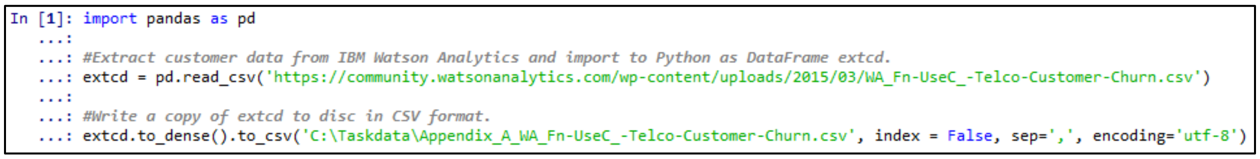
Customer Attrition in Telecommunications

Paul A. Wilson

Western Governors University

I. Tool Selection – This analysis of customer data uses Python 3.6.4. Figure 1 displays the code written to extract the data, create a Python DataFrame (extcd), and export a CSV formatted copy to the local C: drive. Appendix A contains a copy of the file 'Appendix_A_WA_Fn-UseC_-Telco-Customer-Churn.csv'



```
In [1]: import pandas as pd
   ...:
   ...: #Extract customer data from IBM Watson Analytics and import to Python as DataFrame extcd.
   ...: extcd = pd.read_csv('https://community.watsonanalytics.com/wp-content/uploads/2015/03/WA_Fn-UseC_-Telco-Customer-Churn.csv')
   ...:
   ...: #Write a copy of extcd to disc in CSV format.
   ...: extcd.to_dense().to_csv('C:\Taskdata\Appendix_A_WA_Fn-UseC_-Telco-Customer-Churn.csv', index = False, sep=',', encoding='utf-8')
```

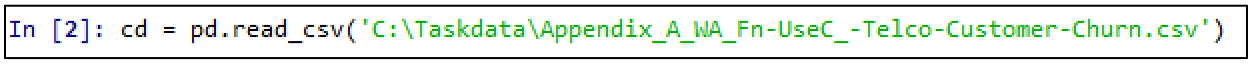
Figure 1. Python code used to extract customer data, create a DataFrame, and export a copy.

A. While SAS, R, or Python can accomplish the required analysis, this analyst's choice of Python is primarily driven by the following factors. The first is cost. SAS requires either commercially purchased client/server software or SAS's University Edition which, according to the license agreement, is free for non-commercial use, but SAS has the right to end this license at any time (SAS Institute Incorporated, 2015). In contrast Python Software Foundation (2018) and The R Foundation (n.d.), feature open source licensing and are free to download and use. The second factor is that for developers with prior object-oriented programming experience, Python is easier to learn than R (Willems, 2015). Priyadharshini (2017) believes that Python's ease of learning also equates to writing less lines of code, which makes debugging easier. Python's extensive documentation is also readily available online and there are many active user groups.

B. The overall goal of this analysis is to help the company quickly identify the major contributors of customer churn. Customer churn is believed to result from customers dropping landline service in favor of cable service provided by competitors. The specific objectives are as follows: 1) Successfully scrape customer data from the internet. 2) Review and clean the data. 3) Perform exploratory data analysis (EDA). 4) Apply a descriptive analysis method to explain the relationship between independent variables. 5) Apply a predictive analysis method to model the relationship between the dependent variable and the most critical independent variables. 6) Summarize findings of why customers are churning. 7) Identify current customers most likely to churn.

C. The descriptive method is principle component analysis (PCA). PCA can assist with determining the number of principle components required to sufficiently explain the dataset so that the independent variables may be reduced. The non-descriptive method is logistic regression. Logistic regression shows the likelyhood that the set of independent variables will affect customer churn.

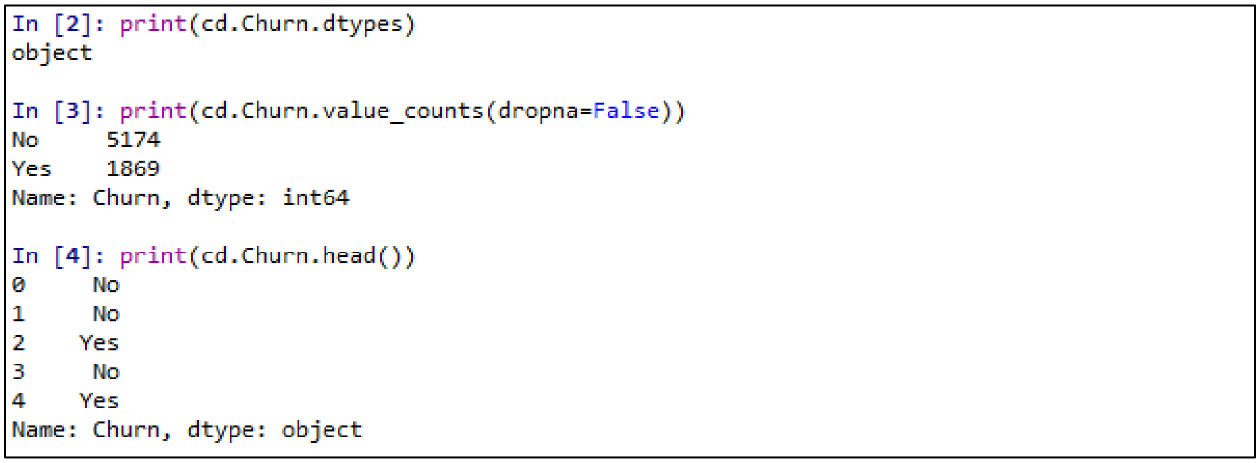
II. Data Exploration and Preparation – The first step in exploring and preparing the customer data is to import the local CSV file into a Pandas DataFrame (cd) as shown in figure 2.



```
In [2]: cd = pd.read_csv('C:\Taskdata\Appendix_A_WA_Fn-UseC_-Telco-Customer-Churn.csv')
```

Figure 2. Pandas DataFrame created from local customer data CSV file.

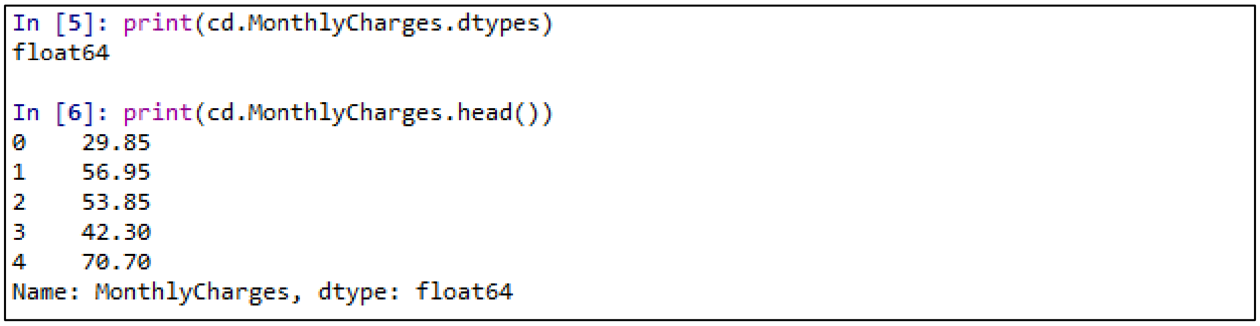
D. The target variable is ‘Churn’. Churn is a qualitative categorical variable that features only two distinct nominal values – either “Yes” or “No”. This provides a binary logical indicator of whether the customer has or has not churned. Figure 3 shows how Python imports Churn as datatype ‘object’, the count of values, and the first five observations. The third and fifth customers have churned while the first, second, and fourth have not.



```
In [2]: print(cd.Churn.dtypes)
object

In [3]: print(cd.Churn.value_counts(dropna=False))
No     5174
Yes    1869
Name: Churn, dtype: int64

In [4]: print(cd.Churn.head())
0     No
1     No
2    Yes
3     No
4    Yes
Name: Churn, dtype: object
```

Figure 3. Churn variable's data type, count of values, and first five observations as imported. Two of the first five customer have churned.

E. An example of an independent variable is MonthlyCharges. It is quantitative and continuous. Python imports MonthlyCharges as the 'float64' datatype. Figure 4 shows the datatype and the first five observations.

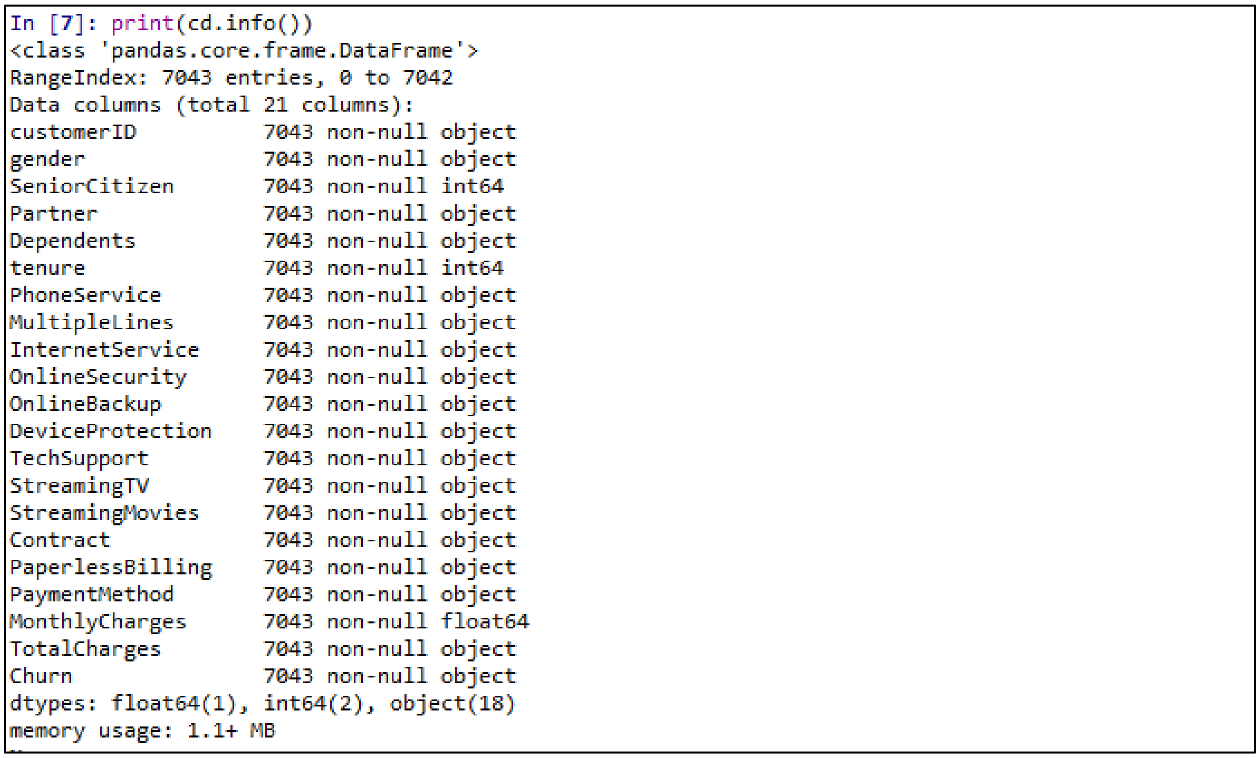


```
In [5]: print(cd.MonthlyCharges.dtypes)
float64

In [6]: print(cd.MonthlyCharges.head())
0    29.85
1    56.95
2    53.85
3    42.30
4    70.70
Name: MonthlyCharges, dtype: float64
```

Figure 4. MonthlyCharges variable's datatype and first five observations as imported. This variable shows how much the customer pays per month.

F. The data manipulation goal is to refine DataFrame cd into a form that accommodates robust analysis using Python's available features. Data preparation aims are to remove unnecessary variables, mitigate erroneous data, eliminate duplicate observations, transform binary categorical string data to numeric, and lower the required memory usage. Variables not needed for this analysis will be removed. Erroneous data may be mitigated by transforming it with computation, substitution, imputation, or elimination. Duplicate observations will be deleted. Binary variables imported with the object datatype will be transformed to numeric values and datatype. Other variables with the object datatype will be converted to character to lower required memory usage.

G. The phenomena to predict is how likely Churn is true or false based on the values of one or more predictor variables. Figure 5 and table 1 show the statistical identity of the customer data. Figure 5 gives an overview of cd and its 7,043 observations, 21 variables, datatypes, and non-null counts. Note that cd requires 1.1+ MB of memory.



```
In [7]: print(cd.info())
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 21 columns):
customerID          7043 non-null object
gender              7043 non-null object
SeniorCitizen       7043 non-null int64
Partner             7043 non-null object
Dependents          7043 non-null object
tenure              7043 non-null int64
PhoneService        7043 non-null object
MultipleLines       7043 non-null object
InternetService     7043 non-null object
OnlineSecurity      7043 non-null object
OnlineBackup        7043 non-null object
DeviceProtection    7043 non-null object
TechSupport         7043 non-null object
StreamingTV         7043 non-null object
StreamingMovies     7043 non-null object
Contract            7043 non-null object
PaperlessBilling    7043 non-null object
PaymentMethod       7043 non-null object
MonthlyCharges      7043 non-null float64
TotalCharges        7043 non-null object
Churn               7043 non-null object
dtypes: float64(1), int64(2), object(18)
memory usage: 1.1+ MB
```

Figure 5. DataFrame info: count of observations and variables, count of non-missing values, and datatypes of each variable.

Table 1 provides details of each variable: its type, datatype, values, and prescribed data cleanup treatment.

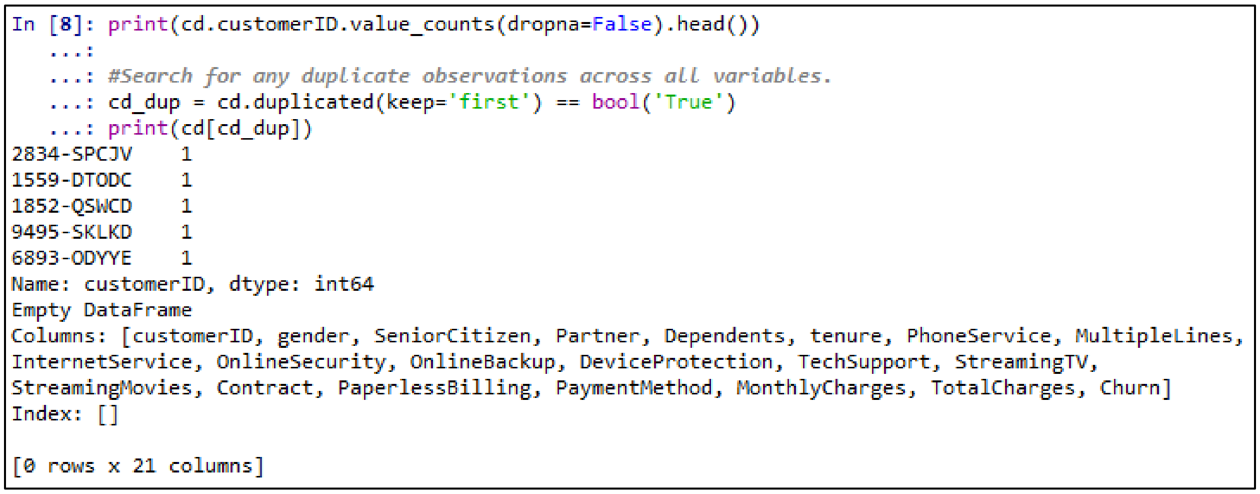
Table 1

Statistical Identity of Raw Customer Data

Variable			
Type and Name	Datatype	Values	Treatment
Qualitative			
customerID	object	7,043 nominal unique values, no nulls	1
gender	object	Male, Female, no nulls	2
Partner	object	Yes, No, no nulls	2
Dependents	object	Yes, No, no nulls	2
PhoneService	object	Yes, No, no nulls	2
MultipleLines	object	Yes, No, No phone service, no nulls	3
InternetService	object	Fiber optic, DSL, No	3
OnlineSecurity	object	Yes, No, No internet service, no nulls	3
OnlineBackup	object	Yes, No, No internet service, no nulls	3
DeviceProtection	object	Yes, No, No internet service, no nulls	3
TechSupport	object	Yes, No, No internet service, no nulls	3
StreamingTV	object	Yes, No, No internet service, no nulls	3
StreamingMovies	object	Yes, No, No internet service, no nulls	3
Contract	object	One year, Two year, Month-to-month, no nulls	3
PaperlessBilling	object	Yes, No, no nulls	2
PaymentMethod	object	Credit card (automatic), Bank transfer (automatic), Mailed check, Electronic check	3
TotalCharges	object	Nominal values, no nulls	4
Churn	object	Yes, No, no nulls	2
Quantitative			
SeniorCitizen	int64	1 (true), 0 (false), no nulls	None
tenure	int64	Distinct values, no nulls.	None
MonthlyCharges	float64	Distinct values, no nulls.	None

Note: Treatment 1 = Delete, not needed. Treatment 2 = Transform to binary numeric int64, 1 (yes), 0 (no). Treatment 3 = Convert to character datatype. Treatment 4 = Convert to numeric float64.

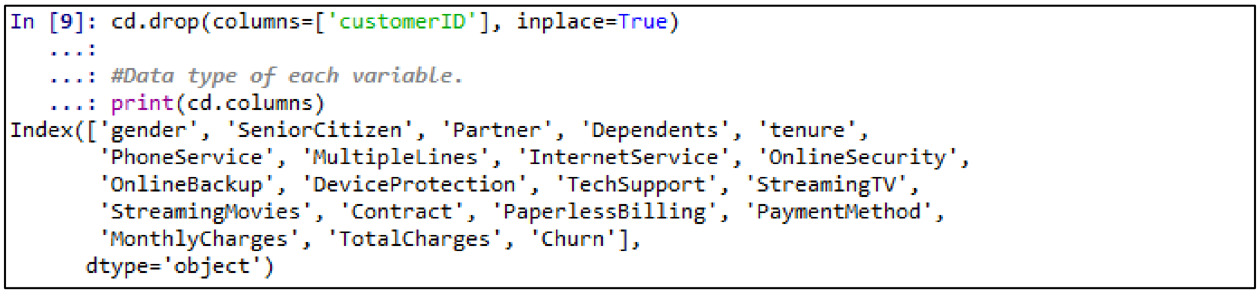
H. Python's panda package allows quick analysis of the raw data to achieve data preparation aims. The first step in data preparation is to check the data for duplicate observations. The CustomerID variable appears to contain unique values making each observation unique. If two customers had duplicate values in all other variables, it would be a coincidence. Figure 6 shows two methods to determine that observations are unique. The first shows that the five most frequent CustomerIDs occur only once. The last method returns an empty DataFrame showing that all observations in cd are unique when comparing all variables row by row.



```
In [8]: print(cd.customerID.value_counts(dropna=False).head())
   ...:
   ...: #Search for any duplicate observations across all variables.
   ...: cd_dup = cd.duplicated(keep='first') == bool('True')
   ...: print(cd[cd_dup])
2834-SPCJV    1
1559-DTODC    1
1852-QSWCD    1
9495-SKLKD    1
6893-ODYYE    1
Name: customerID, dtype: int64
Empty DataFrame
Columns: [customerID, gender, SeniorCitizen, Partner, Dependents, tenure, PhoneService, MultipleLines,
InternetService, OnlineSecurity, OnlineBackup, DeviceProtection, TechSupport, StreamingTV,
StreamingMovies, Contract, PaperlessBilling, PaymentMethod, MonthlyCharges, TotalCharges, Churn]
Index: []

[0 rows x 21 columns]
```

Figure 6. Two methods to search for duplicate observations. .value_counts shows that the most frequent CustomerIDs occur once. .duplicated returns an empty DataFrame when analyzing all variables for all rows.

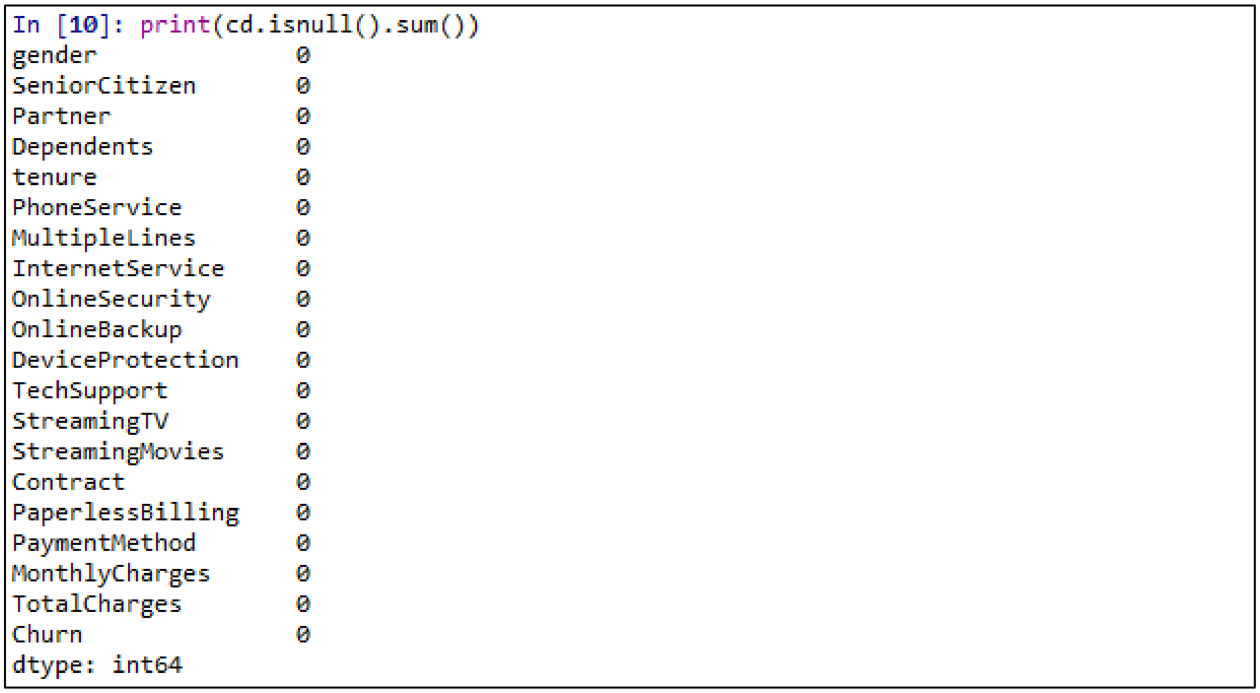
The variable customerID is no longer needed for this analysis and may be deleted. Figure 7 illustrates the deletion and examination of the remaining variable names.



```
In [9]: cd.drop(columns=['customerID'], inplace=True)
   ...:
   ...: #Data type of each variable.
   ...: print(cd.columns)
Index(['gender', 'SeniorCitizen', 'Partner', 'Dependents', 'tenure',
       'PhoneService', 'MultipleLines', 'InternetService', 'OnlineSecurity',
       'OnlineBackup', 'DeviceProtection', 'TechSupport', 'StreamingTV',
       'StreamingMovies', 'Contract', 'PaperlessBilling', 'PaymentMethod',
       'MonthlyCharges', 'TotalCharges', 'Churn'],
      dtype='object')
```

Figure 7. Deleting variable ‘customerID’ and inspection or remaining variable names.

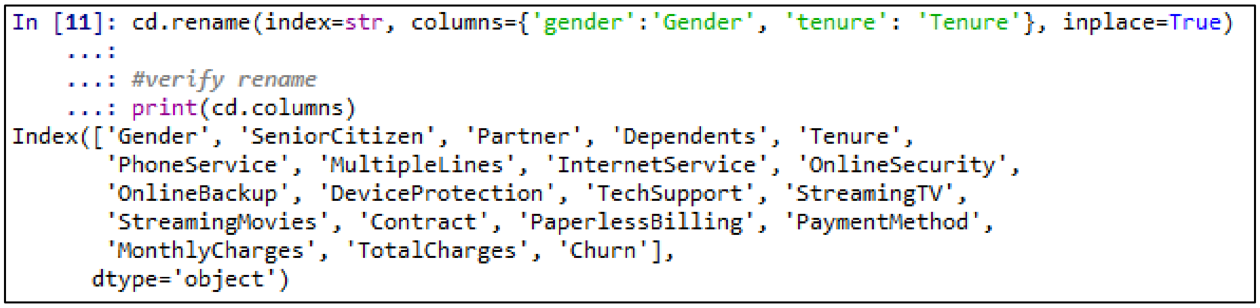
The next step is to search for any missing (null) values. Figure 8 shows that there are no missing values in the DataFrame.



```
In [10]: print(cd.isnull().sum())
gender              0
SeniorCitizen       0
Partner             0
Dependents          0
tenure              0
PhoneService        0
MultipleLines       0
InternetService     0
OnlineSecurity      0
OnlineBackup        0
DeviceProtection    0
TechSupport         0
StreamingTV         0
StreamingMovies     0
Contract            0
PaperlessBilling    0
PaymentMethod       0
MonthlyCharges      0
TotalCharges        0
Churn               0
dtype: int64
```

Figure 8. Verification that there are no nulls in the DataFrame.

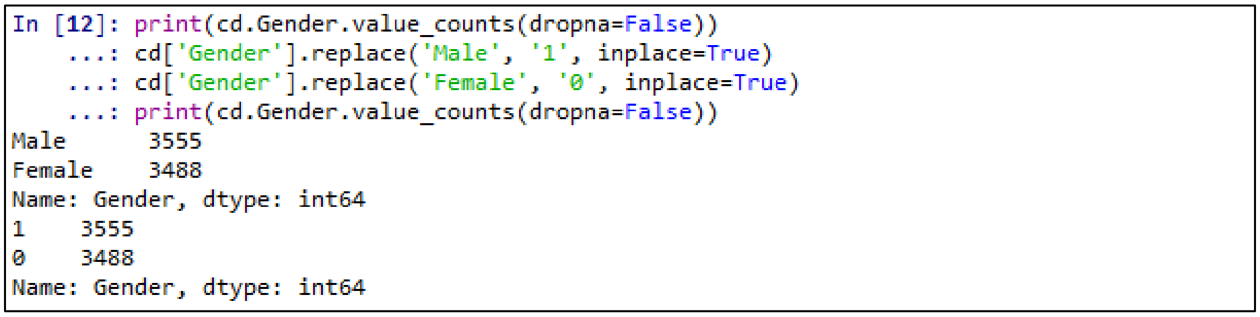
A visual inspection of variable names shows that there are no extra spaces in the variable names but that ‘gender’ and ‘tenure’ begin with lower case characters which is not consistent with the CamelCase naming convention of the other variables. Figure 9 shows these variable names corrected with pandas rename method.



```
In [11]: cd.rename(index=str, columns={'gender':'Gender', 'tenure': 'Tenure'}, inplace=True)
    ...:
    ...: #verify rename
    ...: print(cd.columns)
Index(['Gender', 'SeniorCitizen', 'Partner', 'Dependents', 'Tenure',
       'PhoneService', 'MultipleLines', 'InternetService', 'OnlineSecurity',
       'OnlineBackup', 'DeviceProtection', 'TechSupport', 'StreamingTV',
       'StreamingMovies', 'Contract', 'PaperlessBilling', 'PaymentMethod',
       'MonthlyCharges', 'TotalCharges', 'Churn'],
      dtype='object')
```

Figure 9. Variables ‘gender’ and ‘tenure’ corrected to ‘Gender’ and ‘Tenure’.

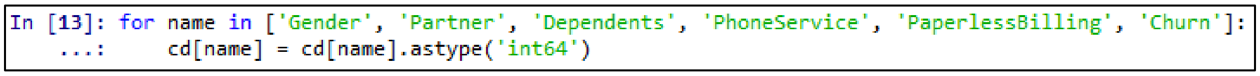
The next step is to transform values in Gender from ‘Male’ and ‘Female’ to 1 or 0 as illustrated in figure 10. The variables Partner, Dependents, PhoneService, PaperlessBilling, and Churn have their ‘Yes’ and ‘No values similarly transformed to 1 and 0.



```
In [12]: print(cd.Gender.value_counts(dropna=False))
    ...: cd['Gender'].replace('Male', '1', inplace=True)
    ...: cd['Gender'].replace('Female', '0', inplace=True)
    ...: print(cd.Gender.value_counts(dropna=False))
Male      3555
Female    3488
Name: Gender, dtype: int64
1    3555
0    3488
Name: Gender, dtype: int64
```

Figure 10. 'Gender' values transformed to 1 for male and 0 for female. 'Yes' and 'No' values in five other variables are also transformed to 1 and 0 in the same way.

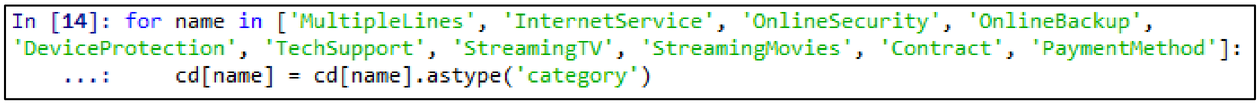
Figure 11 shows the conversion of binary object variables Gender, Partner, Dependents, PhoneService, PaperlessBilling, and Churn into numeric int64 datatypes.



```
In [13]: for name in ['Gender', 'Partner', 'Dependents', 'PhoneService', 'PaperlessBilling', 'Churn']:
    ...:     cd[name] = cd[name].astype('int64')
```

Figure 11. Datatype conversion of six object variables to numeric int64 variables.

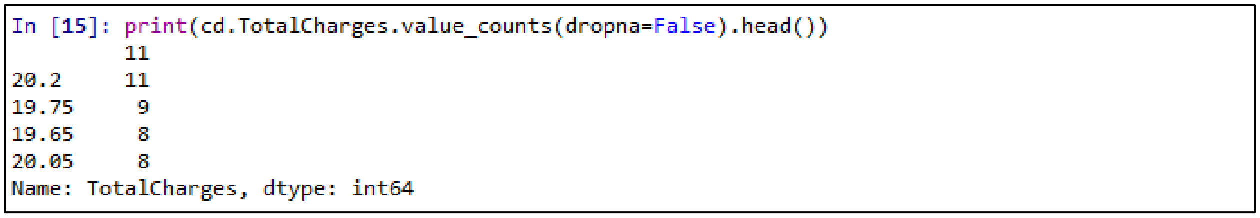
The variables MultipleLines, InternetService, OnlineSecurity, OnlineBackup, DeviceProtection, TechSupport, StreamingTV, StreamingMovies, Contract, and PaymentMethod may be converted from the generic object data type to the category data type to reduce the amount of memory usage that cd requires as shown in figure 12.



```
In [14]: for name in ['MultipleLines', 'InternetService', 'OnlineSecurity', 'OnlineBackup',
'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovies', 'Contract', 'PaymentMethod']:
    ...:     cd[name] = cd[name].astype('category')
```

Figure 12. Object data type variables converted to category data type.

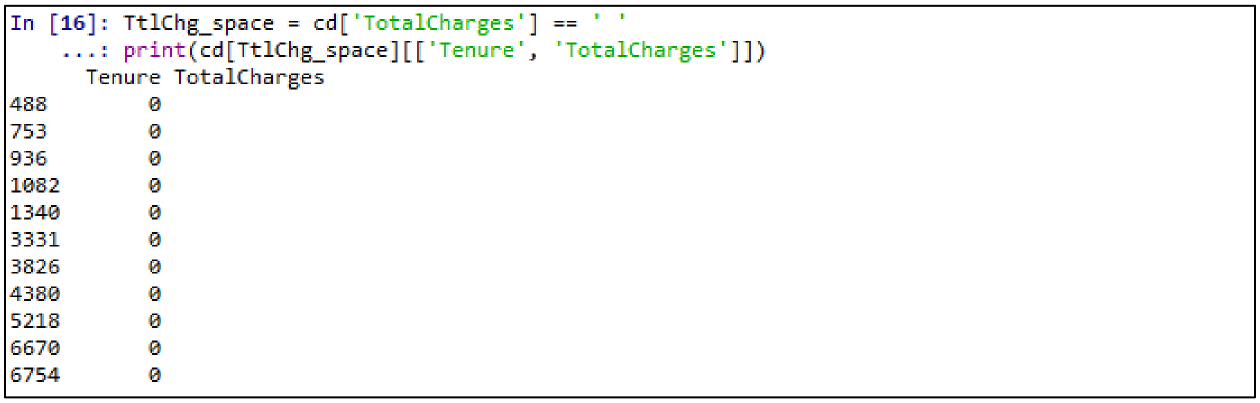
Python's misclassification of TotalCharges signals the presence of erroneous values that would not interpret numerically. There are no missing values, but a search of the most frequent values shows that there are eleven containing a space (figure 13).



```
In [15]: print(cd.TotalCharges.value_counts(dropna=False).head())
         11
20.2     11
19.75     9
19.65     8
20.05     8
Name: TotalCharges, dtype: int64
```

Figure 13. The five most frequent TotalCharges values. Eleven contain a space.

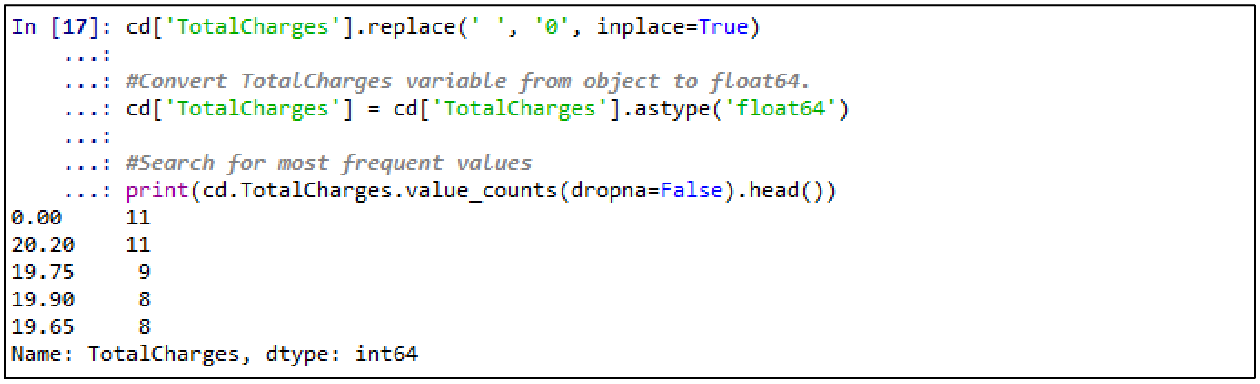
The eleven observations with spaces in TotalCharges also have a value of 0 for Tenure. It is logical that customers would not have accumulated total charges while still in their first month of service (figure 14).



```
In [16]: TtlChg_space = cd['TotalCharges'] == ' '
    ...: print(cd[TtlChg_space][['Tenure', 'TotalCharges']])
      Tenure TotalCharges
488        0
753        0
936        0
1082       0
1340       0
3331       0
3826       0
4380       0
5218       0
6670       0
6754       0
```

Figure 14. Eleven values with spaces in TotalCharges also have 0 months of Tenure.

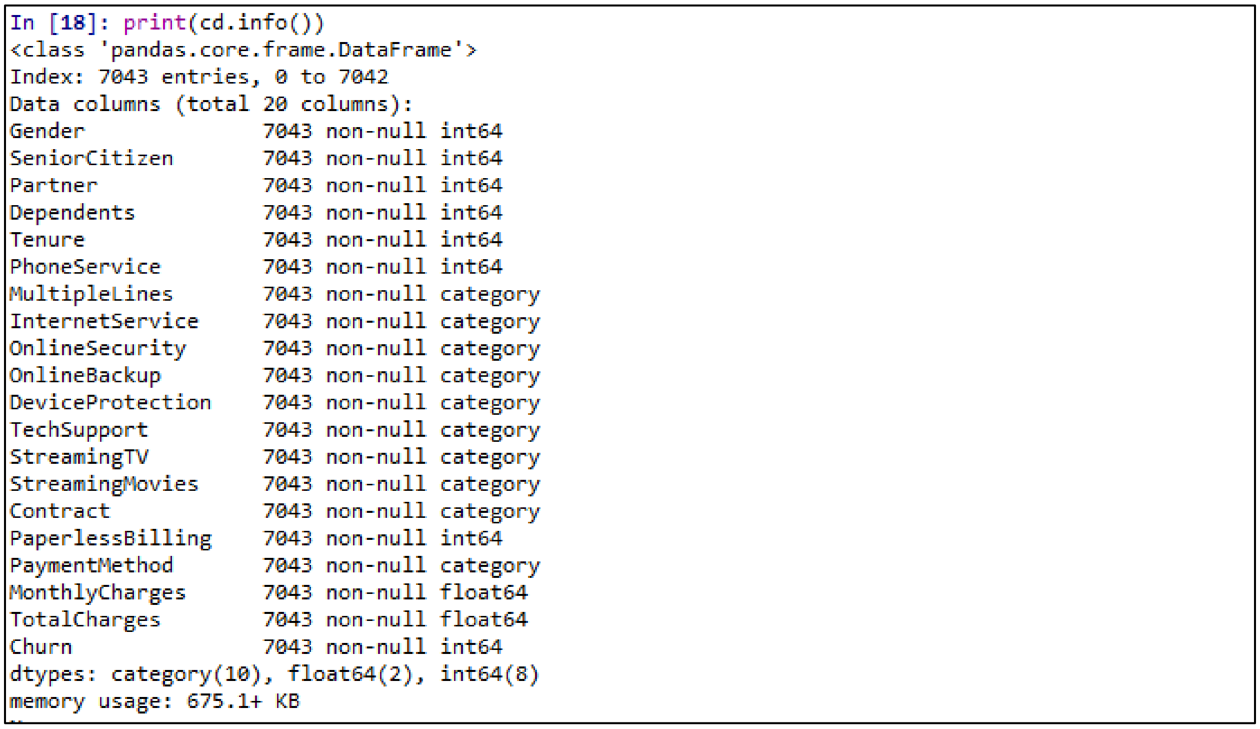
Figure 15 illustrates imputing the eleven TotalCharges with spaces to 0, converting the datatype to float64, and displaying the five most frequent values.



```
In [17]: cd['TotalCharges'].replace(' ', '0', inplace=True)
    ...:
    ...: #Convert TotalCharges variable from object to float64.
    ...: cd['TotalCharges'] = cd['TotalCharges'].astype('float64')
    ...:
    ...: #Search for most frequent values
    ...: print(cd.TotalCharges.value_counts(dropna=False).head())
0.00     11
20.20    11
19.75     9
19.90     8
19.65     8
Name: TotalCharges, dtype: int64
```

Figure 15. Eleven TotalCharge values with spaces imputed to 0. Most frequent values now show 0.00.

Figure 16 shows the completed conversion of variables with ten qualitative category variables, two quantitative float64 variables, and eight quantitative int64 variables. Note the 40.1% reduction in memory usage from 1.1+ MB to 675.1+ KB.



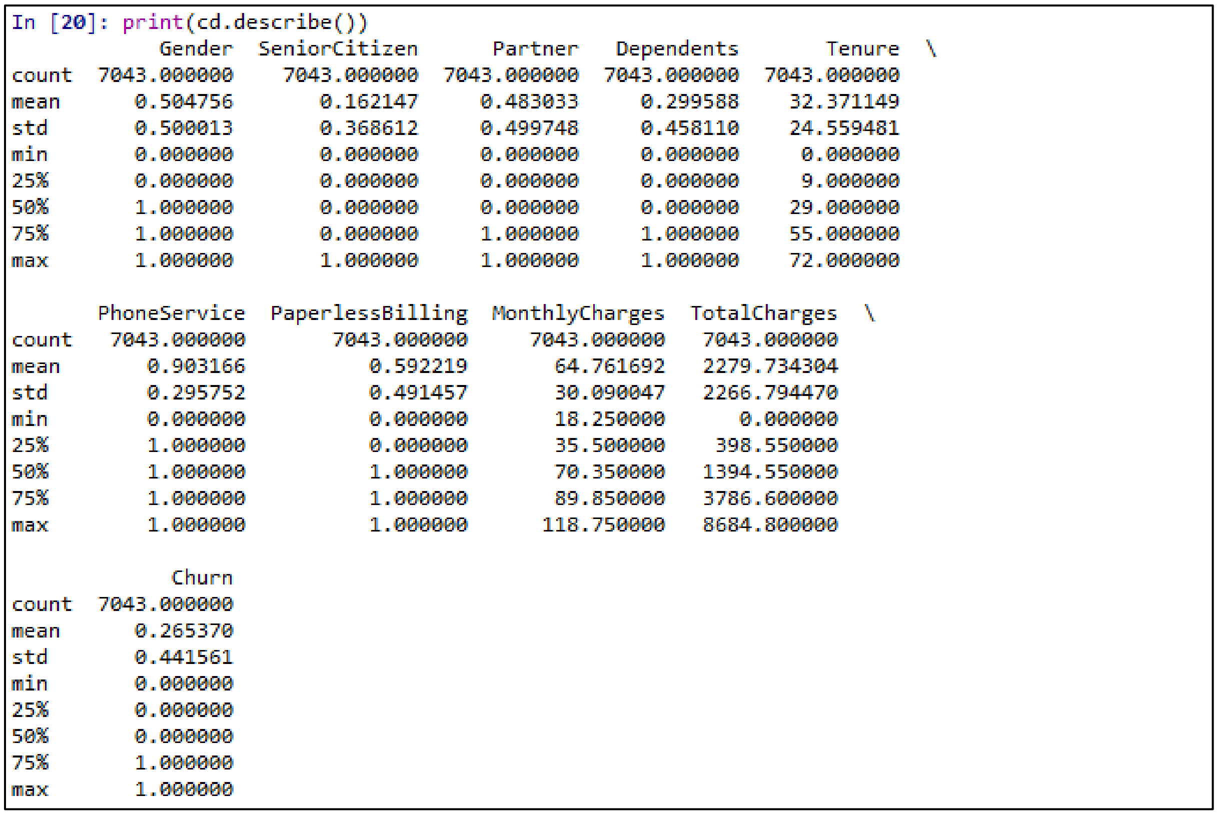
```
In [18]: print(cd.info())
<class 'pandas.core.frame.DataFrame'>
Index: 7043 entries, 0 to 7042
Data columns (total 20 columns):
Gender              7043 non-null int64
SeniorCitizen       7043 non-null int64
Partner             7043 non-null int64
Dependents          7043 non-null int64
Tenure              7043 non-null int64
PhoneService        7043 non-null int64
MultipleLines       7043 non-null category
InternetService     7043 non-null category
OnlineSecurity      7043 non-null category
OnlineBackup        7043 non-null category
DeviceProtection    7043 non-null category
TechSupport         7043 non-null category
StreamingTV         7043 non-null category
StreamingMovies     7043 non-null category
Contract            7043 non-null category
PaperlessBilling    7043 non-null int64
PaymentMethod       7043 non-null category
MonthlyCharges      7043 non-null float64
TotalCharges        7043 non-null float64
Churn               7043 non-null int64
dtypes: category(10), float64(2), int64(8)
memory usage: 675.1+ KB
```

Figure 16. Variables converted to new data types. Memory usage is reduced by 40.1% due to conversion of object datatypes to category datatypes.

Appendix B contains a copy of the cleaned customer data in .xlsx format named 'Appendix_B_Cleaned_WA_Fn-UseC_-Telco-Customer-Churn.xlsx'.

III. Data Analysis

I. Univariate analysis – Continuing with DataFrame cd, Python's matplotlab and seaborn packages provide visualizations of the univariate statistics for each variable. Figure 17 shows the summary statistics of interest for the ten numerical variables. As expected, the min and max values for the binary variables are 0 and 1. The mean for the binary variables also provides an indication of the distribution. For example, Gender's mean of 0.50 follows the expected distribution of sexes among the general population. By contrast, SeniorCitizen's mean of only 0.16 indicates a more skewed distribution. The standard deviation can help describe the spread of the data around the mean.

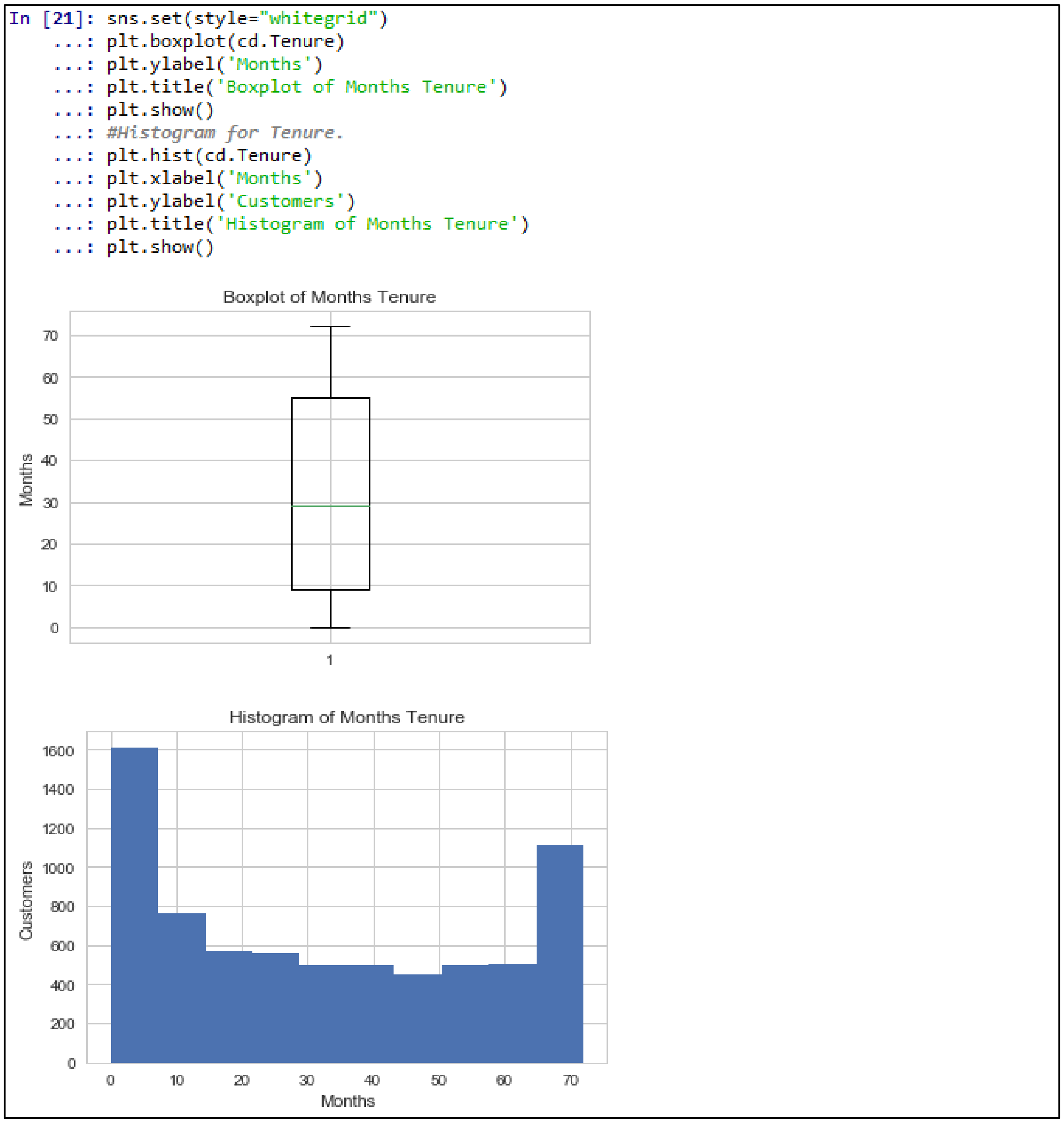


```
In [20]: print(cd.describe())
            Gender  SeniorCitizen      Partner   Dependents       Tenure  \
count  7043.000000    7043.000000  7043.000000  7043.000000  7043.000000
mean      0.504756       0.162147     0.483033     0.299588    32.371149
std       0.500013       0.368612     0.499748     0.458110    24.559481
min       0.000000       0.000000     0.000000     0.000000     0.000000
25%       0.000000       0.000000     0.000000     0.000000     9.000000
50%       1.000000       0.000000     0.000000     0.000000    29.000000
75%       1.000000       0.000000     1.000000     1.000000    55.000000
max       1.000000       1.000000     1.000000     1.000000    72.000000

       PhoneService  PaperlessBilling  MonthlyCharges  TotalCharges  \
count   7043.000000       7043.000000     7043.000000   7043.000000
mean       0.903166          0.592219       64.761692   2279.734304
std        0.295752          0.491457       30.090047   2266.794470
min        0.000000          0.000000       18.250000      0.000000
25%        1.000000          0.000000       35.500000    398.550000
50%        1.000000          1.000000       70.350000   1394.550000
75%        1.000000          1.000000       89.850000   3786.600000
max        1.000000          1.000000      118.750000   8684.800000

             Churn
count  7043.000000
mean      0.265370
std       0.441561
min       0.000000
25%       0.000000
50%       0.000000
75%       1.000000
max       1.000000
```

Figure 17. Summary statistics of numerical variables. As expected, the min and max of binary variables is 0 and 1 respectively.

The boxplots and histograms in figures 18 through 21 illustrate more information for the remaining numerical variables Tenure, MonthlyCharges, and TotalCharges. The boxes represent the observations in the inter quartile range (IQR) between the 25th and 75th percentile. The horizontal line inside the box shows the 50th percentile. The top and bottom horizontal lines at the ends of the whiskers represent the minimum and maximum extents of the data beyond 1.5 times the IQR. Values plotted above or below the extent lines would be outliers however, none of the three variables show outliers. The histograms show the distribution of values in bins.

In figure 18, Tenure's histogram reveals that there are many new customers compared to middle and long-term customers. It also indicates a potential problem for the analysis. The cluster of long-term customers in the highest bin makes it appear that there are no customer accounts older than six years, which is not reasonable. This erroneous data can have several causes: a software bug may prevent this data point from increasing past 72, a software upgrade or systems conversion six years ago may have caused Tenure to start counting from zero, or there may be a management decision to not count past six years. Regardless of the cause, the inaccuracy can affect the analysis and needs to be addressed.



```
In [21]: sns.set(style="whitegrid")
    ...: plt.boxplot(cd.Tenure)
    ...: plt.ylabel('Months')
    ...: plt.title('Boxplot of Months Tenure')
    ...: plt.show()
    ...: #Histogram for Tenure.
    ...: plt.hist(cd.Tenure)
    ...: plt.xlabel('Months')
    ...: plt.ylabel('Customers')
    ...: plt.title('Histogram of Months Tenure')
    ...: plt.show()
```

Figure 18. Python code, boxplot, and histogram of variable Tenure. The highest Tenure bin is suspiciously large.

Fortunately, Tenure's reasonableness can be verified with a new variable TenureCalc created by dividing TotalCharges by MonthlyCharges. Although discounts and penalties such as late fees will prevent an exact calculation, the results provide a reasonable smoothing of the 72-month values without substantially altering the mean and standard deviation (figure19). Also, the vertical size of the IQR shrinks and its position moves down the vertical scale in response to a more realistic left skew of the data.

```
            Tenure   TenureCalc
count  7043.000000  7043.000000
mean     32.371149    32.368025
std      24.559481    24.596800
min       0.000000     0.000000
25%       9.000000     9.000000
50%      29.000000    29.000000
75%      55.000000    55.000000
max      72.000000    79.000000
```

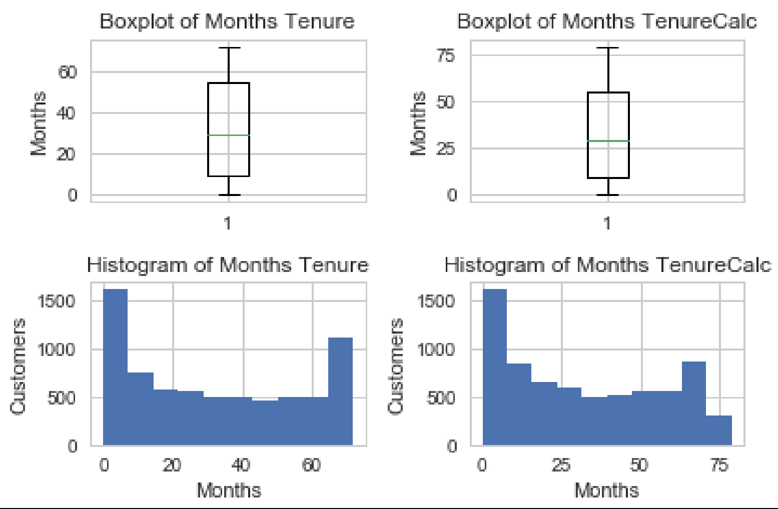


Figure 19. Summary statistics, boxplots, and histograms of variables Tenure and TenureCalc. TenureCalc's max increases by 7 while min, 25%, 50% and 75% remain the same. Mean and std change is very small. The smoothing effect on 72-month Tenures is clearly visible.

Contrast the skew of data in TenureCalc with the boxplot and histogram for TotalCharges shown in figure 20. The IQR in the boxplot occupies a much smaller proportion of the vertical scale and is situated near the bottom indicating a skewed distribution of revenue. The TotalCharges histogram confirms this showing an extremely left-skewed distribution curve with a low population of high revenue customers and very large population of low revenue customers.

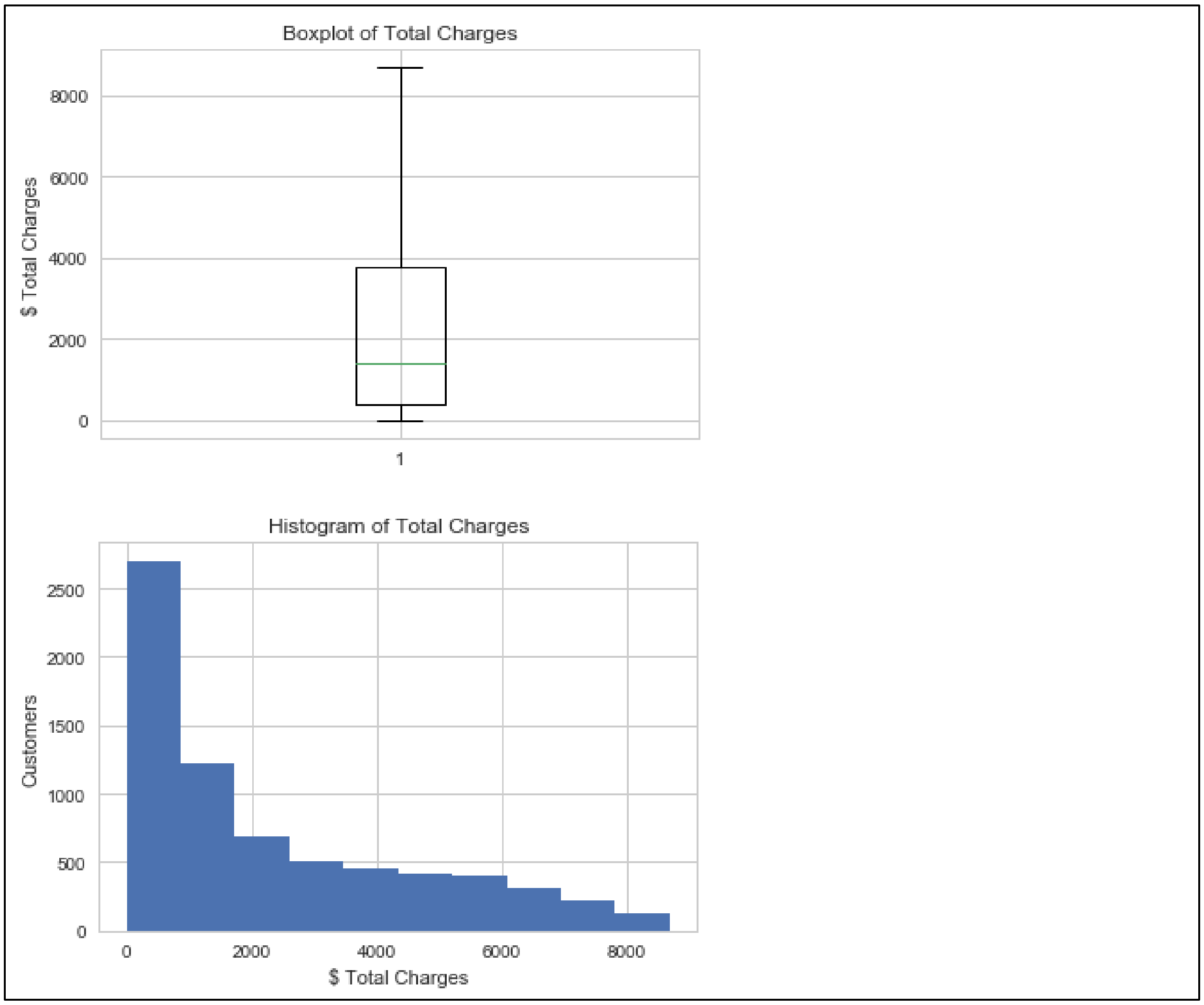


Figure 20. Boxplot and histogram of variable TotalCharges.

The MonthlyCharges boxplot hints at the relationship between TenureCalc and TotalCharges by showing that the largest proportion of the IQR box lies between the 50th and 25th percentiles and the overall position of the IQR box is low on the vertical line (figure 21). Again, the histogram clarifies the relationship. It also shows that the number of customers paying the minimum amount per month is far greater than any of the higher cost MonthlyCharge bins. There is an obvious pricing gap between the lowest monthly charges and the mid-range monthly charges.

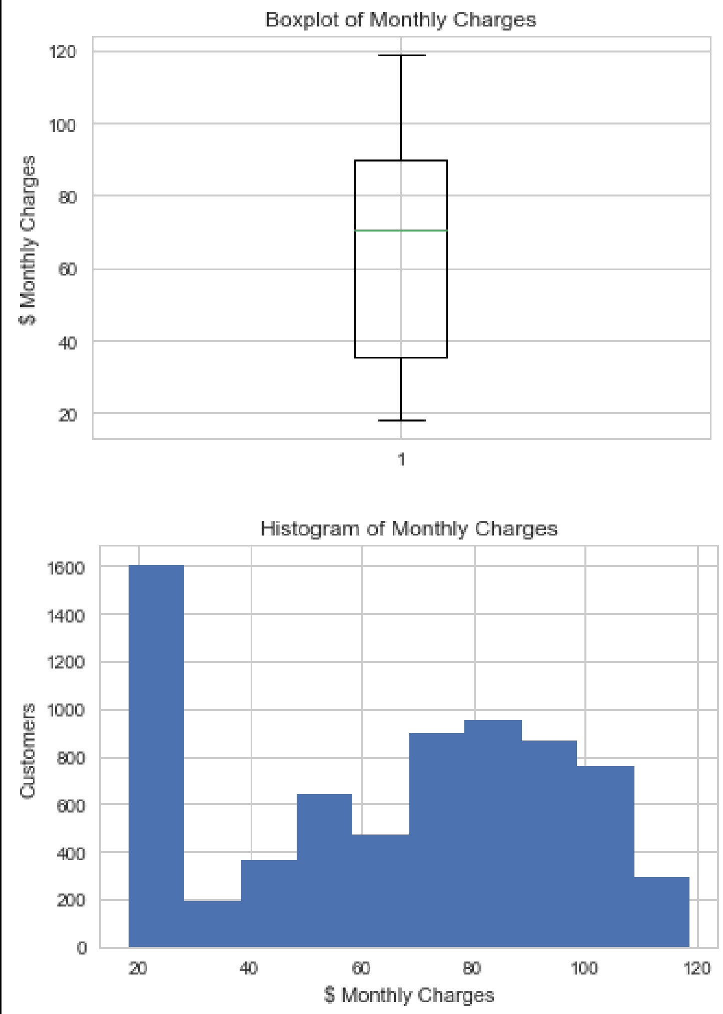


Figure 21. Boxplot and histogram of variable MonthlyCharges. There is a large gap between the lowest and the mid-range pricing plans.

Pandas value_counts method is useful for visualizing the remaining quantitative variables as well as the qualitative categorical variables.

The variables Gender, SeniorCitizen, Partner, and Dependents help define the customer demographically. Figure 22 shows the count of customers by gender. The split between Female and Male customers is almost even. There are 1.9% more male customers than female customers.

```
In [26]: fig, ax = plt.subplots()
    ...: male = mpatches.Patch(color='C0', label='Male')
    ...: female = mpatches.Patch(color='C1', label='Female')
    ...: plt.legend(handles=[female,male])
    ...: print(cd.Gender.value_counts(dropna=False).plot(kind='barh', figsize=(11, 2), grid=True,
zorder=2, title='Gender'))
    ...: plt.xlabel('Customers')
    ...: plt.ylabel('Sex')
    ...: plt.legend(handles=[male,female], bbox_to_anchor=(0., 1.08, 1., .102), ncol=2,
borderaxespad=0.)
    ...: for i in ax.patches:
    ...:     ax.text(i.get_width()+50, i.get_y()+.3, \
    ...:             str(round((i.get_width()), 2)), fontsize=11)
    ...:
    ...: plt.margins(0.09)
    ...: ax.invert_yaxis()
    ...: plt.show()
AxesSubplot(0.125,0.125;0.775x0.755)
```

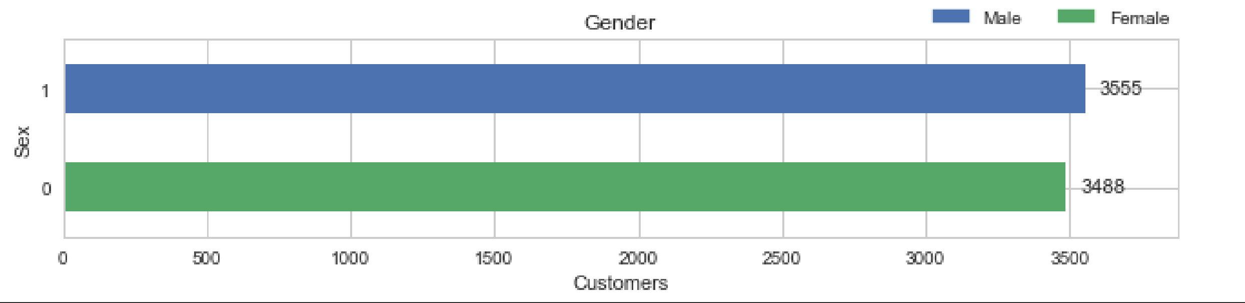


Figure 22. Python code and bar plot showing value counts of variable Gender. There is an almost even split between male and female customers. Python code of the remaining univariate plots is similar.

Figure 23 shows that 16.2% of customers are senior citizens.

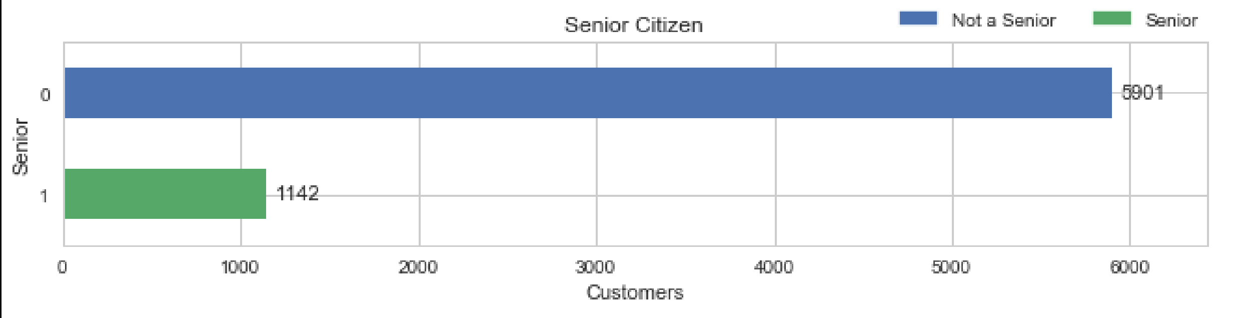


Figure 23. Value counts and bar plot of variable SeniorCitizen showing that senior citizens make up 16.2% of customers.

Figure 24 shows that more customers do not have a partner than do have a partner, but the difference is only 7%.

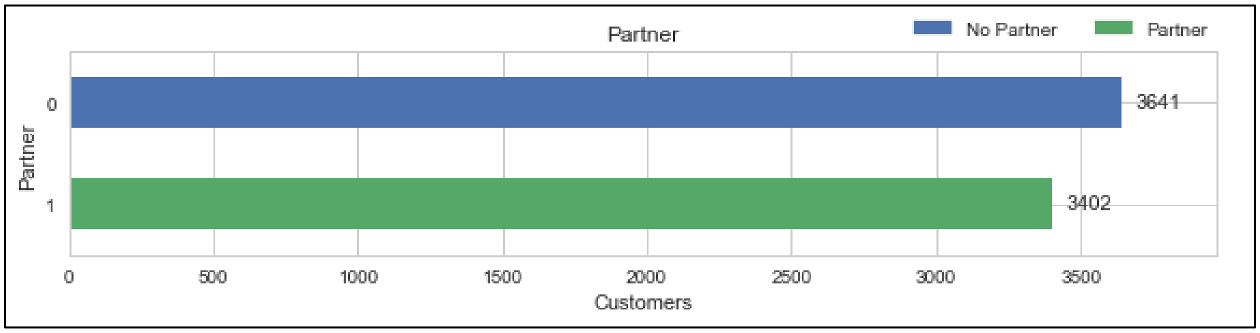


Figure 24. Value counts and bar plot of variable Partner. Most customers do not have a partner, but the difference is small.

The last demographic variable is Dependents. Figure 25 shows that more than two-thirds of customers (70%) do not have dependents.

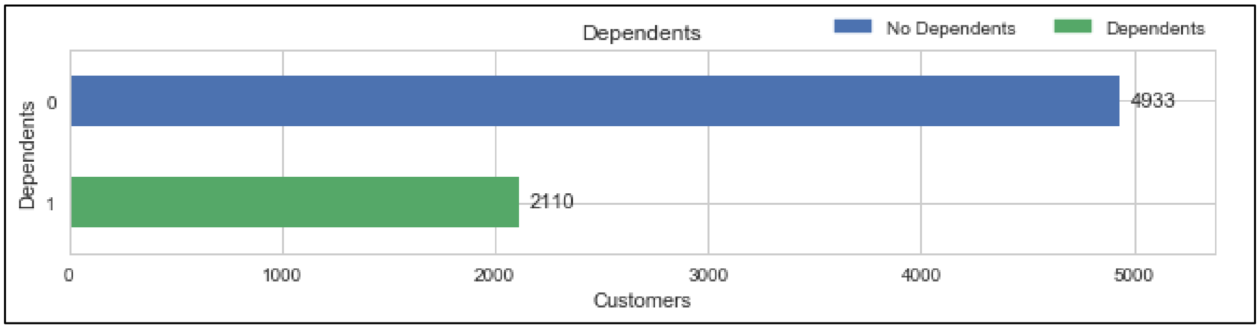


Figure 25. Value counts and bar plot of variable Dependents. 70% of customers do not have dependents.

The variables PhoneService and MultipleLines help define customer engagement for the company's core product offering. Figure 26 shows that, as expected for a telecom provider, 90% of customers have phone service.

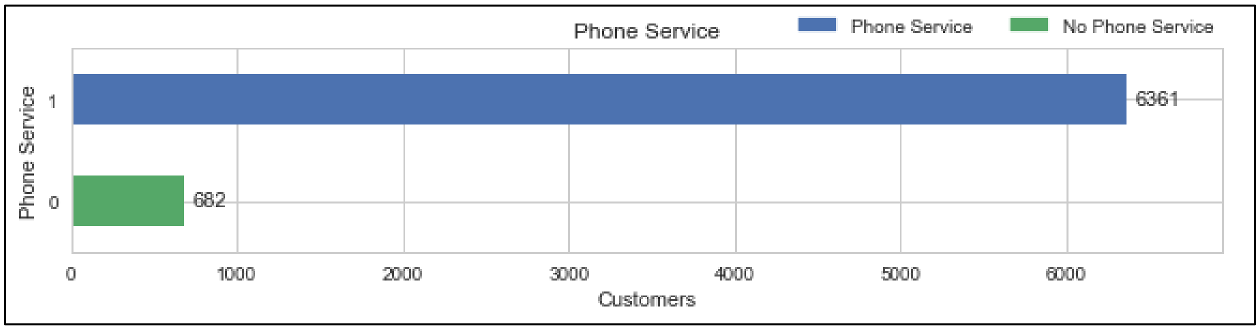


Figure 26. Value counts and bar plot of variable PhoneService. 90% of customers have phone service, which is expected for a telecom provider.

The variable MultipleLines helps define the level of service for customers that have phone service. Figure 27 shows that 46.7% of customers with phone service have multiple lines while 53.3% do not. The count of 682 without phone service matches the count from the PhoneService variable adding validity to the quality of the data.

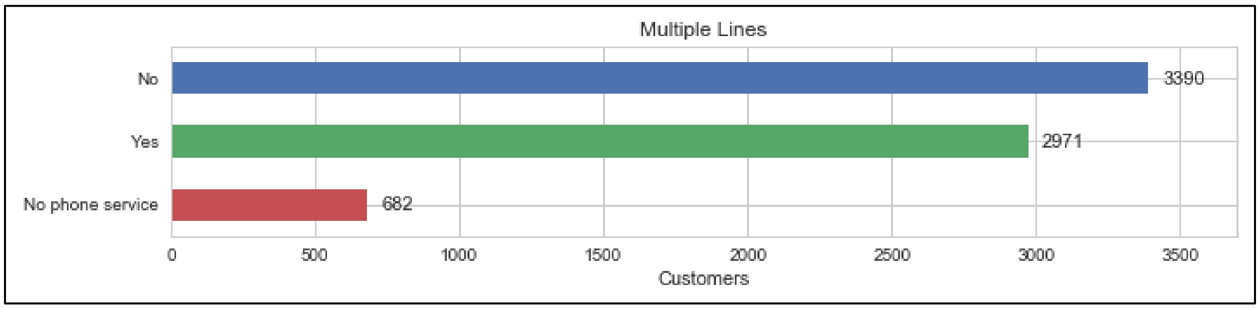


Figure 27. Value counts and bar plot of variable MultipleLines. Of the customers who have phone service, 47% have more than one line.

The variables InternetService, OnlineSecurity, OnlineBackup, DeviceProtection, TechSupport, StreamingTV, and StreamingMovies delve into customer engagement of the company's online product offerings. All of these variables feature the value 'No internet service' with a count of 1,526 (21.7%) of all customer observations. This agreement across multiple variables lends credibility to the data.

The variable InternetService indicates how many customers purchase internet access and which technology they chose. Figure 28 shows that 56.1% of internet service customers take advantage of faster fiber optic service.

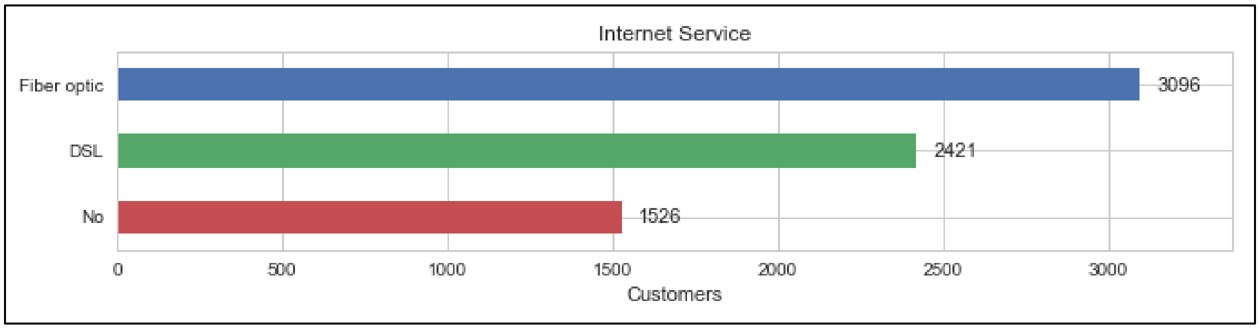


Figure 28. Value counts and bar plot of variable InternetService. 21.7% of customers do not purchase internet service. Of those that purchase internet services, most purchase faster fiber optic service.

Figure 29 shows that 63.4% of customers with internet service opt out of the company's online security service while only 36.6% take advantage of this service.

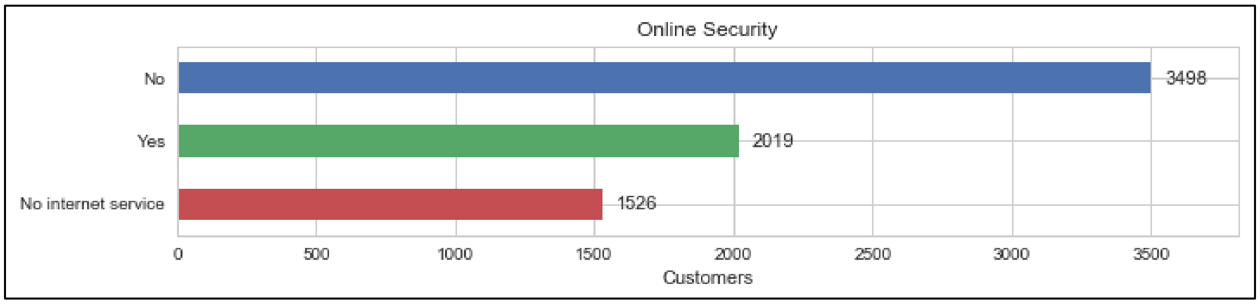


Figure 29. Value counts and bar plot of variable OnlineSecurity. Only 36.6% of customers that purchase internet access take advantage of this service.

Online backup has better acceptance among those who purchase online service than Online security. Figure 30 shows that 44.0% purchase online backup as part of their online services.

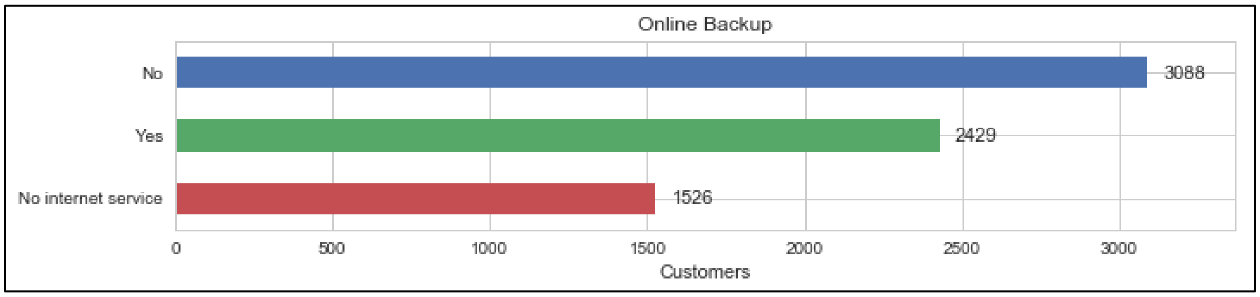


Figure 30. Value counts and bar plot of variable OnlineBackup. 44% of customers that purchase online services also purchase online backup.

Device protection has almost identical appeal with online service customers as online backup. Figure 31 shows that 43.9% choose this option (compared to 44.0% for online backup) while 56.1% do not.

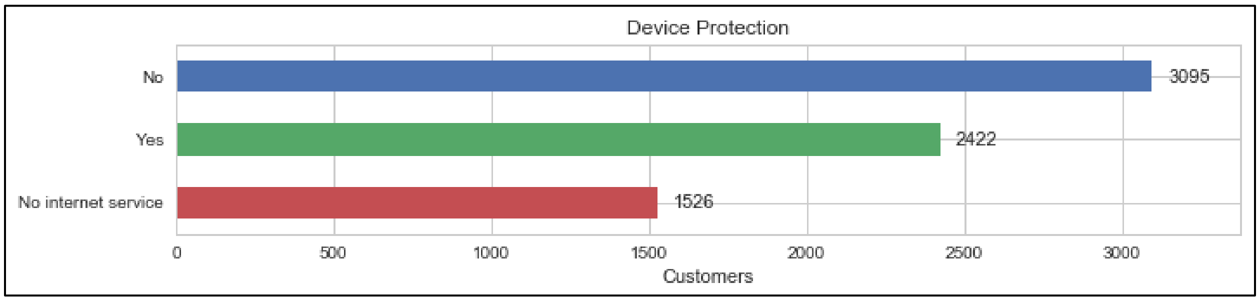


Figure 31. Value counts and bar plot of variable DeviceProtection. This option has almost identical appeal with online service customers as online backup.

Tech support is about as popular as the company's online security offering. Figure 32 shows that 63.0% of online service customers chose not to purchase this option and support themselves. Only 37.0% opt for this service.

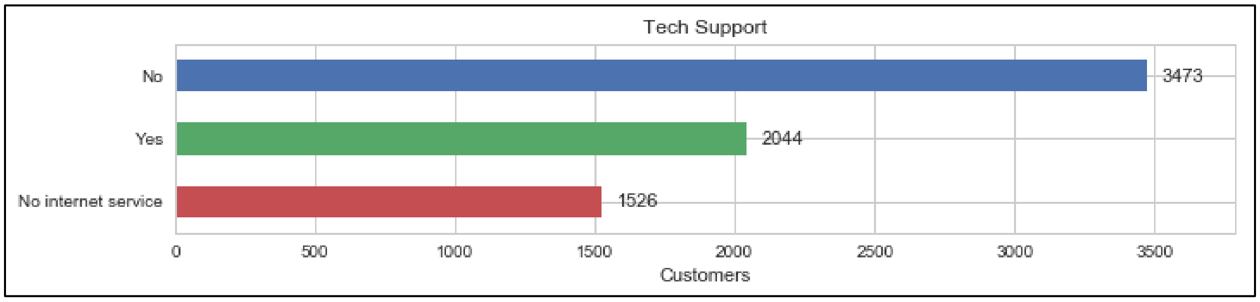


Figure 32. Value counts and bar plot of variable TechSupport.

Close to half (49.1%) of the customers who purchase internet service also purchase Streaming TV. Figure 33 shows how the split is close to even with internet service customers who do not subscribe to Streaming TV (50.9%).

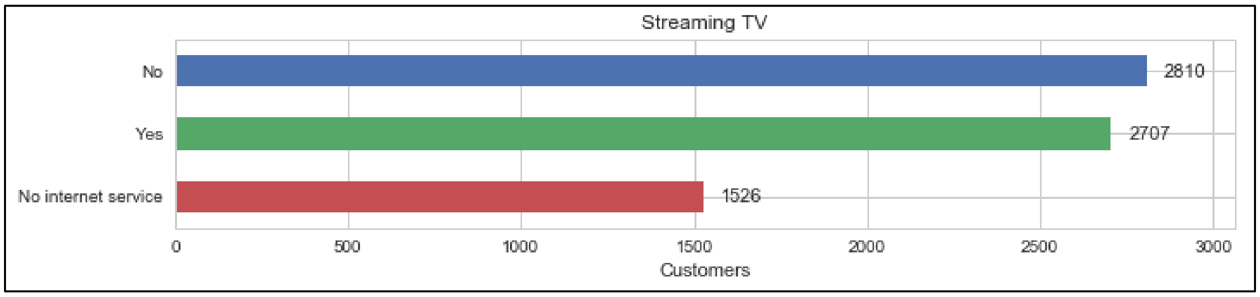


Figure 33. .Value counts and bar plot of variable StreamingTV. A little less than half of internet service customers purchase this service.

Streaming movies edges out Streaming TV in popularity by 0.4% among internet service customers with 49.5% opting for this service. Figure 34 illustrates how those who do not opt for streaming movies are only slightly greater in number.

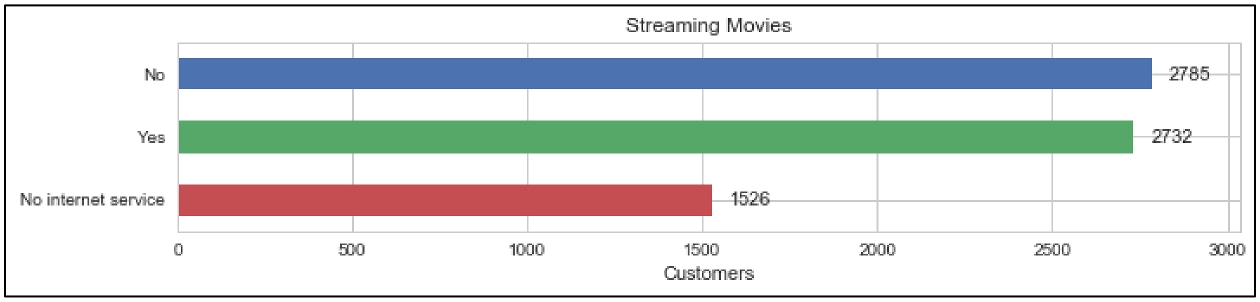


Figure 34. Value counts and bar plot of variable StreamingMovies. The difference between internet service customers opting in or out of streaming movies is negligible.

The qualitative variables Contract, PaperlessBilling, and PaymentMethod speak to the customer's financial relationship with the company.

Figure 35 indicates that more than half (55.0%) of customers are signed up for services with month-to month agreements. The remaining customers are signed up with two-year (24.1%) and one-year (20.9%) contracts.

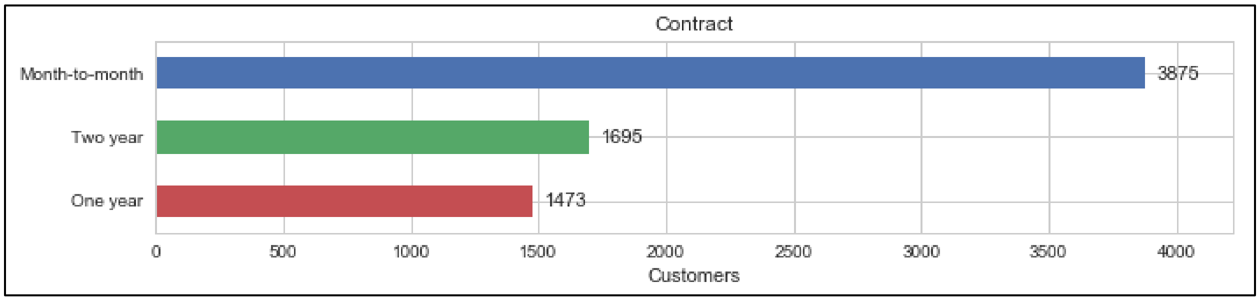


Figure 35. Value counts and bar plot of variable Contract. Most customers do not have a contract and are signed up as month-to-month.

Figure 36 shows that most customers (59.2%) have responded positively to paperless billing and opted for this option.

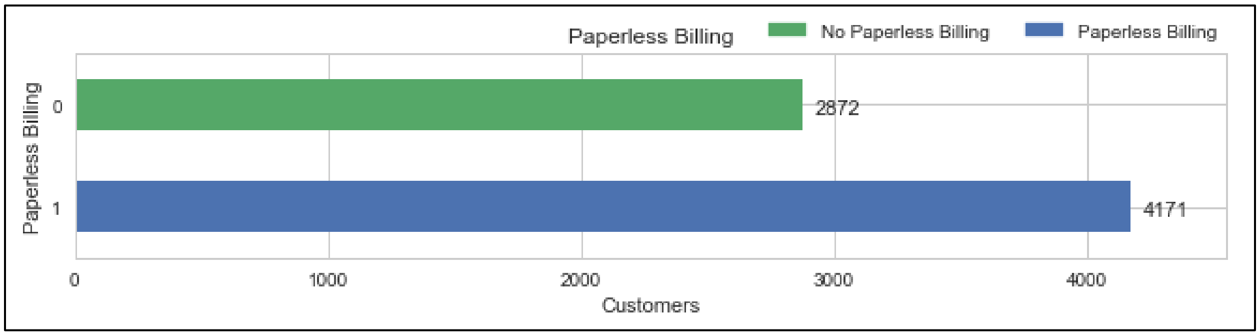


Figure 36. Value counts and bar plot of variable PaperlessBilling. Most customers have opted to take advantage of this convenience.

Figure 37 shows the various payment methods customers use. More than two-third (77.1%) of customer's chose electronic methods leaving the company to process paper checks for the remaining 22.9%.

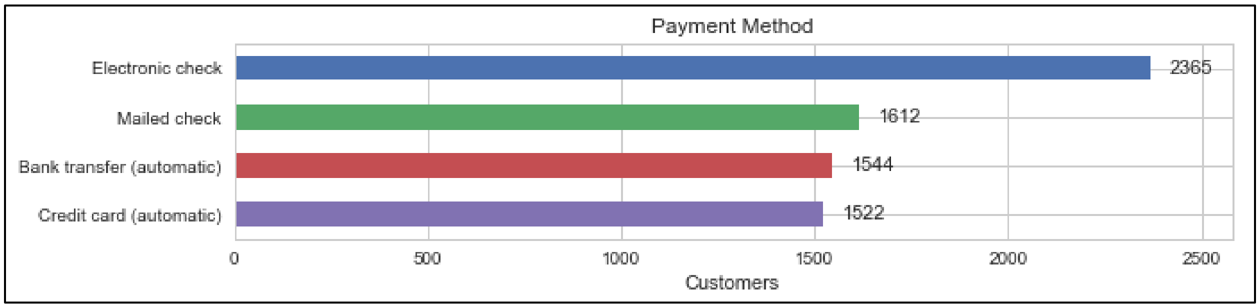


Figure 37. Value counts and bar plot of variable PaymentMethod. Most payments are by electronic means.

Finally, figure 38 shows that more than one-third (36.1%) of all customers have churned (the dependent variable). This is very great concern.

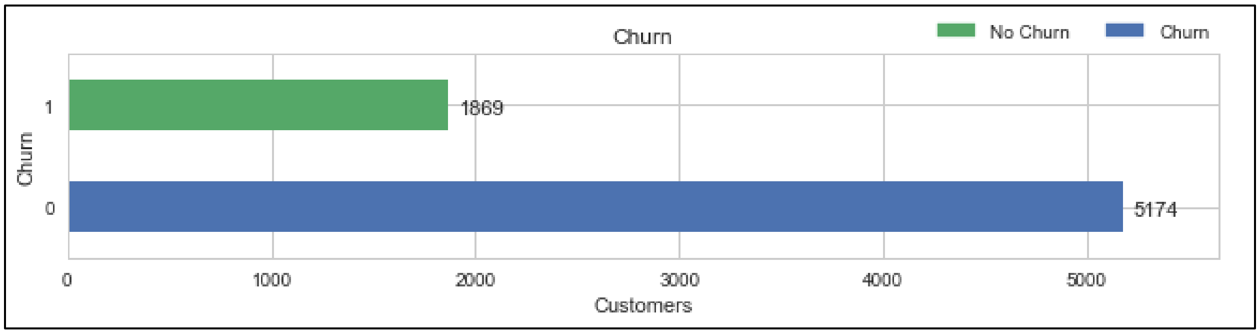


Figure 38. Value counts and bar plot of variable Churn. 36.1% of customers churned.

J. Bivariate analysis – The dependent variable Churn is plotted in relation to each of the other variables. Seaborn violin plots illustrate the effect that each of the quantitative continuous variables TenureCalc, MonthlyCharges, and TotalCharges have on the dependent variable Churn.

Figure 39 illustrates Churn's relationship to TenureCalc. The larger volume of the blue left plot versus the green right plot clearly shows that most customers have not churned. The shape of the left plot reflects the previous univariate histogram with the largest concentrations of customers either new or long-term. The shape of the smaller right plot confirms that short tenured customers churn more than long tenured customers. TenureCalc may be a good predictor of Churn.

```
In [44]: sns.violinplot(x='Churn',y='TenureCalc', data=cd, scale='count')
    ...: plt.xlabel('Churn: 0=No 1=Yes')
    ...: plt.ylabel('TenureCalc')
    ...: plt.title('Customer Churn by TenureCalc')
    ...: plt.show()
```

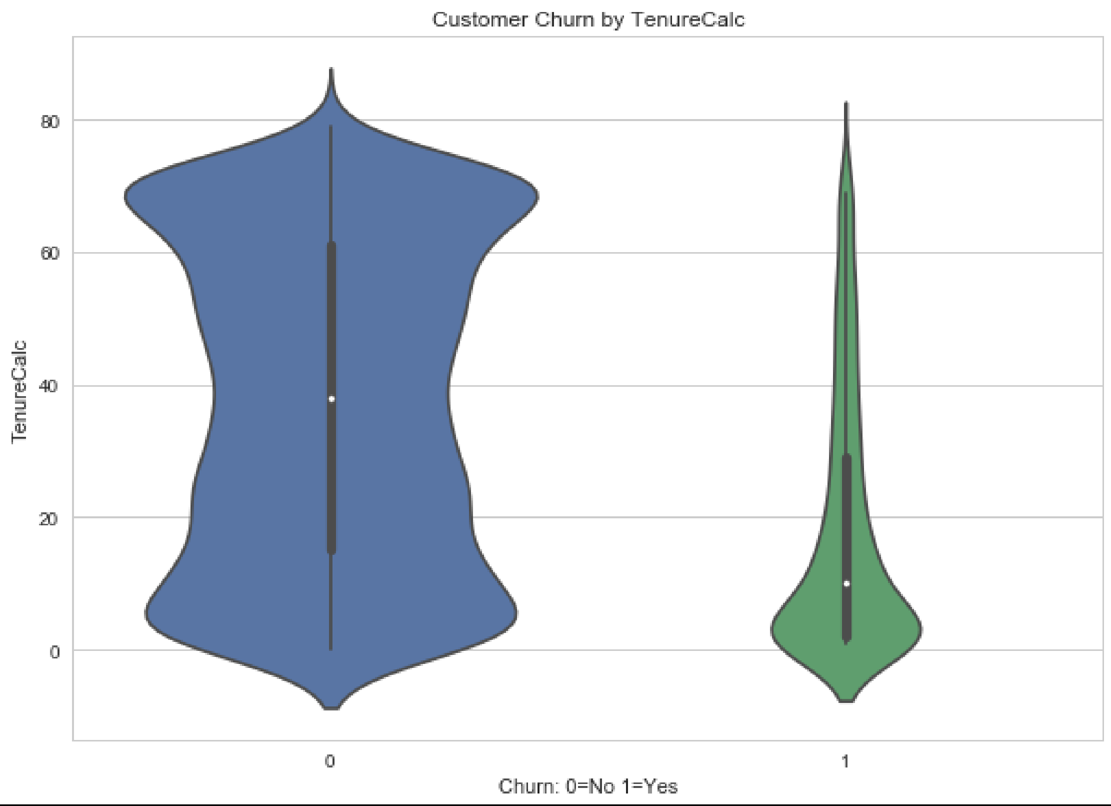


Figure 39. Violin plot of continuous quantitative variable TenureCalc and Churn. As months of tenure increase, customer churn decreases. TenureCalc is likely correlated with Churn. Python code for the remaining quantitative variables is similar.

Figure 40 illustrates how higher MonthlyCharges can correlate with greater churn. The shape of the blue left plot mimics the previous univariate histogram since more customers have the lowest monthly charge offered by the company. The lack of width in the lower $20 area of the right plot suggests that customers with lower monthly bills churn less. MonthlyCharges may be a good predictor of Churn.

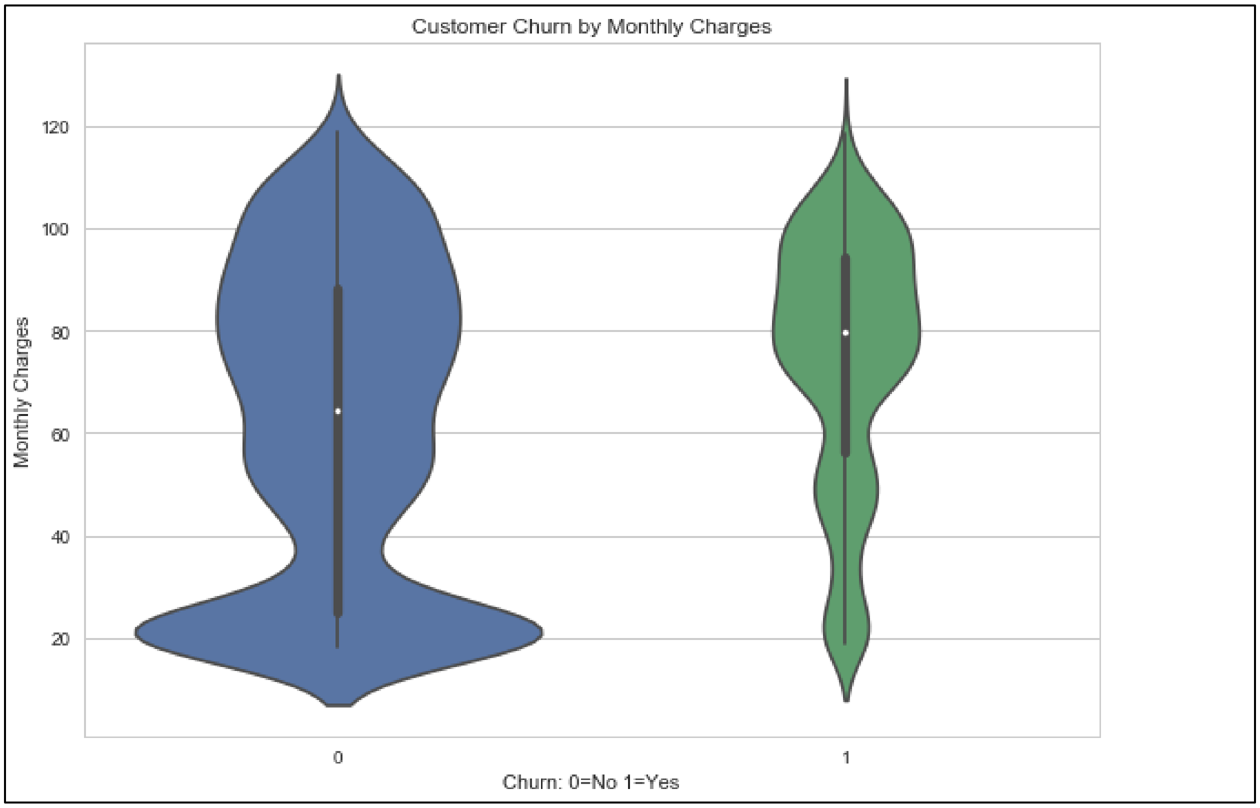


Figure 40. Violin plot of continuous quantitative variable MonthlyCharges and Churn. Customer churn increases among customers with higher monthly charges. Higher monthly bills may cause customers to shop for a better deal elsewhere. MonthlyCharges is likely correlated with Churn.

Figure 41 shows the relationship of Churn to TotalCharges. There are clearly more low-TotalCharge customers churning than high-TotalCharge customers. However, the similar shapes between left and right plots suggests that the same characteristic is true for those that have not churned. TotalCharges is not a good predictor of Churn.

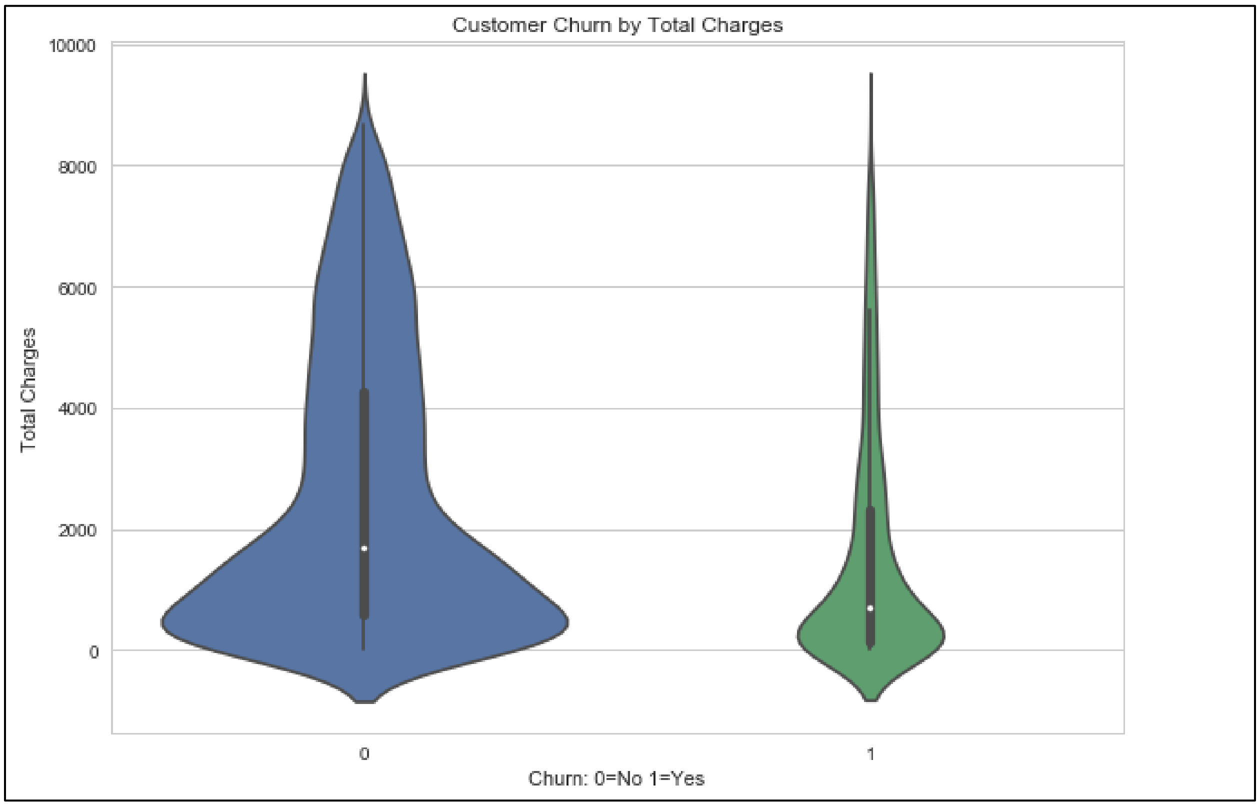
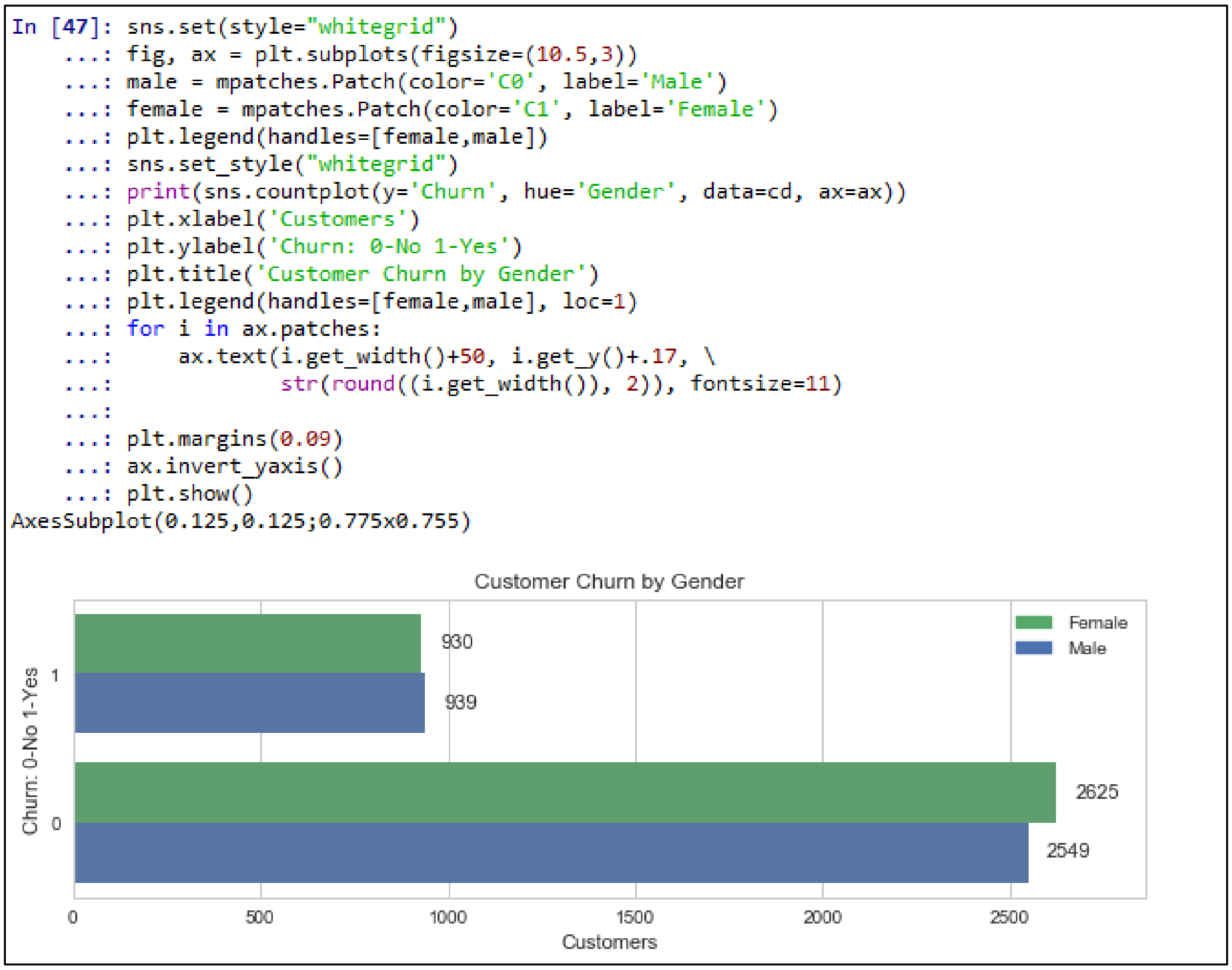


Figure 41. Violin plot of continuous quantitative variable TotalCharges and Churn. Customer churn decreases among customers with higher total charges, However, the two plots are similar in shape suggesting a lack of correlation between TotalCharges and Churn.

The customer demographic variables follow. Figure 42 shows the variables Churn and Gender. The difference between males and females in the churned and non-churned populations is extremely small. Gender is not be a good predictor.



```
In [47]: sns.set(style="whitegrid")
    ...: fig, ax = plt.subplots(figsize=(10.5,3))
    ...: male = mpatches.Patch(color='C0', label='Male')
    ...: female = mpatches.Patch(color='C1', label='Female')
    ...: plt.legend(handles=[female,male])
    ...: sns.set_style("whitegrid")
    ...: print(sns.countplot(y='Churn', hue='Gender', data=cd, ax=ax))
    ...: plt.xlabel('Customers')
    ...: plt.ylabel('Churn: 0-No 1-Yes')
    ...: plt.title('Customer Churn by Gender')
    ...: plt.legend(handles=[female,male], loc=1)
    ...: for i in ax.patches:
    ...:     ax.text(i.get_width()+50, i.get_y()+.17, \
    ...:             str(round((i.get_width()), 2)), fontsize=11)
    ...:
    ...: plt.margins(0.09)
    ...: ax.invert_yaxis()
    ...: plt.show()
AxesSubplot(0.125,0.125;0.775x0.755)
```

Figure 42. Python code, and bar plot of variables Churn and Gender. Gender does not appear to differ among churned/non-churned customers. Python code for the remaining bivariate analysis plots is similar.

Figure 43 illustrates the distribution of SeniorCitizen related to Churn. Senior citizens comprise a larger proportion of churned customers than of non-churned customers. SeniorCitizen may be a predictor.

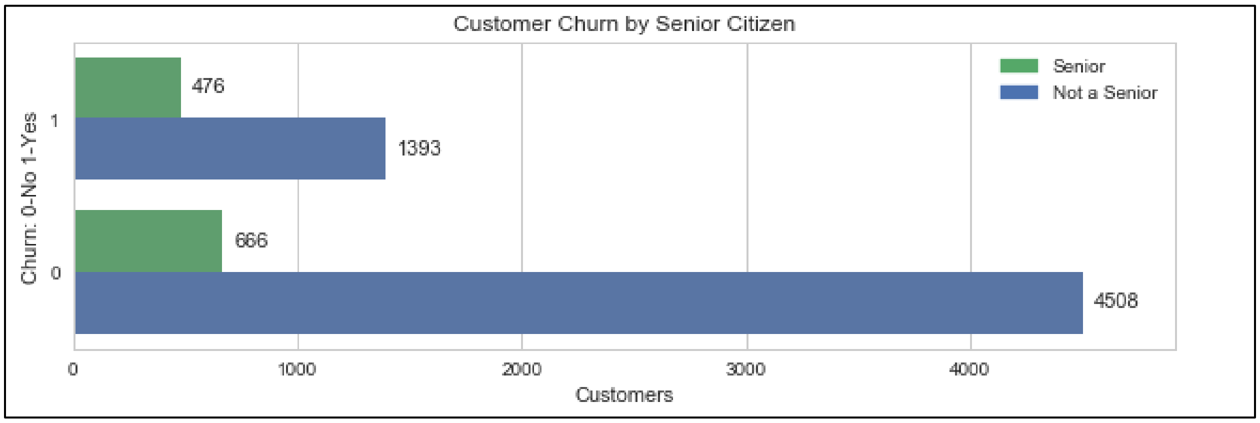


Figure 43. Bar plot of variables Churn and SeniorCitizen. As a group, senior citizens churn more than other customers.

Customers with partners make up about half of churn customers as opposed to making up most of the non-churned customers (figure 44). This difference suggests that single customers are more likely to switch companies. Partner may be a predictor.

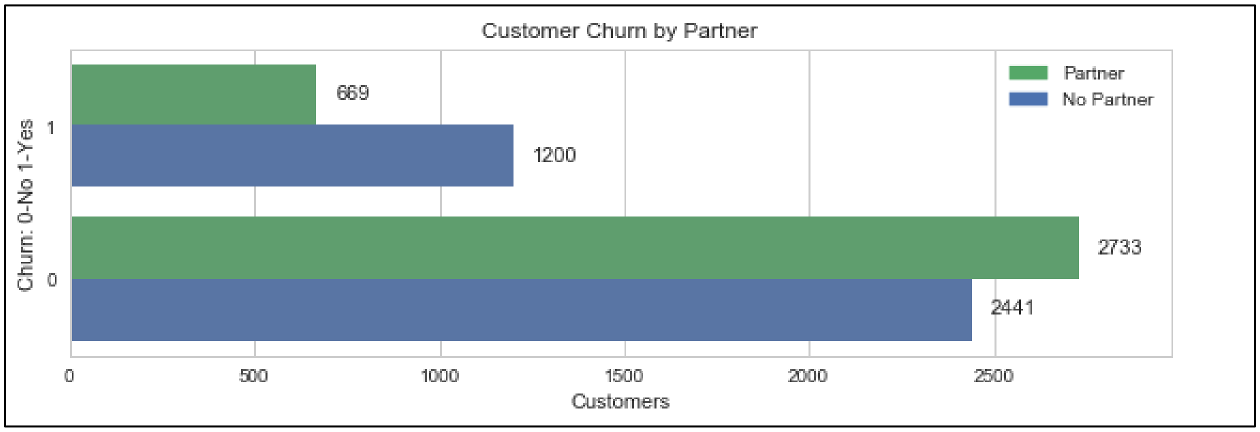


Figure 44. Bar plot of variables Churn and Partner. Single customers (without a partner) churn more than those with partners.

Figure 45 shows the significant difference in churn among customers with and without dependents. Only a small portion of churn occurred with customers that have dependents. In contrast, customers with dependents make up roughly half of non-churn customers. This difference suggests that customers without dependents are more likely to churn. Dependents may be a predictor.

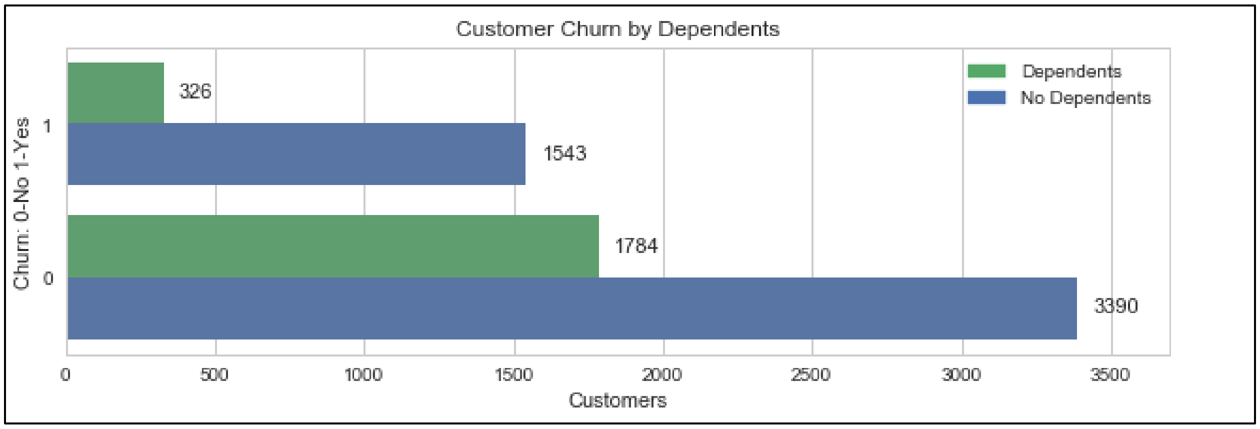


Figure 45. Bar plot of variables Churn and Dependents. Customers with dependents are far less likely to churn than those without dependents.

Figure 46 illustrates that the proportion of customers that have phone service is roughly equal in churn/non-churn populations. PhoneService in not a likely predictor.

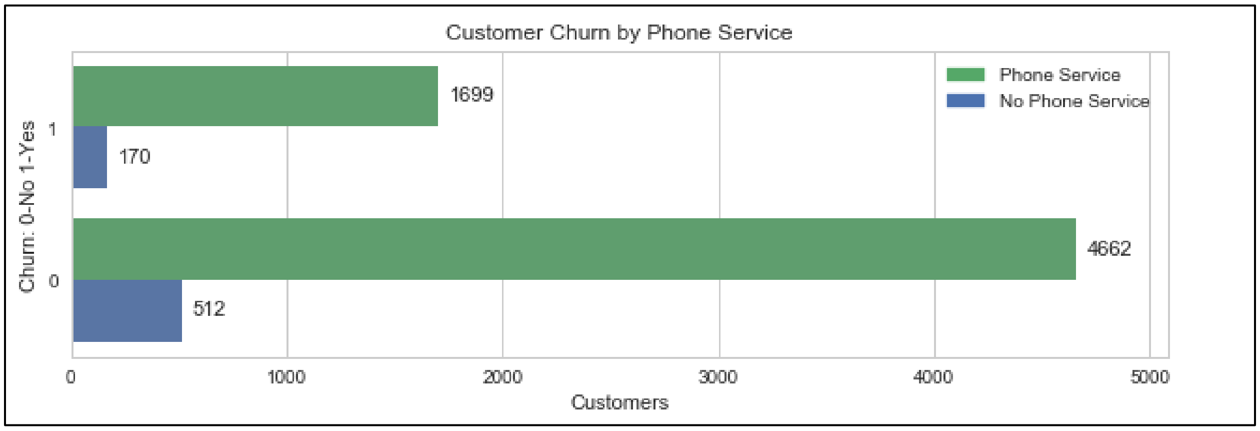


Figure 46. Bar plot of variables Churn and PhoneService. Most churned customers have phone service which is not surprising since the same percentage of all customers have phone service.

Figure 47 shows that among customers with phone service, the difference is trivial between those with and without multiple lines. MultipleLines does not appear to predict Churn.

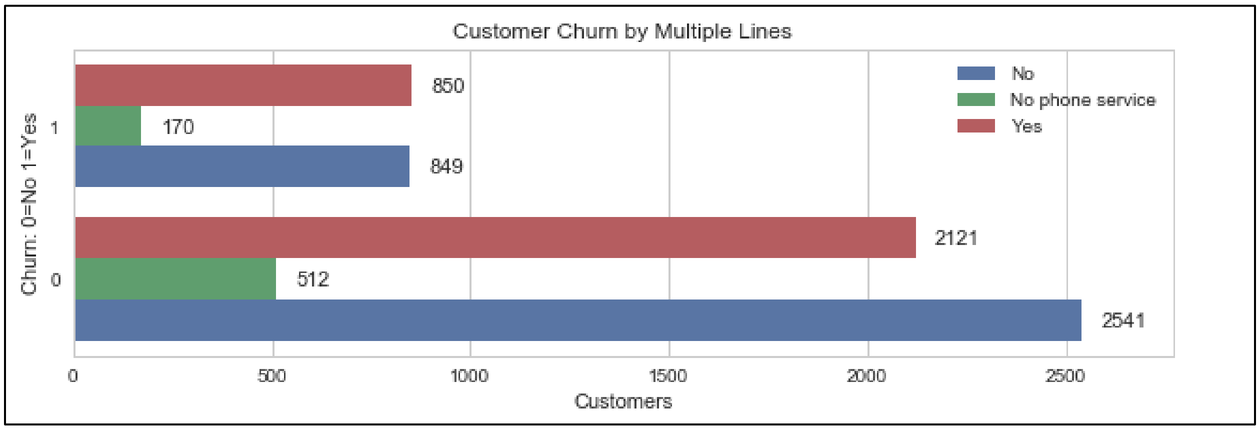


Figure 47. Bar plot of variables Churn and MultipleLines. Among phone service customers there is virtually no difference in churn.

More than twice as many customers with fiber optic internet service churned than those with DSL (figure 48). InternetService may be a predictor.

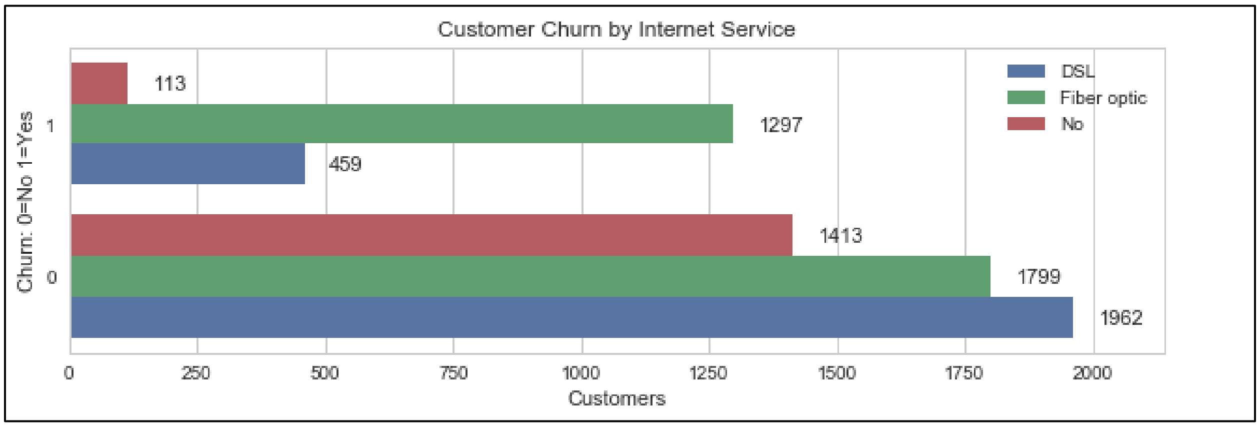


Figure 48. .Bar plot of variables Churn and InternetService. More than twice as many Fiber Optic customers churned as DSL customers.

Figure 49 shows that only a small proportion of customers that churned have online security compared to the substantial number with this feature who have not churned. OnlineSecurity may be a contributor.

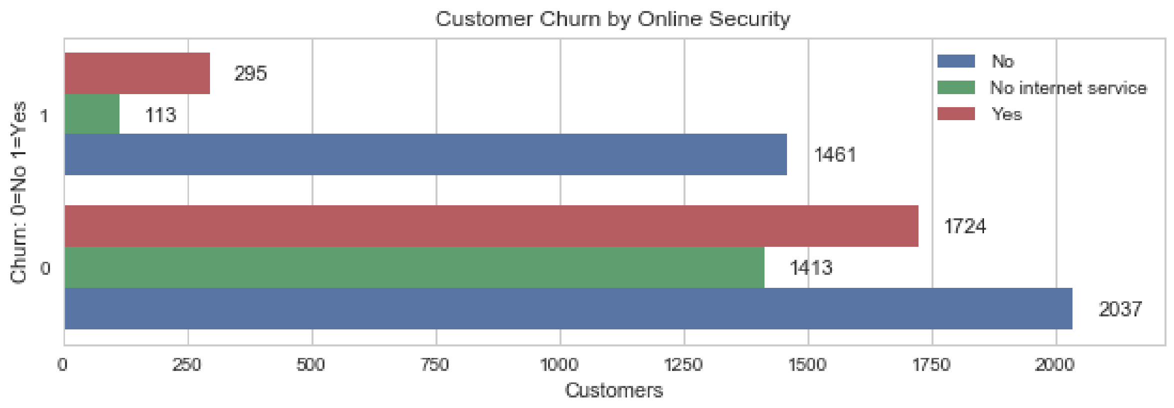


Figure 49. Bar plot of variables Churn and OnlineSecurity. Online security may contribute to Churn.

TechSupport (figure 50) shows similar characteristics to OnlineSecurity and may be a contributor to Churn.

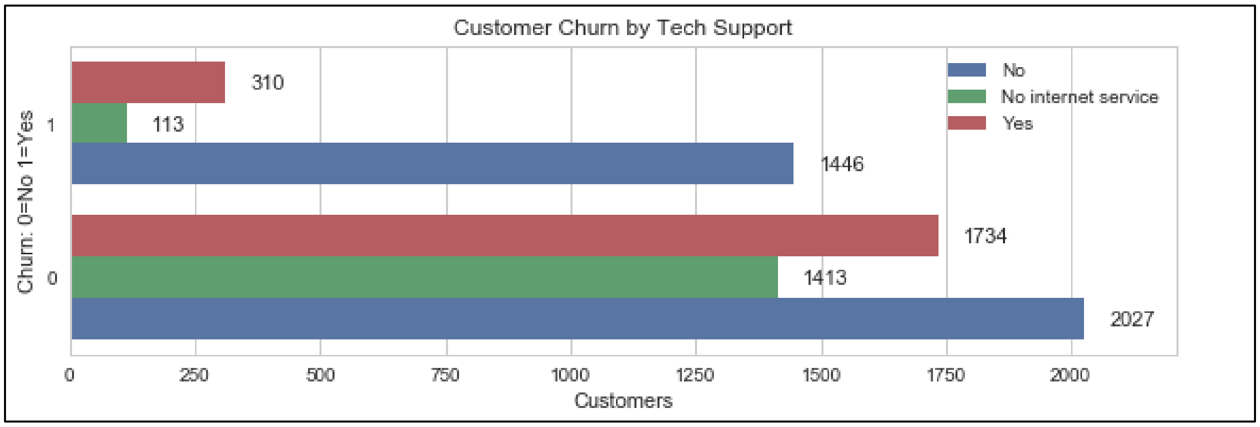


Figure 50. Bar plot of variables Churn and TechSupport This variable has almost identical distribution as OnlineSecurity and may contribute to Churn.

The proportion of OnlineBackup that churned is less than half of those that don't have this service (figure 51). This variable may also be a contributor of Churn. The variables DeviceProtection (figure 52), StreamingTV (figure 53), and StreamingMovies (figure 54) share the same characteristics.

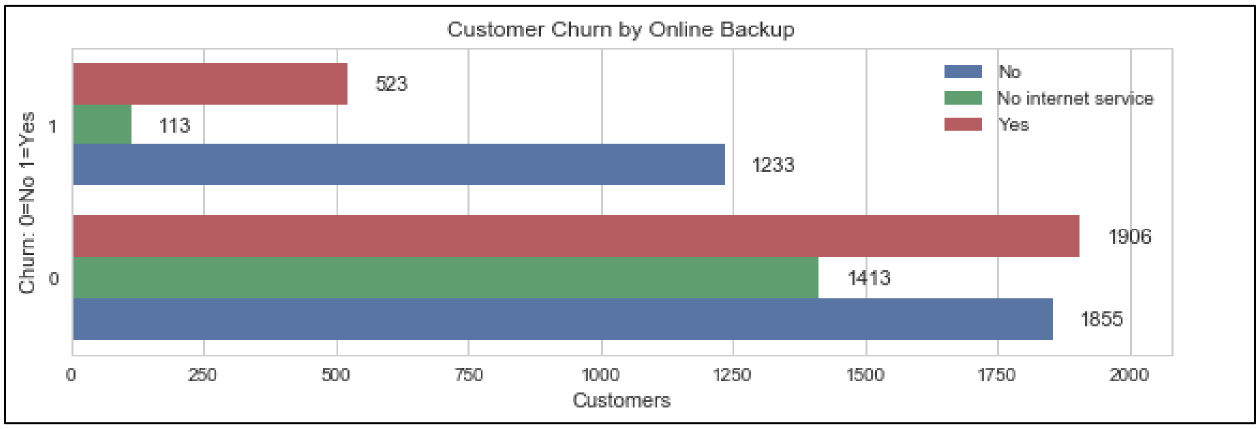


Figure 51. Bar plot of variables Churn and OnlineBackup. This variable may also contribute to Churn.

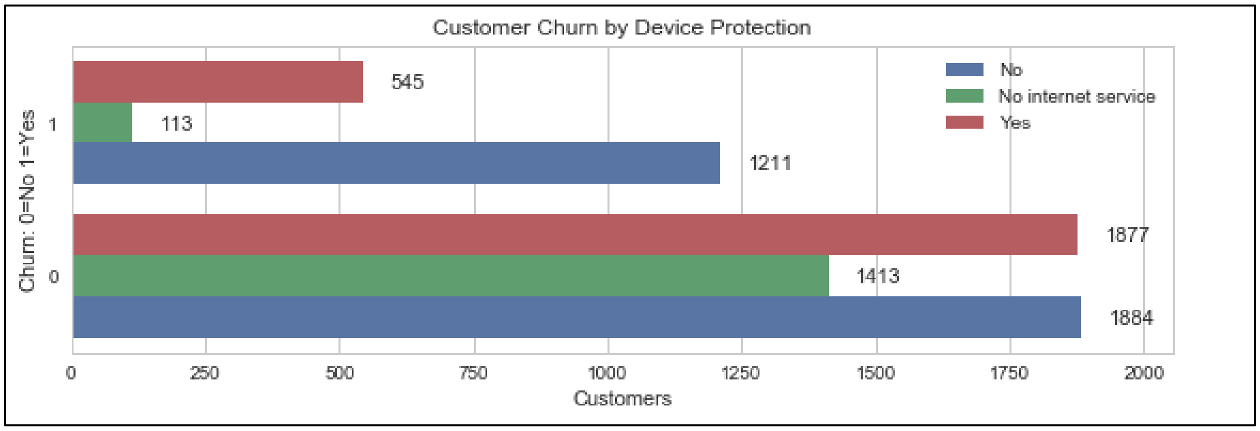


Figure 52. .Bar plot of variables Churn and DeviceProtection. This variable may also contribute to Churn.

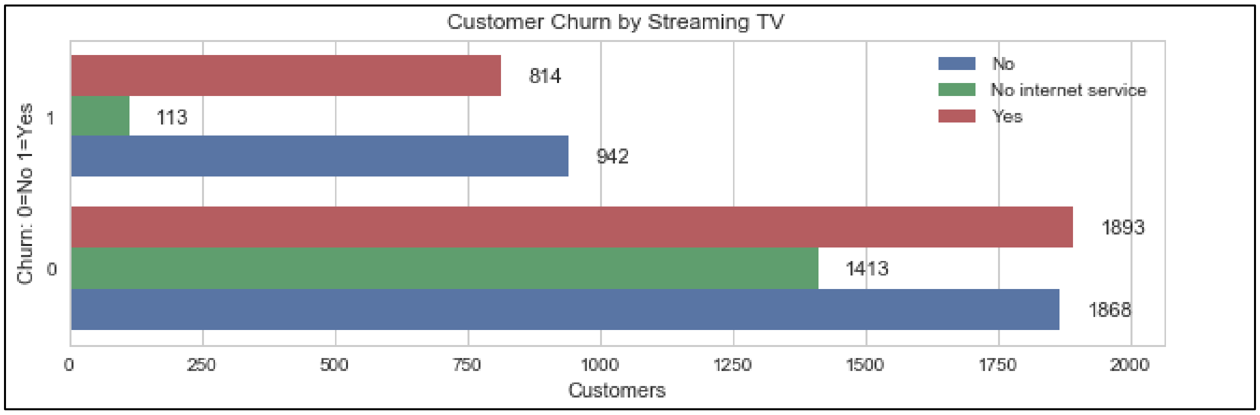


Figure 53. Bar plot of variables Churn and StreamingTV. This variable may contribute to Churn.

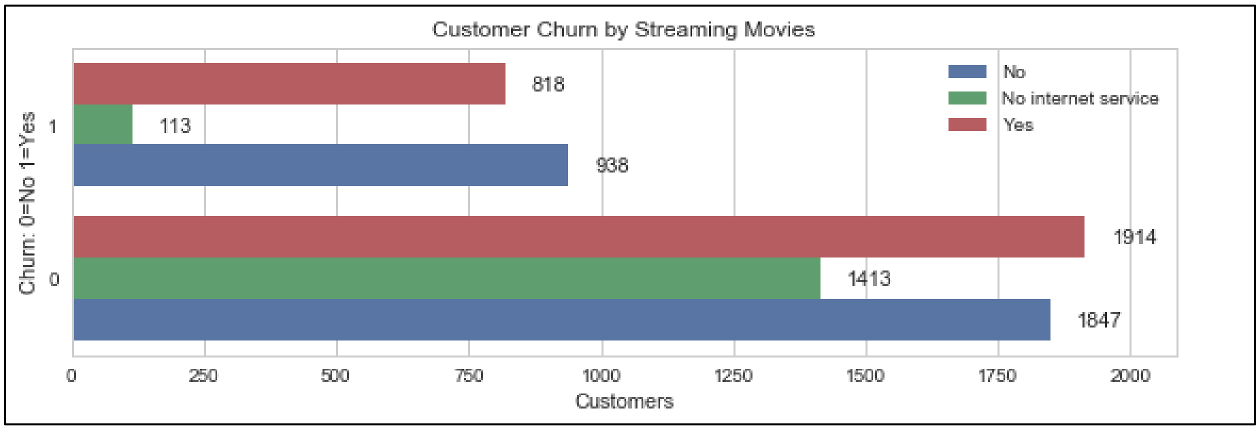


Figure 54. Bar plot of variables Churn and StreamingMovies. This variable has a very similar distribution to StreamingTV and may contribute to Churn.

Figure 55 illustrates that customers with month-to-month contracts account for the majority of Churn. This indicates a strong likelihood that it contributes to Churn.

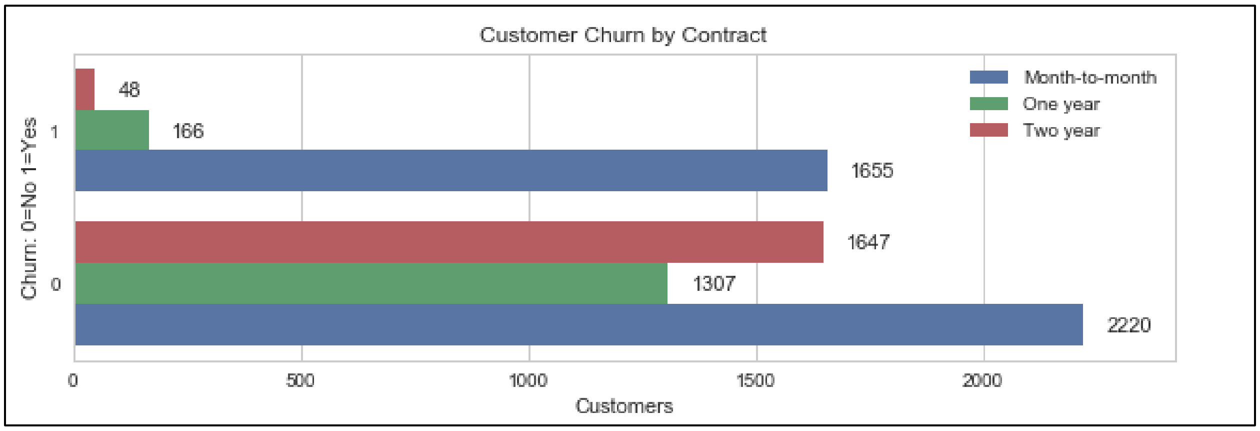


Figure 55. Bar plot of variables Churn and Contract. The clear majority of customer churn occurs with month-to-month customers. This is a likely contributor to Churn.

Figure 56 shows that paperless billing customers account for almost three times as many churns as non-paperless billing customers. This variable may be a contributor.

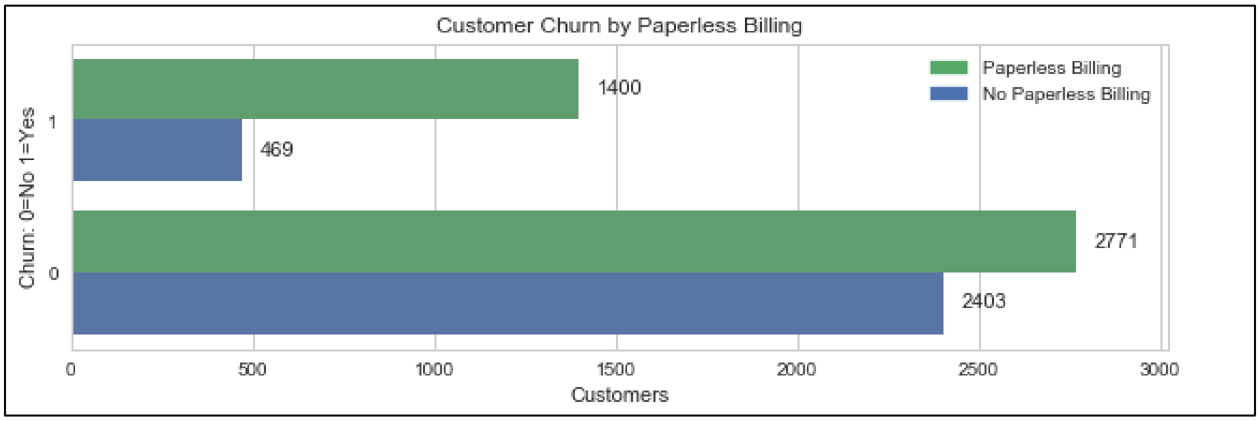


Figure 56. Bar plot of variables Churn and PaperlessBilling. Almost three times as many Paperless Billing customers churn as non-paperless billing customers.

As illustrated in figure 57, the payment method associated with the majority of customer churn is electronic check. Electronic check may contribute to Churn.

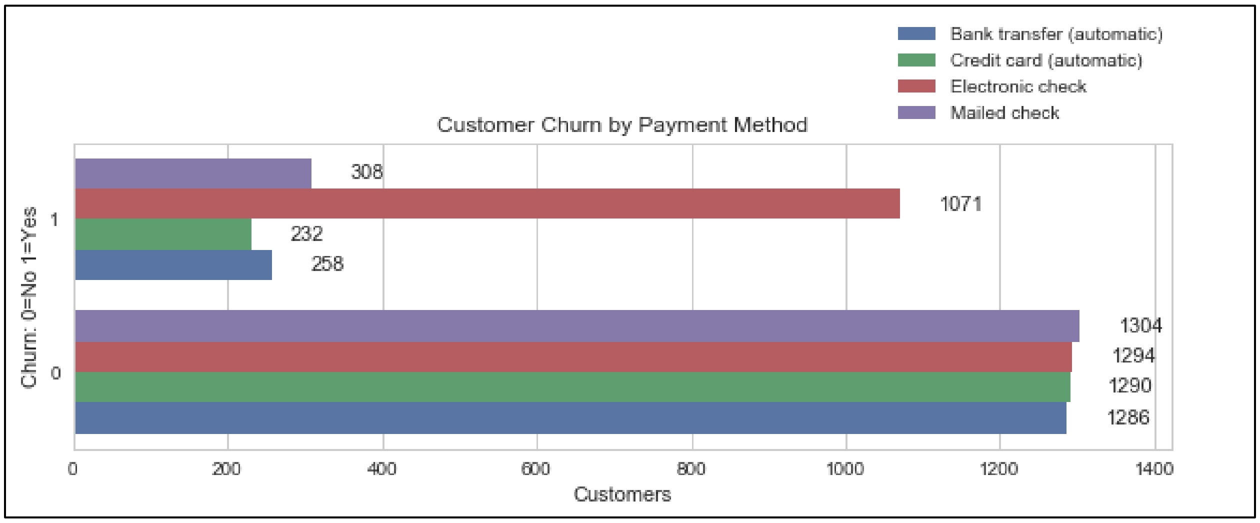
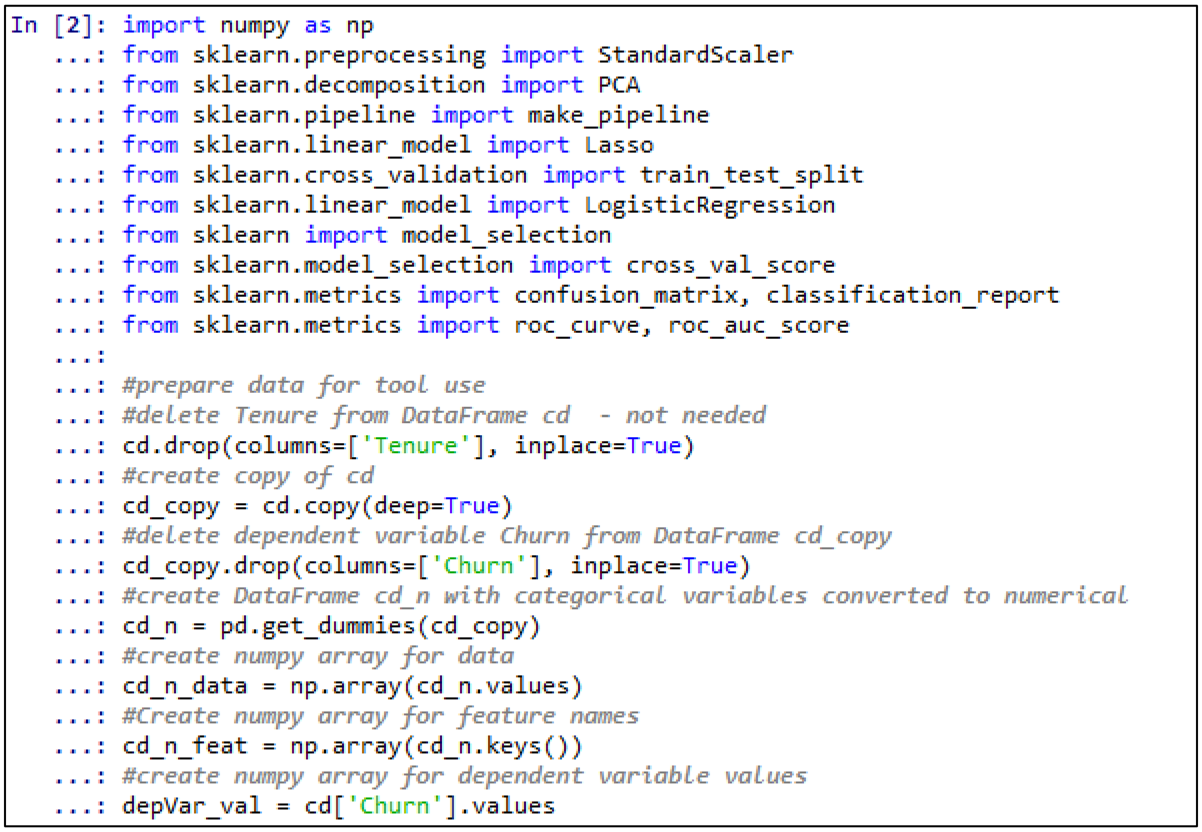


Figure 57. Bar plot of variables Churn and PaymentMethod. Electronic check accounts for more churn (57.3%) than the other three payment methods combined.

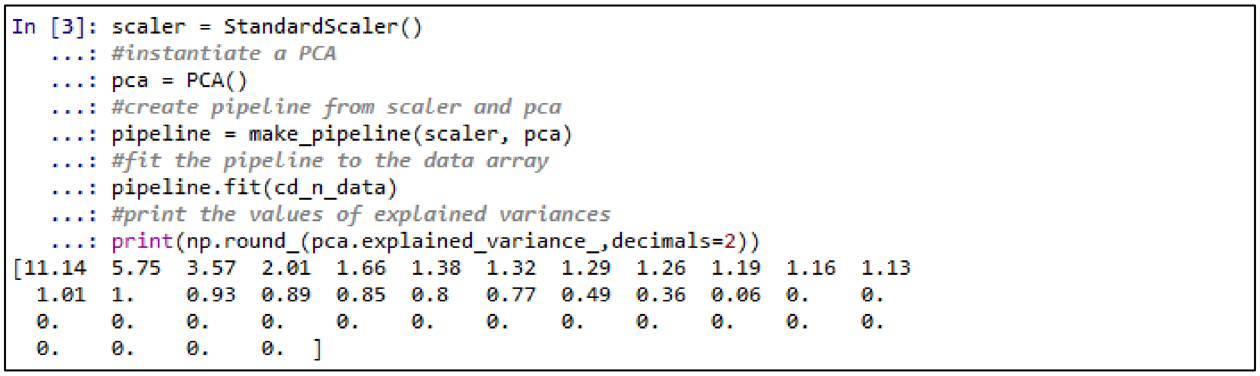
K. Python packages numpy and scikit-learn contribute to the application of analytic and evaluative methods. The methods and packages chosen for this stage require only numerical data. To that end, a copy of cd is made without Tenure (replaced by TenureCalc) and without Churn (the dependent variable). The pandas .get_dummies method converts all categorical variables to new binary numeric variables resulting in 40 independent variables. Numpy creates separate arrays for the data, variable names, and dependent variable data values. Figure 58 shows this process.



```
In [2]: import numpy as np
   ...: from sklearn.preprocessing import StandardScaler
   ...: from sklearn.decomposition import PCA
   ...: from sklearn.pipeline import make_pipeline
   ...: from sklearn.linear_model import Lasso
   ...: from sklearn.cross_validation import train_test_split
   ...: from sklearn.linear_model import LogisticRegression
   ...: from sklearn import model_selection
   ...: from sklearn.model_selection import cross_val_score
   ...: from sklearn.metrics import confusion_matrix, classification_report
   ...: from sklearn.metrics import roc_curve, roc_auc_score
   ...:
   ...: #prepare data for tool use
   ...: #delete Tenure from DataFrame cd  - not needed
   ...: cd.drop(columns=['Tenure'], inplace=True)
   ...: #create copy of cd
   ...: cd_copy = cd.copy(deep=True)
   ...: #delete dependent variable Churn from DataFrame cd_copy
   ...: cd_copy.drop(columns=['Churn'], inplace=True)
   ...: #create DataFrame cd_n with categorical variables converted to numerical
   ...: cd_n = pd.get_dummies(cd_copy)
   ...: #create numpy array for data
   ...: cd_n_data = np.array(cd_n.values)
   ...: #Create numpy array for feature names
   ...: cd_n_feat = np.array(cd_n.keys())
   ...: #create numpy array for dependent variable values
   ...: depVar_val = cd['Churn'].values
```

Figure 58. Python code importing packages for data analysis and preparing new data arrays for tool use.

Analytic method – PCA helps to identify the number of principal components (PCs) needed to adequately represent the data. These components may be a single original variable or a combination of original variables. In this way, it can reduce the number of original variables needed in a subsequent predictive model. Figure 59 illustrates the PCA code and list the variances of each PC in descending order. The first component listed with variance 11.14 has the most variation from the rest of the components.



```
In [3]: scaler = StandardScaler()
   ...: #instantiate a PCA
   ...: pca = PCA()
   ...: #create pipeline from scaler and pca
   ...: pipeline = make_pipeline(scaler, pca)
   ...: #fit the pipeline to the data array
   ...: pipeline.fit(cd_n_data)
   ...: #print the values of explained variances
   ...: print(np.round_(pca.explained_variance_,decimals=2))
[11.14  5.75  3.57  2.01  1.66  1.38  1.32  1.29  1.26  1.19  1.16  1.13
  1.01  1.    0.93  0.89  0.85  0.8   0.77  0.49  0.36  0.06  0.    0.
  0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
  0.    0.    0.    0.  ]
```

Figure 59. Python code creating a PCA and fitting it to the dataset without the dependent variable. The list of principle components shown by descending order of variance illustrates how well each one may describe the original data. The higher the variance the greater the descriptive ability.

Figure 60 graphs PCs in descending order of variance and corresponds to the list above. A best practice is to select PCs from the left up to the "elbow" shaped area of the graph. In this case, from 0 to 3 or 4. Adding more PCs risk overfitting the model and adds unnecessary complexity.

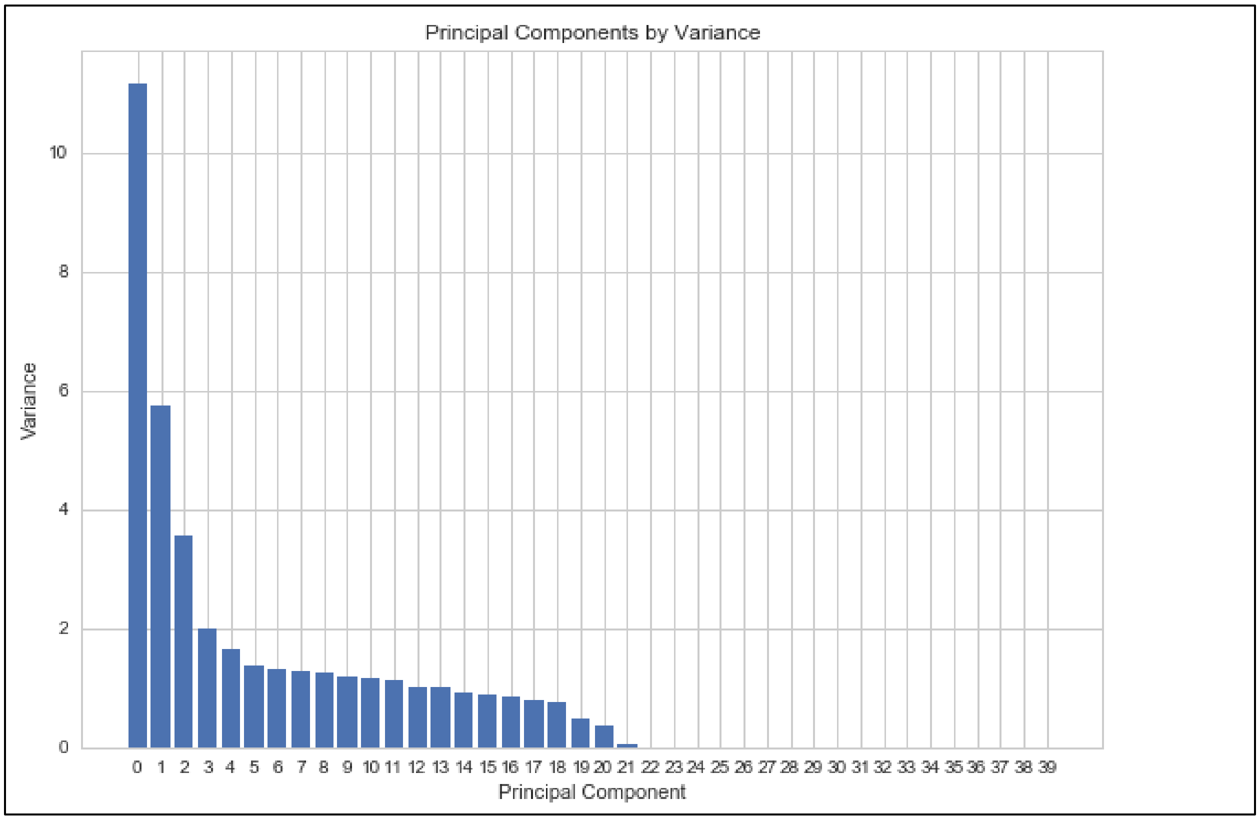


Figure 60. Plot of principle components by variance. Select components 0 to 3 or 4. The remaining components add complexity and risk overfitting.

Plotting the cumulative ratios of PC variances can show how much of the data may be explained by adding more PCs. The graph in figure 61 enhances the previous plot by showing that it would take 14 PCs to explain 90% of the variance and approximately 60% to 70% could be explained with four to six PCs.

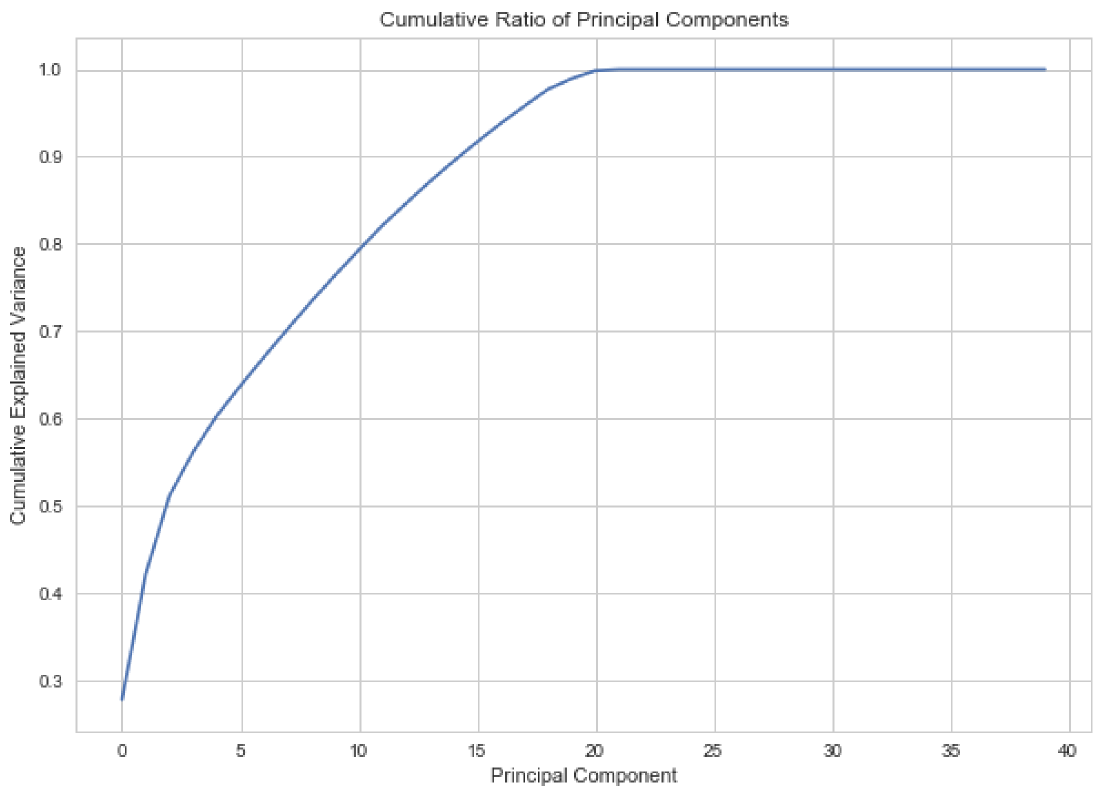


Figure 61. Plot of the cumulative ratio of principle components. Selecting five PCs would explain approximately 65% of the data.

Evaluative method – Scikit-learn's regularized Lasso regression locates the five most explanatory variables based on the optimal number from the prior PCA analysis. Figure 62 displays the code to generate the analysis and plot results. Lasso analysis reveals these five variables: MonthlyCharges, TenureCalc, TechSupport_No, Contract_Month-to-month, and PaymentMethod_Electronic check.

```
In [4]: lasso = Lasso(alpha=0.023)
   ...: #lasso = Lasso(alpha=0.022)
   ...: #lasso = Lasso(alpha=0.024)
   ...: #instantiate a Lasso coefficient to extract the coefficients
   ...: lasso_coef = lasso.fit(cd_n_data, depVar_val).coef_
   ...: #Plot the coefficients as a function of the feature names
   ...: plt.rcParams['figure.figsize'] = 11,7
   ...: plt.plot(range(len(cd_n_feat)), lasso_coef)
   ...: plt.xticks(range(len(cd_n_feat)), cd_n_feat, rotation=90)
   ...: plt.xlabel('Independent Variables')
   ...: plt.ylabel('Coefficients')
   ...: plt.title('Lasso Regression for Independent Variable Selection')
   ...: plt.show()
```

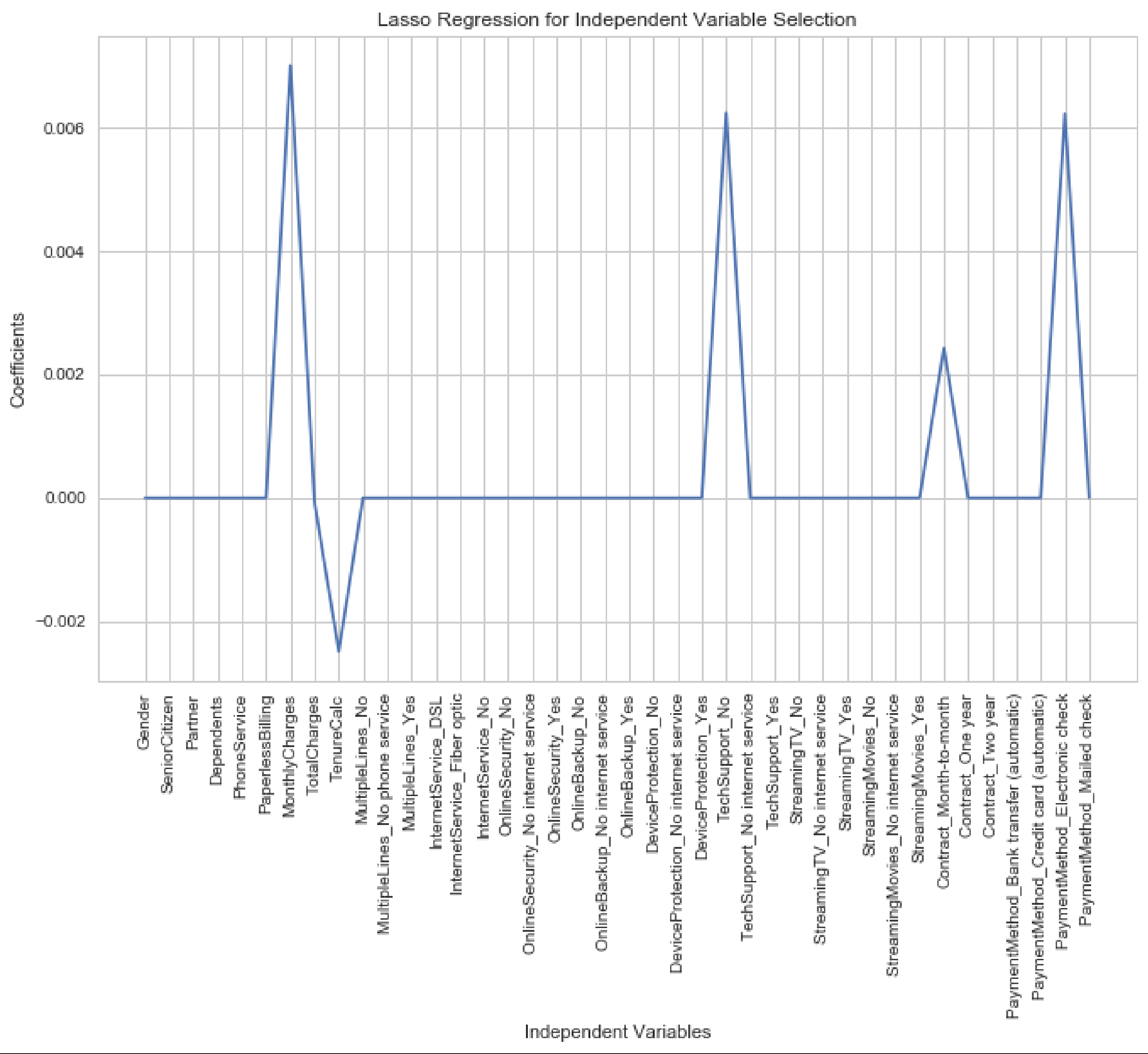
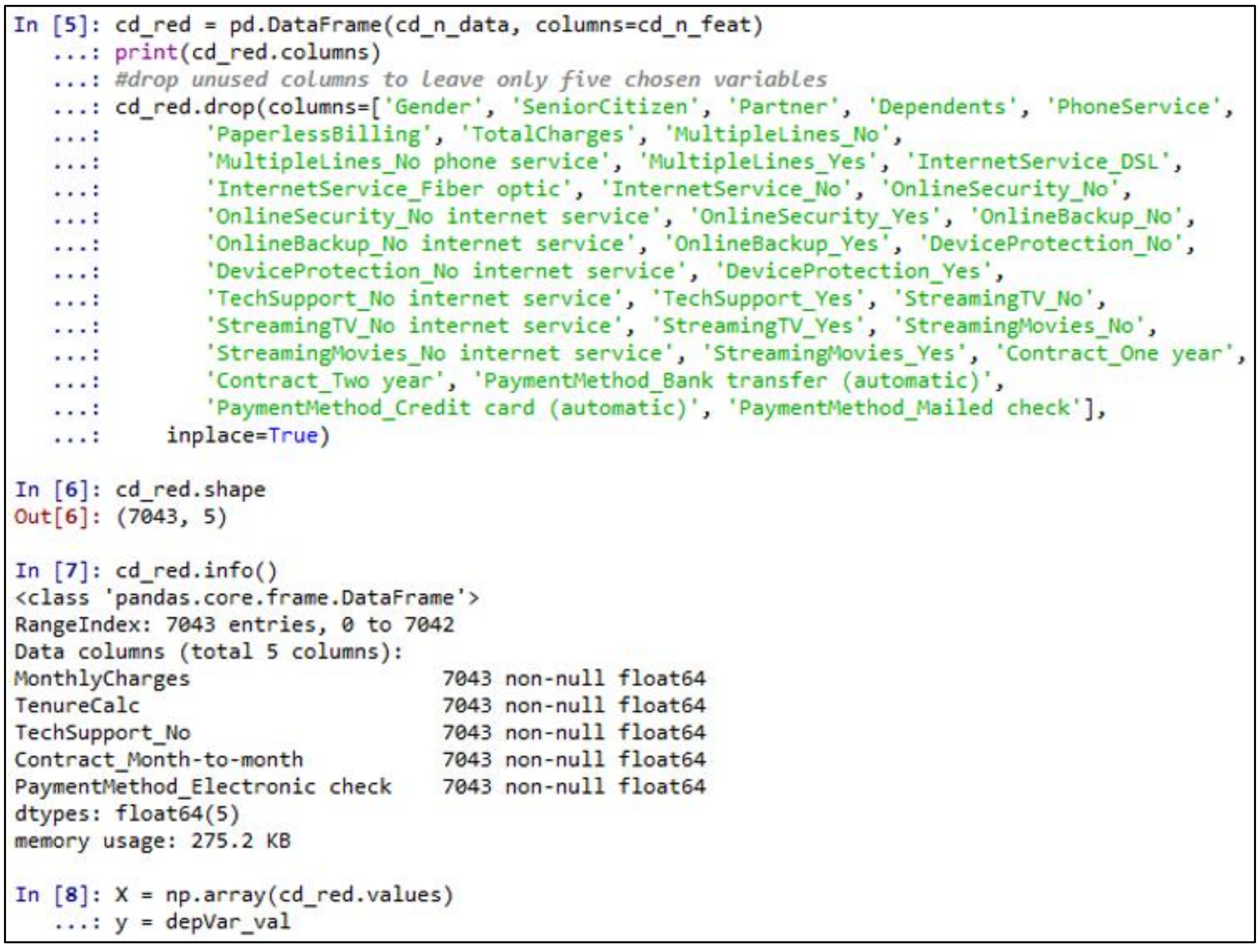


Figure 62. Python code for scikit-learn regularized Lasso regression with alpha value = 0.023. Plot of the five most descriptive variables.

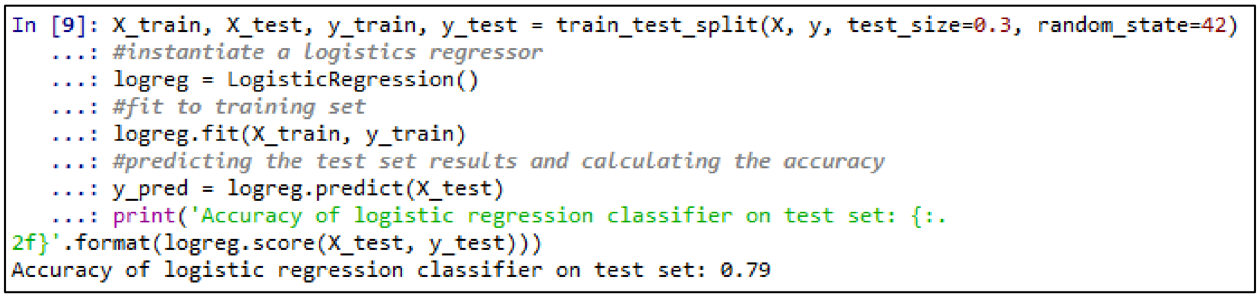
To build the logistic regression model, a new DataFrame (cd_red) is created with these five independent variables and their labels. Next, the five variables are set in a new numpy array X for the independent variable values along with a new numpy array y for the dependent variable Churn values (figure 63).



```
In [5]: cd_red = pd.DataFrame(cd_n_data, columns=cd_n_feat)
   ...: print(cd_red.columns)
   ...: #drop unused columns to leave only five chosen variables
   ...: cd_red.drop(columns=['Gender', 'SeniorCitizen', 'Partner', 'Dependents', 'PhoneService',
   ...:        'PaperlessBilling', 'TotalCharges', 'MultipleLines_No',
   ...:        'MultipleLines_No phone service', 'MultipleLines_Yes', 'InternetService_DSL',
   ...:        'InternetService_Fiber optic', 'InternetService_No', 'OnlineSecurity_No',
   ...:        'OnlineSecurity_No internet service', 'OnlineSecurity_Yes', 'OnlineBackup_No',
   ...:        'OnlineBackup_No internet service', 'OnlineBackup_Yes', 'DeviceProtection_No',
   ...:        'DeviceProtection_No internet service', 'DeviceProtection_Yes',
   ...:        'TechSupport_No internet service', 'TechSupport_Yes', 'StreamingTV_No',
   ...:        'StreamingTV_No internet service', 'StreamingTV_Yes', 'StreamingMovies_No',
   ...:        'StreamingMovies_No internet service', 'StreamingMovies_Yes', 'Contract_One year',
   ...:        'Contract_Two year', 'PaymentMethod_Bank transfer (automatic)',
   ...:        'PaymentMethod_Credit card (automatic)', 'PaymentMethod_Mailed check'],
   ...:     inplace=True)

In [6]: cd_red.shape
Out[6]: (7043, 5)

In [7]: cd_red.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 5 columns):
MonthlyCharges                  7043 non-null float64
TenureCalc                      7043 non-null float64
TechSupport_No                  7043 non-null float64
Contract_Month-to-month         7043 non-null float64
PaymentMethod_Electronic check  7043 non-null float64
dtypes: float64(5)
memory usage: 275.2 KB

In [8]: X = np.array(cd_red.values)
   ...: y = depVar_val
```

Figure 63. Python code to create DataFrame cd_red with only the five variables and their labels. Numpy arrays X and y created from the independent variable values and dependent variable values.

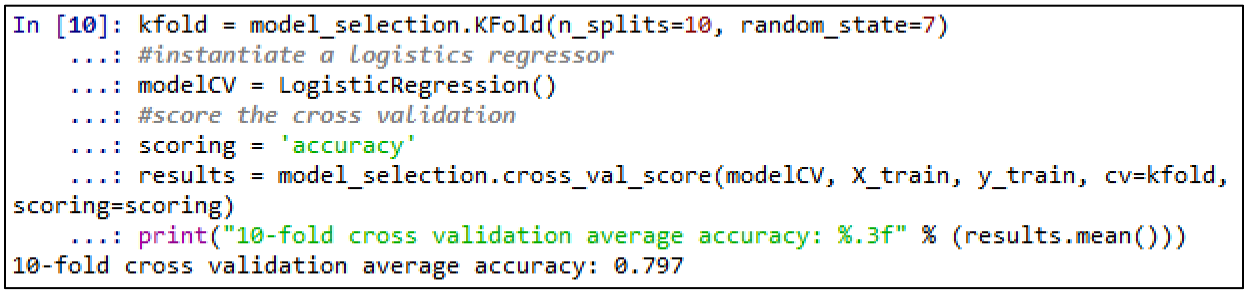
The next step is to split the data into training and test datasets. This will allow the model to train on 70% of the data, be tested on the remaining 30% of the data, and compare the results. Scikit-learn splits the data and creates a new logistic regression instance. Model fitting occurs with the training data, model prediction occurs with the independent variable test data, and model accuracy is checked with the dependent variable test data. Figure 64 shows that the mean accuracy of the model is 79%.



```
In [9]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
   ...: #instantiate a logistics regressor
   ...: logreg = LogisticRegression()
   ...: #fit to training set
   ...: logreg.fit(X_train, y_train)
   ...: #predicting the test set results and calculating the accuracy
   ...: y_pred = logreg.predict(X_test)
   ...: print('Accuracy of logistic regression classifier on test set: {:.
2f}'.format(logreg.score(X_test, y_test)))
Accuracy of logistic regression classifier on test set: 0.79
```

Figure 64. Python code to split the data 70/30 into train and test data sets, create a new logistical regression, fit to the train data, predict with the test data, and check the accuracy. The model's mean accuracy is 79%.

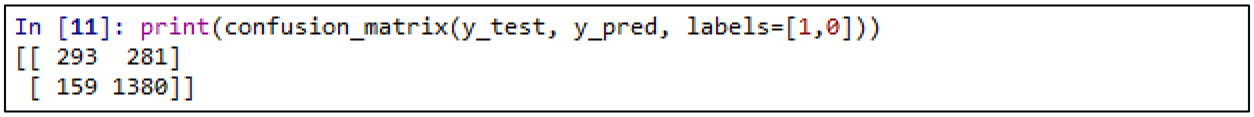
Next, 10-fold cross-validation checks the model for overfitting. Figure 65 illustrates the process of creating a k-fold classifier, a new logistic regression instance, calculating the cross-validation score for the training data, and printing the score. The average mean accuracy score of 0.797 is very close to the accuracy score from the previous step and shows that the model generalizes well and is not overfitted.



```
In [10]: kfold = model_selection.KFold(n_splits=10, random_state=7)
    ...: #instantiate a logistics regressor
    ...: modelCV = LogisticRegression()
    ...: #score the cross validation
    ...: scoring = 'accuracy'
    ...: results = model_selection.cross_val_score(modelCV, X_train, y_train, cv=kfold,
scoring=scoring)
    ...: print("10-fold cross validation average accuracy: %.3f" % (results.mean()))
10-fold cross validation average accuracy: 0.797
```

Figure 65. Python code to create a kfold cross validation, create a new logistical regression, and score the model using the training data. The model's cross validation mean accuracy is 79.7%, which varies only .007% from the previous accuracy score.

A Scikit-learn confusion matrix compares the dependent variable's test data to the dependent variable's predicted data and shows the cross tabulation of counts. The top left is true positive (tp), top right is false negative (fn), bottom left is false positive (fp), and bottom right is true negative (tn). Figure 66 illustrates the code and the matrix.



```
In [11]: print(confusion_matrix(y_test, y_pred, labels=[1,0]))
[[ 293  281]
 [ 159 1380]]
```

Figure 66. Python code to print the confusion matrix. The model made 1,673 correct predictions and 440 incorrect predictions.

The overall accuracy is 79.2% and is calculated as:

Overall accuracy = (tp + tn) / (tp + tn + fp + fn).

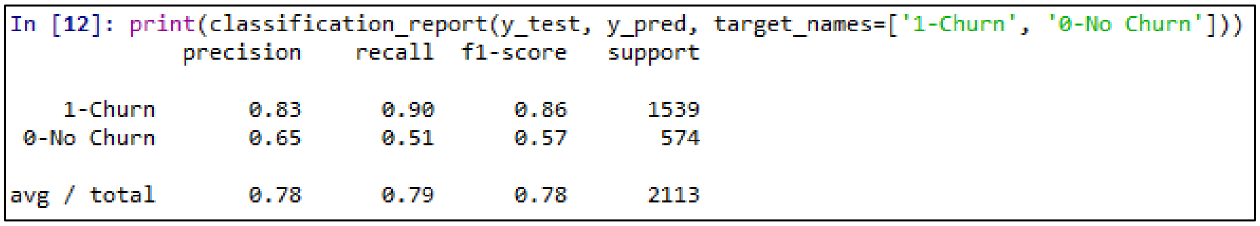
Table 2 illustrates the results showing that the model made 293 + 1,380 = 1,673 true predictions and 159 + 281 = 440 false predictions.

Table 2

Confusion Matrix of Test and Predicted Dependent Variable

	Predicted	
Actual	1-Churn	0-No Churn
1-Churn	True Positive 293	False Negative 281
0-No Churn	False Positive 159	True Negative 1,380

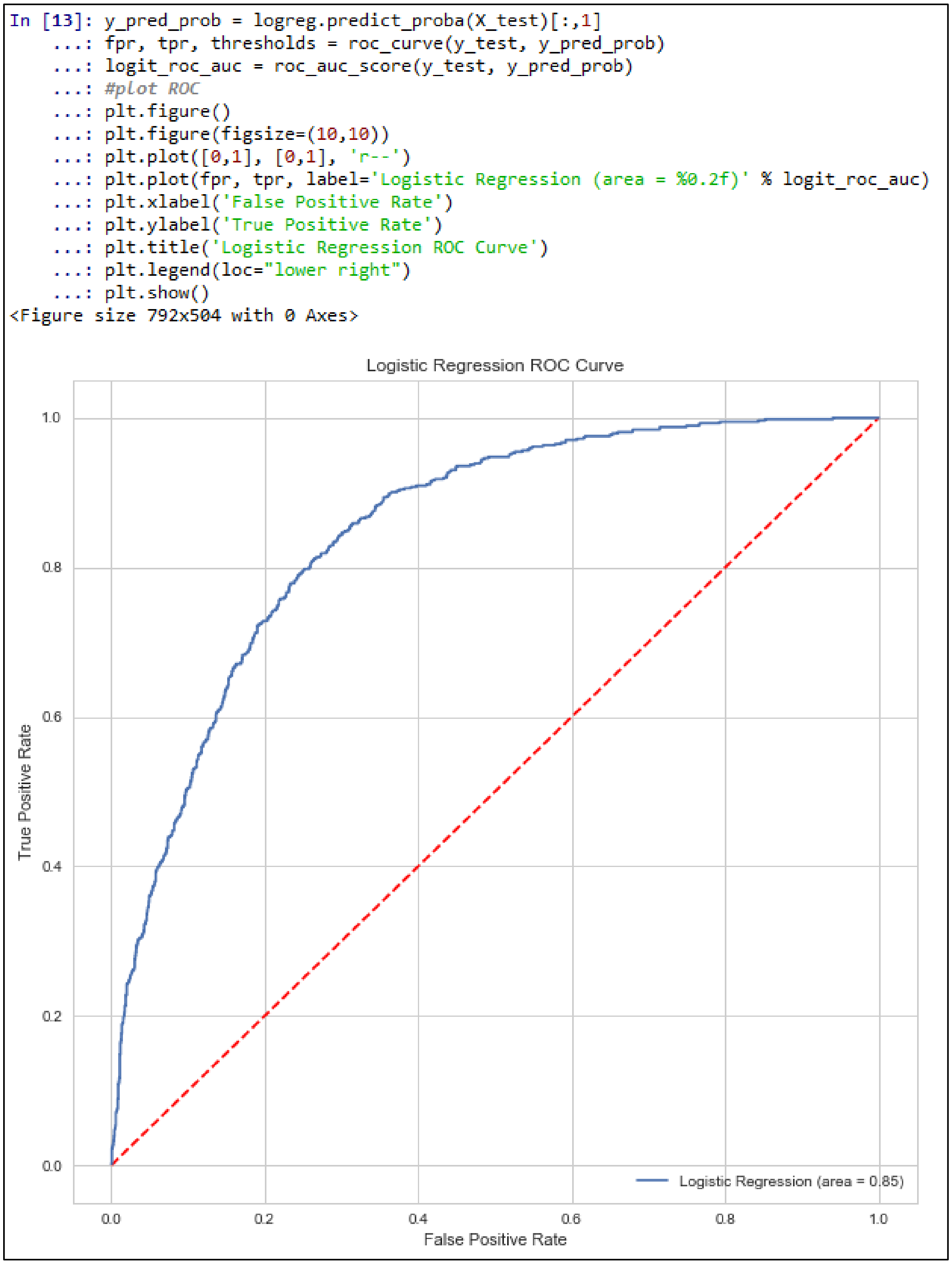
A Scikit-learn classification report helps quantify the results from the confusion matrix. Figure 67 shows the single line of code to produce the report and results. Precision, aka Positive Predictive Value (PPV), is the ratio tp / (tp + fp). Precision scores how well the model avoids labeling a sample as positive when it is negative. Recall is the ratio tp / (tp + fn). Recall scores how well the model identifies positive samples. The fi-score is the mean of precision and recall. Support is the count of samples in each class of the test data. The higher the PPV, the better the model is at avoiding false positives. The higher the recall, the better the model is at avoiding false negatives.



```
In [12]: print(classification_report(y_test, y_pred, target_names=['1-Churn', '0-No Churn']))
             precision    recall  f1-score   support

    1-Churn       0.83      0.90      0.86      1539
 0-No Churn       0.65      0.51      0.57       574

avg / total       0.78      0.79      0.78      2113
```

Figure 67. Python code to print the classification report. The average precision (PPV) indicates that the model avoids false positives 78% of the time. The average recall indicates that the model avoids false negatives 79% of the time.

The logistic regression model returns probabilities that an observation is likely to churn or not churn based on a default p-value threshold of 0.5. The Scikit-learn Receiver Operator Character (ROC) curve shown in figure 68 plots true positive rate versus false positive rate for all possible p-value thresholds from 0.0 to 1.0. The blue line represents the ratio of the model's performance at all possible p-values. The farther the model's performance is above the dotted red line (performance of a purely random model) the better. The measurement of performance above the dotted red line is calculated and shown as the proportion of the total plot area under the curve (logistic regression area = 0.85), which is 0.35 higher than the score of 0.5 for a random model.



```
In [13]: y_pred_prob = logreg.predict_proba(X_test)[:,1]
    ...: fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)
    ...: logit_roc_auc = roc_auc_score(y_test, y_pred_prob)
    ...: #plot ROC
    ...: plt.figure()
    ...: plt.figure(figsize=(10,10))
    ...: plt.plot([0,1], [0,1], 'r--')
    ...: plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % logit_roc_auc)
    ...: plt.xlabel('False Positive Rate')
    ...: plt.ylabel('True Positive Rate')
    ...: plt.title('Logistic Regression ROC Curve')
    ...: plt.legend(loc="lower right")
    ...: plt.show()
<Figure size 792x504 with 0 Axes>
```

Logistic Regression ROC Curve
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Logistic Regression (area = 0.85)

Figure 68. Python code to calculate and print the Receiver Operator Character (ROC) curve report. The blue line shows that for all possible p-values, the model performs better than a random model. The model's area under curve score is 0.85.

L. There are two main advantages of using PCA for this analysis. First is its ability to quickly analyze high dimension data. PCA analyzed the variances between 40 dependent variables containing 7,043 data values in under one second. The second advantage is that PCA exposes the variances between each principal component. These can be graphed allowing an easy visual determination of the number of desired components. Other techniques exist to examine the interactions between independent variables such as Linear discriminant analysis (LDA). However, LDA assumes multivariate normality as well as homoscedasticity between variables to achieve the best results (Wikipedia 2018).

There are several reasons to choose logistic regression for this analysis. The first is that it can determine the likelihood of the binary dependent variable Churn being a 1 or 0 based on multiple independent variables. It does not assume a linear relationship between Churn and the independent variables nor does it assume that the dependent variable and the independent variables are normally distributed. An alternative to logistic regression would have been Decision Trees. The disadvantage of decision trees is that it is easy to overfit the model due to their sensitivity to outliers. This can occur when new data has more variance than the training and test data.

M. Visual presentations in this analysis utilize Python's Matplotlib. The addition of the Seaborn package adds additional plot types and its default color palette softens colors to a more visually pleasing mix. Each type of visualization has two aims. First, to convey the most essential information as quickly as possible. Second, to minimize the amount of space used. This second goal drove the decision to choose horizontal bar plots instead of a vertical bar plots for several visuals. Either method would illustrate differences in quantity, but only the horizontal layout compresses the overall aspect ratio to a greater width vs height to save space, especially when there are significant value differences to compare.

Boxplots and histograms are well suited to displaying univariate data and convey the distribution of continuous variables in a complimentary fashion. Both show the extents of the data. Boxplots show the data's central tendency and variability by marking the mean, quartiles, outliers, and the box's position along the vertical scale. The histogram losses the boxplot's numerical detail but helps clarify the distribution of the values with binned counts shown in adjoining vertical bars. The vertical bars also help the viewer determine normality (or lack of normality) of the data. See figure 21 as an example. There are several alternatives to boxplots such as violin plots, bee swarm plots, bean plots, and vase plots which conveys the distribution of the data well enough. However, they are not as familiar to most users and (depending on software) may not be able to display the mean, quartile, and outlier detail. An alternative to histograms may be Frequency polygons with the point denoting each bin's frequency joined together with a line. Small markers joined by lines are less intuitive to view than the histogram's filled vertical bars.

Horizontal bar plots effectively convey the magnitude of different quantities. Figure 37 illustrates how the horizontal layout also avoids rotating long axis labels. They are especially useful when comparing quantities as they relate to a categorical group. The categorical group is arrayed on the vertical axis with the quantities plotted along the horizontal axis. As an example, in about 2.5 inches of vertical space, figure 57 instantly conveys that there is something very different happening with electronic checks in the

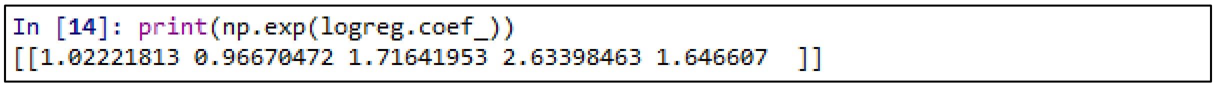
Churn group when compared to the other payment methods. Secondarily, the user can quickly see all eight counts of payment method per Churn/No churn classification.

Violin plots illustrate the distribution of a large volume of quantitative continuous data. Especially when comparing to a categorical group. See figure 39 as an example. The differences between the overall size and shape of the left and right plots highlight differences in population and distribution by values. An alternative to the violin plot might be to use boxplots but they do not convey the distribution of the data as clearly.

Line plots connect many values together to show an accumulation as in figure 61 or a relationship between two variables such as in figure 68 where the axis sizes are set equal. By setting the axis sizes equal, the true relationship is more apparent. Using another type of plot such as a bar chart would force the viewer to focus on the top of the bars to discern the shape of the curve.

IV. Data Summary

N. This logistic regression model uses a combination of five independent variables to predict the phenomena of customer churn with an accuracy of 79%. See figures 64 and 65. Additional insight into the importance of the five independent variables may be gained by observing their odds ratios as shown in figure 69 and table 3.



```
In [14]: print(np.exp(logreg.coef_))
[[1.02221813 0.96670472 1.71641953 2.63398463 1.646607  ]]
```

Figure 69. Python code to calculate odds ratios for the five independent variables.

Table 3

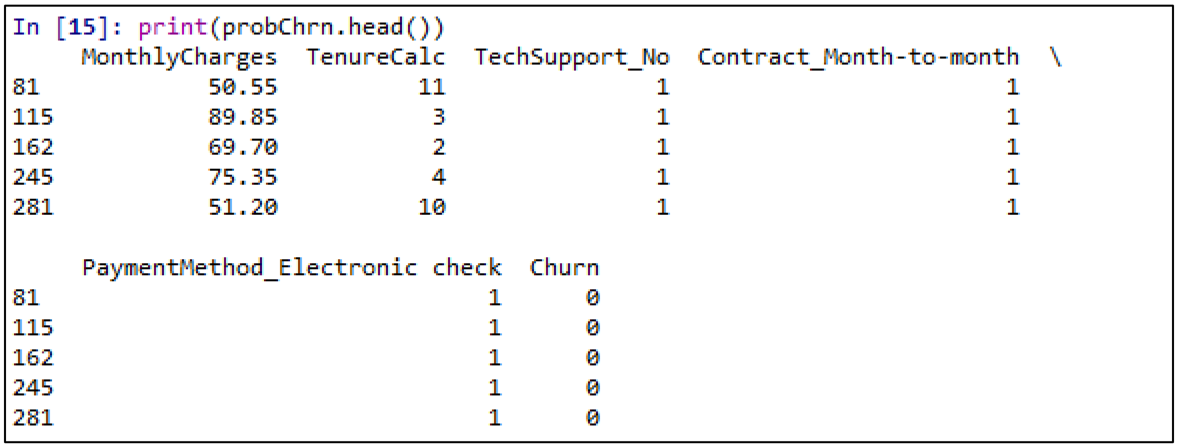
Odds Ratios for Independent Variables

Variable	Odds Ratio
MonthlyCharges	1.02
TenureCalc	0.97
TechSupport_No	1.72
Contract_Month-to-month	2.63
PaymentMethod_Electronic check	1.65

Based on the odds ratios, the largest predictor of churn is that a customer has a month-to-month contract, followed by not having technical support, followed by paying by electronic check. Given the presence of the first three conditions, the odds are almost even for customers with a given MonthlyCharge and Tenure. However, observing MonthlyCharge and TenureCalc versus Churn in isolation (figures 40 and 39), shows how being a new customer with a high monthly charge can influence the customer to churn. Therefore, the recommendation to the company is to promote longer contracts and the advantages of technical support services. Also, to keep monthly charges low and possibly offering longevity incentives. Finally, to monitor customer payment methods since electronic check is a characteristic of churning customers.

We can apply the criteria above to existing customers and filter for monthly charges greater than $35 (first quartile) plus tenure of one year or less to produce a list of customers that the company can target in its retention efforts. The list contains 253 customers. Note that this is very close to the 293 true positives predicted by the model and shown in table 2 above. Appendix C contains these observations in an Excel file

'Appendix_C_ProbableChurn_WA_Fn-UseC_-Telco-Customer-Churn.xlsx'. Figure 70 illustrates the first five observations from the file.



```
In [15]: print(probChrn.head())
     MonthlyCharges  TenureCalc  TechSupport_No  Contract_Month-to-month  \
81            50.55          11               1                        1
115           89.85           3               1                        1
162           69.70           2               1                        1
245           75.35           4               1                        1
281           51.20          10               1                        1

     PaymentMethod_Electronic check  Churn
81                                1      0
115                               1      0
162                               1      0
245                               1      0
281                               1      0
```

Figure 70. First five observations from dataset of customers at greatest risk of churning.

O. After using PCA to determine that four to five principal components adequately describe customer churn, logistic regression's Lasso defined five specific independent variables. Lasso accomplished the task by scoring variable coefficients. Lasso adds a penalty of the absolute value of each variable's coefficient multiplied by an alpha value. This reduces the coefficients of less key variables to zero. Lasso's results are very sensitive to the alpha value. Alpha = 0.23 reveals the five selected variables shown in figure 71. Lowering the alpha only slightly to 0.22 permits OnlineSecurity_No to show as a sixth variable (figure 72), while raising the alpha to 0.24 drops the variable Contract_Month-to-month as shown in figure 73.

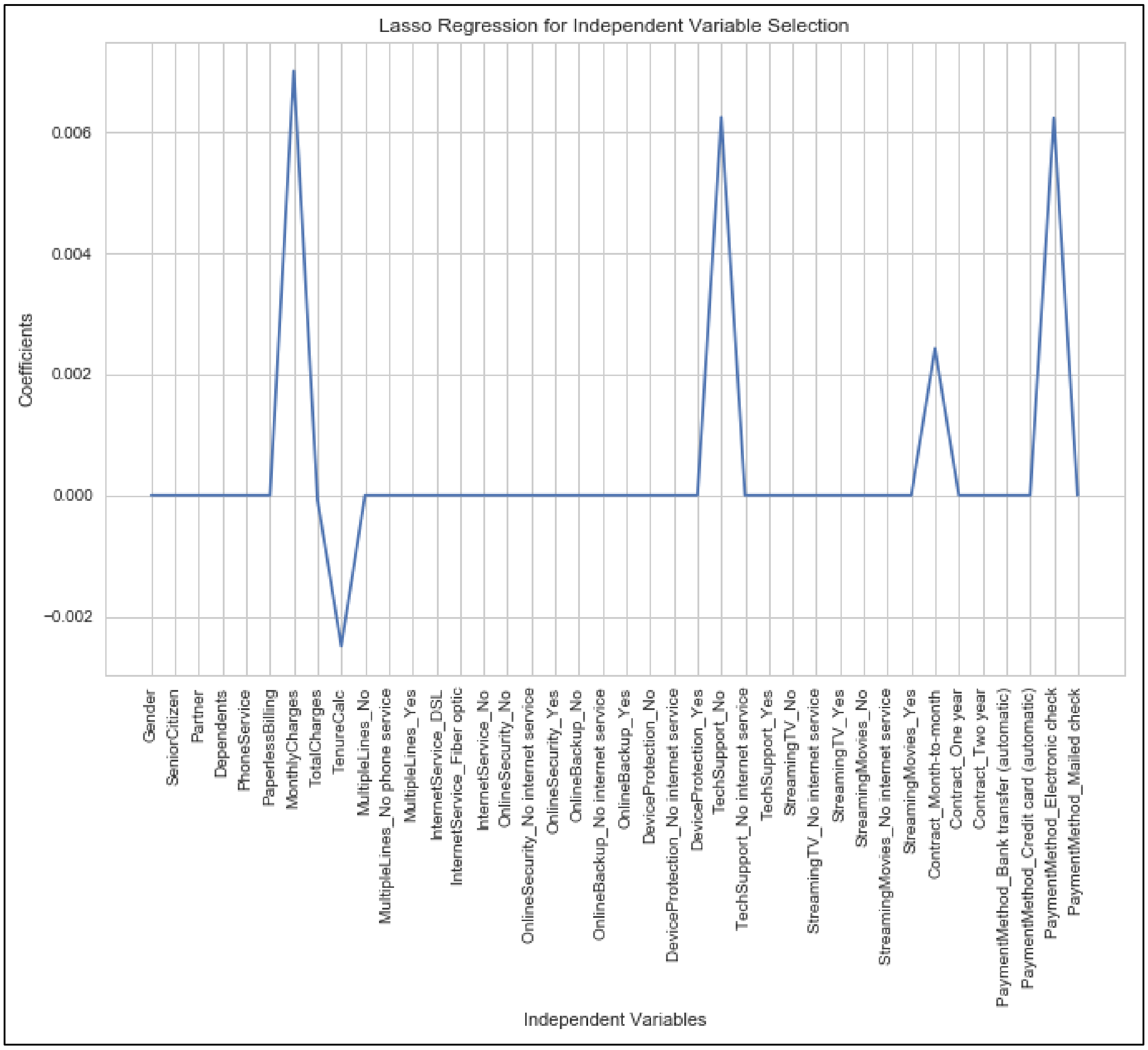


Figure 71. Lasso regression for independent variable selection using alpha = 0.23. Five variables show distinctly.

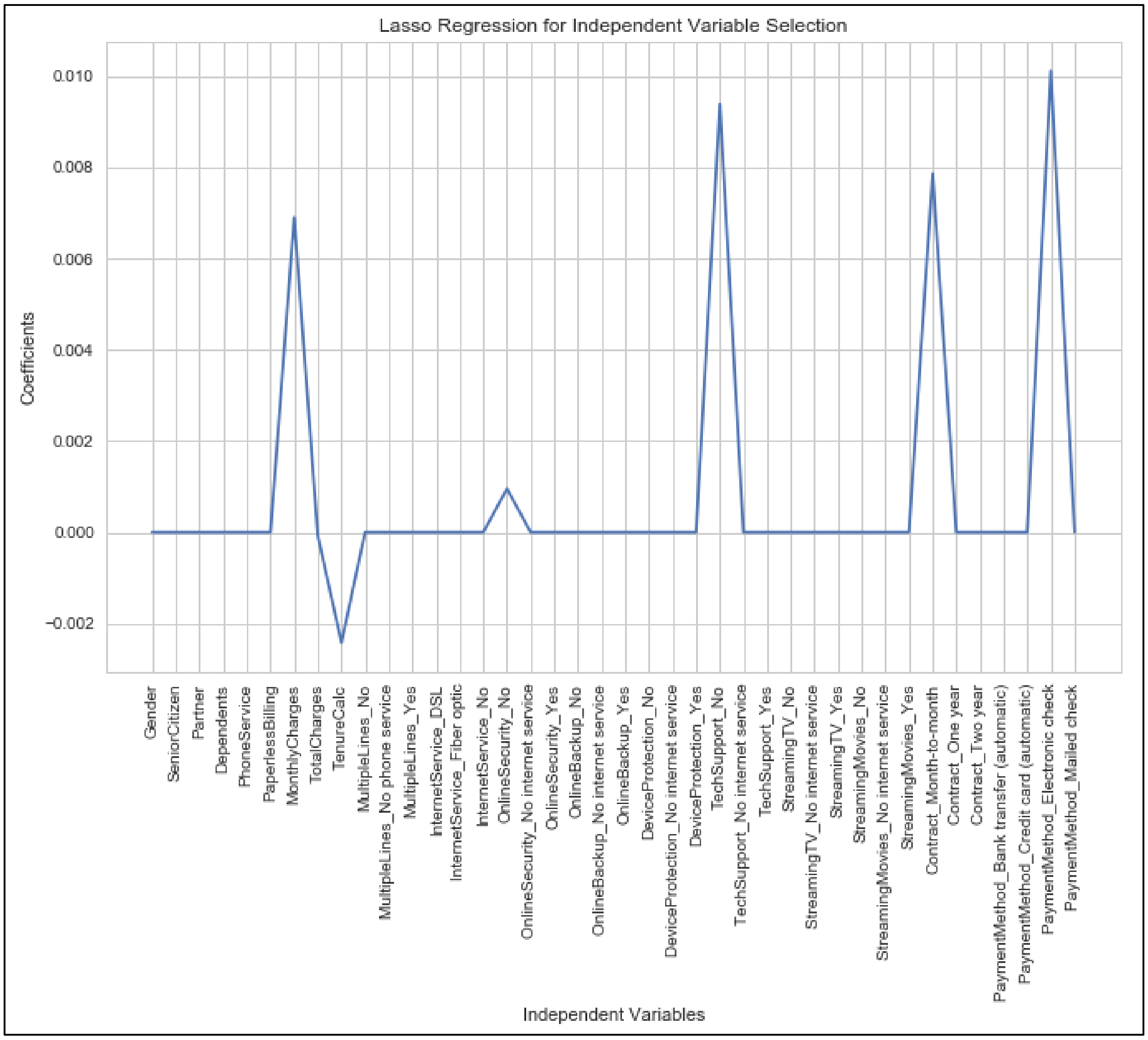


Figure 72. Lasso regression for independent variable selection using alpha = 0.22. Variable OnlineSecurity_No begins to emerge as a sixth variable.

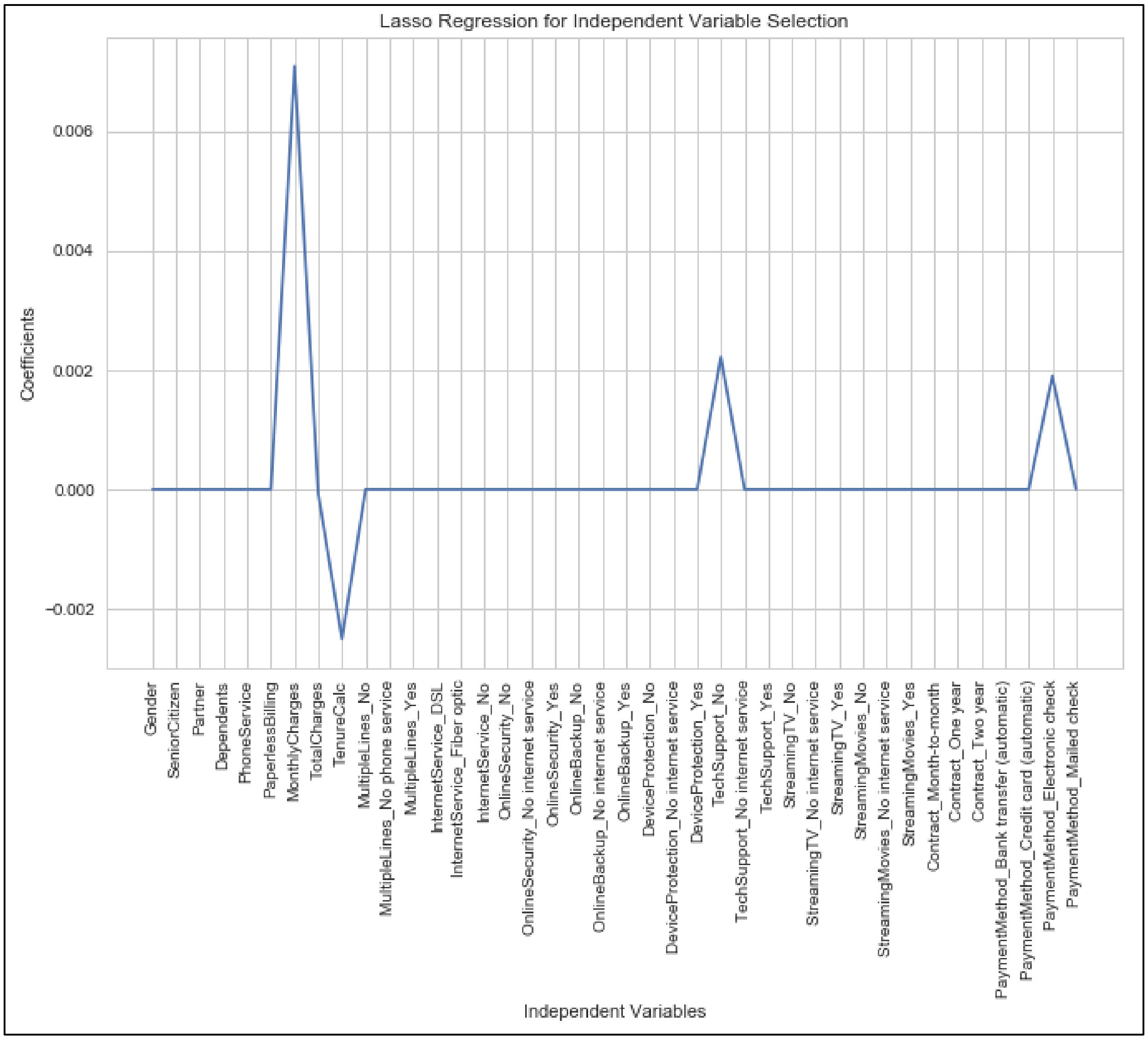


Figure 73. Lasso regression for independent variable selection using alpha = 0.24. Variable Contract_Month-to-month has disappeared.

Reference

Priyadharshini, G. (2017). *21 Reasons You Should Learn R, Python, and Hadoop.* Retrieved from https://www.simplilearn.com/21-reasons-to-learn-r-python-and-hadoop-article

Python Software Foundation (2018). *History and License* (Revised 2018, May 08). Retrieved from https://docs.python.org/3/license.html

SAS Institute Incorporated (2015). *License Agreement for the SAS® University Edition software* (Revised 2015, August 24). Retrieved from http://support.sas.com/legaldocs/univeditionlicense.pdf

The R Foundation (n.d.). *What is R? Introduction to R.* Retrieved from https://www.r-project.org/about.html

Wikipedia: The free encyclopedia (2018). *Linear discriminant analysis.* Retrieved from https://en.wikipedia.org/wiki/Linear_discriminant_analysis

Willems, K. (2015). *Choosing R or Python for Data Analysis? An Infographic.* Retrieved from https://www.datacamp.com/community/tutorials/r-or-python-for-data-analysis