Vnedor Log Analysis - Clutering Costomers Using Unsupervised Machine Learning

Shigeki Kamata 2020/09/29

Table of Contents

Introduction

Let's say you are an analyst at a company that provides various online services in the form of sending data at the request of the clients. The company receives millions of online requests and the transaction log is stored in the company's database. You are given a sample vendor log for a specific month and are asked to gain some insights about the customers, products and transactions. The stakeholders might be wondering which products were successful and how the volume of transactions changes based on the day. They also might be wondering whether there are some behavioral tendencies for each customer and if they can create a new sales plan based on those tendencies. As an analyst, your job is to answer those questions using statistics and machine learning algorithms.

In this article I will attempt to analyze the behaviors of each customer based on the information provided by the vendor log. The analysis consists of two parts. The first part studies the overall characteristics of the data set using basic statistical information and visualization. The second part focuses on customer-specific behaviors. This part has two sections. First, I took a closer look at the behaviors of the top three customers, which makes up 60% of the total transaction activity. Then I attempted to divide the customers into clusters based on their March 2020 behaviors and study the characteristics of each cluster. I utilized two unsupervised machine learning algorithms (K-means and Hierarchical Clustering) to conduct this cluster analysis.

Basic Analysis

Data Preparation

First, we'll need to import the necessary library and load the data file below:

In [2]:
```
# Import the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
```

In [3]:
```
# Load the dataset
df = pd.read_csv('VendorLogs-raw_202003_copy.txt', sep = "\t", header = None)
# Name columns
df.columns = ['Date', 'CustID', 'TFound', 'TCount',
              'TDetail', 'TResult', 'TType']
# Limit the data to March of 2020
df = df.loc[(df['Date'] >= 20200301) & (df['Date'] <= 20200331)]
```

The following columns are located in the data set:

Date
CustID
TFound: How the request was handled; 0=unknown, 1=find, 2=not find, 3=not find filtered, 4=error and so on
TCount: The number of service transaction processed
TDetail: The detail of request
TResult: The result of each request
TType: The type of transaction

Next, let's take a look at the data set below:

In [4]:
```
df.head(10)
```

Out[4]:

	Date	CustID	TFound	TCount	TDetail	TResult	TType
0	20200301	1932	1	1	4012(2310,120,120,122),4030(1710,211)	1(1,1,1,1),1(2,1)	RT:r
1	20200301	2024	0	0	4030(1710,211,478,2805)	2(2,2,2,2)	RT:r
2	20200301	1980	0	0	4012(2310,601,120,122),4030(1710,211,2805)	1(5,1,1,1),2(2,2,2)	RT:r
3	20200301	1932	1	1	4012(2310,120,120,122),4030(1710,211)	1(1,1,1,1),1(2,1)	RT:r
4	20200301	1980	1	1	4012(2310,120,122),4030(1710,211)	1(1,1,1),1(2,1)	RT:r
5	20200301	1980	1	1	4012(2310,120,122),4030(1710,211)	1(1,1,1),1(2,1)	RT:r
6	20200301	1980	1	1	4012(2310,120,122),4030(1710,211,2805)	1(1,1,1),1(2,2,1)	RT:r
7	20200301	2024	0	0	4030(1710,211,478,2805)	2(2,2,2,2)	RT:r
8	20200301	1932	1	1	4012(2310,120,120,122),4030(1710,211)	1(1,1,1,1),1(2,1)	RT:r
9	20200301	1980	1	1	4012(2310,120,120,122),4030(1710,211)	1(1,1,1,1),1(2,1)	RT:r

In order to decipher if there is a day-of-the-week-related pattern in the transaction activity, let's append a day column to the data:

In [5]:
```
# Append day of week to df
from datetime import date
import calendar

def get_day_of_week(timestamp):
    year = int(timestamp / 10000)
    month =  int((timestamp / 100) % 100)
    day =  int(timestamp % 100)
    mydate = date(year, month, day)
    return calendar.day_name[mydate.weekday()]
day = df['Date'].map(get_day_of_week)
df.insert(loc=1, column='Day', value=day )
df.head(10)
```

Out[5]:

	Date	Day	CustID	TFound	TCount	TDetail	TResult	TType
0	20200301	Sunday	1932	1	1	4012(2310,120,120,122),4030(1710,211)	1(1,1,1,1),1(2,1)	RT:r
1	20200301	Sunday	2024	0	0	4030(1710,211,478,2805)	2(2,2,2,2)	RT:r
2	20200301	Sunday	1980	0	0	4012(2310,601,120,122),4030(1710,211,2805)	1(5,1,1,1),2(2,2,2)	RT:r
3	20200301	Sunday	1932	1	1	4012(2310,120,120,122),4030(1710,211)	1(1,1,1,1),1(2,1)	RT:r
4	20200301	Sunday	1980	1	1	4012(2310,120,122),4030(1710,211)	1(1,1,1),1(2,1)	RT:r
5	20200301	Sunday	1980	1	1	4012(2310,120,122),4030(1710,211)	1(1,1,1),1(2,1)	RT:r
6	20200301	Sunday	1980	1	1	4012(2310,120,122),4030(1710,211,2805)	1(1,1,1),1(2,2,1)	RT:r
7	20200301	Sunday	2024	0	0	4030(1710,211,478,2805)	2(2,2,2,2)	RT:r
8	20200301	Sunday	1932	1	1	4012(2310,120,120,122),4030(1710,211)	1(1,1,1,1),1(2,1)	RT:r
9	20200301	Sunday	1980	1	1	4012(2310,120,120,122),4030(1710,211)	1(1,1,1,1),1(2,1)	RT:r

Next, we'll check for any null data:

In [7]:
```
df.isnull().sum()
```

Out[7]:
```
Date            0
Day             0
CustID          0
TFound          0
TCount          0
TDetail    607244
TResult    607244
TType           0
dtype: int64
```

In [8]:
```
df[df['TDetail'].isnull()]
```

Out[8]:

	Date	Day	CustID	TFound	TCount	TDetail	TResult	TType
3463336	20200303	Tuesday	2169	0	0	NaN	NaN	RP:f
3464317	20200304	Wednesday	2169	0	0	NaN	NaN	RP:f
3464318	20200304	Wednesday	2169	0	0	NaN	NaN	RP:f
3464319	20200304	Wednesday	2169	0	0	NaN	NaN	RP:f
3465319	20200304	Wednesday	2169	0	0	NaN	NaN	RP:f
...	...	...	...	...	...	...	...	...
9084126	20200331	Tuesday	2707	0	0	NaN	NaN	RS:b
9084299	20200331	Tuesday	2707	0	0	NaN	NaN	RS:b
9087922	20200331	Tuesday	2707	0	0	NaN	NaN	RS:b
9089062	20200331	Tuesday	2707	0	0	NaN	NaN	RS:b
9089825	20200331	Tuesday	2707	0	0	NaN	NaN	RS:b

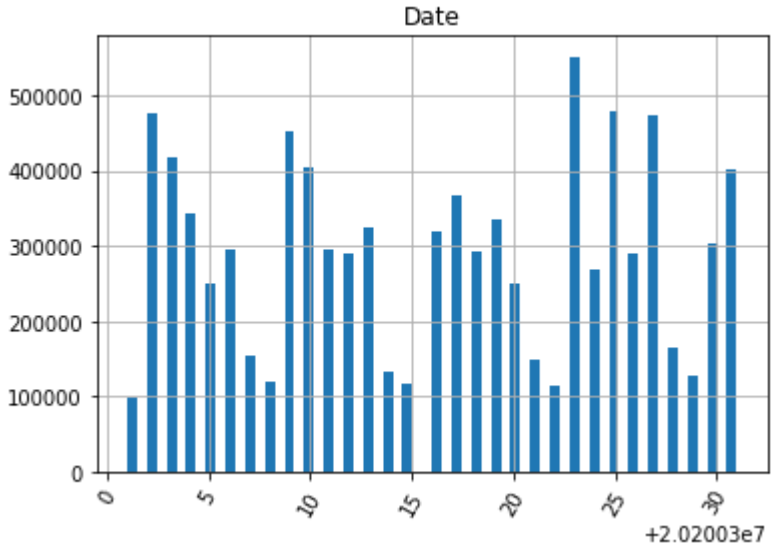
607244 rows × 8 columns

Based on these results, it looks like TDetail and TResult get NaN when TCount is 0, which makes sense because no transaction detail can be added if the requested data was not found.

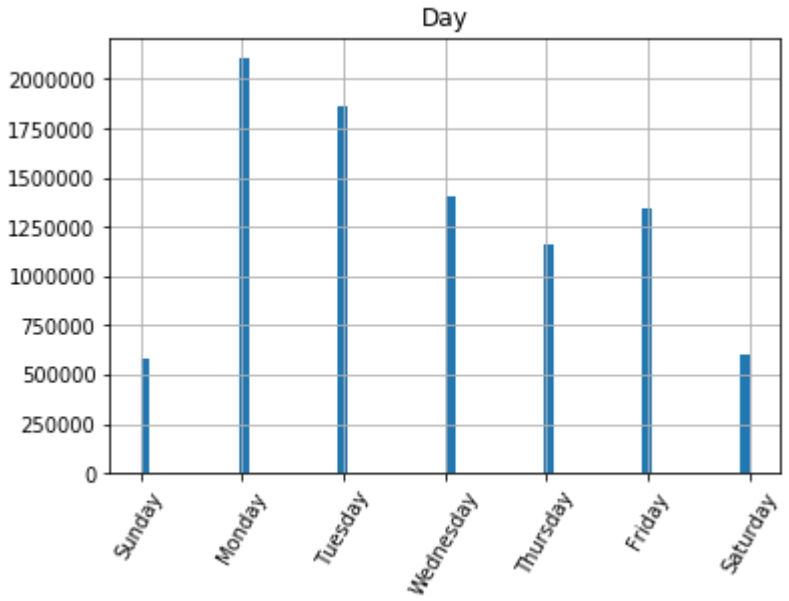
Next, let's observe some basic statistics for each column:

In [9]:
```
for col in ['Date', 'Day', 'CustID', 'TFound', 'TCount', 'TType']:
    print(col)
    print(df[col].astype(str).describe())
    print('Unique values:')
    print(df[col].astype(str).unique())
    # Plot Histogram
    if col in ['Date', 'Day', 'TType']:
        plt.hist(df[col], bins = 62 )
        plt.xticks(rotation=60)
        plt.title(col)
        plt.grid(True)
        plt.show()
    if col in ['CustID', 'TFound', 'TCount']:
        plot_data = df.groupby(col).Date.count().sort_values(axis = 0, ascending = False)
        plot_data = plot_data.head(10)
        plt.bar(plot_data.index.astype(str) , plot_data )
        plt.title(col)
        plt.xticks(rotation=60)
        plt.xlabel(col)
        plt.ylabel('Transaction Count')
        plt.grid(True)
        plt.show()
    print("--------------------------------------------------------------------------------")
```

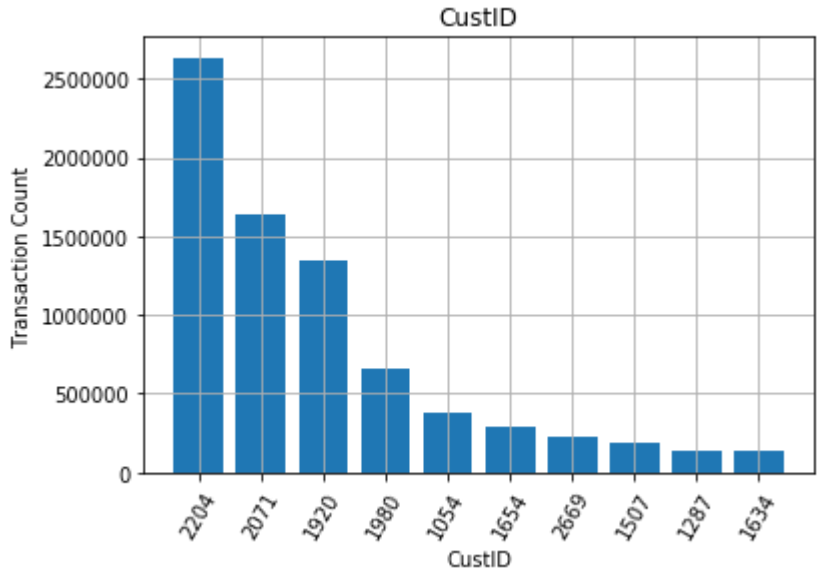
```
Date
count      9056295
unique          31
top       20200323
freq        551459
Name: Date, dtype: object
Unique values:
['20200301' '20200302' '20200303' '20200304' '20200305' '20200306'
 '20200307' '20200308' '20200309' '20200310' '20200311' '20200312'
 '20200313' '20200314' '20200315' '20200316' '20200317' '20200318'
 '20200319' '20200320' '20200321' '20200322' '20200323' '20200324'
 '20200325' '20200326' '20200327' '20200328' '20200329' '20200330'
 '20200331']
```



