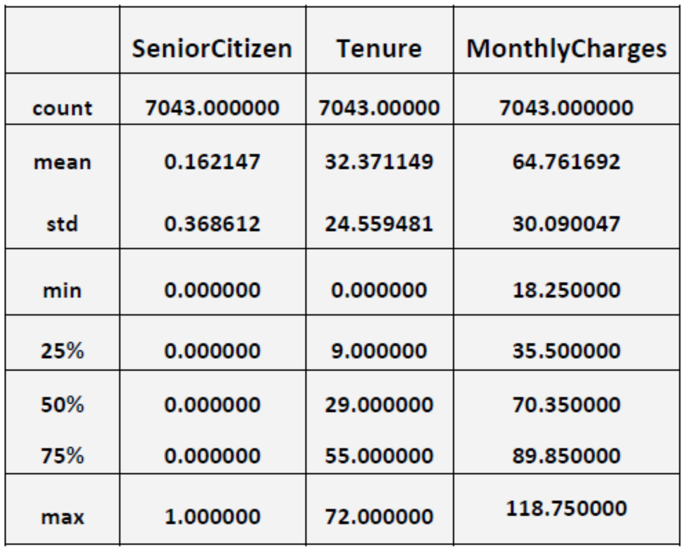
IME672A: DATA MINING AND KNOWLEDGE DISCOVERY
Customer Churn Rate in Telecom Industry

Submitted by: Ayushi Singh, 200258

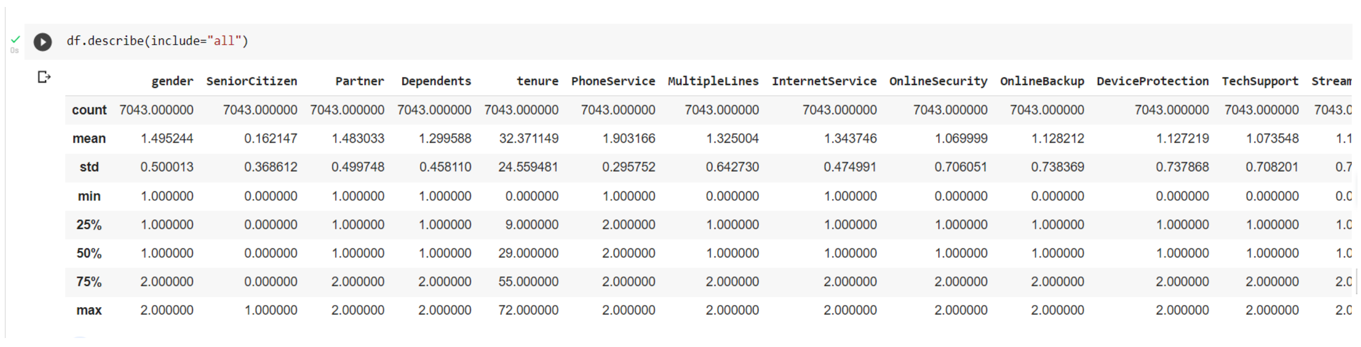
The provided data is from a Telecom Company which is used by them to predict which customers are most likely to switch/quit their services, i.e., to analyse the 'churn rate'. This dataset contains a total of 7,043 customers and 21 attributes, which includes some personal characteristics such as 'Gender', 'Senior Citizen'; services applied, and contract details. Out of the 21 attributes, 3 are numeric and 18 are categorical. I further converted 'TotalCharges' to numeric. 'customerID' does not provide any insights therefore I dropped it.

Out of the entries, 5,174 are active customers and 1,869 are churned, which demonstrates that the dataset is highly unbalanced. Our analysis would revolve around the feature 'Churn'. Also, 11 entries were missing for the feature 'TotalCharges', therefore we can either replace them with the median value or drop these rows since 11 is a very small number for our dataset. I have dropped these rows in my analysis.

First, I analysed the numerical variables statistically, from which I got the following statistical measures.

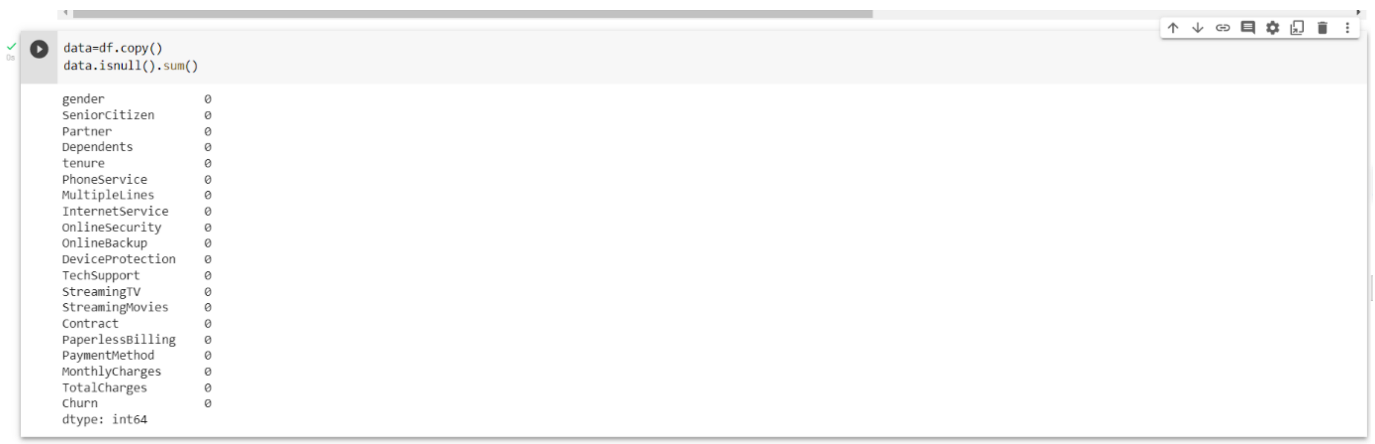


	SeniorCitizen	Tenure	MonthlyCharges
count	7043.000000	7043.00000	7043.000000
mean	0.162147	32.371149	64.761692
std	0.368612	24.559481	30.090047
min	0.000000	0.000000	18.250000
25%	0.000000	9.000000	35.500000
50%	0.000000	29.000000	70.350000
75%	0.000000	55.000000	89.850000
max	1.000000	72.000000	118.750000



```
df.describe(include="all")
```

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtection	TechSupport	Stream
count	7043.000000	7043.000000	7043.000000	7043.000000	7043.000000	7043.000000	7043.000000	7043.000000	7043.000000	7043.000000	7043.000000	7043.000000	7043.0
mean	1.495244	0.162147	1.483033	1.299588	32.371149	1.903166	1.325004	1.343746	1.069999	1.128212	1.127219	1.073548	1.1
std	0.500013	0.368612	0.499748	0.458110	24.559481	0.295752	0.642730	0.474991	0.706051	0.738369	0.737868	0.708201	0.7
min	1.000000	0.000000	1.000000	1.000000	0.000000	1.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.0
25%	1.000000	0.000000	1.000000	1.000000	9.000000	2.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.0
50%	1.000000	0.000000	1.000000	1.000000	29.000000	2.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.0
75%	2.000000	0.000000	2.000000	2.000000	55.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.0
max	2.000000	1.000000	2.000000	2.000000	72.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.0

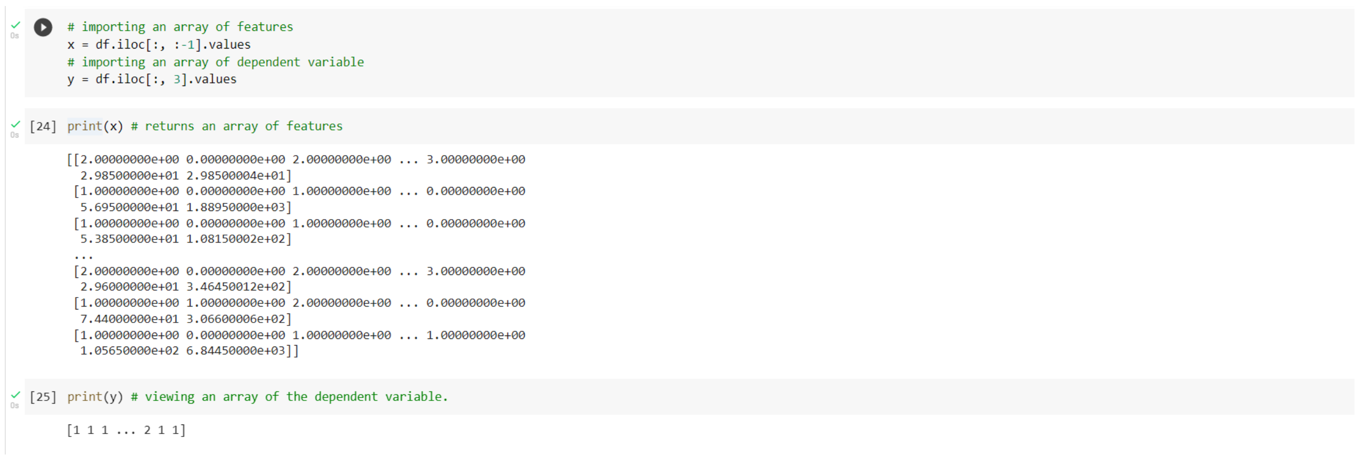


```
data=df.copy()
data.isnull().sum()
```

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

An easy way to check for missing values is to use the method isnull. We will get a data frame with true (1) and false (0) values, so we will sum the values and we can see in which column we have missing values.

Importing an array of features and an array of dependent variable



```
# importing an array of features
x = df.iloc[:, :-1].values
# importing an array of dependent variable
y = df.iloc[:, 3].values
```

```
[24] print(x) # returns an array of features
```

```
[[2.00000000e+00 0.00000000e+00 2.00000000e+00 ... 3.00000000e+00
  2.98500000e+01 2.98500004e+01]
 [1.00000000e+00 0.00000000e+00 1.00000000e+00 ... 0.00000000e+00
  5.69500000e+01 1.88950000e+03]
 [1.00000000e+00 0.00000000e+00 1.00000000e+00 ... 0.00000000e+00
  5.38500000e+01 1.08150002e+02]
 ...
 [2.00000000e+00 0.00000000e+00 2.00000000e+00 ... 3.00000000e+00
  2.96000000e+01 3.46450012e+02]
 [1.00000000e+00 1.00000000e+00 2.00000000e+00 ... 0.00000000e+00
  7.44000000e+01 3.06600006e+02]
 [1.00000000e+00 0.00000000e+00 1.00000000e+00 ... 1.00000000e+00
  1.05650000e+02 6.84450000e+03]]
```

```
[25] print(y) # viewing an array of the dependent variable.
```

```
[1 1 1 ... 2 1 1]
```

```python
[26] from sklearn.impute import SimpleImputer
     imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
```

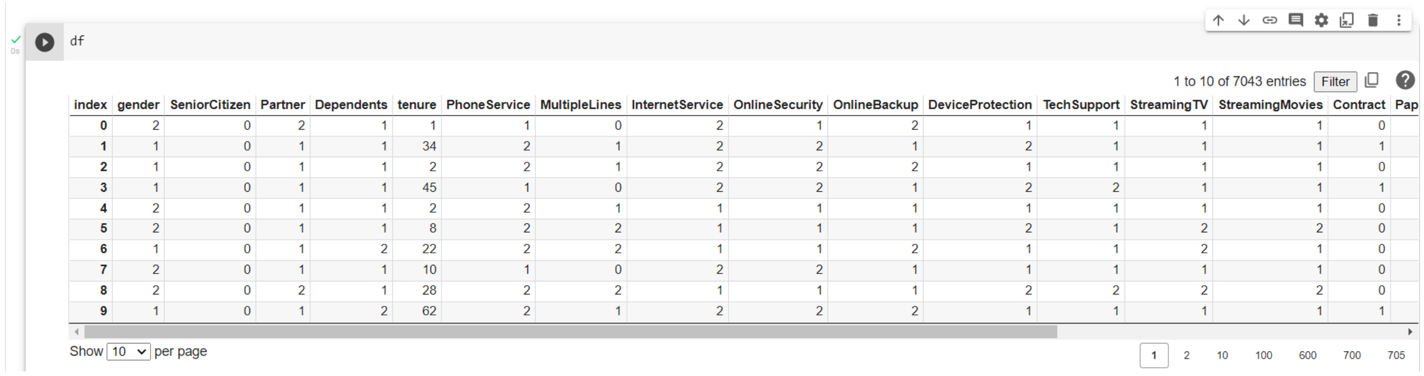
The data we get is rarely homogenous. Sometimes data can be missing, and it needs to be handled so that it does not reduce the performance of our machine learning model. To do this we need to replace the missing data by the Mean or Median of the entire column. For this we will be using the sklearn.preprocessing Library which contains a class called Imputer which will help us in taking care of our missing data.

```python
[27] imputer = imputer.fit(x[:, 1:3])
```

Our object name is imputer. Now we fit the imputer object to our data.

```python
[28] x[:, 1:3] = imputer.transform(x[:, 1:3])
```

Now replacing the missing values with the mean of the column by using transform method.



```python
df
```

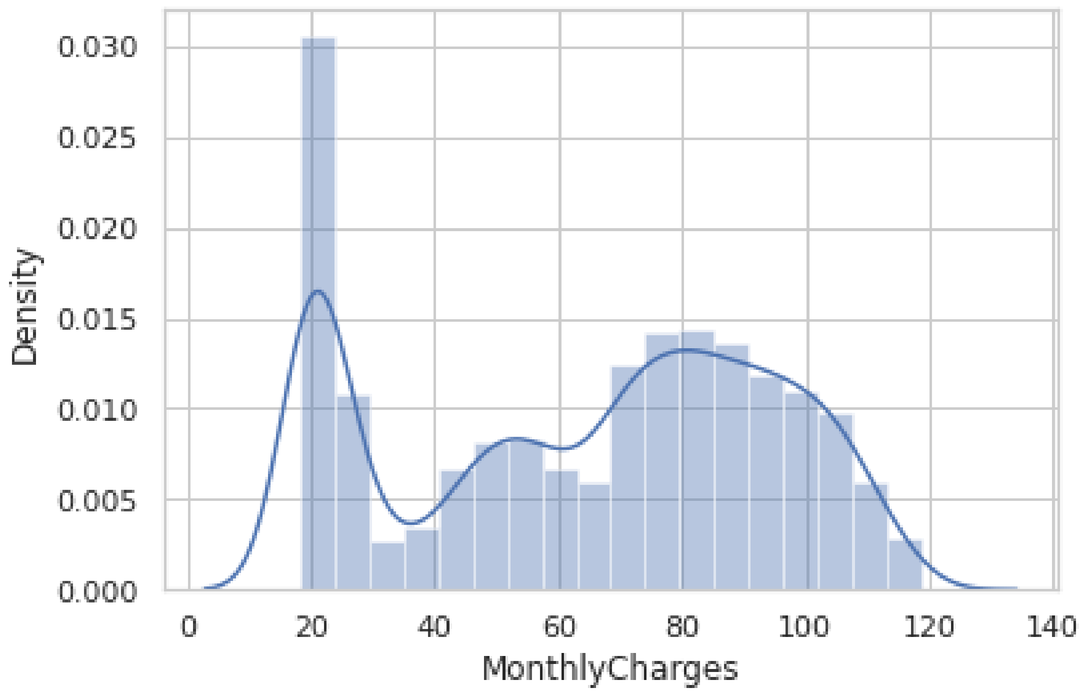
1 to 10 of 7043 entries

index	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtection	TechSupport	StreamingTV	StreamingMovies	Contract	Pap
0	2	0	2	1	1	1	0	2	1	2	1	1	1	1	0	
1	1	0	1	1	34	2	1	2	2	1	2	1	1	1	1	
2	1	0	1	1	2	2	1	2	2	2	1	1	1	1	0	
3	1	0	1	1	45	1	0	2	2	1	2	2	1	1	1	
4	2	0	1	1	2	2	1	1	1	1	1	1	1	1	0	
5	2	0	1	1	8	2	2	1	1	1	2	1	2	2	0	
6	1	0	1	2	22	2	2	1	1	2	1	1	2	1	0	
7	2	0	1	1	10	1	0	2	2	1	1	1	1	1	0	
8	2	0	2	1	28	2	2	1	1	1	2	2	2	2	0	
9	1	0	1	2	62	2	1	2	2	2	1	1	1	1	1	

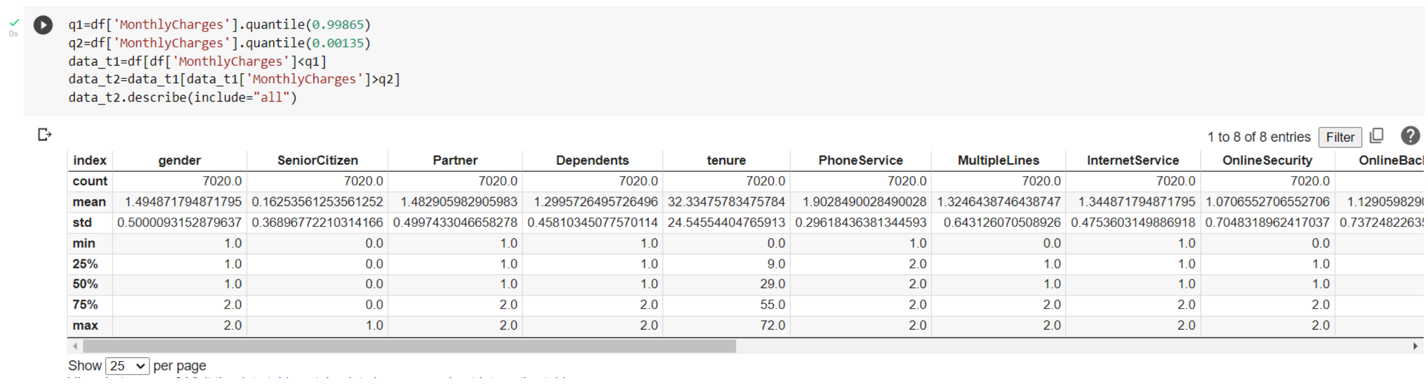
Show 10 per page

1 2 10 100 600 700 705

The minimum value for the MonthlyCharges is 18.25, the maximum is 118.75 and the mean value is approximately 64.76. Also, we can see that 25% of the values are under 35.5 and 75% are under 89.85. So, in this case, we have outliers. Outliers are observations that lie on abnormal distance from other observations in the data and they will affect the regression dramatically. Because of this, the regression will try to place the line closer to these values.



The common rule is to remove everything that is 3 * standard deviations far from the mean. For normally distributed values there is a known rule: 68–95–99.7. Based on this, we can say that the values that are out of the interval [mean — 3*std, mean + 3*std] are outliers and these values can be removed.



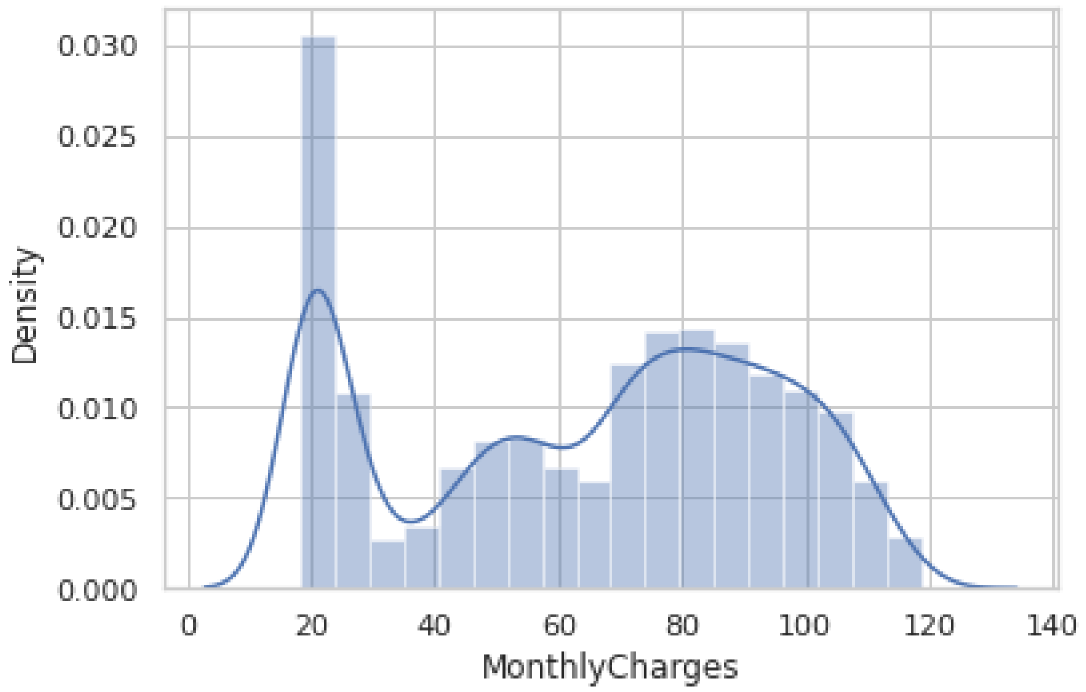
```
q1=df['MonthlyCharges'].quantile(0.99865)
q2=df['MonthlyCharges'].quantile(0.00135)
data_t1=df[df['MonthlyCharges']<q1]
data_t2=data_t1[data_t1['MonthlyCharges']>q2]
data_t2.describe(include="all")
```

1 to 8 of 8 entries

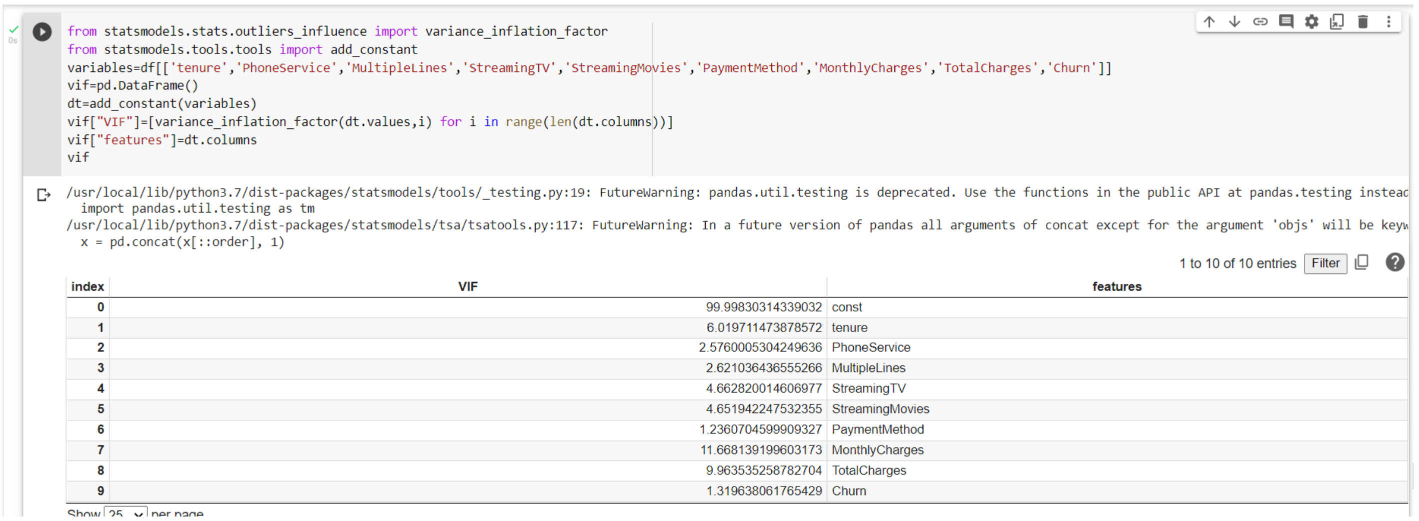
index	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBac
count	7020.0	7020.0	7020.0	7020.0	7020.0	7020.0	7020.0	7020.0	7020.0	
mean	1.494871794871795	0.16253561253561252	1.482905982905983	1.2995726495726496	32.33475783475784	1.9028490028490028	1.3246438746438747	1.344871794871795	1.0706552706552706	1.129059829
std	0.5000093152879637	0.36896772210314166	0.4997433046658278	0.45810345077570114	24.54554404765913	0.29618436381344593	0.643126070508926	0.4753603149886918	0.7048318962417037	0.7372482263
min	1.0	0.0	1.0	1.0	0.0	1.0	0.0	1.0	0.0	
25%	1.0	0.0	1.0	1.0	9.0	2.0	1.0	1.0	1.0	
50%	1.0	0.0	1.0	1.0	29.0	2.0	1.0	1.0	1.0	
75%	2.0	0.0	2.0	2.0	55.0	2.0	2.0	2.0	2.0	
max	2.0	1.0	2.0	2.0	72.0	2.0	2.0	2.0	2.0	

Show 25 per page

The maximum value is still far away from the mean, but it is acceptably closer. We can now plot the distribution and we can see that the data is still distributed the same way, but with fewer outliers.



Multicollinearity exists whenever an independent variable is highly correlated with one or more of the other independent variables in a multiple regression equation. Multicollinearity is a problem because it undermines the statistical significance of an independent variable. When VIF value is equal to 1, there is no multicollinearity at all. Values between 1 and 5 are considered perfectly okay. VIFs greater than 5 represent critical levels of multicollinearity where the coefficients are poorly estimated, and the p-values are questionable.



```python
from statsmodels.stats.outliers_influence import variance_inflation_factor
from statsmodels.tools.tools import add_constant
variables=df[['tenure','PhoneService','MultipleLines','StreamingTV','StreamingMovies','PaymentMethod','MonthlyCharges','TotalCharges','Churn']]
vif=pd.DataFrame()
dt=add_constant(variables)
vif["VIF"]=[variance_inflation_factor(dt.values,i) for i in range(len(dt.columns))]
vif["features"]=dt.columns
vif
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning: pandas.util.testing is deprecated. Use the functions in the public API at pandas.testing insteac
  import pandas.util.testing as tm
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:117: FutureWarning: In a future version of pandas all arguments of concat except for the argument 'objs' will be keyw
  x = pd.concat(x[::order], 1)
```

1 to 10 of 10 entries Filter

index	VIF	features
0	99.99830314339032	const
1	6.019711473878572	tenure
2	2.5760005304249636	PhoneService
3	2.621036436555266	MultipleLines
4	4.662820014606977	StreamingTV
5	4.651942247532355	StreamingMovies
6	1.2360704599909327	PaymentMethod
7	11.668139199603173	MonthlyCharges
8	9.963535258782704	TotalCharges
9	1.319638061765429	Churn

Show 25 per page

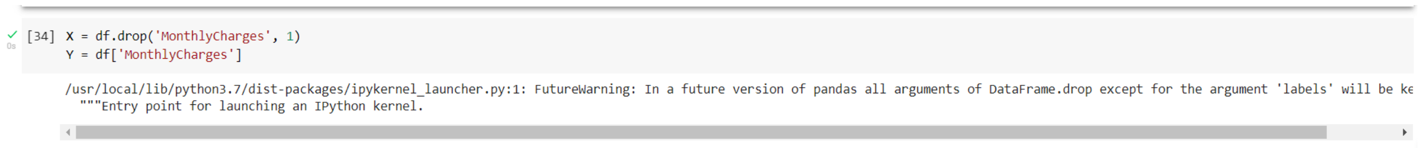
These results show that tenure, MonthlyCharges, TotalCharges and the interaction between them are statistically significant. However, the VIFs indicate that our model has severe multicollinearity for some of the independent variables. Notice that Churn, PaymentMethod has a VIF near 1, which shows that multicollinearity does not affect it and we can trust this coefficient and p-value with no further action. However, the coefficients and p-values for the other terms are suspect!

Principal component analysis, or PCA, is a statistical technique to convert high dimensional data to low dimensional data by selecting the most important features that capture maximum information about the dataset. The features are selected based on variance that they cause in the output.

There are two main advantages of dimensionality reduction with PCA.

- The training time of the algorithms reduces significantly with less number of features.
- It is not always possible to analyse data in high dimensions.

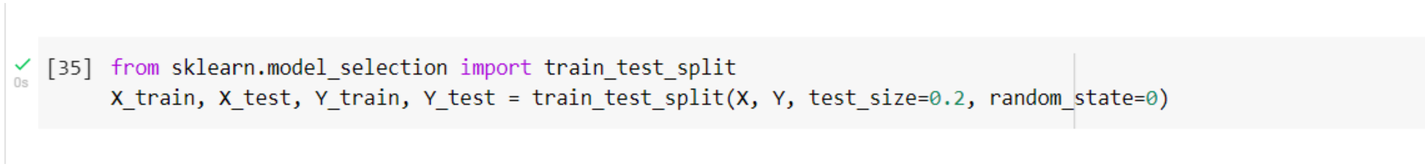
Divide the dataset into a feature set and corresponding labels.



```python
[34] X = df.drop('MonthlyCharges', 1)
     Y = df['MonthlyCharges']
```

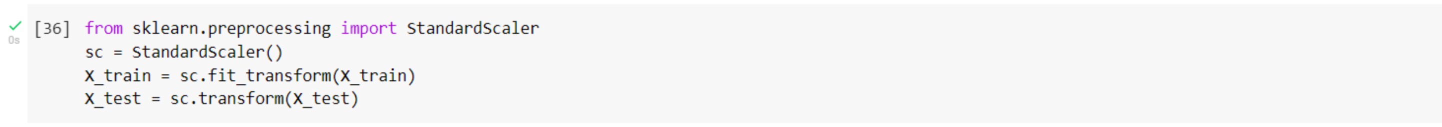
```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: FutureWarning: In a future version of pandas all arguments of DataFrame.drop except for the argument 'labels' will be ke
  """Entry point for launching an IPython kernel.
```

Divide data into training and test sets.



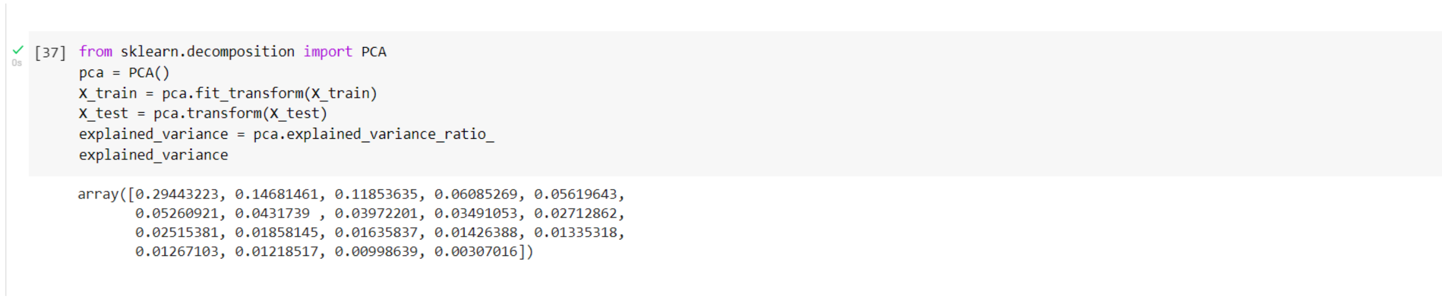
```
[35] from sklearn.model_selection import train_test_split
     X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=0)
```

PCA performs best with a normalized feature set. We will perform standard scalar normalization to normalize our feature set.



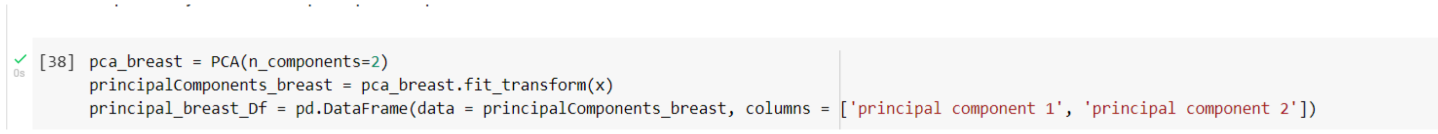
```
[36] from sklearn.preprocessing import StandardScaler
     sc = StandardScaler()
     X_train = sc.fit_transform(X_train)
     X_test = sc.transform(X_test)
```

We create a PCA object named pca. We did not specify the number of components in the constructor. Hence, all the features in the feature set will be returned for both the training and test sets. The PCA class contains explained_variance_ratio_ which returns the variance caused by each of the principal components.



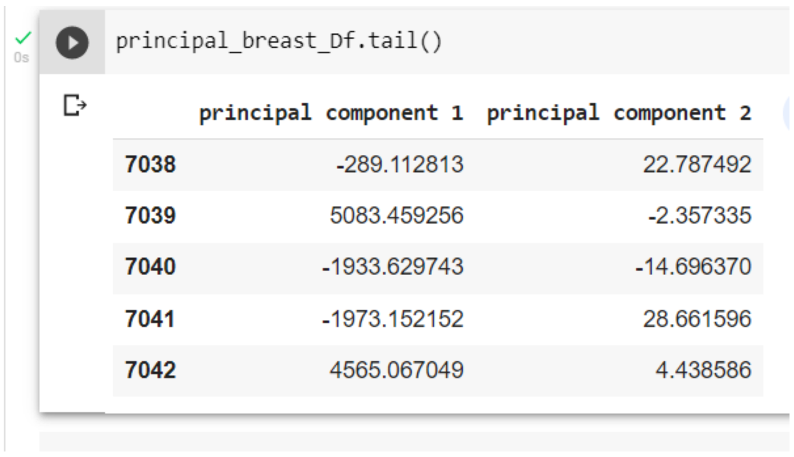
```
[37] from sklearn.decomposition import PCA
     pca = PCA()
     X_train = pca.fit_transform(X_train)
     X_test = pca.transform(X_test)
     explained_variance = pca.explained_variance_ratio_
     explained_variance

     array([0.29443223, 0.14681461, 0.11853635, 0.06085269, 0.05619643,
            0.05260921, 0.0431739 , 0.03972201, 0.03491053, 0.02712862,
            0.02515381, 0.01858145, 0.01635837, 0.01426388, 0.01335318,
            0.01267103, 0.01218517, 0.00998639, 0.00307016])
```

It can be seen here that first principal component is responsible for 29.44% variance. Similarly, the second principal component causes 14.68% variance in the dataset and so on. Collectively we can say that (29.44 + 14.68) 44.12% of the classification information contained in the feature set is captured by the first two principal components.



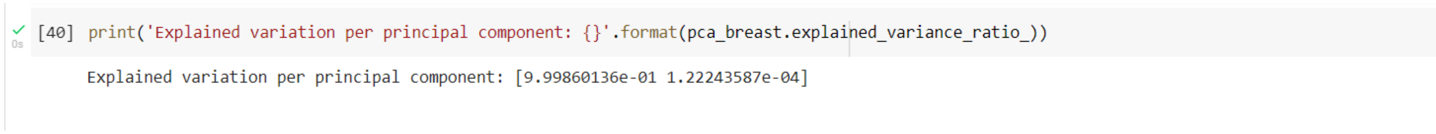
```
[38] pca_breast = PCA(n_components=2)
     principalComponents_breast = pca_breast.fit_transform(x)
     principal_breast_Df = pd.DataFrame(data = principalComponents_breast, columns = ['principal component 1', 'principal component 2'])
```

Now here comes the critical part, we will be projecting the twenty-one - dimensional data to two-dimensional principal components. Once we have the principal components, we can find the explained_variance_ratio. It will provide us with the amount of information or variance each principal component holds after projecting the data to a lower dimensional subspace.



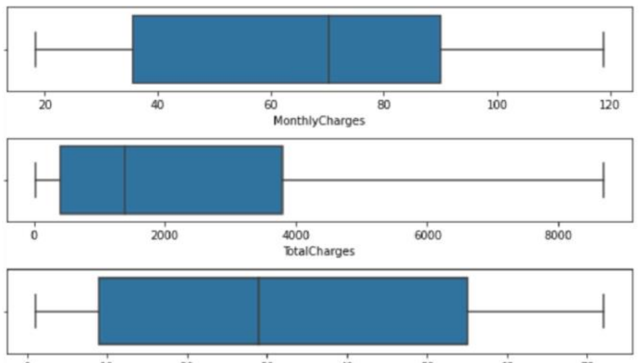
```
principal_breast_Df.tail()
```

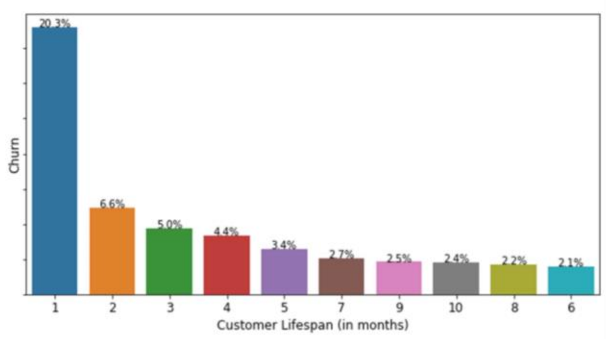
	principal component 1	principal component 2
7038	-289.112813	22.787492
7039	5083.459256	-2.357335
7040	-1933.629743	-14.696370
7041	-1973.152152	28.661596
7042	4565.067049	4.438586



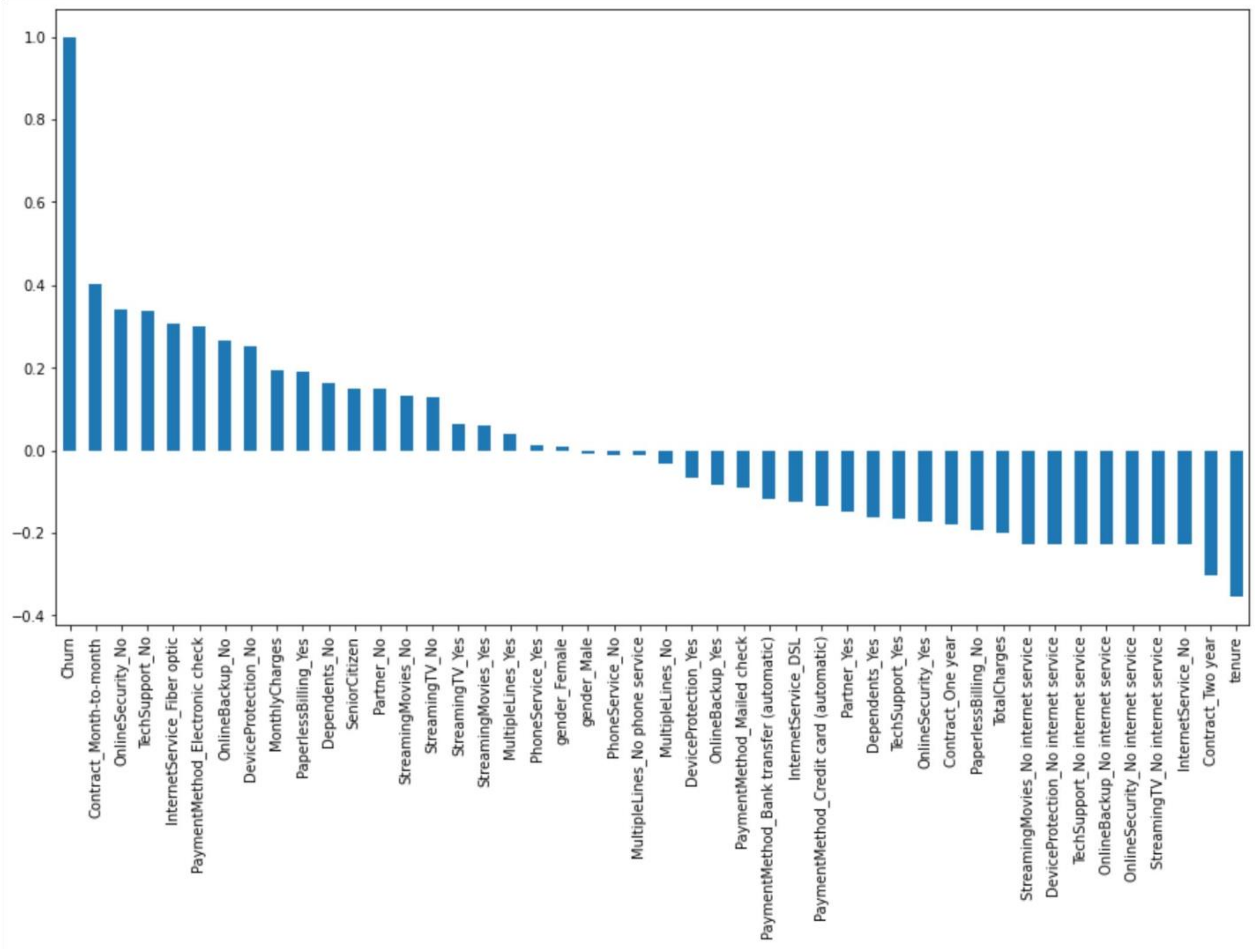
```
[40] print('Explained variation per principal component: {}'.format(pca_breast.explained_variance_ratio_))

     Explained variation per principal component: [9.99860136e-01 1.22243587e-04]
```

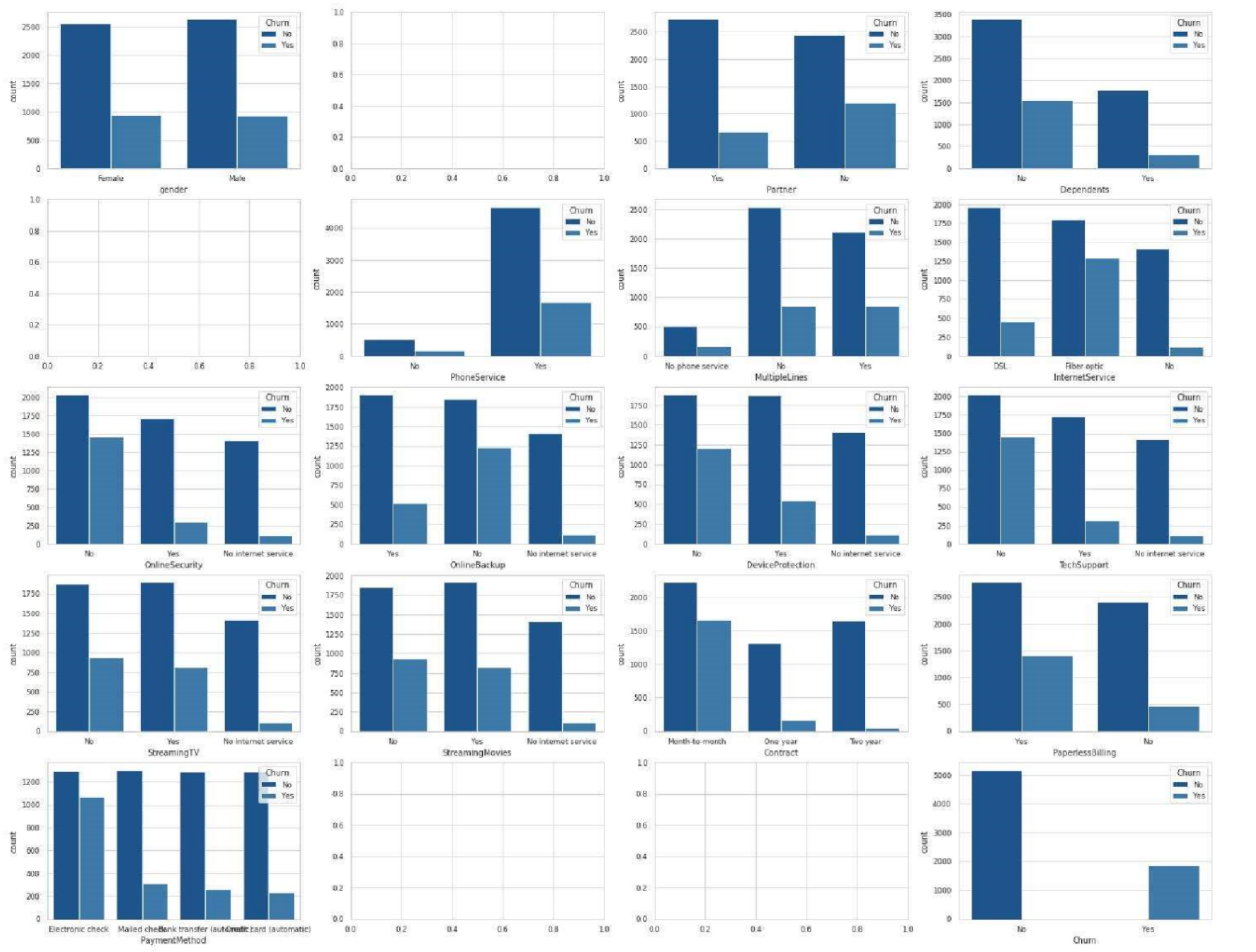
From the above output, we can observe that the principal component 1 holds 99.98% of the information while the principal component 2 holds only 0.012% of the information.





Plotting the correlation of churn with other attributes we get the following plot from which we can observe that service and contract details affect churn rate the most, whereas personal characteristics such as gender does not play a role in determining.

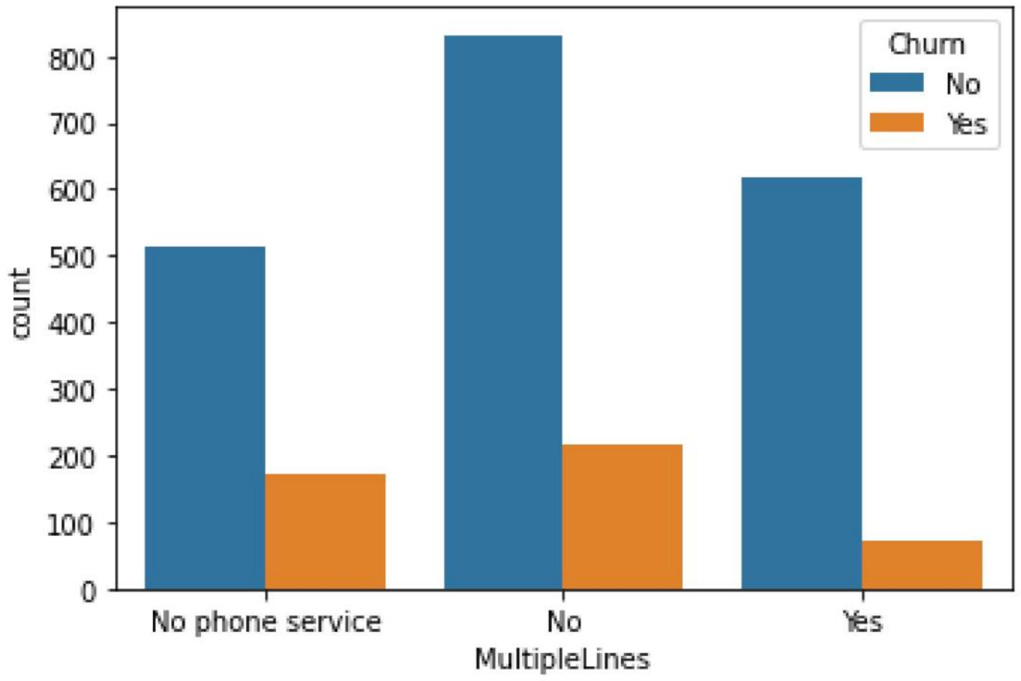


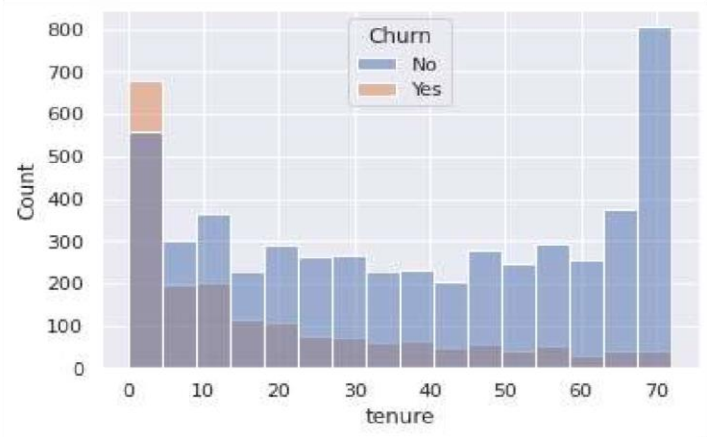


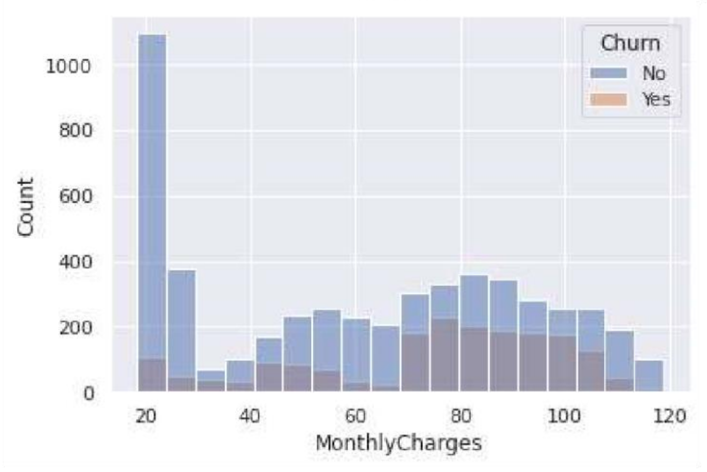
Observation from the above plot:

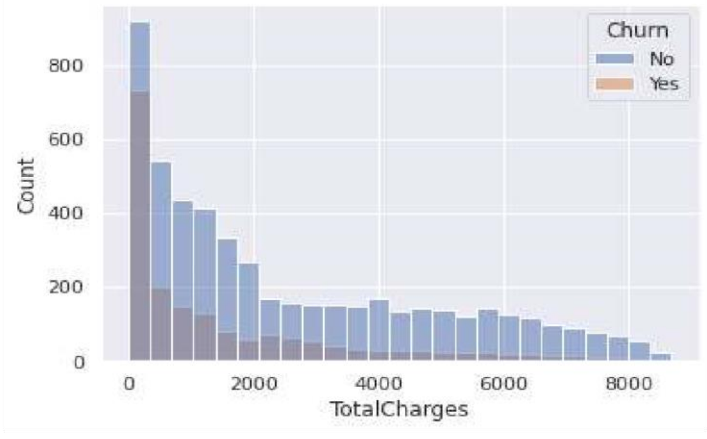
- In the given data, around 71% people **didn't churn**
- Churn distribution is almost similar for both genders
- Electronic payment method has more churn rate as compared to other payment methods
- Month-to-Month contract has higher churn rate than other contracts. It is logical too, cause once you have bought the service for an year or two, people generally do not prefer changing the service due to the hassle it might cause.
- Customer with no Internet service (wherever this value is present) have a very low churn rate. (Maybe because they are not used to the modern technology, and do not prefer changing services)

- People with device protection and Tech-Support (Yes) have less churn rate than people with no device protection or Tech-Support (No), indicating that people are satisfied with the services provided
- Around 650 people didn't take phone service, number is relatively small compared to the total size of the dataset and won't be analyzed.
- People with dependencies or partners have comparatively less churn rate (Around 12.5% for dependencies and 69.5% for partners), than people with no partners or dependencies.
- Customers with no internet services are generally from rural area or want to use their service for just calling or other purpose, thus leaving a very small margin for dissatisfaction and changing the service.
- Another interesting pattern in internet-services is customer with DSL have relatively very less churn rate as compared to Fiber-optics, indicating dissatisfaction with the latter.
- A general patter can be observed: Customers who didn't take services like Online-Security, Online-Backup, Device-Support, Tech-Support, have higher churn rate.



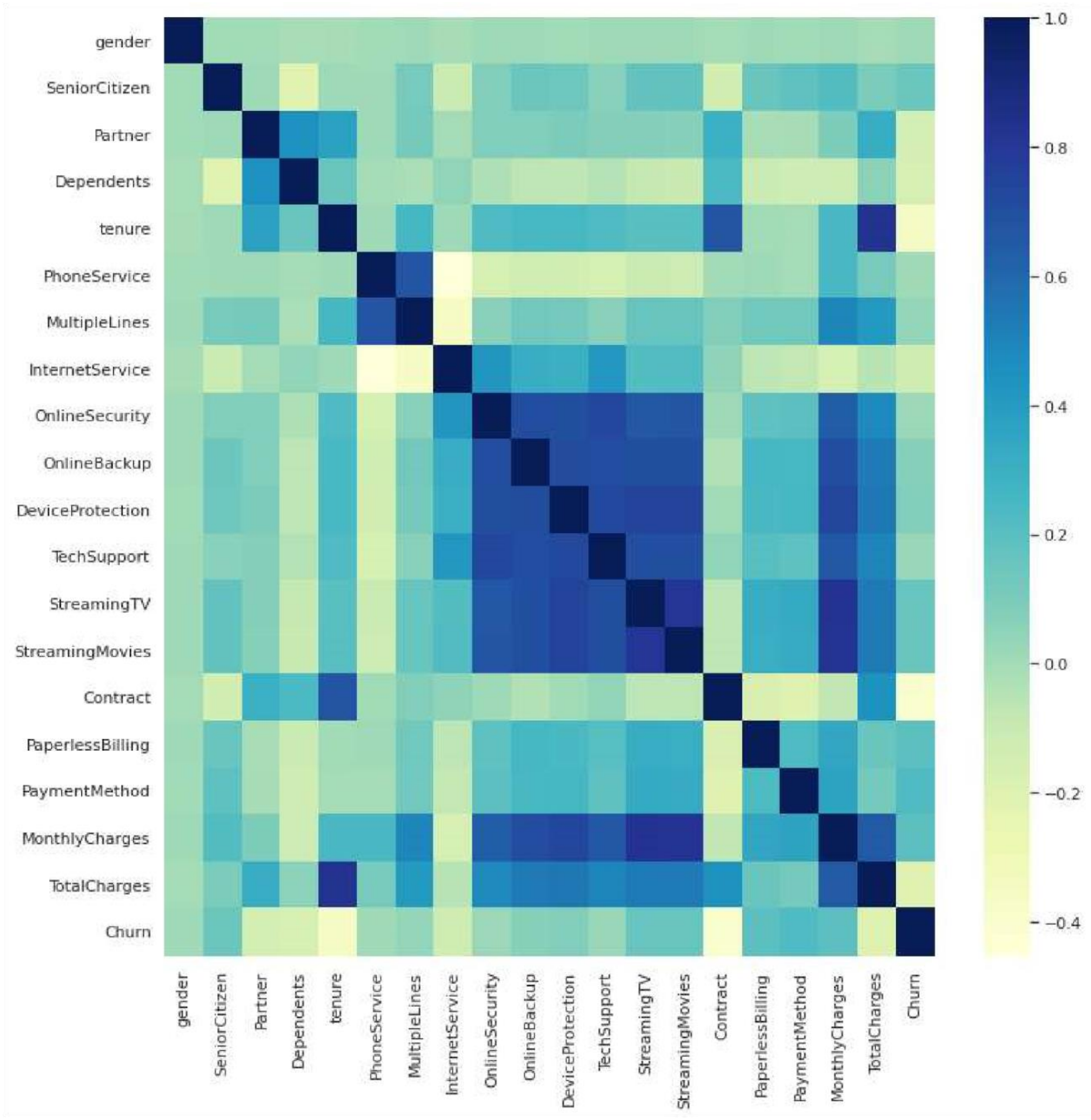






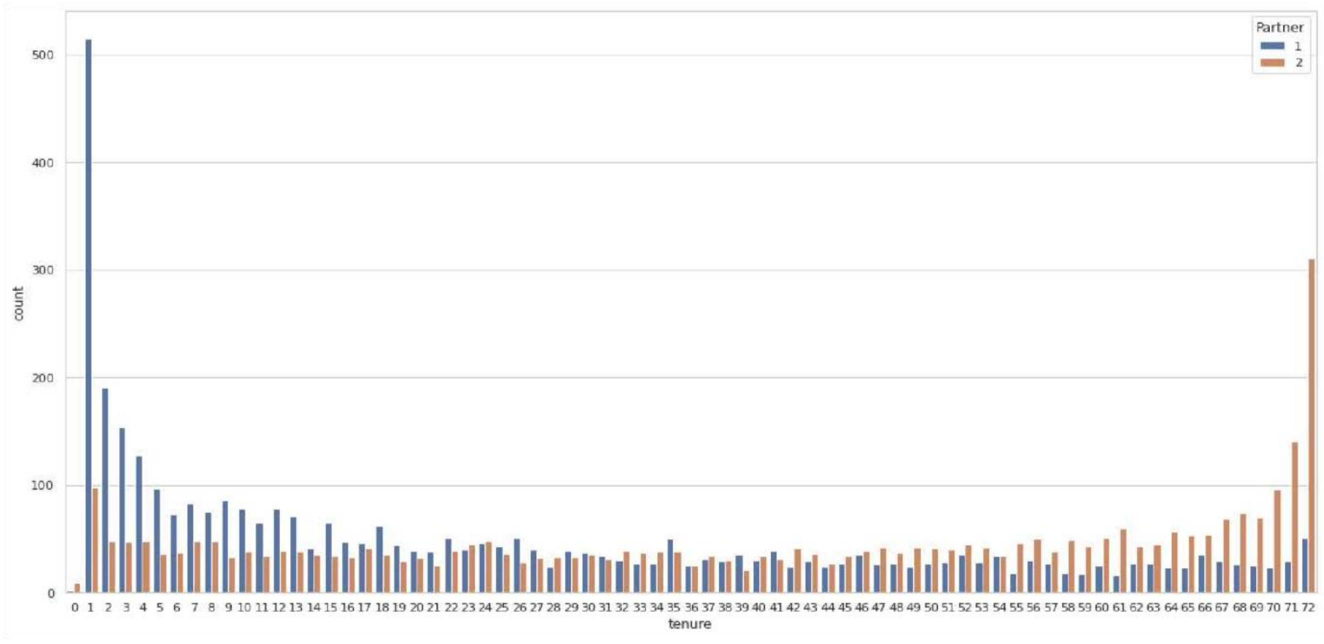
Observations from the above plots:

- People with less value of tenure are more likely to churn.
- Customer's whose monthly bill is less (<= 70) or among the greatest (>=110) are less likely to churn, than customers in the middle range.
- Senior citizens are more likely to churn.

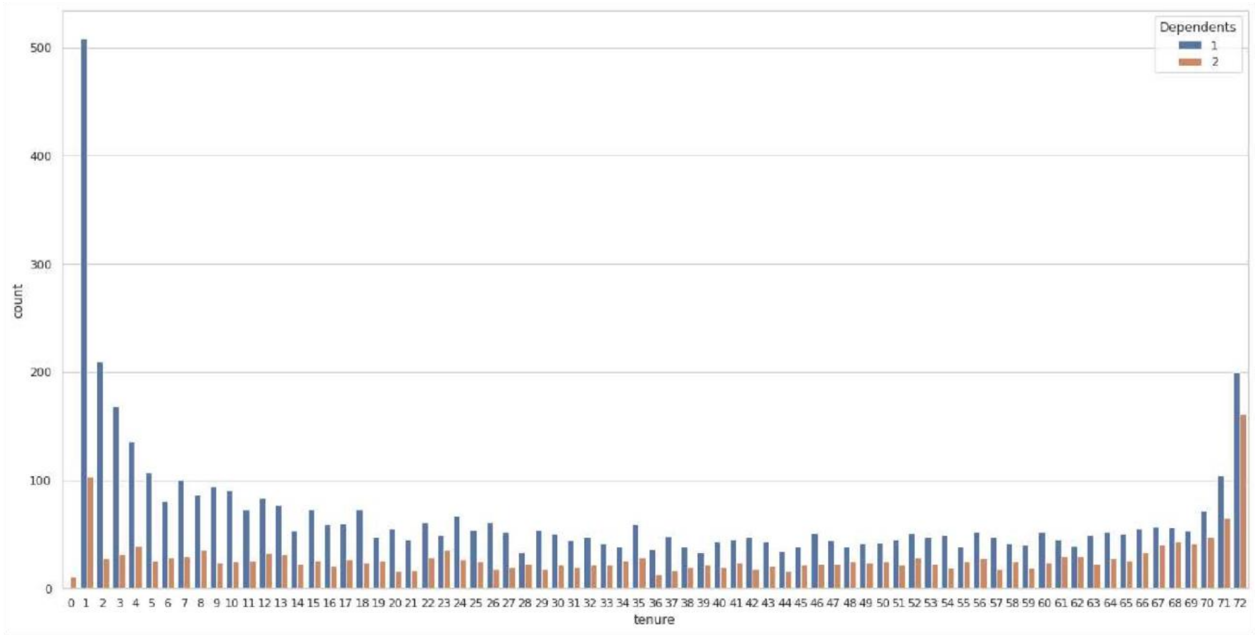


Observations from the above heatmap:

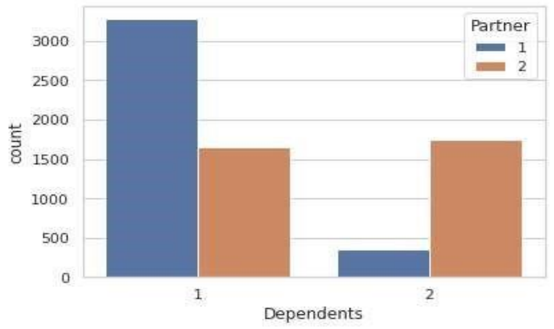
- Contract, Total Charges and Tenure are highly correlated
- Monthly charges increases when customers bought other services (Tech-Support, Online backup, Streaming, etc.)
- Most Senior citizens have no dependents
- All the offered services are correlated, we can just replace them with one attribute Services
- Tenure and Churn are negatively correlated as expected

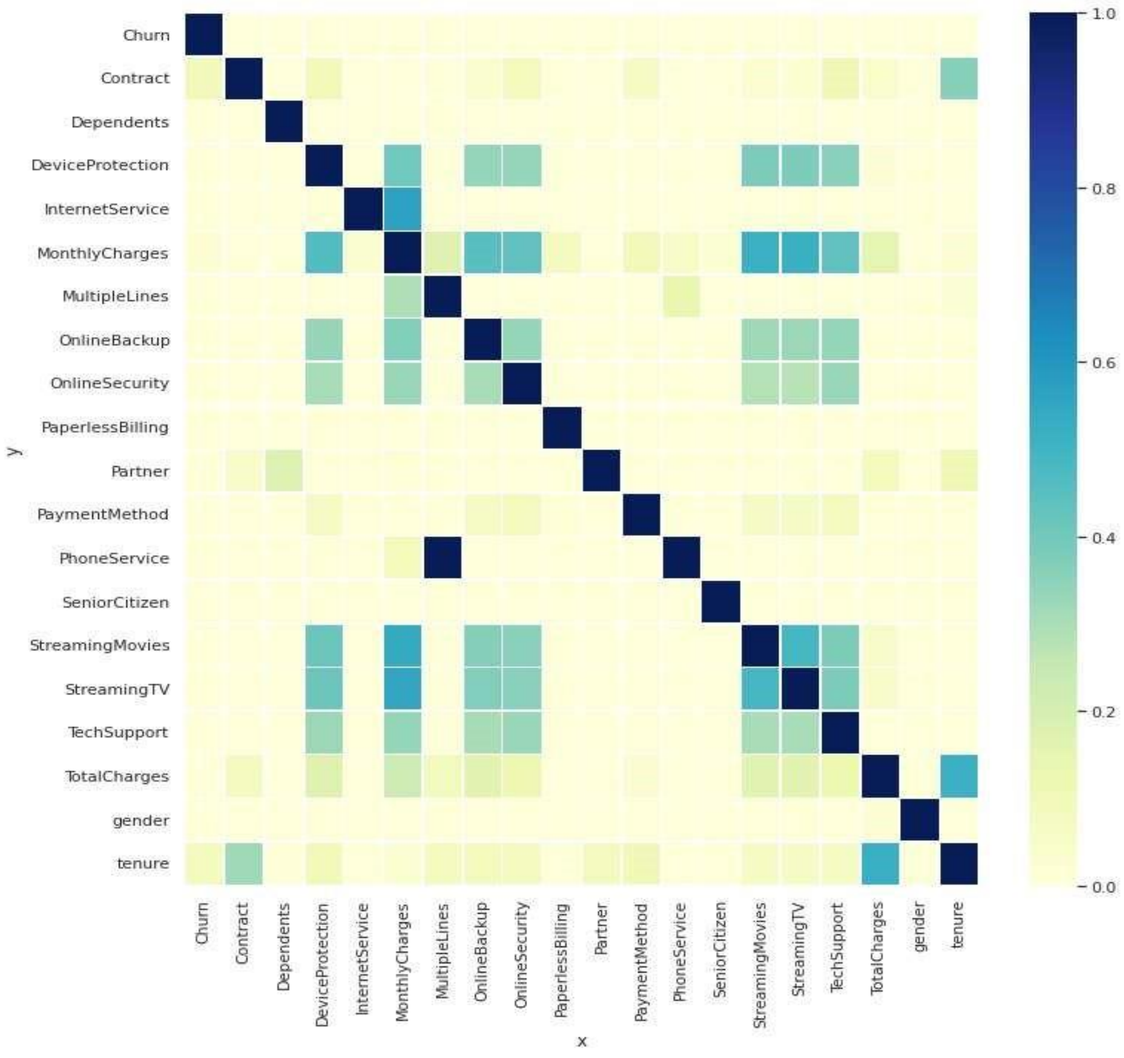


If customers have partner, then their tenure is more likely to be greater than to the ones without a partner.



Customers with no dependent are likely to have less tenure.





Observation from the above plot:

- Services are good predictor of monthly charges.
- Total charges and tenure are good predictors of each other
- Dependents can predict Partners, but the opposite is not True
- Most relation observed are among services provided, charges (monthly and tenure