Plotly Express Arguments in Python

Input data arguments accepted by Plotly Express functions

e with

.bar

ames

umPy

s the

Plotly Express (px) is the high-level interface to Plotly and provides functions for generating charts. px functions support data provided in a number of different formats (long, wide, and mixed) and as different types of objects, including pandas and Polars dataframes.

Data for the Examples

The examples on this page use datasets available in the data package in px. px.data contains functions that when called return a dataset as a dataframe. Some of the datasets included in px.data are:

- carshare - Each row represents the availability of car-sharing services near the centroid of a zone in Montreal over a month-long period.
- election - Each row represents voting results for an electoral district in the 2013 Montreal mayoral election.
- iris - Each row represents a flower.

To access the iris dataset, we call its function and assign it to a variable:

```
import plotly.express as px

df = px.data.iris()
df.head()
```

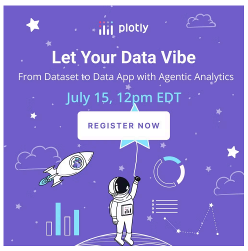
	sepal_length	sepal_width	petal_length	petal_width	species	species_id
0	5.1	3.5	1.4	0.2	setosa	1
1	4.9	3.0	1.4	0.2	setosa	1
2	4.7	3.2	1.3	0.2	setosa	1
3	4.6	3.1	1.5	0.2	setosa	1
4	5.0	3.6	1.4	0.2	setosa	1

By default px.data functions return a pandas DataFrame object, but you can specify an alternative dataframe type using return_type. pandas, polars, pyarrow, modin, and cuDF are supported return types.

```
df = px.data.iris(return_type='polars')
df.head()
```

shape: (5, 6)

sepal_length	sepal_width	petal_length	petal_width	species	species_id
f64	f64	f64	f64	str	i64
5.1	3.5	1.4	0.2	"setosa"	1
4.9	3.0	1.4	0.2	"setosa"	1
4.7	3.2	1.3	0.2	"setosa"	1
4.6	3.1	1.5	0.2	"setosa"	1
5.0	3.6	1.4	0.2	"setosa"	1



Long, Wide, and Mixed-Form Data

There are three common conventions for storing column-oriented data, usually in a data frame with column names:

- **long-form data** has one row per observation, and one column per variable. This is suitable for storing and displaying multivariate data i.e. with dimension greater than 2. This format is sometimes called "tidy".
- **wide-form data** has one row per value of one of the first variable, and one column per value of the second variable. This is suitable for storing and displaying 2-dimensional data.
- **mixed-form data** is a hybrid of long-form and wide-form data, with one row per value of one variable, and some columns representing values of another, and some columns representing more variables. See the wide-form documentation (/python/wide-form/) for examples of how to use Plotly Express to visualize this kind of data.

Every Plotly Express function can operate on long-form data (other than px.imshow which operates only on wide-form input), and in addition, the following 2D-Cartesian functions can operate on wide-form and mixed-form data: px.scatter, px.line, px.area, px.bar, px.histogram, px.violin, px.box, px.strip, px.funnel, px.density_heatmap and px.density_contour.

By way of example here is the same data, represented in long-form first, and then in wide-form:

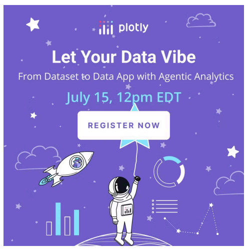
```
import plotly.express as px
long_df = px.data.medals_long()
long_df
```

	nation	medal	count
0	South Korea	gold	24
1	China	gold	10
2	Canada	gold	9
3	South Korea	silver	13
4	China	silver	15
5	Canada	silver	12
6	South Korea	bronze	11
7	China	bronze	8
8	Canada	bronze	12

```
import plotly.express as px
wide_df = px.data.medals_wide()
wide_df
```

	nation	gold	silver	bronze
0	South Korea	24	13	11
1	China	10	15	8
2	Canada	9	12	12

Plotly Express can produce the same plot from either form:



```
import plotly.express as px
long_df = px.data.medals_long()

fig = px.bar(long_df, x="nation", y="count", color="medal", title="Long-Form Input")
fig.show()
```

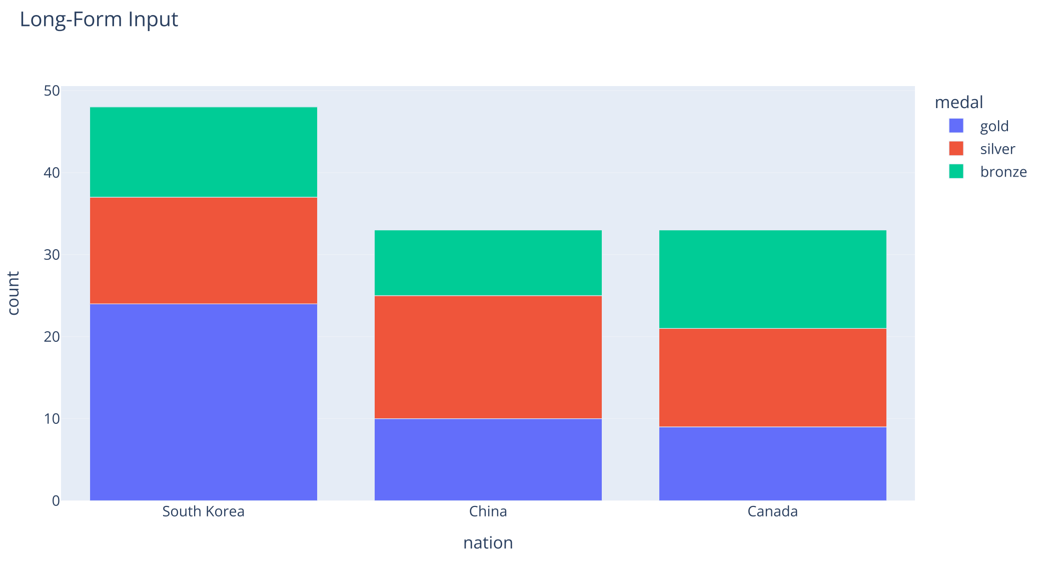
e with

.bar

ames

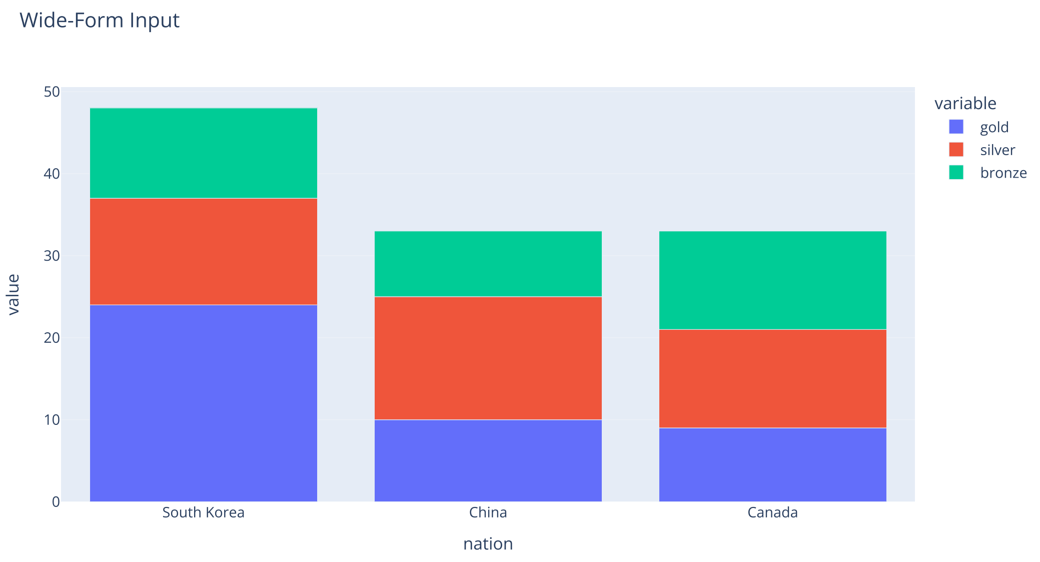
umPy

s the

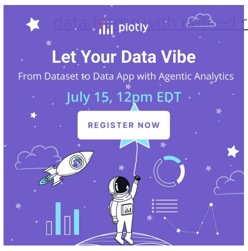


```
import plotly.express as px
wide_df = px.data.medals_wide()

fig = px.bar(wide_df, x="nation", y=["gold", "silver", "bronze"], title="Wide-Form Input")
fig.show()
```

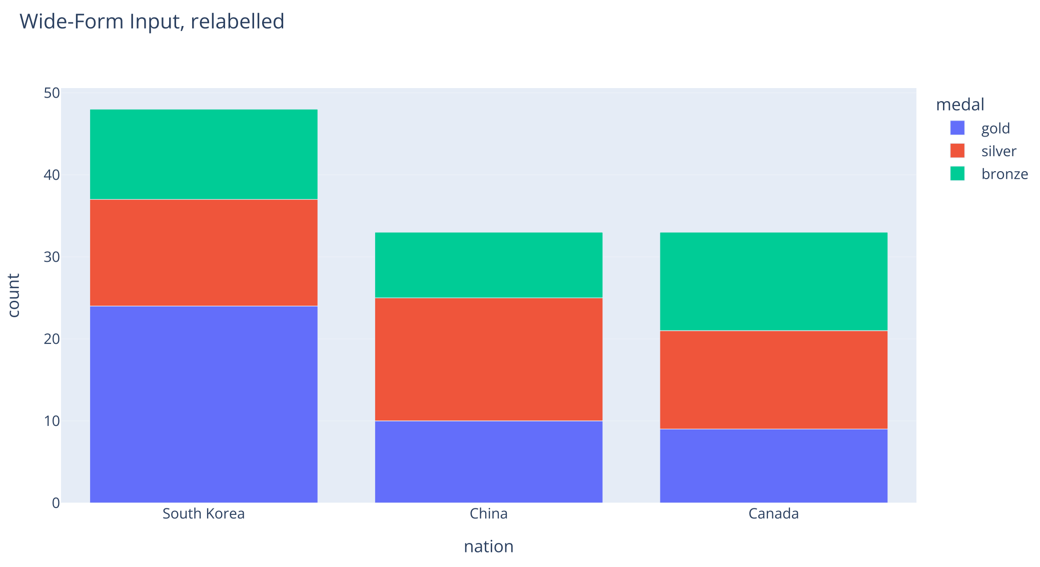


You might notice that y-axis and legend labels are slightly different for the second plot: they are "value" and "variable", respectively, and this is also reflected in the hoverlabel text. Note that the labels "medal" and "count" do not appear in the wide-form data frame, so in this case, you must supply these yourself, or you can use a row- and column-indexes (/python/wide-form/). You can rename these labels with the labels argument (/python/styling-plotly-express/):



```
import plotly.express as px
wide_df = px.data.medals_wide()

fig = px.bar(wide_df, x="nation", y=["gold", "silver", "bronze"], title="Wide-Form Input, relabelled",
             labels={"value": "count", "variable": "medal"})
fig.show()
```



Many more examples of wide-form and messy data input can be found in our detailed wide-form support documentation (/python/wide-form/).

Dataframe Input

The first argument of every px function is data_frame. If you provide a dataframe as a px function's first argument, you can then specify column names as strings from the dataframe as other arguments.

Supported DataFrame Types

px functions natively support pandas, Polars, and PyArrow dataframes. px uses Narwhals (https://narwhals-dev.github.io/narwhals/) to provide this native dataframe support. Other types of dataframes that are currently supported by Narwhals, for example cuDF and Modin, may also work with px.

You can also pass dataframes that are not natively supported, but which support the dataframe interchange protocol (https://data-apis.org/dataframe-protocol/latest/).

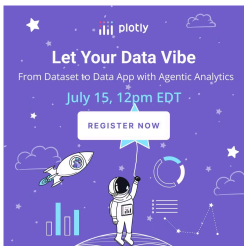
PySpark dataframes are also supported and are converted to pandas dataframes internally by Plotly Express.

Additional Dependencies Required

- Plotly Express requires NumPy. You can install it with pip install numpy if it's not installed by the dataframe library you are using.
- To use trendlines (/python/linear-fits/), you'll also need to have pandas installed.
- To use PySpark dataframes, you'll need to have pandas installed. To use dataframes that support the dataframe interchange protocol, you'll need to have PyArrow installed.

Example: Using a Pandas DataFrame with px.bar

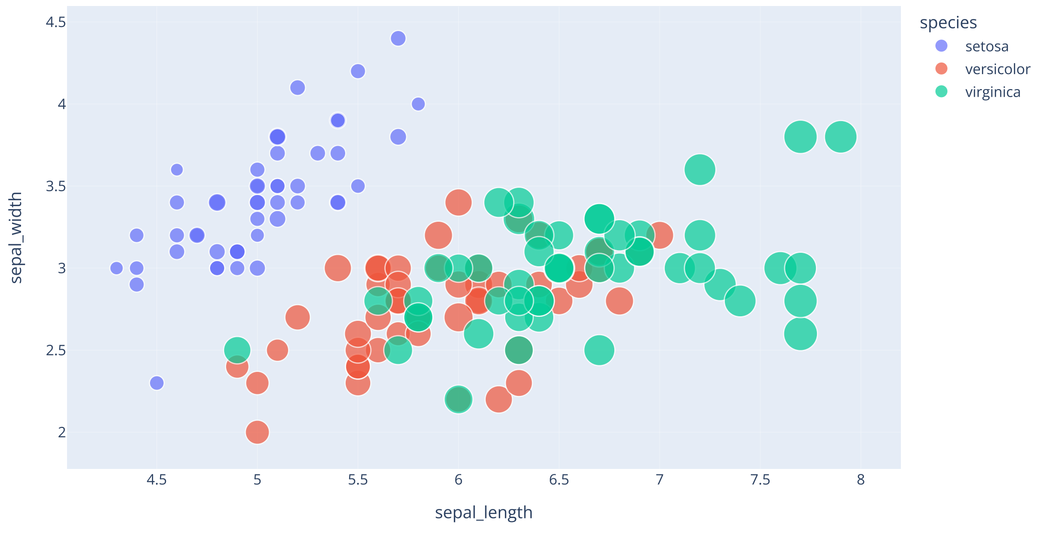
Here, we create a pandas DataFrame, pass it to px.bar as its first argument, and then use the "sepal_length" column for the x-axis and the "sepal_width" for the y-axis.



```
import plotly.express as px

df = px.data.iris()

fig = px.scatter(df, x='sepal_length', y='sepal_width', color='species', size='petal_length')
fig.show()
```



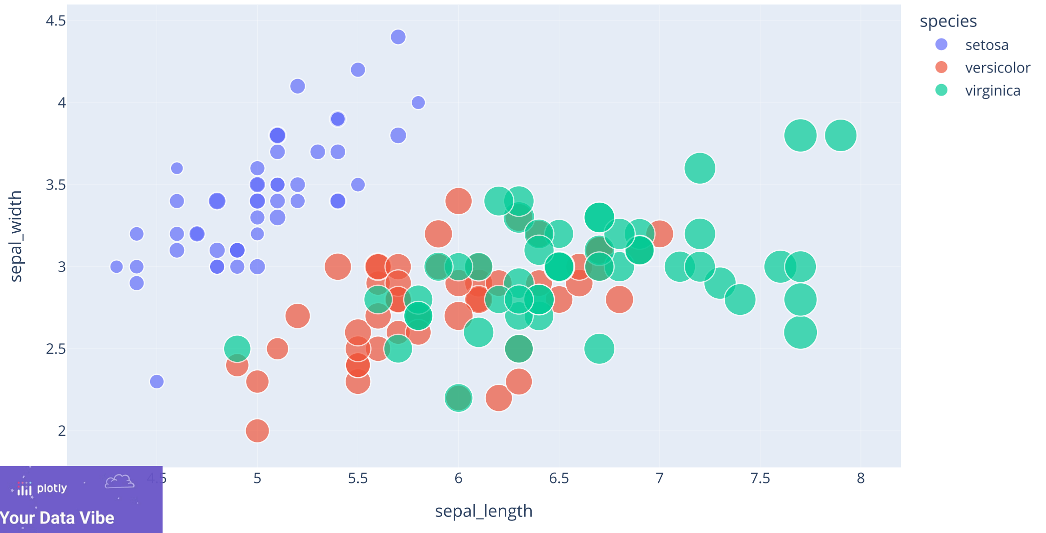
Example: Polars DataFrame with px.bar

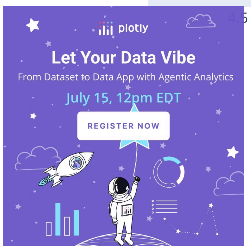
px provides native support for dataframe types other than pandas, including Polars:

```
import plotly.express as px

df = px.data.iris(return_type='polars')

fig = px.scatter(df, x='sepal_length', y='sepal_width', color='species', size='petal_length')
fig.show()
```

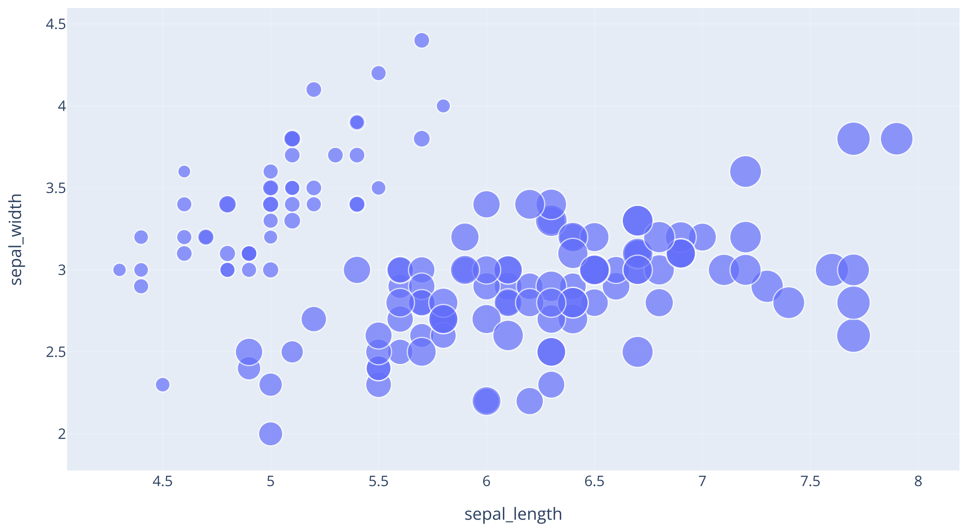




Using the Index of a Dataframe

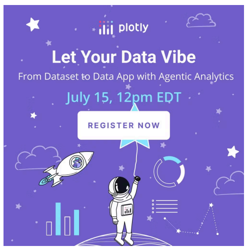
If the dataframe you are using has an index, it is also possible to use that index as an argument. In the following example, the index is used for the hover data.

```
import plotly.express as px
df = px.data.iris()
fig = px.scatter(df, x=df.sepal_length, y=df.sepal_width, size=df.petal_length,
                 hover_data=[df.index])
fig.show()
```



Using Columns from Multiple Dataframes

You can also use columns from multiple dataframes in one px function, as long as all the dataframe columns you use have the same length. In this example, we pass df1 as the data_frame argument to px.bar and then us a column from df2 for the y argument.



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

df1 = pd.DataFrame(dict(time=[10, 20, 30], sales=[10, 8, 30]))
df2 = pd.DataFrame(dict(market=[4, 2, 5]))
fig = px.bar(df1, x="time", y=df2.market, color="sales")
fig.show()
```

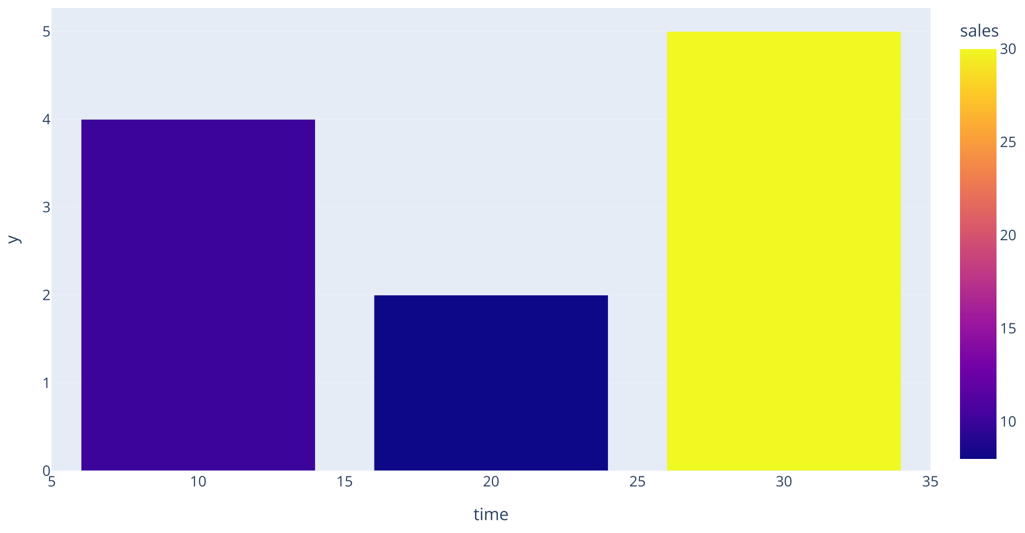
e with

.bar

rames

umPy

s the

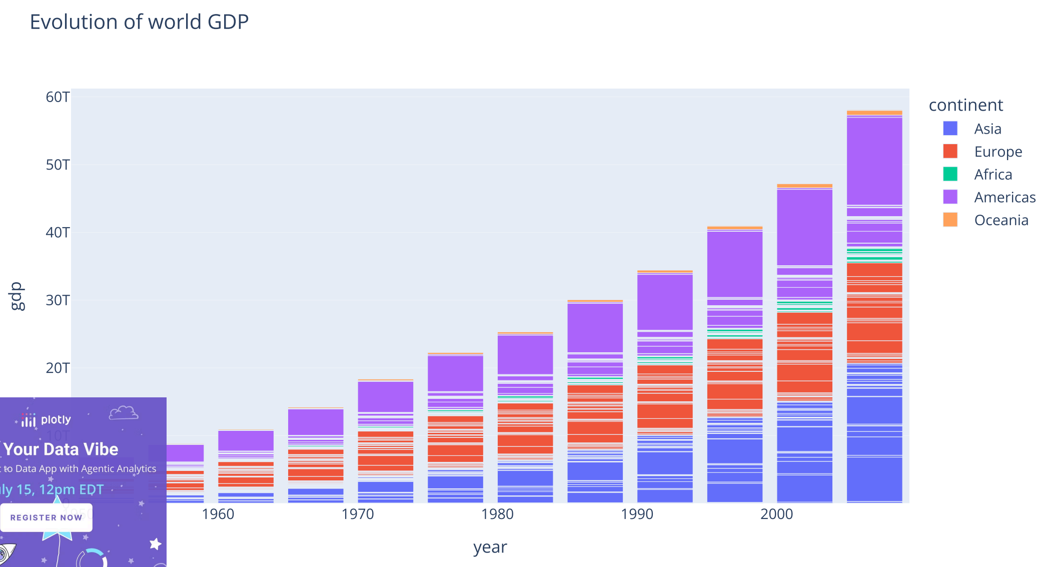


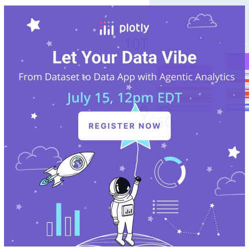
Using labels to pass names

The labels argument can be used to override the names used for axis titles, legend entries and hovers.

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

df = px.data.gapminder()
gdp = df['pop'] * df['gdpPercap']
fig = px.bar(df, x='year', y=gdp, color='continent', labels={'y':'gdp'},
             hover_data=['country'],
             title='Evolution of world GDP')
fig.show()
```





```
## Other Input Data
```

Input Data as array-like columns: NumPy arrays, lists...

px arguments can also be array-like objects such as lists, NumPy arrays, in both long-form or wide-form (for certain functions).

```
import plotly.express as px

# List arguments
fig = px.line(x=[1, 2, 3, 4], y=[3, 5, 4, 8])
fig.show()
```

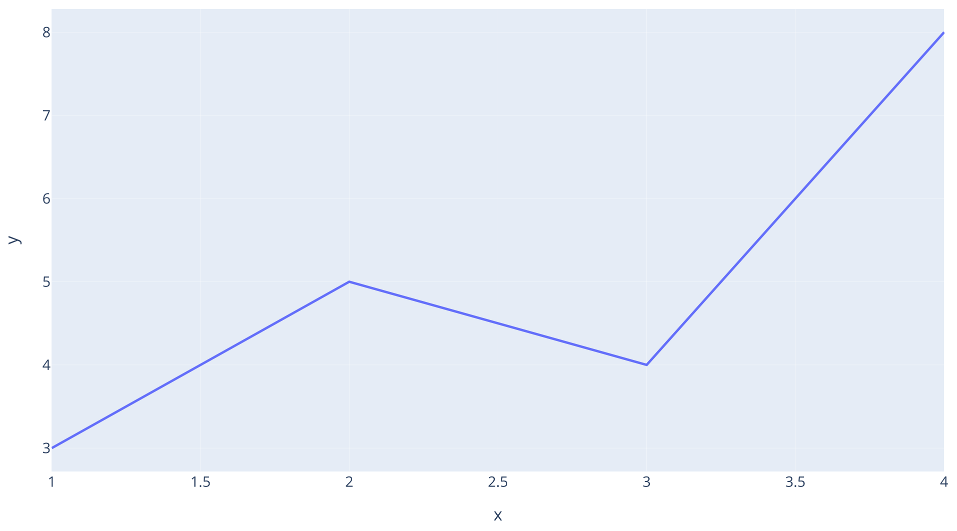
e with

.bar

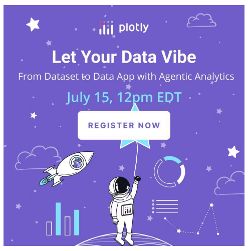
ames

umPy

s the



List arguments can also be passed in as a list of lists, which triggers wide-form data processing (/python/wide-form/), with the downside that the resulting traces will need to be manually renamed via fig.data[<n>].name = "name".



```
import plotly.express as px

# List arguments in wide form
series1 = [3, 5, 4, 8]
series2 = [5, 4, 8, 3]
fig = px.line(x=[1, 2, 3, 4], y=[series1, series2])
fig.show()
```

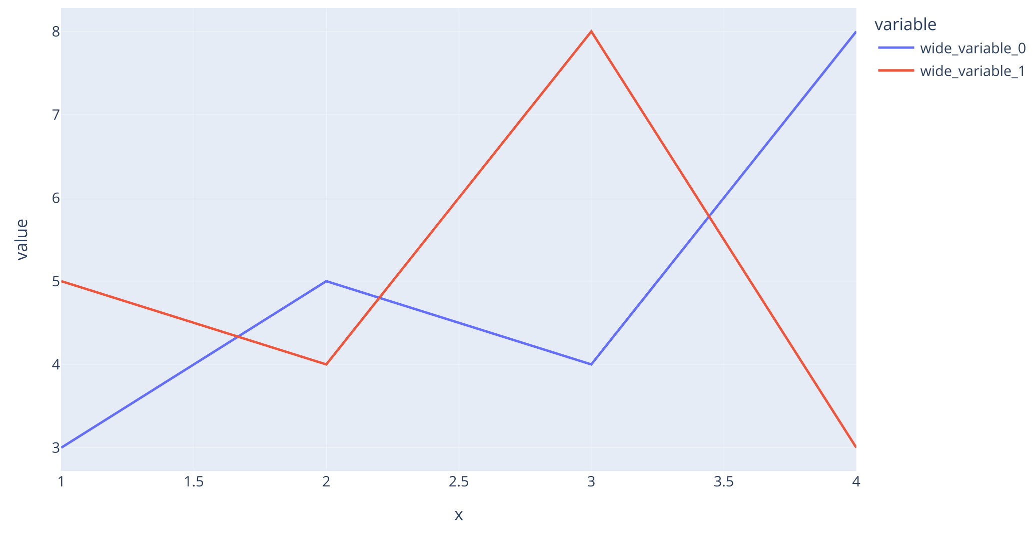
e with

.bar

ames

ımPy

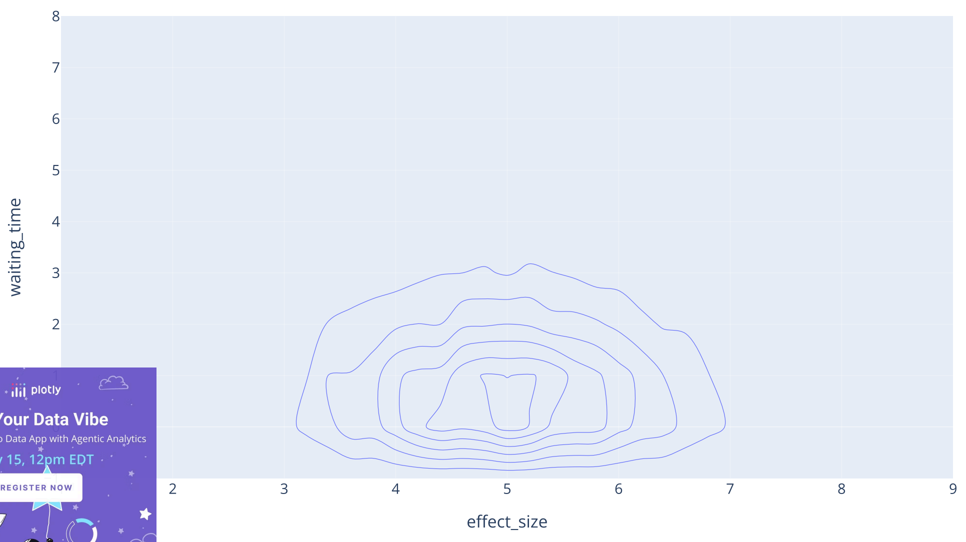
s the



Passing dictionaries or array-likes as the data_frame argument

The data_frame argument can also accept a dict or array in addition to DataFrame objects. Using a dictionary can be a convenient way to pass column names used in axis titles, legend entries and hovers without creating a dataframe.

```
import plotly.express as px
import numpy as np
N = 10000
np.random.seed(0)
fig = px.density_contour(dict(effect_size=5 + np.random.randn(N),
                              waiting_time=np.random.poisson(size=N)),
                         x="effect_size", y="waiting_time")
fig.show()
```



To pass a dict or an array (such as a NumPy ndarray) to the data_frame parameter, you'll need to have pandas installed, because plotly.express internally converts the dict or array to a pandas DataFrame.

Integer column names

When the data_frame argument is a NumPy array, column names are integer corresponding to the columns of the array. In this case, keyword names are used in axis, legend and hovers. This is also the case for a pandas DataFrame with integer column names. Use the labels argument to override these names.

e with

.bar

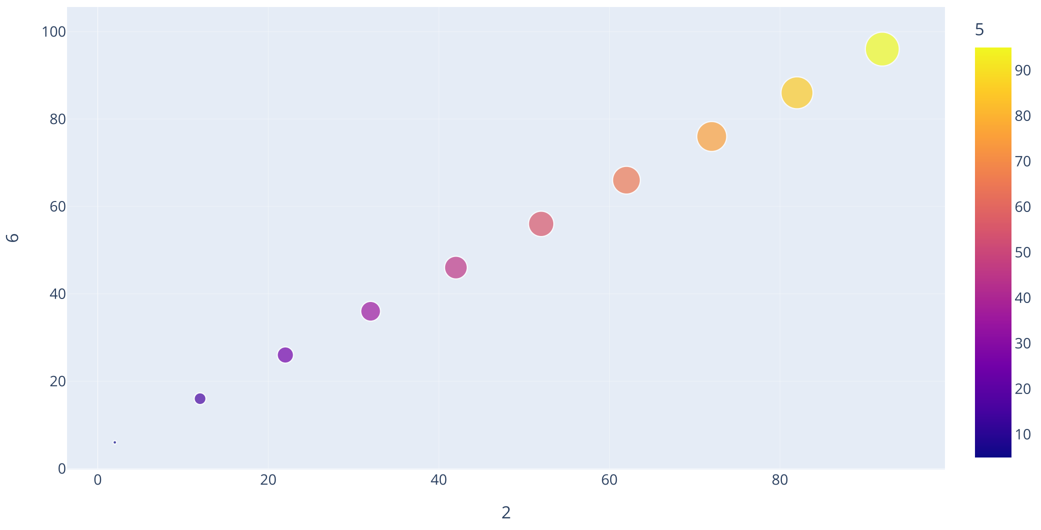
ames

umPy

s the

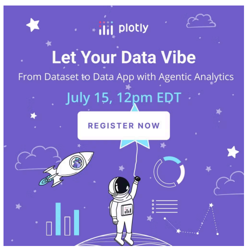
```
import numpy as np
import plotly.express as px

ar = np.arange(100).reshape((10, 10))
fig = px.scatter(ar, x=2, y=6, size=1, color=5)
fig.show()
```



Mixing dataframes and other types

It is possible to mix dataframe columns, NumPy arrays and lists as arguments. Remember that the only column names to be used correspond to columns in the data_frame argument, use labels to override names displayed in axis titles, legend entries or hovers.



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

df = px.data.gapminder()
gdp = np.log(df['pop'] * df['gdpPercap'])  # NumPy array
fig = px.bar(df, x='year', y=gdp, color='continent', labels={'y':'log gdp'},
             hover_data=['country'],
             title='Evolution of world GDP')
fig.show()
```

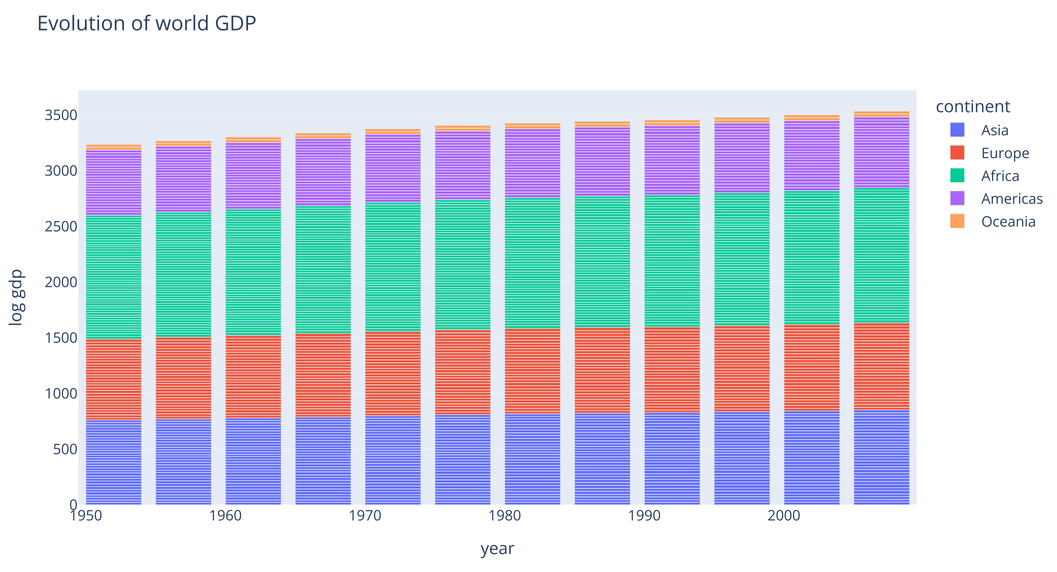
e with

.bar

rames

umPy

s the



What About Dash?

Dash (https://dash.plot.ly/) is an open-source framework for building analytical applications, with no Javascript required, and it is tightly integrated with the Plotly graphing library.

Learn about how to install Dash at https://dash.plot.ly/installation (https://dash.plot.ly/installation).

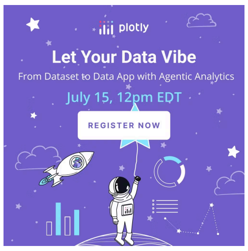
Everywhere in this page that you see fig.show(), you can display the same figure in a Dash application by passing it to the figure argument of the Graph component (https://dash.plot.ly/dash-core-components/graph) from the built-in dash_core_components package like this:

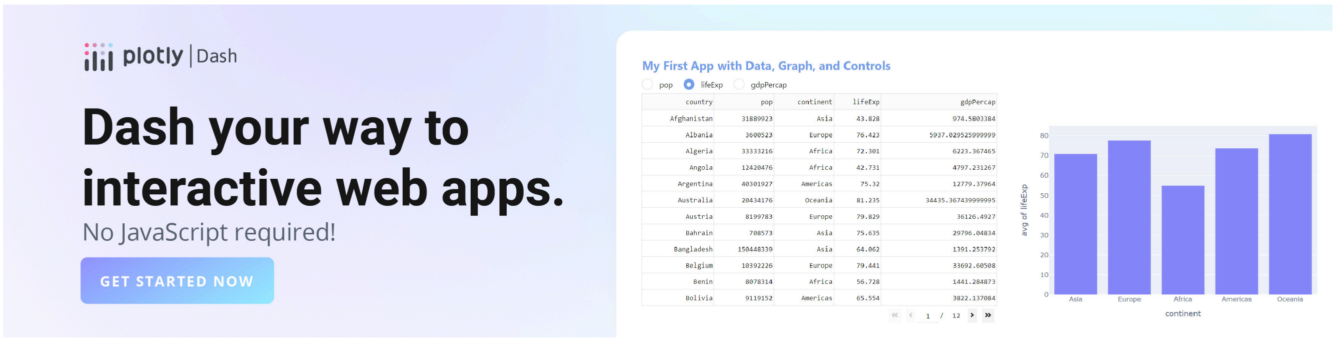
```
import plotly.graph_objects as go # or plotly.express as px
fig = go.Figure() # or any Plotly Express function e.g. px.bar(...)
# fig.add_trace( ... )
# fig.update_layout( ... )

from dash import Dash, dcc, html

app = Dash()
app.layout = html.Div([
    dcc.Graph(figure=fig)
])

app.run(debug=True, use_reloader=False)  # Turn off reloader if inside Jupyter
```





My First App with Data, Graph, and Controls

pop [✓ lifeExp] gdpPercap

country	pop	continent	lifeExp	gdpPercap
Afghanistan	31889923	Asia	43.828	974.5803384
Albania	3600523	Europe	76.423	5937.029525999999
Algeria	33333216	Africa	72.301	6223.367465
Angola	12420476	Africa	42.731	4797.231267
Argentina	40301927	Americas	75.32	12779.37964
Australia	20434176	Oceania	81.235	34435.367439999995
Austria	8199783	Europe	79.829	36126.4927
Bahrain	708573	Asia	75.635	29796.04834
Bangladesh	150448339	Asia	64.062	1391.253792
Belgium	10392226	Europe	79.441	33692.60508
Benin	8078314	Africa	56.728	1441.284873
Bolivia	9119152	Americas	65.554	3822.137084

1 / 12

(https://dash.plotly.com/tutorial?utm_medium=graphing_libraries&utm_content=python_footer)

e with

JOIN OUR MAILING LIST

.bar

Sign up to stay in the loop with all things Plotly — from Dash Club to product updates, webinars, and more!

rames

SUBSCRIBE (HTTPS://GO.PLOT.LY/SUBSCRIPTION)

umPy

Products

Dash (https://plotly.com/dash/)
Consulting and Training (https://plotly.com/consulting-and-oem/)

Pricing

Enterprise Pricing (https://plotly.com/get-pricing/)

About Us

s the

Careers (https://plotly.com/careers)
Resources (https://plotly.com/resources/)
Blog (https://medium.com/@plotlygraphs)

Support

Community Support (https://community.plot.ly/)
Documentation (https://plotly.com/graphing-libraries)