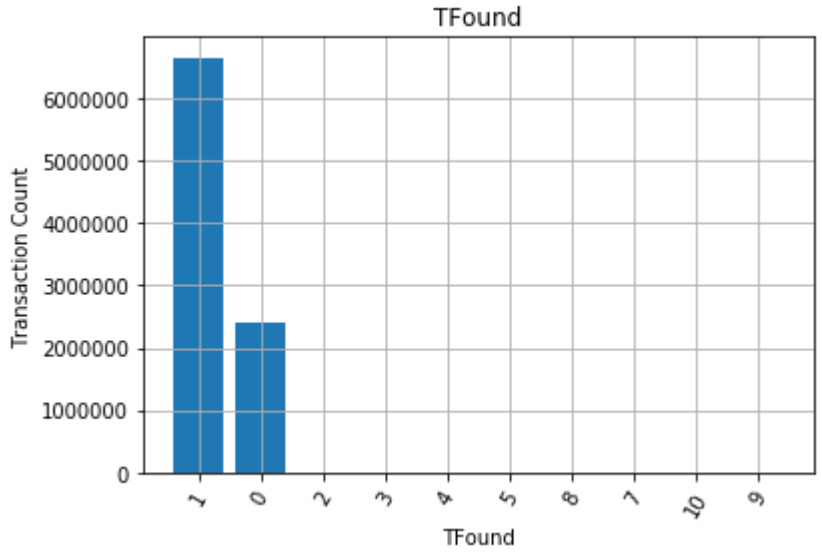

```
--------------------------------------------------------------------------------
Day
count     9056295
unique          7
top        Monday
freq      2105881
Name: Day, dtype: object
Unique values:
['Sunday' 'Monday' 'Tuesday' 'Wednesday' 'Thursday' 'Friday' 'Saturday']
```



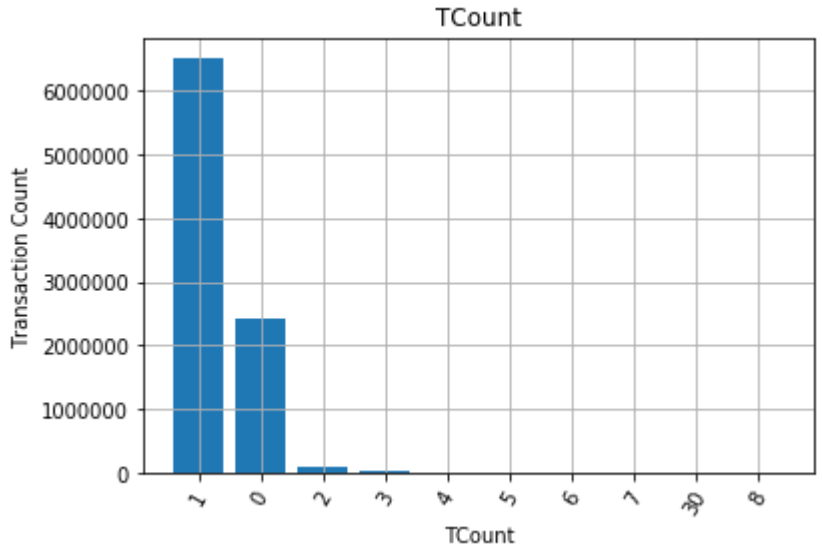

```
--------------------------------------------------------------------------------
CustID
count     9056295
unique         91
top          2204
freq      2634629
Name: CustID, dtype: object
Unique values:
['1932' '2024' '1980' '2184' '2159' '2070' '2117' '2180' '2204' '2056'
 '2216' '2080' '1927' '2223' '1587' '1628' '1751' '1596' '1752' '260'
 '198' '1655' '1585' '1603' '1653' '1654' '1563' '1734' '2169' '1793'
 '1287' '1675' '1692' '1794' '1569' '1661' '1920' '1967' '2071' '1634'
 '1507' '1010' '2669' '2276' '2493' '2662' '1054' '1' '2413' '2424' '2520'
 '1717' '2187' '2509' '1768' '1095' '2704' '2707' '2362' '1749' '2282'
 '2046' '1885' '1923' '2537' '1021' '2623' '2197' '1173' '2684' '2310'
 '1017' '1285' '2343' '2382' '2507' '2680' '2588' '1052' '2574' '2245'
 '1074' '1966' '1018' '1205' '2256' '2593' '2481' '2405' '2498' '2633']
```



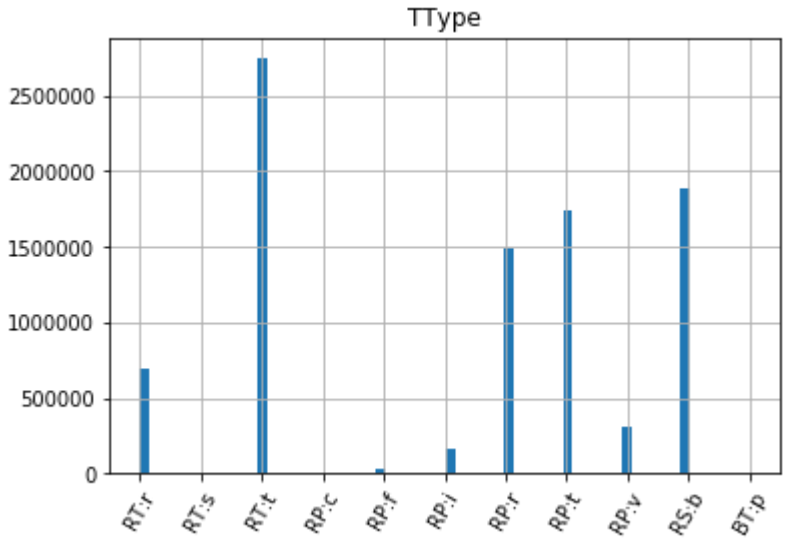

```
--------------------------------------------------------------------------------
TFound
count     9056295
unique       1958
top             1
freq      6643888
Name: TFound, dtype: object
Unique values:
['1' '0' '1186' ... '41107' '6801' '54685']
```




```
--------------------------------------------------------------------------------
TCount
count     9056295
unique         32
top             1
freq      6517639
Name: TCount, dtype: object
Unique values:
['1' '0' '7' '3' '2' '5' '10' '4' '22' '16' '8' '6' '9' '17' '14' '12'
 '11' '24' '18' '28' '15' '13' '21' '30' '20' '27' '26' '19' '23' '25'
 '29' '50']
```




```
--------------------------------------------------------------------------------
TType
count     9056295
unique         11
top          RT:t
freq      2746296
Name: TType, dtype: object
Unique values:
['RT:r' 'RT:s' 'RT:t' 'RP:c' 'RP:f' 'RP:i' 'RP:r' 'RP:t' 'RP:v' 'RS:b'
 'BT:p']
```



Our set defined values for TFound include the following: 0=unknown, 1=find, 2=no find, 3=no find filtered, 4=error, 5=timeout and 6=not completed
Since TFound doesn't exist within these set values, we can convert the TFound undefined values to 0 (=unknown).

In [10]:
```
# Find undefined TFound Value and replace with zeros
illigal_values = list(df.loc[(df.TFound < 0) | (df.TFound > 6)].TFound)
df.TFound = df.TFound.replace(to_replace= illigal_values, value = 0)
```

Let's take a look at the unique values in the TFound column again:

In [11]:
```
# Check the result
list(df.TFound.unique())
```

Out[11]:
```
[1, 0, 2, 4, 3, 6, 5]
```

CustID-wise Analysis

In this next section, we will attempt to divide the customers in this sameple into groups based on their behaviors in the month. In order to archive this, we will utilize two unsupervised machine learning algorithms: K-means and Hierarchical Clustering. We will review the basic methodologies of these algorithms as we apply them to the data set below.

Data Preparation

Since we are now focusing on the characteristics of each customer's behavior, we wil need another version of the VendorLog data set; one in which the data is grouped by CustID and each column corresponds to the number of transaction requests that satisfy the column requirement. For example, in the dataset below we can see that for CustID 1 in row 0, there were 80 transactions on March 3rd, 2020, and the total number of transactions in March 2020 was 49,595 (shown in the TCount column).

In [12]:
```
# Import the dataset and add columns
df = pd.read_csv('VendorLogs_custwise.csv')
df['TotalRequest'] = df.iloc[:,33:40].apply(lambda row: sum(row), axis =1)
df['Percentage'] = df['TotalRequest'].apply(lambda value: str(round(value / df['TotalRequest'].sum() * 100, 1
)) + " %")
col_names = list(df.columns)
col_names[0] = 'CustID'
df.columns = col_names
df.head(10)
```

Out[12]:

	CustID	TCount	20200301	20200302	20200303	20200304	20200305	20200306	20200307	20200308	20200309	20200310	20200311
0	1	49595	0	0	80	0	2	0	0	0	2	2	0
1	198	39	0	0	2	0	0	5	9	0	1	1	0
2	260	31	0	4	0	0	0	5	0	0	0	2	0
3	1010	48098	0	16671	0	0	0	18681	0	0	0	1400	0
4	1017	104	0	0	4	0	24	8	0	0	0	16	4
5	1018	9	0	0	0	0	0	0	0	0	0	0	0
6	1021	36	0	12	0	0	0	0	0	0	0	0	12
7	1052	29	0	0	0	0	0	4	0	0	0	0	0
8	1054	332442	0	40531	0	0	0	0	0	0	19653	0	0
9	1074	6	0	0	0	0	0	0	0	0	0	4	0

Top 3 Customers

As we saw in the previous part, approximately 60% of the total transactions are requested by the one of the top three customers. Hence, it is worth observing their behaviors individually.

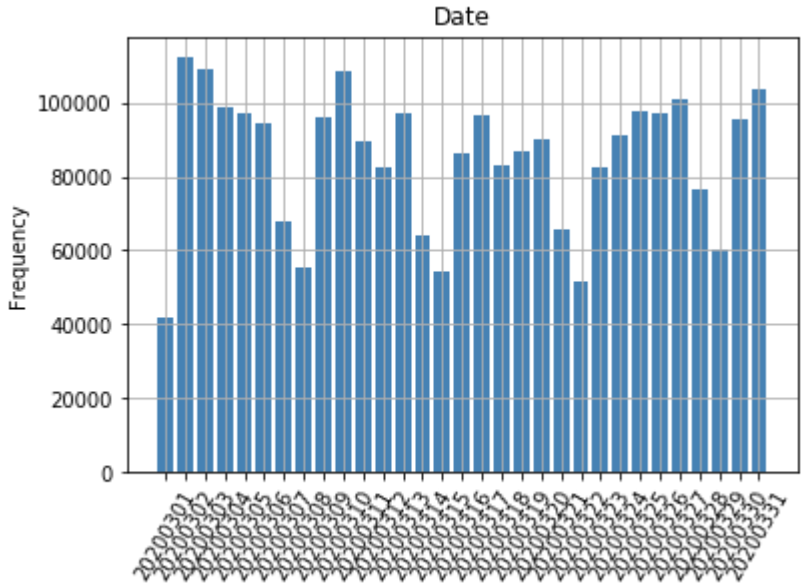
In [13]:
```
top_cust = df.sort_values(by=['TotalRequest'], ascending=False).iloc[:,[0,-2,-1]].head(10)
top_cust
```

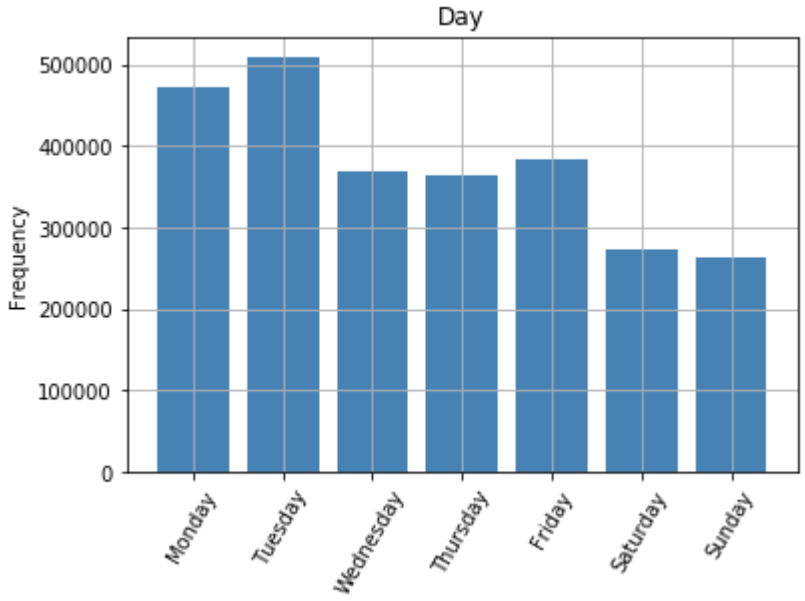
Out[13]:

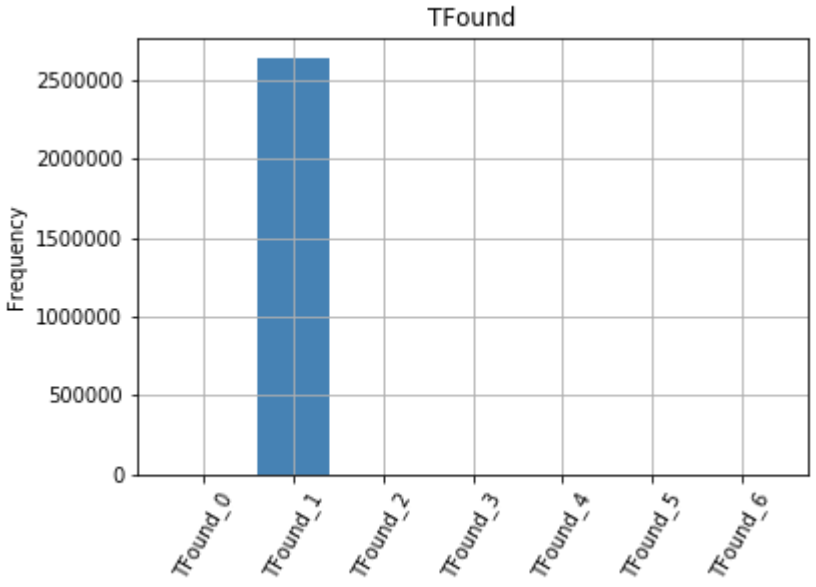
	CustID	TotalRequest	Percentage
59	2204	2634629	29.1 %
50	2071	1639529	18.1 %
39	1920	1347440	14.9 %
45	1980	659262	7.3 %
8	1054	373627	4.1 %
25	1654	288305	3.2 %
86	2669	225406	2.5 %
15	1507	181421	2.0 %
14	1287	132346	1.5 %
23	1634	132342	1.5 %

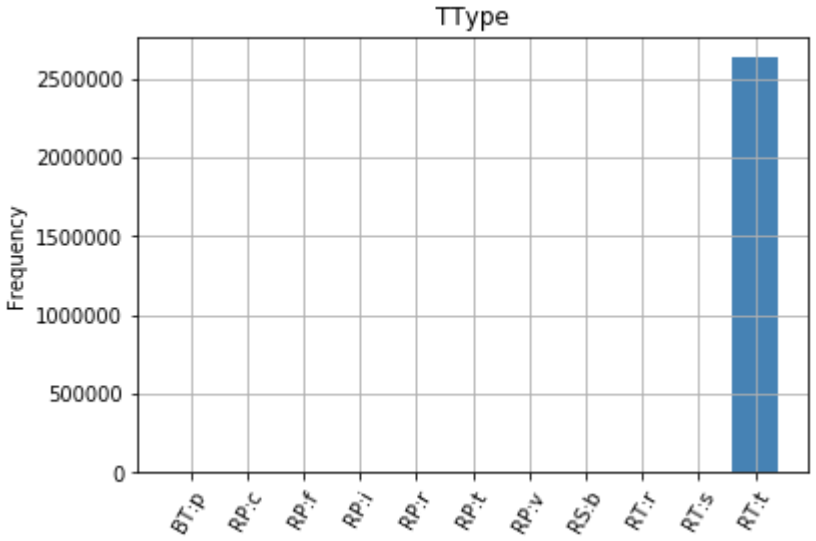
In [14]:
```
categories = {'Date': list(df.columns[2:33]),'Day': list(df.columns[33:40]),
              'TFound': list(df.columns[40:47]),'TType': list(df.columns[47:58]),
             'TDetail': list(df.columns[58:205])}
top_custid = top_cust['CustID'].head(3).tolist()

for i in range(len(top_custid)):
    print('CustID: ' + str(top_custid[i]))
    for key in categories:
        if key != 'TDetail':
            plt.bar(categories[key],
                   df[categories[key]].loc[df.CustID == top_custid[i]].values.tolist()[0],
                    color='steelblue')
            plt.xticks(rotation=60)
            plt.ylabel('Frequency')
            plt.title(key)
            plt.grid(True)
            plt.show()
        else:
            row = df[categories[key]].loc[df.CustID == top_custid[i]]
            row = row.sort_values(by = row.index[0],axis=1, ascending=False, inplace=False, kind='quicksort',
na_position='last')
            plt.bar(row.columns[0:10],row.iloc[0,0:10],color='steelblue')
            plt.xticks(rotation=80)
            plt.title(key)
            plt.ylabel('Frequency')
            plt.grid(True)
            plt.show()
```

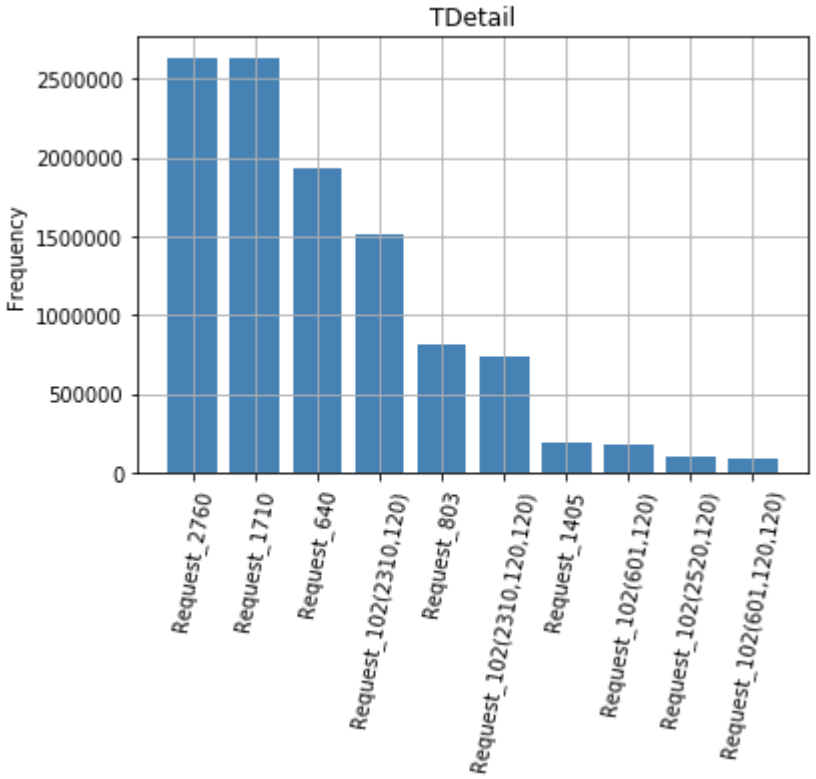
```
CustID: 2204
```


Date
Frequency
0
20000
40000
60000
80000
100000

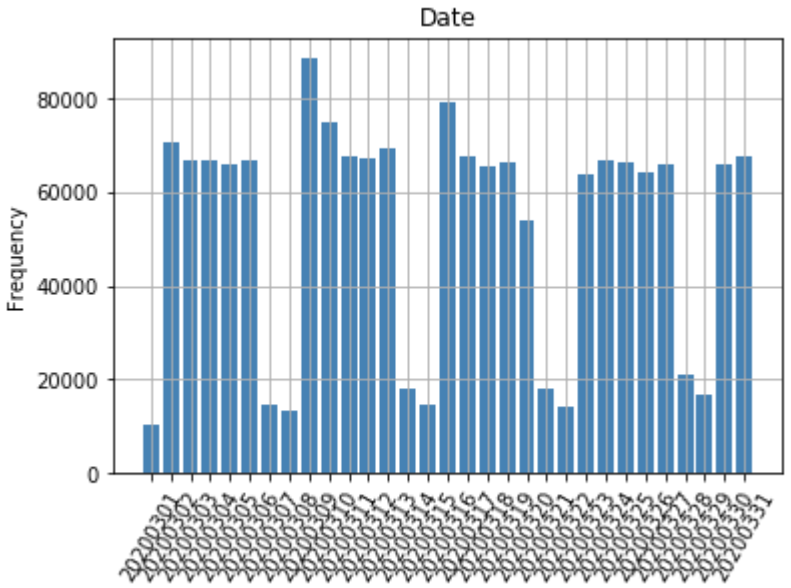

Day
Frequency
0
100000
200000
300000
400000
500000
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday

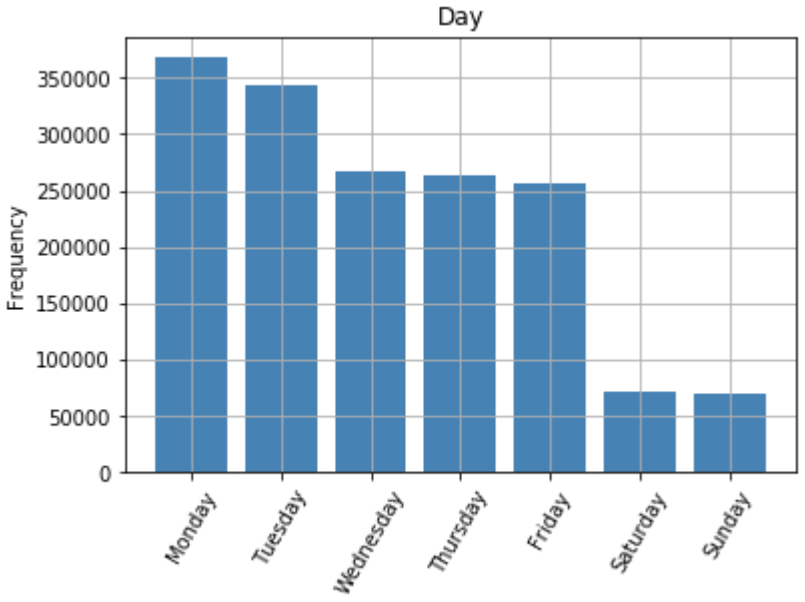

TFound
Frequency
2500000
2000000
1500000
1000000
500000
0
TFound_0
TFound_1
TFound_2
TFound_3
TFound_4
TFound_5
TFound_6

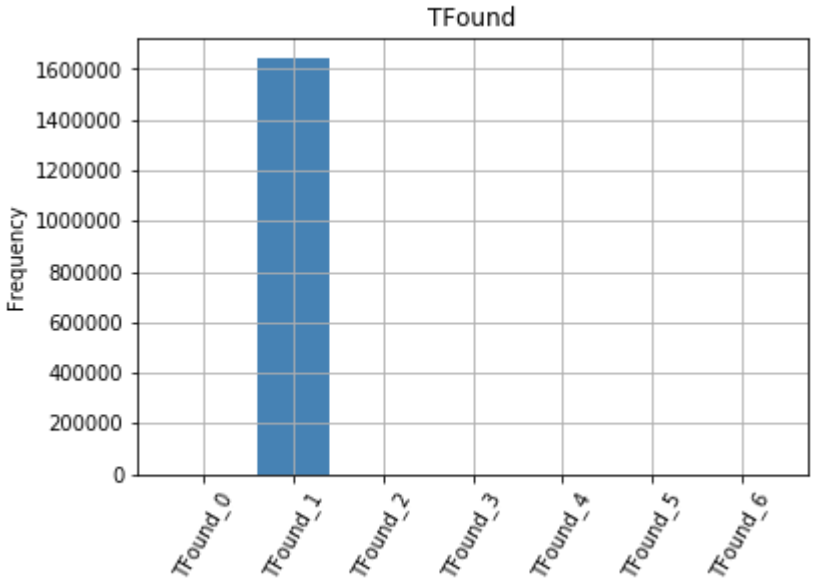

TType
Frequency
2500000
2000000
1500000
1000000
500000
0
BT:p
RP:c
RP:f
RP:i
RP:r
RP:t
RP:v
RS:b
RT:r
RT:s
RT:t

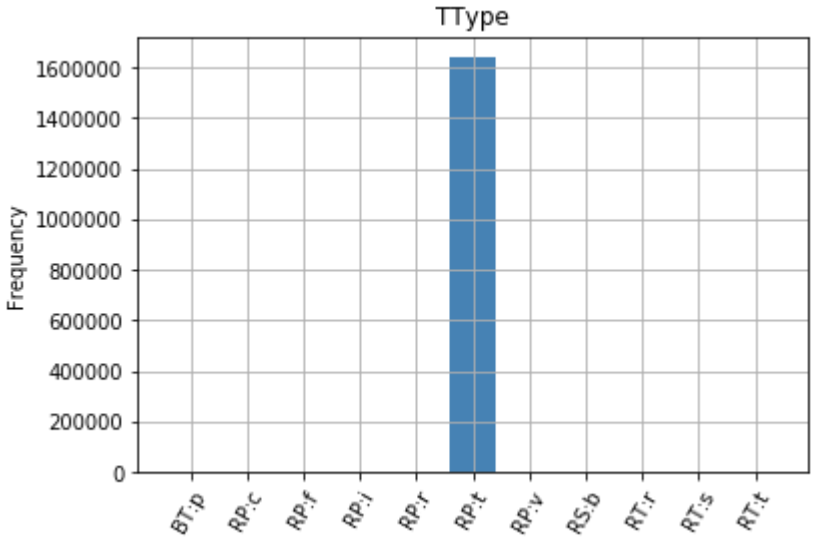

TDetail
Frequency
2500000
2000000
1500000
1000000
500000
0
Request_2760
Request_1710
Request_640
Request_102(2310,120)
Request_803
Request_102(2310,120,120)
Request_1405
Request_102(601,120)
Request_102(2520,120)
Request_102(601,120,120)

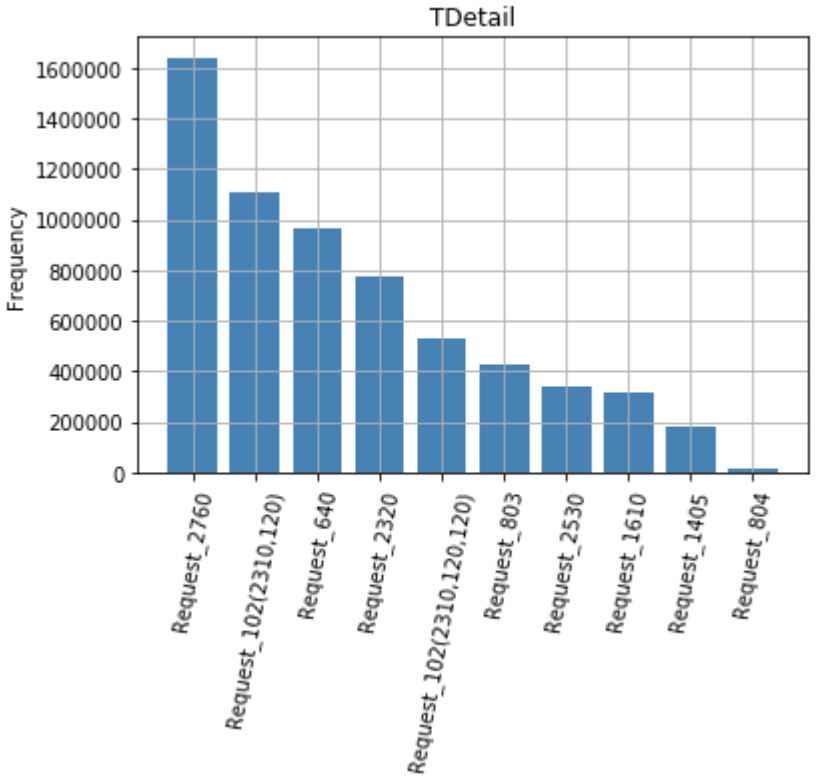
```
CustID: 2071
```


Date
Frequency
0
20000
40000
60000
80000
20200301
20200302
20200303
20200304
20200305
20200306
20200307
20200308
20200309
20200310
20200311
20200312
20200313
20200314
20200315
20200316
20200317
20200318
20200319
20200320
20200321
20200322
20200323
20200324
20200325
20200326
20200327
20200328
20200329
20200330
20200331

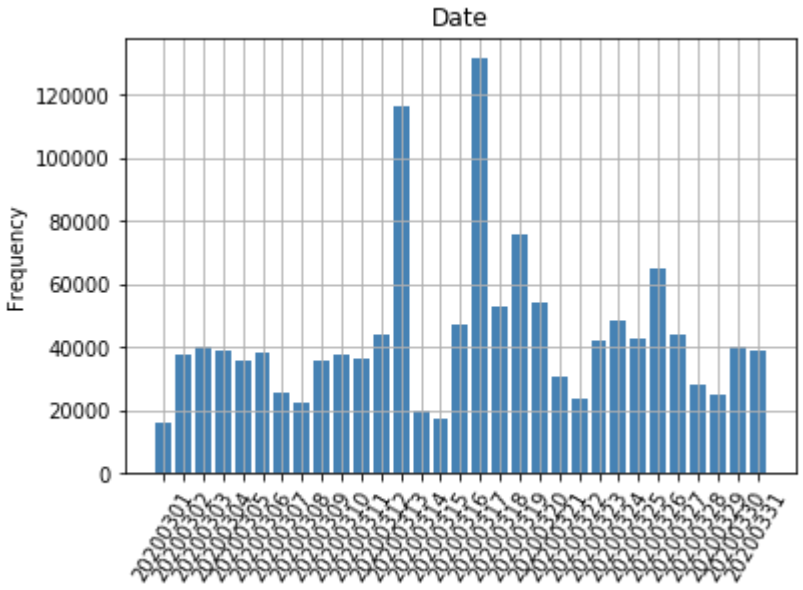

Day
Frequency
0
50000
100000
150000
200000
250000
300000
350000
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday

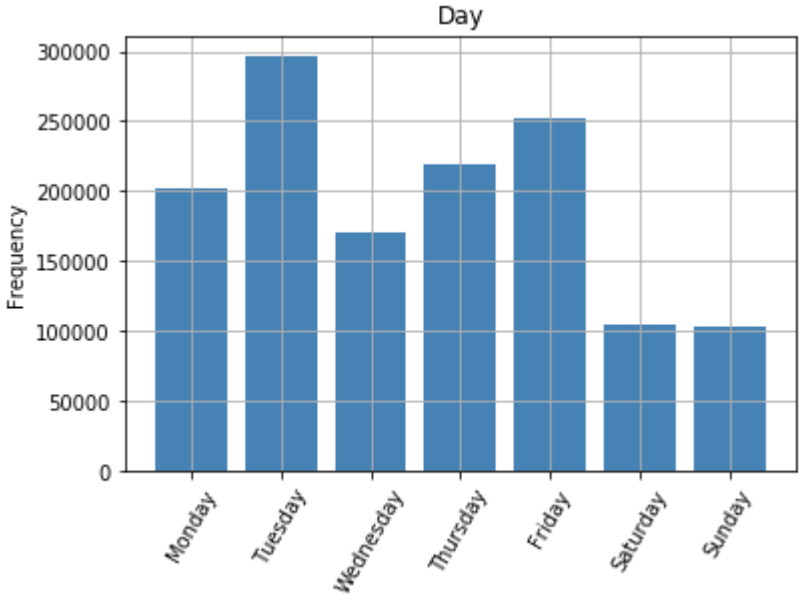

TFound
Frequency
1600000
1400000
1200000
1000000
800000
600000
400000
200000
0
TFound_0
TFound_1
TFound_2
TFound_3
TFound_4
TFound_5
TFound_6

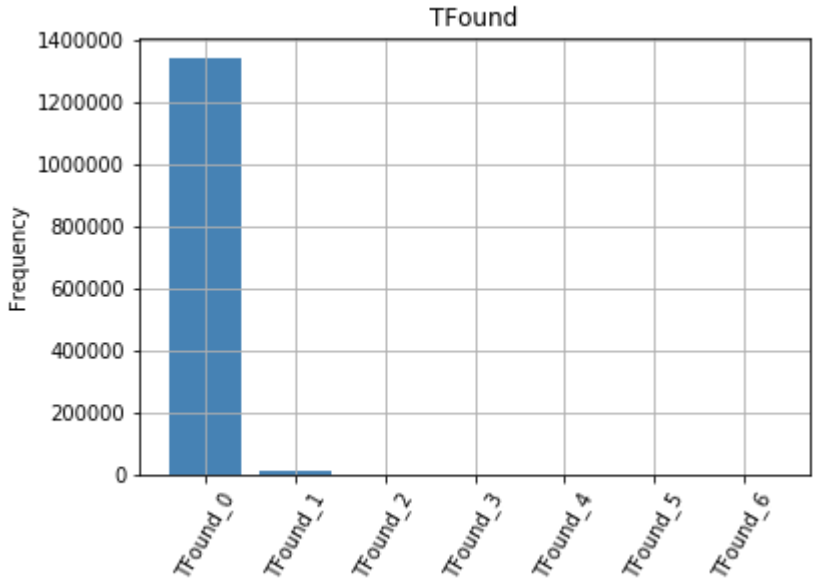

TType
Frequency
1600000
1400000
1200000
1000000
800000
600000
400000
200000
0
BT:p
RP:c
RP:f
RP:i
RP:r
RP:t
RP:v
RS:b
RT:r
RT:s
RT:t

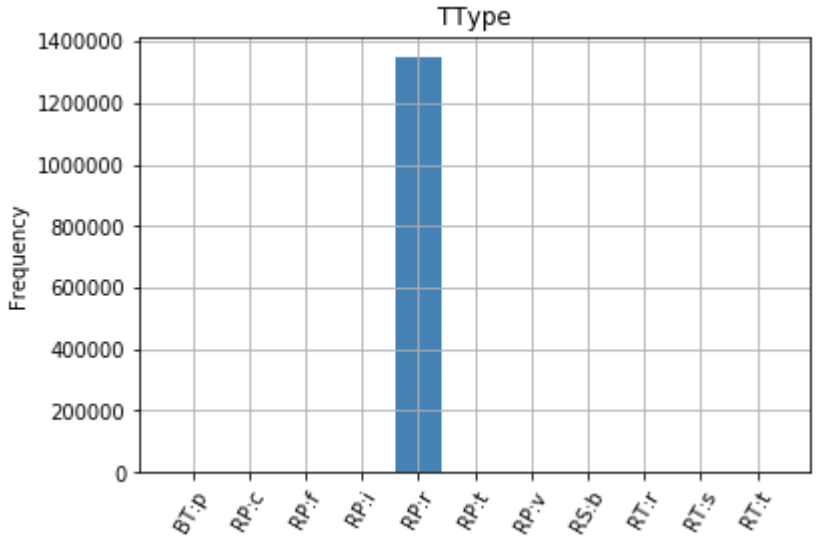

TDetail
Frequency
1600000
1400000
1200000
1000000
800000
600000
400000
200000
0
Request_2760
Request_102(2310,120)
Request_640
Request_2320
Request_102(2310,120,120)
Request_803
Request_2530
Request_1610
Request_1405
Request_804

