DATA SOCIETY®

Introduction to visualization in Python - day 2

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

Module completion checklist

Objective	Complete
Describe uses and strengths of plotly and cufflinks packages	
Transform Costa Rican dataset for visualizations	
Create basic interactive visualizations using cufflinks	
Visualize multiple metrics using cufflinks	
Create complex interactive visualizations using plotly	
Generate interactive visualizations with transformed summary data	
Construct interactive choropleth maps	
Save your cufflinks and plotly graphs	

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 `af-werx` folder

```
# Set `main_dir` to the location of your `af-werx` folder (for Linux).
main_dir = "/home/[username]/Desktop/af-werx"
```

```
# Set `main_dir` to the location of your `af-werx` folder (for Mac).
main_dir = '/Users/[username]/Desktop/af-werx'
```

```
# Set `main_dir` to the location of your `af-werx` folder (for Windows).
main_dir = "C:\\Users\\[username]\\Desktop\\af-werx"
```

```
# Make `data_dir` from the `main_dir` and
# remainder of the path to data directory.
data_dir = main_dir + "/data"
```

```
# Create a plot directory to save our plots
plot_dir = main_dir + "/plots"
```

Loading packages

- Install plotly using `pip` in your terminal

```
pip install plotly
```

- Load the packages we will be using

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

```
import plotly
import plotly.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
```

Working directory

- Set working directory to `data_dir`

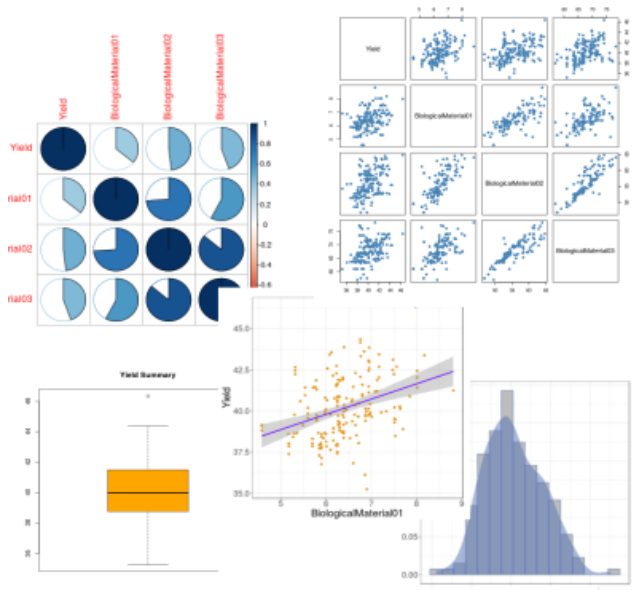
```
# Set working directory.
os.chdir(data_dir)
```

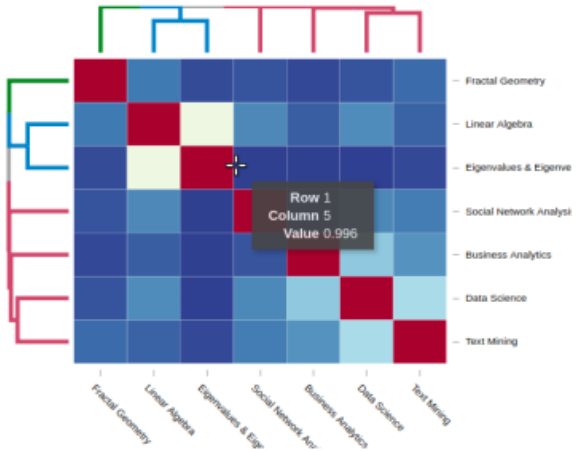
```
# Check working directory.
print(os.getcwd())
```

```
/home/[user-name]/Desktop/af-werx/data
```

Recap: why build a visualization?

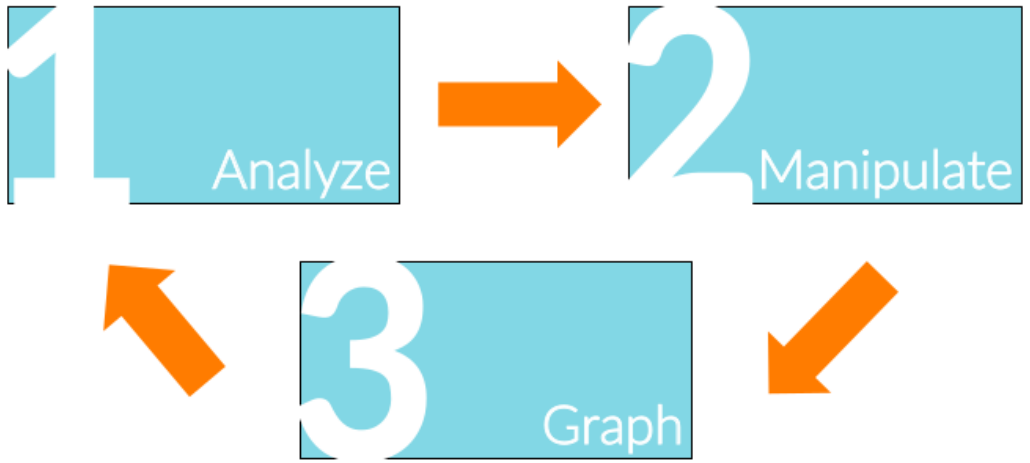
- To provide valuable insights that are **interpretable** and **relevant**
- To give a visual or graphical representation of data / concepts
- To communicate ideas
- To provide an accessible way to see and understand trends, outliers, and patterns in data
- To confirm a hypothesis about the data





Exploratory data analysis (EDA)

- Python is a powerful tool for EDA because the graphics tie in with the functions used to analyze data
- You can create graphs without breaking your train of thought as you explore your data
- Visualization is an iterative process and consists of repeated processes



Exploratory data analysis in Python

Python's capabilities

1. Visualization tools available through a multitude of packages (e.g. `matplotlib`, `plotly`, `seaborn`)
2. The visualizations created are high quality graphics that can be saved as `SVG`, `PNG`, `JPEG`, `BMP`, `PDF`

What we will cover

1. Visualize Costa Rican poverty dataset by using `plotly` package
2. Save our graphs as `HTML` images
3. Interpret the graphs and create a story as if we were to publish a report

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

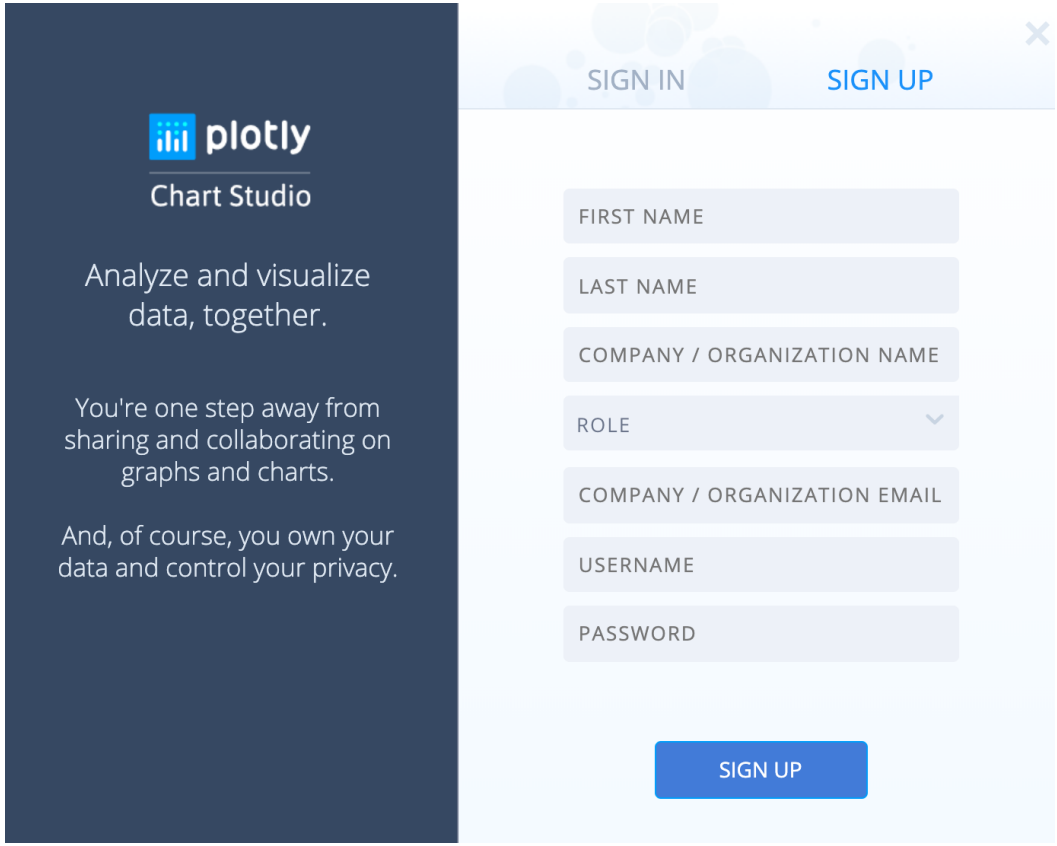
- `cufflinks` is another library used to bind `plotly` directly with a `pandas` dataframe
- This allows us to create easy, interactive visualizations
- We are going to use `3.10.0` version of plotly and `0.16.0` for cufflinks
- You can see all the different plots covered by `cufflinks` ***here***

```
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



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
Describe uses and strengths of plotly and cufflinks packages	✔
Transform Costa Rican dataset for visualizations	
Create basic interactive visualizations using cufflinks	
Visualize multiple metrics using cufflinks	
Create complex interactive visualizations using plotly	
Generate interactive visualizations with transformed summary data	
Construct interactive choropleth maps	
Save your cufflinks and plotly graphs	

Load the dataset

- We are going to use `costa_rica_poverty` dataset today
- Let's load the entire dataset now
- For visualizations, we will be taking a specific subset
- We are now going to use the function `read_csv` to read in our `costa_rica_poverty` dataset

```
household_poverty = pd.read_csv("costa_rica_poverty.csv")
print(household_poverty.head())
```

```
  household_id        ind_id  rooms  ...  age  Target  monthly_rent
0     21eb7fcc1  ID_279628684      3  ...   43       4      190000.0
1     0e5d7a658  ID_f29eb3ddd      4  ...   67       4      135000.0
2     2c7317ea8  ID_68de51c94      8  ...   92       4           NaN
3     2b58d945f  ID_d671db89c      5  ...   17       4      180000.0
4     2b58d945f  ID_d56d6f5f5      5  ...   37       4      180000.0

[5 rows x 84 columns]
```

- The entire dataset consists of `9557` observations and `84` variables

Subsetting data

- Once again, we will explore a subset of this dataset, which includes the following variables:
  - **household id**
  - **ppl_total**
  - **dependency_rate**
  - **num_adults**
  - **monthly rent**
  - **rooms**
  - **age**
  - *Target*

Subsetting data

- Let's subset our data so that we have the variables we need
- Let's name this subset `costa_viz`

```
costa_viz = household_poverty[['household_id',
                               'ppl_total',
                               'dependency_rate',
                               'num_adults',
                               'rooms',
                               'age',
                               'monthly_rent',
                               'Target']]
print(costa_viz.head())
```

```
  household_id  ppl_total  dependency_rate  ...  age  monthly_rent  Target
0     21eb7fcc1          1               37  ...   43      190000.0       4
1     0e5d7a658          1               36  ...   67      135000.0       4
2     2c7317ea8          1               36  ...   92           NaN       4
3     2b58d945f          4               38  ...   17      180000.0       4
4     2b58d945f          4               38  ...   37      180000.0       4

[5 rows x 8 columns]
```

Remove labels

- Let's prepare the data for visualizations by removing any labels, removing variable `household_id`, and keeping the remaining variables

```
costa_viz = costa_viz.drop('household_id', axis = 1)
print(costa_viz.head())
```

```
   ppl_total  dependency_rate  num_adults  rooms  age  monthly_rent  Target
0          1               37           1      3   43      190000.0       4
1          1               36           1      4   67      135000.0       4
2          1               36           1      8   92           NaN       4
3          4               38           2      5   17      180000.0       4
4          4               38           2      5   37      180000.0       4
```

Data prep: clean NAs

- Depending on **subject matter**, missing values might mean something
- Let's define **how to handle columns with NAs**:
  - Drop columns that contain any `NA`s
  - Drop columns with a certain % of `NA`s
  - Impute missing values
  - Convert column with missing values to categorical
- Let's look at the count of NAs by column first:

```
print(costa_viz.isnull().sum())
```

```
ppl_total              0
dependency_rate        0
num_adults             0
rooms                  0
age                    0
monthly_rent        6860
Target                 0
dtype: int64
```

Data cleaning: NAs

- We'll keep `monthly_rent` and **impute missing values** using the mean of the column

```
# Set the dataframe equal to the imputed dataset.
costa_viz = costa_viz.fillna(costa_viz.mean())
# Check how many values are null in monthly_rent.
print(costa_viz.isnull().sum())
```

```
ppl_total          0
dependency_rate    0
num_adults         0
rooms              0
age                0
monthly_rent       0
Target             0
dtype: int64
```

Converting the target variable

- Let's convert our `Target` to a binary class variable, which will help to balance it out
- The four original levels would also increase the complexity of the visualizations and the code
- For this reason, we will convert all 1,2 and 3 to `vulnerable` and 4 to `non_vulnerable`
- The levels translate to 1,2 and 3 as being **vulnerable** households
- Level 4 is **non vulnerable**

```
costa_viz['Target'] = np.where(costa_viz['Target'] <= 3, 'vulnerable', 'non_vulnerable')
```

```
print(costa_viz['Target'].head())
```

```
0    non_vulnerable
1    non_vulnerable
2    non_vulnerable
3    non_vulnerable
4    non_vulnerable
Name: Target, dtype: object
```

Data prep: target

- The next step of our data cleanup is to ensure the target variable is binary and has a label
- Let's look at the `dtype` of `Target`

```
print(costa_viz.Target.dtypes)
```

```
object
```

- We want to convert this to `bool` so that it is a binary class

```
costa_viz["Target"] = np.where(costa_viz["Target"] == "non_vulnerable", True, False)

# Check class again.
print(costa_viz.Target.dtypes)
```

```
bool
```

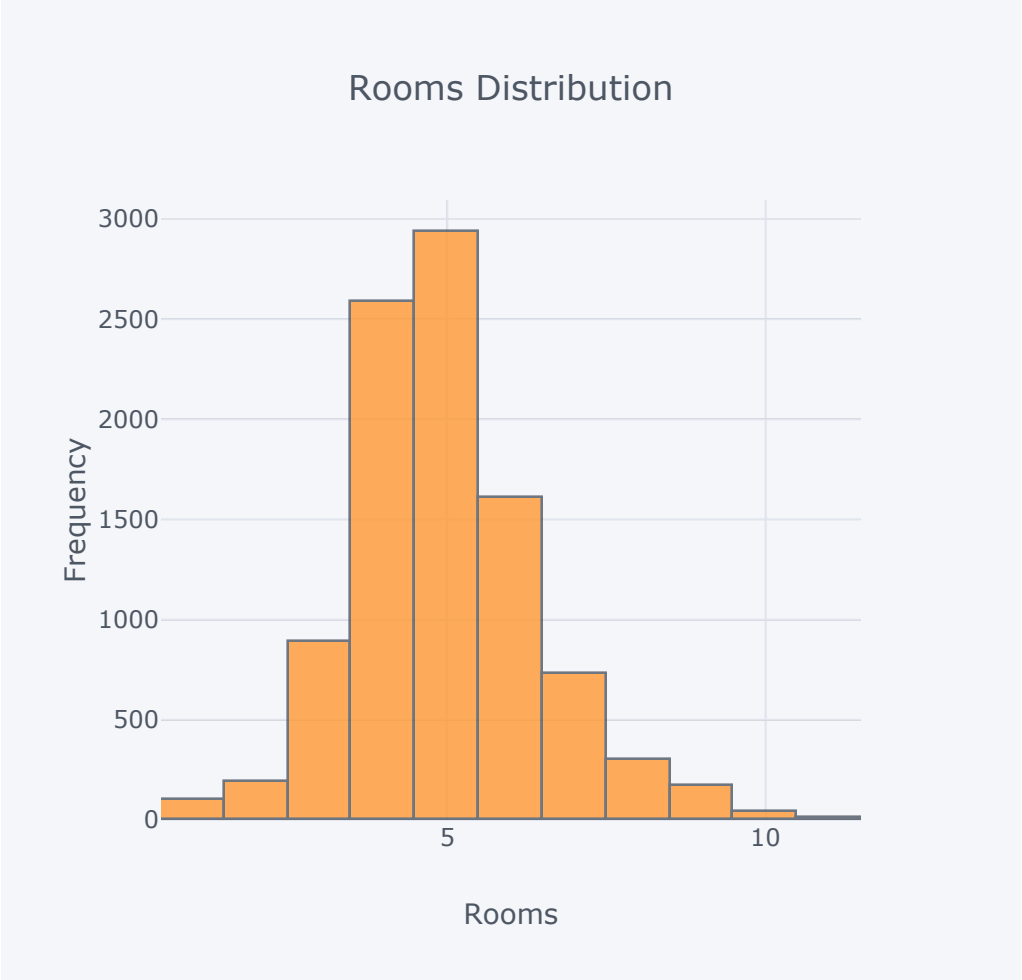
Module completion checklist

Objective	Complete
Describe uses and strengths of plotly and cufflinks packages	✔
Transform Costa Rican dataset for visualizations	✔
Create basic interactive visualizations using cufflinks	
Visualize multiple metrics using cufflinks	
Create complex interactive visualizations using plotly	
Generate interactive visualizations with transformed summary data	
Construct interactive choropleth maps	
Save your cufflinks and plotly graphs	

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

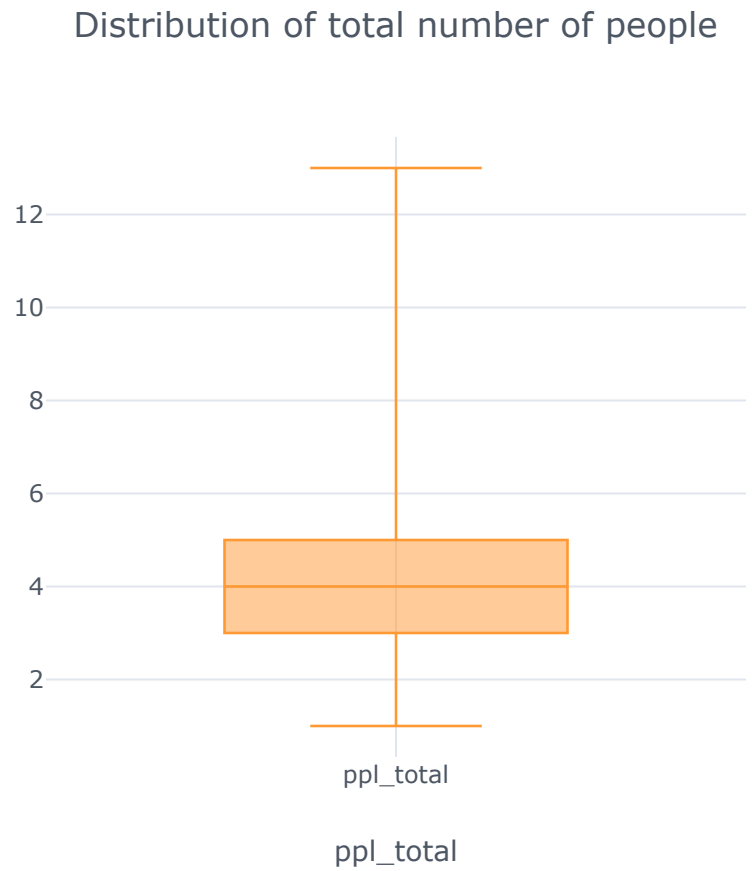
```
costa_viz['rooms'].iplot(kind = 'hist',
                         xTitle = 'Rooms',
                         yTitle = 'Frequency',
                         title = 'Rooms
Distribution')
```



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

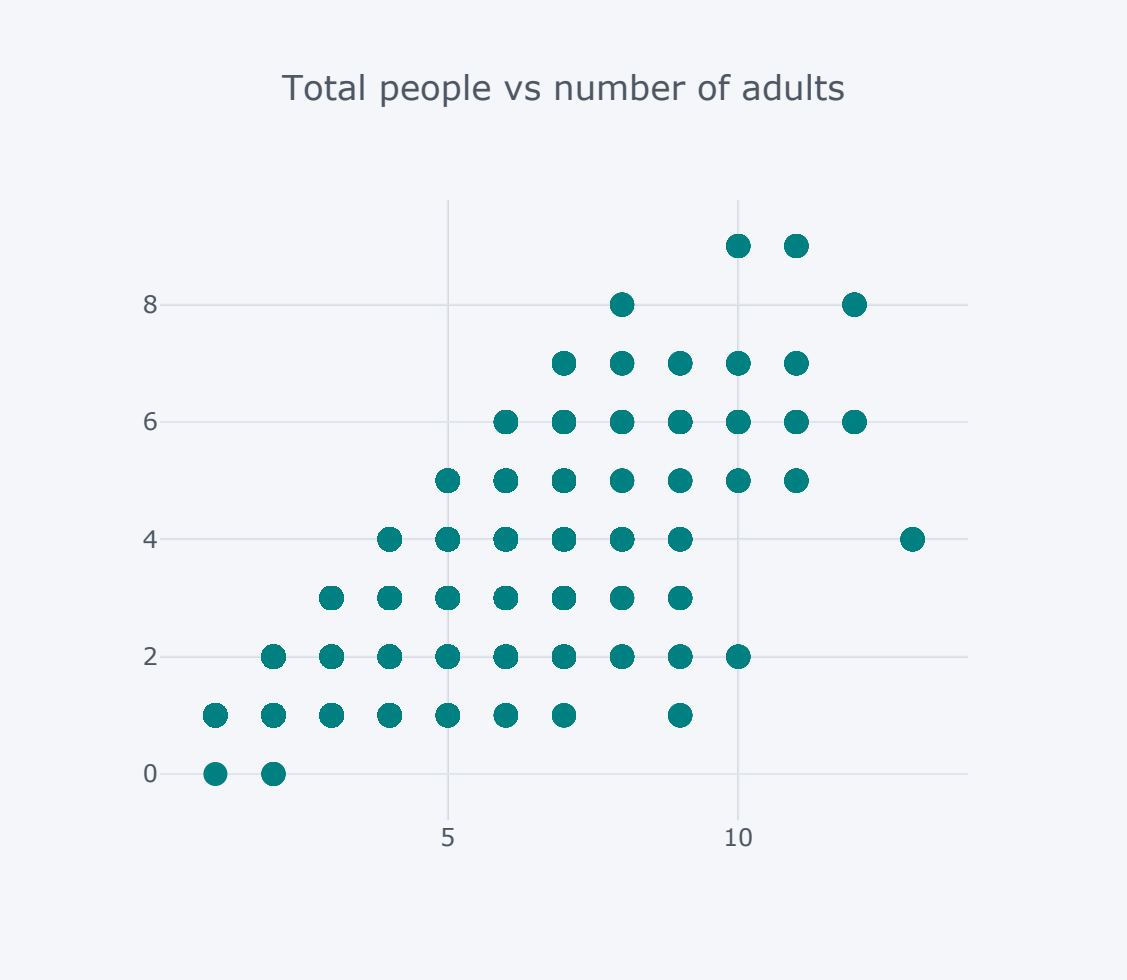
- Now we can create a simple bar chart of the discrete variables
- **How might this be more helpful than the static plot we created earlier?**

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

Bivariate plots: scatterplot

- Scatterplots 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 1

Exercise 1

Module completion checklist

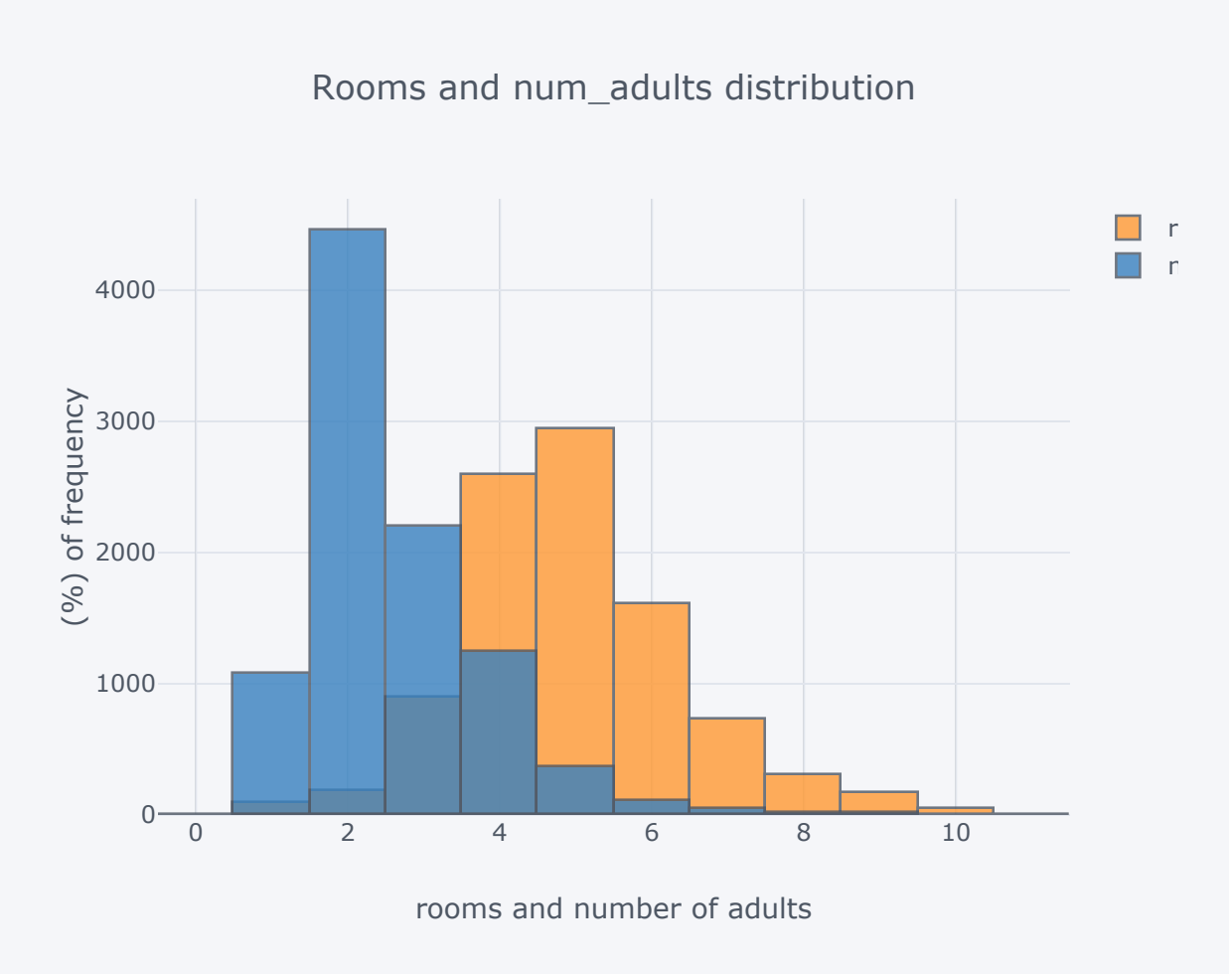
Objective	Complete
Describe uses and strengths of plotly and cufflinks packages	✔
Transform Costa Rican dataset for visualizations	✔
Create basic interactive visualizations using cufflinks	✔
Visualize multiple metrics using cufflinks	
Create complex interactive visualizations using plotly	
Generate interactive visualizations with transformed summary data	
Construct interactive choropleth maps	
Save your cufflinks and plotly graphs	

Multivariate plots: overlaid histogram

- We can also plot histograms of different variables on top of each other

```
# Plot overlaid histograms.
costa_viz[['rooms',
'num_adults']].iplot(

kind = 'hist',

barmode = 'overlay',

xTitle = 'rooms and number of adults',

yTitle = '(%) of frequency',

title = 'Rooms and num_adults
distribution')
```

- **How might this be useful to explore your data?**



Multivariate plots: boxplot with multiple variables

```
# Multiple boxplots.
costa_viz[['ppl_total','dependency_rate','num_adults','rooms','age']].iplot(
                                                                    kind = 'box',
                                                                    title = 'Costa Rican
distribution')
```

Multivariate plots: 3D scatterplot

- It is similar to a scatterplot, but has **three dimensions** instead

```
costa_viz[['age','monthly_rent','rooms']].iplot(kind = 'surface',
                                                colorscale = 'bupu')
```

Customize colors

- You can also use any color by providing its ***RGB code***
- The list of named colors is also available in this handy ***reference table / color map visualization***

Customize color: map colors

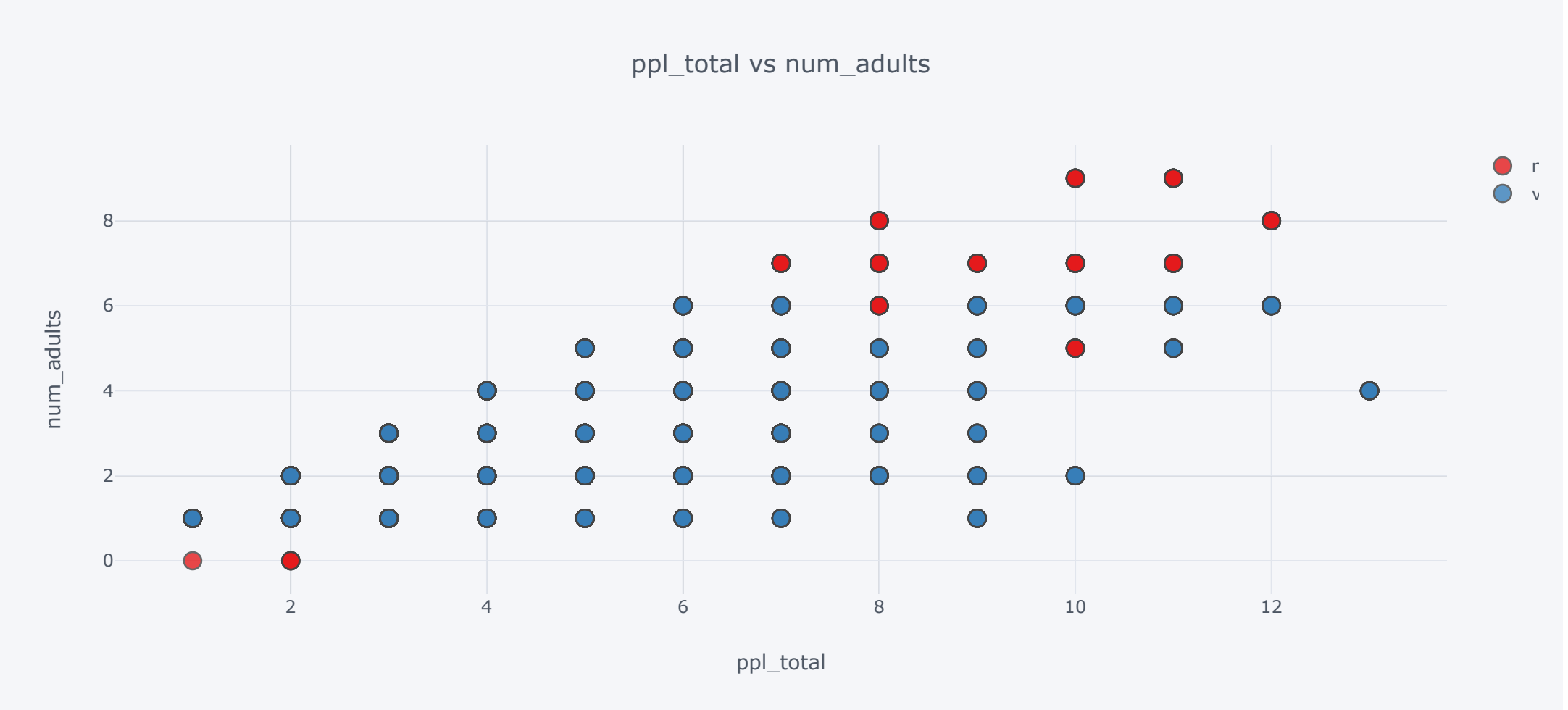
- When plotting data using scatterplots, we might want to see values corresponding to 2 or more distinct categories
- We can achieve that by coloring observations that belong to different categories
- A `cufflinks` scatterplot doesn't accept `bool` values, so let's convert `Target` to type `string` and save it to another variable `Target_class` just for this plot

```
costa_viz["Target_class"] = np.where(costa_viz["Target"] == True, 'non-vulnerable', 'vulnerable')
```

```
costa_viz.iplot(kind='scatter',
                x = 'ppl_total',
                y = 'num_adults',
                categories = 'Target_class',
                title = 'ppl_total vs num_adults',
                xTitle='ppl_total',
                yTitle='num_adults',
                mode='markers',
                colorscale = 'set1')
```

Customize color: map colors

- What are some trends we can identify from this plot?



Knowledge check 2

Exercise 2

Module completion checklist

Objective	Complete
Describe uses and strengths of plotly and cufflinks packages	✔
Transform Costa Rican dataset for visualizations	✔
Create basic interactive visualizations using cufflinks	✔
Visualize multiple metrics using cufflinks	✔
Create complex interactive visualizations using plotly	
Generate interactive visualizations with transformed summary data	
Construct interactive choropleth maps	
Save your cufflinks and plotly graphs	

Choosing between cufflinks and plotly

- Cufflinks is designed for simple one-line charting with Pandas and Plotly
- Hence, all of the Plotly chart attributes are not directly assignable in the `df.iplot` call signature
- This makes grouping and plotting multiple graphs difficult
- **Plotly's native syntax** is better for highly customized graphs

Graphing with plotly

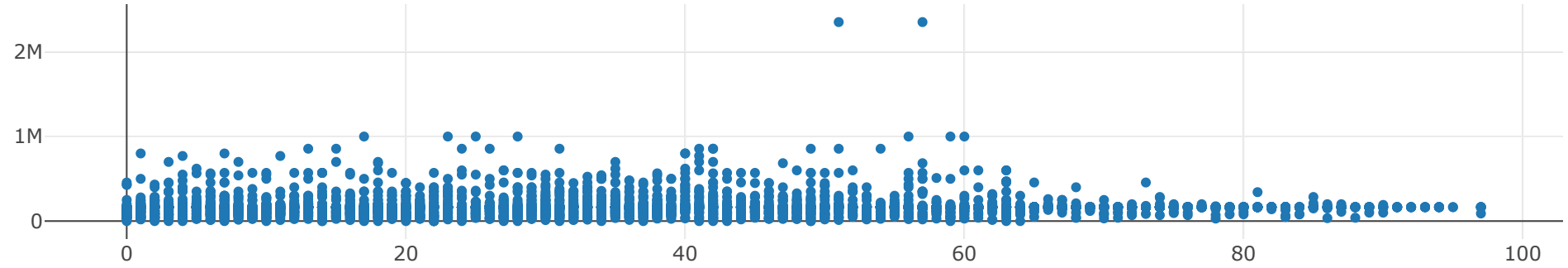
- Plotly charts have two main components: **data** and **layout**
- All traces/plots are saved in the **data** component, while all customizable attributes are in the **layout** component
- We can view the plotly cheat sheet ***here***
- Let's plot a simple scatterplot using plotly's syntax

Simple scatterplot using plotly

- `trace`: an object that contains a collection of data and specifies how we want that data plotted
  - `x`: x-axis
  - `y`: y-axis
  - `mode`: drawing mode for the trace - could be `lines`, `markers`, etc.
- `data` : list where we add traces into it
- `iplot()` : plots the figure (fig) that is created by data and layout

```
trace = go.Scatter(
    x = costa_viz['age'],
    y = costa_viz['monthly_rent'],
    mode = 'markers'
)

data = [trace]

iplot(data, filename='basic-scatter.html')
```

Simple scatterplot using plotly



- Now let's add a **layout** to this plot
- We can specify the labels of the axes and the title in layout

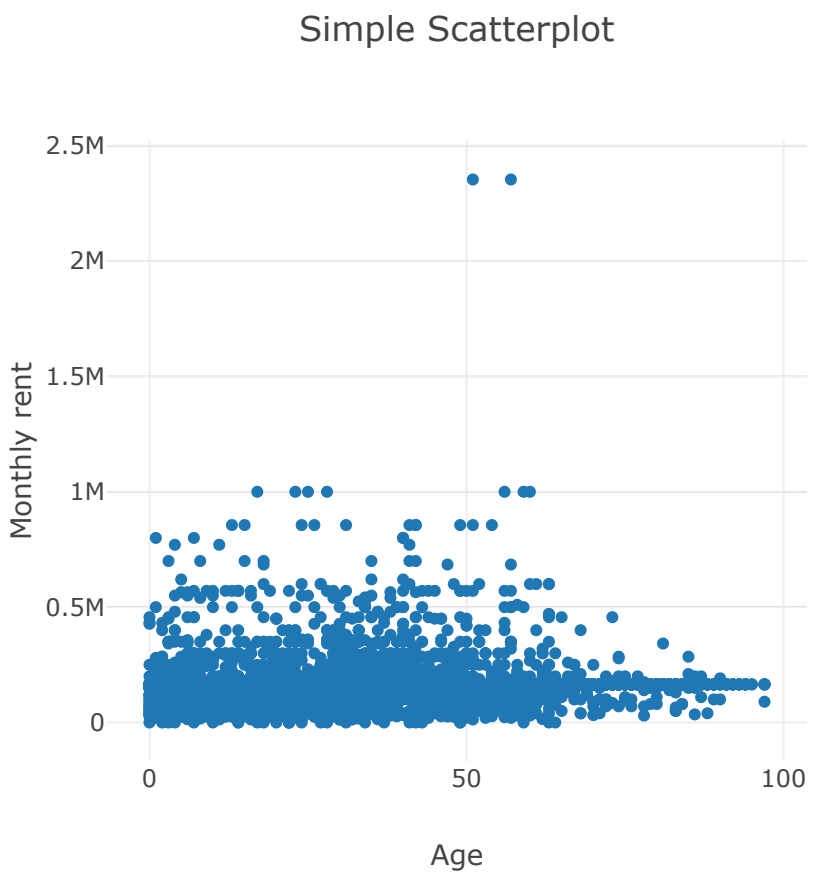
Simple scatterplot using plotly

- `layout`: dictionary of the layout attributes
  - `title`: title of the graph
  - `xaxis`: dictionary with x-axis layout attributes
  - `yaxis`: dictionary with y-axis layout attributes
- `fig`: dictionary of `data` and `layout`
- `iplot()`: plots the figure (fig) that is created by data and layout

```
# Create a trace.
trace = go.Scatter(
    x = costa_viz['age'],
    y = costa_viz['monthly_rent'],
    mode = 'markers'
)

data = [trace]

layout = dict(title = 'Simple Scatterplot',
              xaxis = dict(title = 'Age',
                           zeroline = False),
              yaxis = dict(title = 'Monthly
rent',
                           zeroline = False))

fig = dict(data=data, layout=layout)


# Plot and embed in iPython notebook!
iplot(fig, filename='basic-scatter')
```

Simple scatterplot using plotly

```
# Create a trace.
trace = go.Scatter(
    x = costa_viz['age'],
    y = costa_viz['monthly_rent'],
    mode = 'markers')

data = [trace]

layout = dict(title = 'Simple Scatterplot',
              xaxis = dict(title = 'Age',
                           zeroline = False),
              yaxis = dict(title = 'Monthly
rent',
                           zeroline = False)
              )

fig = dict(data = data, layout = layout)


# Plot and embed in iPython notebook!
iplot(fig, filename = 'basic-scatter')
```



Simple bar-chart using plotly

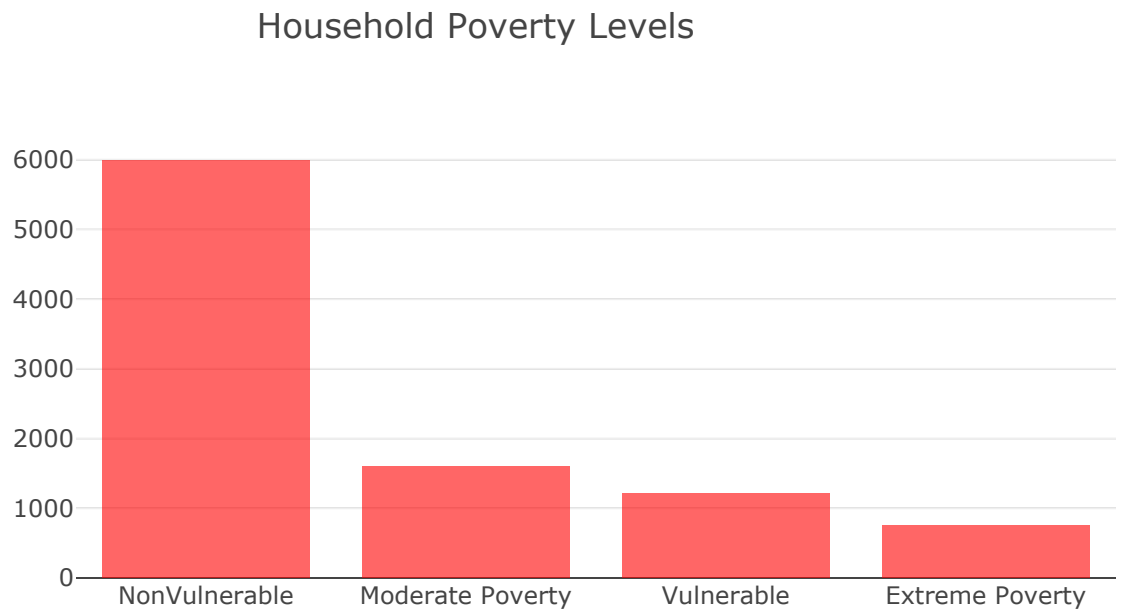
- We're going to create a barbchart with the original four levels of our Target variable from the `household_poverty` dataset
- Here are the new attributes used for this plot:
- `trace`:
  - `marker`: dictionary which sets the style of the plot
  - sets the `color` and `opacity`
- `layout`:
  - `margin`: sets the margin of the graph
  - `l` sets the left margin

```
trace = go.Bar(
                y =
household_poverty.Target.value_counts(),
                x = ["NonVulnerable", "Moderate
Poverty", "Vulnerable", "Extreme Poverty"],
                marker = dict(color = 'red',
                              opacity = 0.6))

layout = dict(title = "Household Poverty Levels",
              margin=dict(l = 200),
              width = 800,
              height = 400)

data = [trace]
```

- We can view plotly attribute reference chart ***here***

Simple bar chart using plotly

```
fig = go.Figure(data = data, layout = layout)
iplot(fig, filename = 'basic-barchart')
```



Module completion checklist

Objective	Complete
Describe uses and strengths of plotly and cufflinks packages	✔
Transform Costa Rican dataset for visualizations	✔
Create basic interactive visualizations using cufflinks	✔
Visualize multiple metrics using cufflinks	✔
Create complex interactive visualizations using plotly	✔
Generate interactive visualizations with transformed summary data	
Construct interactive choropleth maps	
Save your cufflinks and plotly graphs	

Compound visualizations: layered plots

- Now let's prepare our data to create a layered bar chart

```
grouped = costa_viz.groupby('Target')
# Compute mean on the listed variables using the grouped data.
costa_grouped_mean = grouped.mean()[['ppl_total', 'dependency_rate', 'num_adults', 'rooms', 'age']]
print(costa_grouped_mean)
```

```
        ppl_total  dependency_rate  num_adults     rooms        age
Target
False    4.358607        26.011233    2.388093  4.533839  31.314238
True     3.796531        25.425284    2.713809  5.205971  36.078886
```

```
# Reset index of the dataset.
costa_grouped_mean = costa_grouped_mean.reset_index()
print(costa_grouped_mean)
```

```
   Target  ppl_total  dependency_rate  num_adults     rooms        age
0   False   4.358607        26.011233    2.388093  4.533839  31.314238
1    True   3.796531        25.425284    2.713809  5.205971  36.078886
```

Compound visualizations: layered plots

```
costa_grouped_mean_long = pd.melt(costa_grouped_mean,        #<- wide dataset
                                  id_vars = ['Target'],      #<- identifying variable
                                  var_name = 'metric',       #<- contains col names of wide data
                                  value_name = 'mean')       #<- contains values from above columns
print(costa_grouped_mean_long)
```

```
   Target           metric       mean
0   False        ppl_total   4.358607
1    True        ppl_total   3.796531
2   False  dependency_rate  26.011233
3    True  dependency_rate  25.425284
4   False       num_adults   2.388093
5    True       num_adults   2.713809
6   False            rooms   4.533839
7    True            rooms   5.205971
8   False              age  31.314238
9    True              age  36.078886
```

Compound visualizations: layered plots

```
costa_true_means = costa_grouped_mean_long.query('Target == True')[['metric', 'mean']]
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
```

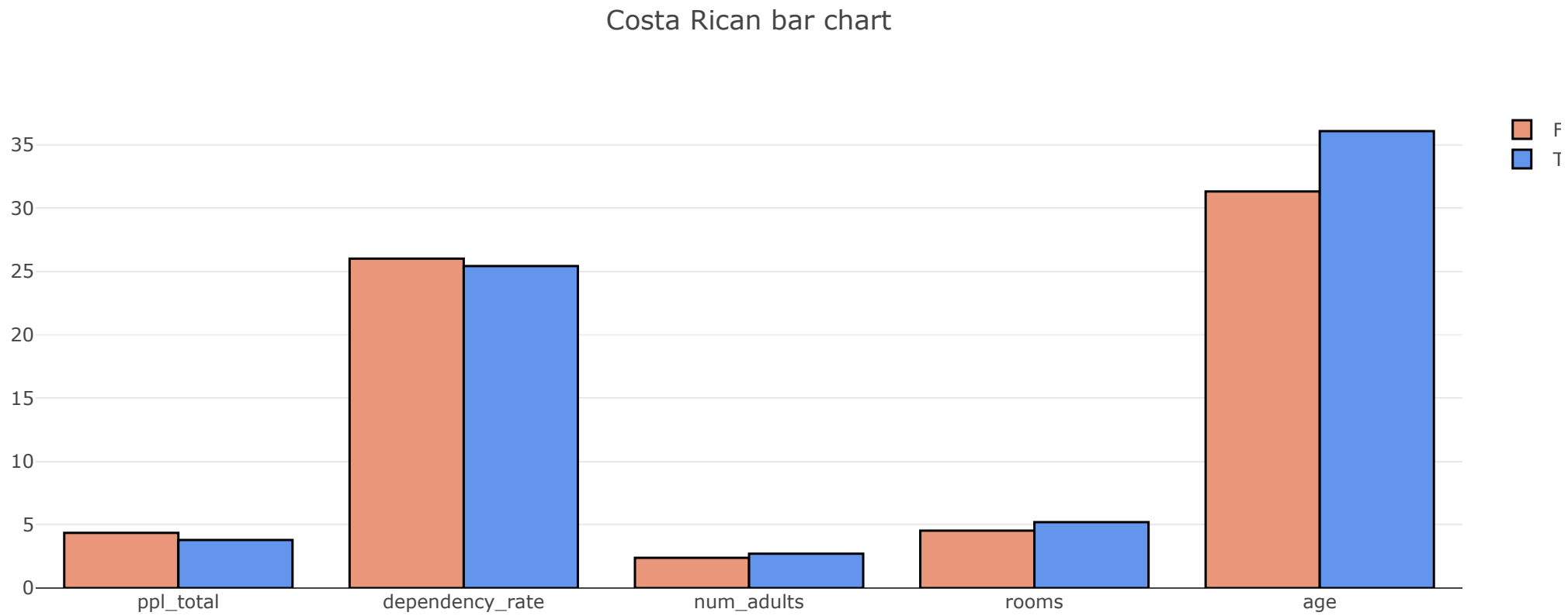
```
# Let's get the `Target` = `False` mean data.
costa_false_means = costa_grouped_mean_long.query('Target == False')[['metric', 'mean']]
print(costa_false_means)
```

```
            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
```

Compound visualizations: create traces for layered plots

```
# Create trace1.
trace1 = go.Bar(
                x = costa_false_means['metric'],
                y = costa_false_means['mean'],
                name = "False",
                marker = dict(color = 'darksalmon',
                              line=dict(color='rgb(0,0,0)',width=1.5)))
# Create trace2.
trace2 = go.Bar(x = costa_true_means['metric'],
                y = costa_true_means['mean'],
                name = "True",
                marker = dict(color = 'cornflowerblue',
                              line=dict(color='rgb(0,0,0)',width=1.5)))


data = [trace1, trace2]

layout = dict(title = 'Costa Rican bar chart',
              barmode = "group")
fig = go.Figure(data = data, layout = layout)
iplot(fig)
```

Compound visualizations: layered plots



Compound visualizations: subplots

- Let's use the plots created before to build a subplot so that we can display multiple charts at one time!

```
trace1 = go.Bar(
                y = household_poverty.Target.value_counts(),
                x = ["NonVulnerable", "Moderate Poverty", "Vulnerable", "Extreme Poverty"],
                name = "trace1",
                marker = dict(color = 'red', opacity = 0.6)
                )

trace2 = go.Scatter(
    x = costa_viz['age'],
    y = costa_viz['monthly_rent'],
    name = 'trace2',
    mode = 'markers'
)

trace3 = go.Bar(
                 x = costa_false_means['metric'],
                 y = costa_false_means['mean'],
                 name = "trace3",
                 marker = dict(color = 'darksalmon',
                 line = dict(color='rgb(0,0,0)',width = 1.5)

                         ))
```

Compound visualizations: subplots

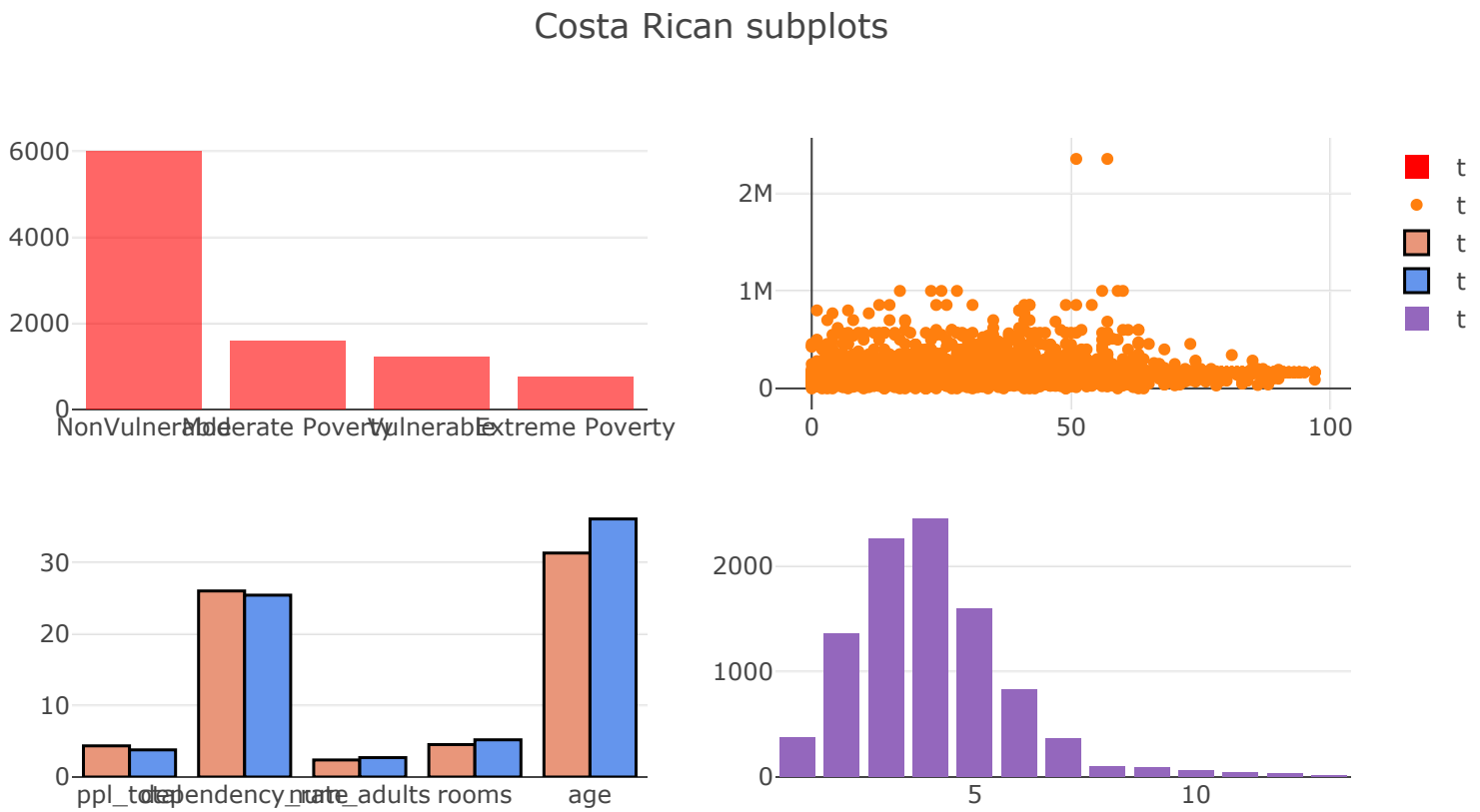
```
trace4 = go.Bar(x = costa_true_means['metric'],
                y = costa_true_means['mean'],
                name = "trace4",
                marker = dict(color = 'cornflowerblue',
                              line = dict(color ='rgb(0,0,0)',width = 1.5
                                          )))

trace5 = go.Histogram(
                    x = costa_viz['ppl_total'],
                    name = 'trace5')

data = [trace1, trace2, trace3, trace4, trace5]
```

Compound visualizations: subplots

- Next, we create the subplots and append the traces to them

```
fig = tools.make_subplots(rows=2, cols=2)

fig.append_trace(trace1, 1, 1)
fig.append_trace(trace2, 1, 2)
fig.append_trace(trace3, 2, 1)
fig.append_trace(trace4, 2, 1)
fig.append_trace(trace5, 2, 2)

fig['layout'].update(height = 600, width = 1000, title = 'Costa Rican subplots')
iplot(fig, filename ='simple-subplot')
```

Compound visualizations: subplots



Compound visualizations: labeling axes

- We use the same traces we used to create the subplots before

```
data = [trace1, trace2, trace3, trace4, trace5]

fig = tools.make_subplots(rows=2, cols=2, subplot_titles=('Poverty level bar chart', 'Age vs monthly
rent',
                                                          'Layered bar chart', 'Total people
distribution'))

fig.append_trace(trace1, 1, 1)
fig.append_trace(trace2, 1, 2)
fig.append_trace(trace3, 2, 1)
fig.append_trace(trace4, 2, 1)
fig.append_trace(trace5, 2, 2)

fig['layout']['xaxis1'].update(title='Poverty levels')
fig['layout']['yaxis1'].update(title='Count')

fig['layout']['xaxis2'].update(title='Age')
fig['layout']['yaxis2'].update(title='Monthly rent')

fig['layout']['yaxis3'].update(title='Mean values')

fig['layout']['xaxis4'].update(title='ppl_total')
fig['layout']['yaxis4'].update(title='Frequency')

fig['layout'].update(height=800, width=1000, title='Costa Rican subplots')

iplot(fig, filename='customized-subplot')
```

Compound visualizations: labeling axes

Inset plots

- An inset plot is a layer which is added to an existing layer in a graph window
- This means that we **inset a smaller graphic representation within a larger one**
- These plots are visually strong, making it easy to **reference both figures**
- We will use the two bar charts we created earlier to build an inset plot

Inset plots: create traces

- First, let's create the traces of the bar charts
- `trace1` and `trace2` are the traces of the layered bar chart we created earlier

```
# Create trace1.
trace1 = go.Bar(
                x = costa_false_means['metric'],
                y = costa_false_means['mean'],
                name = "False",
                marker = dict(color =
'darksalmon',

line=dict(color='rgb(0,0,0)',width=.5)))
# Create trace2.
trace2 = go.Bar(x = costa_true_means['metric'],
                y = costa_true_means['mean'],
                name = "True",
                marker = dict(color =
'cornflowerblue',

line=dict(color='rgb(0,0,0)',width=.5)))
```

Inset plots: create traces

- `trace3` is the simple bar chart of the four levels of our `Target` variable, which we wish to inset into the layered plot
- Notice the `xaxis` and `yaxis` of `trace3` has been set as `x3` and `y3`
- We will use them to set the margins of the smaller plot in `layout`

```
# Create trace3.
trace3 = go.Bar(
                y =
household_poverty.Target.value_counts(),
                x = ["NonVulnerable", "Moderate
Poverty", "Vulnerable", "Extreme Poverty"],
                name = 'Poverty level',
                marker=dict(color='red',
opacity=0.6),
                xaxis='x3',
                yaxis='y3')

data = [trace1, trace2, trace3]
```

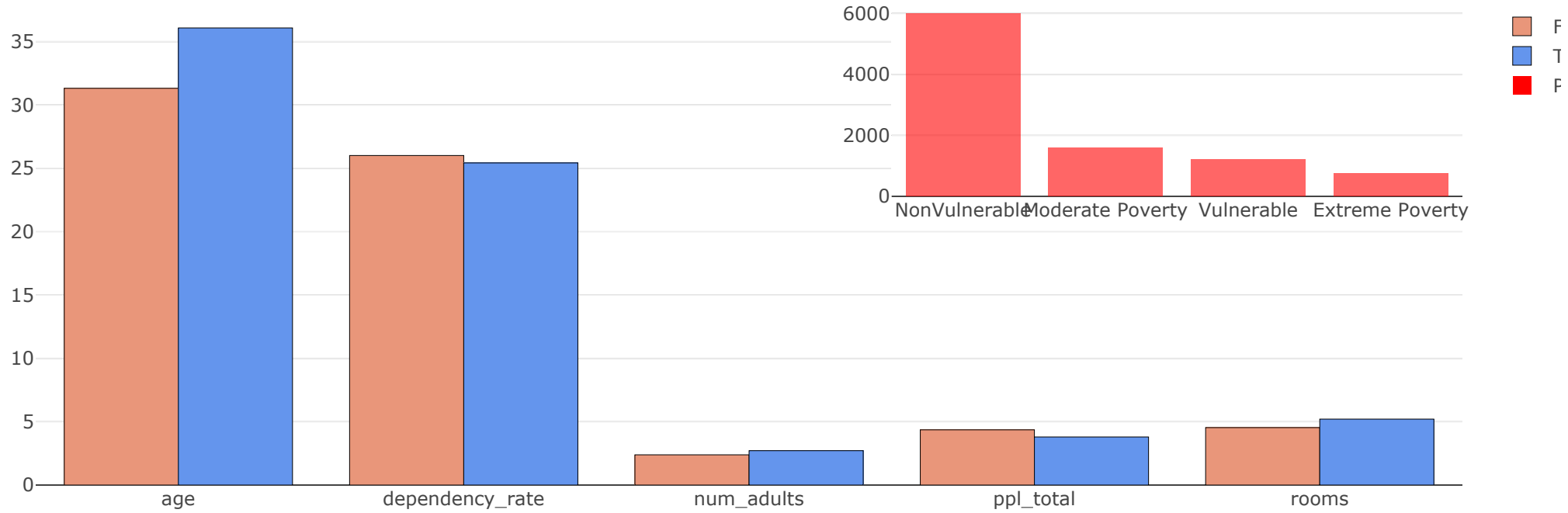
- Next, we will create the `layout`

Inset plots: create layout

- We set the `categoryorder` as ascending so that the two plots do not overlap
- It sets the categories alphabetically in ascending order
- `domain` sets the **horizontal** and **vertical** domain of the trace
- We also use `anchor` to bind the axes to their respective positions

```
layout = go.Layout(
     barmode = "group",
     xaxis = dict(
     categoryorder = "category ascending"),
     xaxis3=dict(
        domain=[0.6, 1],
        anchor='y3',

    ),
    yaxis3=dict(
        domain=[0.6, 1],
        anchor='x3'
    )
)
fig = go.Figure(data=data, layout=layout)
iplot(fig, filename='simple-inset')
```

Inset plots


35
30
25
20
15
10
5
0
age
dependency_rate
num_adults
ppl_total
rooms
6000
4000
2000
0
NonVulnerable
Moderate Poverty
Vulnerable
Extreme Poverty
F
T
F

Knowledge check 3

Exercise 3

Module completion checklist

Objective	Complete
Describe uses and strengths of plotly and cufflinks packages	✔
Transform Costa Rican dataset for visualizations	✔
Create basic interactive visualizations using cufflinks	✔
Visualize multiple metrics using cufflinks	✔
Create complex interactive visualizations using plotly	✔
Generate interactive visualizations with transformed summary data	✔
Construct interactive choropleth maps	
Save your cufflinks and plotly graphs	

Interactive maps

- **Choropleth maps** provide an easy way to visualize how a **measurement varies across a geographic area** or **show the level of variability within a region**
- In this map, we color the area within the border of a region based on an **aggregated metric of that region**
- You can view the different types of maps in plotly ***here***
- Plotly maps are easy to implement, but creating maps based on custom geographical areas is complicated and outside the scope of this class

Interactive maps

- We do not have a explicit region variable in our Costa Rican dataset, so let's use another dataset named `2014_world_gdp_with_codes.csv`
- It has three variables:
  - **Country**
  - **GDP (In billions)**
  - **Code**

Read in our dataset

- Let's read in our dataset directly from plotly's github

```
gdp =
pd.read_csv('https://raw.githubusercontent.com/plotly/datasets/master/2014_world_gdp_with_codes.csv')
```

```
print(gdp.head())
```

```
          COUNTRY  GDP (BILLIONS) CODE
0     Afghanistan           21.71  AFG
1         Albania           13.40  ALB
2         Algeria          227.80  DZA
3  American Samoa            0.75  ASM
4         Andorra            4.80  AND
```

Interactive maps using `choropleth`

- Let's create a simple interactive map using `CODE` and `GDP`
- We use variable `COUNTRY` as the `text` element for each data point

```
data = [go.Choropleth(
    locations = gdp['CODE'],
    z = gdp['GDP (BILLIONS)'],
    text = gdp['COUNTRY'],
    )]
iplot(data, filename = 'simple-map')
```

- **What can we learn from this visualization?**

Customizing map

- We can change the colorscale from **grey** to **blue** by setting the GDP levels as shown below
- We can also specify the `title` and `tickprefix` of the **colorbar**

```
data = [go.Choropleth(
    locations = gdp['CODE'],
    z = gdp['GDP (BILLIONS)'],
    text = gdp['COUNTRY'],
    colorscale = [
        [0, "rgb(5, 10, 172)"],
        [0.5, "rgb(70, 100, 245)"],
        [1, "rgb(220, 220, 220)"]
    ],
    autocolorscale = False,
    reversescale = True,
    marker = go.choropleth.Marker(
        line = go.choropleth.marker.Line(
            color = 'rgb(180,180,180)',
            width = 0.5
        )),
    colorbar = go.choropleth.ColorBar(
        tickprefix = '$',
        title = 'GDP<br>Billions US$'),
)]

layout = go.Layout(
    title = go.layout.Title(
        text = '2014 Global GDP'
    )
)
```

Customizing map

```
fig = go.Figure(data = data, layout = layout)
iplot(fig, filename = 'customized-map')
```

Customizing map: change projection

- We can also change the projection type in the layout of our map
- Adding attribute `geo` to our layout specifies the `scope` and `projection` as shown

```
data = [go.Choropleth(
    locations = gdp['CODE'],
    z = gdp['GDP (BILLIONS)'],
    text = gdp['COUNTRY'],
    colorscale = [
        [0, "rgb(5, 10, 172)"],
        [0.5, "rgb(70, 100, 245)"],
        [1, "rgb(220, 220, 220)"]
    ],
    autocolorscale = False,
    reversescale = True,
    marker = go.choropleth.Marker(
        line = go.choropleth.marker.Line(
            color = 'rgb(180,180,180)',
            width = 0.5
        )),
    colorbar = go.choropleth.ColorBar(
        tickprefix = '$',
        title = 'GDP<br>Billions US$'),
)]
```

```
layout = go.Layout(
    title = go.layout.Title(
         text = '2014 Global GDP'
    ),
    geo = go.layout.Geo(
        scope = 'world',
        projection =
go.layout.geo.Projection(type = 'orthographic')
    )
)

fig = go.Figure(data = data, layout = layout)
iplot(fig, filename = 'world-map')
```

Customizing map: change projection

Interactive USA map

- Plotly allows us to set the layout of the map we wish to plot
- We can create maps **outlining the world** or **only a specific continent**
- Plotly choropleth maps are intuitive in setting the data into the map, as long as there is a specified **country code** or **state code**

scope (`enumerated : "world" | "usa" | "europe" | "asia" | "africa" | "north america" | "south america"`)

default: `"world"`

Set the scope of the map.

Interactive maps: USA choropleth map

- We will need another dataset to illustrate a US map
- We will read in the dataset and save as `state_data`

```
state_data = pd.read_csv('state_data.csv')
```

```
print(state_data.head())
```

```
   Population  Income  Illiteracy  Life Exp  ...  Frost    Area       State  code
0        3615    3624         2.1     69.05  ...     20   50708     Alabama    AL
1         365    6315         1.5     69.31  ...    152  566432      Alaska    AK
2        2212    4530         1.8     70.55  ...     15  113417     Arizona    AZ
3        2110    3378         1.9     70.66  ...     65   51945    Arkansas    AR
4       21198    5114         1.1     71.71  ...     20  156361  California    CA

[5 rows x 10 columns]
```

```
# Create custom colorscale.
scl = [
    [0.0, 'rgb(242,240,247)'],
    [0.2, 'rgb(218,218,235)'],
    [0.4, 'rgb(188,189,220)'],
    [0.6, 'rgb(158,154,200)'],
    [0.8, 'rgb(117,107,177)'],
    [1.0, 'rgb(84,39,143)']
]
```

Interactive maps: USA choropleth map

```
data = [go.Choropleth(
    colorscale = scl,
    autocolorscale = False,
    locations = state_data['code'],
    z = state_data['Income'],
    locationmode = 'USA-states',
    text = state['State'],
    marker = go.choropleth.Marker(
        line = go.choropleth.marker.Line(
            color = 'white',
            width = 2
        )),
    colorbar = go.choropleth.ColorBar(
        title = "USD")
)]

layout = go.Layout(
    title = go.layout.Title(
        text = 'US State Income'
    ),
    geo = go.layout.Geo(
        scope = 'usa',
        projection = go.layout.geo.Projection(type = 'albers usa'),
        showlakes = True,
        lakecolor = 'rgb(255, 255, 255)'),
)

fig = go.Figure(data = data, layout = layout)
iplot(fig, filename = 'usa-map')
```

Interactive maps: USA choropleth map

Module completion checklist

Objective	Complete
Describe uses and strengths of plotly and cufflinks packages	✔
Transform Costa Rican dataset for visualizations	✔
Create basic interactive visualizations using cufflinks	✔
Visualize multiple metrics using cufflinks	✔
Create complex interactive visualizations using plotly	✔
Generate interactive visualizations with transformed summary data	✔
Construct interactive choropleth maps	✔
Save your cufflinks and plotly graphs	

Saving your cufflinks plots

- You will be saving all of your graphs as an `HTML` file in the `plots` folder
- Set `asFigure` as `True` and save it using `py.offline.plot()`
- Change your directory to `plot_dir` and save your cufflinks plots as shown below:

```
os.chdir(plot_dir)

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

plotly.offline.plot(fig, filename = "pyplot-hist.html", auto_open = False)
```

Saving your plotly plots

- Save your plotly graphs using `plot()` and specify the file name using `HTML` extension

```
trace = go.Scatter(
    x = costa_viz['age'],
    y = costa_viz['monthly_rent'],
    mode = 'markers')

data = [trace]

plotly.offline.plot(data, filename = 'basic-scatter.html')
```

Knowledge check 4

Exercise 4

Module completion checklist

Objective	Complete
Describe uses and strengths of plotly and cufflinks packages	✔
Transform Costa Rican dataset for visualizations	✔
Create basic interactive visualizations using cufflinks	✔
Visualize multiple metrics using cufflinks	✔
Create complex interactive visualizations using plotly	✔
Generate interactive visualizations with transformed summary data	✔
Construct interactive choropleth maps	✔
Save your cufflinks and plotly graphs	✔

Workshop: next steps!

- **Now we are at the workshop portion of the day**

Workshops are to be completed in the afternoon either with a dataset for a capstone project or with another dataset of your choosing. Make sure to annotate and comment your code so that it is easy for others to understand what you are doing. This is an exploratory exercise to get you comfortable with the content we discussed today

- Create basic plots using cufflinks and then head on to plots with multiple metrics
- Construct complex visualizations using plotly's native syntax
- Map data to choropleth maps and choose its scope and projection

This completes our module

Congratulations!