I: Tool Selection

This project is done in Python using the various libraries in the scientific Python ecosystem of scikit-learn with a few other libraries for various uses. Specifically, this involves using pandas for data wrangling, manipulation, and cleaning, matplotlib and Seaborn for data visualization, and scikit-learn itself for model creation and analysis.

Like R and unlike SAS, all of these are easily available, free, and open-source. These methods have been chosen over R for ease of explanation, as Python code is often understood more readily than R, and because of the potential of integrating this project directly into a program or software for future use. While R is highly specialized for statistics and mathematics, Python is a general-purpose programming language with specialized libraries for the needed tools, and this facilitates project expansion in the future.

Data Exploration and Preparation

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as scs

%matplotlib inline
```

This dataset has 21 columns, with a number of which may be useful for predicting attrition, which is represented by the final column "Churn." Furthermore, none of the values are missing, or more specifically, NaN. While there is still potential for outliers, no data simply isn't there.

In [2]:

```
df = pd.read_csv("WA_Fn-UseC_-Telco-Customer-Churn.csv")
df.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
```

II: Project Goals

The goal of this data mining and analysis project is to examine customer attrition patterns and to determine what indicators can be used to make better business decisions to prevent loss. Furthermore, a model will be produced in order to predict whether or not a given customer might discontinue business based on a model built from this analysis. This process will involve cleaning and tidying the dataset, testing for significance and correlation, finding and removing interactions, preparing the cleaned dataset for the modeling, and examining the results of the data mining for conclusions.

The primary descriptive method used in this project is FAMD, or Factor Analysis of Mixed Data. FAMD will be performed to both examine the variables in order to determine which are the most important, and to reduce the number of variables to examine without overly reducing the explained variance so that the predictive method will be easier to create and use.

The primary predictive method will be logistic regression, and this technique will be used to build the predictive model. Logistic regression is appropriate because the dependent variable is Boolean and is easily predicted using the probability of binary results that logistic regression provides. Additionally, three of the variables are continuous, which factor easily into any type of linear regression, and the remaining are nominal categorical, which can easily factor into the regression by encoding them in dummy variables. Finally, the coefficients of each parameter that the model will provide can be used to see how much that variable affects the chance of churn. The result is a model that is robust, easy to understand, and is simple to use in predicting if a customer will churn or not.

III: Data Exploration and Preparation

The target and dependent variable in this data is the Churn column, which indicates whether or not that given customer has churned and ceased business. It is a nominal categorical variable that is originally loaded as a string or object dtype with yes/no values but is then tidied into a proper Boolean with True/False values.

There are many potential independent variables that can be used as predictor variables in this dataset. Two of the most important independent variables overall are the MonthlyCharges and the Tenure columns, which contains the continuous quantitative values that represent how much each customer is paying by month according to contract, and how long that customer has had a business relationship with this company. These are among the most important because of how strongly financial figures factor in to customers' decisions. A higher MonthlyCharges value may easy be representative of more pressure to leave to a cheaper competitor as soon as a contract is over, or in the case of a month by month contract, to end it sooner rather than later. Meanwhile, a higher Tenure value is indicative of a customer who has invested more thus far and is less likely to leave based on any single factor.

For other independent variables, PhoneService and InternetService do an excellent job of grouping customer type, and the various columns of data that require one of these grouping columns to have a value indicating that they're paying for that type of service can indicate that a disproportionate number of customers that are churning are all a very specific subset of customer. An example of a possibility is that if all customers with internet service, specifically DSL service, through this company are leaving and choosing to not renew contracts after previous ones are up, then reasons as to why this is need to specifically be looked into. Additionally, that DSL customers could be prime targets for customer relations programs, or improved sales, because of this relation. A competitor may be offering a sale far better than the company currently offers or is willing to match.

The first goal and requirement in manipulating the data is to have a dataset that can be considered both tidy and clean. For data to be tidy, each variable must have its own column, each observation must have its own row, and each value must have its own cell. (Wickham & Grolemund, 2017, 12.2) For data to be clean, it needs to be free from obvious errors, be formatted into the correct form of data, be devoid of missing values, have irrelevant columns removed, and to be consistent within itself. Additionally, the data will be broken down so that dummy encodings primarily represent the categorical data as opposed to the string values that exist in the initial dataset so that it can easily be implemented in an appropriate model. Beyond this, outliers and interactions must be found and removed if any exist to ensure the integrity of the resulting analyses and predictive model.

The base dataset has 1 identifier, the "customerID" column and 3 quantitative, continuous independent variables, "tenure," "MonthlyCharges," and "TotalCharges." Additionally, it has 16 nominal categorical independent variables, "gender", "SeniorCitizen", "Partner", "Dependents", "PhoneService", "MultipleLines", "InternetService", "OnlineSecurity", "OnlineBackup", "DeviceProtection", "TechSupport", "StreamingTV", "StreamingMovies", "Contract", “PaperlessBilling”, and “PaymentMethod.” Finally, the dataset has the 1 nominal categorical dependent variable “Churn.”

The essential criteria and phenomenon to be predicted are what factors are present with customers who are leaving, indicated by the Churn column, which can be detected by finding which independent variables have a high positive correlation with the Churn column. Additionally, finding the variables that are associated with a lower churn rate is useful for customer retention, and is an equally important goal. Beyond this, it is important to find what combination of all of these factors is most associated with Churn rate so that smart business decisions can be made in the future.

To begin the data cleaning process after the initial loading and examining of the dataframe structure, examine each column and flag any of the columns that have null values.

In [4]:
```
for col in df.columns.values:
    print(col + ": " + str(df[col].isnull().values.any()))
```

```
customerID: False
gender: False
SeniorCitizen: False
Partner: False
Dependents: False
tenure: False
PhoneService: False
MultipleLines: False
InternetService: False
OnlineSecurity: False
OnlineBackup: False
DeviceProtection: False
TechSupport: False
StreamingTV: False
StreamingMovies: False
Contract: False
PaperlessBilling: False
PaymentMethod: False
MonthlyCharges: False
TotalCharges: False
Churn: False
```

In [5]:
```
n_rows = len(df)
n_rows
```

Out[5]:
```
7043
```

While the customerID column is extremely important in relational databases and would be useful if we planned on any dataframe splits and subsequent joins, it is not useful for analysis and will be removed early on to simplify later steps.

The column customerID appears to be a structured string simply used for identifying customers. It isn't relevant to this exploratory data analysis, so it will be dropped.

In [6]:
```
df.drop('customerID', axis = 1, inplace = True)
```

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 20 columns):
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(17)
memory usage: 1.1+ MB
```

As the customerID column is the only one that will need removed at this stage of the data exploration, the remainder of this first step is to go through each column, one by one, and do the most basic cleaning and explorative tasks. This includes:

- Inferring the purpose of each column in the larger perspective of the business in order to interpret the information contained within properly.
- Changing the column titles to be consistent.
- Changing categorical values to be consistent like with Boolean values to True and False.
- Changing the data types, or dtypes, of some dataframe columns to ensure the data is handled appropriately.
- Examining the numbers and proportion of each categorical value in categorical columns.
- Examining the distribution and calculate summary statistics like the mean, median, and quartiles of the values of the quantitative columns.
- Finding and filling in any NaN or null values.

gender is a binary column and has a nearly perfect split between the values. 50.48% of customers are male and the remaining 49.52% are female. To make everything consistant, the column title will be updated.

In [8]:
```
df = df.rename(columns = {"gender": "Gender"})
```

In [9]:
```
df['Gender'].value_counts()
```

Out[9]:
```
Male      3555
Female    3488
Name: Gender, dtype: int64
```

In [10]:
```
df['Gender'].value_counts(normalize = True)
```

Out[10]:
```
Male      0.504756
Female    0.495244
Name: Gender, dtype: float64
```

SeniorCitizen appears to be a boolean value indicating whether or not the row corresponds to a customer that is a senior citizen, but what exact age that represents is unknown. 16.21% of customers are senior citizens in this dataset. This is the first example of a boolean column in this dataset, and they overall fail to follow a single convention. The boolean-like object columns will all be changed to an actual bool dtype as they come up.

In [11]:
```
df['SeniorCitizen'] = df['SeniorCitizen'].astype('bool')
df.info()
```

...

In [12]:
```
df['SeniorCitizen'].value_counts()
```

Out[12]:
```
False    5901
True     1142
Name: SeniorCitizen, dtype: int64
```

In [13]:
```
df['SeniorCitizen'].value_counts(normalize = True)
```

Out[13]:
```
False    0.837853
True     0.162147
Name: SeniorCitizen, dtype: float64
```

The Partner column may relate to a business partnership or some similar idea, but its position in between age related columns and dependants when the columns seem ordered by topic lead the belief of it indicating the existance of a spouse or established relationship partner instead in boolean form. 48.3% of customers in this dataset have a partner. It is another column that needs changed to the bool dtype.

In [14]:
```
df['Partner'] = df['Partner'].replace({"No":False, "Yes":True})
```

In [15]:
```
df['Partner'] = df['Partner'].astype('bool')
df.info()
```

...

In [16]:
```
df['Partner'].value_counts()
```

Out[16]:
```
False    3641
True     3402
Name: Partner, dtype: int64
```

In [17]:
```
df['Partner'].value_counts(normalize = True)
```

Out[17]:
```
False    0.516967
True     0.483033
Name: Partner, dtype: float64
```

Likewise, the Dependents column appears to be a boolean value of whether children or other legal dependents exist under the customer a given row represents that also needs converted to bool. 29.96% of customers in this dataset have dependents.

In [18]:
```
df['Dependents'] = df['Dependents'].replace({"No":False, "Yes":True})
df['Dependents'] = df['Dependents'].astype('bool')
df.info()
```

...

In [19]:
```
df['Dependents'].value_counts()
```

Out[19]:
```
False    4933
True     2110
Name: Dependents, dtype: int64
```

In [20]:
```
df['Dependents'].value_counts(normalize = True)
```

Out[20]:
```
False    0.700412
True     0.299588
Name: Dependents, dtype: float64
```

The tenure column, whose name notably does not follow the column naming conventions, seems to be the business meaning of the term, or the number of years that the customer has been with the company. Interestingly, the numbers range from 0 to 72. The highest value, 72, is the second most common while 0, is the lowest value and the least common. To make everything consistant, the column title will be updated.

In [21]:
```
df = df.rename(columns = {"tenure": "Tenure"})
```

In [22]:
```
df['Tenure'].value_counts()
```

Out[22]:
```
1     613
72    362
2     238
3     200
4     176
     ...
28     57
39     56
44     51
36     50
0      11
Name: Tenure, Length: 73, dtype: int64
```

PhoneServices is a boolean value indicating whether or not a customer is paying for phone service or not, with 90.32% of customers at this telecommunications company actively having a phone plan. Like others, it needs converted to the bool dtype.

In [23]:
```
df['PhoneService'] = df['PhoneService'].replace({"No":False, "Yes":True})
df['PhoneService'] = df['PhoneService'].astype('bool')
df.info()
```

...

In [24]:
```
df['PhoneService'].value_counts()
```

Out[24]:
```
True     6361
False     682
Name: PhoneService, dtype: int64
```

In [25]:
```
df['PhoneService'].value_counts(normalize = True)
```

Out[25]:
```
True     0.903166
False    0.096834
Name: PhoneService, dtype: float64
```

MultipleLines is a string value that indicates whether the given customer has multiple lines on their account, has only a single line on their account, or if this column isn't applicable for the customer, that they do not have phone service through this company. 42.18% of total customers in this dataset have multiple lines, and 46.71% of customers with phone service in this dataset have multiple lines.

In [26]:
```
df['MultipleLines'].value_counts()
```

Out[26]:
```
No                  3390
Yes                 2971
No phone service     682
Name: MultipleLines, dtype: int64
```

In [27]:
```
n_multiple_lines = df['MultipleLines'].value_counts()[1]
lines_proportion = n_multiple_lines / n_rows
print("{0:.4f}".format(lines_proportion) + "%")
```

```
0.4218%
```

In [28]:
```
lines_proportion2 = n_multiple_lines / (n_rows - df['MultipleLines'].value_coun
print("{0:.4f}".format(lines_proportion2) + "%")
```

```
0.4671%
```

InternetService is a string value indicating whether or not a customer is paying for internet service or not, and whether or not this service is for a fiber optic plan or a DSL plan. 90.32% of customers in this data service have an internet plan. 43.96% and 34.37% of total customers in this dataset have a fiber or DSL plan respectively, and 56.12% and 43.88% of customers with internet service have fiber or DSL plans respectively.

Many of the columns in this dataset have a number of possibilities that signify whether or not a service is included in the given customer's plan. Many of these columns would be boolean, but have a third value indicating that they do not have the required service plan for it to be a possibility. While these may be converted into bool dtype columns for later analysis, the boolean-like yes and no values will be changed to True or False on this first pass.

In [29]:
```
df['InternetService'] = df['InternetService'].replace({"No":"False"})
```

In [30]:
```
df['InternetService'].value_counts()
```

Out[30]:
```
Fiber optic    3096
DSL            2421
False          1526
Name: InternetService, dtype: int64
```

In [31]:
```
n_internet = n_rows - df['InternetService'].value_counts()[2]
internet_proportion = n_internet / n_rows
print("{0:.4f}".format(internet_proportion) + "%")
```

```
0.7833%
```

In [32]:
```
n_fiber = df['InternetService'].value_counts()[0]
n_dsl = df['InternetService'].value_counts()[1]
fiber_proportion = n_fiber / n_rows
dsl_proportion = n_dsl / n_rows
print("Fiber: {0:.4f}".format(fiber_proportion) + "%\nDSL: {0:.4f}".format(dsl_
```

```
Fiber: 0.4396%
DSL: 0.3437%
```

In [33]:
```
fiber_proportion2 = n_fiber / n_internet
dsl_proportion2 = n_dsl / n_internet
print("Fiber: {0:.4f}".format(fiber_proportion2) + "%\nDSL: {0:.4f}".format(dsl
```

```
Fiber: 0.5612%
DSL: 0.4388%
```

OnlineSecurity is a string value that indicates whether the given customer has an online security package in addition to their internet service, did not opt for the online security package, or if this column isn't applicable for the customer, that they do not have internet service through this company. 28.67% of total customers in this dataset have the online security package, and 36.6% of customers with internet service in this dataset have the online security package. Its values will be updated to the more boolean-like format.

In [34]:
```
df['OnlineSecurity'] = df['OnlineSecurity'].replace({"No":"False", "Yes":"True"
```

In [35]:
```
df['OnlineSecurity'].value_counts()
```

Out[35]:
```
False                  3498
True                   2019
No internet service    1526
Name: OnlineSecurity, dtype: int64
```

In [36]:
```
n_security = df['OnlineSecurity'].value_counts()[1]
security_proportion = n_security / n_rows
print("{0:.4f}".format(security_proportion) + "%")
```

```
0.2867%
```

In [37]:
```
security_proportion2 = n_security / (n_rows - df['OnlineSecurity'].value_counts
print("{0:.4f}".format(security_proportion2) + "%")
```

```
0.3660%
```

OnlineBackup is a string value that indicates whether the given customer has an online backup package in addition to their internet service, did not opt for the online backup package, or if this column isn't applicable for the customer, that they do not have internet service through this company. 34.49% of total customers in this dataset have the online security package, and 44.03% of customers with internet service in this dataset have the online security package. Its values will be updated to the more boolean-like format.

In [38]:
```
df['OnlineBackup'] = df['OnlineBackup'].replace({"No":"False", "Yes":"True"})
```

In [39]:
```
df['OnlineBackup'].value_counts()
```

Out[39]:
```
False                  3088
True                   2429
No internet service    1526
Name: OnlineBackup, dtype: int64
```

In [40]:
```
n_backup = df['OnlineBackup'].value_counts()[1]
backup_proportion = n_backup / n_rows
print("{0:.4f}".format(backup_proportion) + "%")
```

```
0.3449%
```

In [41]:
```
backup_proportion2 = n_backup / (n_rows - df['OnlineBackup'].value_counts()[2])
print("{0:.4f}".format(backup_proportion2) + "%")
```

```
0.4403%
```

DeviceProtection is a string value that indicates whether the given customer has a device protection plan in addition to their internet service, did not opt for any protection plan, or if this column isn't applicable for the customer, that they do not have internet service through this company. 34.39% of total customers in this dataset have a device protection plan, and 43.90% of customers with internet service in this dataset have a device protection plan. Its values will be updated to the more boolean-like format.

In [42]:
```
df['DeviceProtection'] = df['DeviceProtection'].replace({"No":"False", "Yes":"T
```

In [43]:
```
df['DeviceProtection'].value_counts()
```

Out[43]:
```
False                  3095
True                   2422
No internet service    1526
Name: DeviceProtection, dtype: int64
```

In [44]:
```
n_protection = df['DeviceProtection'].value_counts()[1]
protection_proportion = n_protection / n_rows
print("{0:.4f}".format(protection_proportion) + "%")
```

```
0.3439%
```

In [45]:
```
protection_proportion2 = n_protection / (n_rows - df['DeviceProtection'].value_
print("{0:.4f}".format(protection_proportion2) + "%")
```

```
0.4390%
```

OnlineBackup and DeviceProtection appear to be the first potential major issue for future analysis. Both require that the customer have a value other than no in the InternetService column, and also have extremely similar numbers, 2429 and 2422, 44.03% and 43.90% respectively. The correlation between the types of column values, namely the No internet service ones, are going to prevent proper testing and model building if tested together.

In [46]:
```
contingency_table = pd.crosstab(df['OnlineBackup'], df['DeviceProtection'])
contingency_table
```

Out[46]:

DeviceProtection	False	No internet service	True
OnlineBackup			
False	1984	0	1104
No internet service	0	1526	0
True	1111	0	1318

As we will see shortly, a number of the columns require internet service to have meaningful values and are in the same situation as the above section, so the data will have to be restructured to account for this.

TechSupport is a string value that indicates whether the given customer has a tech support package in addition to their internet service, did not opt for any tech support package, or if this column isn't applicable for the customer, that they do not have internet service through this company. 29.02% of total customers in this dataset have a tech support package, and 37.05% of customers with internet service in this dataset have a tech support package. Its values will be updated to the more boolean-like format.

In [47]:
```
df['TechSupport'] = df['TechSupport'].replace({"No":"False", "Yes":"True"})
```

In [48]:
```
df['TechSupport'].value_counts()
```

Out[48]:
```
False                  3473
True                   2044
No internet service    1526
Name: TechSupport, dtype: int64
```

In [49]:
```
n_support = df['TechSupport'].value_counts()[1]
support_proportion = n_support / n_rows
print("{0:.4f}".format(support_proportion) + "%")
```

```
0.2902%
```

In [50]:
```
support_proportion2 = n_support / (n_rows - df['TechSupport'].value_counts()[2]
print("{0:.4f}".format(support_proportion2) + "%")
```

```
0.3705%
```

StreamingTV is a string value that indicates whether the given customer has a TV streaming package in addition to their internet service, did not opt for any TV streaming package, or if this column isn't applicable for the customer, that they do not have internet service through this company. 38.44% of total customers in this dataset have a TV streaming package, and 49.07% of customers with internet service in this dataset have a TV streaming package. Its values will be updated to the more boolean-like format.

In [51]:
```
df['StreamingTV'] = df['StreamingTV'].replace({"No":"False", "Yes":"True"})
```

In [52]:
```
df['StreamingTV'].value_counts()
```

Out[52]:
```
False                  2810
True                   2707
No internet service    1526
Name: StreamingTV, dtype: int64
```

In [53]:
```
n_tv = df['StreamingTV'].value_counts()[1]
tv_proportion = n_tv / n_rows
print("{0:.4f}".format(tv_proportion) + "%")
```

```
0.3844%
```

In [54]:
```
tv_proportion2 = n_tv / (n_rows - df['StreamingTV'].value_counts()[2])
print("{0:.4f}".format(tv_proportion2) + "%")
```

```
0.4907%
```

StreamingMovies is a string value that indicates whether the given customer has a movie streaming package in addition to their internet service, did not opt for any movie streaming package, or if this column isn't applicable for the customer, that they do not have internet service through this company. 38.79% of total customers in this dataset have a movie streaming package, and 49.52% of customers with internet service in this dataset have a movie streaming package. Its values will be updated to the more boolean-like format.

In [55]:
```
df['StreamingMovies'] = df['StreamingMovies'].replace({"No":"False", "Yes":"Tru
```

In [56]:
```
df['StreamingMovies'].value_counts()
```

Out[56]:
```
False                  2785
True                   2732
No internet service    1526
Name: StreamingMovies, dtype: int64
```

In [57]:
```
n_movie = df['StreamingMovies'].value_counts()[1]
movie_proportion = n_movie / n_rows
print("{0:.4f}".format(movie_proportion) + "%")
```

```
0.3879%
```

In [58]:
```
movie_proportion2 = n_movie / (n_rows - df['StreamingMovies'].value_counts()[2]
print("{0:.4f}".format(movie_proportion2) + "%")
```

```
0.4952%
```

Contract is a string value that indicates what type of contract the given customer is under. The contract type values include Month-to-month, One year, and two year. 55.02% of customers are under a month to month contract, 24.07% are under a single year contract, and the remaining 20.91% are under a two year contract.

In [59]:
```
df['Contract'].value_counts()
```

Out[59]:
```
Month-to-month    3875
Two year          1695
One year          1473
Name: Contract, dtype: int64
```

In [60]:
```
df['Contract'].value_counts(normalize = True)
```

Out[60]:
```
Month-to-month    0.550192
Two year          0.240664
One year          0.209144
Name: Contract, dtype: float64
```

PaperlessBilling is a boolean value indicating whether or not the given customer has opted for paperless billing or not, with 59.22% of customers at this telecommunications company actively enrolled in paperless billing. As a true boolean column, it will be completely changed to the boolean dtype.

In [61]:
```
df['PaperlessBilling'] = df['PaperlessBilling'].replace({"No":False, "Yes":True
df['PaperlessBilling'] = df['PaperlessBilling'].astype('bool')
df.info()
```

...

In [62]:
```
df['PaperlessBilling'].value_counts()
```

Out[62]:
```
True     4171
False    2872
Name: PaperlessBilling, dtype: int64
```

In [63]:
```
df['PaperlessBilling'].value_counts(normalize = True)
```

Out[63]:
```
True     0.592219
False    0.407781
Name: PaperlessBilling, dtype: float64
```

PaymentMethod is a string value that indicates what type of payment method the given customer uses. The payment method values include electronic checks, traditional checks via mail, automatic bank transfer, and automatic payment with a credit card. 33.58% of customers pay via electronic check, 22.89% pay via mailed check, 21.92% pay automatically via bank transfer, and the remaining 21.61% pay automatically via credit card.

In [64]:
```
df['PaymentMethod'].value_counts()
```

Out[64]:
```
Electronic check             2365
Mailed check                 1612
Bank transfer (automatic)    1544
Credit card (automatic)      1522
Name: PaymentMethod, dtype: int64
```

In [65]:
```
df['PaymentMethod'].value_counts(normalize = True)
```

Out[65]:
```
Electronic check             0.335794
Mailed check                 0.228880
Bank transfer (automatic)    0.219225
Credit card (automatic)      0.216101
Name: PaymentMethod, dtype: float64
```

MonthlyCharges is a float value indicating the required monthly payment of a given customer. The mean value is 64.76, the mode value 20.05, the minimum 18.25, and the maximum value 118.75. This distribution is extremely skewed to the right.

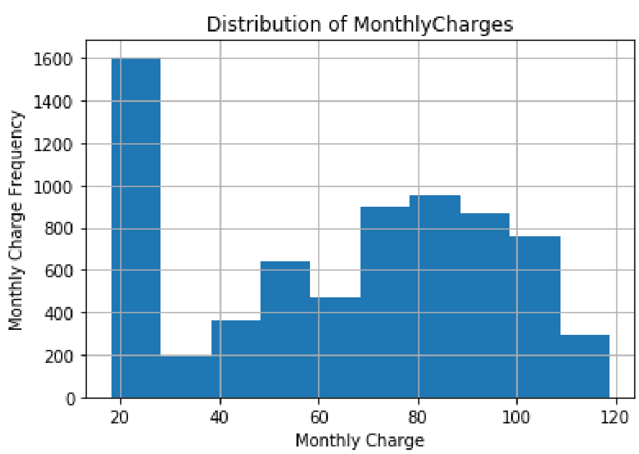
In [66]:
```
df['MonthlyCharges'].value_counts()
```

Out[66]:
```
20.05     61
19.85     45
19.95     44
19.90     44
20.00     43
          ..
114.75     1
103.60     1
113.40     1
57.65      1
113.30     1
Name: MonthlyCharges, Length: 1585, dtype: int64
```

In [67]:
```
df['MonthlyCharges'].describe()
```

Out[67]:
```
count    7043.000000
mean       64.761692
std        30.090047
min        18.250000
25%        35.500000
50%        70.350000
75%        89.850000
max       118.750000
Name: MonthlyCharges, dtype: float64
```

In [68]:
```
plt.hist(df['MonthlyCharges'])
plt.title('Distribution of MonthlyCharges')
plt.xlabel('Monthly Charge')
plt.ylabel('Monthly Charge Frequency')
plt.grid(True)
plt.show()
```



Here, 11 rows were found to have missing values in the TotalCharges column. As seen in the screenshots below, it was determined that these customers had simply yet to have been charged at the time this dataset was assembled, and as such a value of 0 was filled in for the TotalCharges column.

TotalCharges appears to be a column that represents the total amount a given customer has paid throughout all transactions of the business-customer relationship. However, the column is loaded as the object dtype, or a string, instead of a float. This will have to be changed to examine the data effectively.

In [69]:
```python
df['TotalCharges'].describe()
```

Out[69]:
```
count      7043
unique     6531
top        20.2
freq         11
Name: TotalCharges, dtype: object
```

These 11 rows have missing values for their TotalCharges column, but this appears to be the only missing columns. Notably, these customers' Tenure values are all 0, and none of them have churned. It seems likely that these customers don't have a TotalCharges value simply because they haven't been charged yet. As such, their TotalCharges value will be set to 0 to allow for later analysis.

In [70]:
```python
df.loc[pd.to_numeric(df['TotalCharges'], errors='coerce').isnull()]
```

...

In [71]:
```python
tofix = [488, 753, 936, 1082, 1340, 3331, 3826, 4380, 5218, 6670, 6754]

df.loc[tofix, 'TotalCharges'] = 0
```

In [72]:
```python
df.iloc[tofix]['TotalCharges']
```

Out[72]:
```
488     0
753     0
936     0
1082    0
1340    0
3331    0
3826    0
4380    0
5218    0
6670    0
6754    0
Name: TotalCharges, dtype: object
```

Now we can change the column to the float dtype and then examine the data. The mean TotalCharges value is 2283.30, the minimum value is 18.80, and the maximum is 8684.80. Overall, the column is very right-skewed.

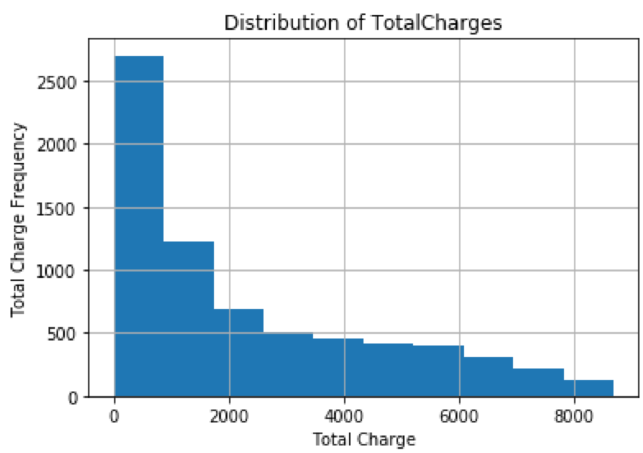
In [73]:
```
# Correct the TotalCharges column's dtype from object to float.
df['TotalCharges'] = df['TotalCharges'].astype('float')
df.info()
```

...

In [74]:
```
df['TotalCharges'].describe()
```

Out[74]:
```
count    7043.000000
mean     2279.734304
std      2266.794470
min         0.000000
25%       398.550000
50%      1394.550000
75%      3786.600000
max      8684.800000
Name: TotalCharges, dtype: float64
```

In [75]:
```
plt.hist(df['TotalCharges'])
plt.title('Distribution of TotalCharges')
plt.xlabel('Total Charge')
plt.ylabel('Total Charge Frequency')
plt.grid(True)
plt.show()
```



Finally, the last column of the dataset is the Churn column, which is a boolean value indicating on whether or not the customer has ceased business. 26.54% customers in this dataset have churned. As a true boolean column, it will be completely changed to the boolean dtype.

```
In [76]: df['Churn'] = df['Churn'].replace({"No":False, "Yes":True})
         df['Churn'] = df['Churn'].astype('bool')
         df.info()
```

...

```
In [77]: df['Churn'].value_counts()

Out[77]: False    5174
         True     1869
         Name: Churn, dtype: int64
```

```
In [78]: df['Churn'].value_counts(normalize = True)

Out[78]: False    0.73463
         True     0.26537
         Name: Churn, dtype: float64
```

```
In [79]: df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 20 columns):
Gender              7043 non-null object
SeniorCitizen       7043 non-null bool
Partner             7043 non-null bool
Dependents          7043 non-null bool
Tenure              7043 non-null int64
PhoneService        7043 non-null bool
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 bool
PaymentMethod       7043 non-null object
MonthlyCharges      7043 non-null float64
TotalCharges        7043 non-null float64
Churn               7043 non-null bool
dtypes: bool(6), float64(2), int64(1), object(11)
memory usage: 811.7+ KB
```

The next stage of cleaning and exploring the data is to look for outliers. Based on both IQR and z-scores, no outliers were found.

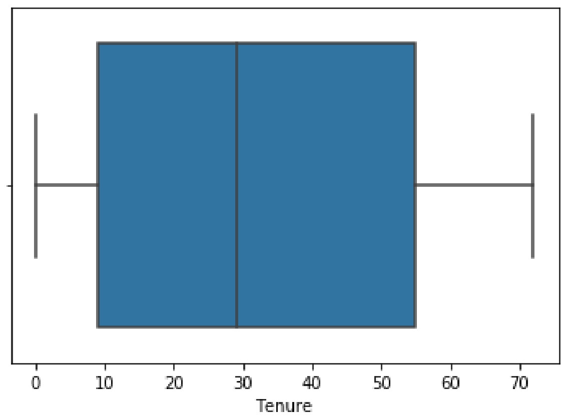
Outlier Detection

In this dataset, the three continuous variable columns do not have any outliers. None are present based on IQR range, as seen on the boxplots, and none are present based on the z-score with a threshold of 3 standard deviations.

Tenure

In [80]:
```
sns.boxplot(df['Tenure'])
```

Out[80]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x167e4173400>
```



In [81]:
```
tenure_z = np.abs(scs.zscore(df['Tenure']))
tenure_z
```

Out[81]:
```
array([1.27744458, 0.06632742, 1.23672422, ..., 0.87024095, 1.15528349,
       1.36937906])
```

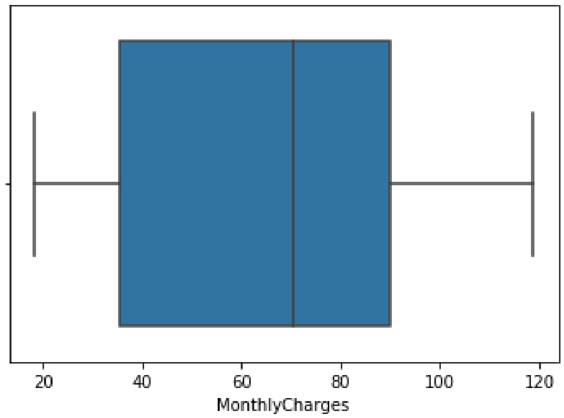
In [82]:
```
print(np.where(tenure_z > 3))
```

```
(array([], dtype=int64),)
```

MonthlyCharges

In [83]:
```
sns.boxplot(df['MonthlyCharges'])
```

Out[83]: <matplotlib.axes._subplots.AxesSubplot at 0x167e41d31d0>



In [84]:
```
monthly_z = np.abs(scs.zscore(df['MonthlyCharges']))
monthly_z
```

Out[84]:
```
array([1.16032292, 0.25962894, 0.36266036, ..., 1.1686319 , 0.32033821,
       1.35896134])
```

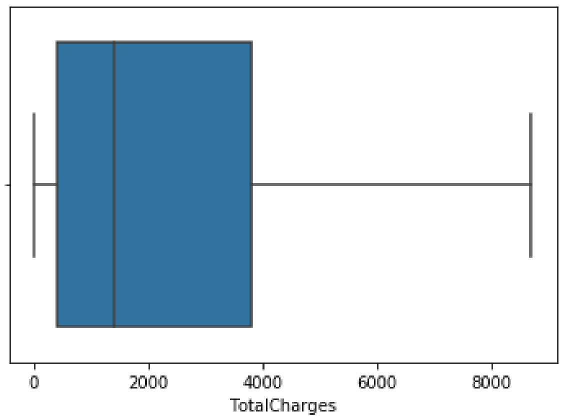
In [85]:
```
print(np.where(monthly_z > 3))
```

```
(array([], dtype=int64),)
```

TotalCharges

In [86]:
```
sns.boxplot(df['TotalCharges'])
```

Out[86]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x167e4249358>
```



In [87]:
```
total_z = np.abs(scs.zscore(df['TotalCharges']))
total_z
```

Out[87]:
```
array([0.99261052, 0.17216471, 0.9580659 , ..., 0.85293201, 0.87051315,
       2.01389665])
```

In [88]:
```
print(np.where(total_z > 3))
```

```
(array([], dtype=int64),)
```

The next step in this process is to transform the data into the structure that the models need. As logistic regression will be performed, the categorical columns will need to be encoded into dummy variables, which are expressed in Boolean values. In addition to the information encoding needing changed a number of these variables are inherently correlated with each other; e.g. MultipleLines with PhoneService, and OnlineSecurity with InternetService. To do meaningful tests of significance, this intercorrelation needs to be removed before the tests are run. Specifically, the problem is that the values of "no phone service" and "no internet service" are also factoring into the tests. There are a few approaches available in order deal with this.

The first is to split the dataframe into three distinct dataframes, one for cutomers with only phone service, one for customers with only internet service, and the last for customers with both phone and internet service through this company. This is less than ideal for several reasons. Rather than a single dataframe with single calculations, this would then create three dataframes and require three tests for each calculation, further complicating how all of the tests would be combined.

The second option is to simply remove a subsection of the rows, effectively removing columns of data in the process. This would technically work but it would also remove a significant amount of information in the process.

The remaining option is to manipulate the encoding and representation of the data. The categories that these all involve can be split into multiple boolean columns representing the "positive" values of the original columns. As an example, the MultipleLines column would be changed to be a boolean True/False option, with all rows that do not have phone service also being represented as False. Ultimately, the multiple logistic regression model is only going to adjust the calculation with a parameter if the value is true, and to create the model, these dummy variables must be created regardless, making this option the best.

In [93]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 20 columns):
Gender              7043 non-null object
SeniorCitizen       7043 non-null bool
Partner             7043 non-null bool
Dependents          7043 non-null bool
Tenure              7043 non-null int64
PhoneService        7043 non-null bool
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 bool
PaymentMethod       7043 non-null object
MonthlyCharges      7043 non-null float64
TotalCharges        7043 non-null float64
Churn               7043 non-null bool
dtypes: bool(6), float64(2), int64(1), object(11)
memory usage: 811.7+ KB
```

In [94]:
```
original_df = df.copy(deep = True)
```

Gender

In [95]:
```
df['Gender'].value_counts()
```

Out[95]:
```
Male      3555
Female    3488
Name: Gender, dtype: int64
```

In [96]:
```
df['Gender'] = df['Gender'].replace({"Male":False, "Female":True})
df['Gender'] = df['Gender'].astype('bool')
df['Gender'].value_counts()
```

Out[96]:
```
False    3555
True     3488
Name: Gender, dtype: int64
```

MultipleLines

In [97]:
```
df['MultipleLines'].value_counts()
```

Out[97]:
```
No                  3390
Yes                 2971
No phone service     682
Name: MultipleLines, dtype: int64
```

In [98]:
```
df.loc[df['MultipleLines'] == "No phone service", 'MultipleLines'] = np.NaN
```

In [99]:
```
df['MultipleLines'] = df['MultipleLines'].replace({"No":False, "Yes":True})
df['MultipleLines'] = df['MultipleLines'].fillna(False).astype('bool')
df['MultipleLines'].value_counts()
```

Out[99]:
```
False    4072
True     2971
Name: MultipleLines, dtype: int64
```

FiberOpticService

In [100]:
```
df['InternetService'].value_counts()
```

Out[100]:
```
Fiber optic    3096
DSL            2421
False          1526
Name: InternetService, dtype: int64
```

In [101]:
```
df['FiberOpticService'] = False
```

In [102]:
```
df.loc[df['InternetService'] == "Fiber optic", 'FiberOpticService'] = True
```

In [103]:
```
df['FiberOpticService'].value_counts()
```

Out[103]:
```
False    3947
True     3096
Name: FiberOpticService, dtype: int64
```

DSLService

In [104]:
```
df['DSLService'] = False
```

In [105]:
```
df.loc[df['InternetService'] == "DSL", 'DSLService'] = True
```

In [106]:
```
df['DSLService'].value_counts()
```

Out[106]:
```
False    4622
True     2421
Name: DSLService, dtype: int64
```

In [107]:
```
df.drop('InternetService', axis = 1, inplace = True)
```

OnlineSecurity

In [108]:
```
df['OnlineSecurity'].value_counts()
```

Out[108]:
```
False                  3498
True                   2019
No internet service    1526
Name: OnlineSecurity, dtype: int64
```

In [109]:
```
df.loc[df['OnlineSecurity'] == "No internet service", 'OnlineSecurity'] = np.Na
```

In [110]:
```
df['OnlineSecurity'] = df['OnlineSecurity'].replace({"False":False, "True":True
df['OnlineSecurity'] = df['OnlineSecurity'].fillna(False).astype('bool')
df['OnlineSecurity'].value_counts()
```

Out[110]:
```
False    5024
True     2019
Name: OnlineSecurity, dtype: int64
```

OnlineBackup

In [111]:
```
df['OnlineBackup'].value_counts()
```

Out[111]:
```
False                  3088
True                   2429
No internet service    1526
Name: OnlineBackup, dtype: int64
```

In [112]:
```
df.loc[df['OnlineBackup'] == "No internet service", 'OnlineBackup'] = np.NaN
```

In [113]:
```
df['OnlineBackup'] = df['OnlineBackup'].replace({"False":False, "True":True})
df['OnlineBackup'] = df['OnlineBackup'].fillna(False).astype('bool')
df['OnlineBackup'].value_counts()
```

Out[113]:
```
False    4614
True     2429
Name: OnlineBackup, dtype: int64
```

DeviceProtection

In [114]:
```
df['DeviceProtection'].value_counts()
```

Out[114]:
```
False                  3095
True                   2422
No internet service    1526
Name: DeviceProtection, dtype: int64
```

In [115]:
```
df.loc[df['DeviceProtection'] == "No internet service", 'DeviceProtection'] = n
```

In [116]:
```
df['DeviceProtection'] = df['DeviceProtection'].replace({"False":False, "True":
df['DeviceProtection'] = df['DeviceProtection'].fillna(False).astype('bool')
df['DeviceProtection'].value_counts()
```

Out[116]:
```
False    4621
True     2422
Name: DeviceProtection, dtype: int64
```

TechSupport

In [117]:
```
df['TechSupport'].value_counts()
```

Out[117]:
```
False                  3473
True                   2044
No internet service    1526
Name: TechSupport, dtype: int64
```

In [118]:
```
df.loc[df['TechSupport'] == "No internet service", 'TechSupport'] = np.NaN
```

In [119]:
```
df['TechSupport'] = df['TechSupport'].replace({"False":False, "True":True})
df['TechSupport'] = df['TechSupport'].fillna(False).astype('bool')
df['TechSupport'].value_counts()
```

Out[119]:
```
False    4999
True     2044
Name: TechSupport, dtype: int64
```

StreamingTV

In [120]:
```
df['StreamingTV'].value_counts()
```

Out[120]:
```
False                  2810
True                   2707
No internet service    1526
Name: StreamingTV, dtype: int64
```

In [121]:
```
df.loc[df['StreamingTV'] == "No internet service", 'StreamingTV'] = np.NaN
```

In [122]:
```
df['StreamingTV'] = df['StreamingTV'].replace({"False":False, "True":True})
df['StreamingTV'] = df['StreamingTV'].fillna(False).astype('bool')
df['StreamingTV'].value_counts()
```

Out[122]:
```
False    4336
True     2707
Name: StreamingTV, dtype: int64
```

StreamingMovies

In [123]:
```
df['StreamingMovies'].value_counts()
```

Out[123]:
```
False                  2785
True                   2732
No internet service    1526
Name: StreamingMovies, dtype: int64
```

In [124]:
```
df.loc[df['StreamingMovies'] == "No internet service", 'StreamingMovies'] = np.
```

In [125]:
```
df['StreamingMovies'] = df['StreamingMovies'].replace({"False":False, "True":Tr
df['StreamingMovies'] = df['StreamingMovies'].fillna(False).astype('bool')
df['StreamingMovies'].value_counts()
```

Out[125]:
```
False    4311
True     2732
Name: StreamingMovies, dtype: int64
```

Contract

The most common type of contract is the Month-to-month contract, so it will be the reference level, having both the new one year and two year contract columns as False.

In [126]:
```
df['Contract'].value_counts()
```

Out[126]:
```
Month-to-month    3875
Two year          1695
One year          1473
Name: Contract, dtype: int64
```

In [127]:
```
df['OneYearContract'] = False
df['TwoYearContract'] = False
```

In [128]:
```
df.loc[df['Contract'] == "One year", 'OneYearContract'] = True
df.loc[df['Contract'] == "Two year", 'TwoYearContract'] = True
```

In [129]:
```
df['OneYearContract'].value_counts()
```

Out[129]:
```
False    5570
True     1473
Name: OneYearContract, dtype: int64
```

In [130]:
```
df['TwoYearContract'].value_counts()
```

Out[130]:
```
False    5348
True     1695
Name: TwoYearContract, dtype: int64
```

In [131]:
```
df.drop('Contract', axis = 1, inplace = True)
```

PaymentMethod

The most common type of payment method is by electronic check, so it will be the reference level, having the new columns mailed check, bank transfer, and credit card as False.

In [132]:
```
df['PaymentMethod'].value_counts()
```

Out[132]:
```
Electronic check             2365
Mailed check                 1612
Bank transfer (automatic)    1544
Credit card (automatic)      1522
Name: PaymentMethod, dtype: int64
```

In [133]:
```
df['MailedCheckPayment'] = False
df['BankTransferPayment'] = False
df['CreditCardPayment'] = False
```

In [134]:
```
df.loc[df['PaymentMethod'] == "Mailed check", 'MailedCheckPayment'] = True
df.loc[df['PaymentMethod'] == "Bank transfer (automatic)", 'BankTransferPayment
df.loc[df['PaymentMethod'] == "Credit card (automatic)", 'CreditCardPayment'] =
```

In [135]:
```
df['MailedCheckPayment'].value_counts()
```

Out[135]:
```
False    5431
True     1612
Name: MailedCheckPayment, dtype: int64
```

In [136]:
```
df['BankTransferPayment'].value_counts()
```

Out[136]:
```
False    5499
True     1544
Name: BankTransferPayment, dtype: int64
```

In [137]:
```
df['CreditCardPayment'].value_counts()
```

Out[137]:
```
False    5521
True     1522
Name: CreditCardPayment, dtype: int64
```

In [138]:
```
df.drop('PaymentMethod', axis = 1, inplace = True)
```

Finally, reorganize all of the columns back to the order that the original dataset was in.

In [139]:
```
cols = df.columns
cols
```

Out[139]:
```
Index(['Gender', 'SeniorCitizen', 'Partner', 'Dependents', 'Tenure',
       'PhoneService', 'MultipleLines', 'OnlineSecurity', 'OnlineBackup',
       'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovies',
       'PaperlessBilling', 'MonthlyCharges', 'TotalCharges', 'Churn',
       'FiberOpticService', 'DSLService', 'OneYearContract', 'TwoYearContrac
t',
       'MailedCheckPayment', 'BankTransferPayment', 'CreditCardPayment'],
      dtype='object')
```

In [140]:
```
# Reorders the columns back to how the original dataframe had them.
df = df[['Gender', 'SeniorCitizen', 'Partner', 'Dependents', 'Tenure',
         'PhoneService', 'MultipleLines', 'FiberOpticService', 'DSLService',
         'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport',
         'StreamingTV', 'StreamingMovies', 'OneYearContract', 'TwoYearContract'
         'PaperlessBilling', 'MailedCheckPayment', 'BankTransferPayment',
         'CreditCardPayment', 'MonthlyCharges', 'TotalCharges', 'Churn']]
```

This section culminates into the dataframe below. All categorical variables have been recoded into Boolean values for the future regression modeling and the continuous variables are still intact as integer for the Tenure column, and floating point for the two charges columns.

```
In [141]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 24 columns):
Gender                 7043 non-null bool
SeniorCitizen          7043 non-null bool
Partner                7043 non-null bool
Dependents             7043 non-null bool
Tenure                 7043 non-null int64
PhoneService           7043 non-null bool
MultipleLines          7043 non-null bool
FiberOpticService      7043 non-null bool
DSLService             7043 non-null bool
OnlineSecurity         7043 non-null bool
OnlineBackup           7043 non-null bool
DeviceProtection       7043 non-null bool
TechSupport            7043 non-null bool
StreamingTV            7043 non-null bool
StreamingMovies        7043 non-null bool
OneYearContract        7043 non-null bool
TwoYearContract        7043 non-null bool
PaperlessBilling       7043 non-null bool
MailedCheckPayment     7043 non-null bool
BankTransferPayment    7043 non-null bool
CreditCardPayment      7043 non-null bool
MonthlyCharges         7043 non-null float64
TotalCharges           7043 non-null float64
Churn                  7043 non-null bool
dtypes: bool(21), float64(2), int64(1)
memory usage: 309.6 KB
```

Later on, during the bivariate data visualization, contingency tables were created for each categorical column of the dataframe, and chi squared tests were performed on each of these to check for statistical significance in their relationship with Churn rate. Gender and PhoneService were found to not be significant by wide margins, and were removed.

The two columns that did not have significance will be removed to simplify later calculations.

```
In [213]: df.drop(['Gender', 'PhoneService'], axis = 1, inplace = True)
```

As an iterative process, data cleaning and manipulation can occur throughout an entire project. After the MCA, PCA, and FAMD were performed, two more columns were removed after finding the interactions they caused.

Now the columns identified as interactions or flagged for removal with the MCA, PCA, and correlation tests can be removed.

```
In [234]: df.drop(['StreamingTV', 'TotalCharges'], axis = 1, inplace = True)
```

IV: Data Analysis

As all of the distributions have been examined in the data exploration and cleaning stage, this section acts as a recap and a point at which to actually represent them graphically. The following code reformats the cleaned dataframe into its long form so that the resulting graphs are generated together in a single chart with shared axes. This should allow for easy comparison of proportions. The large chart of all categorical variables is followed by the summary statistics and a histogram showing the distribution of each of the continuous variables.

Univariate Distribution Visualization

In [142]:
```
viz_df = df.drop(['Tenure', 'MonthlyCharges', 'TotalCharges'], axis = 1)
viz_df.columns
```

Out[142]:
```
Index(['Gender', 'SeniorCitizen', 'Partner', 'Dependents', 'PhoneService',
       'MultipleLines', 'FiberOpticService', 'DSLService', 'OnlineSecurity',
       'OnlineBackup', 'DeviceProtection', 'TechSupport', 'StreamingTV',
       'StreamingMovies', 'OneYearContract', 'TwoYearContract',
       'PaperlessBilling', 'MailedCheckPayment', 'BankTransferPayment',
       'CreditCardPayment', 'Churn'],
      dtype='object')
```

In [143]:
```
melted = viz_df.melt()
```

In [144]:
```
melted
```

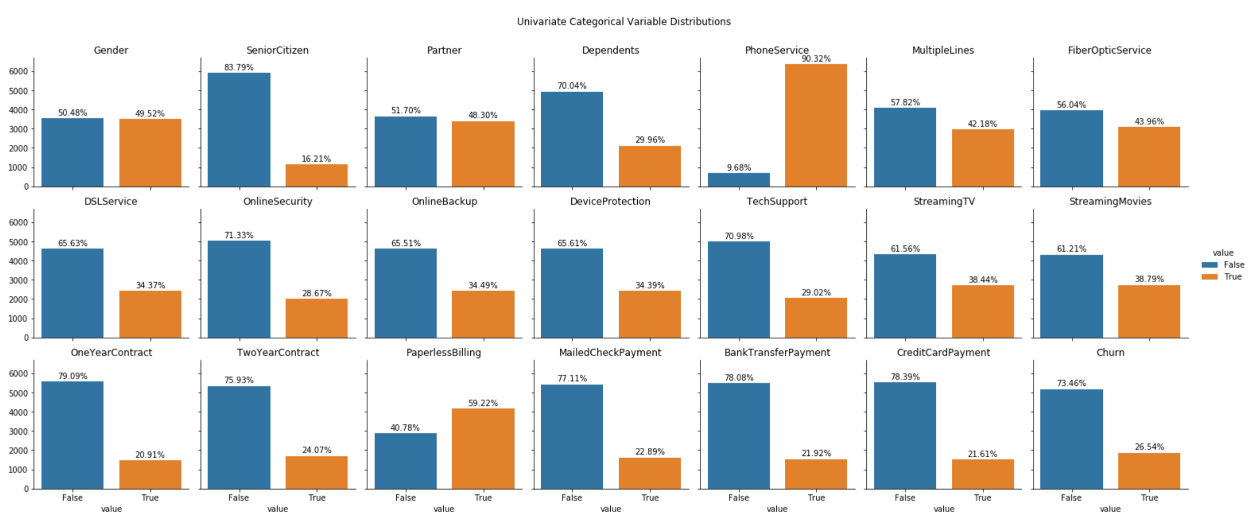
Out[144]:

	variable	value
0	Gender	True
1	Gender	False
2	Gender	False
3	Gender	False
4	Gender	True
...	...	...
147898	Churn	False
147899	Churn	False
147900	Churn	False
147901	Churn	True
147902	Churn	False

147903 rows × 2 columns

In [145]:
```python
g = sns.FacetGrid(
    melted,
    col = 'variable',
    hue = 'value',
    sharey = 'row',
    sharex = 'col',
    col_wrap = 7,
    legend_out = True,
)

g = g.map(sns.countplot, 'value', order = [False, True]).add_legend()

plt.subplots_adjust(top = 0.9)
g.fig.suptitle('Univariate Categorical Variable Distributions')

# Add the percentage proportion value for each column in each variable plot.
for i, ax in enumerate(g.axes):
    # Set each plot's title to its corresponding column name.
    ax.set_title(viz_df.columns[i])
    for p in ax.patches:
        percentage = "{0:.2f}".format((p.get_height() / n_rows) * 100) + "%"

        ax.annotate(percentage, (p.get_x() + p.get_width() / 2., p.get_height()
                    ha = 'center', va = 'center', xytext = (0, 7), textcoords =


g.savefig("Univariate Distributions.png")
g
```

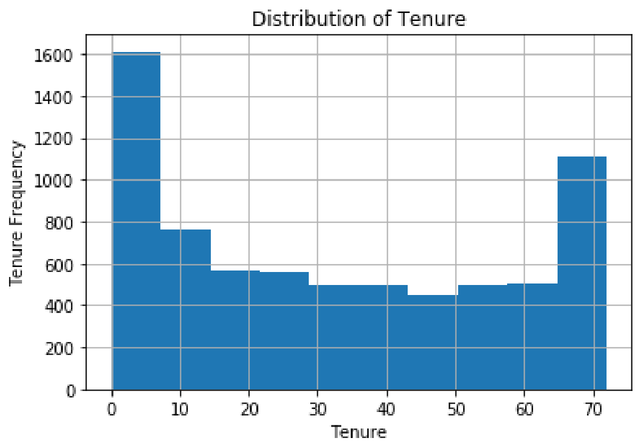
Out[145]: <seaborn.axisgrid.FacetGrid at 0x243d67b2dd8>



In [146]:
```
df['Tenure'].describe()
```

Out[146]:
```
count    7043.000000
mean       32.371149
std        24.559481
min         0.000000
25%         9.000000
50%        29.000000
75%        55.000000
max        72.000000
Name: Tenure, dtype: float64
```

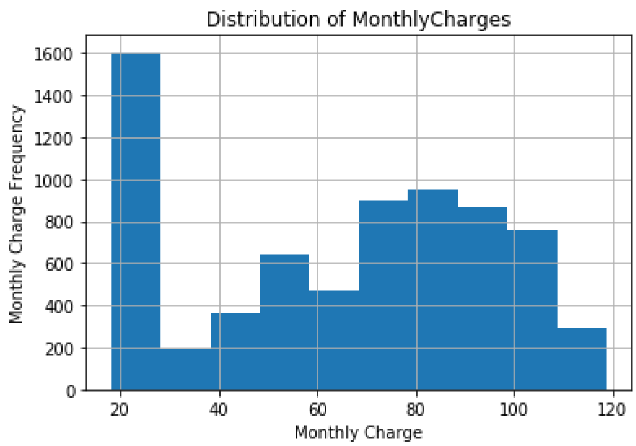
In [147]:
```
plt.hist(df['Tenure'])
plt.title('Distribution of Tenure')
plt.xlabel('Tenure')
plt.ylabel('Tenure Frequency')
plt.grid(True)
plt.show()
```



In [148]:
```
df['MonthlyCharges'].describe()
```

Out[148]:
```
count    7043.000000
mean       64.761692
std        30.090047
min        18.250000
25%        35.500000
50%        70.350000
75%        89.850000
max       118.750000
Name: MonthlyCharges, dtype: float64
```

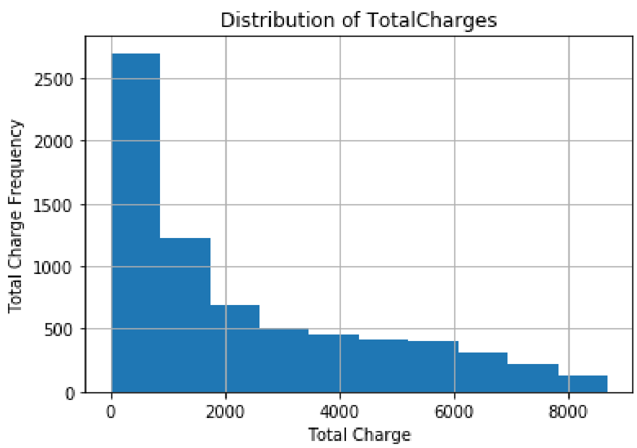
In [149]:
```
plt.hist(df['MonthlyCharges'])
plt.title('Distribution of MonthlyCharges')
plt.xlabel('Monthly Charge')
plt.ylabel('Monthly Charge Frequency')
plt.grid(True)
plt.show()
```



In [150]:
```
df['TotalCharges'].describe()
```

Out[150]:
```
count    7043.000000
mean     2279.734304
std      2266.794470
min         0.000000
25%       398.550000
50%      1394.550000
75%      3786.600000
max      8684.800000
Name: TotalCharges, dtype: float64
```

In [151]:
```
plt.hist(df['TotalCharges'])
plt.title('Distribution of TotalCharges')
plt.xlabel('Total Charge')
plt.ylabel('Total Charge Frequency')
plt.grid(True)
plt.show()
```



For ease of interpretation and usage given the 2x2 relationships between churn and the other categorical variables, simple contingency tables were used to visualize the categorical bivariate distributions. Bar, pie, and mosaic plots were all considered, but all seemed to overcomplicate the presentation. In each of the following sections, the first table is the raw numbers of the categorical variables, and the second table contains the same information but normalized into proportions. The two columns Contract and PaymentMethod, have more values than just True/False. These have been visualized with bar plots grouped by the rows' Churn values.

Gender

In [152]:
```
gender_contingency = pd.crosstab(df["Gender"], df["Churn"])
gender_contingency
```

Out[152]:

Churn	False	True
Gender		
False	2625	930
True	2549	939

In [153]:
```
pd.crosstab(df["Gender"], df["Churn"], normalize = "index")
```

Out[153]:

Churn	False	True
Gender		
False	0.738397	0.261603
True	0.730791	0.269209

SeniorCitizen

In [155]:
```
senior_contingency = pd.crosstab(df["SeniorCitizen"], df["Churn"])
senior_contingency
```

Out[155]:

Churn	False	True
SeniorCitizen		
False	4508	1393
True	666	476

In [156]:
```
pd.crosstab(df["SeniorCitizen"], df["Churn"], normalize = "index")
```

Out[156]:

Churn	False	True
SeniorCitizen		
False	0.763938	0.236062
True	0.583187	0.416813

Partner

In [158]:
```
partner_contingency = pd.crosstab(df["Partner"], df["Churn"])
partner_contingency
```

Out[158]:

Churn	False	True
Partner		
False	2441	1200
True	2733	669

In [159]:
```
pd.crosstab(df["Partner"], df["Churn"], normalize = "index")
```

Out[159]:

Churn	False	True
Partner		
False	0.670420	0.329580
True	0.803351	0.196649

Dependents

In [161]:
```
dependents_contingency = pd.crosstab(df["Dependents"], df["Churn"])
dependents_contingency
```

Out[161]:

Churn	False	True
Dependents		
False	3390	1543
True	1784	326

In [162]:
```
pd.crosstab(df["Dependents"], df["Churn"], normalize = "index")
```

Out[162]:

Churn	False	True
Dependents		
False	0.687209	0.312791
True	0.845498	0.154502

PhoneService

In [164]:
```
phone_contingency = pd.crosstab(df["PhoneService"], df["Churn"])
phone_contingency
```

Out[164]:

Churn	False	True
PhoneService		
False	512	170
True	4662	1699

In [165]:
```
pd.crosstab(df["PhoneService"], df["Churn"], normalize = "index")
```

Out[165]:

Churn	False	True
PhoneService		
False	0.750733	0.249267
True	0.732904	0.267096

MultipleLines

In [167]:
```
lines_contingency = pd.crosstab(df["MultipleLines"], df["Churn"])
lines_contingency
```

Out[167]:

Churn	False	True
MultipleLines		
False	3053	1019
True	2121	850

In [168]:
```
pd.crosstab(df["MultipleLines"], df["Churn"], normalize = "index")
```

Out[168]:

Churn	False	True
MultipleLines		
False	0.749754	0.250246
True	0.713901	0.286099

InternetService

In [170]:
```
fiber_contingency = pd.crosstab(df["FiberOpticService"], df["Churn"])
fiber_contingency
```

Out[170]:

Churn	False	True
FiberOpticService		
False	3375	572
True	1799	1297

In [171]:
```
pd.crosstab(df["FiberOpticService"], df["Churn"], normalize = "index")
```

Out[171]:

Churn	False	True
FiberOpticService		
False	0.855080	0.144920
True	0.581072	0.418928

In [173]:
```
dsl_contingency = pd.crosstab(df["DSLService"], df["Churn"])
dsl_contingency
```

Out[173]:

Churn	False	True
DSLService		
False	3212	1410
True	1962	459

In [174]:
```
pd.crosstab(df["DSLService"], df["Churn"], normalize = "index")
```

Out[174]:

Churn	False	True
DSLService		
False	0.694937	0.305063
True	0.810409	0.189591

OnlineSecurity

In [176]:
```
security_contingency = pd.crosstab(df["OnlineSecurity"], df["Churn"])
security_contingency
```

Out[176]:

Churn	False	True
OnlineSecurity		
False	3450	1574
True	1724	295

In [177]:
```
pd.crosstab(df["OnlineSecurity"], df["Churn"], normalize = "index")
```

Out[177]:

Churn	False	True
OnlineSecurity		
False	0.686704	0.313296
True	0.853888	0.146112

OnlineBackup

In [179]:
```
backup_contingency = pd.crosstab(df["OnlineBackup"], df["Churn"])
backup_contingency
```

Out[179]:

Churn	False	True
OnlineBackup		
False	3268	1346
True	1906	523

In [180]:
```
pd.crosstab(df["OnlineBackup"], df["Churn"], normalize = "index")
```

Out[180]:

Churn	False	True
OnlineBackup		
False	0.708279	0.291721
True	0.784685	0.215315

DeviceProtection

In [182]:
```
dev_protection_contingency = pd.crosstab(df["DeviceProtection"], df["Churn"])
dev_protection_contingency
```

Out[182]:

Churn	False	True
DeviceProtection		
False	3297	1324
True	1877	545

In [183]:
```
pd.crosstab(df["DeviceProtection"], df["Churn"], normalize = "index")
```

Out[183]:

Churn	False	True
DeviceProtection		
False	0.713482	0.286518
True	0.774979	0.225021

TechSupport

In [185]:
```
support_contingency = pd.crosstab(df["TechSupport"], df["Churn"])
support_contingency
```

Out[185]:

Churn	False	True
TechSupport		
False	3440	1559
True	1734	310

In [186]:
```
pd.crosstab(df["TechSupport"], df["Churn"], normalize = "index")
```

Out[186]:

Churn	False	True
TechSupport		
False	0.688138	0.311862
True	0.848337	0.151663

StreamingTV

In [188]:
```
tv_contingency = pd.crosstab(df["StreamingTV"], df["Churn"])
tv_contingency
```

Out[188]:

Churn StreamingTV	False	True
False	3281	1055
True	1893	814

In [189]:
```
pd.crosstab(df["StreamingTV"], df["Churn"], normalize = "index")
```

Out[189]:

Churn StreamingTV	False	True
False	0.756688	0.243312
True	0.699298	0.300702

StreamingMovies

In [191]:
```
movies_contingency = pd.crosstab(df["StreamingMovies"], df["Churn"])
movies_contingency
```

Out[191]:

Churn StreamingMovies	False	True
False	3260	1051
True	1914	818

In [192]:
```
pd.crosstab(df["StreamingMovies"], df["Churn"], normalize = "index")
```

Out[192]:

Churn StreamingMovies	False	True
False	0.756205	0.243795
True	0.700586	0.299414

Contract

In [194]:
```
contract_contingency = pd.crosstab(original_df["Contract"], original_df["Churn"
contract_contingency
```

Out[194]:

Churn	False	True
Contract		
Month-to-month	2220	1655
One year	1307	166
Two year	1647	48

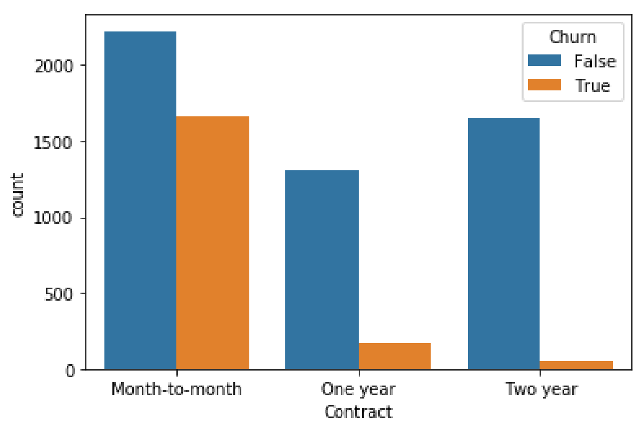
In [195]:
```
pd.crosstab(original_df["Contract"], original_df["Churn"], normalize = "index")
```

Out[195]:

Churn	False	True
Contract		
Month-to-month	0.572903	0.427097
One year	0.887305	0.112695
Two year	0.971681	0.028319

In [196]:
```
sns.countplot(x = original_df["Contract"], hue = original_df["Churn"])
```

Out[196]: <matplotlib.axes._subplots.AxesSubplot at 0x26494207160>



The χ^2 test shows significance, so this column should be examined in more detail later. Customers who had been in a month to month contract in this data set have a substantial difference in churn rate, likely because of the lack of legal agreements preventing ease of churn, and should be examined in much greater detail.

PaperlessBilling

In [198]:
```
paperless_contingency = pd.crosstab(df["PaperlessBilling"], df["Churn"])
paperless_contingency
```

Out[198]:

Churn	False	True
PaperlessBilling		
False	2403	469
True	2771	1400

In [199]:
```
pd.crosstab(df["PaperlessBilling"], df["Churn"], normalize = "index")
```

Out[199]:

Churn	False	True
PaperlessBilling		
False	0.836699	0.163301
True	0.664349	0.335651

PaymentMethod

In [201]:
```
payment_contingency = pd.crosstab(original_df["PaymentMethod"], original_df["Ch
payment_contingency
```

Out[201]:

Churn	False	True
PaymentMethod		
Bank transfer (automatic)	1286	258
Credit card (automatic)	1290	232
Electronic check	1294	1071
Mailed check	1304	308

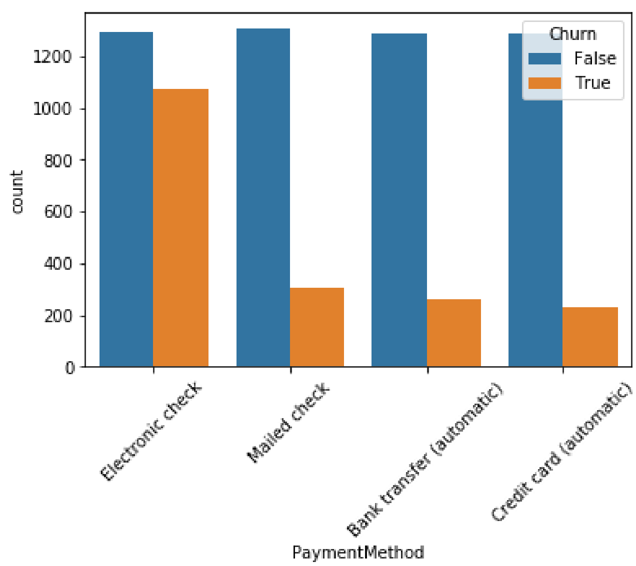
In [202]:
```
pd.crosstab(original_df["PaymentMethod"], original_df["Churn"], normalize = "in
```

Out[202]:

Churn	False	True
PaymentMethod		
Bank transfer (automatic)	0.832902	0.167098
Credit card (automatic)	0.847569	0.152431
Electronic check	0.547146	0.452854
Mailed check	0.808933	0.191067

In [203]:
```
fig = sns.countplot(x = original_df["PaymentMethod"], hue = original_df["Churn"
fig.set_xticklabels(fig.get_xticklabels(), rotation = 45)
```

Out[203]:
```
[Text(0, 0, 'Electronic check'),
 Text(0, 0, 'Mailed check'),
 Text(0, 0, 'Bank transfer (automatic)'),
 Text(0, 0, 'Credit card (automatic)')]
```



The three continuous variables, Tenure, MonthlyCharges, and TotalCharges, are visualized through Seaborn's distplot function, which overlays a histogram of the data with a kernel density estimate. The dataframe is divided by the Churn column, and for each continuous column, both dataframes' distributions via histograms are superimposed for direct comparison.

Continuous Variable Churn Distribution

- Tenure
- MonthlyCharges
- TotalCharges

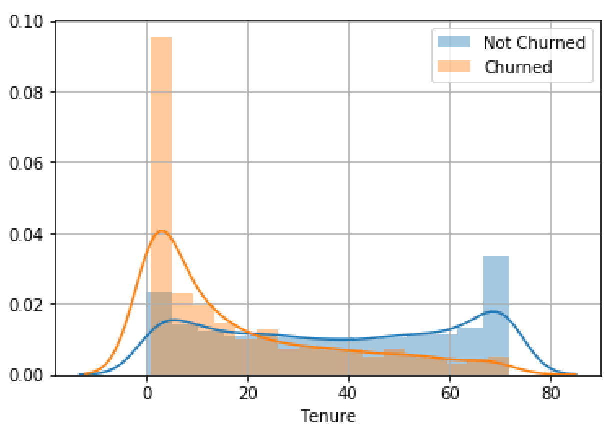
In [205]:
```
churn_df = df[df['Churn'] == True]
churn_df.info()
```
...

In [206]:
```
not_churn_df = df[df['Churn'] == False]
not_churn_df.info()
```
...

Tenure

In regards to the customers who churned and those that did not in this data set, there is a very substantial difference in Tenure. As expected, the lower the tenure, the higher the relative percentage of churn rate.

In [207]:
```
sns.distplot(not_churn_df['Tenure'])
sns.distplot(churn_df['Tenure'])
plt.grid(True)
plt.legend(["Not Churned", "Churned"])
```
Out[207]:
```
<matplotlib.legend.Legend at 0x264942fb630>
```

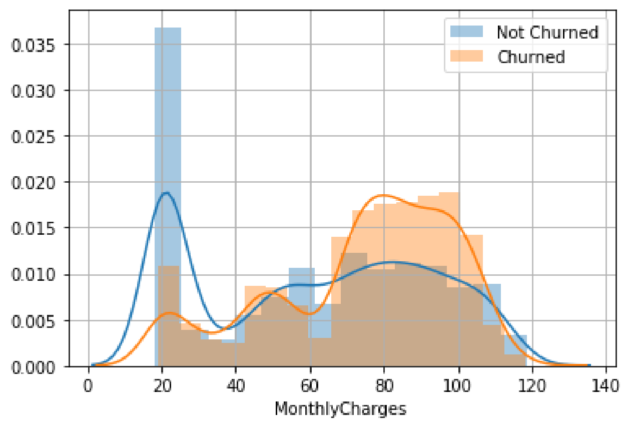


MonthlyCharges

In looking at the MonthlyCharges and TotalCharges values, it becomes clear that the churned and not churned distribution differ on these columns. The churn rate is much higher when a customer has a higher MonthlyCharges value. This is to be expected since a higher monthly bill causes more financial pressure on the customer and gives a greater performance expectation on the end of the service provider. It appears that over 35% of customers who have not churned have 20-25 as their MonthlyCharges value.

In [209]:
```
sns.distplot(not_churn_df['MonthlyCharges'])
sns.distplot(churn_df['MonthlyCharges'])
plt.grid(True)
plt.legend(["Not Churned", "Churned"])
```

Out[209]:
```
<matplotlib.legend.Legend at 0x26494439da0>
```

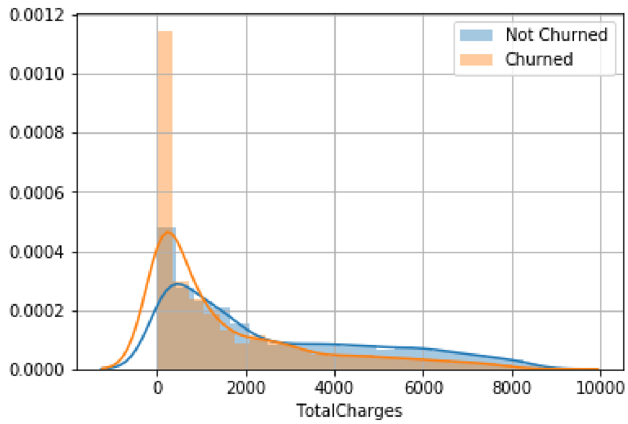


Total Charges

With the TotalCharges column, the distribution in churned and not churned is less of a difference than in the MonthlyCharges column, but is still substantial. The lower the TotalCharges, the higher the churn rate. It appears that over 10% of all customers who have churned had less than 400 TotalCharges.

In [211]:
```
sns.distplot(not_churn_df['TotalCharges'])
sns.distplot(churn_df['TotalCharges'])
plt.grid(True)
plt.legend(["Not Churned", "Churned"])
```

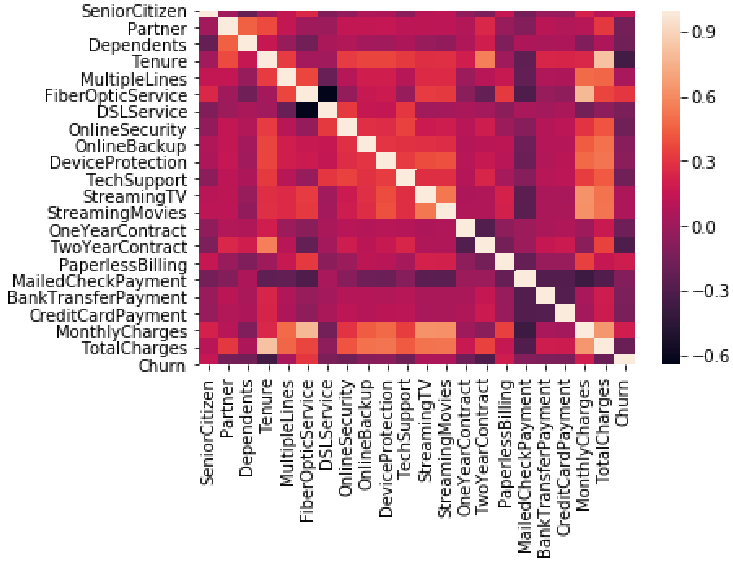
Out[211]: <matplotlib.legend.Legend at 0x264949e8e80>



Finally, for multivariate visualization, a heatmap of the dataframe's correlation coefficient matrix was created and used to look for correlation between the columns. Given the scenario, the most important correlations to look for are those pertaining to the Churn column, which is the focus in prediction. This visualization is very import to look for interactions that need removed, like the extremely high correlation coefficient that exists between the Tenure and TotalCharges columns.

Correlation

```
In [215]: sns.heatmap(
              df.corr(),
              xticklabels = df.columns,
              yticklabels = df.columns
          )

Out[215]: <matplotlib.axes._subplots.AxesSubplot at 0x264941859e8>
```



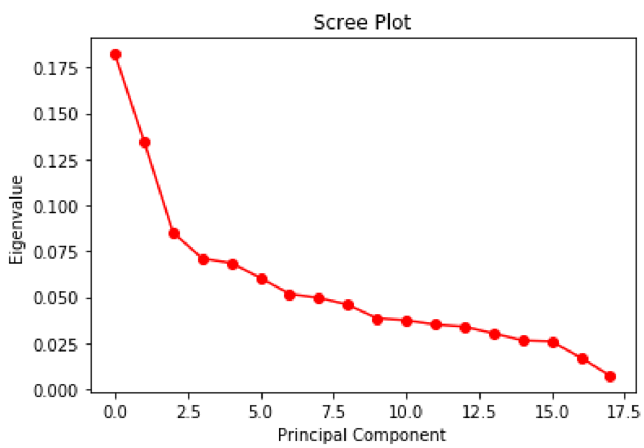
The primary methods of analysis in this report are factor analysis of mixed data, or FAMD, and logistic regression. Multiple correspondence analysis, MCA, was used to explore and examine the cleaned categorical variables. Principal component analysis, PCA, was used to explore and examine the cleaned continuous variables. FAMD was used for dimensionality reduction. Correlation analysis was used throughout the cleaning, exploration, and analysis portions. Finally, as a culmination of all of these methods, logistic regression was used to produce a predictive model with a tested accuracy of 0.8113 and an ROC AUC of 0.8422. The coefficients of the logistic regression model were also examined to come to conclusions about the dataset.

MCA

In [217]:
```
mca_df = df[['SeniorCitizen', 'Partner', 'Dependents', 'MultipleLines',
             'FiberOpticService', 'DSLService', 'OnlineSecurity', 'OnlineBackup
             'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovies
             'OneYearContract', 'TwoYearContract', 'PaperlessBilling',
             'MailedCheckPayment', 'BankTransferPayment', 'CreditCardPayment']]
```

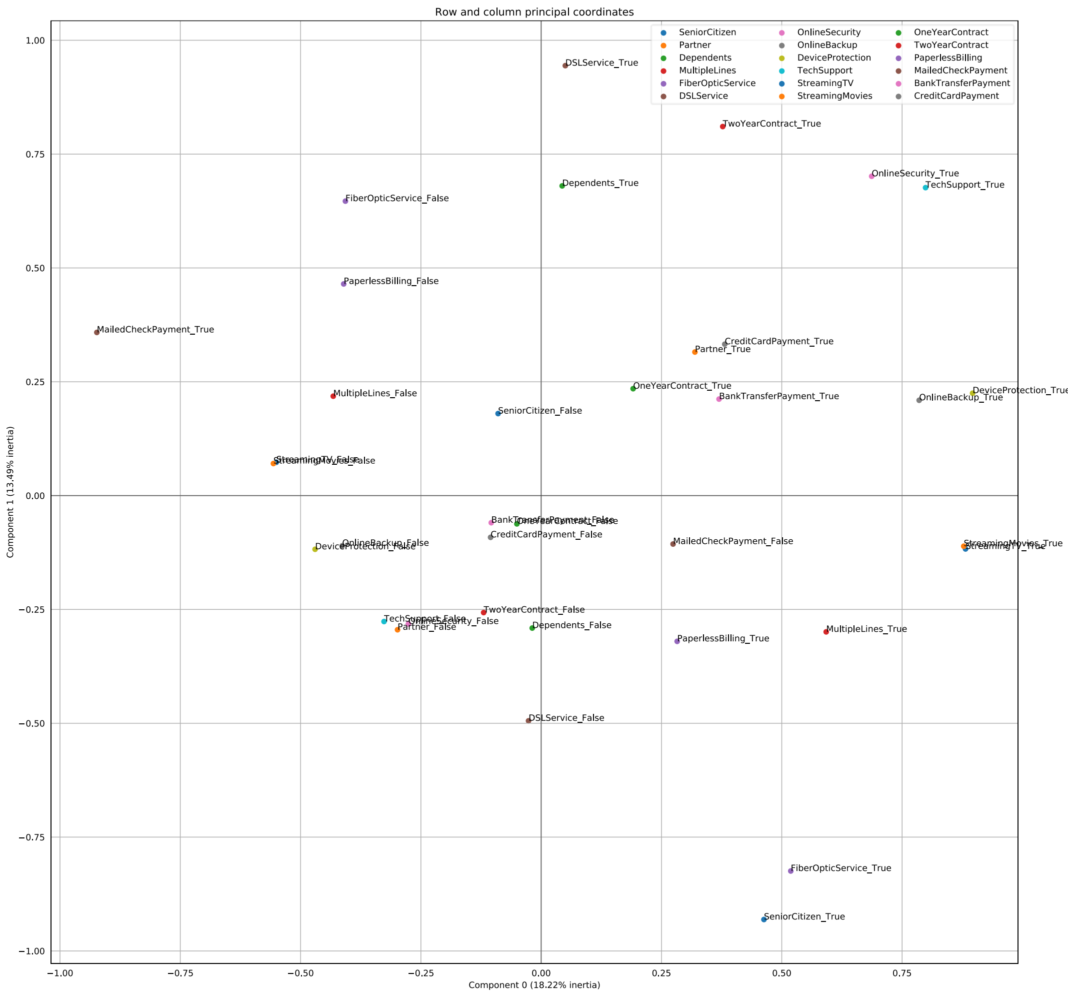
In [218]:
```
mca = prince.MCA(
    n_components = 18,
    n_iter = 3,
    copy = True
)
mca = mca.fit(mca_df)
```

In [221]:
```
plt.plot(np.arange(18), mca_eigenvalues, 'ro-')
plt.title("Scree Plot")
plt.xlabel("Principal Component")
plt.ylabel("Eigenvalue")
plt.show()
```



In [222]:
```
mca = prince.MCA(
    n_components = 3,
    n_iter = 3,
    copy = True
)
mca = mca.fit(mca_df)
```

In [223]:
```
ax = mca.plot_coordinates(
    X = mca_df,
    ax = None,
    figsize=(20, 20),
    show_row_points=False,
    show_row_labels=False,
    show_column_points=True,
    column_points_size=30,
    show_column_labels=True,
    legend_n_cols=3
)
plt.savefig('Charts/MCA.svg')
```



Row and column principal coordinates
SeniorCitizen
Partner
Dependents
MultipleLines
FiberOpticService
DSLService
OnlineSecurity
OnlineBackup
DeviceProtection
TechSupport
StreamingTV
StreamingMovies
OneYearContract
TwoYearContract
PaperlessBilling
MailedCheckPayment
BankTransferPayment
CreditCardPayment
DSLService_True
TwoYearContract_True
OnlineSecurity_True
TechSupport_True
Dependents_True
FiberOpticService_False
PaperlessBilling_False
MailedCheckPayment_True
CreditCardPayment_True
Partner_True
OneYearContract_True
BankTransferPayment_True
DeviceProtection_True
OnlineBackup_True
MultipleLines_False
SeniorCitizen_False
StreamingTV_False
StreamingMovies_False
BankTransferPayment_False
OneYearContract_False
CreditCardPayment_False
MailedCheckPayment_False
OnlineBackup_False
DeviceProtection_False
StreamingMovies_True
StreamingTV_True
TwoYearContract_False
TechSupport_False
OnlineSecurity_False
Partner_False
Dependents_False
PaperlessBilling_True
MultipleLines_True
DSLService_False
FiberOpticService_True
SeniorCitizen_True
Component 1 (13.49% inertia)
Component 0 (18.22% inertia)
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75

PCA

In [224]:
```
pca_df = df[['Tenure', 'MonthlyCharges', 'TotalCharges']]
```

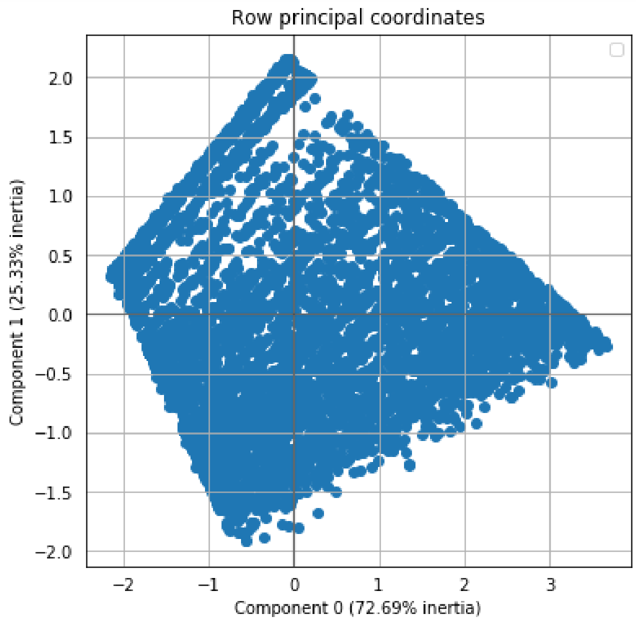
In [225]:
```
pca_df = preprocessing.MinMaxScaler().fit_transform(pca_df)
pca_df
```

Out[225]:
```
array([[0.01388889, 0.11542289, 0.00343704],
       [0.47222222, 0.38507463, 0.21756402],
       [0.02777778, 0.35422886, 0.01245279],
       ...,
       [0.15277778, 0.11293532, 0.03989153],
       [0.05555556, 0.55870647, 0.03530306],
       [0.91666667, 0.86965174, 0.78810105]])
```

In [226]:
```
# Rewrite the normalized columns back to the base dataframe for the upcoming ar
df['Tenure'] = (df['Tenure'] - df['Tenure'].mean()) / df['Tenure'].std()
df['MonthlyCharges'] = (df['MonthlyCharges'] - df['MonthlyCharges'].mean()) / d
df['TotalCharges'] = (df['TotalCharges'] - df['TotalCharges'].mean()) / df['Tot
```

In [227]:
```
pca = prince.PCA(
     n_components=3,
     n_iter=3,
     copy=True,
     check_input=True,
)
pca = pca.fit(pca_df)
```

In [228]:
```
ax = pca.plot_row_coordinates(
    pca_df,
    ax = None,
    figsize = (6, 6),
    x_component = 0,
    y_component = 1
)
```



In [229]:
```
pca.explained_inertia_
```

Out[229]: [0.726888379280512, 0.25325809879467087, 0.019853521924817454]

In [230]:
```
pca.column_correlations(pca_df)
```

Out[230]:

	0	1	2
0	0.837734	0.528903	0.135878
1	0.717447	-0.690778	0.089971
2	0.981905	0.053484	-0.181667

In the process of looking into principal components and variable interactions, it is worth looking at the continuous variable columns and reconsidering what they may represent. Tenure may be better decribed as the amount of bills a customer has had, and if that is the case, then TotalCharges may be nothing more than Tenure * MonthlyCharges.

As seen in the test below, this isn't exactly the case, but the correlation between Tenure and MonthlyCharges, and TotalCharges is very high, with a Pearson correlation coefficient of 0.83 between Tenure and TotalCharges. Given the high correlation between these, TotalCharges will be removed from the dataframe before the logistic regression is performed.

In [231]:
```
df[['Tenure', 'MonthlyCharges', 'TotalCharges']].corr()
```

Out[231]:

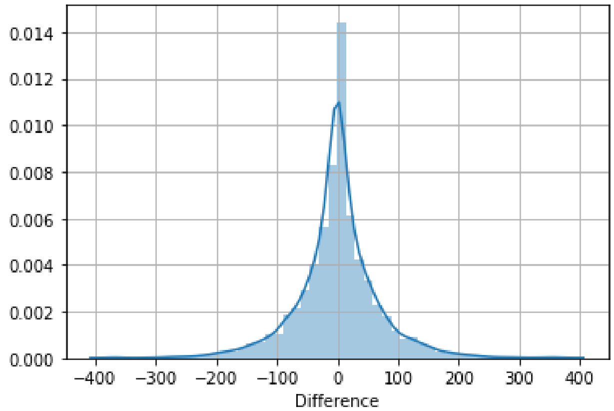
	Tenure	MonthlyCharges	TotalCharges
Tenure	1.000000	0.247900	0.826178
MonthlyCharges	0.247900	1.000000	0.651174
TotalCharges	0.826178	0.651174	1.000000

In [232]:
```
test_df = original_df[['Tenure', 'MonthlyCharges', 'TotalCharges']]
test_df['TenureMonthlyCharges'] = test_df['Tenure'] * test_df['MonthlyCharges']
test_df['Difference'] = test_df['TenureMonthlyCharges'] - test_df['TotalCharges
test_df.describe()
```

Out[232]:

	Tenure	MonthlyCharges	TotalCharges	TenureMonthlyCharges	Difference
count	7043.000000	7043.000000	7043.000000	7043.000000	7043.000000
mean	32.371149	64.761692	2279.734304	2279.581350	-0.152953
std	24.559481	30.090047	2266.794470	2264.729447	67.202778
min	0.000000	18.250000	0.000000	0.000000	-373.250000
25%	9.000000	35.500000	398.550000	394.000000	-28.500000
50%	29.000000	70.350000	1394.550000	1393.600000	0.000000
75%	55.000000	89.850000	3786.600000	3786.100000	28.600000
max	72.000000	118.750000	8684.800000	8550.000000	370.850000

In [233]:
```
sns.distplot(test_df['Difference'])
plt.grid(True)
```



Now the columns identified as interactions or flagged for removal with the MCA, PCA, and correlation tests can be removed.

In [234]:
```
df.drop(['StreamingTV', 'TotalCharges'], axis = 1, inplace = True)
```

FAMD

To continue exploring the data and to find the most important variables for the predictive analysis, FAMD, or Factor Analysis of Mixed Data will be used.

In [235]:
```
famd = prince.FAMD(
    n_components = 20,
    n_iter = 3,
    copy = True
)
famd = famd.fit(df)
```

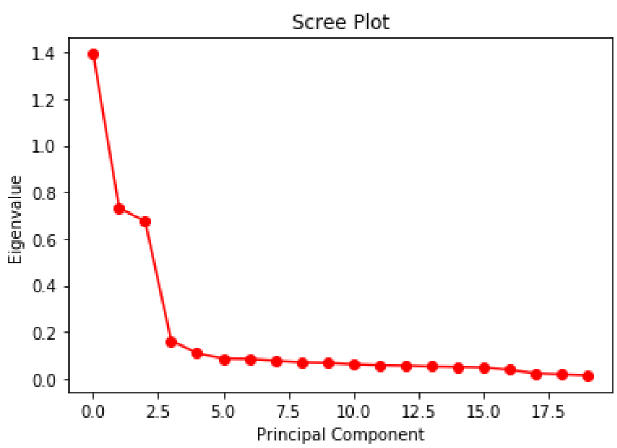
With the factor analysis performed, we can examine the eigenvalues of the dataset and determine what eigenvectors to consider. Judging by the scree plot, the top 4 vectors should be used.

In [236]:
```
eigenvalues = famd.eigenvalues_
eigenvalues
```

Out[236]:
```
[1.3934409203989457,
 0.7335955145699103,
 0.674781156917683 8,
 0.1630003150493670 2,
 0.1091211154955826 6,
 0.0859771329481749 4,
 0.0847133725275702 9,
 0.0768948016782065 7,
 0.0699433521031053 4,
 0.0679031961146765 2,
 0.0618211699750119 5,
 0.057821381747245,
 0.0557418027933077 8,
 0.0525089409302258 2,
 0.0507763852537253 1,
 0.04853565052231533 6,
 0.0382703824929452 6,
 0.0228904534625687 2,
 0.01836689489956057 4,
 0.01417362189952114 6]
```

The scree plot below shows that 4 factors are an adequate amount to use for modeling.

In [237]:
```
plt.plot(np.arange(20), eigenvalues, 'ro-')
plt.title("Scree Plot")
plt.xlabel("Principal Component")
plt.ylabel("Eigenvalue")
plt.show()
```



In [238]:
```
famd = prince.FAMD(
    n_components = 4,
    n_iter = 3,
    copy = True
)
famd = famd.fit(df)
```

In [239]:
```
eigenvalues = famd.eigenvalues_
eigenvalues
```

Out[239]:
```
[1.393440920721763, 0.7335955121345971, 0.674781155124393, 0.162998265176196
2]
```

In [240]:
```
famd.explained_inertia_
```

Out[240]:
```
[0.3591085684198537,
 0.18905748370403094,
 0.1739002285707967,
 0.042006857120303685]
```

However, this only equates to 76.4% of variance of the dataset explained. A 5th factor may be necessary for proper accuracy.

In [241]:
```
sum(famd.explained_inertia_)
```

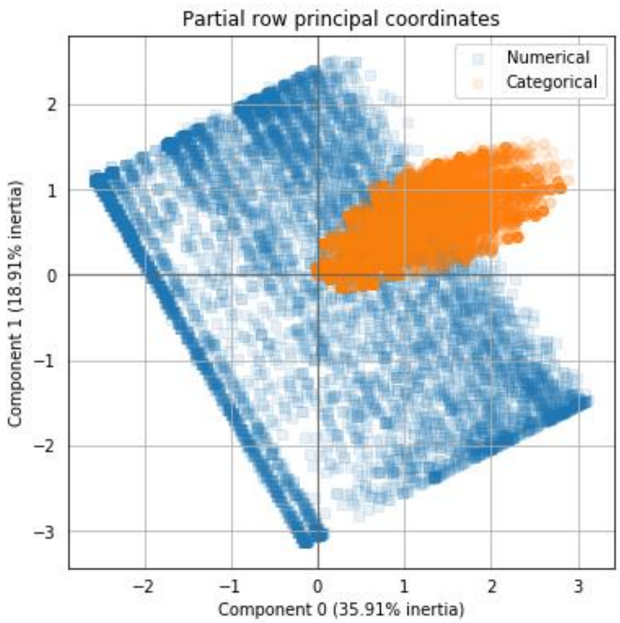
Out[241]:
```
0.7640731378149851
```

In [242]:
```
famd.column_correlations(df)
```

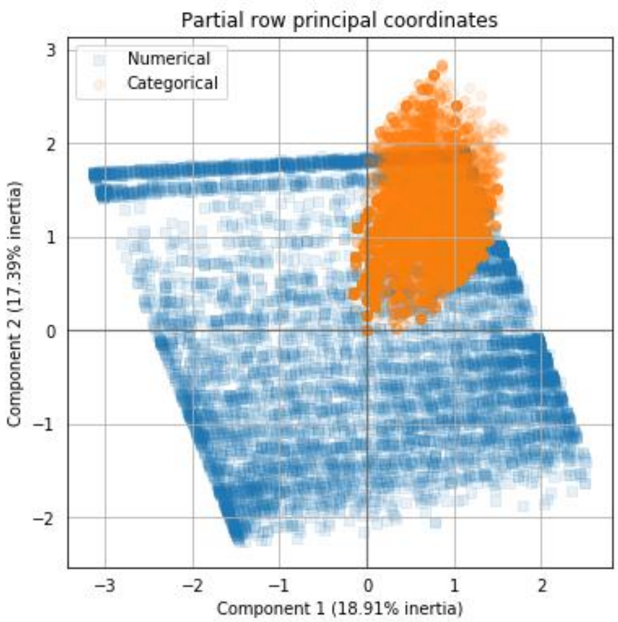
Out[242]:

	0	1	2	3
BankTransferPayment	0.169347	-0.234801	-0.055352	-0.150342
TechSupport	0.419837	-0.212003	-0.349523	0.058267
Partner	0.285479	-0.355039	-0.114051	-0.212548
MailedCheckPayment	-0.394541	0.104815	0.385285	-0.141558
SeniorCitizen	0.161476	0.062453	-0.218131	0.169914
OnlineSecurity	0.392665	-0.230035	-0.308667	0.019311
TwoYearContract	0.268859	-0.600647	0.043416	-0.491380
FiberOpticService	0.553599	0.262111	-0.777801	0.645556
StreamingMovies	0.597270	-0.068711	-0.633186	0.323681
PaperlessBilling	0.247266	0.121161	-0.346142	0.291160
Dependents	0.013797	-0.204942	0.105630	-0.220951

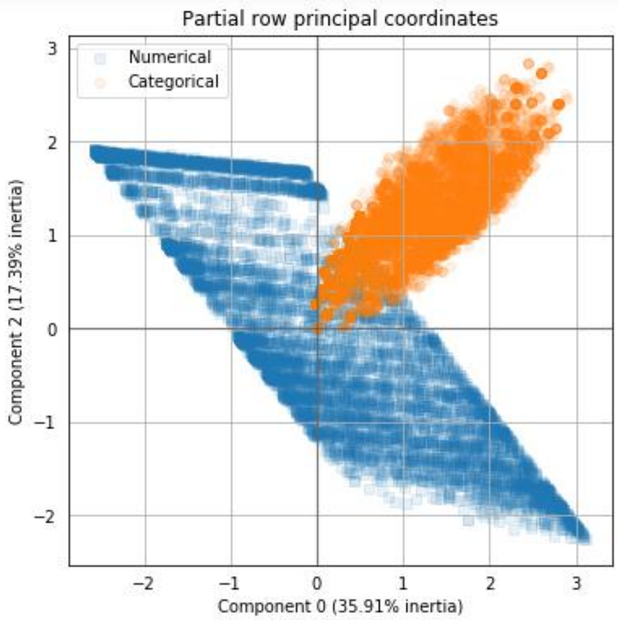
In [243]:
```
plot = famd.plot_partial_row_coordinates(
     df,
     ax=None,
     figsize=(6, 6),
     x_component=0,
     y_component=1,
     alpha = 0.1
)
```



In [244]:
```
plot2 = famd.plot_partial_row_coordinates(
      df,
      ax=None,
      figsize=(6, 6),
      x_component=1,
      y_component=2,
      alpha = 0.1
)
```



In [245]:
```
plot3 = famd.plot_partial_row_coordinates(
      df,
      ax=None,
      figsize=(6, 6),
      x_component=0,
      y_component=2,
      alpha = 0.1
)
```



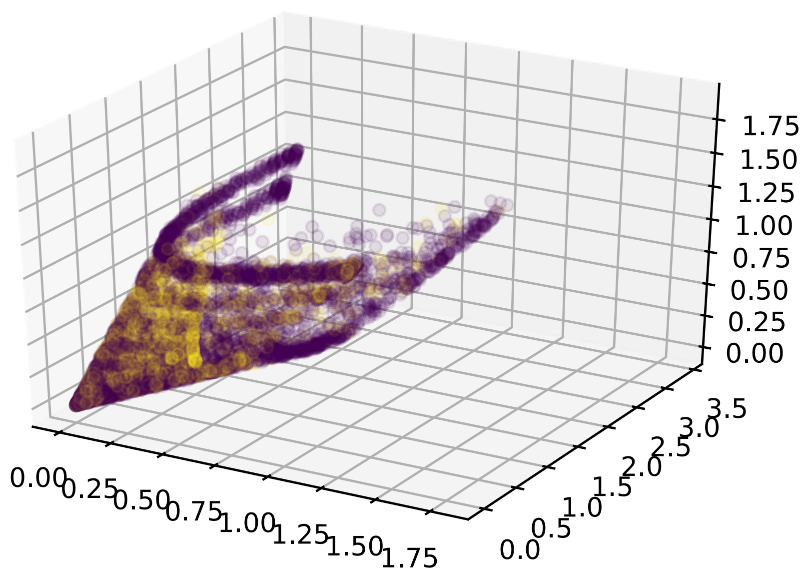
In [246]:
```
three_dimension_fa = famd.row_contributions(df)
three_dimension_fa
```

Out[246]:

	0	1	2	3
0	0.882797	0.559957	0.811650	0.000063
1	0.008367	0.019372	0.034541	0.003680
2	0.347024	0.914568	0.102901	0.077133
3	0.018341	0.447505	0.273310	0.120836
4	0.126795	1.262457	0.010533	0.167698

In [249]:
```
three_dimension_fa_churn = three_dimension_fa.copy(deep = True)
three_dimension_fa_churn['Churn'] = df['Churn']
```

In [250]:
```
fig = plt.figure()
ax = fig.add_subplot(111, projection = '3d')
x = three_dimension_fa_churn[0]
y = three_dimension_fa_churn[1]
z = three_dimension_fa_churn[2]
color = three_dimension_fa_churn['Churn']
ax.scatter(x, y, z, c = color, alpha = 0.1)

pickle.dump(fig, open('3D FAMD Churn.fig.pickle', 'wb'))
fig.savefig("Charts/FAMD Churn.svg")
```



Logistic Regression

Two models will be created using Logistic Regression here, once with the basic cleaned dataframe, and another using the Factor Analysis of Mixed Data results.

In [251]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, precision_score, classification_r
```

Base Dataframe Model

In [252]:
```
x_df = df.drop('Churn', axis = 1)
```

In [253]:
```
# Random_state is set to allow exact reproducibility.
x_train, x_test, y_train, y_test = train_test_split(x_df, df['Churn'],
    test_size = 0.1, random_state = 1)
```

In [254]:
```
regression = LogisticRegression()
regression.fit(x_train, y_train)
```

Out[254]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [255]:
```
regression.intercept_
```

Out[255]:
```
array([-2.14121805])
```

In [256]:
```
regression.coef_
```

Out[256]:
```
array([[ 0.22680933, -0.00440932, -0.11375933, -0.79849974,  0.20898317,
         1.683634  ,  0.88671329, -0.40286786, -0.09805871,  0.03363374,
        -0.30891087,  0.37943566, -0.66759031, -1.36620076,  0.3358983 ,
        -0.35053947, -0.34015657, -0.39539662,  0.01770752]])
```

In [257]:
```python
coefs = pd.concat([pd.DataFrame(df.columns.drop('Churn')),
                   pd.DataFrame(np.transpose(regression.coef_))], axis = 1)

coefs.columns = ["Column", "Coefficient"]
coefs
```

Out[257]:

	Column	Coefficient
0	SeniorCitizen	0.226809
1	Partner	-0.004409
2	Dependents	-0.113759
3	Tenure	-0.798500
4	MultipleLines	0.208983
5	FiberOpticService	1.683634
6	DSLService	0.886713
7	OnlineSecurity	-0.402868
8	OnlineBackup	-0.098059
9	DeviceProtection	0.033634
10	TechSupport	-0.308911
11	StreamingMovies	0.379436
12	OneYearContract	-0.667590
13	TwoYearContract	-1.366201
14	PaperlessBilling	0.335898
15	MailedCheckPayment	-0.350539
16	BankTransferPayment	-0.340157
17	CreditCardPayment	-0.395397
18	MonthlyCharges	0.017708

In testing the model, we get an accuracy of 81.13%. This is 31.13% better than chance on average.

In [258]:
```
accuracy = regression.score(x_test, y_test)
accuracy
```

Out[258]:
```
0.8113475177304964
```

In [259]:
```
predictions = regression.predict(x_test)
actual = y_test
```

In [260]:
```
confusion = confusion_matrix(actual, predictions)
confusion
```

Out[260]:
```
array([[479,  54],
       [ 79,  93]], dtype=int64)
```

In [261]:
```
precision = precision_score(actual, predictions)
precision
```

Out[261]:
```
0.6326530612244898
```

Though the accuracy is high, the precision and recall on True Churns could use some improvement and indicate that the model could be adjusted for better results. Having a larger proportion of customers that churned in the training dataset would help a lot and would be easier to get the data for in this situation than many others.

In [262]:
```
print(classification_report(actual, predictions))
```

```
              precision    recall  f1-score   support

       False       0.86      0.90      0.88       533
        True       0.63      0.54      0.58       172

    accuracy                           0.81       705
   macro avg       0.75      0.72      0.73       705
weighted avg       0.80      0.81      0.81       705
```

In [263]:
```
probabilities = regression.predict_proba(x_test)
predictions = probabilities[:,1]
```

In [264]:
```
false_positive_rate, true_positive_rate, threshold =\
    roc_curve(y_test, predictions)
```

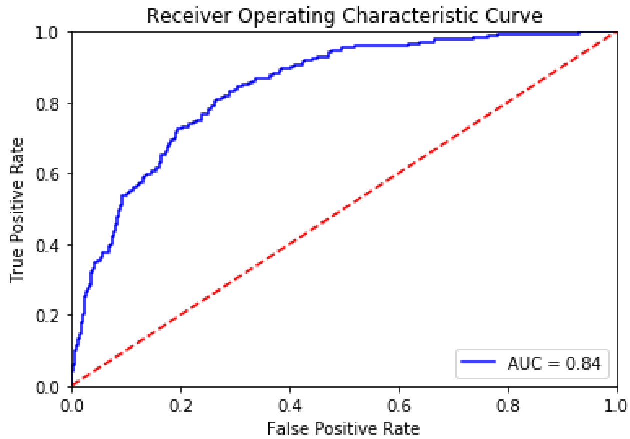
Finally, by calculating the ROC and AUC, we get an AUC of 0.84, which indicates an excellent model relative to chance.

In [265]:
```
roc_auc = auc(false_positive_rate, true_positive_rate)
```

In [266]:
```
roc_auc
```

Out[266]:
```
0.8421942493127973
```

In [267]:
```
plt.title('Receiver Operating Characteristic Curve')
plt.plot(false_positive_rate, true_positive_rate, 'blue',
         label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.savefig('Charts/ROC AUC.svg')
plt.show()
```



FAMD Based Logistic Regression Model

In [268]:
```
famd_df = df.copy(deep = True)
```

In [269]:
```
# Random_state is set to allow exact reproducibility.
famd_x_train, famd_x_test, famd_y_train, famd_y_test = train_test_split(famd_df
    test_size = 0.1, random_state = 1)
```

In [270]:
```
famd = prince.FAMD(
    n_components = 5,
    n_iter = 3,
    copy = True
)
famd = famd.fit(famd_x_train)
```

In [271]:
```
famd.column_correlations(famd_df)
```

Out[271]:

	0	1	2	3	4
OneYearContract	0.121108	0.184091	-0.114338	-0.125124	-0.133100
OnlineSecurity	0.393211	0.103995	-0.389110	0.079937	0.055341
BankTransferPayment	0.170907	0.196701	-0.164094	-0.121275	-0.125411
MailedCheckPayment	-0.393740	0.037719	0.397323	-0.194282	-0.186250
PaperlessBilling	0.244976	-0.232041	-0.251903	0.316004	0.309229
SeniorCitizen	0.160099	-0.133385	-0.165643	0.186766	0.185383
FiberOpticService	0.548373	-0.511973	-0.571199	0.701783	0.712974
Partner	0.287750	0.286865	-0.275241	-0.166686	-0.152204
CreditCardPayment	0.156499	0.195271	-0.149484	-0.122166	-0.130665
Dependents	0.015837	0.225284	-0.004727	-0.213484	-0.195332
DeviceProtection	0.539898	0.009307	-0.539978	0.223947	0.202365

In [272]:
```
famd_x = famd.transform(famd_x_train)
famd_x
```

1144	-0.658755	-0.483233	1.359564	0.248222	-0.432911
4867	1.347853	-0.716199	0.004795	-0.255215	0.387264
4793	1.789362	-0.263440	0.183534	-0.037310	-0.005700
5304	0.151471	-0.704028	0.577791	-0.410748	0.269204
6192	0.395527	-0.824545	0.495055	-0.282386	-0.382560
...	...	...	...	...	...
905	0.977050	-1.574818	0.435212	-0.257343	-0.269082
5192	-0.139433	1.653680	0.645255	-0.357436	0.093522
3980	1.473054	-1.110007	0.138667	-0.117483	-0.244078
235	-0.216571	-0.722301	1.319013	0.197074	-0.377867
5157	0.660279	-0.213306	1.209996	0.749978	0.699765

6338 rows × 5 columns

This regression will be fitted with the FAMD transformed version of the training split data instead of the base split.

In [273]:
```
famd_regression = LogisticRegression()
famd_regression.fit(famd_x, famd_y_train)
```

Out[273]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [274]:
```
famd_regression.coef_
```

Out[274]:
```
array([[-0.14499965, -1.7984952 ,  0.05868958, -0.68208019, -0.50552033]])
```

In [275]:
```
famd_accuracy = famd_regression.score(famd.transform(famd_x_test), famd_y_test)
famd_accuracy
```

Out[275]:
```
0.8
```

In [276]:
```
famd_predictions = regression.predict(famd_x_test)
famd_actual = famd_y_test
```

In [277]:
```
famd_confusion = confusion_matrix(famd_actual, famd_predictions)
famd_confusion
```

Out[277]:
```
array([[479,  54],
       [ 79,  93]], dtype=int64)
```

In [278]:
```
famd_precision = precision_score(famd_actual, famd_predictions)
famd_precision
```

Out[278]:
```
0.6326530612244898
```

Notice that the results are exactly the same as the model trained on the base dataset.

In [279]:
```
print(classification_report(famd_actual, famd_predictions))
```

```
              precision    recall  f1-score   support

       False       0.86      0.90      0.88       533
        True       0.63      0.54      0.58       172

    accuracy                           0.81       705
   macro avg       0.75      0.72      0.73       705
weighted avg       0.80      0.81      0.81       705
```

In [280]:
```
famd_probabilities = regression.predict_proba(famd_x_test)
famd_predictions = famd_probabilities[:,1]
```

In [281]:
```
famd_false_positive_rate, famd_true_positive_rate, famd_threshold =\
    roc_curve(famd_y_test, famd_predictions)
```

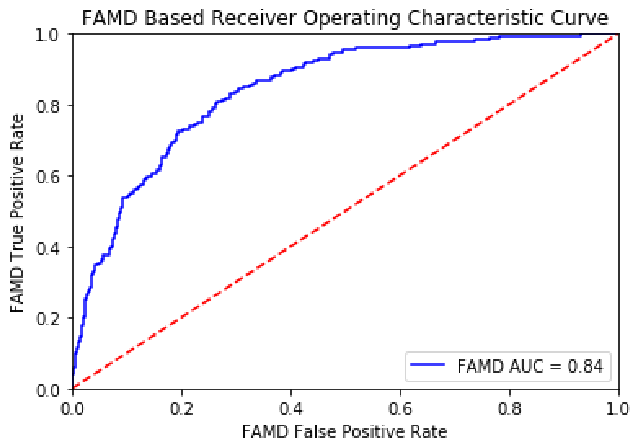
Finally, by calculating the ROC and AUC, we get an AUC of 0.84, exactly the same as before.

In [282]:
```
famd_roc_auc = auc(famd_false_positive_rate, famd_true_positive_rate)
```

In [283]:
```
famd_roc_auc
```

Out[283]:
```
0.8421942493127973
```

In [284]:
```
plt.title('FAMD Based Receiver Operating Characteristic Curve')
plt.plot(famd_false_positive_rate, famd_true_positive_rate, 'blue',
         label = 'FAMD AUC = %0.2f' % famd_roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('FAMD True Positive Rate')
plt.xlabel('FAMD False Positive Rate')
plt.savefig('Charts/FAMD ROC AUC.svg')
plt.show()
```



There are many data mining methods to explore and analyze data, but they all have various requirements of the data in order to successfully and meaningful do. For this dataset. FAMD was chosen as a descriptive method. This dataset and the scenario around it required the dependent variable, the Churn column, to be Boolean in nature. This means that predictive methods are going to naturally focus on classification. Additionally, the independent variables are 16 nominal categorical variables and 3 continuous quantitative variables. After cleaning, 13 categorical and 2 quantitative independent variables remained for data mining. While there are 2 continuous variables, clustering techniques would not work well on this dataset because of the number of significant categorical variables. Association analysis may have been used, but the ultimate goal was to specifically examine the interactions between the numerous categorical variables to find trends in customer churn and not just what tends to be grouped together.

This naturally led to using MCA to examine the similarities and differences that the various variables had on the variance of the data. Plotting the results of the MCA gives a chart that showed how close the variables were in relation to each other on the primary 2 factors of the dataset. From there, it was a simple jump to FAMD in order to also include the effects of the 2 continuous variables and then use the results as a form of dimensionality reduction for the predictive modeling that followed.

For a predictive method, logistic regression was chosen for a few reasons. One of the main reasons was the ease and simplicity of creating, interpreting, and integrating the model into future projects. The coefficients that it creates are a simple way to compare the effects of the various parameters at play in a classification, and it’s very easy to calculate the classification with the model, even by hand. The number of categorical variables are simple to integrate into the predictive model through dummy variables, but still allow for the easy integration of the continuous variables into the equation. Finally, the classification it gives for any customer put into it are based on the probability that a customer will churn, which is more applicable to the given context and scenario that a clean classification like LDA gives while not requiring as many assumptions about the dataset. Decision trees may have been another good choice, but this was ultimately a predictive classification task and not just a prediction task. Logistic regression results in a model that is far easier to examine the relationship between the variables with whereas a decision tree is much harder to form conclusions with.

Different types of information require different types of visualization. The distribution of continuous variables was performed using histograms and box plots were used to visualize their IQRs and to look for outliers as these two charts are exclusively for this type of data. For the distribution of the categorical variables, bar charts were used. Humans are significantly better at judging length than area, and bar charts are better for showing proportion than pie charts because of this.

For the bivariate relations, the categorical variables were visualized with simple contingency charts. The relationship between these independent variables and their binary dependent variable are the simplest of bivariate relations and the information can be readily absorbed by the reader in the spatially organized text format that a contingency chart is displayed using. Bar, pie, and mosaic plots were all considered, but all seemed to overcomplicate the presentation. Furthermore, two contingency charts were created for each relationship, with the second displaying the normalized proportion percentages instead of the raw numbers like the first for ease of lookup. For the non-binary categorical variables, bar charts were used to plot the frequency of each possibility, grouped by categorical value for simple comparison.

The continuous variable bivariate relations are visualized through Seaborn's distplot function, which overlays a histogram of the data with a kernel density estimate. The dataframe is divided by the Churn column, and for each continuous column, both dataframes' distributions via histograms are superimposed for direct comparison. The histograms are easy to compare by relative area and the plotted KDEs make it easy to follow the trend across the x-axis. Finally, outside of the distributions of the bivariate relations, a heatmap was used to visualize the correlation coefficient matrix of the dataset. Color-coding the coefficients make it extremely easy to pick out the extremes even when a total of 484 coefficients need to be considered.

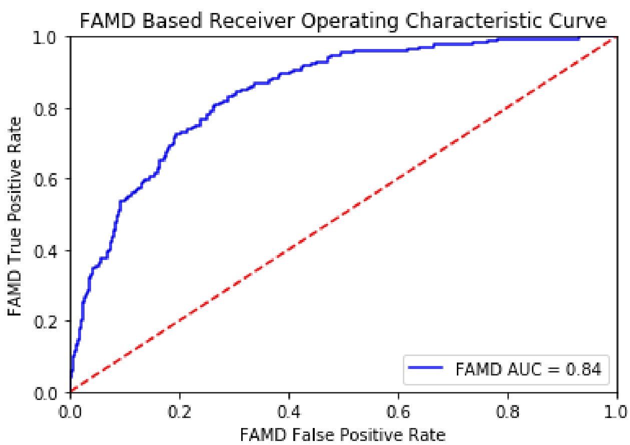
Visualizing the data mining methods is a more complicated process, but this leaves less to be considered when selecting methods. As MCA, PCA, and FAMD are all forms of factor analysis, scree plots were the most natural choice for examining the inertia of the calculated eigenvectors for the dataset and were used to visually conceptualize this information. The columns and rows were visualized by their contributions to the factor calculated by these methods, and this was particularly useful for examining the MCA results in order to see the relative similarities and differences in variance contribution that each column gives.

Finally, ROCs were used to visualize the effectiveness of the created predictive models. These are among the simplest to look at and understand visually, and they pair naturally with the AUC for a numeric calculation of effectiveness. ROC AUC is also an expected pairing with logistic regression.

IV: Data Summary

Aside from the results of the correlation analysis throughout the report, the most decisive proof of discrimination in this analysis is the results of the ROC AUC and the measures of precision, recall, and accuracy that the predictive model built from these discrimination rules achieves. If the analysis was not discriminating, the model would not be significantly better than chance at predicting the dependent variable than chance.

In [284]:
```
plt.title('FAMD Based Receiver Operating Characteristic Curve')
plt.plot(famd_false_positive_rate, famd_true_positive_rate, 'blue',
         label = 'FAMD AUC = %0.2f' % famd_roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('FAMD True Positive Rate')
plt.xlabel('FAMD False Positive Rate')
plt.savefig('Charts/FAMD ROC AUC.svg')
plt.show()
```



As we can see here, the model (the blue curve) performs 34% better than chance (the diagonal red line). The classifications that produce this are calculated with the regression coefficients below.

In [257]:
```python
coefs = pd.concat([pd.DataFrame(df.columns.drop('Churn')),
                   pd.DataFrame(np.transpose(regression.coef_))], axis = 1)

coefs.columns = ["Column", "Coefficient"]
coefs
```

Out[257]:

	Column	Coefficient
0	SeniorCitizen	0.226809
1	Partner	-0.004409
2	Dependents	-0.113759
3	Tenure	-0.798500
4	MultipleLines	0.208983
5	FiberOpticService	1.683634
6	DSLService	0.886713
7	OnlineSecurity	-0.402868
8	OnlineBackup	-0.098059
9	DeviceProtection	0.033634
10	TechSupport	-0.308911
11	StreamingMovies	0.379436
12	OneYearContract	-0.667590
13	TwoYearContract	-1.366201
14	PaperlessBilling	0.335898
15	MailedCheckPayment	-0.350539
16	BankTransferPayment	-0.340157
17	CreditCardPayment	-0.395397
18	MonthlyCharges	0.017708

It's this list of coefficients that show the detection of the phenomena that needed to be found and prompted this analysis. Here we see what factors are most associated with customer churn and from there we can infer what is most likely causing customers to churn. As a 1 indicates a positive customer churn, the positive coefficients are the ones that push the probability closer to churn, and the negative coefficients push the probability closer to not churning.

The biggest issue is with customers with FiberOpticService. It is over twice as influential on the probability of churn than the second highest, DSLService. The combination of these two show that the Internet services of this telecommunications business are the areas that need the most attention. Beyond that, customers that pay for Streaming Movies, and TV based on correlation that caused an interaction requiring the second to be removed from the model, are at the next highest risk of churn. Finally, customers enrolled in paperless billing, senior citizens, and customers with multiple phone lines are also groups to pay attention to.

The first method used for detecting interactions was examining the contingency tables with chi-squared tests to determine whether or not their values are independent or not.

The χ^2 test does not show significance, so this column will not be examined in more detail later.

In [154]:
```
scs.chi2_contingency(gender_contingency)
```
Out[154]:
```
(0.4840828822091383,
 0.48657873605618596,
 1,
 array([[2611.61010933,  943.38989067],
        [2562.38989067,  925.61010933]]))
```

The χ^2 test shows significance, so this column should be examined in more detail later.

In [157]:
```
scs.chi2_contingency(senior_contingency)
```
Out[157]:
```
(159.42630036838742,
 1.510066805092378e-36,
 1,
 array([[4335.05239245, 1565.94760755],
        [ 838.94760755,  303.05239245]]))
```

The χ^2 test shows significance, so this column should be examined in more detail later.

In [160]:
```
scs.chi2_contingency(partner_contingency)
```
Out[160]:
```
(158.7333820309922,
 2.1399113440759935e-36,
 1,
 array([[2674.78830044,  966.21169956],
        [2499.21169956,  902.78830044]]))
```

The χ^2 test shows significance, so this column should be examined in more detail later.

In [163]:
```
scs.chi2_contingency(dependents_contingency)
```
Out[163]:
```
(189.12924940423474,
 4.9249216612154196e-43,
 1,
 array([[3623.93042737, 1309.06957263],
        [1550.06957263,  559.93042737]]))
```

The χ^2 test does not show significance, so this column will not be examined in more detail later.

In [166]:
```
scs.chi2_contingency(phone_contingency)
```
Out[166]:
```
(0.9150329892546948,
 0.3387825358066928,
 1,
 array([[ 501.01774812,  180.98225188],
        [4672.98225188, 1688.01774812]]))
```

The χ^2 test shows significance, so this column should be examined in more detail later. It may be that a greater number of lines increases the probability of churn as a result of the higher cost overall.

In [169]:
```
scs.chi2_contingency(lines_contingency)
```

Out[169]:
```
(11.143251001456251,
 0.0008433795342472428,
 1,
 array([[2991.41388613, 1080.58611387],
        [2182.58611387,  788.41388613]]))
```

The χ^2 test shows significance, so this column should be examined in more detail later. Customers who had been purchasing fiber optic services in this data set have a substantial difference in churn rate and should be examined in much greater detail.

In [172]:
```
scs.chi2_contingency(fiber_contingency)
```

Out[172]:
```
(666.8080208747958,
 4.940476033744708e-147,
 1,
 array([[2899.58511998, 1047.41488002],
        [2274.41488002,  821.58511998]]))
```

The χ^2 test shows significance, so this column should be examined in more detail later. Customers who had not been purchasing DSL services in this data set have a substantial difference in churn rate and should be examined in much greater detail.

In [175]:
```
scs.chi2_contingency(dsl_contingency)
```

Out[175]:
```
(108.07545746651793,
 2.587376621108007e-25,
 1,
 array([[3395.46045719, 1226.53954281],
        [1778.53954281,  642.46045719]]))
```

The χ^2 test shows significance, so this column should be examined in more detail later. Customers who had been purchasing internet service but not the online security package in this data set have a substantial difference in churn rate and should be examined in much greater detail.

In [178]:
```
scs.chi2_contingency(security_contingency)
```

Out[178]:
```
(205.63310416062058,
 1.2320984831180024e-46,
 1,
 array([[3690.78176913, 1333.21823087],
        [1483.21823087,  535.78176913]]))
```

The χ^2 test shows significance, so this column should be examined in more detail later.

In [181]:
```
scs.chi2_contingency(backup_contingency)
```

Out[181]:
```
(47.260854003612764,
 6.214092807254819e-12,
 1,
 array([[3389.58341616, 1224.41658384],
        [1784.41658384,  644.58341616]]))
```

The χ^2 test shows significance, so this column should be examined in more detail later.

In [184]:
```
scs.chi2_contingency(dev_protection_contingency)
```

Out[184]:
```
(30.513394539261306,
 3.315693222362861e-08,
 1,
 array([[3394.72582706, 1226.27417294],
        [1779.27417294,  642.72582706]]))
```

The χ^2 test shows significance, so this column should be examined in more detail later. It appears that customers who has a tech support package had a significantly lower churn rate.

In [187]:
```
scs.chi2_contingency(support_contingency)
```

Out[187]:
```
(190.16684201526067,
 2.9235674453140758e-43,
 1,
 array([[3672.4160159, 1326.5839841],
        [1501.5839841,  542.4160159]]))
```

The χ^2 test shows significance, so this column should be examined in more detail later.

In [190]:
```
scs.chi2_contingency(tv_contingency)
```

Out[190]:
```
(27.862522274233417,
 1.3024835736732686e-07,
 1,
 array([[3185.35624024, 1150.64375976],
        [1988.64375976,  718.35624024]]))
```

The χ^2 test shows significance, so this column should be examined in more detail later.

In [193]:
```
scs.chi2_contingency(movies_contingency)
```

Out[193]:
```
(26.25133601003847,
 2.9974738476267514e-07,
 1,
 array([[3166.99048701, 1144.00951299],
        [2007.00951299,  724.99048701]]))
```

In [197]:
```
scs.chi2_contingency(contract_contingency)
```

Out[197]:
```
(1184.5965720837926,
 5.863038300673391e-258,
 2,
 array([[2846.69175067, 1028.30824933],
        [1082.11018032,  390.88981968],
        [1245.198069  ,  449.801931  ]]))
```

The χ^2 test shows significance, so this column should be examined in more detail later. Customers who had been enrolled in paperless billing have a substantial difference in churn rate and should be examined in more detail later.

In [200]:
```
scs.chi2_contingency(paperless_contingency)
```

Out[200]:
```
(258.27764906707307,
 4.073354668665985e-58,
 1,
 array([[2109.85773108,  762.14226892],
        [3064.14226892, 1106.85773108]]))
```

The χ^2 test shows significance, so this column should be examined in more detail later. Customers who had been paying via electronic check have a substantial difference in churn rate and should be examined in much greater detail later.

In [204]:
```
scs.chi2_contingency(paperless_contingency)
```

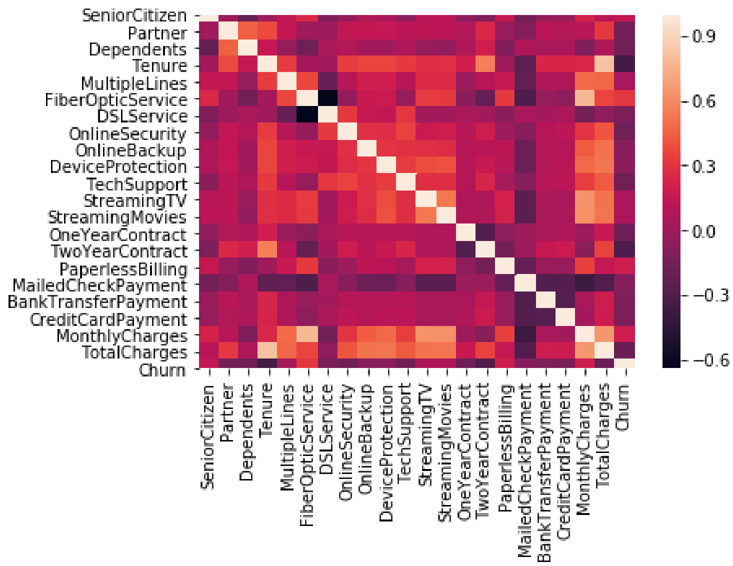
Out[204]:
```
(258.27764906707307,
 4.073354668665985e-58,
 1,
 array([[2109.85773108,  762.14226892],
        [3064.14226892, 1106.85773108]]))
```

This process was followed by examining correlation coefficients with a heatmap for any results with a strong enough linear relationship to skew the model like an interaction would. This was the first direct hint at the interaction between Tenure and TotalCharges.

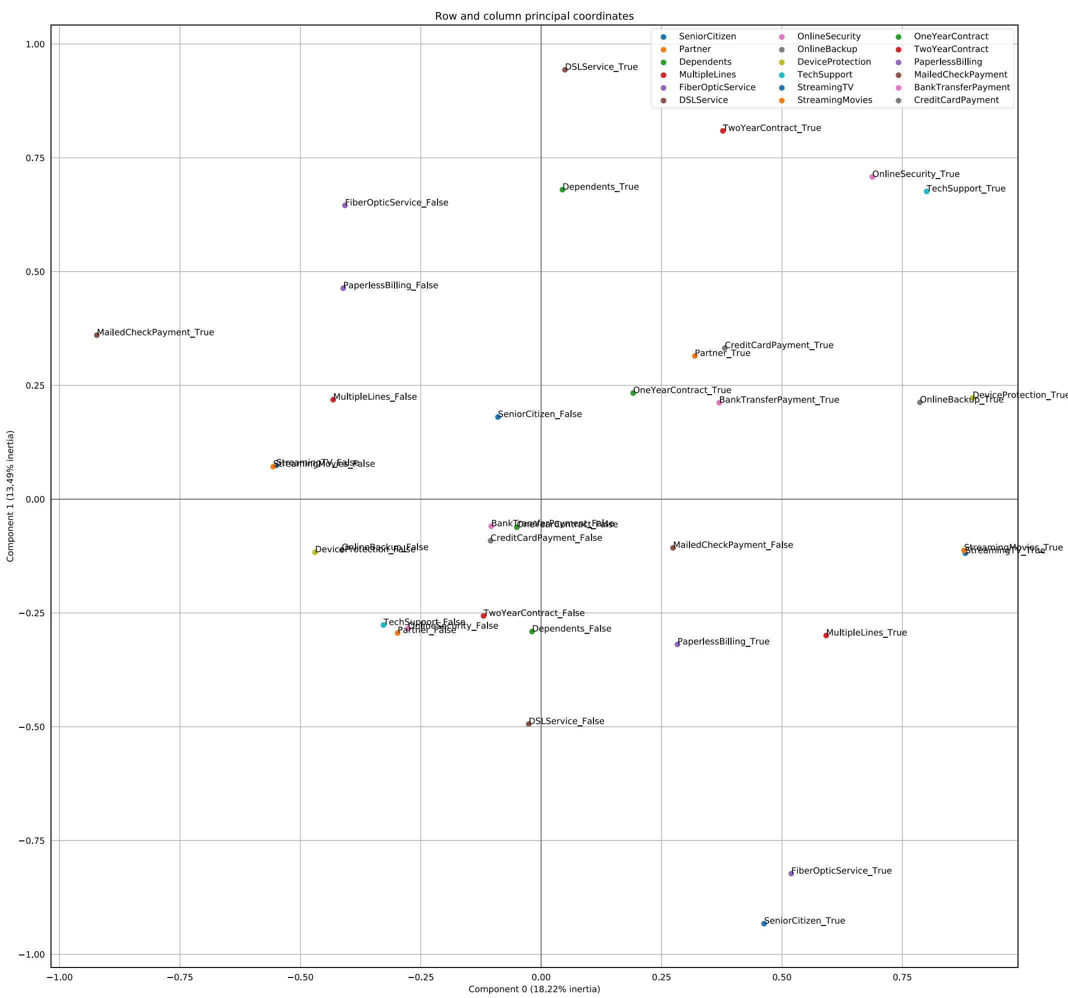
Correlation

In [215]:
```
sns.heatmap(
    df.corr(),
    xticklabels = df.columns,
    yticklabels = df.columns
)
```

Out[215]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x264941859e8>
```



This heatmap was followed by the MCA analysis that revealed a pair of columns, StreamingMovies and StreamingTV, that actually overlapped each other on the first 2 principle factors.



Finally, PCA led to a correlation test between all of the continuous variables that showed an interaction between Tenure and TotalCharges that was reducing the accuracy of the logistic model significantly.

The most important predictor variables were detected by these same processes, but also by examining the logistic model's parameter coefficients. The lowest p-value in the chi-squared tests, the most extreme colors in the correlation coefficient heatmap, and the most extreme values in the model coefficients all showed the strongest predictor variables. These show that the strongest positive predictor variables are having Internet service, particularly Fiber optic service, paying for movie or TV streaming, and being enrolled in paperless billing. Additionally, these also show the strongest negative predictors of churn are having a non-month-to-month contract, particularly a two year contract, having a higher tenure value, paying for an online security package, and having any payment method other than electronic check.

References

Wickham, H., & Grolemund, G. (2017). R for Data Science. Retrieved from https://r4ds.had.co.nz/