

# Subplots

INTRODUCTION TO DATA VISUALIZATION WITH PLOTLY IN PYTHON

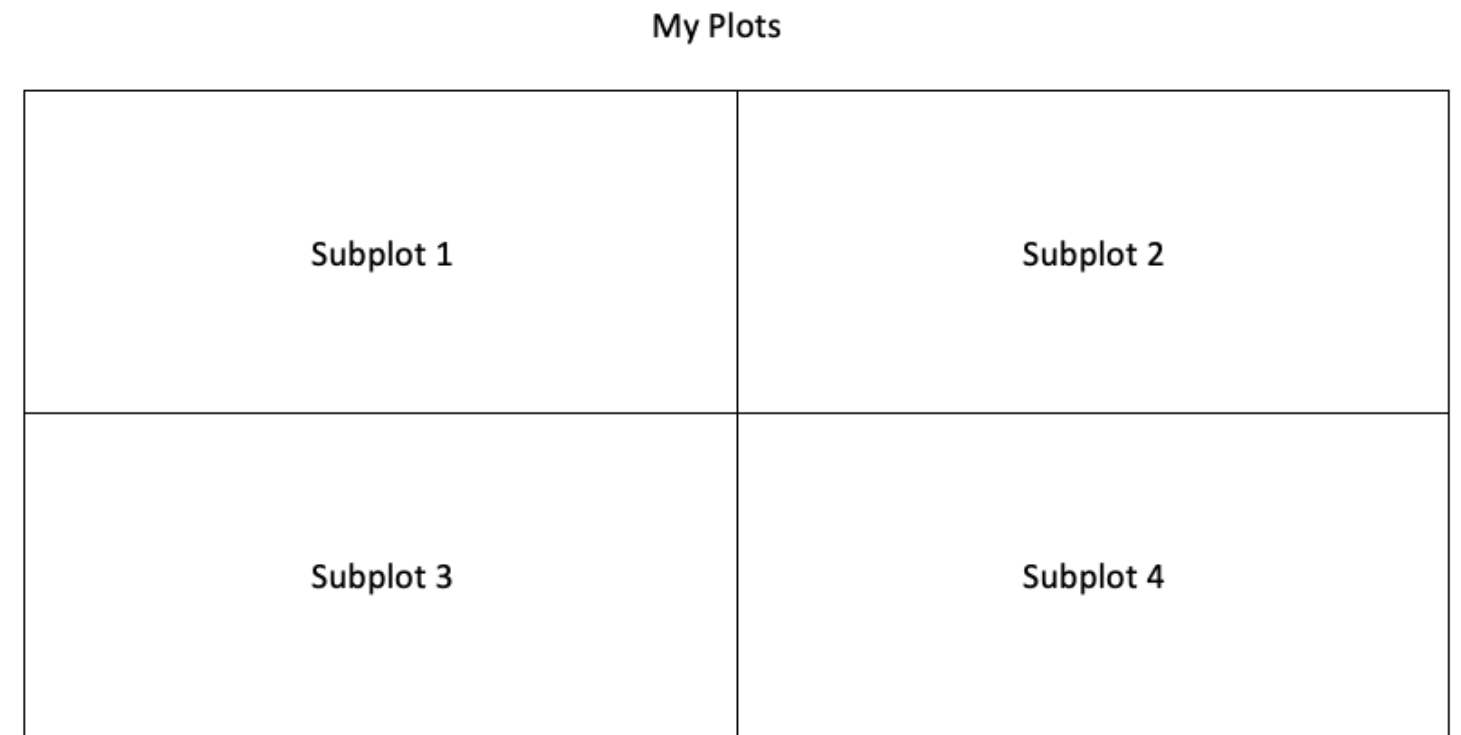


**Alex Scriven**  
Data Scientist

# What are subplots?

- Subplots: 'mini-plots' positioned in a grid arrangement
- Display different plot types (same data) or different data subsets
- Many are possible - but more will make each plot smaller!

For example:



# A reminder of traces

Remember discussing 'traces' earlier?

- Each set of `data` + graph `type` is a trace.
- You can build a plot by using `fig.add_trace(X)` where `X` is a `graph_objects` object (such as `go.Scatter()` or `go.Bar()`)
  - So far we haven't needed to do this.

To add data to each subplot, we will use `.add_trace()`.

# graph\_objects (go) vs plotly.express (px)

`graph_objects` and `plotly.express` often similar but have slight differences:

`add_trace()` takes `px` plots but the code is complex and not best-practice so we will use `go`

Check equivalent documentation for more help (`px histogram` vs `go histogram`)

```
# With graph_objects
go.Histogram(x=revenues['Revenue'],
             nbinsx=5, name='Histogram')

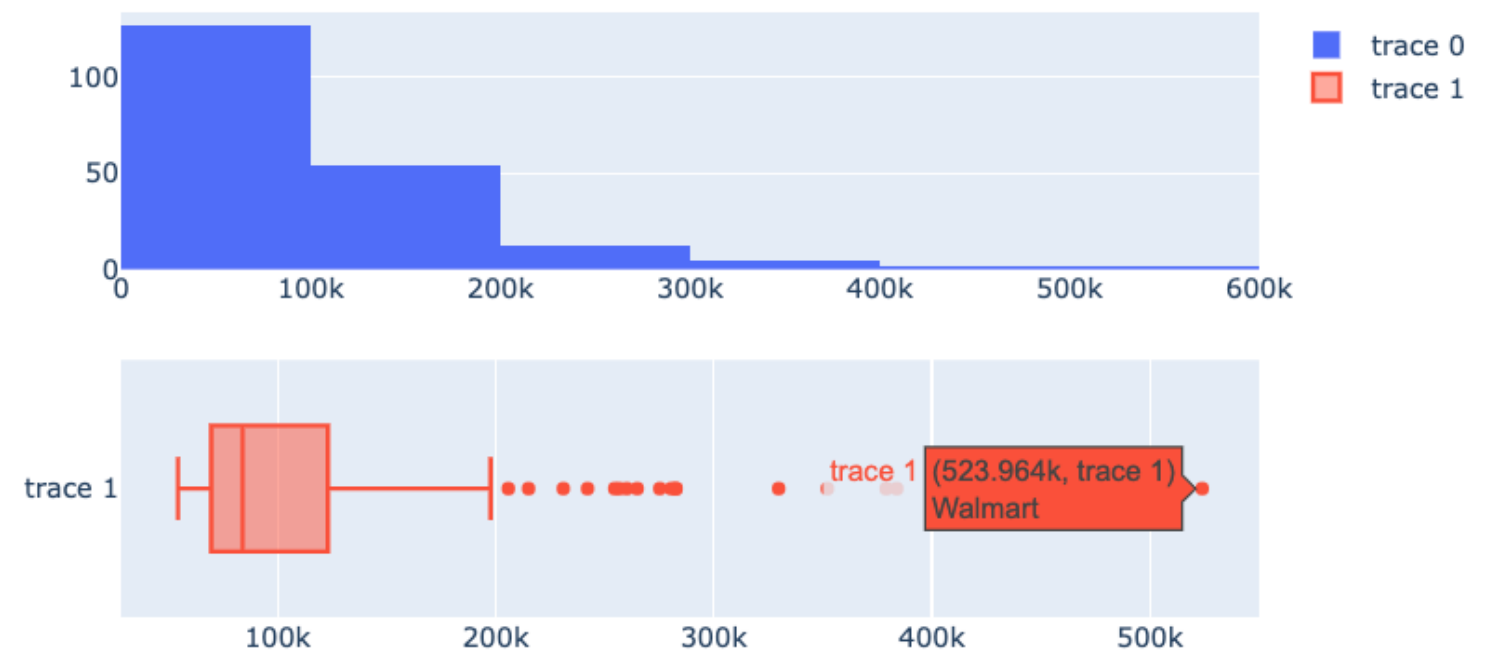
# With plotly.express
px.histogram(data_frame=revenues,
            x='Revenue', nbins=5,
            title='Histogram')
```

# Creating a 1x2 subplot

Let's build a 1x2 subplot (histogram + box plot)  
from the `revenues` DataFrame:

```
from plotly.subplots import make_subplots
fig = make_subplots(rows=2, cols=1)
fig.add_trace(
    go.Histogram(x=revenues['Revenue'], nbinsx=5,
    row=1, col=1)
fig.add_trace(
    go.Box(x=revenues['Revenue'],
    hovertext=revenues['Company']),
    row=2, col=1)
fig.show()
```

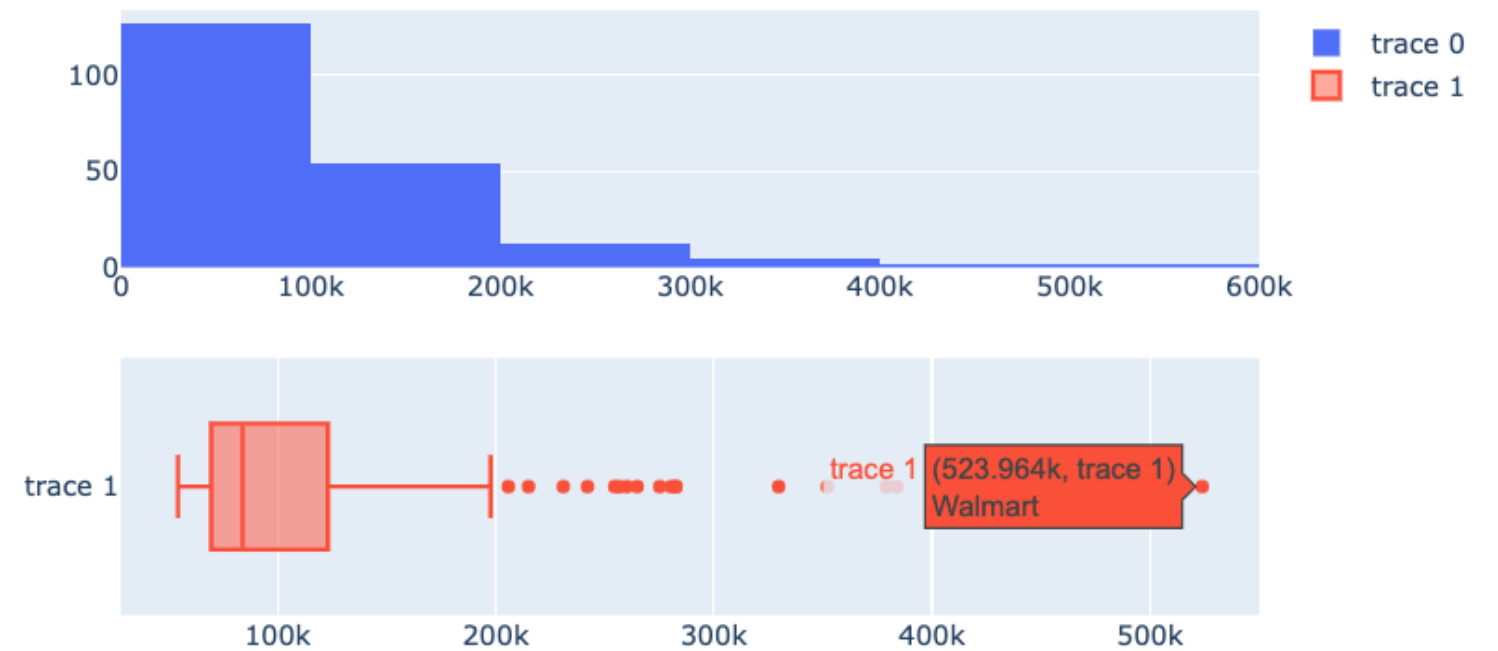
Our plots:



# Customizing subplots

Some stylistic elements that need attention:

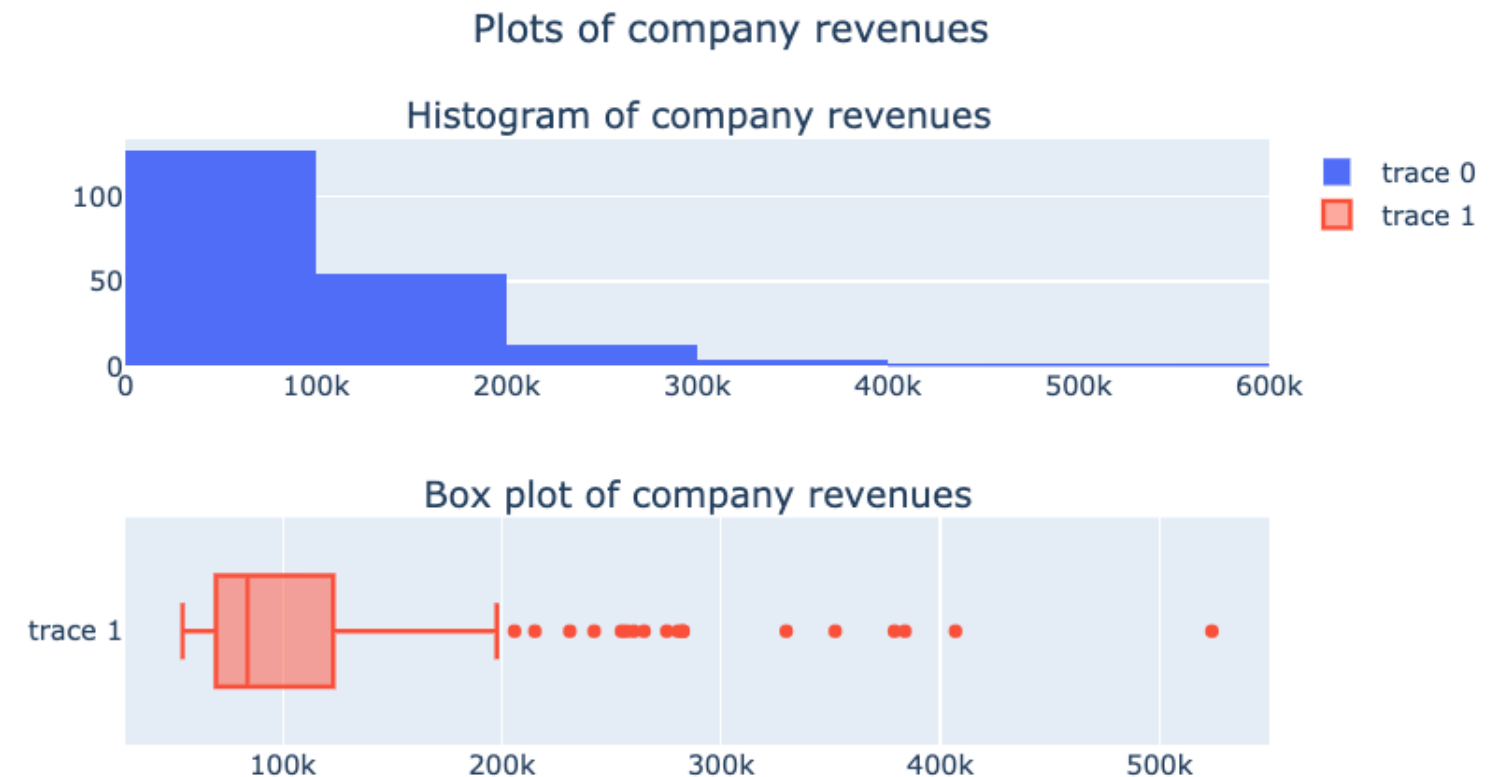
1. No overall plot title
2. No subplot titles
3. The legend says 'trace 1' / 'trace 2'
4. Other customization skills!



# Subplot titles

Let's fix the titles:

```
from plotly.subplots import make_subplots
fig = make_subplots(rows=2, cols=1,
                    subplot_titles=[
                        'Histogram of company revenues',
                        'Box plot of company revenues'])
## Add in traces (fig.add_trace())
fig.update_layout({'title': {'text':
                            'Plots of company revenues',
                            'x': 0.5, 'y': 0.9}}})
fig.show()
```

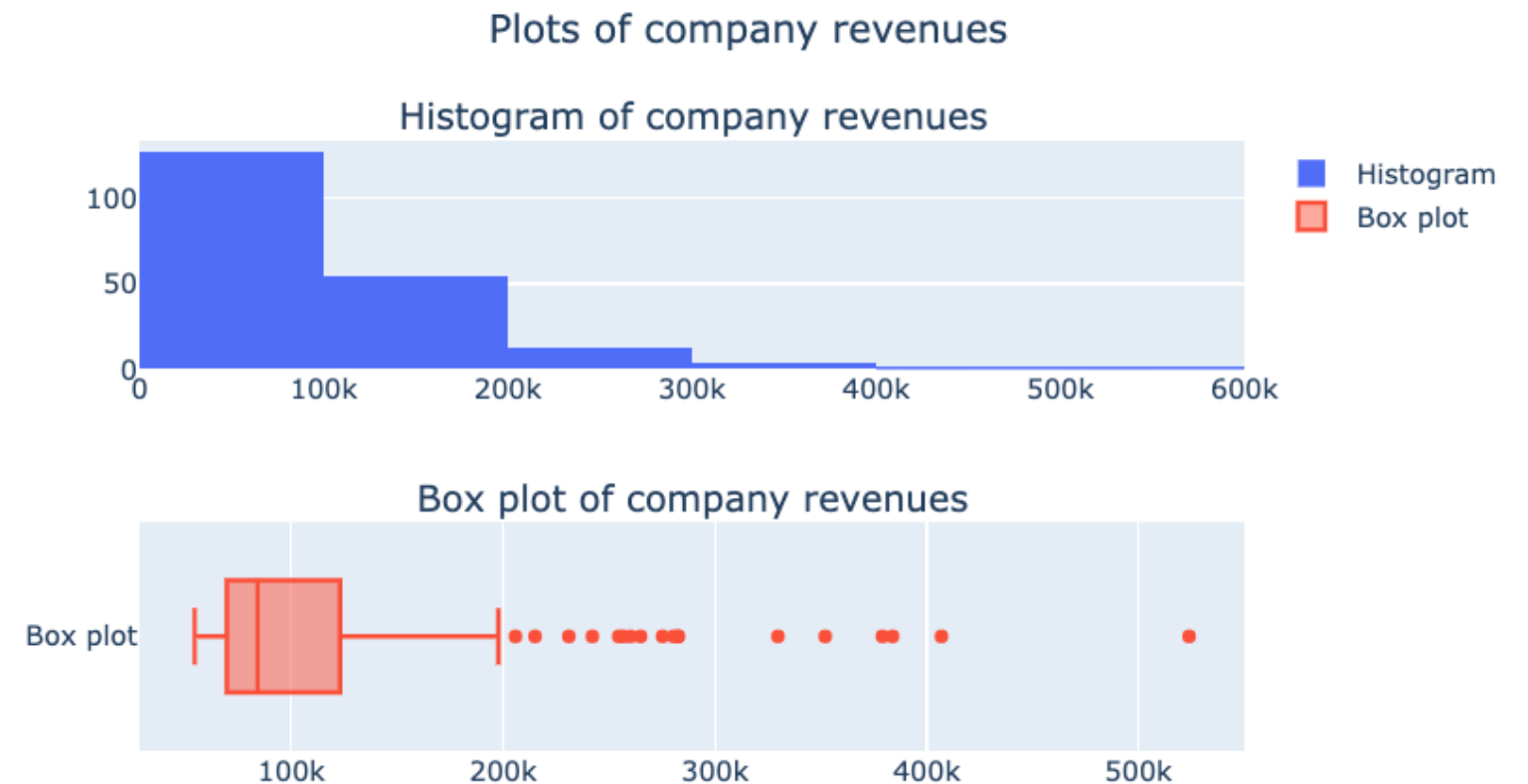


Note: More options available in the [\(documentation\)](#)

# Subplot legends

Let's fix the legend names:

```
fig.add_trace(  
    go.Histogram(x=revenues.Revenue,  
        nbinsx=5, name='Histogram'),  
    row=1, col=1)  
fig.add_trace(  
    go.Box(x=revenues.Revenue,  
        hovertext=revenues['Company'],  
        name='Box plot'),  
    row=2, col=1)
```



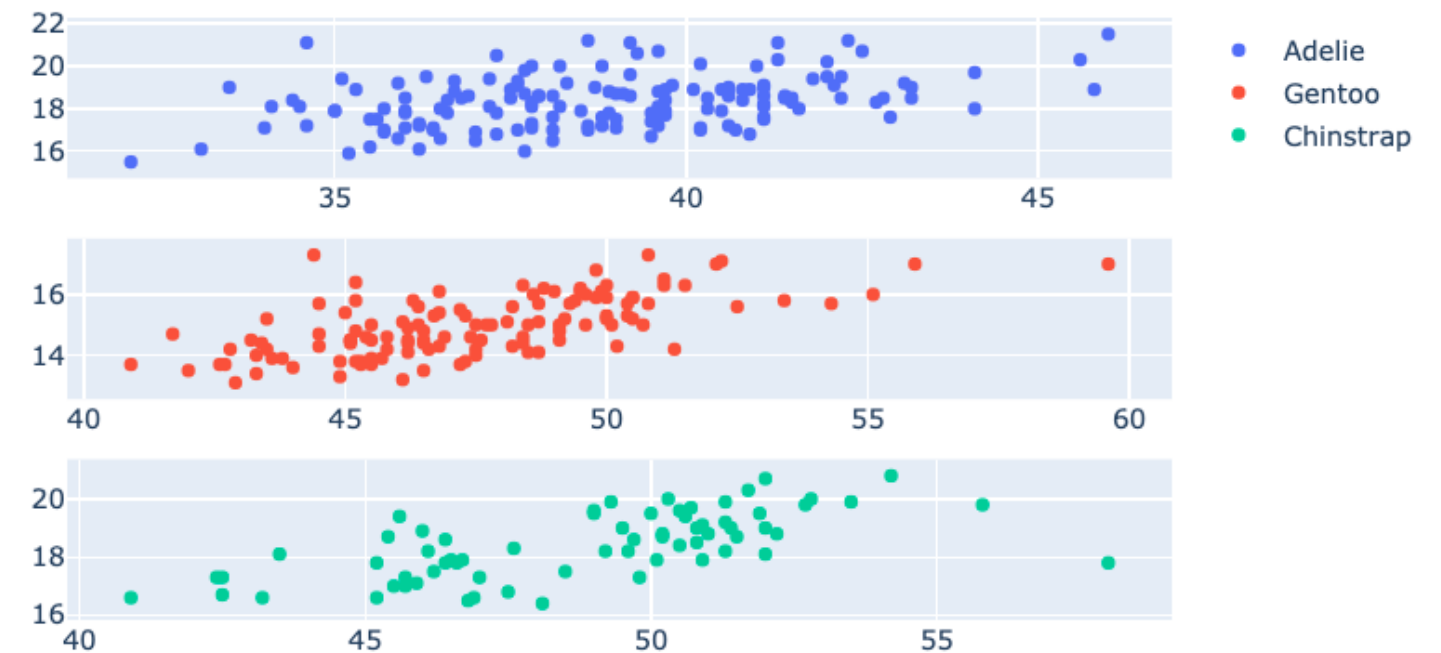


# Stacked subplots

Let's redo our penguins scatterplot with subplots, splitting out the species:

```
fig = make_subplots(rows=3, cols=1)
row_num = 1
for species in ['Adelie', 'Gentoo', 'Chinstrap']:
    df = penguins[penguins['Species'] == species]
    fig.add_trace(
        go.Scatter(x=df['Culmen Length (mm)'],
                    y=df['Culmen Depth (mm)'],
                    name=species, mode='markers'),
        row=row_num, col=1)
    row_num += 1
fig.show()
```

Different x-axes?

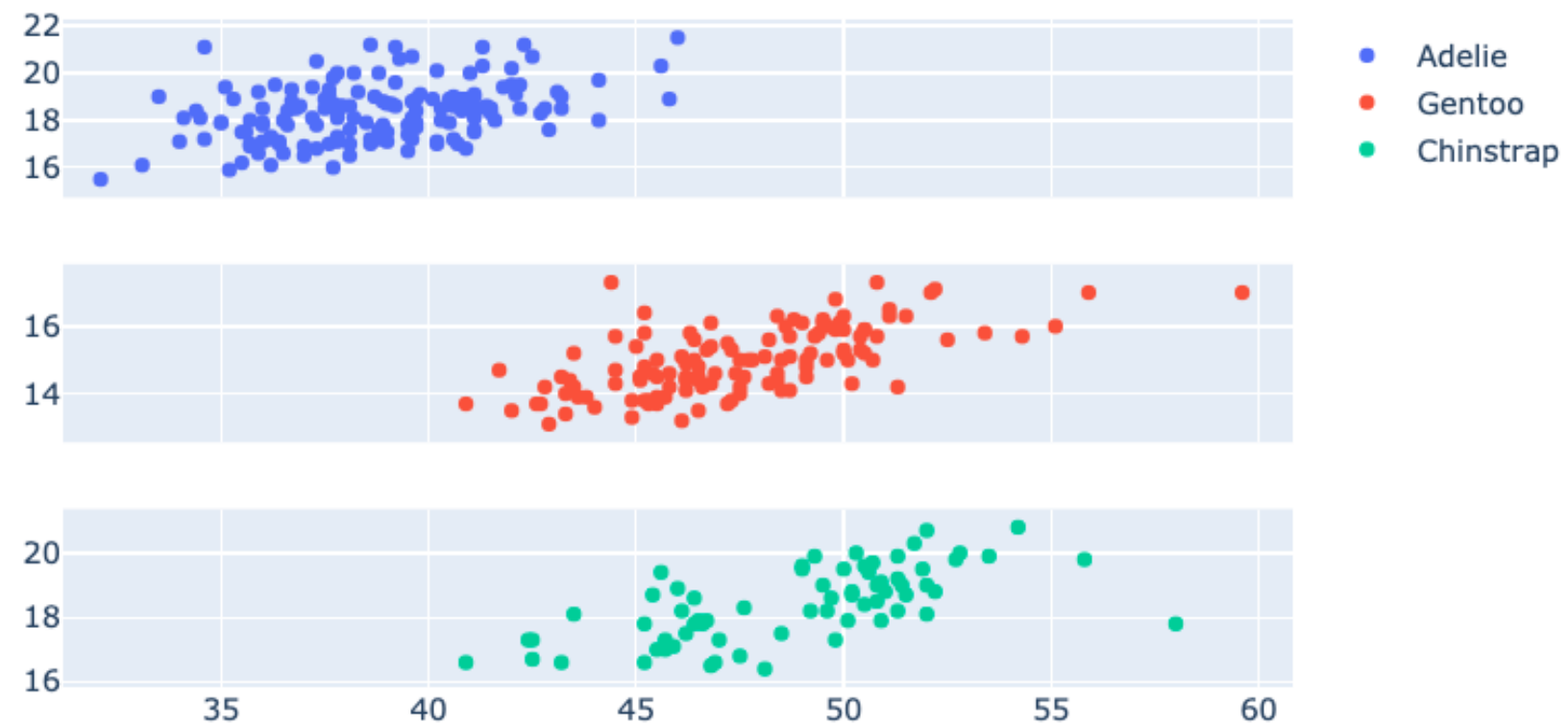


# Subplots with shared axes

Let's fix this by making the x-axis 'shared':

```
fig = make_subplots(rows=3, cols=1, shared_xaxes=True)
```

That's better!



# Let's practice!

INTRODUCTION TO DATA VISUALIZATION WITH PLOTLY IN PYTHON

# Layering multiple plots

INTRODUCTION TO DATA VISUALIZATION WITH PLOTLY IN PYTHON



**Alex Scriven**  
Data Scientist

# What is plot layering?

Layering plots = multiple plots on top of each other

- No separate grid location (or separate plot)
- We use `add_trace()`
- Some 'shortcut' functions exist:
  - `add_bar()` , `add_area()` , `add_box()` , etc.
  - Search for 'add\_' on the figure [documentation](#) for more

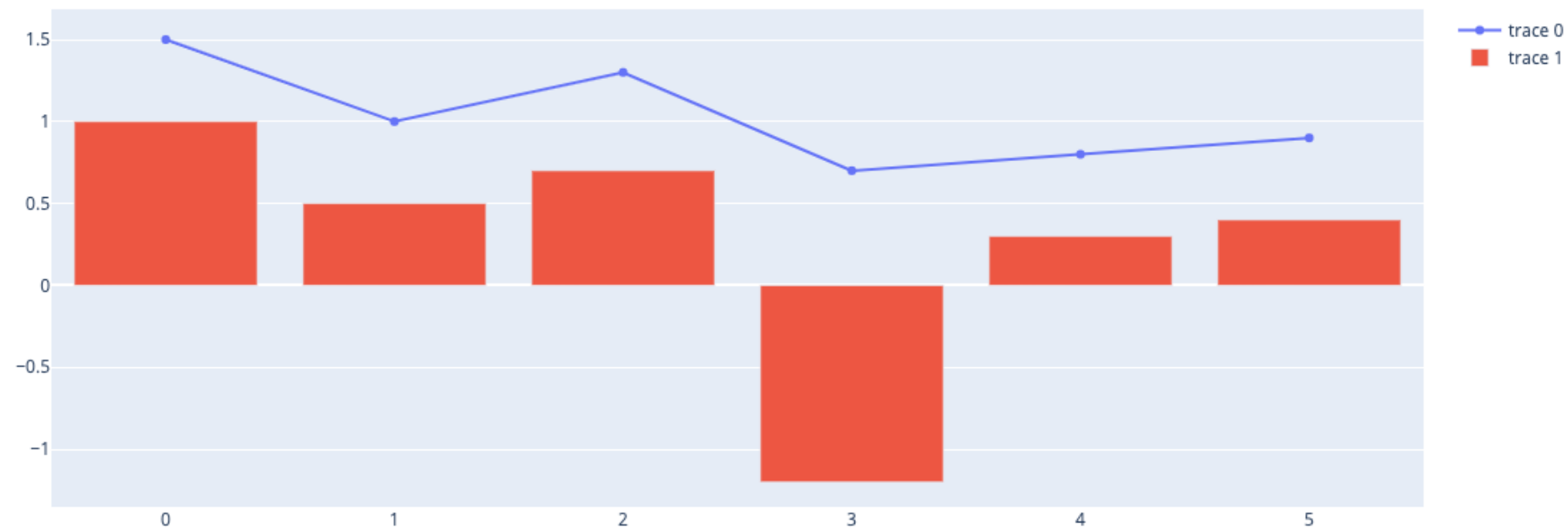
# Why layer plots?

Layering plots is useful for:

- Accessing more customization (same type)
  - For example, layering multiple line charts
- Displaying complementary plot types
- Using different plot types to draw focus
- Keeping visualizations tight for close comparisons
  - Compared to split out subplots or separate plots

# Bar + line layered plot

- A bar chart with a line-chart layered over the top is common
- Allows analyzing trends in multiple variables over time

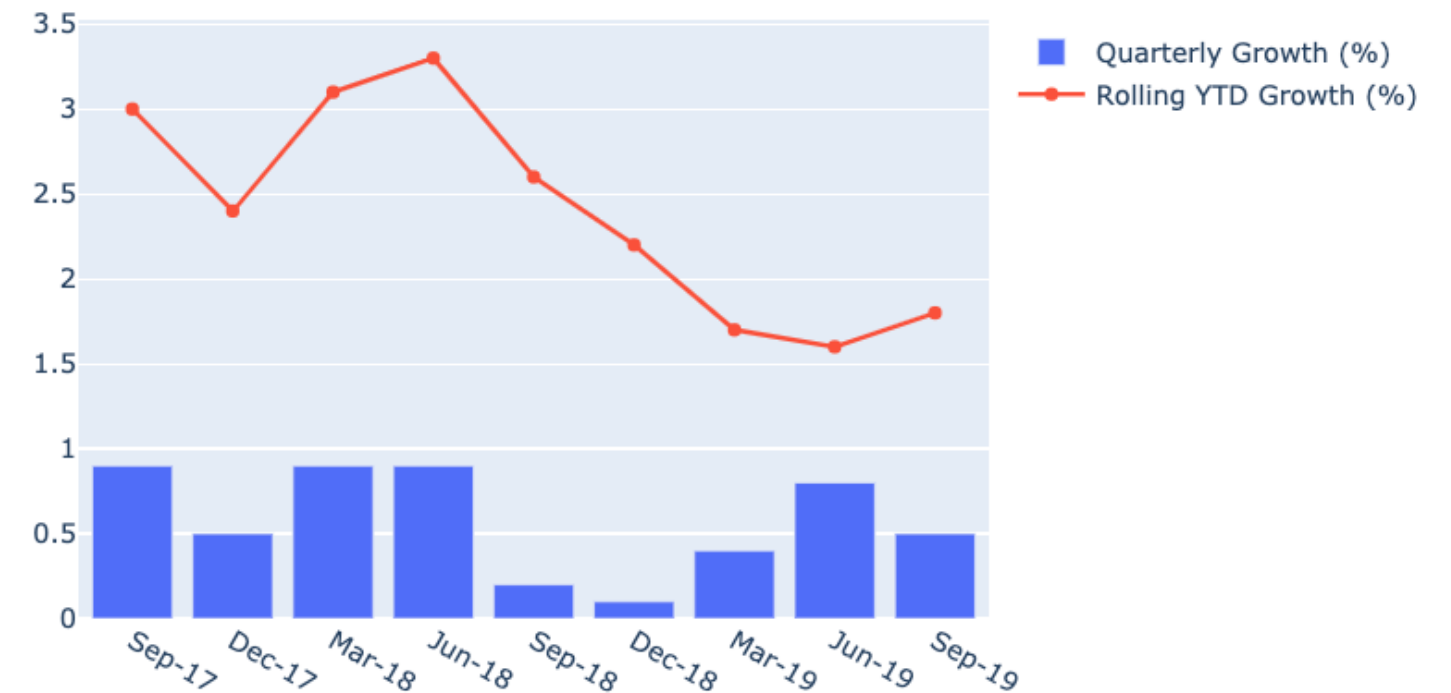


# GDP growth layered plot

Consider the Australian GDP growth per quarter (and yearly rolling growth)

```
fig = go.Figure()
fig.add_trace(go.Bar(x=gdp['Date'],
                    y=gdp['Quarterly growth (%)'],
                    name='Quarterly Growth (%)'))
fig.add_trace(go.Scatter(x=gdp['Date'],
                        y=gdp['Rolling YTD growth (%)'],
                        name='Rolling YTD Growth (%)',
                        mode='lines+markers'))
fig.show()
```

Here is our plot:





# Nonsensical combinations

Layering many types of traces is possible, but stick to those that make sense:

- Line + another plot to show trend, such as
  - Line + bar plots
  - Line + scatterplots
- The same type (line + line, bar + bar)
- Make sure the x and y axes have the same units!

# Let's practice!

INTRODUCTION TO DATA VISUALIZATION WITH PLOTLY IN PYTHON

# Time buttons

INTRODUCTION TO DATA VISUALIZATION WITH PLOTLY IN PYTHON



**Alex Scriven**  
Data Scientist

# What are time buttons?

Time buttons allow filter/zoom in line charts.

Often seen on most stock websites such as Yahoo Finance (TESLA stock);

- 1D = Show data for the last day, 1M = for the last month, 1Y = for the last year, etc.
- YTD = Show data for the 'year to date'



# Time buttons in Plotly

Time buttons in Plotly are a dictionary specifying:

- `label` = Text to appear on the button
- `count` = How many `step` s to take when clicking the button
- `step` = What date period to move ( `'month'` , `'year'` , `'day'` , etc.)
- `stepmode` = Either `'todate'` or `'backward'`
  - `'todate'` = From the beginning of the nearest whole time period denoted in `step` (after going backwards by `count` )
  - `'backward'` = Just go backwards by `count`

# 'todate' vs. 'backward'

To illustrate `todate` vs. `backward`, consider a dataset finishing on October 20th and a 6-month button ( `count=6`, `step='month'` ) with each option.

- `stepmode='backward'` would zoom the plot to start on **April 20th** (6 months backward)
- `stepmode='todate'` would zoom the plot to start on **May 1st** (start of the nearest month to April 20th)

# Sydney rainfall example

Let's chart the rainfall from a weather station in Sydney in 2020.

Create the buttons

- Buttons are specified as a list of dictionaries

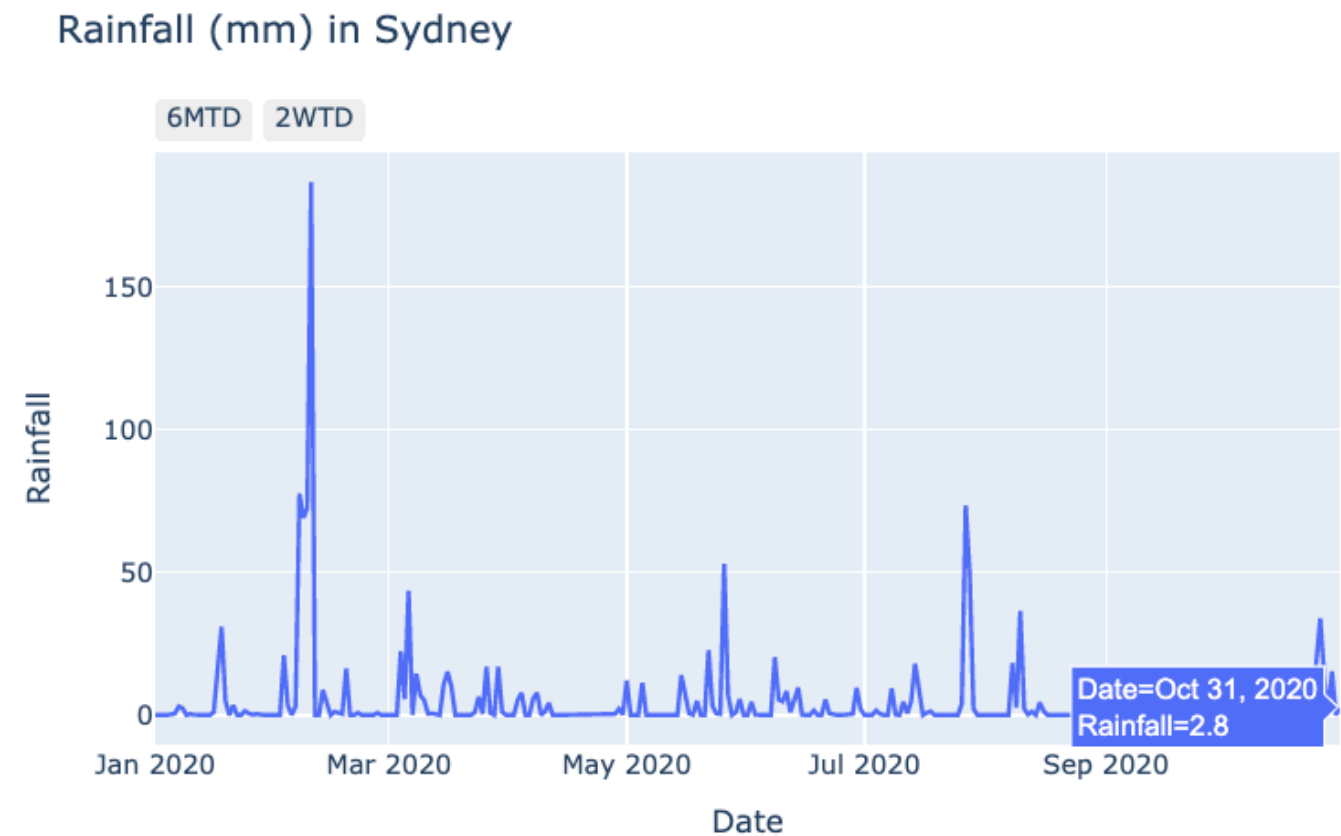
```
date_buttons = [  
    {'count': 6, 'step': "month", 'stepmode': "todate", 'label': "6MTD"},  
    {'count': 14, 'step': "day", 'stepmode': "todate", 'label': "2WTD"}  
]
```

# Adding the time buttons

Now let's create the chart and add them;

```
fig = px.line(data_frame=rain, x='Date',
              y='Rainfall',
              title="Rainfall (mm) in Sydney")
fig.update_layout(
    {'xaxis':
     {'rangelector':
      {'buttons': date_buttons}
     }})
fig.show()
```

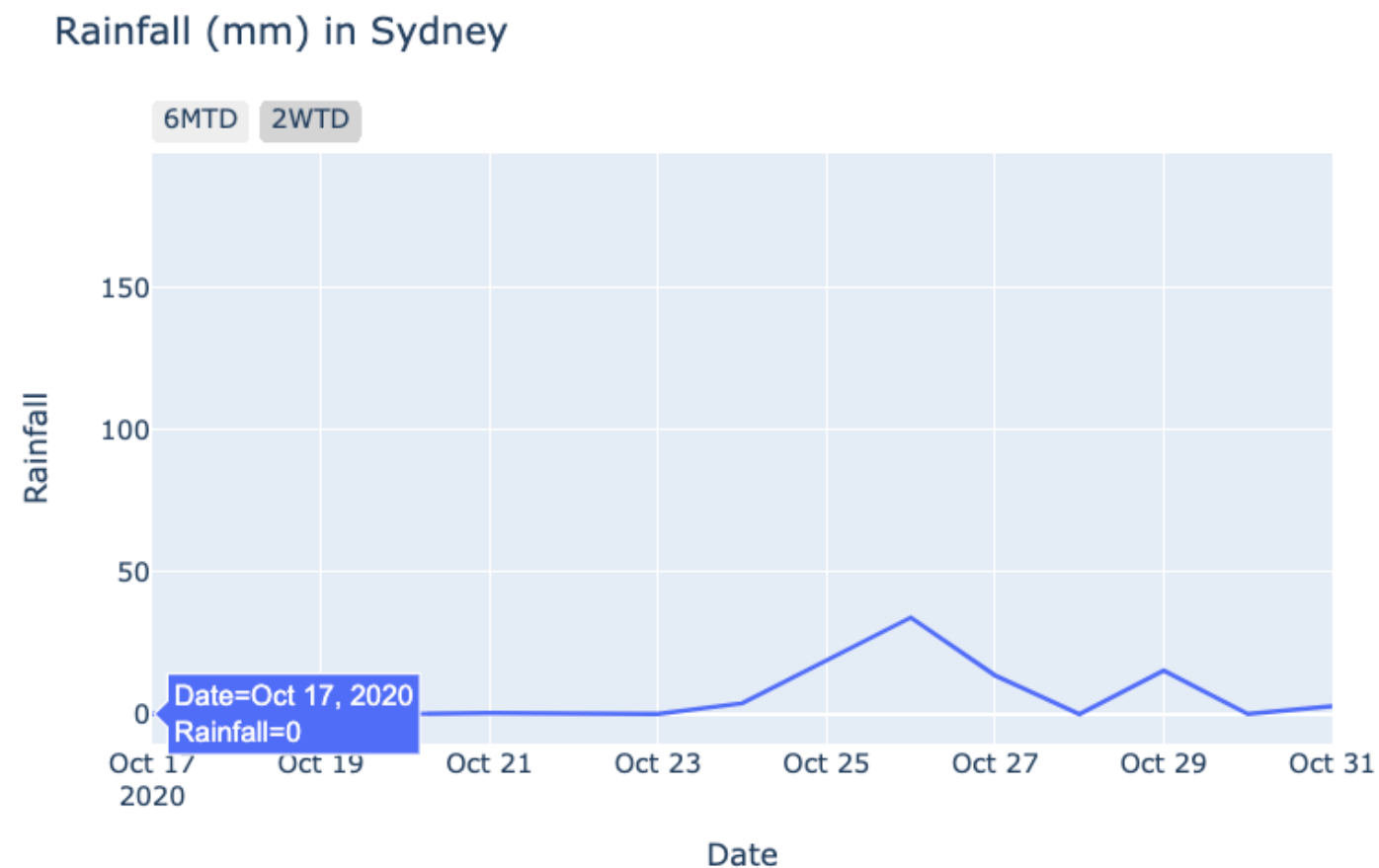
Our line chart has the buttons:



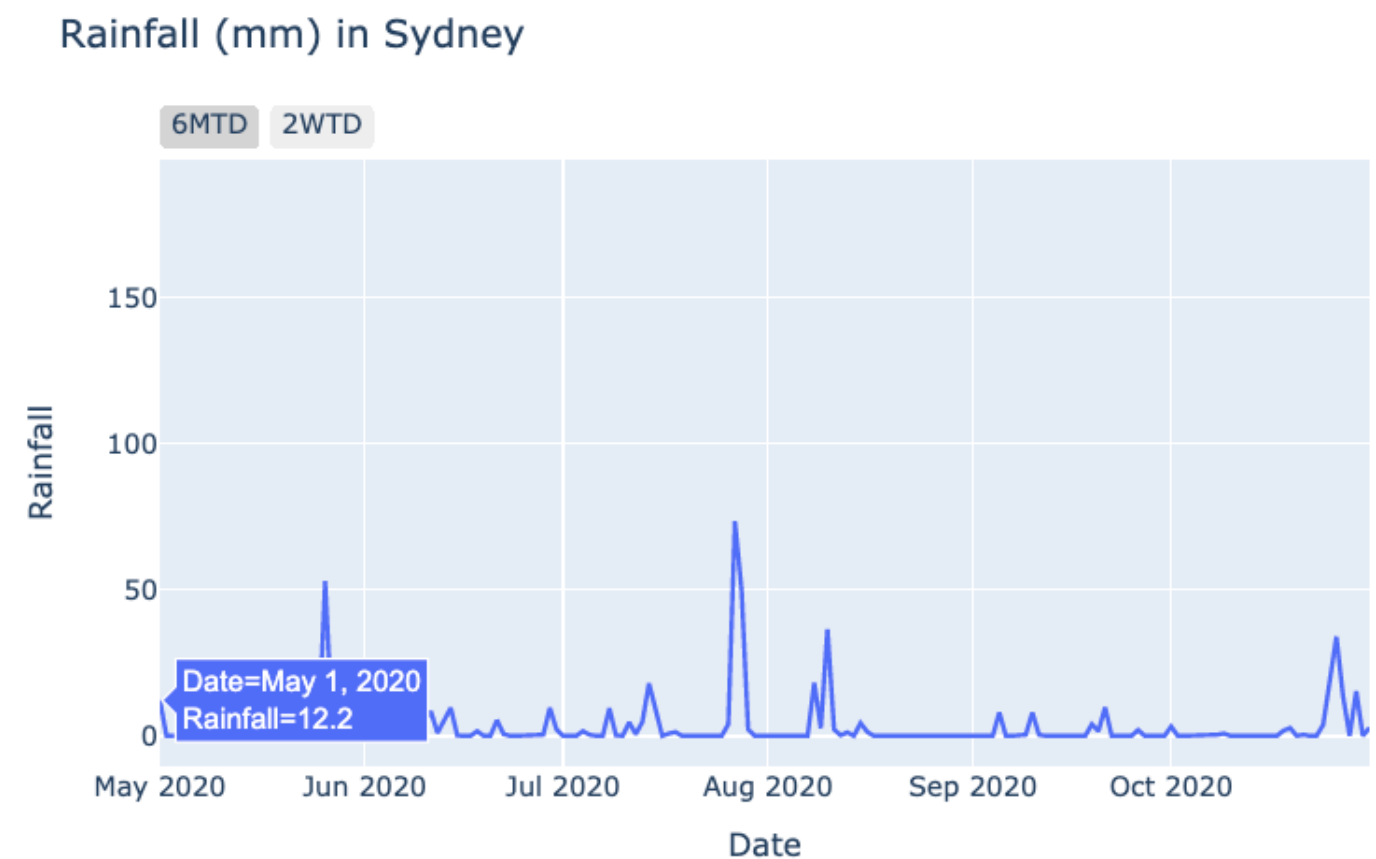


# Clicking our time buttons

Clicking the **2WTD** button:



Clicking the **6MTD** button:



# Let's practice!

INTRODUCTION TO DATA VISUALIZATION WITH PLOTLY IN PYTHON