1/7/2020

Data Mining Report

Describing and Predicting Customer Churn using Python

Nanako Ohashi

Contents

I: Tool Selection

I chose Python as my tool for data analysis.

Tool Selection

```
In [1]: # import all packages and set plots to be embedded inline
        import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sns
        %matplotlib inline
        from sklearn.model_selection import train_test_split
        from sklearn.ensemble import RandomForestClassifier
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\ensemble\weight_boosting.py:29: DeprecationWarning: numpy.core.umath_tests i
s an internal NumPy module and should not be imported. It will be removed in a future NumPy release.
  from numpy.core.umath_tests import inner1d
```

```
In [2]: # load Customer Data dataset into a pandas dataframe
        df = pd.read_csv('WA_Fn-UseC_-Telco-Customer-Churn.csv')
```

Data Wrangling

General Properties

```
In [3]: # visually assess the data set.
        df.head()
```

Out[3]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	...	DeviceProtection	TechSup
0	7590-VHVEG	Female	0	Yes	No	1	No	No phone service	DSL	No	...	No	
1	5575-GNVDE	Male	0	No	No	34	Yes	No	DSL	Yes	...	Yes	
2	3668-QPYBK	Male	0	No	No	2	Yes	No	DSL	Yes	...	No	
3	7795-CFOCW	Male	0	No	No	45	No	No phone service	DSL	Yes	...	Yes	
4	9237-HQITU	Female	0	No	No	2	Yes	No	Fiber optic	No	...	No	

Benefits of Python for Data Analysis

1. **Flexibility:** Python is very flexible (Terra, 2019);
2. **Open Source:** Unlike SAS, Python is free to use (Jain, 2017);
3. **Well-Supported** (Terra, 2019);
4. **Great for research and prototyping** as well as building the production systems (McKinney, 2018);
5. **Great reproducibility:** Python code can be saved as scripts and can be reused repeatedly with better version control (Fong, 2019). Python is superior to R in this aspect;

6. **Great accessibility:** Python makes accessibility easier than R. If the results of my analysis will be used in an application or website, Python is superior (Guru99, n.d.);
7. **Efficient and scalable:** Python handles big data efficiently (Fong, 2019);
8. **Great for predictive models** (Fong, 2019);
9. Python codes are **easier to maintain and are more robust** than R (Guru99, n.d.);
10. **Python works faster**: Python works faster than R (Guru99, n.d.);
11. **Easy to share:** It is easy to share Python code with colleagues using Jupyter Notebook (Guru99, n.d.);
12. **Code Readability:** Python code is generally easier to read than R code (Guru99, n.d.);
13. **Graphical Capabilities:** Python has superior graphical capabilities than SAS (Jain, 2017);
14. **Deep Learning Support:** Deep learning in SAS is still in its infancy. R has some packages for deep learning, whereas Python has made great advancement in this field with packages such as Tensorflow and Keras (Jain, 2017).

Objectives of the data analysis:

- Extracts information that is not easily deducible from raw data;
- Converts and process raw data to make visualizations and predictions;
- Analyzed data is then used to understand the mechanisms of the systems that produced the data, which could then be used for forecasting the evolution of the systems over time (Nelli, 2015). By this step, we will have discovered the variables that correlate the most to churn and will forecast churn for the business;
- Using this analysis, the company can make decisions as to where they should spend their resources and time to slow down the churn rate.

Descriptive Method

Principal Component Analysis (PCA)

I have selected PCA to choose the most important variables influencing the churn rate. It is difficult to visualize a high dimensional dataset. PCA is a great tool to reduce the dimensions of the dataset by finding the two principal components and, as a result, visualize the data in two-dimensional space as a single scatter plot (Choudhary, 2019). PCA will reduce the dataset to fewer dimensions while maintaining as much distance between the variables as possible (Tufféry, 2011).

Non-Descriptive Method

Logistic Regression

I have selected logistic regression as my non-descriptive method as it is a method for predicting binary classes. Churn is a categorical variable with two options (churn or not churn). It is easy to implement and is used to estimate the relationship between a dependent binary variable and independent variables (Navlani, 2019).

II: Data Exploration and Preparation

Target Variable

The target variable in this dataset is churn as it is the dependent variable (Winters, 2017). Churn is a categorical binary variable as it only as two outcomes (`Yes` or `No` churn) which was discovered by using the `df['Churn'].value_counts()` function.

Independent Predictor Variable

A predictor variable is a candidate for inclusion in the model as a predictor of the target variable (Nisbet, Miner, Yale, Elder, & Peterson, 2018).

An independent variable in the dataset is tenure. Tenure is an integer – and, therefore, a discrete variable - in the dataset. This was discovered by using the `df.info()` function in Python 3.

Goal in Manipulation of Data

To predict which customers are likely to churn in order to:

1. Stall churn for these customers (Nisbet, Miner, Yale, Elder, & Peterson, 2018);
2. Focus the business's marketing efforts with the appropriate message (Nisbet, Miner, Yale, Elder, & Peterson, 2018).

Data Preparation Aim

To prepare data for exploration, where the predictors I will use for the model will be determined for analysis (Nisbet, Miner, Yale, Elder, & Peterson, 2018).

Statistical Identity

Column Name	Statistical Identity
CustomerID	Unique Verifier (this column was dropped during data preparation), Categorical/Nominal, Independent Variable
Gender	Categorical/Nominal, Binary/Dichotomous, Independent Variable
SeniorCitizen	Categorical/Nominal, Binary/Dichotomous, Independent Variable
Partner	Categorical/Nominal, Binary/Dichotomous, Independent Variable
Dependents	Categorical/Nominal, Binary/Dichotomous, Independent Variable
Tenure	Quantitative, Discrete, Independent Variable
PhoneService	Categorical/Nominal, Binary/Dichotomous, Independent Variable
MultipleLines	Categorical/Nominal, Independent Variable, Binary (after combining 'No' and 'No phone service' during data preparation)
InternetService	Categorical/Nominal, Independent Variable
OnlineSecurity	Categorical/Nominal, Independent Variable, Binary (after combining 'No' and 'No internet service' during data preparation)
OnlineBackup	Categorical/Nominal, Independent Variable, Binary (after combining 'No' and 'No internet service' during data preparation)
DeviceProtection	Categorical/Nominal, Independent Variable, Binary (after combining 'No' and 'No internet service' during data preparation)
TechSupport	Categorical/Nominal, Independent Variable, Binary (after combining 'No' and 'No internet service' during data preparation)
StreamingTV	Categorical/Nominal, Independent Variable, Binary (after combining 'No' and 'No internet service' during data preparation)
StreamingMovies	Categorical/Nominal, Independent Variable, Binary (after combining 'No' and 'No internet service' during data preparation)
Contract	Categorical/Ordinal, Independent Variable
PaperlessBilling	Categorical/Nominal, Binary/Dichotomous, Independent Variable
PaymentMethod	Categorical/Nominal, Independent Variable
MonthlyCharges	Quantitative, Continuous, Independent Variable
TotalCharges	Quantitative, Continuous, Independent Variable
Churn (target)	Categorical/Nominal, Binary/Dichotomous, Dependent Variable – Phenomenon to be predicted.
(UFHealth, n.d.)	

Steps for Data Cleaning:

1. Detect any "dirty" values

a. *Missing Values*: I did not find any missing values in the data set.

b. *Aberrant Values*: I did not find any aberrant values in the data set.

c. *Rare Values*: I did not find any rare values.

d. *Extreme Values*: I did not find any extreme values.

2. Data transformation

a. *Removing Duplicates*: There were no duplicate entries in the data set.

b. *Computing Indicator/Dummy Variables*: I replaced columns with categorical variables with dummy variables (McKinney, 2018).

c. *Drop Irrelevant Columns*: I dropped the customerID column as it is not relevant to the analysis.

```
In [4]: print(df.shape)

(7043, 21)

In [5]: # summary to get a sense for df's structure and notice the different columns (also known as "features" in machine learning)
        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
PaperlessBilling    7043 non-null object
PaymentMethod       7043 non-null object
MonthlyCharges      7043 non-null float64
TotalCharges        7043 non-null object
Churn               7043 non-null object
dtypes: float64(1), int64(2), object(18)
memory usage: 1.1+ MB
```

```
In [7]: # Count how many churners and non-churners the dataset contains.
        df['Churn'].value_counts()

Out[7]: No     5174
        Yes    1869
        Name: Churn, dtype: int64
```

```
In [8]: df.nunique()

Out[8]: customerID          7043
        gender                 2
        SeniorCitizen          2
        Partner                2
        Dependents             2
        tenure                73
        PhoneService           2
        MultipleLines          3
        InternetService        3
        OnlineSecurity         3
        OnlineBackup           3
        DeviceProtection       3
        TechSupport            3
        StreamingTV            3
        StreamingMovies        3
        Contract               3
        PaperlessBilling       2
        PaymentMethod          4
        MonthlyCharges      1585
        TotalCharges        6531
        Churn                  2
        dtype: int64
```

```
In [9]: sum(df.duplicated())

Out[9]: 0
```

```
In [10]: df.isnull().sum()

Out[10]: customerID          0
         gender              0
         SeniorCitizen       0
         Partner             0
         Dependents          0
         tenure              0
         PhoneService        0
         MultipleLines       0
         InternetService     0
         OnlineSecurity      0
         OnlineBackup        0
         DeviceProtection    0
         TechSupport         0
         StreamingTV         0
         StreamingMovies     0
         Contract            0
         PaperlessBilling    0
         PaymentMethod       0
         MonthlyCharges      0
         TotalCharges        0
         Churn               0
         dtype: int64
```

- This dataset has 7,043 samples, and 21 attributes(2 integers, 1 float, and 18 objects).
- No variable column has null/missing values.

Data Wrangling

In [11]:
```
# Total charges contain entries that are not floats. Use 'coerce' - invalid parsing will be set as NaN
df['TotalCharges'] = pd.to_numeric(df['TotalCharges'], errors = 'coerce')
df.loc[df['TotalCharges'].isna()==True]
```

Out[11]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	...	DeviceProtection	Tech
488	4472-LVYGI	Female	0	Yes	Yes	0	No	No phone service	DSL	Yes	...	Yes	
753	3115-CZMZD	Male	0	No	Yes	0	Yes	No	No	No internet service	...	No internet service	Nc
936	5709-LVOEQ	Female	0	Yes	Yes	0	Yes	No	DSL	Yes	...	Yes	
1082	4367-NUYAO	Male	0	Yes	Yes	0	Yes	Yes	No	No internet service	...	No internet service	Nc
1340	1371-DWPAZ	Female	0	Yes	Yes	0	No	No phone service	DSL	Yes	...	Yes	
3331	7644-OMVMY	Male	0	Yes	Yes	0	Yes	No	No	No internet service	...	No internet service	Nc
3826	3213-VVOLG	Male	0	Yes	Yes	0	Yes	Yes	No	No internet service	...	No internet service	Nc
4380	2520-SGTTA	Female	0	Yes	Yes	0	Yes	No	No	No internet service	...	No internet service	Nc
5218	2923-ARZLG	Male	0	Yes	Yes	0	Yes	No	No	No internet service	...	No internet service	Nc
6670	4075-WKNIU	Female	0	Yes	Yes	0	Yes	Yes	DSL	No	...	Yes	
6754	2775-SEFEE	Male	0	No	Yes	0	Yes	Yes	DSL	Yes	...	No	

11 rows × 21 columns

In [12]:
```
# `tenure = 0` for `TotalCharges` columns. Convert TotalCharges to 0 for these entries.
df[df['TotalCharges'].isna()==True] = 0
```

In [13]:
```
# Unique values for categorical column entries
df.gender.unique()
```

Out[13]: `array(['Female', 'Male', 0], dtype=object)`

In [14]:
```
df.Partner.unique()
```

Out[14]: `array(['Yes', 'No', 0], dtype=object)`

In [15]:
```
df.Dependents.unique()
```

Out[15]: `array(['No', 'Yes', 0], dtype=object)`

In [16]:
```
df.PhoneService.unique()
```

Out[16]: `array(['No', 'Yes', 0], dtype=object)`

In [17]:
```
df.MultipleLines.unique()
```

Out[17]: `array(['No phone service', 'No', 'Yes', 0], dtype=object)`

In [18]:
```
df.InternetService.unique()
```

Out[18]: `array(['DSL', 'Fiber optic', 'No', 0], dtype=object)`

In [19]:
```
df.OnlineSecurity.unique()
```

Out[19]: `array(['No', 'Yes', 'No internet service', 0], dtype=object)`

In [20]:
```
df.OnlineBackup.unique()
```

Out[20]: `array(['Yes', 'No', 'No internet service', 0], dtype=object)`

In [21]:
```
df.DeviceProtection.unique()
```

Out[21]: `array(['No', 'Yes', 'No internet service', 0], dtype=object)`

```
In [22]: df.TechSupport.unique()

Out[22]: array(['No', 'Yes', 'No internet service', 0], dtype=object)

In [23]: df.StreamingTV.unique()

Out[23]: array(['No', 'Yes', 'No internet service', 0], dtype=object)

In [24]: df.StreamingMovies.unique()

Out[24]: array(['No', 'Yes', 'No internet service', 0], dtype=object)

In [25]: df.Contract.unique()

Out[25]: array(['Month-to-month', 'One year', 'Two year', 0], dtype=object)

In [26]: df.Contract.value_counts()

Out[26]: Month-to-month    3875
         Two year          1685
         One year          1472
         0                   11
         Name: Contract, dtype: int64

In [27]: df.PaperlessBilling.unique()

Out[27]: array(['Yes', 'No', 0], dtype=object)

In [28]: df.PaymentMethod.unique()

Out[28]: array(['Electronic check', 'Mailed check', 'Bank transfer (automatic)',
                'Credit card (automatic)', 0], dtype=object)

In [29]: df.Churn.unique()

Out[29]: array(['No', 'Yes', 0], dtype=object)

In [30]: # convert categorical values into numerical values
         df.gender.replace(['Male','Female'],[0,1], inplace=True)
         df.Dependents.replace(['Yes','No'],[1,0], inplace=True)
         df.Partner.replace(['No', 'Yes'],[0,1], inplace=True)
         df.PhoneService.replace(['No', 'Yes'],[0,1], inplace=True)
         df.MultipleLines.replace(['No phone service','No', 'Yes'], [0,0,1], inplace=True)
         df.InternetService.replace(['No', 'DSL', 'Fiber optic'], [0,1,2], inplace=True)
         df.OnlineSecurity.replace(['No internet service','No', 'Yes'], [0,0,1], inplace=True)
         df.OnlineBackup.replace(['No internet service','No', 'Yes'], [0,0,1], inplace=True)
         df.DeviceProtection.replace(['No internet service','No', 'Yes'], [0,0,1], inplace=True)
         df.TechSupport.replace(['No internet service','No', 'Yes'], [0,0,1], inplace=True)
         df.StreamingTV.replace(['No internet service','No', 'Yes'], [0,0,1], inplace=True)
         df.StreamingMovies.replace(['No internet service','No', 'Yes'], [0,0,1], inplace=True)
         df.Contract.replace(['Month-to-month','One year', 'Two year'], [0,1,2], inplace=True)
         df.PaperlessBilling.replace(['No', 'Yes'], [0,1], inplace=True)
         df.PaymentMethod.replace(['Electronic check', 'Mailed check', 'Bank transfer (automatic)', 'Credit card (automatic)'], [1,2,3,4]
         df.Churn.replace(['No', 'Yes'], [0,1], inplace=True)

In [31]: # check that changes were made successfully
         df.gender.unique()

Out[31]: array([1, 0], dtype=int64)

In [32]: df.Partner.unique()

Out[32]: array([1, 0], dtype=int64)

In [33]: df.Dependents.unique()

Out[33]: array([0, 1], dtype=int64)

In [34]: df.PhoneService.unique()

Out[34]: array([0, 1], dtype=int64)

In [35]: df.MultipleLines.unique()

Out[35]: array([0, 1], dtype=int64)
```

```
In [36]: df.InternetService.unique()
Out[36]: array([1, 2, 0], dtype=int64)

In [37]: df.OnlineSecurity.unique()
Out[37]: array([0, 1], dtype=int64)

In [38]: df.OnlineBackup.unique()
Out[38]: array([1, 0], dtype=int64)

In [39]: df.DeviceProtection.unique()
Out[39]: array([0, 1], dtype=int64)

In [40]: df.TechSupport.unique()
Out[40]: array([0, 1], dtype=int64)

In [41]: df.StreamingTV.unique()
Out[41]: array([0, 1], dtype=int64)

In [42]: df.StreamingMovies.unique()
Out[42]: array([0, 1], dtype=int64)

In [43]: df.Contract.unique()
Out[43]: array([0, 1, 2], dtype=int64)
```

```
In [44]: df.PaperlessBilling.unique()
Out[44]: array([1, 0], dtype=int64)

In [45]: df.PaymentMethod.unique()
Out[45]: array([1, 2, 3, 4, 0], dtype=int64)

In [46]: df.Churn.unique()
Out[46]: array([0, 1], dtype=int64)
```

Legend

- `gender` : Male = 0, Female = 1
- `SeniorCitizen` : No = 0, Yes = 1
- `Partner` : No = 0, Yes = 1
- `Dependents` : No = 0, Yes = 1
- `tenure` : # months
- `PhoneService` : No = 0, Yes = 1
- `MultipleLines` : No = 0, Yes = 1
- `InternetService` : No = 0, DSL = 1, Fiber optic = 2
- `OnlineSecurity` : No = 0, Yes = 1
- `OnlineBackup` : No = 0, Yes = 1
- `DeviceProtection` : No = 0, Yes = 1
- `TechSupport` : No = 0, Yes = 1
- `StreamingTV` : No = 0, Yes = 1
- `StreamingMovies` : No = 0, Yes = 1
- `Contract` : Month-to-month = 0, One year = 1, Two year = 2
- `PaperlessBilling` : No = 0, Yes = 1
- `PaymentMethod` : None = 0, Electronic check = 1, Mailed check = 2, Bank transfer (automatic) = 3, Credit Card (automatic) = 4
- `Churn` : No = 0, Yes = 1

```
In [47]: # Drop Customer ID column
         df = df.drop(['customerID'], axis = 1)
```

In [52]:
```
# Group df by 'Churn' and compute the mean
df.groupby(['Churn']).mean()
```

Out[52]:

Churn	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtection
0	0.491689	0.128721	0.526479	0.342675	37.569965	0.899304	0.409161	1.073637	0.332431	0.367607	0.362002
1	0.502408	0.254682	0.357945	0.174425	17.979133	0.909042	0.454789	1.633494	0.157838	0.279829	0.291600

In [53]:
```
# Group df by 'Churn' and compute the standard deviation
df.groupby(['Churn']).std()
```

Out[53]:

Churn	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtection
0	0.499979	0.334923	0.499347	0.474650	24.113777	0.300955	0.491727	0.785149	0.471130	0.482200	0.480626
1	0.500128	0.435799	0.479524	0.379576	19.531123	0.287626	0.498085	0.594381	0.364687	0.449035	0.454621

In [54]:
```
# check to make sure changes were made successfully
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 20 columns):
gender              7043 non-null int64
SeniorCitizen       7043 non-null int64
Partner             7043 non-null int64
Dependents          7043 non-null int64
tenure              7043 non-null int64
PhoneService        7043 non-null int64
MultipleLines       7043 non-null int64
InternetService     7043 non-null int64
OnlineSecurity      7043 non-null int64
OnlineBackup        7043 non-null int64
DeviceProtection    7043 non-null int64
TechSupport         7043 non-null int64
StreamingTV         7043 non-null int64
StreamingMovies     7043 non-null int64
Contract            7043 non-null int64
PaperlessBilling    7043 non-null int64
PaymentMethod       7043 non-null int64
MonthlyCharges      7043 non-null float64
TotalCharges        7043 non-null float64
Churn               7043 non-null int64
dtypes: float64(2), int64(18)
memory usage: 1.1 MB
```

In [55]:
```
df.describe()
```

Out[55]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceP
count	7043.000000	7043.000000	7043.000000	7043.000000	7043.000000	7043.000000	7043.000000	7043.000000	7043.000000	7043.000000	704
mean	0.494534	0.162147	0.481755	0.298026	32.371149	0.901888	0.421269	1.222206	0.286100	0.344314	
std	0.500006	0.368612	0.499702	0.457424	24.559481	0.297487	0.493798	0.779535	0.451969	0.475178	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.000000	0.000000	0.000000	0.000000	9.000000	1.000000	0.000000	1.000000	0.000000	0.000000	
50%	0.000000	0.000000	0.000000	0.000000	29.000000	1.000000	0.000000	1.000000	0.000000	0.000000	
75%	1.000000	0.000000	1.000000	1.000000	55.000000	1.000000	1.000000	2.000000	1.000000	1.000000	
max	1.000000	1.000000	1.000000	1.000000	72.000000	1.000000	1.000000	2.000000	1.000000	1.000000	

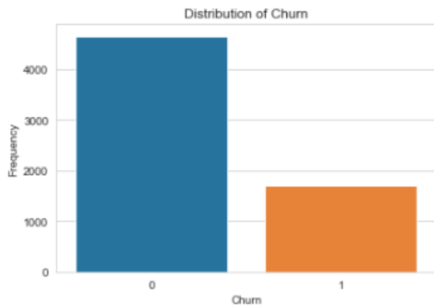
III: Data Analysis

Univariate Statistics

Data Visualization

In [767]:
```
sns.countplot(data = df, x = 'Churn')
plt.title('Distribution of Churn')
plt.ylabel('Frequency')
plt.xlabel('Churn')
```

Out[767]: `Text(0.5, 0, 'Churn')`



In [768]:
```
churn_True = df["Churn"][df["Churn"] == True]
print ("Churn Percentage = "+ str( (churn_True.shape[0] / df["Churn"].shape[0]) * 100 ) +("%"))
```

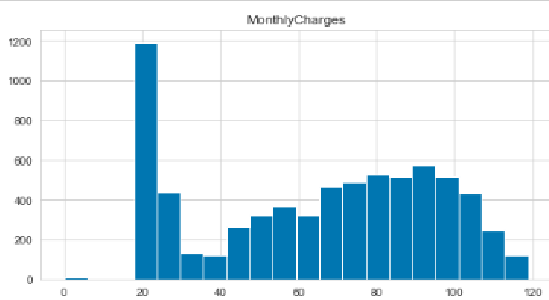
```
Churn Percentage = 26.747481108312343%
```

Imbalanced data - less churners than non-churners.

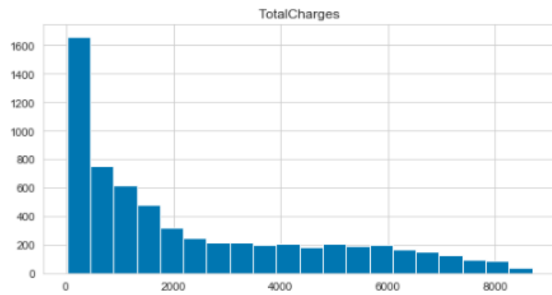
Univariate Exploration

Categorical Variables

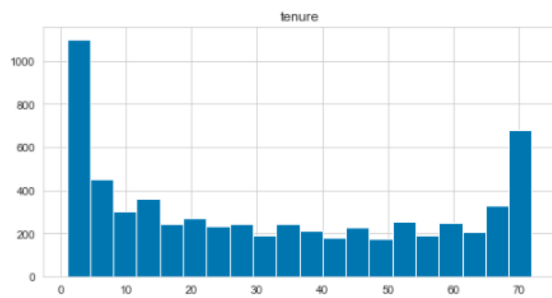
In [1072]:
```
df.hist(column = 'MonthlyCharges', bins=20, figsize=(8,4))
plt.show()
```



In [770]:
```
df.hist(column = 'TotalCharges', bins=20, figsize=(8,4))
plt.show()
```



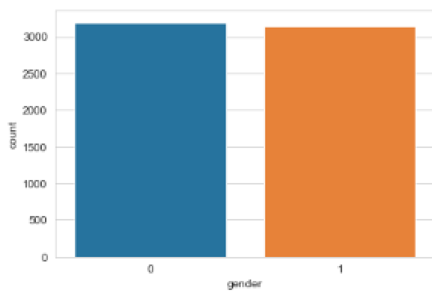
In [771]:
```
df.hist(column = 'tenure', bins=20, figsize=(8,4))
plt.show()
```



Continuous Variables

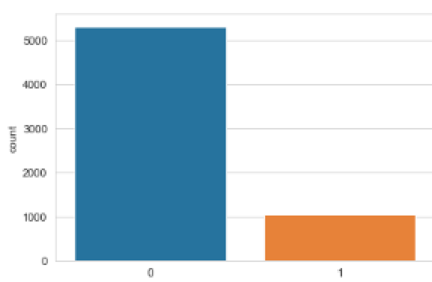
In [772]:
```
sns.countplot(x = 'gender', data = df)
```

Out[772]: <matplotlib.axes._subplots.AxesSubplot at 0x193829b9898>

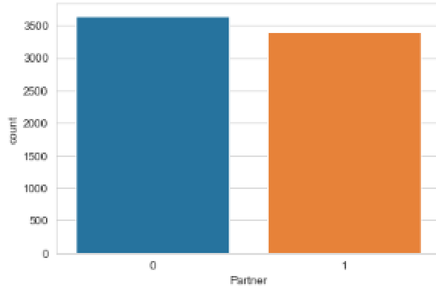


In [773]:
```
sns.countplot(x = 'SeniorCitizen', data = df)
```

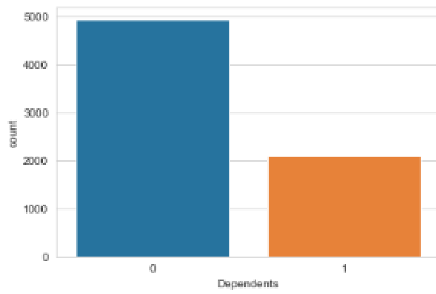
Out[773]: <matplotlib.axes._subplots.AxesSubplot at 0x19382a33c88>



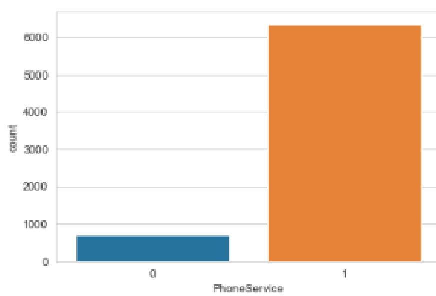
In [1077]:
```
sns.countplot(x = 'Partner', data = df)
```
Out[1077]: <matplotlib.axes._subplots.AxesSubplot at 0x193863a70b8>



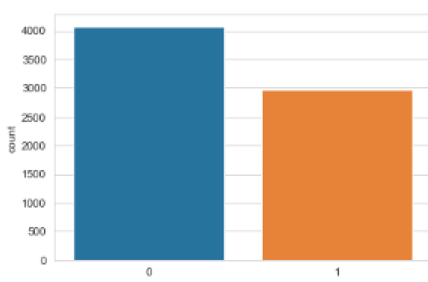
In [1078]:
```
sns.countplot(x = 'Dependents', data = df)
```
Out[1078]: <matplotlib.axes._subplots.AxesSubplot at 0x193863e1438>



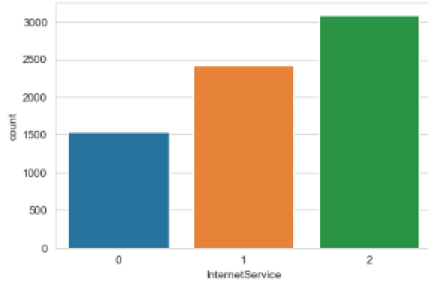
In [1079]:
```
sns.countplot(x = 'PhoneService', data = df)
```
Out[1079]: <matplotlib.axes._subplots.AxesSubplot at 0x193864369e8>



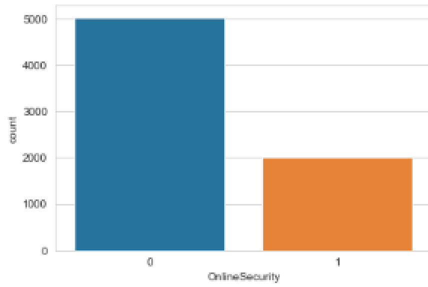
In [1080]:
```
sns.countplot(x = 'MultipleLines', data = df)
```
Out[1080]: <matplotlib.axes._subplots.AxesSubplot at 0x19386484668>



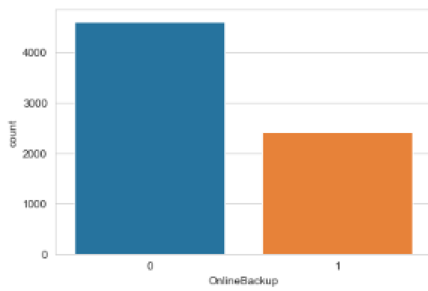
```
In [1081]: sns.countplot(x = 'InternetService', data = df)
Out[1081]: <matplotlib.axes._subplots.AxesSubplot at 0x193864b47f0>
```



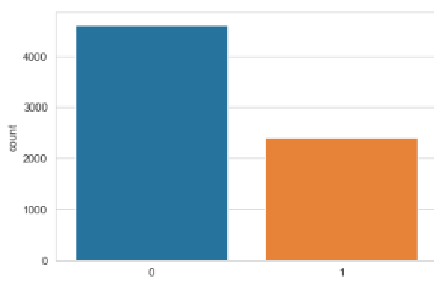

```
In [1082]: sns.countplot(x = 'OnlineSecurity', data = df)
Out[1082]: <matplotlib.axes._subplots.AxesSubplot at 0x193865039b0>
```



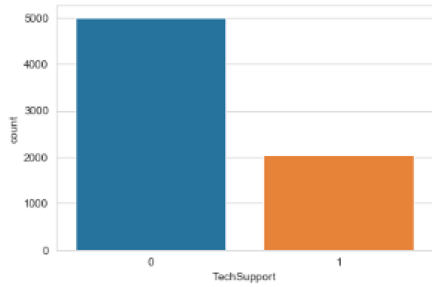

```
In [1083]: sns.countplot(x = 'OnlineBackup', data = df)
Out[1083]: <matplotlib.axes._subplots.AxesSubplot at 0x19386561eb8>
```



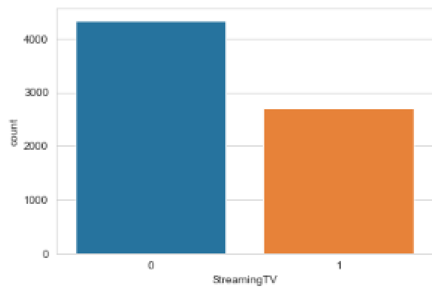

```
In [1084]: sns.countplot(x = 'DeviceProtection', data = df)
Out[1084]: <matplotlib.axes._subplots.AxesSubplot at 0x19384258cf8>
```



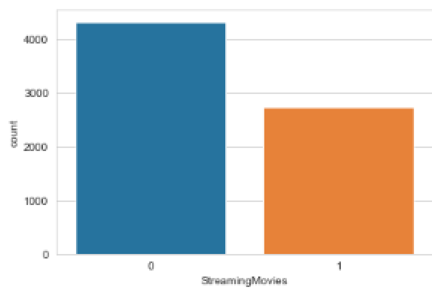
In [1085]:
```
sns.countplot(x = 'TechSupport', data = df)
```
Out[1085]: <matplotlib.axes._subplots.AxesSubplot at 0x193865ce198>



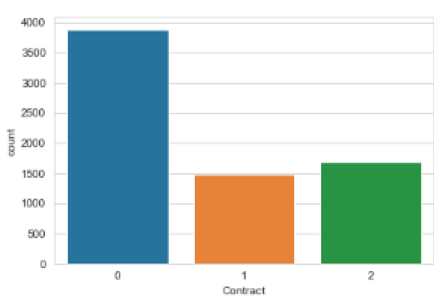
In [1086]:
```
sns.countplot(x = 'StreamingTV', data = df)
```
Out[1086]: <matplotlib.axes._subplots.AxesSubplot at 0x19386604c18>



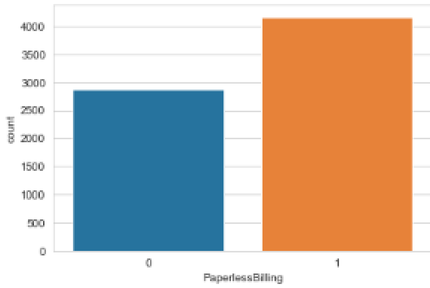
In [1087]:
```
sns.countplot(x = 'StreamingMovies', data = df)
```
Out[1087]: <matplotlib.axes._subplots.AxesSubplot at 0x1938665ceb8>



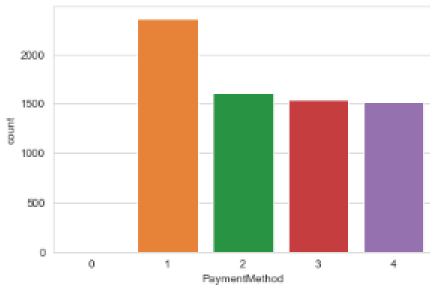
In [1088]:
```
sns.countplot(x = 'Contract', data = df)
```
Out[1088]: <matplotlib.axes._subplots.AxesSubplot at 0x1938669aef0>



In [1089]:
```
sns.countplot(x = 'PaperlessBilling', data = df)
```
Out[1089]: <matplotlib.axes._subplots.AxesSubplot at 0x19387986940>



In [1090]:
```
sns.countplot(x = 'PaymentMethod', data = df)
```
Out[1090]: <matplotlib.axes._subplots.AxesSubplot at 0x193879e30b8>

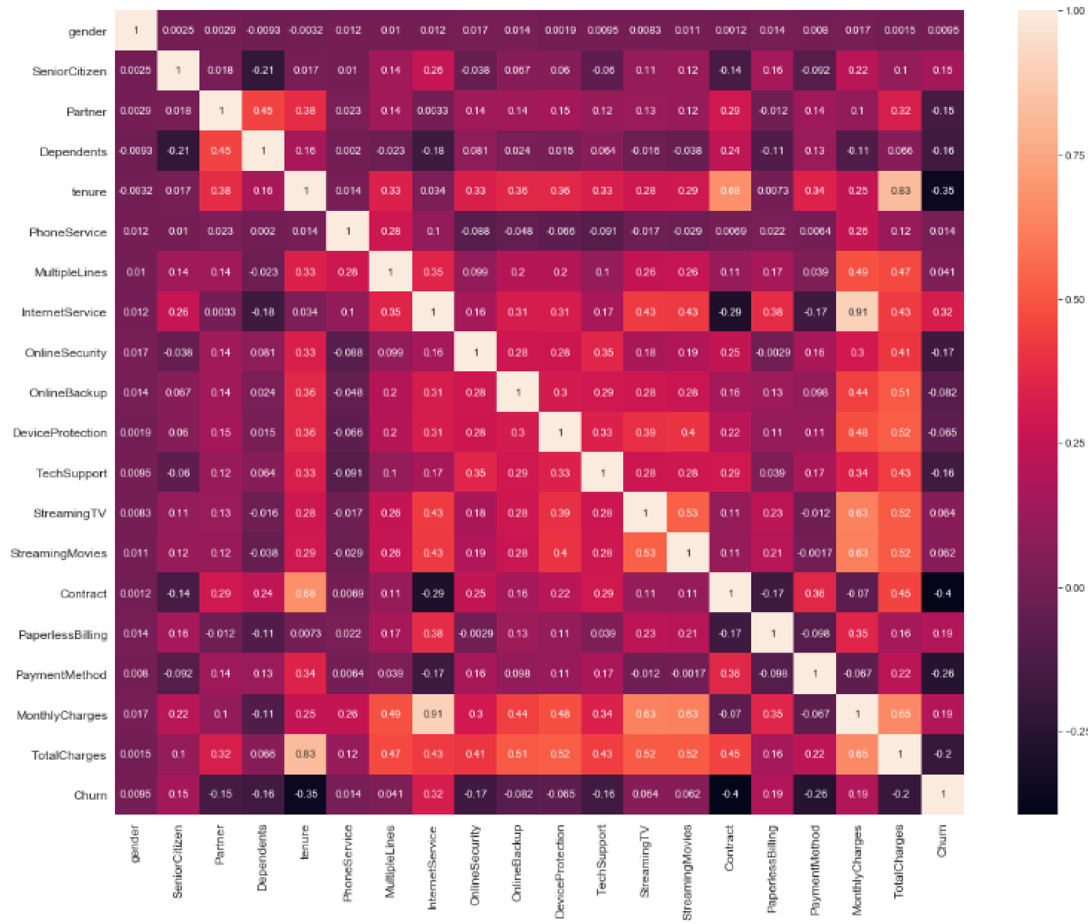


Bivariate Exploration

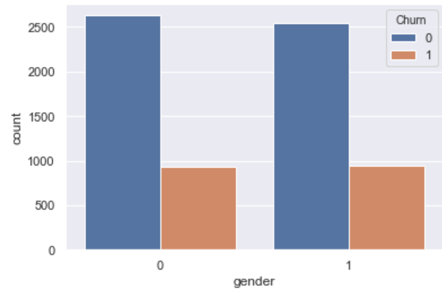
Bivariate Exploration

Explore relationships with each variable and churn

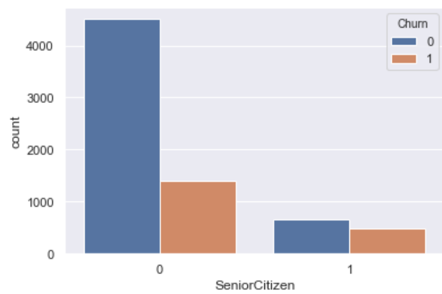
In [1389]:
```
# heatmap
corr = df.corr()
sns.heatmap(corr, xticklabels = corr.columns.values, yticklabels = corr.columns.values, annot = True)
heat_map = plt.gcf()
heat_map.set_size_inches(18,14)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.show()
```



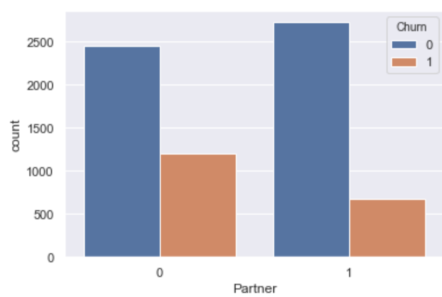
In [480]:
```
sns.countplot(data = df, x = 'gender', hue = 'Churn')
```
Out[480]: <matplotlib.axes._subplots.AxesSubplot at 0x19a15614048>



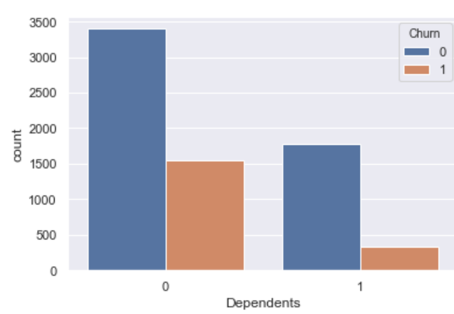
In [481]:
```
sns.countplot(data = df, x = 'SeniorCitizen', hue = 'Churn')
```
Out[481]: <matplotlib.axes._subplots.AxesSubplot at 0x19a11f575f8>



In [482]:
```
sns.countplot(data = df, x = 'Partner', hue = 'Churn')
```
Out[482]: <matplotlib.axes._subplots.AxesSubplot at 0x19a1418f828>

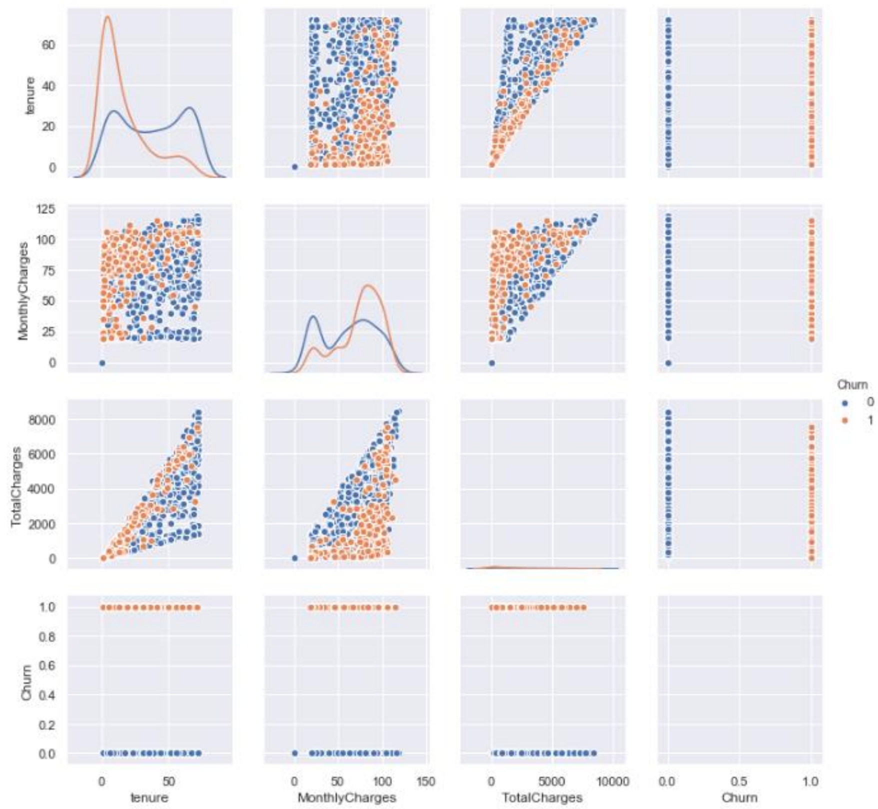


In [483]:
```
sns.countplot(data = df, x = 'Dependents', hue = 'Churn')
```
Out[483]: <matplotlib.axes._subplots.AxesSubplot at 0x19a14052748>



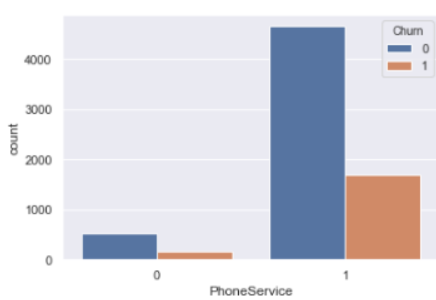
In [484]:
```
# Group all continuous variables
numeric = df[['tenure', 'MonthlyCharges', 'TotalCharges']]
```

In [514]:
```
# Plot continuous variables against churn
numeric = pd.concat([numeric, df["Churn"]],axis=1) #Add the 'Churn' variable to the numeric dataset

g = sns.PairGrid(numeric.sample(n=1000), hue="Churn")
g = g.map_offdiag(plt.scatter, linewidths=1, edgecolor="w", s=40)
g = g.map_diag(sns.kdeplot)
g = g.add_legend()
```



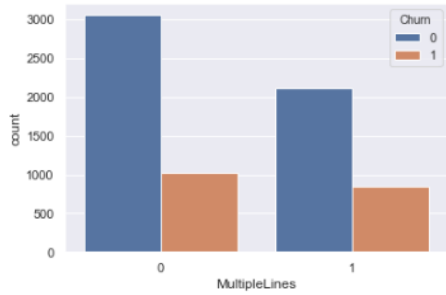
In [492]:
```
sns.countplot(data = df, x = 'PhoneService', hue = 'Churn')
```

Out[492]: <matplotlib.axes._subplots.AxesSubplot at 0x19a15897780>



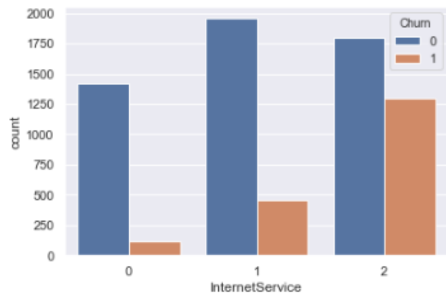
```
In [493]: sns.countplot(data = df, x = 'MultipleLines', hue = 'Churn')
```

```
Out[493]: <matplotlib.axes._subplots.AxesSubplot at 0x19a1595a3c8>
```



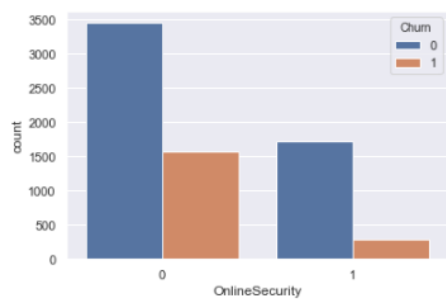

```
In [495]: sns.countplot(data = df, x = 'InternetService', hue = 'Churn')
```

```
Out[495]: <matplotlib.axes._subplots.AxesSubplot at 0x19a1599d8d0>
```



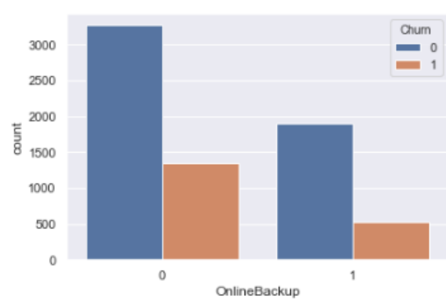

```
In [496]: sns.countplot(data = df, x = 'OnlineSecurity', hue = 'Churn')
```

```
Out[496]: <matplotlib.axes._subplots.AxesSubplot at 0x19a15a01b38>
```

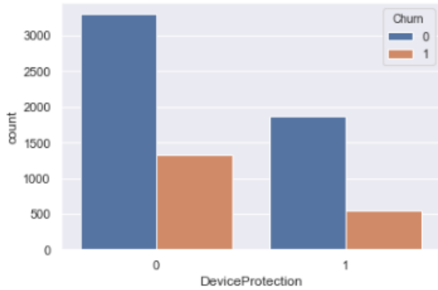



```
In [497]: sns.countplot(data = df, x = 'OnlineBackup', hue = 'Churn')
```

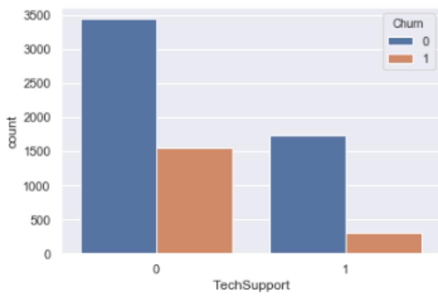
```
Out[497]: <matplotlib.axes._subplots.AxesSubplot at 0x19a15a5bf60>
```



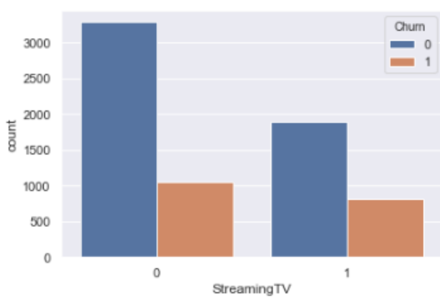
```
In [498]: sns.countplot(data = df, x = 'DeviceProtection', hue = 'Churn')
Out[498]: <matplotlib.axes._subplots.AxesSubplot at 0x19a15abe160>
```



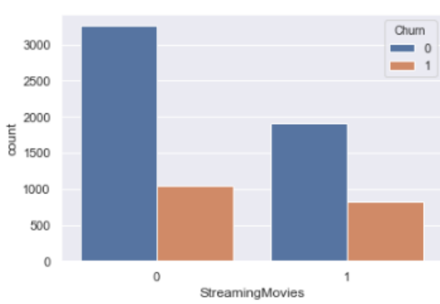

```
In [499]: sns.countplot(data = df, x = 'TechSupport', hue = 'Churn')
Out[499]: <matplotlib.axes._subplots.AxesSubplot at 0x19a15b0c160>
```



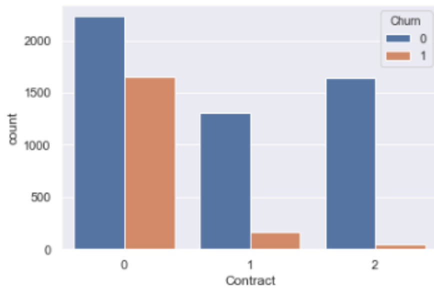

```
In [500]: sns.countplot(data = df, x = 'StreamingTV', hue = 'Churn')
Out[500]: <matplotlib.axes._subplots.AxesSubplot at 0x19a15f268d0>
```



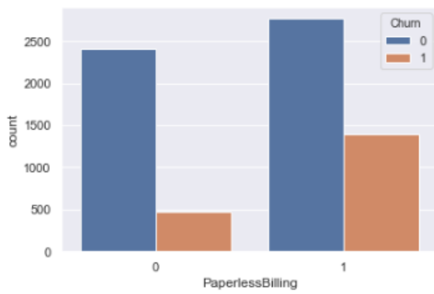

```
In [501]: sns.countplot(data = df, x = 'StreamingMovies', hue = 'Churn')
Out[501]: <matplotlib.axes._subplots.AxesSubplot at 0x19a15f7e668>
```



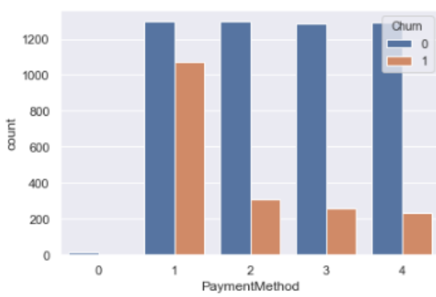
```
In [502]: sns.countplot(data = df, x = 'Contract', hue = 'Churn')
Out[502]: <matplotlib.axes._subplots.AxesSubplot at 0x19a15fd8630>
```



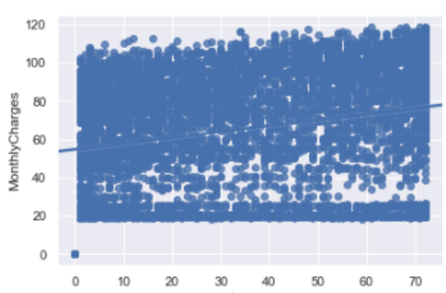

```
In [503]: sns.countplot(data = df, x = 'PaperlessBilling', hue = 'Churn')
Out[503]: <matplotlib.axes._subplots.AxesSubplot at 0x19a16025a58>
```



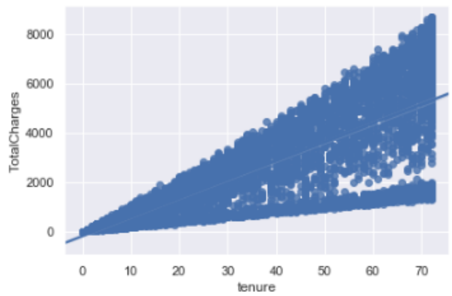

```
In [504]: sns.countplot(data = df, x = 'PaymentMethod', hue = 'Churn')
Out[504]: <matplotlib.axes._subplots.AxesSubplot at 0x19a16076f98>
```



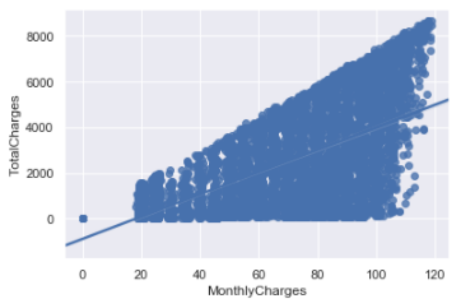

```
In [728]: sns.regplot(data = df, x = 'tenure', y = 'MonthlyCharges')
Out[728]: <matplotlib.axes._subplots.AxesSubplot at 0x19a129eecc0>
```



In [732]:
```
sns.regplot(data = df, x = 'tenure', y = 'TotalCharges')
```
Out[732]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x19a17b49f60>
```



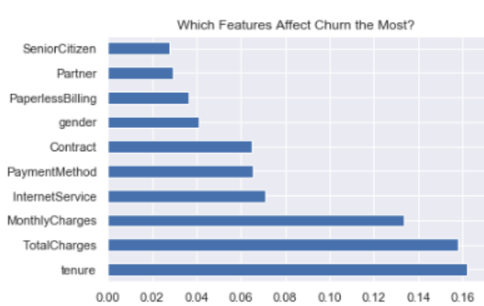
In [733]:
```
sns.regplot(data = df, x = 'MonthlyCharges', y = 'TotalCharges')
```
Out[733]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x19a08fd1898>
```



Feature Selection

In [1291]:
```
# Feature Selection
from sklearn.ensemble import ExtraTreesClassifier
model = ExtraTreesClassifier()
X = df[['gender', 'SeniorCitizen', 'Partner', 'Dependents', 'tenure', 'PhoneService', 'MultipleLines', 'InternetService',
        'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovies', 'Contract',
        'PaperlessBilling', 'PaymentMethod', 'MonthlyCharges', 'TotalCharges']]
y = df.Churn
model.fit(X,y)
print(model.feature_importances_) #use inbuilt class feature_importances of tree based classifiers
#plot graph of feature importances for better visualization
feat_importances = pd.Series(model.feature_importances_, index=X.columns)
feat_importances.nlargest(10).plot(kind='barh')
plt.title('Which Features Affect Churn the Most?')
plt.show()
```

```
[0.04090442 0.02763017 0.02960382 0.02413577 0.16195241 0.00838593
 0.02438348 0.07115406 0.02604704 0.02622974 0.02530902 0.02749641
 0.0240406  0.02465344 0.06485558 0.03629265 0.06540484 0.13363005
 0.15789057]
```



Detecting Multicollinearity

In [101]:
```
X = add_constant(df2)
>>> pd.Series([variance_inflation_factor(X.values, i)
               for i in range(X.shape[1])],
              index=X.columns)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\numpy\core\fromnumeric.py:57: FutureWarning: Method .ptp is deprecated and will be r
emoved in a future version. Use numpy.ptp instead.
  return getattr(obj, method)(*args, **kwds)
```

Out[101]:
```
const              20.462840
tenure              6.908941
TotalCharges        9.971263
MonthlyCharges     11.956848
InternetService     8.368575
PaymentMethod       1.205364
Contract            2.353670
Churn               1.342594
dtype: float64
```

If VIF > 10, then multicollinearity is high.

Ignore columns with dummy variables with high VIFs - If you have high VIFs for dummy variables representing nominal variables with three or more categories, those are usually not a problem (Statistics How To, 2015).

We will remove MonthlyCharges from the analysis.

In [102]:
```
df3 = df2[['tenure', 'TotalCharges', 'InternetService', 'PaymentMethod', 'Contract', 'Churn']]
```

In [103]:
```
# Check if there is any highly correlated variables remaining
X = add_constant(df3)
pd.Series([variance_inflation_factor(X.values, i)
               for i in range(X.shape[1])],
              index=X.columns)
```

Out[103]:
```
const              13.119246
tenure              6.092509
TotalCharges        6.500336
InternetService     2.534570
PaymentMethod       1.205360
Contract            2.295199
Churn               1.339830
dtype: float64
```

Analytic (Descriptive Method): Principal Component Analysis

Method

- **Import packages**: From sklearn.preprocessing import StandardScalar
- **Preprocess the data**: Scale the data so that each feature has unit variance so that one does not have a greater impact than another.
- **Transform data** to its first two principal components.
- **Plot the data** as a scatter plot.
- **Visualize** the PCA components in the form of a heat map (Choudhary, 2019).

Findings

- The results below show that the PC1 holds 45.3% of the information, while the PC2 holds only 26.0% of the information. In addition, 28.7% of the information was lost during this transformation.
- Tenure is most affected by churn, followed by TotalCharges.

Descriptive Method - PCA

In [1402]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(df3)
```

Out[1402]:
```
StandardScaler(copy=True, with_mean=True, with_std=True)
```

In [1403]:
```
scaled_data = scaler.transform(df3)
```

In [1404]:
```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
pca.fit(scaled_data)
```

Out[1404]:
```
PCA(copy=True, iterated_power='auto', n_components=2, random_state=None,
  svd_solver='auto', tol=0.0, whiten=False)
```

In [1405]:
```
x_pca = pca.transform(scaled_data)
x_pca
```

Out[1405]:
```
array([[-1.73936092, -0.73614836],
       [ 0.27236993, -0.50486426],
       [-2.18411141, -0.0772131 ],
       ...,
       [-1.44861952, -0.5986665 ],
       [-2.16402999,  0.92182795],
       [ 2.83210748,  1.30771335]])
```

In [1406]:
```
x_pca_df = pd.DataFrame(data = x_pca, columns = ['principal component 1', 'principal component 2'])
```

In [1407]:
```
x_pca_df.tail()
```

Out[1407]:

	principal component 1	principal component 2
7038	0.067489	-0.555320
7039	2.766124	1.485241
7040	-1.448620	-0.598667
7041	-2.164030	0.921828
7042	2.832107	1.307713

In [1408]:
```
print('Explained variation per principal component: {}'.format(pca.explained_variance_ratio_))
```

```
Explained variation per principal component: [0.45350653 0.25954695]
```

In [1409]:
```
scaled_data.shape
```

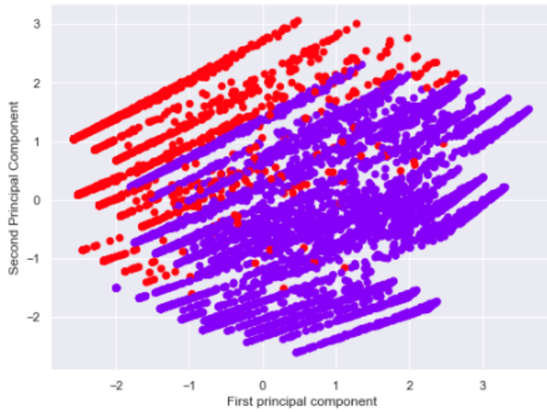
Out[1409]:
```
(7043, 6)
```

In [1410]:
```
x_pca.shape
```

Out[1410]:
```
(7043, 2)
```

In [1411]:
```
plt.figure(figsize=(8,6))
plt.scatter(x_pca[:,0],x_pca[:,1],c=df3['Churn'], cmap='rainbow')
plt.xlabel('First principal component')
plt.ylabel('Second Principal Component')
```

Out[1411]: `Text(0, 0.5, 'Second Principal Component')`

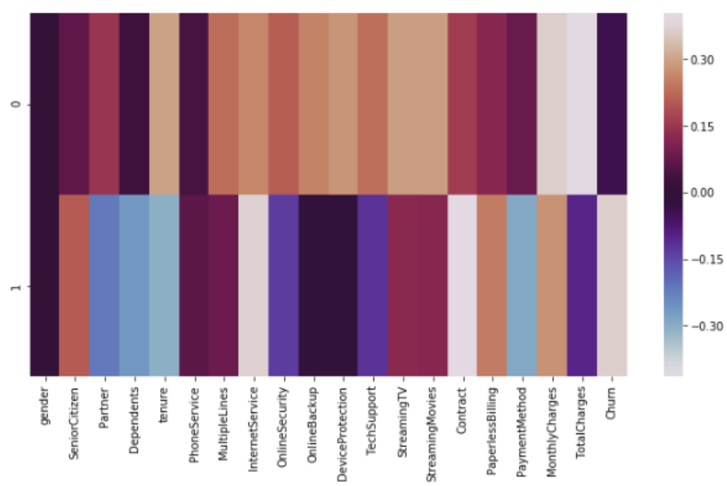


In [1412]:
```
pca.components_
```

Out[1412]:
```
array([[ 0.55413705,  0.46603286, -0.05132902,  0.33227889,  0.49656174,
        -0.34077561],
       [ 0.17559657,  0.47235826,  0.73534876, -0.21063492, -0.17436679,
         0.3612961 ]])
```

In [131]:
```
map= pd.DataFrame(pca.components_,columns=df.columns)
plt.figure(figsize=(12,6))
sns.heatmap(map,cmap='twilight')
```

Out[131]: `<matplotlib.axes._subplots.AxesSubplot at 0x1bbf17a9c18>`



Evaluative (Predictive) Method I: Logistic Regression

Method

- **Import packages:** From scikit-learn import train_test_split, LogisticRegression, confusion_matrix, and classification_report.
- **Create a sample of the dataset** which is 25% the size of the dataset.
- **Perform logistic regression**.
- **Up-sample the minority class** (successful churn).
- **Perform logistic regression**.

Findings

- **Logistic regression pre-up-sample**:
  - Accuracy = 81%
  - Precision score for predicting a positive churn = 66%
  - Recall score for predicting a positive churn = 55%
- **Logistic regression post-up-sample**:
  - Accuracy = 78%
  - Precision score for predicting a positive churn = 75%
  - Recall score fore predicting a positive churn = 81%.

Predictive Analysis - Logistic Regression

In [1159]:

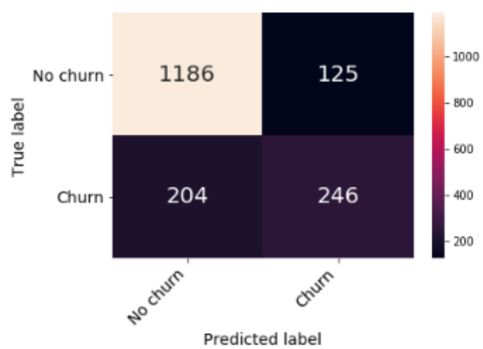
```
from sklearn.model_selection import train_test_split
train, test = train_test_split(df3, test_size = 0.25)

train_y = train['Churn']
test_y = test['Churn']

train_x = train
train_x.pop('Churn')
test_x = test
test_x.pop('Churn')
```

In [117]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, classification_report

logisticRegr = LogisticRegression()
logisticRegr.fit(X=train_x, y=train_y)

test_y_pred = logisticRegr.predict(test_x)
confusion_matrix = confusion_matrix(test_y, test_y_pred)
print('Intercept: ' + str(logisticRegr.intercept_))
print('Regression: ' + str(logisticRegr.coef_))
print('Accuracy of logistic regression classifier on test set: {:.2f}'.format(logisticRegr.score(test_x, test_y)))
print(classification_report(test_y, test_y_pred))

confusion_matrix_df = pd.DataFrame(confusion_matrix, ('No churn', 'Churn'), ('No churn', 'Churn'))
heatmap = sns.heatmap(confusion_matrix_df, annot=True, annot_kws={"size": 20}, fmt="d")
heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=0, ha='right', fontsize = 14)
heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=45, ha='right', fontsize = 14)
plt.ylabel('True label', fontsize = 14)
plt.xlabel('Predicted label', fontsize = 14)
```

```
Intercept: [-0.13438647]
Regression: [[ -6.77924454e-02   4.31672467e-04   6.93175313e-01  -2.93904015e-01
   -6.46006659e-01]]
Accuracy of logistic regression classifier on test set: 0.81
              precision    recall  f1-score   support

           0       0.85      0.90      0.88      1311
           1       0.66      0.55      0.60       450

   micro avg       0.81      0.81      0.81      1761
   macro avg       0.76      0.73      0.74      1761
weighted avg       0.80      0.81      0.81      1761
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:433: FutureWarning: Default solver will be changed
to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
```

Out[117]: Text(0.5, 15.0, 'Predicted label')

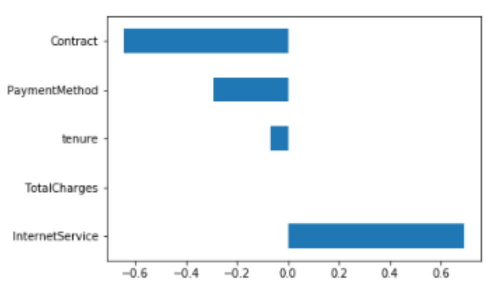


In [118]:
```
print(logisticRegr.coef_)
```

```
[[ -6.77924454e-02   4.31672467e-04   6.93175313e-01  -2.93904015e-01
   -6.46006659e-01]]
```

In [119]:
```
feat_importances = pd.Series(logisticRegr.coef_[0], index=train.columns)
feat_importances.nlargest(10).plot(kind='barh')
```

Out[119]: <matplotlib.axes._subplots.AxesSubplot at 0x1bbf37c4390>



We got 80% classification accuracy from our logistic regression classifier. However, the precision and recall for predictions in the positive class (churn) are relatively low, which suggests our data set may be imbalanced.

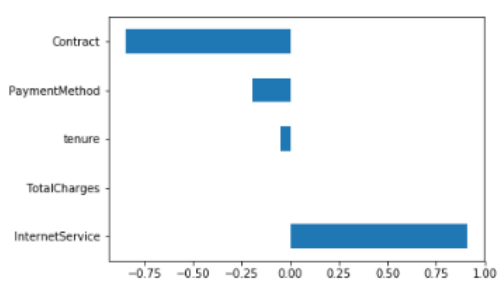
```
In [120]: df.Churn.value_counts()
Out[120]: 0    5174
          1    1869
          Name: Churn, dtype: int64
```

Up-sampling the minority class

```
In [121]: from sklearn.utils import resample

          data_majority = df3[df3['Churn']==0]
          data_minority = df3[df3['Churn']==1]

          data_minority_upsampled = resample(data_minority,
          replace=True,
          n_samples=4653, #same number of samples as majority class
          random_state=1) #set the seed for random resampling
          # Combine resampled results
          data_upsampled = pd.concat([data_majority, data_minority_upsampled])

          data_upsampled['Churn'].value_counts()
Out[121]: 0    5174
          1    4653
          Name: Churn, dtype: int64
```

```
In [122]: train, test = train_test_split(data_upsampled, test_size = 0.25)

          train_y_upsampled = train['Churn']
          test_y_upsampled = test['Churn']

          train_x_upsampled = train
          train_x_upsampled.pop('Churn')
          test_x_upsampled = test
          test_x_upsampled.pop('Churn')

          logisticRegr_balanced = LogisticRegression()
          logisticRegr_balanced.fit(X=train_x_upsampled, y=train_y_upsampled)

          test_y_pred_balanced = logisticRegr_balanced.predict(test_x_upsampled)
          print('Accuracy of logistic regression classifier on test set: {:.2f}'.format(logisticRegr_balanced.score(test_x_upsampled,
                                                                                                                   test_y_upsampled)))
          print(classification_report(test_y_upsampled, test_y_pred_balanced))

          Accuracy of logistic regression classifier on test set: 0.78
                        precision    recall  f1-score   support

                     0       0.82      0.75      0.79      1297
                     1       0.75      0.81      0.78      1160

             micro avg       0.78      0.78      0.78      2457
             macro avg       0.78      0.78      0.78      2457
          weighted avg       0.79      0.78      0.78      2457
```

```
In [123]: logisticRegr2= LogisticRegression()
          logisticRegr2.fit(X=train_x_upsampled, y=train_y_upsampled)
          feat_importances = pd.Series(logisticRegr2.coef_[0], index=train.columns)
          feat_importances.nlargest(10).plot(kind='barh')

          C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:433: FutureWarning: Default solver will be changed
          to 'lbfgs' in 0.22. Specify a solver to silence this warning.
            FutureWarning)
Out[123]: <matplotlib.axes._subplots.AxesSubplot at 0x1bbf38329b0>
```



The overall accuracy has decreased, but the precision and recall scores for predicting a churn have increased.

Evaluative (Predictive) Method I: Random Forest Classifier

Method

- **Import packages:** From scikit-learn import RandomForestClassifier, classification_report, and accuracy_score.
- **Create a sample of the dataset** which is 25% the size of the dataset.
- **Perform random forest classifier test.**

Findings

- Accuracy = 76%
- Precision score for predicting a positive churn = 55%
- Recall score for predicting a positive churn = 47%

Tree-Based Algorithms - Random Forest Classifier

```
In [124]: from sklearn.ensemble import RandomForestClassifier
          classifier = RandomForestClassifier(n_estimators=200, random_state=0)
          classifier.fit(train_x, train_y)
          predictions = classifier.predict(test_x)
```

```
In [125]: from sklearn.metrics import classification_report, accuracy_score
          print(classification_report(test_y, predictions))
          print(accuracy_score(test_y, predictions))

                       precision    recall  f1-score   support

                    0       0.83      0.86      0.85      1311
                    1       0.55      0.48      0.51       450

            micro avg       0.77      0.77      0.77      1761
            macro avg       0.69      0.67      0.68      1761
         weighted avg       0.76      0.77      0.76      1761

          0.76660988075
```

```
In [126]: classifier.feature_importances_
Out[126]: array([ 0.2189393 ,  0.54177526,  0.07887995,  0.05727837,  0.10312712])
```

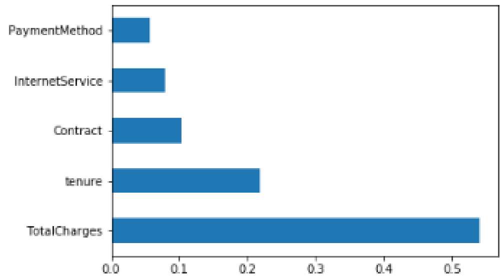

```
In [126]: classifier.feature_importances_
Out[126]: array([ 0.2189393 ,  0.54177526,  0.07887995,  0.05727837,  0.10312712])
```


```
In [127]: X.columns
Out[127]: Index(['const', 'tenure', 'TotalCharges', 'InternetService', 'PaymentMethod',
                 'Contract', 'Churn'],
                dtype='object')
```

```
In [128]: train.columns
Out[128]: Index(['tenure', 'TotalCharges', 'InternetService', 'PaymentMethod',
                 'Contract'],
                dtype='object')
```

In [129]:
```
feat_importances = pd.Series(classifier.feature_importances_, index=train.columns)
feat_importances.nlargest(10).plot(kind='barh')
```

Out[129]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1bbf39f82b0>
```



Justifying the Methods Used

Principal Component Analysis (PCA)

I selected PCA to choose the most important variables influencing the churn rate. It is difficult to visualize a high dimensional dataset. PCA is a great tool to reduce the dimensions of the dataset by finding the two principal components and, as a result, visualize the data in two-dimensional space as a single scatter plot (Choudhary, 2019). PCA will reduce the dataset to fewer dimensions while maintaining as much distance between the variables as possible (Tufféry, 2011).

Logistic Regression:

I selected logistic regression as my non-descriptive method as it is a method for predicting binary classes. Churn is a categorical variable with two options (churn or not churn). It is easy to implement and is used to estimate the relationship between a dependent binary variable and independent variables (Navlani, 2019).

Random Forest:

I also selected random forests as they are good models for unbalanced datasets (Data Skunkworks, 2018). I wanted to compare the two predictive models for further insight into the nuances of the dataset.

Justifying the Visualizations Used

Matplotlib

One of the original Python data visualization libraries. Other libraries are built on top of matplotlib or work in tandem with it. Low level interface; provides the most freedom (Tanner, 2019).

Seaborn

A high-level interface with great default styles (Tanner, 2019). Creates beautiful charts with only a few lines of code. A more aesthetically pleasing data visualization tool than matplotlib (Bierly, 2016).

IV: Data Summary

Eliminating Discrimination

Representative data set

The data set came from the source and I am not aware of the company's policies around data collection. Perhaps some of the data collection may have not been completed properly. However, the data was sufficiently cleaned and easy to work with.

Biases

I did not detect any biases in the original data. However, the models that I used for descriptive and predictive analyses may have placed more value on some values over others. I tried to account for this by using multiple models for prediction and feature selection in combination with PCA.

Continually test data predictions: I tested my two prediction models and up-sampled to create an equal number of churners and non-churners in my sample (Workfront, 2018).

Phenomenon Detection

Tenure was the strongest indicator of churn, followed by total charges. This was discovered during the Descriptive Analysis stage.

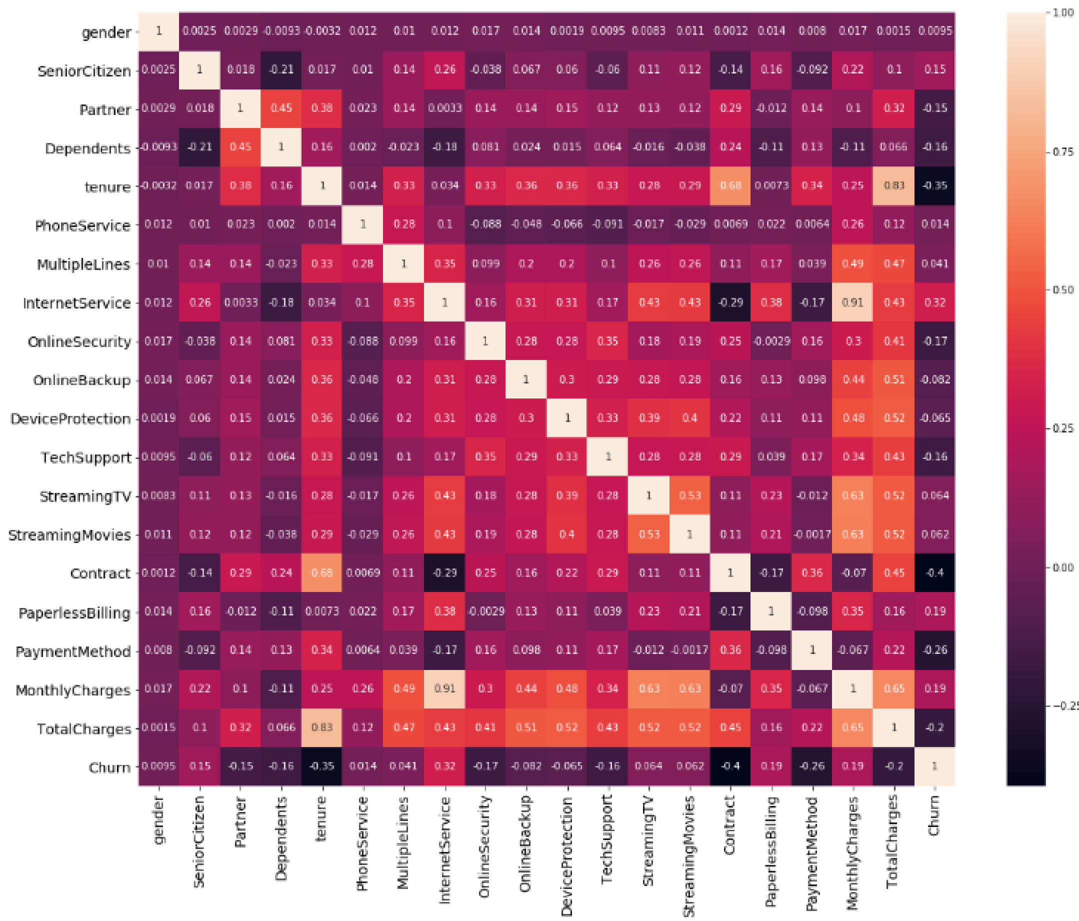
Methods for Detecting Interactions and Feature

Correlation Heatmap

I used a seaborn correlation heatmap to explore the relationships between variables in the initial stages of analysis.

Explore relationships with each variable and churn

In [132]:
```
# heatmap
corr = df.corr()
sns.heatmap(corr, xticklabels = corr.columns.values, yticklabels = corr.columns.values, annot = True)
heat_map = plt.gcf()
heat_map.set_size_inches(18,14)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.show()
```



Multicollinearity Detection

I detected multicollinearity using variance inflation factor analysis. This analysis indicated that MonthlyCharges had a VIF value of over 10 (15.4). MonthlyCharges was removed from the analysis.

Detecting Multicollinearity

```
In [101]: X = add_constant(df2)
          >>> pd.Series([variance_inflation_factor(X.values, i)
                         for i in range(X.shape[1])],
                        index=X.columns)

C:\ProgramData\Anaconda3\lib\site-packages\numpy\core\fromnumeric.py:57: FutureWarning: Method .ptp is deprecated and will be r
emoved in a future version. Use numpy.ptp instead.
  return getattr(obj, method)(*args, **kwds)

Out[101]: const              20.462840
          tenure              6.908941
          TotalCharges        9.971263
          MonthlyCharges     11.956848
          InternetService     8.368575
          PaymentMethod       1.205364
          Contract            2.353670
          Churn               1.342594
          dtype: float64
```

Feature Selection:

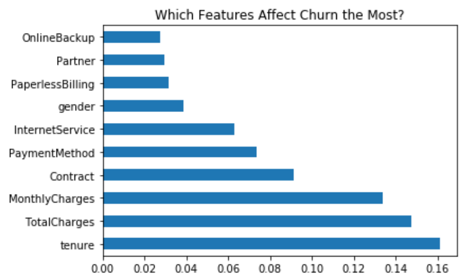
I used scikit-learn to find out which features affect churn the most. This was plotted in a bar graph and was used to decide the most important predictor variables. I chose the six most important predictor variables (tenure, TotalCharges, MonthlyCharges, Contract, and InternetService).

Feature Selection

In [99]:
```
# Feature Selection
from sklearn.ensemble import ExtraTreesClassifier
model = ExtraTreesClassifier()
X = df[['gender', 'SeniorCitizen', 'Partner', 'Dependents', 'tenure', 'PhoneService', 'MultipleLines', 'InternetService',
        'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovies', 'Contract',
        'PaperlessBilling', 'PaymentMethod', 'MonthlyCharges', 'TotalCharges']]
y = df.Churn
model.fit(X,y)
print(model.feature_importances_) #use inbuilt class feature_importances of tree based classifiers
#plot graph of feature importances for better visualization
feat_importances = pd.Series(model.feature_importances_, index=X.columns)
feat_importances.nlargest(10).plot(kind='barh')
plt.title('Which Features Affect Churn the Most?')
plt.show()
```

```
[ 0.03881588  0.0235565   0.02982391  0.02614242  0.16139256  0.00890209
  0.02337465  0.06285403  0.02425069  0.02732285  0.02485598  0.02474202
  0.02300025  0.02346344  0.09127072  0.03177364  0.07353232  0.13359442
  0.14733163]
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\ensemble\forest.py:246: FutureWarning: The default value of n_estimators wil
l change from 10 in version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

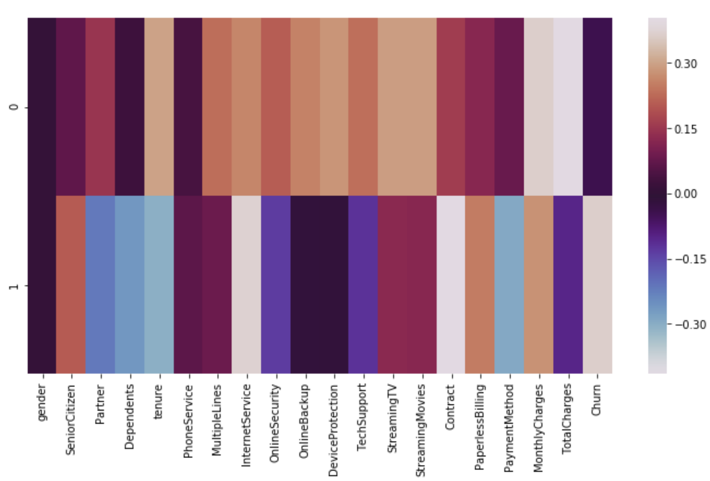


PCA:

PCA helped confirm the most important predictor variables for the next stage of the analysis (predictive analysis.

In [131]:
```
map= pd.DataFrame(pca.components_,columns=df.columns)
plt.figure(figsize=(12,6))
sns.heatmap(map,cmap='twilight')
```

Out[131]: <matplotlib.axes._subplots.AxesSubplot at 0x1bbf17a9c18>



V. References

- Bierly, M. (2016). *10 Useful Python Data Visualization Libraries for Any Discipline.* Retrieved from https://mode.com/blog/python-data-visualization-libraries

- Choudhary, A. (2019). *Principal Component Analysis (PCA) with Python*. Retrieved from https://datascienceplus.com/principal-component-analysis-pca-with-python/

- Data Skunkworks (2018). *Predicting Customer Churn with Python: Logistic Regression, decision trees and random forests. Retrieved from* http://dataskunkworks.com/2018/06/05/predicting-customer-churn-with-python-logistic-regression-decision-trees-and-random-forests/

- Fong, W. (2019). *Excel vs. Python for Data Analysis.* Retrieved from https://datascienceplus.com/principal-component-analysis-pca-with-python/https://xccelerate.co/blog/excel-vs-python-for-data-analysis

- Guru99 (n.d.). *R Vs Python: What's the Difference?* Retrieved from https://www.guru99.com/r-vs-python.html

- Jain, K. (2017). *Python vs. R vs. SAS – which tool should I learn for Data Science?* Retrieved from https://www.analyticsvidhya.com/blog/2017/09/sas-vs-vs-python-tool-learn/

- McKinney, W. (2018). Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython. Sebastopol, CA: O'Reilly Media, Inc.

- Navlani, A. (2018). *Predicting Employee Churn in Python.* Retrieved from https://www.datacamp.com/community/tutorials/predicting-employee-churn-python

- Navlani, A. (2019). *Understanding Logistic Regression in Python.* Retrieved from https://www.datacamp.com/community/tutorials/understanding-logistic-regression-python

- Nelli, F. (2015). Python Data Analytics: Data Analysis and Science Using Pandas, matplotlib, and the Python Programming Language, pp 2-3. Rome, Italy: Apress.

- Nisbet, R., Miner, G., Yale, K., Elder, J. F., & Peterson, A. F. (2018). *Handbook of statistical analysis and data mining applications*. London: Academic Press.

- Tanner, G. (2019). *Introduction to Data Visualization with Python.* Retrieved from https://towardsdatascience.com/introduction-to-data-visualization-in-python-89a54c97fbed

- Terra, J. (2019). *Python for Data Science and Data Analysis.* Retrieved from https://www.simplilearn.com/why-python-is-essential-for-data-analysis-article

- Tufféry, S. (2011). Data Mining and Statistics for Decision Making. West Sussex, UK: Editions Technip.

- UFHealth (n.d.). *Types of Variables*. Retrieved from https://bolt.mph.ufl.edu/6050-6052/preliminaries/types-of-variables/

- Winters, R. (2017). Practical Predictive Analytics. Birmingham, pp 161 UK: Packt.

- Workfront, (2018). *Data Discrimination: The Dark Side of Big Data.* Retrieved from https://www.workfront.com/blog/data-discrimination-the-dark-side-of-big-data