# D209 Task 1

This is the code for my d209 performance assessment task 1. Student id: 012047746

## A1:PROPOSAL OF QUESTION

The research question that I want to ask for this task is "Is it possible to create a KNN model that can predict whether or not a customer will churn using quantiative and binary qualitative factors?". I did a similar analysis in my D208 assessment task 2. This is a useful analysis to explore because it helps us understand what customer factors may possibly affect what causes a customer to be a customer that churns and a customer that doesn't. It can also help with planning for the future, to see if there will be more or less customers that are churning.

## A2:DEFINED GOAL

The goal is to determine if its possible to create a KNN model that can accurately determine whether or not a customer is going to churn given the input of quantitative and binary qualitative factors. In order to do this, I am going to use scikitlearn and the K nearest neighbors method. From the accuracy of the model, we can determine whether or not it will be useful to use this forecast to try and implement into business practices.

# B1:EXPLANATION OF CLASSIFICATION METHOD

For the analysis I am going to be using K nearest neighbors to classify contributing factors as to whether or not a customer is going to churn. K nearest neighbors plots the data points on a graph and tries to categorize the data points by relative distance to one another on the chart.

The user sets the parameter for k number of points which is closest to the test data area, and the largest amount that is nearby is the predicted class. It is like a voting system, where the number of nearby points in a category counts as a vote, and the cateogry with the most vote is what the point being assessed is categorized as.

# B2:SUMMARY OF METHOD ASSUMPTION

One assumption of the k-nearest-neighbors classification method is that similar things are near one another. If this is not iherintly true for the data, then the method does not work. For instance, is a data point is a certain class but it is a distance away from most of the other points in its class, it will not be categorized as the class it should be classified as.

# B3:PACKAGES OR LIBRARIES LIST

I have used the following packages for my analysis:

- Pandas: This library is essential to import the CSV and apply analysis to the data. This is also used to map the qualitative variables to a numeric value depending on whether its 'yes' or 'no'.
- numpy: We use numpy to use arrays and set up the dataframe to be used for statistical analysis
- matplotlib: We use matplotlib for visualization such as histograms
- scikitlearn: This is the library that is used to bring in the k nearest neighbors classification model. I also use the min max scaler to normalize the numeric variables.
- scipy.stats: Used to find z-scores. Z-scores are used to standardize across the board for numeric variables in order to check if they have outliers, as if the z-score is to high or low than the value is too distant from the rest of the values

In [1]:
```
# import the libraries
import pandas as pd
from pandas import DataFrame
import numpy as np
import matplotlib.pyplot as plt
import sklearn
from sklearn.model_selection import train_test_
from sklearn.neighbors import KNeighborsClassif
from sklearn.metrics import roc_auc_score, roc_
import scipy.stats as stats
from sklearn.metrics import classification_repo
from sklearn.preprocessing import MinMaxScaler
```

# C1:DATA PREPROCESSING

We need to one hot encode the binary qualitative variables as we need our variables to be qualitative. For the quantitative variables we need to fill in missing data, fix outliers, and duplicates.

# C2:DATA SET VARIABLES

For the qualitative variables we need:

- Churn: This is a binary variable with either yes or no values. This variable is categorical.
- Techie: This is a binary variable with either yes or no values. This variable is categorical.
- Port_modem: This is a binary variable with either yes or no values. This variable is categorical.

- Tablet: This is a binary variable with either yes or no values. This variable is categorical.
- Phone: This is a binary variable with either yes or no values. This variable is categorical.
- Multiple:This is a binary variable with either yes or no values. This variable is categorical.
- OnlineSecurity: This is a binary variable with either yes or no values. This variable is categorical.
- OnlineBackup: This is a binary variable with either yes or no values. This variable is categorical.
- DeviceProtection: This is a binary variable with either yes or no values. This variable is categorical.
- TechSupport: This is a binary variable with either yes or no values. This variable is categorical.
- StreamingTV: This is a binary variable with either yes or no values. This variable is categorical.
- StreamingMovies: This is a binary variable with either yes or no values. This variable is categorical.
- PaperlessBilling: This is a binary variable with either yes or no values. This variable is categorical.

All these variables are categorical

For the quantitative we need:

- Population: The average population is 6817, the minimum is 2, and the maximum is 38,597. We can once again see the zeroes have been removed as per our treatment of nulls earlier in the analysis. This variable is numeric.
- Children: The mean, which is calculated by adding up all the values and dividing by the n amount, is around 1.7 children. The most children is 6, and the least is 0. This variable is numeric.
- Age: The average age is around 53. This means many of the customers tend to be older. The youngest customer is 18 and the oldest is 89. This variable is numeric.
- Income: The average income is around 35,688 dollars. The lowest is 348 dollars and the highest is 96,190. This variable is numeric.
- Outage_sec_perweek: On average there is an average outage time of 10 seconds per week. The minimum or lowest time is 4 seconds, and the highest is 15. This is interesting as we learn that there is never a point in time where there is a week without outages. This variable is numeric.
- Email: The avereage number of emails is 12. The minimum is 6 and the maximum is 18. This gives us insight that depending on the customer, different amounts of emails are sent. This may be because the business segments its customers or because some customers joined at different

times and thus were not included in previous emails. This variable is numeric.

- Contacts: The average for this variable is .8. The minimum is 0 and the max is 2. This shows us that customers do not frequently contact customer support, with the most a customer contacting them being 2 recorded times. This variable is numeric.
- Yearly_equip_failure: The average for this variable is .3. The minimum is 0 and the maximum is 1. This shows us that its not frequent for a customer's equipment to fail, and that it will most likely not occur more than once according to our recorded history. This variable is numeric.
- Tenure: The average tenure is around 34.5 months. The minimum is 1 and the max is 72. This shows us that tenure of the customer does generally not last for more than a few years according to our data. This variable is numeric.
- MonthlyCharge: The average monthly charge is about 172 dollars a month. The minimum is 80 and max is 290. This could be due to different customers having different plans, customizable services, and offers. This variable is numeric.
- Bandwidth_gb_year: This is the amount of gb a customer uses per year. On average it is 3392, with the lowest being 155.5 and the highest being 7158.98 gb.

- Item1 through Item8: Items 1 through 8 should all have a minimum of 2 and a max of 5. This could be a result of how we cleaned the data, removing any outliers. This variable is numeric.

All these variables are numeric

In [2]:
```
df = pd.read_csv('churn_clean.csv')
```

In [3]:
```
dfq_cq = ['Churn','Techie','Port_modem','Tablet
```

In [4]:
```
# run a for loop that goes through and uses .de
for column in df:
    print('Variable: ', column,'\n', df[column]
```

```
Variable:  CaseOrder
 count    10000.00000
mean      5000.50000
std       2886.89568
min          1.00000
25%       2500.75000
50%       5000.50000
75%       7500.25000
max      10000.00000
Name: CaseOrder, dtype: float64

Variable:  Customer_id
 count       10000
unique      10000
top       K409198
freq            1
Name: Customer_id, dtype: object

Variable:  Interaction
 count                                    10000
unique                                   10000
top       aa90260b-4141-4a24-8e36-b04ce1f4f77b
freq                                         1
Name: Interaction, dtype: object

Variable:  UID
 count                                10000
unique                               10000
top       e885b299883d4f9fb18e39c75155d990
freq                                     1
Name: UID, dtype: object

Variable:  City
 count       10000
unique       6058
top       Houston
freq           34
Name: City, dtype: object

Variable:  State
```

```
 count     10000
unique      52
top         TX
freq       603
Name: State, dtype: object

Variable:  County
 count          10000
unique          1620
top       Washington
freq             111
Name: County, dtype: object

Variable:  Zip
 count    10000.000000
mean     49153.319600
std      27532.196108
min        601.000000
25%      26292.500000
50%      48869.500000
75%      71866.500000
max      99929.000000
Name: Zip, dtype: float64

Variable:  Lat
 count    10000.000000
mean        38.757567
std          5.437389
min         17.966120
25%         35.341828
50%         39.395800
75%         42.106908
max         70.640660
Name: Lat, dtype: float64

Variable:  Lng
 count    10000.000000
mean       -90.782536
std         15.156142
min       -171.688150
```

```
25%        -97.082812
50%        -87.918800
75%        -80.088745
max        -65.667850
Name: Lng, dtype: float64

Variable:  Population
 count     10000.000000
mean       9756.562400
std       14432.698671
min           0.000000
25%         738.000000
50%        2910.500000
75%       13168.000000
max      111850.000000
Name: Population, dtype: float64

Variable:  Area
 count        10000
unique           3
top       Suburban
freq          3346
Name: Area, dtype: object

Variable:  TimeZone
 count                10000
unique                  25
top       America/New_York
freq                  4072
Name: TimeZone, dtype: object

Variable:  Job
 count                         10000
unique                          639
top       Occupational psychologist
freq                             30
Name: Job, dtype: object

Variable:  Children
 count    10000.0000
```

```
mean         2.0877
std          2.1472
min          0.0000
25%          0.0000
50%          1.0000
75%          3.0000
max         10.0000
Name: Children, dtype: float64

Variable:  Age
 count    10000.000000
mean        53.078400
std         20.698882
min         18.000000
25%         35.000000
50%         53.000000
75%         71.000000
max         89.000000
Name: Age, dtype: float64

Variable:  Income
 count     10000.000000
mean      39806.926771
std       28199.916702
min         348.670000
25%       19224.717500
50%       33170.605000
75%       53246.170000
max      258900.700000
Name: Income, dtype: float64

Variable:  Marital
 count        10000
unique           5
top       Divorced
freq          2092
Name: Marital, dtype: object

Variable:  Gender
 count      10000
```

```
unique          3
top        Female
freq         5025
Name: Gender, dtype: object

Variable:  Churn
 count     10000
unique        2
top          No
freq       7350
Name: Churn, dtype: object

Variable:  Outage_sec_perweek
 count    10000.000000
mean        10.001848
std          2.976019
min          0.099747
25%          8.018214
50%         10.018560
75%         11.969485
max         21.207230
Name: Outage_sec_perweek, dtype: float64

Variable:  Email
 count    10000.000000
mean        12.016000
std          3.025898
min          1.000000
25%         10.000000
50%         12.000000
75%         14.000000
max         23.000000
Name: Email, dtype: float64

Variable:  Contacts
 count    10000.000000
mean         0.994200
std          0.988466
min          0.000000
25%          0.000000
```

```
50%            1.000000
75%            2.000000
max            7.000000
Name: Contacts, dtype: float64

Variable:  Yearly_equip_failure
 count    10000.000000
mean         0.398000
std          0.635953
min          0.000000
25%          0.000000
50%          0.000000
75%          1.000000
max          6.000000
Name: Yearly_equip_failure, dtype: float64

Variable:  Techie
 count     10000
unique        2
top          No
freq       8321
Name: Techie, dtype: object

Variable:  Contract
 count              10000
unique                  3
top       Month-to-month
freq                 5456
Name: Contract, dtype: object

Variable:  Port_modem
 count     10000
unique        2
top          No
freq       5166
Name: Port_modem, dtype: object

Variable:  Tablet
 count     10000
unique        2
```

```
top          No
freq       7009
Name: Tablet, dtype: object

Variable:  InternetService
 count           10000
unique               3
top       Fiber Optic
freq              4408
Name: InternetService, dtype: object

Variable:  Phone
 count     10000
unique        2
top         Yes
freq       9067
Name: Phone, dtype: object

Variable:  Multiple
 count     10000
unique        2
top          No
freq       5392
Name: Multiple, dtype: object

Variable:  OnlineSecurity
 count     10000
unique        2
top          No
freq       6424
Name: OnlineSecurity, dtype: object

Variable:  OnlineBackup
 count     10000
unique        2
top          No
freq       5494
Name: OnlineBackup, dtype: object

Variable:  DeviceProtection
```

```
 count     10000
unique         2
top           No
freq        5614
Name: DeviceProtection, dtype: object

Variable:  TechSupport
 count     10000
unique         2
top           No
freq        6250
Name: TechSupport, dtype: object

Variable:  StreamingTV
 count     10000
unique         2
top           No
freq        5071
Name: StreamingTV, dtype: object

Variable:  StreamingMovies
 count     10000
unique         2
top           No
freq        5110
Name: StreamingMovies, dtype: object

Variable:  PaperlessBilling
 count     10000
unique         2
top          Yes
freq        5882
Name: PaperlessBilling, dtype: object

Variable:  PaymentMethod
 count                 10000
unique                    4
top       Electronic Check
freq                   3398
Name: PaymentMethod, dtype: object
```

```
Variable:  Tenure
 count    10000.000000
mean        34.526188
std         26.443063
min          1.000259
25%          7.917694
50%         35.430507
75%         61.479795
max         71.999280
Name: Tenure, dtype: float64

Variable:  MonthlyCharge
 count    10000.000000
mean       172.624816
std         42.943094
min         79.978860
25%        139.979239
50%        167.484700
75%        200.734725
max        290.160419
Name: MonthlyCharge, dtype: float64

Variable:  Bandwidth_GB_Year
 count    10000.000000
mean      3392.341550
std       2185.294852
min        155.506715
25%       1236.470827
50%       3279.536903
75%       5586.141370
max       7158.981530
Name: Bandwidth_GB_Year, dtype: float64

Variable:  Item1
 count    10000.000000
mean         3.490800
std          1.037797
min          1.000000
25%          3.000000
```

```
50%          3.000000
75%          4.000000
max          7.000000
Name: Item1, dtype: float64

Variable:  Item2
 count    10000.000000
mean         3.505100
std          1.034641
min          1.000000
25%          3.000000
50%          4.000000
75%          4.000000
max          7.000000
Name: Item2, dtype: float64

Variable:  Item3
 count    10000.000000
mean         3.487000
std          1.027977
min          1.000000
25%          3.000000
50%          3.000000
75%          4.000000
max          8.000000
Name: Item3, dtype: float64

Variable:  Item4
 count    10000.000000
mean         3.497500
std          1.025816
min          1.000000
25%          3.000000
50%          3.000000
75%          4.000000
max          7.000000
Name: Item4, dtype: float64

Variable:  Item5
 count    10000.000000
```

```
mean          3.492900
std           1.024819
min           1.000000
25%           3.000000
50%           3.000000
75%           4.000000
max           7.000000
Name: Item5, dtype: float64

Variable:  Item6
 count    10000.000000
mean          3.497300
std           1.033586
min           1.000000
25%           3.000000
50%           3.000000
75%           4.000000
max           8.000000
Name: Item6, dtype: float64

Variable:  Item7
 count    10000.000000
mean          3.509500
std           1.028502
min           1.000000
25%           3.000000
50%           4.000000
75%           4.000000
max           7.000000
Name: Item7, dtype: float64

Variable:  Item8
 count    10000.000000
mean          3.495600
std           1.028633
min           1.000000
25%           3.000000
50%           3.000000
75%           4.000000
max           8.000000
```

```
Name: Item8, dtype: float64
```

## C3:STEPS FOR ANALYSIS

First I will deal with the quantitative variables. It can be seperated into 3 steps: treating nulls, treating duplicates, and treating outliers.

In [5]:
```
dfq = df.drop(['CaseOrder','Customer_id','Inter
dfq
```

Out[5]:

| | Population | Children | Age | Income | Outage_sec_ |
|---|---|---|---|---|---|
| **0** | 38 | 0 | 68 | 28561.99 | |
| **1** | 10446 | 1 | 27 | 21704.77 | 1 |
| **2** | 3735 | 4 | 50 | 9609.57 | 1 |
| **3** | 13863 | 1 | 48 | 18925.23 | 1 |
| **4** | 11352 | 0 | 83 | 40074.19 | |
| **...** | ... | ... | ... | ... | |
| **9995** | 640 | 3 | 23 | 55723.74 | |
| **9996** | 77168 | 4 | 48 | 34129.34 | |
| **9997** | 406 | 1 | 48 | 45983.43 | |
| **9998** | 35575 | 1 | 39 | 16667.58 | 1 |
| **9999** | 12230 | 1 | 28 | 9020.92 | 1 |

10000 rows × 19 columns

## Treat Nulls

We begin by checking the dataframe for nulls. We can use .isnlull().sum() to look through the variables and see if there is any missing data. Using this function we can see that there are no nulls present in the data. Another thing I would like to check is population. This is because for a value like this, it cannot be 0 since it should count the customer. Using the nsmallest() function, we can see that zeroes do exist within the data. I would like to drop those zeroes and replace it with the median as the distribution is skewed right. We determine the distribution by creating a histogram of population. After dropping all the zero values from population and replacing them with median, we can see our minimum is no longer zero.

