D206_Churn_Data_Cleaning

August 17, 2021

[1]:
```python
# Standard imports
import pandas as pd
import numpy as np
import scipy.stats as stats
import matplotlib.pyplot as pp
import seaborn as sns
from sklearn.decomposition import PCA
```

[2]:
```python
# Reading the file
df = pd.read_csv('C:/Users/sigsp/OneDrive/Desktop/D206 Data Cleaning/
↪churn_raw_data.csv')
```

[3]:
```python
# Dropping duplicates & checking for changes
print(df.shape)
df.drop_duplicates(inplace=True)
print(df.shape)
```

```
(10000, 52)
(10000, 52)
```

[4]:
```python
# Reviewing columns, nulls, and datatypes
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10000 entries, 0 to 9999
Data columns (total 52 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Unnamed: 0            10000 non-null  int64
 1   CaseOrder             10000 non-null  int64
 2   Customer_id           10000 non-null  object
 3   Interaction           10000 non-null  object
 4   City                  10000 non-null  object
 5   State                 10000 non-null  object
 6   County                10000 non-null  object
 7   Zip                   10000 non-null  int64
 8   Lat                   10000 non-null  float64
 9   Lng                   10000 non-null  float64
 10  Population            10000 non-null  int64
```

```
 11  Area                   10000 non-null  object
 12  Timezone               10000 non-null  object
 13  Job                    10000 non-null  object
 14  Children               7505 non-null   float64
 15  Age                    7525 non-null   float64
 16  Education              10000 non-null  object
 17  Employment             10000 non-null  object
 18  Income                 7510 non-null   float64
 19  Marital                10000 non-null  object
 20  Gender                 10000 non-null  object
 21  Churn                  10000 non-null  object
 22  Outage_sec_perweek     10000 non-null  float64
 23  Email                  10000 non-null  int64
 24  Contacts               10000 non-null  int64
 25  Yearly_equip_failure   10000 non-null  int64
 26  Techie                 7523 non-null   object
 27  Contract               10000 non-null  object
 28  Port_modem             10000 non-null  object
 29  Tablet                 10000 non-null  object
 30  InternetService        10000 non-null  object
 31  Phone                  8974 non-null   object
 32  Multiple               10000 non-null  object
 33  OnlineSecurity         10000 non-null  object
 34  OnlineBackup           10000 non-null  object
 35  DeviceProtection       10000 non-null  object
 36  TechSupport            9009 non-null   object
 37  StreamingTV            10000 non-null  object
 38  StreamingMovies        10000 non-null  object
 39  PaperlessBilling       10000 non-null  object
 40  PaymentMethod          10000 non-null  object
 41  Tenure                 9069 non-null   float64
 42  MonthlyCharge          10000 non-null  float64
 43  Bandwidth_GB_Year      8979 non-null   float64
 44  item1                  10000 non-null  int64
 45  item2                  10000 non-null  int64
 46  item3                  10000 non-null  int64
 47  item4                  10000 non-null  int64
 48  item5                  10000 non-null  int64
 49  item6                  10000 non-null  int64
 50  item7                  10000 non-null  int64
 51  item8                  10000 non-null  int64
dtypes: float64(9), int64(15), object(28)
memory usage: 4.0+ MB
None
```

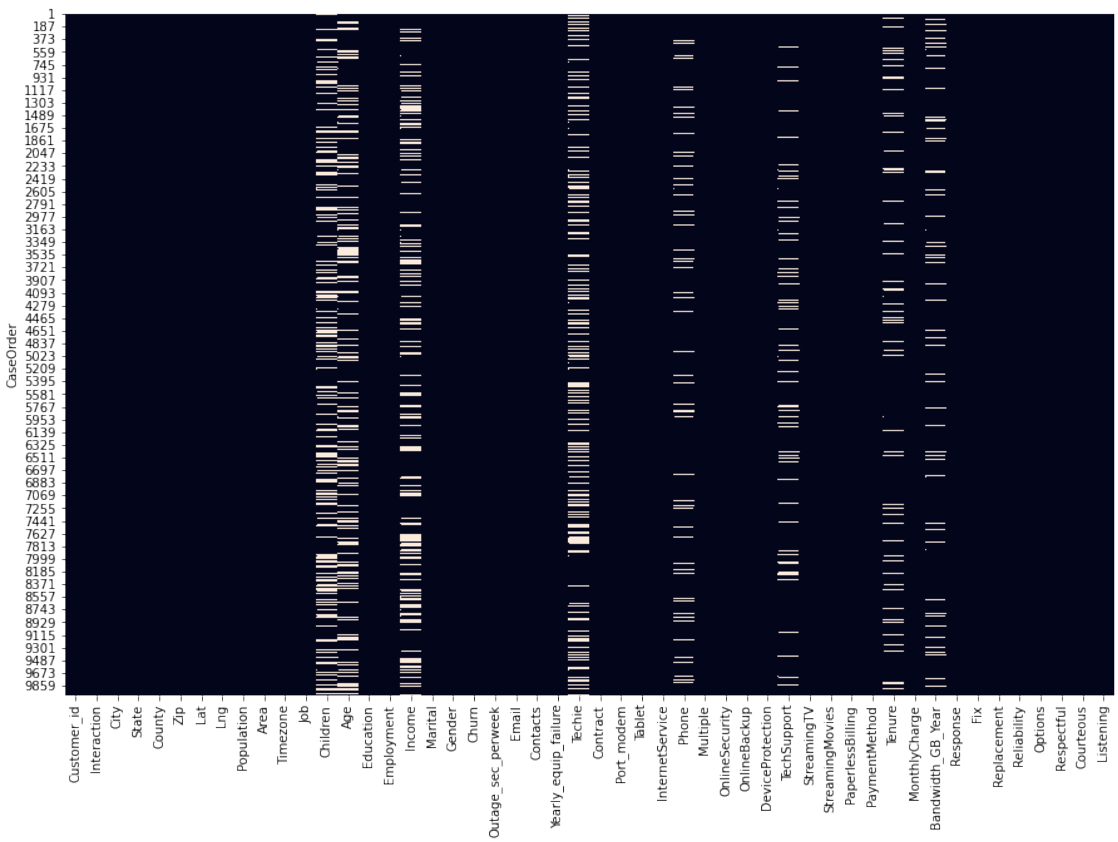
[5]:
```
# Prepping columns: Dropping Unnamed: 0 column, setting index, renaming item1
↪to item8 columns
```

```
df.drop(columns='Unnamed: 0', inplace=True)

df.set_index('CaseOrder', inplace=True)

survey_dict = {'item1': 'Response',
               'item2': 'Fix',
               'item3': 'Replacement',
               'item4': 'Reliability',
               'item5': 'Options',
               'item6': 'Respectful',
               'item7': 'Courteous',
               'item8': 'Listening'}
df.rename(columns=survey_dict, inplace=True)
```

[6]:

```
# Heatmap to visualize location of null values.
fig, ax = pp.subplots(figsize=(15,10))
sns.heatmap(df.isnull(), xticklabels=1, cbar=False, ax=ax)
```

[6]: <AxesSubplot:ylabel='CaseOrder'>



[7]:
```
# Dropping rows with > 75% missing data
nullThresh = df.shape[1]*0.75
df.dropna(thresh=nullThresh, inplace=True)
print(df.shape)
```

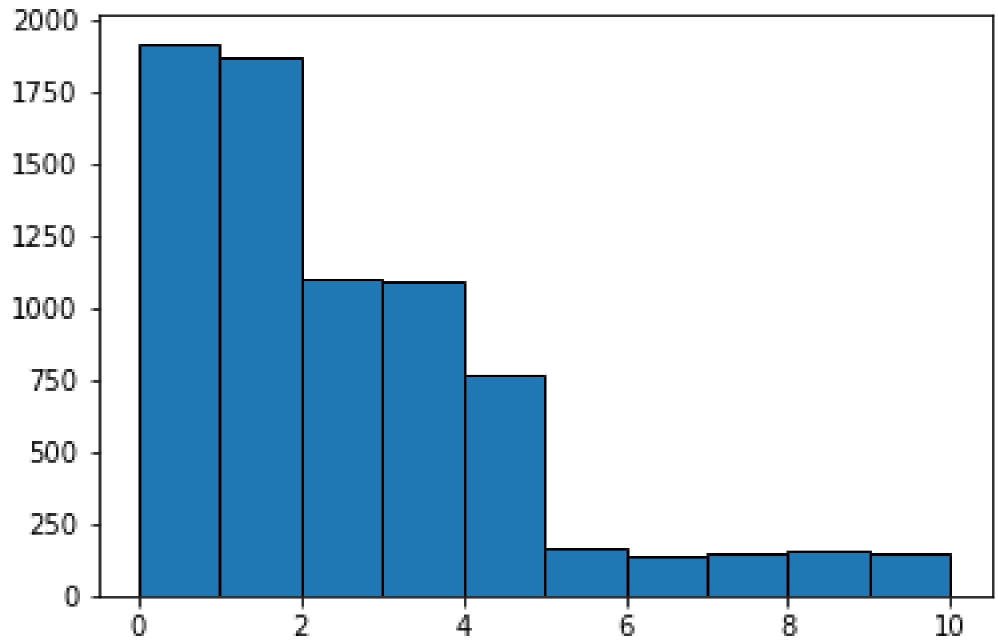
```
(10000, 50)
```

[8]:
```
# The following columns have null values that must be addressed:
# Children, Age, Income, Techie, Phone, TechSupport, Tenure, Bandwith_GB_Year
```

