full_run_report

June 25, 2020

[1]:
```python
# Import necessary files
import numpy as np
from helperFunctions.check_merge_data import merge_check_data
from helperFunctions.clustering import *
from helperFunctions.clv import basic_clv, granular_clv, traditional_clv
from helperFunctions.cohorts import *
from helperFunctions.outlier_detection import outlier_detection
from helperFunctions.predict_customer_transaction_for_last_month import␣
↪train_model_to_predict_sales
from helperFunctions.read_data import import_data
from helperFunctions.rfmt_ihc import *
from helperFunctions.tidy_data import clean_data

# Read row files data
rowConvPath = "data/table_A_conversions.csv"
rowAttrPath = "data/table_B_attribution.csv"
rowConvData, rowAttrData = import_data(rowConvPath, rowAttrPath)

# Merge data
rowDataMerged = merge_check_data(rowConvData, rowAttrData)

# Get tidy data frame and get unique Channels
tidyUserAttr_df, uniqueChannels = clean_data(rowDataMerged)

# Save the new tidy data frame to a new CSV file
tidyUserAttr_df.to_csv("data/cleanUserAttr_df.csv")

# Get some KPI"s and insights
# Revenue over time
plot_rev_over_time(df=tidyUserAttr_df, df_gp_by="Conv_Date", df_col="Revenue",␣
↪df_agg_func="sum",
                   title="Revenue Over Time", xlab="Transaction Month and␣
↪Year", ylab="Total Revenue",
                   save_f_name="Revenue_Over_Time")
print("We can notice that revenue is always high between march and april every␣
↪year thus we conclude seasonal offers "
      "or purchasing during this period")
```

```
# Fraction of return customers
fracReturnCustomers = (tidyUserAttr_df.groupby("User_ID")[
                            "Conv_ID"].nunique() > 1).sum() / tidyUserAttr_df.
 ↪User_ID.nunique()
print("Fraction of return customers is {:.2f} %".format(fracReturnCustomers *␣
 ↪100))

# Add cohort columns to the data frame
tidyUserAttr_df = add_cohort_columns(tidyUserAttr_df, "User_ID")
print(tidyUserAttr_df.head())

# Monthly Active customers in each cohort
cohort_counts = build_time_cohort(df=tidyUserAttr_df, df_grp_by=["CohortMonth",␣
 ↪"CohortIndex"], cohort_slic="User_ID",
                                  func=pd.Series.nunique)
vis_cohort(cohort_counts, "Monthly Active customers in each cohort",␣
 ↪"Monthly_Active_customers_in_each_cohort")
print("We can notice that the first column contains the total of active cohort␣
 ↪customers in each cohort month")
print("We can notice also that April 2017 has the most number of active users␣
 ↪and cohort users")

# Retention Rate Cohort
# Get the total cohort sizes or counts from the first column of the␣
 ↪cohort_counts
cohort_sizes = cohort_counts.iloc[:, 0]
retention = cohort_counts.divide(cohort_sizes, axis=0)
# Plot the Retention Rate Cohort and exclude the first column of the total for␣
 ↪better visualization in the heat map
vis_cohort(retention, "Retention Rate Cohort", "Retention_Rate_Cohort")

# Calculate total revenue by Monthly Cohorts
rev_counts = build_time_cohort(df=tidyUserAttr_df, df_grp_by=["CohortMonth",␣
 ↪"CohortIndex"], cohort_slic="Revenue",
                               func=sum)
vis_cohort(df=rev_counts, plt_title="Total Revenue by Monthly Cohorts", frmt=".
 ↪1f",
           save_f_name="Total_Revenue_by_Monthly_Cohorts")

# Calculate CLV
# Basic CLV calculation
basic_clv(tidyUserAttr_df, "User_ID", "InvoiceMonth", "Revenue", 36)

# Granular CLV calculation
```

```
granular_clv(tidyUserAttr_df, "User_ID", "InvoiceMonth", "Revenue", "Conv_ID", 36)

# Traditional CLV
# # Calculate monthly spend per customer
traditional_clv(tidyUserAttr_df, retention, "User_ID", "InvoiceMonth", "Revenue")

# Get a snapshot of the data collection date which is exactly after the maximum date with 1 day
snapshot_date = extract_snap_date(tidyUserAttr_df)
print("Snapshot date :" + str(snapshot_date))

# RFMT IHC segmentation
# Recency - R - days since last customer transaction
# Frequency - F - number of transactions in the last 13 months
# Monetary Value - M - total spend in the last 13 months
# Tenure - T - time since the first transaction
# IHC means - Initializer, Holder and Closer average values per customer per channel

# Get RFMT data
rfmtDM = build_rfmt(df=tidyUserAttr_df, snapshot_date=snapshot_date)
print(rfmtDM.head())

# Get IHC data
ihcDM = build_ihc(df=tidyUserAttr_df, channels=uniqueChannels)
print(ihcDM.head())

# Get RFMT_IHC data
RfmtIhcDM = build_rfmt_ihc(rfmt_dm=rfmtDM, ihc_dm=ihcDM)

RfmtIhcDM.set_index('key_0', inplace=True)
print(ihcDM.head())

# Data Exploration and cleaning process
# Exploratory Data Analysis (EDA)
for x in RfmtIhcDM.columns:
    print(x + ' Column Outliers Metrics:')
    outlier_detection(RfmtIhcDM[x])
    sns.distplot(RfmtIhcDM[x])
    plt.title('Column ' + x + ' Distribution Plot Before Cleaning')
    plt.savefig("visualizations/" + x + "_Distribution_Plot_b4_clean.png", dpi=600)
    plt.show()
    print('----------------------')
```

```
# By looking visually at these data we see almost 85% of Frequency data are = 1␣
↪so it's better to drop this column
# We also notice that the MonetaryValue also suffer from outliers which has␣
↪been identified by Modified Z-Score to be
# outside of the range [24.65, 1027.34]. Resulting outlier proportion: 0.02855.

rmtihc = RfmtIhcDM[(RfmtIhcDM.MonetaryValue <= 1027.34) & (RfmtIhcDM.
↪MonetaryValue > 24.65)].drop('Frequency', axis=1)
print('Data loss = ' + str(RfmtIhcDM.shape[0] - rmtihc.shape[0]) + " which is "␣
↪+
      str(np.round((RfmtIhcDM.shape[0] - rmtihc.shape[0]) / RfmtIhcDM.shape[0]␣
↪* 100, 2)) + "%")
print(rmtihc.describe().T)

# Save the clean and tidy rmtihc data set
rmtihc.to_csv("data/clean_rmtihc.csv")

# Data distribution after dropping Frequency column, and removing outliers and␣
↪from the MonetaryValue column
for x in rmtihc.columns:
    print(x + ' Showing dist plots of data after cleaning:')
    sns.distplot(rmtihc[x])
    plt.title('Column ' + x + ' Distribution Plot After Cleaning')
    plt.savefig("visualizations/" + x + "_Distribution_Plot_after_clean.png",␣
↪dpi=600)
    plt.show()
    print('----------------------')

# Clustering (using Kmeans)
# Data preprocessing (feature transformation and scaling )
sse, clusters_labels = create_kmeans_clusters(rmtihc, 5)
plot_clusters(sse)
show_clusters_hmap(df=rmtihc, km_labels=clusters_labels, k=2)
show_clusters_hmap(df=rmtihc, km_labels=clusters_labels, k=3)
show_clusters_hmap(df=rmtihc, km_labels=clusters_labels, k=4)
print(
    "We can see from this plot that 2 or 3, or 4 clusters would be appropriate␣
↪to represent our data from the elbow method")

# Train a simple linear regression model to preidct custemer sales for the last␣
↪month
lm_model = train_model_to_predict_sales(df=tidyUserAttr_df,␣
↪channels_to_keep=uniqueChannels)
```

```
Info

<class 'pandas.core.frame.DataFrame'>
```

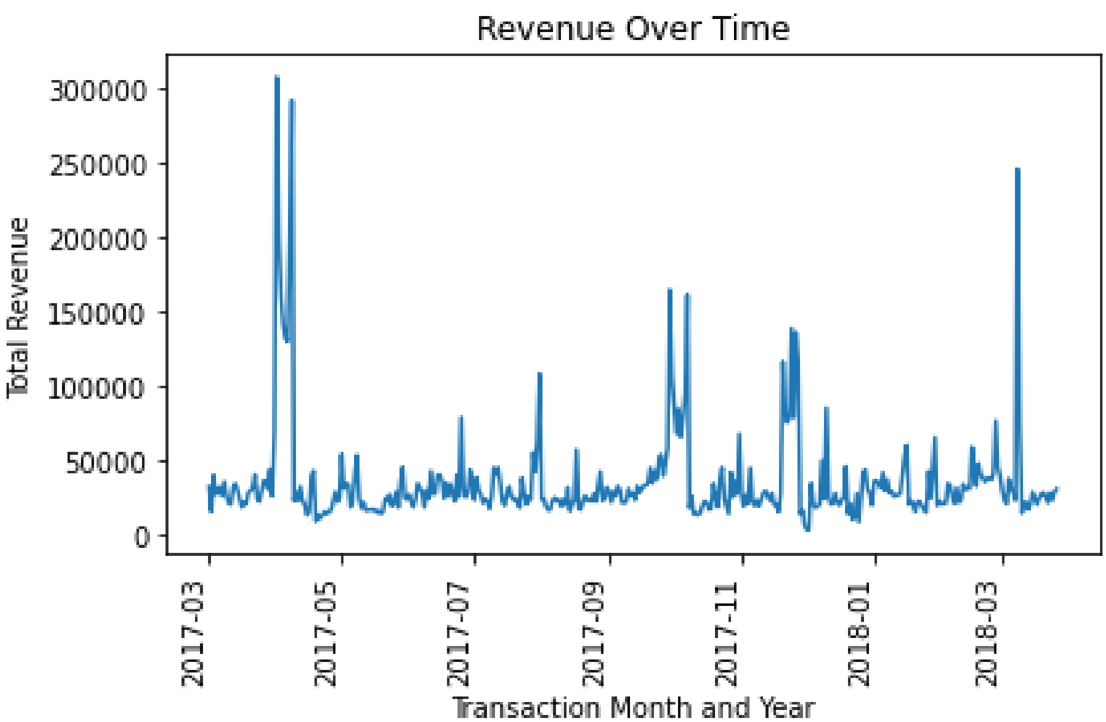
```
Int64Index: 211060 entries, 0 to 211059
Data columns (total 6 columns):
 #   Column     Non-Null Count   Dtype
---  ------     --------------   -----
 0   Conv_Date  211060 non-null  object
 1   Revenue    211060 non-null  float64
 2   User_ID    204422 non-null  object
 3   Conv_ID    211060 non-null  object
 4   Channel    211060 non-null  object
 5   IHC_Conv   211060 non-null  float64
dtypes: float64(2), object(4)
memory usage: 11.3+ MB
None

------------------------------------
Head

    Conv_Date    Revenue                                   User_ID  \
0  2017-03-06   47.00000  5094298f068196c5349d43847de5afc9125cf989
1  2017-03-02   98.00004                                       NaN
2  2017-03-02   98.00004                                       NaN
3  2017-03-02   98.00004                                       NaN
4  2017-03-02  180.35300                                       NaN

                                    Conv_ID Channel  IHC_Conv
0  881152bb20f9b73daafb99d77714f38ac702629c       H  1.000000
1  faf5c1181ea84a32237dff45ca201d2c28f19d7b       I  0.300250
2  faf5c1181ea84a32237dff45ca201d2c28f19d7b       A  0.322839
3  faf5c1181ea84a32237dff45ca201d2c28f19d7b       E  0.376911
4  b0e58a88459ece1b585ca22c93e633dc56273b83       H  1.000000

------------------------------------
Number of unique categories in each field

We Can notice that data is not tidy (rows doesn't represent observations and not
all columns are variables
e.g. All data are duplicated data except IHC, and chennel which are spreaded
across multiple rows
Conv_Date       389
Revenue       39358
User_ID       55332
Conv_ID       79615
Channel          22
IHC_Conv     119574
dtype: int64

------------------------------------
Number of missing values and its portion of the data
```

```
           count  Pct% of the data
Conv_Date      0          0.000000
Revenue        0          0.000000
User_ID     6638          3.145077
Conv_ID        0          0.000000
Channel        0          0.000000
IHC_Conv       0          0.000000
<class 'pandas.core.frame.DataFrame'>
Int64Index: 204422 entries, 0 to 211059
Data columns (total 6 columns):
 #   Column     Non-Null Count   Dtype
---  ------     --------------   -----
 0   Conv_Date  204422 non-null  object
 1   Revenue    204422 non-null  float64
 2   User_ID    204422 non-null  object
 3   Conv_ID    204422 non-null  object
 4   Channel    204422 non-null  object
 5   IHC_Conv   204422 non-null  float64
dtypes: float64(2), object(4)
memory usage: 10.9+ MB
data types before fixing :

Conv_Date     object
Revenue      float64
User_ID       object
Conv_ID       object
Channel       object
IHC_Conv     float64
dtype: object


data types after fixing :

Conv_Date    datetime64[ns]
Revenue             float64
User_ID              object
Conv_ID              object
Channel            category
IHC_Conv            float64
dtype: object

Min snapshot date:2017-03-01 00:00:00; Max snapshot date:2018-03-26 00:00:00
Data duration is about 13 months
Check how many users conversed in more than one day (if 0 means that all users
conversed in the same day)
0
First 5 values of IHC values per unique conversion ID
```

```
Channel                                          A         B         C  \
Conv_ID
0000ccb093df86fd1480a0aa5c2167233f8ab9cf  0.540098       NaN       NaN
0000ea3393004ed1e855e74f5eec5ad96270a816       NaN       NaN       NaN
00011c4ee4c3484ebaf68d328668f9c97c5eaa4f  0.549969  0.450031       NaN
00015d1120d462a27b4a58b4e3b63b3831be28f8  0.323511  0.676489       NaN
00061879cf1e7229b4957a0d31723df0d5767cf3  0.910853  0.025606  0.039688

Channel                                        D         E   F         G   H  \
Conv_ID
0000ccb093df86fd1480a0aa5c2167233f8ab9cf  0.3786       NaN NaN  0.081302 NaN
0000ea3393004ed1e855e74f5eec5ad96270a816     NaN       NaN NaN  1.000000 NaN
00011c4ee4c3484ebaf68d328668f9c97c5eaa4f     NaN       NaN NaN       NaN NaN
00015d1120d462a27b4a58b4e3b63b3831be28f8     NaN       NaN NaN       NaN NaN
00061879cf1e7229b4957a0d31723df0d5767cf3     NaN  0.013145 NaN  0.007938 NaN

Channel                                         I   J  …   M   N   O   P  \
Conv_ID                                                …
0000ccb093df86fd1480a0aa5c2167233f8ab9cf      NaN NaN  … NaN NaN NaN NaN
0000ea3393004ed1e855e74f5eec5ad96270a816      NaN NaN  … NaN NaN NaN NaN
00011c4ee4c3484ebaf68d328668f9c97c5eaa4f      NaN NaN  … NaN NaN NaN NaN
00015d1120d462a27b4a58b4e3b63b3831be28f8      NaN NaN  … NaN NaN NaN NaN
00061879cf1e7229b4957a0d31723df0d5767cf3  0.00277 NaN  … NaN NaN NaN NaN

Channel                                     Q   R   S   T   U   V
Conv_ID
0000ccb093df86fd1480a0aa5c2167233f8ab9cf NaN NaN NaN NaN NaN NaN
0000ea3393004ed1e855e74f5eec5ad96270a816 NaN NaN NaN NaN NaN NaN
00011c4ee4c3484ebaf68d328668f9c97c5eaa4f NaN NaN NaN NaN NaN NaN
00015d1120d462a27b4a58b4e3b63b3831be28f8 NaN NaN NaN NaN NaN NaN
00061879cf1e7229b4957a0d31723df0d5767cf3 NaN NaN NaN NaN NaN NaN

[5 rows x 22 columns]
Range of Sum of all IHC for each Conversion ID is below: so we can conclude that
it equals 1 for each conversion
min    1.0
max    1.0
dtype: float64
Number of complete cases channels columns (columns without missing values):
Channel
A    40484
G    36024
H    28745
I    23885
B    22078
E    15014
C     8493
K     6208
```

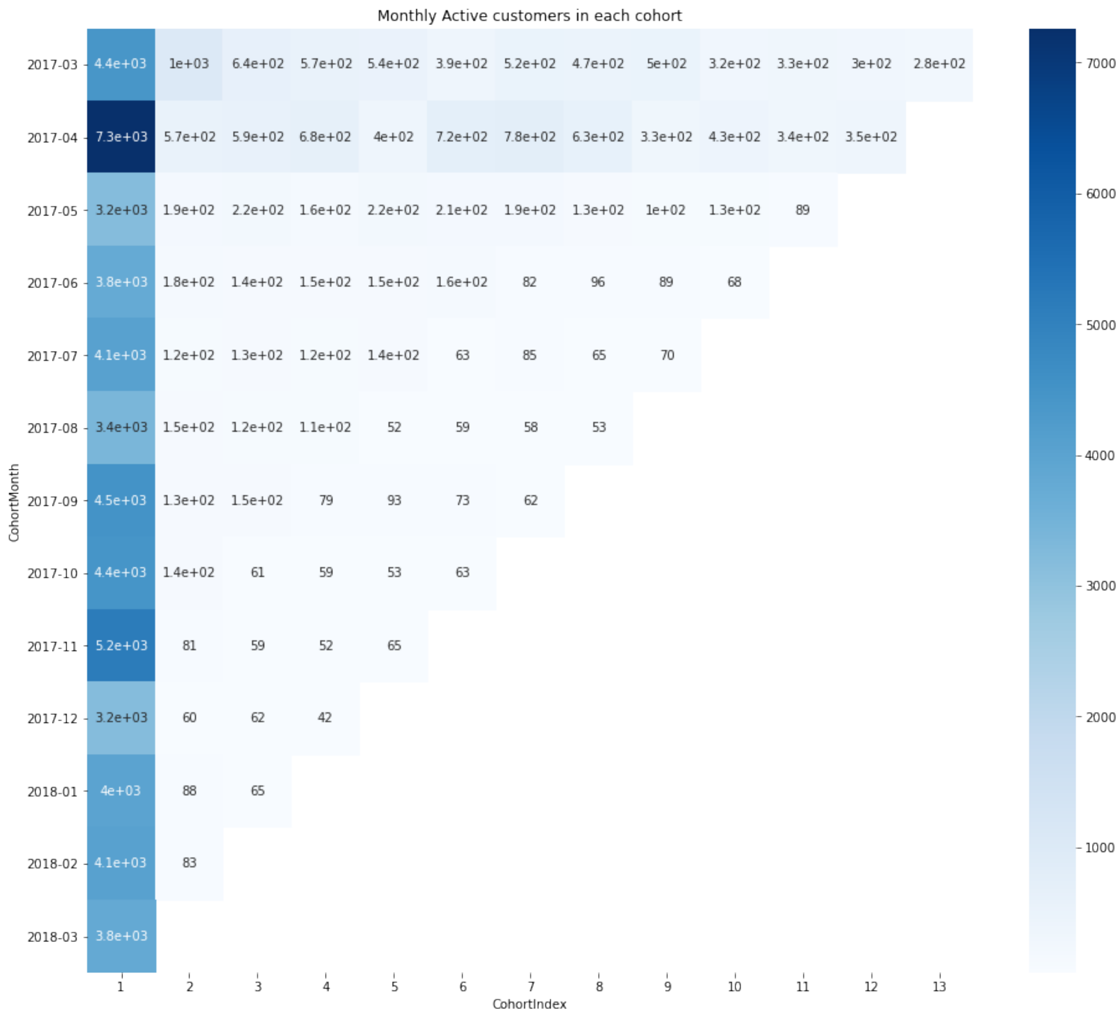
```
J     5290
L     4694
M     4545
D     2546
N     1643
F     1431
S     1365
R      749
P      636
O      313
U      124
T       75
V       43
Q       37
dtype: int64
<class 'pandas.core.frame.DataFrame'>
Index: 77319 entries, 0000ccb093df86fd1480a0aa5c2167233f8ab9cf to
ffff19f83a071ed0ea8011e09b2db089d523a54f
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   A       40484 non-null  float64
 1   B       22078 non-null  float64
 2   G       36024 non-null  float64
 3   H       28745 non-null  float64
 4   I       23885 non-null  float64
dtypes: float64(5)
memory usage: 3.5+ MB
First 5 values of the clean data without Channel, and IHC
                                           Conv_Date    Revenue  \
Conv_ID
0000ccb093df86fd1480a0aa5c2167233f8ab9cf 2017-11-27  230.97600
0000ea3393004ed1e855e74f5eec5ad96270a816 2017-03-12  135.76448
00011c4ee4c3484ebaf68d328668f9c97c5eaa4f 2017-11-25  114.50400
00015d1120d462a27b4a58b4e3b63b3831be28f8 2017-10-17   90.90000
00061879cf1e7229b4957a0d31723df0d5767cf3 2018-03-16  108.03600

User_ID
Conv_ID
0000ccb093df86fd1480a0aa5c2167233f8ab9cf
9e33e0f30f3f76b4581faea2310cce386769fe12
0000ea3393004ed1e855e74f5eec5ad96270a816
7fe7f993b2607fb0a49ddbb2b2836fd3673128a1
00011c4ee4c3484ebaf68d328668f9c97c5eaa4f
5292372b8a4f1e07c91a50e15c7d06ff3f14a7e4
00015d1120d462a27b4a58b4e3b63b3831be28f8
4e6e92b9ce6507da6c68d71871fdd572b2d845ab
00061879cf1e7229b4957a0d31723df0d5767cf3
```

```
310e3421aa1d5ff61b48cc153b460123218c0d10
First 5 values of the final clean data
                                    Conv_ID   Conv_Date    Revenue  \
0  0000ccb093df86fd1480a0aa5c2167233f8ab9cf  2017-11-27  230.97600
1  0000ea3393004ed1e855e74f5eec5ad96270a816  2017-03-12  135.76448
2  00011c4ee4c3484ebaf68d328668f9c97c5eaa4f  2017-11-25  114.50400
3  00015d1120d462a27b4a58b4e3b63b3831be28f8  2017-10-17   90.90000
4  00061879cf1e7229b4957a0d31723df0d5767cf3  2018-03-16  108.03600

                                    User_ID         A         B         G   H  \
0  9e33e0f30f3f76b4581faea2310cce386769fe12  0.540098       NaN  0.081302 NaN
1  7fe7f993b2607fb0a49ddbb2b2836fd3673128a1       NaN       NaN  1.000000 NaN
2  5292372b8a4f1e07c91a50e15c7d06ff3f14a7e4  0.549969  0.450031       NaN NaN
3  4e6e92b9ce6507da6c68d71871fdd572b2d845ab  0.323511  0.676489       NaN NaN
4  310e3421aa1d5ff61b48cc153b460123218c0d10  0.910853  0.025606  0.007938 NaN

         I
0      NaN
1      NaN
2      NaN
3      NaN
4  0.00277
```

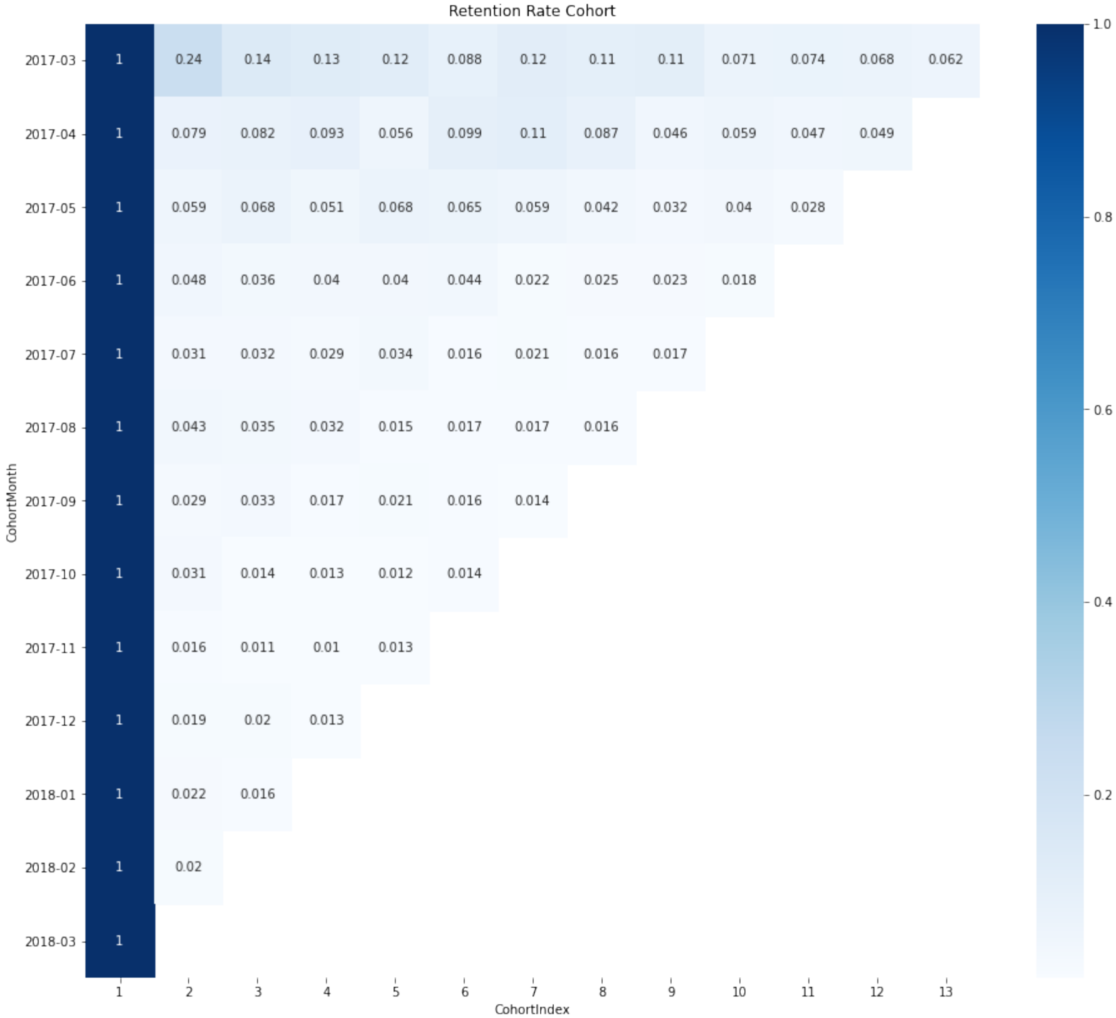


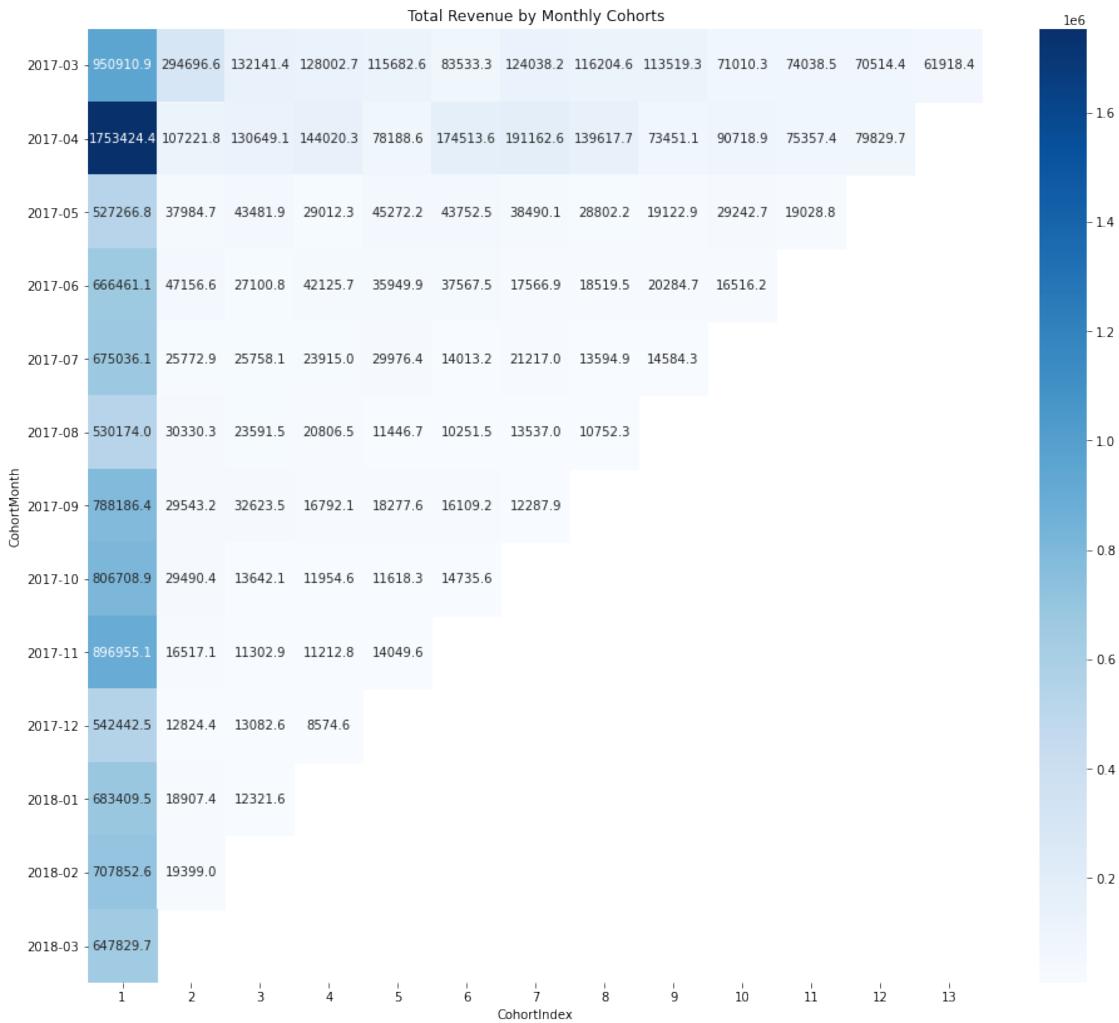
We can notice that revenue is always high between march and april every year

```
thus we conclude seasonal offers or purchasing during this period
Fraction of return customers is 16.12 %
                                    Conv_ID  Conv_Date    Revenue  \
0  0000ccb093df86fd1480a0aa5c2167233f8ab9cf 2017-11-27  230.97600
1  0000ea3393004ed1e855e74f5eec5ad96270a816 2017-03-12  135.76448
2  00011c4ee4c3484ebaf68d328668f9c97c5eaa4f 2017-11-25  114.50400
3  00015d1120d462a27b4a58b4e3b63b3831be28f8 2017-10-17   90.90000
4  00061879cf1e7229b4957a0d31723df0d5767cf3 2018-03-16  108.03600

                                    User_ID         A         B         G   H  \
0  9e33e0f30f3f76b4581faea2310cce386769fe12  0.540098       NaN  0.081302 NaN
1  7fe7f993b2607fb0a49ddbb2b2836fd3673128a1       NaN       NaN  1.000000 NaN
2  5292372b8a4f1e07c91a50e15c7d06ff3f14a7e4  0.549969  0.450031       NaN NaN
3  4e6e92b9ce6507da6c68d71871fdd572b2d845ab  0.323511  0.676489       NaN NaN
4  310e3421aa1d5ff61b48cc153b460123218c0d10  0.910853  0.025606  0.007938 NaN

         I InvoiceMonth CohortMonth InvoiceDay  CohortDay  CohortIndex  \
0      NaN   2017-11-01  2017-04-01 2017-11-27 2017-04-03            8
1      NaN   2017-03-01  2017-03-01 2017-03-12 2017-03-12            1
2      NaN   2017-11-01  2017-11-01 2017-11-25 2017-11-25            1
3      NaN   2017-10-01  2017-10-01 2017-10-17 2017-10-17            1
4  0.00277   2018-03-01  2017-03-01 2018-03-16 2017-03-22           13

   CohortDayIndex
0             239
1               1
2               1
3               1
4             360
```



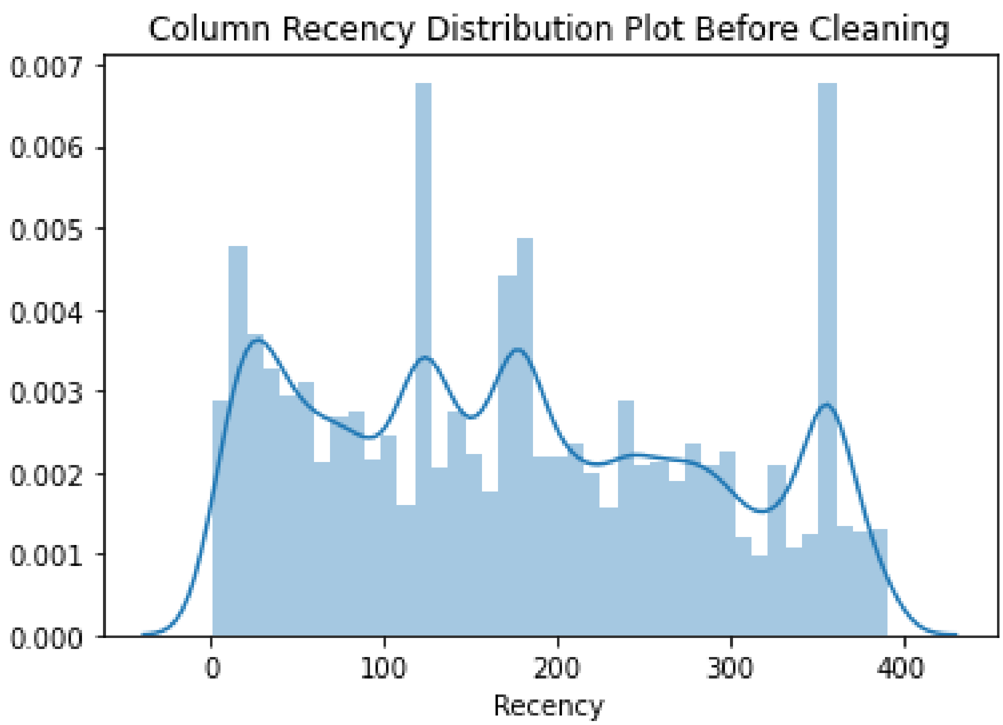
We can notice that the first column contains the total of active cohort customers in each cohort month
We can notice also that April 2017 has the most number of active users and cohort users


Retention Rate Cohort
CohortMonth
CohortIndex
2017-03
2017-04
2017-05
2017-06
2017-07
2017-08
2017-09
2017-10
2017-11
2017-12
2018-01
2018-02
2018-03
1 0.24 0.14 0.13 0.12 0.088 0.12 0.11 0.11 0.071 0.074 0.068 0.062
1 0.079 0.082 0.093 0.056 0.099 0.11 0.087 0.046 0.059 0.047 0.049
1 0.059 0.068 0.051 0.068 0.065 0.059 0.042 0.032 0.04 0.028
1 0.048 0.036 0.04 0.04 0.044 0.022 0.025 0.023 0.018
1 0.031 0.032 0.029 0.034 0.016 0.021 0.016 0.017
1 0.043 0.035 0.032 0.015 0.017 0.017 0.016
1 0.029 0.033 0.017 0.021 0.016 0.014
1 0.031 0.014 0.013 0.012 0.014
1 0.016 0.011 0.01 0.013
1 0.019 0.02 0.013
1 0.022 0.016
1 0.02
1
1 2 3 4 5 6 7 8 9 10 11 12 13
1.0 0.8 0.6 0.4 0.2

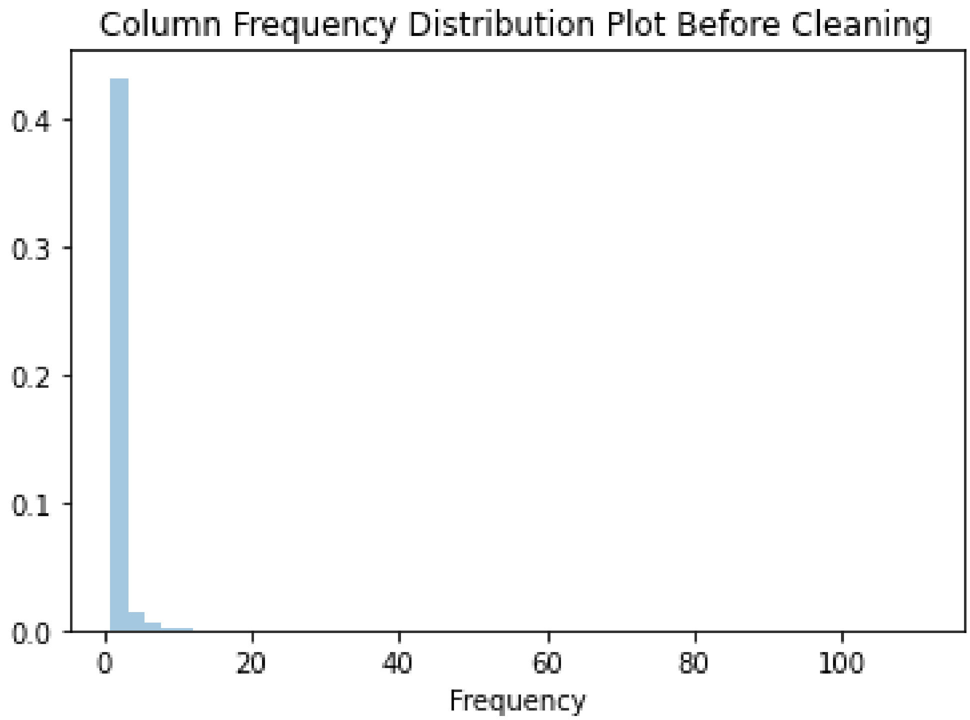



```
Average basic CLV is 6960.8 $ based on a lifespan of 36 months

Average granular CLV is 6960.8 $ based on a lifespan of 36 months

Average traditional CLV is 10.0 $ at 4.9 % retention_rate
Snapshot date :2018-03-27 00:00:00
                                          Recency  Frequency  MonetaryValue  \
User_ID
00003ce67d6b73b2d49f4036f60cb73385a9c96e      146          1        153.840
0003509d64606735e66a3d32f2a1a084f613ee4b       89          2        245.632
00035f943a8a8e176fdd5a44059b38dcc0c73f5a       38          2        833.164
0003f10010cd3dadcb7182ed7b0abf5166393e91      287          1        121.808
0003fc733e4ff3bfb295f2c10c7077fb0763ebcc       28          1        108.720

                                          Tenure
```

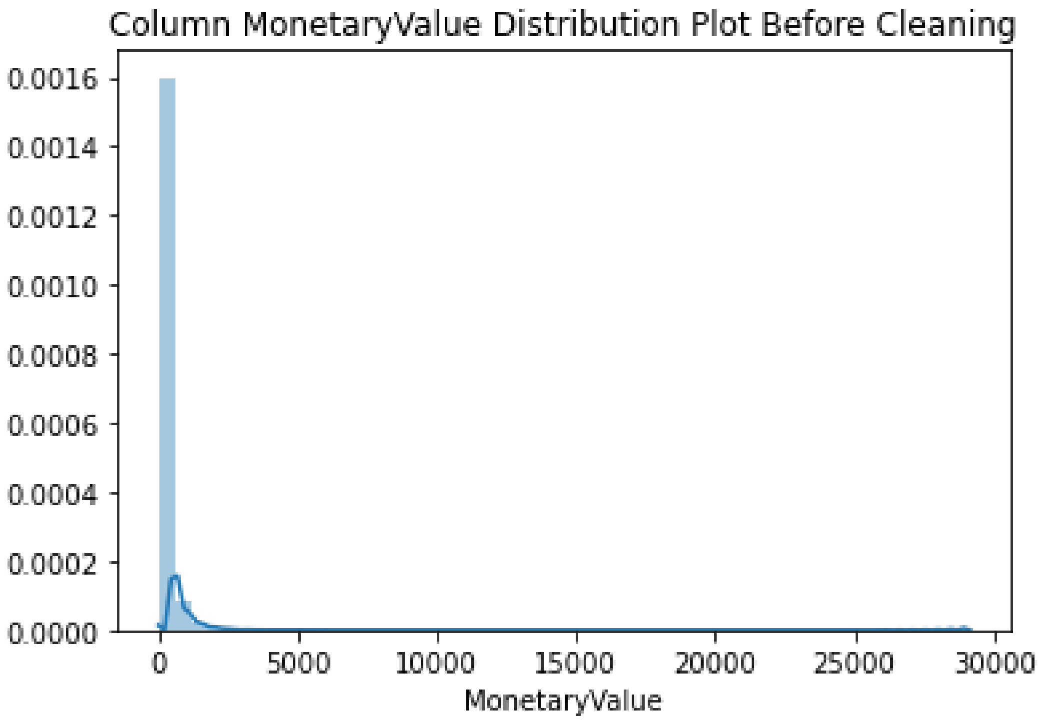
```
User_ID
00003ce67d6b73b2d49f4036f60cb73385a9c96e     146
0003509d64606735e66a3d32f2a1a084f613ee4b     142
00035f943a8a8e176fdd5a44059b38dcc0c73f5a     275
0003f10010cd3dadcb7182ed7b0abf5166393e91     287
0003fc733e4ff3bfb295f2c10c7077fb0763ebcc      28
                                                 A         G         H    I  \
User_ID
00003ce67d6b73b2d49f4036f60cb73385a9c96e  0.302910  0.155305  0.170037  0.0
0003509d64606735e66a3d32f2a1a084f613ee4b  0.000000  1.000000  0.000000  0.5
00035f943a8a8e176fdd5a44059b38dcc0c73f5a  0.231401  0.280338  0.319150  0.0
0003f10010cd3dadcb7182ed7b0abf5166393e91  0.000000  1.000000  0.000000  0.0
0003fc733e4ff3bfb295f2c10c7077fb0763ebcc  1.000000  0.000000  0.000000  0.0

                                            B
User_ID
00003ce67d6b73b2d49f4036f60cb73385a9c96e  0.0
0003509d64606735e66a3d32f2a1a084f613ee4b  0.0
00035f943a8a8e176fdd5a44059b38dcc0c73f5a  0.0
0003f10010cd3dadcb7182ed7b0abf5166393e91  0.0
0003fc733e4ff3bfb295f2c10c7077fb0763ebcc  0.0
                                                 A         G         H    I  \
User_ID
00003ce67d6b73b2d49f4036f60cb73385a9c96e  0.302910  0.155305  0.170037  0.0
0003509d64606735e66a3d32f2a1a084f613ee4b  0.000000  1.000000  0.000000  0.5
00035f943a8a8e176fdd5a44059b38dcc0c73f5a  0.231401  0.280338  0.319150  0.0
0003f10010cd3dadcb7182ed7b0abf5166393e91  0.000000  1.000000  0.000000  0.0
0003fc733e4ff3bfb295f2c10c7077fb0763ebcc  1.000000  0.000000  0.000000  0.0

                                            B
User_ID
00003ce67d6b73b2d49f4036f60cb73385a9c96e  0.0
0003509d64606735e66a3d32f2a1a084f613ee4b  0.0
00035f943a8a8e176fdd5a44059b38dcc0c73f5a  0.0
0003f10010cd3dadcb7182ed7b0abf5166393e91  0.0
0003fc733e4ff3bfb295f2c10c7077fb0763ebcc  0.0
Recency Column Outliers Metrics:
Outlier detection: Z-Score identified outliers outside of the range [1, 391].
Resulting outlier proportion: 0.0.
Outlier detection: Modified Z-Score identified outliers outside of the range [1,
391]. Resulting outlier proportion: 0.0.
Outlier detection: Isolation Forest identified outliers outside of the range
[18, 362]. Resulting outlier proportion: 0.03687.
Outlier detection: Isolation Forest (new version) identified outliers outside of
the range [19, 358]. Resulting outlier proportion: 0.0587.
```



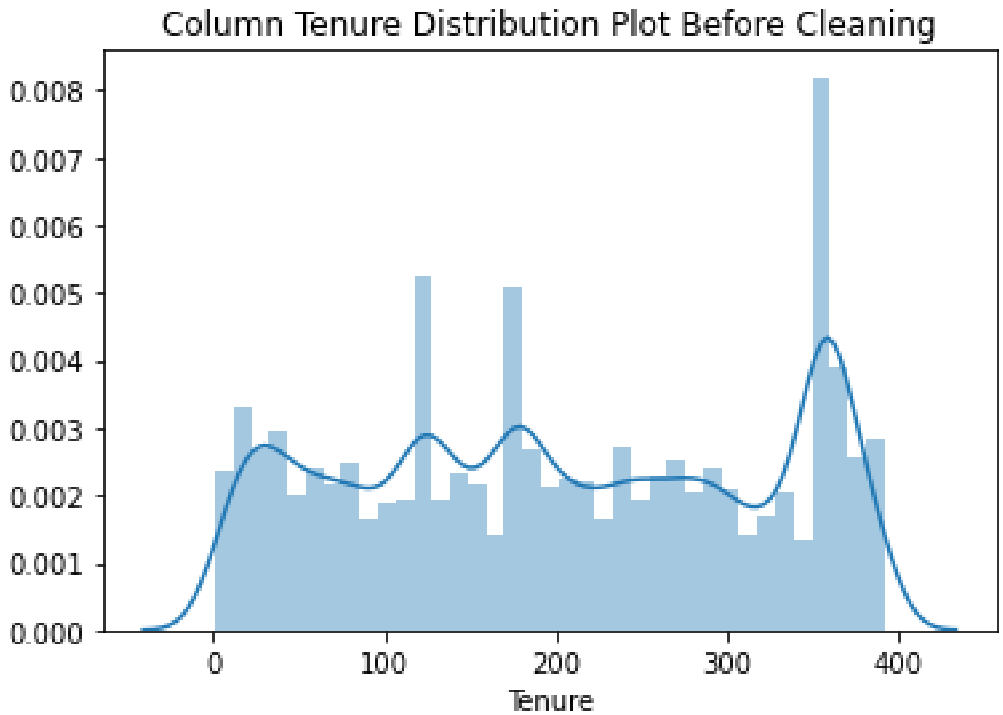

```
----------------------
Frequency Column Outliers Metrics:
Outlier detection: Z-Score identified outliers outside of the range [1, 6].
Resulting outlier proportion: 0.01327.
Outlier detection: Modified Z-Score identified outliers outside of the range [1,
4]. Resulting outlier proportion: 0.03206.
Outlier detection: Isolation Forest identified outliers outside of the range [1,
2]. Resulting outlier proportion: 0.08702.
Outlier detection: Isolation Forest (new version) identified outliers outside of
the range [1, 1]. Resulting outlier proportion: 0.16117.

C:\Users\ahmed\anaconda3\envs\h_ams\lib\site-
packages\seaborn\distributions.py:369: UserWarning: Default bandwidth for data
is 0; skipping density estimation.
  warnings.warn(msg, UserWarning)
```



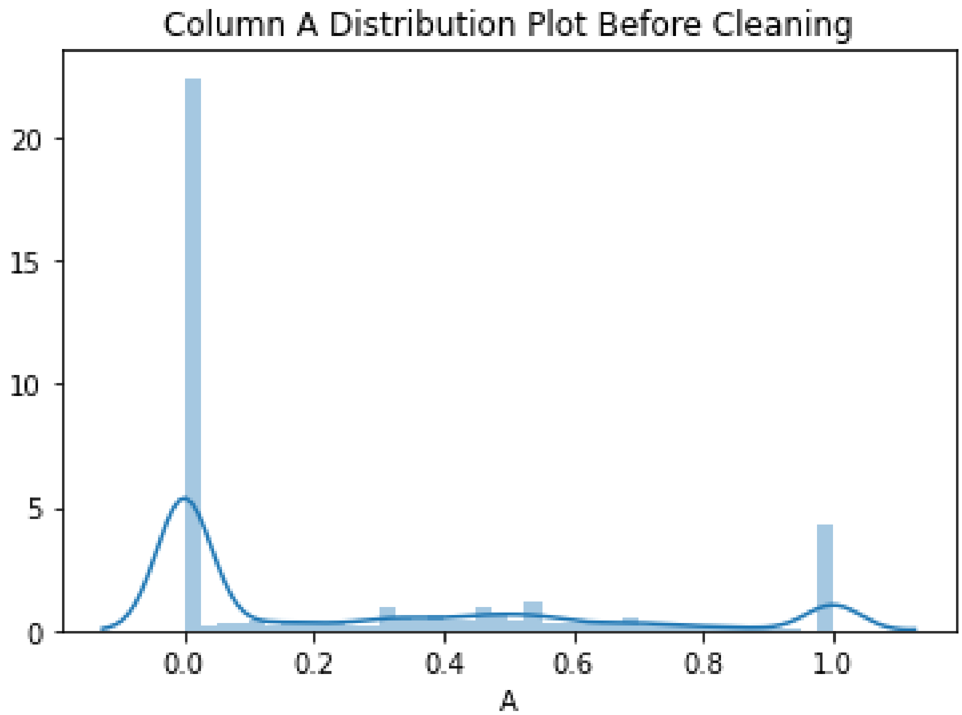

```
----------------------
MonetaryValue Column Outliers Metrics:
Outlier detection: Z-Score identified outliers outside of the range [24.65,
1466.18]. Resulting outlier proportion: 0.01261.
Outlier detection: Modified Z-Score identified outliers outside of the range
[24.65, 1027.34]. Resulting outlier proportion: 0.02855.
Outlier detection: Isolation Forest identified outliers outside of the range
[50.04, 556.95]. Resulting outlier proportion: 0.07986.
Outlier detection: Isolation Forest (new version) identified outliers outside of
the range [64.31, 394.63]. Resulting outlier proportion: 0.13061.
```



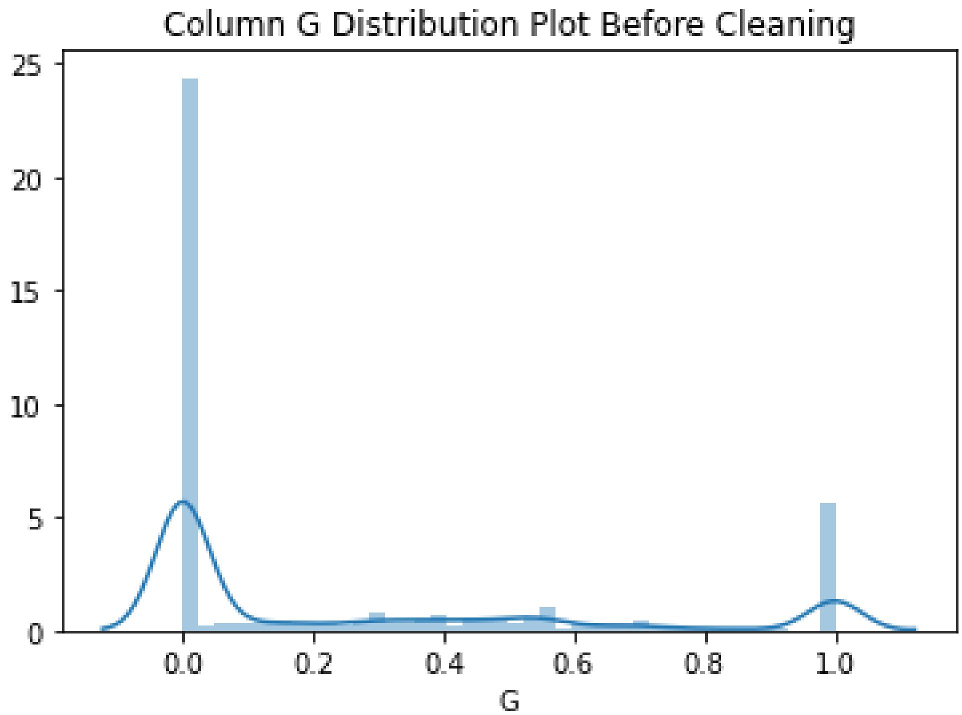

```
----------------------
Tenure Column Outliers Metrics:
Outlier detection: Z-Score identified outliers outside of the range [1, 391].
Resulting outlier proportion: 0.0.
Outlier detection: Modified Z-Score identified outliers outside of the range [1,
391]. Resulting outlier proportion: 0.0.
Outlier detection: Isolation Forest identified outliers outside of the range
[18, 368]. Resulting outlier proportion: 0.05897.
Outlier detection: Isolation Forest (new version) identified outliers outside of
the range [114, 361]. Resulting outlier proportion: 0.0756.
```



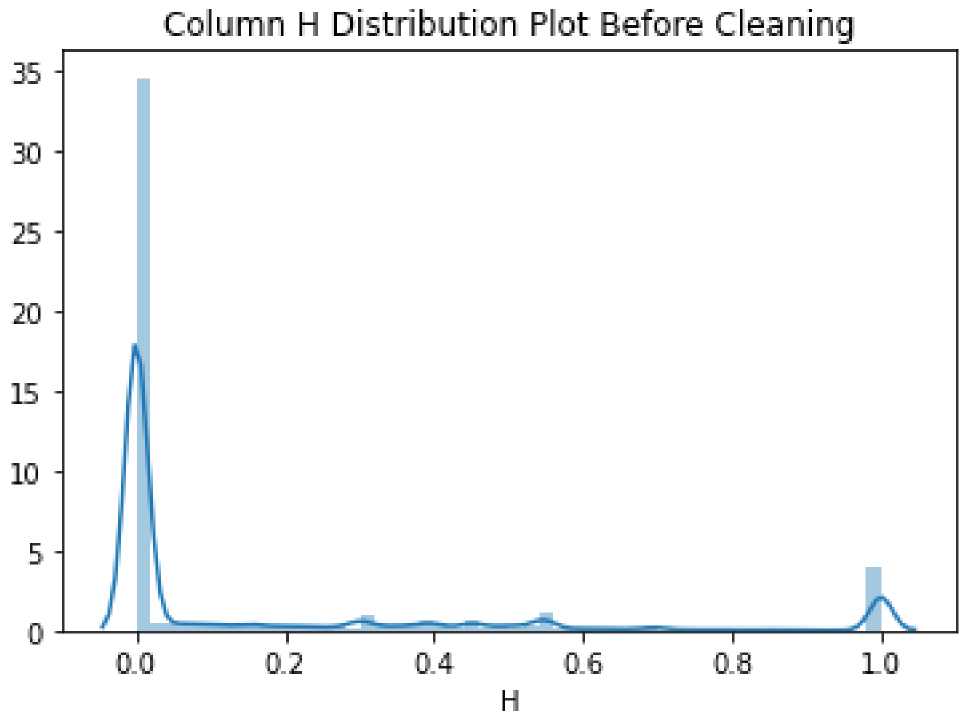

```
----------------------
A Column Outliers Metrics:
Outlier detection: Z-Score identified outliers outside of the range [0.0, 1.0].
Resulting outlier proportion: 0.0.
Outlier detection: Modified Z-Score identified outliers outside of the range
[0.0, 1.0]. Resulting outlier proportion: 0.0.
Outlier detection: Isolation Forest identified outliers outside of the range
[0.0, 1.0]. Resulting outlier proportion: 0.0.
Outlier detection: Isolation Forest (new version) identified outliers outside of
the range [0.0, 0.0]. Resulting outlier proportion: 0.4465.
```



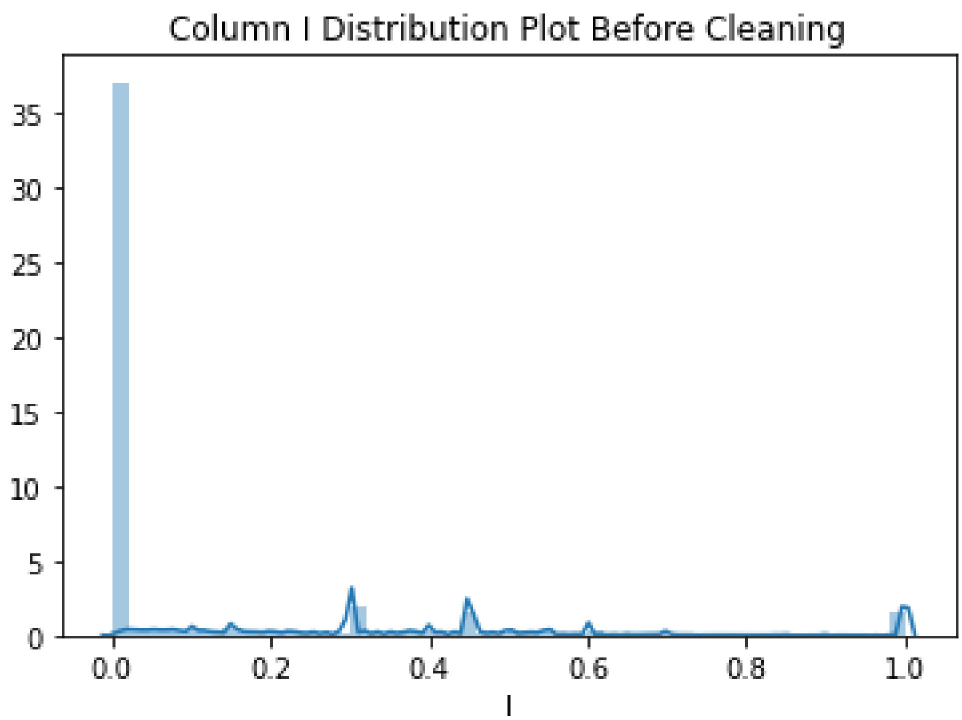

```
----------------------
G Column Outliers Metrics:
Outlier detection: Z-Score identified outliers outside of the range [0.0, 1.0].
Resulting outlier proportion: 0.0.
Outlier detection: Modified Z-Score identified outliers outside of the range
[0.0, 1.0]. Resulting outlier proportion: 0.0.
Outlier detection: Isolation Forest identified outliers outside of the range
[0.0, 1.0]. Resulting outlier proportion: 0.0.
Outlier detection: Isolation Forest (new version) identified outliers outside of
the range [0.0, 0.0]. Resulting outlier proportion: 0.42836.
```



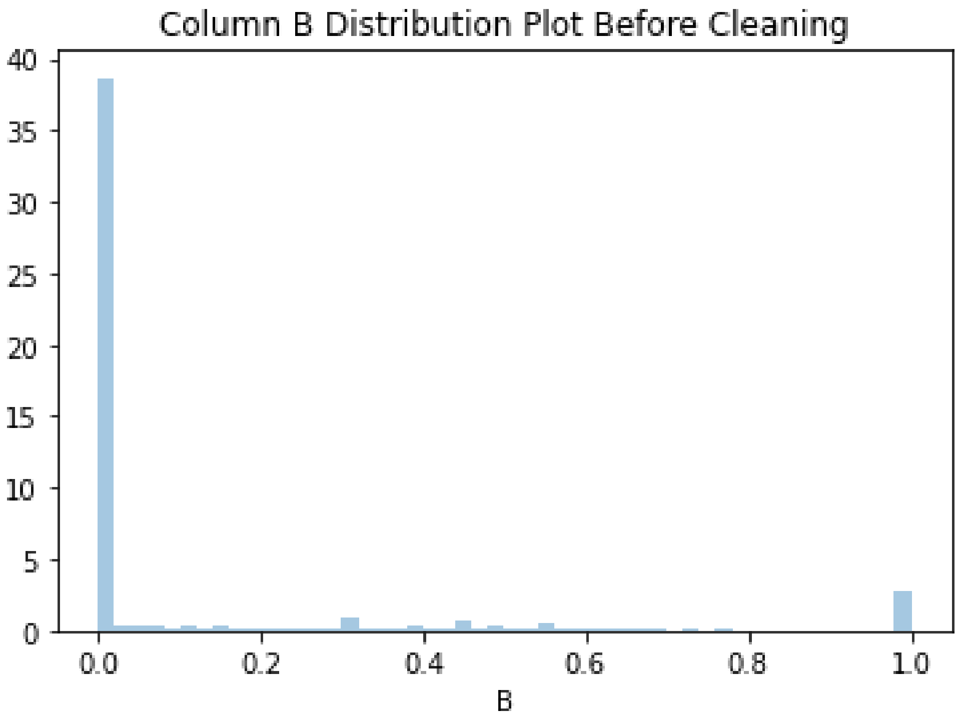

```
----------------------
H Column Outliers Metrics:
Outlier detection: Z-Score identified outliers outside of the range [0.0, 1.0].
Resulting outlier proportion: 0.0.
Outlier detection: Modified Z-Score identified outliers outside of the range
[0.0, 1.0]. Resulting outlier proportion: 0.0.
Outlier detection: Isolation Forest identified outliers outside of the range
[0.0, 1.0]. Resulting outlier proportion: 0.0.
Outlier detection: Isolation Forest (new version) identified outliers outside of
the range [0.0, 0.0]. Resulting outlier proportion: 0.32269.
```



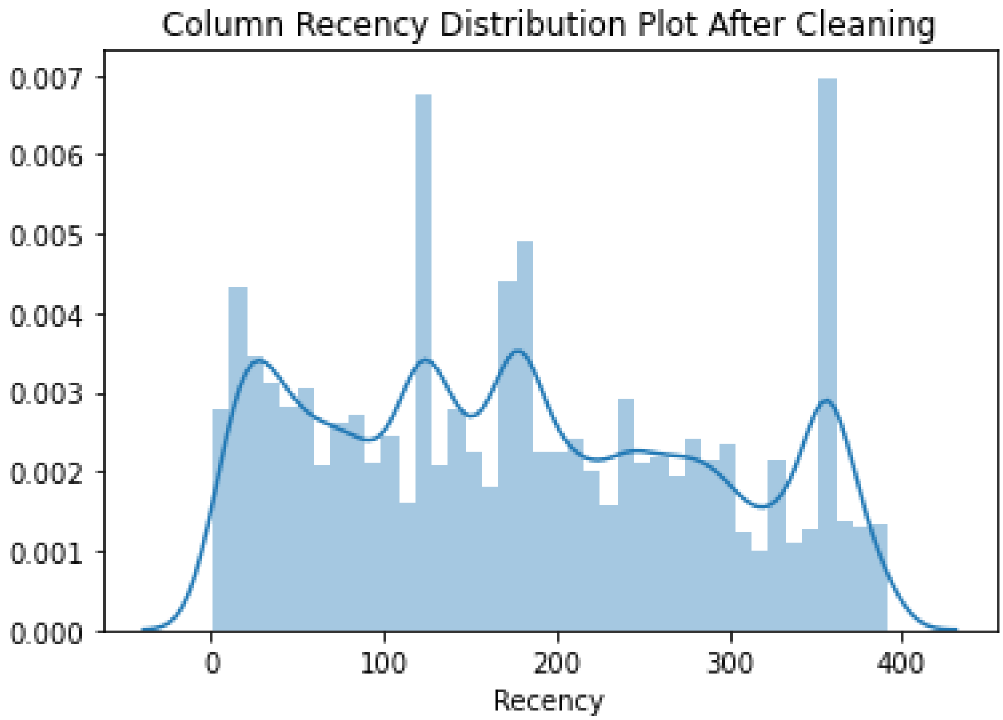

```
----------------------
I Column Outliers Metrics:
Outlier detection: Z-Score identified outliers outside of the range [0.0, 0.92].
Resulting outlier proportion: 0.03222.
Outlier detection: Modified Z-Score identified outliers outside of the range
[0.0, 0.85]. Resulting outlier proportion: 0.03371.
Outlier detection: Isolation Forest identified outliers outside of the range
[0.0, 0.5]. Resulting outlier proportion: 0.07106.
Outlier detection: Isolation Forest (new version) identified outliers outside of
the range [0.0, 0.45]. Resulting outlier proportion: 0.09774.
```



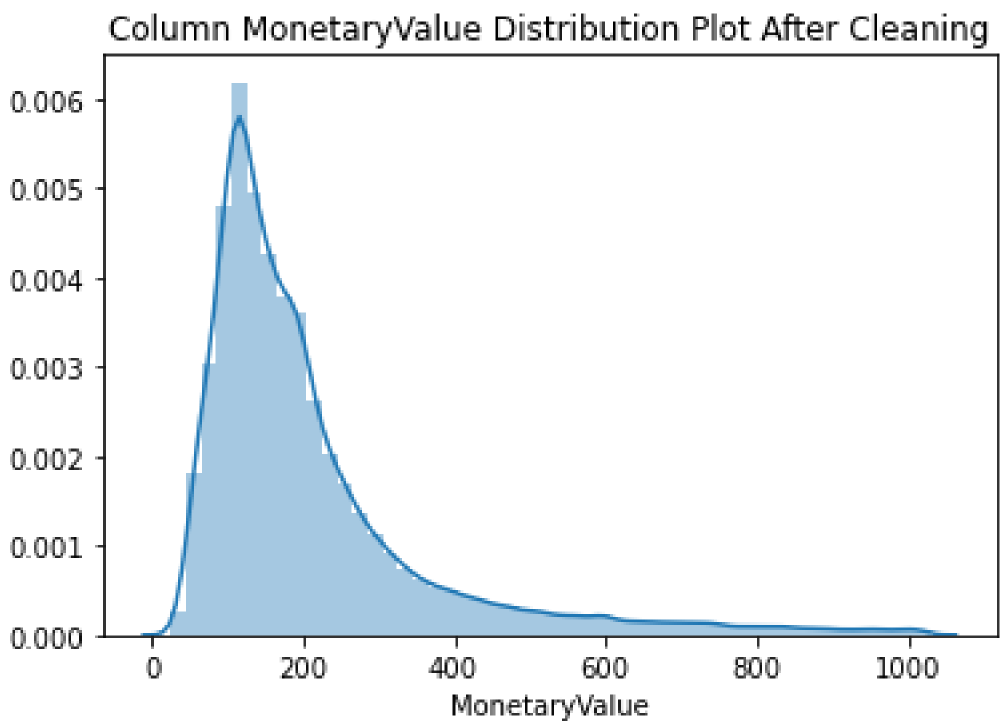

```
----------------------
B Column Outliers Metrics:
Outlier detection: Z-Score identified outliers outside of the range [0.0, 1.0].
Resulting outlier proportion: 0.0.
Outlier detection: Modified Z-Score identified outliers outside of the range
[0.0, 0.95]. Resulting outlier proportion: 0.05425.
Outlier detection: Isolation Forest identified outliers outside of the range
[0.0, 1.0]. Resulting outlier proportion: 0.00067.
Outlier detection: Isolation Forest (new version) identified outliers outside of
the range [0.0, 0.0]. Resulting outlier proportion: 0.2309.

C:\Users\ahmed\anaconda3\envs\h_ams\lib\site-
packages\seaborn\distributions.py:369: UserWarning: Default bandwidth for data
is 0; skipping density estimation.
  warnings.warn(msg, UserWarning)
```



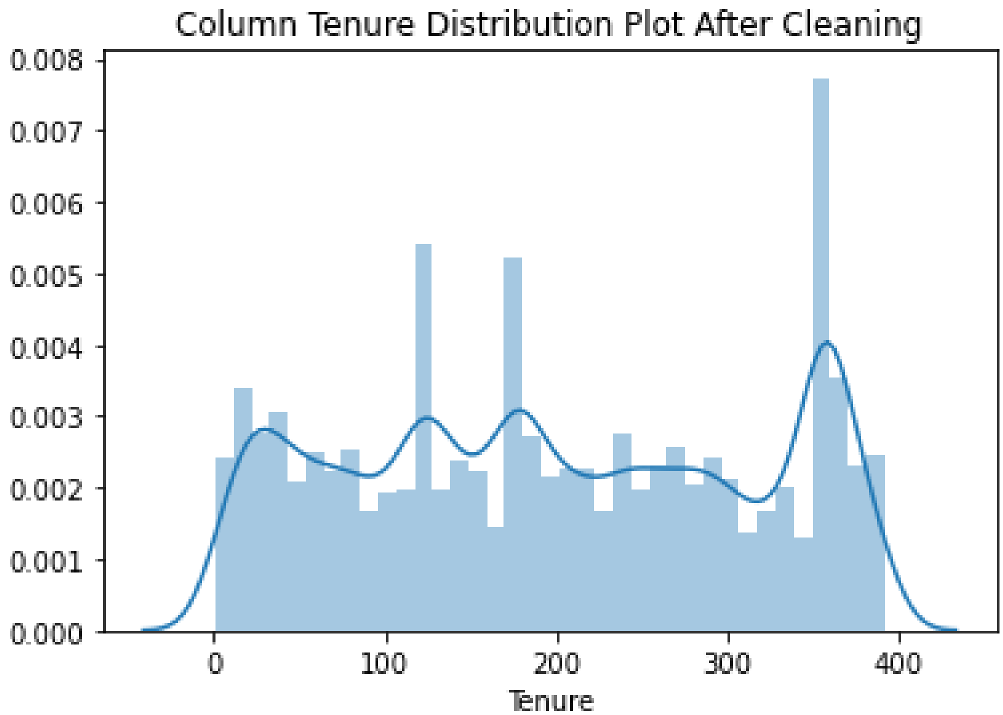

```
----------------------
Data loss = 1581 which is 2.86%
                 count        mean         std        min      25%     50%  \
Recency        53751.0  178.118249  111.076491    1.00000   83.000  173.00
MonetaryValue  53751.0  212.718513  161.261046   24.94088  113.708  163.58
Tenure         53751.0  197.663095  116.117253    1.00000  102.000  188.00
A              53751.0    0.247917    0.349569    0.00000    0.000    0.00
G              53751.0    0.248176    0.362732    0.00000    0.000    0.00
H              53751.0    0.159429    0.304614    0.00000    0.000    0.00
I              53751.0    0.108294    0.234280    0.00000    0.000    0.00
B              53751.0    0.113523    0.265070    0.00000    0.000    0.00

                      75%      max
Recency        269.000000   391.00
MonetaryValue  245.780000  1027.34
Tenure         301.000000   391.00
A                0.473228     1.00
G                0.458626     1.00
H                0.159849     1.00
I                0.022532     1.00
B                0.000000     1.00
Recency Showing dist plots of data after cleaning:
```



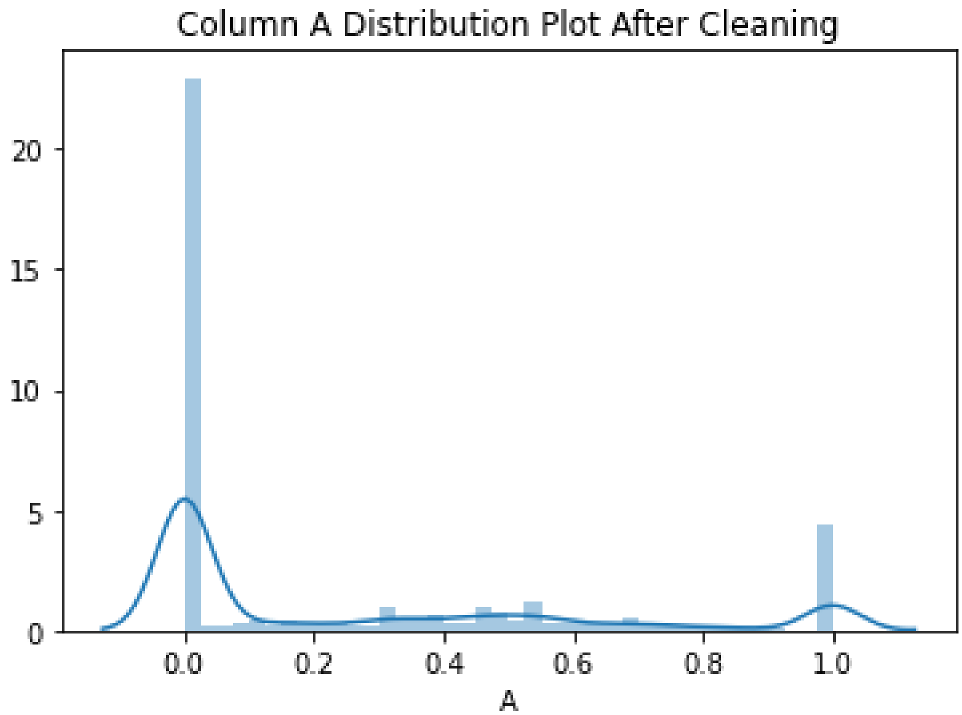

```
----------------------
MonetaryValue Showing dist plots of data after cleaning:
```



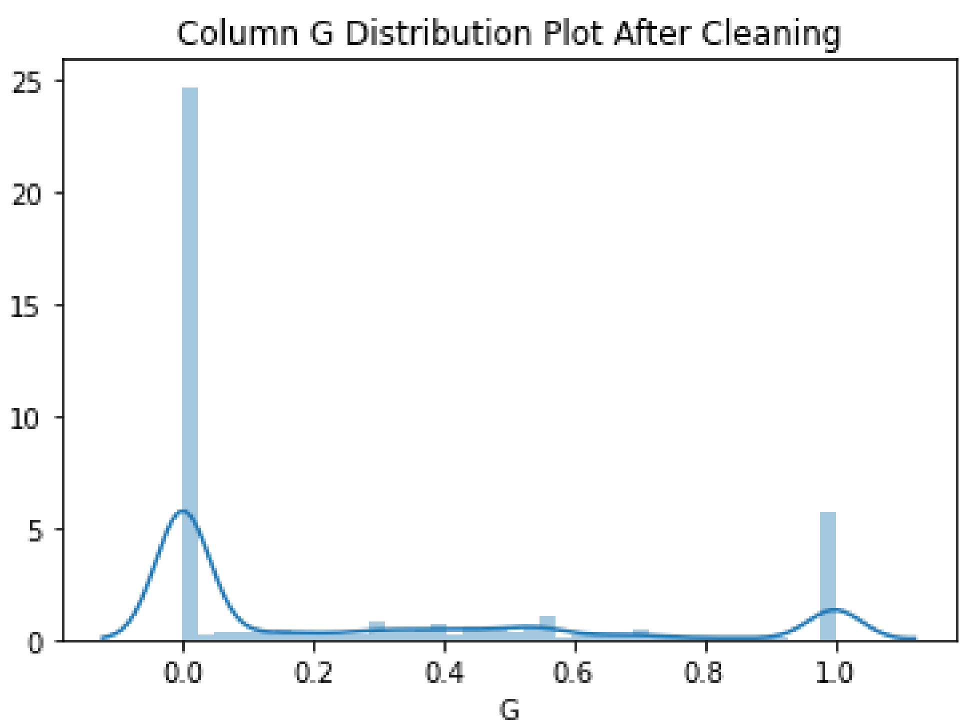
Tenure Showing dist plots of data after cleaning:



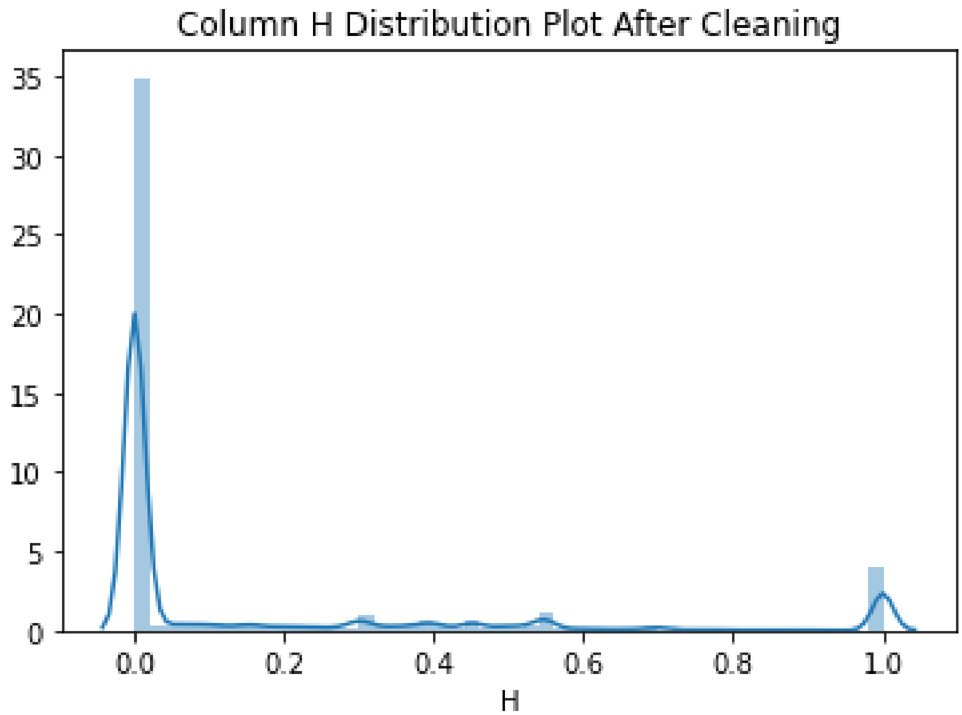
A Showing dist plots of data after cleaning:



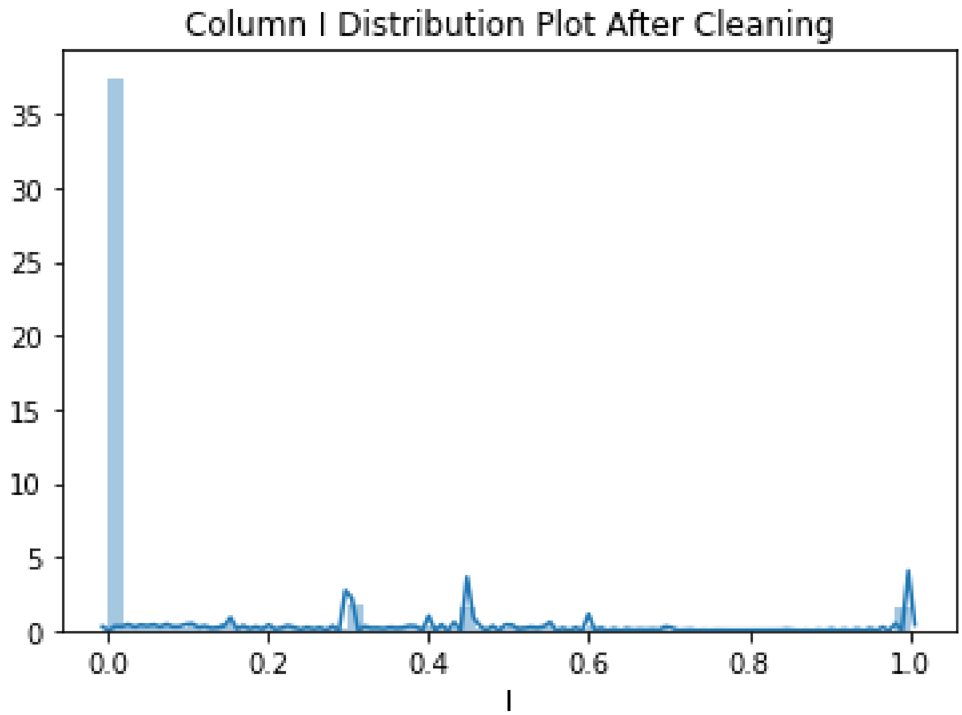

```
----------------------
G Showing dist plots of data after cleaning:
```



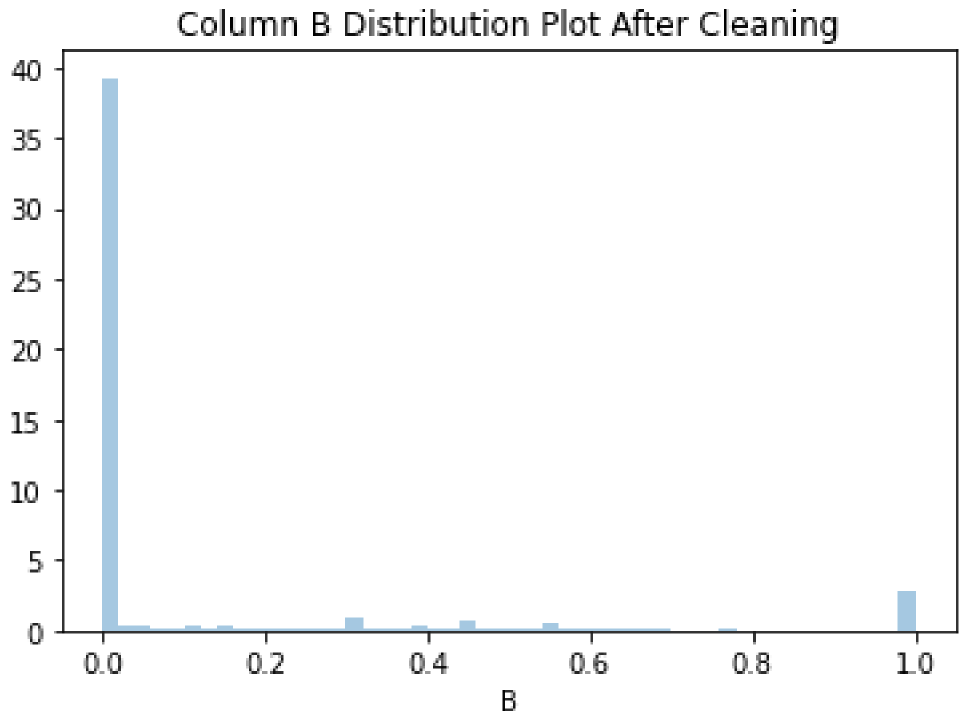
```
----------------------
H Showing dist plots of data after cleaning:
```

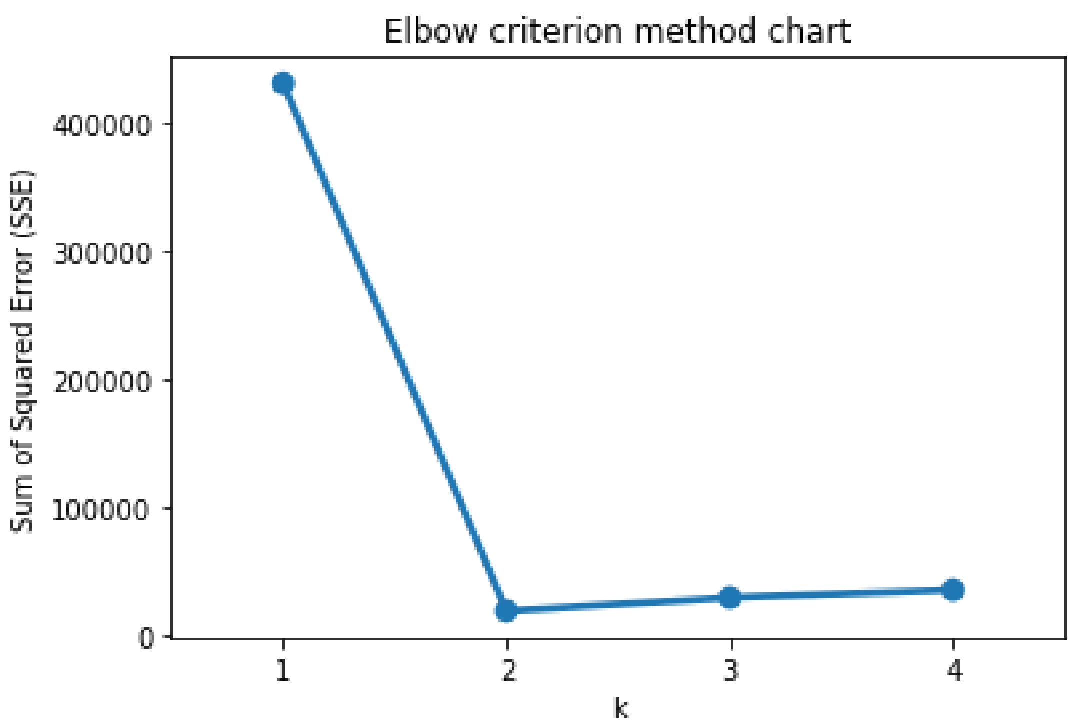


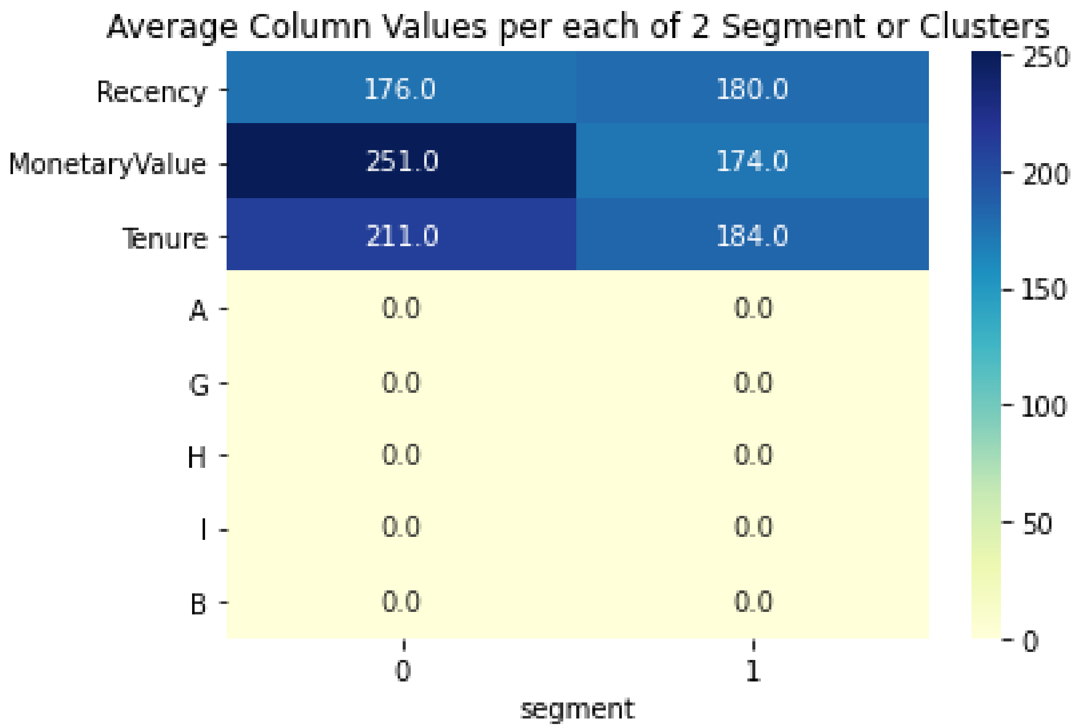

```
----------------------
I Showing dist plots of data after cleaning:
```

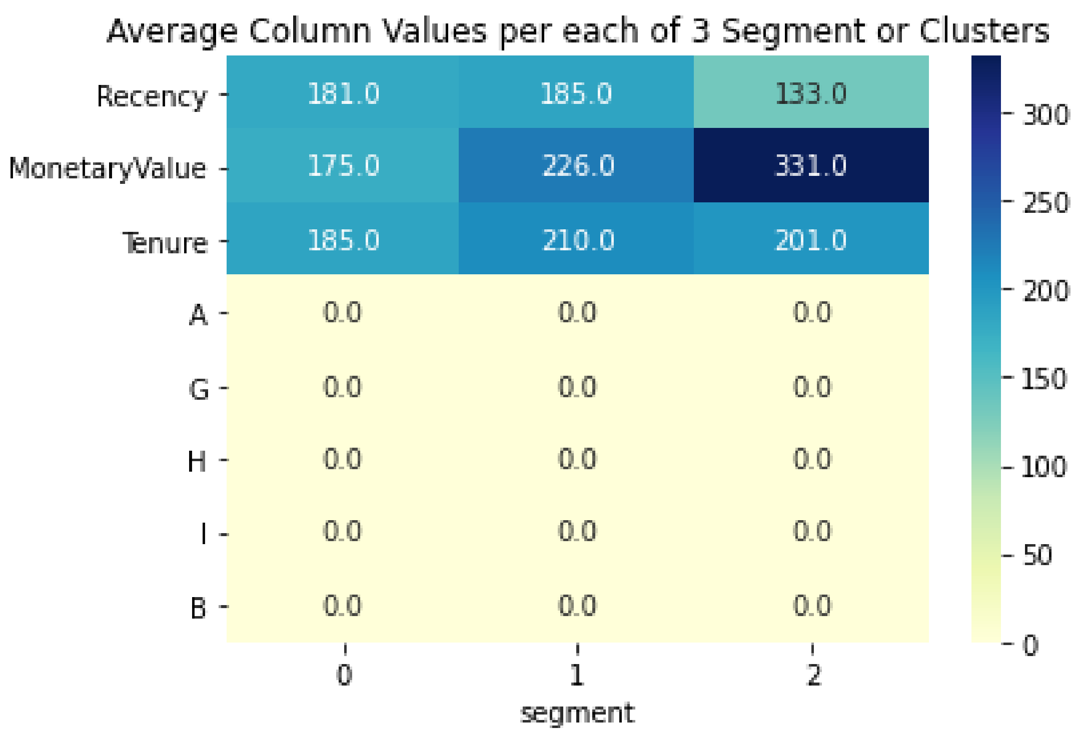


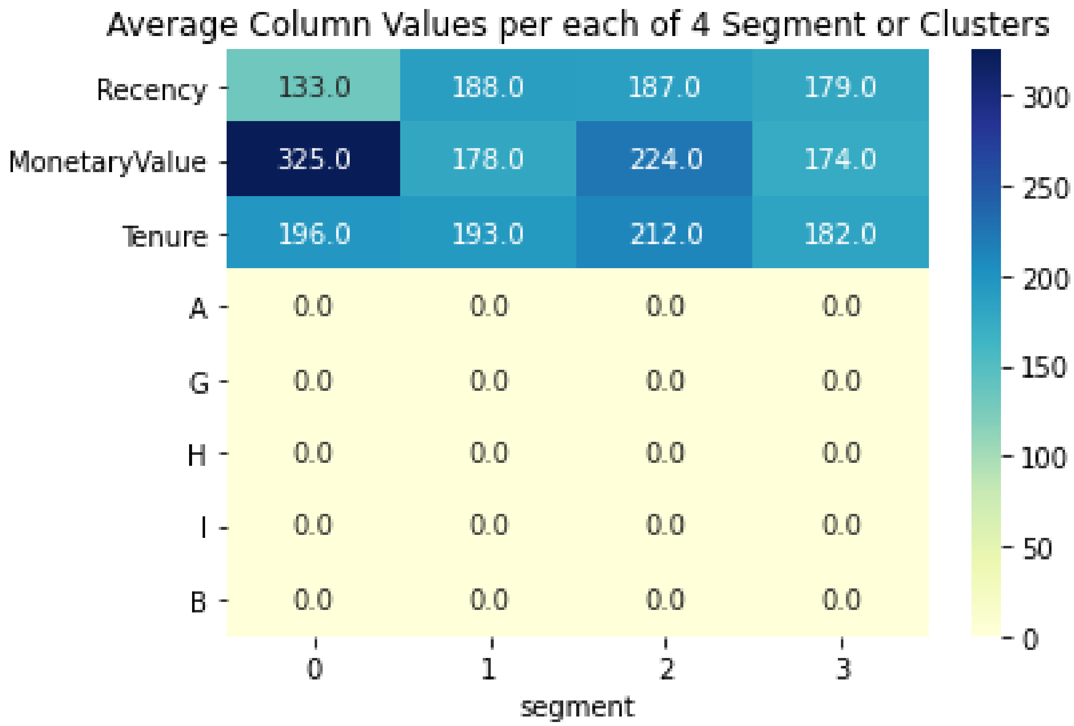

```
----------------------
B Showing dist plots of data after cleaning:

C:\Users\ahmed\anaconda3\envs\h_ams\lib\site-
packages\seaborn\distributions.py:369: UserWarning: Default bandwidth for data
is 0; skipping density estimation.
  warnings.warn(msg, UserWarning)
```


Column B Distribution Plot After Cleaning
40
35
30
25
20
15
10
5
0
0.0
0.2
0.4
0.6
0.8
1.0
B


Elbow criterion method chart
Sum of Squared Error (SSE)
400000
300000
200000
100000
0
1
2
3
4
k


Average Column Values per each of 2 Segment or Clusters
Recency
MonetaryValue
Tenure
A
G
H
I
B
176.0
180.0
251.0
174.0
211.0
184.0
0.0
0
1
segment
250
200
150
100
50
0


Average Column Values per each of 3 Segment or Clusters
Recency
MonetaryValue
Tenure
A
G
H
I
B
181.0
185.0
133.0
175.0
226.0
331.0
185.0
210.0
201.0
0.0
0
1
2
segment
300
250
200
150
100
50
0




```
We can see from this plot that 2 or 3, or 4 clusters would be appropriate to
represent our data from the elbow method
RMSE train: 0.1464754378467411; RMSE test: 0.15721296117534553
MAE train: 0.0392831706098676, MAE test: 0.0397795373372464

Although it's an extremely simple and easy model , it achieved a very good
results


Get model coefficients:

Recency         -0.000314
MonetaryValue    0.000110
Tenure           0.000267
A               -0.000924
G               -0.004147
H               -0.002282
I               -0.002848
B                0.002229
dtype: float64
```

[ ]: