Analyzing and Preprocessing Data

January 13, 2026

##Pre-processing Data In this project, we will explore several methods of data preprocessing for Machine Learning.

Problem: Ensuring proper data cleaning and preprocessing to prepare datasets for Machine Learning applications.

[ ]:
```
#Importing python libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Here, we're going to use the E-commerce Customer Behavior dataset from Kaggle. Courtesy: https://www.kaggle.com/datasets/dhairyajeetsingh/ecommerce-customer-behavior-dataset

[ ]:
```
# Reading Dataset from local csv file
df_cust = pd.read_csv('ecommerce_customer_churn_dataset.csv')
```

Note: Pandas can read datasets in various formats:

CSV - read_csv(<filename>) : Reads comma separated files

TSV - read_csv(<filename>, sep= "̂") : Reads a"tab" delimited file, you can replace separator with :, | and read different types of delimited file

Excel - read_excel(<filename>, sheet_name=<worksheet_name>) : Reads .xls or .xlsx file

Let us check dimension of Dataset

[ ]:
```
print(df_cust.shape)
```

```
(50000, 25)
```

This data set has 25 features and 50000 samples

Let' examine few rows from this dataset

[ ]:
```
print(df_cust.head())
```

```
    Age  Gender Country        City  Membership_Years  Login_Frequency  \
0  43.0    Male  France   Marseille               2.9             14.0
1  36.0    Male      UK  Manchester               1.6             15.0
2  45.0  Female  Canada   Vancouver               2.9             10.0
3  56.0  Female     USA    New York               2.6             10.0
```

```
4  35.0    Male   India       Delhi              3.1              29.0

   Session_Duration_Avg  Pages_Per_Session  Cart_Abandonment_Rate  \
0                  27.4                6.0                   50.6
1                  42.7               10.3                   37.7
2                  24.8                1.6                   70.9
3                  38.4               14.8                   41.7
4                  51.4                NaN                   19.1

   Wishlist_Items  Total_Purchases  Average_Order_Value  \
0             3.0              9.0                94.72
1             1.0             19.5                82.45
2             1.0              9.1               165.52
3             9.0             15.0               147.33
4             9.0             32.5               141.30

   Days_Since_Last_Purchase  Discount_Usage_Rate  Returns_Rate  \
0                      34.0                46.40           2.0
1                      71.0                57.96           9.2
2                      11.0                12.24          11.5
3                      47.0                44.10           5.4
4                      73.0                25.20           5.5

   Email_Open_Rate  Customer_Service_Calls  Product_Reviews_Written  \
0             17.9                     9.0                      4.0
1             42.8                     7.0                      3.0
2              0.0                     4.0                      1.0
3             41.4                     2.0                      5.0
4             37.9                     1.0                     11.0

   Social_Media_Engagement_Score  Mobile_App_Usage  Payment_Method_Diversity  \
0                           16.3              20.8                       1.0
1                            NaN              23.3                       3.0
2                            NaN               8.8                       NaN
3                           85.9              31.0                       3.0
4                           83.0              50.4                       4.0

   Lifetime_Value  Credit_Balance  Churned Signup_Quarter
0          953.33          2278.0        0             Q1
1         1067.47          3028.0        0             Q4
2         1289.75          2317.0        0             Q4
3         2340.92          2674.0        0             Q1
4         3041.29          5354.0        0             Q4
```

By default, the head() function displays the first 5 rows and up to 20 columns. To view additional rows and columns, these defaults can be adjusted, allowing the DataFrame to be formatted into a more readable table.

[ ]:
```python
pd.set_option('display.max_columns', 25)
print(df_cust.head(3).to_string())
```

```
    Age  Gender Country        City  Membership_Years  Login_Frequency
Session_Duration_Avg  Pages_Per_Session  Cart_Abandonment_Rate  Wishlist_Items
Total_Purchases  Average_Order_Value  Days_Since_Last_Purchase
Discount_Usage_Rate  Returns_Rate  Email_Open_Rate  Customer_Service_Calls
Product_Reviews_Written  Social_Media_Engagement_Score  Mobile_App_Usage
Payment_Method_Diversity  Lifetime_Value  Credit_Balance  Churned Signup_Quarter
0  43.0    Male  France   Marseille               2.9              14.0
27.4               6.0                   50.6             3.0             9.0
94.72                     34.0                 46.40           2.0
17.9                     9.0                      4.0
16.3               20.8                       1.0         953.33          2278.0
0             Q1
1  36.0    Male      UK  Manchester               1.6              15.0
42.7              10.3                   37.7             1.0            19.5
82.45                     71.0                 57.96           9.2
42.8                     7.0                      3.0
NaN               23.3                        3.0        1067.47          3028.0
0             Q4
2  45.0  Female  Canada   Vancouver               2.9              10.0
24.8               1.6                   70.9             1.0             9.1
165.52                     11.0                 12.24           11.5
0.0                     4.0                      1.0
NaN                8.8                        NaN        1289.75          2317.0
0             Q4
```

Let us see the information about dataset

info() - gives us a quick snapshot of DataFrame. It shows us how many rows, columns & its data type, and whether there are any missing values

[ ]:
```python
print(df_cust.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 25 columns):
 #   Column                         Non-Null Count  Dtype
---  ------                         --------------  -----
 0   Age                            47505 non-null  float64
 1   Gender                         50000 non-null  object
 2   Country                        50000 non-null  object
 3   City                           50000 non-null  object
 4   Membership_Years               50000 non-null  float64
 5   Login_Frequency                50000 non-null  float64
 6   Session_Duration_Avg           46601 non-null  float64
 7   Pages_Per_Session              47000 non-null  float64
 8   Cart_Abandonment_Rate          50000 non-null  float64
```

```
 9   Wishlist_Items               46000 non-null  float64
 10  Total_Purchases              50000 non-null  float64
 11  Average_Order_Value          50000 non-null  float64
 12  Days_Since_Last_Purchase     47000 non-null  float64
 13  Discount_Usage_Rate          46500 non-null  float64
 14  Returns_Rate                 45509 non-null  float64
 15  Email_Open_Rate              47472 non-null  float64
 16  Customer_Service_Calls       49832 non-null  float64
 17  Product_Reviews_Written      46500 non-null  float64
 18  Social_Media_Engagement_Score 44000 non-null  float64
 19  Mobile_App_Usage             45000 non-null  float64
 20  Payment_Method_Diversity     47500 non-null  float64
 21  Lifetime_Value               50000 non-null  float64
 22  Credit_Balance               44500 non-null  float64
 23  Churned                      50000 non-null  int64
 24  Signup_Quarter               50000 non-null  object
dtypes: float64(20), int64(1), object(4)
memory usage: 9.5+ MB
None
```

Looking at the dataset, you'll notice a mix of float, integer, and text/object columns. Some columns also have missing entries, which shows up when the non-null count is below 50,000.

[ ]:
```
df_cust.columns[df_cust.isna().any()].tolist()
```

[ ]:
```
['Age',
 'Session_Duration_Avg',
 'Pages_Per_Session',
 'Wishlist_Items',
 'Days_Since_Last_Purchase',
 'Discount_Usage_Rate',
 'Returns_Rate',
 'Email_Open_Rate',
 'Customer_Service_Calls',
 'Product_Reviews_Written',
 'Social_Media_Engagement_Score',
 'Mobile_App_Usage',
 'Payment_Method_Diversity',
 'Credit_Balance']
```

To clean things up, let's replace the nulls in Session_Duration_Avg, Returns_Rate, and Email_Open_Rate with their average values.

[ ]:
```
avg_features = ['Session_Duration_Avg','Returns_Rate','Email_Open_Rate',
 ↪'Discount_Usage_Rate', 'Payment_Method_Diversity','Credit_Balance',
 ↪'Mobile_App_Usage']
```

describe() : generates summary statistics for numeric columns, like mean, median, min, max, and quartiles. It's a quick way to understand the distribution and spread of your dataset's values.

[ ]:
```
df_cust[avg_features].describe()
```

[ ]:
```
       Session_Duration_Avg  Returns_Rate  Email_Open_Rate  \
count          46601.000000  45509.000000     47472.000000
mean              27.660754      6.680913        20.937980
std               10.871013      6.143027        14.252561
min                1.000000      0.000000         0.000000
25%               19.700000      2.900000         9.900000
50%               26.800000      5.400000        19.700000
75%               34.700000      9.100000        30.400000
max               75.600000     99.615734        91.700000

       Discount_Usage_Rate  Payment_Method_Diversity  Credit_Balance  \
count         46500.000000              47500.000000    44500.000000
mean             41.997485                  2.353874     1966.233258
std              21.373642                  1.110012     1225.072166
min               0.240000                  1.000000        0.000000
25%              25.300000                  2.000000     1049.000000
50%              40.200000                  2.000000     1896.000000
75%              57.000000                  3.000000     2791.000000
max             116.640000                  5.000000     7197.000000

       Mobile_App_Usage
count      45000.000000
mean          19.371607
std            9.419252
min            0.000000
25%           12.500000
50%           18.600000
75%           25.500000
max           61.900000
```

Let's take a look at how the average feature values have changed in the data after we applied the transformation.

[ ]:
```
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer

avgimputer = SimpleImputer(strategy='mean')
col_trans = ColumnTransformer([('avg_imputer', avgimputer, avg_features)],
 ↪remainder='passthrough').set_output(transform="pandas")
df_cust_trans = col_trans.fit_transform(df_cust)
```

Here, We've imported SimpleImputer from the scikit-learn library.

Scikit-learn is a free and open-source Python package offering efficient tools for machine learning and data analysis.

SimpleImputer serves as a preprocessing utility that helps us handle missing numerical values in a dataset.

[ ]:
```
df_cust_trans.describe()
```

[ ]:
```
       avg_imputer__Session_Duration_Avg  avg_imputer__Returns_Rate  \
count                       50000.000000               50000.000000
mean                           27.660754                   6.680913
std                            10.494997                   5.860648
min                             1.000000                   0.000000
25%                            20.200000                   3.100000
50%                            27.660754                   6.000000
75%                            34.000000                   8.600000
max                            75.600000                  99.615734

       avg_imputer__Email_Open_Rate  avg_imputer__Discount_Usage_Rate  \
count                  50000.000000                      50000.000000
mean                      20.937980                         41.997485
std                       13.887576                         20.611979
min                        0.000000                          0.240000
25%                       10.500000                         26.500000
50%                       20.800000                         41.997485
75%                       29.700000                         55.560000
max                       91.700000                        116.640000

       avg_imputer__Payment_Method_Diversity  avg_imputer__Credit_Balance  \
count                           50000.000000                 50000.000000
mean                                2.353874                  1966.233258
std                                 1.081905                  1155.729342
min                                 1.000000                     0.000000
25%                                 2.000000                  1164.000000
50%                                 2.000000                  1966.233258
75%                                 3.000000                  2664.000000
max                                 5.000000                  7197.000000

       avg_imputer__Mobile_App_Usage  remainder__Age  \
count                   50000.000000    47505.000000
mean                       19.371607       37.802968
std                         8.935878       11.834668
min                         0.000000        5.000000
25%                        13.200000       29.000000
50%                        19.371607       38.000000
75%                        24.600000       46.000000
max                        61.900000      200.000000

       remainder__Membership_Years  remainder__Login_Frequency  \
count                 50000.000000                50000.000000
mean                      2.984009                   11.624660
std                       2.059105                    7.810657
```

```
min                     0.100000                    0.000000
25%                     1.400000                    6.000000
50%                     2.500000                   11.000000
75%                     4.000000                   17.000000
max                    10.000000                   46.000000

       remainder__Pages_Per_Session  remainder__Cart_Abandonment_Rate  \
count                  47000.000000                      50000.000000
mean                       8.737811                         57.079973
std                        3.778220                         16.282723
min                        1.000000                          0.000000
25%                        6.000000                         46.400000
50%                        8.400000                         58.100000
75%                       11.200000                         68.700000
max                       24.100000                        143.743350

       remainder__Wishlist_Items  remainder__Total_Purchases  \
count               46000.000000                50000.000000
mean                    4.298391                   13.111576
std                     3.189754                    7.017312
min                     0.000000                  -13.000000
25%                     2.000000                    8.000000
50%                     4.000000                   12.000000
75%                     6.000000                   17.000000
max                    28.000000                  128.700000

       remainder__Average_Order_Value  remainder__Days_Since_Last_Purchase  \
count                    50000.000000                         47000.000000
mean                       123.117330                            29.792872
std                        175.569714                            29.695062
min                         26.380000                             0.000000
25%                         87.050000                             9.000000
50%                        112.970000                            21.000000
75%                        144.440000                            41.000000
max                       9666.379178                           287.000000

       remainder__Customer_Service_Calls  remainder__Product_Reviews_Written  \
count                       49832.000000                        46500.000000
mean                            5.681831                            2.853312
std                             2.676052                            2.328948
min                             0.000000                            0.000000
25%                             4.000000                            1.000000
50%                             5.000000                            2.000000
75%                             7.000000                            4.000000
max                            21.000000                           21.000000

       remainder__Social_Media_Engagement_Score  remainder__Lifetime_Value  \
```

```
count                                      44000.000000               50000.000000
mean                                          29.364466                1440.626292
std                                           20.574021                 907.249443
min                                            0.000000                   0.000000
25%                                           13.200000                 789.817500
50%                                           27.600000                1243.415000
75%                                           43.100000                1874.000000
max                                          100.000000                8987.240000

       remainder__Churned
count        50000.000000
mean             0.289000
std              0.453302
min              0.000000
25%              0.000000
50%              0.000000
75%              1.000000
max              1.000000
```

Now let's use SimpleImputer to clean up the dataset by replacing nulls in Age, Wishlist_Items, and Customer_service_calls with the median values.

[ ]:
```
med_features = ['Age','Wishlist_Items','Customer_Service_Calls']
medimputer = SimpleImputer(strategy='median')
col_trans = ColumnTransformer([('avg_imputer', avgimputer, avg_features),
                               ('med_imputer', medimputer, med_features)],␣
 ↪remainder='passthrough').set_output(transform="pandas")
df_cust_trans = col_trans.fit_transform(df_cust)
```

ColumnTransformer() - lets you apply different preprocessing steps to different columns at once in a single pipeline.

[ ]:
```
df_cust_trans.describe()
```

[ ]:
```
       avg_imputer__Session_Duration_Avg  avg_imputer__Returns_Rate  \
count                       50000.000000               50000.000000
mean                           27.660754                   6.680913
std                            10.494997                   5.860648
min                             1.000000                   0.000000
25%                            20.200000                   3.100000
50%                            27.660754                   6.000000
75%                            34.000000                   8.600000
max                            75.600000                  99.615734

       avg_imputer__Email_Open_Rate  avg_imputer__Discount_Usage_Rate  \
count                  50000.000000                      50000.000000
mean                      20.937980                         41.997485
std                       13.887576                         20.611979
```

```
min                      0.000000                          0.240000
25%                     10.500000                         26.500000
50%                     20.800000                         41.997485
75%                     29.700000                         55.560000
max                     91.700000                        116.640000

       avg_imputer__Payment_Method_Diversity  avg_imputer__Credit_Balance  \
count                           50000.000000                 50000.000000
mean                                2.353874                  1966.233258
std                                 1.081905                  1155.729342
min                                 1.000000                     0.000000
25%                                 2.000000                  1164.000000
50%                                 2.000000                  1966.233258
75%                                 3.000000                  2664.000000
max                                 5.000000                  7197.000000

       avg_imputer__Mobile_App_Usage  med_imputer__Age  \
count                   50000.000000      50000.000000
mean                       19.371607         37.812800
std                         8.935878         11.535688
min                         0.000000          5.000000
25%                        13.200000         30.000000
50%                        19.371607         38.000000
75%                        24.600000         45.000000
max                        61.900000        200.000000

       med_imputer__Wishlist_Items  med_imputer__Customer_Service_Calls  \
count                 50000.000000                         50000.000000
mean                      4.274520                             5.679540
std                       3.060573                             2.671844
min                       0.000000                             0.000000
25%                       2.000000                             4.000000
50%                       4.000000                             5.000000
75%                       6.000000                             7.000000
max                      28.000000                            21.000000

       remainder__Membership_Years  remainder__Login_Frequency  \
count                 50000.000000                50000.000000
mean                      2.984009                   11.624660
std                       2.059105                    7.810657
min                       0.100000                    0.000000
25%                       1.400000                    6.000000
50%                       2.500000                   11.000000
75%                       4.000000                   17.000000
max                      10.000000                   46.000000

       remainder__Pages_Per_Session  remainder__Cart_Abandonment_Rate  \
```

```
count                  47000.000000                        50000.000000
mean                       8.737811                           57.079973
std                        3.778220                           16.282723
min                        1.000000                            0.000000
25%                        6.000000                           46.400000
50%                        8.400000                           58.100000
75%                       11.200000                           68.700000
max                       24.100000                          143.743350

       remainder__Total_Purchases  remainder__Average_Order_Value  \
count                50000.000000                    50000.000000
mean                    13.111576                      123.117330
std                      7.017312                      175.569714
min                    -13.000000                       26.380000
25%                      8.000000                       87.050000
50%                     12.000000                      112.970000
75%                     17.000000                      144.440000
max                    128.700000                     9666.379178

       remainder__Days_Since_Last_Purchase  \
count                         47000.000000
mean                             29.792872
std                              29.695062
min                               0.000000
25%                               9.000000
50%                              21.000000
75%                              41.000000
max                             287.000000

       remainder__Product_Reviews_Written  \
count                        46500.000000
mean                             2.853312
std                              2.328948
min                              0.000000
25%                              1.000000
50%                              2.000000
75%                              4.000000
max                             21.000000

       remainder__Social_Media_Engagement_Score  remainder__Lifetime_Value  \
count                              44000.000000               50000.000000
mean                                  29.364466                1440.626292
std                                   20.574021                 907.249443
min                                    0.000000                   0.000000
25%                                   13.200000                 789.817500
50%                                   27.600000                1243.415000
75%                                   43.100000                1874.000000
```

```
max                                    100.000000                8987.240000

       remainder__Churned
count        50000.000000
mean             0.289000
std              0.453302
min              0.000000
25%              0.000000
50%              0.000000
75%              1.000000
max              1.000000
```

Next, we'll apply SimpleImputer to fill missing values in selected columns using their most frequent entries.

[ ]:
```
mode_features =␣
  ↪['Pages_Per_Session','Days_Since_Last_Purchase','Product_Reviews_Written','Social_Media_Eng
modeimputer = SimpleImputer(strategy='most_frequent')
col_trans = ColumnTransformer([('avg_imputer', avgimputer, avg_features),
                               ('med_imputer', medimputer, med_features),
                               ('mode_imputer', modeimputer, mode_features)],
                               remainder='passthrough').
  ↪set_output(transform="pandas")
df_cust_trans = col_trans.fit_transform(df_cust)
```

When you set remainder="passthrough", it just means any columns you didn't transform will still be carried forward instead of being dropped.

And if you use set_output(transform="pandas"), it will give you back a pandas DataFrame with proper column names, which makes the results much easier to read and work with.

[ ]:
```
df_cust_trans.describe()
```

[ ]:
```
       avg_imputer__Session_Duration_Avg  avg_imputer__Returns_Rate  \
count                       50000.000000               50000.000000
mean                           27.660754                   6.680913
std                            10.494997                   5.860648
min                             1.000000                   0.000000
25%                            20.200000                   3.100000
50%                            27.660754                   6.000000
75%                            34.000000                   8.600000
max                            75.600000                  99.615734

       avg_imputer__Email_Open_Rate  avg_imputer__Discount_Usage_Rate  \
count                  50000.000000                      50000.000000
mean                      20.937980                         41.997485
std                       13.887576                         20.611979
min                        0.000000                          0.240000
```

```
25%                         10.500000                            26.500000
50%                         20.800000                            41.997485
75%                         29.700000                            55.560000
max                         91.700000                           116.640000

       avg_imputer__Payment_Method_Diversity  avg_imputer__Credit_Balance  \
count                           50000.000000                 50000.000000
mean                                2.353874                  1966.233258
std                                 1.081905                  1155.729342
min                                 1.000000                     0.000000
25%                                 2.000000                  1164.000000
50%                                 2.000000                  1966.233258
75%                                 3.000000                  2664.000000
max                                 5.000000                  7197.000000

       avg_imputer__Mobile_App_Usage  med_imputer__Age  \
count                   50000.000000      50000.000000
mean                       19.371607         37.812800
std                         8.935878         11.535688
min                         0.000000          5.000000
25%                        13.200000         30.000000
50%                        19.371607         38.000000
75%                        24.600000         45.000000
max                        61.900000        200.000000

       med_imputer__Wishlist_Items  med_imputer__Customer_Service_Calls  \
count                 50000.000000                         50000.000000
mean                      4.274520                             5.679540
std                       3.060573                             2.671844
min                       0.000000                             0.000000
25%                       2.000000                             4.000000
50%                       4.000000                             5.000000
75%                       6.000000                             7.000000
max                      28.000000                            21.000000

       mode_imputer__Pages_Per_Session  \
count                     50000.000000
mean                          8.705542
std                           3.665344
min                           1.000000
25%                           6.200000
50%                           8.200000
75%                          11.000000
max                          24.100000

       mode_imputer__Days_Since_Last_Purchase  \
count                            50000.000000
```

```
mean                                 28.065300
std                                  29.591317
min                                   0.000000
25%                                   7.000000
50%                                  19.000000
75%                                  39.000000
max                                 287.000000

       mode_imputer__Product_Reviews_Written  \
count                           50000.000000
mean                                2.723580
std                                 2.295194
min                                 0.000000
25%                                 1.000000
50%                                 2.000000
75%                                 4.000000
max                                21.000000

       mode_imputer__Social_Media_Engagement_Score  \
count                                 50000.000000
mean                                     25.840730
std                                      21.530263
min                                       0.000000
25%                                       5.700000
50%                                      23.700000
75%                                      40.700000
max                                     100.000000

       remainder__Membership_Years  remainder__Login_Frequency  \
count                 50000.000000                50000.000000
mean                      2.984009                   11.624660
std                       2.059105                    7.810657
min                       0.100000                    0.000000
25%                       1.400000                    6.000000
50%                       2.500000                   11.000000
75%                       4.000000                   17.000000
max                      10.000000                   46.000000

       remainder__Cart_Abandonment_Rate  remainder__Total_Purchases  \
count                      50000.000000                50000.000000
mean                          57.079973                   13.111576
std                           16.282723                    7.017312
min                            0.000000                  -13.000000
25%                           46.400000                    8.000000
50%                           58.100000                   12.000000
75%                           68.700000                   17.000000
max                          143.743350                  128.700000
```

```
       remainder__Average_Order_Value  remainder__Lifetime_Value  \
count                     50000.000000               50000.000000
mean                        123.117330                1440.626292
std                         175.569714                 907.249443
min                          26.380000                   0.000000
25%                          87.050000                 789.817500
50%                         112.970000                1243.415000
75%                         144.440000                1874.000000
max                        9666.379178                8987.240000

       remainder__Churned
count        50000.000000
mean             0.289000
std              0.453302
min              0.000000
25%              0.000000
50%              0.000000
75%              1.000000
max              1.000000
```

Let's quickly check if there are still any columns with missing data.

[ ]:
```
df_cust_trans.columns[df_cust_trans.isna().any()].tolist()
```

[ ]:
```
[]
```

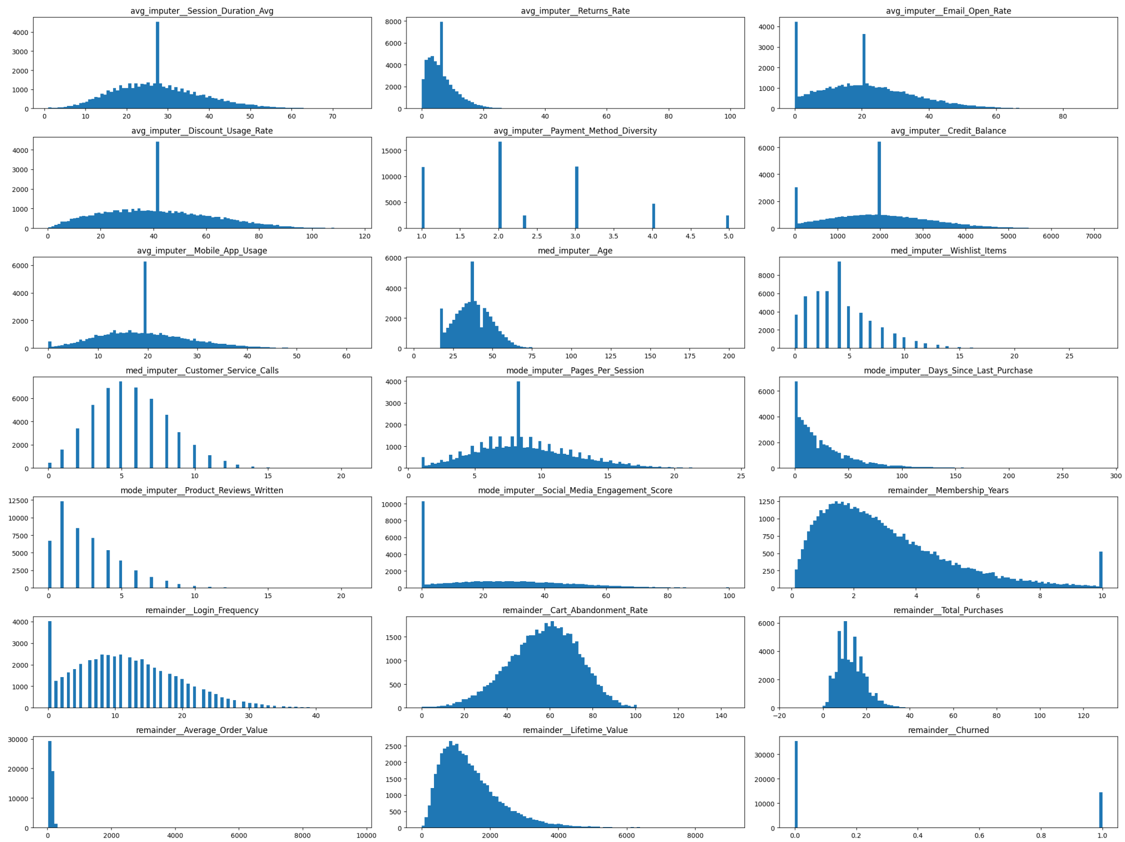
isna() - checks for missing values in a DataFrame or Series. It returns a boolean mask, with True where values are null and False otherwise.

any() - checks if at least one value in a Series or DataFrame is True.

tolist() - converts a pandas Series or DataFrame values into a plain Python list

Next, let's create visualizations to examine how the values are distributed across each column.

[ ]:
```
numeric_cols=df_cust_trans.columns[df_cust_trans.dtypes != 'object']
fig, axes = plt.subplots(nrows=7,ncols=3, figsize=(24,18))
axes=axes.ravel()
idx=0
for col in numeric_cols:
    if idx < len(axes): # Added a check to prevent IndexError
        axes[idx].set_title(col)
        axes[idx].hist(df_cust_trans[col],bins=100)
    idx+=1
plt.tight_layout()
plt.show()
```

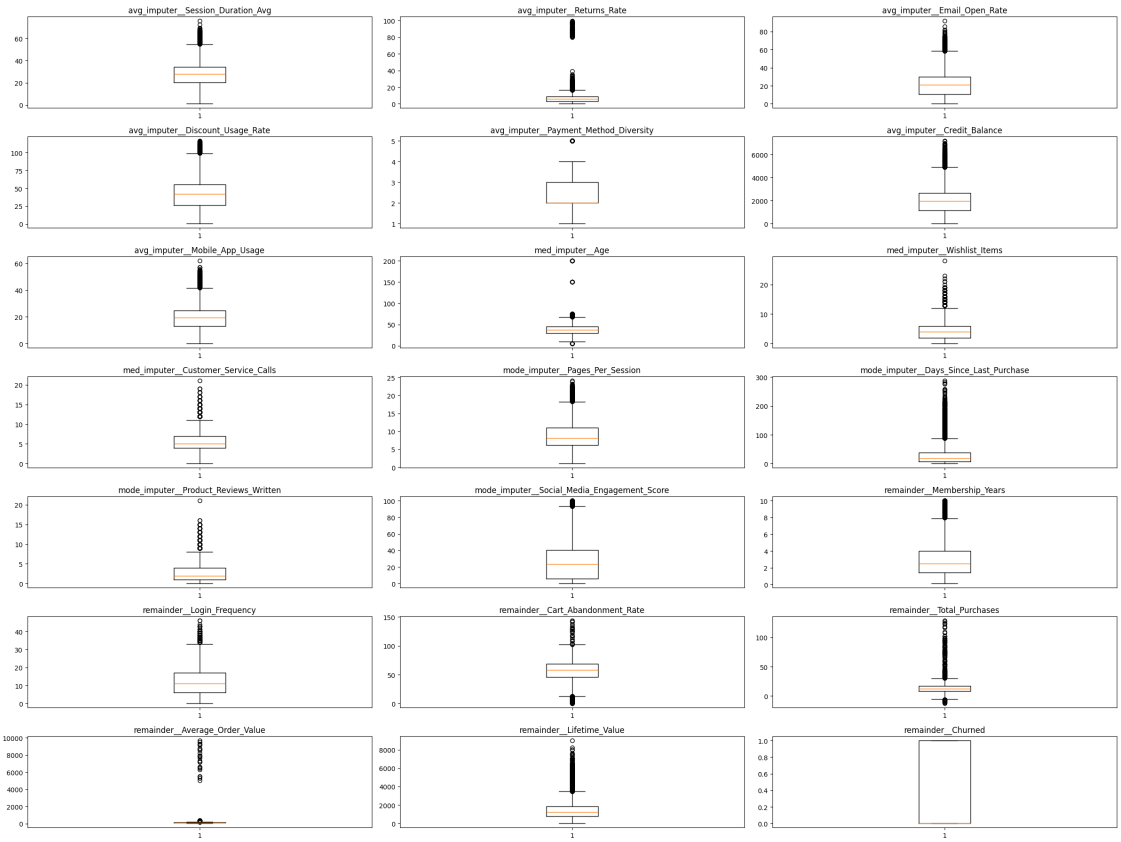


The above code builds a grid of histograms where each subplot shows the distribution of one numeric column. By arranging them side-by-side, thus you can easily see how features are spread out, spot skewness, or detect outliers.

- **Type of chart**: Histograms are used to show how values are distributed for each numeric column.
- **Grid**: A grid of plots is created (7 rows × 3 columns) to organize multiple charts on one canvas.
- **Subplot**: Each subplot corresponds to one column, with its own title and histogram.

The data isn't normalized — the ranges vary a lot and some columns have outliers. So, we'll start by spotting outliers with boxplots, and then normalize the data.

[ ]:
```
fig, axes = plt.subplots(nrows=7,ncols=3, figsize=(24,18))
axes=axes.ravel()
idx=0
for col in numeric_cols:
    axes[idx].set_title(col)
    axes[idx].boxplot(df_cust_trans[col])
    idx+=1
plt.tight_layout()
plt.show()
```



[ ]:

```
```

Next, let's analyze scatter plots of the numeric columns in relation to the Age column.

[ ]:

```python
y=df_cust_trans['med_imputer__Age']
fig, axes = plt.subplots(nrows=6,ncols=4, figsize=(24,18))
axes=axes.ravel()
idx=0
for col in numeric_cols:
    axes[idx].set_title(col)
    axes[idx].scatter(df_cust_trans[col],y)
    idx+=1
plt.tight_layout()
plt.show()
```

Next, we'll handle the outliers by applying the Winsorize technique.

- Winsorizing is a statistical method used to limit extreme values in data.
- Instead of removing outliers completely, it replaces extreme values (both very high and very low) with the nearest values within a chosen percentile range.
- Example: If you Winsorize at the 5th and 95th percentiles, values below the 5th percentile are set to the 5th percentile value, and values above the 95th percentile.

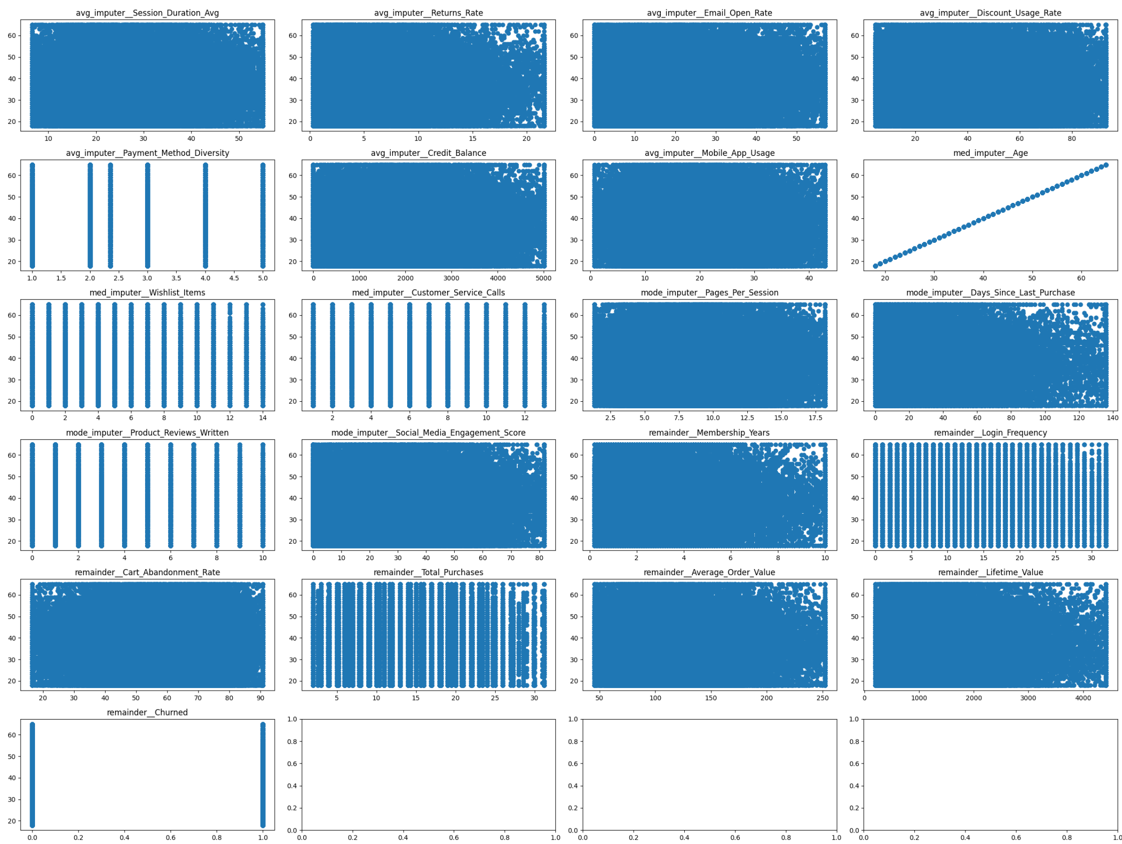
[ ]:
```
def winsorize(df, col, lower=0.01, upper=0.99):
    low_val = df[col].quantile(lower)
    high_val = df[col].quantile(upper)
    df[col] = np.clip(df[col], low_val, high_val)
    return df
```

[ ]:
```
for col in numeric_cols:
  df_cust_trans = winsorize(df_cust_trans,col)
```

Now, we'll re-plot the scatter plots after removing the outliers to examine the cleaned data.

[ ]:
```
y=df_cust_trans['med_imputer__Age']
fig, axes = plt.subplots(nrows=6,ncols=4, figsize=(24,18))
axes=axes.ravel()
```

```python
idx=0
for col in numeric_cols:
    axes[idx].set_title(col)
    axes[idx].scatter(df_cust_trans[col],y)
    idx+=1
plt.tight_layout()
plt.show()
```



Let's scale the numeric columns so they're centered at 0 with variance 1, giving all features the same influence in training. We'll use StandardScaler to do this.

[ ]:
```python
from sklearn.preprocessing import StandardScaler

col_scaler = ColumnTransformer([("num_data", StandardScaler(), numeric_cols),],
                               remainder='passthrough').set_output(transform="pandas")
df_cust_scaled = col_scaler.fit_transform(df_cust_trans)
```

StandardScaler() -transforms numeric features to have mean=0 and variance=1. This ensures all columns are on the same scale, preventing large-range features from dominating model training

[ ]:
```
df_cust_scaled.describe()
```

[ ]:
```
       num_data__avg_imputer__Session_Duration_Avg  \
count                                 5.000000e+04
mean                                 -4.017409e-16
std                                   1.000010e+00
min                                  -2.036323e+00
25%                                  -7.203269e-01
50%                                   1.608806e-03
75%                                   6.150224e-01
max                                   2.647076e+00

       num_data__avg_imputer__Returns_Rate  \
count                         5.000000e+04
mean                          4.033751e-16
std                           1.000010e+00
min                          -1.379414e+00
25%                          -7.591560e-01
50%                          -1.167458e-01
75%                           4.592082e-01
max                           3.338978e+00

       num_data__avg_imputer__Email_Open_Rate  \
count                            5.000000e+04
mean                             1.818989e-17
std                              1.000010e+00
min                             -1.521137e+00
25%                             -7.563769e-01
50%                             -6.183219e-03
75%                              6.420424e-01
max                              2.630420e+00

       num_data__avg_imputer__Discount_Usage_Rate  \
count                                5.000000e+04
mean                                -1.826095e-17
std                                  1.000010e+00
min                                 -1.826400e+00
25%                                 -7.573748e-01
50%                                  2.588236e-03
75%                                  6.676646e-01
max                                  2.503690e+00

       num_data__avg_imputer__Payment_Method_Diversity  \
count                                     5.000000e+04
mean                                     -2.642508e-16
std                                       1.000010e+00
min                                      -1.251392e+00
```

```
25%                                          -3.270872e-01
50%                                          -3.270872e-01
75%                                           5.972177e-01
max                                           2.445828e+00

       num_data__avg_imputer__Credit_Balance  \
count                           5.000000e+04
mean                            1.989520e-17
std                             1.000010e+00
min                            -1.716946e+00
25%                            -6.982743e-01
50%                             3.798246e-03
75%                             6.144471e-01
max                             2.662301e+00

       num_data__avg_imputer__Mobile_App_Usage  num_data__med_imputer__Age  \
count                             5.000000e+04                5.000000e+04
mean                              1.830358e-16               -2.596323e-16
std                               1.000010e+00                1.000010e+00
min                              -2.113401e+00               -1.782070e+00
25%                              -6.964337e-01               -6.985067e-01
50%                               3.163578e-03                2.386910e-02
75%                               5.958406e-01                6.559479e-01
max                               2.670281e+00                2.461887e+00

       num_data__med_imputer__Wishlist_Items  \
count                           5.000000e+04
mean                            1.251976e-16
std                             1.000010e+00
min                            -1.414477e+00
25%                            -7.506744e-01
50%                            -8.687184e-02
75%                             5.769307e-01
max                             3.232141e+00

       num_data__med_imputer__Customer_Service_Calls  \
count                                   5.000000e+04
mean                                   -4.938272e-17
std                                     1.000010e+00
min                                    -1.781862e+00
25%                                    -6.396575e-01
50%                                    -2.589226e-01
75%                                     5.025473e-01
max                                     2.786957e+00

       num_data__mode_imputer__Pages_Per_Session  \
count                               5.000000e+04
```

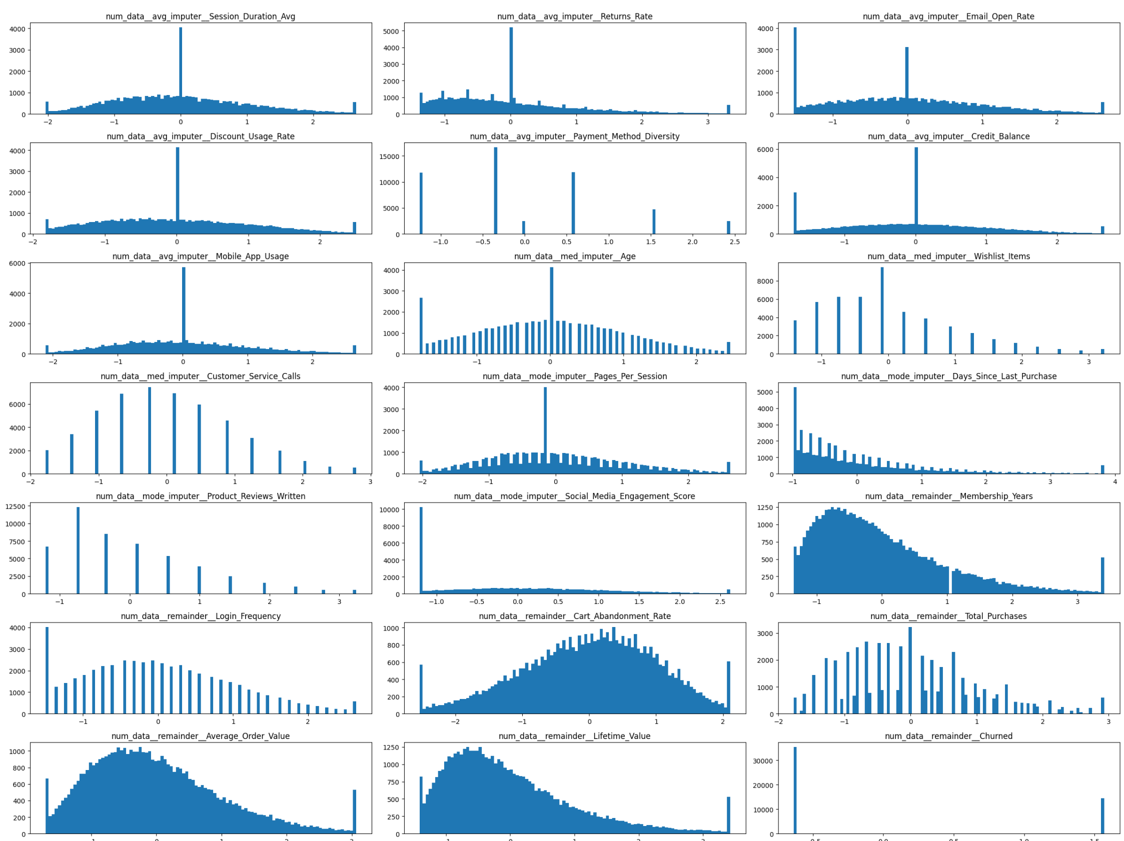
```
mean                                5.641709e-16
std                                 1.000010e+00
min                                -2.042588e+00
25%                                -6.891226e-01
50%                                -1.366878e-01
75%                                 6.367208e-01
max                                 2.625486e+00

       num_data__mode_imputer__Days_Since_Last_Purchase  \
count                                      5.000000e+04
mean                                       7.929657e-17
std                                        1.000010e+00
min                                       -9.840524e-01
25%                                       -7.360205e-01
50%                                       -3.108229e-01
75%                                        3.978396e-01
max                                        3.834853e+00

       num_data__mode_imputer__Product_Reviews_Written  \
count                                     5.000000e+04
mean                                     -4.131806e-17
std                                       1.000010e+00
min                                      -1.205141e+00
25%                                      -7.607444e-01
50%                                      -3.163480e-01
75%                                       5.724447e-01
max                                       3.238823e+00

       num_data__mode_imputer__Social_Media_Engagement_Score  \
count                                       5.000000e+04
mean                                        8.697043e-17
std                                         1.000010e+00
min                                        -1.208506e+00
25%                                        -9.411857e-01
50%                                        -9.701635e-02
75%                                         7.002547e-01
max                                         2.623085e+00

       num_data__remainder__Membership_Years  \
count                           5.000000e+04
mean                           -2.651035e-16
std                             1.000010e+00
min                            -1.352749e+00
25%                            -7.697732e-01
50%                            -2.353785e-01
75%                             4.933415e-01
max                             3.408222e+00
```

```
       num_data__remainder__Login_Frequency  \
count                          5.000000e+04
mean                           3.478107e-17
std                            1.000010e+00
min                           -1.500666e+00
25%                           -7.242240e-01
50%                           -7.718872e-02
75%                            6.992536e-01
max                            2.640359e+00

       num_data__remainder__Cart_Abandonment_Rate  \
count                                5.000000e+04
mean                                 7.673862e-18
std                                  1.000010e+00
min                                 -2.537946e+00
25%                                 -6.681313e-01
50%                                  6.351092e-02
75%                                  7.263663e-01
max                                  2.108357e+00

       num_data__remainder__Total_Purchases  \
count                          5.000000e+04
mean                           2.316369e-16
std                            1.000010e+00
min                           -1.771100e+00
25%                           -8.045689e-01
50%                           -1.602148e-01
75%                            6.452278e-01
max                            2.932685e+00

       num_data__remainder__Average_Order_Value  \
count                              5.000000e+04
mean                               9.407586e-17
std                                1.000010e+00
min                               -1.704752e+00
25%                               -7.419689e-01
50%                               -1.443825e-01
75%                                5.811593e-01
max                                3.067877e+00

       num_data__remainder__Lifetime_Value  num_data__remainder__Churned
count                         5.000000e+04                  5.000000e+04
mean                         -9.762857e-17                  6.870948e-17
std                           1.000010e+00                  1.000010e+00
min                          -1.411868e+00                 -6.375498e-01
25%                          -7.370380e-01                 -6.375498e-01
```

```
50%                                -2.173839e-01                -6.375498e-01
75%                                 5.050320e-01                 1.568505e+00
max                                 3.422027e+00                 1.568505e+00
```

Now, we'll re-plot the histogram plots after normalizing to examine the cleaned data.

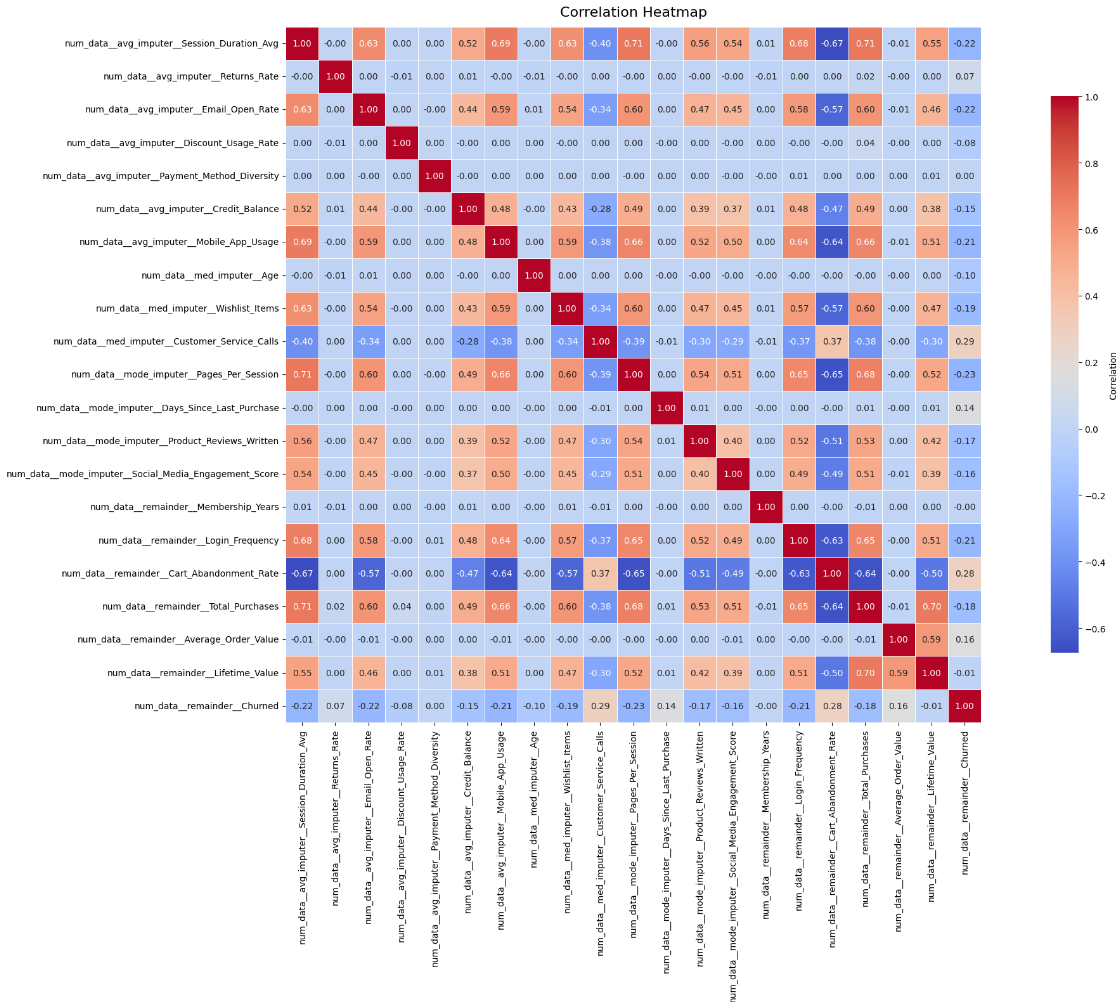
[ ]:
```python
numeric_cols_scaled=df_cust_scaled.columns[df_cust_scaled.dtypes != 'object']
fig, axes = plt.subplots(nrows=7,ncols=3, figsize=(24,18))
axes=axes.ravel()
idx=0
for col in numeric_cols_scaled:
    if idx < len(axes): # Added a check to prevent IndexError
        axes[idx].set_title(col)
        axes[idx].hist(df_cust_scaled[col],bins=100)
    idx+=1
plt.tight_layout()
plt.show()
```



Now let's check how the numeric columns are related to each other using correlation.

Correlation measures how strongly two variables move together. A positive correlation means both increase or decrease together, while a negative correlation means one increases as the other decreases.

[ ]:
```python
import seaborn as sns

correlations = {}

for col in numeric_cols_scaled:
    correlations[col] = df_cust_scaled[numeric_cols_scaled].corr()[col].
 ↪to_dict()



# Convert dict-of-dicts to a DataFrame (matrix)
corr_df = pd.DataFrame(correlations)

# Ensure consistent ordering of rows/columns
corr_df = corr_df.loc[numeric_cols_scaled, numeric_cols_scaled]

# Plot heatmap
plt.figure(figsize=(24, 16))
sns.heatmap(corr_df,
            annot=True,
            cmap='coolwarm',
            fmt='.2f',
            linewidths=0.5,
            linecolor='white',
            square=True,
            cbar_kws={'shrink': .8, 'label': 'Correlation'})
plt.title('Correlation Heatmap', fontsize=16, pad=12)
plt.tight_layout()
plt.show()
```



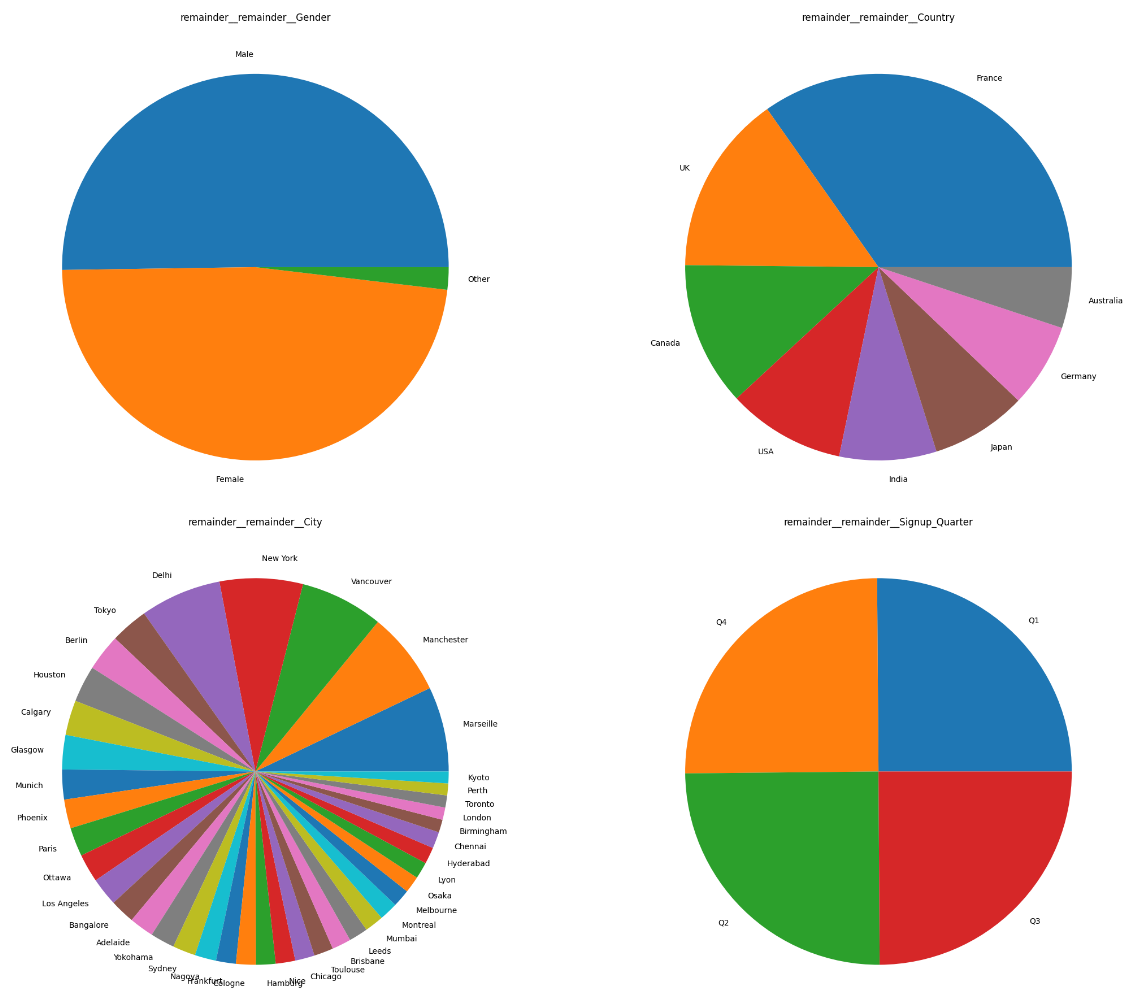
Seaborn is a Python library that makes it easy to create beautiful charts. It's built on top of Matplotlib and adds simple functions for common plots like histograms, scatter plots, and heatmaps.

Next, we'll examine the non-numeric features in the dataset.

We will use pie chart to visualize categorical data. Each slice represents a category, and its size shows the proportion or frequency of that category. This makes it easy to compare groups at a glance and understand how the dataset is distributed across categories.

[ ]:
```
categ_cols=df_cust_scaled.select_dtypes(include=['object', 'category']).columns.
  ↪tolist()
fig, axes = plt.subplots(nrows=2,ncols=2, figsize=(24,18))
axes=axes.ravel()
idx=0
for col in categ_cols:
    axes[idx].set_title(col)
```

```
    axes[idx].pie(df_cust_scaled[col].value_counts(),labels=df_cust_scaled[col].unique())
    idx+=1
plt.tight_layout()
plt.show()
```



Since machine learning models cannot directly process text data, we will convert categorical variables into numerical form using OneHotEncoder.

[ ]:
```
!pip install category_encoders
```

```
Collecting category_encoders
  Downloading category_encoders-2.9.0-py3-none-any.whl.metadata (7.9 kB)
Requirement already satisfied: numpy>=1.14.0 in /usr/local/lib/python3.12/dist-packages (from category_encoders) (2.0.2)
Requirement already satisfied: pandas>=1.0.5 in /usr/local/lib/python3.12/dist-packages (from category_encoders) (2.2.2)
```

```
Requirement already satisfied: patsy>=0.5.1 in /usr/local/lib/python3.12/dist-
packages (from category_encoders) (1.0.2)
Requirement already satisfied: scikit-learn>=1.6.0 in
/usr/local/lib/python3.12/dist-packages (from category_encoders) (1.6.1)
Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.12/dist-
packages (from category_encoders) (1.16.3)
Requirement already satisfied: statsmodels>=0.9.0 in
/usr/local/lib/python3.12/dist-packages (from category_encoders) (0.14.6)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.12/dist-packages (from pandas>=1.0.5->category_encoders)
(2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.12/dist-
packages (from pandas>=1.0.5->category_encoders) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.12/dist-
packages (from pandas>=1.0.5->category_encoders) (2025.3)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.12/dist-
packages (from scikit-learn>=1.6.0->category_encoders) (1.5.3)
Requirement already satisfied: threadpoolctl>=3.1.0 in
/usr/local/lib/python3.12/dist-packages (from scikit-
learn>=1.6.0->category_encoders) (3.6.0)
Requirement already satisfied: packaging>=21.3 in
/usr/local/lib/python3.12/dist-packages (from
statsmodels>=0.9.0->category_encoders) (25.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-
packages (from python-dateutil>=2.8.2->pandas>=1.0.5->category_encoders)
(1.17.0)
Downloading category_encoders-2.9.0-py3-none-any.whl (85 kB)
                         85.9/85.9 kB
6.5 MB/s eta 0:00:00
Installing collected packages: category_encoders
Successfully installed category_encoders-2.9.0
```

[ ]:
```
from category_encoders import OneHotEncoder

col_trans = ColumnTransformer([("categ_data", OneHotEncoder(), categ_cols)],
                              remainder='passthrough').
 ↪set_output(transform="pandas")
df_cust_categ = col_trans.fit_transform(df_cust_scaled)
```

Let's check what happens to our categorical columns once we use OneHotEncoder.

[ ]:
```
df_cust_categ.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 76 columns):
 #   Column                                                             Non-Null
Count  Dtype
```

```
---  ------
--------------  -----
 0   categ_data__remainder__remainder__Gender_1                         50000
non-null  int64
 1   categ_data__remainder__remainder__Gender_2                         50000
non-null  int64
 2   categ_data__remainder__remainder__Gender_3                         50000
non-null  int64
 3   categ_data__remainder__remainder__Country_1                        50000
non-null  int64
 4   categ_data__remainder__remainder__Country_2                        50000
non-null  int64
 5   categ_data__remainder__remainder__Country_3                        50000
non-null  int64
 6   categ_data__remainder__remainder__Country_4                        50000
non-null  int64
 7   categ_data__remainder__remainder__Country_5                        50000
non-null  int64
 8   categ_data__remainder__remainder__Country_6                        50000
non-null  int64
 9   categ_data__remainder__remainder__Country_7                        50000
non-null  int64
 10  categ_data__remainder__remainder__Country_8                        50000
non-null  int64
 11  categ_data__remainder__remainder__City_1                           50000
non-null  int64
 12  categ_data__remainder__remainder__City_2                           50000
non-null  int64
 13  categ_data__remainder__remainder__City_3                           50000
non-null  int64
 14  categ_data__remainder__remainder__City_4                           50000
non-null  int64
 15  categ_data__remainder__remainder__City_5                           50000
non-null  int64
 16  categ_data__remainder__remainder__City_6                           50000
non-null  int64
 17  categ_data__remainder__remainder__City_7                           50000
non-null  int64
 18  categ_data__remainder__remainder__City_8                           50000
non-null  int64
 19  categ_data__remainder__remainder__City_9                           50000
non-null  int64
 20  categ_data__remainder__remainder__City_10                          50000
non-null  int64
 21  categ_data__remainder__remainder__City_11                          50000
non-null  int64
 22  categ_data__remainder__remainder__City_12                          50000
non-null  int64
```

```
 23  categ_data__remainder__remainder__City_13                                    50000
non-null  int64
 24  categ_data__remainder__remainder__City_14                                    50000
non-null  int64
 25  categ_data__remainder__remainder__City_15                                    50000
non-null  int64
 26  categ_data__remainder__remainder__City_16                                    50000
non-null  int64
 27  categ_data__remainder__remainder__City_17                                    50000
non-null  int64
 28  categ_data__remainder__remainder__City_18                                    50000
non-null  int64
 29  categ_data__remainder__remainder__City_19                                    50000
non-null  int64
 30  categ_data__remainder__remainder__City_20                                    50000
non-null  int64
 31  categ_data__remainder__remainder__City_21                                    50000
non-null  int64
 32  categ_data__remainder__remainder__City_22                                    50000
non-null  int64
 33  categ_data__remainder__remainder__City_23                                    50000
non-null  int64
 34  categ_data__remainder__remainder__City_24                                    50000
non-null  int64
 35  categ_data__remainder__remainder__City_25                                    50000
non-null  int64
 36  categ_data__remainder__remainder__City_26                                    50000
non-null  int64
 37  categ_data__remainder__remainder__City_27                                    50000
non-null  int64
 38  categ_data__remainder__remainder__City_28                                    50000
non-null  int64
 39  categ_data__remainder__remainder__City_29                                    50000
non-null  int64
 40  categ_data__remainder__remainder__City_30                                    50000
non-null  int64
 41  categ_data__remainder__remainder__City_31                                    50000
non-null  int64
 42  categ_data__remainder__remainder__City_32                                    50000
non-null  int64
 43  categ_data__remainder__remainder__City_33                                    50000
non-null  int64
 44  categ_data__remainder__remainder__City_34                                    50000
non-null  int64
 45  categ_data__remainder__remainder__City_35                                    50000
non-null  int64
 46  categ_data__remainder__remainder__City_36                                    50000
non-null  int64
```

```
 47  categ_data__remainder__remainder__City_37                          50000
non-null  int64
 48  categ_data__remainder__remainder__City_38                          50000
non-null  int64
 49  categ_data__remainder__remainder__City_39                          50000
non-null  int64
 50  categ_data__remainder__remainder__City_40                          50000
non-null  int64
 51  categ_data__remainder__remainder__Signup_Quarter_1                 50000
non-null  int64
 52  categ_data__remainder__remainder__Signup_Quarter_2                 50000
non-null  int64
 53  categ_data__remainder__remainder__Signup_Quarter_3                 50000
non-null  int64
 54  categ_data__remainder__remainder__Signup_Quarter_4                 50000
non-null  int64
 55  remainder__num_data__avg_imputer__Session_Duration_Avg             50000
non-null  float64
 56  remainder__num_data__avg_imputer__Returns_Rate                     50000
non-null  float64
 57  remainder__num_data__avg_imputer__Email_Open_Rate                  50000
non-null  float64
 58  remainder__num_data__avg_imputer__Discount_Usage_Rate              50000
non-null  float64
 59  remainder__num_data__avg_imputer__Payment_Method_Diversity         50000
non-null  float64
 60  remainder__num_data__avg_imputer__Credit_Balance                   50000
non-null  float64
 61  remainder__num_data__avg_imputer__Mobile_App_Usage                 50000
non-null  float64
 62  remainder__num_data__med_imputer__Age                              50000
non-null  float64
 63  remainder__num_data__med_imputer__Wishlist_Items                   50000
non-null  float64
 64  remainder__num_data__med_imputer__Customer_Service_Calls           50000
non-null  float64
 65  remainder__num_data__mode_imputer__Pages_Per_Session               50000
non-null  float64
 66  remainder__num_data__mode_imputer__Days_Since_Last_Purchase        50000
non-null  float64
 67  remainder__num_data__mode_imputer__Product_Reviews_Written         50000
non-null  float64
 68  remainder__num_data__mode_imputer__Social_Media_Engagement_Score   50000
non-null  float64
 69  remainder__num_data__remainder__Membership_Years                   50000
non-null  float64
 70  remainder__num_data__remainder__Login_Frequency                    50000
non-null  float64
```

```
 71  remainder__num_data__remainder__Cart_Abandonment_Rate              50000
non-null  float64
 72  remainder__num_data__remainder__Total_Purchases                    50000
non-null  float64
 73  remainder__num_data__remainder__Average_Order_Value                50000
non-null  float64
 74  remainder__num_data__remainder__Lifetime_Value                     50000
non-null  float64
 75  remainder__num_data__remainder__Churned                            50000
non-null  float64
dtypes: float64(21), int64(55)
memory usage: 29.0 MB
```

As we can see each value in category features has converted into a column (3 value in Gender is completed into 3 columns)

```
[ ]: print(df_cust_categ[['categ_data__remainder__remainder__Gender_1',␣
     ↪'categ_data__remainder__remainder__Gender_2','categ_data__remainder__remainder__Gender_3']]
     ↪head(5).to_string())
```

```
   categ_data__remainder__remainder__Gender_1
categ_data__remainder__remainder__Gender_2
categ_data__remainder__remainder__Gender_3
0                                            1
0                                            0
1                                            1
0                                            0
2                                            0
1                                            0
3                                            0
1                                            0
4                                            1
0                                            0
```

Apart from above, listed below is some encoder used for non-numerical data

- Label Encoding: Assigns each category a unique integer. Best for ordered data where numbers reflect ranking.
- One-Hot Encoding: Creates binary columns for each category. Useful for nominal data without any natural order.
- Ordinal Encoding: Maps categories to integers based on order. Ideal when categories have a clear progression.
- Binary Encoding: Converts categories into binary digits across fewer columns. Helps reduce dimensionality for large sets.
- Target/Mean Encoding: Replaces categories with target variable averages. Effective but must avoid data leakage.
- Frequency/Count Encoding: Substitutes categories with their occurrence count. Simple and works well with tree-based models.

Note: These are some of the ways to pre-process data before using the same for training the model.

#End of Chapter-2