WGU D208 Predictive Modeling

Task 1 - Linear Regression

Ednalyn C. De Dios

August 6, 2023

A1. Research Question

> ***Can we predict how much the customer will pay in monthly charges?***

A2. Goals

The organization will benefit from knowing how much a customer might pay in monthly charges. This will inform the decisions of stakeholders in matters where customer lifetime value needs to be taken into account. For example, a marketing campaign might target customers who are not spending enough money to be upsold with other telecom services.

B1. Summary of Assumptions

According to Statology.org, assumptions of a multiple linear regression model include the following:

- A linear relationship between the independent and dependent variables
- The independent variables do not have multicollinearity
- Independence of observations
- Residuals have a constant variance at every point in the linear model (homoscedasticity)
- The residuals of the model are normally distributed

B2. Tool Benefits

Jupyter Notebooks and the Python programming language will be used in this analysis. I chose to program in Python because it is very readable. It ranks among the most popular languages worldwide because it's powerful, flexible, and easy to use. (Geeksforgeeks.org, 2023) Moreover, the Python community is active (Geeksforgeeks.org, 2023) and the language sports a vast system of mature packages for data science and machine learning.

B3. Appropriate Technique

The target variable, MonthlyCharge, is a continuous variable and hence multiple linear regression is the right tool to analyze if we can predict howw much a customer will pay in monthly charges. In addition, the data set has several good candidates of explanatory variables that will inform our predictions. We will determine if the independent variables have a positive or negative relationship to the target variable. This can perhaps affect the organization's decisions on marketing segmentation.

C1. Data Cleaning

Our goal for cleaning the data set is to have a dataframe free of duplicates, missing values, outliers, and irrelevant variables. To do so, we will execute the following goals and steps:

1. Find and remove duplicates.
2. Handle missing values.
3. Remove outliers where necessary.
4. Drop irrelevant features.

In [1]:
```
# setting the random seed for reproducibility
import random
random.seed(493)

# for manipulating dataframes
import pandas as pd
import numpy as np

# for visualizations
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
sns.set(style="whitegrid")

# for modeling
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.feature_selection import RFE
import statsmodels.api as sm
from sklearn.metrics import r2_score

# to print out all the outputs
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

# set display options
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
pd.set_option('display.max_colwidth', None)
```

In [2]:
```
# read the csv file
```

```
df = pd.read_csv('churn_clean.csv')
```

Find and remove duplicates

In [3]:
```
# select rows that are duplicated based on all columns. Any records after the first
dup = df[df.duplicated()]

# find out how many rows are duplicated
dup.shape
```

Out[3]:
```
(0, 50)
```

Handle missing values

In [4]:
```
def show_missing(df):
    """
    Takes a dataframe and returns a dataframe with stats
    on missing and null values with their percentages.
    """
    null_count = df.isnull().sum()
    null_percentage = (null_count / df.shape[0]) * 100
    empty_count = pd.Series(((df == ' ') | (df == '')).sum())
    empty_percentage = (empty_count / df.shape[0]) * 100
    nan_count = pd.Series(((df == 'nan') | (df == 'NaN')).sum())
    nan_percentage = (nan_count / df.shape[0]) * 100
    dfx = pd.DataFrame({'num_missing': null_count, 'missing_percentage': null_perce
                        'num_empty': empty_count, 'empty_percentage': empty_percen
                        'nan_count': nan_count, 'nan_percentage': nan_percentage})
    return dfx

show_missing(df)
```

Out[4]:

	num_missing	missing_percentage	num_empty	empty_percentage	na
CaseOrder	0	0.00	0	0.0	
Customer_id	0	0.00	0	0.0	
Interaction	0	0.00	0	0.0	
UID	0	0.00	0	0.0	
City	0	0.00	0	0.0	
State	0	0.00	0	0.0	
County	0	0.00	0	0.0	
Zip	0	0.00	0	0.0	
Lat	0	0.00	0	0.0	
Lng	0	0.00	0	0.0	
Population	0	0.00	0	0.0	
Area	0	0.00	0	0.0	
TimeZone	0	0.00	0	0.0	
Job	0	0.00	0	0.0	
Children	0	0.00	0	0.0	
Age	0	0.00	0	0.0	
Income	0	0.00	0	0.0	
Marital	0	0.00	0	0.0	
Gender	0	0.00	0	0.0	
Churn	0	0.00	0	0.0	
Outage_sec_perweek	0	0.00	0	0.0	
Email	0	0.00	0	0.0	
Contacts	0	0.00	0	0.0	
Yearly_equip_failure	0	0.00	0	0.0	
Techie	0	0.00	0	0.0	
Contract	0	0.00	0	0.0	
Port_modem	0	0.00	0	0.0	
Tablet	0	0.00	0	0.0	
InternetService	2129	21.29	0	0.0	
Phone	0	0.00	0	0.0	

	num_missing	missing_percentage	num_empty	empty_percentage	na
Multiple	0	0.00	0	0.0	
OnlineSecurity	0	0.00	0	0.0	
OnlineBackup	0	0.00	0	0.0	
DeviceProtection	0	0.00	0	0.0	
TechSupport	0	0.00	0	0.0	
StreamingTV	0	0.00	0	0.0	
StreamingMovies	0	0.00	0	0.0	
PaperlessBilling	0	0.00	0	0.0	
PaymentMethod	0	0.00	0	0.0	
Tenure	0	0.00	0	0.0	
MonthlyCharge	0	0.00	0	0.0	
Bandwidth_GB_Year	0	0.00	0	0.0	
Item1	0	0.00	0	0.0	
Item2	0	0.00	0	0.0	
Item3	0	0.00	0	0.0	
Item4	0	0.00	0	0.0	
Item5	0	0.00	0	0.0	
Item6	0	0.00	0	0.0	
Item7	0	0.00	0	0.0	
Item8	0	0.00	0	0.0	

In [5]:
```
# count of values in the column
df['InternetService'].value_counts()
```

Out[5]:
```
InternetService
Fiber Optic    4408
DSL            3463
Name: count, dtype: int64
```

In [6]:
```
# fill missing values with None as in no service
df = df.fillna("None")

df['InternetService'].value_counts()
show_missing(df)
```

Out[6]:
```
InternetService
Fiber Optic    4408
DSL            3463
None           2129
Name: count, dtype: int64
```

Out[6]:

	num_missing	missing_percentage	num_empty	empty_percentage	na
CaseOrder	0	0.0	0	0.0	
Customer_id	0	0.0	0	0.0	
Interaction	0	0.0	0	0.0	
UID	0	0.0	0	0.0	
City	0	0.0	0	0.0	
State	0	0.0	0	0.0	
County	0	0.0	0	0.0	
Zip	0	0.0	0	0.0	
Lat	0	0.0	0	0.0	
Lng	0	0.0	0	0.0	
Population	0	0.0	0	0.0	
Area	0	0.0	0	0.0	
TimeZone	0	0.0	0	0.0	
Job	0	0.0	0	0.0	
Children	0	0.0	0	0.0	
Age	0	0.0	0	0.0	
Income	0	0.0	0	0.0	
Marital	0	0.0	0	0.0	
Gender	0	0.0	0	0.0	
Churn	0	0.0	0	0.0	
Outage_sec_perweek	0	0.0	0	0.0	
Email	0	0.0	0	0.0	
Contacts	0	0.0	0	0.0	
Yearly_equip_failure	0	0.0	0	0.0	
Techie	0	0.0	0	0.0	
Contract	0	0.0	0	0.0	
Port_modem	0	0.0	0	0.0	
Tablet	0	0.0	0	0.0	
InternetService	0	0.0	0	0.0	
Phone	0	0.0	0	0.0	

	num_missing	missing_percentage	num_empty	empty_percentage	na
Multiple	0	0.0	0	0.0	
OnlineSecurity	0	0.0	0	0.0	
OnlineBackup	0	0.0	0	0.0	
DeviceProtection	0	0.0	0	0.0	
TechSupport	0	0.0	0	0.0	
StreamingTV	0	0.0	0	0.0	
StreamingMovies	0	0.0	0	0.0	
PaperlessBilling	0	0.0	0	0.0	
PaymentMethod	0	0.0	0	0.0	
Tenure	0	0.0	0	0.0	
MonthlyCharge	0	0.0	0	0.0	
Bandwidth_GB_Year	0	0.0	0	0.0	
Item1	0	0.0	0	0.0	
Item2	0	0.0	0	0.0	
Item3	0	0.0	0	0.0	
Item4	0	0.0	0	0.0	
Item5	0	0.0	0	0.0	
Item6	0	0.0	0	0.0	
Item7	0	0.0	0	0.0	
Item8	0	0.0	0	0.0	

Remove outliers where necessary

In [7]:
```
df.info()
```

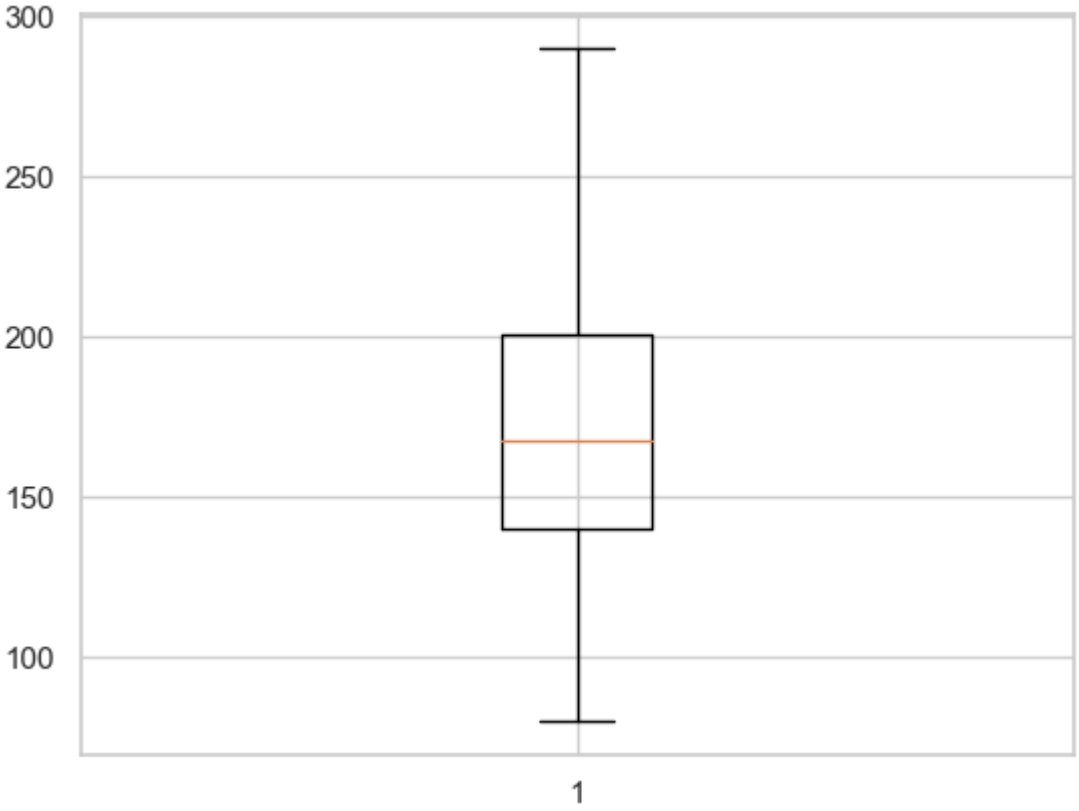
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 50 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   CaseOrder             10000 non-null  int64
 1   Customer_id           10000 non-null  object
 2   Interaction           10000 non-null  object
 3   UID                   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              10000 non-null  int64
 15  Age                   10000 non-null  int64
 16  Income                10000 non-null  float64
 17  Marital               10000 non-null  object
 18  Gender                10000 non-null  object
 19  Churn                 10000 non-null  object
 20  Outage_sec_perweek    10000 non-null  float64
 21  Email                 10000 non-null  int64
 22  Contacts              10000 non-null  int64
 23  Yearly_equip_failure  10000 non-null  int64
 24  Techie                10000 non-null  object
 25  Contract              10000 non-null  object
 26  Port_modem            10000 non-null  object
 27  Tablet                10000 non-null  object
 28  InternetService       10000 non-null  object
 29  Phone                 10000 non-null  object
 30  Multiple              10000 non-null  object
 31  OnlineSecurity        10000 non-null  object
 32  OnlineBackup          10000 non-null  object
 33  DeviceProtection      10000 non-null  object
 34  TechSupport           10000 non-null  object
 35  StreamingTV           10000 non-null  object
 36  StreamingMovies       10000 non-null  object
 37  PaperlessBilling      10000 non-null  object
 38  PaymentMethod         10000 non-null  object
 39  Tenure                10000 non-null  float64
 40  MonthlyCharge         10000 non-null  float64
 41  Bandwidth_GB_Year     10000 non-null  float64
 42  Item1                 10000 non-null  int64
 43  Item2                 10000 non-null  int64
 44  Item3                 10000 non-null  int64
 45  Item4                 10000 non-null  int64
 46  Item5                 10000 non-null  int64
 47  Item6                 10000 non-null  int64
 48  Item7                 10000 non-null  int64
 49  Item8                 10000 non-null  int64
```

```
dtypes: float64(7), int64(16), object(27)
memory usage: 3.8+ MB
```

Remove outliers

In [8]:
```
# visualize the distribution of column values
plt.boxplot(df['MonthlyCharge'])
fig = plt.figure(figsize =(10, 7))
```

Out[8]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x23ff2425f10>,
  <matplotlib.lines.Line2D at 0x23ff24461f0>],
 'caps': [<matplotlib.lines.Line2D at 0x23ff2446490>,
  <matplotlib.lines.Line2D at 0x23ff2446730>],
 'boxes': [<matplotlib.lines.Line2D at 0x23ff2416b50>],
 'medians': [<matplotlib.lines.Line2D at 0x23ff24469d0>],
 'fliers': [<matplotlib.lines.Line2D at 0x23ff2446c70>],
 'means': []}
```




```
<Figure size 1000x700 with 0 Axes>
```

Drop irrelevant features

In [9]:
```
# drop columns
df.drop(columns=['CaseOrder', 'Customer_id', 'Interaction', 'UID', 'City', 'State',
                 'Zip', 'Lat', 'Lng', 'Population', 'Area', 'TimeZone', 'Job',
                 'Churn', 'Income', 'PaperlessBilling', 'PaymentMethod',
                 'Item1', 'Item2', 'Item3', 'Item4', 'Item5',
                 'Item6', 'Item7', 'Item8'], inplace=True)
```

C2. Summary Statistics

In [10]:
```
df.describe()
```

Out[10]:

	Children	Age	Outage_sec_perweek	Email	Contacts	Yearly_e
count	10000.0000	10000.000000	10000.000000	10000.000000	10000.000000	1
mean	2.0877	53.078400	10.001848	12.016000	0.994200	
std	2.1472	20.698882	2.976019	3.025898	0.988466	
min	0.0000	18.000000	0.099747	1.000000	0.000000	
25%	0.0000	35.000000	8.018214	10.000000	0.000000	
50%	1.0000	53.000000	10.018560	12.000000	1.000000	
75%	3.0000	71.000000	11.969485	14.000000	2.000000	
max	10.0000	89.000000	21.207230	23.000000	7.000000	

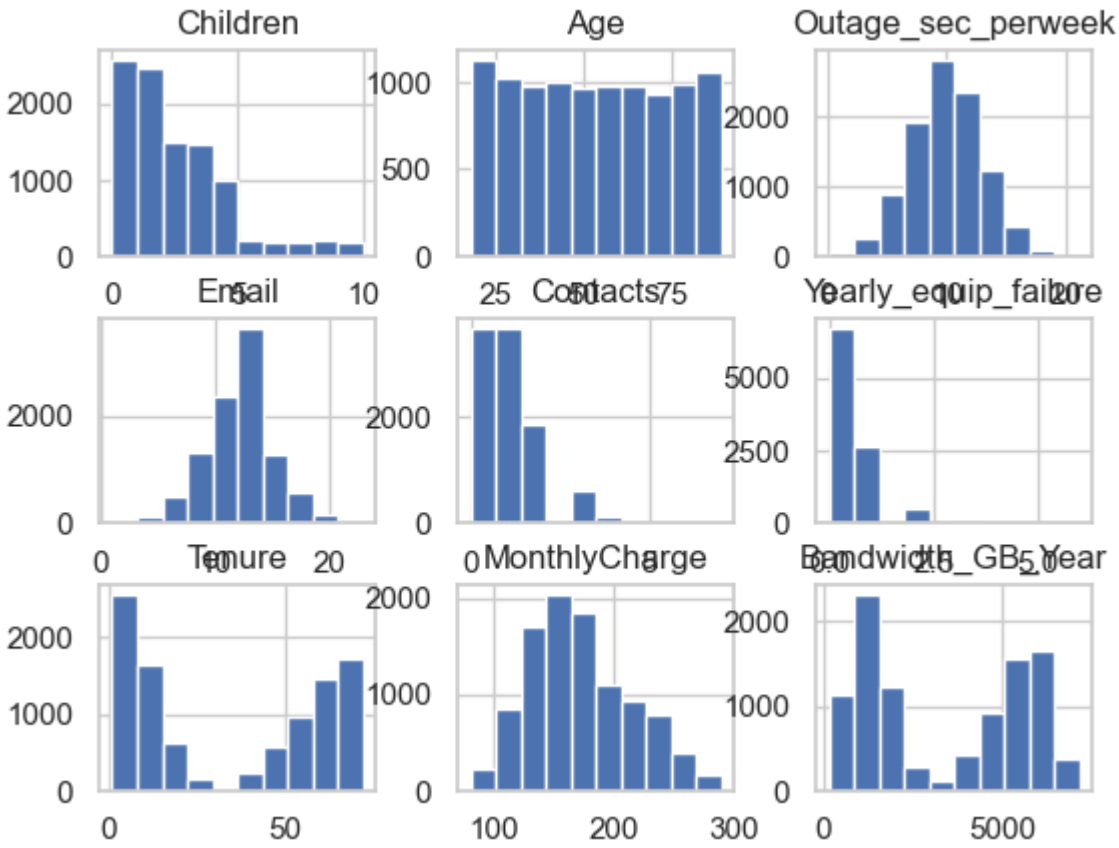
C3. Visualizations

In [11]:
```
df.columns
```

Out[11]:
```
Index(['Children', 'Age', 'Marital', 'Gender', 'Outage_sec_perweek', 'Email',
       'Contacts', 'Yearly_equip_failure', 'Techie', 'Contract', 'Port_modem',
       'Tablet', 'InternetService', 'Phone', 'Multiple', 'OnlineSecurity',
       'OnlineBackup', 'DeviceProtection', 'TechSupport', 'StreamingTV',
       'StreamingMovies', 'Tenure', 'MonthlyCharge', 'Bandwidth_GB_Year'],
      dtype='object')
```

In [12]:
```
# make historgrams and save the plot
df[['Children',
    'Age',
    'Outage_sec_perweek',
    'Email',
    'Contacts',
    'Yearly_equip_failure',
    'Tenure',
    'MonthlyCharge',
    'Bandwidth_GB_Year'
   ]].hist()
plt.savefig('churn_univariate_hist.jpg')
```

Out[12]:
```
array([[<Axes: title={'center': 'Children'}>,
        <Axes: title={'center': 'Age'}>,
        <Axes: title={'center': 'Outage_sec_perweek'}>],
       [<Axes: title={'center': 'Email'}>,
        <Axes: title={'center': 'Contacts'}>,
        <Axes: title={'center': 'Yearly_equip_failure'}>],
       [<Axes: title={'center': 'Tenure'}>,
        <Axes: title={'center': 'MonthlyCharge'}>,
        <Axes: title={'center': 'Bandwidth_GB_Year'}>]], dtype=object)
```



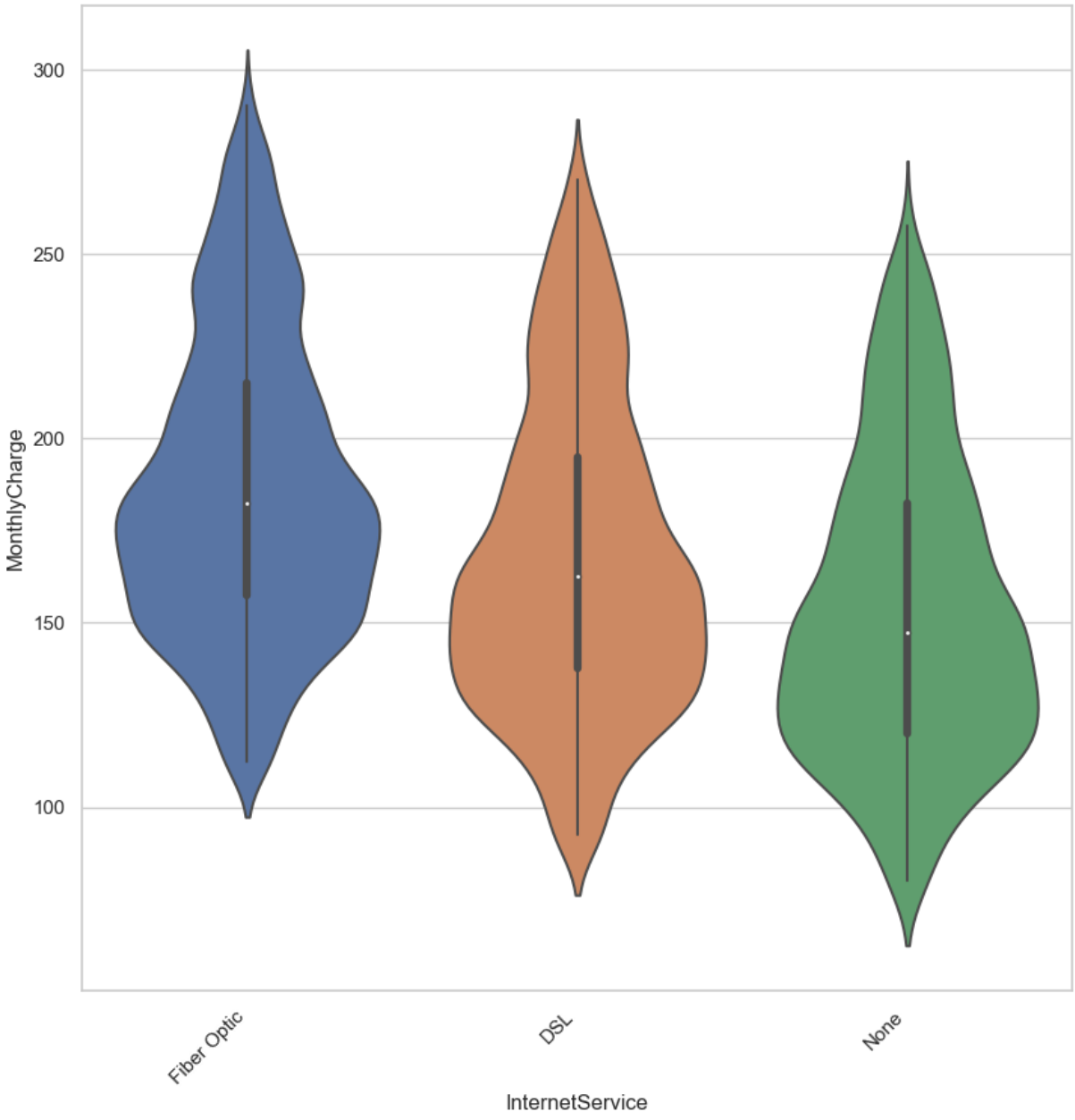
In [13]:
```
df.columns
```

Out[13]:
```
Index(['Children', 'Age', 'Marital', 'Gender', 'Outage_sec_perweek', 'Email',
       'Contacts', 'Yearly_equip_failure', 'Techie', 'Contract', 'Port_modem',
       'Tablet', 'InternetService', 'Phone', 'Multiple', 'OnlineSecurity',
       'OnlineBackup', 'DeviceProtection', 'TechSupport', 'StreamingTV',
       'StreamingMovies', 'Tenure', 'MonthlyCharge', 'Bandwidth_GB_Year'],
      dtype='object')
```

In [14]:
```
# make violin plot and save
plt.figure(figsize=(10,10))
ax = sns.violinplot(x="InternetService", y="MonthlyCharge", data=df)
ax.set_xticklabels(ax.get_xticklabels(), rotation=45, horizontalalignment='right')
plt.savefig('churn_bivariate_internetservice.jpg')
```

Out[14]:
```
<Figure size 1000x1000 with 0 Axes>
```

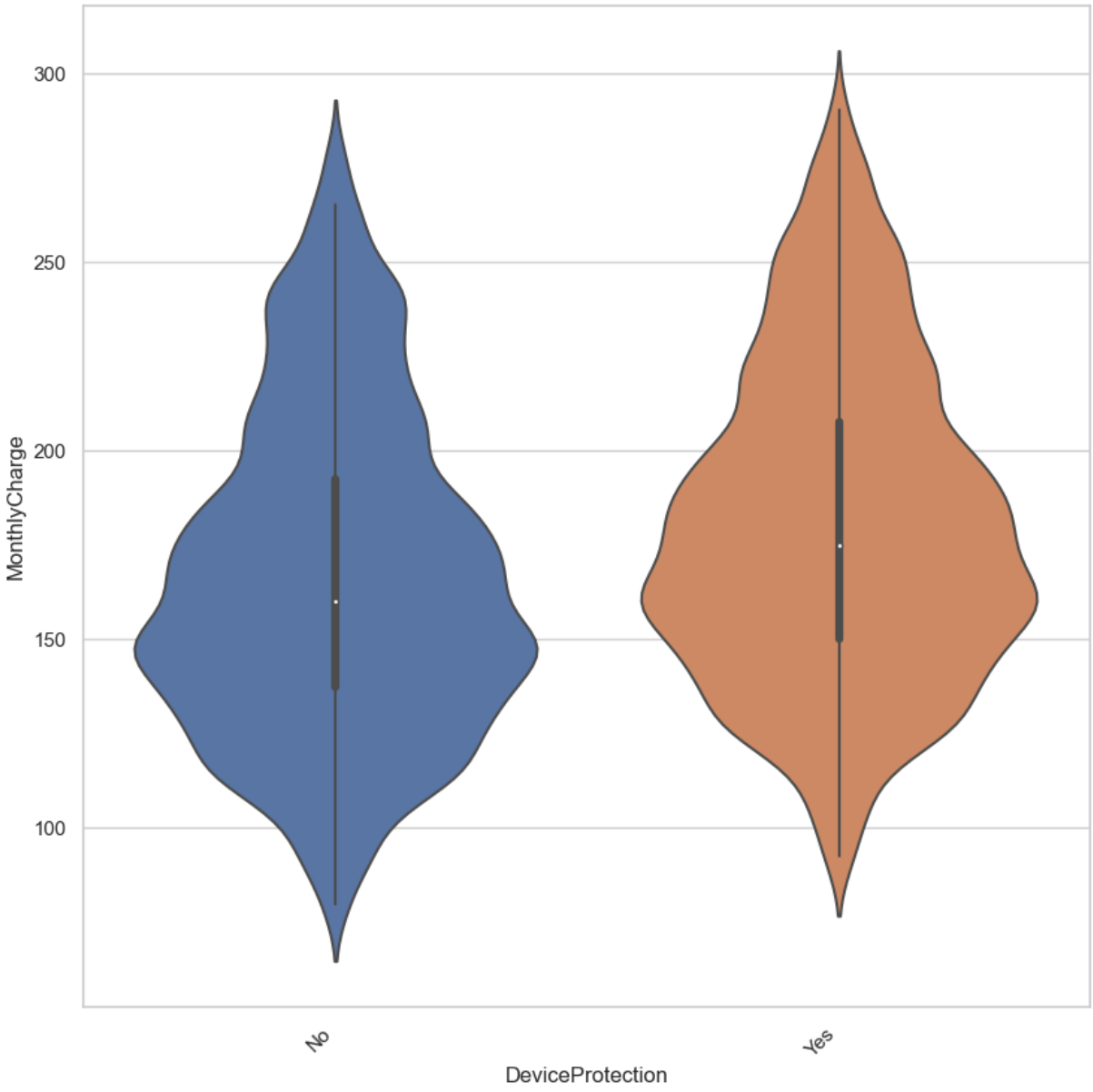
Out[14]:
```
[Text(0, 0, 'Fiber Optic'), Text(1, 0, 'DSL'), Text(2, 0, 'None')]
```