[9]:
```
# Analyzing Children variable
maxChild = np.nanmax(df.Children)
minChild = np.nanmin(df.Children)
avgChild = np.nanmean(df.Children)
medChild = np.nanmedian(df.Children)
print("The maximum value in the Children column is ", maxChild)
print("The minimum value in the Children column is ", minChild)
print("The arithmetic mean value in the Children column is ", avgChild)
print("The median value in the Children column is ", medChild)
pp.hist(df.Children, edgecolor='black')
```

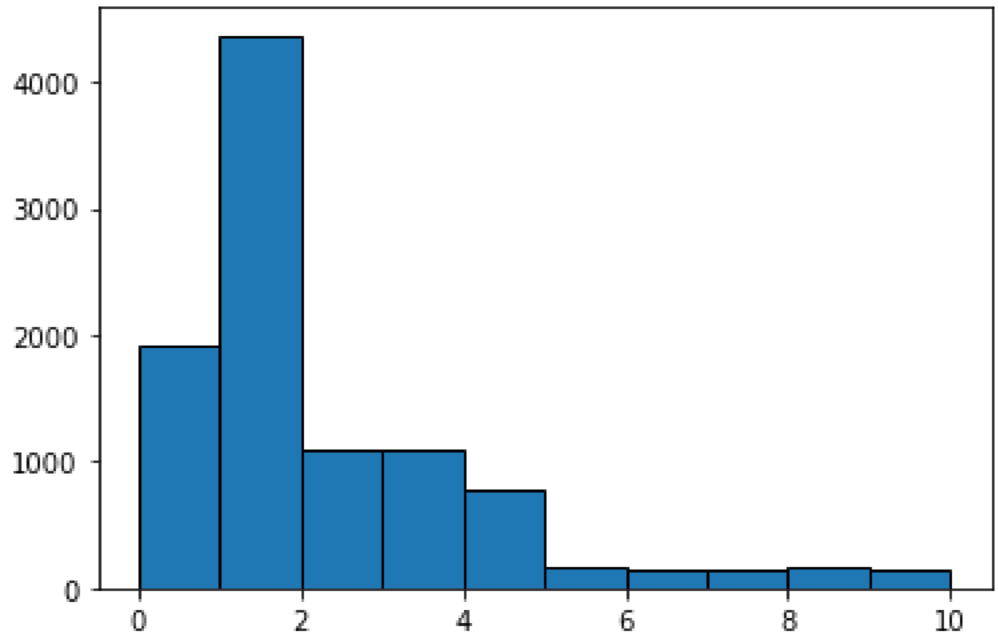
```
The maximum value in the Children column is  10.0
The minimum value in the Children column is  0.0
The arithmetic mean value in the Children column is  2.095936042638241
The median value in the Children column is  1.0
```

[9]:
```
(array([1919., 1874., 1100., 1096.,  769.,  161.,  135.,  149.,  158.,
         144.]),
 array([ 0.,  1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10.]),
 <BarContainer object of 10 artists>)
```



[10]:
```
# Imputing using the median, converting to int64, & confirming no nulls remain
df.Children.fillna(medChild, inplace=True)
df['Children'] = df['Children'].astype('int64')
print(df.Children.dtypes)
pp.hist(df.Children, edgecolor='black')
print(df.Children.isnull().any())
```

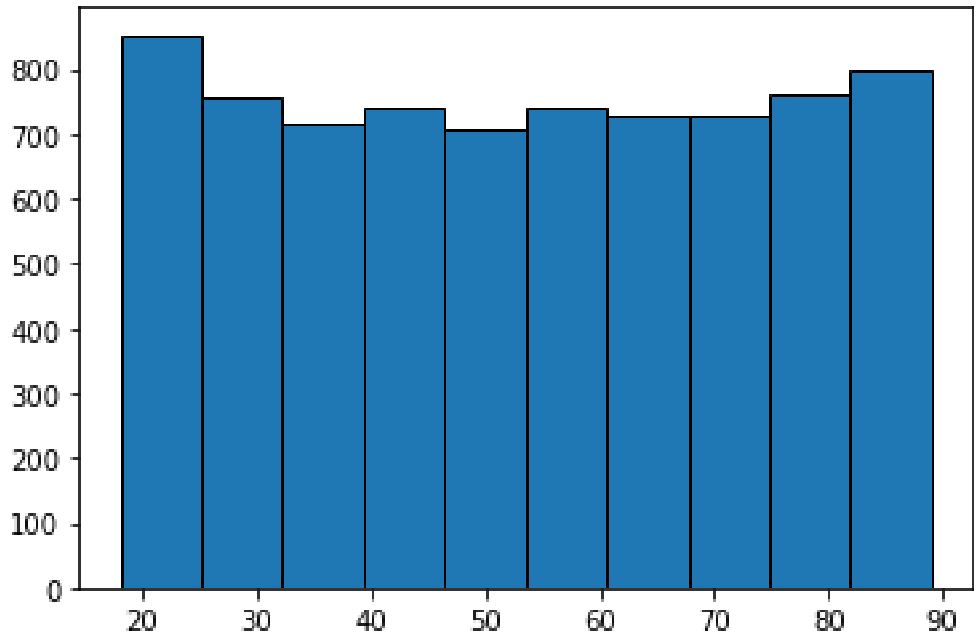
```
int64
False
```



[11]:
```
# Analyzing the Age variable
maxAge = np.nanmax(df.Age)
minAge = np.nanmin(df.Age)
avgAge = np.nanmean(df.Age)
medAge = np.nanmedian(df.Age)
print("The maximum value in the Age column is ", maxAge)
print("The minimum value in the Age column is ", minAge)
print("The arithmetic mean value in the Age column is ", avgAge)
print("The median value in the Age column is ", medAge)
pp.hist(df.Age, edgecolor='black')
```

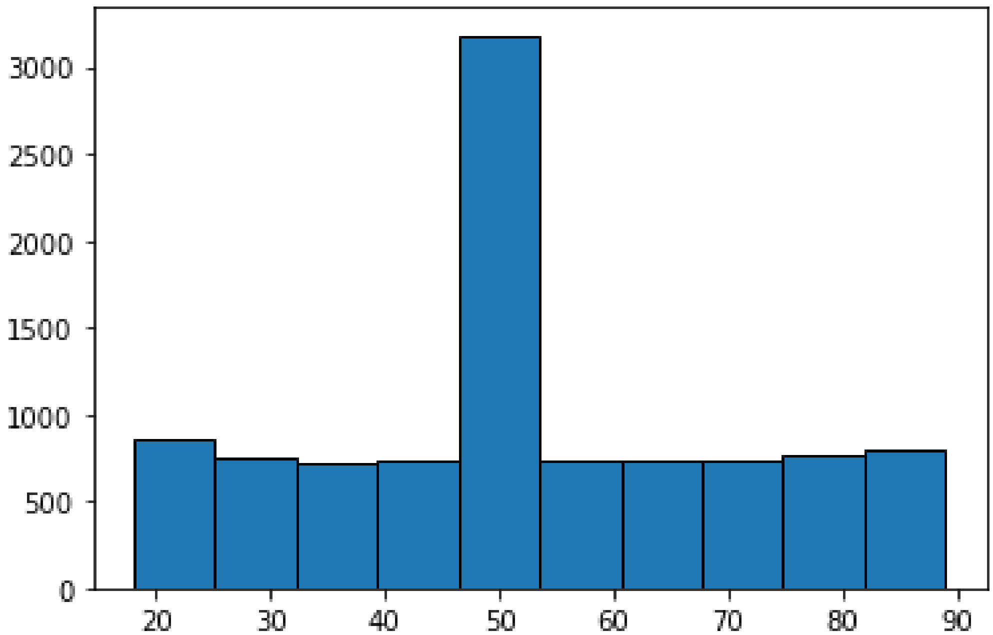
```
The maximum value in the Age column is  89.0
The minimum value in the Age column is  18.0
The arithmetic mean value in the Age column is  53.27574750830565
The median value in the Age column is  53.0
```

[11]:
```
(array([853., 756., 714., 739., 708., 739., 728., 728., 761., 799.]),
 array([18. , 25.1, 32.2, 39.3, 46.4, 53.5, 60.6, 67.7, 74.8, 81.9, 89. ]),
 <BarContainer object of 10 artists>)
```



[12]:
```
# Imputing using the median, converting to int64, & confirming no nulls remain.
df.Age.fillna(medAge, inplace=True)
df['Age'] = df['Age'].astype('int64')
print(df.Age.dtypes)
pp.hist(df.Age, edgecolor='black')
print(df.Age.isnull().any())
```

