DATA CLEANING — D206

Eric D. Otten

Student ID: 011399183

Task 1

A. Describe one question or decision that could be addressed using the data set you chose. The summarized question or decision must be relevant to a realistic organizational need or situation.

My research question is: "Which customer factors have the greatest impact on a customer's decision to churn?" This question is relevant to an organization because decision-makers use customer behavioral patterns to increase customer retention.

B. Describe all variables in the data set (regardless of the research question) and indicate the data type for each variable. Use examples from the data set to support your claims.

This analysis utilizes the "churn_raw_data.csv" file containing 10,000 rows and 52 columns. We will focus on the "Outage_sec_perweek" and "Churn" columns. A description of all columns is shown below.

1. Column #1: CaseOrder – Integer (Index-Like). Example: 1. This column serves as a unique identifier.
2. Column #2: Customer_id – String (Alphanumeric Identifier). Example: K409198. This column identifies each customer with a unique alphanumeric code.
3. Column #3: Interaction – String (Alphanumeric Identifier). Example: aa90260b-4141-4a24-8e36-b04ce1f4f77b. This column uniquely identifies each customer interaction.
4. Column #4: City – String. Example: Point Baker. This column records the city where the customer is located.
5. Column #5: State – String. Example: AK. This column indicates the state where the customer is located using two-letter state codes.
6. Column #6: County – String. Example: Prince of Wales-Hyder. This column provides the county name associated with each customer.
7. Column #7: Zip – String. Example: 99927. This column stores the zip code.
8. Column #8: Lat – Float (Geographical Latitude). Example: 56.25100. This column represents the geographical latitude coordinate.
9. Column #9: Lng – Float (Geographical Longitude). Example: -133.37571. This column represents the geographical longitude coordinate.

10. Column #10: Population – Integer. Example: 500. This column indicates the population of the customer's area.
11. Column #11: Area – Integer. Example: 1000. This column represents the area size (in square miles).
12. Column #12: Timezone – String. Example: AKST. This column indicates the customer's time zone abbreviation.
13. Column #13: Job – String. Example: Engineer. This column contains the occupation/job title of the customer.
14. Column #14: Children – Integer. Example: 2. This column indicates the number of children the customer has.
15. Column #15: Age – Integer. Example: 35. This column represents the customer's age.
16. Column #16: Education – String. Example: Bachelor's. This column indicates the customer's education level.
17. Column #17: Employment – String. Example: Full-Time. This column reflects the customer's employment status.
18. Column #18: Income – Integer. Example: 60000. This column stores the annual income in dollars.
19. Column #19: Marital – String. Example: Married. This column provides the marital status of the customer.
20. Column #20: Gender – String. Example: Female. This column represents the customer's gender.
21. Column #21: Churn – Boolean. Example: Yes. This column indicates whether the customer has churned (left the service).
22. Column #22: Outage_sec_perweek – Float (Duration). Example: 300.0. This column indicates the average outage time per week in seconds.
23. Column #23: Email – Integer. Example: 10. This column counts the number of email interactions with the customer.
24. Column #24: Contacts – Integer. Example: 5. This column indicates the total number of contacts with the customer.
25. Column #25: Yearly_equip_failure – Integer. Example: 2. This column counts the number of equipment failures the customer experienced annually.
26. Column #26: Techie – Boolean. Example: Yes. This column indicates if the customer identifies as tech-savvy.
27. Column #27: Contract – String. Example: Month-to-Month. This column indicates the type of service contract the customer has.
28. Column #28: Port_modem – Boolean. Example: No. This column reflects whether the customer ports their modem.
29. Column #29: Tablet – Boolean. Example: Yes. This column shows whether the customer owns a tablet.

30. Column #30: InternetService – String. Example: Fiber. This column represents the customer's type of internet service.
31. Column #31: Phone – Boolean. Example: Yes. This column indicates whether the customer uses phone service.
32. Column #32: Multiple – Boolean. Example: No. This column shows whether the customer has multiple services.
33. Column #33: OnlineSecurity – Boolean. Example: Yes. This column indicates if the customer has subscribed to online security services.
34. Column #34: OnlineBackup – Boolean. Example: Yes. This column reflects whether the customer has subscribed to online backup services.
35. Column #35: DeviceProtection – Boolean. Example: No. This column indicates if the customer has device protection.
36. Column #36: TechSupport – Boolean. Example: Yes. This column shows whether the customer receives technical support.
37. Column #37: StreamingTV – Boolean. Example: Yes. This column indicates if the customer uses streaming TV services.
38. Column #38: StreamingMovies – Boolean. Example: No. This column shows if the customer streams movies.
39. Column #39: PaperlessBilling – Boolean. Example: Yes. This column reflects if the customer has opted for paperless billing.
40. Column #40: PaymentMethod – String. Example: Credit Card. This column indicates the customer's preferred payment method.
41. Column #41: Tenure – Integer. Example: 12. This column reflects the number of months the customer has been with the company.
42. Column #42: MonthlyCharge – Float. Example: 171.45. This column indicates the monthly charge for the customer's services.
43. Column #43: Bandwidth_GB_Year – Float. Example: 904.54. This column represents the customer's yearly bandwidth usage in gigabytes.
44. Columns #44-51: item1 to item8 – Integer (Rating or Categorical Data). Examples: 5, 4, 3. These columns could be ratings or represent some categorically relevant information about customers.

C. Explain the plan for cleaning the data by doing the following:

1. Propose a plan that includes the relevant techniques and specific steps needed to assess the quality of the data in the data set.

I plan to begin assessing the quality of the data set by identifying missing data. This can be done using a histogram or missingno matrix. Dean and Illowsky (2009) stated, "A histogram consists of contiguous boxes... The histogram can give you the shape of the data, the center, and the spread of the data." Additionally, I can run `.value_counts()` on a DataFrame column to list all distinct values and their count within the column. This can be easily used to identify data anomalies such as typos. Additionally, I can use the `.duplicated()` method to identify duplicate rows within the CaseOrder, and Customer_id, and Interaction columns as these columns should contain unique rows. Lastly, I can run `.describe()` on a DataFrame column to list the mean, standard deviation, and minimum to identify any outlier data which may result from an anomaly.

2. Justify your approach for assessing the quality of the data, including the following: characteristics of the data being assessed and the approach used to assess the quality of the data.

This plan for identifying data anomalies addresses multiple types of data anomalies using tools designed for this purpose. The missingno matrix will allow me to view the existence of data within the DataFrame, allowing me to identify columns in which there is missing data. Dean and Illowsky (2009) observed, "The histogram displays the heights on the x-axis and relative frequency on the y-axis." The `.value_counts()` method allows me to identify data anomalies by listing out categorical data. The `.duplicated()` method allows me to identify duplicate rows within columns where duplicate rows should not exist. The `.describe()` method allows me to identify data anomalies such as outliers that result from improperly entered data.

3. Justify your selected programming language and any libraries and packages that will support the data-cleaning process.

I have selected Python as the programming language that will support my data clearing process due to its simple and flexible nature. Additionally, I will use the `pandas`, `scikit-learn`, `missingno` and `matplotlib` libraries which allow for data manipulation, principal component analysis, missing data detection, and plotting functionality respectively.

4. Provide the annotated code you will use to assess the quality of the data in an executable script file.

Create a missingno matrix:

Code

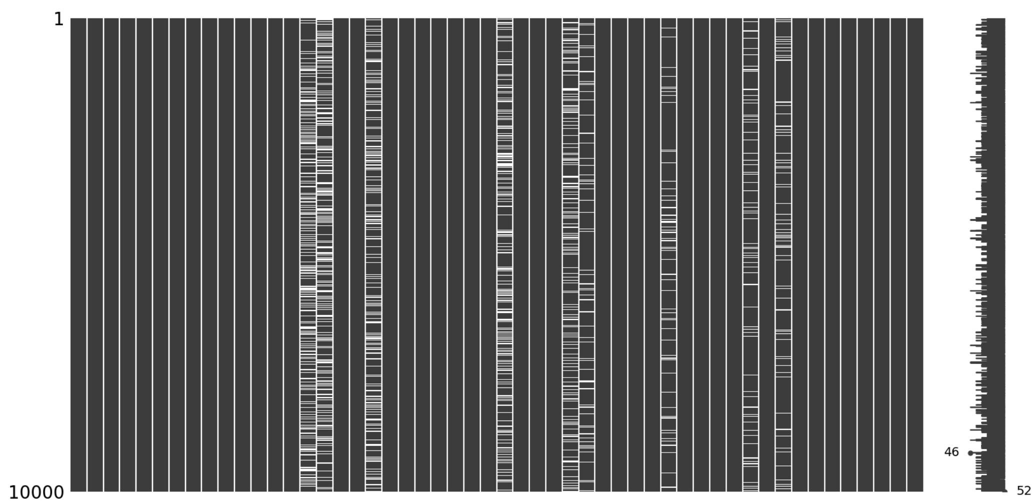
```
import pandas as pd
import missingno as msno
import matplotlib.pyplot as plt

# Assess Data Quality

# Read the CSV file
filename = "churn_raw_data.csv"
df = pd.read_csv(filename, keep_default_na = False, na_values=['NA'])

# Identify missing data
msno.matrix(df)
plt.show()
```

Result



Identify duplicate rows:

Code

```
# Identify duplicate rows based on specified columns
duplicates = df.duplicated(subset=['CaseOrder', 'Customer_id',
'Interaction'], keep=False)

# Display duplicate rows
duplicate_rows = df[duplicates]
print("Duplicate Rows based on 'CaseOrder', 'Customer_id', and
'Interaction':")
print(duplicate_rows)
```

Result

```
Empty DataFrame
Columns: [Unnamed: 0, CaseOrder, Customer_id, Interaction, City, State,
County, Zip, Lat, Lng, Population, Area, Timezone, Job, Children, Age,
Education, Employment, Income, Marital, Gender, Churn, Outage_sec_perweek,
Email, Contacts, Yearly_equip_failure, Techie, Contract, Port_modem, Tablet,
InternetService, Phone, Multiple, OnlineSecurity, OnlineBackup,
DeviceProtection, TechSupport, StreamingTV, StreamingMovies,
PaperlessBilling, PaymentMethod, Tenure, MonthlyCharge, Bandwidth_GB_Year,
item1, item2, item3, item4, item5, item6, item7, item8]
Index: []
```

Identify data types:

Code

```
# Identify data types and non-null count
df.info()
```

Result

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

Run `.value_counts()` on all string columns to identify anomalies.

Code

```
# For string (object) columns, use .value_counts()
string_columns = df.select_dtypes(include=['object']).columns
string_summary = {col: df[col].value_counts() for col in string_columns}

# Iterate through string_summary dictionary to print each string column's
value counts
for col, counts in string_summary.items():
    print(f"Column: {col}")
    print(counts)
    print()
```

Result

```
Column: Customer_id
Customer_id
K409198    1
X300173    1
M155745    1
G126132    1
O148559    1
          ..
F454437    1
W845098    1
P854487    1
K983374    1
T38070     1
Name: count, Length: 10000, dtype: int64

Column: Interaction
Interaction
aa90260b-4141-4a24-8e36-b04ce1f4f77b    1
26769b47-8eda-4e14-9baf-7348b64b7da3    1
6d65ca83-1001-4d01-a3f9-c3ae5ac33a83    1
448944cf-10f6-4a04-a8e0-4079b6791e26    1
a9890702-06c6-4337-9d5b-65f7d1e30466    1
                                       ..
c650b63b-2d68-48f2-911d-6e8c838c8185    1
3006986f-69e9-4c80-8dcb-1f8d917f2071    1
0e3b8690-177a-4bce-a4e9-823682ce8aec    1
25400298-b615-407d-9e79-25fb89b38429    1
9de5fb6e-bd33-4995-aec8-f01d0172a499    1
Name: count, Length: 10000, dtype: int64

Column: City
City
Houston         34
New York        24
Springfield     23
Buffalo         23
San Antonio     22
                ..
Cottontown       1
San Dimas        1
Fort Hill        1
Webster          1
Clarkesville     1
Name: count, Length: 6058, dtype: int64
```

```
Column: State
State
TX    603
NY    558
PA    550
CA    526
IL    413
OH    359
FL    324
MO    310
VA    285
NC    280
IA    279
MI    279
MN    264
WV    247
IN    241
GA    238
KY    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
NM    114
OR    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: count, dtype: int64

Column: County
County
Washington     111
Jefferson      100
Montgomery      99
Franklin        92
Los Angeles     91
              ...
Rooks            1
Cochise          1
Yauco            1
Hoke             1
Briscoe          1
Name: count, Length: 1620, dtype: int64

Column: Area
Area
Suburban    3346
Urban       3327
Rural       3327
Name: count, dtype: int64

Column: Timezone
Timezone
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/Knox                1
America/Indiana/Winamac             1
America/Indiana/Marengo             1
America/Ojinaga                     1
Name: count, dtype: int64

Column: Job
Job
Occupational psychologist              30
Comptroller                            28
Hospital pharmacist                    28
Horticultural therapist                28
Ranger/warden                          27
                                       ..
Control and instrumentation engineer    6
Travel agency manager                   6
Accountant, chartered certified         6
Arboriculturist                         6
Toxicologist                            6
Name: count, Length: 639, dtype: int64

Column: Education
Education
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: count, dtype: int64

Column: Employment
Employment
Full Time     5992
Part Time     1042
Retired       1011
```

```
Unemployed     991
Student        964
Name: count, dtype: int64

Column: Marital
Marital
Divorced         2092
Widowed          2027
Separated        2014
Never Married    1956
Married          1911
Name: count, dtype: int64

Column: Gender
Gender
Female                  5025
Male                    4744
Prefer not to answer     231
Name: count, dtype: int64

Column: Churn
Churn
No     7350
Yes    2650
Name: count, dtype: int64

Column: Techie
Techie
No     6266
Yes    1257
Name: count, dtype: int64

Column: Contract
Contract
Month-to-month    5456
Two Year          2442
One year          2102
Name: count, dtype: int64

Column: Port_modem
Port_modem
No     5166
Yes    4834
Name: count, dtype: int64

Column: Tablet
Tablet
No     7009
```

```
Yes    2991
Name: count, dtype: int64

Column: InternetService
InternetService
Fiber Optic    4408
DSL            3463
Name: count, dtype: int64

Column: Phone
Phone
Yes    8128
No      846
Name: count, dtype: int64

Column: Multiple
Multiple
No     5392
Yes    4608
Name: count, dtype: int64

Column: OnlineSecurity
OnlineSecurity
No     6424
Yes    3576
Name: count, dtype: int64

Column: OnlineBackup
No     5494
Yes    4506
Name: count, dtype: int64

Column: DeviceProtection
DeviceProtection
No     5614
Yes    4386
Name: count, dtype: int64

Column: TechSupport
TechSupport
No     5635
Yes    3374
Name: count, dtype: int64

Column: StreamingTV
StreamingTV
No     5071
Yes    4929
```

```
Name: count, dtype: int64

Column: StreamingMovies
StreamingMovies
No     5110
Yes    4890
Name: count, dtype: int64

Column: PaperlessBilling
PaperlessBilling
Yes    5882
No     4118
Name: count, dtype: int64

Column: PaymentMethod
PaymentMethod
Electronic Check             3398
Mailed Check                 2290
Bank Transfer(automatic)     2229
Credit Card (automatic)      2083
Name: count, dtype: int64
```

Run `.describe()` on numerical columns to identify outliers resulting from data anomalies.

Code

```
# For numerical (int64/float64) columns, use .describe()
numerical_columns = df.select_dtypes(include=['int64', 'float64']).columns
numerical_summary = df[numerical_columns].describe()

# Iterate through numerical_summary dictionary to print each string column's
value counts
for col, counts in numerical_summary.items():
    print(f"Column: {col}")
    print(counts)
    print()
```

Result

```
Column: Unnamed: 0
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: Unnamed: 0, dtype: float64

Column: 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

Column: 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

Column: 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

Column: 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
```

```
Column: 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

Column: Children
count    7505.000000
mean        2.095936
std         2.154758
min         0.000000
25%         0.000000
50%         1.000000
75%         3.000000
max        10.000000
Name: Children, dtype: float64

Column: Age
count    7525.000000
mean       53.275748
std        20.753928
min        18.000000
25%        35.000000
50%        53.000000
75%        71.000000
max        89.000000
Name: Age, dtype: float64

Column: Income
count      7510.000000
mean      39936.762226
std       28358.469482
min         740.660000
25%       19285.522500
50%       33186.785000
75%       53472.395000
max      258900.700000
Name: Income, dtype: float64

Column: Outage_sec_perweek
count    10000.000000
mean        11.452955
std          7.025921
```

```
min        -1.348571
25%         8.054362
50%        10.202896
75%        12.487644
max        47.049280
Name: Outage_sec_perweek, dtype: float64

Column: 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

Column: 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

Column: 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

Column: Tenure
count    9069.000000
mean       34.498858
std        26.438904
min         1.000259
25%         7.890442
50%        36.196030
75%        61.426670
```

```
max        71.999280
Name: Tenure, dtype: float64

Column: MonthlyCharge
count    10000.000000
mean       174.076305
std         43.335473
min         77.505230
25%        141.071078
50%        169.915400
75%        203.777441
max        315.878600
Name: MonthlyCharge, dtype: float64

Column: Bandwidth_GB_Year
count    8979.000000
mean     3398.842752
std      2187.396807
min       155.506715
25%      1234.110529
50%      3382.424000
75%      5587.096500
max      7158.982000
Name: Bandwidth_GB_Year, dtype: float64

Column: 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

Column: 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

Column: 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

Column: 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

Column: item5
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

Column: 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

Column: 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

Column: 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
```

Quantify the number of zip codes which have lost their leading zeroes:

Code

```
# Investigate anomalies in the zip column
df['Zip_str'] = df['Zip'].apply(lambda x: str(int(x)))

# Count the number of anomalies where the length of the zip code is not 5
anomaly_count = (df['Zip_str'].apply(len) != 5).sum()

# Print the number of anomalies
print("Number of zip code anomalies (missing leading zeros):",
anomaly_count)
```

Result

```
Number of zip code anomalies (missing leading zeros): 773
```

Investigate Outage_sec_perweek anomalies

Code

```
# Investigate outliers in Outage_sec_perweek column
df.Outage_sec_perweek.nsmallest(n=20)
```

Result

```
4167   -1.348571
1904   -1.195428
4427   -1.099934
```

```
6093   -0.787115
6577   -0.527396
4184   -0.352431
1997   -0.339214
8194   -0.214328
3069   -0.206145
3629   -0.152845
6463   -0.144644
7339    0.113821
908     0.169351
4697    0.278712
7070    0.359073
9402    0.683623
2984    0.840953
8191    0.852520
8180    0.915846
7389    0.994552
Name: Outage_sec_perweek, dtype: float64
```

D. Summarize the data-cleaning process by doing the following:

1. Describe the findings for the data quality issues found from the implementation of the data-cleaning plan from part C.

- Columns #44-51 are improperly named as item1 to item8.
- Based on the analysis of `df.info()`, the following columns contain missing data:
  - Children: There are 2,495 missing values (10,000 - 7,505).
  - Age: There are 2,475 missing values (10,000 - 7,525).
  - Income: There are 2,490 missing values (10,000 - 7,510).
  - Techie: There are 2,477 missing values (10,000 - 7,523).
  - Phone: There are 1,026 missing values (10,000 - 8,974).
  - TechSupport: There are 991 missing values (10,000 - 9,009).
  - Tenure: There are 931 missing values (10,000 - 9,069).
  - Bandwidth_GB_Year: There are 1,021 missing values (10,000 - 8,979).
- The following rows in the Outage_sec_perweek column contain several negative values which are anomalous because it's impossible to have negative outage time:
  - 4167 -1.348571
  - 1904 -1.195428
  - 4427 -1.099934
  - 6093 -0.787115
  - 6577 -0.527396

- 4184 -0.352431
- 1997 -0.339214
- 8194 -0.214328
- 3069 -0.206145
- 3629 -0.152845
- 6463 -0.144644

- City, State, County, Area, Timezone, Job, Education, Employment, Marital, Gender, Contract, InternetService, and PaymentMethod would be more efficiently stored as a category instead of a string.
- Churn, Techie, Port_modem, Tablet, Phone, Multiple, OnlineSecurity, OnlineBackup, DeviceProtection, TechSupport, StreamingTV, StreamingMovies, and PaperlessBilling would be more efficiently stored as a boolean instead of a string.
- Zip code is stored as a float instead of a string and as a result, 773 rows are missing their leading zeroes.
- CaseOrder, Population, Children, Age, Email, and Contacts would be more accurately stored as an integer as they consist of whole numbers.

2. Justify your methods for mitigating the data quality issues in the data set.

Mitigating the data quality issues found in the naming of item1 through item8 would require additional information.

To mitigate the remaining data anomalies, I will perform the following steps:

1. Convert Churn, Techie, Port_modem, Tablet, Phone, Multiple, OnlineSecurity, OnlineBackup, DeviceProtection, TechSupport, StreamingTV, StreamingMovies, and PaperlessBilling to booleans with Yes being True and No being False.
2. Convert City, State, County, Area, Timezone, Job, Education, Employment, Marital, Gender, Contract, InternetService, and PaymentMethod columns to categories.
3. Convert Zip Code to a string and add leading zeros so each value is 5 digits in length.
4. Convert CaseOrder, Population, Children, Age, Email, and Contacts to integers.
5. Convert negative values in the Outage_sec_perweek column to absolute value
6. Impute Children, Age, Income, Tenure, and Bandwidth_GB_Year with averages.
7. Impute Techie, Phone, TechSupport with most common values.

3. Summarize the outcome from the implementation of each data-cleaning step.

- By converting columns with repeated values to booleans or categories, categorical data will be more memory-efficient, quicker to analyze, and useful for modeling.

- By converting Zip Codes to strings, each row will be consistently formatted to five digits with leading zeros.
- By converting floats that represent whole numbers to integers, columns intended to represent whole numbers will now be accurate integers.
- By converting negative values in outage time to positive numbers, one can resolve logical inconsistencies.
- By imputing averages and most frequent categories in columns, missing values will be filled to provide consistent, usable data.

4. Provide the annotated code you will use to mitigate the data quality issues—including anomalies—in the data set in an executable script file.

1. Convert Churn, Techie, Port_modem, Tablet, Phone, Multiple, OnlineSecurity, OnlineBackup, DeviceProtection, TechSupport, StreamingTV, StreamingMovies, and PaperlessBilling to booleans with Yes being True and No being False.

Code

```
boolean_columns = ['Churn', 'Techie', 'Port_modem', 'Tablet', 'Phone',
'Multiple', 'OnlineSecurity', 'OnlineBackup', 'DeviceProtection',
'TechSupport', 'StreamingTV', 'StreamingMovies', 'PaperlessBilling']

for col in boolean_columns:
    df[col] = df[col].map({'Yes': True, 'No': False})
```

2. Convert City, State, County, Area, Timezone, Job, Education, Employment, Marital, Gender, Contract, InternetService, and PaymentMethod columns to categories.

Code

```
category_columns = ['City', 'State', 'County', 'Area', 'Timezone', 'Job',
                    'Education', 'Employment', 'Marital', 'Gender',
'Contract', 'InternetService', 'PaymentMethod']

for col in category_columns:
    df[col] = df[col].astype('category')
```

3. Convert Zip Code to a string and add leading zeros so each value is 5 digits in length.

Code
`df['Zip'] = df['Zip'].astype(str).str.zfill(5)`

4. Convert CaseOrder, Population, Children, Age, Email, and Contacts to integers.

Code

```
integer_columns = ['CaseOrder', 'Population', 'Children', 'Age', 'Email',
'Contacts']
df[integer_columns] = df[integer_columns].fillna(0).astype(int)
```

5. Convert negative values in the Outage_sec_perweek column to absolute value

Code

```
df['Outage_sec_perweek'] = df['Outage_sec_perweek'].abs()
```

6. Impute Children, Age, Income, Tenure, and Bandwidth_GB_Year with averages.

Code

```
mean_impute_cols = ['Children', 'Age', 'Income', 'Tenure',
'Bandwidth_GB_Year']

for col in mean_impute_cols:
    df[col].fillna(df[col].mean(), inplace=True)
```

7. Impute Techie, Phone, TechSupport with most common values.

Code

```
mode_impute_cols = ['Techie', 'Phone', 'TechSupport']

for col in mode_impute_cols:
    df[col].fillna(df[col].mode()[0], inplace=True)
```

5. Provide a copy of the cleaned data set as a CSV file.

See churn_cleaned.csv.

6. Summarize the limitations of the data-cleaning process.

The limitations of this data-cleaning process are as follows:

- **Imputation bias:** Filling missing values with averages or modes can introduce bias into the dataset.
- **Inaccurate assumptions:** Converting negative outage times to absolute values might not apply universally. In some cases, negative values might indicate specific data points (e.g., refunds or negative billing).

7. Discuss how the limitations summarized in part D6 could affect the analysis of the question or decision from part A.

The analysis of my research question, “Which customer factors have the greatest impact on a customer's decision to churn?” may be affected by the assumption that negative outage times are the result of a data entry error and are intended to be positive. This will affect the average outage time for customers grouped by their churn status.

E. Apply principal component analysis (PCA) to identify the significant features of the data set by doing the following:

1. Identify the total number of principal components and provide the output of the principal components loading matrix.

The variables used for principal component analysis were all of the numerical variables: Population, Children, Age, Income, Tenure, Outage_sec_perweek, MonthlyCharge, Bandwidth_GB_Year, item1, item2, item3, item4, item5, item6, item7, and item8.

Shown below is the code used to generate the loading matrix along with the loading matrix itself.

Code

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

# Numerical columns
numerical_columns = ["Population", "Children", "Age", "Income", "Tenure",
"Outage_sec_perweek", "MonthlyCharge", "Bandwidth_GB_Year", "item1",
"item2", "item3", "item4", "item5", "item6", "item7", "item8"]
```

```
numerical_data = df[numerical_columns].fillna(0)

# Standardize the data
scaler = StandardScaler()
scaled_data = scaler.fit_transform(numerical_data)

# Initialize and fit PCA
n_components = 16
pca = PCA(n_components=n_components)
pca.fit(scaled_data)

# Create the loading matrix as a DataFrame
loading_matrix = pd.DataFrame(
    pca.components_,
    columns=numerical_columns,
    index=[f"PC{i + 1}" for i in range(n_components)]
)

# Transpose to have principal components as columns and features as rows
transposed_loading_matrix = loading_matrix.T
print(transposed_loading_matrix)
```

Result

	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	PC9	PC10	PC11	PC12	PC13	PC14	PC15	PC16
Population	-0.002187	0.000905	0.015631	-0.056449	-0.496185	0.157321	0.790092	-0.311578	-0.021384	-0.056142	0.018351	-0.017404	0.001238	-0.005661	-0.002368	-0.000858
Children	-0.000315	-0.003816	0.007443	0.036606	0.658063	0.061918	0.115011	-0.738345	-0.001347	0.033165	-0.035065	-0.017612	0.011844	0.012843	-0.011067	-0.016110
Age	-0.001116	-0.020808	0.003405	-0.023816	-0.347046	0.720894	-0.488779	-0.326847	0.111623	0.011214	0.017274	-0.004656	-0.007102	0.001133	0.002408	0.012654
Income	-0.000842	0.007394	0.025236	0.008565	0.431122	0.669633	0.333610	0.496404	-0.059572	0.007337	-0.058878	-0.002057	0.000354	0.012147	0.004949	0.001383
Tenure	-0.010662	0.701426	-0.069741	-0.058802	-0.009652	0.012199	-0.011598	-0.001016	0.038292	0.002778	-0.009977	0.007617	-0.012061	0.007023	0.003550	-0.705231
Outage_sec_perweek	-0.013164	0.018255	-0.048206	0.701817	-0.000445	-0.011558	0.079407	0.039001	0.696505	-0.103913	0.012359	-0.013566	-0.016624	0.010258	-0.005175	0.000497
MonthlyCharge	-0.000571	0.043757	-0.024204	0.702347	-0.087278	0.040171	-0.045689	-0.042710	-0.697468	0.037863	0.009191	-0.013759	0.000443	0.021151	-0.012794	-0.047798
Bandwidth_GB_Year	-0.012406	0.702967	-0.072416	-0.009964	0.004980	0.002751	-0.002531	-0.015955	-0.010953	0.012052	0.002886	0.003265	-0.001896	-0.008417	0.008381	0.706836
item1	0.458928	0.031107	0.280033	0.031005	-0.001500	-0.008449	-0.008671	-0.011413	-0.001994	-0.071750	-0.118594	-0.047075	0.025531	-0.239828	0.793182	-0.004298
item2	0.434154	0.042309	0.281573	0.019953	-0.003602	-0.001186	-0.015546	0.001994	-0.000127	-0.108503	-0.169470	-0.066194	0.074137	-0.591882	-0.573652	-0.002511
item3	0.400741	0.034367	0.280753	-0.012042	-0.017661	-0.018950	-0.018972	-0.003151	-0.005045	-0.175566	-0.251419	-0.146383	-0.396382	0.673564	-0.178022	0.015195
item4	0.145667	-0.049390	-0.567283	-0.032898	-0.023587	-0.001978	-0.015405	0.002975	-0.017541	-0.174449	-0.479689	-0.443782	0.432083	0.088517	0.018012	0.001990
item5	-0.175499	0.065343	0.586383	0.027648	-0.018742	-0.013914	-0.009477	0.012018	0.034736	0.132196	0.062174	-0.209112	0.693511	0.265532	-0.042357	-0.002656
item6	0.405090	-0.011981	-0.183640	0.004045	0.001838	0.013172	0.011061	-0.010021	0.000438	-0.064512	0.065471	0.757894	0.402645	0.229280	-0.063804	0.001923
item7	0.358327	-0.003469	-0.180819	-0.031737	0.059479	0.023694	0.002416	0.017912	-0.021475	-0.171509	0.806059	-0.378569	0.066408	0.066272	-0.041115	-0.006827
item8	0.308769	-0.016871	-0.132046	0.032614	-0.043982	-0.007209	0.051590	0.014157	0.091430	0.924096	-0.018513	-0.111831	-0.043921	0.046342	-0.042210	-0.002842

[2.94784283 1.8997551 1.63686462 1.13110437 1.0245581 1.00379203

2. Justify the reduced number of the principal components and include a screenshot of a scree plot.

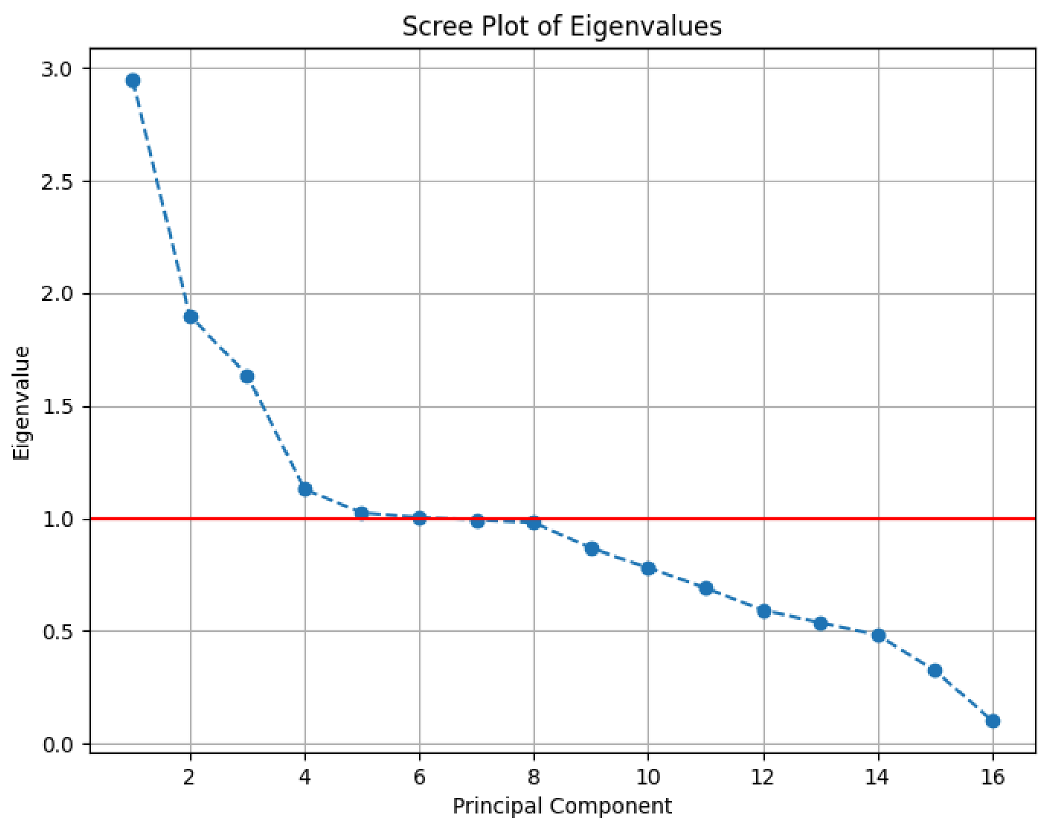
Code

```
# Calculate eigenvalues
eigenvalues = pca.explained_variance_

# Plot the eigenvalues
plt.figure(figsize=(8, 6))
plt.plot(range(1, len(eigenvalues) + 1), eigenvalues, marker="o",
linestyle="--")
```

```
plt.title("Scree Plot of Eigenvalues")
plt.xlabel("Principal Component")
plt.ylabel("Eigenvalue")
plt.axhline(y=1, color="red")
plt.grid(True)
plt.show()
```

Result



Code

```
eigenvalues
```

Result

```
[2.94784283 1.8997551  1.63686462 1.13110437 1.0245581  1.00379203
 0.99369793 0.98261636 0.8684206  0.78029542 0.69121166 0.59248071
 0.5382088  0.48242352 0.32508315 0.10324495]
```

Based on the scree plot and raw eigenvalues, we can determine that principal components 1 through 6 should be used as they have an eigenvalue greater than 1. Principal components 7-16 have an eigenvalue that is less than 1 and should be discarded. Silva et al. (2020) acknowledged, "A frequent yet open issue that arises from supervised-based problems is how many PCA axes are required."

3. Describe how the organization would benefit from the use of PCA.

The greatest benefit of PCA for an organization is dimensionality reduction. PCA reduces the number of variables in a dataset, making it easier for stakeholders to visualize data trends and grasp concepts more easily. Richardson (2009) noted, "Principal Component Analysis (PCA) is the general name for a technique which uses sophisticated underlying mathematical principles to transform... variables into a smaller number of principal components." Furthermore, PCA provides a compressed representation of the data, which reduces storage needs. Lastly, PCA filters out random noise by emphasizing only the significant components, leading to clearer insights and more reliable analysis. Richardson (2009) claimed, "PCA successfully reduced the dimensionality of our data set down from 17 to 2."

G. Acknowledge web sources, using in-text citations and references, for segments of third-party code used to support the application. Be sure the web sources are reliable.

WGU Course Material was used for Python implementation of PCA.

Scikit-Learn documentation was referenced for identification of principal components.

H. Acknowledge sources, using in-text citations and references, for content that is quoted, paraphrased, or summarized.

Dean, S., & Illowsky, B. (2009). Descriptive Statistics: Histogram. Retrieved from the Connexions Web site: http://cnx. org/content/m16298/1.11.

Richardson, M. (2009). Principal component analysis. URL: http://people. maths. ox. ac. uk/richardsonm/SignalProcPCA. pdf (last access: 3.5. 2013). Aleš Hladnik Dr., Ass. Prof., Chair of Information and Graphic Arts Technology, Faculty of Natural Sciences and Engineering, University of Ljubljana, Slovenia ales. hladnik@ ntf. uni-lj. si, 6(16), 4.

Silva, R. B., Oliveira, D. D., Santos, D. P. D., Santos, L. F. D., Wilson, R. E., & Bêdo, M. V. N. (2020). Criteria for choosing the number of dimensions in a principal component analysis: An empirical assessment. Anais.