Interactive Data Visualization

ironhack.school/courses/course-

Lesson Goals

- Learn about interactive visualizations.
- Introduce the plotly and cufflinks libraries.
- Create interactive histograms.
- Create interactive bar charts.
- Create interactive line charts.
- Create interactive scatter plots.
- Create interactive bubble charts.
- Create interactive heatmaps.

Introduction

So far in the program, we have covered a variety of ways to visualize information. We have seen histograms, bar charts, line charts, scatter plots, and even some more advanced visualizations like box plots, violin plots, and scatter matrices.

However, these visualizations have been static and, while perfectly suitable and informative, the world is moving in the direction of interactive visualization. When we add interactivity to our visualizations, one of the benefits we immediately derive is the ability to view values of different data points simply by hovering over them. We also get the ability to zoom in and out as well as remove and add back different series of data points, which allows us to explore our data in ways that we simply could not with static visualizations.

In the past, in order to construct interactive visualizations in the browser, you had to learn D3 or JavaScript. These days, there are Python libraries that sit on top of D3 which give us the ability to generate interactive visualizations directly from Python. In this lesson, we will be using a couple of these new libraries (Plotly and cufflinks) to construct our interactive visualizations. Plotly is an open source visualization library, and cufflinks is a wrapper around Plotly that allows you to easily produce interactive visualizations - often with a single line of code!

Before we jump into the lesson, let's make sure both of these libraries are installed.

```
$ pip install plotly
$ pip install cufflinks
```

Let's also import the libraries we are going to need for this lesson and set cufflinks to offline mode so that we can just visualize our interactive charts in the Jupyter Notebook. If we did not specify, it would push each visualization created up to your Plotly account.

```
import plotly.plotly as py
import cufflinks as cf
import pandas as pd

cf.go_offline()
```

The Data

The data set we will be using for this lesson is the Telco Customer Churn data set from Kaggle. Let's go ahead and import the data set using the Pandas `read_csv` method and take a look at what columns we have to work with.

```
df = pd.read_csv('./data/churn.csv')

df.columns
```

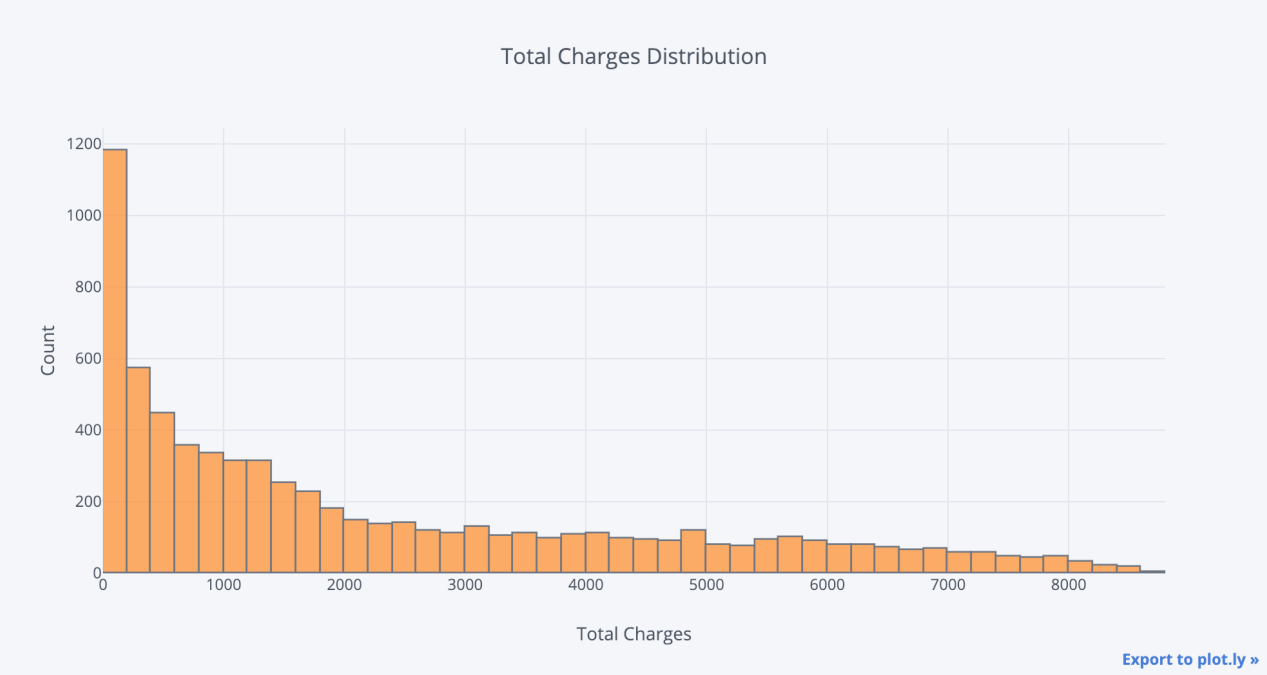
```
Index(['customerID', 'gender', 'SeniorCitizen', 'Partner', 'Dependents',
       'tenure', 'PhoneService', 'MultipleLines', 'InternetService',
       'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport',
       'StreamingTV', 'StreamingMovies', 'Contract', 'PaperlessBilling',
       'PaymentMethod', 'MonthlyCharges', 'TotalCharges', 'Churn',
       'MonthLevel', 'TotalLevel', 'TenureLevel', 'ChurnBinary'],
      dtype='object')
```

Interactive Histograms

The first type of interactive visualization we will be generating with Plotly and cufflinks is a histogram. As mentioned in prior lessons, histograms help us visualize how the values in a specific field are distributed. For example, let's take the TotalCharges column of our data set and create a basic interactive histogram for it.

To do that we would call the `iplot` method and pass "hist" to the `kind` argument along with what we want the x and y axis to be named (to the xTitle and yTitle arguments respectively) and what title we want displayed on our visualization to the title argument. These last three arguments are present in every interactive chart we will generate.

```
data = df['TotalCharges']

data.iplot(kind='hist', xTitle='Total Charges', yTitle='Count',
           title='Total Charges Distribution')
```

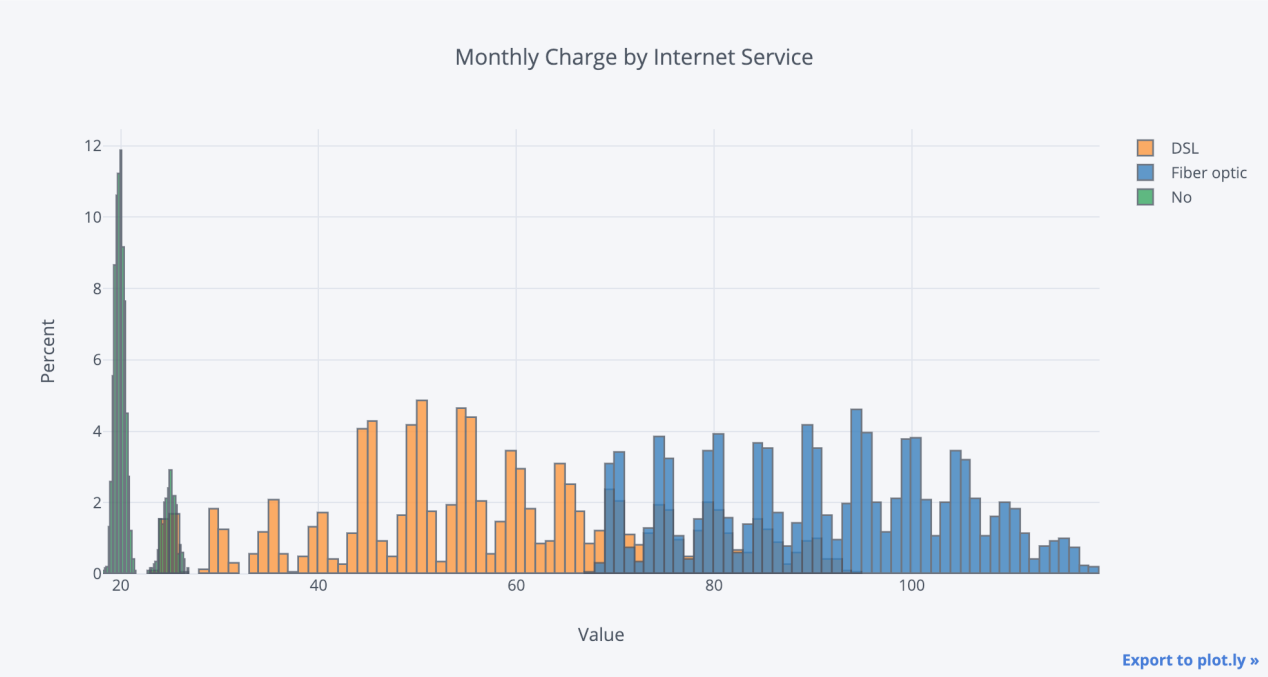


The resulting plot should look like the image above, only yours should be interactive! When you hover over each bar of the histogram, it should display the number of records that fall into the bin that bar represents.

Plotly and cufflinks allow us to do a few other interesting things with histograms. In the example below, we are creating a pivot table from our original data frame that has the different Internet Services represented in separate columns and Monthly Charges by customer as the values in the table.

We then create a histogram from this data set and, by default, it creates overlay histograms representing the Monthly Charge ranges for all three Internet Services. Note also that we have added a new `histnorm` argument and have passed it the value "percent." What this does is converts the y axis to show the percentage instead of the number of records each bin makes up.

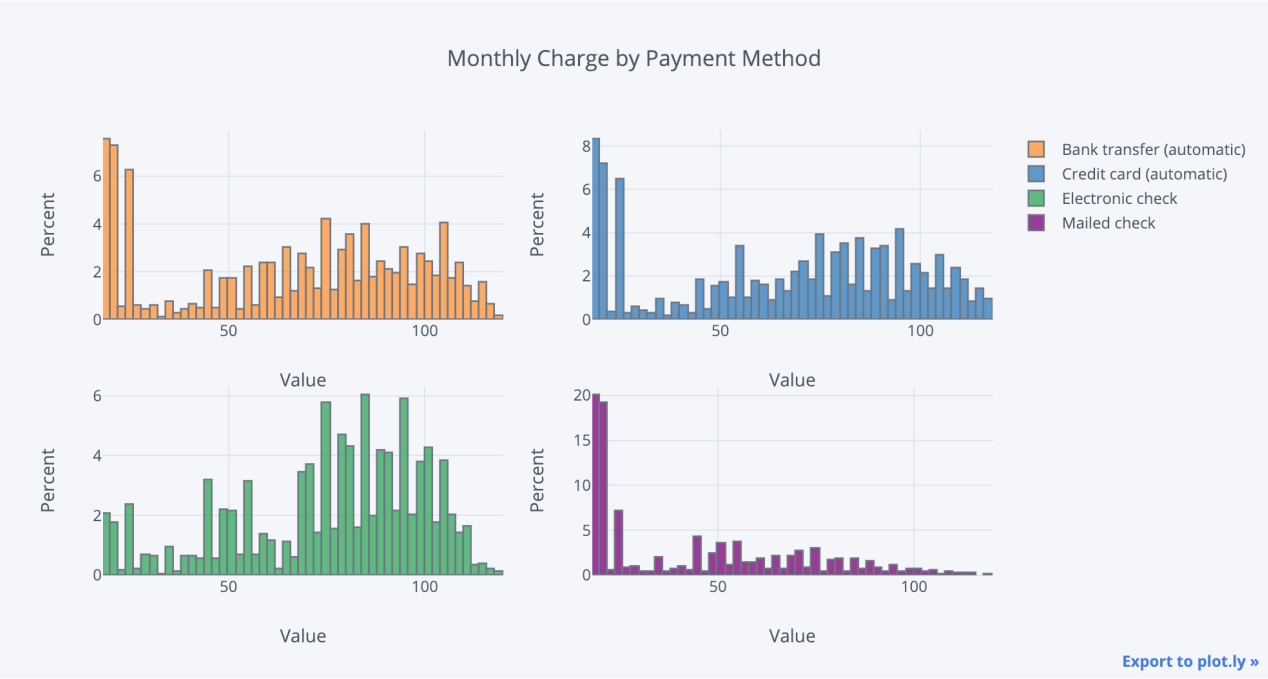
```
data = df.pivot_table(values='MonthlyCharges', columns='InternetService',
                      index='customerID', aggfunc='sum')

data.iplot(kind='hist', histnorm='percent', xTitle='Value',
           yTitle='Percent', title='Monthly Charge by Internet Service')
```



In addition to being able to hover over components in the visualization and see their values, another useful feature of these types of interactive visualizations is that you can filter categories when there are more than one of them and they are included in a legend. For example, to remove the No category, simply click on it. It disappears and the visualization automatically rescales itself according to the remaining data points. If you click it again, it appears in the visualization again.

Another useful feature that is great for histograms is the `subplot` feature. When this value is included and set to True, instead of overlapping histograms, you get back a grid of histograms. In the example below, we create a pivot table where the columns represent different payment methods and the values represent monthly charges by customer. When we call the `iplot` method to generate histograms from this data and include the `subplot` argument, it returns four separate histograms - one for each type of payment method.

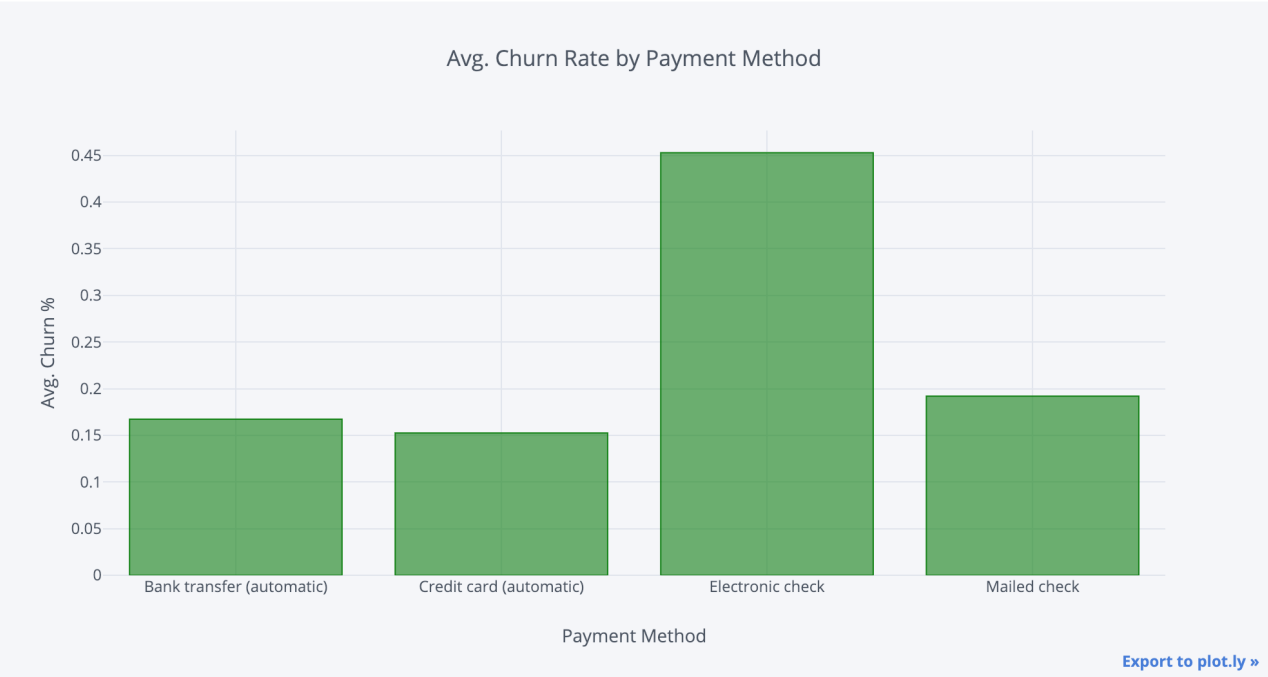
```
data = df.pivot_table(values='MonthlyCharges', columns='PaymentMethod', index='customerID',
aggfunc='sum')

data.iplot(kind='hist', histnorm='percent', xTitle='Value', subplots=True,
yTitle='Percent', title='Monthly Charge by Payment Method')
```



Interactive Bar Charts

In addition to histograms, another useful type of interactive visualization is the interactive bar chart. Below is a basic example where we calculate the average churn rate for each payment method and then create a bar chart by setting the `kind` argument to "bar."

```
data = df.groupby('PaymentMethod', as_index=False).agg({'ChurnBinary':'mean'})

data.iplot(kind='bar', x='PaymentMethod', xTitle='Payment Method', color='green',
           yTitle='Avg. Churn %', title='Avg. Churn Rate by Payment Method')
```



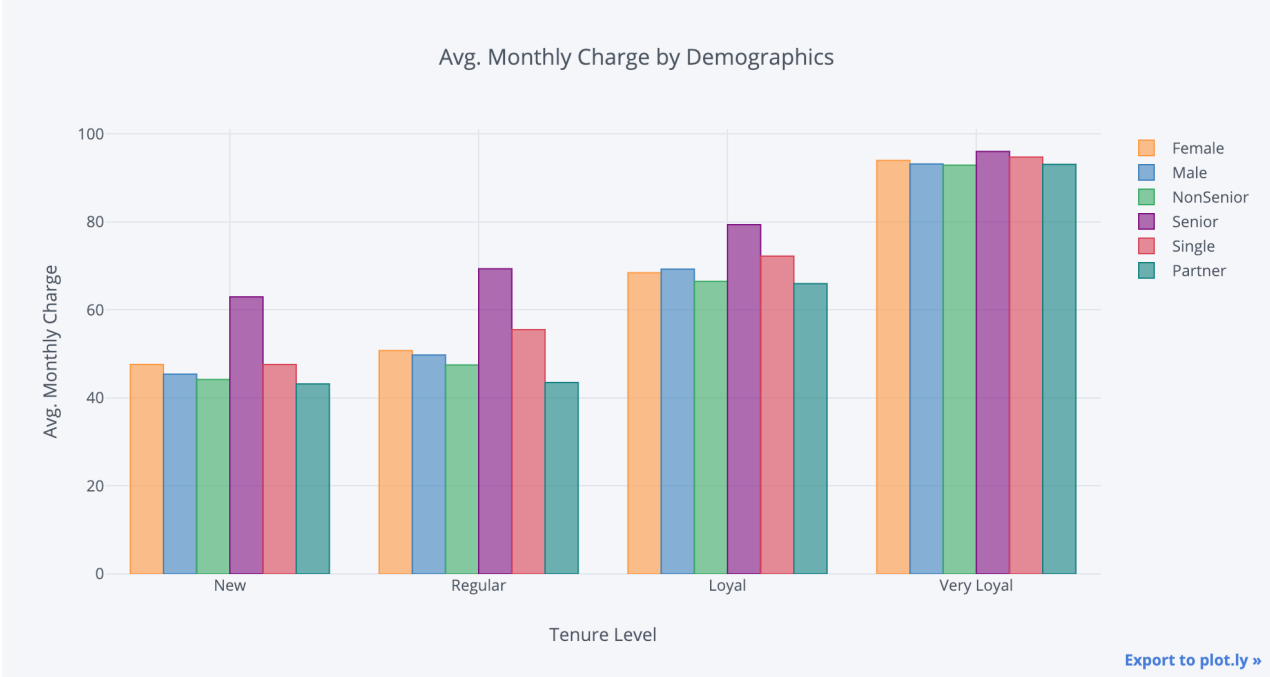
Note that we also included a color argument to specify that we would like the bars to be green. Feel free to try changing it to whatever your favorite color is.

Basic bar charts are very helpful, but sometimes the need arises to create more complex bar charts, such as when we need to see bars in groups. To set this up with the data set we are currently working with, below we are creating three sets of pivot tables that each calculate average monthly charges by tenure level but for different demographic variables (gender, senior citizen, and partner). We combine all the three pivot tables together (concat columns) and sort according to the intuitive order of customer tenure level - new customers, regular customers, loyal customers, and very loyal customers.

```
gender = df.pivot_table(values='MonthlyCharges', columns='gender',
                        index='TenureLevel', aggfunc='mean')

senior = df.pivot_table(values='MonthlyCharges', columns='SeniorCitizen',
                        index='TenureLevel', aggfunc='mean')

partner = df.pivot_table(values='MonthlyCharges', columns='Partner',
                         index='TenureLevel', aggfunc='mean')

data = pd.concat([gender, senior, partner], axis=1)
data.columns = ['Female', 'Male', 'NonSenior', 'Senior', 'Single', 'Partner']
data = data.reset_index()

custom_dict = {'New': 0, 'Regular': 1, 'Loyal': 2, 'Very Loyal' : 3}
data = data.iloc[data['TenureLevel'].map(custom_dict).argsort()]
```

Now that are data is formatted how we need it to be, we can simply call `iplot` to generate a bar chart that will create a bar for each of the demographic columns, group them by tenure level, and even display the tenure levels in the order we assigned them. This lets us see the overall increase in monthly charges from new customers to very loyal customers as well as the variations across demographics within each one.

```
data.iplot(kind='bar', x='TenureLevel', xTitle='Tenure Level',
           yTitle='Avg. Monthly Charge', title='Avg. Monthly Charge by Demographics')
```

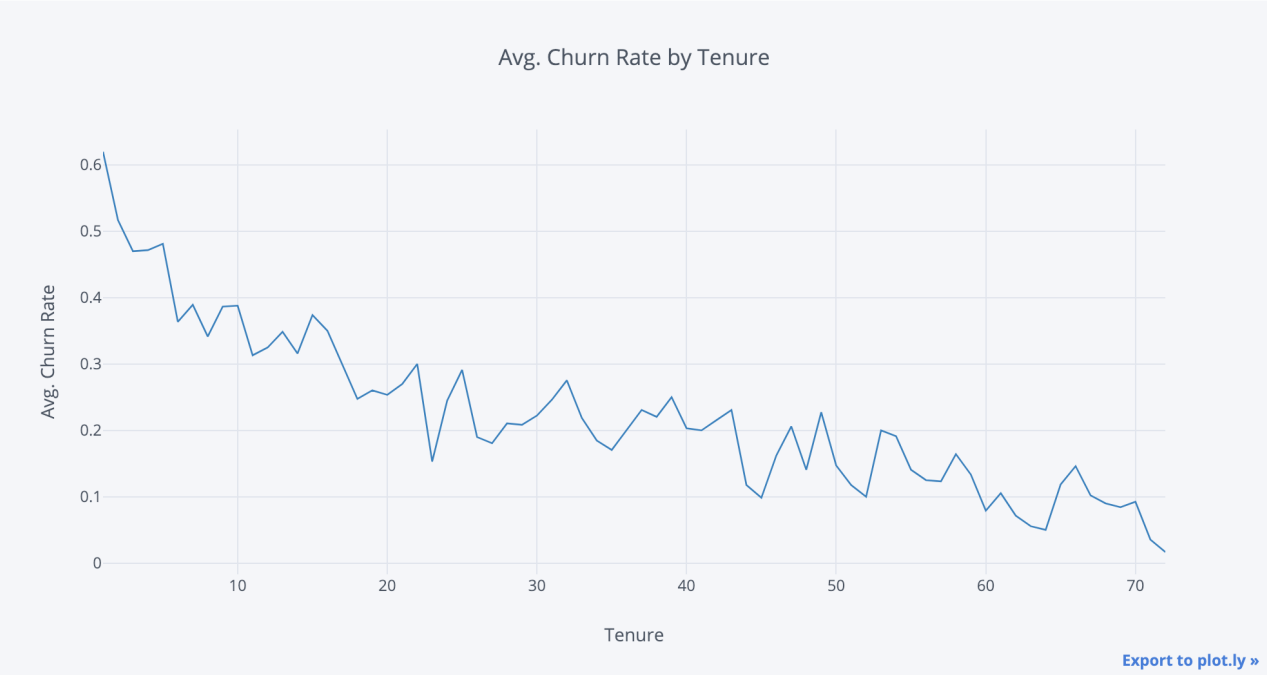


Interactive Line Charts

In addition to histograms and bar charts, another very useful interactive visualization is the line chart. Interactive line charts allow us to not only visualize how a variable changes, but it can also let us hover over and find out the exact values even when there are a lot of points across the x axis.

In the example below, we are creating calculating the average churn rate by tenure and then creating a basic single line chart that shows the overall decrease in churn rate as tenure increases. Note that all we had to do was pass "line" to the `kind` argument and then the appropriate fields, titles, and colors to their respective arguments.

```
data = df.groupby('tenure', as_index=False).agg({'ChurnBinary':'mean'})

data.iplot(kind='line', x='tenure', xTitle='Tenure', color='blue',
           yTitle='Avg. Churn Rate', title='Avg. Churn Rate by Tenure')
```

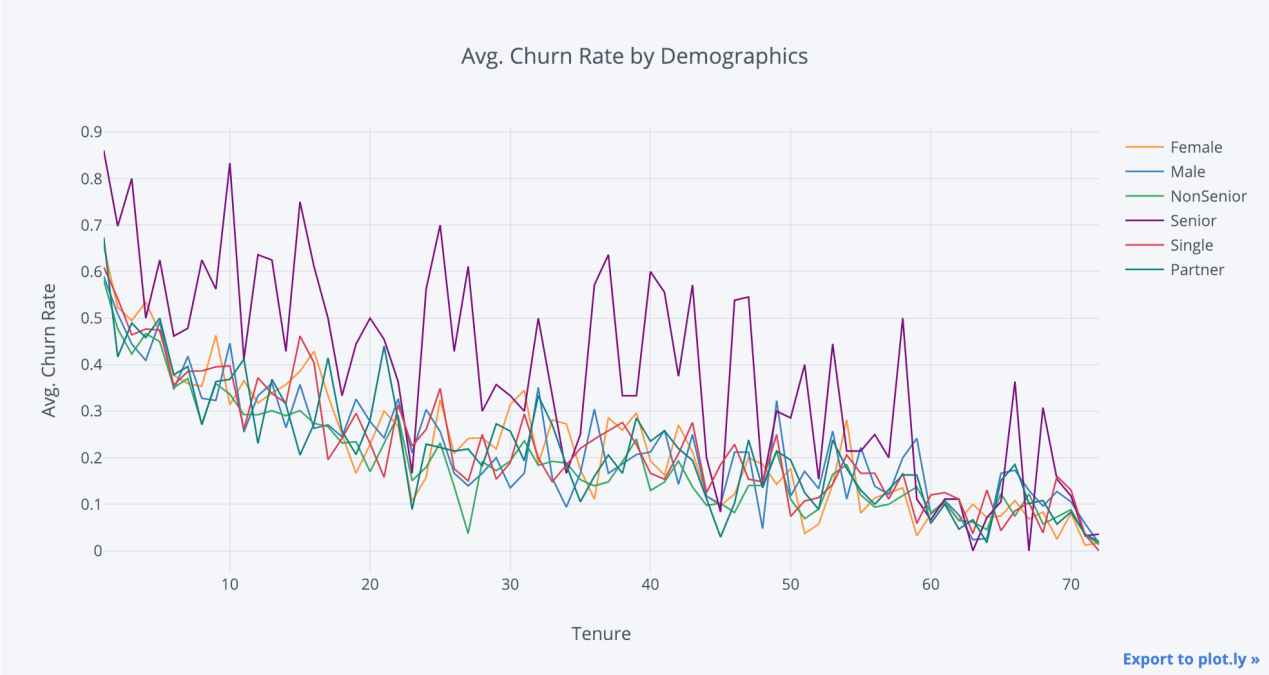


This gives us a sense of what happens with aggregate churn rates, but how does that differ by demographics? To find out, we can take the same approach as we did with our grouped bar chart. Below, we create three pivot tables again and combine them.

```
gender = df.pivot_table(values='ChurnBinary', columns='gender',
                        index='tenure', aggfunc='mean')

senior = df.pivot_table(values='ChurnBinary', columns='SeniorCitizen',
                        index='tenure', aggfunc='mean')

partner = df.pivot_table(values='ChurnBinary', columns='Partner',
                         index='tenure', aggfunc='mean')

data = pd.concat([gender, senior, partner], axis=1)
data.columns = ['Female', 'Male', 'NonSenior', 'Senior', 'Single', 'Partner']
data = data.reset_index()
```

We now have the data in the format we need to create an interactive multi-line chart. To do that, we call the `iplot` method again and pass the appropriate values to each argument.

```
data.iplot(kind='line', x='tenure', xTitle='Tenure',
        yTitle='Avg. Churn Rate', title='Avg. Churn Rate by Demographics')
```



Interactive Scatter Plots

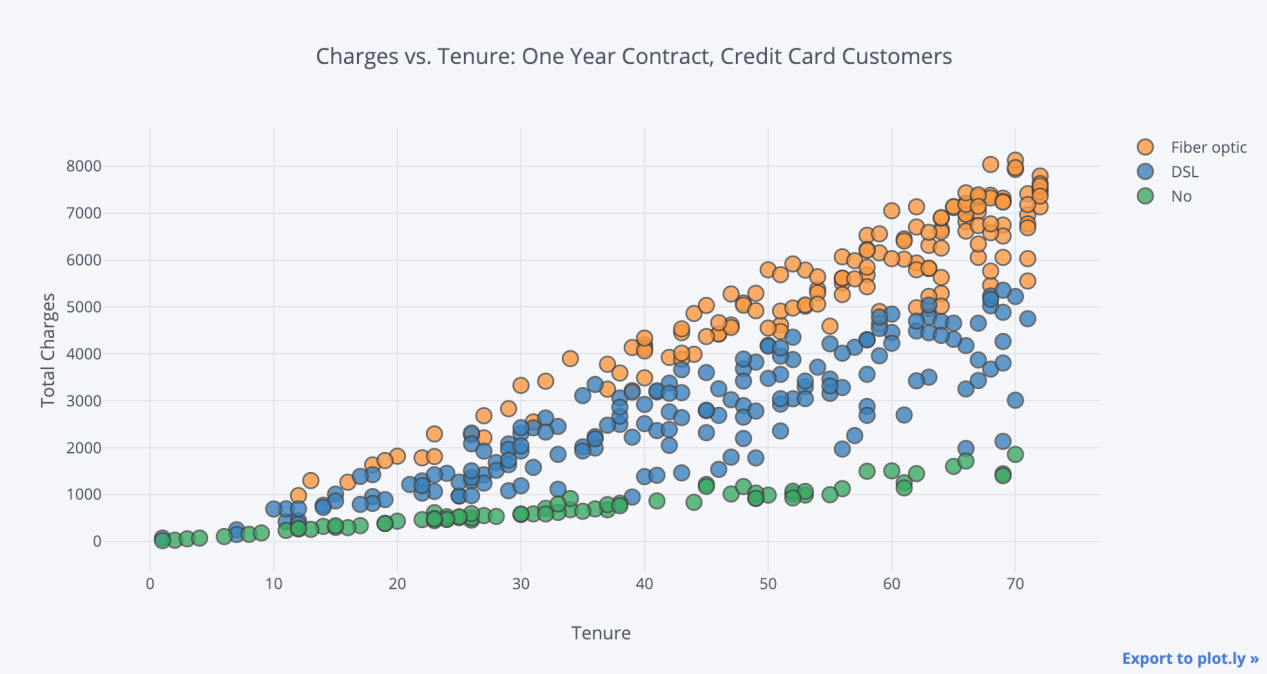
Another useful type of interactive visualization is the scatter plot. Like with line charts, one of the valuable features of interactivity with scatter plots is the ability to hover over each of the points and see their value. It is often also useful to apply the filtering by group like we did with the histograms earlier. However, one drawback about creating interactive scatter plots is that scatter plots tend to have a lot of data points, so creating interactive ones can quickly get computationally intensive and the results can look a bit cluttered.

In the example below, we capitalize on the advantages mentioned above while avoiding the disadvantages by filtering our data set for a subset that we would like to investigate (customers on one year contracts that pay with credit cards).

```
data = df[(df['Contract']=='One year') & (df['PaymentMethod']=='Credit card (automatic)')]
```

We then call the `iplot` method to generate our interactive visualization. Note that to create a scatter plot, we did not need to pass anything to the `kind` argument but we did need to specify which fields would be represented on both the x and y axes. We also added a new `categories` argument that we passed the type of Internet Service to so that we can distinguish between those groups in our visualization.

```
data.iplot(x='tenure', y='TotalCharges', categories='InternetService',
        xTitle='Tenure', yTitle='Total Charges',
        title='Charges vs. Tenure: One Year Contract, Credit Card Customers')
```



Due to our filtering of the data, the resulting visualization contains a reasonable number of data points and allows us to clearly distinguish between the different groups in our data.

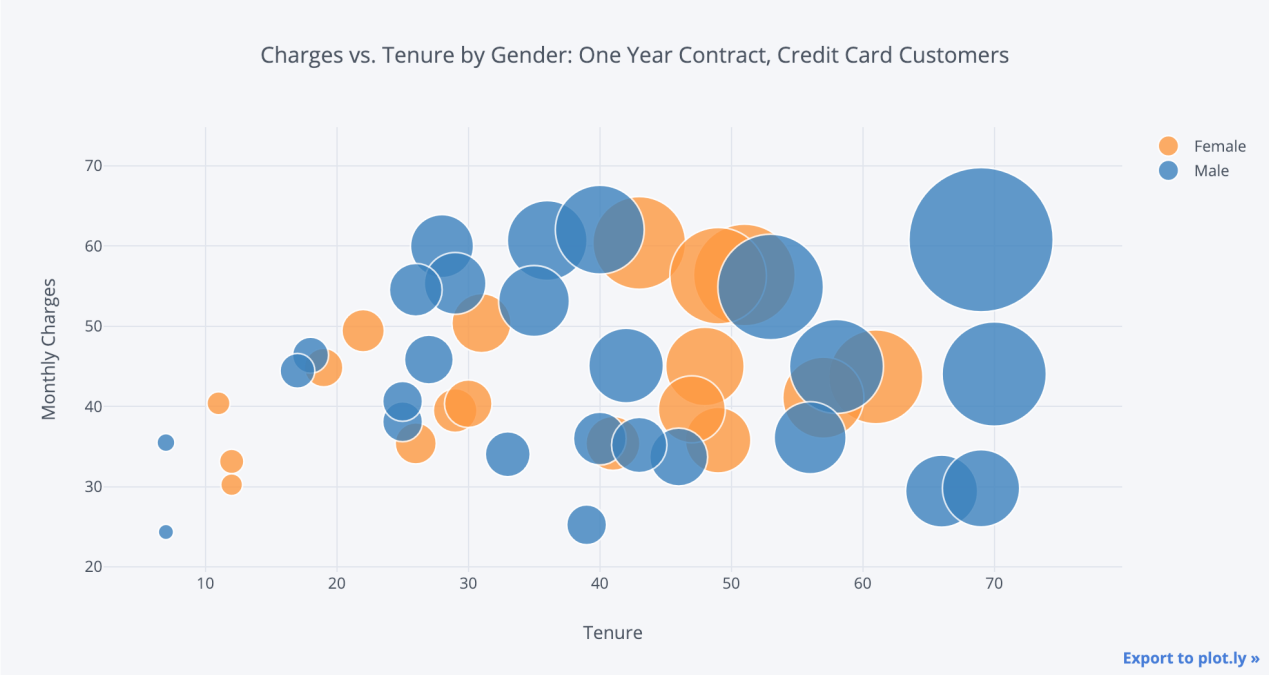
Interactive Bubble Charts

Bubble charts are essentially scatter plots with an additional dimension - the size of the bubbles. This can provide us with additional insights but also makes it so that the visualization has the potential to get cluttered even easier than with a scatter plot. Because of this, it is best to use interactive bubble charts when we have filtered our data significantly to a number of data points that will be reasonable for this type of visualization.

Below, we are starting with the data subset that we used for our scatter plot above and we are filtering it even further by keeping only the customers that do not have phone service. This makes our data sufficiently small to visualize using a bubble chart.

To create the bubble chart, we call `iplot`, pass "bubble" to the `kind` argument, and then fill in all the rest of the arguments with appropriate values.

```
data = data[data['PhoneService']=='No']

data.iplot(kind='bubble', x='tenure', y='MonthlyCharges', size='TotalCharges',
        categories='gender', xTitle='Tenure', yTitle='Total Charges',
        title='Charges vs. Tenure: One Year Contract, Credit Card Customers')
```



The result is an informative, interactive bubble chart that conveys information about both monthly and total charges, tenure, and gender for this specific segment of customers.

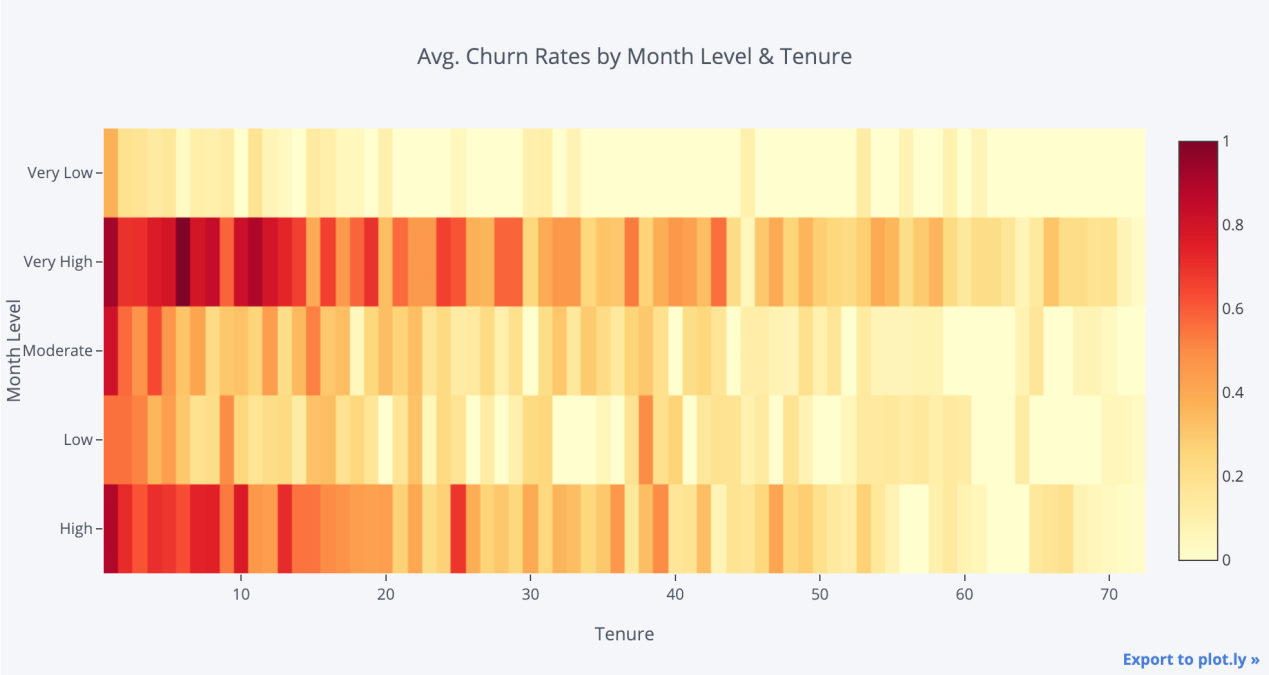
Interactive Heatmaps

The last type of interactive visualization we will cover in this lesson is the heatmap. Heatmaps are useful for seeing how relatively high or low values are across combinations of categories. For example, below we create a pivot table containing average churn rates by level of monthly charges and tenure.

```
data = df.pivot_table(values='ChurnBinary', columns='MonthLevel',
                      index='tenure', aggfunc='mean')
```

From there, we call `iplot`, pass "heatmap" to the `kind` argument, specify an appropriate colorscale, and then fill in the rest of the arguments. This generates an informative heatmap that looks like the following.

```
data.iplot(kind='heatmap', colorscale='YlOrRd', xTitle='Tenure',
           yTitle='Month Level', title='Avg. Churn Rates by Month Level & Tenure')
```



In this heatmap, we can see that the High and Very High categories are where the highest churn rates are occurring, particularly early in the customer tenure.

Summary

In this lesson, we have explored a variety of interactive visualizations. We began the lesson by introducing the plotly and cufflinks libraries, which when used together make generating interactive visualizations as easy as writing a single line of Python code. We then looked at examples of how to create interactive histograms, bar charts, line charts, scatter plots, bubble charts, and heatmaps. We hope that the ease with which we were able to create all of these makes you strongly consider incorporating interactive visualizations into your analytical workflow.