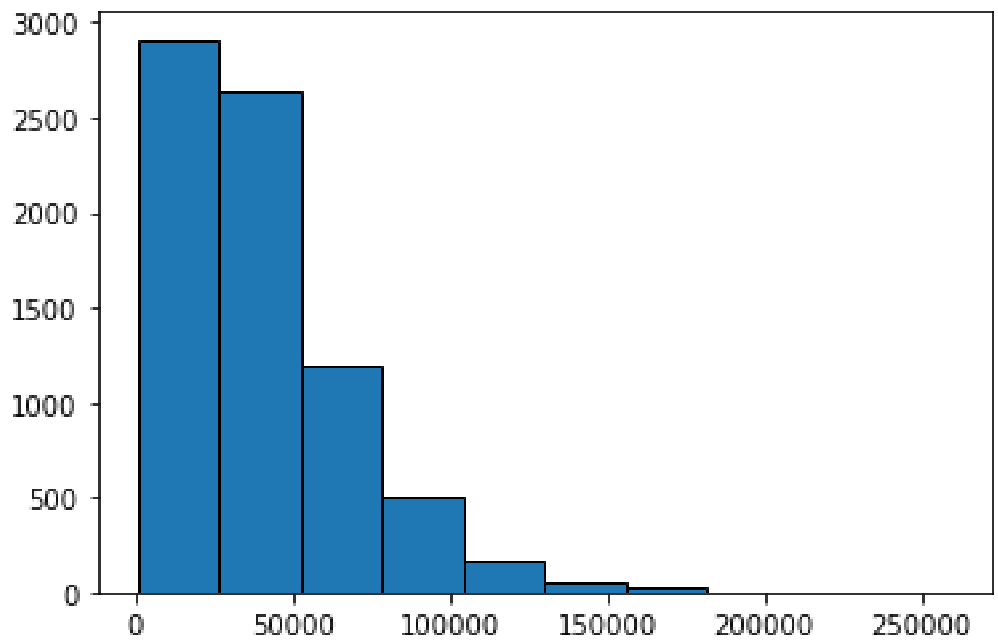
```
int64
False
```



[13]:
```
# Analyzing the Income variable.
maxInc = np.nanmax(df.Income)
minInc = np.nanmin(df.Income)
avgInc = np.nanmean(df.Income)
medInc = np.nanmedian(df.Income)
print("The maximum value in the Income column is ", maxInc)
print("The minimum value in the Income column is ", minInc)
print("The arithmetic mean value in the Income column is ", avgInc)
print("The median value in the Income column is ", medInc)
pp.hist(df.Income, edgecolor='black')
```

```
The maximum value in the Income column is  258900.7
The minimum value in the Income column is  740.66
The arithmetic mean value in the Income column is  39936.76222636485
The median value in the Income column is  33186.785
```

[13]:
```
(array([2.912e+03, 2.642e+03, 1.193e+03, 5.080e+02, 1.670e+02, 6.000e+01,
        2.100e+01, 4.000e+00, 1.000e+00, 2.000e+00]),
 array([   740.66 ,  26556.664,  52372.668,  78188.672, 104004.676,
        129820.68 , 155636.684, 181452.688, 207268.692, 233084.696,
        258900.7  ]),
 <BarContainer object of 10 artists>)
```



[14]:
```
# Histogram is heavily skewed right with visible outliers. Further analysis is␣
↪required.
```

[15]:
```
# Identifying & analyzing outliers
df['Income_z'] = stats.zscore(df['Income'], nan_policy='omit')
income_outliers = df.query('Income_z > 3 | Income_z < -3')
income_outliers_sort = income_outliers[['Customer_id', 'Income', 'Income_z']].
↪sort_values(['Income_z'], ascending=False)
print("The percentage of outliers in the data set is", len(income_outliers)/df.
↪shape[0]*100)
```

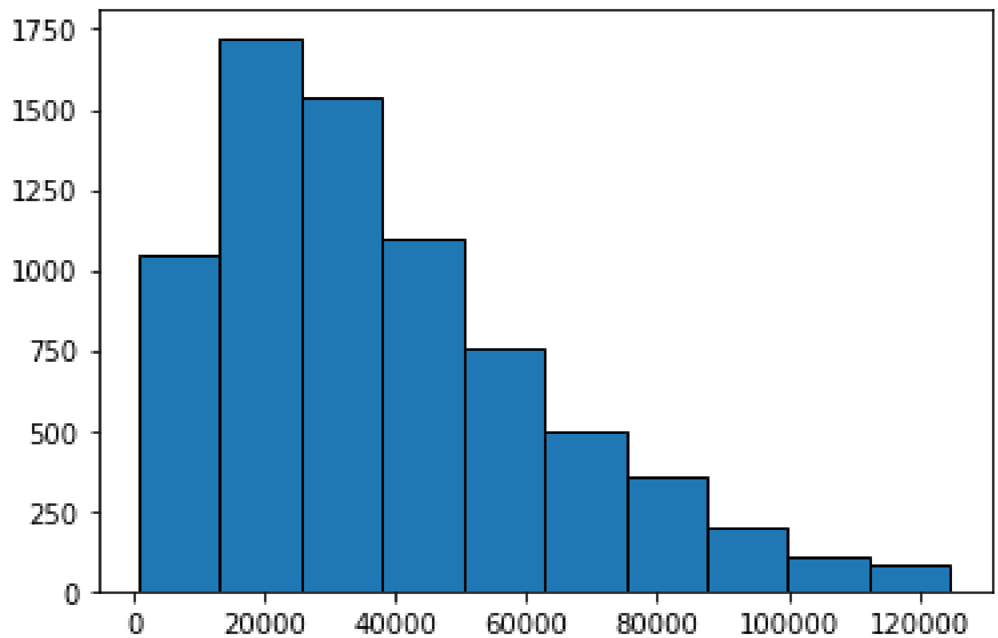
```
The percentage of outliers in the data set is 1.0999999999999999
```

[16]:
```
# Removing outliers and recalculating values.
filtered_income = df.drop(df[abs(df['Income_z']) > 3].index)
filteredMaxInc = np.nanmax(filtered_income['Income'])
filteredMinInc = np.nanmin(filtered_income['Income'])
filteredAvgInc = np.nanmean(filtered_income['Income'])
filteredMedInc = np.nanmedian(filtered_income['Income'])
print("The filtered maximum value in the Income column is ", filteredMaxInc)
print("The filtered minimum value in the Income column is ", filteredMinInc)
print("The filtered arithmetic mean value in the Income column is ",␣
↪filteredAvgInc)
print("The filtered median value in the Income column is ", filteredMedInc)
pp.hist(filtered_income['Income'], edgecolor='black')
```

```
The filtered maximum value in the Income column is  124735.8
```

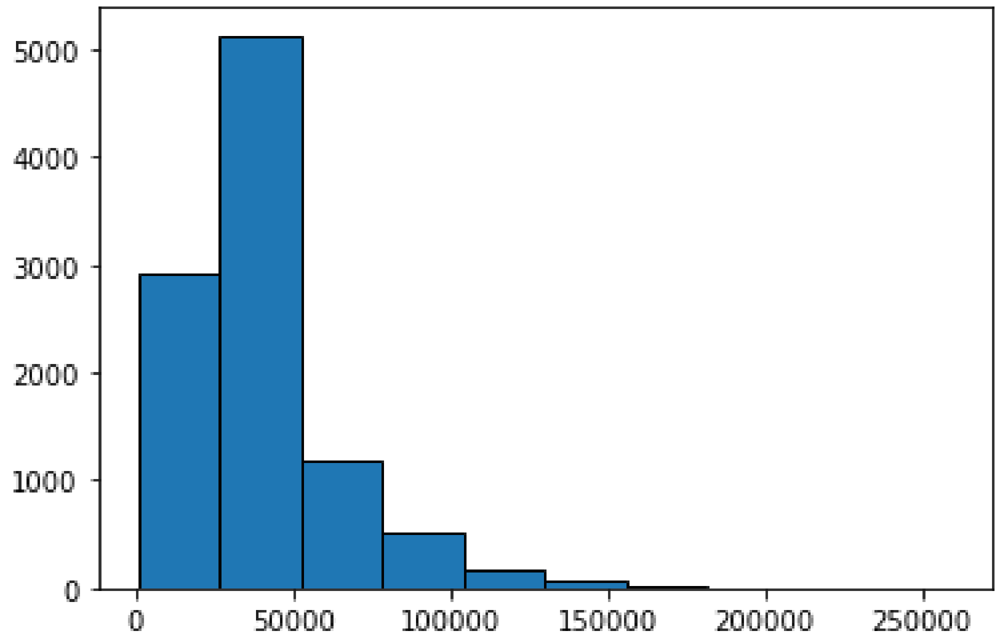
```
The filtered minimum value in the Income column is  740.66
The filtered arithmetic mean value in the Income column is  38344.443281081076
The filtered median value in the Income column is  32769.490000000005
```

[16]:
```
(array([1049., 1723., 1536., 1097.,  754.,  497.,  356.,  199.,  105.,
          84.]),
 array([   740.66 ,  13140.174,  25539.688,  37939.202,  50338.716,
         62738.23 ,  75137.744,  87537.258,  99936.772, 112336.286,
        124735.8  ]),
 <BarContainer object of 10 artists>)
```



[17]:
```
# Imputing missing values, dropping Income_z column, and confirming changes␣
 ↪have been made.
df.Income.fillna(filteredMedInc, inplace=True)
df.drop(columns='Income_z', inplace=True)

pp.hist(df.Income, edgecolor='black')
print(df.Age.isnull().any())
print(df.shape)
```

```
False
(10000, 50)
```



[18]:
```
# Analyzing Techie column.
print(df.Techie.value_counts())
print(df.Techie.isnull().sum())
```

```
No      6266
Yes     1257
Name: Techie, dtype: int64
2477
```

[19]:
```
# Dropping column due to large amount of missing data (nearly 25%) & values are␣
↪self-reported (therefore biased)
# and confirming it has been dropped
print(df.shape)
df.drop(columns='Techie', inplace=True)
print(df.shape)
```

```
(10000, 50)
(10000, 49)
```

[20]:
```
# Analyzing the Phone column using the Multiple column.
print("The Phone column contains", df.Phone.isnull().sum(),"missing values.")
print("The Multiple column contains", df.Multiple.isnull().sum(), "missing␣
↪values.")
```

```
The Phone column contains 1026 missing values.
The Multiple column contains 0 missing values.
```

[21]:
```
# Those with Multiple lines have phone service.
# Finding Customer_ids with multiple lines but phone is null & imputing with␣
↪'Yes'
phoneNull = df.loc[df['Phone'].isnull()]
phoneNullMultYes = phoneNull.loc[phoneNull['Multiple'] == 'Yes']
filteredPhoneIds = phoneNullMultYes['Customer_id']
df.loc[df.Customer_id.isin(filteredPhoneIds), 'Phone'] = 'Yes'
print("There are", df.Phone.isnull().sum(), "remaining missing values.")
```

```
There are 576 remaining missing values.
```

[22]:
```
# Checking mode
print(df.Phone.mode())
```

```
0    Yes
dtype: object
```

[23]:
```
# Imputing remaining missing values with mode & confirming no nulls remain.
df.Phone.fillna('Yes', inplace=True)
print(df.Phone.value_counts())
print(df.Phone.isnull().any())
```

```
Yes    9154
No      846
Name: Phone, dtype: int64
False
```

[24]:
```
# Analyzing TechSupport column.
print(df.TechSupport.value_counts())
print(df.TechSupport.isnull().sum())
```

```
No     5635
Yes    3374
Name: TechSupport, dtype: int64
991
```

[25]:
```
# Checking mode
print(df.TechSupport.mode())
```

```
0    No
dtype: object
```

[26]:
```
# Imputing with mode & confirming no nulls remain.
df.TechSupport.fillna('No', inplace=True)
print(df.TechSupport.value_counts())
print(df.TechSupport.isnull().any())
```

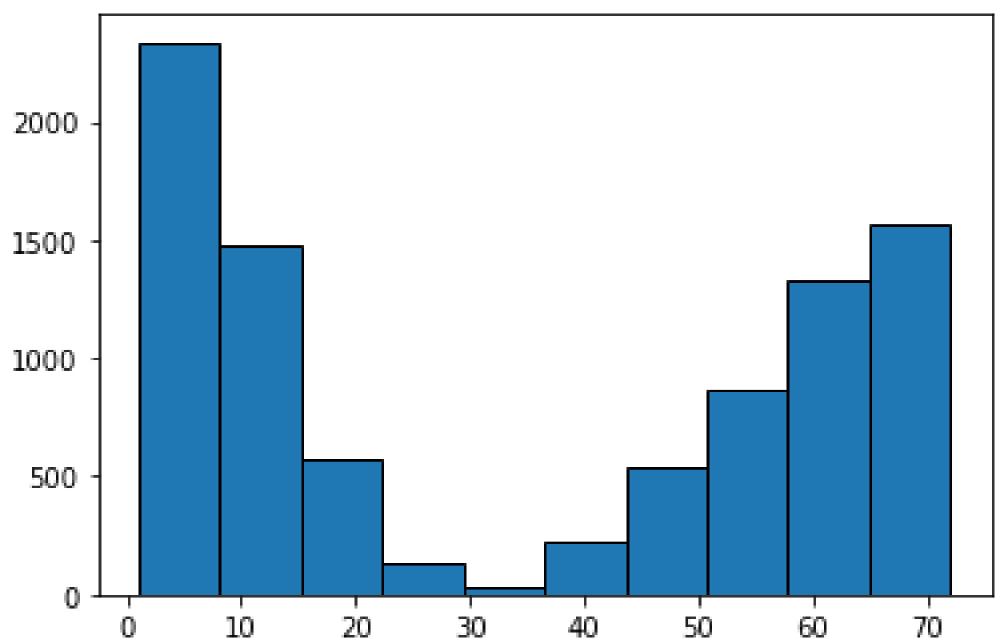
```
No     6626
Yes    3374
```

```
Name: TechSupport, dtype: int64
False
```

[27]:
```python
# Analyzing Tenure column.
maxTenure = np.nanmax(df['Tenure'])
minTenure = np.nanmin(df['Tenure'])
avgTenure = np.nanmean(df['Tenure'])
medTenure = np.nanmedian(df['Tenure'])
print("The maximum value in the Tenure column is ", maxTenure)
print("The minimum value in the Tenure column is ", minTenure)
print("The arithmetic mean value in the Tenure column is ", avgTenure)
print("The median value in the Tenure column is ", medTenure)
pp.hist(df['Tenure'], edgecolor='black')
```

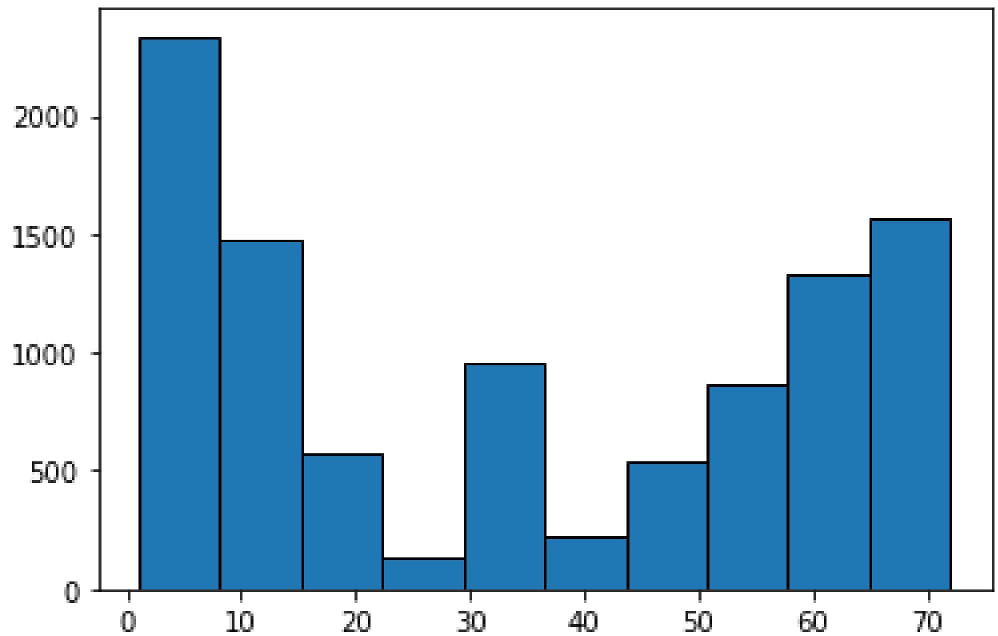
```
The maximum value in the Tenure column is  71.99928
The minimum value in the Tenure column is  1.00025934
The arithmetic mean value in the Tenure column is  34.49885764604521
The median value in the Tenure column is  36.19603
```

[27]:
```
(array([2337., 1473.,  568.,  132.,   30.,  221.,  540.,  870., 1330.,
        1568.]),
 array([ 1.00025934,  8.10016141, 15.20006347, 22.29996554, 29.3998676 ,
        36.49976967, 43.59967174, 50.6995738 , 57.79947587, 64.89937793,
        71.99928   ]),
 <BarContainer object of 10 artists>)
```



[28]:
```
# Imputing with mode & confirming no nulls remain.
df.Tenure.fillna(medTenure, inplace=True)
pp.hist(df['Tenure'], edgecolor='black')
print(df.Tenure.isnull().any())
```

```
False
```

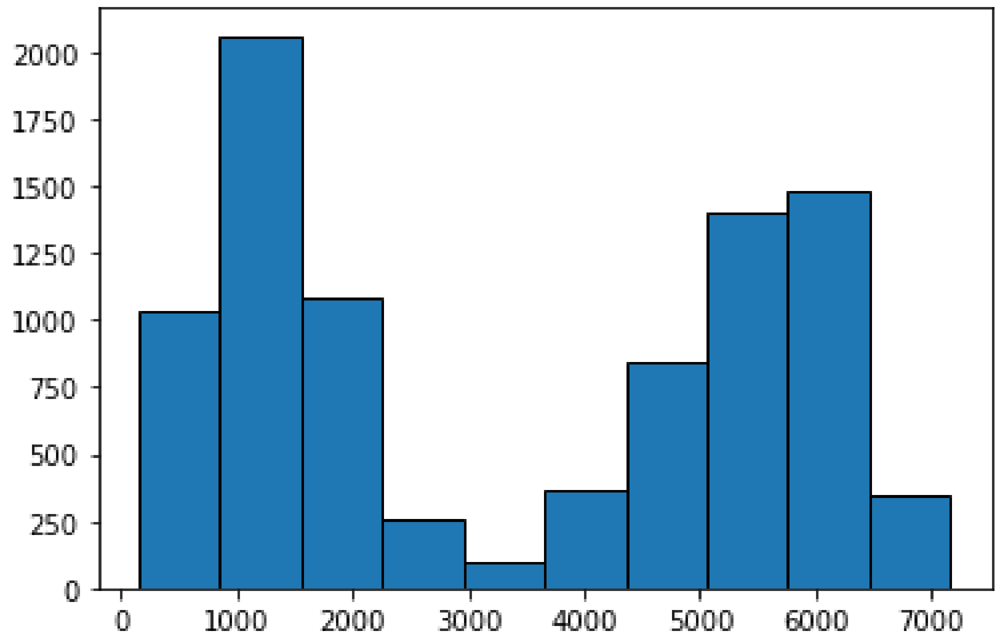


[29]:
```
# Analyzing Bandwidth_GB_Year column.
maxBandwidth = np.nanmax(df['Bandwidth_GB_Year'])
minBandwidth = np.nanmin(df['Bandwidth_GB_Year'])
avgBandwidth = np.nanmean(df['Bandwidth_GB_Year'])
medBandwidth = np.nanmedian(df['Bandwidth_GB_Year'])
print("The maximum value in the Bandwidth_GB_Year column is ", maxBandwidth)
print("The minimum value in the Bandwidth_GB_Year column is ", minBandwidth)
print("The arithmetic mean value in the Bandwidth_GB_Year column is ",␣
 ↪avgBandwidth)
print("The median value in the Bandwidth_GB_Year column is ", medBandwidth)
pp.hist(df['Bandwidth_GB_Year'], edgecolor='black')
```

```
The maximum value in the Bandwidth_GB_Year column is  7158.982
The minimum value in the Bandwidth_GB_Year column is  155.5067148
The arithmetic mean value in the Bandwidth_GB_Year column is  3398.842752015135
The median value in the Bandwidth_GB_Year column is  3382.424
```

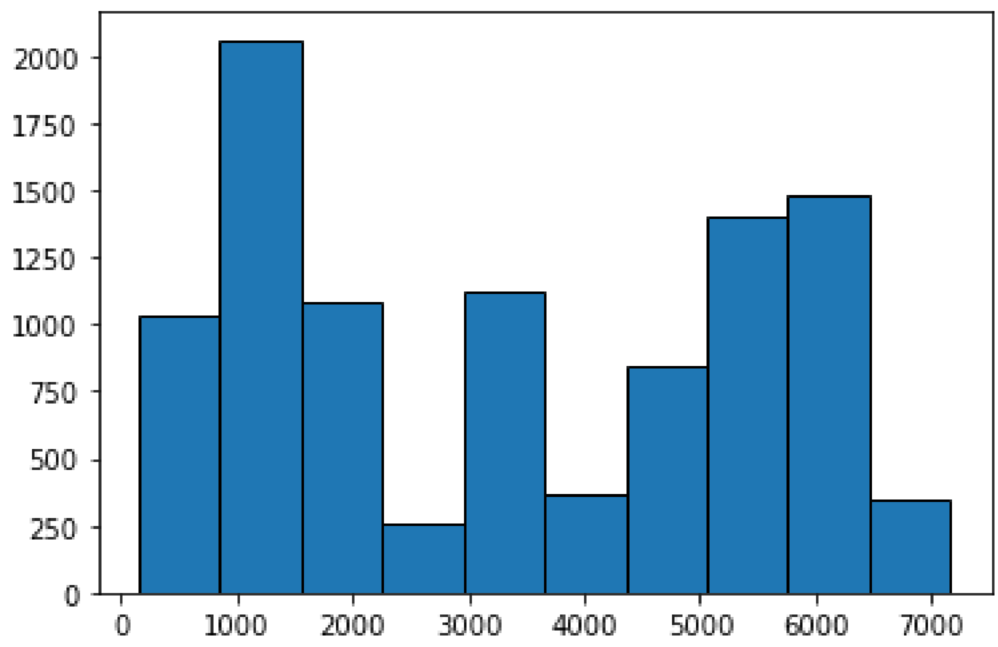
[29]:
```
(array([1031., 2062., 1087.,  255.,   98.,  370.,  844., 1399., 1484.,
          349.]),
```

```
array([ 155.5067148 ,  855.85424332, 1556.20177184, 2256.54930036,
        2956.89682888, 3657.2443574 , 4357.59188592, 5057.93941444,
        5758.28694296, 6458.63447148, 7158.982     ]),
 <BarContainer object of 10 artists>)
```



[30]:
```
# Imputing with median & confirming no nulls remain
df['Bandwidth_GB_Year'].fillna(medBandwidth, inplace=True)
pp.hist(df['Bandwidth_GB_Year'], edgecolor='black')
print(df.Bandwidth_GB_Year.isnull().any())
```

```
False
```



[31]:
```
# Confirming all nulls have been replaced in the dataframe
print(df.isnull().any())
```

```
Customer_id                False
Interaction                False
City                       False
State                      False
County                     False
Zip                        False
Lat                        False
Lng                        False
Population                 False
Area                       False
Timezone                   False
Job                        False
Children                   False
Age                        False
Education                  False
Employment                 False
Income                     False
Marital                    False
Gender                     False
Churn                      False
Outage_sec_perweek         False
Email                      False
Contacts                   False
```

```
Yearly_equip_failure    False
Contract                False
Port_modem              False
Tablet                  False
InternetService         False
Phone                   False
Multiple                False
OnlineSecurity          False
OnlineBackup            False
DeviceProtection        False
TechSupport             False
StreamingTV             False
StreamingMovies         False
PaperlessBilling        False
PaymentMethod           False
Tenure                  False
MonthlyCharge           False
Bandwidth_GB_Year       False
Response                False
Fix                     False
Replacement             False
Reliability             False
Options                 False
Respectful              False
Courteous               False
Listening               False
dtype: bool
```

[32]:
```
# All nulls have been imputed. Review remaining variables for errors and/or␣
↪outliers.
```

[33]:
```
# Analyzing City column.
print(len(df.City.unique()))
```

```
6058
```

[34]:
```
# This list accounts for > 60% of the values in the column. It is unreasonable␣
↪to review this manually,
# so the column will be left as-is.
```

[35]:
```
# Analyzing State column.
print(df.State.value_counts())
print(len(df.State.value_counts()))
```

```
TX    603
NY    558
PA    550
CA    526
IL    413
```

```
OH    359
FL    324
MO    310
VA    285
NC    280
MI    279
IA    279
MN    264
WV    247
IN    241
KY    238
GA    238
WI    228
OK    203
KS    195
NJ    190
TN    185
AL    181
NE    181
AR    176
WA    175
MA    172
CO    155
LA    141
MS    126
SC    124
MD    123
ND    118
OR    114
NM    114
AZ    112
ME    112
SD    101
MT     96
NH     85
VT     84
ID     81
AK     77
CT     71
UT     66
NV     48
WY     43
PR     40
HI     35
DE     21
RI     19
DC     14
Name: State, dtype: int64
```

```
52
```

[36]:
```python
# No apparent errors in state column.
```

[37]:
```python
# Analyzing County column.
print(len(df.County.unique()))
```

```
1620
```

[38]:
```python
# This list accounts for > 16% of the values in the column. It is unreasonable␣
↪to review this manually,
# so the column will be left as-is.
```

[39]:
```python
# Analyzing Zip column.
# The range for zip codes in the US is 00001 to 99950
print(min(df.Zip))
print(max(df.Zip))
```

```
601
99929
```

[40]:
```python
# The minimum and maximum values are within the defined range for zip codes.
```

[41]:
```python
# Analyzing Lat and Lng columns.
# Latitude range: -90 to 90, Longitude range: -180 to 180
print(max(df.Lat))
print(min(df.Lat))
print(max(df.Lng))
print(min(df.Lng))
```

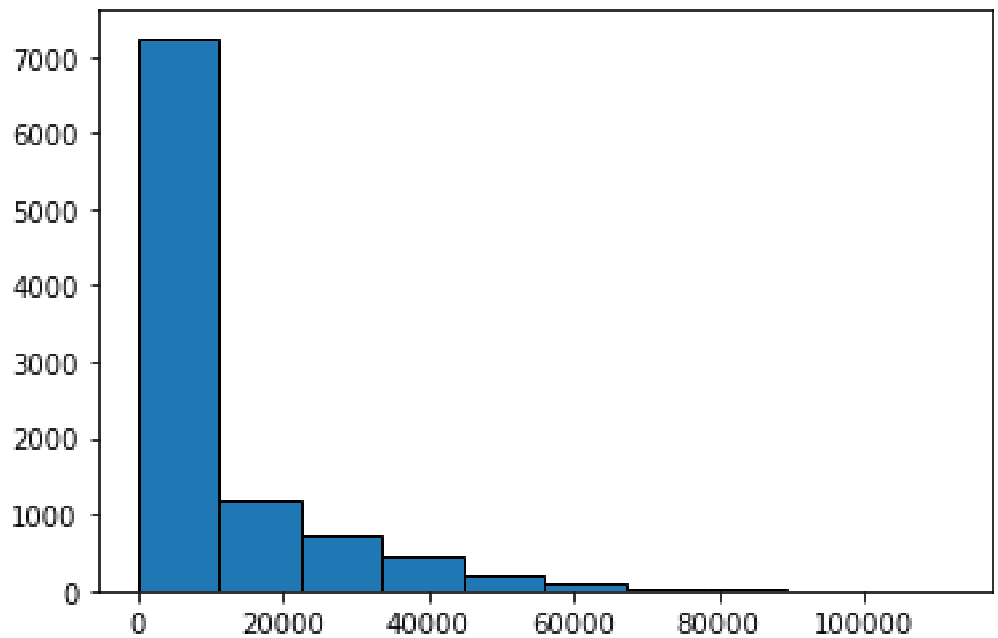
```
70.64066
17.96612
-65.66785
-171.68815
```

[42]:
```python
# These values are within the respective ranges.
```

[43]:
```python
# Analyzing Popluation column.
# Analyzing Bandwidth_GB_Year column.
maxPop = np.nanmax(df['Population'])
minPop = np.nanmin(df['Population'])
avgPop = np.nanmean(df['Population'])
medPop = np.nanmedian(df['Population'])
print("The maximum value in the Population column is ", maxPop)
print("The minimum value in the Population column is ", minPop)
print("The arithmetic mean value in the Population column is ", avgPop)
print("The median value in the Population column is ", medPop)
pp.hist(df.Population, edgecolor='black')
```

```
The maximum value in the Population column is  111850
The minimum value in the Population column is  0
The arithmetic mean value in the Population column is  9756.5624
The median value in the Population column is  2910.5
```

[43]:
```
(array([7.243e+03, 1.178e+03, 7.340e+02, 4.610e+02, 2.090e+02, 1.050e+02,
        3.500e+01, 2.400e+01, 8.000e+00, 3.000e+00]),
 array([     0.,  11185.,  22370.,  33555.,  44740.,  55925.,  67110.,
         78295.,  89480., 100665., 111850.]),
 <BarContainer object of 10 artists>)
```



[44]:
```
# Analyzing the 20 largest Population values.
cityPopSorted = df[['City', 'State','Population']].sort_values(by='Population',
 ascending=False)
print(cityPopSorted.head(20))
```

```
                    City State  Population
CaseOrder
8140             Chicago    IL      111850
8321               Bronx    NY      103732
6289        Bell Gardens    CA      102433
1776            Brooklyn    NY       98660
6611             Fontana    CA       96575
8131         Los Angeles    CA       96436
7442           Riverside    CA       94512
2403           Riverside    CA       94512
```

```
1894            Chicago    IL       94395
4350      Lawrenceville    GA       90675
204             Chicago    IL       90517
1399           Brooklyn    NY       89075
443            Brooklyn    NY       88349
5899            Cypress    TX       88344
9988            Chicago    IL       87509
6465            Chicago    IL       87509
1212          La Puente    CA       87240
158         League City    TX       86926
8080          San Diego    CA       86811
7455        Watsonville    CA       86703
```

[45]:
```
# These values correspond to highly populated areas and do not appear to be
# a recording error. While the zeroes in the population column seem unlikely,
# the values were pulled from census data and I do not feel properly
# equipped to replace them.
```

[46]:
```
# Analyzing Timezone column.
print(df.Area.value_counts())
print(df.Timezone.value_counts())
```

```
Suburban    3346
Rural       3327
Urban       3327
Name: Area, dtype: int64
America/New_York                  4072
America/Chicago                   3672
America/Los_Angeles                887
America/Denver                     552
America/Detroit                    265
America/Indiana/Indianapolis       186
America/Phoenix                    104
America/Boise                       57
America/Anchorage                   55
America/Puerto_Rico                 40
Pacific/Honolulu                    35
America/Menominee                   16
America/Nome                        12
America/Kentucky/Louisville         10
America/Sitka                        8
America/Indiana/Vincennes            6
America/Indiana/Tell_City            6
America/Toronto                      5
America/Indiana/Petersburg           4
America/Juneau                       2
America/North_Dakota/New_Salem       2
America/Indiana/Winamac              1
```

```
America/Ojinaga                      1
America/Indiana/Marengo              1
America/Indiana/Knox                 1
Name: Timezone, dtype: int64
```

[47]:
```python
# The outputs are as expected.
```

[48]:
```python
# Analyzing Job column.
print(len(df.Job.unique()))
print(df.Job.head(20))
```

```
639
CaseOrder
1        Environmental health practitioner
2                   Programmer, multimedia
3                  Chief Financial Officer
4                                Solicitor
5                      Medical illustrator
6                 Chief Technology Officer
7                   Surveyor, hydrographic
8        Sales promotion account executive
9           Teaching laboratory technician
10                 Museum education officer
11      Teacher, special educational needs
12                    Maintenance engineer
13     Engineer, broadcasting (operations)
14               Learning disability nurse
15                     Automotive engineer
16                  Amenity horticulturist
17                  Applications developer
18                            Immunologist
19                    Engineer, electrical
20                     Broadcast presenter
Name: Job, dtype: object
```

[49]:
```python
# These values are self-reported and categorical. They may no longer be true,
# or may never have been true. With such a wide variance in entries, it would
# be challenging to draw any reasonable conclusions based on this variable.
# Therefore, it will be dropped.
```

[50]:
```python
# Dropping Job column & confirming it has been dropped.
print(df.shape)
df.drop(columns='Job', inplace=True)
print(df.shape)
```

```
(10000, 49)
(10000, 48)
```

[51]:
```
# Analyzing Education, Employment, Marital, Gender, and Churn columns.
print(df.Education.value_counts())
print(df.Employment.value_counts())
print(df.Marital.value_counts())
print(df.Gender.value_counts())
print(df.Churn.value_counts())
```

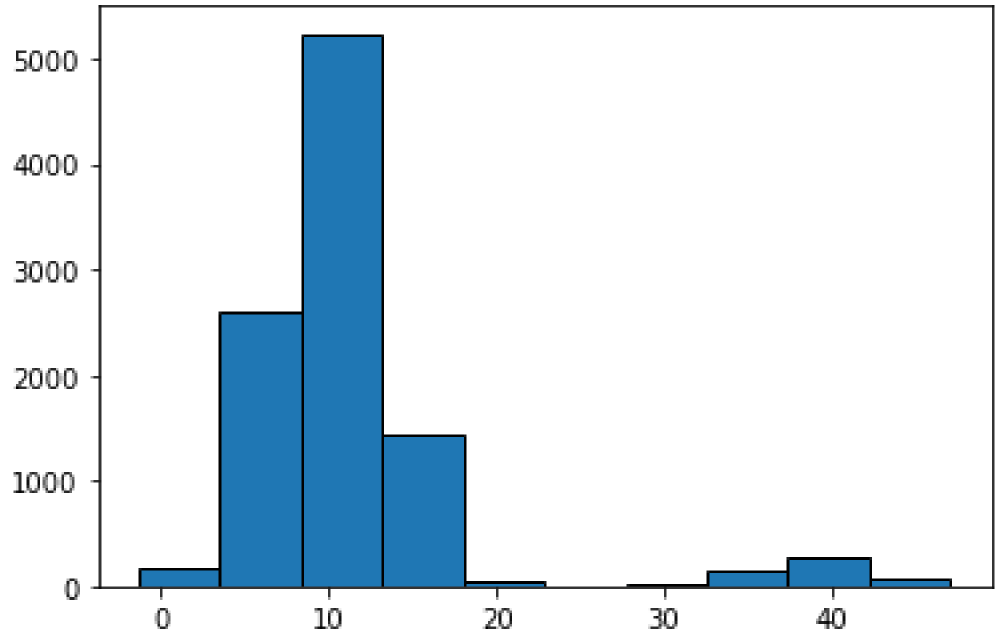
```
Regular High School Diploma                2421
Bachelor's Degree                          1703
Some College, 1 or More Years, No Degree   1562
9th Grade to 12th Grade, No Diploma         870
Master's Degree                             764
Associate's Degree                          760
Some College, Less than 1 Year              652
Nursery School to 8th Grade                 449
GED or Alternative Credential               387
Professional School Degree                  198
No Schooling Completed                      118
Doctorate Degree                            116
Name: Education, dtype: int64
Full Time     5992
Part Time     1042
Retired       1011
Unemployed     991
Student        964
Name: Employment, dtype: int64
Divorced         2092
Widowed          2027
Separated        2014
Never Married    1956
Married          1911
Name: Marital, dtype: int64
Female                       5025
Male                         4744
Prefer not to answer          231
Name: Gender, dtype: int64
No     7350
Yes    2650
Name: Churn, dtype: int64
```

[52]:
```
# There are no apparent errors in these outputs.
```

[53]:
```
# Analyzing Outage_sec_perweek
print(df.Outage_sec_perweek.max())
print(df.Outage_sec_perweek.min())
print(df.Outage_sec_perweek.mean())
print(df.Outage_sec_perweek.median())
pp.hist(df.Outage_sec_perweek, edgecolor='black')
```

```
pp.show()
```

```
47.04928
-1.348571
11.452955137651369
10.20289623
```



[54]:
```
# It appears that there are some negative values for time which cannot be true.
# There is a small grouping of values on the high end but the maximum value is
# still reasonable.
```

[55]:
```
negOutage = df.query('Outage_sec_perweek < 0')
print(negOutage.shape)
print(negOutage.Outage_sec_perweek)
```

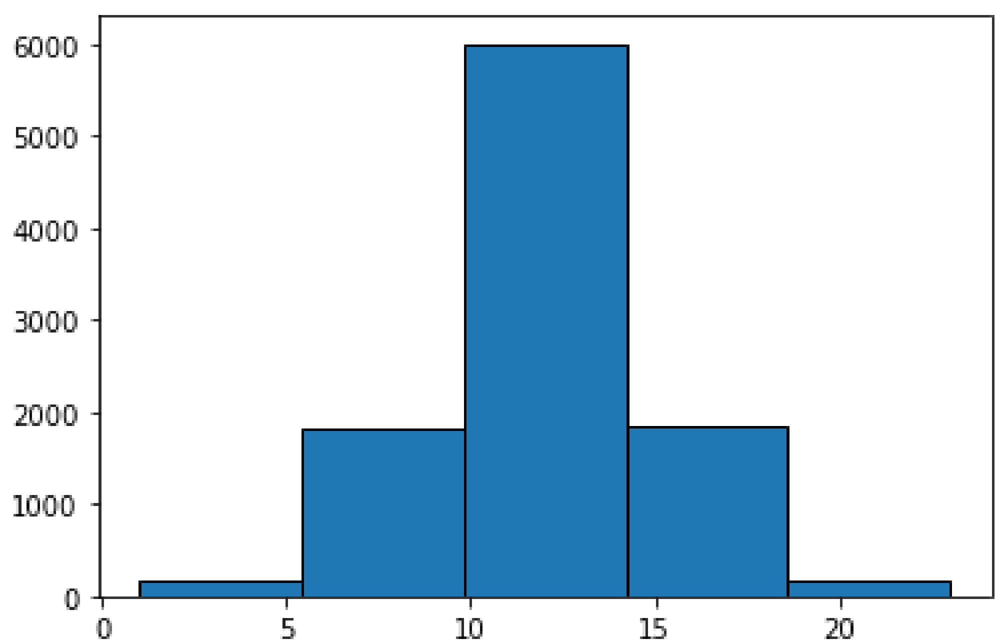
```
(11, 48)
CaseOrder
1905   -1.195428
1998   -0.339214
3070   -0.206145
3630   -0.152845
4168   -1.348571
4185   -0.352431
4428   -1.099934
6094   -0.787115
6464   -0.144644
6578   -0.527396
```

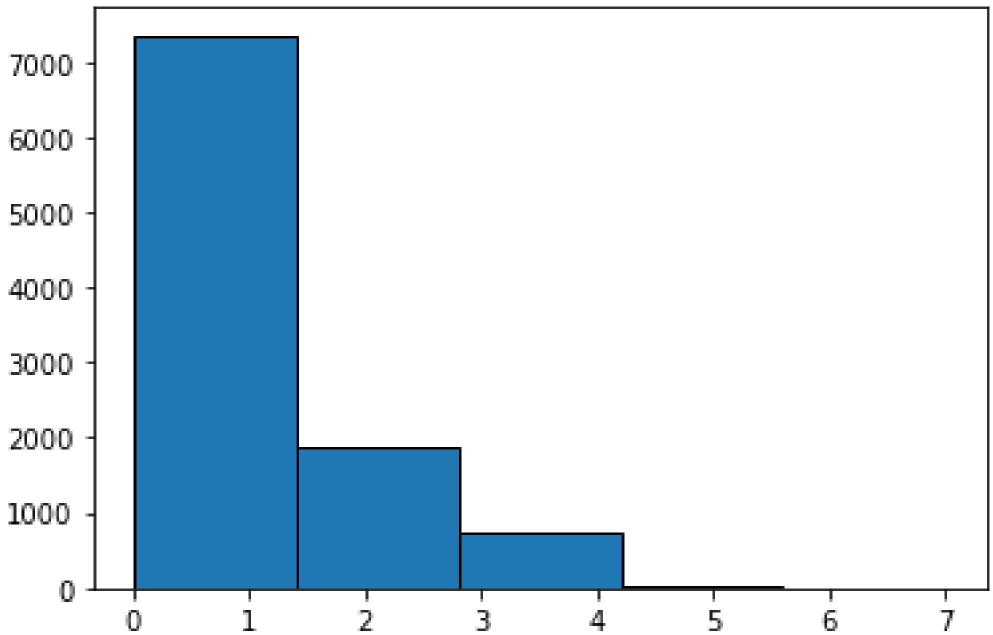
```
8195   -0.214328
Name: Outage_sec_perweek, dtype: float64
```

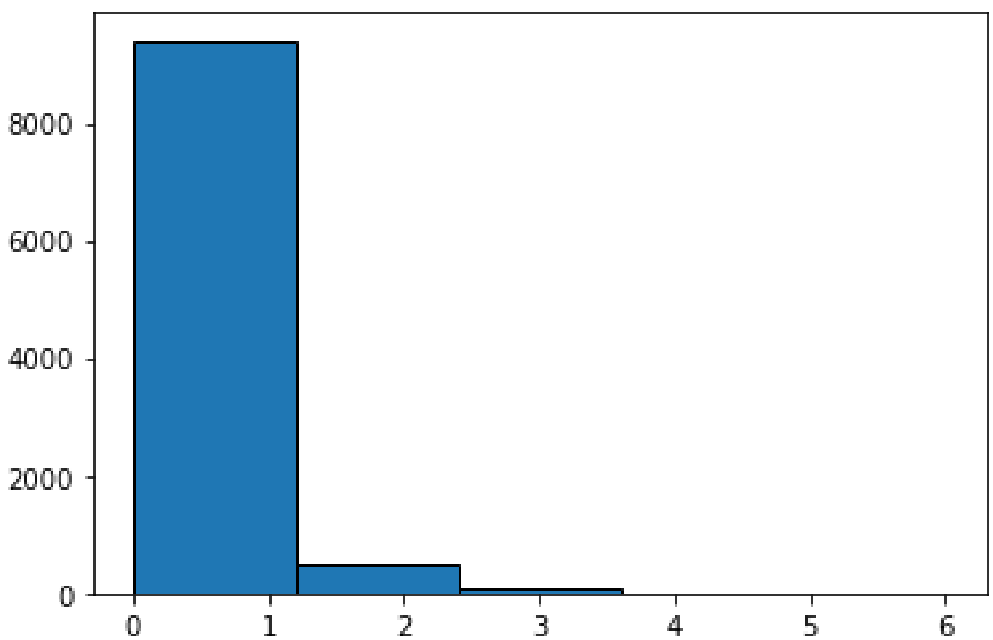
[56]:
```
# There are only 11 entries that are negative, all of which are close to zero.
# I will replace these with zero as they are likely due to some recording error
# and such a minor change will not dramatically affect the data set.
```

[57]:
```
# Replacing negative values in Outage_sec_perweek
negOutageIds = negOutage.Customer_id
df.loc[df.Customer_id.isin(negOutageIds), 'Outage_sec_perweek'] = '0'
```

[58]:
```
# Analyzing Email, Contacts, and Yearly_equip_failure columns.
pp.hist(df.Email, bins=5, edgecolor='black')
pp.show()

pp.hist(df.Contacts, bins=5, edgecolor='black')
pp.show()

pp.hist(df.Yearly_equip_failure, bins=5, edgecolor='black')
pp.show()
```







[59]:
```
# Email appears normally distributed. Contacts and Yearly_equip_failure are␣
↪skewed right, but reasonable.
```

[60]:

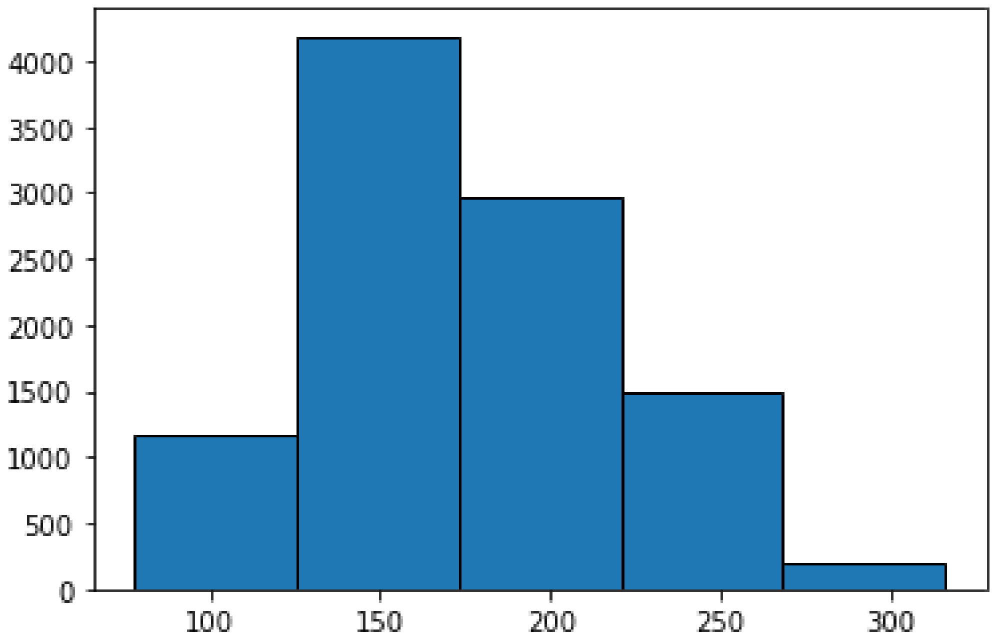
```python
# Analyzing several categorical columns using value_counts
cols = ['Contract', 'Port_modem', 'Tablet', 'InternetService', 'Multiple',
        'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'StreamingTV',
        'StreamingMovies', 'PaperlessBilling', 'PaymentMethod']

for col in cols:
   print(df[col].value_counts())
```

```
Month-to-month    5456
Two Year          2442
One year          2102
Name: Contract, dtype: int64
No     5166
Yes    4834
Name: Port_modem, dtype: int64
No     7009
Yes    2991
Name: Tablet, dtype: int64
Fiber Optic    4408
DSL            3463
None           2129
Name: InternetService, dtype: int64
No     5392
Yes    4608
Name: Multiple, dtype: int64
No     6424
Yes    3576
Name: OnlineSecurity, dtype: int64
No     5494
Yes    4506
Name: OnlineBackup, dtype: int64
No     5614
Yes    4386
Name: DeviceProtection, dtype: int64
No     5071
Yes    4929
Name: StreamingTV, dtype: int64
No     5110
Yes    4890
Name: StreamingMovies, dtype: int64
Yes    5882
No     4118
Name: PaperlessBilling, dtype: int64
Electronic Check            3398
Mailed Check                2290
Bank Transfer(automatic)    2229
Credit Card (automatic)     2083
Name: PaymentMethod, dtype: int64
```

