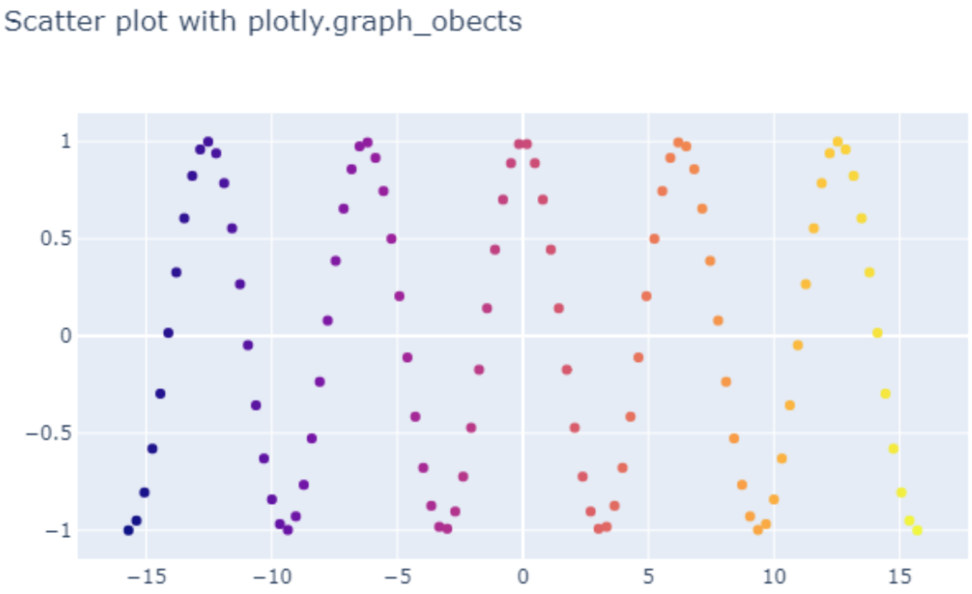
DATA SOCIETY®

Day 3 - Data visualization with Python

“One should look for what is and not what he thinks should be.”
-Albert Einstein.

Introducing plotly

- Toward the end of class today, we're going to be learning how to use `plotly` to create interactive, aesthetically pleasing visualizations
- While we wait for class to start, ***read this article*** to get familiar with the tool
- ***https://medium.com/python-in-plain-english/learn-plotly-basics-in-5-minutes-or-less-de1bf8d20436***



Recap and today's agenda

So far, we've learned about:

- Visualizing data with matplotlib
- Creating histograms, boxplots, and bar charts
- Creating scatter plots
- Customizing graphs for impact
- Creating violin plots and compound visualizations

Today, we'll explore:

- Layered plots
- Saving plots into the plot directory
- Data visualization best practices
- Creating interactive plots

Module completion checklist

Objective	Complete
Create layered plots	
Save your plots and your data	
Best practices of data visualization	
Describe uses and strengths of plotly and cufflinks packages	
Create basic interactive visualizations using cufflinks	

Recap of Matplotlib

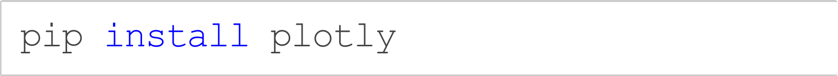
- **Univariate plots:**
  - **Histogram:** represents the distribution of numeric data. We use `plt.hist(x)` to create the plot
  - **Boxplot:** a visual summary of the 25th, 50th, and 75th percentiles of the data. We use `plt.boxplot(x)` to create the plot
  - **Bar chart:** most commonly used when visualizing survey data, or summary data. We use `plt.bar(x, height)` to create the plot
  - **Violin plot:** primarily used to look at the variations in the data. We use `plt.violinplot(x)` to create the plot
- **Bivariate plot:**
  - **Scatter plot:** shows patterns between 2 variables. We use `plt.scatter(x,y)` to create the plot

Layered plot and interactive plot

- In our last session, we created a grid from multiple plots
- Today, we are going to learn how to **layer** multiple plots into the same figure
- We are also going to learn how to use `plotly` to create **interactive** visualizations
- You can use `plotly` directly with `pandas` data frames by using a library called `cufflinks`

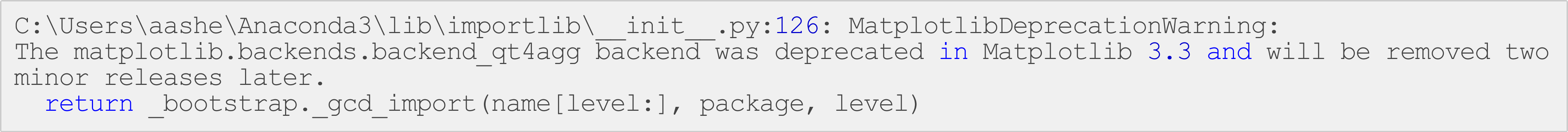
Loading packages

- Install plotly using `pip` in your terminal

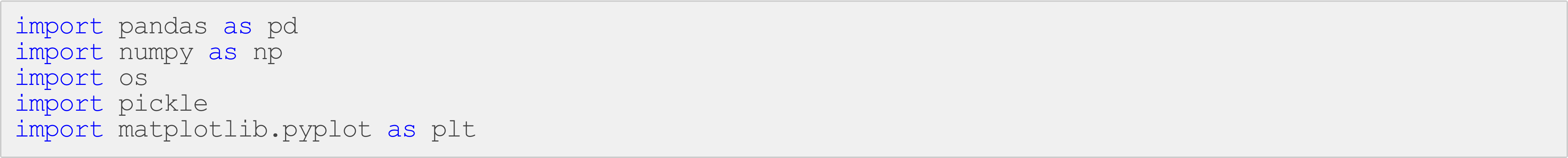


```
pip install plotly
```

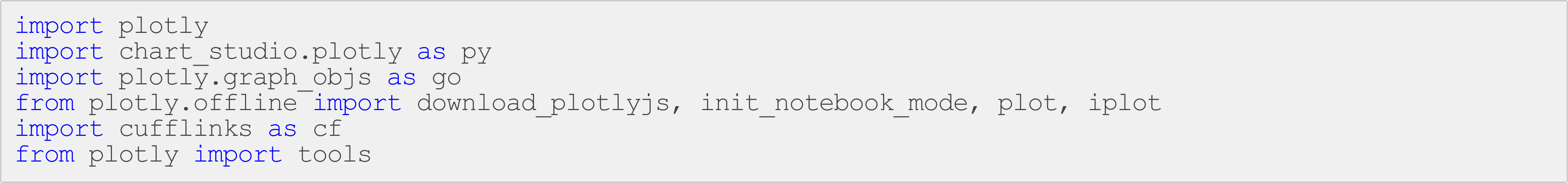
- Load the packages we will be using



```
C:\Users\aashe\Anaconda3\lib\importlib\__init__.py:126: MatplotlibDeprecationWarning:
The matplotlib.backends.backend_qt4agg backend was deprecated in Matplotlib 3.3 and will be removed two
minor releases later.
  return _bootstrap._gcd_import(name[level:], package, level)
```



```
import pandas as pd
import numpy as np
import os
import pickle
import matplotlib.pyplot as plt
```



```
import plotly
import chart_studio.plotly as py
import plotly.graph_objs as go
from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
import cufflinks as cf
from plotly import tools
```

Directory settings

- In order to maximize the efficiency of your workflow, you should encode your directory structure into `variables`
- Let the `main_dir` be the variable corresponding to your `skillsoft-data-viz-with-python` folder

```
# Set `home_dir` to the root directory of your computer.
home_dir = os.path.expanduser("~")
# Set `main_dir` to the location of your `skillsoft-data-viz-with-python` folder.
main_dir = os.path.join(home_dir, "Desktop", "skillsoft-data-viz-with-python")
```

```
# Make `data_dir` from the `main_dir` and
# remainder of the path to data directory.
data_dir = os.path.join(main_dir, "data")

# Create a plot directory to save our plots
plot_dir = os.path.join(main_dir, "plots")
```

Working directory

- Set working directory to `data_dir`

```
# Set working directory.
os.chdir(data_dir)
```

```
# Check working directory.
print(os.getcwd())
```

```
/home/[user-name]/Desktop/skillsoft-data-viz-with-python/data
```

Loading datasets

- Before creating visualizations in Python, let's load the cleaned Costa Rican data set and the long and wide grouped data sets we pickled earlier

```
costa_viz = pickle.load(open("costa_viz.sav","rb"))
```

```
costa_grouped_mean_long = pickle.load(open("costa_grouped_mean_long.sav","rb"))
```

```
costa_grouped_mean_wide = pickle.load(open("costa_grouped_mean_wide.sav","rb"))
```

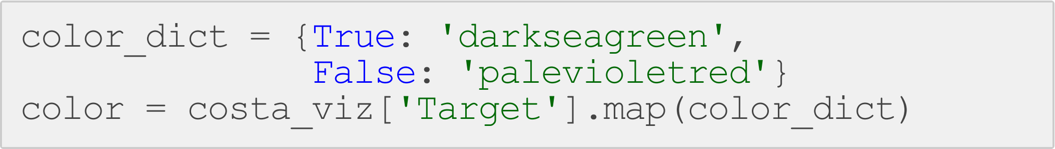
Compound visualizations: layered plots

- We can create figures containing multiple plots, layered on top of each other using the same plotting area `plt.subplots()`
- Layered plots allow any number of plotting layers, which makes them very flexible, especially for those datasets where looking at patterns across multiple categories is important!

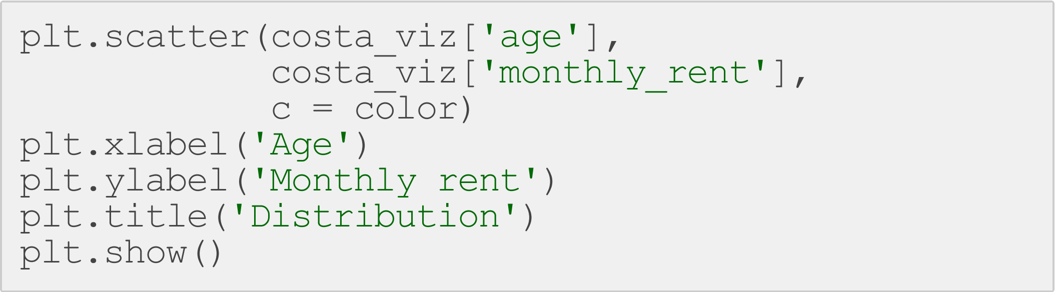
```
plt.clf()                    #<- clear plotting area
fig, axes = plt.subplots() #<- create a new figure and axes objects for plotting
```

Scatter plot from day 2

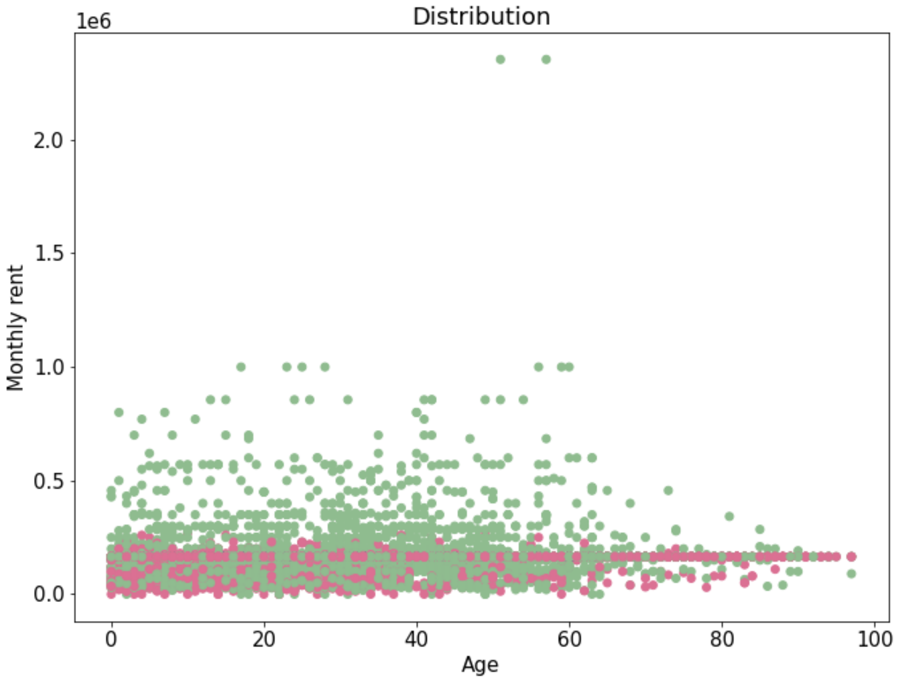
We will create the same scatter plot with a different approach.



```
color_dict = {True: 'darkseagreen',
              False: 'palevioletred'}
color = costa_viz['Target'].map(color_dict)
```



```
plt.scatter(costa_viz['age'],
            costa_viz['monthly_rent'],
            c = color)
plt.xlabel('Age')
plt.ylabel('Monthly rent')
plt.title('Distribution')
plt.show()
```



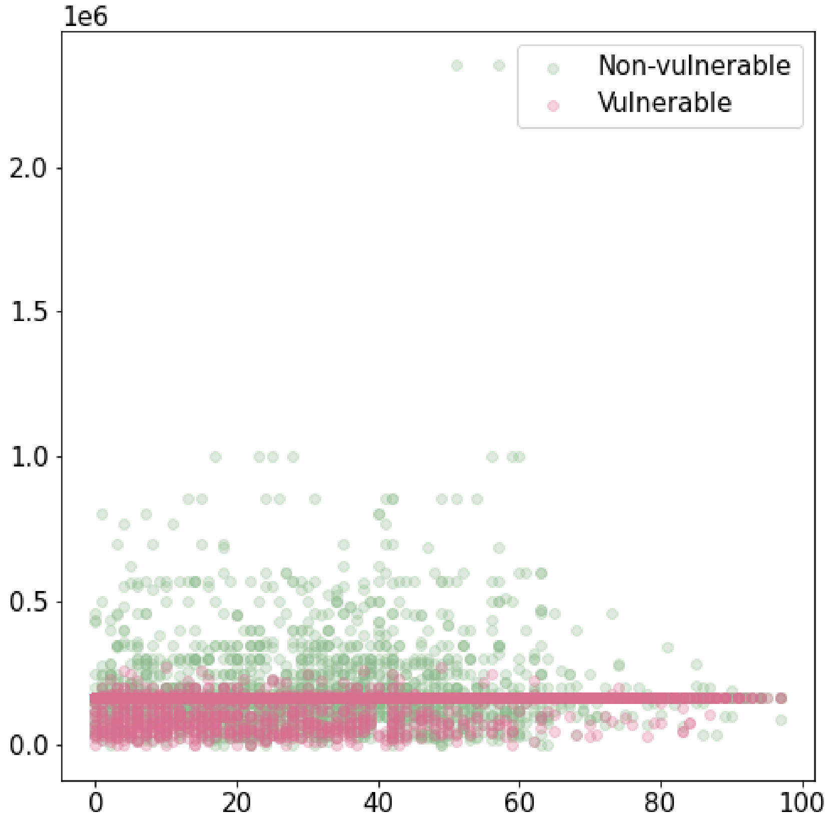
Layered plot 1: scatter plot

- We'll now create a layered plot based on the scatter plot we created earlier
- With this layered approach, we will actually create 2 layers based on age and monthly_rent:
  - The first layer for `Target` is True
  - The second layer for `Target` is False and will be added on top of the first layer.
- Compare the plots on previous and next slide. Do you notice any difference?

```
for key, value in color_dict.items():

    age = costa_viz.query('Target=='+str(key))['age']
    monthly_rent = costa_viz.query('Target=='+str(key))['monthly_rent']

    if key == 0:
        Flag = "Vulnerable"
    else:
        Flag = "Non-vulnerable"

    axes.scatter(age,
                 monthly_rent,
                 c = value,
                 label = Flag,
                 alpha = 0.3)
axes.legend()  #<- add a legend that automatically gets labels and colors from layers!
```

Layered scatter plot

```
plt.show()
```



Bar chart from day 2

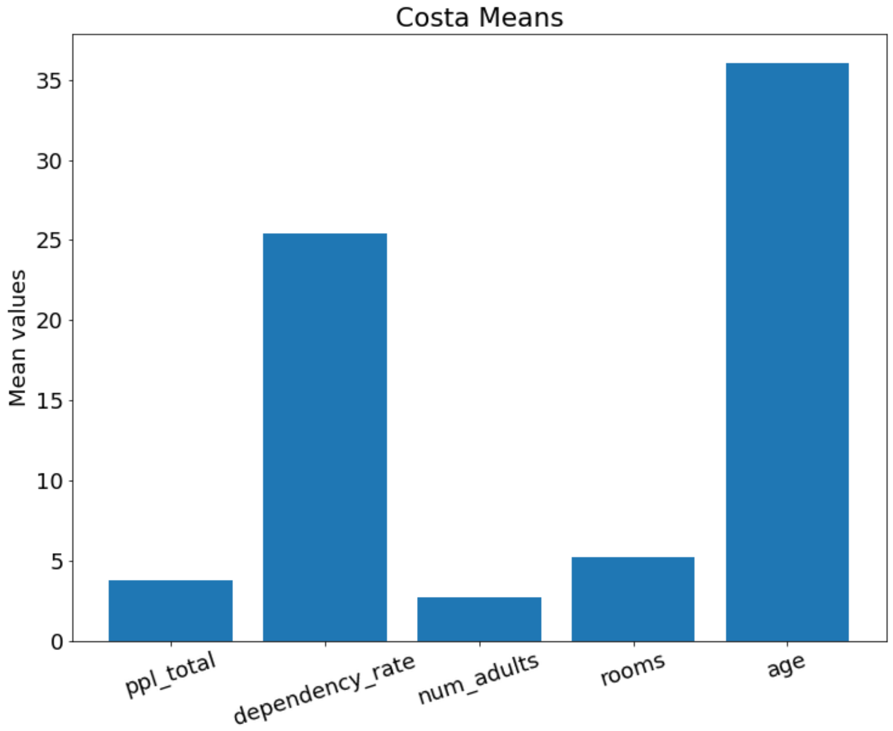
- Now recall how we made a simple bar chart

```
costa_true_means =
costa_grouped_mean_long.query('Target == True')
[['metric','mean']]
bar_labels = costa_true_means['metric']     #<-
1
bar_heights = costa_true_means['mean']      #<-
2
num_bars = len(bar_heights)
bar_positions = np.arange(num_bars)         #<-
3
```

```
# Adjust figure size before plotting.
plt.figure(figsize = (12, 9))
plt.bar(bar_positions, bar_heights)
```

```
plt.xticks(bar_positions,
           bar_labels,
           rotation = 18)
```

```
plt.ylabel('Mean values')
plt.title('Costa Means')  #<- add plot title
plt.show()
```



Layered plot 2: bar chart

- Let's try creating a layered bar chart to compare mean values between True and False

```
# We already have `Target` = `True` mean data.
print(costa_true_means)
```

```
            metric       mean
1        ppl_total   3.796531
3  dependency_rate  25.425284
5       num_adults   2.713809
7            rooms   5.205971
9              age  36.078886
```

```
# Let's get the `Target` = `False` mean data.
costa_false_means = costa_grouped_mean_long.query('Target == False')[['metric','mean']]
print(costa_false_means)
```

```
            metric       mean
0        ppl_total   4.358607
2  dependency_rate  26.011233
4       num_adults   2.388093
6            rooms   4.533839
8              age  31.314238
```

Layered bar chart: setup

```
# Mean values for `Target` = `False` data.
false_bar_heights = costa_false_means['mean']
# Mean values for `Target` = `True` data.
true_bar_heights = costa_true_means['mean']
# Labels of bars, their width, and positions are shared for both categories.
bar_labels = costa_false_means['metric']
num_bars = len(bar_labels)
bar_positions = np.arange(num_bars)
width = 0.35
```

```
# Clear the plotting area for the new plot.
plt.clf()
# Create the figure and axes objects.
fig, axes = plt.subplots()
```

Layered bar chart: layout

```
false_bar_chart = axes.bar(bar_positions,           #<- set `false` bar positions
                            false_bar_heights,      #<- set `false` bar heights
                            width,                  #<- set width of the bars
                            color = color_dict[0]) #<- set color corresponding to `False` in dictionary
```

```
true_bar_chart = axes.bar(bar_positions + width, #<- set `true` bar positions
                          true_bar_heights,      #<- set `true` bar heights
                          width,                 #<- set width of the bars
                          color = color_dict[1]) #<- set color corresponding to `True` in dictionary
```

Layered bar chart: labels

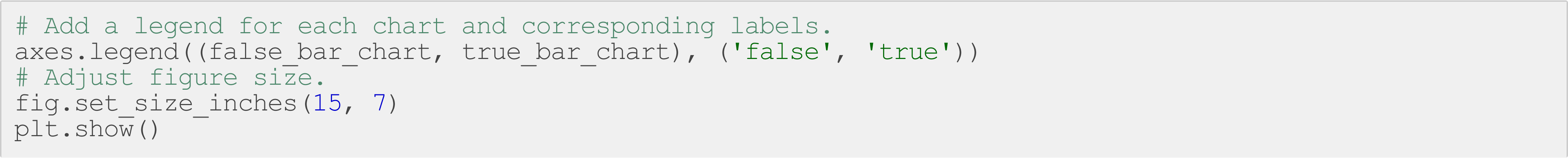
```python
# Add text for labels, title and axes ticks.
axes.set_ylabel('Mean values')
axes.set_title('Costa metrics summary by Target')
axes.set_xticks(bar_positions + width/2)
```

```
[<matplotlib.axis.XTick object at 0x00000000710CFD30>, <matplotlib.axis.XTick object at
0x00000000710CF908>, <matplotlib.axis.XTick object at 0x00000000710CF550>, <matplotlib.axis.XTick
object at 0x000000007F377B00>, <matplotlib.axis.XTick object at 0x000000007F36F240>]
```

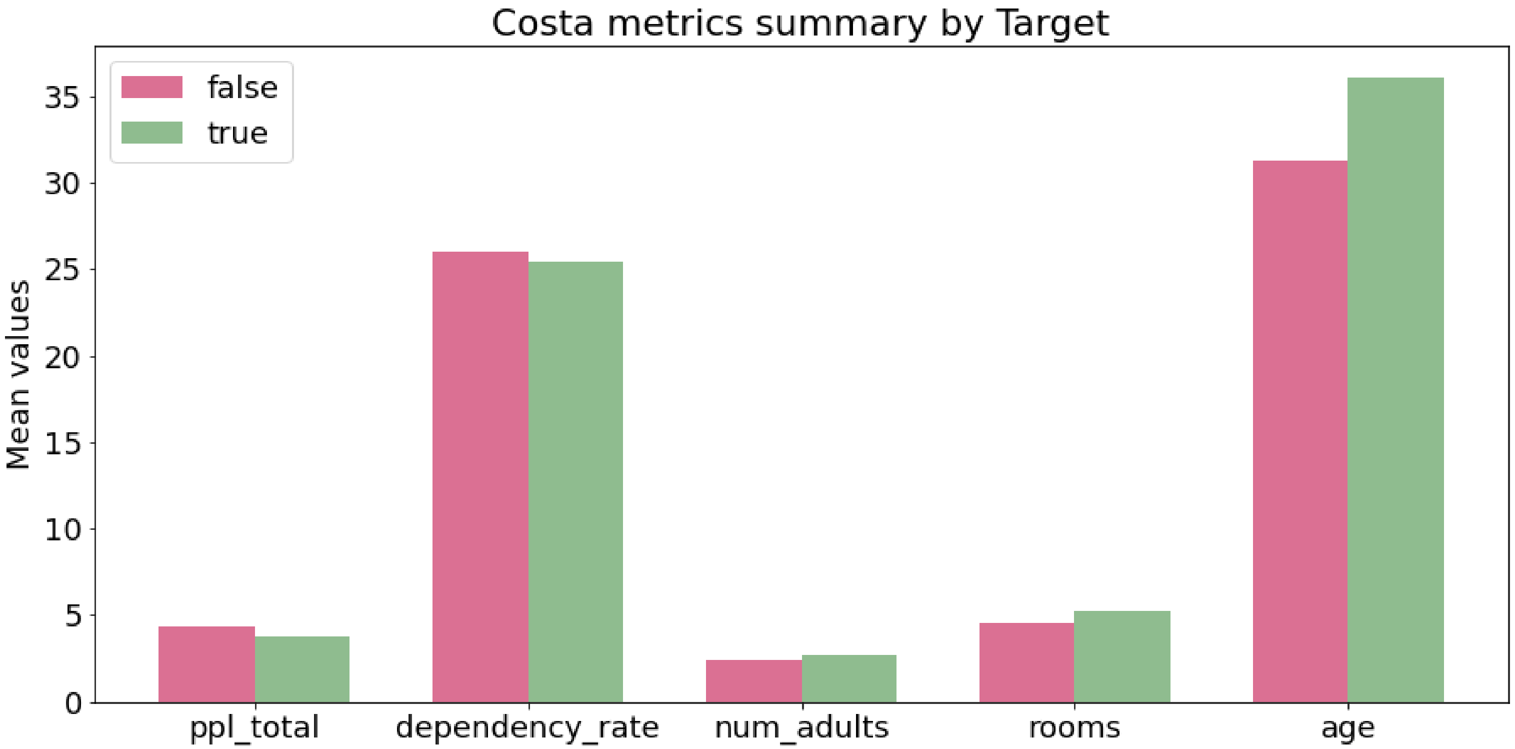
```python
axes.set_xticklabels(bar_labels)
```

```
[Text(0.175, 0, 'ppl_total'), Text(1.175, 0, 'dependency_rate'), Text(2.175, 0, 'num_adults'),
Text(3.175, 0, 'rooms'), Text(4.175, 0, 'age')]
```

Layered bar chart: legend and render



```
# Add a legend for each chart and corresponding labels.
axes.legend((false_bar_chart, true_bar_chart), ('false', 'true'))
# Adjust figure size.
fig.set_size_inches(15, 7)
plt.show()
```



Module completion checklist

Objective	Complete
Create layered plots	✓
Save your plots and your data	
Best practices of data visualization	
Describe uses and strengths of plotly and cufflinks packages	
Create basic interactive visualizations using cufflinks	

Saving your plots

- You will be saving all of your graphs in the `plots` folder
- Save the current plot with `fig.savefig()`, where `fig` is any figure you want to save

```
fig.savefig(main_dir + '/plots/costa_metrics_by_target.png')
```

- Now open your plots folder and look at the file you have saved

Saving your data

- To save your data to a `csv` file, we will use a simple `df.to_csv()` function, where `df` is any dataframe
- When saving to a `csv` format, make sure to provide:
  - the path to your file with its name
  - the `index` argument (if it is set to `True`, the dataframe will be written with its index as the leftmost column)

```
costa_grouped_mean_long.to_csv(data_dir + '/costa_summary_by_target.csv',
                               index = False)
```

- Now open your data folder and look at the file you have saved

Knowledge check 1

Exercise 1

Module completion checklist

Objective	Complete
Create layered plots	✓
Save your plots and your data	✓
Best practices of data visualization	
Describe uses and strengths of plotly and cufflinks packages	
Create basic interactive visualizations using cufflinks	

Visualization best practices

Four pillars

- **Purpose** – identify the stakeholders and their objectives
- **Content** – pull out the content that matters most to the stakeholders
- **Structure** – which chart best displays the content you want to display?
- **Formatting** – are the titles and axes easily readable? Are the colors aesthetically pleasing?

Purpose

- Know your **audience** and understand how it processes visual information
- Consider audience **familiarity**:
  - High-level executives are generally well-versed in visual data, so use a variety of methods to stand out
  - Less-experienced audiences will want it kept simple (e.g., pie charts, bar graphs, and word maps)
- Consider how the visualization will be **used** by the audience:
  - Is it for executives to use to make decisions?
  - Is it to inform the public?

Content

- Determine what you're trying to visualize and **what kind of information** you want to communicate
- Remember, the audience only knows as much as you tell them:
  - Do you want them to **explore** the data on their own? (**exploratory** analysis)
  - Do you want to tell a **specific story** about the data? (**explanatory purposes**)
- If the message is **explanatory**, consider:
  - What type of data you have on which to base the analysis?
  - What are the audience's topmost concerns or requirements?
  - What decisions can be made based on the results you provide?

Structure

- Choose a type of visual that conveys the information in the **best and simplest form** for your audience
- The type of visual you use depends primarily on two things:
  - the data you want to communicate
  - what you want to convey about that data
- Then, choose the visual that will be **easiest** for your audience to read
  - Aim for them to “get it” in **30 seconds or less**

Format: choose the right type of visual

- Remember, we first define the **purpose** and **content**, and then the **structure**
- The type of visual you choose should depend on the **data** you want to communicate and the **message**
- Consider:
  - How many variables do I want to show?
  - How many data points are there?
  - Should values be displayed over time?
  - Should similar items be grouped?

Comparisons

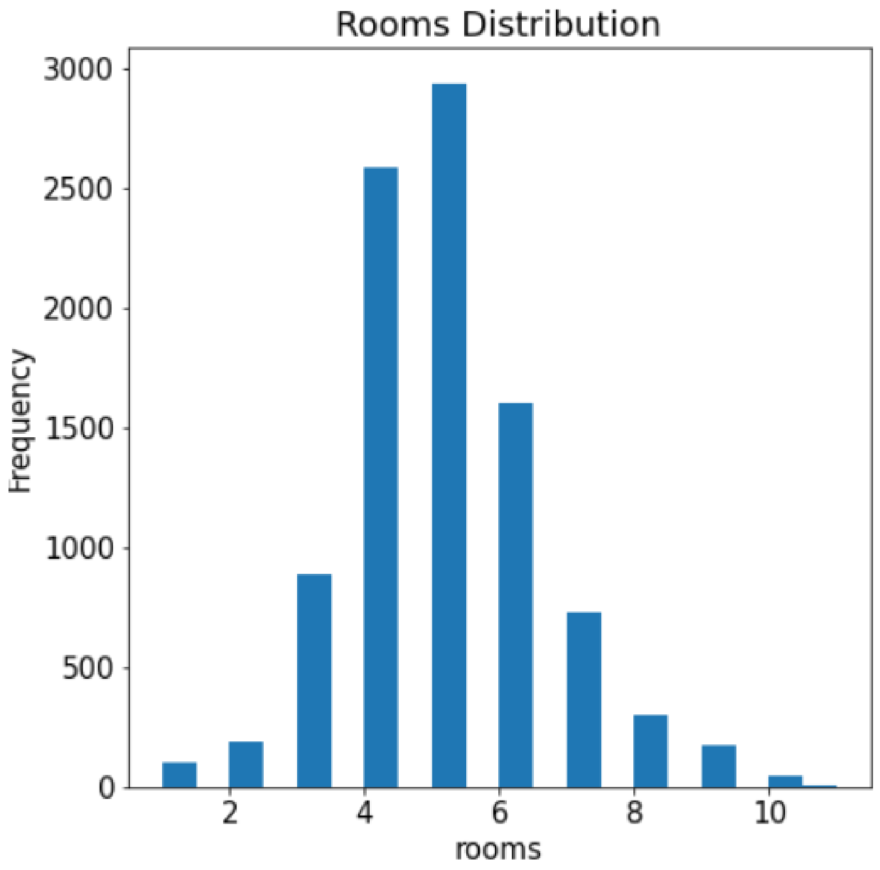
- Comparisons help us evaluate and compare values between two or more data points
- Examples include:
  - Total number of visitors per month, grouped by country of residence, to see where most visitors come from and where to put more efforts
  - Quarterly expenditures for a particular project, to spot trends or performance issues
  - Number of COVID-19 patients by city, highlighting the prevention efforts undertaken in that area
- Go-to visualizations: bar charts, pie charts, & line charts

Composition

- Composition will show how individual parts make up the whole
- Examples include:
  - Advertising spend, by medium, for a given year
  - Total country population by religions, languages, or ethnical groups
  - Total budget, by strategic objective, department, or region
- Go-to visualizations: bar charts, pie charts, waterfall charts, & stacked area charts

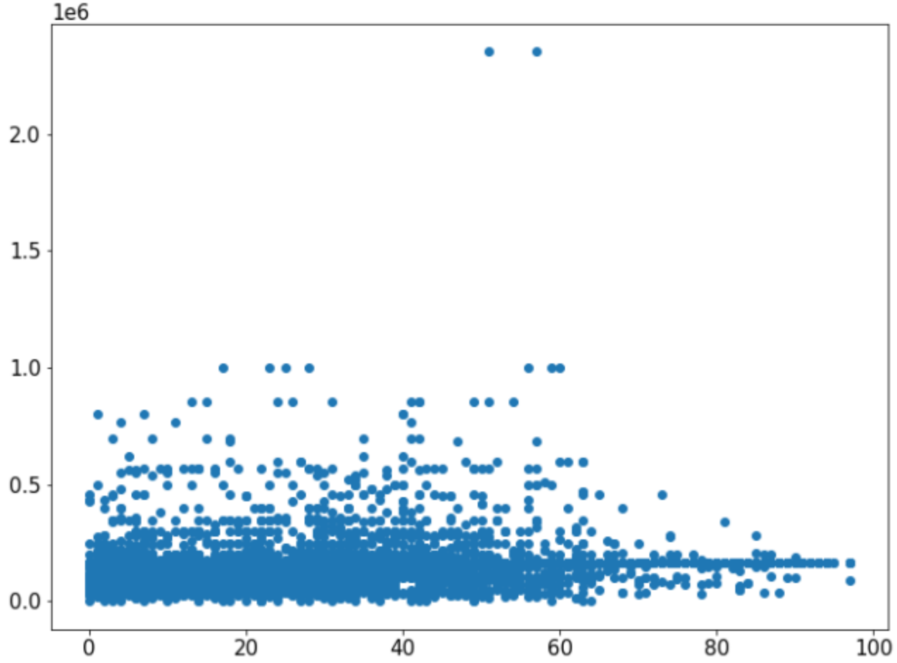
Distributions

- Distributions combine comparison and composition
- Examples include:
  - The distribution of ages in a group of people
  - Identifying problems or constraints in quality control systems
- Go-to visualizations: histograms, line charts, area charts, scatter plots, & maps

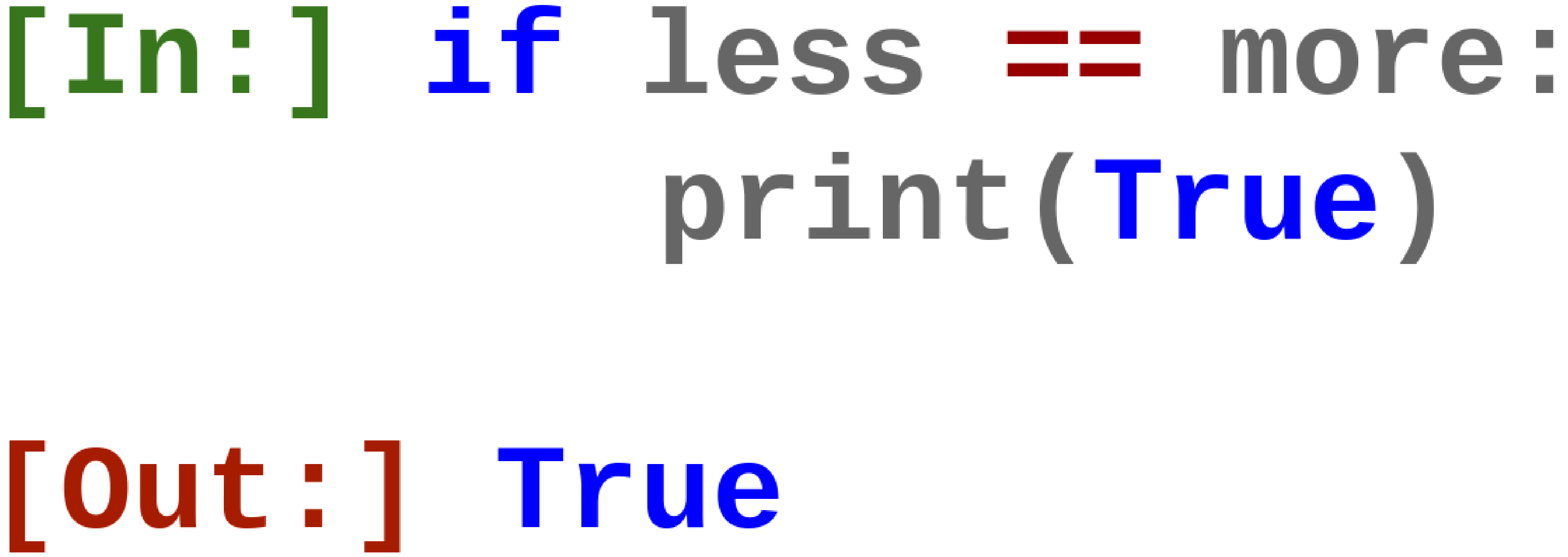


Relationships

- Sometimes we want to see the relationships, correlation, or connection of two or more variables and their properties
- Examples include:
  - Estimating how expenditures in advertising affect sales
  - Spotting trouble areas by evaluating budget vs. expenses by department or region
  - Answering questions such as, “Does income level depend on education level?”
- Go-to visualizations: scatter plots, bubble charts, & line charts



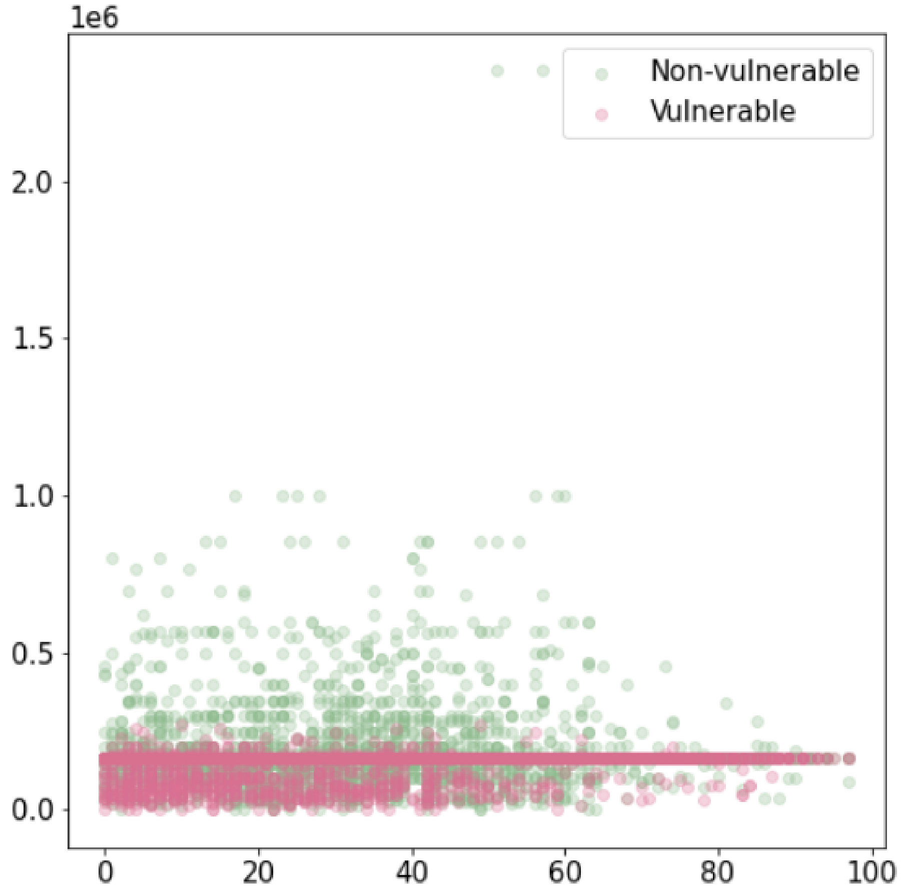
And remember, more often than not …



```
[In:] if less == more:
              print(True)


[Out:] True
```

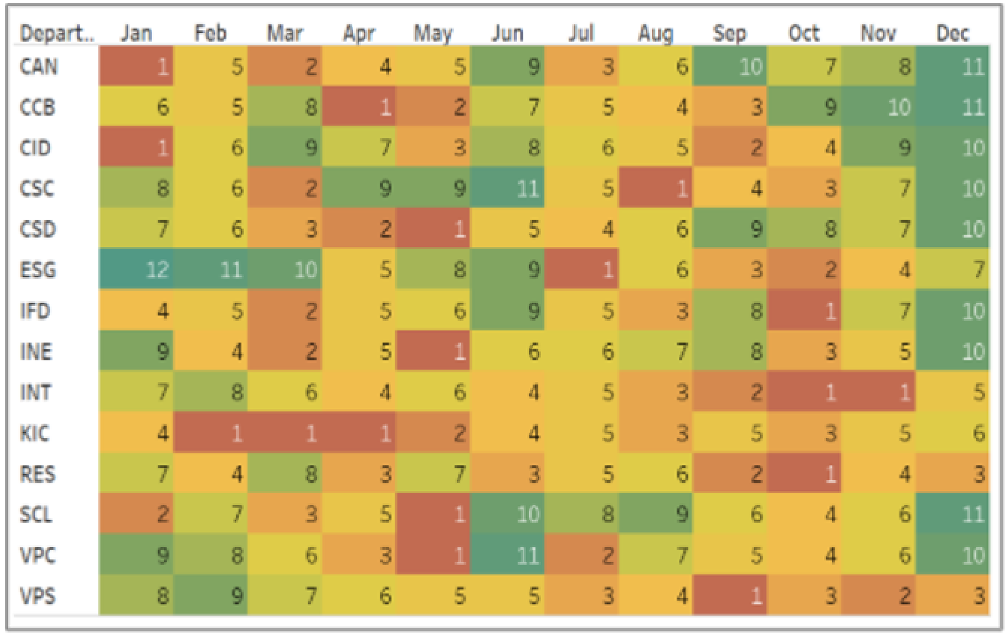
Encode data with color

- Use color schemes to encode data as sequential, diverging, or categorical
- Use a categorical color scheme for discrete data values representing distinct categories
- These schemes use different hues with consistent steps in lightness and saturation

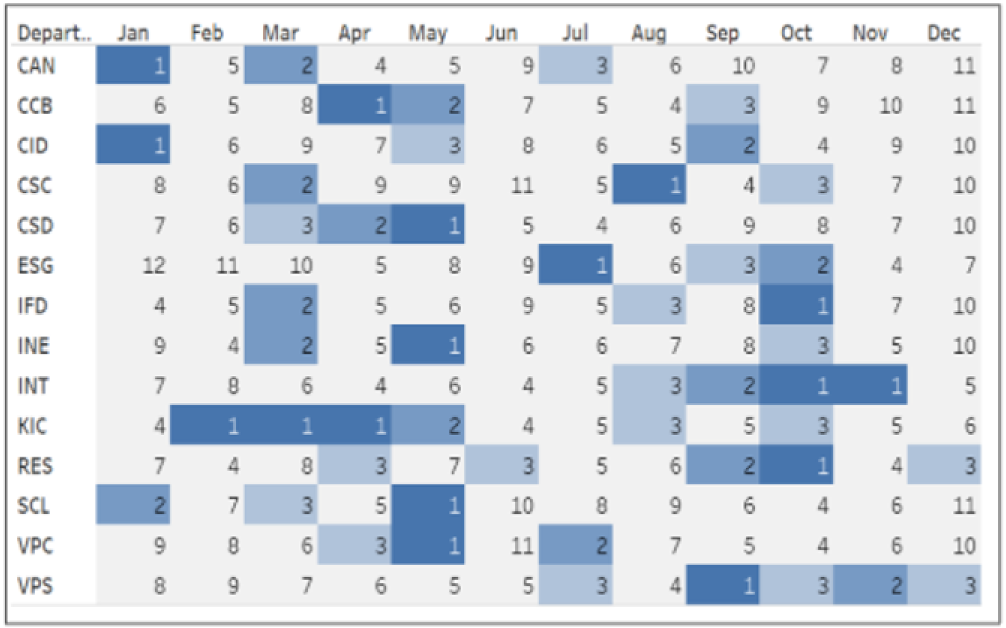


Format: color

- Color is another powerful tool used to draw the audience's attention
- However, keep the following in mind:
  - Use it **sparingly**: too much variety prevents anything from standing out
  - Use it **consistently**: a color change can be used to visually reinforce change in topic or tone



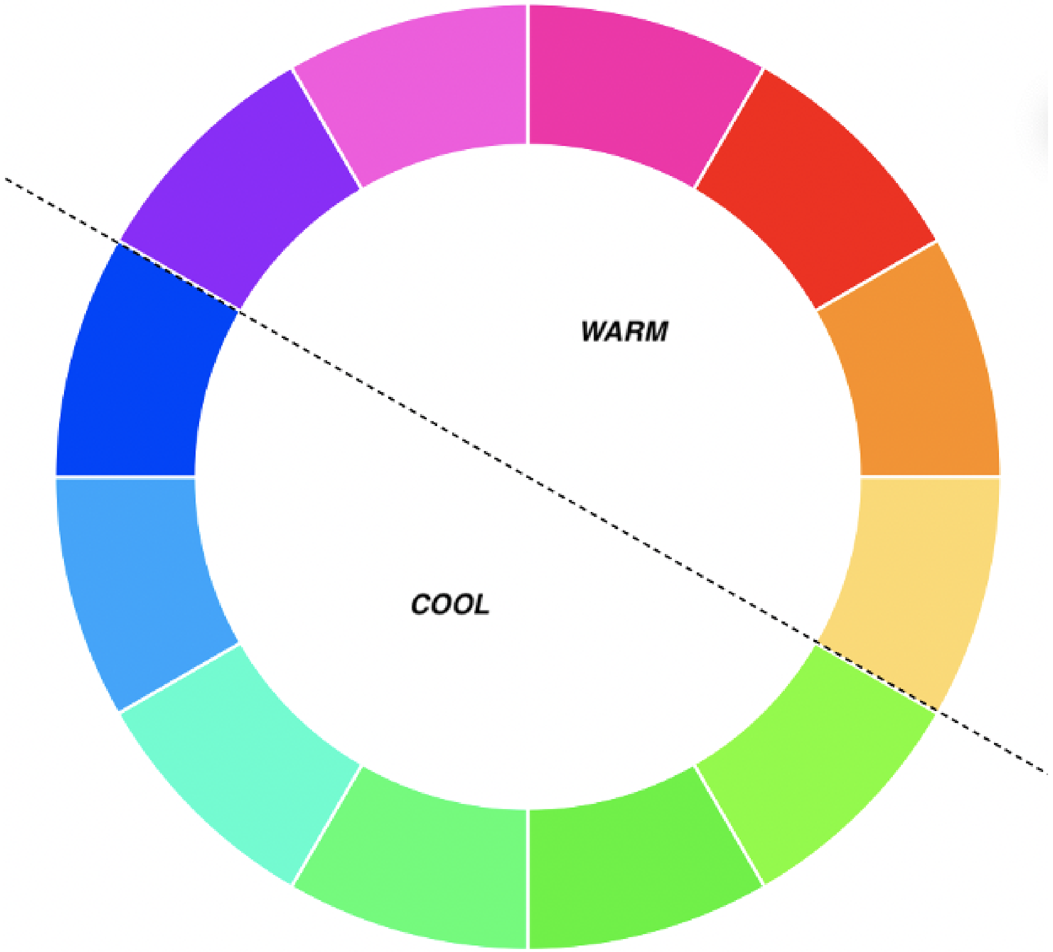
Depart..	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
CAN	1	5	2	4	5	9	3	6	10	7	8	11
CCB	6	5	8	1	2	7	5	4	3	9	10	11
CID	1	6	9	7	3	8	6	5	2	4	9	10
CSC	8	6	2	9	9	11	5	1	4	3	7	10
CSD	7	6	3	2	1	5	4	6	9	8	7	10
ESG	12	11	10	5	8	9	1	6	3	2	4	7
IFD	4	5	2	5	6	9	5	3	8	1	7	10
INE	9	4	2	5	1	6	6	7	8	3	5	10
INT	7	8	6	4	6	4	5	3	2	1	1	5
KIC	4	1	1	1	2	4	5	3	5	3	5	6
RES	7	4	8	3	7	3	5	6	2	1	4	3
SCL	2	7	3	5	1	10	8	9	6	4	6	11
VPC	9	8	6	3	1	11	2	7	5	4	6	10
VPS	8	9	7	6	5	5	3	4	1	3	2	3



Depart..	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
CAN	1	5	2	4	5	9	3	6	10	7	8	11
CCB	6	5	8	1	2	7	5	4	3	9	10	11
CID	1	6	9	7	3	8	6	5	2	4	9	10
CSC	8	6	2	9	9	11	5	1	4	3	7	10
CSD	7	6	3	2	1	5	4	6	9	8	7	10
ESG	12	11	10	5	8	9	1	6	3	2	4	7
IFD	4	5	2	5	6	9	5	3	8	1	7	10
INE	9	4	2	5	1	6	6	7	8	3	5	10
INT	7	8	6	4	6	4	5	3	2	1	1	5
KIC	4	1	1	1	2	4	5	3	5	3	5	6
RES	7	4	8	3	7	3	5	6	2	1	4	3
SCL	2	7	3	5	1	10	8	9	6	4	6	11
VPC	9	8	6	3	1	11	2	7	5	4	6	10
VPS	8	9	7	6	5	5	3	4	1	3	2	3

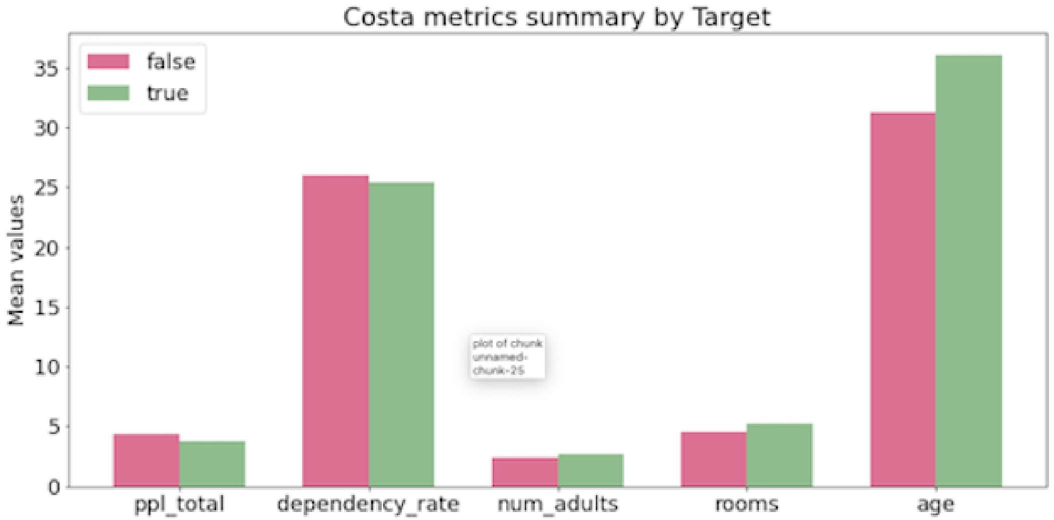
Use color to evoke emotion

- Color evokes emotion, so choose the one that helps reinforce the emotion you want to arouse in your audience
- **Warm colors** represent **energy**
- **Cool colors** represent **calmness**



Don’t forget color-blindness

- Color-blindness impacts roughly 8% of men and half a percent of women and results in difficulty distinguishing shades of red and green
- Design with color-blindness in mind by varying boldness, saturation, or brightness to distinguish colors
- What seems like a problem with the layered bar chart that we’ve created?



Module completion checklist

Objective	Complete
Recap from previous lecture	✓
Create layered plots	✓
Save your plots and your data	✓
Best practices of data visualization	✓
Describe uses and strengths of plotly and cufflinks packages	
Create basic interactive visualizations using cufflinks	

Visualizing data with plotly

- `plotly` is a popular graphing library which makes interactive, publication-quality graphs online
- Plotly also integrates with IPython to create interactive graphs in a Jupyter Notebook.
- You can begin to explore the different types of plots you can create with `plotly` by browsing their ***gallery***
- It also gives us the flexibility to plot online and offline

Using cufflinks with plotly

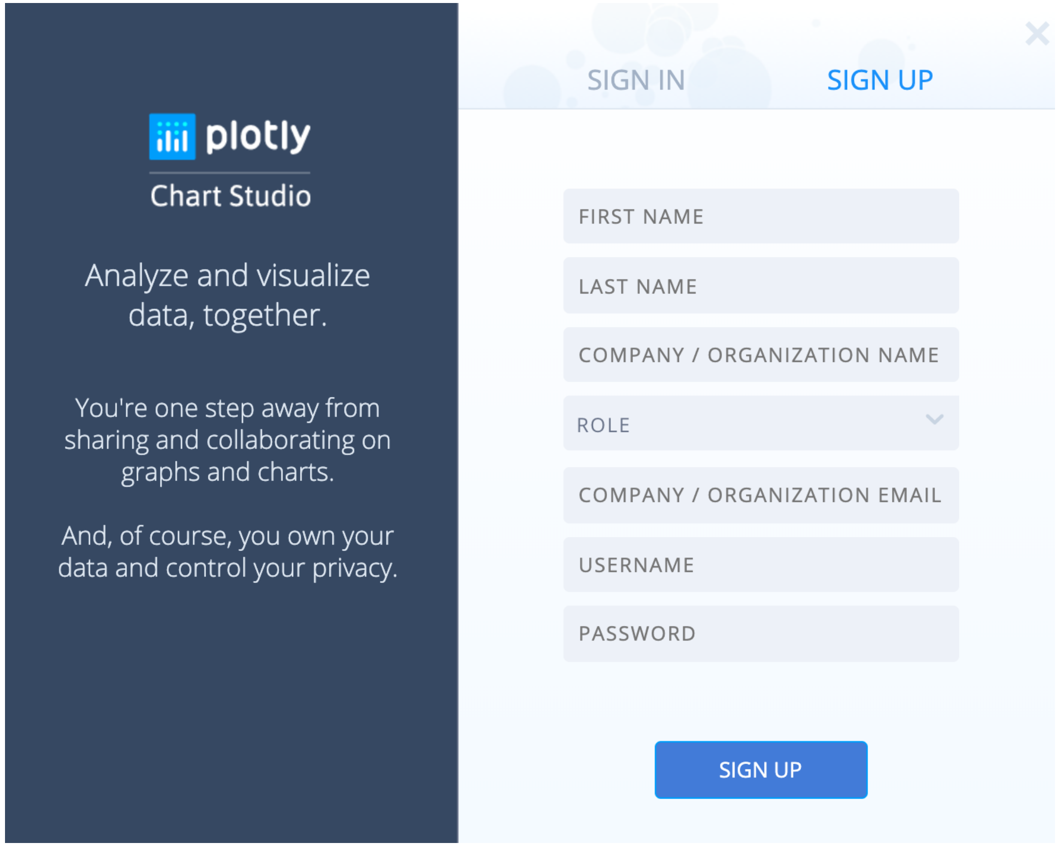
- `cufflinks` is another library used to bind `plotly` directly with a `pandas` dataframe
- This allows us to create easy, interactive visualizations
- You can see all the different plots covered by `cufflinks` ***here***

```
import plotly as py
import cufflinks as cf
```

- We will first create simple plots using `cufflinks`
- Then we will create some complex visualizations using `plotly` directly

Online plotting with plotly

- We can create a free account ***here*** and set your credentials in the notebook before plotting
- Graphs created will be saved on your plotly server account
- Keep in mind that all default plots are set to **public**
- You can only keep one private plot with a free account



Analyze and visualize data, together.

You're one step away from sharing and collaborating on graphs and charts.

And, of course, you own your data and control your privacy.

FIRST NAME

LAST NAME

COMPANY / ORGANIZATION NAME

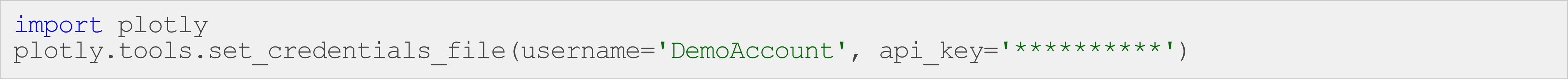
COMPANY / ORGANIZATION EMAIL

USERNAME

PASSWORD

Online plotting with plotly

- We can set our `username` and `api_key` as shown in our notebook



```
import plotly
plotly.tools.set_credentials_file(username='DemoAccount', api_key='**********')
```

- Plots can have three types of privacy levels: **public**, **private** and **secret**
- There are two methods for online plotting:
  - `plotly.plot()` returns the unique URL of the plot
  - `plotly.iplot()` displays the plot in Jupyter notebook

Offline plotting with plotly

- We can also create plots **offline and save them locally**
- Here's how we can handle offline plotting:
  - `plotly.offline.plot()` creates standalone HTML file which is saved locally
  - `plotly.offline.iplot()` is used to display the plot in Jupyter Notebook
- This method is more feasible for today's session, so let's stay offline

Initialization steps for offline plotting

- Run additional initialization codes shown below which will help us with plotting offline
- We initialize the Plotly Notebook mode by injecting the JavaScript `plotly.js` into our notebook

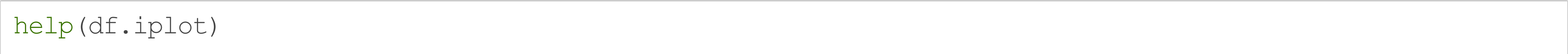
```
init_notebook_mode(connected = True)
```

- The code below allows us to use `cufflinks` offline

```
cf.go_offline()
```

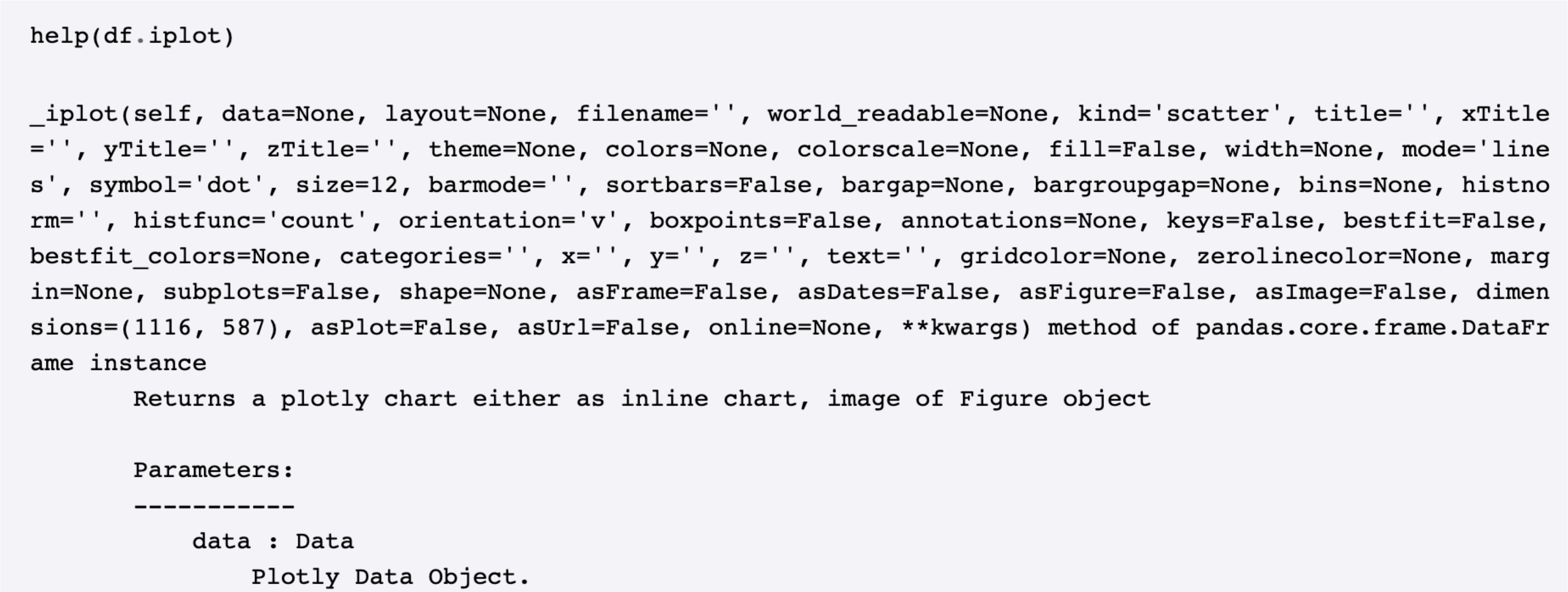
Using cufflinks with plotly

- We can view all the parameters and available options in cufflinks as shown below:



```
help(df.iplot)
```

- Where `df` is the dataframe we want to work on



```
help(df.iplot)

_iplot(self, data=None, layout=None, filename='', world_readable=None, kind='scatter', title='', xTitle
='', yTitle='', zTitle='', theme=None, colors=None, colorscale=None, fill=False, width=None, mode='line
s', symbol='dot', size=12, barmode='', sortbars=False, bargap=None, bargroupgap=None, bins=None, histno
rm='', histfunc='count', orientation='v', boxpoints=False, annotations=None, keys=False, bestfit=False,
bestfit_colors=None, categories='', x='', y='', z='', text='', gridcolor=None, zerolinecolor=None, marg
in=None, subplots=False, shape=None, asFrame=False, asDates=False, asFigure=False, asImage=False, dimen
sions=(1116, 587), asPlot=False, asUrl=False, online=None, **kwargs) method of pandas.core.frame.DataFr
ame instance
       Returns a plotly chart either as inline chart, image of Figure object

       Parameters:
       -----------
           data : Data
               Plotly Data Object.
```

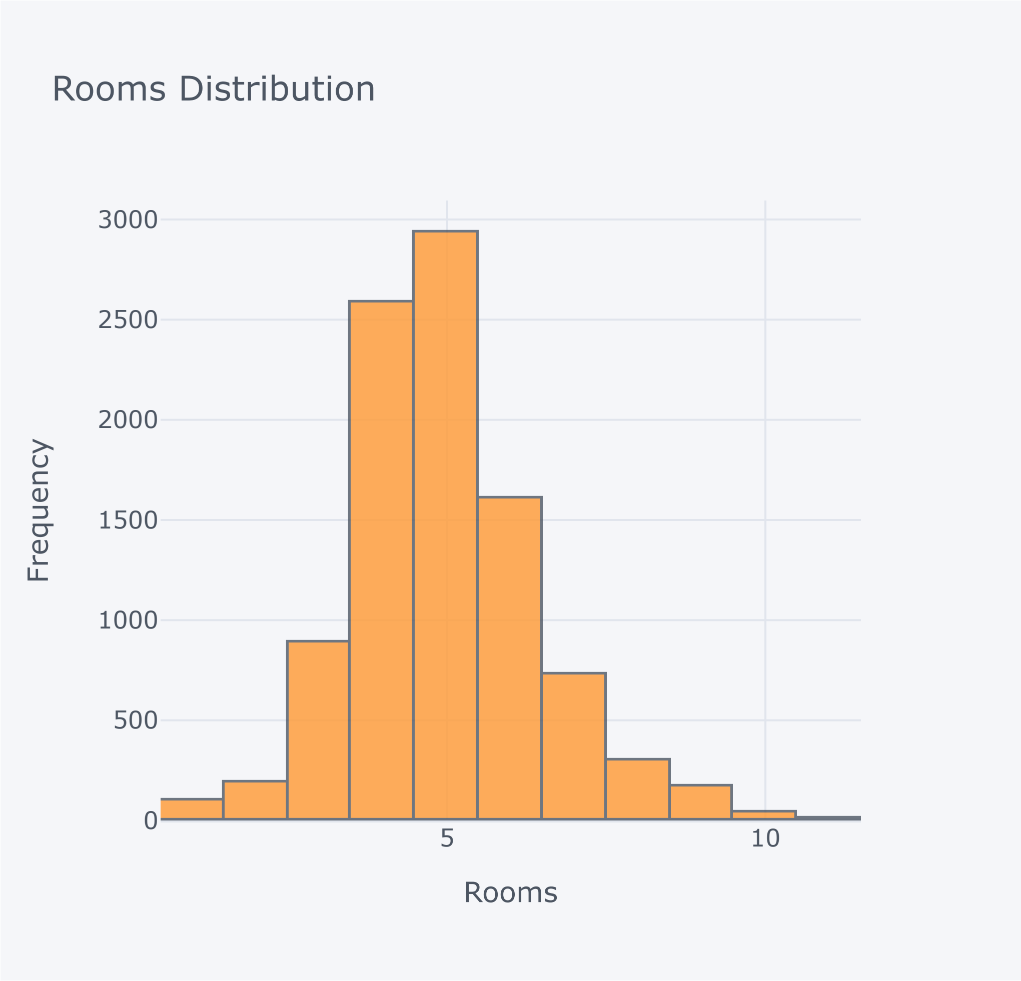
Module completion checklist

Objective	Complete
Recap from previous lecture	✓
Create layered plots	✓
Save your plots and your data	✓
Best practices of data visualization	✓
Describe uses and strengths of plotly and cufflinks packages	✓
Create basic interactive visualizations using cufflinks	

Univariate plots: histogram

- We've already covered the visualization concepts in the previous class, so let's go ahead and create a simple histogram of `rooms`
- We can use `.iplot()` to produce a basic histogram of any *numeric* variable

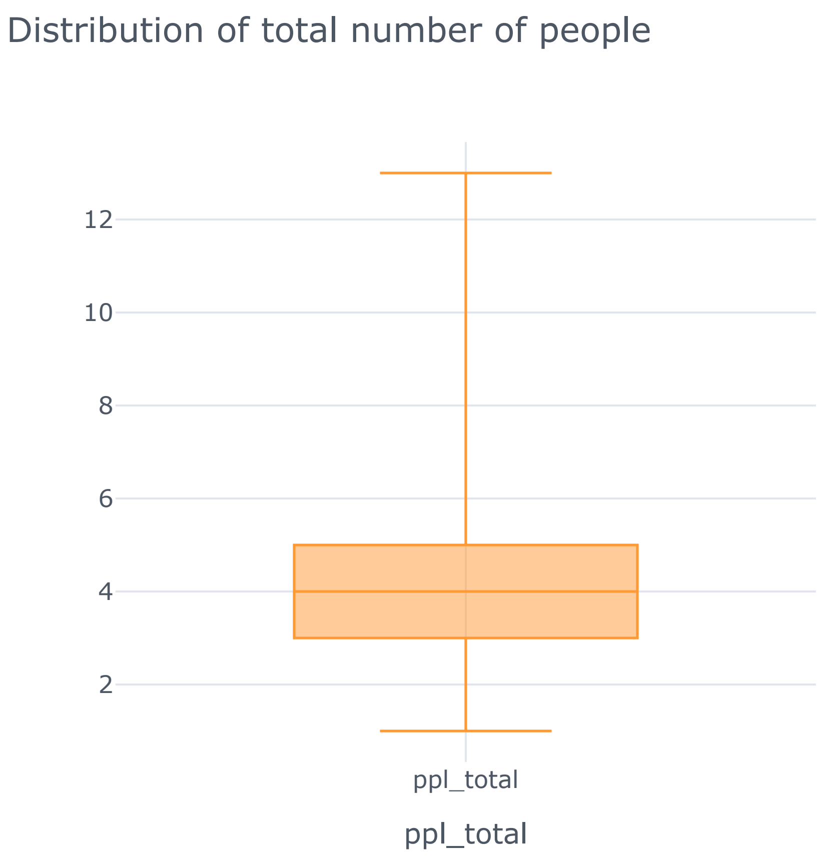
```
costa_viz['rooms'].iplot(kind = 'hist',
                         xTitle = 'Rooms',
                         yTitle = 'Frequency',
                         title = 'Rooms
Distribution')
```



Univariate plots: boxplot

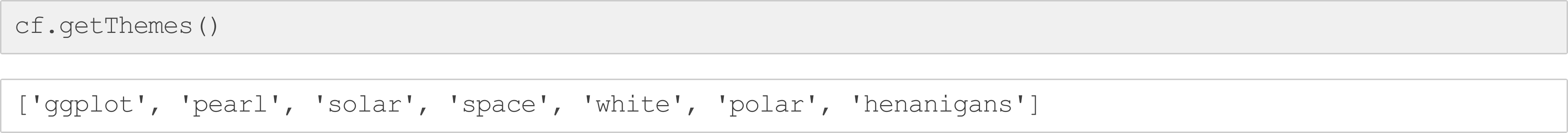
- Similarly, let's create a boxplot of `ppl_total`
- Let's also set the theme as `white`

```
costa_viz['ppl_total'].iplot(kind ='box',
                             theme = 'white',
                             xTitle =
'ppl_total',
                             title =
'Distribution of total number of people')
```



Customize themes

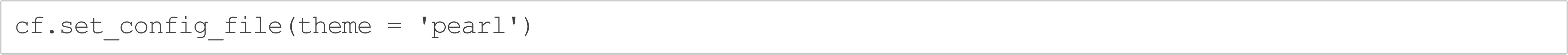
- We can see all the available themes using `cf.getThemes()`



```
cf.getThemes()
```

```
['ggplot', 'pearl', 'solar', 'space', 'white', 'polar', 'henanigans']
```

- We can also set the global, default settings for all plots as shown



```
cf.set_config_file(theme = 'pearl')
```

Univariate plots: bar chart

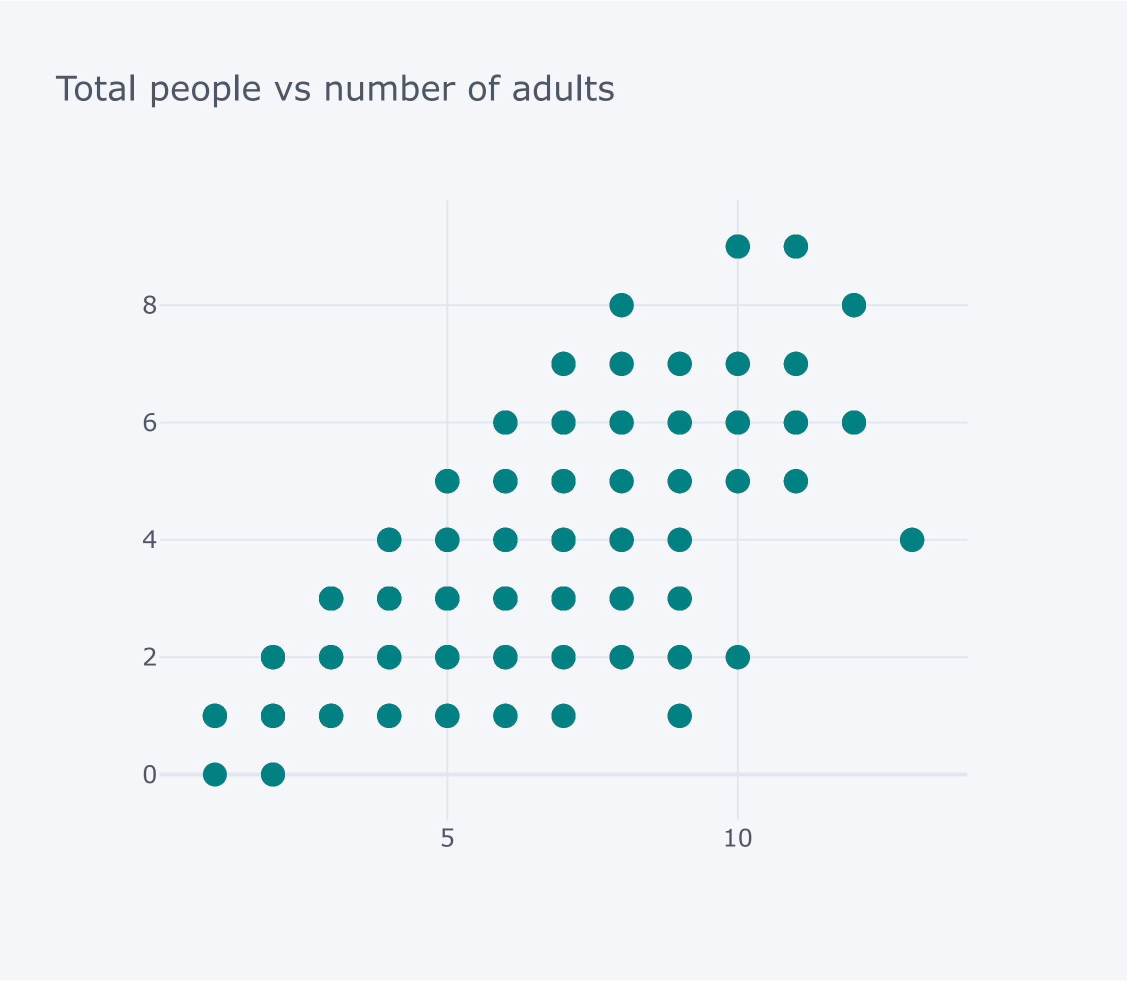
- Now we can create a simple bar chart of the discrete variables

```
costa_viz[['ppl_total', 'dependency_rate', 'num_adults', 'rooms', 'age']].mean().iplot(kind = 'bar',
                                                                                   color = 'firebrick')
```

Bivariate plots: scatter plot

- Scatter plots are great for showing **patterns between 2 variables** (hence *bivariate*)
- Let's plot `ppl_total` against `num_adults` for each observation

```
costa_viz.iplot(
              kind = 'scatter',
              x = 'ppl_total',
              y = 'num_adults',
              color = 'teal',
              title = 'Total people vs
number of adults',
              mode = 'markers')
```



Knowledge check 2

Exercise 2

Module completion checklist

Objective	Complete
Recap from previous lecture	✓
Create layered plots	✓
Save your plots and your data	✓
Best practices of data visualization	✓
Describe uses and strengths of plotly and cufflinks packages	✓
Create basic interactive visualizations using cufflinks	✓

Summary

So far, we have:

1. Created layered plots
2. Learned how to save plots and the data
3. Discussed the best practices for data visualization
4. Created basic interactive plots using plotly and cufflinks

In the next session, we will continue learning about those two packages to build more interactive plots.

This completes our module

Congratulations!