Code - Telecom Churn

In [1]:

```python
import pandas as pd

# #Reading the dataset

cust_data_df = pd.read_csv('Telecom_customer churn.csv')
print(cust_data_df.head())

print(cust_data_df.head())
```

```
   rev_Mean  mou_Mean  totmrc_Mean  da_Mean  ovrmou_Mean  ovrrev_Mean  \
0   23.9975    219.25       22.500   0.2475         0.00          0.0   
1   57.4925    482.75       37.425   0.2475        22.75          9.1   
2   16.9900     10.25       16.990   0.0000         0.00          0.0   
3   38.0000      7.50       38.000   0.0000         0.00          0.0   
4   55.2300    570.50       71.980   0.0000         0.00          0.0   

   vceovr_Mean  datovr_Mean  roam_Mean  change_mou  ...  forgntvl  ethnic  \
0          0.0          0.0        0.0     -157.25  ...       0.0       N   
1          9.1          0.0        0.0      532.25  ...       0.0       Z   
2          0.0          0.0        0.0       -4.25  ...       0.0       N   
3          0.0          0.0        0.0       -1.50  ...       0.0       U   
4          0.0          0.0        0.0       38.50  ...       0.0       I   

   kid0_2  kid3_5  kid6_10  kid11_15  kid16_17  creditcd  eqpdays  Customer_ID  
0       U       U        U         U         U         Y    361.0      1000001  
1       U       U        U         U         U         Y    240.0      1000002  
2       U       Y        U         U         U         Y   1504.0      1000003  
3       Y       U        U         U         U         Y   1812.0      1000004  
4       U       U        U         U         U         Y    434.0      1000005  

[5 rows x 100 columns]
   rev_Mean  mou_Mean  totmrc_Mean  da_Mean  ovrmou_Mean  ovrrev_Mean  \
0   23.9975    219.25       22.500   0.2475         0.00          0.0   
1   57.4925    482.75       37.425   0.2475        22.75          9.1   
2   16.9900     10.25       16.990   0.0000         0.00          0.0   
3   38.0000      7.50       38.000   0.0000         0.00          0.0   
4   55.2300    570.50       71.980   0.0000         0.00          0.0   

   vceovr_Mean  datovr_Mean  roam_Mean  change_mou  ...  forgntvl  ethnic  \
0          0.0          0.0        0.0     -157.25  ...       0.0       N   
1          9.1          0.0        0.0      532.25  ...       0.0       Z   
2          0.0          0.0        0.0       -4.25  ...       0.0       N   
3          0.0          0.0        0.0       -1.50  ...       0.0       U   
4          0.0          0.0        0.0       38.50  ...       0.0       I   

   kid0_2  kid3_5  kid6_10  kid11_15  kid16_17  creditcd  eqpdays  Customer_ID  
0       U       U        U         U         U         Y    361.0      1000001  
1       U       U        U         U         U         Y    240.0      1000002  
2       U       Y        U         U         U         Y   1504.0      1000003  
3       Y       U        U         U         U         Y   1812.0      1000004  
4       U       U        U         U         U         Y    434.0      1000005  

[5 rows x 100 columns]
```

In [2]:

```python
#Printing basic information about the dataset
print(f'The dataset has {cust_data_df.shape[0]} rows and {cust_data_df.shape[1]} columns.')
```

```python
print ("Number of pepole who stay at the telecom company: "+
       str(cust_data_df[(cust_data_df['churn'] ==0) ].count()[1]))
print ("Number of pepole who leave the telecom company: "+
       str(cust_data_df[(cust_data_df['churn'] ==1) ].count()[1]))
```

```
The dataset has 100000 rows and 100 columns.
Number of pepole who stay at the telecom company: 50326
Number of pepole who leave the telecom company: 49317
```

In [3]:

```python
#Inspecting datatypes of each column
print(cust_data_df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 100 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   rev_Mean          99643 non-null   float64
 1   mou_Mean          99643 non-null   float64
 2   totmrc_Mean       99643 non-null   float64
 3   da_Mean           99643 non-null   float64
 4   ovrmou_Mean       99643 non-null   float64
 5   ovrrev_Mean       99643 non-null   float64
 6   vceovr_Mean       99643 non-null   float64
 7   datovr_Mean       99643 non-null   float64
 8   roam_Mean         99643 non-null   float64
 9   change_mou        99109 non-null   float64
 10  change_rev        99109 non-null   float64
 11  drop_vce_Mean     100000 non-null  float64
 12  drop_dat_Mean     100000 non-null  float64
 13  blck_vce_Mean     100000 non-null  float64
 14  blck_dat_Mean     100000 non-null  float64
 15  unan_vce_Mean     100000 non-null  float64
 16  unan_dat_Mean     100000 non-null  float64
 17  plcd_vce_Mean     100000 non-null  float64
 18  plcd_dat_Mean     100000 non-null  float64
 19  recv_vce_Mean     100000 non-null  float64
 20  recv_sms_Mean     100000 non-null  float64
 21  comp_vce_Mean     100000 non-null  float64
 22  comp_dat_Mean     100000 non-null  float64
 23  custcare_Mean     100000 non-null  float64
 24  ccrndmou_Mean     100000 non-null  float64
 25  cc_mou_Mean       100000 non-null  float64
 26  inonemin_Mean     100000 non-null  float64
 27  threeway_Mean     100000 non-null  float64
 28  mou_cvce_Mean     100000 non-null  float64
 29  mou_cdat_Mean     100000 non-null  float64
 30  mou_rvce_Mean     100000 non-null  float64
 31  owylis_vce_Mean   100000 non-null  float64
 32  mouowylisv_Mean   100000 non-null  float64
 33  iwylis_vce_Mean   100000 non-null  float64
 34  mouiwylisv_Mean   100000 non-null  float64
 35  peak_vce_Mean     100000 non-null  float64
 36  peak_dat_Mean     100000 non-null  float64
 37  mou_peav_Mean     100000 non-null  float64
 38  mou_pead_Mean     100000 non-null  float64
 39  opk_vce_Mean      100000 non-null  float64
 40  opk_dat_Mean      100000 non-null  float64
 41  mou_opkv_Mean     100000 non-null  float64
 42  mou_opkd_Mean     100000 non-null  float64
 43  drop_blk_Mean     100000 non-null  float64
 44  attempt_Mean      100000 non-null  float64
 45  complete_Mean     100000 non-null  float64
 46  callfwdv_Mean     100000 non-null  float64
 47  callwait_Mean     100000 non-null  float64
 48  churn             100000 non-null  int64
 49  months            100000 non-null  int64
 50  uniqsubs          100000 non-null  int64
 51  actvsubs          100000 non-null  int64
 52  new_cell          100000 non-null  object
 53  crclscod          100000 non-null  object
 54  asl_flag          100000 non-null  object
 55  totcalls          100000 non-null  int64
 56  totmou            100000 non-null  float64
```

```
 57  totrev           100000 non-null  float64
 58  adjrev           100000 non-null  float64
 59  adjmou           100000 non-null  float64
 60  adjqty           100000 non-null  int64
 61  avgrev           100000 non-null  float64
 62  avgmou           100000 non-null  float64
 63  avgqty           100000 non-null  float64
 64  avg3mou          100000 non-null  int64
 65  avg3qty          100000 non-null  int64
 66  avg3rev          100000 non-null  int64
 67  avg6mou          97161 non-null   float64
 68  avg6qty          97161 non-null   float64
 69  avg6rev          97161 non-null   float64
 70  prizm_social_one 92612 non-null   object
 71  area             99960 non-null   object
 72  dualband         99999 non-null   object
 73  refurb_new       99999 non-null   object
 74  hnd_price        99153 non-null   float64
 75  phones           99999 non-null   float64
 76  models           99999 non-null   float64
 77  hnd_webcap       89811 non-null   object
 78  truck            98268 non-null   float64
 79  rv               98268 non-null   float64
 80  ownrent          66294 non-null   object
 81  lor              69810 non-null   float64
 82  dwlltype         68091 non-null   object
 83  marital          98268 non-null   object
 84  adults           76981 non-null   float64
 85  infobase         77921 non-null   object
 86  income           74564 non-null   float64
 87  numbcars         50634 non-null   float64
 88  HHstatin         62077 non-null   object
 89  dwllsize         61692 non-null   object
 90  forgntvl         98268 non-null   float64
 91  ethnic           98268 non-null   object
 92  kid0_2           98268 non-null   object
 93  kid3_5           98268 non-null   object
 94  kid6_10          98268 non-null   object
 95  kid11_15         98268 non-null   object
 96  kid16_17         98268 non-null   object
 97  creditcd         98268 non-null   object
 98  eqpdays          99999 non-null   float64
 99  Customer_ID      100000 non-null  int64
dtypes: float64(69), int64(10), object(21)
memory usage: 76.3+ MB
None
```

In [4]:

```python
#Dropping Customer ID column because it is unique and it is not contributing to the task
cust_data_df.drop("Customer_ID", axis=1, inplace=True)
```

In [5]:

```python
#Finding null values in the dataset
print(cust_data_df.isnull().sum())
```

```
rev_Mean          357
mou_Mean          357
totmrc_Mean       357
da_Mean           357
ovrmou_Mean       357
                 ...
kid6_10          1732
kid11_15         1732
kid16_17         1732
creditcd         1732
eqpdays             1
Length: 99, dtype: int64
```

In [6]:
```
#Finding null values in the dataset
print(cust_data_df.isnull().sum())
```

```
rev_Mean          357
mou_Mean          357
totmrc_Mean       357
da_Mean           357
ovrmou_Mean       357
                 ...
kid6_10          1732
kid11_15         1732
kid16_17         1732
creditcd         1732
eqpdays             1
Length: 99, dtype: int64
```

In [7]:
```
#Getting the columns having NA values and then checking which columns are categorical
#and numerical
print('Printing the Missing values statistics-')
op = cust_data_df.isnull().sum().sort_values(ascending = False).head(43)
miss_per = (op/len(cust_data_df))*100

# Percentage of missing values
temp = pd.DataFrame({'No. missing values': op, '% of missing data': miss_per.values})
print(temp)
```

```
Printing the Missing values statistics-
                    No. missing values  % of missing data
numbcars                          49366             49.366
dwllsize                          38308             38.308
HHstatin                          37923             37.923
ownrent                           33706             33.706
dwlltype                          31909             31.909
lor                               30190             30.190
income                            25436             25.436
adults                            23019             23.019
infobase                          22079             22.079
hnd_webcap                        10189             10.189
prizm_social_one                   7388              7.388
avg6qty                            2839              2.839
avg6rev                            2839              2.839
avg6mou                            2839              2.839
kid6_10                            1732              1.732
kid16_17                           1732              1.732
rv                                 1732              1.732
kid3_5                             1732              1.732
marital                            1732              1.732
creditcd                           1732              1.732
kid11_15                           1732              1.732
forgntvl                           1732              1.732
ethnic                             1732              1.732
kid0_2                             1732              1.732
truck                              1732              1.732
change_rev                          891              0.891
change_mou                          891              0.891
hnd_price                           847              0.847
totmrc_Mean                         357              0.357
da_Mean                             357              0.357
ovrmou_Mean                         357              0.357
vceovr_Mean                         357              0.357
ovrrev_Mean                         357              0.357
datovr_Mean                         357              0.357
roam_Mean                           357              0.357
mou_Mean                            357              0.357
rev_Mean                            357              0.357
area                                 40              0.040
models                                1              0.001
phones                                1              0.001
refurb_new                            1              0.001
dualband                              1              0.001
eqpdays                               1              0.001
```

In [8]:
```python
#Printing the column names that have missing values in them

print('Columns having null values-')
columns_having_null_values = cust_data_df.columns[cust_data_df.isnull().any()]
print(columns_having_null_values.values)
```

```
Columns having null values-
['rev_Mean' 'mou_Mean' 'totmrc_Mean' 'da_Mean' 'ovrmou_Mean' 'ovrrev_Mean'
 'vceovr_Mean' 'datovr_Mean' 'roam_Mean' 'change_mou' 'change_rev'
 'avg6mou' 'avg6qty' 'avg6rev' 'prizm_social_one' 'area' 'dualband'
 'refurb_new' 'hnd_price' 'phones' 'models' 'hnd_webcap' 'truck' 'rv'
 'ownrent' 'lor' 'dwlltype' 'marital' 'adults' 'infobase' 'income'
 'numbcars' 'HHstatin' 'dwllsize' 'forgntvl' 'ethnic' 'kid0_2' 'kid3_5'
 'kid6_10' 'kid11_15' 'kid16_17' 'creditcd' 'eqpdays']
```

In [9]:
```python
def get_categorical_columns(dataframe):
```

```
  """
     Function that identifies and returns categorical columns in dataset
  """
  categorical_columns = []
  for col in dataframe.columns:
    if dataframe[col].dtypes=='object':
      categorical_columns.append(col)
  return categorical_columns

def get_numerical_columns(dataframe):
  """
    Function that identifies and returns numerical columns in the dataset

  """
  numerical_columns = []

  for col in dataframe.columns:
    if dataframe[col].dtypes !='object':
      numerical_columns.append(col)
  return numerical_columns

def get_intersection_cont_values(dataframe, columns_input):
  """
    Function that identifies categorical and numerical columns that have missing
    values

  """


  vals = get_numerical_columns(dataframe)
  num_res = []
  for col in columns_input:
      if col in vals:
          num_res.append(col)

  vals = get_categorical_columns(dataframe)
  cat_res = []

  for col in columns_input:
      if col in vals:
          cat_res.append(col)

  return num_res, cat_res

#Getting the categorical and numerical columns that have missing values

numerical_null_columns, categorical_null_columns = get_intersection_cont_values(cust_data_df,
                                                     columns_having_null_values)

print('Numerical columns with null values in them-')
print(numerical_null_columns)

print('\nCategorical columns with missing values-')
print(categorical_null_columns)
```

```
Numerical columns with null values in them-
['rev_Mean', 'mou_Mean', 'totmrc_Mean', 'da_Mean', 'ovrmou_Mean', 'ovrrev_Mean', 'vceovr_Mean',
'datovr_Mean', 'roam_Mean', 'change_mou', 'change_rev', 'avg6mou', 'avg6qty', 'avg6rev', 'hnd_pr
ice', 'phones', 'models', 'truck', 'rv', 'lor', 'adults', 'income', 'numbcars', 'forgntvl', 'eqp
days']

Categorical columns with missing values-
['prizm_social_one', 'area', 'dualband', 'refurb_new', 'hnd_webcap', 'ownrent', 'dwlltype', 'mar
ital', 'infobase', 'HHstatin', 'dwllsize', 'ethnic', 'kid0_2', 'kid3_5', 'kid6_10', 'kid11_15',
'kid16_17', 'creditcd']
```

In [10]:

```python
#Handling Missing Data for numerical values
#Since the data is left skewed, we will fill up the missing values with mean values

for cols in numerical_null_columns:
    cust_data_df[cols] = cust_data_df[cols].fillna(cust_data_df[cols].mean())
```

In [11]:

```python
#checking for null values after impuding
print(cust_data_df[get_numerical_columns(cust_data_df)].isnull().sum())
```

```
rev_Mean       0
mou_Mean       0
totmrc_Mean    0
da_Mean        0
ovrmou_Mean    0
              ..
adults         0
income         0
numbcars       0
forgntvl       0
eqpdays        0
Length: 78, dtype: int64
```

In [12]:

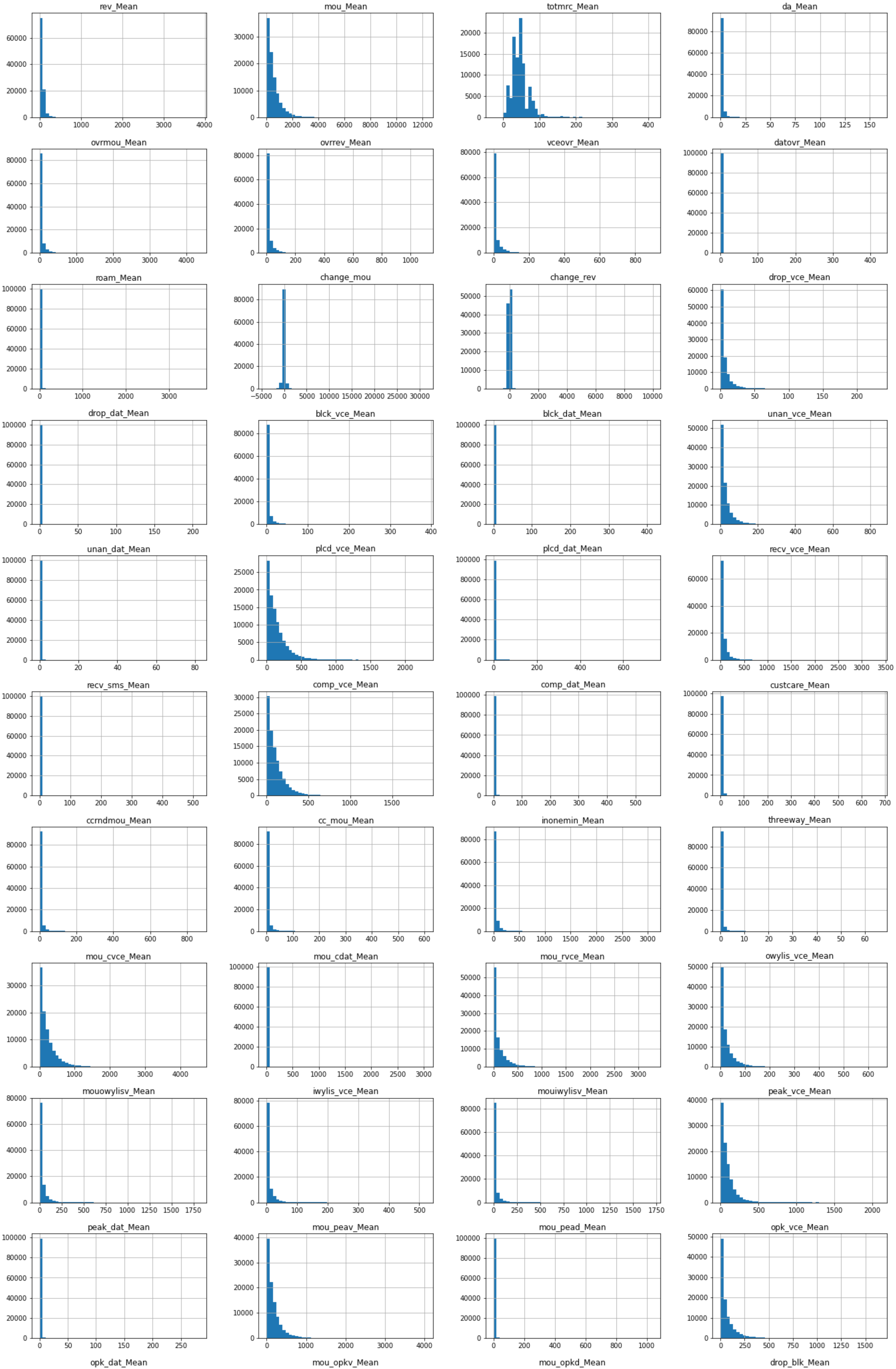
```python
#Description of Dataframe
print(cust_data_df.describe())
```

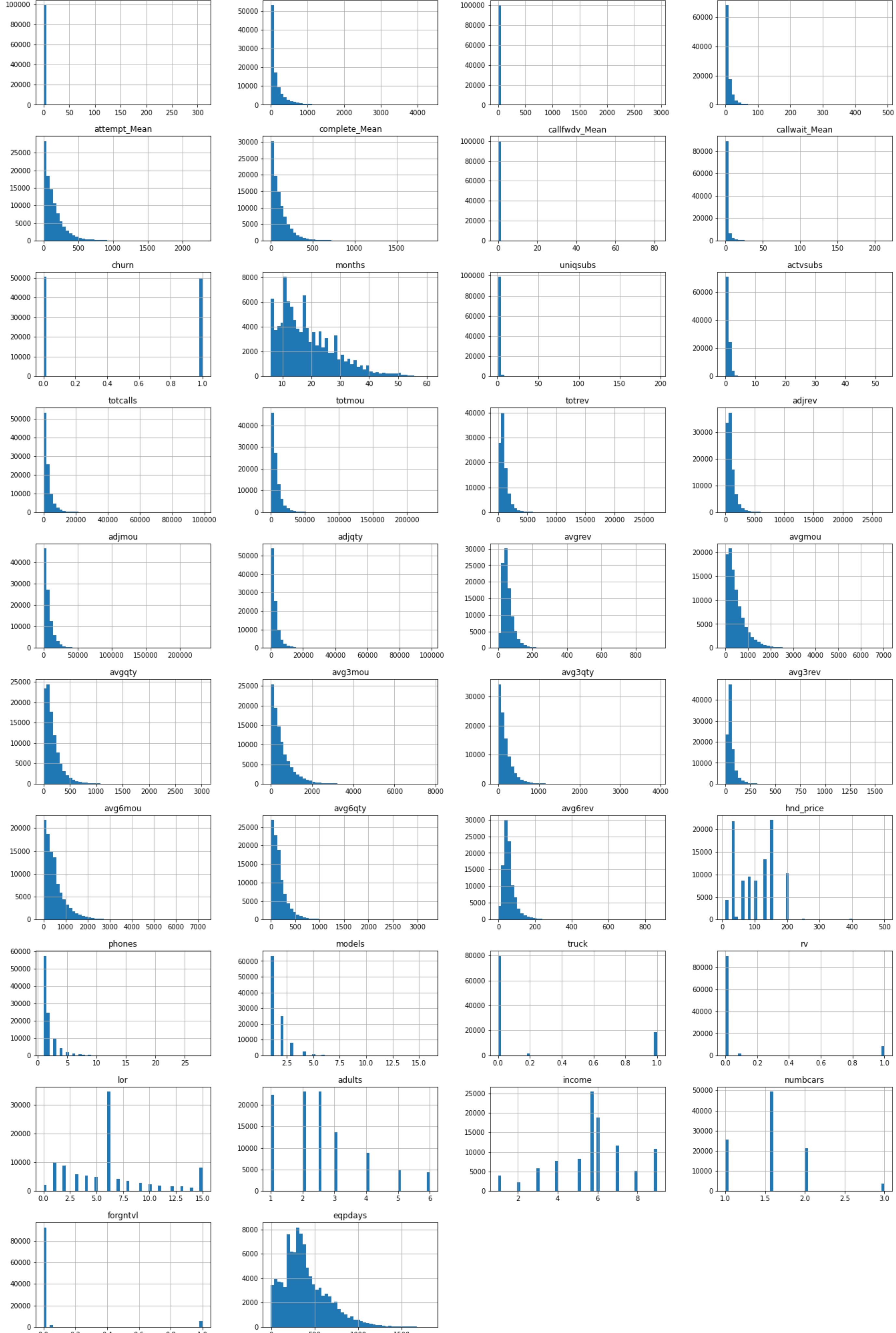
```
          rev_Mean      mou_Mean    totmrc_Mean       da_Mean  \
count  100000.000000  100000.000000  100000.000000  100000.000000
mean       58.719985     513.559937      46.179136       0.888828
std        46.208972     524.229868      23.581283       2.173729
min        -6.167500       0.000000     -26.915000       0.000000
25%        33.311875     151.500000      30.000000       0.000000
50%        48.377500     357.500000      44.990000       0.247500
75%        70.630000     701.250000      59.990000       0.888828
max      3843.262500   12206.750000     409.990000     159.390000

         ovrmou_Mean    ovrrev_Mean    vceovr_Mean    datovr_Mean  \
count  100000.000000  100000.000000  100000.000000  100000.000000
mean       41.072247      13.559560      13.295062       0.261318
std        97.122320      30.446392      30.002391       3.120946
min         0.000000       0.000000       0.000000       0.000000
25%         0.000000       0.000000       0.000000       0.000000
50%         3.000000       1.050000       0.700000       0.000000
75%        42.000000      14.350000      13.950000       0.000000
max      4320.750000    1102.400000     896.087500     423.540000

           roam_Mean     change_mou  ...         phones         models  \
count  100000.000000  100000.000000  ...  100000.000000  100000.000000
mean        1.286405     -13.933818  ...       1.787118       1.545825
std        14.685090     274.854774  ...       1.313971       0.898391
min         0.000000   -3875.000000  ...       1.000000       1.000000
25%         0.000000     -86.000000  ...       1.000000       1.000000
50%         0.000000      -7.000000  ...       1.000000       1.000000
75%         0.257500      61.750000  ...       2.000000       2.000000
max      3685.200000   31219.250000  ...      28.000000      16.000000

               truck             rv            lor         adults  \
count  100000.000000  100000.000000  100000.000000  100000.000000
mean        0.188820       0.082580       6.177238       2.530326
std         0.387964       0.272854       3.956420       1.274685
min         0.000000       0.000000       0.000000       1.000000
25%         0.000000       0.000000       3.000000       2.000000
50%         0.000000       0.000000       6.177238       2.530326
75%         0.000000       0.000000       7.000000       3.000000
max         1.000000       1.000000      15.000000       6.000000

              income       numbcars       forgntvl        eqpdays
count  100000.000000  100000.000000  100000.000000  100000.000000
mean        5.783112       1.567563       0.057974     391.932309
std         1.884277       0.445057       0.231663     256.480910
min         1.000000       1.000000       0.000000      -5.000000
25%         5.000000       1.000000       0.000000     212.000000
50%         5.783112       1.567563       0.000000     342.000000
75%         7.000000       2.000000       0.000000     530.000000
max         9.000000       3.000000       1.000000    1823.000000

[8 rows x 78 columns]
```

In [13]:

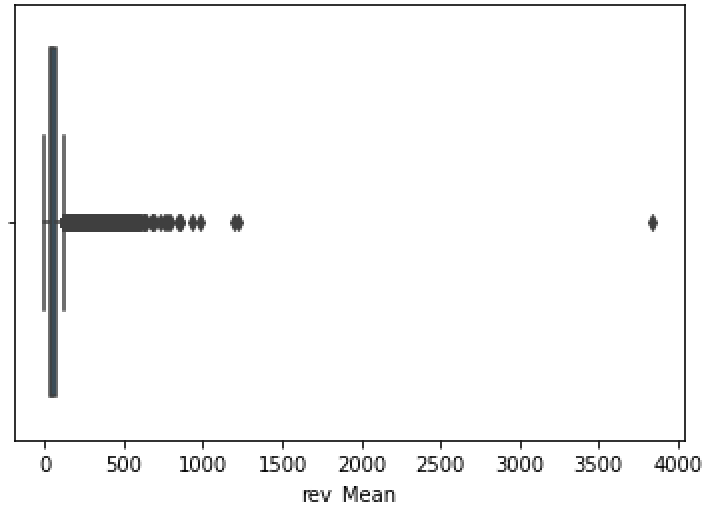
```python
import seaborn as sns
import matplotlib.pyplot as plt

def studying_numerical_columns(df):
    """
      Function that visualizes numerical columns with respect to target variable

    """
    cont_vals = get_numerical_columns(df)
    df.iloc[:,:].hist(bins=50,figsize=(23,74),layout=(20,4));
```

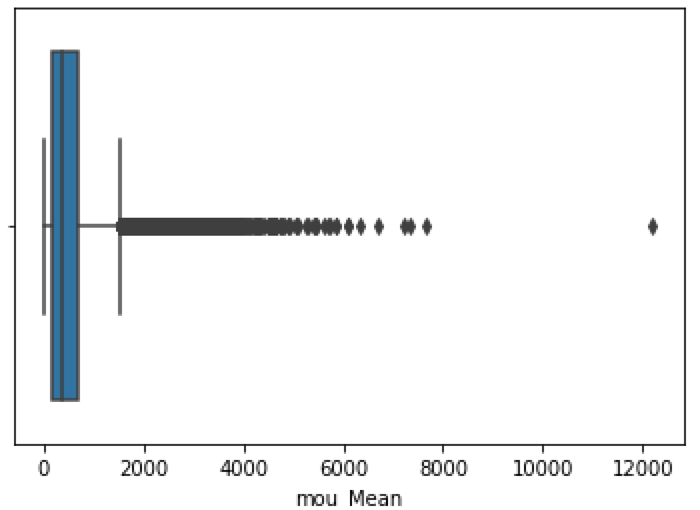
```python
#Visualizing Numerical values
studying_numerical_columns(cust_data_df)
```

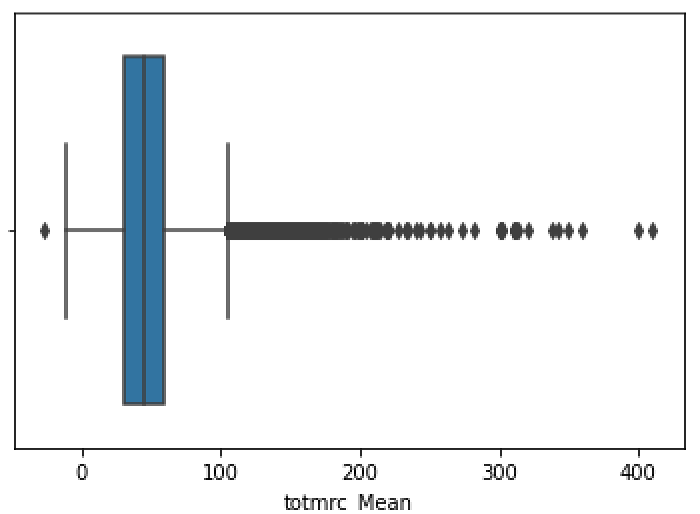

rev_Mean
mou_Mean
totmrc_Mean
da_Mean
ovrmou_Mean
ovrrev_Mean
vceovr_Mean
datovr_Mean
roam_Mean
change_mou
change_rev
drop_vce_Mean
drop_dat_Mean
blck_vce_Mean
blck_dat_Mean
unan_vce_Mean
unan_dat_Mean
plcd_vce_Mean
plcd_dat_Mean
recv_vce_Mean
recv_sms_Mean
comp_vce_Mean
comp_dat_Mean
custcare_Mean
ccrndmou_Mean
cc_mou_Mean
inonemin_Mean
threeway_Mean
mou_cvce_Mean
mou_cdat_Mean
mou_rvce_Mean
owylis_vce_Mean
mouowylisv_Mean
iwylis_vce_Mean
mouiwylisv_Mean
peak_vce_Mean
peak_dat_Mean
mou_peav_Mean
mou_pead_Mean
opk_vce_Mean
opk_dat_Mean
mou_opkv_Mean
mou_opkd_Mean
drop_blk_Mean

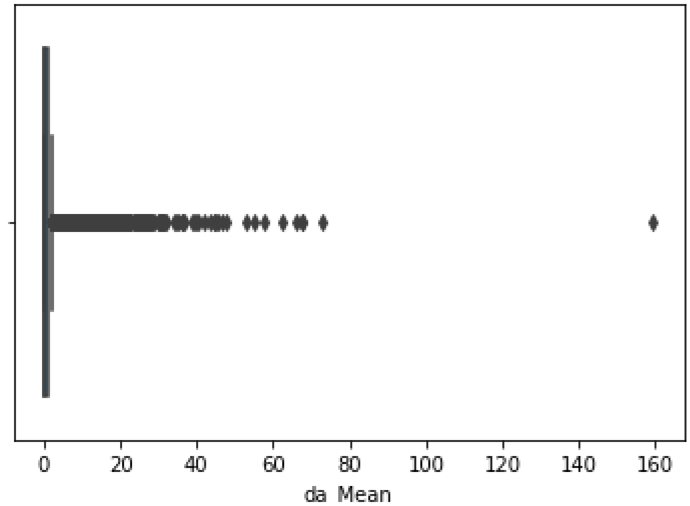

attempt_Mean
complete_Mean
callfwdv_Mean
callwait_Mean
churn
months
uniqsubs
actvsubs
totcalls
totmou
totrev
adjrev
adjmou
adjqty
avgrev
avgmou
avgqty
avg3mou
avg3qty
avg3rev
avg6mou
avg6qty
avg6rev
hnd_price
phones
models
truck
rv
lor
adults
income
numbcars
forgntvl
eqpdays

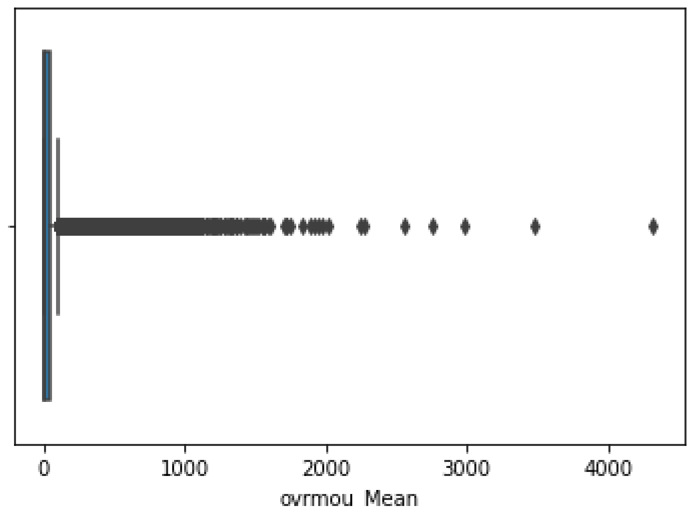
In [14]:
```
import numpy as np
from collections import Counter

def visualize_outliers(df):
  """
    Function that uses box plot to visualize outliers

  """

  vals = get_numerical_columns(df)

  for col in vals:
      sns.boxplot(x=df[col])
      plt.show()

def remove_outliers(df):
  """
    Function that uses IQR statistical method to identify and remove outliers

  """

  visualize_outliers(df)

  targ_cols = get_numerical_columns(df)
  indices = []
  #The data is left skewed so to remove outliers we will use IQR technique

  for c in targ_cols:
      # 1st quartile
      Q1 = np.percentile(df[c],25)
      # 3rd quartile
      Q3 = np.percentile(df[c],75)
      # IQR
      IQR = Q3 - Q1
      # Outlier step
      outlier_step = IQR * 1.5
      # detect outlier and their indeces
      outlier_list_col = df[(df[c] < Q1 - outlier_step) | (df[c] > Q3 + outlier_step)].index
      # store indeces
      indices.extend(outlier_list_col)

  indices = Counter(indices)
  multiple_outliers = list(i for i, v in indices.items() if v > 2)

  return indices

#Outlier Detection and removal in continuous column values To be experimented with & without thi
outlier_index = remove_outliers(cust_data_df)
cust_data_df = cust_data_df.drop(outlier_index, axis=0).reset_index(drop=True)
```

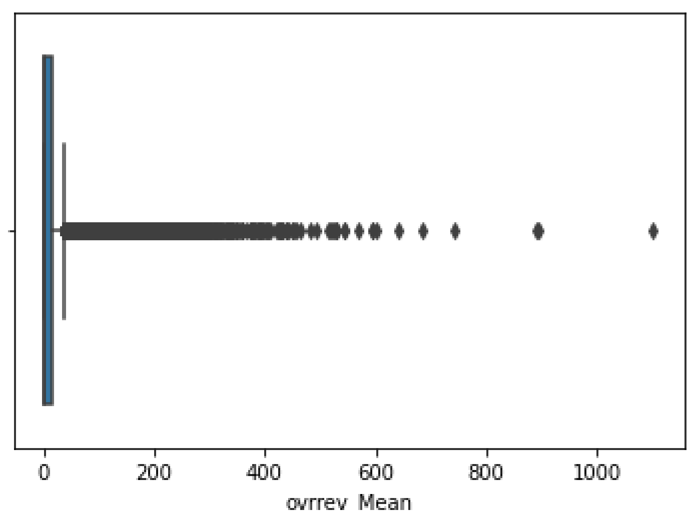

0
500
1000
1500
2000
2500
3000
3500
4000
rev_Mean

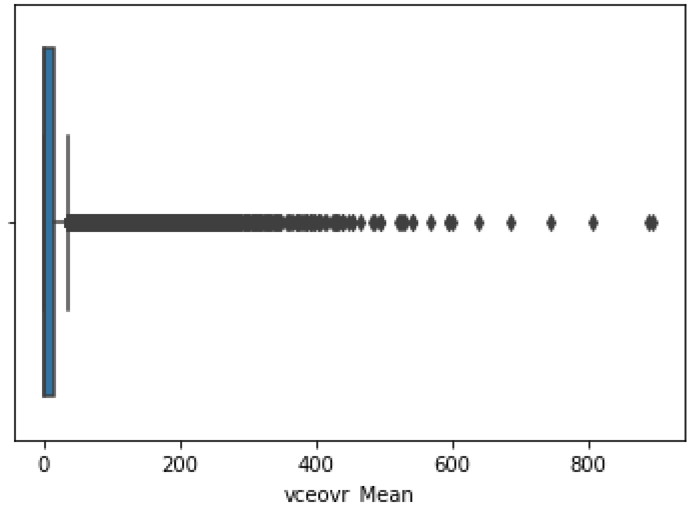

0
2000
4000
6000
8000
10000
12000
mou_Mean

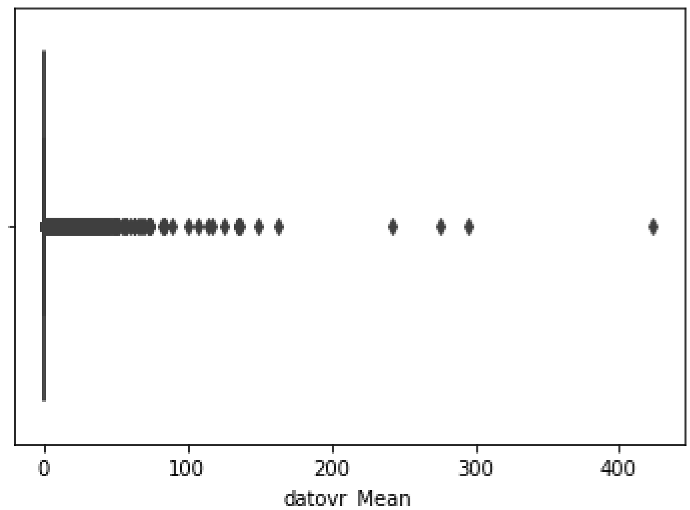

0
100
200
300
400
totmrc_Mean

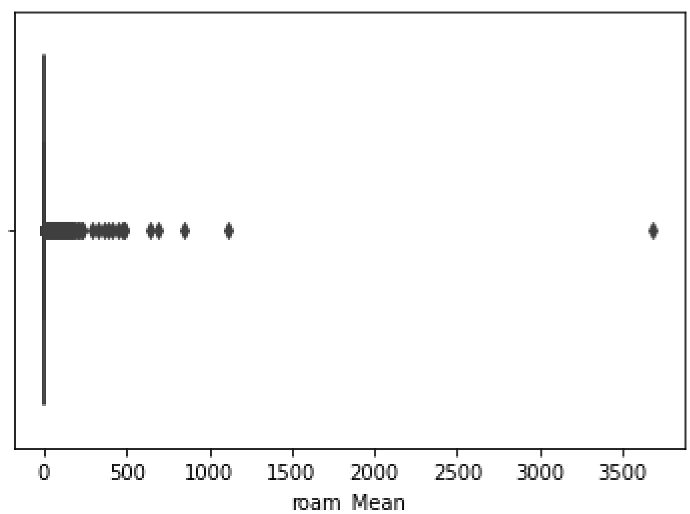

0
20
40
60
80
100
120
140
160
da_Mean

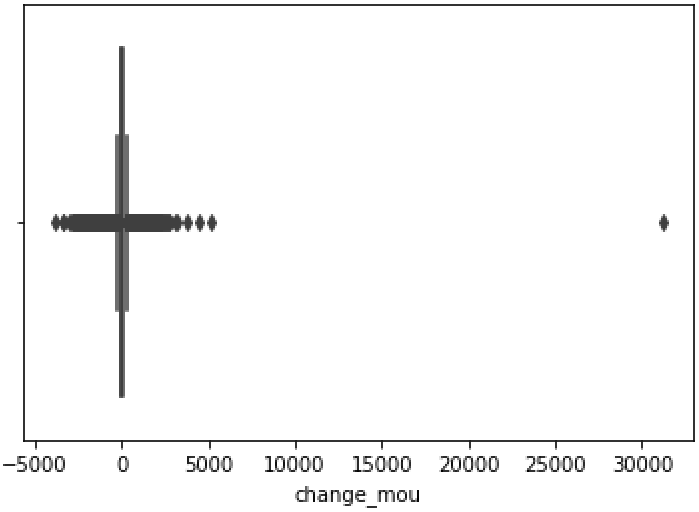

0
1000
2000
3000
4000
ovrmou_Mean

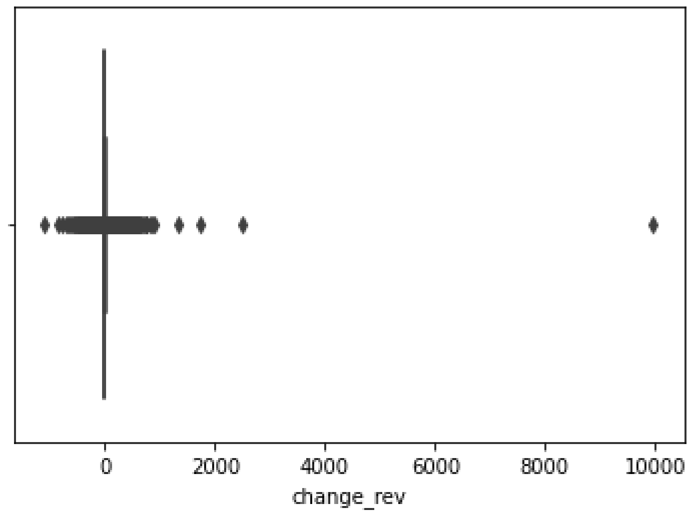

0
200
400
600
800
1000
ovrrev_Mean

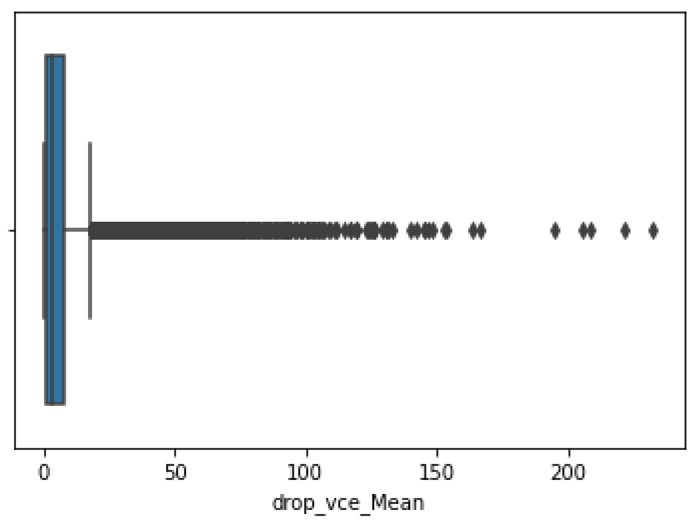

0
200
400
600
800
vceovr_Mean

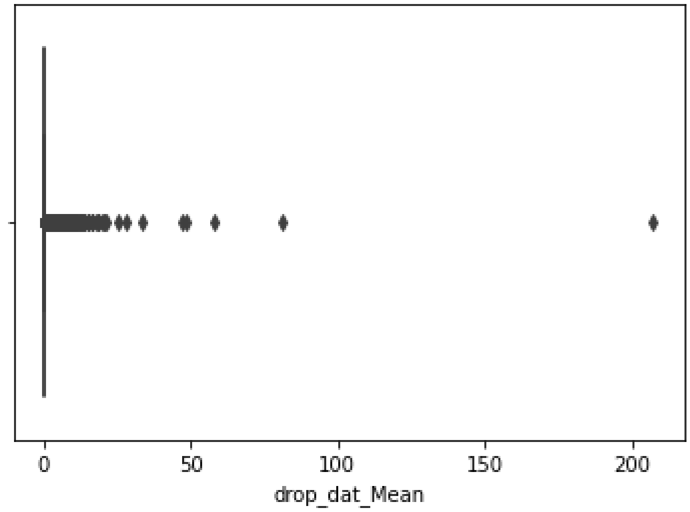

0
100
200
300
400
datovr_Mean

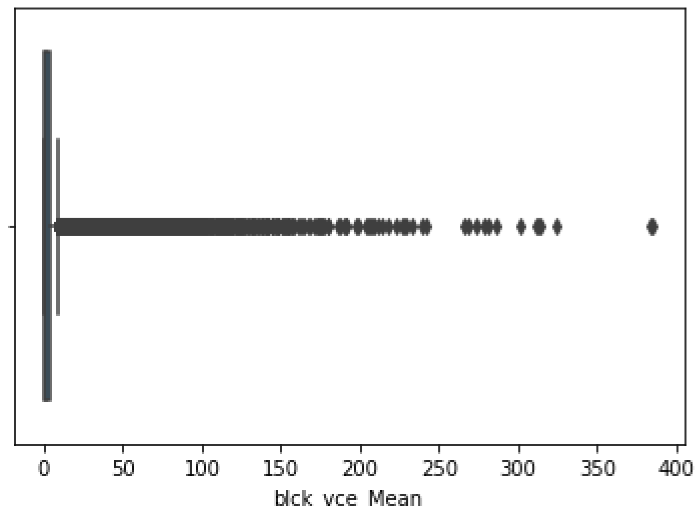

0
500
1000
1500
2000
2500
3000
3500
roam_Mean

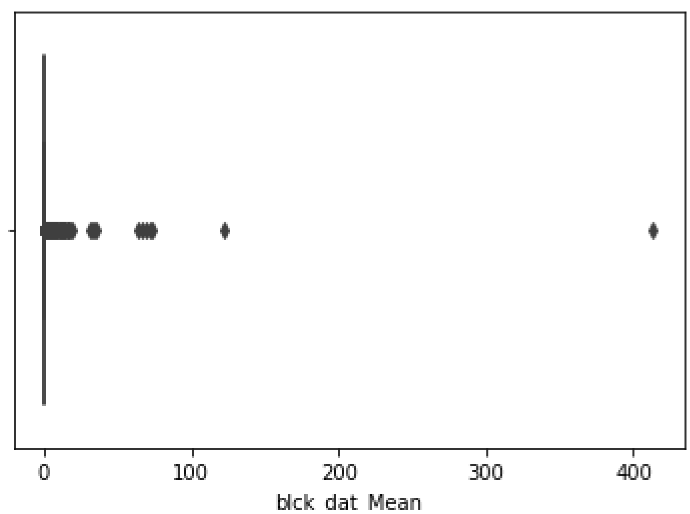

−5000
0
5000
10000
15000
20000
25000
30000
change_mou

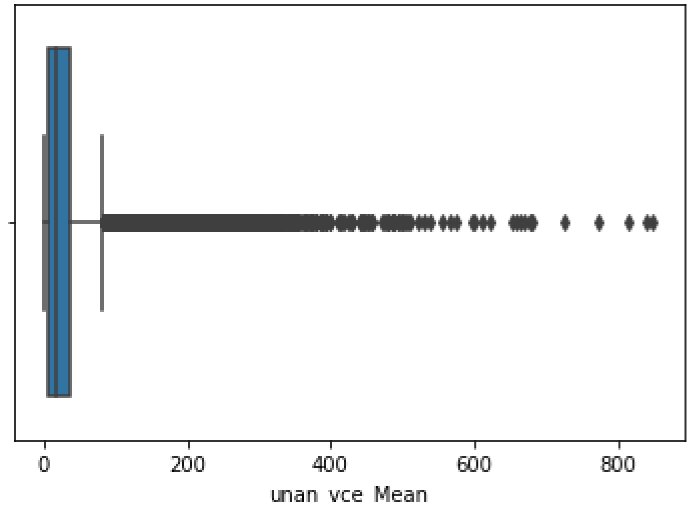

0
2000
4000
6000
8000
10000
change_rev

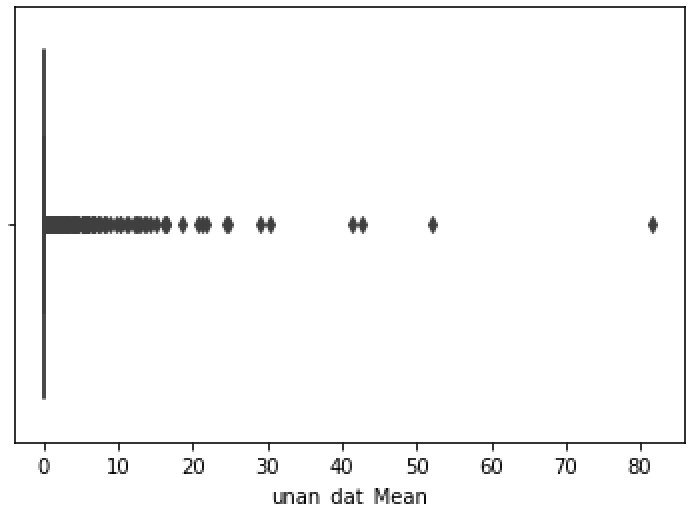

0
50
100
150
200
drop_vce_Mean

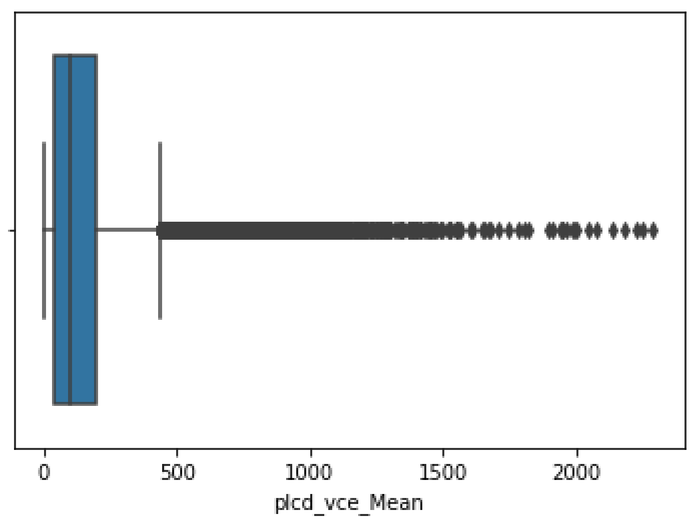

0
50
100
150
200
drop_dat_Mean

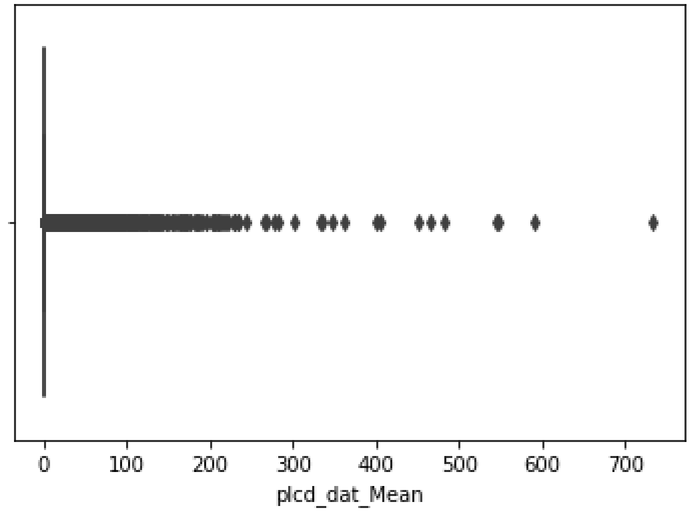

0
50
100
150
200
250
300
350
400
blck_vce_Mean

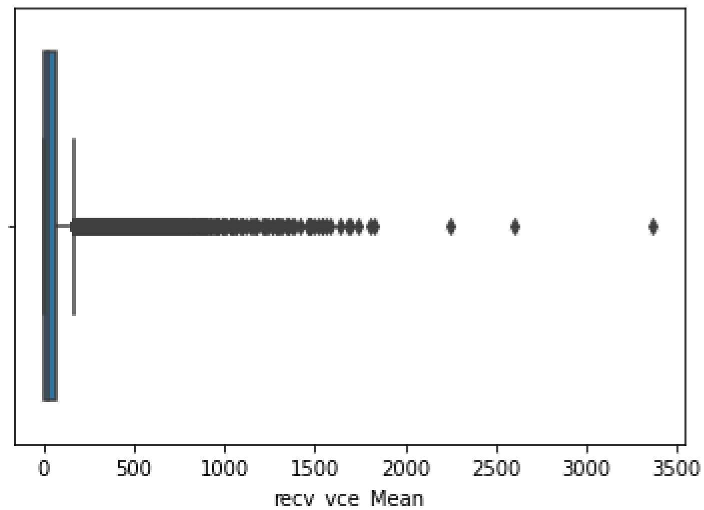

0
100
200
300
400
blck_dat_Mean

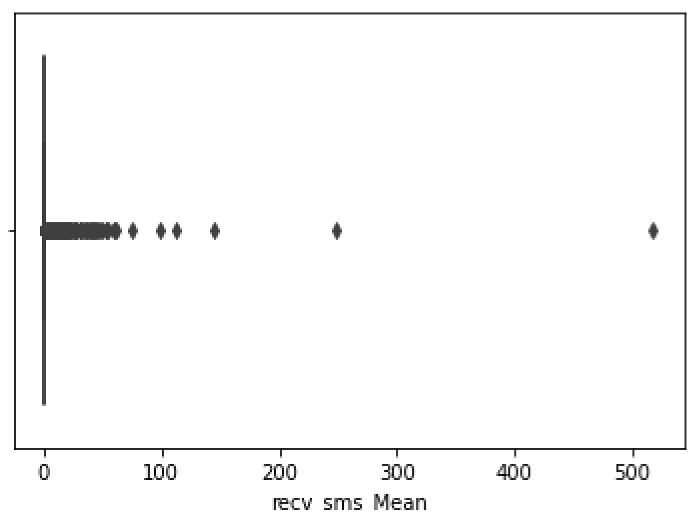

0
200
400
600
800
unan_vce_Mean

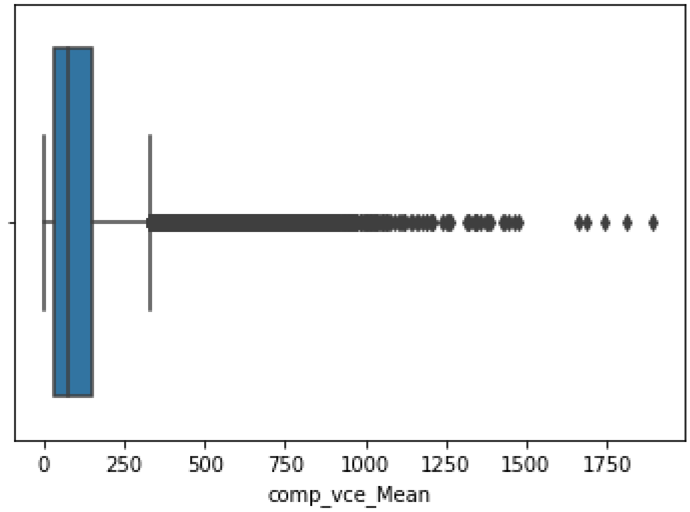

0
10
20
30
40
50
60
70
80
unan_dat_Mean

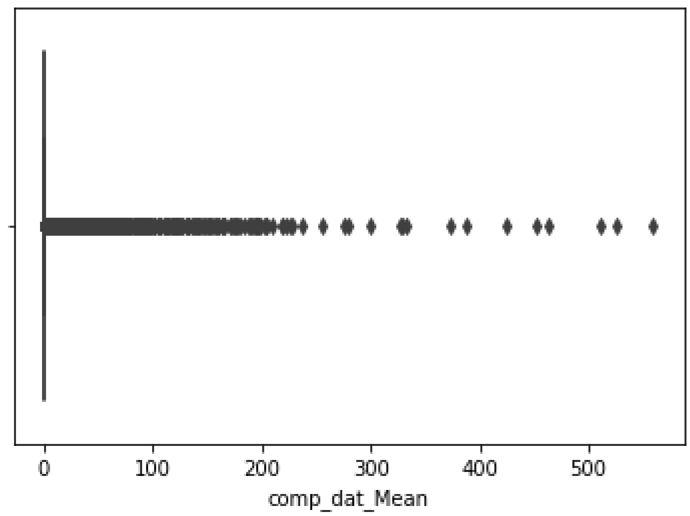

0
500
1000
1500
2000
plcd_vce_Mean

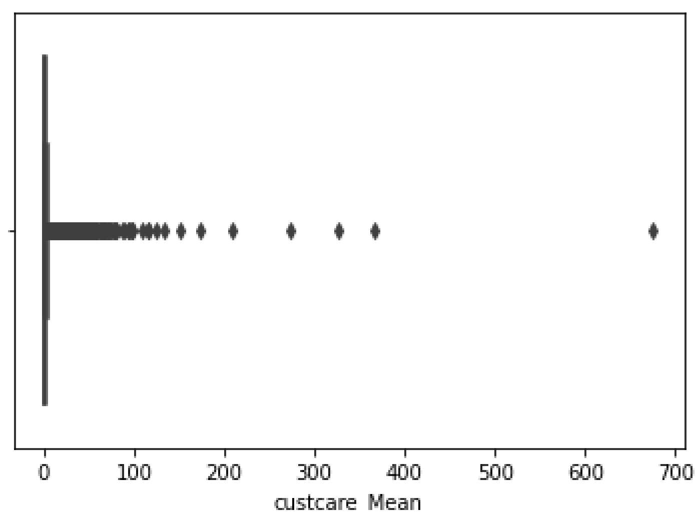

0 100 200 300 400 500 600 700
plcd_dat_Mean

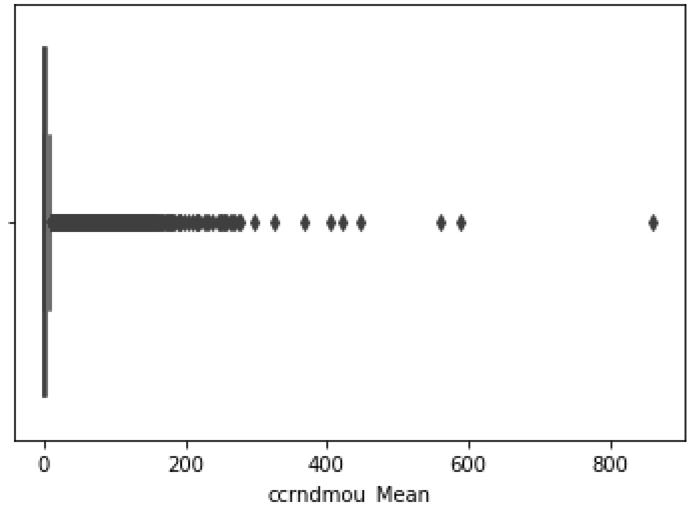

0 500 1000 1500 2000 2500 3000 3500
recv_vce_Mean


0 100 200 300 400 500
recv_sms_Mean

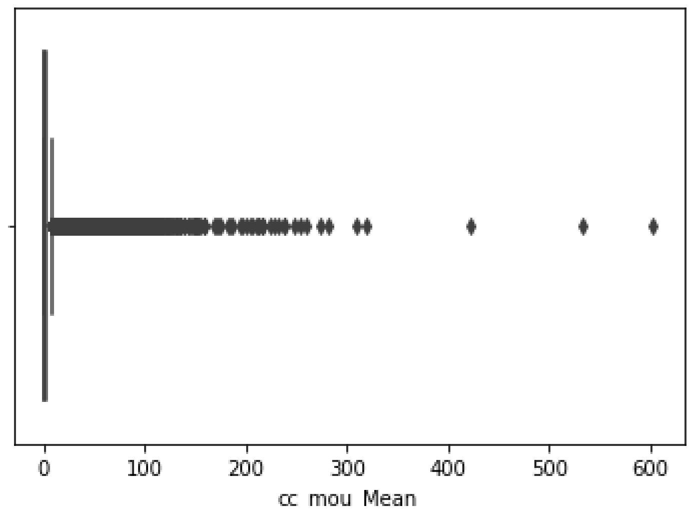

0
250
500
750
1000
1250
1500
1750
comp_vce_Mean


0
100
200
300
400
500
comp_dat_Mean

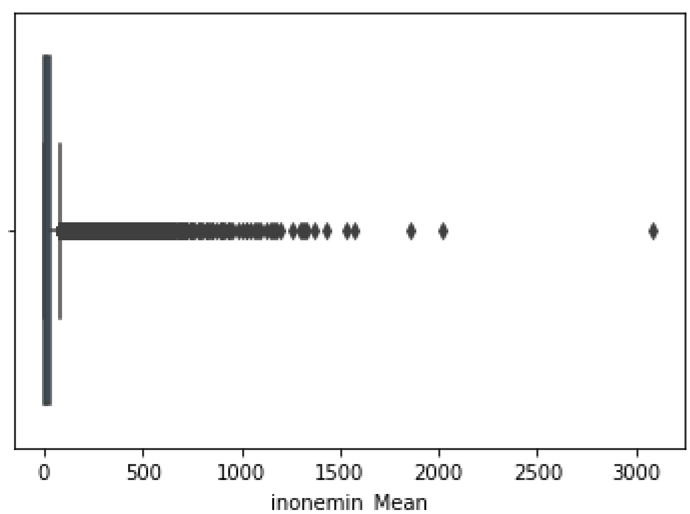

0
100
200
300
400
500
600
700
custcare_Mean



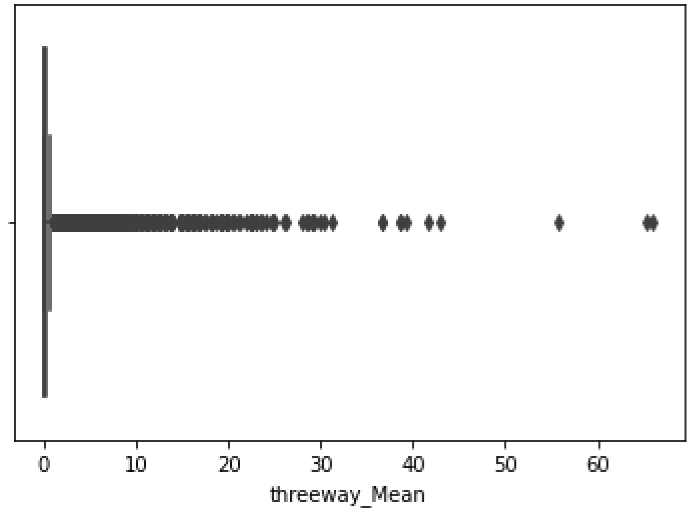
0
200
400
600
800
ccrndmou_Mean



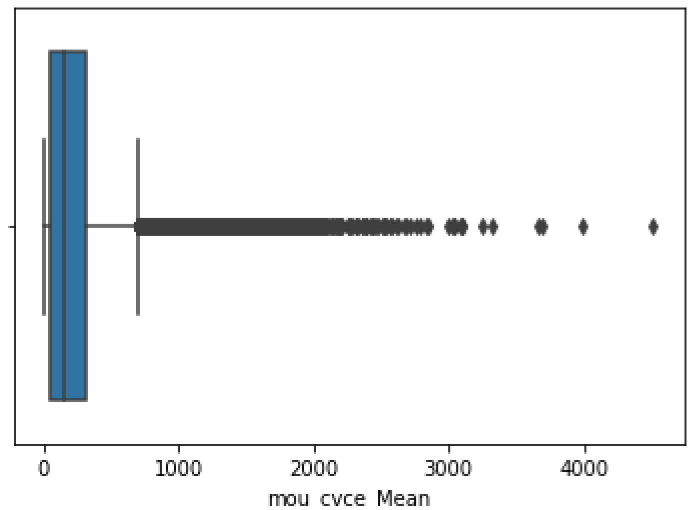
0
100
200
300
400
500
600
cc_mou_Mean



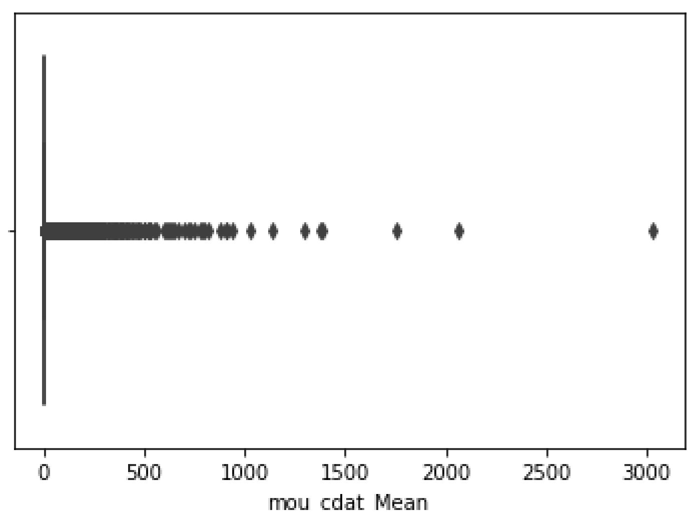
0
500
1000
1500
2000
2500
3000
inonemin_Mean



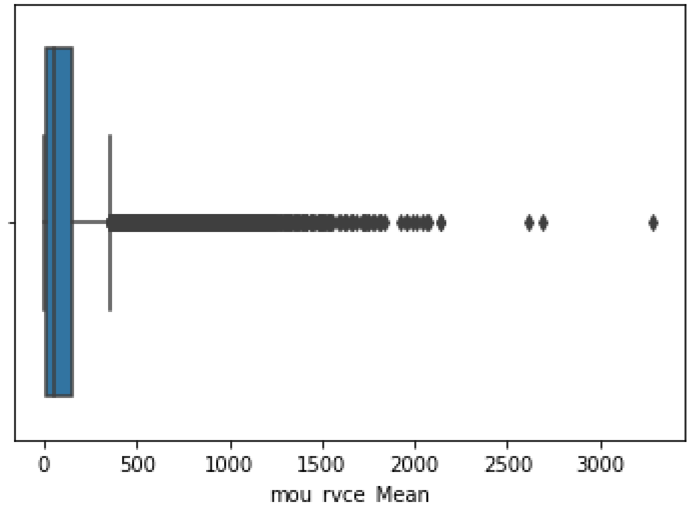
0 10 20 30 40 50 60
threeway_Mean

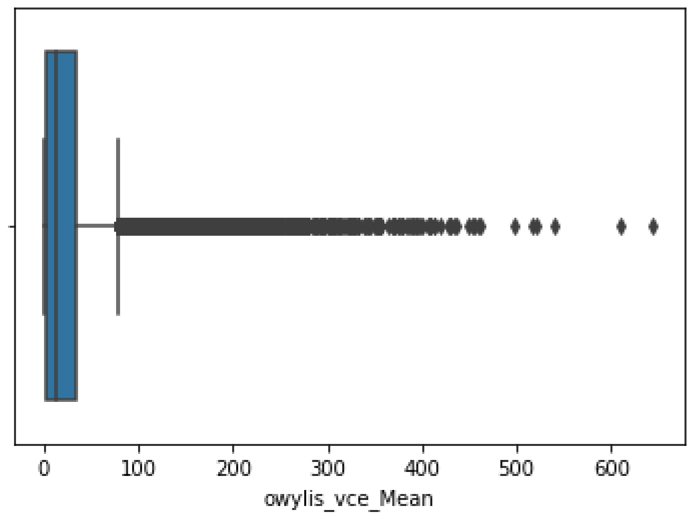


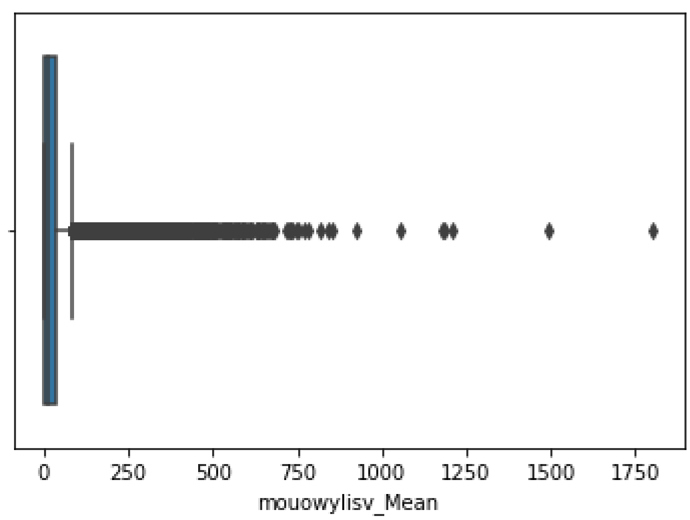
0 1000 2000 3000 4000
mou_cvce_Mean

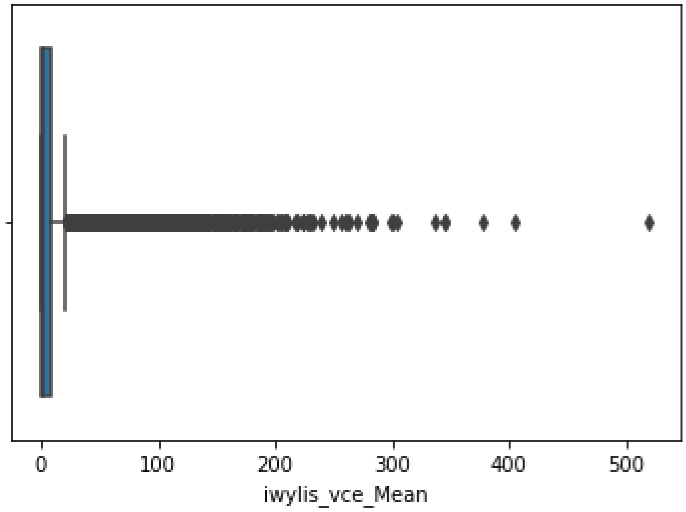


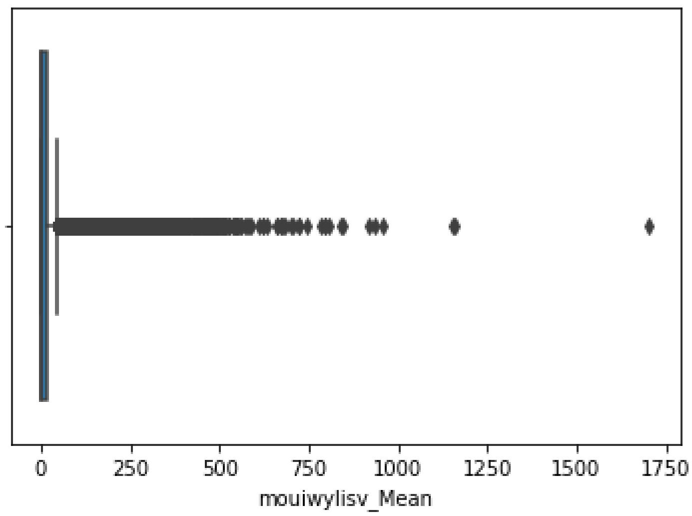
0 500 1000 1500 2000 2500 3000
mou_cdat_Mean

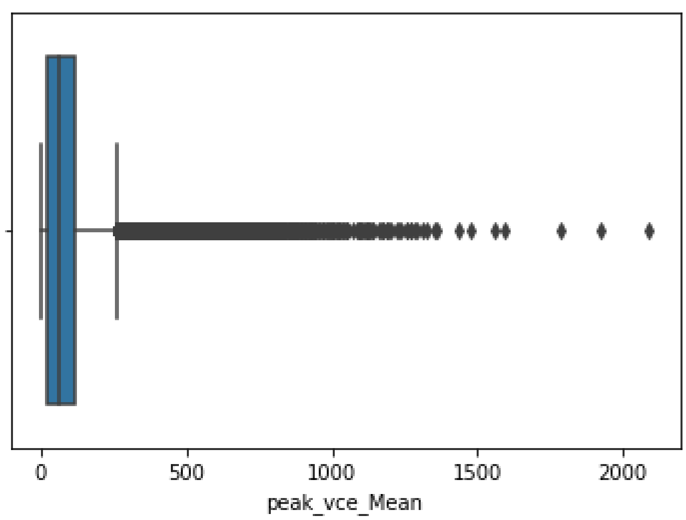

0
500
1000
1500
2000
2500
3000
mou_rvce_Mean

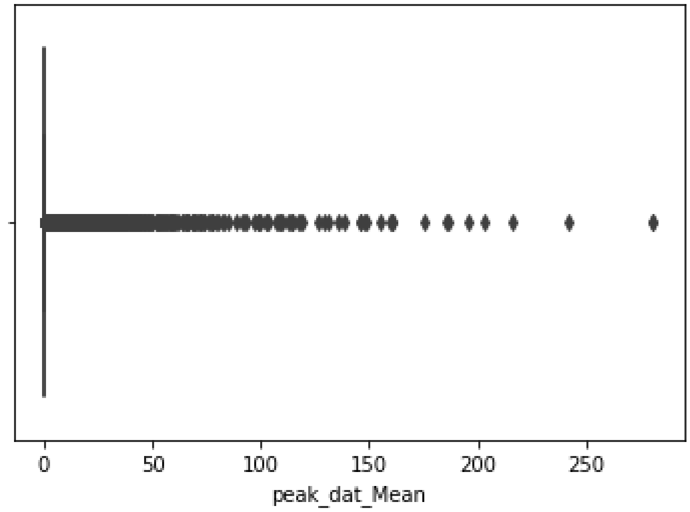

0
100
200
300
400
500
600
owylis_vce_Mean

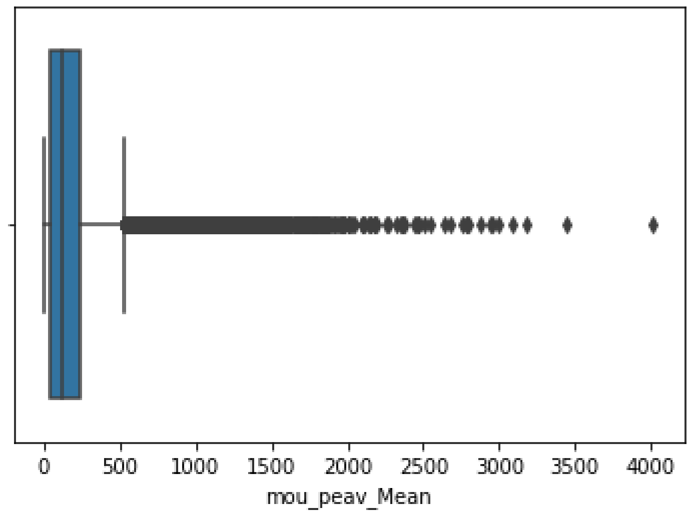

0
250
500
750
1000
1250
1500
1750
mouowylisv_Mean

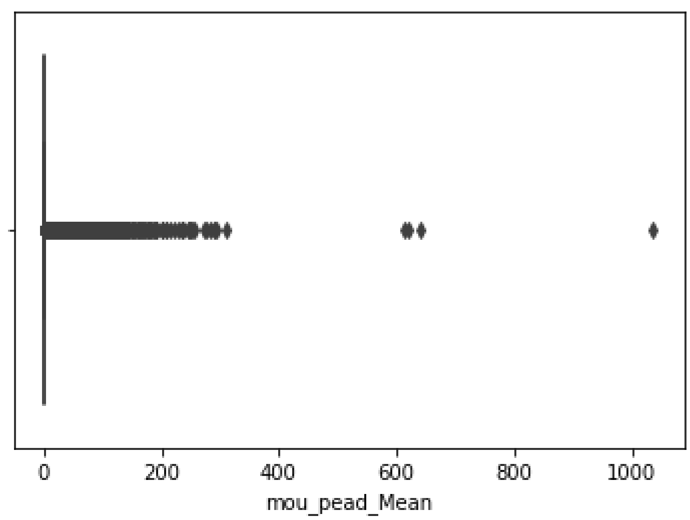

0
100
200
300
400
500
iwylis_vce_Mean

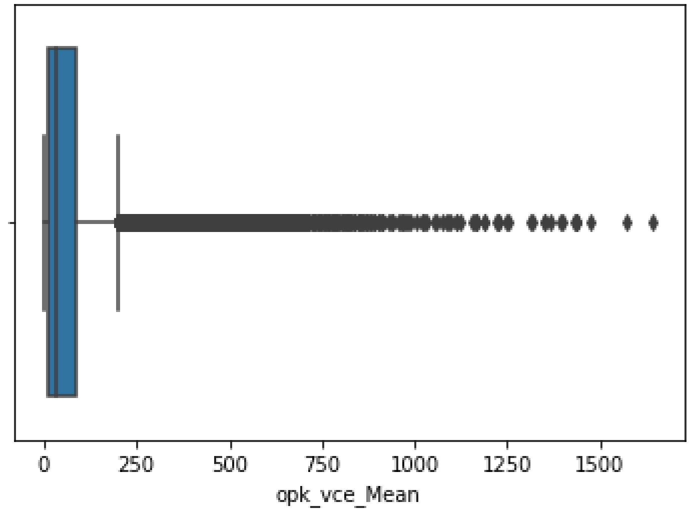

0
250
500
750
1000
1250
1500
1750
mouiwylisv_Mean

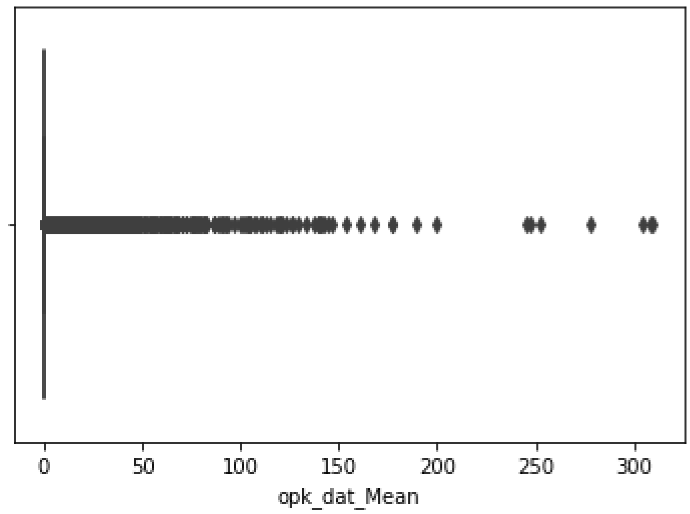

0
500
1000
1500
2000
peak_vce_Mean

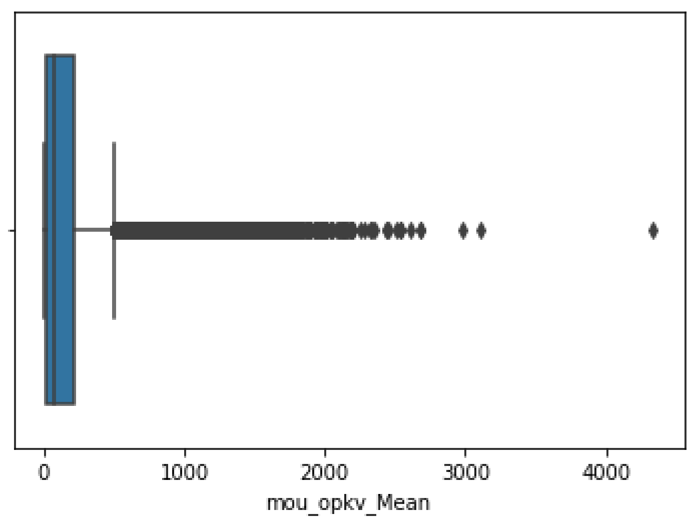

0
50
100
150
200
250
peak_dat_Mean

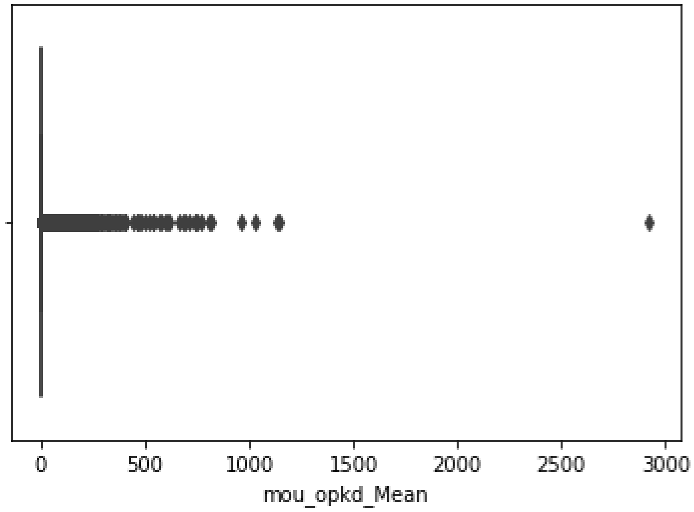

0
500
1000
1500
2000
2500
3000
3500
4000
mou_peav_Mean


0
200
400
600
800
1000
mou_pead_Mean

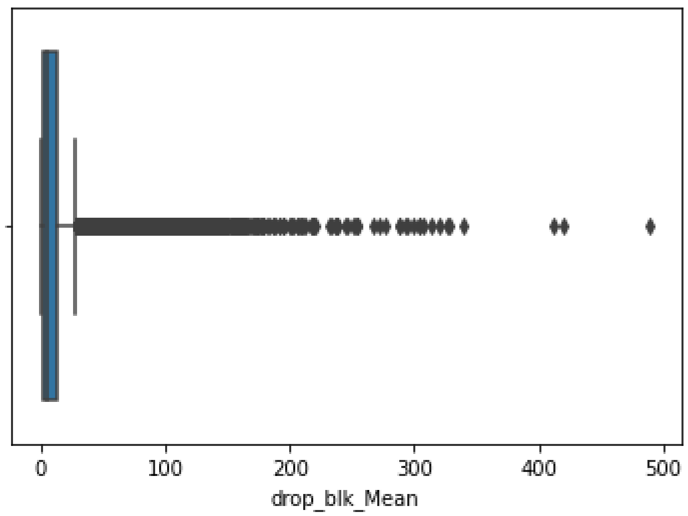

0
250
500
750
1000
1250
1500
opk_vce_Mean


0
50
100
150
200
250
300
opk_dat_Mean

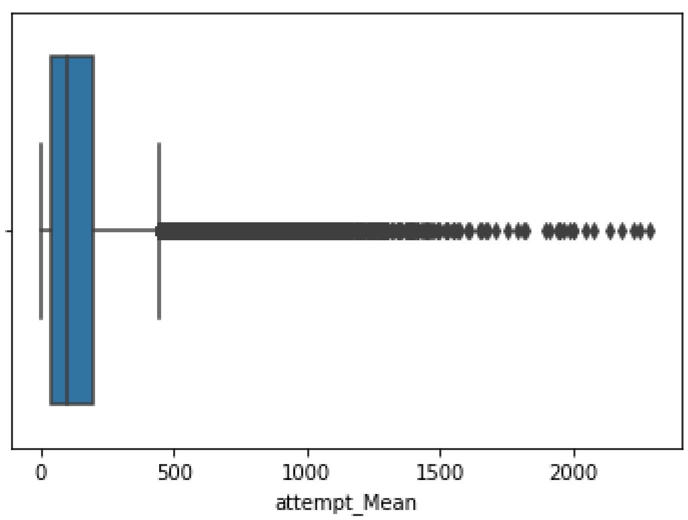

0
1000
2000
3000
4000
mou_opkv_Mean



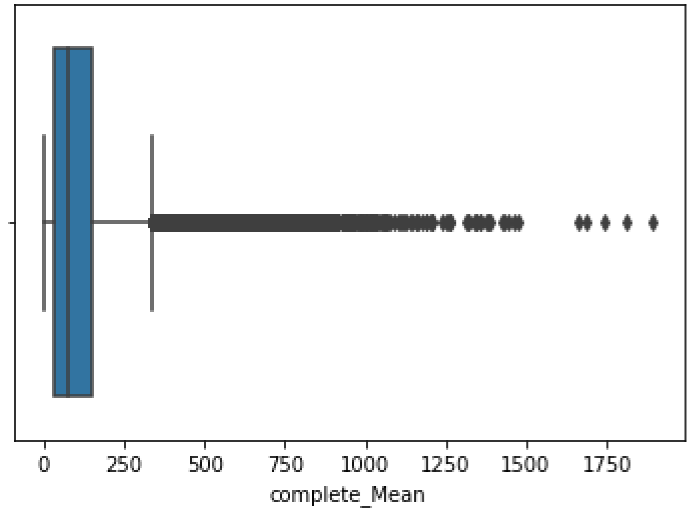
0
500
1000
1500
2000
2500
3000
mou_opkd_Mean

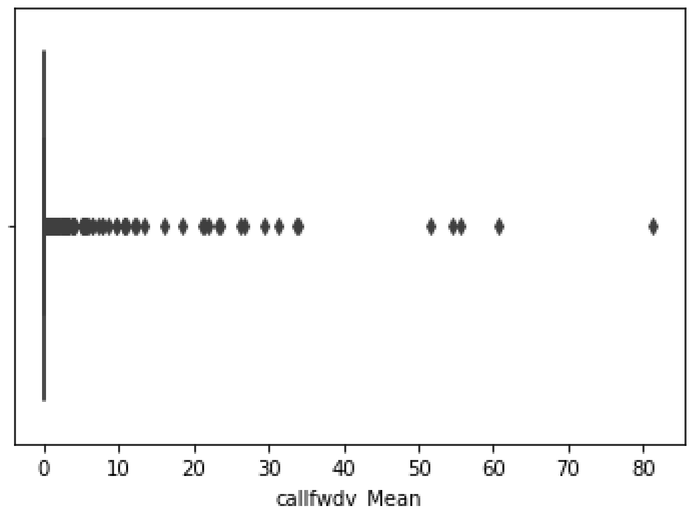


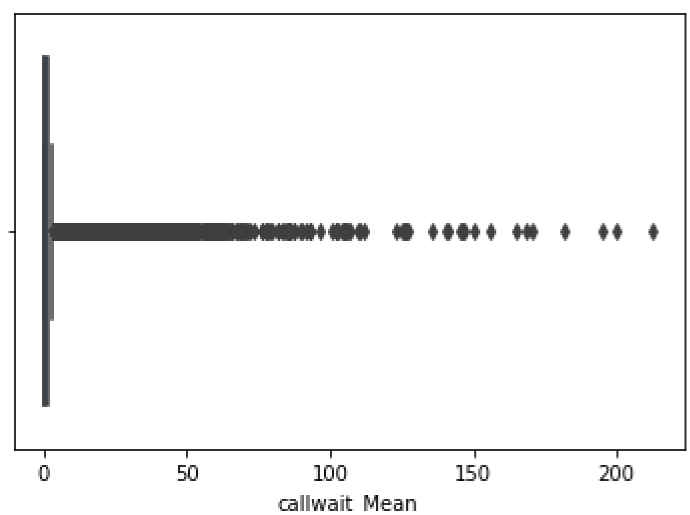
0
100
200
300
400
500
drop_blk_Mean

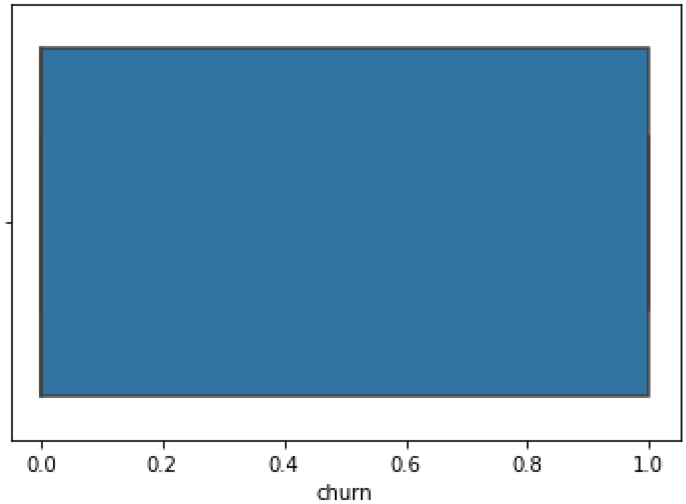


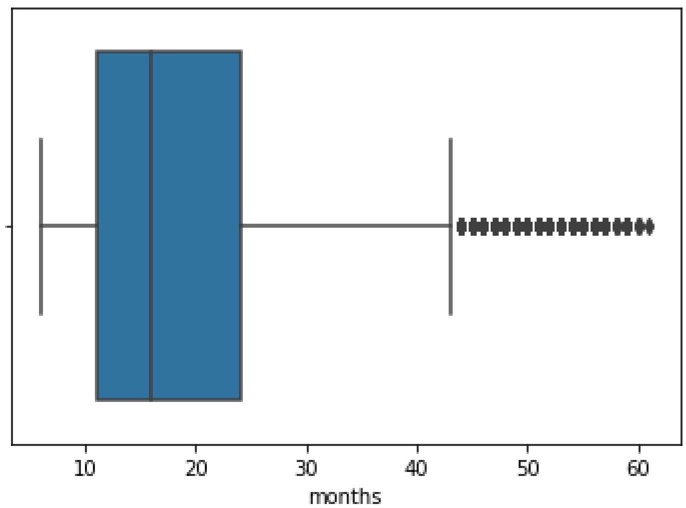
0
500
1000
1500
2000
attempt_Mean

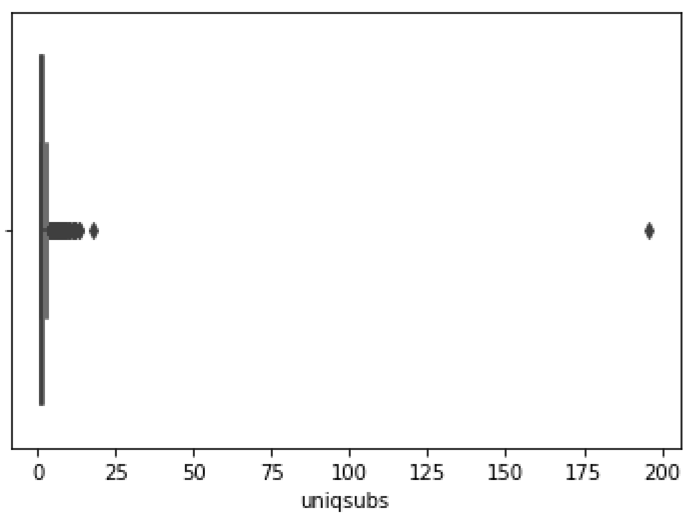

0
250
500
750
1000
1250
1500
1750
complete_Mean

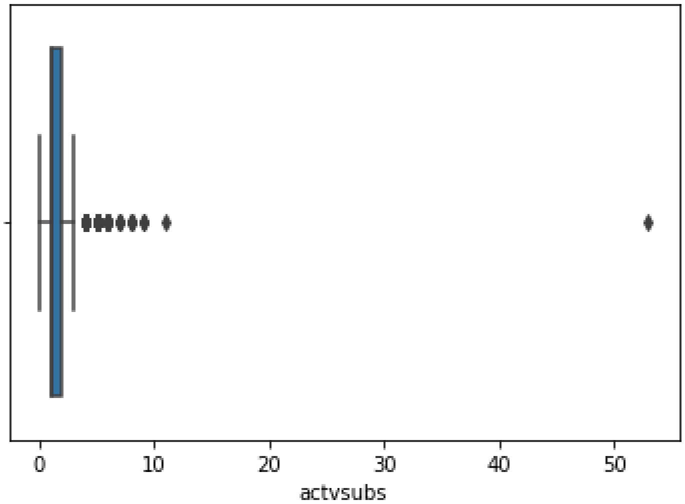

0
10
20
30
40
50
60
70
80
callfwdv_Mean

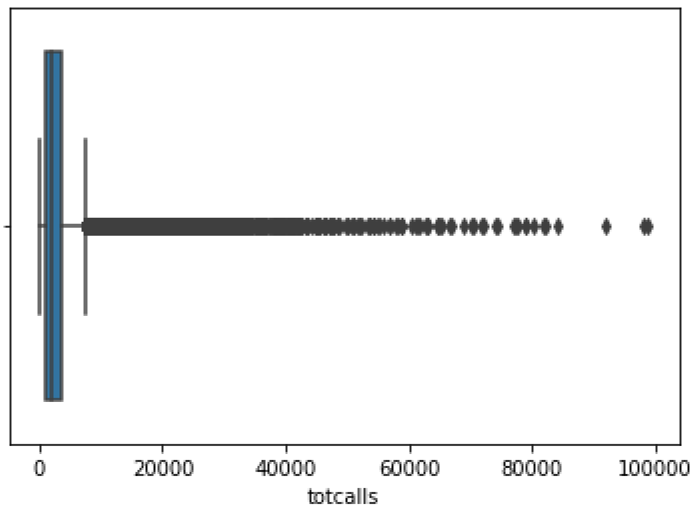

0
50
100
150
200
callwait_Mean

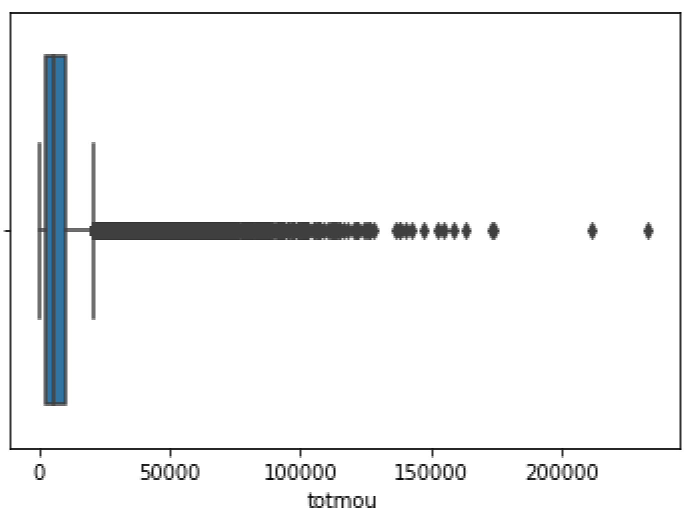

0.0
0.2
0.4
0.6
0.8
1.0
churn

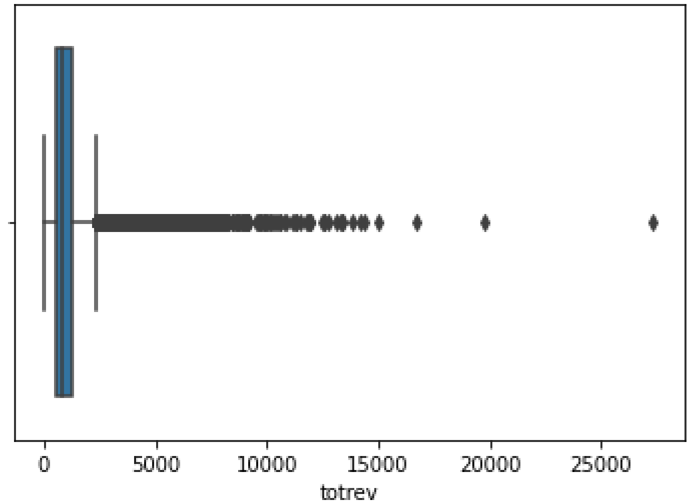

10
20
30
40
50
60
months

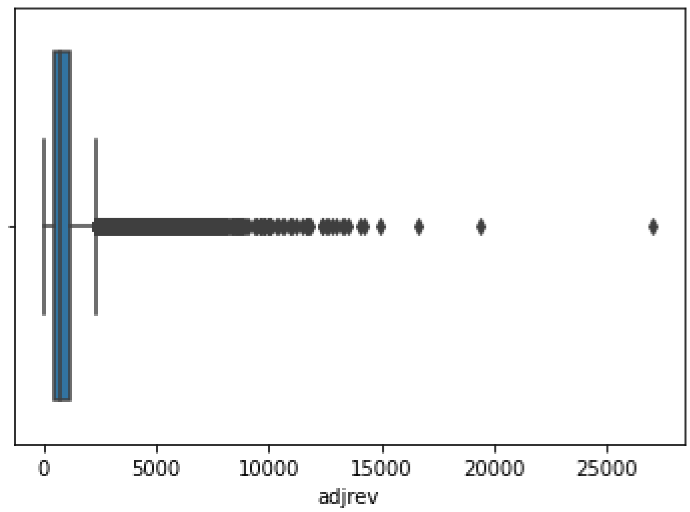

0
25
50
75
100
125
150
175
200
uniqsubs

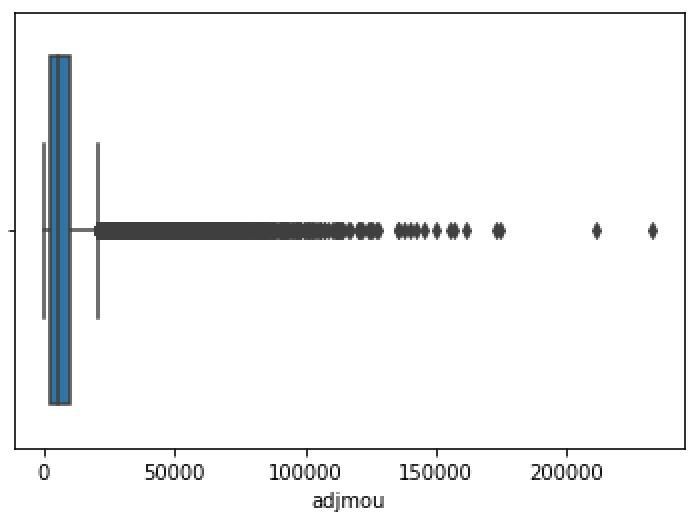

0
10
20
30
40
50
actvsubs

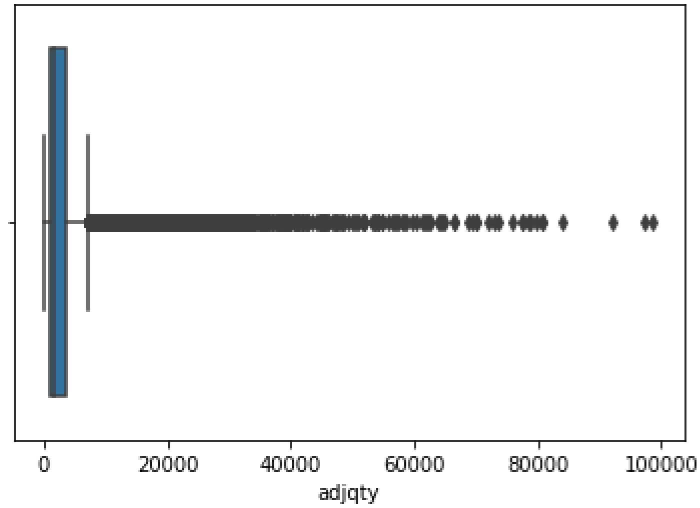

0
20000
40000
60000
80000
100000
totcalls

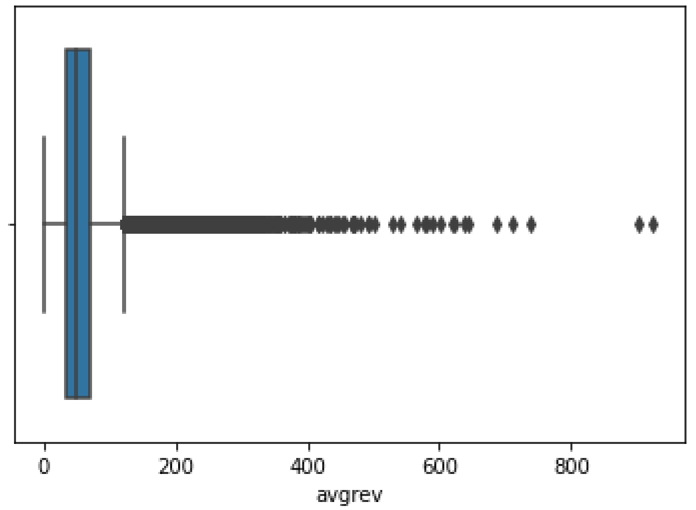

0
50000
100000
150000
200000
totmou

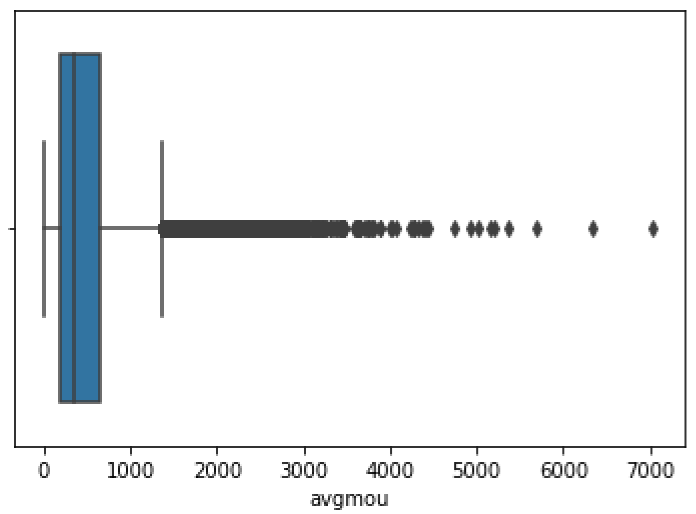

0
5000
10000
15000
20000
25000
totrev

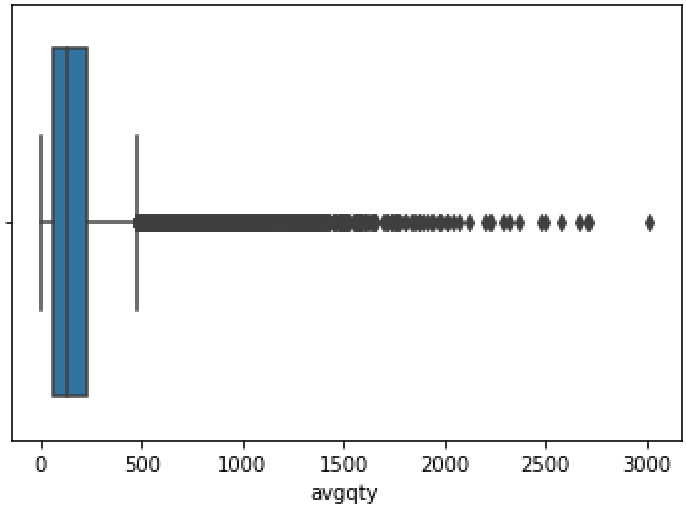

0
5000
10000
15000
20000
25000
adjrev

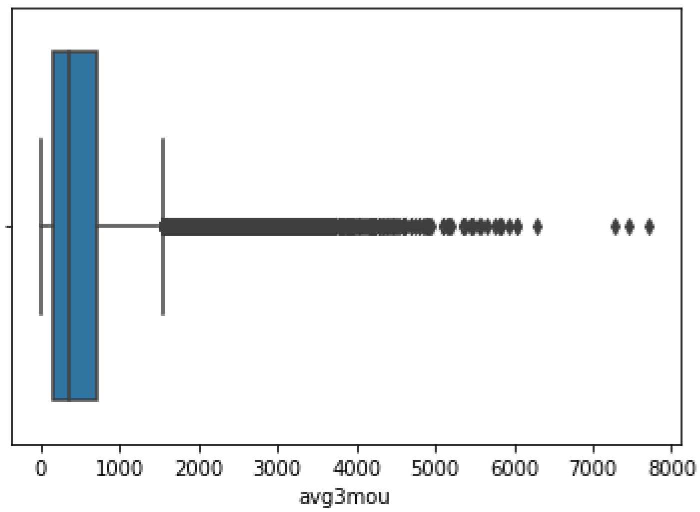

0
50000
100000
150000
200000
adjmou

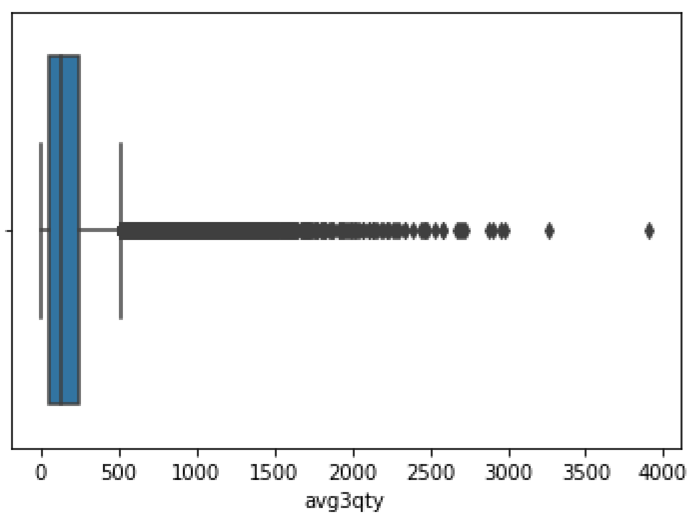

0
20000
40000
60000
80000
100000
adjqty

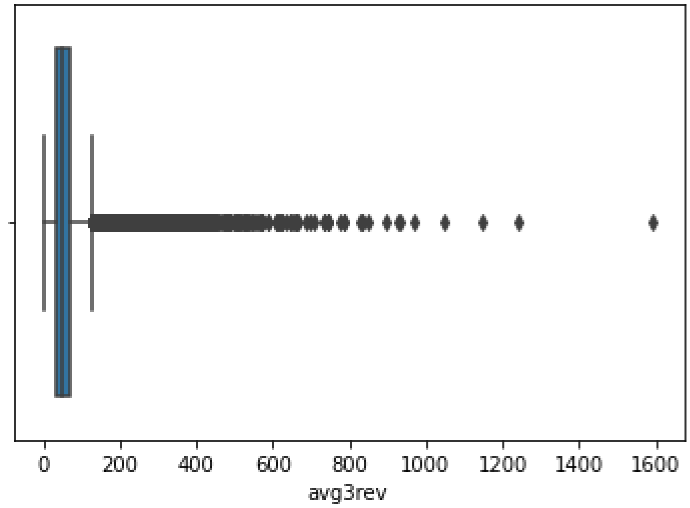

0
200
400
600
800
avgrev

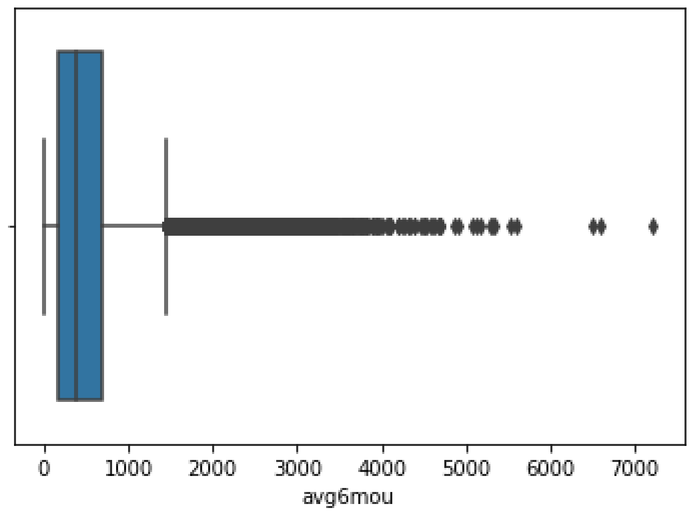

0
1000
2000
3000
4000
5000
6000
7000
avgmou

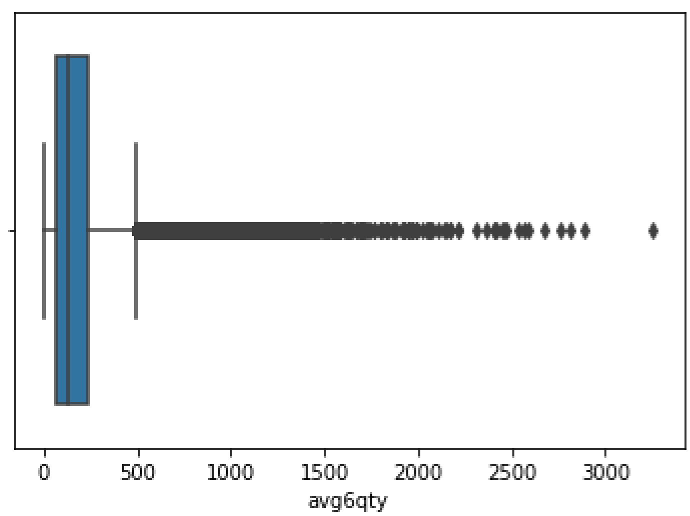

0
500
1000
1500
2000
2500
3000
avgqty

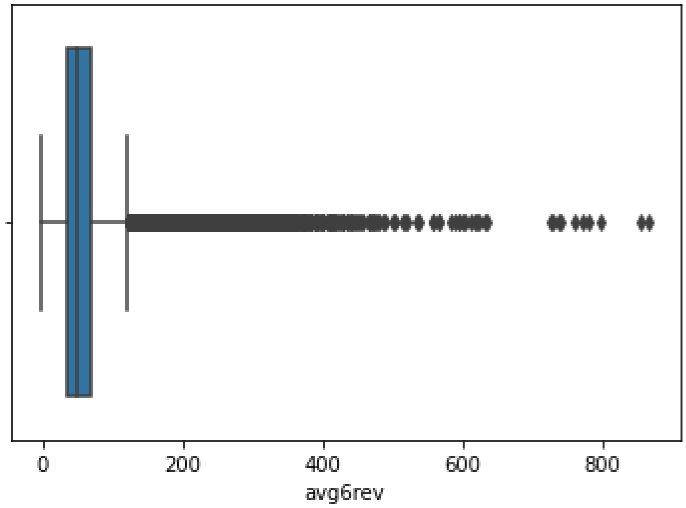

0
1000
2000
3000
4000
5000
6000
7000
8000
avg3mou

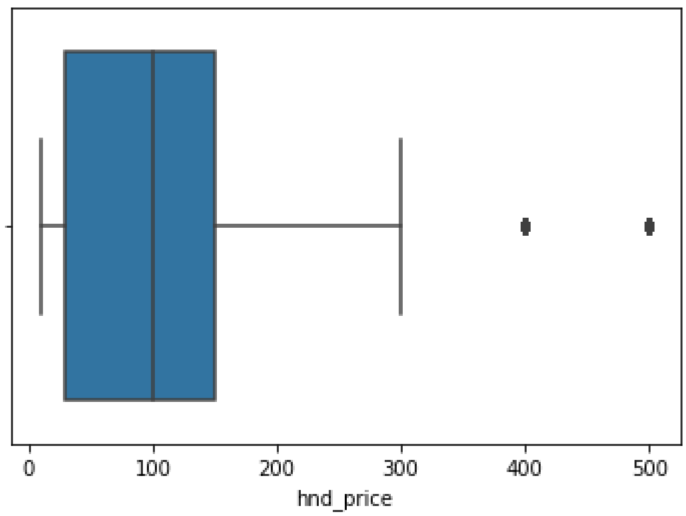

0
500
1000
1500
2000
2500
3000
3500
4000
avg3qty

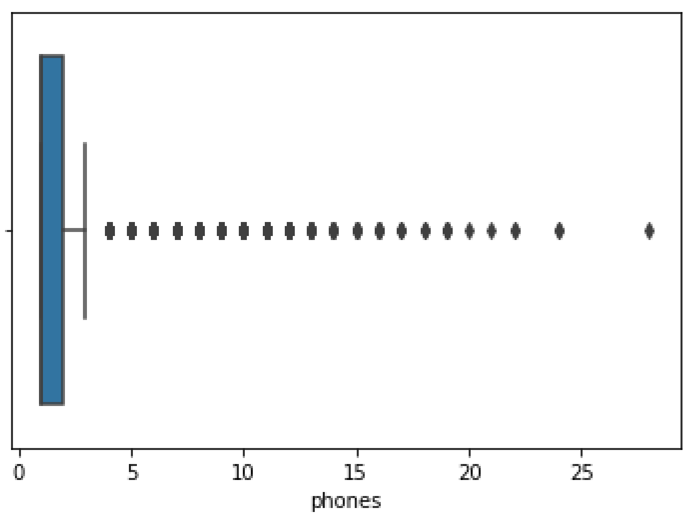

0
200
400
600
800
1000
1200
1400
1600
avg3rev

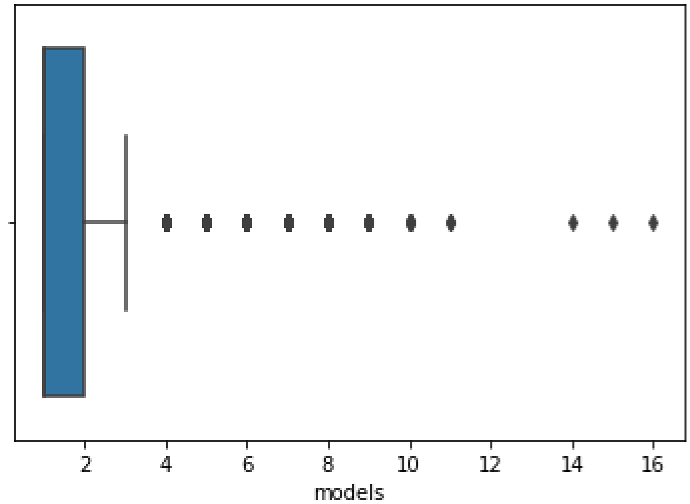

0
1000
2000
3000
4000
5000
6000
7000
avg6mou


0
500
1000
1500
2000
2500
3000
avg6qty

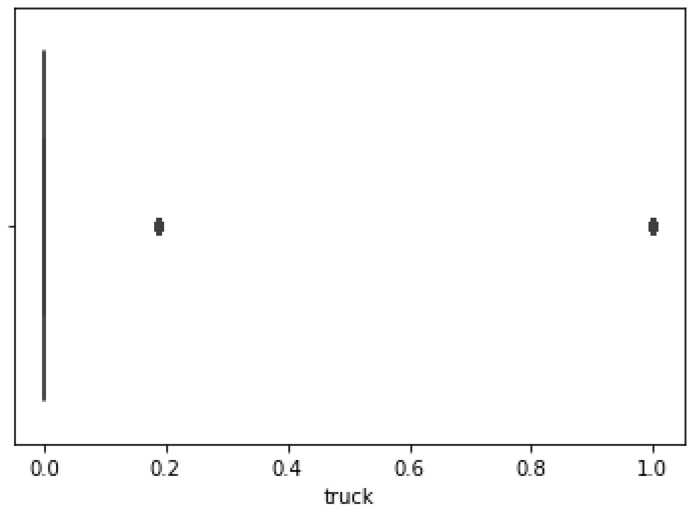

0
200
400
600
800
avg6rev


0
100
200
300
400
500
hnd_price

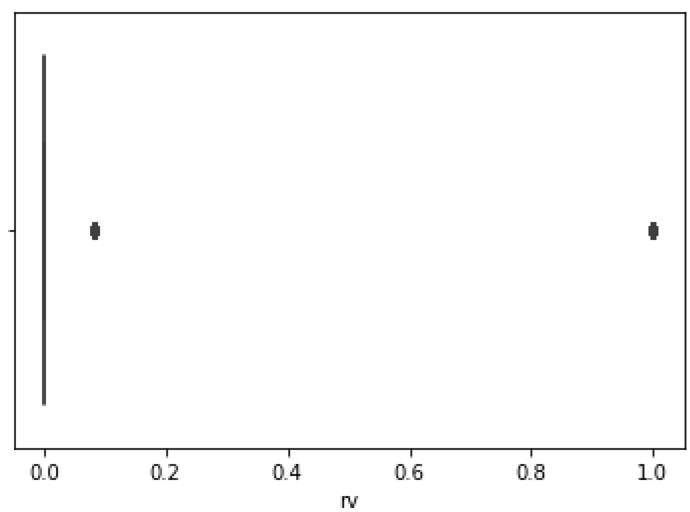

0
5
10
15
20
25
phones



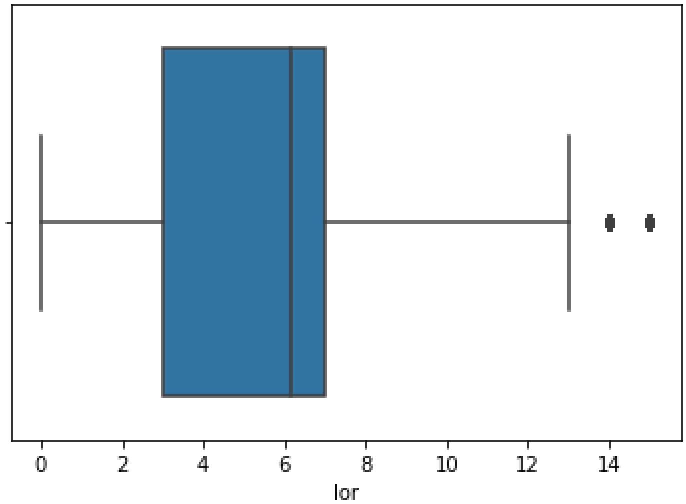
2
4
6
8
10
12
14
16
models

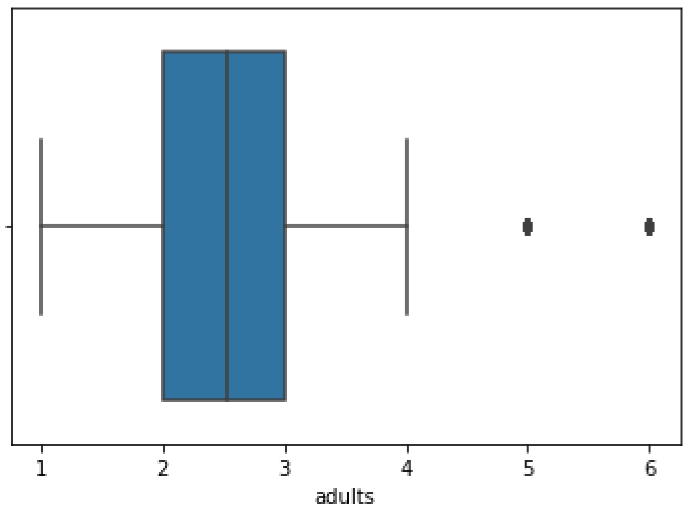


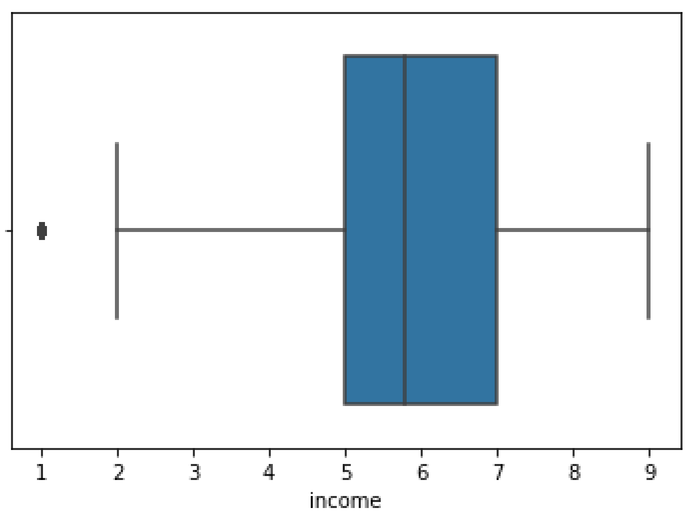
0.0
0.2
0.4
0.6
0.8
1.0
truck

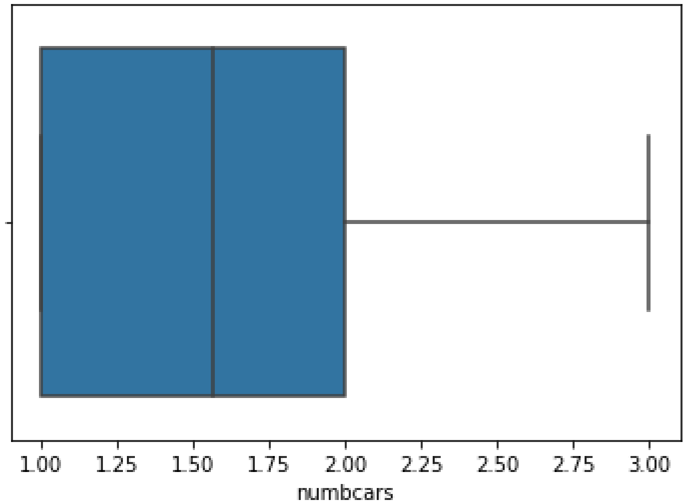


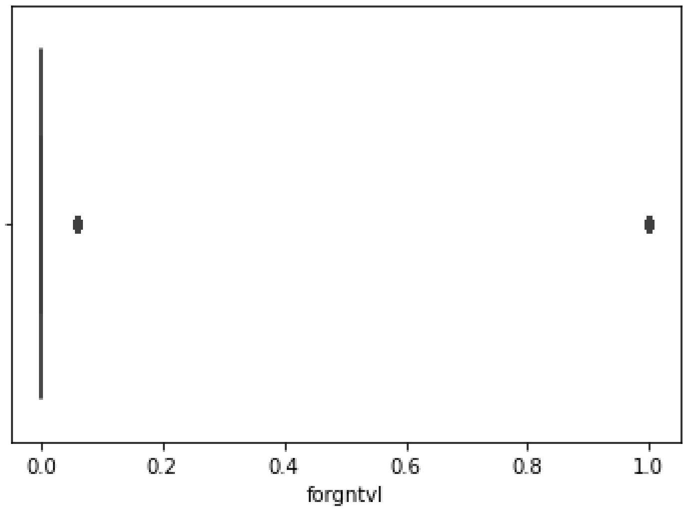
0.0
0.2
0.4
0.6
0.8
1.0
rv

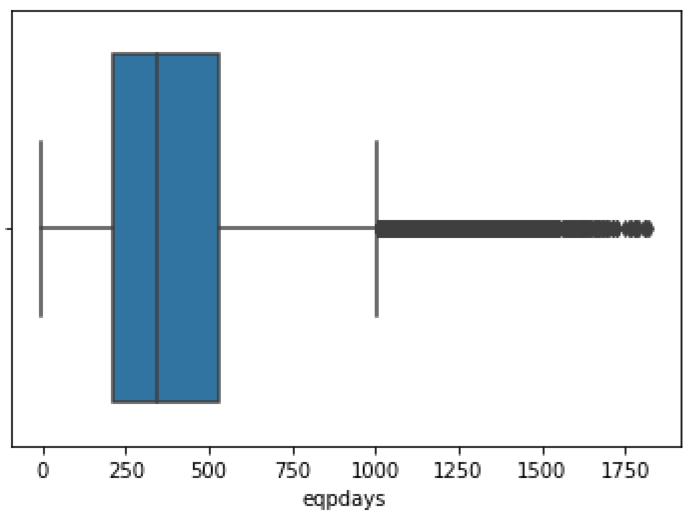

0
2
4
6
8
10
12
14
lor


1
2
3
4
5
6
adults


1
2
3
4
5
6
7
8
9
income







In [15]:

```
#Converting the Categorical data to one hot encoding
cat_vals = get_categorical_columns(cust_data_df)
print(cat_vals)

from sklearn.preprocessing import LabelEncoder

l_c=LabelEncoder()

def Label_Enoder(column):
    for i in range (len(column)):
        cust_data_df[column[i]]=l_c.fit_transform(cust_data_df[column[i]])
    return cust_data_df[column]

Label_Enoder(cat_vals)
```

```
['new_cell', 'crclscod', 'asl_flag', 'prizm_social_one', 'area', 'dualband', 'refurb_new', 'hnd_
webcap', 'ownrent', 'dwlltype', 'marital', 'infobase', 'HHstatin', 'dwllsize', 'ethnic', 'kid0_
2', 'kid3_5', 'kid6_10', 'kid11_15', 'kid16_17', 'creditcd']
```

Out[15]:

	new_cell	crclscod	asl_flag	prizm_social_one	area	dualband	refurb_new	hnd_webcap	ownrent	dwlltyp
0	0	0	0	2	14	3	0	2	0	
1	2	0	0	4	11	3	0	3	0	
2	0	0	0	2	9	0	1	3	0	
3	0	0	0	5	16	0	0	1	2	
4	2	0	0	5	9	3	1	3	0	
...	...	...	...	...	...	...	...	...	...	
18110	0	12	1	0	17	3	0	2	2	
18111	1	0	0	3	8	0	0	2	2	
18112	1	0	0	4	8	0	0	1	0	
18113	0	4	0	4	17	0	0	1	2	
18114	1	0	0	2	8	1	0	2	0	

18115 rows × 21 columns

In [16]:

```
#Removing unfilled rows after impuding
cust_data_df = cust_data_df.dropna()
print ("num of pepole who stay at the telecom company: "+
       str(cust_data_df[(cust_data_df['churn'] ==0) ].count()[1]))
print ("num of pepole who leave the telecom company: "+
       str(cust_data_df[(cust_data_df['churn'] ==1) ].count()[1]))
print(f'The dataset has {cust_data_df.shape[0]} rows and {cust_data_df.shape[1]} columns after dı
```

```
num of pepole who stay at the telecom company: 8682
num of pepole who leave the telecom company: 9433
The dataset has 18115 rows and 99 columns after dropping na rows
```

In [17]:

```
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
#Splitting data into features and labels where X are features and y is the label
#Normalizing numerical features using MinMax Scaler

#Performing oversampling to balance the dataset
count_0, count_1 = cust_data_df.churn.value_counts()

df_class_0 = cust_data_df[cust_data_df['churn']==0]
df_class_1 = cust_data_df[cust_data_df['churn']==1]

class_0_ = df_class_0
class_1_ = df_class_1.sample(count_0, replace=True)

df_balanced_1= pd.concat([class_0_,class_1_], axis=0)
df_balanced_1

#Normalizing numerical columns for better performance
scaler = MinMaxScaler()
df_balanced_1[get_numerical_columns(df_balanced_1)] = scaler.fit_transform(
                    df_balanced_1[get_numerical_columns(df_balanced_1)])

X = df_balanced_1.drop("churn", axis=1)
```

```
y = df_balanced_1["churn"]

print('Printing the first feature values')
print('The shape ', X.shape)
print(X.head())

print('Printing the label values')
print('The shape ', y.shape)

print(y.head())
```

```
Printing the first feature values
The shape  (18115, 98)
    rev_Mean  mou_Mean  totmrc_Mean  da_Mean  ovrmou_Mean  ovrrev_Mean  \
0   0.486448  0.272074     0.490126    0.000     0.205811     0.216206
4   0.309506  0.053730     0.359583    0.125     0.000000     0.000000
7   0.477674  0.082307     0.533757    0.000     0.000000     0.000000
12  0.307144  0.085729     0.446583    0.125     0.000000     0.000000
13  0.367151  0.085216     0.315952    0.000     0.288136     0.319985

    vceovr_Mean  datovr_Mean  roam_Mean  change_mou  ...  dwllsize  forgntvl  \
0      0.216206          0.0        0.0    0.607552  ...  0.066667       0.0
4      0.000000          0.0        0.0    0.554943  ...  1.000000       0.0
7      0.000000          0.0        0.0    0.533305  ...  0.000000       0.0
12     0.000000          0.0        0.0    0.567246  ...  0.066667       0.0
13     0.319985          0.0        0.0    0.344506  ...  0.933333       0.0

    ethnic  kid0_2  kid3_5  kid6_10  kid11_15  kid16_17  creditcd   eqpdays
0   0.5625     0.0     0.0      0.0       0.0       0.0       1.0  0.086053
4   0.8750     0.0     0.0      0.0       0.0       0.0       1.0  0.204748
7   0.3750     0.0     0.0      1.0       0.0       0.0       1.0  0.355094
12  0.5625     0.0     0.0      0.0       0.0       0.0       1.0  0.309594
13  0.1875     0.0     0.0      0.0       0.0       0.0       1.0  0.087043

[5 rows x 98 columns]
Printing the label values
The shape  (18115,)
0     0.0
4     0.0
7     0.0
12    0.0
13    0.0
Name: churn, dtype: float64
```

In [18]:
```
!pip install mrmr_selection
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simpl
e/
Collecting mrmr_selection
  Downloading mrmr_selection-0.2.5-py3-none-any.whl (24 kB)
Requirement already satisfied: numpy>=1.18.1 in /usr/local/lib/python3.8/dist-packages (from mrm
r_selection) (1.21.6)
Requirement already satisfied: pandas>=1.0.3 in /usr/local/lib/python3.8/dist-packages (from mrm
r_selection) (1.3.5)
Collecting sklearn
  Downloading sklearn-0.0.post1.tar.gz (3.6 kB)
Requirement already satisfied: scipy in /usr/local/lib/python3.8/dist-packages (from mrmr_select
ion) (1.7.3)
Requirement already satisfied: joblib in /usr/local/lib/python3.8/dist-packages (from mrmr_selec
tion) (1.2.0)
Collecting category-encoders
  Downloading category_encoders-2.5.1.post0-py2.py3-none-any.whl (72 kB)
     |████████████████████████████████| 72 kB 938 kB/s
Requirement already satisfied: jinja2 in /usr/local/lib/python3.8/dist-packages (from mrmr_selec
tion) (2.11.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.8/dist-packages (from mrmr_selecti
on) (4.64.1)
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.8/dist-packages
(from pandas>=1.0.3->mrmr_selection) (2.8.2)
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.8/dist-packages (from pand
as>=1.0.3->mrmr_selection) (2022.6)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.8/dist-packages (from python-d
ateutil>=2.7.3->pandas>=1.0.3->mrmr_selection) (1.15.0)
Requirement already satisfied: statsmodels>=0.9.0 in /usr/local/lib/python3.8/dist-packages (fro
m category-encoders->mrmr_selection) (0.12.2)
Requirement already satisfied: scikit-learn>=0.20.0 in /usr/local/lib/python3.8/dist-packages (f
rom category-encoders->mrmr_selection) (1.0.2)
Requirement already satisfied: patsy>=0.5.1 in /usr/local/lib/python3.8/dist-packages (from cate
gory-encoders->mrmr_selection) (0.5.3)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.8/dist-packages (f
rom scikit-learn>=0.20.0->category-encoders->mrmr_selection) (3.1.0)
Requirement already satisfied: MarkupSafe>=0.23 in /usr/local/lib/python3.8/dist-packages (from
jinja2->mrmr_selection) (2.0.1)
Building wheels for collected packages: sklearn
  Building wheel for sklearn (setup.py) ... done
  Created wheel for sklearn: filename=sklearn-0.0.post1-py3-none-any.whl size=2344 sha256=388cc7
3cdeb53615605635dfc56631c7a284cfb27f69e5685eafcdd8369d208b
  Stored in directory: /root/.cache/pip/wheels/14/25/f7/1cc0956978ae479e75140219088deb7a36f60459
df242b1a72
Successfully built sklearn
Installing collected packages: sklearn, category-encoders, mrmr-selection
Successfully installed category-encoders-2.5.1.post0 mrmr-selection-0.2.5 sklearn-0.0.post1
```

In [19]:

```
#Performing feature selection

from mrmr import mrmr_classif

selected_features = mrmr_classif(X, y, K=44)
print(selected_features)
```

```
100%|██████████| 44/44 [00:10<00:00,  4.22it/s]
['hnd_price', 'change_mou', 'totcalls', 'dwllsize', 'eqpdays', 'mou_cvce_Mean', 'uniqsubs', 'dua
lband', 'mou_Mean', 'adjqty', 'marital', 'refurb_new', 'totmrc_Mean', 'ethnic', 'mou_opkv_Mean',
'asl_flag', 'mouiwylisv_Mean', 'ownrent', 'threeway_Mean', 'adjrev', 'mou_peav_Mean', 'months',
'infobase', 'iwylis_vce_Mean', 'totrev', 'avg3mou', 'comp_vce_Mean', 'numbcars', 'dwlltype', 'to
tmou', 'complete_Mean', 'adjmou', 'owylis_vce_Mean', 'peak_vce_Mean', 'HHstatin', 'plcd_vce_Mea
n', 'avgqty', 'attempt_Mean', 'creditcd', 'models', 'mou_rvce_Mean', 'mouowylisv_Mean', 'da_Mea
n', 'opk_vce_Mean']
```

In [20]:
```python
from sklearn.model_selection import train_test_split

#https://towardsdatascience.com/predicting-customer-churn-using-logistic-regression-c6076f37eaca
final_df = X.filter(selected_features)
X_train, X_test, y_train, y_test = train_test_split(final_df,
                                                    y,
                                                    test_size=0.2,
                                                    random_state=100)
```

In [21]:
```python
#https://towardsdatascience.com/predicting-customer-churn-using-logistic-regression-c6076f37eaca
#Logistic Regression
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, recall_score
from sklearn.metrics import classification_report, confusion_matrix

clf_logi=LogisticRegression(fit_intercept = False, C = 1e12, solver = 'liblinear')
clf_logi.fit(X_train, y_train)

y_pred_logi = clf_logi.predict(X_test)

print('Accuracy for Logistic Regression:', clf_logi.score(X_test, y_test))

print(classification_report(y_test, y_pred_logi))
```

```
Accuracy for Logistic Regression: 0.6075075903947005
              precision    recall  f1-score   support

         0.0       0.61      0.55      0.58      1772
         1.0       0.61      0.66      0.63      1851

    accuracy                           0.61      3623
   macro avg       0.61      0.61      0.61      3623
weighted avg       0.61      0.61      0.61      3623
```

In [ ]:
```python
from sklearn.model_selection import cross_validate
#Performing 10-fold cross validation and using these scoring parameters.

def perform_cross_validation(estimator_name):
  _scoring = ['accuracy', 'precision', 'recall', 'f1']
  results = cross_validate(estimator=estimator_name,
                               X=X,
                               y=y,
                               cv=10,
                               scoring=_scoring,
                               return_train_score=True)

  output_dict =  {"Training Accuracy scores": results['train_accuracy'],
                "Mean Training Accuracy": results['train_accuracy'].mean()*100,
                "Training Precision scores": results['train_precision'],
                "Mean Training Precision": results['train_precision'].mean(),
                "Training Recall scores": results['train_recall'],
                "Mean Training Recall": results['train_recall'].mean(),
                "Training F1 scores": results['train_f1'],
                "Mean Training F1 Score": results['train_f1'].mean(),
                "Validation Accuracy scores": results['test_accuracy'],
                "Mean Validation Accuracy": results['test_accuracy'].mean()*100,
                "Validation Precision scores": results['test_precision'],
                "Mean Validation Precision": results['test_precision'].mean(),
                "Validation Recall scores": results['test_recall'],
```

```
                    "Mean Validation Recall": results['test_recall'].mean(),
                    "Validation F1 scores": results['test_f1'],
                    "Mean Validation F1 Score": results['test_f1'].mean()
                    }
  return output_dict

perform_cross_validation(clf_logi)
```

Out[ ]:

```
{'Training Accuracy scores': array([0.62246212, 0.62743053, 0.62565172, 0.62319818, 0.62264614,
        0.61353042, 0.62052257, 0.61395976, 0.60457556, 0.61285574]),
 'Mean Training Accuracy': 61.868327642726605,
 'Training Precision scores': array([0.63081232, 0.63431176, 0.63078272, 0.62788923, 0.62683165,
        0.61854219, 0.62493219, 0.61826799, 0.60977969, 0.61844681]),
 'Mean Training Precision': 0.6240596555999184,
 'Training Recall scores': array([0.6631331 , 0.67196702, 0.67781835, 0.67840735, 0.68029214,
        0.67267373, 0.67844523, 0.67608952, 0.66831567, 0.66972909]),
 'Mean Training Recall': 0.6736871203515504,
 'Training F1 scores': array([0.64656905, 0.65259666, 0.65345523, 0.65217145, 0.65246865,
        0.64447328, 0.65059016, 0.64588725, 0.63770722, 0.64306718]),
 'Mean Training F1 Score': 0.6478986132079105,
 'Validation Accuracy scores': array([0.43487859, 0.47019868, 0.51103753, 0.53532009, 0.5673289
2,
        0.5963556 , 0.55438984, 0.57702927, 0.66261734, 0.64715627]),
 'Mean Validation Accuracy': 55.563121127570916,
 'Validation Precision scores': array([0.46975355, 0.49316171, 0.52377049, 0.54404145, 0.5727272
7,
        0.60153257, 0.55964912, 0.58202039, 0.67849462, 0.65932914]),
 'Mean Validation Precision': 0.5684480311505551,
 'Validation Recall scores': array([0.66702015, 0.65005302, 0.67690678, 0.66737288, 0.66737288,
        0.6659597 , 0.67656416, 0.6659597 , 0.66914104, 0.66702015]),
 'Mean Validation Recall': 0.6673370463899924,
 'Validation F1 scores': array([0.55127082, 0.56084172, 0.59057301, 0.59942912, 0.61643836,
        0.63210871, 0.61257801, 0.62116716, 0.67378537, 0.66315235]),
 'Mean Validation F1 Score': 0.6121344616515926}
```

In [22]:

```
##
## Neural Networks
##

import tensorflow as tf
from tensorflow import keras

#Neural Networks model
nn_model = keras.Sequential([
    keras.layers.BatchNormalization(),
    keras.layers.Dense(44, input_shape=(44,), activation='relu'),
    keras.layers.Dropout(0.25),
    keras.layers.BatchNormalization(),
    keras.layers.Dense(22,activation = 'relu'),
    keras.layers.Dropout(0.25),

    # we use sigmoid for binary output
    # output layer
    keras.layers.Dense(1, activation='sigmoid')
]
)

nn_model.compile(optimizer = tf.keras.optimizers.Adagrad(
    learning_rate=0.05, epsilon=0.01), loss='binary_crossentropy',
        metrics=['accuracy'])
```

In [23]:
```
nn_model.fit(X_train, y_train, epochs=25, verbose=2)
```

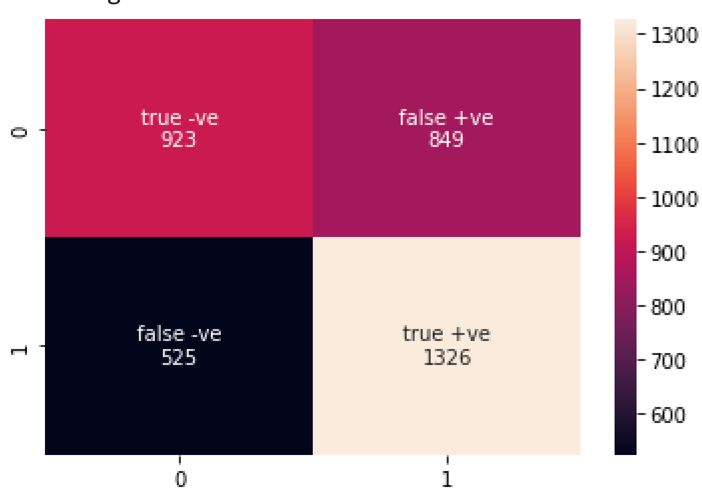
```
Epoch 1/25
453/453 - 4s - loss: 0.6895 - accuracy: 0.5655 - 4s/epoch - 10ms/step
Epoch 2/25
453/453 - 2s - loss: 0.6706 - accuracy: 0.5836 - 2s/epoch - 4ms/step
Epoch 3/25
453/453 - 2s - loss: 0.6661 - accuracy: 0.5905 - 2s/epoch - 3ms/step
Epoch 4/25
453/453 - 1s - loss: 0.6632 - accuracy: 0.5916 - 1s/epoch - 3ms/step
Epoch 5/25
453/453 - 1s - loss: 0.6592 - accuracy: 0.6016 - 1s/epoch - 3ms/step
Epoch 6/25
453/453 - 1s - loss: 0.6603 - accuracy: 0.6018 - 1s/epoch - 3ms/step
Epoch 7/25
453/453 - 1s - loss: 0.6591 - accuracy: 0.6032 - 1s/epoch - 3ms/step
Epoch 8/25
453/453 - 1s - loss: 0.6594 - accuracy: 0.6005 - 1s/epoch - 3ms/step
Epoch 9/25
453/453 - 1s - loss: 0.6594 - accuracy: 0.6041 - 1s/epoch - 3ms/step
Epoch 10/25
453/453 - 1s - loss: 0.6577 - accuracy: 0.6057 - 1s/epoch - 3ms/step
Epoch 11/25
453/453 - 1s - loss: 0.6576 - accuracy: 0.6036 - 1s/epoch - 3ms/step
Epoch 12/25
453/453 - 1s - loss: 0.6548 - accuracy: 0.6069 - 1s/epoch - 3ms/step
Epoch 13/25
453/453 - 1s - loss: 0.6540 - accuracy: 0.6112 - 1s/epoch - 3ms/step
Epoch 14/25
453/453 - 1s - loss: 0.6508 - accuracy: 0.6154 - 1s/epoch - 3ms/step
Epoch 15/25
453/453 - 1s - loss: 0.6540 - accuracy: 0.6107 - 1s/epoch - 3ms/step
Epoch 16/25
453/453 - 1s - loss: 0.6506 - accuracy: 0.6104 - 1s/epoch - 3ms/step
Epoch 17/25
453/453 - 1s - loss: 0.6523 - accuracy: 0.6140 - 1s/epoch - 3ms/step
Epoch 18/25
453/453 - 1s - loss: 0.6488 - accuracy: 0.6152 - 1s/epoch - 3ms/step
Epoch 19/25
453/453 - 1s - loss: 0.6496 - accuracy: 0.6200 - 1s/epoch - 3ms/step
Epoch 20/25
453/453 - 1s - loss: 0.6489 - accuracy: 0.6122 - 1s/epoch - 3ms/step
Epoch 21/25
453/453 - 1s - loss: 0.6504 - accuracy: 0.6137 - 1s/epoch - 3ms/step
Epoch 22/25
453/453 - 1s - loss: 0.6485 - accuracy: 0.6155 - 1s/epoch - 3ms/step
Epoch 23/25
453/453 - 1s - loss: 0.6493 - accuracy: 0.6180 - 1s/epoch - 3ms/step
Epoch 24/25
453/453 - 1s - loss: 0.6484 - accuracy: 0.6159 - 1s/epoch - 3ms/step
Epoch 25/25
453/453 - 1s - loss: 0.6491 - accuracy: 0.6146 - 1s/epoch - 3ms/step
```

Out[23]:
```
<keras.callbacks.History at 0x7f2200639670>
```

In [24]:
```
#getting the probability values and applying thresholding
import numpy as np
preds = nn_model.predict(X_test)
final_output = []
for pred in preds:
  if pred > 0.5:
    final_output.append(1)
```

```
    else:
      final_output.append(0)

#Printing evaluation metrics

print('Printing the accuracy of the model')
print(accuracy_score(y_test, final_output))
print('Printing Classification report')
print(classification_report(y_test, final_output))
print('Printing the recall score')
print(recall_score(y_test, final_output))
print('\nPrinting the confusion Matrix of the model')
cf_matrix = confusion_matrix(y_test, final_output)

def plot_confusion_matrix(matrix):
    """
        Plots confusion matrix
    """
    grp_names = ['true -ve','false +ve','false -ve','true +ve']
    group_counts = ['{0:0.0f}'.format(value) for value in matrix.flatten()]
    labels = [f'{val1}\n{val2}' for val1, val2 in zip(grp_names,group_counts)]
    labels = np.asarray(labels).reshape(2,2)
    sns.heatmap(matrix, annot=labels, fmt='')
    plt.show()

plot_confusion_matrix(cf_matrix)
```

```
114/114 [==============================] - 0s 2ms/step
Printing the accuracy of the model
0.620756279326525
Printing Classification report
              precision    recall  f1-score   support

         0.0       0.64      0.52      0.57      1772
         1.0       0.61      0.72      0.66      1851

    accuracy                           0.62      3623
   macro avg       0.62      0.62      0.62      3623
weighted avg       0.62      0.62      0.62      3623

Printing the recall score
0.7163695299837926

Printing the confusion Matrix of the model
```



In [ ]:
```
##
## Random Forest
```

```
##

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

max_depth_values = [i for i in range (5,16)]

recall_values = []
for val in max_depth_values:
    print(val)
    rf_clf = RandomForestClassifier(max_depth=val)
    rf_clf.fit(X_train, y_train)

    y_pred_rf = rf_clf.predict(X_test)

    print('Printing the recall score')
    score = recall_score(y_test, y_pred_rf)
    print(score)
    recall_values.append(score)

print(recall_values)
```

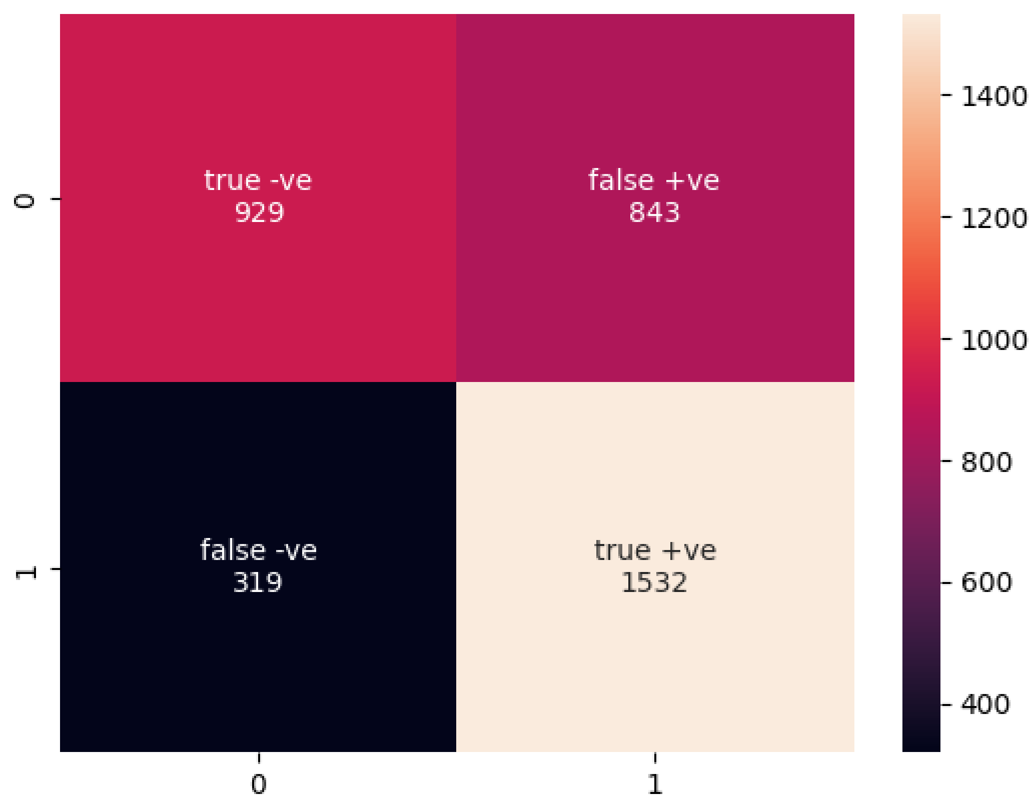
```
5
Printing the recall score
0.8627768773635872
6
Printing the recall score
0.8492706645056726
7
Printing the recall score
0.8276607239330092
8
Printing the recall score
0.8379254457050244
9
Printing the recall score
0.8319827120475418
10
Printing the recall score
0.8314424635332253
11
Printing the recall score
0.8292814694759589
12
Printing the recall score
0.8217179902755267
13
Printing the recall score
0.8119935170178282
14
Printing the recall score
0.8206374932468936
15
Printing the recall score
0.8146947595894112
[0.8627768773635872, 0.8492706645056726, 0.8276607239330092, 0.8379254457050244, 0.8319827120475
418, 0.8314424635332253, 0.8292814694759589, 0.8217179902755267, 0.8119935170178282, 0.820637493
2468936, 0.8146947595894112]
```

In [ ]:

```
#From the above values we can say that max_depth 10 is optimal as after that
#the recall values kind of stabilizes
```

```python
rf_clf = RandomForestClassifier(max_depth=10)
rf_clf.fit(X_train, y_train)

y_pred_rf = rf_clf.predict(X_test)

print('Printing the accuracy of the model')
print(accuracy_score(y_test, y_pred_rf))
print('Printing Classification report')
print(classification_report(y_test, y_pred_rf))
print('Printing the recall score')
score = recall_score(y_test, y_pred_rf)
print(score)

cf_matrix = confusion_matrix(y_test, y_pred_rf)
plot_confusion_matrix(cf_matrix)

perform_cross_validation(rf_clf)

# !pip install lightgbm
```

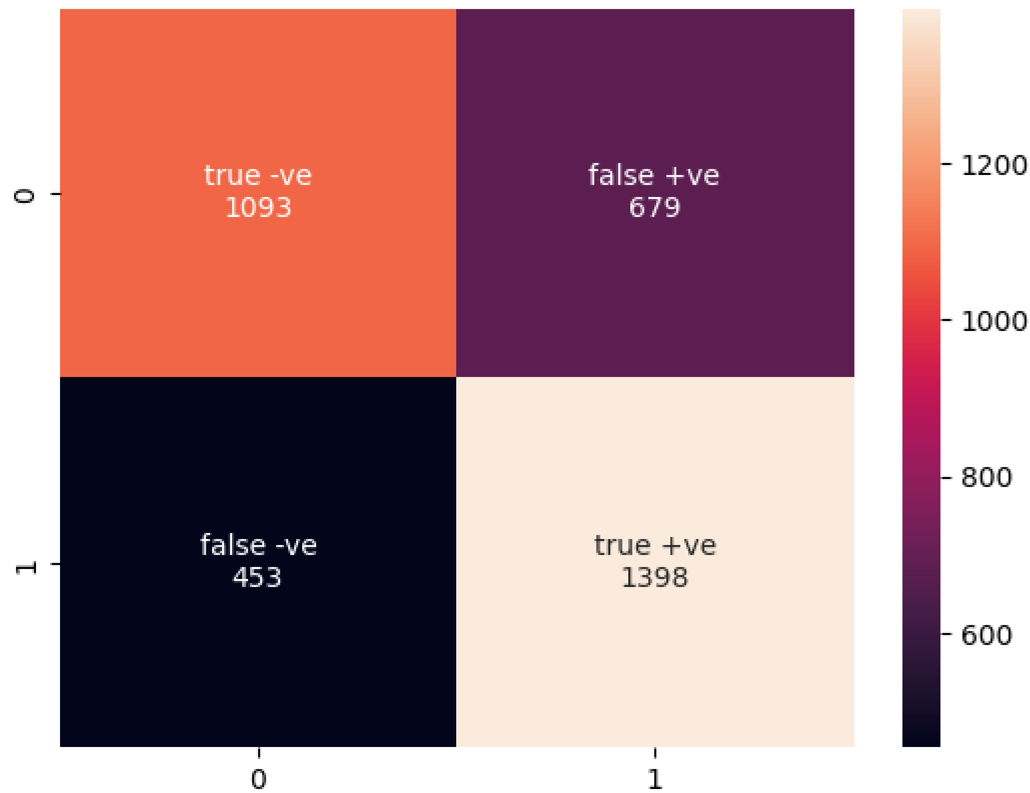
```
Printing the accuracy of the model
0.6792713221087496
Printing Classification report
              precision    recall  f1-score   support

         0.0       0.74      0.52      0.62      1772
         1.0       0.65      0.83      0.73      1851

    accuracy                           0.68      3623
   macro avg       0.69      0.68      0.67      3623
weighted avg       0.69      0.68      0.67      3623

Printing the recall score
0.8276607239330092
```



Out[ ]:
```
{'Training Accuracy scores': array([0.85622278, 0.84370975, 0.83162608, 0.83420229, 0.82512421,
        0.82090285, 0.84396467, 0.84089794, 0.84108194, 0.84739941]),
 'Mean Training Accuracy': 83.85131923767982,
 'Training Precision scores': array([0.8250476 , 0.80125735, 0.77824065, 0.78092834, 0.77294733,
        0.76742846, 0.80454825, 0.80229696, 0.80625065, 0.81616098]),
 'Mean Training Precision': 0.7955106565971357,
 'Training Recall scores': array([0.91872792, 0.93074205, 0.94628343, 0.94734362, 0.94039345,
        0.94134276, 0.92508834, 0.92155477, 0.91460542, 0.91248528]),
 'Mean Training Recall': 0.9298567022437823,
 'Training F1 scores': array([0.86937138, 0.86115955, 0.85407474, 0.85612392, 0.84848807,
        0.84553534, 0.86061801, 0.85780068, 0.85701672, 0.86163942]),
 'Mean Training F1 Score': 0.8571827821313741,
 'Validation Accuracy scores': array([0.48289183, 0.53642384, 0.57560706, 0.57615894, 0.5651214
1,
        0.54555494, 0.56156819, 0.62451684, 0.72335726, 0.80839315]),
 'Mean Validation Accuracy': 59.99593482556318,
 'Validation Precision scores': array([0.50196335, 0.53568954, 0.55964554, 0.56179775, 0.5546984
6,
        0.54126547, 0.55302491, 0.60528423, 0.71128107, 0.83037694]),
 'Mean Validation Precision': 0.5955027255458151,
 'Validation Recall scores': array([0.81336161, 0.81972428, 0.86970339, 0.84745763, 0.83792373,
        0.83457052, 0.82396607, 0.80169671, 0.78897137, 0.79427359]),
 'Mean Validation Recall': 0.8231648902708629,
 'Validation F1 scores': array([0.6208013 , 0.64794635, 0.68104521, 0.67567568, 0.66751055,
        0.65665415, 0.66183986, 0.68978102, 0.74811463, 0.81192412]),
 'Mean Validation F1 Score': 0.6861292868750374}
```

In [ ]:
```
##
## LGMM
##

import lightgbm as lgb

lgm_clf = lgb.LGBMClassifier()
lgm_clf.fit(X_train, y_train)

y_pred_lgm = lgm_clf.predict(X_test)

print('Printing the accuracy of the model')
print(accuracy_score(y_test, y_pred_lgm))
print('Printing Classification report')
print(classification_report(y_test, y_pred_lgm))
print('Printing the recall score')
print(recall_score(y_test, y_pred_lgm))

cf_matrix = confusion_matrix(y_test, y_pred_lgm)
plot_confusion_matrix(cf_matrix)

perform_cross_validation(lgm_clf)
```

```
Printing the accuracy of the model
0.68755175269114
Printing Classification report
              precision    recall  f1-score   support

         0.0       0.71      0.62      0.66      1772
         1.0       0.67      0.76      0.71      1851

    accuracy                           0.69      3623
   macro avg       0.69      0.69      0.69      3623
weighted avg       0.69      0.69      0.69      3623

Printing the recall score
0.7552674230145867
```



Out[ ]:
```
{'Training Accuracy scores': array([0.84039747, 0.83782126, 0.8402748 , 0.84328038, 0.83782126,
        0.8408366 , 0.83930324, 0.83899657, 0.83353778, 0.8294897 ]),
 'Mean Training Accuracy': 83.81759052696673,
 'Training Precision scores': array([0.83363554, 0.82015334, 0.81793625, 0.82173064, 0.81645912,
        0.82273076, 0.82203204, 0.82264275, 0.82743764, 0.82238031]),
 'Mean Training Precision': 0.8227138396819548,
 'Training Recall scores': array([0.8664311 , 0.8819788 , 0.89174225, 0.89268465, 0.88820827,
        0.88504122, 0.88244994, 0.88068316, 0.85959953, 0.85783274]),
 'Mean Training Recall': 0.8786651664920487,
 'Training F1 scores': array([0.84971699, 0.84994325, 0.85324617, 0.85573937, 0.85082374,
        0.85274925, 0.85117019, 0.8506741 , 0.84321202, 0.8397325 ]),
 'Mean Training F1 Score': 0.8497007574750274,
 'Validation Accuracy scores': array([0.45198675, 0.52980132, 0.58664459, 0.58719647, 0.5706401
8,
        0.54555494, 0.56101601, 0.61844285, 0.68856985, 0.74323578]),
 'Mean Validation Accuracy': 58.83088752448552,
 'Validation Precision scores': array([0.48310811, 0.53372869, 0.57238307, 0.57716535, 0.5650470
2,
        0.54477612, 0.556231  , 0.61033275, 0.68743818, 0.75643777]),
 'Mean Validation Precision': 0.5886648066060822,
 'Validation Recall scores': array([0.75821845, 0.76352068, 0.81673729, 0.77648305, 0.76377119,
        0.77412513, 0.77624602, 0.73913043, 0.73700954, 0.747614  ]),
 'Mean Validation Recall': 0.7652855788414185,
 'Validation F1 scores': array([0.59017747, 0.62827225, 0.67306853, 0.66214995, 0.64954955,
        0.63950942, 0.64807437, 0.66858513, 0.71136131, 0.752     ]),
 'Mean Validation F1 Score': 0.6622747979319912}
```

In [ ]:
```
clf_types = ['Random Forest', 'LGM Classifier', 'Logistic Regression']
models_done = set()
dict_models = {'Random Forest': rf_clf,
               'LGM Classifier': lgm_clf,
               'Logistic Regression': clf_logi}
# Classification comparison result
from mlxtend.evaluate import paired_ttest_5x2cv

for clf_1 in range(len(clf_types)):
    for clf_2 in range(clf_1+1, len(clf_types)):
        if clf_1 != clf_2:
          print(f'The statistical analysis is applied for {clf_types[clf_1]} and {clf_types[clf_
          statistic, p_value = paired_ttest_5x2cv(
              estimator1=dict_models[clf_types[clf_1]],
              estimator2=dict_models[clf_types[clf_2]],
              X=X.to_numpy(),
              y=y.to_numpy()
              )
          print("Statistic:", statistic)
          print("PValue:", p_value)
```

```
The statistical analysis is applied for Random Forest and LGM Classifier
Statistic: -4.286002041148445
PValue: 0.007818733432560726
The statistical analysis is applied for Random Forest and Logistic Regression
Statistic: 7.68133069515328
PValue: 0.0005961067326729515
The statistical analysis is applied for LGM Classifier and Logistic Regression
Statistic: 11.708557521988473
PValue: 7.987851061802866e-05
```