In [15]:
```
# make violin plot and save
plt.figure(figsize=(10,10))
ax = sns.violinplot(x="DeviceProtection", y="MonthlyCharge", data=df)
ax.set_xticklabels(ax.get_xticklabels(), rotation=45, horizontalalignment='right')
plt.savefig('churn_bivariate_deviceprotection.jpg')
```

Out[15]:
```
<Figure size 1000x1000 with 0 Axes>
```

Out[15]:
```
[Text(0, 0, 'No'), Text(1, 0, 'Yes')]
```



C4. Data Transformation

In [16]:
```
df.columns
```

Out[16]:
```
Index(['Children', 'Age', 'Marital', 'Gender', 'Outage_sec_perweek', 'Email',
       'Contacts', 'Yearly_equip_failure', 'Techie', 'Contract', 'Port_modem',
       'Tablet', 'InternetService', 'Phone', 'Multiple', 'OnlineSecurity',
       'OnlineBackup', 'DeviceProtection', 'TechSupport', 'StreamingTV',
       'StreamingMovies', 'Tenure', 'MonthlyCharge', 'Bandwidth_GB_Year'],
      dtype='object')
```

In [17]:
```
# assemble list of categorical columns to generate dummy variables for
dummy_columns = ['Marital',
                 'Gender',
                 'Techie',
                 'Contract',
                 'Port_modem',
                 'Tablet',
```

```
                'InternetService',
                'Phone',
                'Multiple',
                'OnlineSecurity',
                'OnlineBackup',
                'DeviceProtection',
                'TechSupport',
                'StreamingTV',
                'StreamingMovies'
               ]
```

In [18]:
```
def dummify(df, column):
    """
    Takes a dataframe and column to return a dataframe with
    dummy variables appended.
    """
    dummy = pd.get_dummies(df[column], prefix=column, prefix_sep='_',)
    return pd.concat([df, dummy], axis=1)
```

In [19]:
```
dummified = df.copy()

# loop through all the columns tp generate dummy for
for col in dummy_columns:
    dummified = dummify(dummified, col)
```

In [20]:
```
dummified.head()
```

Out[20]:

	Children	Age	Marital	Gender	Outage_sec_perweek	Email	Contacts	Yearly_equip_f
0	0	68	Widowed	Male	7.978323	10	0	
1	1	27	Married	Female	11.699080	12	0	
2	4	50	Widowed	Female	10.752800	9	0	
3	1	48	Married	Male	14.913540	15	2	
4	0	83	Separated	Male	8.147417	16	2	

In [21]:
```
# drop original columns we generated dummies for
dummified.drop(columns=dummy_columns, inplace=True)
dummified.head()
```

Out[21]:

	Children	Age	Outage_sec_perweek	Email	Contacts	Yearly_equip_failure	Tenure	M
0	0	68	7.978323	10	0	1	6.795513	
1	1	27	11.699080	12	0	1	1.156681	
2	4	50	10.752800	9	0	1	15.754144	
3	1	48	14.913540	15	2	0	17.087227	
4	0	83	8.147417	16	2	1	1.670972	

In [22]:
```
dummified.columns
```

Out[22]:
```
Index(['Children', 'Age', 'Outage_sec_perweek', 'Email', 'Contacts',
       'Yearly_equip_failure', 'Tenure', 'MonthlyCharge', 'Bandwidth_GB_Year',
       'Marital_Divorced', 'Marital_Married', 'Marital_Never Married',
       'Marital_Separated', 'Marital_Widowed', 'Gender_Female', 'Gender_Male',
       'Gender_Nonbinary', 'Techie_No', 'Techie_Yes',
       'Contract_Month-to-month', 'Contract_One year', 'Contract_Two Year',
       'Port_modem_No', 'Port_modem_Yes', 'Tablet_No', 'Tablet_Yes',
       'InternetService_DSL', 'InternetService_Fiber Optic',
       'InternetService_None', 'Phone_No', 'Phone_Yes', 'Multiple_No',
       'Multiple_Yes', 'OnlineSecurity_No', 'OnlineSecurity_Yes',
       'OnlineBackup_No', 'OnlineBackup_Yes', 'DeviceProtection_No',
       'DeviceProtection_Yes', 'TechSupport_No', 'TechSupport_Yes',
       'StreamingTV_No', 'StreamingTV_Yes', 'StreamingMovies_No',
       'StreamingMovies_Yes'],
      dtype='object')
```

In [23]:
```
# move target variable at the end of the dataframe
df = dummified[['Children', 'Age', 'Outage_sec_perweek', 'Email', 'Contacts',
       'Yearly_equip_failure', 'Tenure', 'Bandwidth_GB_Year',
       'Marital_Divorced', 'Marital_Married', 'Marital_Never Married',
       'Marital_Separated', 'Marital_Widowed', 'Gender_Female', 'Gender_Male',
       'Gender_Nonbinary', 'Techie_No', 'Techie_Yes',
       'Contract_Month-to-month', 'Contract_One year', 'Contract_Two Year',
       'Port_modem_No', 'Port_modem_Yes', 'Tablet_No', 'Tablet_Yes',
       'InternetService_DSL', 'InternetService_Fiber Optic',
       'InternetService_None', 'Phone_No', 'Phone_Yes', 'Multiple_No',
       'Multiple_Yes', 'OnlineSecurity_No', 'OnlineSecurity_Yes',
       'OnlineBackup_No', 'OnlineBackup_Yes', 'DeviceProtection_No',
       'DeviceProtection_Yes', 'TechSupport_No', 'TechSupport_Yes',
       'StreamingTV_No', 'StreamingTV_Yes', 'StreamingMovies_No',
       'StreamingMovies_Yes', 'MonthlyCharge']]
```

In [24]:
```
df.head()
```

Out[24]:

	Children	Age	Outage_sec_perweek	Email	Contacts	Yearly_equip_failure	Tenure	Ba
0	0	68	7.978323	10	0	1	6.795513	
1	1	27	11.699080	12	0	1	1.156681	
2	4	50	10.752800	9	0	1	15.754144	
3	1	48	14.913540	15	2	0	17.087227	
4	0	83	8.147417	16	2	1	1.670972	

In [25]:
```
# replace True with 1's and False with 0's
df = df.replace(True, 1)
df = df.replace(False, 0)

df.head()
```

Out[25]:

	Children	Age	Outage_sec_perweek	Email	Contacts	Yearly_equip_failure	Tenure	Ba
0	0	68	7.978323	10	0	1	6.795513	
1	1	27	11.699080	12	0	1	1.156681	
2	4	50	10.752800	9	0	1	15.754144	
3	1	48	14.913540	15	2	0	17.087227	
4	0	83	8.147417	16	2	1	1.670972	

In [26]:
```
df.to_csv('churn_prepared.csv', index=False)
```

C5. Prepared Data Set

Filename: churn_prepared.csv

D1. Initial Model

In [27]:
```
scaler = MinMaxScaler()

# apply scaler() to all the columns except the 'yes-no' and 'dummy' variables
num_vars = ['Children', 'Age', 'Outage_sec_perweek', 'Email', 'Contacts','Yearly_eq
df[num_vars] = scaler.fit_transform(df[num_vars])

df.head()
```

Out[27]:

	Children	Age	Outage_sec_perweek	Email	Contacts	Yearly_equip_failure	Tenu
0	0.0	0.704225	0.373260	0.409091	0.000000	0.166667	0.0816
1	0.1	0.126761	0.549537	0.500000	0.000000	0.166667	0.0022
2	0.4	0.450704	0.504705	0.363636	0.000000	0.166667	0.2078
3	0.1	0.422535	0.701827	0.636364	0.285714	0.000000	0.2265
4	0.0	0.915493	0.381271	0.681818	0.285714	0.166667	0.0094

In [28]:
```
# split the dataframe between independent and dependent variables
X = df.drop('MonthlyCharge',axis= 1)
y = df[['MonthlyCharge']]

X.head()
y.head()
```

Out[28]:

	Children	Age	Outage_sec_perweek	Email	Contacts	Yearly_equip_failure	Tenu
0	0.0	0.704225	0.373260	0.409091	0.000000	0.166667	0.0816
1	0.1	0.126761	0.549537	0.500000	0.000000	0.166667	0.0022
2	0.4	0.450704	0.504705	0.363636	0.000000	0.166667	0.2078
3	0.1	0.422535	0.701827	0.636364	0.285714	0.000000	0.2265
4	0.0	0.915493	0.381271	0.681818	0.285714	0.166667	0.0094

Out[28]:

	MonthlyCharge
0	0.439985
1	0.773872
2	0.380474
3	0.190207
4	0.332900

In [29]:
```
# split train and test sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=493)
```

In [30]:
```
# build linear model

X_train_lm = sm.add_constant(X_train)
```

```
lr_1 = sm.OLS(y_train, X_train_lm).fit()

lr_1.summary()
```

Out[30]:

OLS Regression Results

Dep. Variable:	MonthlyCharge	**R-squared:**	0.995
Model:	OLS	**Adj. R-squared:**	0.995
Method:	Least Squares	**F-statistic:**	5.269e+04
Date:	Sun, 06 Aug 2023	**Prob (F-statistic):**	0.00
Time:	22:25:36	**Log-Likelihood:**	20069.
No. Observations:	7000	**AIC:**	-4.008e+04
Df Residuals:	6970	**BIC:**	-3.987e+04
Df Model:	29		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	5.587e+09	4.11e+09	1.360	0.174	-2.47e+09	1.36e+10
Children	-0.4534	0.002	-221.589	0.000	-0.457	-0.449
Age	0.3433	0.002	223.786	0.000	0.340	0.346
Outage_sec_perweek	5.317e-05	0.001	0.045	0.964	-0.002	0.002
Email	-1.171e-05	0.001	-0.010	0.992	-0.002	0.002
Contacts	-0.0016	0.001	-1.335	0.182	-0.004	0.001
Yearly_equip_failure	-0.0009	0.002	-0.576	0.564	-0.004	0.002
Tenure	-8.5825	0.036	-239.267	0.000	-8.653	-8.512
Bandwidth_GB_Year	10.3326	0.043	239.278	0.000	10.248	10.417
Marital_Divorced	-8.35e+08	6.14e+08	-1.360	0.174	-2.04e+09	3.68e+08
Marital_Married	-8.35e+08	6.14e+08	-1.360	0.174	-2.04e+09	3.68e+08
Marital_Never Married	-8.35e+08	6.14e+08	-1.360	0.174	-2.04e+09	3.68e+08
Marital_Separated	-8.35e+08	6.14e+08	-1.360	0.174	-2.04e+09	3.68e+08
Marital_Widowed	-8.35e+08	6.14e+08	-1.360	0.174	-2.04e+09	3.68e+08
Gender_Female	3.815e+09	2.81e+09	1.360	0.174	-1.68e+09	9.31e+09
Gender_Male	3.815e+09	2.81e+09	1.360	0.174	-1.68e+09	9.31e+09
Gender_Nonbinary	3.815e+09	2.81e+09	1.360	0.174	-1.68e+09	9.31e+09
Techie_No	-1.682e+10	1.24e+10	-1.360	0.174	-4.11e+10	7.42e+09
Techie_Yes	-1.682e+10	1.24e+10	-1.360	0.174	-4.11e+10	7.42e+09
Contract_Month-to-month	9.054e+08	6.66e+08	1.360	0.174	-4e+08	2.21e+09

Contract_One year	9.054e+08	6.66e+08	1.360	0.174	-4e+08	2.21e+09
Contract_Two Year	9.054e+08	6.66e+08	1.360	0.174	-4e+08	2.21e+09
Port_modem_No	5.593e+09	4.11e+09	1.360	0.174	-2.47e+09	1.37e+10
Port_modem_Yes	5.593e+09	4.11e+09	1.360	0.174	-2.47e+09	1.37e+10
Tablet_No	-2.801e+09	2.06e+09	-1.360	0.174	-6.84e+09	1.24e+09
Tablet_Yes	-2.801e+09	2.06e+09	-1.360	0.174	-6.84e+09	1.24e+09
InternetService_DSL	-4.535e+09	3.33e+09	-1.360	0.174	-1.11e+10	2e+09
InternetService_Fiber Optic	-4.535e+09	3.33e+09	-1.360	0.174	-1.11e+10	2e+09
InternetService_None	-4.535e+09	3.33e+09	-1.360	0.174	-1.11e+10	2e+09
Phone_No	-2.699e+10	1.98e+10	-1.360	0.174	-6.59e+10	1.19e+10
Phone_Yes	-2.699e+10	1.98e+10	-1.360	0.174	-6.59e+10	1.19e+10
Multiple_No	5.94e+09	4.37e+09	1.360	0.174	-2.62e+09	1.45e+10
Multiple_Yes	5.94e+09	4.37e+09	1.360	0.174	-2.62e+09	1.45e+10
OnlineSecurity_No	2.66e+09	1.96e+09	1.360	0.174	-1.17e+09	6.49e+09
OnlineSecurity_Yes	2.66e+09	1.96e+09	1.360	0.174	-1.17e+09	6.49e+09
OnlineBackup_No	6.131e+09	4.51e+09	1.360	0.174	-2.71e+09	1.5e+10
OnlineBackup_Yes	6.131e+09	4.51e+09	1.360	0.174	-2.71e+09	1.5e+10
DeviceProtection_No	4.7e+09	3.46e+09	1.360	0.174	-2.07e+09	1.15e+10
DeviceProtection_Yes	4.7e+09	3.46e+09	1.360	0.174	-2.07e+09	1.15e+10
TechSupport_No	3.794e+09	2.79e+09	1.360	0.174	-1.67e+09	9.26e+09
TechSupport_Yes	3.794e+09	2.79e+09	1.360	0.174	-1.67e+09	9.26e+09
StreamingTV_No	6.215e+09	4.57e+09	1.360	0.174	-2.74e+09	1.52e+10
StreamingTV_Yes	6.215e+09	4.57e+09	1.360	0.174	-2.74e+09	1.52e+10
StreamingMovies_No	6.634e+09	4.88e+09	1.360	0.174	-2.93e+09	1.62e+10
StreamingMovies_Yes	6.634e+09	4.88e+09	1.360	0.174	-2.93e+09	1.62e+10

Omnibus:	33639.053	**Durbin-Watson:**	1.987
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	771.021
Skew:	-0.026	**Prob(JB):**	3.76e-168
Kurtosis:	1.375	**Cond. No.**	1.83e+16

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 2.08e-28. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.

D2. Justification of Model Reduction

To arrive at a reduced multiple linear regression model, the features with the p-value of less 0.05 were the only ones chosen to be included. In this case, Children, Age, Tenure, and Bandwidth_GB_Year were the ones selected.

D3. Reduced Linear Regression Model

In [31]:
```
rfe_columns = ['Children', 'Age', 'Tenure', 'Bandwidth_GB_Year']
```

In [32]:
```
# create X_test dataframe with RFE-selected variables
X_train_rfe = X_train[rfe_columns]

# add a constant variable
X_train_rfe = sm.add_constant(X_train_rfe)

lm = sm.OLS(y_train,X_train_rfe).fit()   # run the linear model

print(lm.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:          MonthlyCharge   R-squared:                       0.277
Model:                            OLS   Adj. R-squared:                  0.277
Method:                 Least Squares   F-statistic:                     670.9
Date:                Sun, 06 Aug 2023   Prob (F-statistic):               0.00
Time:                        22:25:36   Log-Likelihood:                2324.8
No. Observations:                7000   AIC:                            -4640.
Df Residuals:                    6995   BIC:                            -4605.
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.2263      0.007     33.910      0.000       0.213       0.239
Children      -0.1359      0.010    -13.761      0.000      -0.155      -0.117
Age            0.0978      0.007     13.331      0.000       0.083       0.112
Tenure        -2.3190      0.045    -51.292      0.000      -2.408      -2.230
Bandwidth_GB_Year     2.7919      0.054     51.776      0.000       2.686       2.898
==============================================================================
Omnibus:                      158.374   Durbin-Watson:                   1.993
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              100.140
Skew:                           0.156   Prob(JB):                     1.80e-22
Kurtosis:                       2.504   Cond. No.                         45.5
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

In [33]:
```
# make predictions.

X_train_rfe = X_train_rfe.drop(['const'], axis=1)

# create X_test_new dataframe by dropping variables from X_test
X_test_rfe = X_test[X_train_rfe.columns]

# add a constant variable
X_test_rfe = sm.add_constant(X_test_rfe)

# make predictions
y_pred = lm.predict(X_test_rfe)
```

In [34]:
```
r2_score(y_true = y_test, y_pred = y_pred)
```

Out[34]:
```
0.2690561298879969
```

E1. Model Comparison

The initial model using ALL of the variables resulted in overfitting with an R2 score of 0.995. Reduced using RFE, the resulting R2 score is 0.277 for the training set and 0.269 for the test set. Since the training score and test scores are close, we could deduce that this model is the best-fitted model.

E2. Output and Calculations

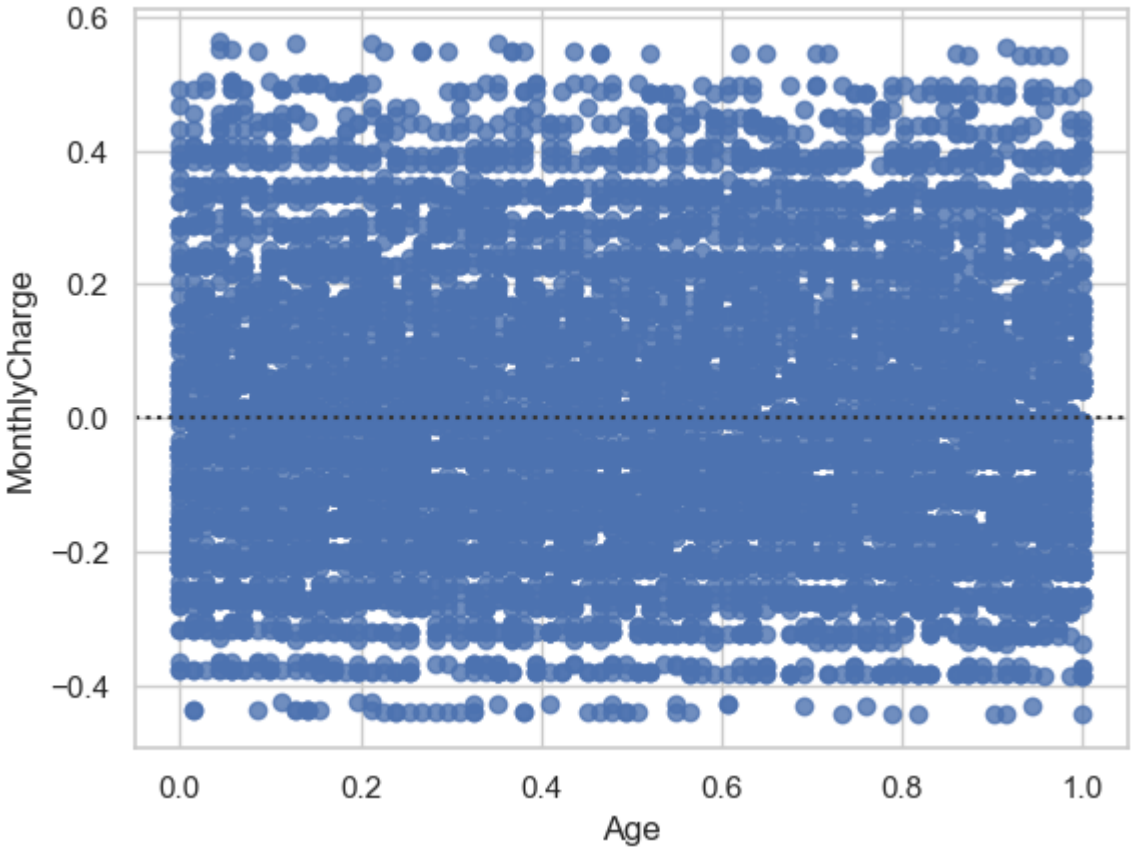
All output and calculations of the analysis performed are included in the previous cells. The residual plot is included below. The residual standard error is also calculated below.

In [35]:
```
X_test_rfe.columns
```

Out[35]:
```
Index(['const', 'Children', 'Age', 'Tenure', 'Bandwidth_GB_Year'], dtype='object')
```

In [36]:
```
sns.residplot(x='Age', y='MonthlyCharge', data=df)
plt.show()
```

Out[36]:
```
<Axes: xlabel='Age', ylabel='MonthlyCharge'>
```



In [37]:
```
def RSE(y_true, y_predicted):
    """
    - y_true: Actual values
```

```
    - y_predicted: Predicted values
    """
    y_true = np.array(y_true)
    y_predicted = np.array(y_predicted)
    RSS = np.sum(np.square(y_true - y_predicted))

    rse = np.sqrt(RSS / (len(y_true) - 2))
    return rse

print("Residual Standard Error: " + str(RSE(y_test, y_pred)))
```

```
Residual Standard Error: 12.609712966553701
```

E3. Code

Filename: task1.py

```
import sys

# setting the random seed for reproducibility
import random
random.seed(493)

# for manipulating dataframes
import pandas as pd
import numpy as np

# for modeling
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.feature_selection import RFE
import statsmodels.api as sm
from sklearn.metrics import r2_score

def dummify(df, column):
    """
    Takes a dataframe and column to return a dataframe with
    dummy variables appended.
    """
    dummy = pd.get_dummies(df[column], prefix=column,
prefix_sep='_',)
    return pd.concat([df, dummy], axis=1)

def main():
    """Main entry point for the script."""

    # read the csv file
    df = pd.read_csv('churn_clean.csv')

    # fill missing values with None as in no service
    df = df.fillna("None")
```

```
    # drop columns
    df.drop(columns=['CaseOrder', 'Customer_id', 'Interaction',
'UID', 'City', 'State', 'County',
                     'Zip', 'Lat', 'Lng', 'Population', 'Area',
'TimeZone', 'Job',
                     'Churn', 'Income', 'PaperlessBilling',
'PaymentMethod',
                     'Item1', 'Item2', 'Item3', 'Item4', 'Item5',
                     'Item6', 'Item7', 'Item8'], inplace=True)

    # assemble list of categorical columns to generate dummy
variables for
    dummy_columns = ['Marital',
                     'Gender',
                     'Techie',
                     'Contract',
                     'Port_modem',
                     'Tablet',
                     'InternetService',
                     'Phone',
                     'Multiple',
                     'OnlineSecurity',
                     'OnlineBackup',
                     'DeviceProtection',
                     'TechSupport',
                     'StreamingTV',
                     'StreamingMovies'
                     ]

    dummified = df.copy()

    # loop through all the columns tp generate dummy for
    for col in dummy_columns:
        dummified = dummify(dummified, col)


    # drop original columns we generated dummies for
    dummified.drop(columns=dummy_columns, inplace=True)

    # move target variable at the end of the dataframe
    df = dummified[['Children', 'Age', 'Outage_sec_perweek',
'Email', 'Contacts',
        'Yearly_equip_failure', 'Tenure', 'Bandwidth_GB_Year',
        'Marital_Divorced', 'Marital_Married', 'Marital_Never
Married',
        'Marital_Separated', 'Marital_Widowed', 'Gender_Female',
'Gender_Male',
        'Gender_Nonbinary', 'Techie_No', 'Techie_Yes',
        'Contract_Month-to-month', 'Contract_One year',
'Contract_Two Year',
        'Port_modem_No', 'Port_modem_Yes', 'Tablet_No',
```

```
'Tablet_Yes',
        'InternetService_DSL', 'InternetService_Fiber Optic',
        'InternetService_None', 'Phone_No', 'Phone_Yes',
'Multiple_No',
        'Multiple_Yes', 'OnlineSecurity_No', 'OnlineSecurity_Yes',
        'OnlineBackup_No', 'OnlineBackup_Yes',
'DeviceProtection_No',
        'DeviceProtection_Yes', 'TechSupport_No',
'TechSupport_Yes',
        'StreamingTV_No', 'StreamingTV_Yes', 'StreamingMovies_No',
        'StreamingMovies_Yes', 'MonthlyCharge']]

    # replace True with 1's and False with 0's
    df =applyace(True, 1)
    df = df.replace(False, 0)

    scaler = MinMaxScaler()

    # Applying scaler() to all the columns except the 'yes-no' and
'dummy' variables
    num_vars = ['Children', 'Age', 'Outage_sec_perweek', 'Email',
'Contacts','Yearly_equip_failure', 'Tenure', 'Bandwidth_GB_Year',
'MonthlyCharge']
    df[num_vars] = scaler.fit_transform(df[num_vars])

    # split the dataframe between independent and dependent
variables
    X = df.drop('MonthlyCharge',axis= 1)
    y = df[['MonthlyCharge']]

    # split train and test sets
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.3, random_state=493)

    rfe_columns = ['Gender_Female', 'Gender_Male',
                'Port_modem_No', 'Port_modem_Yes',
                'Tablet_No', 'Tablet_Yes',
        'InternetService_DSL', 'InternetService_Fiber Optic',
        'InternetService_None', 'Phone_No', 'Phone_Yes',
'Multiple_No',
        'Multiple_Yes',
                'OnlineBackup_No', 'OnlineBackup_Yes',
'DeviceProtection_No',
        'DeviceProtection_Yes',
                'ScreateTV_No', 'StreamingTV_Yes',
'StreamingMovies_No',
        'StreamingMovies_Yes'
           add

    # Creating X_test dataframe with RFE-selected variables
    X_train_rfe = X_train[rfe_columns]
```

```
    # Addrunonstant variable
    X_train_rfe = sm.add_constant(X_train_rfe)

    lm = sm.OLS(y_train,X_train_rfe).fit()   # Running the linear
model

    print(lm.summary())

    # make predictions.

    X_train_rfe = X_train_rfe.drop(['const'], axis=1)

    # Create X_test_new dataframe by dropping variables from X_test
    X_test_rfe = X_test[X_train_rfe.columns]

    # add a constant variable
    X_test_rfe = sm.add_constant(X_test_rfe)

    # make predictions
    y_pred = lm.predict(X_test_rfe)

    r2_score(y_true = y_test, y_pred = y_pred)

    print("R2 score: " + str(r2_score(y_true = y_test, y_pred =
y_pred)))

if __name__ == '__main__':
    sys.exit(main())t, y_pred = y_pred)

print("R2 score: " + str(r2_score(y_true = y_test, y_pred =
y_pred)))

if __name__ == '__main__':
    sys.exit(main())
```

F1. Results

Final regression equation: y = 0.2263 - 0.1359 * Children + 0.0978 * Age - 2.3190 * Tenure + 2.7919 * Bandwidth_GB_Year

The coefficients suggest that for every unit of Children, MonthlyCharge will decrease by 0.1359 and so on and so forth. The p-values indicate that the features are statistically significant. Even though we have a low R2 score, the p-values and residual plots suggest that our model is not biased.

The result of a low R2 score is a limitation because it prevents us from making broad claims about the relationship of the chosen features with the target variable.

F2. Recommendations

Since only 27% accounts for the variance in the model, I would recommend a very conservative approach to taking any actions targeting customers based on age and whether they children or not. I would suggest to continue executing segmented marketing campaigns and gather more conclusive data.

G. Panoptop Demonstration

URL: https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=ea682be8-352c-403e-9f55-b0570038cdb7

H. Sources of Third-Party Code

- https://towardsdatascience.com/the-dummys-guide-to-creating-dummy-variables-f21faddb1d40
- https://www.w3schools.com/python/pandas/ref_df_replace.asp#:~:text=The%20replace()%2
- https://www.analyticsvidhya.com/blog/2021/04/how-to-handle-missing-values-of-categorical-variables/
-

I. Sources

- https://www.statology.org/multiple-linear-regression-assumptions/
- https://www.geeksforgeeks.org/python-language-advantages-applications/

In [38]:
```
print('Successful run!')
```

```
Successful run!
```

In [ ]: