2     4801
9     4780
Name: count, dtype: int64
```

```
'time_of_day'
print("time_of_day")
print("number of unique values :",df['time_of_day'].nunique())
print("value counts are :\n",df['time_of_day'].value_counts(),sep='')
print("datatype :",df['time_of_day'].dtype)
```

```
time_of_day
number of unique values : 4
value counts are :
time_of_day
Night        15095
Evening      15025
Morning      14968
Afternoon    14912
```

```
Name: count, dtype: int64
datatype : object
```

```
df.drop(columns=['purchase_time'],axis=1,inplace=True)
```

```
basic_info=pd.DataFrame()
basic_info['feature_name']=df.columns
basic_info['missing_values']=df.isnull().sum().values
basic_info['data_types']=df.dtypes.values
basic_info['number_of_uniquevalues']=df.nunique().values
basic_info
```

```
              feature_name  missing_values data_types
number_of_uniquevalues
0                  user_id               0      int64
45154
1               product_id               0      int64
4999
2                 category               0     object
5
3                    price               0    float64
45014
4                   rating               0      int64
5
5             review_count               0      int64
500
6                 user_age               0      int64
52
7              user_gender               0     object
3
8            user_location               0     object
3
9         purchase_history               0       bool
2
10            time_on_page               0    float64
2951
11       add_to_cart_count               0      int64
10
12         search_keywords               0     object
5
13        discount_applied               0       bool
2
14         user_membership               0     object
4
15            user_browser               0     object
4
16             user_device               0     object
3
17        session_duration               0    float64
55235
```

```
18             clicks_on_ads                 0      int64
20
19                page_views                 0      int64
99
20           referral_source                 0     object
4
21        wishlist_additions                 0      int64
20
22     cart_abandonment_rate                 0    float64
101
23             average_spent                 0    float64
56634
24               user_income                 0    float64
59912
25            user_education                 0     object
4
26       user_marital_status                 0     object
4
27      product_availability                 0     object
3
28              stock_status                 0     object
3
29       product_return_rate                 0    float64
101
30             product_color                 0     object
5
31              product_size                 0     object
3
32             is_top_seller                 0       bool
2
33       discount_percentage                 0    float64
5001
34          time_to_purchase                 0    float64
25810
35             delivery_time                 0    float64
1401
36              shipping_fee                 0    float64
5001
37             seller_rating                 0      int64
5
38      seller_response_time                 0    float64
7100
39           seller_location                 0     object
3
40  product_rating_variance                  0    float64
201
41    review_sentiment_score                 0    float64
201
42     user_engagement_score                 0    float64
```

```
101
43              ad_click_rate                   0    float64
101
44                time_of_day                   0     object
4
45                day_of_week                   0     object
7
46                     season                   0     object
4
47             payment_method                   0     object
4
48                coupon_used                   0       bool
2
49         product_popularity                   0    float64
101
50        purchase_time_month                   0      int32
12
```

```
y=df['purchase_history']
x=df.drop(columns=['purchase_history'],axis=1)
```

```
categorical=[]
numerical=[]
descrete=[]
x.drop(columns=['user_id', 'product_id'],axis=1,inplace=True)
for i in x.columns:
    if x[i].nunique()>25 and i not in ['user_id','product_id']:
        numerical.append(i)
    elif x[i].nunique()<25 and x[i].dtype=='object':
        categorical.append(i)
    else:
        descrete.append(i)
```

```
print(numerical)
print()
print("total numerical features are =",len(numerical))
```

```
['price', 'review_count', 'user_age', 'time_on_page',
'session_duration', 'page_views', 'cart_abandonment_rate',
'average_spent', 'user_income', 'product_return_rate',
'discount_percentage', 'time_to_purchase', 'delivery_time',
'shipping_fee', 'seller_response_time', 'product_rating_variance',
'review_sentiment_score', 'user_engagement_score', 'ad_click_rate',
'product_popularity']
```

```
total numerical features are = 20
```

```
print(categorical)
print()
print("total categorical features are =",len(categorical))
```

```
['category', 'user_gender', 'user_location', 'search_keywords',
'user_membership', 'user_browser', 'user_device', 'referral_source',
'user_education', 'user_marital_status', 'product_availability',
'stock_status', 'product_color', 'product_size', 'seller_location',
'time_of_day', 'day_of_week', 'season', 'payment_method']

total categorical features are = 19
```

```
print(descrete)
print()
print("total descrete features are =",len(descrete))
```

```
['rating', 'add_to_cart_count', 'discount_applied', 'clicks_on_ads',
'wishlist_additions', 'is_top_seller', 'seller_rating', 'coupon_used',
'purchase_time_month']

total descrete features are = 9
```

II] Feature Selection & Multi-Colinearity

1) Multi-Colinearity in Numerical Features

1.a) Corellation Matrix

```
corrmatrix=df[numerical].corr()
corrmatrix
```

```
                            price  review_count  user_age
time_on_page  \
price                    1.000000      0.000553 -0.005285     -
0.002197
review_count             0.000553      1.000000 -0.000759
0.002428
user_age                -0.005285     -0.000759  1.000000
0.004028
time_on_page            -0.002197      0.002428  0.004028
1.000000
session_duration        -0.000339     -0.000119  0.000812     -
0.001737
page_views               0.002720     -0.001148 -0.002092
0.002644
cart_abandonment_rate    0.004704     -0.004129 -0.001691
0.002740
average_spent           -0.002186      0.001596 -0.002669     -
0.000698
user_income             -0.002944     -0.001625  0.001072     -
0.001668
product_return_rate      0.002338      0.004427  0.002673     -
```

```
0.004414
discount_percentage      0.006372      0.000887 -0.003565
0.002988
time_to_purchase        -0.004822      0.001550 -0.004478
0.003364
delivery_time            0.000883      0.001659 -0.000821     -
0.002803
shipping_fee             0.001692      0.003383 -0.000638
0.009305
seller_response_time     0.000019     -0.000159 -0.003831     -
0.000462
product_rating_variance  0.011372     -0.004260  0.002916
0.003512
review_sentiment_score  -0.009115      0.002156 -0.003348
0.000330
user_engagement_score   -0.000325      0.006545 -0.001121     -
0.000154
ad_click_rate            0.004739      0.005577  0.001147     -
0.000799
product_popularity      -0.001021      0.004671  0.002571     -
0.004101

                         session_duration  page_views
cart_abandonment_rate  \
price                           -0.000339    0.002720
0.004704
review_count                    -0.000119   -0.001148               -
0.004129
user_age                         0.000812   -0.002092               -
0.001691
time_on_page                    -0.001737    0.002644
0.002740
session_duration                 1.000000   -0.005290               -
0.003657
page_views                      -0.005290    1.000000
0.005704
cart_abandonment_rate           -0.003657    0.005704
1.000000
average_spent                    0.003399    0.005138               -
0.010053
user_income                      0.001579    0.002168
0.003268
product_return_rate              0.003760    0.000223
0.001191
discount_percentage              0.002069    0.003101               -
0.000313
time_to_purchase                -0.000845   -0.003670
0.004395
delivery_time                    0.008153    0.002657               -
```

```
0.005104
shipping_fee                       -0.000676    0.002222
0.001536
seller_response_time                0.007093    0.000090
0.003947
product_rating_variance            -0.000166    0.000418
0.007186
review_sentiment_score              0.001741   -0.002711              -
0.003193
user_engagement_score              -0.000691   -0.003882
0.003944
ad_click_rate                       0.002220    0.003758
0.007245
product_popularity                 -0.003204   -0.005096
0.005538

                         average_spent  user_income
product_return_rate  \
price                        -0.002186    -0.002944
0.002338
review_count                  0.001596    -0.001625
0.004427
user_age                     -0.002669     0.001072
0.002673
time_on_page                 -0.000698    -0.001668              -
0.004414
session_duration              0.003399     0.001579
0.003760
page_views                    0.005138     0.002168
0.000223
cart_abandonment_rate        -0.010053     0.003268
0.001191
average_spent                 1.000000    -0.002158
0.003798
user_income                  -0.002158     1.000000              -
0.004187
product_return_rate           0.003798    -0.004187
1.000000
discount_percentage           0.008415     0.000450              -
0.001619
time_to_purchase             -0.004868     0.000561
0.002045
delivery_time                 0.009656    -0.000427
0.001719
shipping_fee                  0.003372     0.003680
0.004030
seller_response_time          0.000699    -0.003094
0.001394
product_rating_variance      -0.002859     0.001301
```

```
0.003918
review_sentiment_score         0.006323     0.005753
0.003248
user_engagement_score         -0.000966    -0.000097
0.001266
ad_click_rate                 -0.005269    -0.004528
0.001472
product_popularity             0.004336    -0.002315              -
0.002570

                         discount_percentage  time_to_purchase
delivery_time  \
price                               0.006372         -0.004822
0.000883
review_count                        0.000887          0.001550
0.001659
user_age                           -0.003565         -0.004478       -
0.000821
time_on_page                        0.002988          0.003364       -
0.002803
session_duration                    0.002069         -0.000845
0.008153
page_views                          0.003101         -0.003670
0.002657
cart_abandonment_rate              -0.000313          0.004395       -
0.005104
average_spent                       0.008415         -0.004868
0.009656
user_income                         0.000450          0.000561       -
0.000427
product_return_rate                -0.001619          0.002045
0.001719
discount_percentage                 1.000000         -0.001379       -
0.003327
time_to_purchase                   -0.001379          1.000000
0.003150
delivery_time                      -0.003327          0.003150
1.000000
shipping_fee                       -0.005994         -0.005498       -
0.005238
seller_response_time               -0.007436         -0.001043       -
0.003248
product_rating_variance             0.008739          0.004788
0.004401
review_sentiment_score             -0.008945         -0.003292       -
0.000429
user_engagement_score              -0.000484         -0.003193       -
0.005483
ad_click_rate                      -0.002277          0.003012       -
```

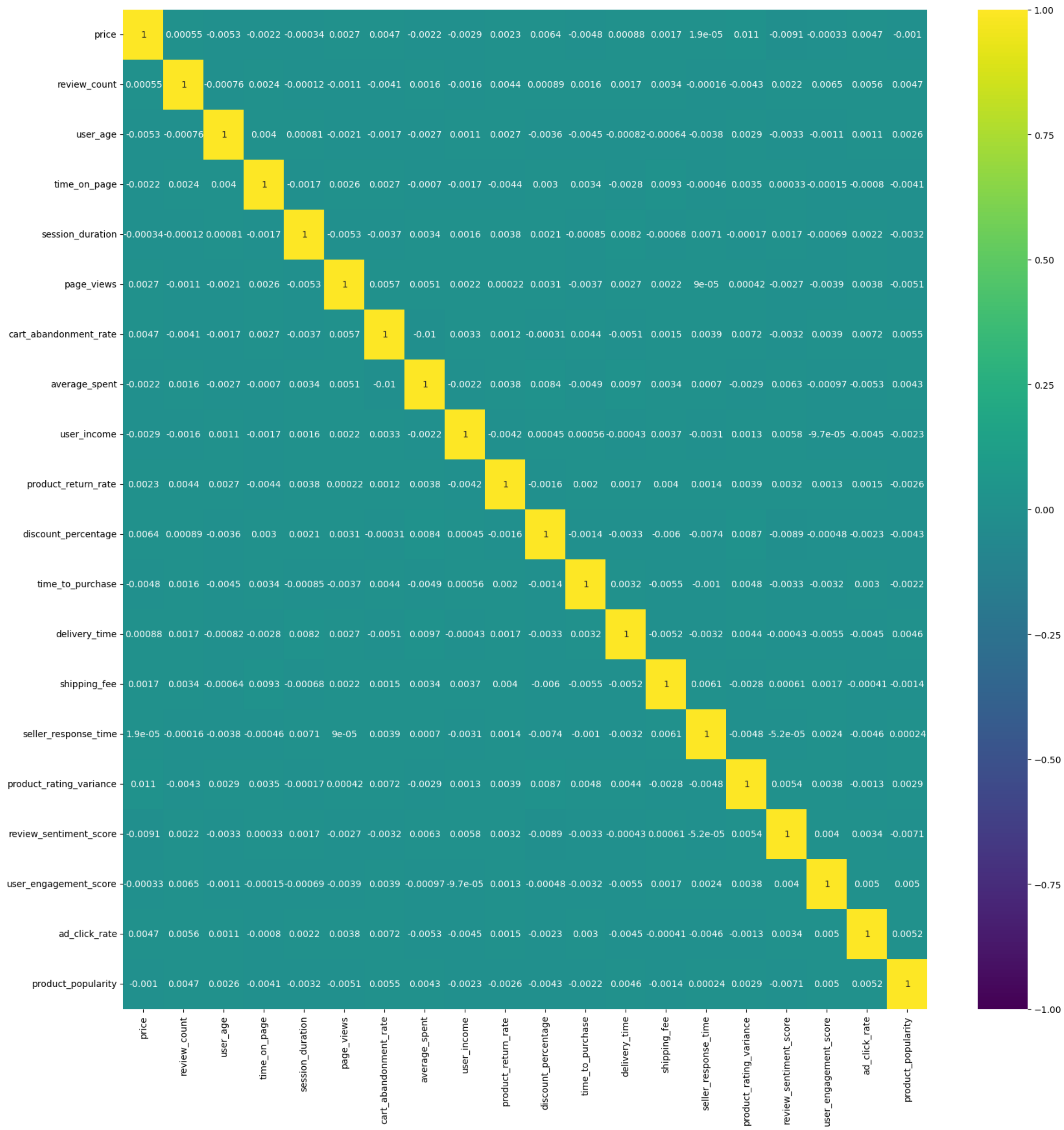
```
0.004506
product_popularity                 -0.004310        -0.002184
0.004569

                         shipping_fee  seller_response_time  \
price                        0.001692              0.000019
review_count                 0.003383             -0.000159
user_age                    -0.000638             -0.003831
time_on_page                 0.009305             -0.000462
session_duration            -0.000676              0.007093
page_views                   0.002222              0.000090
cart_abandonment_rate        0.001536              0.003947
average_spent                0.003372              0.000699
user_income                  0.003680             -0.003094
product_return_rate          0.004030              0.001394
discount_percentage         -0.005994             -0.007436
time_to_purchase            -0.005498             -0.001043
delivery_time               -0.005238             -0.003248
shipping_fee                 1.000000              0.006113
seller_response_time         0.006113              1.000000
product_rating_variance     -0.002822             -0.004816
review_sentiment_score       0.000610             -0.000052
user_engagement_score        0.001722              0.002366
ad_click_rate               -0.000414             -0.004610
product_popularity          -0.001354              0.000244

                         product_rating_variance
review_sentiment_score  \
price                                   0.011372                -
0.009115
review_count                           -0.004260
0.002156
user_age                                0.002916                -
0.003348
time_on_page                            0.003512
0.000330
session_duration                       -0.000166
0.001741
page_views                              0.000418                -
0.002711
cart_abandonment_rate                   0.007186                -
0.003193
average_spent                          -0.002859
0.006323
user_income                             0.001301
0.005753
product_return_rate                     0.003918
0.003248
discount_percentage                     0.008739                -
```

```
0.008945
time_to_purchase                                0.004788                -
0.003292
delivery_time                                   0.004401                -
0.000429
shipping_fee                                   -0.002822
0.000610
seller_response_time                           -0.004816                -
0.000052
product_rating_variance                         1.000000
0.005371
review_sentiment_score                          0.005371
1.000000
user_engagement_score                           0.003763
0.004033
ad_click_rate                                  -0.001256
0.003371
product_popularity                              0.002944                -
0.007109

                         user_engagement_score  ad_click_rate  \
price                                -0.000325       0.004739
review_count                          0.006545       0.005577
user_age                             -0.001121       0.001147
time_on_page                         -0.000154      -0.000799
session_duration                     -0.000691       0.002220
page_views                           -0.003882       0.003758
cart_abandonment_rate                 0.003944       0.007245
average_spent                        -0.000966      -0.005269
user_income                          -0.000097      -0.004528
product_return_rate                   0.001266       0.001472
discount_percentage                  -0.000484      -0.002277
time_to_purchase                     -0.003193       0.003012
delivery_time                        -0.005483      -0.004506
shipping_fee                          0.001722      -0.000414
seller_response_time                  0.002366      -0.004610
product_rating_variance               0.003763      -0.001256
review_sentiment_score                0.004033       0.003371
user_engagement_score                 1.000000       0.004989
ad_click_rate                         0.004989       1.000000
product_popularity                    0.004979       0.005217

                         product_popularity
price                             -0.001021
review_count                       0.004671
user_age                           0.002571
time_on_page                      -0.004101
session_duration                  -0.003204
page_views                        -0.005096
```

```
cart_abandonment_rate          0.005538
average_spent                  0.004336
user_income                   -0.002315
product_return_rate           -0.002570
discount_percentage           -0.004310
time_to_purchase              -0.002184
delivery_time                  0.004569
shipping_fee                  -0.001354
seller_response_time           0.000244
product_rating_variance        0.002944
review_sentiment_score        -0.007109
user_engagement_score          0.004979
ad_click_rate                  0.005217
product_popularity             1.000000
```

1.b) Heatmap

```
plt.figure(figsize=(20,20))
sns.heatmap(corrmatrix,cmap='viridis',vmin=-1,vmax=1,annot=True)
plt.show()
```



2) Feature Selection using Stats

2.a) ANNOVA Testing

```
import scipy.stats as stats
from scipy.stats import chi2_contingency

print("ANNOVA on Numerical vs Target Column")
print()
annovanumerical={}
```

```
numerical_to_keep=[]
numerical_can_be_removed=[]
for i in numerical:
    group1=df[df['purchase_history']==True][i]
    group2=df[df['purchase_history']==False][i]
    f_stat, p_value = stats.f_oneway(group1,group2)
    annovanumerical[i]=p_value
    if p_value<0.05:
        numerical_to_keep.append(i)
    elif p_value>0.05:
        numerical_can_be_removed.append(i)
print("column is important for prediction \n",numerical_to_keep)
print()
print("column is not important, can be removed \
n",numerical_can_be_removed)
print()
print(annovanumerical)

ANNOVA on Numerical vs Target Column

column is important for prediction
 ['average_spent', 'user_income', 'time_to_purchase',
'seller_response_time', 'product_popularity']

column is not important, can be removed
 ['price', 'review_count', 'user_age', 'time_on_page',
'session_duration', 'page_views', 'cart_abandonment_rate',
'product_return_rate', 'discount_percentage', 'delivery_time',
'shipping_fee', 'product_rating_variance', 'review_sentiment_score',
'user_engagement_score', 'ad_click_rate']

{'price': 0.8899465739986773, 'review_count': 0.5770768106977051,
'user_age': 0.7854081232478979, 'time_on_page': 0.5085345397926913,
'session_duration': 0.7497831381683305, 'page_views':
0.1449642634414101, 'cart_abandonment_rate': 0.07615900506164756,
'average_spent': 0.03134234160871035, 'user_income':
0.015616516056511025, 'product_return_rate': 0.5786243875252175,
'discount_percentage': 0.5028842277197112, 'time_to_purchase':
0.0009285761046717416, 'delivery_time': 0.715906387899155,
'shipping_fee': 0.12814595886083166, 'seller_response_time':
0.011110530302949694, 'product_rating_variance': 0.9687752069566598,
'review_sentiment_score': 0.5063007444466396, 'user_engagement_score':
0.43557617269905147, 'ad_click_rate': 0.478543850161665,
'product_popularity': 0.019631567972866765}
```

2.b) ChiSquare Testing

```
print("Chi Square Test on Categorical Data")
print()
```

```
chisquarecategorical={}
categorical_to_keep=[]
categorical_to_remove=[]
for i in categorical:
    contingency_table=pd.crosstab(df[i],df['purchase_history'])
    chi2, p_value, dof, expected =chi2_contingency(contingency_table)
    chisquarecategorical[i]=p_value
    if p_value<0.05:
        categorical_to_keep.append(i)

    elif p_value>0.05:
        categorical_to_remove.append(i)
print("column is important for prediction \n",categorical_to_keep)
print()
print("column is not important, can be removed \
n",categorical_to_remove)
print()
print(chisquarecategorical)
```

```
Chi Square Test on Categorical Data

column is important for prediction
 []

column is not important, can be removed
 ['category', 'user_gender', 'user_location', 'search_keywords',
'user_membership', 'user_browser', 'user_device', 'referral_source',
'user_education', 'user_marital_status', 'product_availability',
'stock_status', 'product_color', 'product_size', 'seller_location',
'time_of_day', 'day_of_week', 'season', 'payment_method']

{'category': 0.4953927713533365, 'user_gender': 0.5103166694448208,
'user_location': 0.6412943409626277, 'search_keywords':
0.11971165254357352, 'user_membership': 0.14738046643188624,
'user_browser': 0.8750665879204095, 'user_device': 0.6032563538695197,
'referral_source': 0.5704558268399278, 'user_education':
0.05188126377538571, 'user_marital_status': 0.5860140869575095,
'product_availability': 0.9171695966427554, 'stock_status':
0.5717932793932304, 'product_color': 0.5774925654273336,
'product_size': 0.736638141818841, 'seller_location':
0.6536341963997004, 'time_of_day': 0.6395547722074488, 'day_of_week':
0.42747766253353114, 'season': 0.9392576700028451, 'payment_method':
0.3067340087346109}
```

```
print("Chi Square Test on descrete Data")
print()

chisquaredescrete={}
descrete_to_keep=[]
descrete_to_remove=[]
```

```
for i in descrete:
    contingency_table=pd.crosstab(df[i],df['purchase_history'])
    chi2, p_value, dof, expected =chi2_contingency(contingency_table)
    chisquaredescrete[i]=p_value
    if p_value<0.05:
        descrete_to_keep.append(i)
    elif p_value>0.05:
        descrete_to_remove.append(i)
print("column is important for prediction \n",descrete_to_keep)
print()
print("column is not important, can be removed \n",descrete_to_remove)
print()
print(chisquaredescrete)
```

```
Chi Square Test on descrete Data

column is important for prediction
 []

column is not important, can be removed
 ['rating', 'add_to_cart_count', 'discount_applied', 'clicks_on_ads',
'wishlist_additions', 'is_top_seller', 'seller_rating', 'coupon_used',
'purchase_time_month']

{'rating': 0.5379206436964865, 'add_to_cart_count':
0.7324985410378212, 'discount_applied': 0.09075982734615025,
'clicks_on_ads': 0.6597429536690044, 'wishlist_additions':
0.372469260506758, 'is_top_seller': 0.5561143004611644,
'seller_rating': 0.550055705756747, 'coupon_used': 0.7631196614510456,
'purchase_time_month': 0.12279738470928081}
```

5) Observations:

total duplicate rows in dataset is = 0

total missing values rows in dataset is = 0

target column "purchase_history" is perfectly balanced

multicolinearity is not there in numerical data

boxplot - No Outliers Present

ANNOVA on Numerical Dependent Variable vs Target Categorical Column: 5 column are important for prediction 'average_spent' 'user_income', 'time_to_purchase', 'seller_response_time', 'product_popularity We can try building a model using these columns only'##### Chi Square Test on Categorical Variable vs Target Categorical Column0 5 column are important for prediction

II] Encoding, Train-Test-Split, & Standardization

1) Label Encoding

```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
for i in categorical:
    x[i]=le.fit_transform(x[i])

y=pd.DataFrame(np.where(y==True,1,0))

for i in descrete:
    if x[i].dtype=='bool':
        x[i] = x[i].astype(int)

x.head()
```

```
   category   price  rating  review_count  user_age  user_gender  \
0         0  842.23       2           155        24            2
1         0  253.76       3           331        43            2
2         0  483.65       2           236        64            0
3         4  459.37       2           227        34            0
4         4  150.11       2           214        51            0

   user_location  time_on_page  add_to_cart_count
search_keywords  ...  \
0              2         13.86                  6
4  ...
1              1         13.03                  3
1  ...
2              0          3.75                  7
4  ...
3              2          6.01                  0
0  ...
4              2          6.89                  9
2  ...

   review_sentiment_score  user_engagement_score  ad_click_rate
time_of_day  \
0                   -0.28                   0.68           0.04
3
1                    0.28                   0.11           0.89
2
2                    0.23                   0.35           0.99
1
3                    0.93                   0.73           0.16
0
4                    0.11                   0.26           0.17
3
```

```
   day_of_week  season  payment_method  coupon_used  
product_popularity  \
0            4       2               2            0
0.54
1            2       2               2            0
0.77
2            5       0               2            0
0.14
3            5       1               1            0
0.18
4            6       1               3            0
0.66

   purchase_time_month
0                    8
1                    7
2                    9
3                    9
4                    9

[5 rows x 48 columns]
```

2) Train Test Split

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,rando
m_state=42)
x_train2,x_test2,y_train2,y_test2=train_test_split(x,y,test_size=0.3,r
andom_state=42)
for i in [x_train,x_test,y_train,y_test]:
    print(i.shape)

(42000, 48)
(18000, 48)
(42000, 1)
(18000, 1)
```

3) Standard Scaling

```
x_train.shape

(42000, 48)

x_test.shape

(18000, 48)

from sklearn.preprocessing import StandardScaler
ss=StandardScaler()
for i in numerical:
```

```
    x_train[i]=ss.fit_transform(x_train[[i]])
    x_test[i]=ss.transform(x_test[[i]])
```

```
x_train.shape
```

```
(42000, 48)
```

```
x_test.shape
```

```
(18000, 48)
```

```
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report,precision_score,recall_score,f1_score


def evaluation_matrix(a,b):
    print(f"accuracy = {accuracy_score(a,b)}")
    print(f"precision = {precision_score(a,b)}")
    print(f"recall = {recall_score(a,b)}")
    print(f"f1 score = {f1_score(a,b)}")
    print("Confusion Matrix:\n", confusion_matrix(a,b))
```

4) Model Evaluation Metrics Function

```
model_metrics=pd.DataFrame(columns=['Model','Accuracy','Precision','Re
call','F1 Score'])
model_metrics
```

```
Empty DataFrame
Columns: [Model, Accuracy, Precision, Recall, F1 Score]
Index: []
```

```
def model_evaluation_metrics(model_name,ytrain,ypredict):
    accuracy=accuracy_score(ytrain,ypredict)
    precision=precision_score(ytrain,ypredict)
    recall=recall_score(ytrain,ypredict)
    f1score=f1_score(ytrain,ypredict)

metrics=pd.DataFrame([[model_name,accuracy,precision,recall,f1score]],
columns=['Model','Accuracy','Precision','Recall','F1 Score'])
    global model_metrics
    model_metrics=pd.concat([model_metrics,metrics],ignore_index=True)
```

III] Model Building

1) Logistic Regression

```
from sklearn.linear_model import LogisticRegression
lr=LogisticRegression()
```

```
lr.fit(x_train[numerical_to_keep],y_train)
y_train_predict=lr.predict(x_train[numerical_to_keep])
y_test_predict=lr.predict(x_test[numerical_to_keep])
```

```
evaluation_matrix(y_train,y_train_predict)
```

```
accuracy = 0.5131190476190476
precision = 0.5135257191889084
recall = 0.5402413530976815
f1 score = 0.5265448820356092
Confusion Matrix:
 [[10180 10772]
 [ 9677 11371]]
```

```
evaluation_matrix(y_test,y_test_predict)
```

```
accuracy = 0.5043888888888889
precision = 0.5063076433796247
recall = 0.5283185840707965
f1 score = 0.5170789801331673
Confusion Matrix:
 [[4303 4657]
 [4264 4776]]
```

```
model_evaluation_metrics("Logistic Regression - 
Train",y_train,y_train_predict)
model_evaluation_metrics("Logistic Regression - 
Test",y_test,y_test_predict)
```

```
model_metrics
```

```
                         Model  Accuracy  Precision    Recall  F1 
Score
0  Logistic Regression - Train  0.513119   0.513526  0.540241  
0.526545
1   Logistic Regression - Test  0.504389   0.506308  0.528319  
0.517079
```

2) Logistic Regression With Standard Scaling on Categorical Columns

```
from sklearn.preprocessing import StandardScaler
ss=StandardScaler()
for i in x_train2.columns:
    x_train2[i]=ss.fit_transform(x_train[[i]])
    x_test2[i]=ss.transform(x_test[[i]])

lr2=LogisticRegression()
lr2.fit(x_train2[numerical_to_keep],y_train2)
```

```
y_train2_predict=lr.predict(x_train2[numerical_to_keep])
y_test2_predict=lr.predict(x_test2[numerical_to_keep])

evaluation_matrix(y_train2,y_train2_predict)

accuracy = 0.5131190476190476
precision = 0.5135257191889084
recall = 0.5402413530976815
f1 score = 0.5265448820356092
Confusion Matrix:
 [[10180 10772]
 [ 9677 11371]]

evaluation_matrix(y_test2,y_test2_predict)

accuracy = 0.5043888888888889
precision = 0.5063076433796247
recall = 0.5283185840707965
f1 score = 0.5170789801331673
Confusion Matrix:
 [[4303 4657]
 [4264 4776]]

model_evaluation_metrics("Logistic Regression with SS -
Train",y_train2,y_train2_predict)

model_evaluation_metrics("Logistic Regression with SS -
Test",y_test2,y_test2_predict)

model_metrics

                                 Model  Accuracy  Precision    Recall
\
0          Logistic Regression - Train  0.513119   0.513526  0.540241

1           Logistic Regression - Test  0.504389   0.506308  0.528319

2  Logistic Regression with SS - Train  0.513119   0.513526  0.540241

3   Logistic Regression with SS - Test  0.504389   0.506308  0.528319


   F1 Score
0  0.526545
1  0.517079
2  0.526545
3  0.517079
```

3) Stats Model

```
import statsmodels.api as sm
x_train3 = sm.add_constant(x_train)
logistic=sm.Logit(y_train,x_train3)
model=logistic.fit()
print(model.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.692129
         Iterations 3
                           Logit Regression Results

======================================================================
========
Dep. Variable:                      0   No. Observations:
42000
Model:                          Logit   Df Residuals:
41951
Method:                           MLE   Df Model:
48
Date:                Sun, 16 Feb 2025   Pseudo R-squ.:
0.001465
Time:                        21:14:56   Log-Likelihood:
-29069.
converged:                       True   LL-Null:
-29112.
Covariance Type:            nonrobust   LLR p-value:
0.0007366
======================================================================
=====================
                              coef    std err          z      P>|z|
[0.025      0.975]
----------------------------------------------------------------------
---------------------
const                       0.0828      0.075      1.108      0.268
-0.064       0.229
category                   -0.0088      0.007     -1.273      0.203
-0.022       0.005
price                      -0.0039      0.010     -0.401      0.688
-0.023       0.015
rating                     -0.0011      0.007     -0.156      0.876
-0.015       0.012
review_count               -0.0035      0.010     -0.362      0.717
-0.023       0.016
user_age                   -0.0062      0.010     -0.631      0.528
-0.025       0.013
user_gender                 0.0155      0.012      1.297      0.195
-0.008       0.039
user_location              -0.0120      0.012     -1.003      0.316
```

```
-0.036       0.011
time_on_page                   -0.0112      0.010     -1.145      0.252
-0.030       0.008
add_to_cart_count               0.0040      0.003      1.182      0.237
-0.003       0.011
search_keywords                -0.0117      0.007     -1.682      0.093
-0.025       0.002
discount_applied               -0.0385      0.020     -1.971      0.049
-0.077      -0.000
user_membership                 0.0087      0.009      1.002      0.317
-0.008       0.026
user_browser                    0.0043      0.009      0.489      0.625
-0.013       0.021
user_device                    -0.0057      0.012     -0.477      0.633
-0.029       0.018
session_duration                0.0021      0.010      0.215      0.830
-0.017       0.021
clicks_on_ads                  -0.0021      0.002     -1.213      0.225
-0.005       0.001
page_views                      0.0083      0.010      0.849      0.396
-0.011       0.027
referral_source                -0.0060      0.009     -0.684      0.494
-0.023       0.011
wishlist_additions           5.303e-05      0.002      0.031      0.975
-0.003       0.003
cart_abandonment_rate           0.0063      0.010      0.648      0.517
-0.013       0.025
average_spent                  -0.0262      0.010     -2.676      0.007
-0.045      -0.007
user_income                     0.0204      0.010      2.087      0.037
0.001       0.040
user_education                  0.0172      0.009      1.966      0.049
5.15e-05       0.034
user_marital_status             0.0091      0.009      1.046      0.296
-0.008       0.026
product_availability            0.0104      0.012      0.866      0.387
-0.013       0.034
stock_status                    0.0097      0.012      0.809      0.418
-0.014       0.033
product_return_rate            -0.0031      0.010     -0.319      0.750
-0.022       0.016
product_color                   0.0053      0.007      0.769      0.442
-0.008       0.019
product_size                    0.0123      0.012      1.026      0.305
-0.011       0.036
is_top_seller                  -0.0088      0.020     -0.451      0.652
-0.047       0.029
discount_percentage             0.0023      0.010      0.237      0.812
-0.017       0.021
```

```
time_to_purchase              0.0356      0.010      3.637      0.000
0.016       0.055
delivery_time                -0.0104      0.010     -1.065      0.287
-0.030       0.009
shipping_fee                 -0.0218      0.010     -2.227      0.026
-0.041      -0.003
seller_rating                -0.0094      0.007     -1.357      0.175
-0.023       0.004
seller_response_time          0.0255      0.010      2.613      0.009
0.006       0.045
seller_location               0.0078      0.012      0.654      0.513
-0.016       0.031
product_rating_variance       0.0036      0.010      0.367      0.714
-0.016       0.023
review_sentiment_score       -0.0060      0.010     -0.610      0.542
-0.025       0.013
user_engagement_score         0.0109      0.010      1.112      0.266
-0.008       0.030
ad_click_rate                 0.0042      0.010      0.434      0.664
-0.015       0.023
time_of_day                  -0.0130      0.009     -1.481      0.139
-0.030       0.004
day_of_week                  -0.0070      0.005     -1.432      0.152
-0.017       0.003
season                       -0.0014      0.009     -0.162      0.871
-0.019       0.016
payment_method               -0.0151      0.009     -1.726      0.084
-0.032       0.002
coupon_used                   0.0058      0.020      0.298      0.766
-0.032       0.044
product_popularity            0.0271      0.010      2.770      0.006
0.008       0.046
purchase_time_month          -0.0028      0.003     -0.985      0.325
-0.008       0.003
======================================================================
=====================

# Predict probabilities
y_train_prob = model.predict(x_train3)  # Predict on train set
y_test_prob = model.predict(sm.add_constant(x_test))  # Add constant
to test set before prediction

# Convert probabilities to class labels (threshold = 0.5)
y_train_pred = (y_train_prob > 0.5).astype(int)
y_test_pred = (y_test_prob > 0.5).astype(int)

evaluation_matrix(y_train,y_train_pred)

accuracy = 0.5175952380952381
precision = 0.5179787088225888
```

```
recall = 0.5386259977194983
f1 score = 0.5281006172120647
Confusion Matrix:
 [[10402 10550]
 [ 9711 11337]]
```

```
evaluation_matrix(y_test,y_test_pred)
```

```
accuracy = 0.5028888888888889
precision = 0.5049601035152038
recall = 0.5180309734513274
f1 score = 0.5114120345091188
Confusion Matrix:
 [[4369 4591]
 [4357 4683]]
```

```
model_evaluation_metrics("Stats Model (LR) -
Train",y_train,y_train_predict)
model_evaluation_metrics("Stats Model (LR) -
Test",y_test,y_test_predict)
model_metrics
```

```
                                 Model  Accuracy  Precision    Recall
\
0          Logistic Regression - Train  0.513119   0.513526  0.540241

1           Logistic Regression - Test  0.504389   0.506308  0.528319

2  Logistic Regression with SS - Train  0.513119   0.513526  0.540241

3   Logistic Regression with SS - Test  0.504389   0.506308  0.528319

4               Stats Model (LR) - Train  0.513119   0.513526  0.540241

5                Stats Model (LR) - Test  0.504389   0.506308  0.528319


   F1 Score
0  0.526545
1  0.517079
2  0.526545
3  0.517079
4  0.526545
5  0.517079
```

```
# # Get the p-values
# p_values = model.pvalues

# # Define the significance level (e.g., 0.05)
# alpha = 0.05
```

```
# # Get the columns with p-values less than alpha
# significant_columns = p_values[p_values <= alpha].index

# print(significant_columns)
```

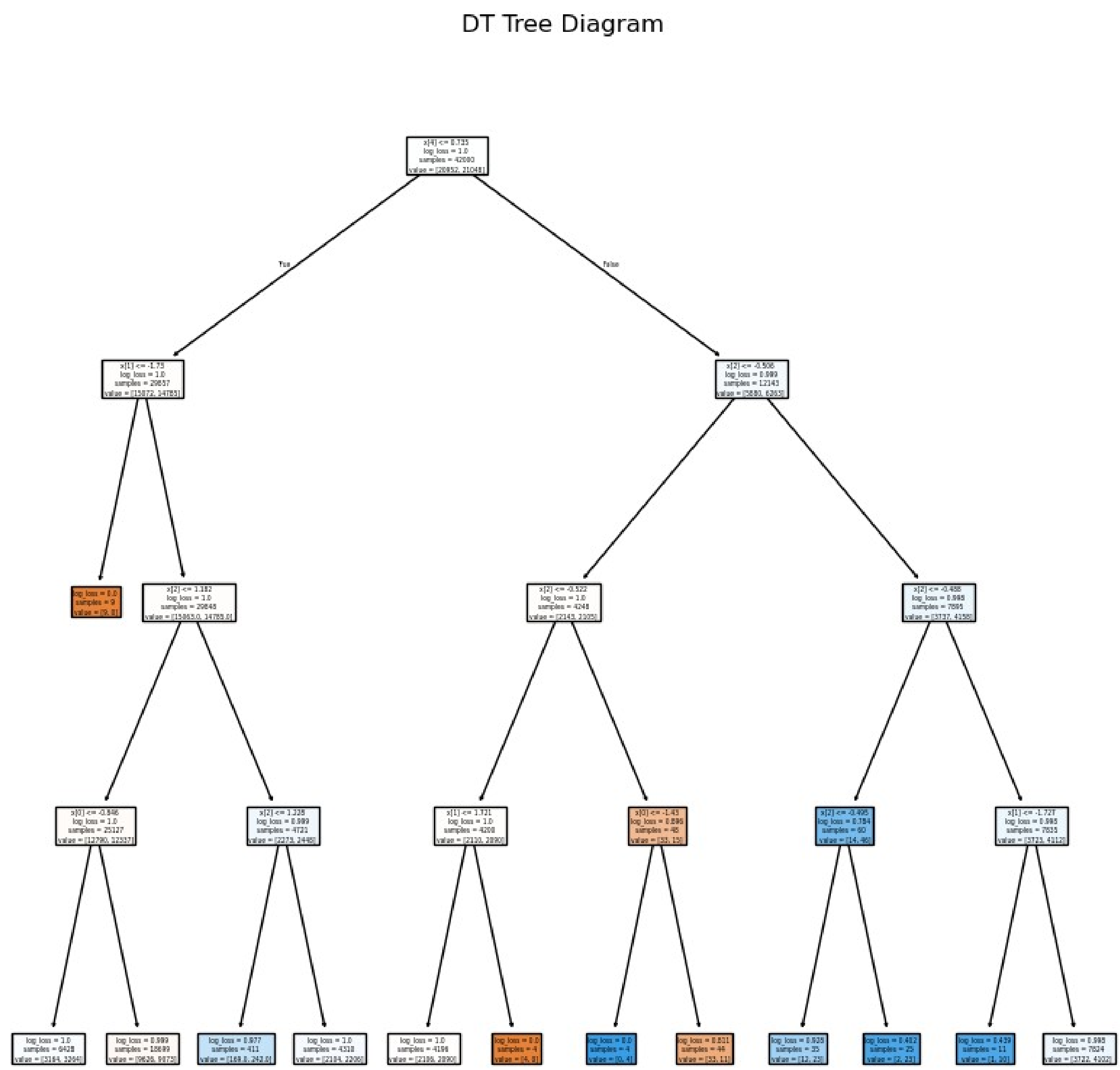
4) Decision Tree

```
from sklearn.tree import DecisionTreeClassifier
dt=DecisionTreeClassifier(criterion='log_loss',max_depth=4)

dt.fit(x_train[numerical_to_keep],y_train)
```

```
DecisionTreeClassifier(criterion='log_loss', max_depth=4)
```

4.a) DT Tree Diagram

```
from sklearn.tree import plot_tree
plt.figure(figsize=(10,10))
plot_tree(dt,filled=True)
plt.title("DT Tree Diagram")
plt.show()
```




```
y_train_predict=lr.predict(x_train[numerical_to_keep])
evaluation_matrix(y_train,y_train_predict)
```

```
accuracy = 0.5131190476190476
precision = 0.5135257191889084
recall = 0.5402413530976815
f1 score = 0.5265448820356092
Confusion Matrix:
 [[10180 10772]
 [ 9677 11371]]
```

```
y_test_predict=lr.predict(x_test[numerical_to_keep])
evaluation_matrix(y_test,y_test_predict)
```

```
accuracy = 0.5043888888888889
precision = 0.5063076433796247
recall = 0.5283185840707965
f1 score = 0.5170789801331673
Confusion Matrix:
 [[4303 4657]
 [4264 4776]]
```

```
model_evaluation_metrics("DecisionTree -
Train",y_train,y_train_predict)
model_evaluation_metrics("DecisionTree - Test",y_test,y_test_predict)
model_metrics
```

```
                                 Model  Accuracy  Precision    Recall  \
0           Logistic Regression - Train  0.513119   0.513526  0.540241

1            Logistic Regression - Test  0.504389   0.506308  0.528319

2  Logistic Regression with SS - Train  0.513119   0.513526  0.540241

3   Logistic Regression with SS - Test  0.504389   0.506308  0.528319

4              Stats Model (LR) - Train  0.513119   0.513526  0.540241

5               Stats Model (LR) - Test  0.504389   0.506308  0.528319

6                  DecisionTree - Train  0.513119   0.513526  0.540241

7                   DecisionTree - Test  0.504389   0.506308  0.528319


   F1 Score
0  0.526545
1  0.517079
2  0.526545
3  0.517079
4  0.526545
5  0.517079
6  0.526545
7  0.517079
```

5) Random Forest

```
from sklearn.ensemble import RandomForestClassifier

i=40
rfc=RandomForestClassifier(n_estimators=i,max_depth=4,random_state=42)
```

```
rfc.fit(x_train,y_train)
y_train_predict=rfc.predict(x_train)
y_test_predict=rfc.predict(x_test)
print()
print(f"n_estimator = {i}")
print(f"train_accuracy={accuracy_score(y_train,y_train_predict)},
test_accuracy={accuracy_score(y_test,y_test_predict)}")
```

```
n_estimator = 40
train_accuracy=0.5601666666666667, test_accuracy=0.5037777777777778
```

```
model_evaluation_metrics("Random Forest -
Train",y_train,y_train_predict)
model_evaluation_metrics("Random Forest - Test",y_test,y_test_predict)
model_metrics
```

```
                                  Model  Accuracy  Precision    Recall  \
0           Logistic Regression - Train  0.513119   0.513526  0.540241

1            Logistic Regression - Test  0.504389   0.506308  0.528319

2  Logistic Regression with SS - Train  0.513119   0.513526  0.540241

3   Logistic Regression with SS - Test  0.504389   0.506308  0.528319

4                Stats Model (LR) - Train  0.513119   0.513526  0.540241

5                 Stats Model (LR) - Test  0.504389   0.506308  0.528319

6                    DecisionTree - Train  0.513119   0.513526  0.540241

7                     DecisionTree - Test  0.504389   0.506308  0.528319

8                   Random Forest - Train  0.560167   0.554401  0.623385

9                    Random Forest - Test  0.503778   0.505365  0.562721


   F1 Score
0  0.526545
1  0.517079
2  0.526545
3  0.517079
4  0.526545
5  0.517079
6  0.526545
7  0.517079
8  0.586872
9  0.532503
```

5.a) Feature Importance

```
rfc.feature_importances_
```

```
array([0.00514383, 0.0219449 , 0.00305131, 0.03706279, 0.02525855,
       0.01459486, 0.00212649, 0.03665709, 0.0065069 , 0.0062489 ,
       0.00159372, 0.00837013, 0.00221248, 0.00408166, 0.03408216,
       0.01703656, 0.0219626 , 0.00187657, 0.03152289, 0.02445949,
       0.06159432, 0.05973389, 0.00633194, 0.00126244, 0.00744943,
       0.00182547, 0.01785313, 0.01209383, 0.00579391, 0.00582219,
       0.0477129 , 0.09691251, 0.03756426, 0.03618254, 0.00475532,
       0.05316106, 0.00230782, 0.04540263, 0.03572277, 0.04502118,
       0.02894014, 0.00756018, 0.01414072, 0.00401029, 0.00344366,
       0.00396357, 0.03187965, 0.0157644 ])
```

```
feature_importance=rfc.feature_importances_
features=x_train.columns
important_feature=pd.DataFrame({'Feature':features,'Importance':featur
e_importance})
important_feature
```

	Feature	Importance
0	category	0.005144
1	price	0.021945
2	rating	0.003051
3	review_count	0.037063
4	user_age	0.025259
5	user_gender	0.014595
6	user_location	0.002126
7	time_on_page	0.036657
8	add_to_cart_count	0.006507
9	search_keywords	0.006249
10	discount_applied	0.001594
11	user_membership	0.008370
12	user_browser	0.002212
13	user_device	0.004082
14	session_duration	0.034082
15	clicks_on_ads	0.017037
16	page_views	0.021963
17	referral_source	0.001877
18	wishlist_additions	0.031523
19	cart_abandonment_rate	0.024459
20	average_spent	0.061594
21	user_income	0.059734
22	user_education	0.006332
23	user_marital_status	0.001262
24	product_availability	0.007449
25	stock_status	0.001825
26	product_return_rate	0.017853
27	product_color	0.012094
28	product_size	0.005794

```
29           is_top_seller    0.005822
30     discount_percentage    0.047713
31        time_to_purchase    0.096913
32           delivery_time    0.037564
33            shipping_fee    0.036183
34           seller_rating    0.004755
35    seller_response_time    0.053161
36         seller_location    0.002308
37  product_rating_variance   0.045403
38   review_sentiment_score   0.035723
39    user_engagement_score   0.045021
40           ad_click_rate    0.028940
41             time_of_day    0.007560
42             day_of_week    0.014141
43                  season    0.004010
44          payment_method    0.003444
45             coupon_used    0.003964
46      product_popularity    0.031880
47     purchase_time_month    0.015764
```

```
important_feature[important_feature['Importance']>=0.1]
```

```
Empty DataFrame
Columns: [Feature, Importance]
Index: []
```

```
(rfc.feature_importances_>=0.1).sum()
```

```
0
```

```
(rfc.feature_importances_>=0.07).sum()
```

```
1
```

```
important_feature[important_feature['Importance']>=0.07]
```

```
             Feature  Importance
31  time_to_purchase    0.096913
```

```
(rfc.feature_importances_>=0.05).sum()
```

```
4
```

```
important_feature[important_feature['Importance']>=0.05]
```

```
                 Feature  Importance
20         average_spent    0.061594
21           user_income    0.059734
31      time_to_purchase    0.096913
35  seller_response_time    0.053161
```

```
(rfc.feature_importances_>=0.03).sum()
```

```
15
```

```
important_feature[important_feature['Importance']>=0.03].Feature
```

```
3                  review_count
7                  time_on_page
14             session_duration
18           wishlist_additions
20                average_spent
21                  user_income
30          discount_percentage
31             time_to_purchase
32                delivery_time
33                 shipping_fee
35           seller_response_time
37        product_rating_variance
38         review_sentiment_score
39          user_engagement_score
46           product_popularity
Name: Feature, dtype: object
```

6) Gradient Boosting

```
from sklearn.ensemble import GradientBoostingClassifier
```

```
gbc=GradientBoostingClassifier(learning_rate=0.001,
n_estimators=150,max_depth=4)
gbc.fit(x_train,y_train)
```

```
GradientBoostingClassifier(learning_rate=0.001, max_depth=4,
n_estimators=150)
```

```
y_train_predict=gbc.predict(x_train)
evaluation_matrix(y_train,y_train_predict)
```

```
accuracy = 0.5262619047619047
precision = 0.5192237031098641
recall = 0.7385024705435196
f1 score = 0.609747965087771
Confusion Matrix:
 [[ 6559 14393]
 [ 5504 15544]]
```

```
y_test_predict=gbc.predict(x_test)
evaluation_matrix(y_test,y_test_predict)
```

```
accuracy = 0.5017777777777778
precision = 0.5027985074626866
recall = 0.7154867256637168
f1 score = 0.5905770635500365
Confusion Matrix:
```

```
[[2564 6396]
 [2572 6468]]
```

```
model_evaluation_metrics("Gradient Boosting -
Train",y_train,y_train_predict)
model_evaluation_metrics("Gradient Boosting  -
Test",y_test,y_test_predict)
model_metrics
```

	Model	Accuracy	Precision	Recall
0	Logistic Regression - Train	0.513119	0.513526	0.540241
1	Logistic Regression - Test	0.504389	0.506308	0.528319
2	Logistic Regression with SS - Train	0.513119	0.513526	0.540241
3	Logistic Regression with SS - Test	0.504389	0.506308	0.528319
4	Stats Model (LR) - Train	0.513119	0.513526	0.540241
5	Stats Model (LR) - Test	0.504389	0.506308	0.528319
6	DecisionTree - Train	0.513119	0.513526	0.540241
7	DecisionTree - Test	0.504389	0.506308	0.528319
8	Random Forest - Train	0.560167	0.554401	0.623385
9	Random Forest - Test	0.503778	0.505365	0.562721
10	Gradient Boosting - Train	0.526262	0.519224	0.738502
11	Gradient Boosting - Test	0.501778	0.502799	0.715487

	F1 Score
0	0.526545
1	0.517079
2	0.526545
3	0.517079
4	0.526545
5	0.517079
6	0.526545
7	0.517079
8	0.586872
9	0.532503
10	0.609748
11	0.590577

7) Adaboost

```
from sklearn.ensemble import AdaBoostClassifier

abc=AdaBoostClassifier(n_estimators=100,learning_rate=0.1,)
abc.fit(x_train,y_train)

AdaBoostClassifier(learning_rate=0.1, n_estimators=100)

y_train_predict=abc.predict(x_train)
evaluation_matrix(y_train,y_train_predict)

accuracy = 0.5242142857142857
precision = 0.5244928936111495
recall = 0.5417616875712656
f1 score = 0.5329874500455725
Confusion Matrix:
 [[10614 10338]
 [ 9645 11403]]

y_test_predict=abc.predict(x_test)
evaluation_matrix(y_test,y_test_predict)

accuracy = 0.5043333333333333
precision = 0.5063250428816467
recall = 0.5224557522123894
f1 score = 0.51426393728223
Confusion Matrix:
 [[4355 4605]
 [4317 4723]]

my_lr=LogisticRegression()
abc=AdaBoostClassifier(estimator=my_lr,n_estimators=40,learning_rate=0
.1,)
abc.fit(x_train,y_train)

AdaBoostClassifier(estimator=LogisticRegression(), learning_rate=0.1,
                   n_estimators=40)

y_train_predict=abc.predict(x_train)
evaluation_matrix(y_train,y_train_predict)

accuracy = 0.518095238095238
precision = 0.5181573033707865
recall = 0.5477480045610034
f1 score = 0.5325419187953254
Confusion Matrix:
 [[10231 10721]
 [ 9519 11529]]

y_test_predict=abc.predict(x_test)
evaluation_matrix(y_test,y_test_predict)
```

```
accuracy = 0.5007777777777778
precision = 0.5028559339961921
recall = 0.5258849557522124
f1 score = 0.5141126851951985
Confusion Matrix:
 [[4260 4700]
 [4286 4754]]
```

```python
my_rfc=RandomForestClassifier(n_estimators=40,max_depth=4,random_state
=42)
abc=AdaBoostClassifier(estimator=my_rfc,n_estimators=20,learning_rate=
0.1,)
abc.fit(x_train,y_train)
```

```
AdaBoostClassifier(estimator=RandomForestClassifier(max_depth=4,
                                                    n_estimators=40,
                                                    random_state=42),
                   learning_rate=0.1, n_estimators=20)
```

```python
y_train_predict=abc.predict(x_train)
evaluation_matrix(y_train,y_train_predict)
```

```
accuracy = 0.594
precision = 0.5898542903399892
recall = 0.6231470923603193
f1 score = 0.6060438037149987
Confusion Matrix:
 [[11832  9120]
 [ 7932 13116]]
```

```python
y_test_predict=abc.predict(x_test)
evaluation_matrix(y_test,y_test_predict)
```

```
accuracy = 0.5049444444444444
precision = 0.5067901884408885
recall = 0.5325221238938053
f1 score = 0.5193376126004638
Confusion Matrix:
 [[4275 4685]
 [4226 4814]]
```

```python
model_evaluation_metrics("AdaBoost - Train",y_train,y_train_predict)
model_evaluation_metrics("AdaBoost - Test",y_test,y_test_predict)
model_metrics
```

```
                                  Model  Accuracy  Precision    Recall
\
0           Logistic Regression - Train  0.513119   0.513526  0.540241

1            Logistic Regression - Test  0.504389   0.506308  0.528319
```

```
2   Logistic Regression with SS - Train  0.513119   0.513526  0.540241

3    Logistic Regression with SS - Test  0.504389   0.506308  0.528319

4                 Stats Model (LR) - Train  0.513119   0.513526  0.540241

5                  Stats Model (LR) - Test  0.504389   0.506308  0.528319

6                      DecisionTree - Train  0.513119   0.513526  0.540241

7                       DecisionTree - Test  0.504389   0.506308  0.528319

8                     Random Forest - Train  0.560167   0.554401  0.623385

9                      Random Forest - Test  0.503778   0.505365  0.562721

10                Gradient Boosting - Train  0.526262   0.519224  0.738502

11                Gradient Boosting  - Test  0.501778   0.502799  0.715487

12                         AdaBoost - Train  0.594000   0.589854  0.623147

13                          AdaBoost - Test  0.504944   0.506790  0.532522


    F1 Score
0   0.526545
1   0.517079
2   0.526545
3   0.517079
4   0.526545
5   0.517079
6   0.526545
7   0.517079
8   0.586872
9   0.532503
10  0.609748
11  0.590577
12  0.606044
13  0.519338
```

8) XGBoost

```
from xgboost import XGBClassifier

xgb=XGBClassifier(max_depth=2,max_leaf_nodes=3)
xgb.fit(x_train,y_train)

XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None,
```

```
early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None,
feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None,
max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=2, max_leaf_nodes=3,
              max_leaves=None, min_child_weight=None, missing=nan,
              monotone_constraints=None, multi_strategy=None,
n_estimators=None,
              n_jobs=None, num_parallel_tree=None, ...)
```

```
y_train_predict=xgb.predict(x_train)
evaluation_matrix(y_train,y_train_predict)
```

```
accuracy = 0.5687857142857143
precision = 0.5693244582920266
recall = 0.572976054732041
f1 score = 0.5711444199758471
Confusion Matrix:
 [[11829  9123]
 [ 8988 12060]]
```

```
y_test_predict=xgb.predict(x_test)
evaluation_matrix(y_test,y_test_predict)
```

```
accuracy = 0.5013888888888889
precision = 0.5035923510555985
recall = 0.5039823008849558
f1 score = 0.503787250511417
Confusion Matrix:
 [[4469 4491]
 [4484 4556]]
```

```
model_evaluation_metrics("XGBoost - Train",y_train,y_train_predict)
model_evaluation_metrics("XGBoost - Test",y_test,y_test_predict)
model_metrics
```

```
                                  Model  Accuracy  Precision    Recall  \
0            Logistic Regression - Train  0.513119   0.513526  0.540241

1             Logistic Regression - Test  0.504389   0.506308  0.528319

2    Logistic Regression with SS - Train  0.513119   0.513526  0.540241

3     Logistic Regression with SS - Test  0.504389   0.506308  0.528319

4               Stats Model (LR) - Train  0.513119   0.513526  0.540241
```

```
5                  Stats Model (LR) - Test  0.504389   0.506308  0.528319

6                     DecisionTree - Train  0.513119   0.513526  0.540241

7                      DecisionTree - Test  0.504389   0.506308  0.528319

8                    Random Forest - Train  0.560167   0.554401  0.623385

9                     Random Forest - Test  0.503778   0.505365  0.562721

10               Gradient Boosting - Train  0.526262   0.519224  0.738502

11               Gradient Boosting  - Test  0.501778   0.502799  0.715487

12                        AdaBoost - Train  0.594000   0.589854  0.623147

13                         AdaBoost - Test  0.504944   0.506790  0.532522

14                         XGBoost - Train  0.568786   0.569324  0.572976

15                          XGBoost - Test  0.501389   0.503592  0.503982


    F1 Score
0   0.526545
1   0.517079
2   0.526545
3   0.517079
4   0.526545
5   0.517079
6   0.526545
7   0.517079
8   0.586872
9   0.532503
10  0.609748
11  0.590577
12  0.606044
13  0.519338
14  0.571144
15  0.503787
```

9) KNN

```
from sklearn.neighbors import KNeighborsClassifier

for i in range(5,30,5):
    knn=KNeighborsClassifier(n_neighbors=i)
    knn.fit(x_train,y_train)
    y_train_predict=knn.predict(x_train)
    print()
```

```
    print(i,"Negihbors")
    print("train accuracy=",accuracy_score(y_train,y_train_predict),";
test accuracy=",accuracy_score(y_test,y_test_predict))
    y_test_predict=knn.predict(x_test)
```

```
5 Negihbors
train accuracy= 0.6852380952380952 ; test accuracy= 0.5013888888888889

10 Negihbors
train accuracy= 0.6198571428571429 ; test accuracy=
0.49994444444444447

15 Negihbors
train accuracy= 0.601 ; test accuracy= 0.49955555555555553

20 Negihbors
train accuracy= 0.5849047619047619 ; test accuracy= 0.5032777777777778

25 Negihbors
train accuracy= 0.5777142857142857 ; test accuracy= 0.5002777777777778
```

```
y_train_predict=knn.predict(x_train)
evaluation_matrix(y_train,y_train_predict)
```

```
accuracy = 0.5777142857142857
precision = 0.5811844298460633
recall = 0.5632364120106423
f1 score = 0.5720696810307387
Confusion Matrix:
 [[12409  8543]
 [ 9193 11855]]
```

```
y_test_predict=knn.predict(x_test)
evaluation_matrix(y_test,y_test_predict)
```

```
accuracy = 0.5025
precision = 0.5048844960349386
recall = 0.48595132743362834
f1 score = 0.49523702158841104
Confusion Matrix:
 [[4652 4308]
 [4647 4393]]
```

```
model_evaluation_metrics("KNN - Train",y_train,y_train_predict)
model_evaluation_metrics("KNN - Test",y_test,y_test_predict)
model_metrics
```

```
                                  Model  Accuracy  Precision    Recall  
\
0           Logistic Regression - Train  0.513119   0.513526  0.540241
```

1	Logistic Regression - Test	0.504389	0.506308	0.528319
2	Logistic Regression with SS - Train	0.513119	0.513526	0.540241
3	Logistic Regression with SS - Test	0.504389	0.506308	0.528319
4	Stats Model (LR) - Train	0.513119	0.513526	0.540241
5	Stats Model (LR) - Test	0.504389	0.506308	0.528319
6	DecisionTree - Train	0.513119	0.513526	0.540241
7	DecisionTree - Test	0.504389	0.506308	0.528319
8	Random Forest - Train	0.560167	0.554401	0.623385
9	Random Forest - Test	0.503778	0.505365	0.562721
10	Gradient Boosting - Train	0.526262	0.519224	0.738502
11	Gradient Boosting - Test	0.501778	0.502799	0.715487
12	AdaBoost - Train	0.594000	0.589854	0.623147
13	AdaBoost - Test	0.504944	0.506790	0.532522
14	XGBoost - Train	0.568786	0.569324	0.572976
15	XGBoost - Test	0.501389	0.503592	0.503982
16	KNN - Train	0.577714	0.581184	0.563236
17	KNN - Test	0.502500	0.504884	0.485951

	F1 Score
0	0.526545
1	0.517079
2	0.526545
3	0.517079
4	0.526545
5	0.517079
6	0.526545
7	0.517079
8	0.586872
9	0.532503
10	0.609748
11	0.590577
12	0.606044
13	0.519338

```
14  0.571144
15  0.503787
16  0.572070
17  0.495237
```

10) SVM

```
from sklearn.svm import SVC

svc= SVC(kernel='rbf')
svc.fit(x_train,y_train)

SVC()

y_train_predict=svc.predict(x_train)
evaluation_matrix(y_train,y_train_predict)

accuracy = 0.5626666666666666
precision = 0.5626578135228655
recall = 0.5716932725199544
f1 score = 0.5671395579016826
Confusion Matrix:
 [[11599  9353]
 [ 9015 12033]]

y_test_predict=svc.predict(x_test)
evaluation_matrix(y_test,y_test_predict)

accuracy = 0.5041111111111111
precision = 0.5063136907399203
recall = 0.5056415929203539
f1 score = 0.5059774186406907
Confusion Matrix:
 [[4503 4457]
 [4469 4571]]

model_evaluation_metrics("SVM - Train",y_train,y_train_predict)
model_evaluation_metrics("SVM - Test",y_test,y_test_predict)
model_metrics

                                  Model  Accuracy  Precision    Recall  \
0            Logistic Regression - Train  0.513119   0.513526  0.540241

1             Logistic Regression - Test  0.504389   0.506308  0.528319

2    Logistic Regression with SS - Train  0.513119   0.513526  0.540241

3     Logistic Regression with SS - Test  0.504389   0.506308  0.528319

4               Stats Model (LR) - Train  0.513119   0.513526  0.540241
```

5	Stats Model (LR) - Test	0.504389	0.506308	0.528319
6	DecisionTree - Train	0.513119	0.513526	0.540241
7	DecisionTree - Test	0.504389	0.506308	0.528319
8	Random Forest - Train	0.560167	0.554401	0.623385
9	Random Forest - Test	0.503778	0.505365	0.562721
10	Gradient Boosting - Train	0.526262	0.519224	0.738502
11	Gradient Boosting - Test	0.501778	0.502799	0.715487
12	AdaBoost - Train	0.594000	0.589854	0.623147
13	AdaBoost - Test	0.504944	0.506790	0.532522
14	XGBoost - Train	0.568786	0.569324	0.572976
15	XGBoost - Test	0.501389	0.503592	0.503982
16	KNN - Train	0.577714	0.581184	0.563236
17	KNN - Test	0.502500	0.504884	0.485951
18	SVM - Train	0.562667	0.562658	0.571693
19	SVM - Test	0.504111	0.506314	0.505642

	F1 Score
0	0.526545
1	0.517079
2	0.526545
3	0.517079
4	0.526545
5	0.517079
6	0.526545
7	0.517079
8	0.586872
9	0.532503
10	0.609748
11	0.590577
12	0.606044
13	0.519338
14	0.571144
15	0.503787
16	0.572070
17	0.495237

```
18  0.567140
19  0.505977
```

```
model_evaluation_metrics("SVM - Train",y_train,y_train_predict)
model_evaluation_metrics("SVM - Test",y_test,y_test_predict)
model_metrics
```

```
                                  Model  Accuracy  Precision    Recall  \
0            Logistic Regression - Train  0.513119   0.513526  0.540241
1             Logistic Regression - Test  0.504389   0.506308  0.528319
2    Logistic Regression with SS - Train  0.513119   0.513526  0.540241
3     Logistic Regression with SS - Test  0.504389   0.506308  0.528319
4                 Stats Model (LR) - Train  0.513119   0.513526  0.540241
5                  Stats Model (LR) - Test  0.504389   0.506308  0.528319
6                     DecisionTree - Train  0.513119   0.513526  0.540241
7                      DecisionTree - Test  0.504389   0.506308  0.528319
8                    Random Forest - Train  0.560167   0.554401  0.623385
9                     Random Forest - Test  0.503778   0.505365  0.562721
10               Gradient Boosting - Train  0.526262   0.519224  0.738502
11               Gradient Boosting  - Test  0.501778   0.502799  0.715487
12                        AdaBoost - Train  0.594000   0.589854  0.623147
13                         AdaBoost - Test  0.504944   0.506790  0.532522
14                         XGBoost - Train  0.568786   0.569324  0.572976
15                          XGBoost - Test  0.501389   0.503592  0.503982
16                             KNN - Train  0.577714   0.581184  0.563236
17                              KNN - Test  0.502500   0.504884  0.485951
18                             SVM - Train  0.562667   0.562658  0.571693
19                              SVM - Test  0.504111   0.506314  0.505642
20                             SVM - Train  0.562667   0.562658  0.571693
21                              SVM - Test  0.504111   0.506314  0.505642
```

	F1 Score
0	0.526545
1	0.517079
2	0.526545
3	0.517079
4	0.526545
5	0.517079
6	0.526545
7	0.517079
8	0.586872
9	0.532503
10	0.609748
11	0.590577
12	0.606044
13	0.519338
14	0.571144
15	0.503787
16	0.572070
17	0.495237
18	0.567140
19	0.505977
20	0.567140
21	0.505977

```
file_name=r'C:\Users\venky\Downloads\model_metrics.csv'
model_metrics.to_csv(file_name, index=False)
```