In [6]:
```
dfq.isnull().sum()
```

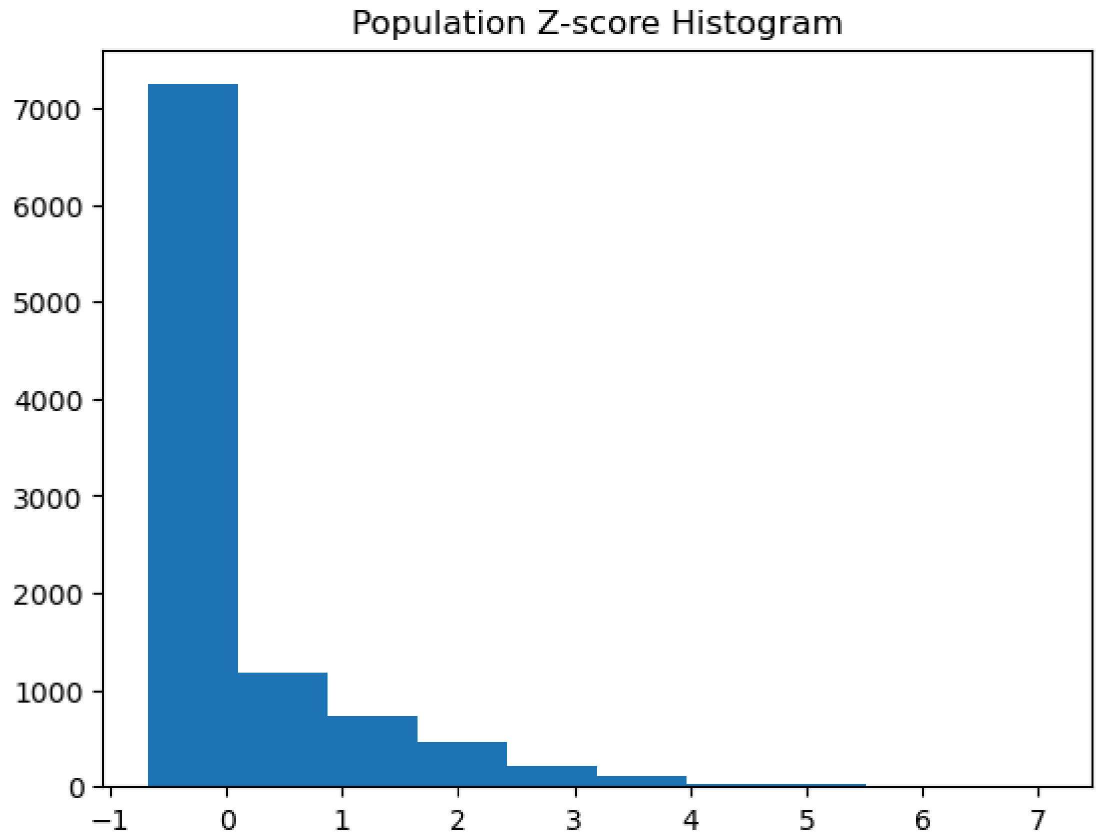
```
Out[6]: Population              0
Children                0
Age                     0
Income                  0
Outage_sec_perweek      0
Email                   0
Contacts                0
Yearly_equip_failure    0
Tenure                  0
MonthlyCharge           0
Bandwidth_GB_Year       0
Item1                   0
Item2                   0
Item3                   0
Item4                   0
Item5                   0
Item6                   0
Item7                   0
Item8                   0
dtype: int64
```

```
In [7]: # Check the poulation for zeroes
dfq.Population.nsmallest(n=10)
```

```
Out[7]: 13     0
422    0
428    0
434    0
446    0
682    0
694    0
719    0
814    0
839    0
Name: Population, dtype: int64
```

```
In [8]: # create hist for population
dfq['zscore'] = stats.zscore(dfq['Population'])
plt.hist(dfq['zscore'])
```

```
plt.title('Population Z-score Histogram')
plt.show()
```




In [9]:
```
# drop all zeroes
dfq['Population'] = np.where(dfq['Population']
# fill with median as it is skewed right
dfq['Population'] = dfq['Population'].fillna(df
```

In [10]:
```
# Check the poulation for zeroes
dfq.Population.nsmallest(n=10)
```

Out[10]:
```
4453    2.0
261     4.0
3475    4.0
6018    4.0
2613    5.0
2092    6.0
2192    6.0
5054    6.0
5149    6.0
6048    6.0
Name: Population, dtype: float64
```

In [11]:
```
#drop zscore
dfq = dfq.drop(['zscore'],axis=1)
```

## Treat Duplicates

Next, we will check to see if there are any duplicates in the data. We can do this by using .duplicated().value_counts() which will output a true or false depending on whether or not duplicates exist within the dataframe. We can see from the output of false 10,000 times that there are no duplicates within the data.

In [12]:
```
dfq.duplicated().value_counts()
```

Out[12]:
```
False    10000
dtype: int64
```

## Treat Outliers

We can start by checking the histograms of all of our quantiative variables. After looking through it, the distributions are as follows:

- 'Population' - skewed right
- 'Children' - skewed right
- 'Age' - uniform
- 'Income' - skewed right
- 'Outage_sec_perweek' - normal
- 'Email' - normal
- 'Contacts' - skewed right
- 'Yearly_equip_failure' - skewed right
- 'Tenure' - bimodal
- 'MonthlyCharge' - normal
- 'Bandwidth_GB_Year' - bimodal
- 'Item1' - normal
- 'Item2' - normal
- 'Item3' - normal
- 'Item4' - normal
- 'Item5' - normal
- 'Item6' - normal
- 'Item7' - normal
- 'Item8' - normal

This is useful information to note for later. We can also identify from our histograms if the data passes 3 standard deviations. I will use that as a cutoff for what we identify as outliers. Using this benchmark, the following variables contain outliers:

- Population, Children, Income, Outage_sec_perweek, Email, Contacts, Yearly_equip_failure, Item1, Item2, Item3, Item4, Item5, Item6, Item7, and Item8

Now that I know what variables are the ones that need to be solved, I can run a for loop to drop the outliers which are values equivalent to a z-score greater or less than 3 and -3 three respectively. We also need to know the distribution to understand what we need to imputer these variables with. For population we impute with median since it is skewed right. For children, we use median since it's skewed right. For income we use median. For outage_sec_perweek we use mean since it is distributed normally. For Email we use mean. For Contacts we use median. For Yearly_equip_failure we use median. For item1 through item8 we use mean.

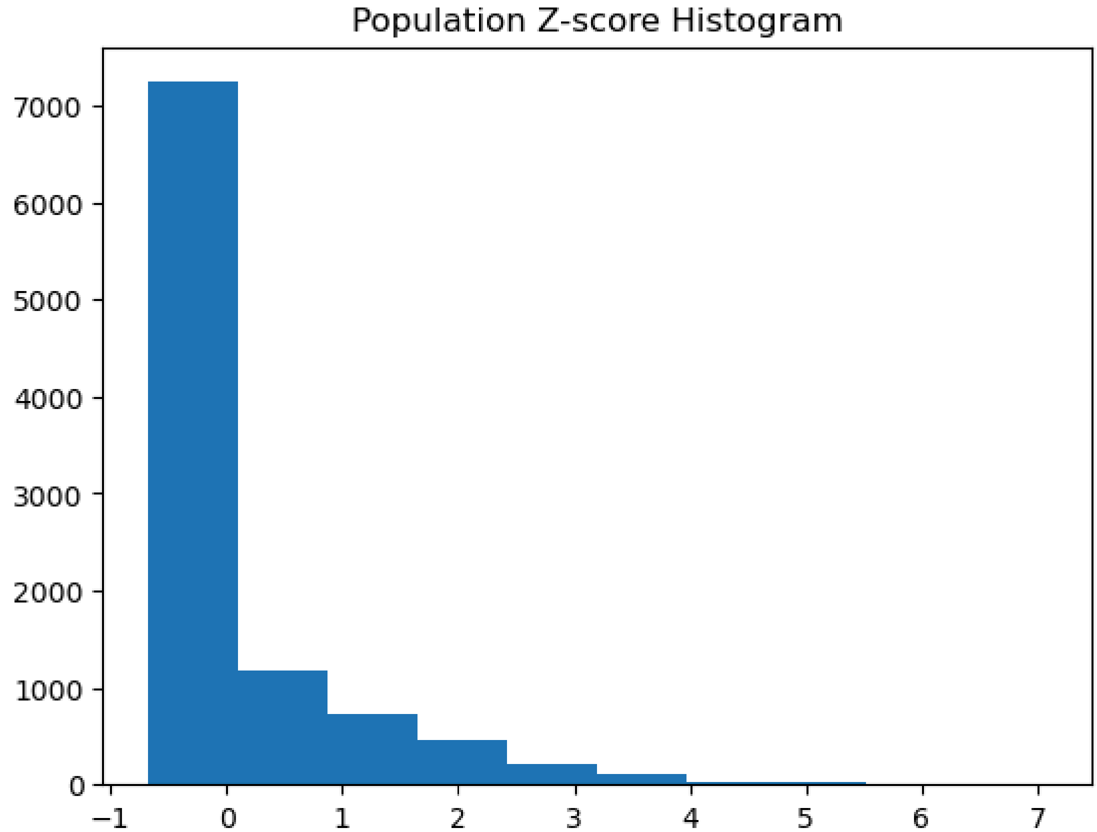
After the for loop runs for both median and for mean, we are able to see that the histograms are fixed and the outliers have been treated.

In [13]:

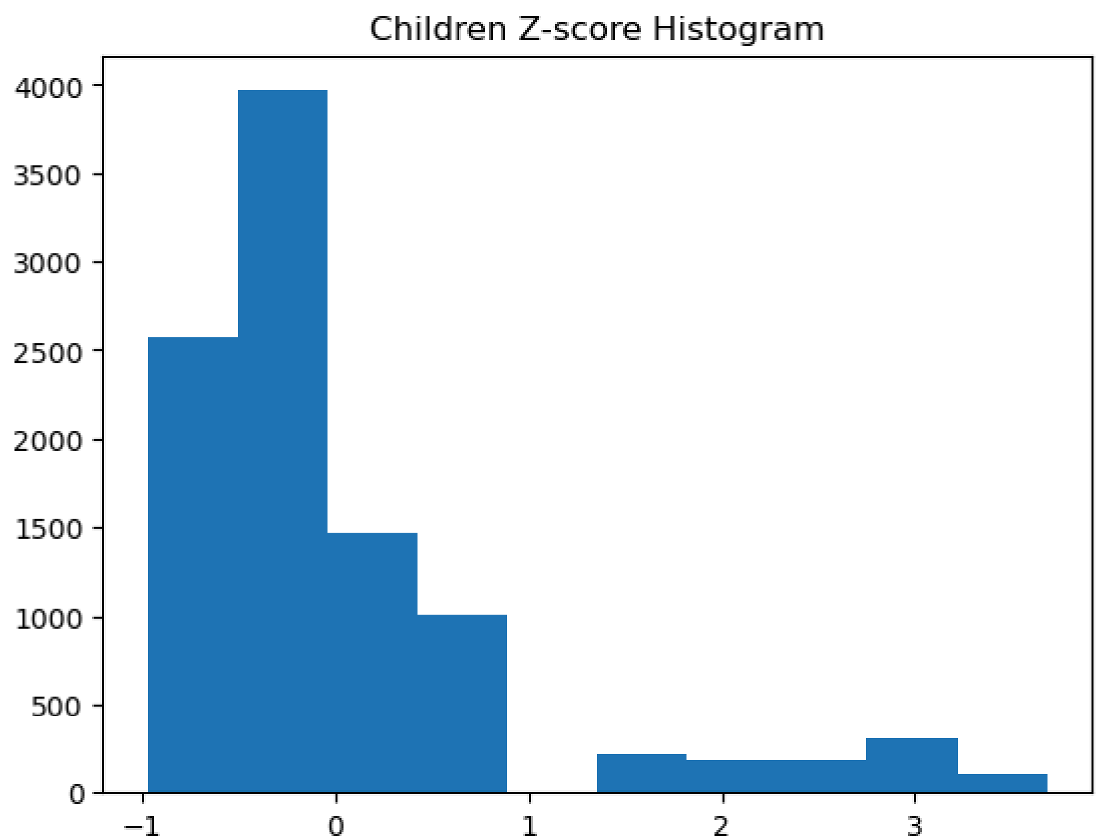
```
# create a list of columns
dfq_c = dfq.columns.tolist()
dfq_c
```

Out[13]:
```
['Population',
 'Children',
 'Age',
 'Income',
 'Outage_sec_perweek',
 'Email',
 'Contacts',
 'Yearly_equip_failure',
 'Tenure',
 'MonthlyCharge',
 'Bandwidth_GB_Year',
 'Item1',
 'Item2',
 'Item3',
 'Item4',
 'Item5',
 'Item6',
 'Item7',
 'Item8']
```

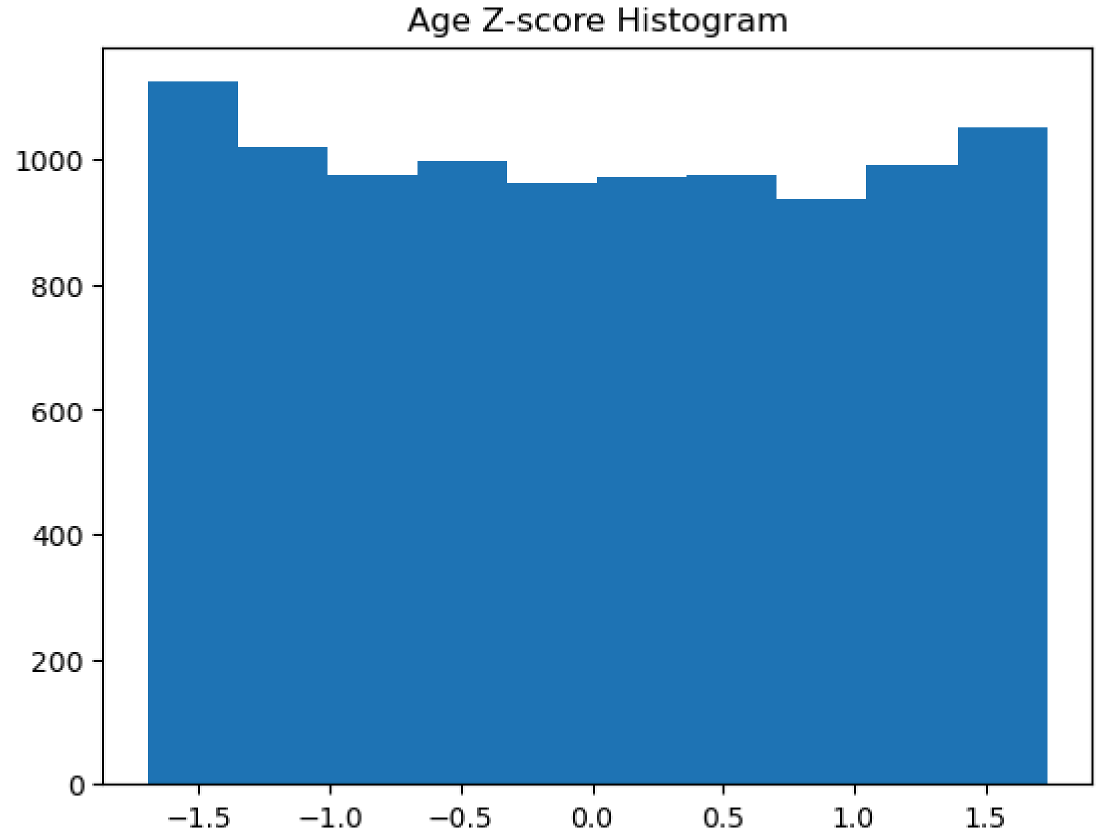
In [14]:
```
for column in dfq_c:
    dfq['zscore'] = stats.zscore(dfq[column])
    plt.hist(dfq['zscore'])
    plt.title(column + ' Z-score Histogram')
    plt.show()
```

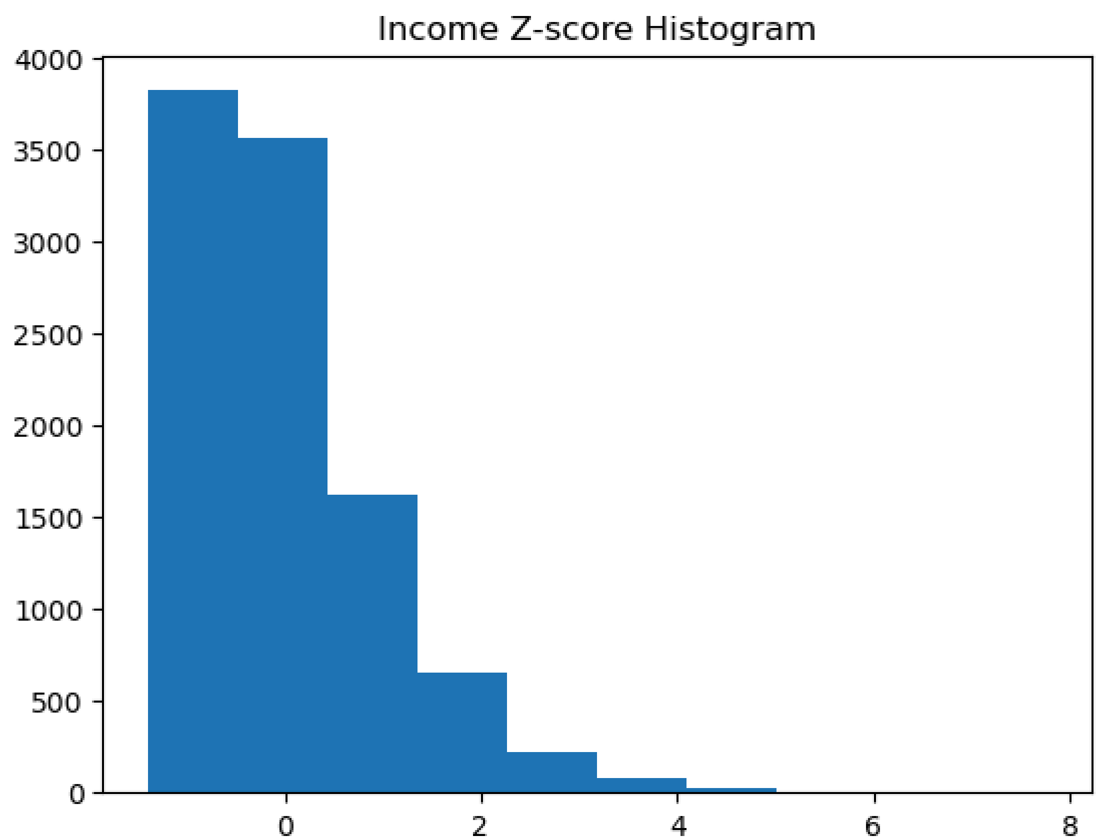


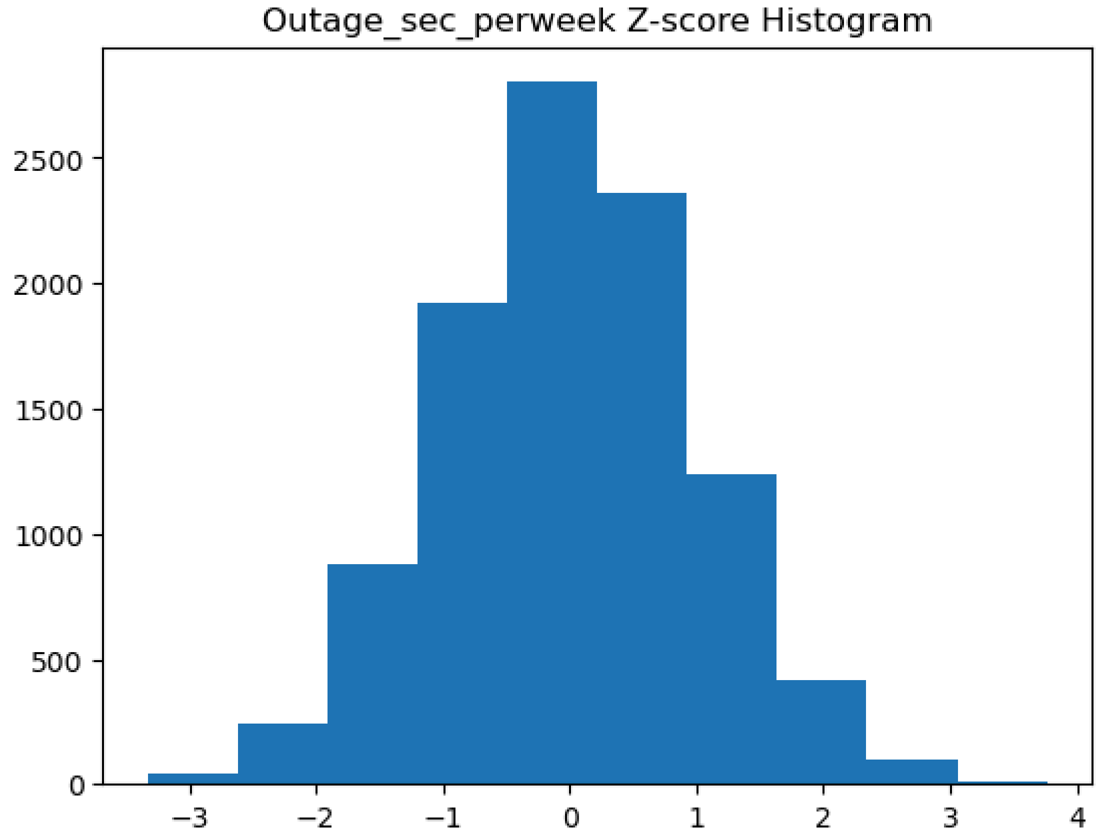
Population Z-score Histogram
7000
6000
5000
4000
3000
2000
1000
0
−1
0
1
2
3
4
5
6
7

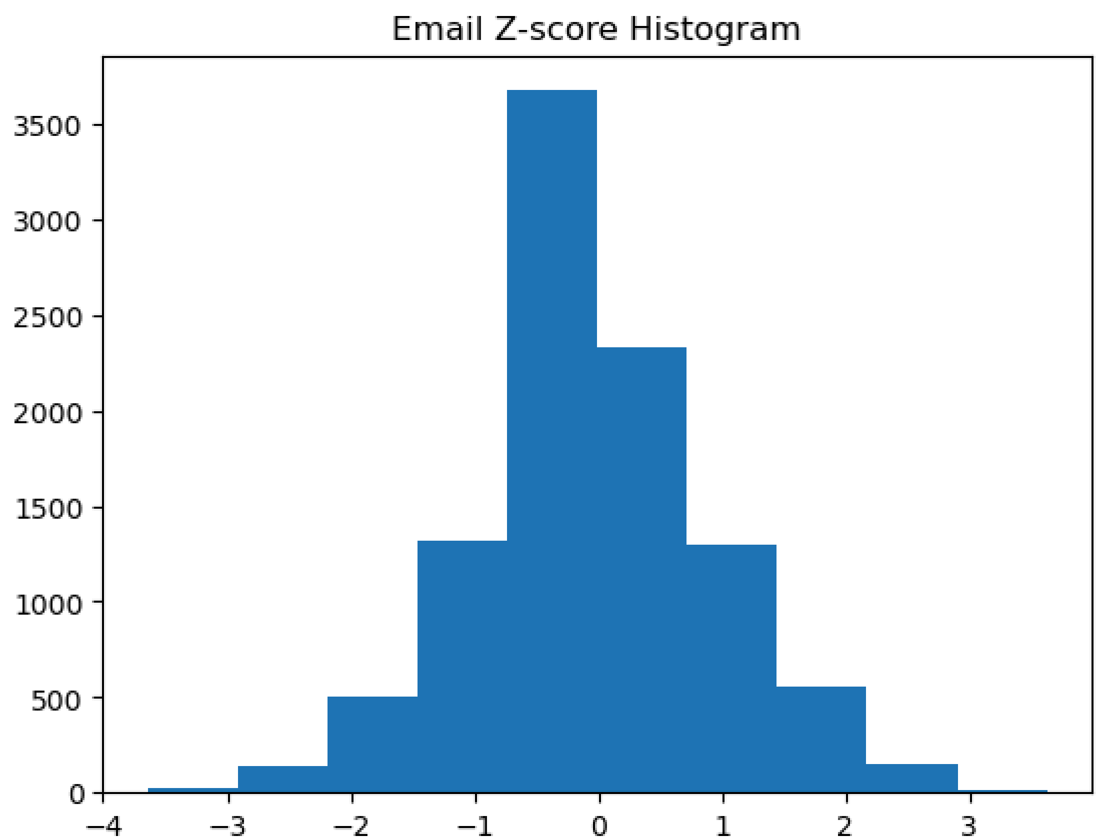



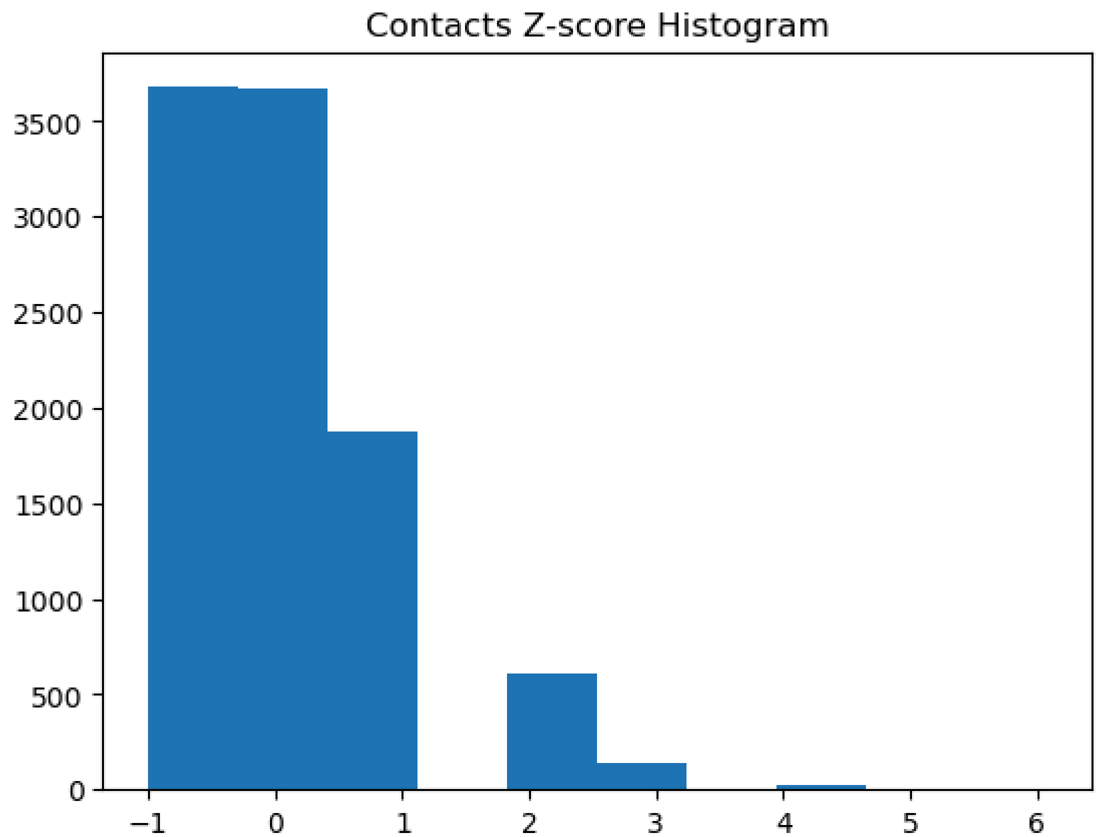
Children Z-score Histogram
4000
3500
3000
2500
2000
1500
1000
500
0
−1
0
1
2
3

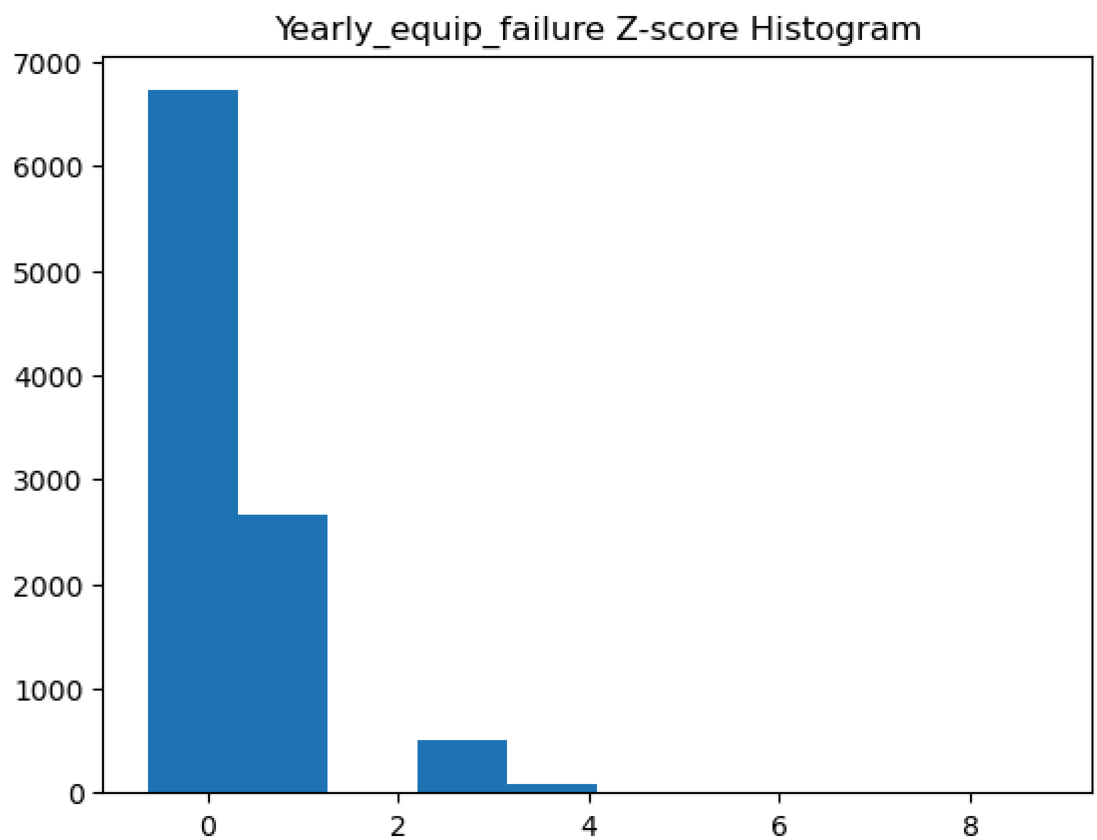

Age Z-score Histogram
0
200
400
600
800
1000
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

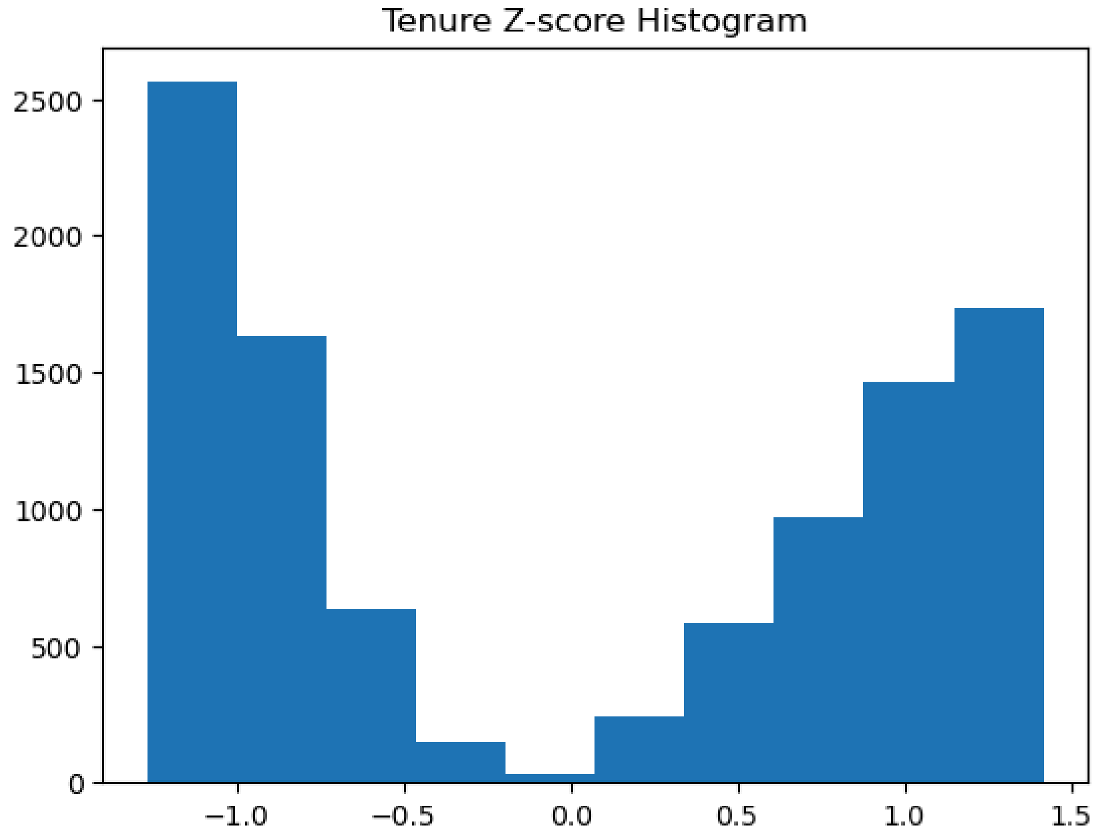

Income Z-score Histogram
0
500
1000
1500
2000
2500
3000
3500
4000
0
2
4
6
8

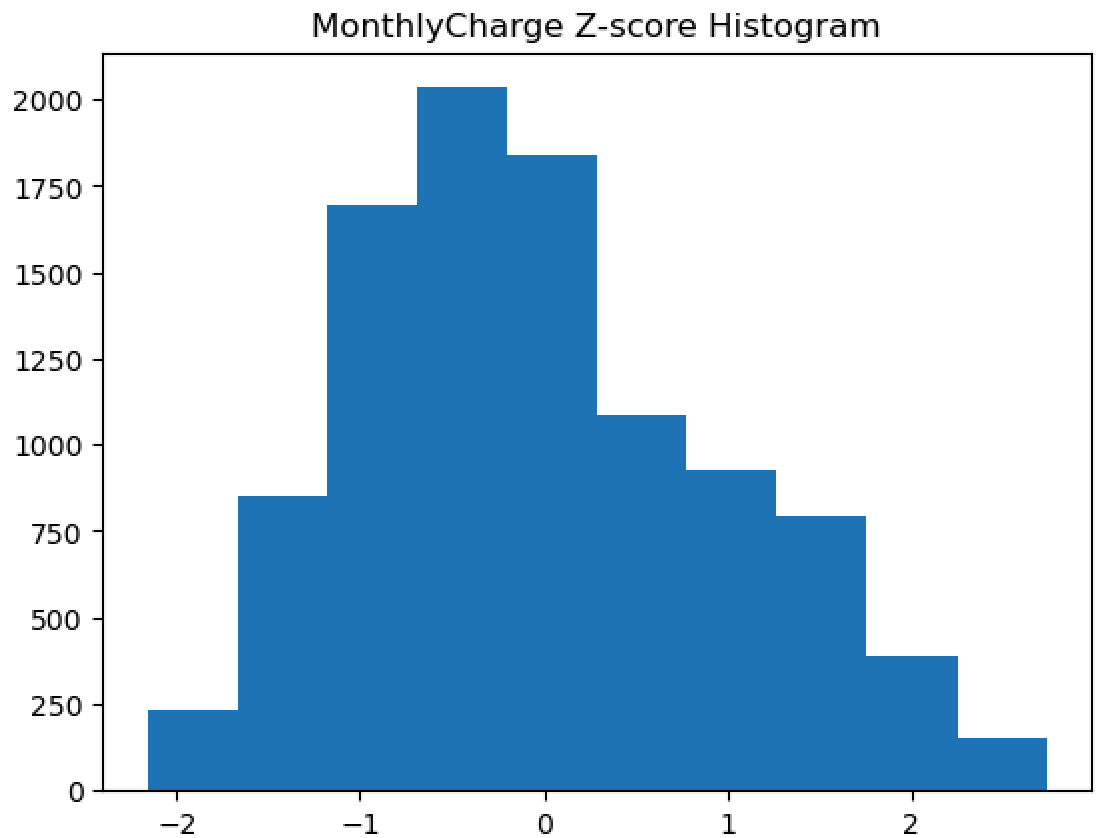

Outage_sec_perweek Z-score Histogram
0
500
1000
1500
2000
2500
−3
−2
−1
0
1
2
3
4

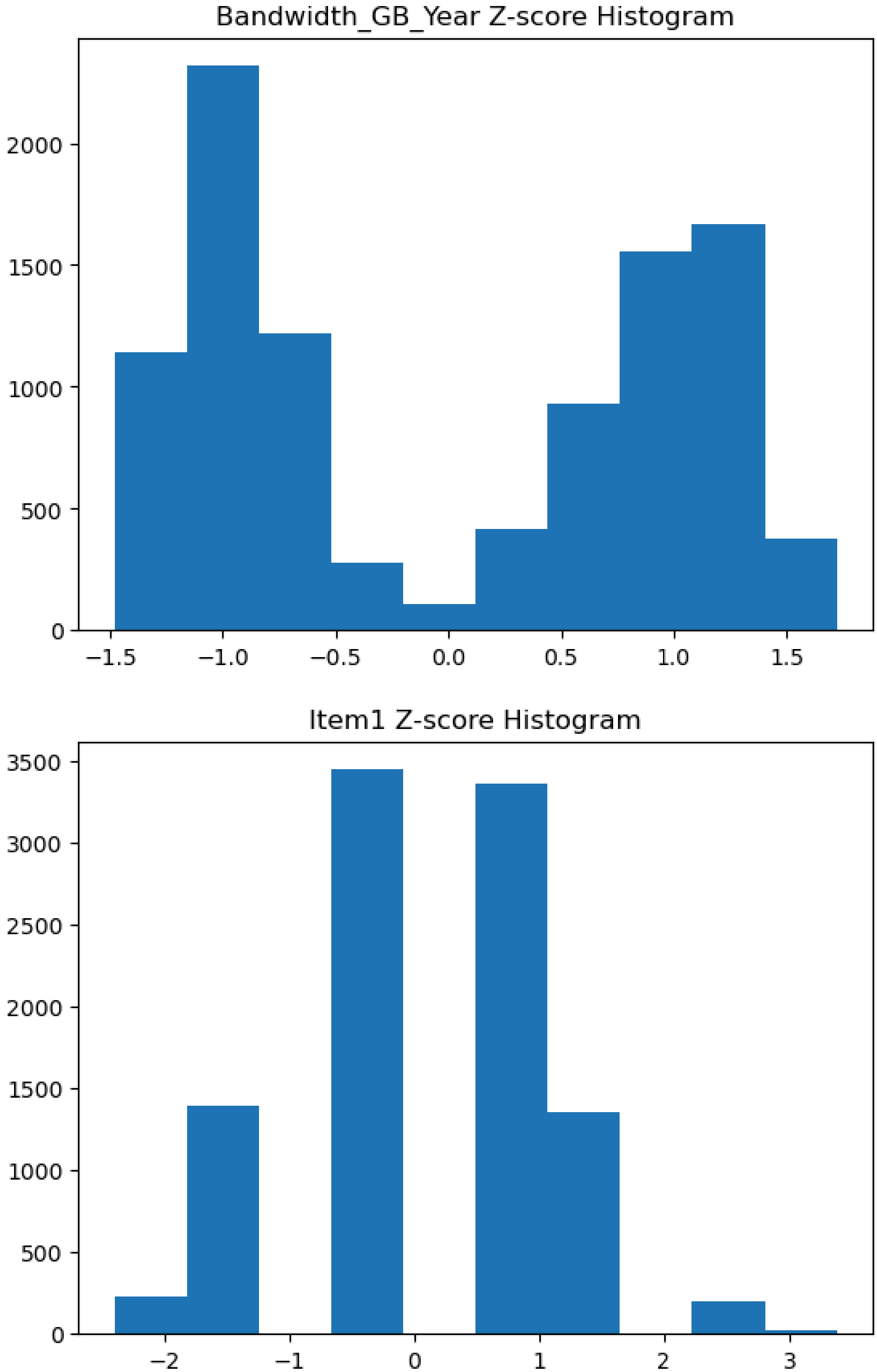

Email Z-score Histogram
0
500
1000
1500
2000
2500
3000
3500
−4
−3
−2
−1
0
1
2
3

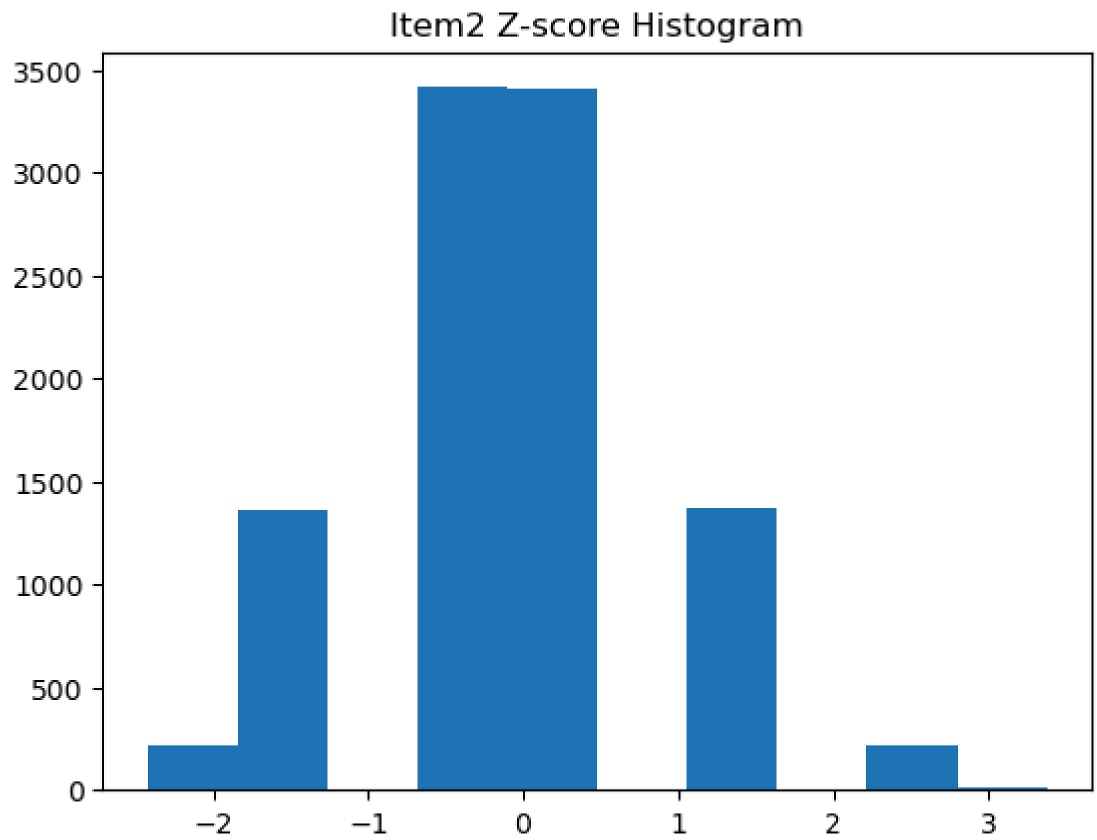

Contacts Z-score Histogram
3500
3000
2500
2000
1500
1000
500
0
−1
0
1
2
3
4
5
6

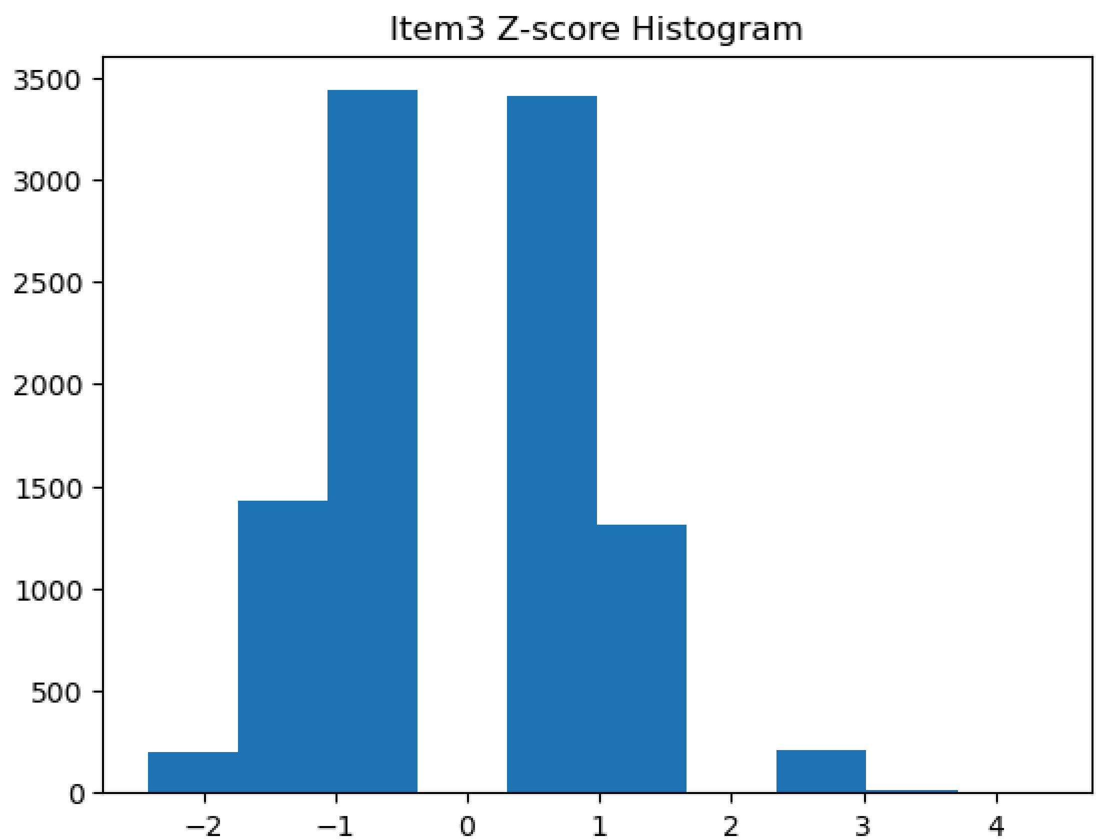

Yearly_equip_failure Z-score Histogram
7000
6000
5000
4000
3000
2000
1000
0
0
2
4
6
8

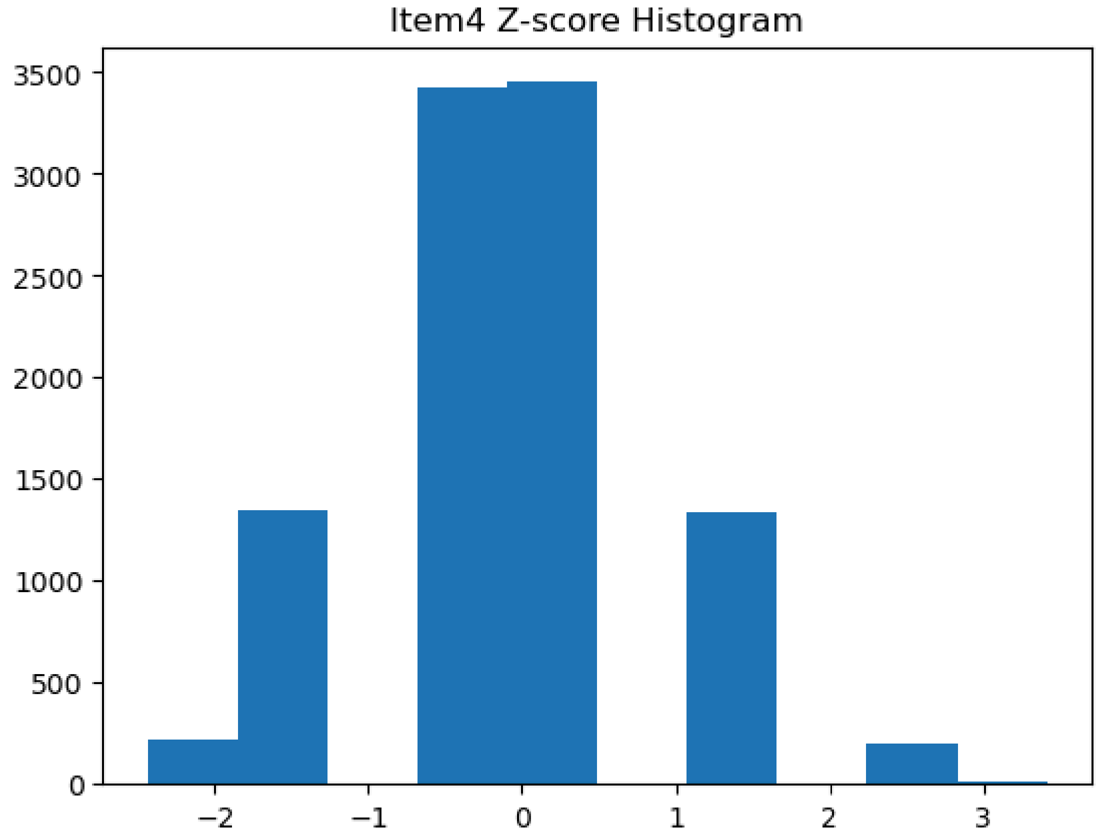

Tenure Z-score Histogram
2500
2000
1500
1000
500
0
−1.0
−0.5
0.0
0.5
1.0
1.5

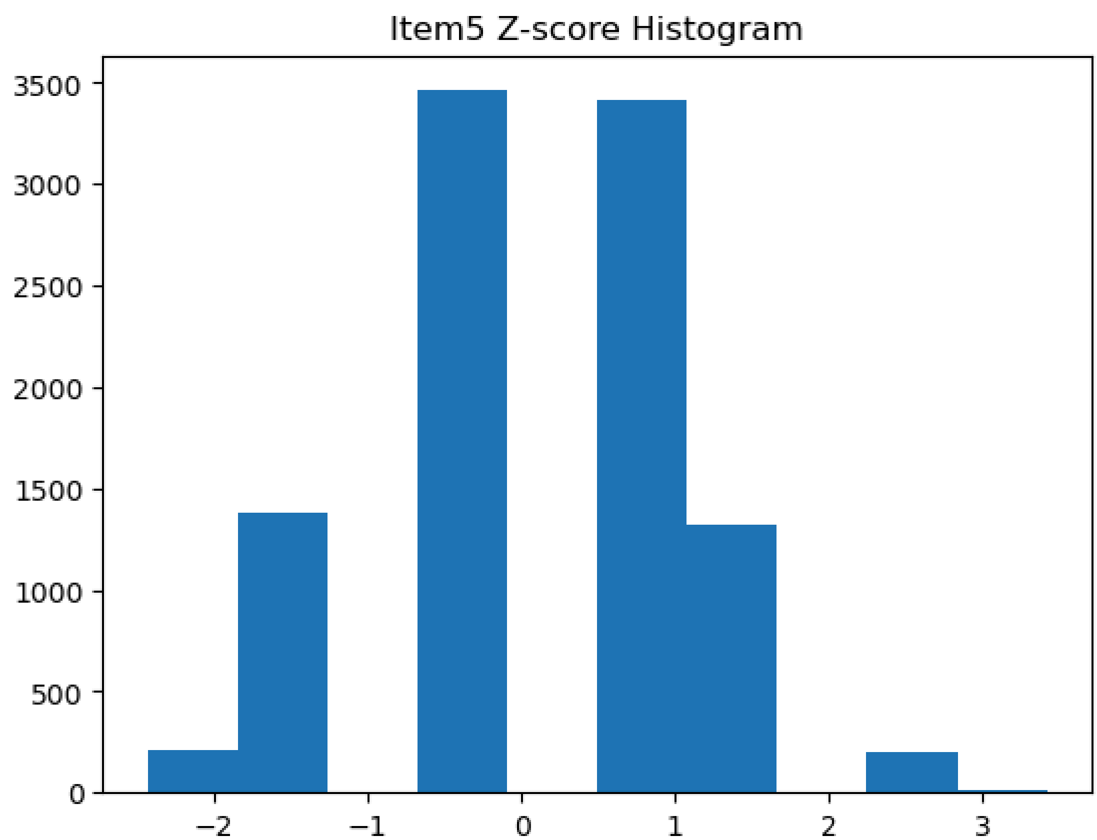

MonthlyCharge Z-score Histogram
2000
1750
1500
1250
1000
750
500
250
0
−2
−1
0
1
2

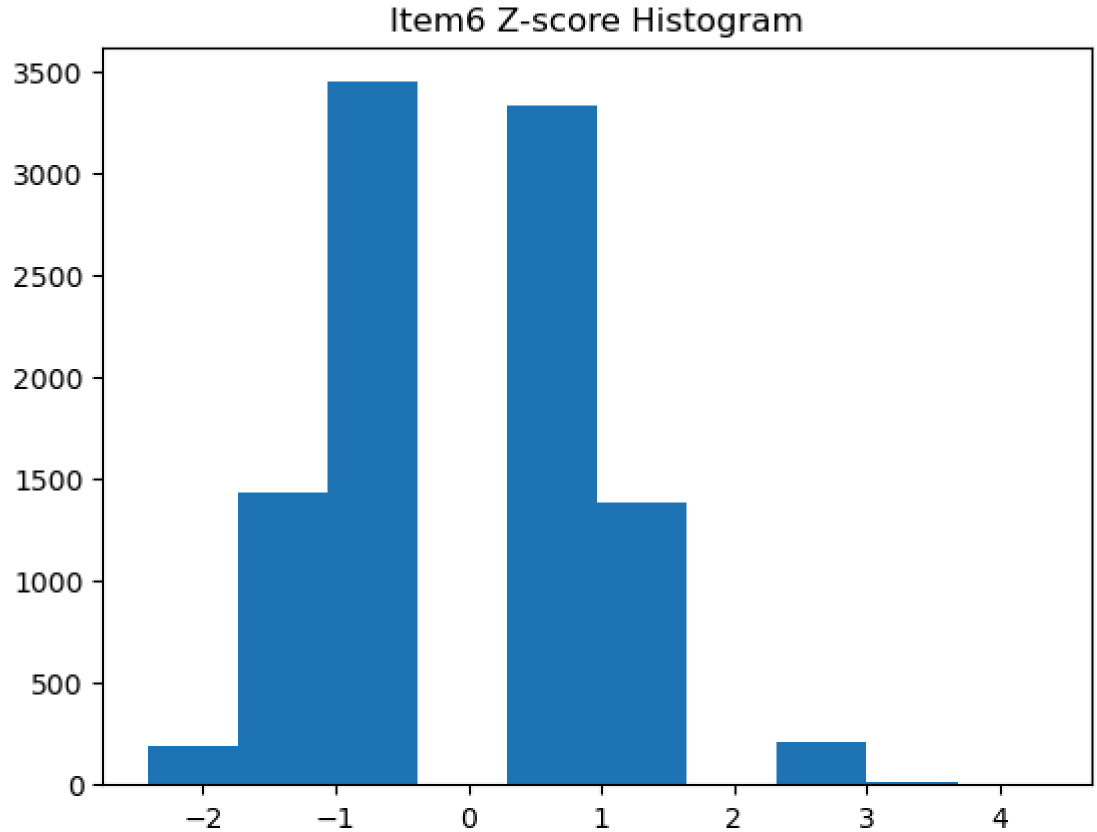

Bandwidth_GB_Year Z-score Histogram
0
500
1000
1500
2000
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
Item1 Z-score Histogram
0
500
1000
1500
2000
2500
3000
3500
−2
−1
0
1
2
3

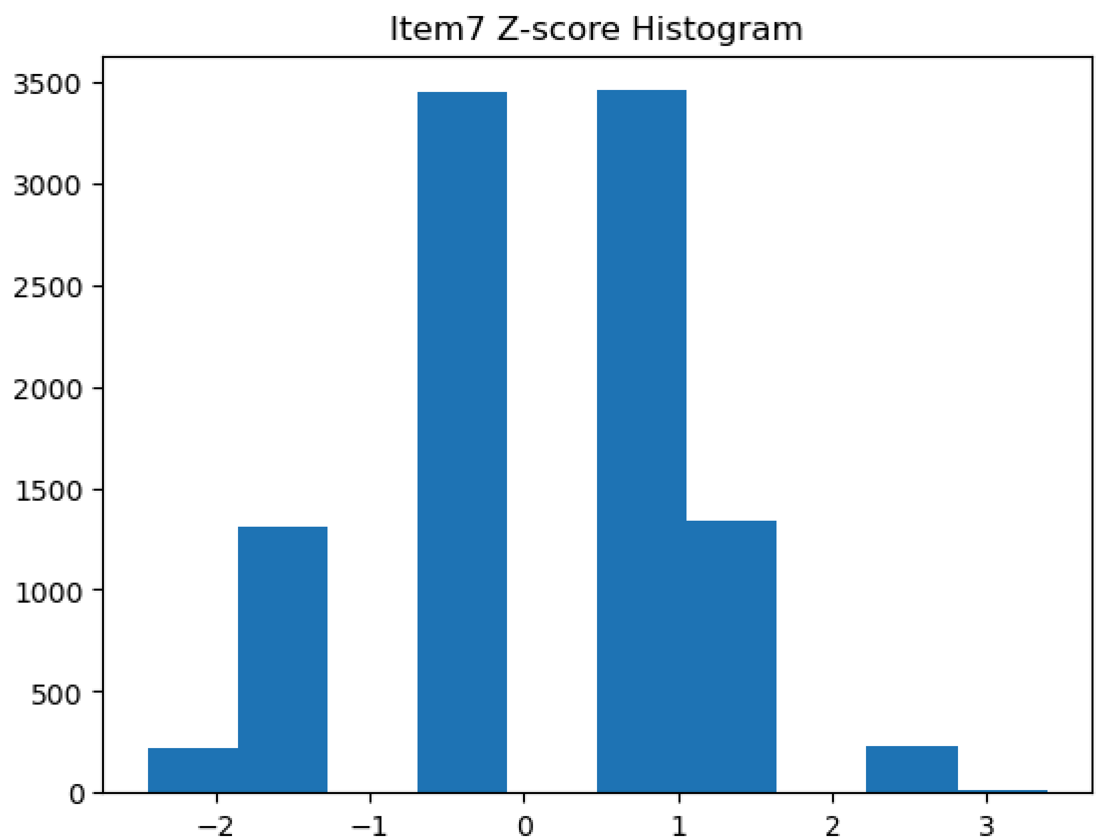

Item2 Z-score Histogram
3500
3000
2500
2000
1500
1000
500
0
−2
−1
0
1
2
3

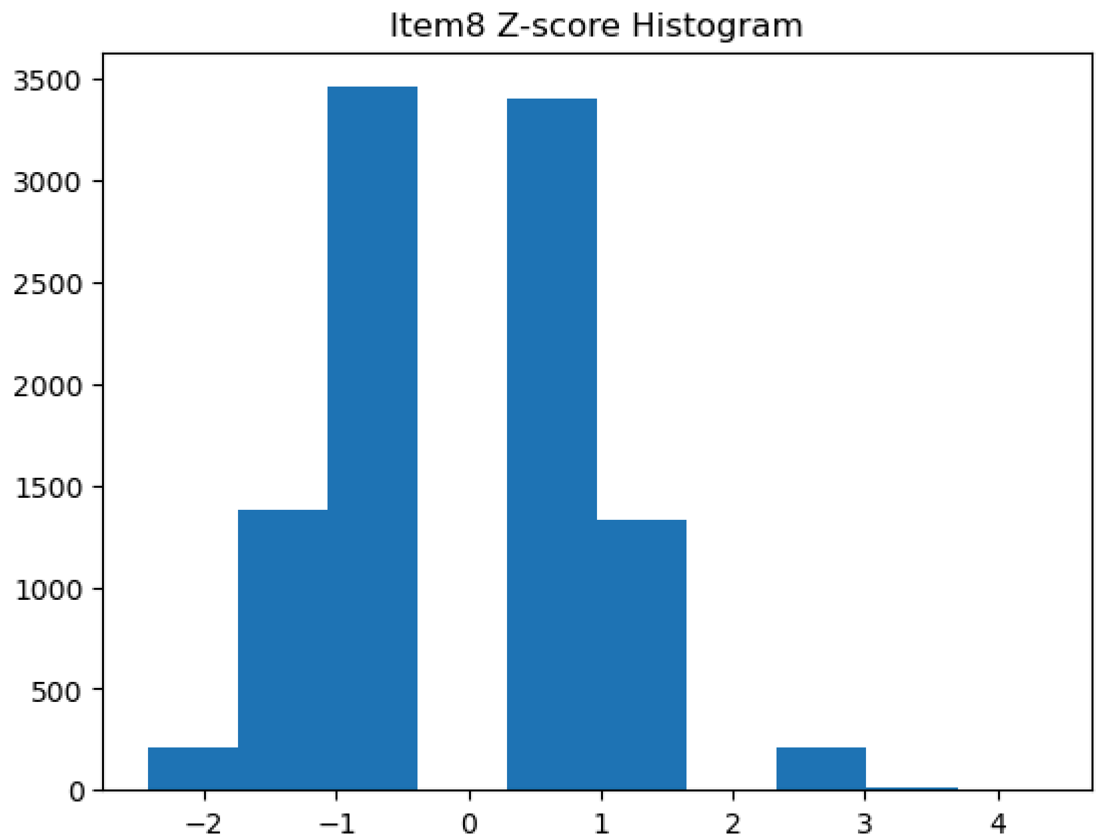

Item3 Z-score Histogram
3500
3000
2500
2000
1500
1000
500
0
−2
−1
0
1
2
3
4


Item4 Z-score Histogram
3500
3000
2500
2000
1500
1000
500
0
−2
−1
0
1
2
3


Item5 Z-score Histogram
3500
3000
2500
2000
1500
1000
500
0
−2
−1
0
1
2
3


Item6 Z-score Histogram
3500
3000
2500
2000
1500
1000
500
0
−2
−1
0
1
2
3
4


Item7 Z-score Histogram
3500
3000
2500
2000
1500
1000
500
0
−2
−1
0
1
2
3

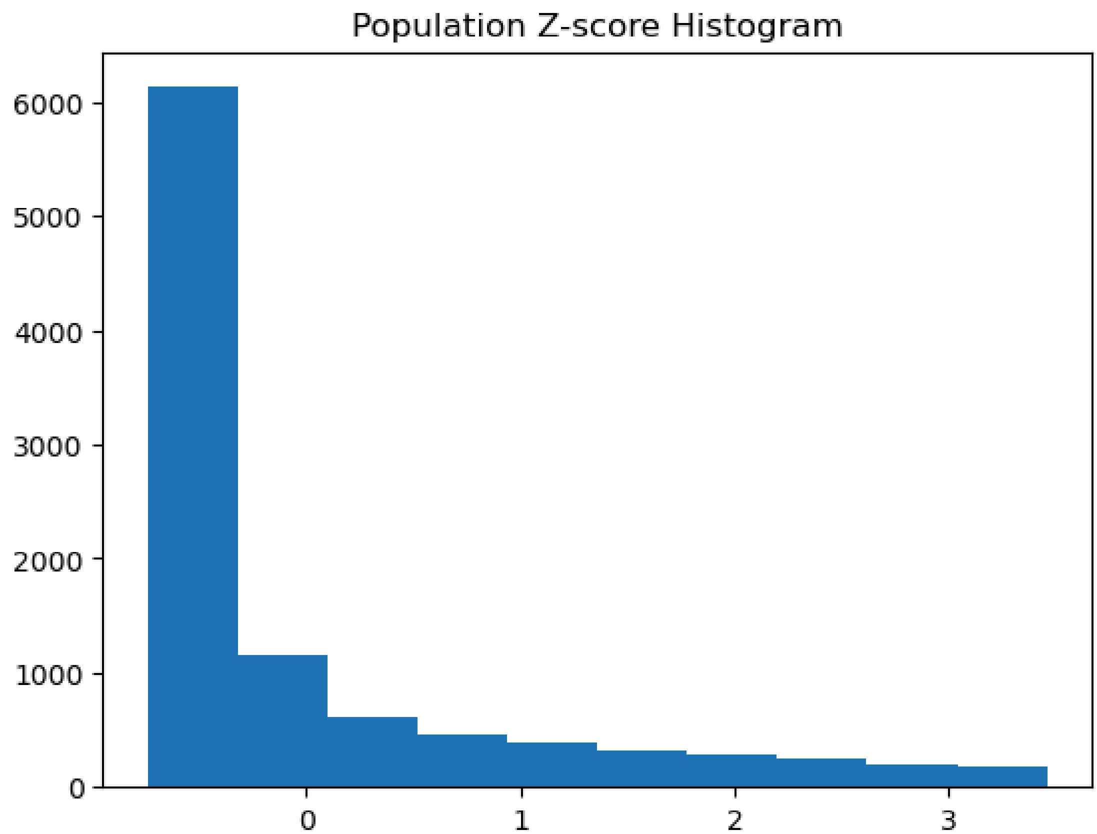


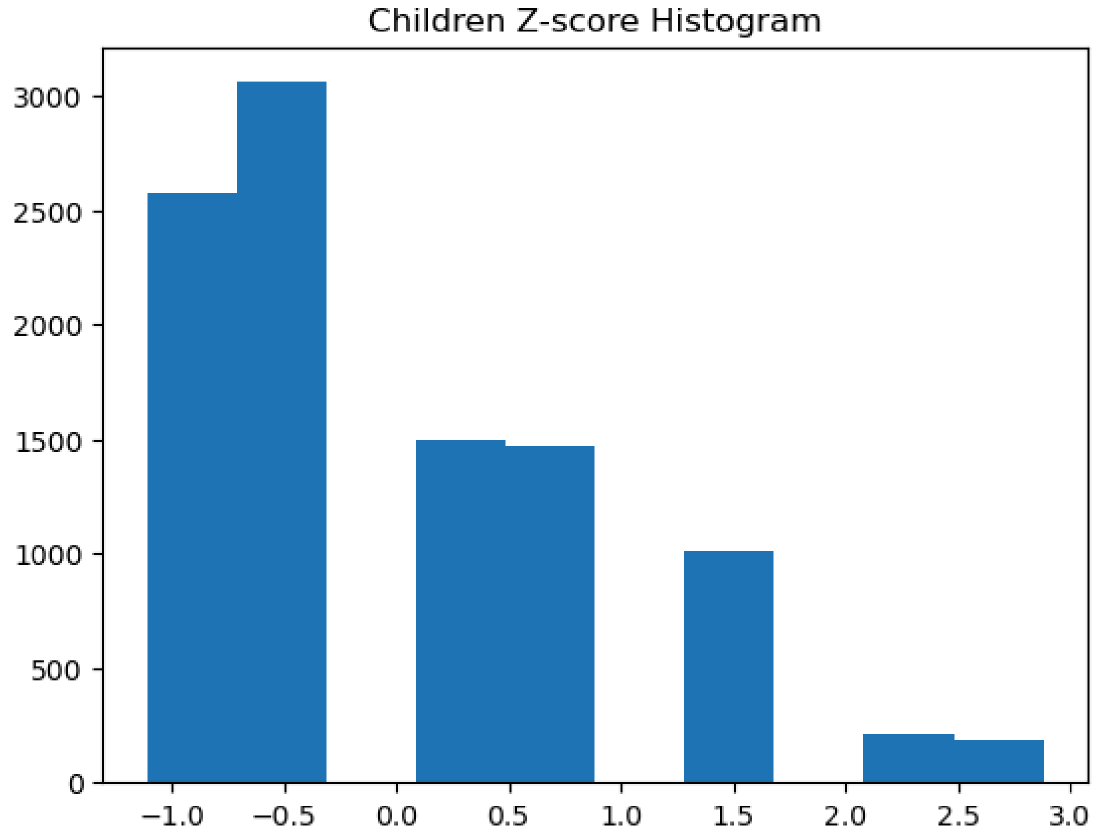

In [15]:
```
# Run a for loop for all the identified variabl
dfq_z_median = ['Population', 'Children', 'Inco
dfq_z_median
for column in dfq_z_median:
    # create nulls for outliers in population
    dfq['zscore'] = stats.zscore(dfq[column])
    dfq[column] = np.where(dfq['zscore'] > 2, n
    dfq[column] = np.where(dfq['zscore'] < -2,
    # use fillna function to impute outliers wi
    dfq[column] = dfq[column].fillna(dfq[column
```

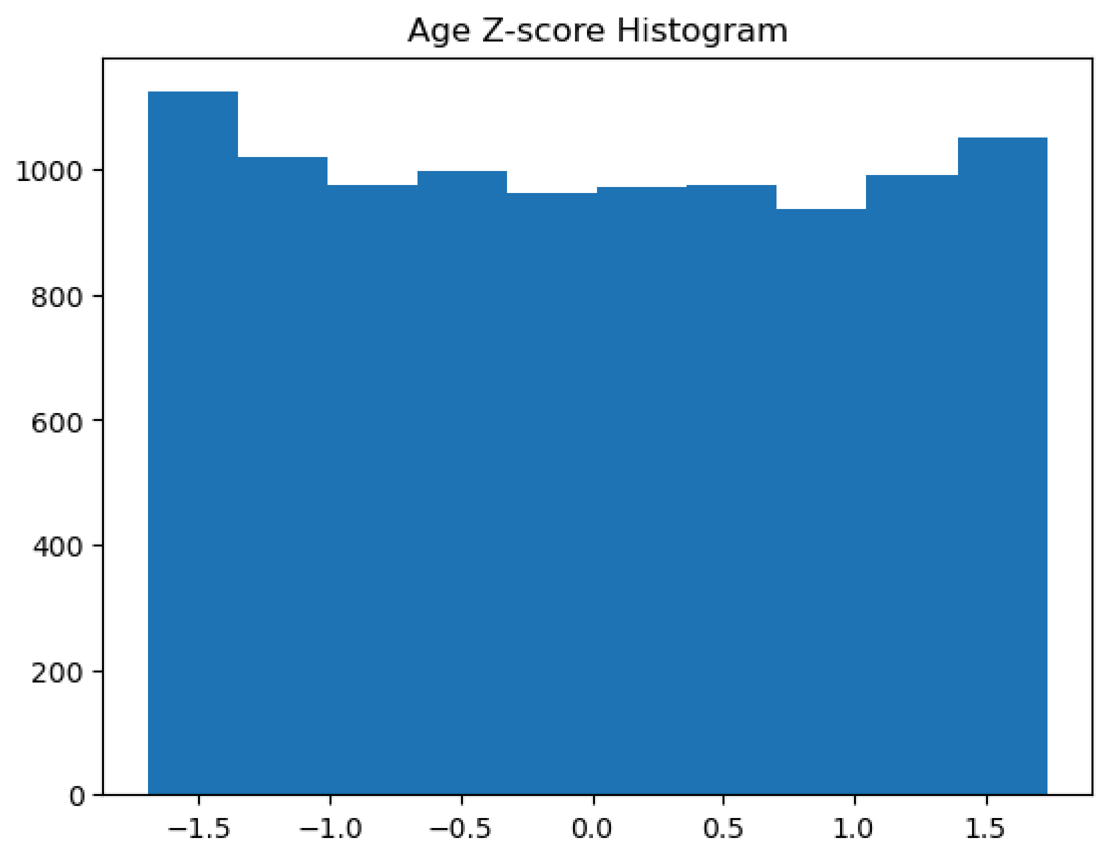
In [16]:
```
# Run a for loop for all the identified variabl
dfq_z_mean = ['Outage_sec_perweek', 'Email', 'I
dfq_z_mean
for column in dfq_z_mean:
     # create nulls for outliers in population
    dfq['zscore'] = stats.zscore(dfq[column])
    dfq[column] = np.where(dfq['zscore'] > 2, n
    dfq[column] = np.where(dfq['zscore'] < -2,
```

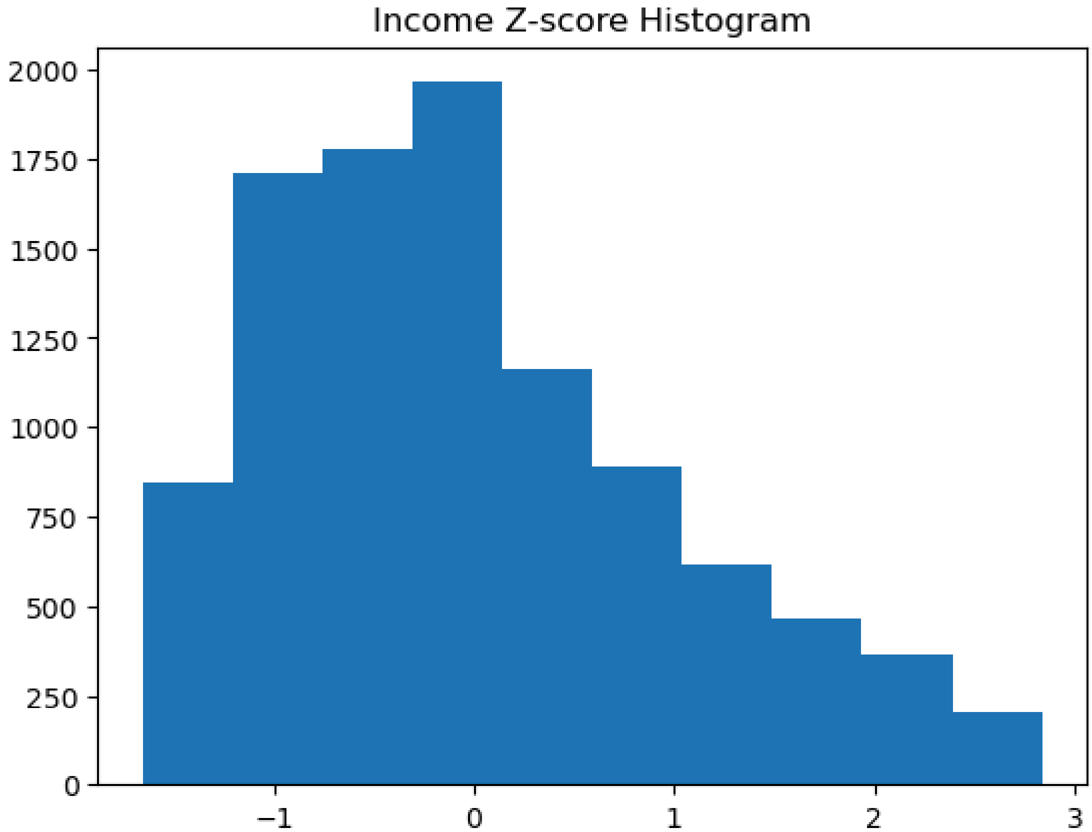
```
    # use fillna function to impute outliers wi
    dfq[column] = dfq[column].fillna(dfq[column
```

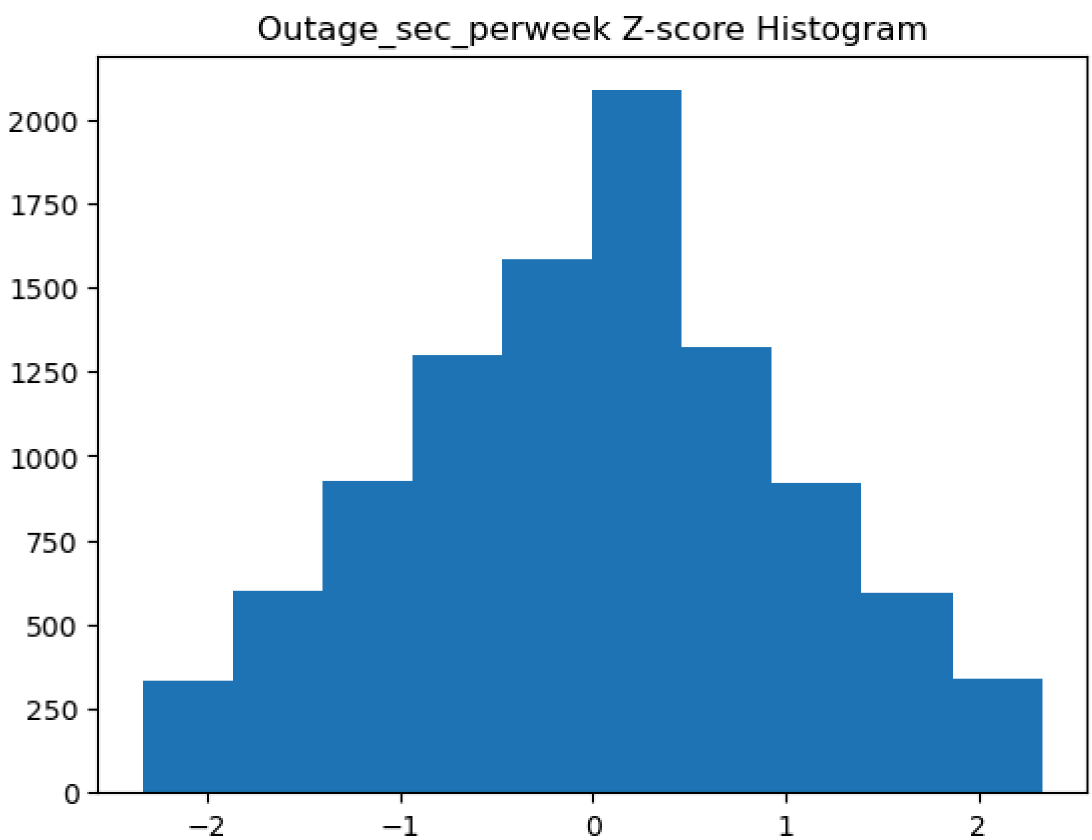
In [17]:
```
# check new histograms
for column in dfq_c:
    dfq['zscore'] = stats.zscore(dfq[column])
    plt.hist(dfq['zscore'])
    plt.title(column + ' Z-score Histogram')
    plt.show()
```

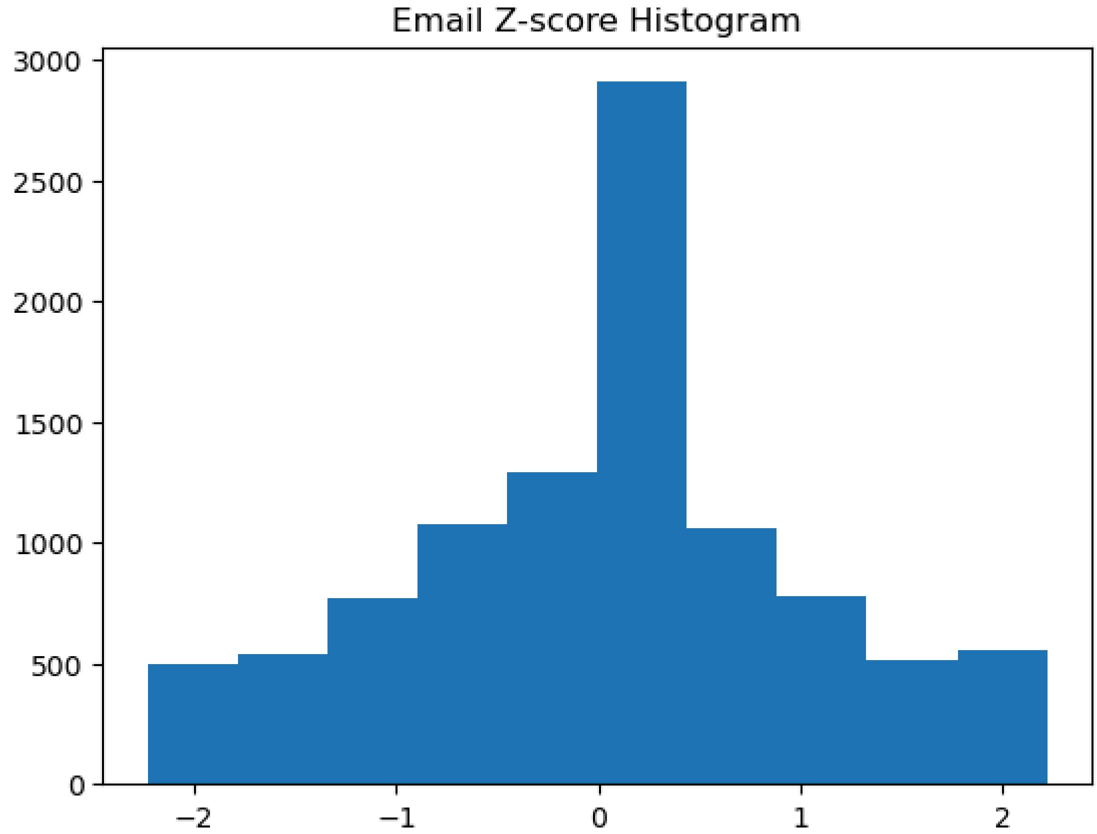



Children Z-score Histogram
3000
2500
2000
1500
1000
500
0
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
3.0

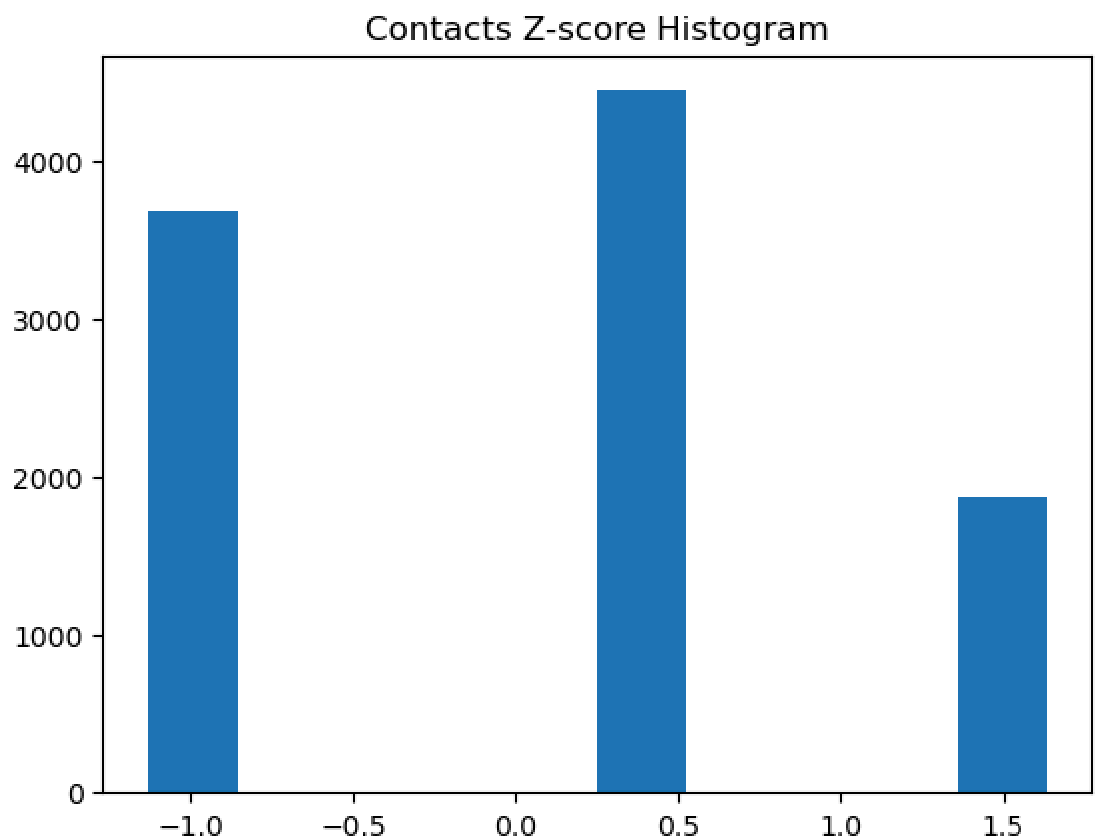

Age Z-score Histogram
1000
800
600
400
200
0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5


Income Z-score Histogram
0
250
500
750
1000
1250
1500
1750
2000
−1
0
1
2
3

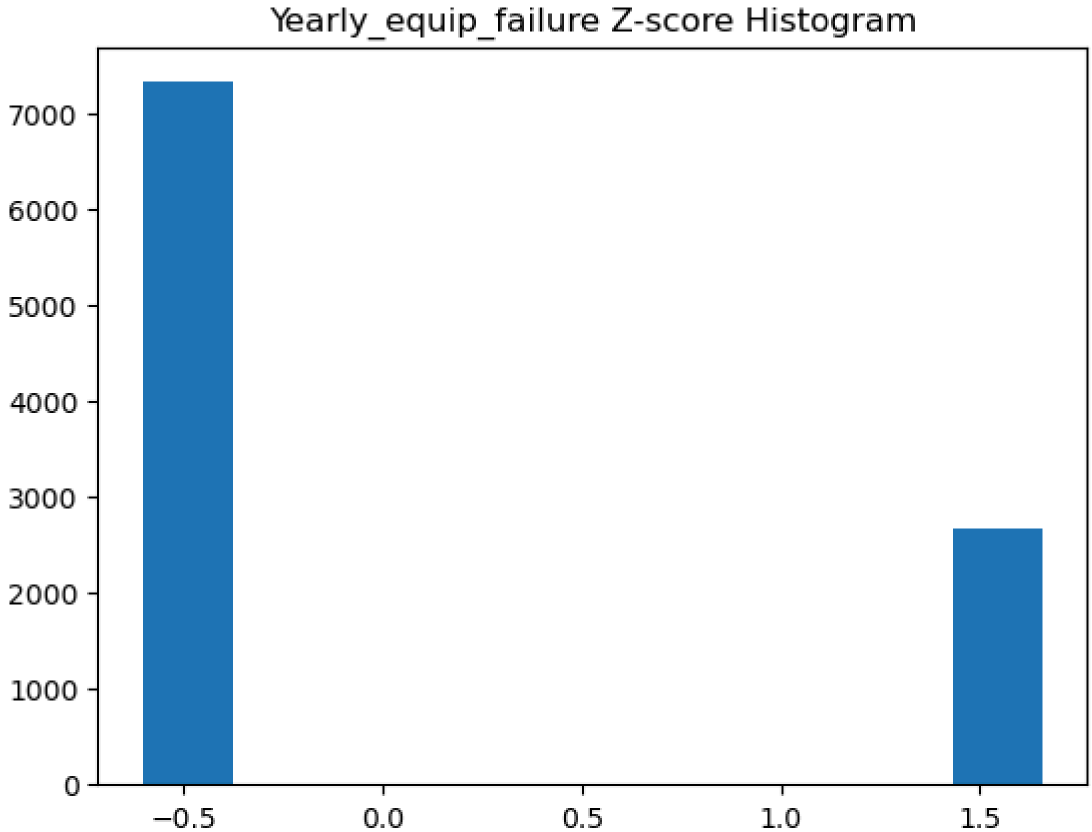

Outage_sec_perweek Z-score Histogram
0
250
500
750
1000
1250
1500
1750
2000
−2
−1
0
1
2

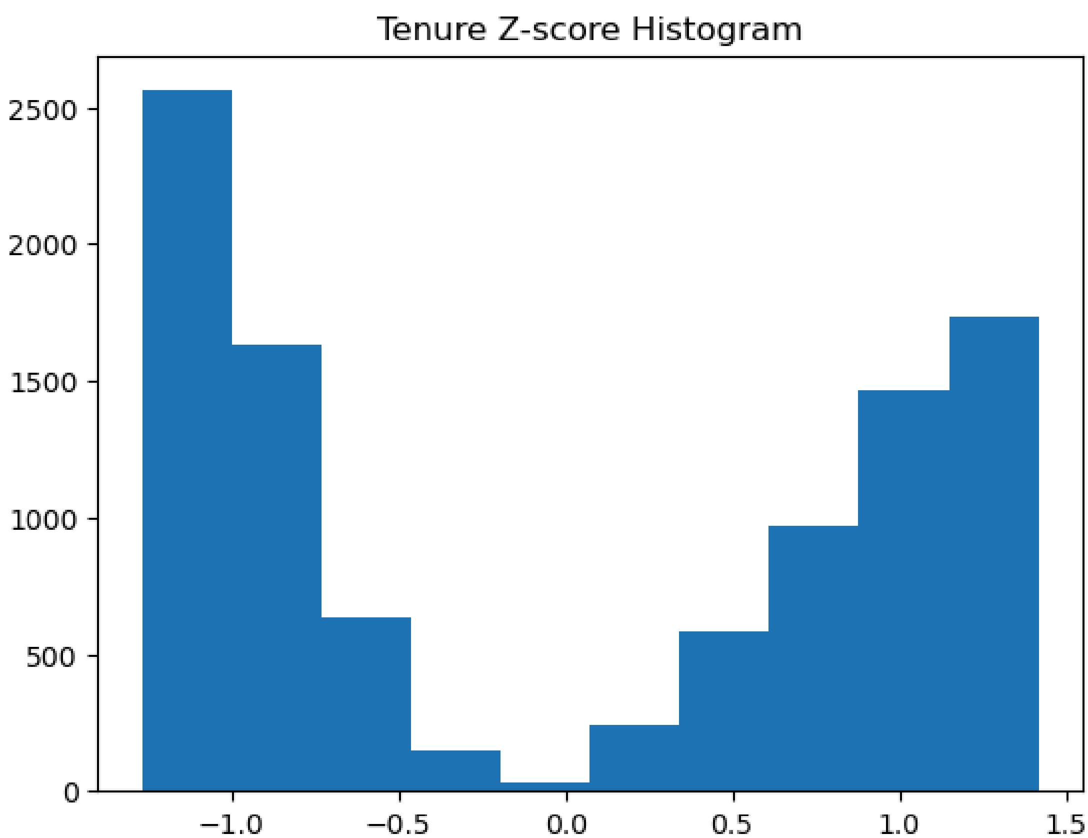


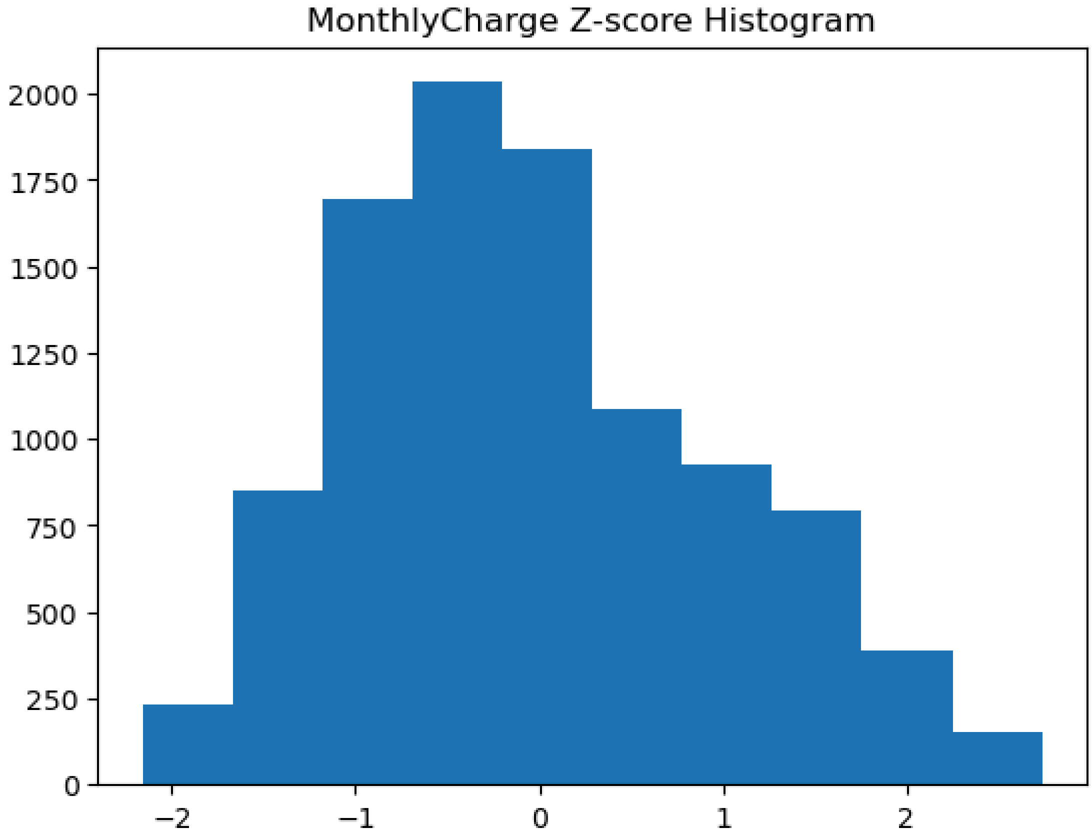
Email Z-score Histogram
3000
2500
2000
1500
1000
500
0
−2
−1
0
1
2