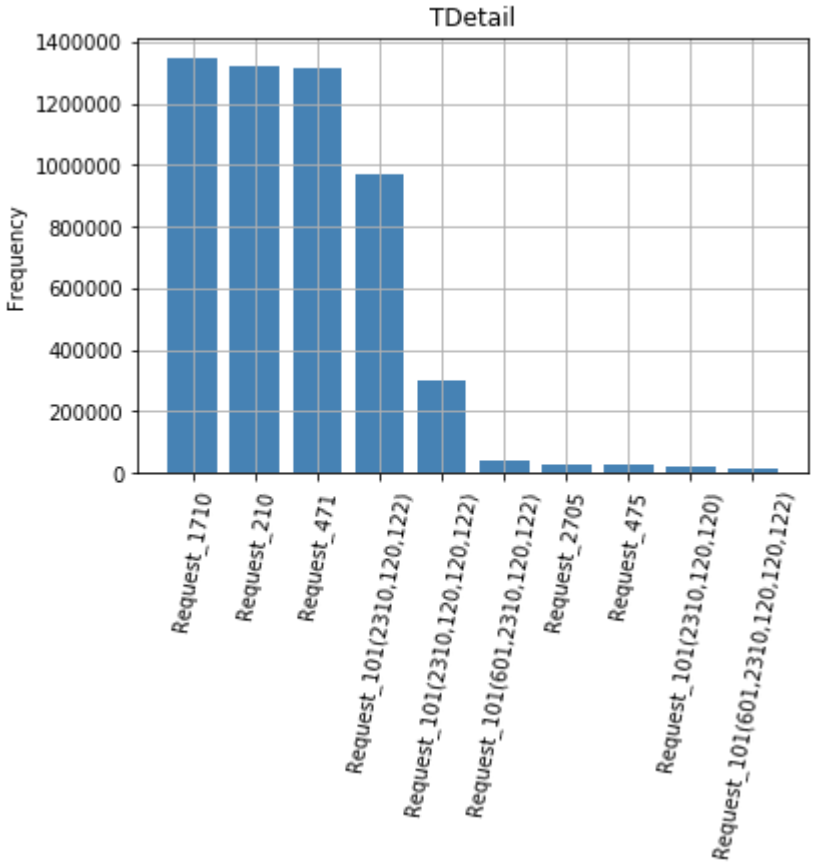
```
CustID: 1920
```


Date
Frequency
120000
100000
80000
60000
40000
20000
0
20200301
20200302
20200303
20200304
20200305
20200306
20200307
20200308
20200309
20200310
20200311
20200312
20200313
20200314
20200315
20200316
20200317
20200318
20200319
20200320
20200321
20200322
20200323
20200324
20200325
20200326
20200327
20200328
20200329
20200330
20200331


Day
Frequency
300000
250000
200000
150000
100000
50000
0
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday


TFound
Frequency
1400000
1200000
1000000
800000
600000
400000
200000
0
TFound_0
TFound_1
TFound_2
TFound_3
TFound_4
TFound_5
TFound_6


TType
Frequency
1400000
1200000
1000000
800000
600000
400000
200000
0
BT:p
RP:c
RP:f
RP:i
RP:r
RP:t
RP:v
RS:b
RT:r
RT:s
RT:t


TDetail
Frequency
1400000
1200000
1000000
800000
600000
400000
200000
0
Request_1710
Request_210
Request_471
Request_101(2310,120,122)
Request_101(2310,120,120,122)
Request_101(601,2310,120,122)
Request_2705
Request_475
Request_101(2310,120,120)
Request_101(601,2310,120,120,122)

Clustering the Customers into Groups Using Machine Learning

Now we will attempt to divide the data set into groups using unsupervised machine learning algorithms. We are going to use the following columns for this clustering analysis:

Date
Day
TFound
TType
TDetail

Since the total request counts vary vastly depending on the customer, we need to scale the data so that each data has an equal impact on the final clustering result. To do this we will take each row value and divide it by the total number of March transactions for that customer. This way we are able to convert the count of each entity into a percentage-based ratio for each customer. Below is the scaled data set:

In [15]:
```
scaled_df = df.iloc[:,2:-2].apply(lambda row: row / (df['TotalRequest'] + 1), axis = 0)
scaled_df.head(10)
```

Out[15]:

	20200301	20200302	20200303	20200304	20200305	20200306	20200307	20200308	20200309	20200310	20200311	20200312	2020
0	0.0	0.000000	0.000859	0.0	0.000021	0.000000	0.00000	0.0	0.000021	0.000021	0.000000	0.000000	0.00
1	0.0	0.000000	0.062500	0.0	0.000000	0.156250	0.28125	0.0	0.031250	0.031250	0.000000	0.125000	0.00
2	0.0	0.060606	0.000000	0.0	0.000000	0.075758	0.00000	0.0	0.000000	0.030303	0.000000	0.000000	0.06
3	0.0	0.212554	0.000000	0.0	0.000000	0.238181	0.00000	0.0	0.000000	0.017850	0.000000	0.000000	0.00
4	0.0	0.000000	0.049383	0.0	0.296296	0.098765	0.00000	0.0	0.000000	0.197531	0.049383	0.049383	0.00
5	0.0	0.000000	0.000000	0.0	0.000000	0.000000	0.00000	0.0	0.000000	0.000000	0.000000	0.000000	0.00
6	0.0	0.480000	0.000000	0.0	0.000000	0.000000	0.00000	0.0	0.000000	0.000000	0.480000	0.000000	0.00
7	0.0	0.000000	0.000000	0.0	0.000000	0.190476	0.00000	0.0	0.000000	0.000000	0.000000	0.190476	0.00
8	0.0	0.108480	0.000000	0.0	0.000000	0.000000	0.00000	0.0	0.052600	0.000000	0.000000	0.000000	0.00
9	0.0	0.000000	0.000000	0.0	0.000000	0.000000	0.00000	0.0	0.000000	0.800000	0.000000	0.000000	0.00

In [17]:
```
# Make nparray X from df
X = scaled_df.values
```

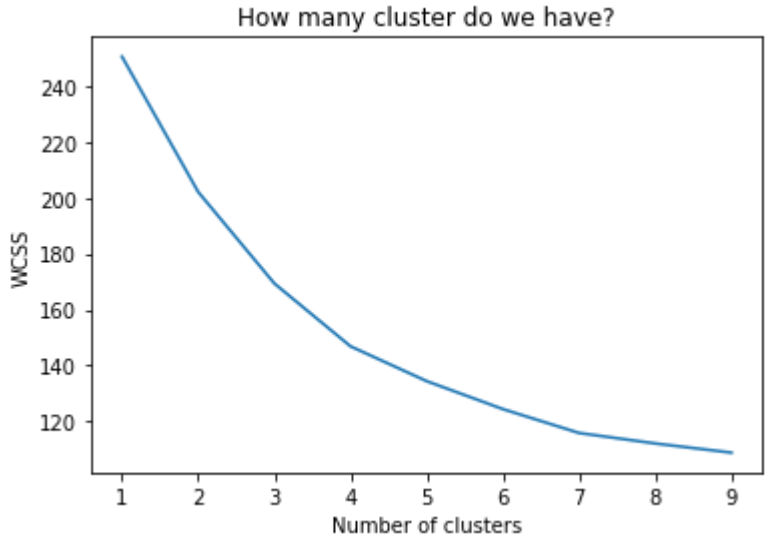
K Means

We may now apply unsupervised machine learning to our data set. We will start with the K-mean algorithm. Before we begin, I will briefly review the technical side of how this argorithm works:

Model Methodology
To use the K Mean algorithm, we begin by specifying the number of clusters. Let's say we want to divide the data into K number of clusters. Next, we randomly place the K points in the data space. We call these K points centroids. Then we assign each data point to the closest centroid by measuring the distance between the point and the centroids (the distance can be Euclidean, Manhattan, etc). Now all our data points should belong to one the K centroids. Next, we recalculate the centroids of each cluster. To do so, we take the average of all the points in the cluster and assign this midpoint to be the new centroid. After that, we assign each data point to the closest new cluster and recalculate the centroids again. This process repeats itself until the centroids converge to their most reasonable points.
Note:
-We need to specify the number of clusters beforehand
-The random placement of initial centroids can affect the final result, which is unfortunate

Since we need to specify the initial number of centroids, we should try several different numbers of centroids to train the model. In order to evaluate how good the model fits, we use Within-Cluster-Sum-of-Squares (WCSS). WCSS is the sum of the distance between each data point and its centroids. The lower the WCSS score is, the better each observation is clustered.

In [18]:
```
# Using the elbow method to find the optimal number of clusters
from sklearn.cluster import KMeans
wcss = []
for i in range(1, 10):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 42)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 10), wcss)
plt.title('How many cluster do we have?')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```



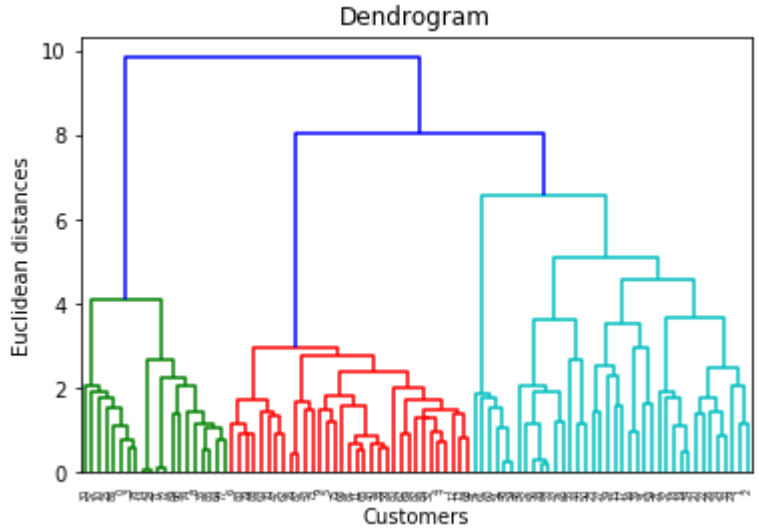
Unfortunately, the graph above does not suggest any clear-cut conclusion about how many groups we should divide the data set into. Moving forward, we will arbitrarily set the number of clusters to be 3, instantiate a k mean object and fit it to the data set.

In [19]:
```
# Fitting K-Means to the dataset
kmeans = KMeans(n_clusters = 3, init = 'k-means++', random_state = 42)
y_kmeans = kmeans.fit_predict(X)
```

Hierarchical Clustering

Next, we will apply Hierarchical Clustering to our dataset.
Model Methodology
Unlike the K Means algorithm, where we start by deciding the number of clusters to search for, Hierarchical Clustering starts from each data point and goes up. That is, for a dataset of N data points, we start with the trivial conclusion that the dataset forms N clusters. Next, we take the two closest clusters (still only two data points) and make them form a cluster, which revises our conclusion a little (we now have N – 1 clusters). Now, out of these N – 1 clusters, we merge the two closest clusters into one cluster. We now have N - 2 clusters. This process continues until there is only one cluster left. Repeating these steps naturally corresponds to forming a tree diagram from the bottom up. This tree diagram shows the history of our merging process. Using this tree diagram, we can observe the possible groupings of our data set by cutting the tree horizontally at an arbitrary height.
Let us start the hierarching process and take a look at the tree diagram:

In [20]:
```
# Using the dendrogram to find the optimal number of clusters
import scipy.cluster.hierarchy as sch
dendrogram = sch.dendrogram(sch.linkage(X, method = 'ward'))
plt.title('Dendrogram')
plt.xlabel('Customers')
plt.ylabel('Euclidean distances')
plt.show()
```



In this diagram, the distance between two neighboring clusters is represented by the length of the vertical lines that connect the nodes of each cluster. ie, the longer the vertical line connecting to the node, the further away it is from the other clusters. Therefore, based on the above we can naturally conclude that we should divide the dataset into three groups- green, red, and blue. Next, let's instantiate a hierarchical clustering object and fit it to the data set:

In [21]:
```
# Fitting Hierarchical Clustering to the dataset
from sklearn.cluster import AgglomerativeClustering
hc = AgglomerativeClustering(n_clusters = 3, affinity = 'euclidean', linkage = 'ward')
y_hc = hc.fit_predict(X)
```

Compare the Result

Having fitted the two models to the data set, we can now append the cluster labeling onto the original data set and see how the two models predict the categories:

In [22]:
```
# Concatenate the prediction onto df
df['KM_pred'] = y_kmeans
df['HC_pred'] = y_hc
df[['CustID','KM_pred','HC_pred' ]]
```

Out[22]:

	CustID	KM_pred	HC_pred
0	1	1	1
1	198	2	0
2	260	2	0
3	1010	1	1
4	1017	0	2
5	1018	0	2
6	1021	0	2
7	1052	0	2
8	1054	1	1
9	1074	0	2
10	1095	1	1
11	1173	0	2
12	1205	0	2
13	1285	0	2
14	1287	1	1
15	1507	2	0
16	1563	2	0
17	1569	2	0
18	1585	2	0
19	1587	2	0
20	1596	2	0
21	1603	2	0
22	1628	2	0
23	1634	0	0
24	1653	2	0
25	1654	1	1

	CustID	KM_pred	HC_pred
26	1655	2	0
27	1661	2	0
28	1675	1	1
29	1692	2	0
30	1717	1	1
31	1734	2	0
32	1749	0	2
33	1751	2	0
34	1752	2	0
35	1768	1	1
36	1793	2	0
37	1794	2	0
38	1885	0	2
39	1920	2	0
40	1923	0	2
41	1927	2	0
42	1932	2	0
43	1966	0	2
44	1967	2	0
45	1980	2	0
46	2024	2	0
47	2046	0	2
48	2056	2	0
49	2070	2	0
50	2071	2	0
51	2080	2	0
52	2117	2	0

	CustID	KM_pred	HC_pred
53	2159	2	0
54	2169	2	0
55	2180	2	0
56	2184	2	0
57	2187	1	1
58	2197	0	2
59	2204	2	0
60	2216	2	0
61	2223	2	0
62	2245	0	2
63	2256	0	2
64	2276	1	1
65	2282	0	2
66	2310	0	2
67	2343	0	2
68	2362	0	2
69	2382	0	2
70	2405	0	2
71	2413	1	1
72	2424	1	1
73	2481	0	2
74	2493	1	1
75	2498	0	2
76	2507	0	2
77	2509	1	1
78	2520	1	1
79	2537	0	2

	CustID	KM_pred	HC_pred
80	2574	0	2
81	2588	0	2
82	2593	0	2
83	2623	0	2
84	2633	0	2
85	2662	1	1
86	2669	1	1
87	2680	0	2
88	2684	0	2
89	2704	1	1
90	2707	1	1

The above shows that we have exactly the same labeling from both algorithms (ie, the K-means labeling of 0, 1 and 2 correspond to the Hierarchical Clustering labeling of 2, 1 and 0, respectively).

Behavioral Characteristics of Each Group

Next, we will divide the customers into three different groups according to the results from our K-means model. Then we examine each group to find its behavioral tendencies. We group each cluster together and take the mean value of each cell. Since the top 3 customers have a tremendous impact on the mean value of entities, we will exclude those three customers from our analysis moving forward. I should note that although the rest of the customers have relatively similar impacts on the mean values, each customer's individual behavior varries greatly compared to the other customers, which makes it somewhat harder for the three groups to represent their members' individual characteristics.
The data after grouping is shown below:

In [23]:
```
# Remove top three customers from the data set
for custid in top_custid:
    df = df.loc[df.CustID != custid]

# Group the data by the K Mean label
grouped_km =  df.groupby('KM_pred')
groups = [grouped_km.mean().iloc[i,1:-1] for i in range(len(grouped_km))]
df_group = pd.DataFrame(groups)
df_group
```

Out[23]:

	TCount	20200301	20200302	20200303	20200304	20200305	20200306	20200307	20200308	20200309
0	57.470588	85.058824	129.235294	128.764706	130.323529	129.411765	129.323529	127.323529	127.029412	129.676471
1	71700.600000	165.200000	7329.300000	6039.450000	2543.750000	367.550000	2882.600000	243.100000	245.250000	9760.950000
2	33958.558824	700.000000	3104.676471	2231.441176	2426.529412	1115.882353	963.735294	1112.088235	590.588235	980.735294

Finally, we examine the characteristics of each group by observing the histograms and bar plots of each category (i.e. Date, Day, TFound, TType and TDetail). Again, it is very important to note that we should be cautious in describing the tendency of each group; although we took the mean of each value, we should keep in mind that each customer has its own very skewed behavior.

Group A

Here are the customers labeled as 0. We'll call them Group A:

In [24]:
```
df.loc[df.KM_pred == 0][['CustID', 'KM_pred']]
```

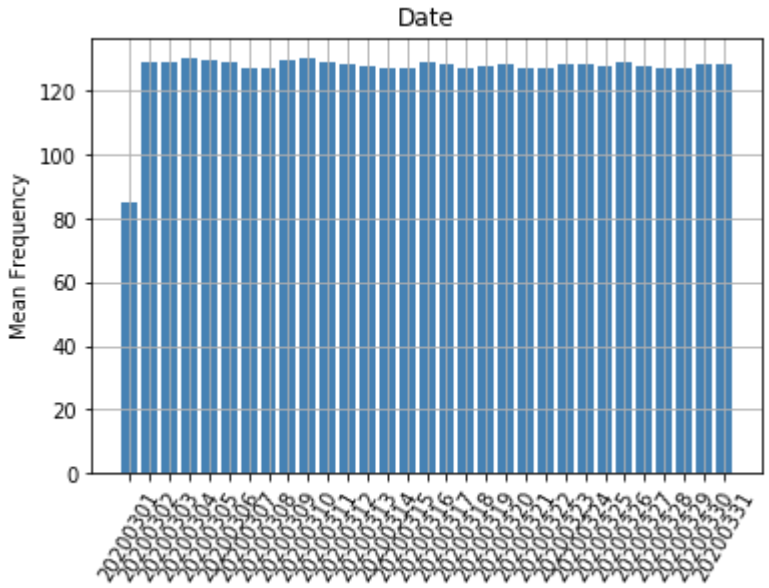
Out[24]:

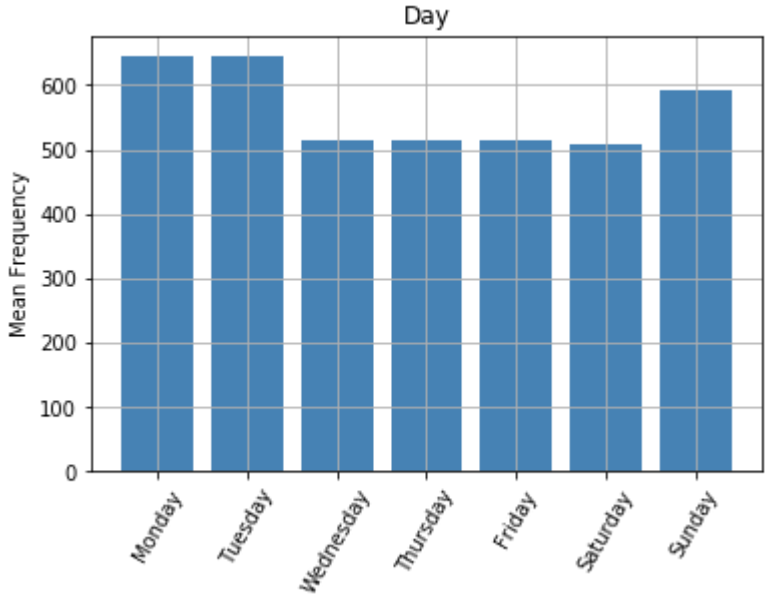
	CustID	KM_pred
4	1017	0
5	1018	0
6	1021	0
7	1052	0
9	1074	0
11	1173	0
12	1205	0
13	1285	0
23	1634	0
32	1749	0
38	1885	0
40	1923	0
43	1966	0
47	2046	0
58	2197	0
62	2245	0
63	2256	0
65	2282	0
66	2310	0
67	2343	0
68	2362	0
69	2382	0
70	2405	0
73	2481	0
75	2498	0
76	2507	0

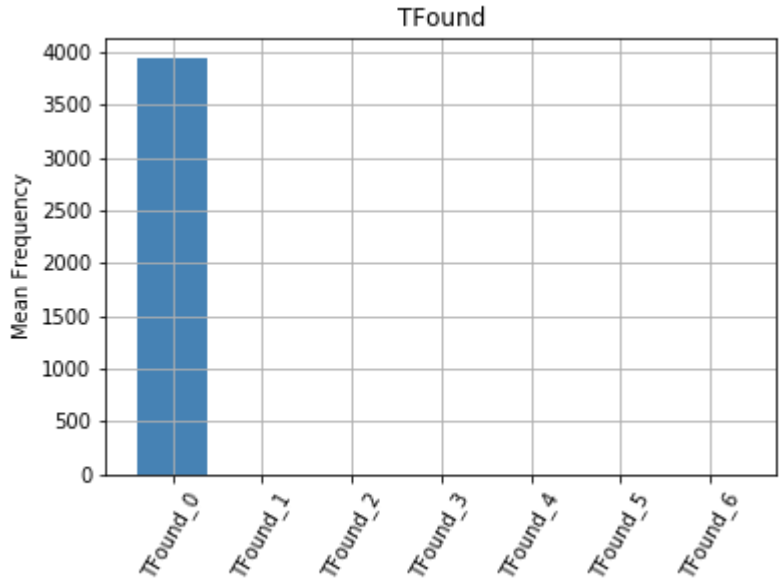
	CustID	KM_pred
79	2537	0
80	2574	0
81	2588	0
82	2593	0
83	2623	0
84	2633	0
87	2680	0
88	2684	0

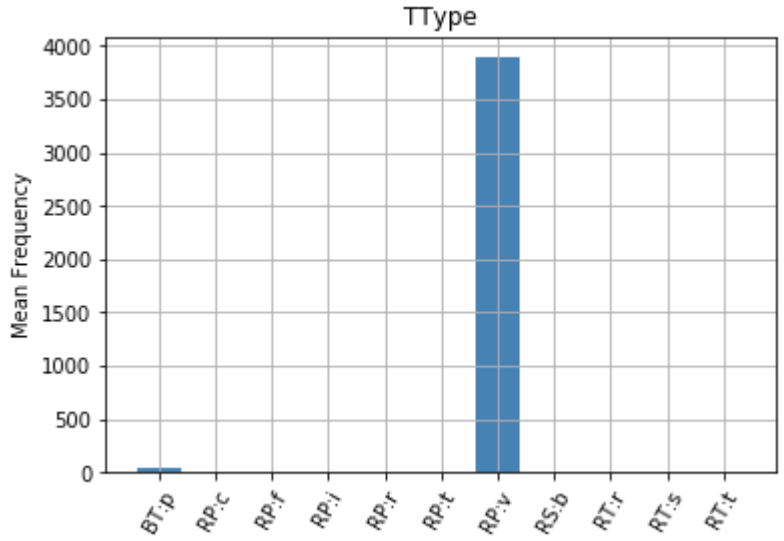
Let's look at the behavior of each category for Group A:

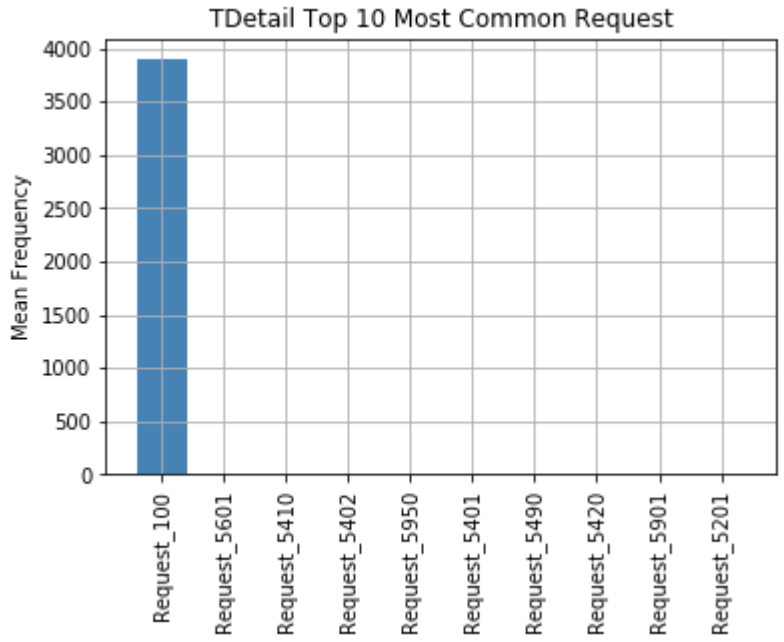
In [25]:
```
# Define categories to look at
categories = {'Date': list(df_group.columns[1:32]), 'Day': list(df_group.columns[32:39]),
              'TFound':list(df_group.columns[39:46]), 'TType': list(df_group.columns[46:57]),
             'TDetail' : list(df_group.columns[57:-2]) }

i = 0
# For each category, plot bar graph for each group
for name in categories:
    if name != 'TDetail':
        plt.bar(categories[name],
                df_group.loc[i,categories[name]].values,
                color='steelblue')
        plt.xticks(rotation=60)
        plt.ylabel('Mean Frequency')
        plt.title(name)
        plt.grid(True)
        plt.show()
    else:
        row = df_group.loc[i,categories[name]]
        row = row.sort_values(ascending=False)
        plt.bar(row.index[0:10],row.values[0:10],color='steelblue')
        plt.title(name + str(' Top 10 Most Common Request'))
        plt.xticks(rotation=90)
        plt.ylabel('Mean Frequency')
        plt.grid(True)
        plt.show()
```


Date
Mean Frequency
0
20
40
60
80
100
120
20200301
20200302
20200303
20200304
20200305
20200306
20200307
20200308
20200309
20200310
20200311
20200312
20200313
20200314
20200315
20200316
20200317
20200318
20200319
20200320
20200321
20200322
20200323
20200324
20200325
20200326
20200327
20200328
20200329
20200330
20200331


Day
Mean Frequency
0
100
200
300
400
500
600
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday


TFound
Mean Frequency
4000
3500
3000
2500
2000
1500
1000
500
0
TFound_0
TFound_1
TFound_2
TFound_3
TFound_4
TFound_5
TFound_6


TType
Mean Frequency
4000
3500
3000
2500
2000
1500
1000
500
0
BT:p
RP:c
RP:f
RP:i
RP:r
RP:t
RP:v
RS:b
RT:r
RT:s
RT:t



Group A shows the following patterns:

- The number of transaction requests is consistently low throughout the month (or week) compared to the other two groups.
- All the transactions are unknown (TFound = 0)
- Most of the transaction types are RP:V
- All the transaction requests are Request_100

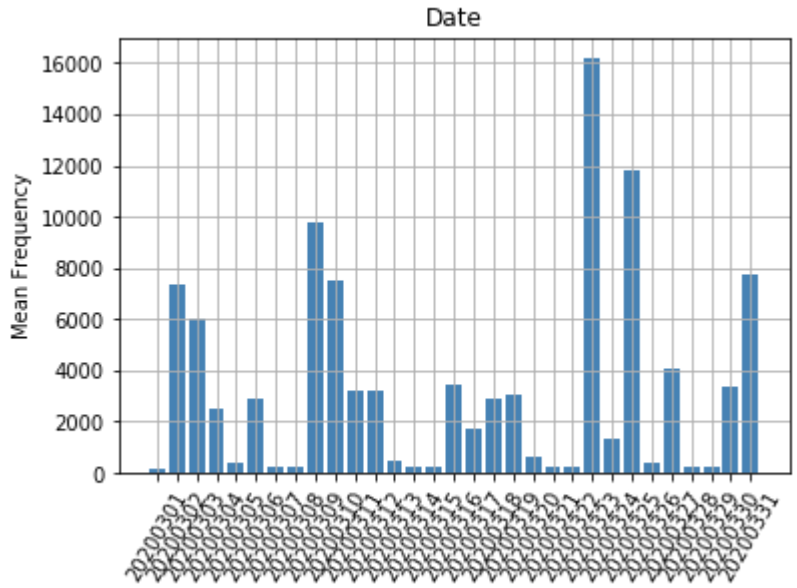
Group B

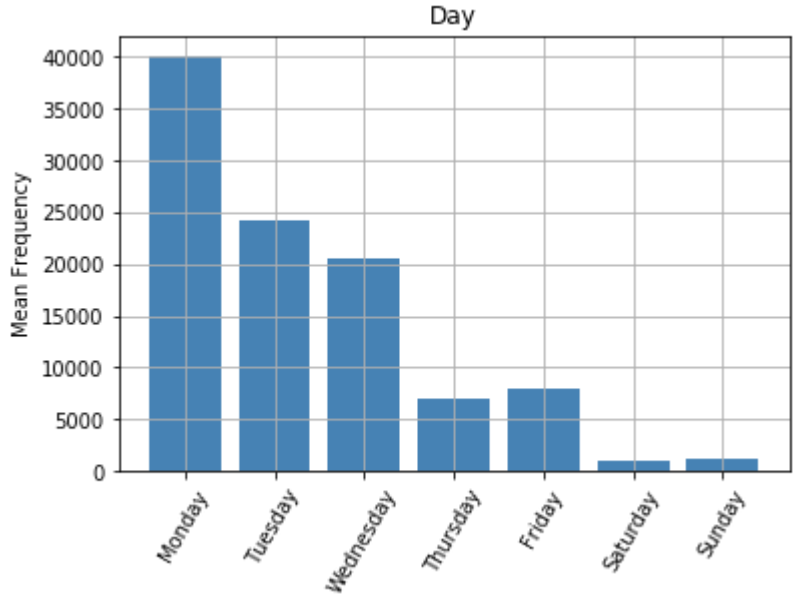
In [26]:
```
df.loc[df.KM_pred == 1][['CustID', 'KM_pred']]
```

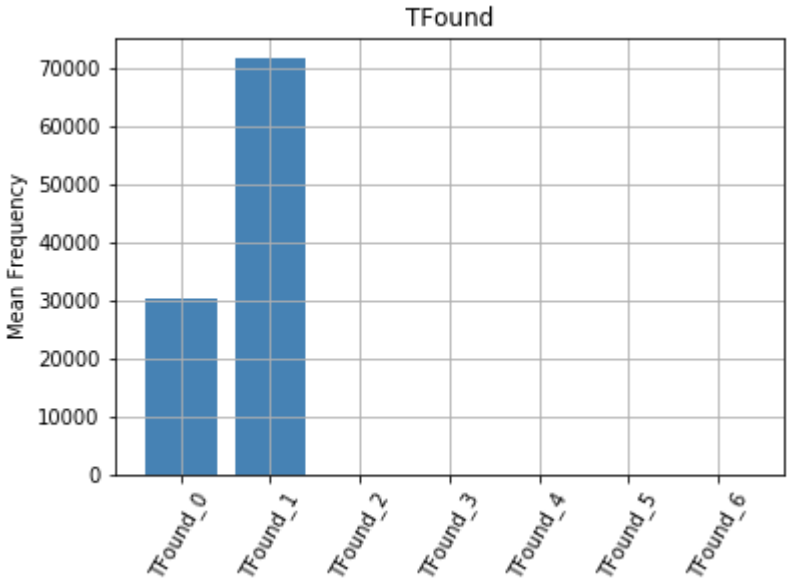
Out[26]:

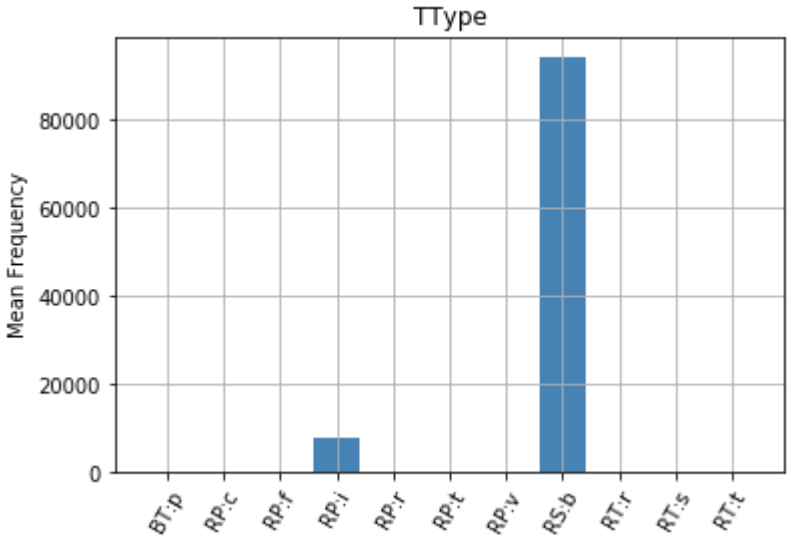
	CustID	KM_pred
0	1	1
3	1010	1
8	1054	1
10	1095	1
14	1287	1
25	1654	1
28	1675	1
30	1717	1
35	1768	1
57	2187	1
64	2276	1
71	2413	1
72	2424	1
74	2493	1
77	2509	1
78	2520	1
85	2662	1
86	2669	1
89	2704	1
90	2707	1

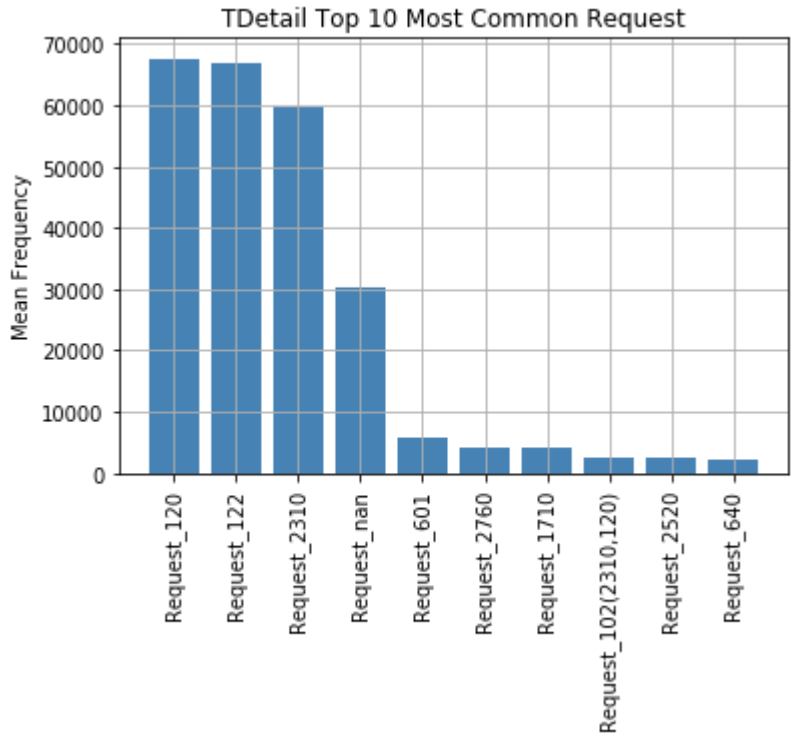
In [27]:
```
i = 1
# For each category, plot bar graph for each group
for name in categories:
    if name != 'TDetail':
        plt.bar(categories[name],
                df_group.loc[i,categories[name]].values,
                color='steelblue')
        plt.xticks(rotation=60)
        plt.ylabel('Mean Frequency')
        plt.title(name)
        plt.grid(True)
        plt.show()
    else:
        row = df_group.loc[i,categories[name]]
        row = row.sort_values(ascending=False)
        plt.bar(row.index[0:10],row.values[0:10],color='steelblue')
        plt.title(name + str(' Top 10 Most Common Request'))
        plt.xticks(rotation=90)
        plt.ylabel('Mean Frequency')
        plt.grid(True)
        plt.show()
```


Date
Mean Frequency
16000
14000
12000
10000
8000
6000
4000
2000
0
20200301
20200302
20200303
20200304
20200305
20200306
20200307
20200308
20200309
20200310
20200311
20200312
20200313
20200314
20200315
20200316
20200317
20200318
20200319
20200320
20200321
20200322
20200323
20200324
20200325
20200326
20200327
20200328
20200329
20200330
20200331


Day
Mean Frequency
40000
35000
30000
25000
20000
15000
10000
5000
0
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday


TFound
Mean Frequency
70000
60000
50000
40000
30000
20000
10000
0
TFound_0
TFound_1
TFound_2
TFound_3
TFound_4
TFound_5
TFound_6


TType
Mean Frequency
80000
60000
40000
20000
0
BT:p
RP:c
RP:f
RP:i
RP:r
RP:t
RP:v
RS:b
RT:r
RT:s
RT:t



Group B has the following patterns:

- The number of transactions are sugnificantly more than that of Group A.
- The number of transaction requests changes throughout the month. It generally increases towards the end of the month with multiple spikes around the end of the month.
- It has the highest number of requests on Monday and the number decreases as the week continues. There are very few transactions over the weekend.
- Approximately 70% of the transactions are found (TFound = 1), the rest are unknown.
- The majority have the type "RS:b".
- Most of the requests are either 120, 122 or 2310.

Group C

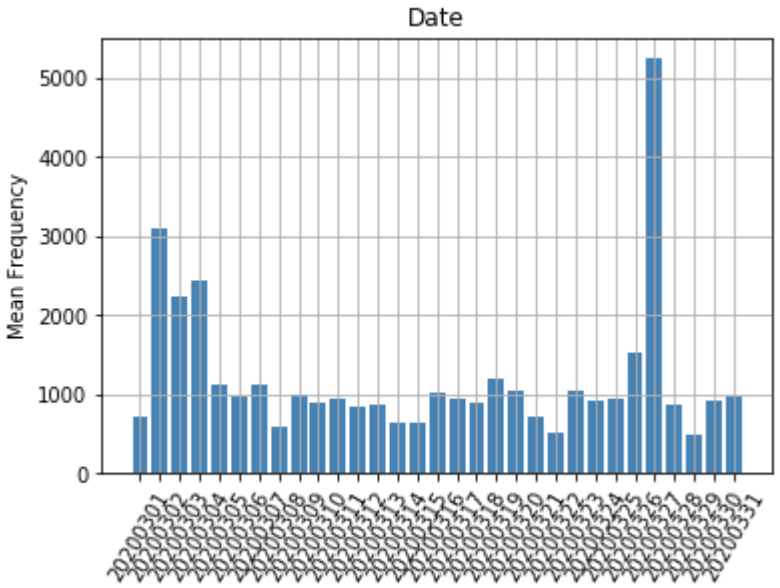
In [28]:
```
df.loc[df.KM_pred == 2][['CustID', 'KM_pred']]
```

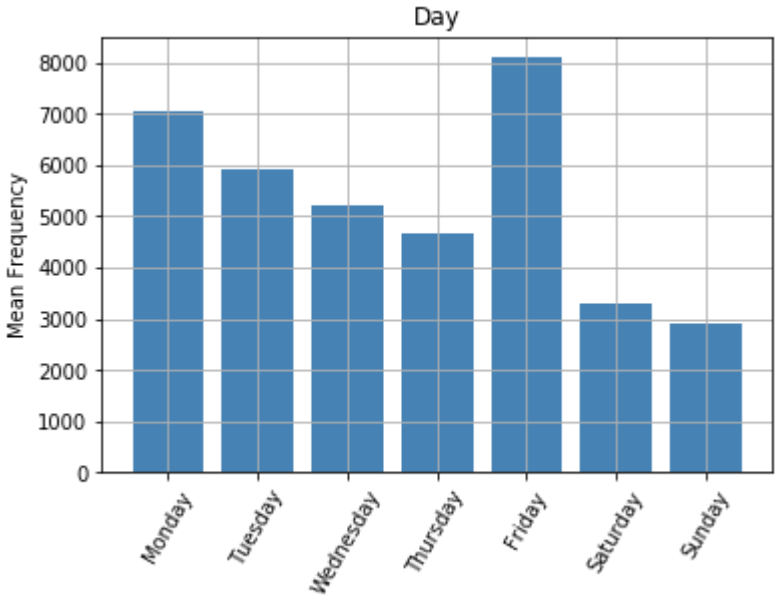
Out[28]:

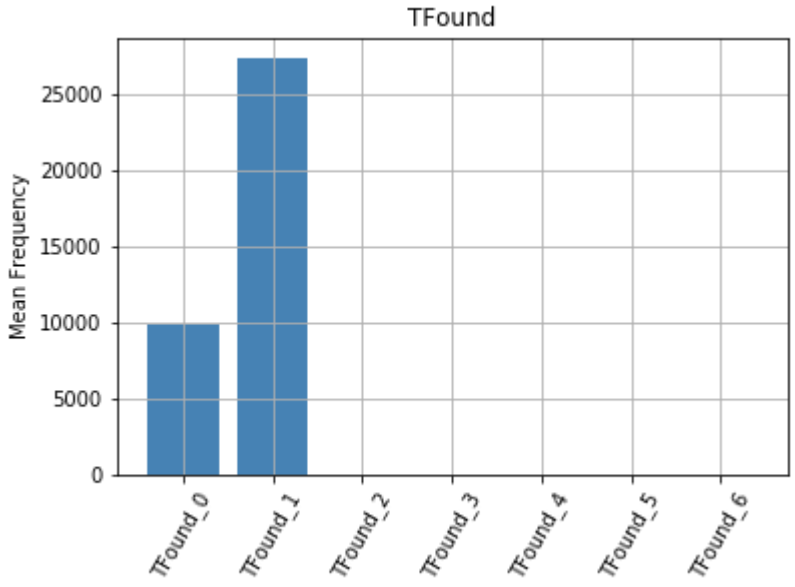
	CustID	KM_pred
1	198	2
2	260	2
15	1507	2
16	1563	2
17	1569	2
18	1585	2
19	1587	2
20	1596	2
21	1603	2
22	1628	2
24	1653	2
26	1655	2
27	1661	2
29	1692	2
31	1734	2
33	1751	2
34	1752	2
36	1793	2
37	1794	2
41	1927	2
42	1932	2
44	1967	2
45	1980	2
46	2024	2
48	2056	2
49	2070	2

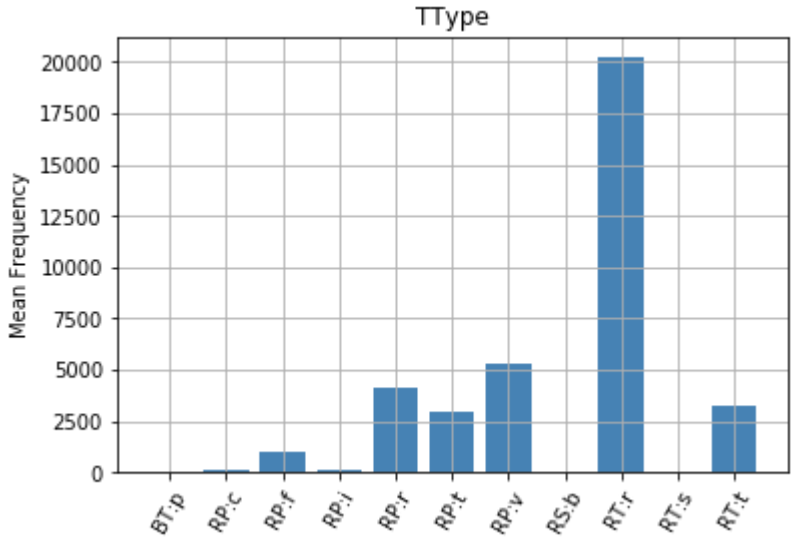
	CustID	KM_pred
51	2080	2
52	2117	2
53	2159	2
54	2169	2
55	2180	2
56	2184	2
60	2216	2
61	2223	2

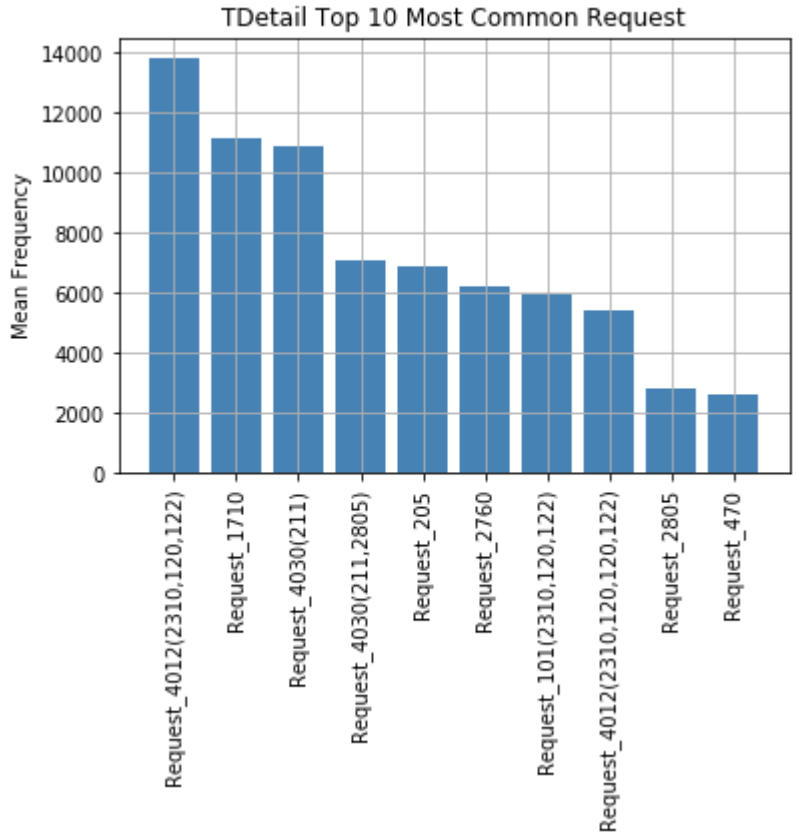
In [29]:
```
i = 2
# For each category, plot bar graph for each group
for name in categories:
    if name != 'TDetail':
        plt.bar(categories[name],
                df_group.loc[i,categories[name]].values,
                color='steelblue')
        plt.xticks(rotation=60)
        plt.ylabel('Mean Frequency')
        plt.title(name)
        plt.grid(True)
        plt.show()
    else:
        row = df_group.loc[i,categories[name]]
        row = row.sort_values(ascending=False)
        plt.bar(row.index[0:10],row.values[0:10],color='steelblue')
        plt.title(name + str(' Top 10 Most Common Request'))
        plt.xticks(rotation=90)
        plt.ylabel('Mean Frequency')
        plt.grid(True)
        plt.show()
```


Date
Mean Frequency
0
1000
2000
3000
4000
5000
20200301
20200302
20200303
20200304
20200305
20200306
20200307
20200308
20200309
20200310
20200311
20200312
20200313
20200314
20200315
20200316
20200317
20200318
20200319
20200320
20200321
20200322
20200323
20200324
20200325
20200326
20200327
20200328
20200329
20200330
20200331


Day
Mean Frequency
0
1000
2000
3000
4000
5000
6000
7000
8000
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday


TFound
Mean Frequency
25000
20000
15000
10000
5000
0
TFound_0
TFound_1
TFound_2
TFound_3
TFound_4
TFound_5
TFound_6


TType
Mean Frequency
20000
17500
15000
12500
10000
7500
5000
2500
0
BT:p
RP:c
RP:f
RP:i
RP:r
RP:t
RP:v
RS:b
RT:r
RT:s
RT:t



Group C shows the following patterns:

- The number of daily transactions is more consistent throughout the month than the other two groups, with a spike on March 26.
- The number of transactions goes down as you get closer to the weekend, with the exception of Friday.
- Approximately 75% of the transactions are found (TFound = 1), while the rest are unknown.
- The majority of the transactions are of type RT:r.

Conclusion

We've analyzed the vendor log for March 2020 from both overall and customer-wise perspectives. The basic statistics on the overall data set showed that there are some periodic patterns in the transaction requests and that the majority of the transactions are requested by a few major customers. Next, we grouped the data by CustID and studied the behavior of the top three customers. Finally, we conducted two unsupervised machine learning algorithms to attempt to cluster the customers into groups. The machine learning models indicataed that there are three groups, which we later found had distinctive behavioral characteristics. Although we succeeded in clustering the customers into groups, it should be noted that each customer has unique behavioral tendencies and hence the traits of each cluster do not always represent all the members of the group accurately.