D206_Performance_Assessment_revision

July 11, 2021

1 Performance Assessment | D206 Data Cleaning

Ryan L. Buchanan Student ID: 001826691 Masters Data Analytics (12/01/2020) Program Mentor: Dan Estes (385) 432-9281 (MST) rbuch49@wgu.edu

1.1 Part I: Research Question

1.1.1 A. Question or Decision:

Can we determine which individual customers are at high risk of churn? And, can we determine which features are most significant to churn?

1.1.2 B. Required Variables:

The data set is 10,000 customer records of a popular telecommunications company. The dependent variable (target) in question is whether or not each customer has continued or discontinued service within the last month. This column is titled "Churn."
Independent variables or predictors that may lead to identifying a relationship with the dependent variable of "Churn" within the dataset include: 1. Services that each customer signed up for (for example, multiple phone lines, technical support add-ons or streaming media) 2. Customer account information (customers' tenure with the company, payment methods, bandwidth usage, etc.) 3. Customer demographics (gender, marital status, income, etc.).
4. Finally, there are eight independent variables that represent responses customer-perceived importance of company services and features.

The data is both numerical (as in the yearly GB bandwidth usage; customer annual income) and categorical (a "Yes" or "No" for Churn; customer job).

1.2 Part II: Data-Cleaning Plan

1.2.1 C1. Plan to Find Anomalies:

My approach will include: 1. Back up my data and the process I am following as a copy to my machine and, since this is a manageable dataset, to GitHub using command line and gitbash. 2. Read the data set into Python using Pandas' read_csv command. 3. Evaluate the data struture to better understand input data. 4. Naming the dataset as a the variable "churn_df" and subsequent useful slices of the dataframe as "df". 5. Examine potential misspellings, awkward variable naming & missing data. 6. Find outliers that may create or hide statistical significance using histograms. 7. Imputing records missing data with meaningful measures of central tendency (mean, median or mode) or simply remove outliers that are several standard deviations above the mean. *Referenced & paraphrased within above plan (Larose, 2019, p. 29-43).

1.2.2 C2. Justification of Approach:

Though the data seems to be inexplicably missing quite a bit of data (such as the many NAs in customer tenure with the company) from apparently random columns, this approach seems like a good first approach in order to put the data in better working order without needing to involve methods of initial data collection or querying the data-gatherers on reasons for missing information. Also, this the first dataset that I've clean, so I followed the procedures practice in the performance lab as well as tips from StackOverflow and other tutorial resources.

1.2.3 C3. Justification of Tools:

I will use the Python programming language as I have a bit of a background in Python having studied machine learning independently over the last year before beginning this masters program and its ability to perform many things right "out of the box" (Poulson, 2016, section 2). Python provides clean, intuitive and readable syntax that has become ubiquitous across in the data science industry. Also, I find the Jupyter notebooks a convenient way to run code visually, in its attractive single document markdown format, the ability to display results of code and graphic visualizations and provide crystal-clear running documentation for future reference. A thorough installation and importation of Python packages and libraries will provide specially designed code to perfom complex data science tasks rather than personally building them from scratch. This will include: NumPy - to work with arrays Pandas - to load datasets Matplotlib - to plot charts Scikit-learn - for machine learning model classes SciPy - for mathematical problems, specifically linear algebra transformations Seaborn - for high-level interface and atttractive visualizations

A quick, precise example of loading a dataset and creating a variable efficiently is using to call the Pandas library and its subsequent "read_csv" function in order to manipulate our data as a dataframe: import pandas as pd df = pd.read_csv('Data.csv')

1.2.4 C4. Provide the Code:

[1]:
```
# Install necessary packages
!pip install pandas
!pip install numpy
!pip install scipy
!pip install sklearn
!pip install matplotlib
```

```
Requirement already satisfied: pandas in /usr/local/lib/python3.7/dist-packages
(1.1.5)
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.7/dist-
packages (from pandas) (2018.9)
Requirement already satisfied: python-dateutil>=2.7.3 in
/usr/local/lib/python3.7/dist-packages (from pandas) (2.8.1)
Requirement already satisfied: numpy>=1.15.4 in /usr/local/lib/python3.7/dist-
packages (from pandas) (1.19.5)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-
```

```
packages (from python-dateutil>=2.7.3->pandas) (1.15.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages
(1.19.5)
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-packages
(1.4.1)
Requirement already satisfied: numpy>=1.13.3 in /usr/local/lib/python3.7/dist-
packages (from scipy) (1.19.5)
Requirement already satisfied: sklearn in /usr/local/lib/python3.7/dist-packages
(0.0)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.7/dist-
packages (from sklearn) (0.22.2.post1)
Requirement already satisfied: numpy>=1.11.0 in /usr/local/lib/python3.7/dist-
packages (from scikit-learn->sklearn) (1.19.5)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-
packages (from scikit-learn->sklearn) (1.0.1)
Requirement already satisfied: scipy>=0.17.0 in /usr/local/lib/python3.7/dist-
packages (from scikit-learn->sklearn) (1.4.1)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.7/dist-
packages (3.2.2)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7
/dist-packages (from matplotlib) (1.3.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-
packages (from matplotlib) (0.10.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in
/usr/local/lib/python3.7/dist-packages (from matplotlib) (2.4.7)
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.7
/dist-packages (from matplotlib) (2.8.1)
Requirement already satisfied: numpy>=1.11 in /usr/local/lib/python3.7/dist-
packages (from matplotlib) (1.19.5)
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages
(from cycler>=0.10->matplotlib) (1.15.0)
```

[2]:
```
# Standard imports
import numpy as np
import pandas as pd
from sklearn.preprocessing import scale
from sklearn.decomposition import PCA

import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

[3]:
```
# Load data set into Pandas dataframe
churn_df = pd.read_csv('churn_raw_data.csv')
```

[4]:
```
# Display Churn dataframe
churn_df
```

[4]:
```
      Unnamed: 0  CaseOrder Customer_id  ... item6 item7 item8
0              1          1     K409198  ...     4     3     4
1              2          2     S120509  ...     3     4     4
2              3          3     K191035  ...     3     3     3
3              4          4      D90850  ...     4     3     3
4              5          5     K662701  ...     4     4     5
...          ...        ...         ...  ...   ...   ...   ...
9995        9996       9996     M324793  ...     3     2     3
9996        9997       9997     D861732  ...     5     2     5
9997        9998       9998     I243405  ...     4     4     5
9998        9999       9999     I641617  ...     3     5     4
9999       10000      10000      T38070  ...     3     4     1

[10000 rows x 52 columns]
```

[5]:
```python
# List of Dataframe Columns
df = churn_df.columns
print(df)
```

```
Index(['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'],
      dtype='object')
```

[6]:
```python
# Remove redundant "Unnamed" column at beginning & display first five records
df = churn_df.drop(churn_df.columns[0], axis = 1)
df.head()
```

[6]:
```
   CaseOrder Customer_id  ... item7 item8
0          1     K409198  ...     3     4
1          2     S120509  ...     4     4
2          3     K191035  ...     3     3
3          4      D90850  ...     3     3
4          5     K662701  ...     4     5

[5 rows x 51 columns]
```

[7]:
```python
# Rename last 8 survey columns for better description of variables
df.rename(columns = {'item1':'Responses',
                    'item2':'Fixes',
                     'item3':'Replacements',
                     'item4':'Reliability',
```

```
                      'item5':'Options',
                      'item6':'Respectfulness',
                      'item7':'Courteous',
                      'item8':'Listening'},
           inplace=True)
```

[8]:
```
# Find number of records and columns of dataset
df.shape
```

[8]:
```
(10000, 51)
```

[9]:
```
# Describe Churn dataset statistics
df.describe()
```

[9]:
```
         CaseOrder           Zip  ...     Courteous     Listening
count  10000.00000  10000.000000  ...  10000.000000  10000.000000
mean    5000.50000  49153.319600  ...      3.509500      3.495600
std     2886.89568  27532.196108  ...      1.028502      1.028633
min        1.00000    601.000000  ...      1.000000      1.000000
25%     2500.75000  26292.500000  ...      3.000000      3.000000
50%     5000.50000  48869.500000  ...      4.000000      3.000000
75%     7500.25000  71866.500000  ...      4.000000      4.000000
max    10000.00000  99929.000000  ...      7.000000      8.000000

[8 rows x 23 columns]
```

[10]:
```
# Remove less meaningful variables from statistics description
df_stats = df.drop(columns=['CaseOrder', 'Zip', 'Lat', 'Lng'])
df_stats.describe()
```

[10]:
```
          Population     Children  ...     Courteous     Listening
count   10000.000000  7505.000000  ...  10000.000000  10000.000000
mean     9756.562400     2.095936  ...      3.509500      3.495600
std     14432.698671     2.154758  ...      1.028502      1.028633
min         0.000000     0.000000  ...      1.000000      1.000000
25%       738.000000     0.000000  ...      3.000000      3.000000
50%      2910.500000     1.000000  ...      4.000000      3.000000
75%     13168.000000     3.000000  ...      4.000000      4.000000
max    111850.000000    10.000000  ...      7.000000      8.000000

[8 rows x 19 columns]
```

[11]:
```
# Calculate Churn Rate
df.Churn.value_counts() / len(df)
```

[11]:
```
No     0.735
Yes    0.265
Name: Churn, dtype: float64
```

[12]:
```
# Review data types (numerical => "int64" & "float64"; & categorical =>␣
 ↪"object") in data set
```

```
df.dtypes
```

[12]:
```
CaseOrder                   int64
Customer_id                object
Interaction                object
City                       object
State                      object
County                     object
Zip                         int64
Lat                       float64
Lng                       float64
Population                  int64
Area                       object
Timezone                   object
Job                        object
Children                  float64
Age                       float64
Education                  object
Employment                 object
Income                    float64
Marital                    object
Gender                     object
Churn                      object
Outage_sec_perweek        float64
Email                       int64
Contacts                    int64
Yearly_equip_failure        int64
Techie                     object
Contract                   object
Port_modem                 object
Tablet                     object
InternetService            object
Phone                      object
Multiple                   object
OnlineSecurity             object
OnlineBackup               object
DeviceProtection           object
TechSupport                object
StreamingTV                object
StreamingMovies            object
PaperlessBilling           object
PaymentMethod              object
Tenure                    float64
MonthlyCharge             float64
Bandwidth_GB_Year         float64
Responses                   int64
Fixes                       int64
```

```
Replacements             int64
Reliability              int64
Options                  int64
Respectfulness           int64
Courteous                int64
Listening                int64
dtype: object
```

[13]:
```
# Re-validate column data types and missing values
df.columns.to_series().groupby(df.dtypes).groups
```

[13]:
```
{int64: ['CaseOrder', 'Zip', 'Population', 'Email', 'Contacts',
'Yearly_equip_failure', 'Responses', 'Fixes', 'Replacements', 'Reliability',
'Options', 'Respectfulness', 'Courteous', 'Listening'], float64: ['Lat', 'Lng',
'Children', 'Age', 'Income', 'Outage_sec_perweek', 'Tenure', 'MonthlyCharge',
'Bandwidth_GB_Year'], object: ['Customer_id', 'Interaction', 'City', 'State',
'County', 'Area', 'Timezone', 'Job', 'Education', 'Employment', 'Marital',
'Gender', 'Churn', 'Techie', 'Contract', 'Port_modem', 'Tablet',
'InternetService', 'Phone', 'Multiple', 'OnlineSecurity', 'OnlineBackup',
'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovies',
'PaperlessBilling', 'PaymentMethod']}
```

[14]:
```
# Find missing values
df.isnull()
```

[14]:
```
      CaseOrder  Customer_id  Interaction  ...  Respectfulness  Courteous
Listening
0         False        False        False  ...           False      False
False
1         False        False        False  ...           False      False
False
2         False        False        False  ...           False      False
False
3         False        False        False  ...           False      False
False
4         False        False        False  ...           False      False
False
...         ...          ...          ...  ...             ...        ...
...
9995      False        False        False  ...           False      False
False
9996      False        False        False  ...           False      False
False
9997      False        False        False  ...           False      False
False
9998      False        False        False  ...           False      False
False
9999      False        False        False  ...           False      False
False
```

```
[10000 rows x 51 columns]
```

[15]:
```
# Access only rows from dataframe containing missing values
df.isnull().any(axis=1)
```

[15]:
```
0        True
1       False
2        True
3       False
4       False
        ...
9995     True
9996     True
9997     True
9998    False
9999     True
Length: 10000, dtype: bool
```

[16]:
```
# Woah, lots of empty fields!  Immediately noticeable as "True" in columns of␣
 ↪"Children", "Age", "Income", "Techie", "Phone", "Tenure"
# Display the specific columns with NAs
df.isna().any()
```

[16]:
```
CaseOrder               False
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                 True
Age                      True
Education               False
Employment              False
Income                   True
Marital                 False
Gender                  False
Churn                   False
Outage_sec_perweek      False
Email                   False
Contacts                False
Yearly_equip_failure    False
```

```
Techie                   True
Contract                False
Port_modem              False
Tablet                  False
InternetService         False
Phone                    True
Multiple                False
OnlineSecurity          False
OnlineBackup            False
DeviceProtection        False
TechSupport              True
StreamingTV             False
StreamingMovies         False
PaperlessBilling        False
PaymentMethod           False
Tenure                   True
MonthlyCharge           False
Bandwidth_GB_Year        True
Responses               False
Fixes                   False
Replacements            False
Reliability             False
Options                 False
Respectfulness          False
Courteous               False
Listening               False
dtype: bool
```

[17]:
```
# Confirm missing observations numbers
data_nulls = df.isnull().sum()
print(data_nulls)
```

```
CaseOrder                  0
Customer_id                0
Interaction                0
City                       0
State                      0
County                     0
Zip                        0
Lat                        0
Lng                        0
Population                 0
Area                       0
Timezone                   0
Job                        0
Children                2495
Age                     2475
Education                  0
```

```
Employment                0
Income                 2490
Marital                   0
Gender                    0
Churn                     0
Outage_sec_perweek        0
Email                     0
Contacts                  0
Yearly_equip_failure      0
Techie                 2477
Contract                  0
Port_modem                0
Tablet                    0
InternetService           0
Phone                  1026
Multiple                  0
OnlineSecurity            0
OnlineBackup              0
DeviceProtection          0
TechSupport             991
StreamingTV               0
StreamingMovies           0
PaperlessBilling          0
PaymentMethod             0
Tenure                  931
MonthlyCharge             0
Bandwidth_GB_Year      1021
Responses                 0
Fixes                     0
Replacements              0
Reliability               0
Options                   0
Respectfulness            0
Courteous                 0
Listening                 0
dtype: int64
```

[18]:
```
# Store rows with missing values in a new variable
rows_with_missing_values = df.isnull().any(axis=1)
df[rows_with_missing_values]
```

[18]:
```
      CaseOrder Customer_id  ... Courteous Listening
0             1     K409198  ...         3         4
2             3     K191035  ...         3         3
5             6     W303516  ...         3         3
6             7     U335188  ...         5         5
7             8     V538685  ...         4         5
...         ...         ...  ...       ...       ...
```

```
9994       9995   P175475  ...          4          4
9995       9996   M324793  ...          2          3
9996       9997   D861732  ...          2          5
9997       9998   I243405  ...          4          5
9999      10000    T38070  ...          4          1

[7867 rows x 51 columns]
```

[19]:
```python
# Examine columns for misspellings in categorical variables using unique()␣
↪method
df['Employment'].unique()
```

[19]:
```
array(['Part Time', 'Retired', 'Student', 'Full Time', 'Unemployed'],
      dtype=object)
```

[20]:
```python
df['Area'].unique()
```

[20]:
```
array(['Urban', 'Suburban', 'Rural'], dtype=object)
```

[21]:
```python
df['Timezone'].unique()
```

[21]:
```
array(['America/Sitka', 'America/Detroit', 'America/Los_Angeles',
       'America/Chicago', 'America/New_York', 'America/Puerto_Rico',
       'America/Denver', 'America/Menominee', 'America/Phoenix',
       'America/Indiana/Indianapolis', 'America/Boise',
       'America/Kentucky/Louisville', 'Pacific/Honolulu',
       'America/Indiana/Petersburg', 'America/Nome', 'America/Anchorage',
       'America/Indiana/Knox', 'America/Juneau', 'America/Toronto',
       'America/Indiana/Winamac', 'America/Indiana/Vincennes',
       'America/North_Dakota/New_Salem', 'America/Indiana/Tell_City',
       'America/Indiana/Marengo', 'America/Ojinaga'], dtype=object)
```

[22]:
```python
# Well then, how many unique jobs are there and will this variable help us out␣
↪much?
len(df['Job'].unique())
```

[22]:
```
639
```

[23]:
```python
df['Children'].unique()
```

[23]:
```
array([nan,  1.,  4.,  0.,  3.,  2.,  7.,  5.,  9.,  6., 10.,  8.])
```

[24]:
```python
df['Age'].unique()
```

[24]:
```
array([68., 27., 50., 48., 83., nan, 49., 86., 23., 56., 30., 39., 63.,
       60., 61., 52., 75., 77., 47., 70., 69., 45., 40., 82., 26., 25.,
       66., 72., 41., 44., 43., 84., 59., 31., 51., 58., 73., 33., 42.,
       81., 87., 54., 67., 46., 24., 20., 71., 32., 29., 80., 53., 79.,
       65., 35., 34., 74., 55., 76., 57., 38., 78., 19., 36., 88., 62.,
       37., 28., 22., 85., 89., 18., 21., 64.])
```

[25]:
```python
# Examine age range
age_range = df['Age'].unique()
```

```
print(sorted(age_range))
```

```
[23.0, 25.0, 26.0, 27.0, 30.0, 31.0, 39.0, 40.0, 41.0, 43.0, 44.0, 45.0, 47.0,
48.0, 49.0, 50.0, 51.0, 52.0, 59.0, 61.0, 68.0, 83.0, nan, 18.0, 19.0, 20.0,
21.0, 22.0, 24.0, 28.0, 29.0, 32.0, 33.0, 34.0, 35.0, 36.0, 37.0, 38.0, 42.0,
46.0, 53.0, 54.0, 55.0, 56.0, 57.0, 58.0, 60.0, 62.0, 63.0, 64.0, 65.0, 66.0,
67.0, 69.0, 70.0, 71.0, 72.0, 73.0, 74.0, 75.0, 76.0, 77.0, 78.0, 79.0, 80.0,
81.0, 82.0, 84.0, 85.0, 86.0, 87.0, 88.0, 89.0]
```

[26]:
```
df['Education'].unique()
```

[26]:
```
array(["Master's Degree", 'Regular High School Diploma',
       'Doctorate Degree', 'No Schooling Completed', "Associate's Degree",
       "Bachelor's Degree", 'Some College, Less than 1 Year',
       'GED or Alternative Credential',
       'Some College, 1 or More Years, No Degree',
       '9th Grade to 12th Grade, No Diploma',
       'Nursery School to 8th Grade', 'Professional School Degree'],
      dtype=object)
```

[27]:
```
df['Employment'].unique()
```

[27]:
```
array(['Part Time', 'Retired', 'Student', 'Full Time', 'Unemployed'],
      dtype=object)
```

[28]:
```
df['Marital'].unique()
```

[28]:
```
array(['Widowed', 'Married', 'Separated', 'Never Married', 'Divorced'],
      dtype=object)
```

[29]:
```
df['Gender'].unique()
```

[29]:
```
array(['Male', 'Female', 'Prefer not to answer'], dtype=object)
```

[30]:
```
df['Contract'].unique()
```

[30]:
```
array(['One year', 'Month-to-month', 'Two Year'], dtype=object)
```

[31]:
```
df['PaymentMethod'].unique()
```

[31]:
```
array(['Credit Card (automatic)', 'Bank Transfer(automatic)',
       'Mailed Check', 'Electronic Check'], dtype=object)
```

[32]:
```
# Display any duplicate rows in the dataframe.
data_duplicates = df.loc[df.duplicated()]
print(data_duplicates)
```

```
Empty DataFrame
Columns: [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, Responses, Fixes, Replacements, Reliability,
Options, Respectfulness, Courteous, Listening]
Index: []
```

[33]:
```python
# Identify the standard deviation of every numeric column in the dataset
data_std = df_stats.std()
print(data_std)
```

```
Population              14432.698671
Children                    2.154758
Age                        20.753928
Income                  28358.469482
Outage_sec_perweek          7.025921
Email                       3.025898
Contacts                    0.988466
Yearly_equip_failure        0.635953
Tenure                     26.438904
MonthlyCharge              43.335473
Bandwidth_GB_Year        2187.396807
Responses                   1.037797
Fixes                       1.034641
Replacements                1.027977
Reliability                 1.025816
Options                     1.024819
Respectfulness              1.033586
Courteous                   1.028502
Listening                   1.028633
dtype: float64
```

[34]:
```python
data_nulls = df_stats.isnull().sum()
print(data_nulls)
```

```
Customer_id                0
Interaction                0
City                       0
State                      0
County                     0
Population                 0
Area                       0
Timezone                   0
Job                        0
Children                2495
Age                     2475
Education                  0
Employment                 0
Income                  2490
```

```
Marital                    0
Gender                     0
Churn                      0
Outage_sec_perweek         0
Email                      0
Contacts                   0
Yearly_equip_failure       0
Techie                  2477
Contract                   0
Port_modem                 0
Tablet                     0
InternetService            0
Phone                   1026
Multiple                   0
OnlineSecurity             0
OnlineBackup               0
DeviceProtection           0
TechSupport              991
StreamingTV                0
StreamingMovies            0
PaperlessBilling           0
PaymentMethod              0
Tenure                   931
MonthlyCharge              0
Bandwidth_GB_Year       1021
Responses                  0
Fixes                      0
Replacements               0
Reliability                0
Options                    0
Respectfulness             0
Courteous                  0
Listening                  0
dtype: int64
```

[35]:

```
# Impute missing fields for variables Children, Age, Income, Tenure and␣
↪Bandwidth_GB_Year with median or mean
df_stats['Children'] = df['Children'].fillna(df['Children'].median())
df_stats['Age'] = df['Age'].fillna(df['Age'].median())
df_stats['Income'] = df['Income'].fillna(df['Income'].median())
df_stats['Tenure'] = df['Tenure'].fillna(df['Tenure'].median())
df_stats['Bandwidth_GB_Year'] = df['Bandwidth_GB_Year'].
↪fillna(df['Bandwidth_GB_Year'].median())
```

[36]:

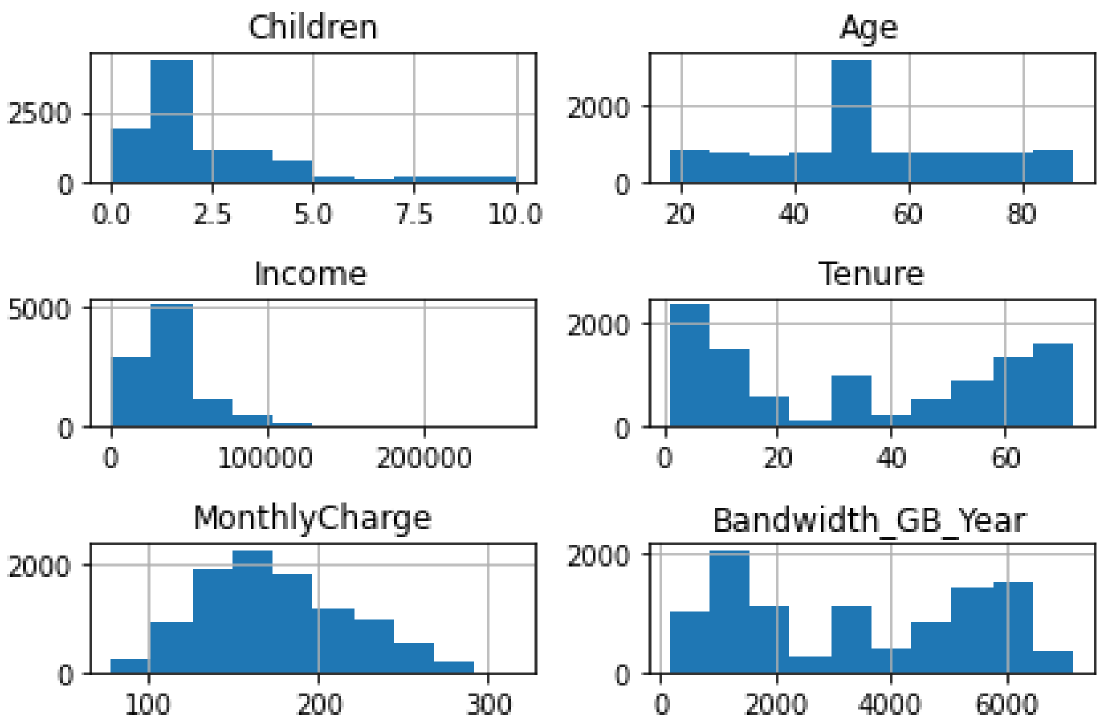
```
data_nulls = df_stats.isnull().sum()
print(data_nulls)
```

```
Customer_id                0
```

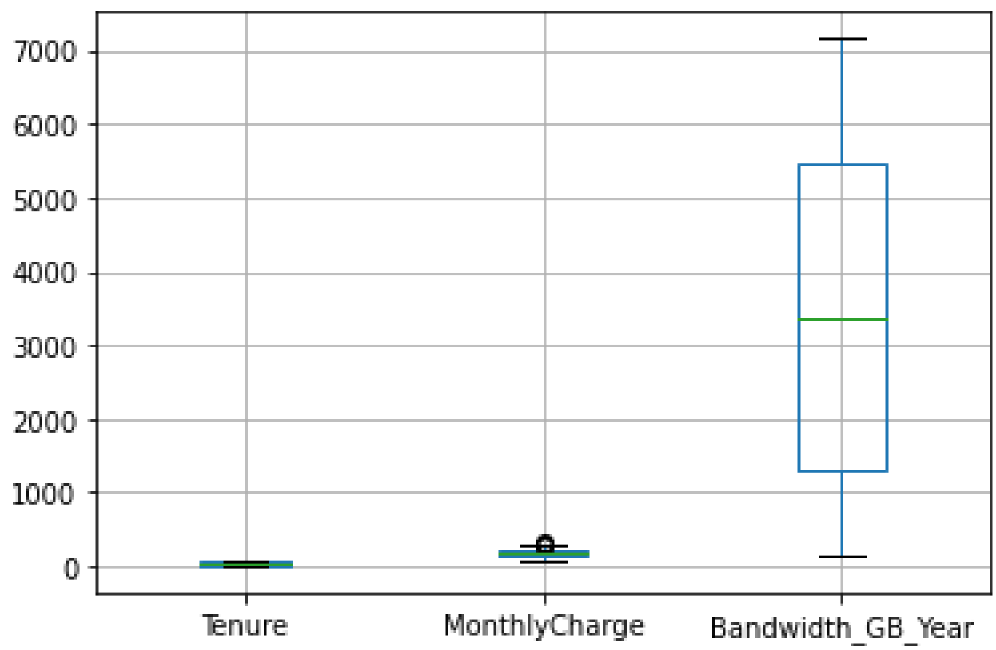
```
Interaction                 0
City                        0
State                       0
County                      0
Population                  0
Area                        0
Timezone                    0
Job                         0
Children                    0
Age                         0
Education                   0
Employment                  0
Income                      0
Marital                     0
Gender                      0
Churn                       0
Outage_sec_perweek          0
Email                       0
Contacts                    0
Yearly_equip_failure        0
Techie                   2477
Contract                    0
Port_modem                  0
Tablet                      0
InternetService             0
Phone                    1026
Multiple                    0
OnlineSecurity              0
OnlineBackup                0
DeviceProtection            0
TechSupport               991
StreamingTV                 0
StreamingMovies             0
PaperlessBilling            0
PaymentMethod               0
Tenure                      0
MonthlyCharge               0
Bandwidth_GB_Year           0
Responses                   0
Fixes                       0
Replacements                0
Reliability                 0
Options                     0
Respectfulness              0
Courteous                   0
Listening                   0
dtype: int64
```

1.3 Anomaly Detection & Data Visualization

[37]:
```
# Create histograms of important variables
df_stats[['Children', 'Age', 'Income', 'Tenure', 'MonthlyCharge',␣
 ↪'Bandwidth_GB_Year']].hist()
plt.savefig('churn_pyplot.jpg')
plt.tight_layout()
# plt.close()
```

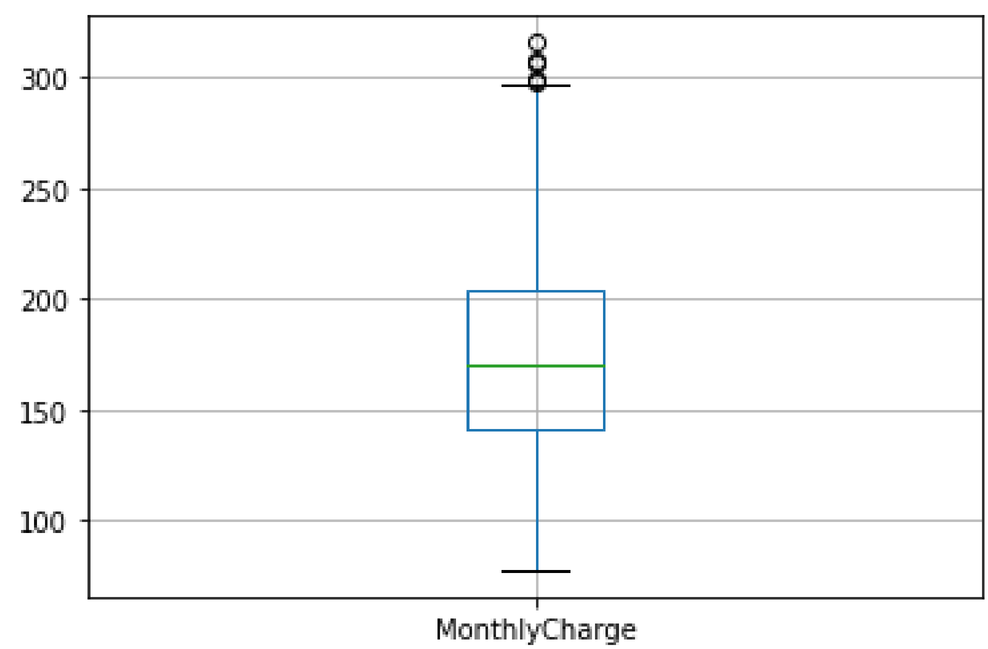


[38]:
```
# Some odd distributions here, let's see some box plots for outliers
# Create a boxplot of user duration, payment & usage variables
df_stats.boxplot(['Tenure', 'MonthlyCharge', 'Bandwidth_GB_Year'])
plt.savefig('churn_boxplots.jpg')
```



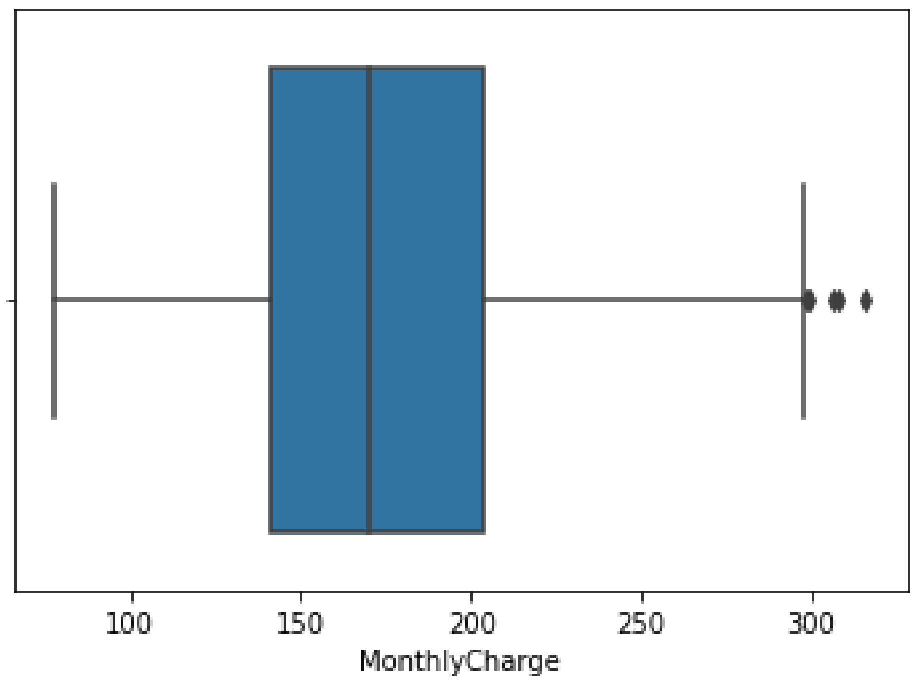
[39]:
```
# Let's see monthly charge separately
df_stats.boxplot(['MonthlyCharge'])
# Definitely outliers but not sure how that effects PCA down the line
```

[39]: <matplotlib.axes._subplots.AxesSubplot at 0x7f51d10c7f90>



[40]:
```
# Let's see a Seaborn boxplot fee & bandwidth
sns.boxplot('MonthlyCharge', data = df_stats)
plt.show()
# Definitely outliers but not sure what to do with them
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning:
Pass the following variable as a keyword arg: x. From version 0.12, the only
valid positional argument will be `data`, and passing other arguments without an
explicit keyword will result in an error or misinterpretation.
  FutureWarning
```



1.4 Extract Clean dataset to 'churn_clean.csv'

[41]:
```
# Extract Clean dataset
df_stats.to_csv('churn_clean.csv')
```

[42]:
```
# Reload cleaned data & remove all variable except user services payment info␣
↪and survey data
churn_user = pd.read_csv('churn_clean.csv')
```

[43]:
```
# Slice off all but last eleven service realted variables
data = churn_user.loc[:, 'Tenure':'Listening']
data.head()
```

[43]:
```
      Tenure  MonthlyCharge  ...  Courteous  Listening
0   6.795513     171.449762  ...          3          4
1   1.156681     242.948015  ...          4          4
2  15.754144     159.440398  ...          3          3
3  17.087227     120.249493  ...          3          3
4   1.670972     150.761216  ...          4          5

[5 rows x 11 columns]
```

1.5 PCA (D206 Performance Lab)

[44]:
```python
# Import Scikit Learn PCA application
from sklearn.decomposition import PCA
```

[45]:
```python
# Normalize the data
churn_normalized = (data - data.mean()) / data.std()
```

[46]:
```python
# Select number of components to extract
pca = PCA(n_components = data.shape[1])
```

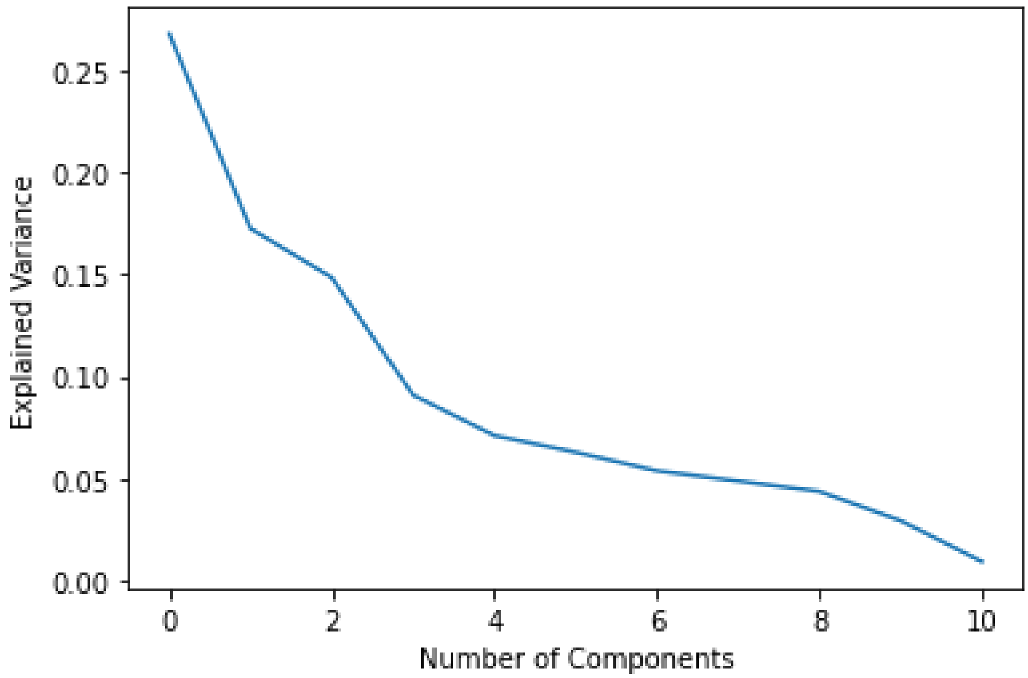
[47]:
```python
# Create a list of PCA names
churn_numeric = data[['Tenure', 'MonthlyCharge', 'Bandwidth_GB_Year',␣
 ↪'Responses',
                      'Fixes', 'Replacements', 'Reliability', 'Options',
                      'Respectfulness', 'Courteous', 'Listening']]
pcs_names = []
for i, col in enumerate(churn_numeric.columns):
    pcs_names.append('PC' + str(i + 1))
print(pcs_names)
```

```
['PC1', 'PC2', 'PC3', 'PC4', 'PC5', 'PC6', 'PC7', 'PC8', 'PC9', 'PC10', 'PC11']
```

[48]:
```python
# Call PCA application & convert the dataset of 11 variables into a dataset of␣
 ↪11 components
pca.fit(churn_normalized)
churn_pca = pd.DataFrame(pca.transform(churn_normalized),
                        columns = pcs_names)
```

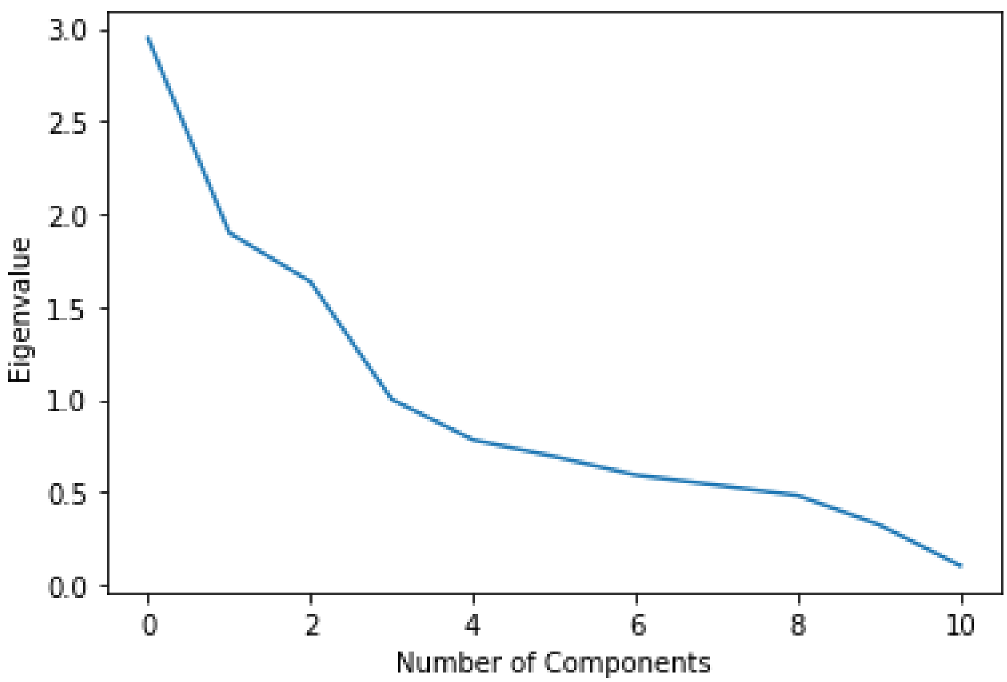
[49]:
```python
# For a scree plot import matplotlib & seaborn libraries
import matplotlib.pyplot as plt
import seaborn as sns
```

[50]:
```python
# Run the scree plot
plt.plot(pca.explained_variance_ratio_)
plt.xlabel('Number of Components')
plt.ylabel('Explained Variance')
plt.show();
```



[51]:
```
# Extract the eigenvalues
cov_matrix = np.dot(churn_normalized.T, churn_normalized) / data.shape[0]
eigenvalues = [np.dot(eigenvector.T, np.dot(cov_matrix, eigenvector)) for␣
 ↪eigenvector in pca.components_]
```

[52]:
```
# Plot the eigenvalues
plt.plot(eigenvalues)
plt.xlabel('Number of Components')
plt.ylabel('Eigenvalue')
plt.show();
```



[53]:
```
# Select the fewest components
for pc, var in zip(pcs_names, np.cumsum(pca.explained_variance_ratio_)):
    print(pc, var)
```

```
PC1 0.2679266062396307
PC2 0.44053732880229357
PC3 0.5891444522248594
PC4 0.6801579842425081
PC5 0.7513217054372495
PC6 0.8143117094101315
PC7 0.8681970531672314
PC8 0.9171387362588826
PC9 0.9610122820002185
PC10 0.9905736363144829
PC11 0.9999999999999998
```

[54]:
```
# Above, we see that 86% of variance is explained by 7 components
# Create a rotation
rotation = pd.DataFrame(pca.components_.T, columns = pcs_names, index =␣
 ↪churn_numeric.columns)
print(rotation)
```

```
                            PC1       PC2       PC3  ...       PC9      PC10
PC11
Tenure                -0.010403  0.701838 -0.072209  ...  0.006935  0.003286
```

```
-0.705445
MonthlyCharge      0.000317  0.041147 -0.014151  ...  0.023690 -0.013785
-0.047865
Bandwidth_GB_Year -0.012166  0.703079 -0.074222  ... -0.008068  0.008529
0.706925
Responses          0.458932  0.031325  0.281154  ... -0.240574  0.793237
-0.004306
Fixes              0.434134  0.042559  0.282404  ... -0.592131 -0.573832
-0.002217
Replacements       0.400639  0.034665  0.281118  ...  0.673088 -0.177665
0.014933
Reliability        0.145799 -0.050367 -0.567815  ...  0.087207  0.018301
0.002283
Options           -0.175633  0.066334  0.587335  ...  0.265474 -0.042012
-0.002514
Respectfulness     0.405207 -0.012680 -0.183447  ...  0.230319 -0.063972
0.001604
Courteous          0.358342 -0.003886 -0.181697  ...  0.067293 -0.040946
-0.006875
Listening          0.308925 -0.017396 -0.131173  ...  0.046107 -0.042251
-0.002357

[11 rows x 11 columns]
```

[55]:
```python
# Output loadings for components
loadings = pd.DataFrame(pca.components_.T,
                        columns = pcs_names,
                        index = data.columns)
loadings
```

[55]:
```
                        PC1       PC2       PC3  ...       PC9      PC10
PC11
Tenure            -0.010403  0.701838 -0.072209  ...  0.006935  0.003286
-0.705445
MonthlyCharge      0.000317  0.041147 -0.014151  ...  0.023690 -0.013785
-0.047865
Bandwidth_GB_Year -0.012166  0.703079 -0.074222  ... -0.008068  0.008529
0.706925
Responses          0.458932  0.031325  0.281154  ... -0.240574  0.793237
-0.004306
Fixes              0.434134  0.042559  0.282404  ... -0.592131 -0.573832
-0.002217
Replacements       0.400639  0.034665  0.281118  ...  0.673088 -0.177665
0.014933
Reliability        0.145799 -0.050367 -0.567815  ...  0.087207  0.018301
0.002283
Options           -0.175633  0.066334  0.587335  ...  0.265474 -0.042012
-0.002514
```

```
Respectfulness     0.405207 -0.012680 -0.183447  ...  0.230319 -0.063972
0.001604
Courteous          0.358342 -0.003886 -0.181697  ...  0.067293 -0.040946
-0.006875
Listening          0.308925 -0.017396 -0.131173  ...  0.046107 -0.042251
-0.002357

[11 rows x 11 columns]
```

[56]:
```python
# Finally, extract reduced dataset & print 3 components
churn_reduced = churn_pca.iloc[ : , 0:3]
print(churn_reduced)
```

```
           PC1       PC2       PC3
0     1.923875 -1.421955  1.903125
1    -0.199798 -1.706801  0.538766
2    -0.667923 -0.985940  0.227390
3     0.046465 -0.730628  2.282040
4     1.326741 -1.924880  0.825729
...        ...       ...       ...
9995 -2.097964  1.961837  0.104147
9996  1.917485  1.645946  0.611009
9997  1.431918  0.323573  0.028288
9998  2.011460  2.187756 -0.079864
9999 -2.266364  1.591986 -0.819973

[10000 rows x 3 columns]
```

1.6 Part III: Data Cleaning

1.6.1 D. Data Cleaning Summary

D1.Cleaning Findings: There was much missing data with meaningful variable fields including Children, Age, Income, Tenure and Bandwidth_GB_Year. Given mean and variance of these variables, it seemed reasonable to impute missing values with median values. Many categorical (such as whether or not the customer was "Techie") & non-numeric (columns for customer ID numbers & related customer transaction IDs) data were not included in analysis given they seemed less meaningful to interpretation and decision-making. The anomalies discovered were not significant & were mitigated as follows.

D2.Justification of Mitigation Methods: Mitigated missing values with imputation using median values. MonthlyCharge variable shows outliers so left alone. This does not seem significant to this analysis.

D3.Summary of Outcomes: Cleaned dataset to leave remaining variables describing customer tenure, monthly service charge, yearly bandwidth usage & responses to survey. It seems pretty straightforward at this point.

D4.Mitigation Code: (see code above and Panopto recording)

D5.Clean Data: (see attached file 'churn_clean.csv')

D6.Limitations: Limitations given the telecom company data set are that the data are not coming from a warehouse. In this scenario, it is as though I initiated and gathered the data. So, I am not able to reach out to the staff that organized & gathered this information to ask them why certain NAs are there, why are fields such as age or yearly bandwidth usage missing information that might be relevant to answering questions about customer retention or churn. In a real world project, you would be able to go down to the department where these folks worked and fill in the empty fields or discover why fields are left blank.

D7. The accurate factual data for missing fields that might be recoverable given the ability to access the staff in the data warehouse, such as company tenure with the company, may give a slighty different overall picture to the analysis. While imputation may provide a path to move forward and give decision-makers reasonable answers, there really is no reason for these data to be missing. The limitations here could be remedied by instituting stricter data acquisition procedures, follow-ups & feedback.

1.6.2 E. PCA Application

E1. Principal Components: The principal components, and what I determined to be "most important", in this dataset include survey responses to:

```
<li>Timely Responses</li>
<li>Timely Fixes</li>
<li>Timely Replacements</li>
<li>Respectful Response</li>
```

E2. Criteria Used: Intuition about customer service suggests that feedback from user survey might offer the most important components when analyzing churn rate. Also, since survey results were the easiest to select as numeric predictors of whether or not a user would leave the company I included the 8 responses as variables for the PCA. And, of course, users' tenure with the company as well as monthly charge & yearly GB use are seem like significant numeric data points for analysis. I used a scree plot & extracted the eigenvalues for visualization of where the "elbow was bending". The elbow bent at about 3 but kept an eigenvalue above 1 until the tenth component. Then, the fewest number of components were selected based on the 86% explanation at 7 components using the Numpy cumsum method. A rotation & loadings were created which suggested the "most important" features of the dataset.

E3. Benefits: The loadings suggest the variables involved in timely action with regard to customer satisfaction (Responses, Fixes, Replacements & Respectfulness) should be given greater emphasis and hopefully help reduce the churn rate from the large number of 27% & "increase the retention period of customers" by targeting more resources in the direction prompt customer service (Ahmad, 2019, p. 1). Again, this seems an intuitive result but now decision-makers in the company of reasonable verification of what might have been a "hunch".

1.7 Part IV: Supporting Documents

1.7.1 F. Video

https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=9c94329b-551f-44de-a598-ad1d013eecff

1.7.2 G. Sources for Third-Party Code

Cmdline. (2018, March 20). How To Change Column Names and Row Indexes in Pandas? Python and R Tips. https://cmdlinetips.com/2018/03/how-to-change-column-names-and-row-indexes-in-pandas/

Larose, C. D. & Larose, D. T. (2019). Data Science: Using Python and R. John Wiley & Sons, Inc.

Poulson, B. (2016). Data Science Foundations: Data Mining LinkedIn Learning. https://www.linkedin.com/learning/data-science-foundations-data-mining/anomaly-detection-in-python?u=2045532

Sree. (2020, October 26). Predict Customer Churn in Python. Towards Data Science. https://towardsdatascience.com/predict-customer-churn-in-python-e8cd6d3aaa7

VanderPlas, J. (2017). Python Data Science Handbook: Essential Tools for Working with Data. O'Reilly Media, Inc.

1.7.3 H. Sources

Ahmad, A. K., Jafar, A & Aljoumaa, K. (2019, March 20). Customer churn prediction in telecom using machine learning in big data platform. Journal of Big Data. https://journalofbigdata.springeropen.com/articles/10.1186/s40537-019-0191-6

Altexsoft. (2019, March 27). Customer Churn Prediction Using Machine Learning: Main Approaches and Models. Altexsoft. https://www.altexsoft.com/blog/business/customer-churn-prediction-for-subscription-businesses-using-machine-learning-main-approaches-and-models/

Frohbose, F. (2020, November 24). Machine Learning Case Study: Telco Customer Churn Prediction. Towards Data Science. https://towardsdatascience.com/machine-learning-case-study-telco-customer-churn-prediction-bc4be03c9e1d

Mountain, A. (2014, August 11). Data Cleaning. Better Evaluation. https://www.betterevaluation.org/en/evaluation-options/data_cleaning

[57]:
```
!wget -nc https://raw.githubusercontent.com/brpy/colab-pdf/master/colab_pdf.py
from colab_pdf import colab_pdf
colab_pdf('D206_Performance_Assessment.ipynb')
```

```
--2021-07-11 21:41:09--  https://raw.githubusercontent.com/brpy/colab-
pdf/master/colab_pdf.py
Resolving raw.githubusercontent.com (raw.githubusercontent.com)...
185.199.108.133, 185.199.109.133, 185.199.110.133, ...
Connecting to raw.githubusercontent.com
(raw.githubusercontent.com)|185.199.108.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1864 (1.8K) [text/plain]
Saving to: colab_pdf.py

colab_pdf.py        100%[===================>]   1.82K  --.-KB/s    in 0s

2021-07-11 21:41:10 (11.6 MB/s) - colab_pdf.py saved [1864/1864]
```

```
␣
↪---------------------------------------------------------------------------

    ValueError                                Traceback (most recent call␣
↪last)

    <ipython-input-57-8848f4a78340> in <module>()
      1 get_ipython().system('wget -nc https://raw.githubusercontent.com/
↪brpy/colab-pdf/master/colab_pdf.py')
      2 from colab_pdf import colab_pdf
----> 3 colab_pdf('D206_Performance_Assessment.ipynb')


    /content/colab_pdf.py in colab_pdf(file_name, notebookpath)
     20     # Check if the notebook exists in the Drive.
     21     if not os.path.isfile(os.path.join(notebookpath, file_name)):
---> 22         raise ValueError(f"file '{file_name}' not found in path␣
↪'{notebookpath}'.")
     23
     24     # Installing all the recommended packages.


    ValueError: file 'D206_Performance_Assessment.ipynb' not found in path '/
↪content/drive/MyDrive/Colab Notebooks/'.
```