datacamp-plotly

Table of Contents

1. Source

2. Plotly and the Plotly Figure

- No need to know JavaScript: Use a Python wrapper
- Fast and easy to implement simple plots
- Methods:
  1. `plotly.express` for quick plots
  2. `plotly.graph_objects` for more customization (link)
  3. `plotly.figure_factory` for specific figures (link)
- Documentation: plotly.com/python - very pythonic (example: graphical object `scatter` in `plotly.graph_objects.scatter` here)
- The Plotly Figure:
  1. `layout` is a Python dictionary controlling the style of the figure.
  2. `data` is a list of dictionaries for (40) type + data = `trace`.
  3. `frames` for animated plots.
- Only one `layout` per figure but more than one `trace` per graph.

- Example for a Plotly `figure` object:

```
import plotly.graph_objects as go
figure_config = dict({
    "data": [{"type": "bar",
              "x": ["Mon","Tue","Wed","Thu","Fri","Sat","Sun"],
              "y": [28, 27, 25, 31, 32, 35, 36]}],
    "layout": {"title": {"text":"Temperatures of the week",
                         "x":0.5,"font":{'color':'red','size':15}
                         }}})
```

```
fig = go.Figure(figure_config)
fig.show()
```

- This object cannot be displayed in Emacs because it's interactive: when you hover over the bars of the plot, you see the values.
- But `figure_config` can be printed:

```
print(figure_config)
```

```
{'data': [{'x': ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun'], 'y': [28, 27, 2
```

- Fixing a plotly figure:

```
## define figure
monthly_sales = dict({
    "data": [{'type': '',
              'x': ['Jan', 'Feb', 'March'],
              'y': [450, 475, 400]}],
    "layout": {'title': {'text': ''}}})
## Update the type
monthly_sales['data'][0]['type'] = 'bar'

## Update the title text
monthly_sales['layout']['title']['text'] = 'Sales for Jan-Mar 2020'

## Create a figure
fig = go.Figure(monthly_sales)

## Print it out!
fig.show()
```

3. Plotly Figure as a "Dictionary of Dictionaries"

In Plotly, a figure is often described as a "dictionary of dictionaries" because it is structured in a nested format, with keys and values that contain further dictionaries or lists within them. This structure allows for a high level of customization for both data and layout properties of the plot.

3.1. Top-Level Structure (`figure`)

The figure itself is essentially a dictionary that has two main keys: `data` and `layout`.

```
figure = {
    'data': [...],     # List of dictionaries, each describing a trace (plot layer)
    'layout': {...}    # Dictionary describing the layout (appearance) of the plot
}
```

3.2. `data` Key

The value associated with `data` is a **list of dictionaries**. Each dictionary in this list represents a single "trace" or data series in the plot (e.g., a line, bar, scatter plot). Each trace dictionary can contain keys like `type`, `x`, `y`, etc., specifying the type of plot (e.g., bar, line), the x-axis data, the y-axis data, and other properties.

Example:

```
'data': [
    {
        'type': 'bar',  # Type of trace
        'x': ['Jan', 'Feb', 'Mar'],  # X-axis data
        'y': [450, 475, 400]         # Y-axis data
    }
]
```

3.3. `layout` Key

The value associated with `layout` is a **dictionary** that contains settings for the overall appearance of the figure, such as title, axis labels, color schemes, margins, and templates. Inside this dictionary, there may be further nested dictionaries. For example, `layout['title']` can itself be a dictionary with properties like `text` (the title text) and font settings.

Example:

```
'layout': {
    'title': {
        'text': 'Sales for Jan-Mar 2020',
        'font': {'color': 'red', 'size': 15}
    },
    'xaxis': {'title': 'Months'},
    'yaxis': {'title': 'Sales'}
}
```

3.4. Why It's Structured This Way

This nested dictionary structure allows Plotly figures to be highly flexible and customizable:

- **Multiple Traces**: You can add multiple traces to the `data` list, allowing for complex plots with multiple data series.
- **Fine-Grained Control**: Each trace and layout property can have detailed settings, allowing you to control every aspect of the plot.

3.5. Example of a Full Figure

Here's what a complete figure dictionary might look like:

```
figure = {
    'data': [
        {
            'type': 'bar',
            'x': ['Jan', 'Feb', 'Mar'],
            'y': [450, 475, 400]
        }
    ],
    'layout': {
        'title': {'text': 'Sales for Jan-Mar 2020'},
        'xaxis': {'title': 'Months'},
        'yaxis': {'title': 'Sales'}
    }
}
```

In this structure:

- **Top Level**: The `figure` dictionary contains `data` and `layout`.
- **Nested Dictionaries**: The `data` key holds a list with one dictionary (a bar chart trace). The `layout` key contains a dictionary with the title and axis labels.

In short, a Plotly figure is a "dictionary of dictionaries" because it organizes plot configuration and data in a hierarchical, nested dictionary format that allows easy access and modification of plot components.

4. Univariate visualizations (one variable only)

- `plotly.express` specifies a `DataFrame` and its columns as arguments, and is less customizable.
- `graph_objects` have `Bar`, `Scatter` constructors, etc. methods with more customization but more code needed.
- Common univariate plots: Bar chart, histogram, box, density plots.
- Bar charts: X-axis with one bar per group. Y-axis height represents the value of a variable.

- Example: Bar graph

```
import plotly.express as px
import pandas as pd
weekly_temps = pd.DataFrame({
    'day': ['Monday', 'Tuesday', 'Wednesday', 'Thursday',
            'Friday', 'Saturday', 'Sunday'],
    'temp': [28, 27, 25, 31, 32, 35, 36]})
fig = px.bar(data_frame=weekly_temps,x='day',y='temp')
fig.show()
```

- Dataset: Palmer penguins

```
pip install palmerpenguins
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: palmerpenguins in /home/aletheia/.local/lib/python3
Requirement already satisfied: pandas in /home/aletheia/.local/lib/python3.10/site
Requirement already satisfied: numpy in /home/aletheia/.local/lib/python3.10/site-
Requirement already satisfied: pytz>=2020.1 in /usr/lib/python3/dist-packages (fro
Requirement already satisfied: python-dateutil>=2.8.2 in /home/aletheia/.local/lib
Requirement already satisfied: tzdata>=2022.7 in /home/aletheia/.local/lib/python3
Requirement already satisfied: six>=1.5 in /usr/lib/python3/dist-packages (from py
```

- Check the data

```
from palmerpenguins import load_penguins
penguins = load_penguins()
penguins.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 344 entries, 0 to 343
Data columns (total 8 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
```

```
 0   species            344 non-null    object
 1   island             344 non-null    object
 2   bill_length_mm     342 non-null    float64
 3   bill_depth_mm      342 non-null    float64
 4   flipper_length_mm  342 non-null    float64
 5   body_mass_g        342 non-null    float64
 6   sex                333 non-null    object
 7   year               344 non-null    int64
dtypes: float64(4), int64(1), object(3)
memory usage: 21.6+ KB
```

- Example: Histogram

```
import plotly.express as px
fig = px.histogram(
    data_frame=penguins,
    x='body_mass_g',
    nbins=10)
fig.show()
```

  - Histogram arguments: `orientation` (vertically or horizontally), `histfunc` to aggregate bins (e.g. average, min, max).
  - Example: Box (and whisker) plot of penguins' flipper length, with

```
fig = px.box(
    data_frame=penguins,
    x='flipper_length_mm')
fig.show()
```

- Other arguments: `hover_data`, a list of column names to display on hover, `points` to specify how to show outliers.

5. Student scores bar graph

- Problem:

  > The school board has asked you to come and look at some test scores. They want an easy way to visualize the score of different students within a small class. This seems like a simple use case to practice your bar chart skills! You will help the school board team by creating a bar chart of school test score values.

- Create `DataFrame`:

```
import pandas as pd
import plotly.express as px

student_scores = pd.DataFrame({
    'name': ['John','Julia','Xuan','Harry'],
    'score': [80,97,90,85]})
print(student_scores)
```

```
    name  score
0   John     80
```

```
1  Julia     97
2   Xuan     90
3  Harry     85
```

- Create the bar plot:

```
fig = px.bar(
    data_frame=student_scores,
    x='name',
    y='score',
    title='Student Scores by Student')
fig.show()
```

6. Box plot of company revenues

- Problem:

> You have been contracted by a New York Stock exchange firm who are interested in upping their data visualization capabilities. They are cautious about this new technology so have tasked you with something simple first. To display the distribution of the revenues of top companies in the USA. They are particularly interested in what kind of revenue puts you in the 'top bracket' of companies. They also want to know if there are any outliers and how they can explore this in the plot. This sounds like a perfect opportunity for a box plot. You will help the investment team by creating a box plot of the revenue of top US companies.

- Create `DataFrame` `revenues`:

```
revenues = pd.DataFrame({
    'Rank': range(1,6),
    'Company': ['Walmart','Sinopec Group','State Grid',
                'China National Petroleum', 'Royal Dutch Shell'],
    'Revenue': [523964.0,407009.0,383906.0,379130.0,352106.0]})
print(revenues)
```

```
   Rank                   Company   Revenue
0     1                   Walmart  523964.0
1     2             Sinopec Group  407009.0
2     3                State Grid  383906.0
3     4  China National Petroleum  379130.0
4     5          Royal Dutch Shell  352106.0
```

- Create box plot: Set the appopriate y-axis for company revenue data, and set the `hover_data`, a list of one string value, to show the company name.

```
fig = px.box(
    data_frame=revenues,
    y='Revenue',
    hover_data=['Company'])
fig.show()
```

- The lesson dataset is much larger. Import it from CSV:

```
revenues2= pd.read_csv("./data/revenue_data.csv")
print(revenues2)
```

```
     Rank                    Company   Revenue
0       1                    Walmart  523964.0
1       2              Sinopec Group  407009.0
2       3                 State Grid  383906.0
3       4  China National Petroleum  379130.0
4       5          Royal Dutch Shell  352106.0
..    ...                        ...       ...
195   196             Auchan Holding   54672.0
196   197           Tencent Holdings   54613.0
197   198  Nippon Steel Corporation    54465.0
198   199             CNP Assurances   54365.0
199   200            Energy Transfer       NaN

[200 rows x 3 columns]
```

- New box plot:

```
fig = px.box(
    data_frame=revenues2,
    y='Revenue',
    hover_data=['Company'])
fig.show()
```

7. Histogram of company revenues

- Problem

  > The New York Stock exchange firm loved your previous box plot and want you to do more work for them. The box plot was a perfect visualization to help them understand the outliers and quartile-related attributes of their company revenue dataset. However, they want to understand a bit more about the distribution of the data. Are there many companies with smaller revenue, or larger revenue? Is it somewhat bell-shaped or skewed towards higher or lower revenues? You will help the investment team by creating a histogram of the revenue of top US companies.

- Make another histogram of `revenues` with `x=Revenues`, and the number of bins `nbin=5`

```
import plotly.express as px
fig = px.histogram(
    data_frame=revenues2,
    x='Revenue',
    nbins=5)
fig.show()
```

8. Customizing color

- Customizing in `plotly`:
  1. At figure creation if argument exists (e.g. `color`)
  2. After figure creation using a function: `fig.update_layout({'title':'A new title'})`
- Customizing color: R[ed]G[reen]B[lue] encoding (0-255) - (0,0,255) is blue, (255,255,0) is yellow.

- Example: Bar chart for student scores with additional column `city`

```
print(student_scores)
student_scores =  student_scores.assign(city=['Sydney','Melbourne','Sydney','Melbo
print(student_scores)
```

```
    name  score
0   John     80
1  Julia     97
2   Xuan     90
3  Harry     85
    name  score       city
0   John     80     Sydney
1  Julia     97  Melbourne
2   Xuan     90     Sydney
3  Harry     85  Melbourne
```

- New bar chart using `city` as `color` value:

```
import plotly.express as px
fig = px.bar(
    data_frame=student_scores,
    x='name',
    y='score',
    title='Student scores by student',
    color='city')
fig.show()
```

- When using the `color` argument with histograms you get stacked bars, and with box plots, multiple boxes (one per category).
- Specific colors in `plotly.express`: RGB codes or characters.

- Pick sandy yellow for 'Sydney', and navy blue for 'Melbourne', and add a dictionary `color_discrete_map`

```
import plotly.express as px
fig = px.bar(
    data_frame=student_scores,
    x='name',
    y='score',
    title='Student scores by student',
    color_discrete_map={
        'Melbourne': 'rgb(0,0,128)',
        'Sydney': 'rgb(235,207,52)'},
    color='city')
fig.show()
```

- Color scales with `color_continuous_scale` for single color scales of different shades, or for multiple colors merging into one another.

- Example:

```
import plotly.express as px
fig = px.bar(
    data_frame=weekly_temps,
    color='temp',
```

```
    color_continuous_scale='inferno')
fig.show()
```

- I got a `TypeError`: `color_continous_scale` not known: Tried `dir(px)` but that only lists the top level methods. To see the parameters of one of `px` functions, use `help(px.bar)` which shows all parameters.

```
[print(_) for _ in dir(px)]
```

- Construct your own color range:

```
## color scale: yellow through orange to red
my_scale=[('rgb(242,238,10)'),
          ('rgb(242,95,10)'),
          ('rgb(255,0,0)')]
fig=px.bar(
    data_frame=weekly_temps,
    x='day', # categorical
    y='temp', # numerical
    color_continuous_scale=my_scale,
    color='temp')
fig.show()
```

```
Opening in existing browser session.
```

9. Coloring student scores bar graph

- Problem:

> The previous plot that you created was well received by the school board, but they are wondering if there is a way for you to visually identify good and bad performers. This would be a great opportunity to utilize color. Specifically, a color scale. You think a scale from red (worst marks) to green (good marks) would be great.

- Check `DataFrame`

```
print(student_scores)
```

- Code:

```
# Create your own continuous color scale
my_scale = ['rgb(255,0,0)', 'rgb(3,252,40)']

# Create the bar plot
fig = px.bar(data_frame=student_scores,
             x='name', y='score', title='Student Scores by Student',
             # Set the color variable and scale
             color='score',
             color_continuous_scale=my_scale
             )

# Show the plot
fig.show()
```

10. Side-by-side revenue box plots with color

- Problem:

  The New York Stock Exchange firm you did work for previously has contracted you to extend on your work building the box plot of company revenues. They want to understand how different industries compare using this same visualization technique from before. They are also particular about what colors are used for what industries. They have prepared a list of industries and the colors as below. Your task is to create a box plot of company revenues, as before, but include the specified colors based on the list of industries given below:

  Industry-color RGB definitions:

  - Tech = 124, 250, 120
  - Oil = 112,128,144
  - Pharmaceuticals = 137, 109, 247
  - Professional Services = 255, 0, 0

- Load extended data set as `revenues3` `DataFrame`

```
import pandas as pd
revenues3 = pd.read_csv("data/revenue_data2.csv")
print(revenues3)
```

```
     Rank                   Company  Revenue  employees Industry  age
0       1                   Walmart   523964  2,300,000     Tech   44
1       2             Sinopec Group   407009     71,200     Tech   56
2       3                State Grid   383906    377,000      Oil   21
3       4  China National Petroleum   379130    123,000     Tech   33
4       5          Royal Dutch Shell   352106    260,000     Tech   70
..    ...                       ...      ...        ...      ...  ...
195   196            Auchan Holding    54672     49,000  Unknown   74
196   197          Tencent Holdings    54613     14,715  Unknown   78
197   198  Nippon Steel Corporation    54465     57,750  Unknown    9
198   199             CNP Assurances    54365     21,900  Unknown   65
199   200           Energy Transfer  Unknown    215,000  Unknown   79

[200 rows x 6 columns]
```

- For the color map, we need to know the exact values of `Industry`:

```
print(revenues3["Industry"].unique())
```

```
['Tech' 'Oil' 'Pharmaceuticals' 'Professional Services' 'Retail' 'Legal'
 'Unknown']
```

- Code:

```
import plotly.express as px
## Create the industry-color map
ind_color_map = {
    'Tech': 'rgb(124,250,120)',
    'Oil': 'rgb(112,128,144)',
```

```
    'Pharmaceuticals' : 'rgb(137, 109, 247)',
    'Professional Services': 'rgb(255, 0, 0)'}
## Create the basic box plot
fig = px.box(
    ## Set the data and y variable
    data_frame=revenues3,
    y='Revenue',
    ## Set the color map and variable
    color='Industry',
    color_discrete_map=ind_color_map)
## Show the plot
fig.show()
```

```
Opening in existing browser session.
```

11. Revenue histogram with stacked bars

- Problem:

  The New York Stock exchange firm thought your previous histogram provided great insight into how the revenue of the firms they are looking at is distributed. However, like before, they are interested in learning a bit more about how the industry of the firms could shed more light on what is happening. Your task is to re-create the histogram of company revenues, as before, but include the specified colors based on the list of industries given below.

  Industry-color RGB definitions:

  - Tech = 124, 250, 120
  - Oil = 112,128,144
  - Pharmaceuticals = 137, 109, 247
  - Professional Services = 255, 0, 0

- Check the data:

```
print(revenues3)
```

```
     Rank                   Company  Revenue  employees Industry  age
0       1                   Walmart   523964  2,300,000     Tech   44
1       2             Sinopec Group   407009     71,200     Tech   56
2       3                State Grid   383906    377,000      Oil   21
3       4  China National Petroleum   379130    123,000     Tech   33
4       5         Royal Dutch Shell   352106    260,000     Tech   70
..    ...                       ...      ...        ...      ...  ...
195   196            Auchan Holding    54672     49,000  Unknown   74
196   197          Tencent Holdings    54613     14,715  Unknown   78
197   198  Nippon Steel Corporation    54465     57,750  Unknown    9
198   199            CNP Assurances    54365     21,900  Unknown   65
199   200           Energy Transfer  Unknown    215,000  Unknown   79

[200 rows x 6 columns]
```

- They look identical to the DataCamp data except that the `employees` column is missing.

```
print(revenues3['Revenue'].unique())
```

- Check the data types:

```
revenues3.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 199 entries, 0 to 198
Data columns (total 6 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Rank       199 non-null    int64  
 1   Company    199 non-null    object 
 2   Revenue    199 non-null    float64
 3   employees  199 non-null    object 
 4   Industry   199 non-null    object 
 5   age        199 non-null    int64  
dtypes: float64(1), int64(2), object(3)
memory usage: 10.9+ KB
```

- This is different! The `Revenue` values don't have the `Dtype` `object` but instead `float64`, and there are `Unknown` values that stop the conversion to `float`, which otherwise would look like this:

```
revenues3['Revenue'] = revenues3['Revenue'].astype(float)
```

- Fixing that: Replace non-numeric values in 'Revenue' with NaN and drop them

```
import pandas as pd
import numpy as np
revenues3['Revenue'] = pd.to_numeric(revenues3['Revenue'], errors='coerce')
revenues3 = revenues3.dropna(subset=['Revenue'])
print(revenues3.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 199 entries, 0 to 198
Data columns (total 6 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Rank       199 non-null    int64  
 1   Company    199 non-null    object 
 2   Revenue    199 non-null    float64
 3   employees  199 non-null    object 
 4   Industry   199 non-null    object 
 5   age        199 non-null    int64  
dtypes: float64(1), int64(2), object(3)
memory usage: 10.9+ KB
None
```

- The code works now:

```
import plotly.express as px
## Create the industry-color map
ind_color_map = {
```

```
    'Tech': 'rgb(124,250,120)',
    'Oil': 'rgb(112,128,144)',
    'Pharmaceuticals' : 'rgb(137, 109, 247)',
    'Professional Services': 'rgb(255, 0, 0)'}
## Create the histogram
fig = px.histogram(
    ## Set the data and x variable
    data_frame=revenues3,
    x='Revenue',
    nbins=5,
    ## Set the color map and variable
    color='Industry',
    color_discrete_map=ind_color_map)
## Show the plot
fig.show()
```

Author: Marcus Birkenkrahe
Created: 2024-11-04 Mon 22:15