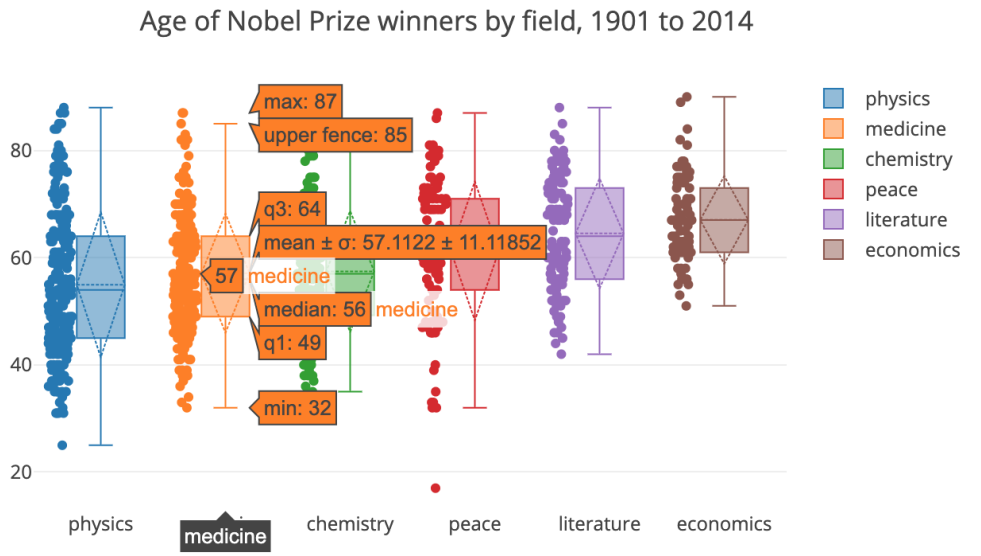
# DATA SOCIETY®

## Day 4 - Data visualization with Python

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

# Welcome back

- We ended our last session learning how to use `cufflinks` in conjunction with `plotly`
- While we wait for class to start, ***take a look at this showcase*** of the kinds of visualizations you can produce using plotly's native syntax
- Make sure to **hover over** each visualization to get a look at the interactive component



# Review

**So far, we have:**

- Explored the plotly and cufflinks packages
- Created some basic visualizations in python using cufflinks

**Today, we will:**

- Visualize multiple metrics using cufflinks
- Create interactive visualizations using plotly
- Generate interactive visualizations with a transformed summary data
- Create inset plots

# Module completion checklist

| Objective | Complete |
| --- | --- |
| Visualize multiple metrics using cufflinks | |
| Create interactive visualizations using plotly | |
| Generate interactive visualizations with transformed summary data | |
| Create inset plots | |
| Save your plotly and cufflinks 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 `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")
```

# Loading packages

- Load the packages we will be using

```
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
import warnings
warnings.filterwarnings("ignore")
```

# Working directory

- Set working directory to `data_dir`

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

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

```
~/Desktop/skillsoft-data-viz-with-python/data
```

# Loading datasets

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

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

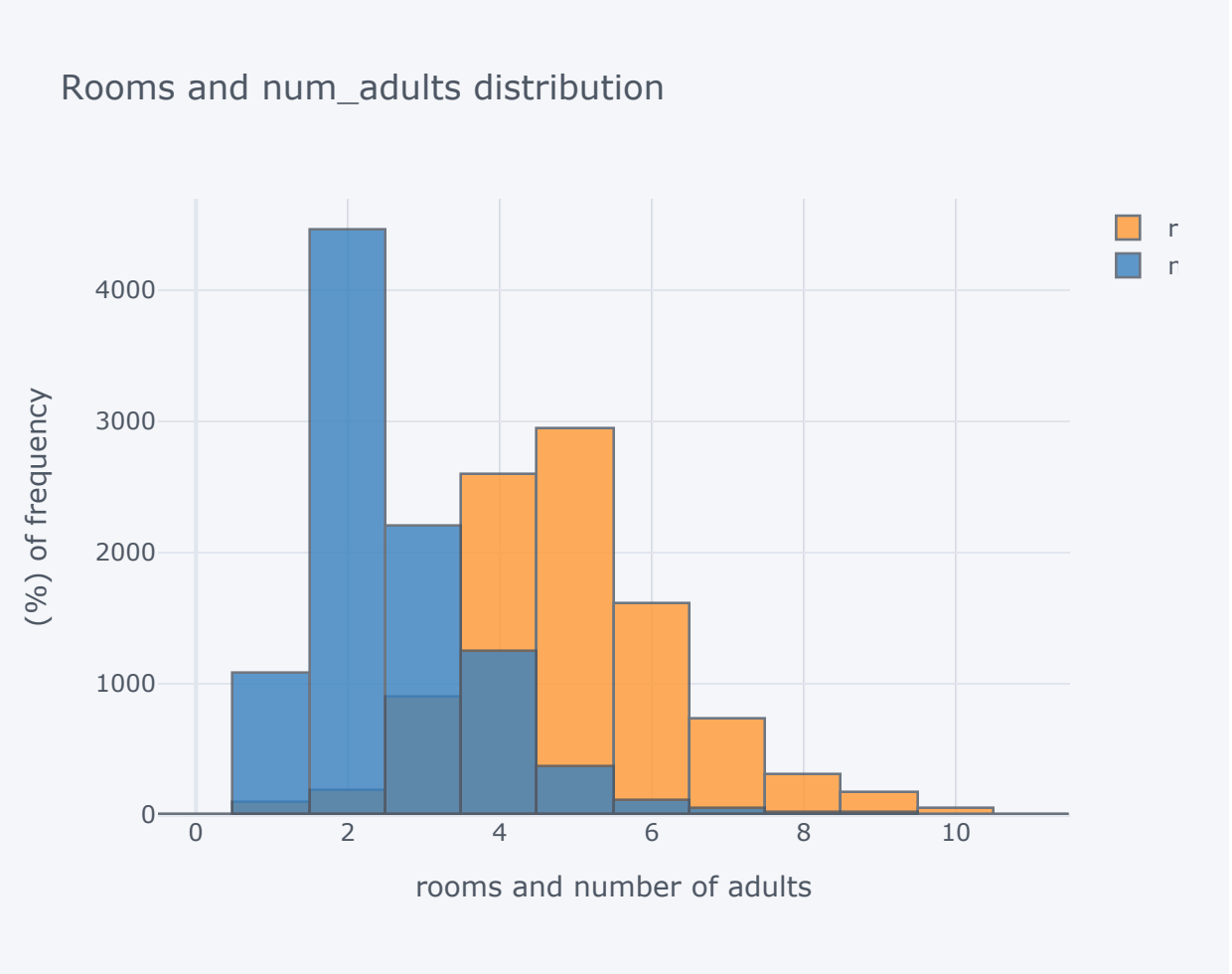
```
  household_id        ind_id  rooms  ...  age  Target  monthly_rent
0    21eb7fcc1  ID_279628684      3  ...   43       4      190000.0
1    0e5d7a658  ID_f29eb3ddd      4  ...   67       4      135000.0
2    2c7317ea8  ID_68de51c94      8  ...   92       4           NaN
3    2b58d945f  ID_d671db89c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

[5 rows x 84 columns]
```

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

# Multivariate plots: overlaid histogram

- Using cufflinks, we can plot histograms of different variables on top of each other
- Before we can do this, we need to set up the initialization for offline plotting

```
init_notebook_mode(connected = True)
cf.go_offline()

# 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')
```



# 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')
```

# 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***

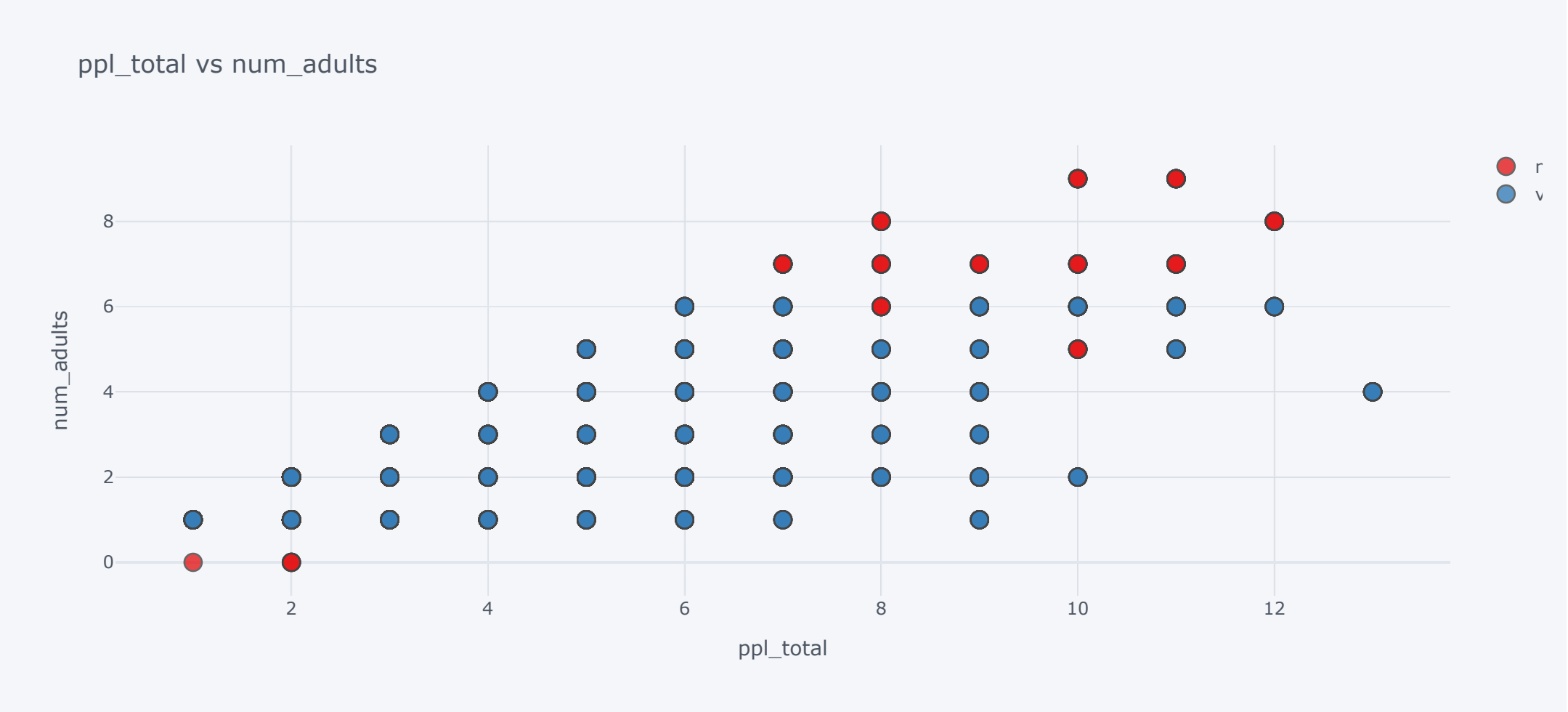
# Customize color: map colors

- When plotting data using scatter plots, 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
- `cufflinks` scatter plot 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')
```

# Color-customized scatter plot



# Module completion checklist

| Objective | Complete |
| --- | --- |
| Visualize multiple metrics using cufflinks | ✔ |
| Create interactive visualizations using plotly | |
| Generate interactive visualizations with transformed summary data | |
| Create inset plots | |
| Save your plotly and cufflinks 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

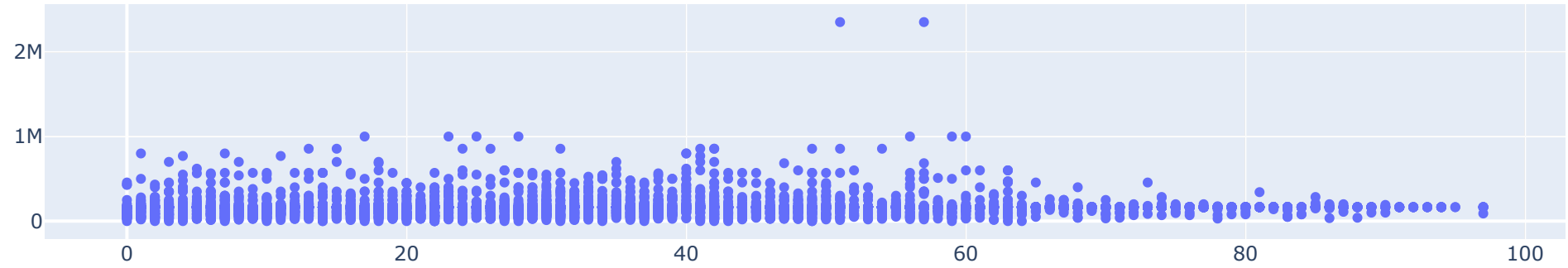
- Plotly charts have two main components: **data** and **layout**
- All traces / plots are saved in the **data** component, while all customizable attributes of the layout are in the **layout** component
  - Traces contain the data we wish to plot as well as its specifications, such as opacity, color, etc.
- We can view the plotly cheat sheet ***here***
- Let's plot a simple scatter plot using plotly's syntax

# Creating a trace

- `trace`: a Plotly-specific term for a dict that could consist of data or of some basic specs about appearance, which we add to variable `data`
  - `x`: x-axis
  - `y`: y-axis
  - `mode`: drawing mode for the trace - could be `lines`, `markers`, etc.
- `format`: a dict that controls figure-wide positioning and configuration of non-data-related parts of the figure
- `data`: a list where we add traces into
- `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]

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

# Simple scatter plot using plotly




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

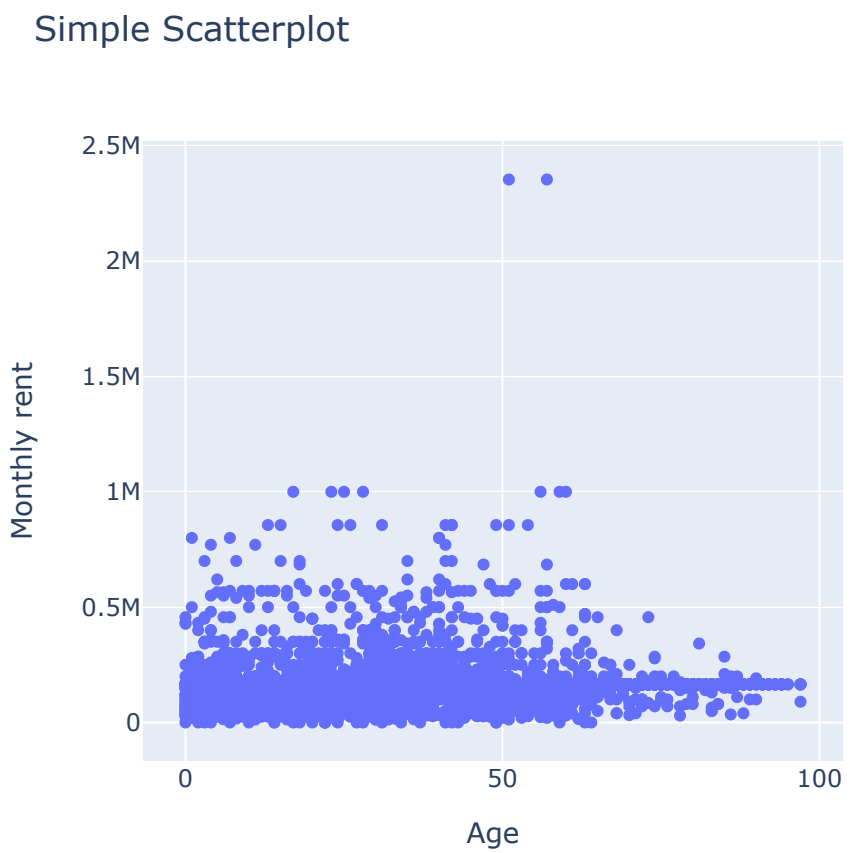
# Altering the layout

- `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]

#Alter the layout.
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')
```

# Final scatter plot

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

data = [trace]

#Alter the layout.
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

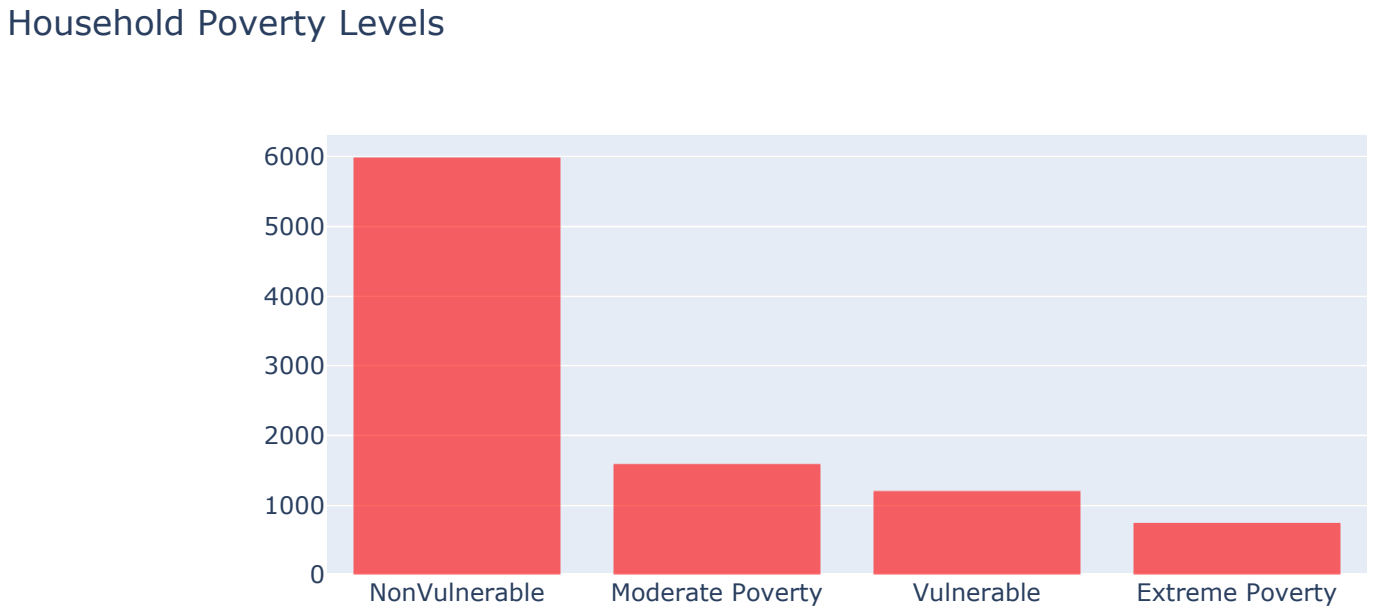
- We're going to create a bar chart with the original four levels of our target variable from the `household_poverty` dataset
- Here are the new attributes used for this plot:
- `trace`:
  - `marker`: dictionary which sets the style of the plot
  - sets the `color` and `opacity`
- `layout`:
  - `margin`: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

# Final bar chart

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



# Knowledge check 1

# Exercise 1

# Module completion checklist

| Objective | Complete |
| --- | --- |
| Visualize multiple metrics using cufflinks | ✔ |
| Create interactive visualizations using plotly | ✔ |
| Generate interactive visualizations with transformed summary data | |
| Create inset plots | |
| Save your plotly and cufflinks graphs | |

# Compound visualizations: layered plots

- Now let's prepare our data to create a layered bar chart
- We will start by grouping the data by the `Target` variable

```
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
```

# Transform into long data

- We now want to transform the wide data into long data for easier data manipulation

```
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
```

# Create data frames

```
# Let's get the `Target` = `True` mean data.
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
```

```
# Now 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
```

# Pickle summary data frames

We'll now pickle the `costa_true_means` and `costa_false_means` data frames so they can be used for later

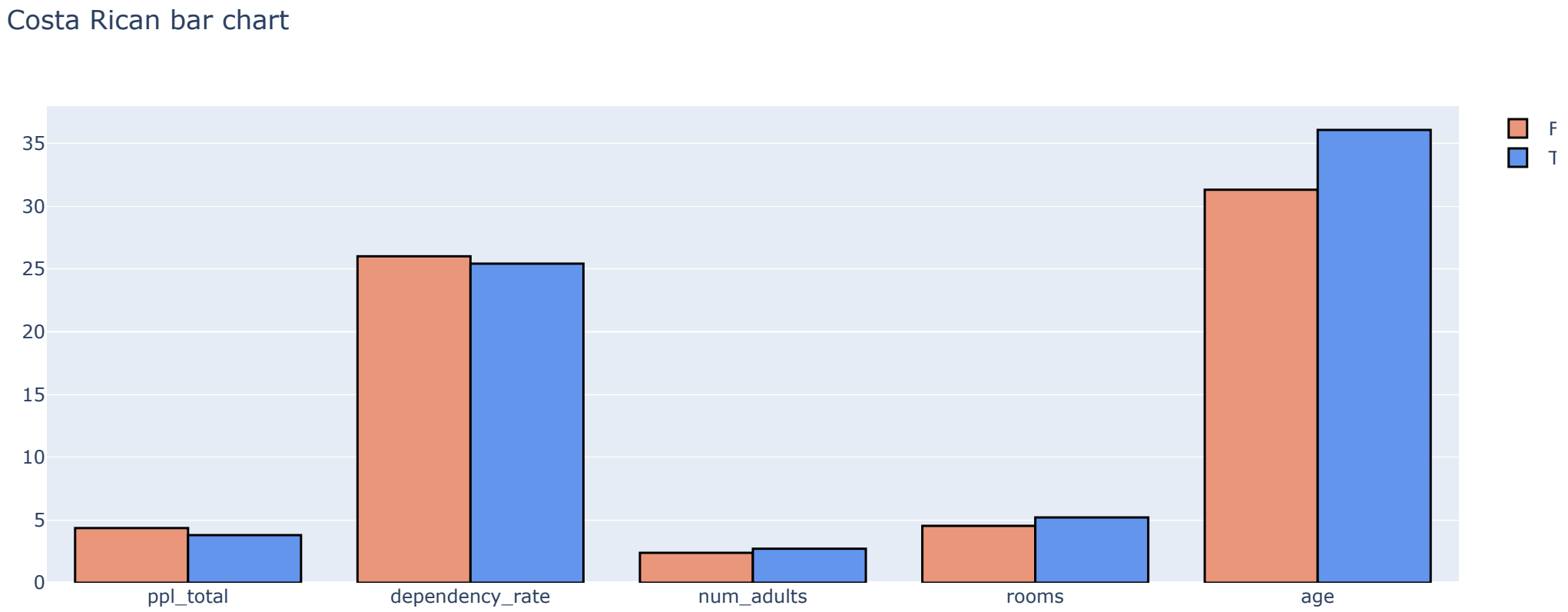
```
pickle.dump(costa_true_means, open("costa_true_means.sav","wb" ))
```

```
pickle.dump(costa_true_means, open("costa_false_means.sav","wb" ))
```

# Create traces from data frames

```
# 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)
```

# Layered bar chart



# Compound visualizations: subplots

- Let's use the plots created before to build a subplot

```
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, ctd.

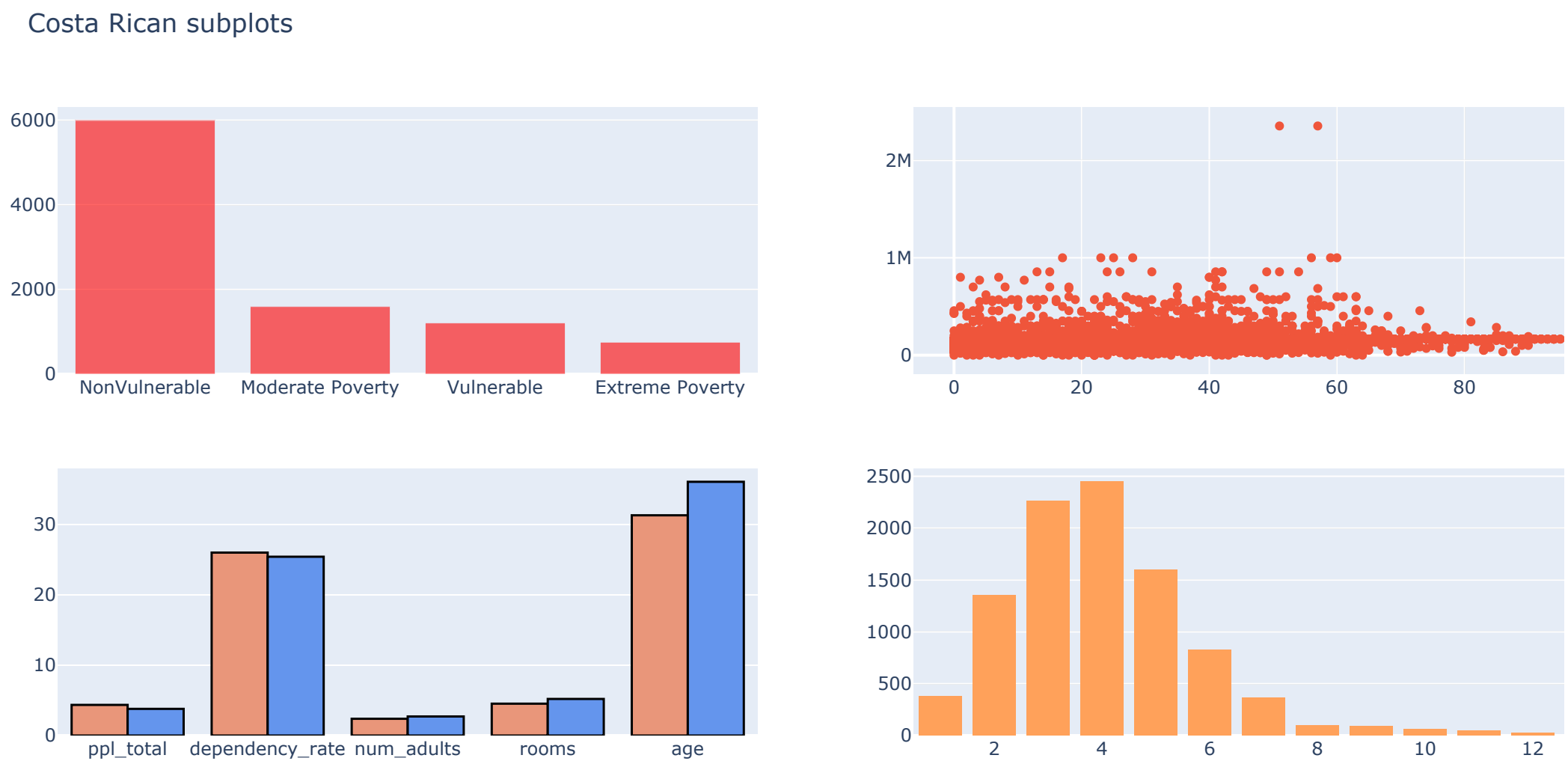
```
trace4 = go.Bar(x = costa_true_means['metric'],
                y = costa_true_means['mean'],
                name = "trace4",
                marker = dict(color = 'cornflowerblue',
                              line = dict(color = 'rgb(0,0,0)', width = 1.5
                                     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

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

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

# Producing a single figure

- Now let's create a single figure made up of multiple tiled subplots
- That way we can see several different data visualizations at a glance
- To do so, we'll start by creating traces of the different parts.

```
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')
```

# Subplots in plotly



# Compound visualizations: labeling axes

- To label the figure, we can use the same traces as when we created the subplots earlier

```
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

# Module completion checklist

| Objective | Complete |
| --- | --- |
| Visualize multiple metrics using cufflinks | ✔ |
| Create interactive visualizations using plotly | ✔ |
| Generate interactive visualizations with transformed summary data | ✔ |
| Create inset plots | |
| Save your plotly and cufflinks graphs | |

# 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

```
# Initialize offline plotting
init_notebook_mode(connected = True)
cf.go_offline()

# 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, ctd.

- `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]
```

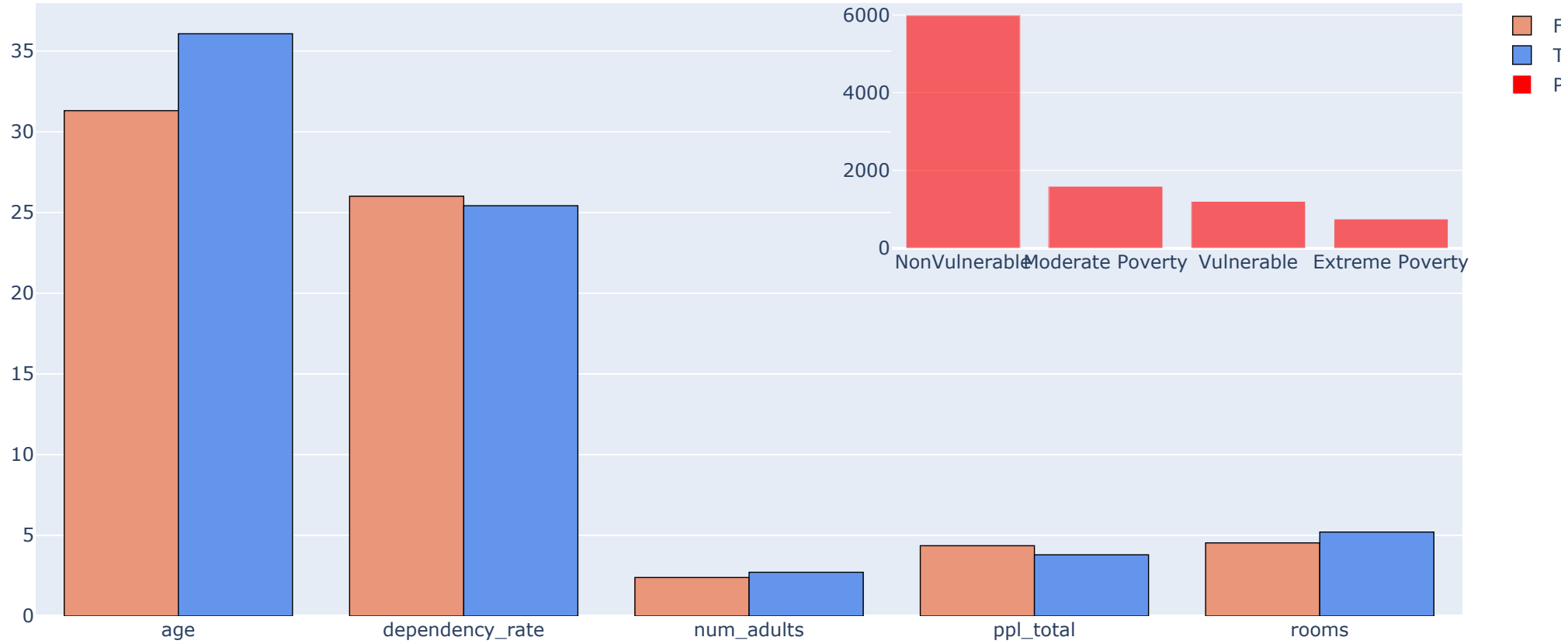
- Next, we will create the `layout`

# 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

# Module completion checklist

| Objective | Complete |
| --- | --- |
| Visualize multiple metrics using cufflinks | ✔ |
| Create interactive visualizations using plotly | ✔ |
| Generate interactive visualizations with transformed summary data | ✔ |
| Create inset plots | ✔ |
| Save your plotly and cufflinks 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 filename using `HTML` extension

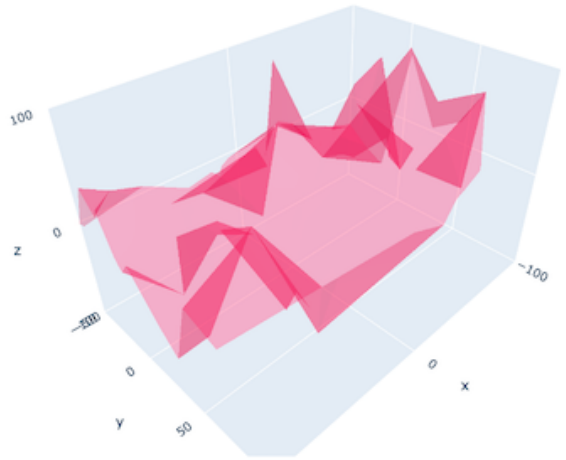
```
trace = go.Scatter(
    x = costa_viz['age'],
    y = costa_viz['monthly_rent'],
    mode = 'markers')

data = [trace]

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

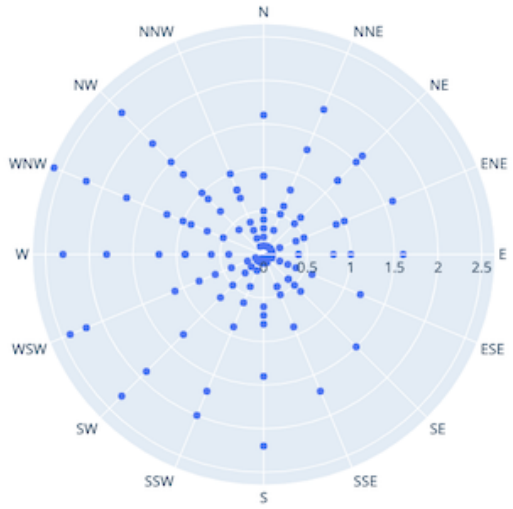
# Knowledge check 2

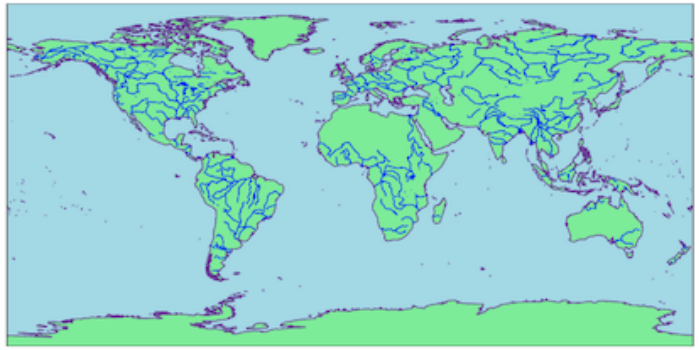
# Exercise 2

# Continuing with plotly

- So far, we've introduced how to create some most popular visualizations with Plotly and Cufflinks
- There are many more robust features that you can explore such as ***3-dimentional plots***, ***polar plots***, and ***geo plots for outline maps***
- Each of those plots has its own advantages and should be selected depending on the specific use case



# Module completion checklist

| Objective | Complete |
| --- | --- |
| Visualize multiple metrics using cufflinks | ✔ |
| Create interactive visualizations using plotly | ✔ |
| Generate interactive visualizations with transformed summary data | ✔ |
| Create inset plots | ✔ |
| Save your plotly and cufflinks graphs | ✔ |

# This completes our module

**Congratulations!**