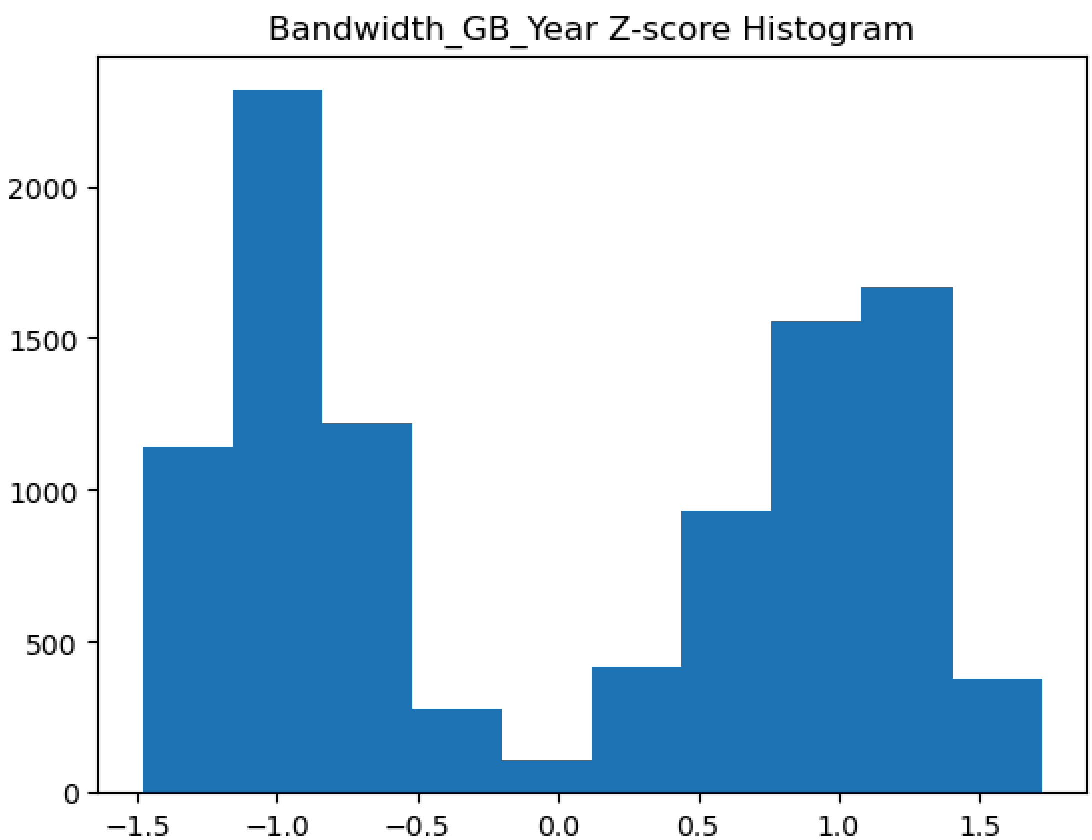



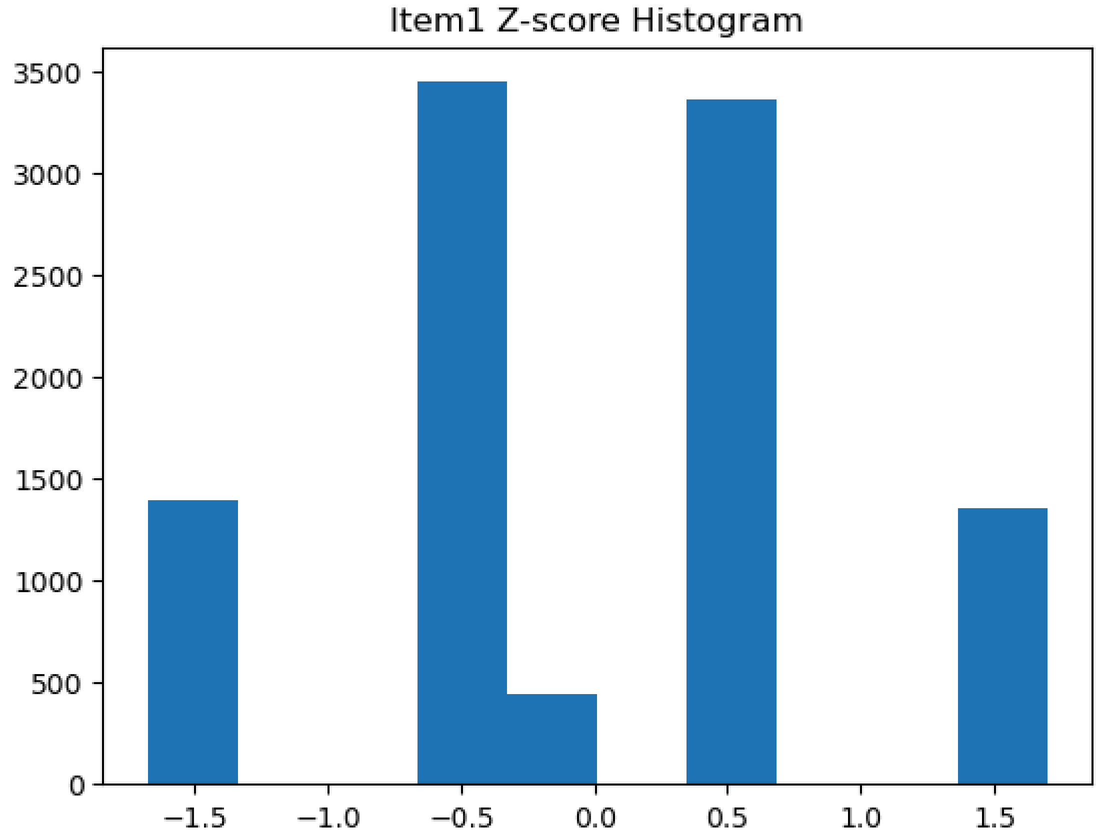
Contacts Z-score Histogram
4000
3000
2000
1000
0
−1.0
−0.5
0.0
0.5
1.0
1.5

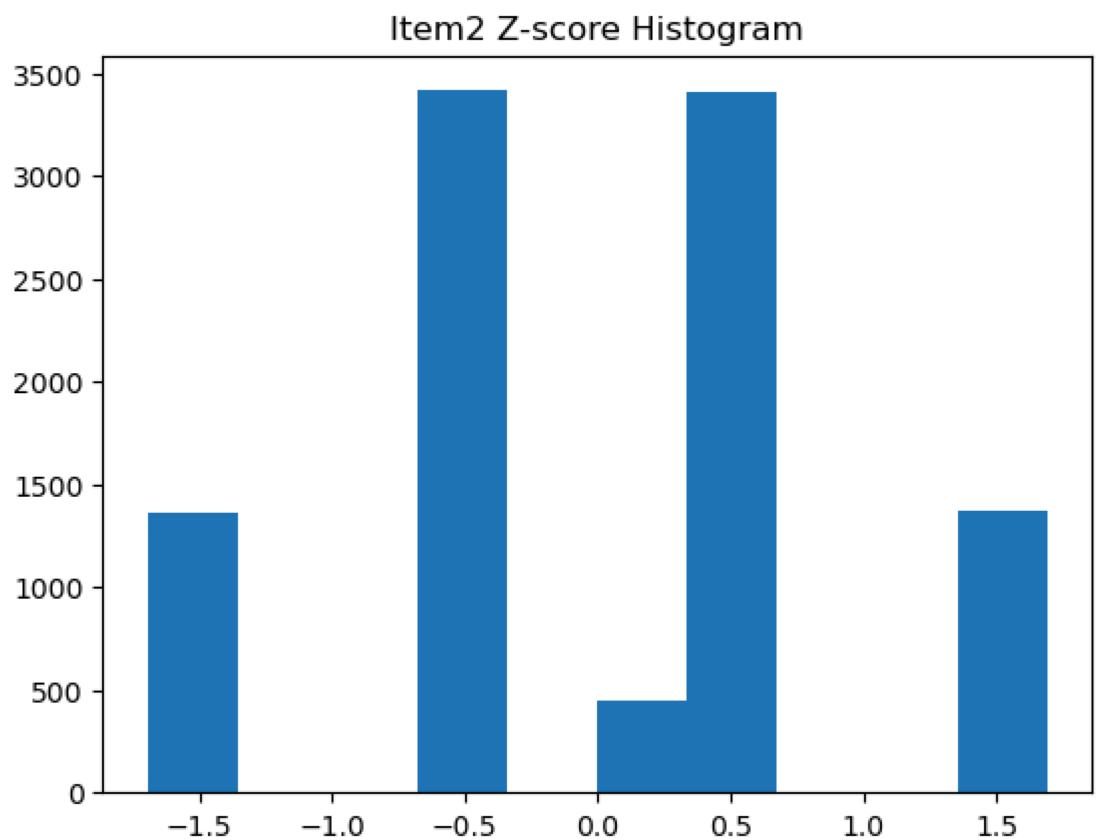

Yearly_equip_failure Z-score Histogram
7000
6000
5000
4000
3000
2000
1000
0
−0.5
0.0
0.5
1.0
1.5


Tenure Z-score Histogram
2500
2000
1500
1000
500
0
−1.0
−0.5
0.0
0.5
1.0
1.5

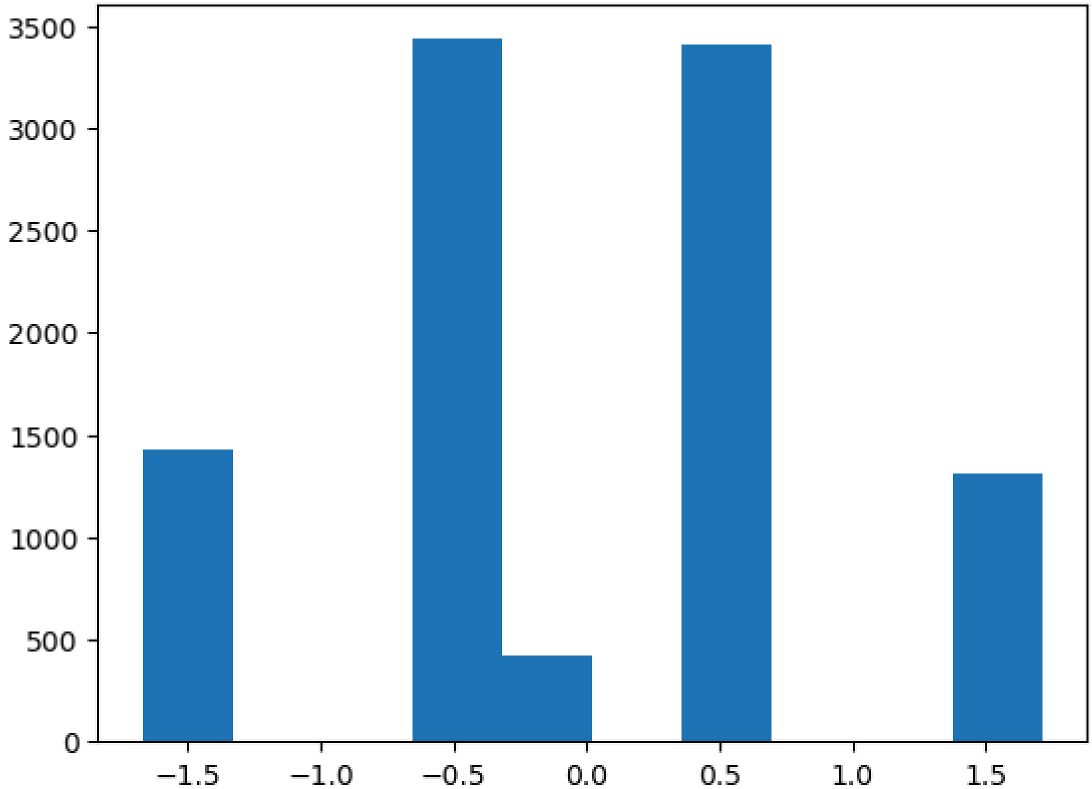

MonthlyCharge Z-score Histogram
0
250
500
750
1000
1250
1500
1750
2000
−2
−1
0
1
2


Bandwidth_GB_Year Z-score Histogram
0
500
1000
1500
2000
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

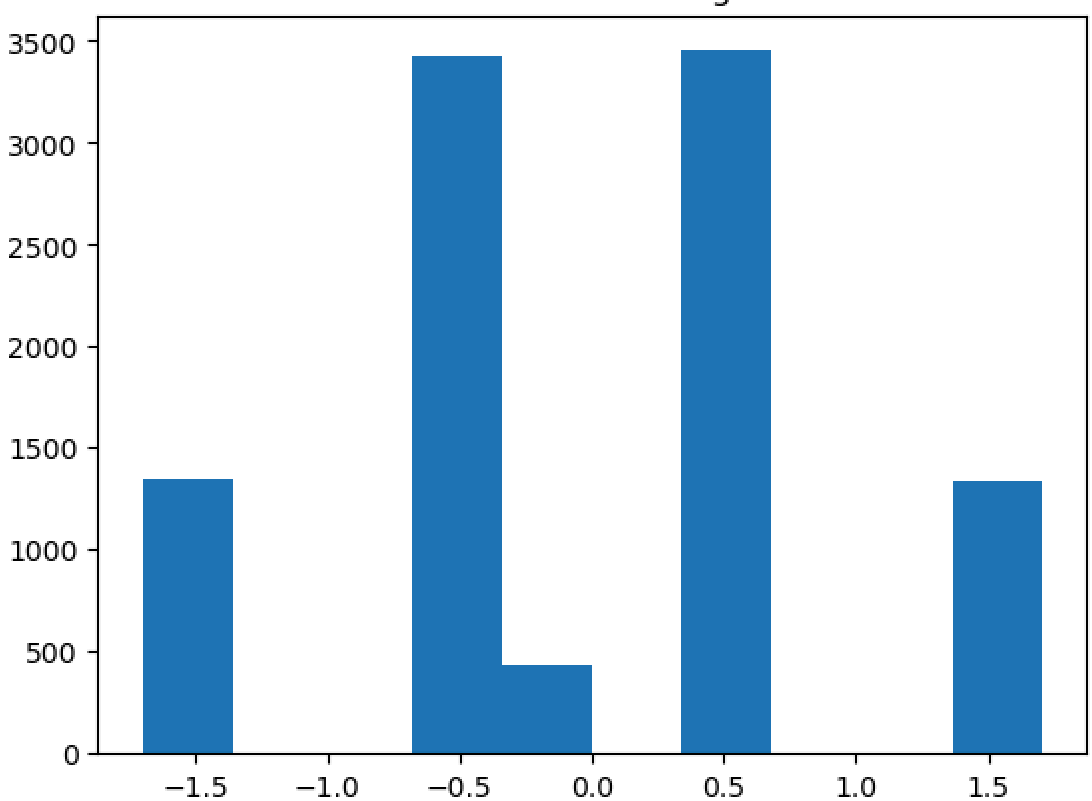

Item1 Z-score Histogram
3500
3000
2500
2000
1500
1000
500
0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

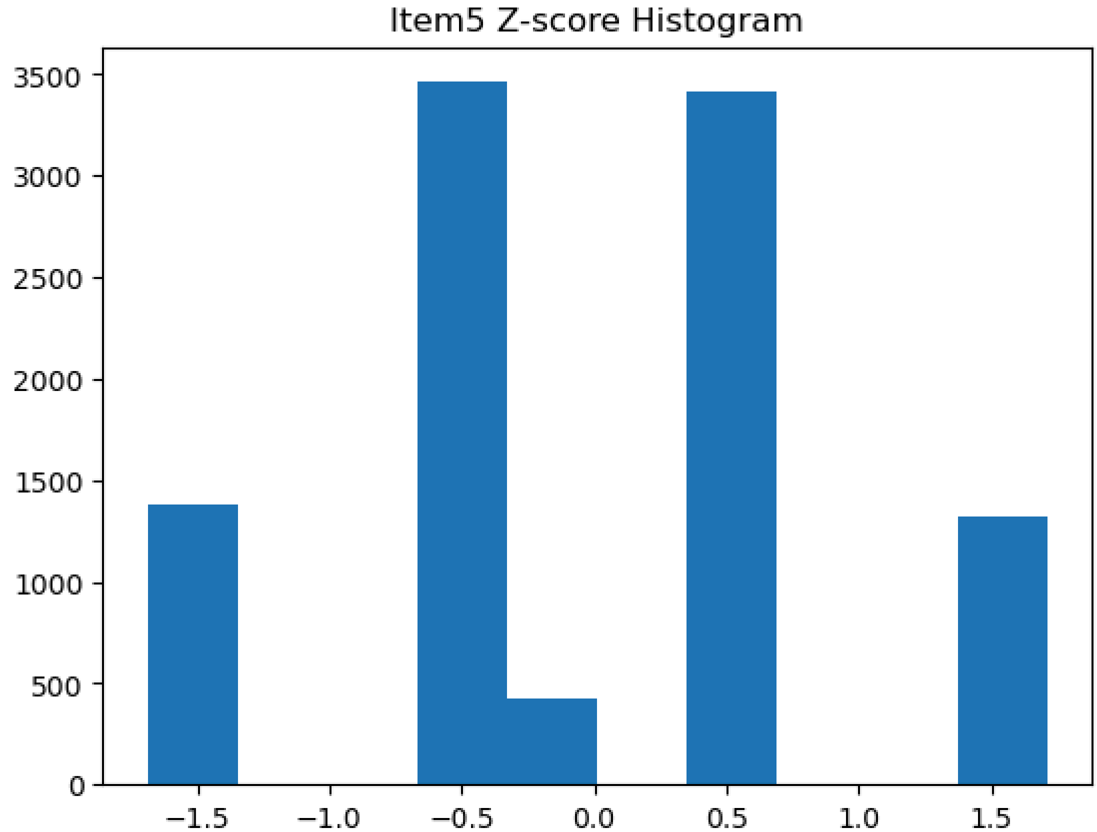

Item2 Z-score Histogram
3500
3000
2500
2000
1500
1000
500
0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

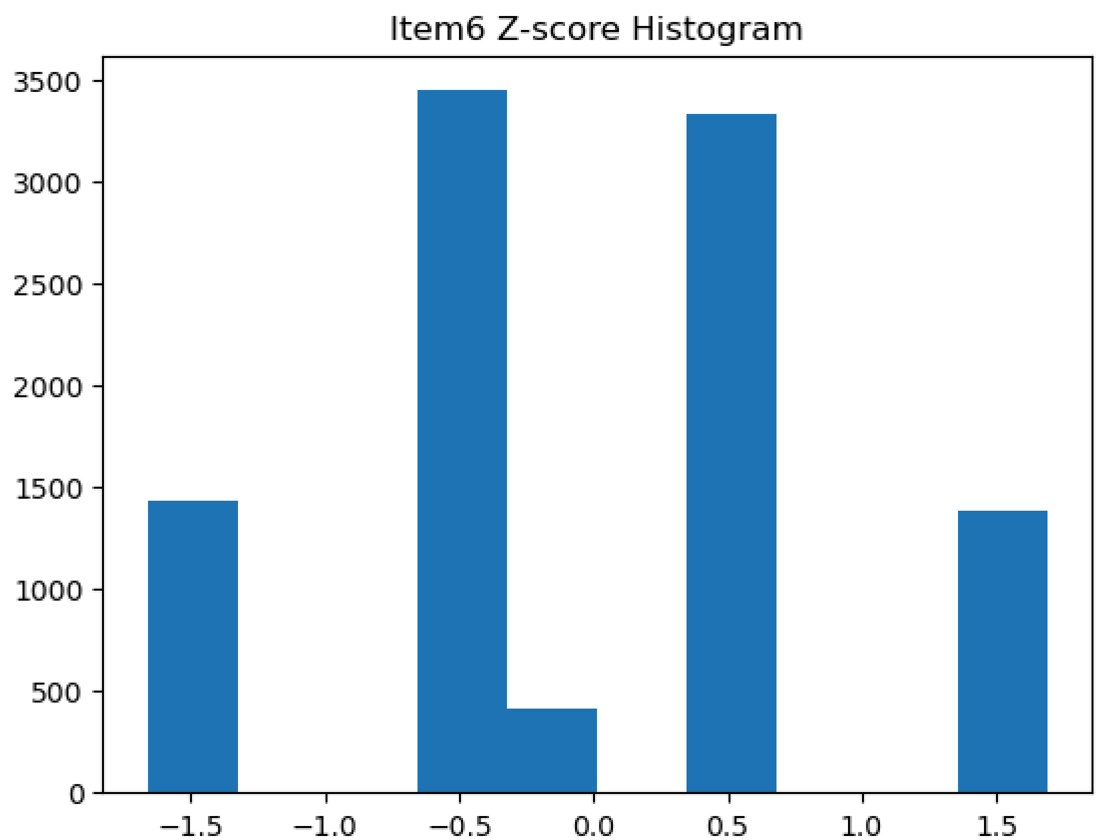
Item3 Z-score Histogram

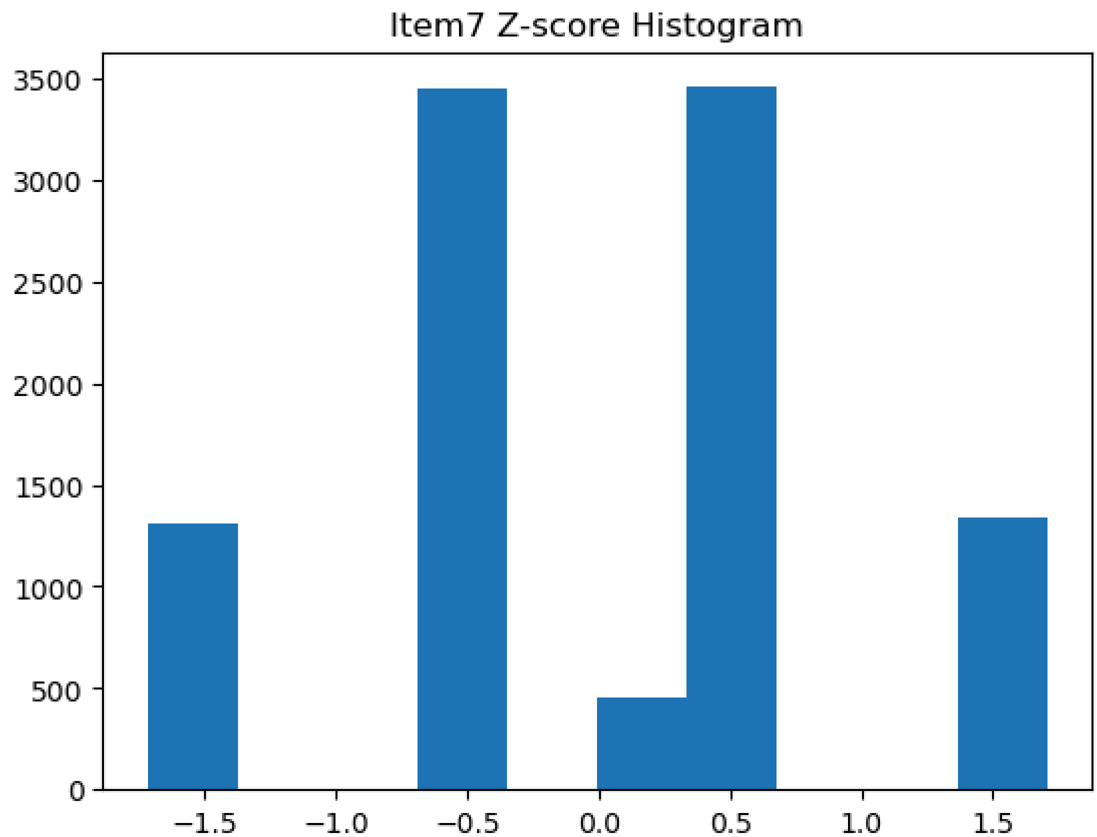


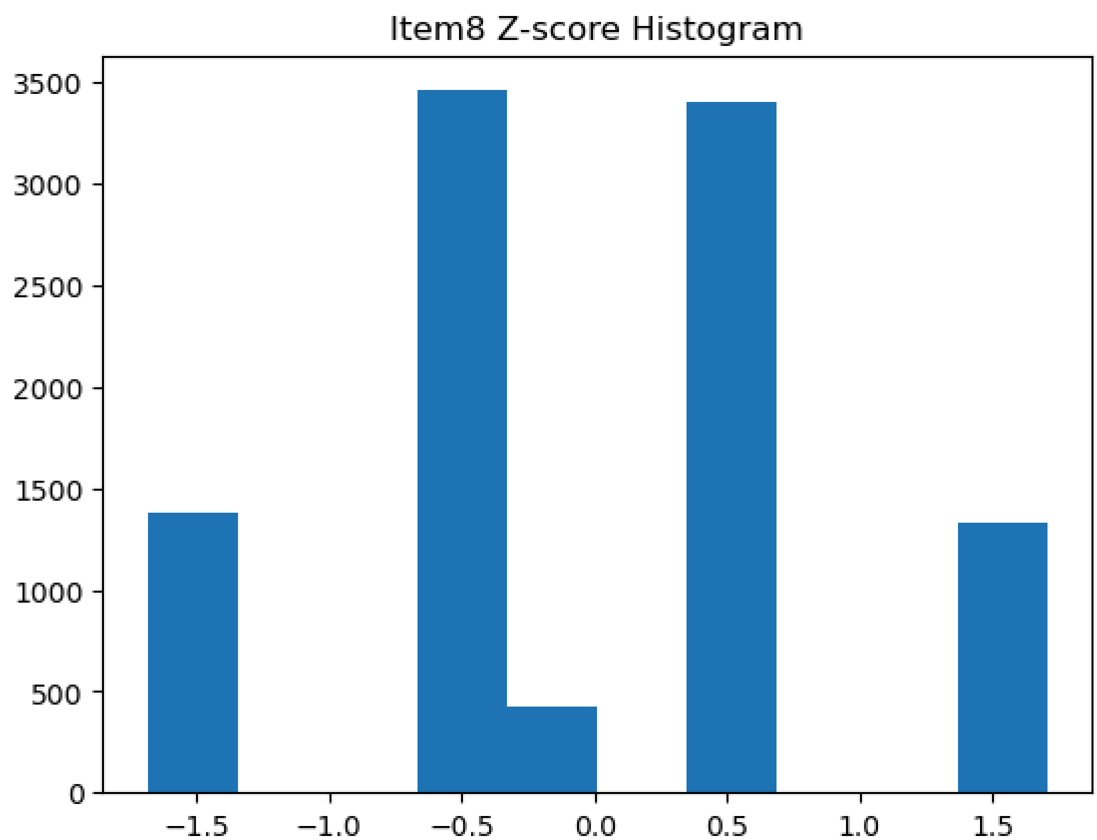

Item4 Z-score Histogram




Item5 Z-score Histogram
3500
3000
2500
2000
1500
1000
500
0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5


Item6 Z-score Histogram
3500
3000
2500
2000
1500
1000
500
0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5




In [18]:
```
# remove a remnant from the z scores
dfq = dfq.drop('zscore', axis=1)
```

## One Hot Encoding the Binary Qualitative Variables

We can one hot encode the rest of the variables that we will be using by using the map function.

In [19]:
```
dfq_ql = ['Churn','Techie','Port_modem','Tablet
for column in dfq_ql:
    dfq[column] = df[column].map({'Yes': 1,'No'
```

In [20]:
```
dfq
```

Out[20]:

| | Population | Children | Age | Income | Outage_sec_ |
|---|---|---|---|---|---|
| **0** | 38.0 | 0.0 | 68 | 28561.99 | |
| **1** | 10446.0 | 1.0 | 27 | 21704.77 | 1 |
| **2** | 3735.0 | 4.0 | 50 | 9609.57 | 1 |
| **3** | 13863.0 | 1.0 | 48 | 18925.23 | 1 |
| **4** | 11352.0 | 0.0 | 83 | 40074.19 | |
| **...** | ... | ... | ... | ... | |
| **9995** | 640.0 | 3.0 | 23 | 55723.74 | |
| **9996** | 2610.0 | 4.0 | 48 | 34129.34 | |
| **9997** | 406.0 | 1.0 | 48 | 45983.43 | |
| **9998** | 35575.0 | 1.0 | 39 | 16667.58 | 1 |
| **9999** | 12230.0 | 1.0 | 28 | 9020.92 | 1 |

10000 rows × 32 columns

In [21]:
```
dfq.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 32 columns):
 #   Column                Non-Null Count  Dtyp
e
---  ------                --------------  ----
-
 0   Population            10000 non-null  floa
t64
 1   Children              10000 non-null  floa
t64
 2   Age                   10000 non-null  int6
4
 3   Income                10000 non-null  floa
t64
 4   Outage_sec_perweek    10000 non-null  floa
t64
 5   Email                 10000 non-null  floa
t64
 6   Contacts              10000 non-null  floa
t64
 7   Yearly_equip_failure  10000 non-null  floa
t64
 8   Tenure                10000 non-null  floa
t64
 9   MonthlyCharge         10000 non-null  floa
t64
 10  Bandwidth_GB_Year     10000 non-null  floa
t64
 11  Item1                 10000 non-null  floa
t64
 12  Item2                 10000 non-null  floa
t64
 13  Item3                 10000 non-null  floa
t64
 14  Item4                 10000 non-null  floa
t64
 15  Item5                 10000 non-null  floa
t64
 16  Item6                 10000 non-null  floa
```

```
t64
 17   Item7                  10000 non-null  floa
t64
 18   Item8                  10000 non-null  floa
t64
 19   Churn                  10000 non-null  int6
4
 20   Techie                 10000 non-null  int6
4
 21   Port_modem             10000 non-null  int6
4
 22   Tablet                 10000 non-null  int6
4
 23   Phone                  10000 non-null  int6
4
 24   Multiple               10000 non-null  int6
4
 25   OnlineSecurity         10000 non-null  int6
4
 26   OnlineBackup           10000 non-null  int6
4
 27   DeviceProtection       10000 non-null  int6
4
 28   TechSupport            10000 non-null  int6
4
 29   StreamingTV            10000 non-null  int6
4
 30   StreamingMovies        10000 non-null  int6
4
 31   PaperlessBilling       10000 non-null  int6
4
dtypes: float64(18), int64(14)
memory usage: 2.4 MB
```

## Standardizing the Numeric Variables

In [22]:
```
dfq_nm = dfq.drop(['Churn','Techie','Port_modem
nm_c = dfq_nm.columns
```

```
In [23]: scaler = MinMaxScaler()
```

```
In [24]: dfq[nm_c] = scaler.fit_transform(dfq[nm_c])
```

```
In [25]: dfq
```

Out[25]:

| | Population | Children | Age | Income | Outage |
|---|---|---|---|---|---|
| **0** | 0.000933 | 0.000000 | 0.704225 | 0.294373 | |
| **1** | 0.270605 | 0.166667 | 0.126761 | 0.222826 | |
| **2** | 0.096722 | 0.666667 | 0.450704 | 0.096627 | |
| **3** | 0.359140 | 0.166667 | 0.422535 | 0.193825 | |
| **4** | 0.294080 | 0.000000 | 0.915493 | 0.414489 | |
| **...** | ... | ... | ... | ... | |
| **9995** | 0.016531 | 0.500000 | 0.070423 | 0.577774 | |
| **9996** | 0.067574 | 0.666667 | 0.422535 | 0.352462 | |
| **9997** | 0.010468 | 0.166667 | 0.422535 | 0.476145 | |
| **9998** | 0.921700 | 0.166667 | 0.295775 | 0.170269 | |
| **9999** | 0.316829 | 0.166667 | 0.140845 | 0.090485 | |

10000 rows × 32 columns

## C4:CLEANED DATA SET

```
In [26]: dfq.to_csv('prepared_data_task1.csv')
```

## D1:SPLITTING THE DATA

In [27]:
```
# Split into X and y
X = dfq.drop("Churn", axis=1)
y = dfq["Churn"]
```

In [28]:
```
# split into test and training data sets for bo
X_train, X_test, y_train, y_test = train_test_s
```

In [29]:
```
pd.DataFrame(X_test).to_csv('task1_test_data.cs
```

In [30]:
```
pd.DataFrame(X_train).to_csv('task1_train_data.
```

# D2 & D3:OUTPUT AND INTERMEDIATE CALCULATIONS AND CODE EXECUTION

Here we set the k number of neighbors and fit to the model. I output a results dataframe that has the true value of whether or not a customer churned and a predicted value. 0 is no and 1 is yes.

In [31]:
```
# set up  model with number of neighbors
knn = KNeighborsClassifier(n_neighbors=5)
```

In [32]:
```
# fir to the training data
knn.fit(X_train, y_train)
```

Out[32]:
```
KNeighborsClassifier()
```

In [33]:
```
# predicts the actual class for churn
y_pred = knn.predict(X_test)
```

```
/Users/rjcalabio/opt/anaconda3/lib/python3.9/site-packages/sklearn/neighbors/_classification.py:228: FutureWarning: Unlike other reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will change: the default value of `keepdims` will become False, the `axis` over which the statistic is taken will be eliminated, and the value None will no longer be accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```

In [34]:

```
# used for class probabilties
y_prob = knn.predict_proba(X_test)[:, 1]
```

In [35]:

```
#prints the results
churn_results_df = pd.DataFrame({
    'Churn Label': y_test,
    'Predicted Label': y_pred,
    'Probability Score': y_prob
})
```

In [36]:

```
churn_results_df
```

Out[36]:

| | Churn Label | Predicted Label | Probability Score |
|---|---|---|---|
| **8158** | 0 | 1 | 0.6 |
| **3484** | 0 | 0 | 0.4 |
| **5443** | 0 | 0 | 0.0 |
| **7278** | 0 | 0 | 0.0 |
| **8278** | 0 | 0 | 0.0 |
| **...** | ... | ... | ... |
| **3759** | 0 | 0 | 0.0 |
| **7659** | 0 | 0 | 0.0 |
| **8081** | 0 | 0 | 0.0 |
| **3072** | 0 | 0 | 0.0 |
| **3530** | 0 | 0 | 0.0 |

2000 rows × 3 columns

# E1:ACCURACY AND AUC

The accuracy and the AUC score for the model is calculated here

In [37]:

```
# calculates false positive rates, true positiv
fpr, tpr, thresholds = roc_curve(y_test, y_prob
roc_auc = auc(fpr, tpr)
```

In [38]:

```
# print the results
print(f"Accuracy: {knn.score(X_test, y_test)}")
print(f"AUC: {roc_auc}")
```

```
Accuracy: 0.8115
AUC: 0.8367417533050956
```

```
/Users/rjcalabio/opt/anaconda3/lib/python3.9/si
te-packages/sklearn/neighbors/_classification.p
y:228: FutureWarning: Unlike other reduction fu
nctions (e.g. `skew`, `kurtosis`), the default 
behavior of `mode` typically preserves the axis
it acts along. In SciPy 1.11.0, this behavior w
ill change: the default value of `keepdims` wil
l become False, the `axis` over which the stati
stic is taken will be eliminated, and the value
None will no longer be accepted. Set `keepdims`
to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=
1)
```

In [39]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   s
upport

           0       0.84      0.91      0.88
1470
           1       0.69      0.53      0.60
530

    accuracy                           0.81
2000
   macro avg       0.77      0.72      0.74
2000
weighted avg       0.80      0.81      0.80
2000
```

# E2:RESULTS AND IMPLICATIONS

The accuracy of the model that was developed for this analysis is 0.8115. Accuracy in relation to this model represents the amount of points that our model correctly categorized based on the test data.

The AUC of the model that was develop for this analysis is 0.8367. AUC stands for area under the curve, and this represents the accuracy of the classification of the model. For AUC we want this to be higher as a higher AUC means more accuracy in the model.

The results of this tell us that we can within an 81% accuracy range predict whether or not a customer will churn based on numeric and binary qualitative data provided by the churn dataset. The implication for this is that we can implement a system like this into the business so that we can make decisions based upon the potential for a customer to churn. For instance, if a customer is predicted to churn by the model, we can create an action that will attempt to offer something that will incentivize the customer to stay.

This has a lot of implications for the business side. We can use customer data to create a table of customers that are highly likely to churn based upon the results of the KNN model. Then we can develop a Tableu or Power BI report to present the data in an interactive way to the business stakeholders. From there, the stakeholders can use this data to make a data driven decision on whether or not to implement certain policies or business decisions that could positively improve churn.

# E3:LIMITATION

A limitation of K Nearest Neighbors is that as the features increase the accuracy of the model may suffer. This is because there are more points muddying up the areas of the graph when the model is determining the closest points, and the more there are of classifications that are not the actual grouping, the more likely the model is to choose a classification that is not the correct grouping.

# E4:COURSE OF ACTION

The next course of action would be to look at what features help in predicting accurately whether or not a customer has churned. These can then be isolated as factors that need to be focused on. It would also be a good idea to reiterate and redevelop the model to improve the accruacy, and then utilize the model to forecast whether or not a customer will be predicted to churn. If that person is a high-value customer, it may be worth it to offer them a deal in order to retain their business. It may also be a good course of action to test multiple models out to see if we can find another model that is more accurate in predicting customer churn.

We can now take this data and implement the model into something like a Power BI report that can be viewed by stakeholders and allow them to make better informed decisions. These decisions can directly lead to less churn, and we can track the results of the decisions in our reports and decide whether or not this has made a large business impact.

# G:SOURCES FOR THIRD-PARTY CODE

No third party sources of code used

# H:SOURCES

Bhandari, Aniruddha. “Guide to AUC ROC Curve in Machine Learning: What Is Specificity?” Analytics Vidhya, 27 Aug. 2024, www.analyticsvidhya.com/blog/2020/06/auc-roc-curve-machine-learning/.

Christopher, Antony. “K-Nearest Neighbor.” Medium, The Startup, 3 Feb. 2021, medium.com/swlh/k-nearest-neighbor-ca2593d7a3c4.

"Custom-Data-Mining-i." DataCamp, app.datacamp.com/learn/custom-tracks/custom-data-mining-i. Accessed 30 Aug. 2024.