[61]:
```
# The outputs are as expected.
```

[62]:
```
# Analyzing MonthlyCharge column.
pp.hist(df.MonthlyCharge, bins=5, edgecolor='black')
```

[62]:
```
(array([1166., 4189., 2960., 1488.,  197.]),
 array([ 77.50523 , 125.179904, 172.854578, 220.529252, 268.203926,
        315.8786  ]),
 <BarContainer object of 5 artists>)
```



[63]:
```
# There are no apparent errors.
```

[64]:
```
# Analyzing Survey Response columns
cols2 = ['Response', 'Fix', 'Replacement', 'Reliability', 'Options',
         'Respectful', 'Courteous', 'Listening']

for col in cols2:
    print(df[col].value_counts())
```

```
3    3448
4    3358
2    1393
5    1359
1     224
6     199
7      19
```

```
Name: Response, dtype: int64
3    3415
4    3412
5    1368
2    1360
1     217
6     215
7      13
Name: Fix, dtype: int64
3    3435
4    3410
2    1424
5    1313
6     203
1     202
7      12
8       1
Name: Replacement, dtype: int64
4    3452
3    3430
2    1350
5    1335
1     221
6     203
7       9
Name: Reliability, dtype: int64
3    3462
4    3417
2    1378
5    1321
1     206
6     204
7      12
Name: Options, dtype: int64
3    3445
4    3333
2    1427
5    1382
6     210
1     190
7      12
8       1
Name: Respectful, dtype: int64
4    3456
3    3446
5    1335
2    1309
6     224
```

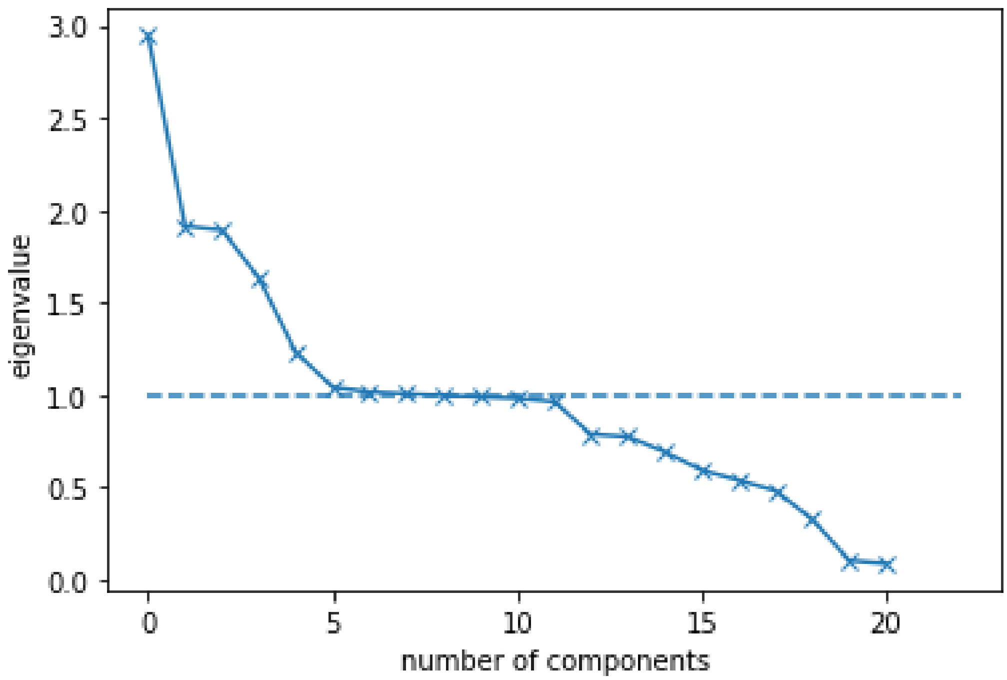
```
1     219
7      11
Name: Courteous, dtype: int64
3    3461
4    3400
2    1378
5    1335
1     206
6     205
7      14
8       1
Name: Listening, dtype: int64
```

[65]:
```
# All outputs are as expected.
```

[66]:
```
# Exporting cleaned data set to CSV
df.to_csv(r'C:/Users/sigsp/OneDrive/Desktop/D206 Data Cleaning/D206 PA/
 ↪churn_data_cleaned.csv')
```

[67]:
```
# Performing PCA on numeric columns
numericCols = df.select_dtypes(exclude=['object']).columns.tolist()

dfNumeric = df[numericCols]

numericCols_normalized = (dfNumeric - dfNumeric.mean())/dfNumeric.std()

pca = PCA(n_components=dfNumeric.shape[1])

# Loop to name PC columns based on the number of numeric columns in the␣
 ↪dataframe
PCcols = []
i = 1
while i < len(numericCols)+1:
    col = 'PC'+str(i)
    PCcols.append(col)
    i+=1

pca.fit(numericCols_normalized)
numeric_pca = pd.DataFrame(pca.transform(numericCols_normalized),␣
 ↪columns=numericCols)

cov_matrix = np.dot(numericCols_normalized.T, numericCols_normalized)/dfNumeric.
 ↪shape[0]
eigenvalues = [np.dot(eigenvector.T, np.dot(cov_matrix, eigenvector)) for␣
 ↪eigenvector in pca.components_]

pp.plot(eigenvalues, marker='x', linestyle='solid')
```

```
pp.hlines(1, xmin=0, xmax=len(numericCols)+1, linestyles='dashed')
pp.xlabel('number of components')
pp.ylabel('eigenvalue')
pp.show()
```



[68]:
```
# Determine the number of eigenvalues that are greater than 1.
pcCount = 0
for eigenvalue in eigenvalues:
    if eigenvalue > 1:
        pcCount +=1
print(pcCount)
```

```
8
```

[69]:
```
# Printing loadings for only the first 8 PCs
loading = pd.DataFrame(pca.components_.T, columns=PCcols, index=dfNumeric.
 ↪columns)
print(loading[['PC1', 'PC2', 'PC3', 'PC4', 'PC5', 'PC6', 'PC7', 'PC8']])
```

```
                     PC1       PC2       PC3       PC4       PC5  \
Zip            -0.018931  0.598838  0.365518  0.045303  0.026119
Lat            -0.001150  0.036502 -0.004353 -0.002031 -0.706089
Lng             0.016862 -0.602131 -0.365082 -0.043200  0.061109
Population     -0.002874  0.051722  0.039456  0.022258  0.681069
Children        0.000786 -0.028179 -0.019273  0.006929 -0.050785
```

```
Age                 0.004930  0.005894 -0.011708 -0.018286  0.018018
Income             -0.000825 -0.003199  0.007001  0.024352 -0.058390
Email               0.008851  0.000745 -0.023759 -0.004232  0.148738
Contacts           -0.008470 -0.009626 -0.001432 -0.011635  0.024561
Yearly_equip_failure -0.007742  0.001445  0.019497  0.009231 -0.005756
Tenure             -0.010451 -0.360696  0.601395 -0.072172 -0.008579
MonthlyCharge       0.000519 -0.032164  0.027864 -0.016139 -0.023028
Bandwidth_GB_Year  -0.012304 -0.361333  0.602778 -0.073755 -0.011388
Response            0.458913 -0.031597  0.017445  0.280261 -0.012679
Fix                 0.433890 -0.021589  0.036494  0.282590 -0.017367
Replacement         0.400713 -0.030163  0.021895  0.280279 -0.002357
Reliability         0.145633  0.056781 -0.024044 -0.566765 -0.006861
Options            -0.175493 -0.065043  0.037391  0.585637 -0.005884
Respectful          0.404781  0.036745  0.008165 -0.181169  0.015380
Courteous           0.358110  0.019892  0.007784 -0.180468 -0.020108
Listening           0.308741  0.024837 -0.004353 -0.130058  0.046584

                          PC6       PC7       PC8
Zip                 0.006616 -0.005832 -0.004627
Lat                -0.063641 -0.059369  0.013276
Lng                 0.001827  0.019417 -0.001259
Population          0.055606  0.054889  0.050688
Children            0.579724 -0.052362 -0.007114
Age                -0.472411  0.438277 -0.113990
Income              0.203470  0.307718  0.699475
Email              -0.165072 -0.546458  0.005485
Contacts           -0.494358  0.222099  0.283918
Yearly_equip_failure  0.259411  0.563864 -0.525024
Tenure             -0.008217  0.002231  0.023086
MonthlyCharge      -0.224926 -0.186772 -0.365263
Bandwidth_GB_Year   0.005902 -0.017990  0.000098
Response           -0.000165  0.003877 -0.023071
Fix                -0.020815  0.007019 -0.000548
Replacement        -0.002100 -0.027489 -0.014955
Reliability        -0.001055 -0.001000 -0.010733
Options            -0.039833  0.004247 -0.003541
Respectful          0.004302  0.004196  0.002666
Courteous           0.018599  0.005635  0.059649
Listening          -0.011927  0.039299 -0.015280
```

[ ]: