



# Histograms in Python

How to make Histograms in Python with Plotly.

Plotly Studio: Transform any dataset into an interactive data application in minutes with AI. [Sign up for early access now.](https://plotly.com/studio/?utm_medium=graphing-libraries&utm_campaign=studio_early_access&utm_content=sidebar) ([https://plotly.com/studio/?utm\\_medium=graphing-libraries&utm\\_campaign=studio\\_early\\_access&utm\\_content=sidebar](https://plotly.com/studio/?utm_medium=graphing-libraries&utm_campaign=studio_early_access&utm_content=sidebar))

In statistics, a [histogram](https://en.wikipedia.org/wiki/Histogram) (<https://en.wikipedia.org/wiki/Histogram>) is representation of the distribution of numerical data, where the data are binned and the count for each bin is represented. More generally, in Plotly a histogram is an aggregated bar chart, with several possible aggregation functions (e.g. sum, average, count...) which can be used to visualize data on categorical and date axes as well as linear axes.

values

an count

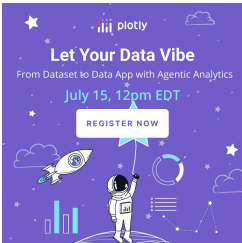
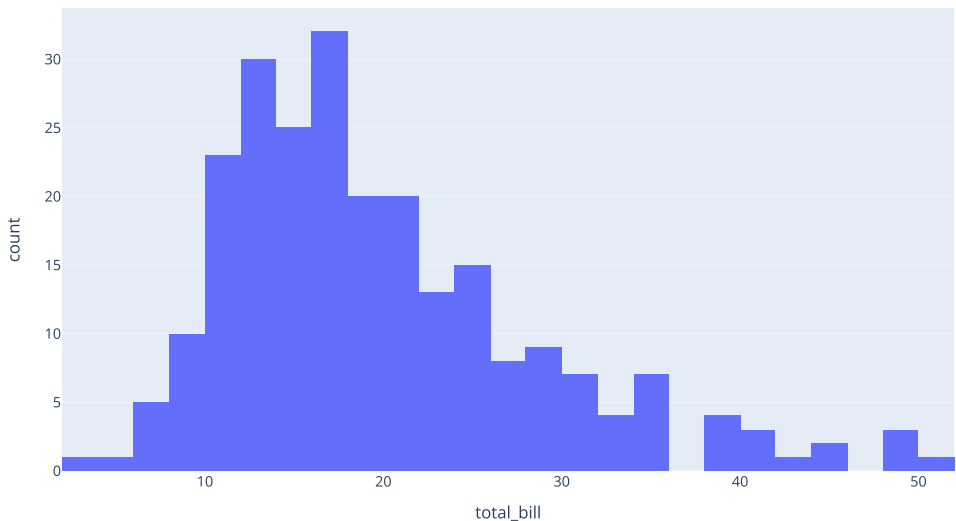
Alternatives to histogram plots for visualizing distributions include [violin plots](https://plotly.com/python/violin/) (<https://plotly.com/python/violin/>), [box plots](https://plotly.com/python/box-plots/) (<https://plotly.com/python/box-plots/>), [ECDF plots](https://plotly.com/python/ecdf-plots/) (<https://plotly.com/python/ecdf-plots/>) and [strip charts](https://plotly.com/python/strip-charts/) (<https://plotly.com/python/strip-charts/>).

If you're looking instead for bar charts, i.e. representing *raw, unaggregated* data with rectangular bar, go to the [Bar Chart tutorial](/python/bar-charts/) ([python/bar-charts/](/python/bar-charts/)).

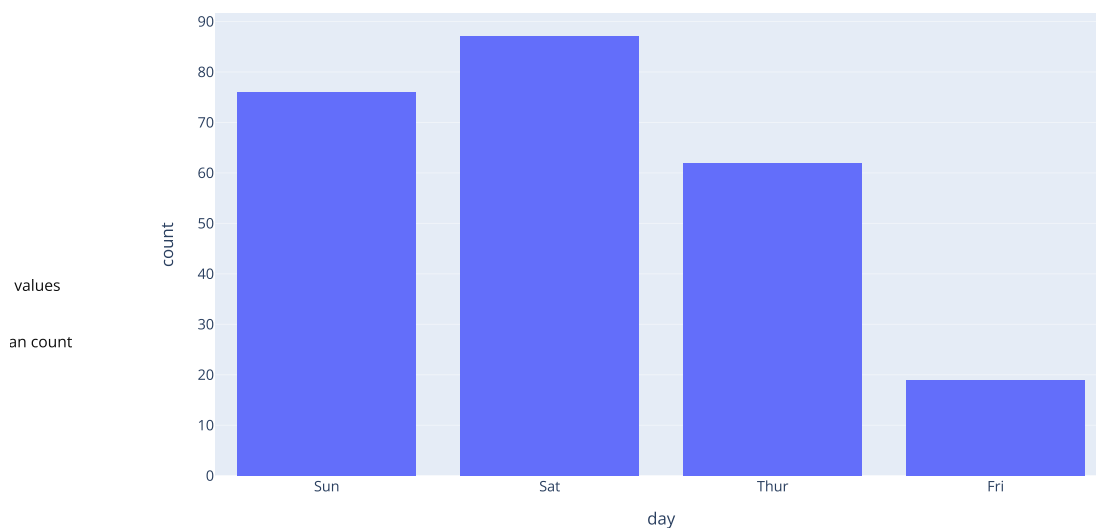
## Histograms with Plotly Express

[Plotly Express](/python/plotly-express/) ([python/plotly-express/](/python/plotly-express/)) is the easy-to-use, high-level interface to Plotly, which [operates on a variety of types of data](/python/px-arguments/) ([python/px-arguments/](/python/px-arguments/)) and produces [easy-to-style figures](/python/styling-plotly-express/) ([python/styling-plotly-express/](/python/styling-plotly-express/)).

```
import plotly.express as px
df = px.data.tips()
fig = px.histogram(df, x="total_bill")
fig.show()
```



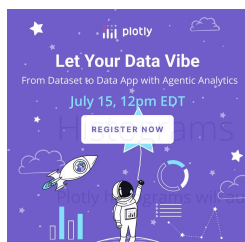
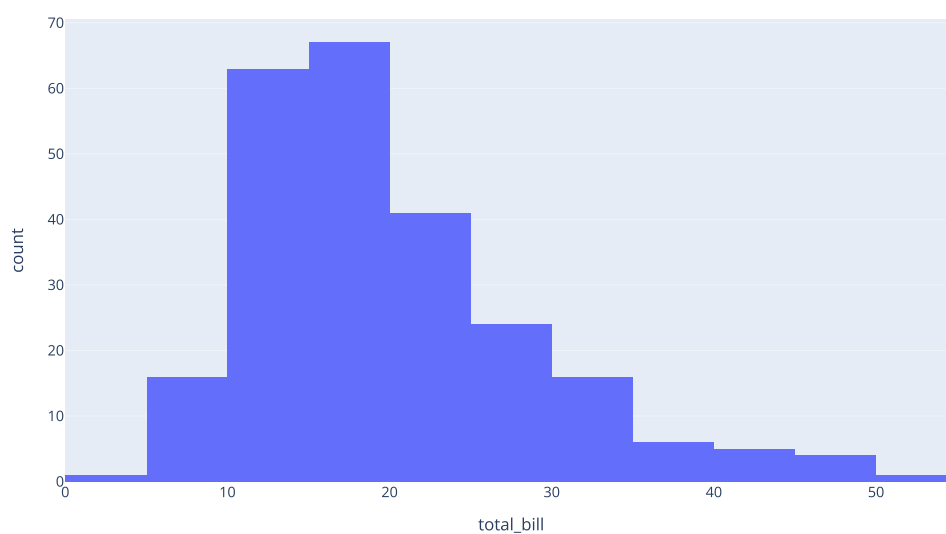
```
import plotly.express as px
df = px.data.tips()
# Here we use a column with categorical data
fig = px.histogram(df, x="day")
fig.show()
```



## Choosing the number of bins

By default, the number of bins is chosen so that this number is comparable to the typical number of samples in a bin. This number can be customized, as well as the range of values.

```
import plotly.express as px
df = px.data.tips()
fig = px.histogram(df, x="total_bill", nbins=20)
fig.show()
```

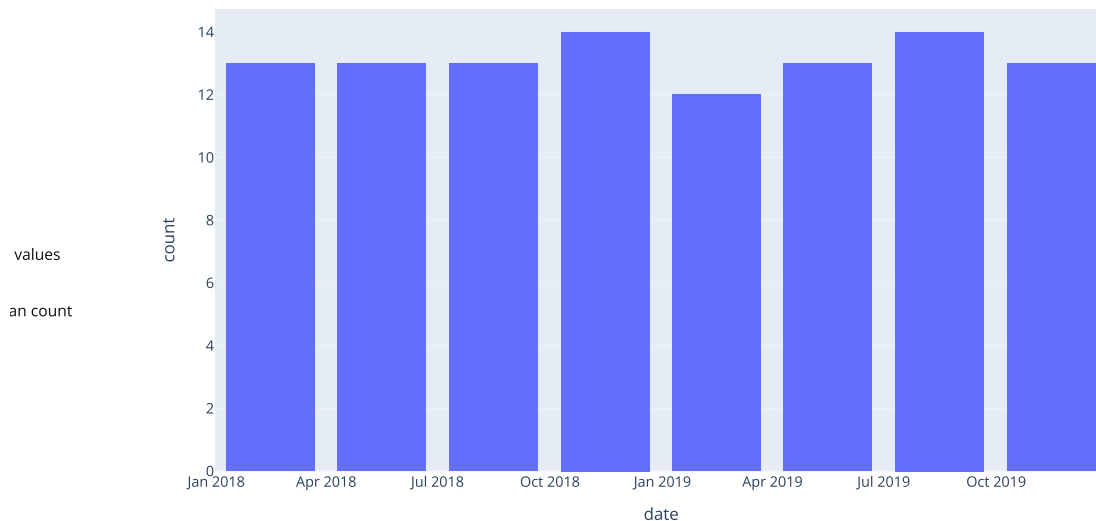


## Bin on Date Data

Plotly histograms will automatically bin date data in addition to numerical data:

```
import plotly.express as px

df = px.data.stocks()
fig = px.histogram(df, x="date")
fig.update_layout(bargap=0.2)
fig.show()
```

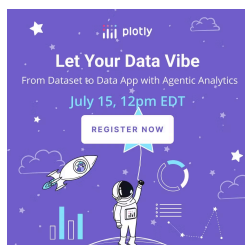
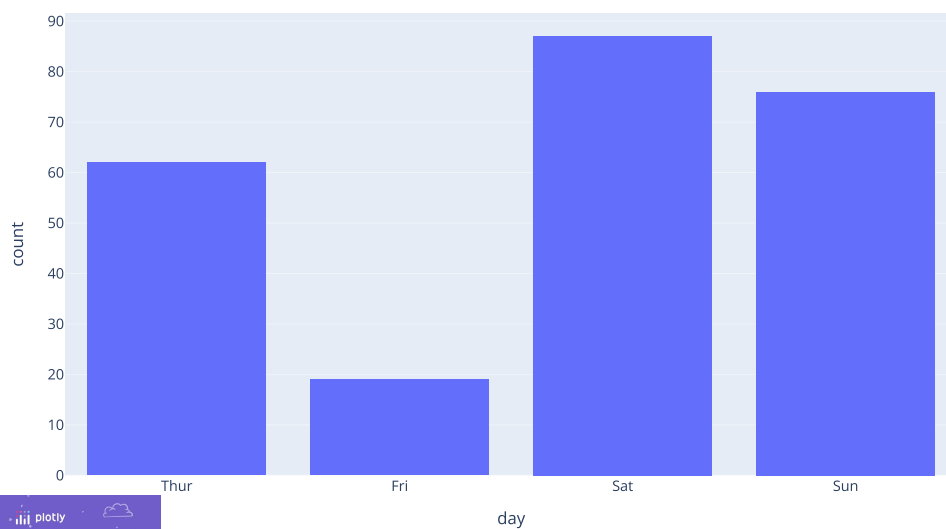


## Histograms on Categorical Data

Plotly histograms will automatically bin numerical or date data but can also be used on raw categorical data, as in the following example, where the X-axis value is the categorical "day" variable:

```
import plotly.express as px

df = px.data.tips()
fig = px.histogram(df, x="day", category_orders=dict(day=["Thur", "Fri", "Sat", "Sun"]))
fig.show()
```



Histograms in Dash

Dash (<https://plotly.com/dash/>) is the best way to build analytical apps in Python using Plotly figures. To run the app below, run pip install dash, click "Download" to get the code and run python app.py.

Get started with [the official Dash docs \(https://dash.plotly.com/installation\)](https://dash.plotly.com/installation) and **learn how to effortlessly** [style \(https://plotly.com/dash/design-kit/\)](https://plotly.com/dash/design-kit/) & [deploy \(https://plotly.com/dash/app-manager/\)](https://plotly.com/dash/app-manager/) apps like this with [Dash Enterprise \(https://plotly.com/dash/\)](https://plotly.com/dash/).

values  
an count

```
from dash import Dash, dcc, html, Input, Output
import plotly.express as px
import numpy as np

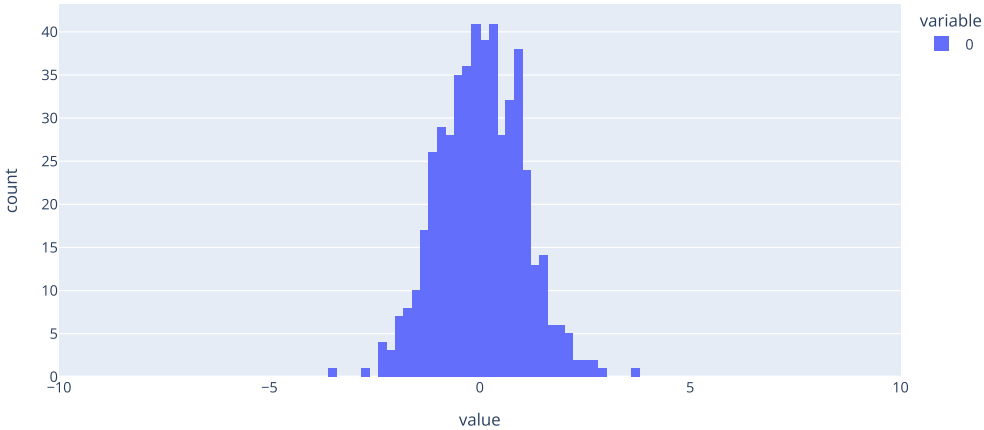
app = Dash(__name__)

app.layout = html.Div([
    html.H4('Interactive normal distribution'),
    dcc.Graph(id="graph"),
    html.P("Mean:"),
    dcc.Slider(id="mean", min=-3, max=3, value=0,
              marks={-3: '-3', 3: '3'}),
    html.P("Standard Deviation:"),
    dcc.Slider(id="std", min=1, max=3, value=1,
              marks={1: '1', 3: '3'}),
])

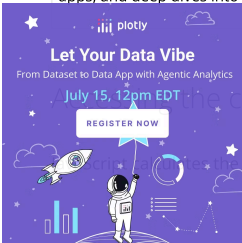
@app.callback(
    Output("graph", "figure"),
    Input("mean", "value"),
    Input("std", "value"))
def display_color(mean, std):
    data = np.random.normal(mean, std, size=500) # replace with your own data source
    fig = px.histogram(data, nbins=50, color_discrete_sequence=['#1f77b4'])
```

DOWNLOAD

Interactive normal distribution



**Sign up for Dash Club** → Free cheat sheets plus updates from Chris Parmer and Adam Schroeder delivered to your inbox every two months. Includes tips and tricks, community apps, and deep dives into the Dash architecture. [Join now \(https://go.plotly.com/dash-club?utm\\_source=Dash+Club+2022&utm\\_medium=graphing\\_libraries&utm\\_content=inline\)](https://go.plotly.com/dash-club?utm_source=Dash+Club+2022&utm_medium=graphing_libraries&utm_content=inline).



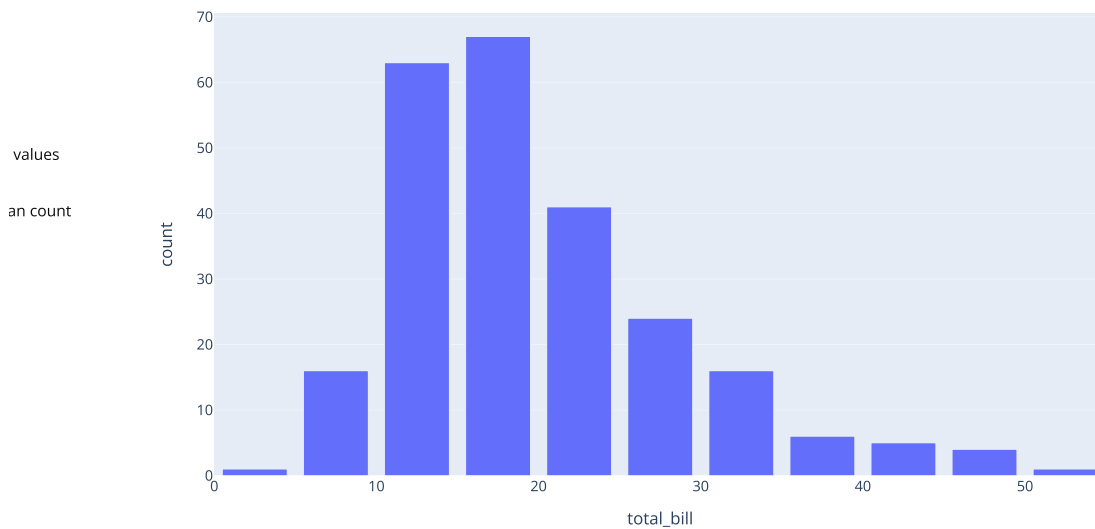
counts (y-axis) values

y-axis (count) values on the fly in the browser, so it's not accessible in the fig. You can manually calculate it using np.histogram.

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

df = px.data.tips()
# create the bins
counts, bins = np.histogram(df.total_bill, bins=range(0, 60, 5))
bins = 0.5 * (bins[:-1] + bins[1:])

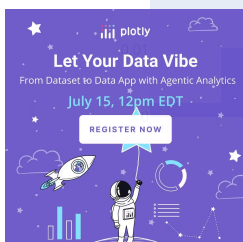
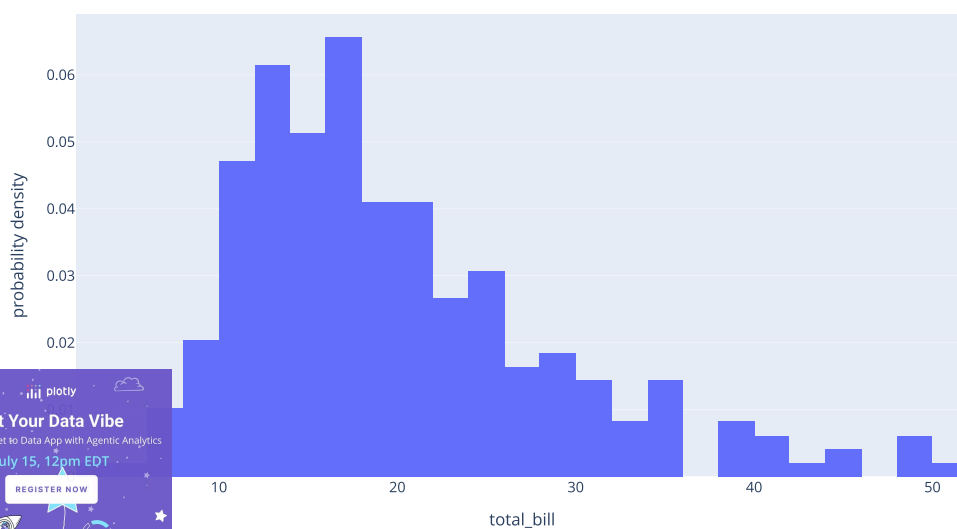
fig = px.bar(x=bins, y=counts, labels={'x': 'total_bill', 'y': 'count'})
fig.show()
```



## Type of normalization

The default mode is to represent the count of samples in each bin. With the `histnorm` argument, it is also possible to represent the percentage or fraction of samples in each bin (`histnorm='percent'` or `probability`), or a density histogram (the sum of all bar areas equals the total number of sample points, density), or a probability density histogram (the sum of all bar areas equals 1, probability density).

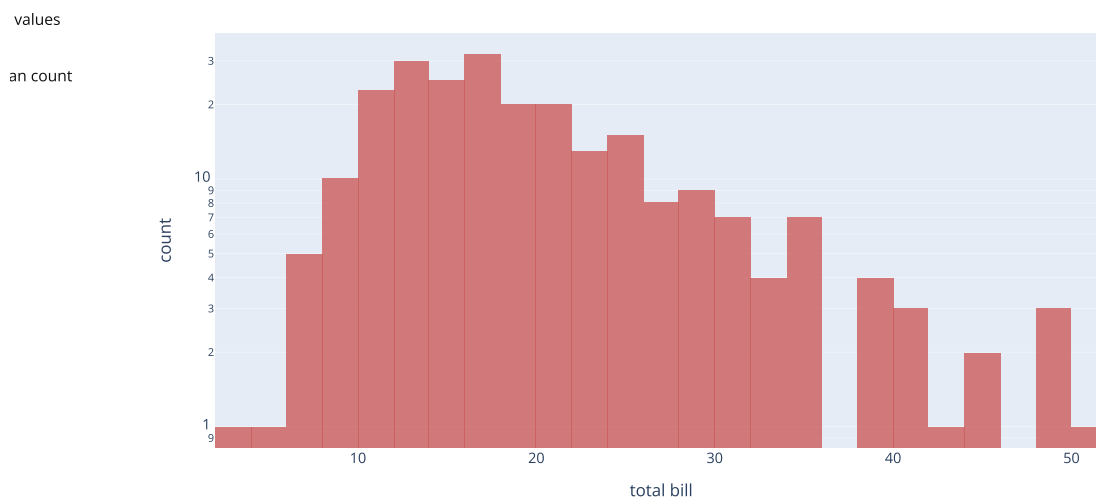
```
import plotly.express as px
df = px.data.tips()
fig = px.histogram(df, x="total_bill", histnorm='probability density')
fig.show()
```



## Aspect of the histogram plot

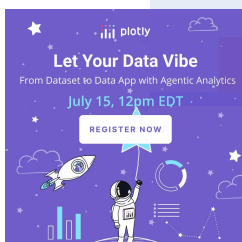
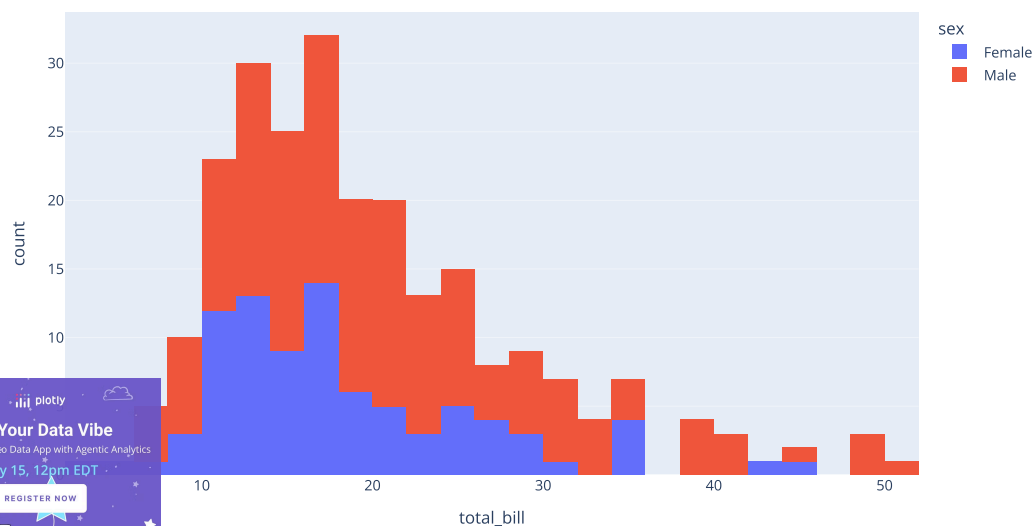
```
import plotly.express as px
df = px.data.tips()
fig = px.histogram(df, x="total_bill",
                  title='Histogram of bills',
                  labels={'total_bill': 'total bill'}, # can specify one label per df column
                  opacity=0.8,
                  log_y=True, # represent bars with log scale
                  color_discrete_sequence=['indianred'] # color of histogram bars
                  )
fig.show()
```

Histogram of bills



## Several histograms for the different values of one column

```
import plotly.express as px
df = px.data.tips()
fig = px.histogram(df, x="total_bill", color="sex")
fig.show()
```

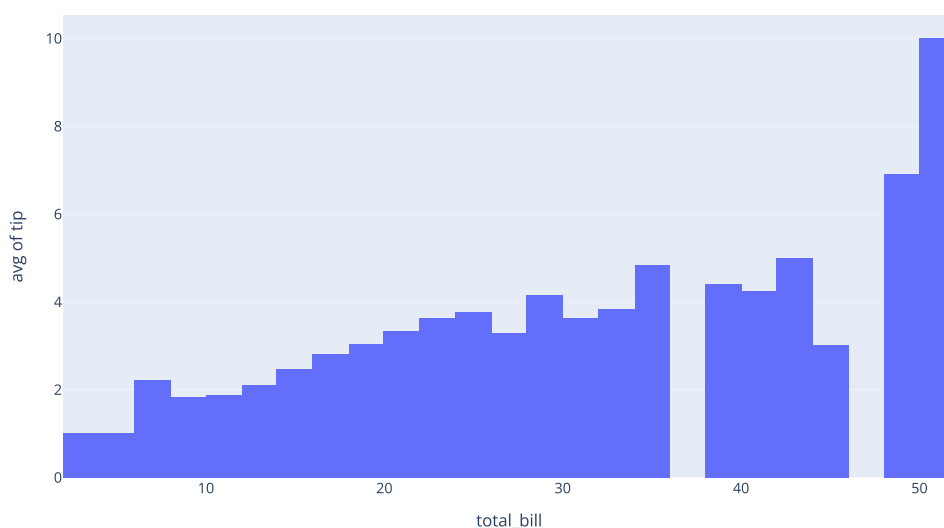


## Aggregating with other functions than count

For each bin of  $x$ , one can compute a function of data using `histfunc`. The argument of `histfunc` is the dataframe column given as the  $y$  argument. Below the plot shows that the average tip increases with the total bill.

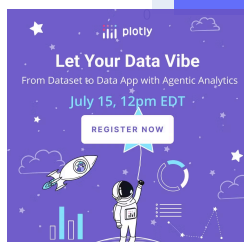
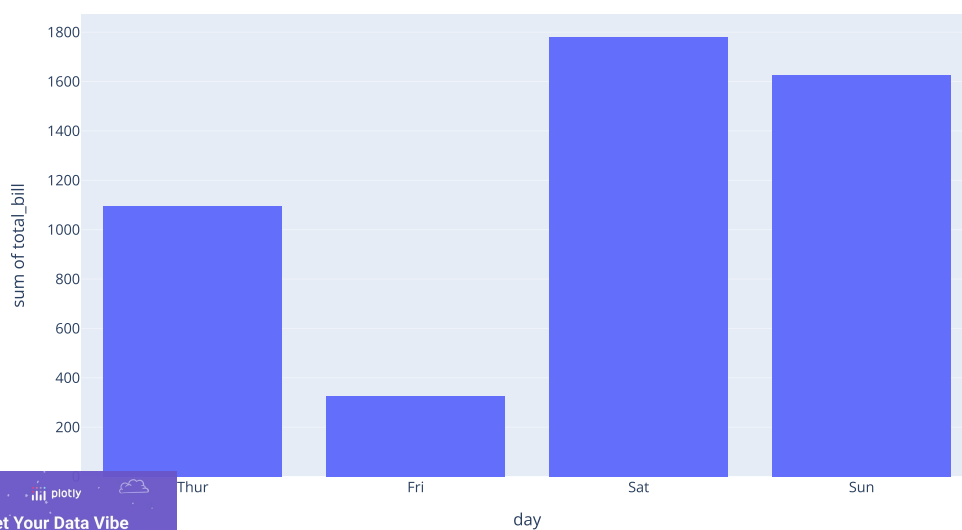
```
import plotly.express as px
df = px.data.tips()
fig = px.histogram(df, x="total_bill", y="tip", histfunc='avg')
fig.show()
```

values  
an count



The default `histfunc` is `sum` if  $y$  is given, and works with categorical as well as binned numeric data on the  $x$  axis:

```
import plotly.express as px
df = px.data.tips()
fig = px.histogram(df, x="day", y="total_bill", category_orders=dict(day=["Thur", "Fri", "Sat", "Sun"]))
fig.show()
```

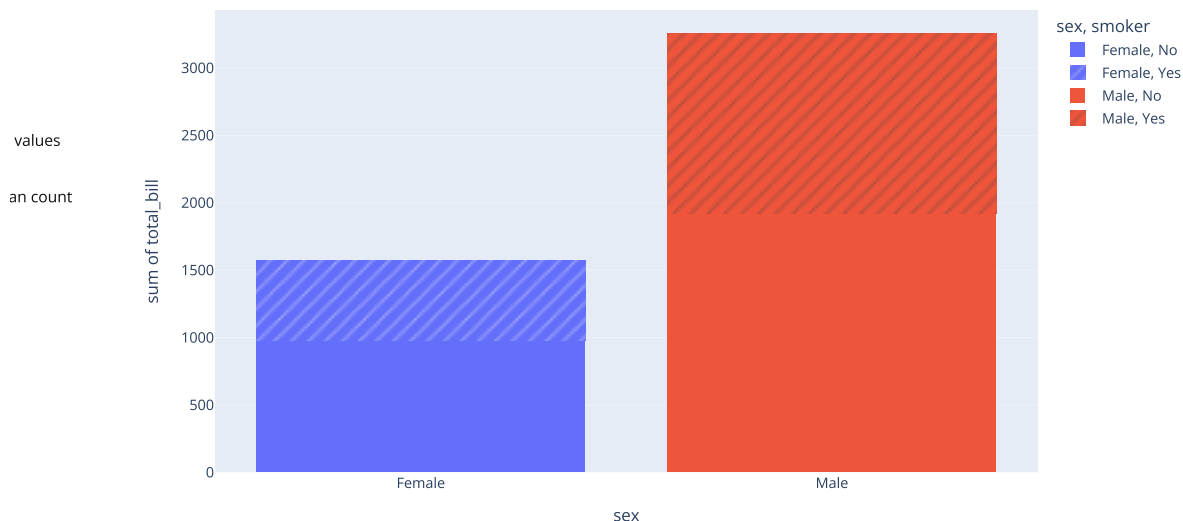


New in v5.0

Histograms afford the use of [patterns \(also known as hatching or texture\)](#) ([python/pattern-hatching-texture/](#)) in addition to color:

```
import plotly.express as px

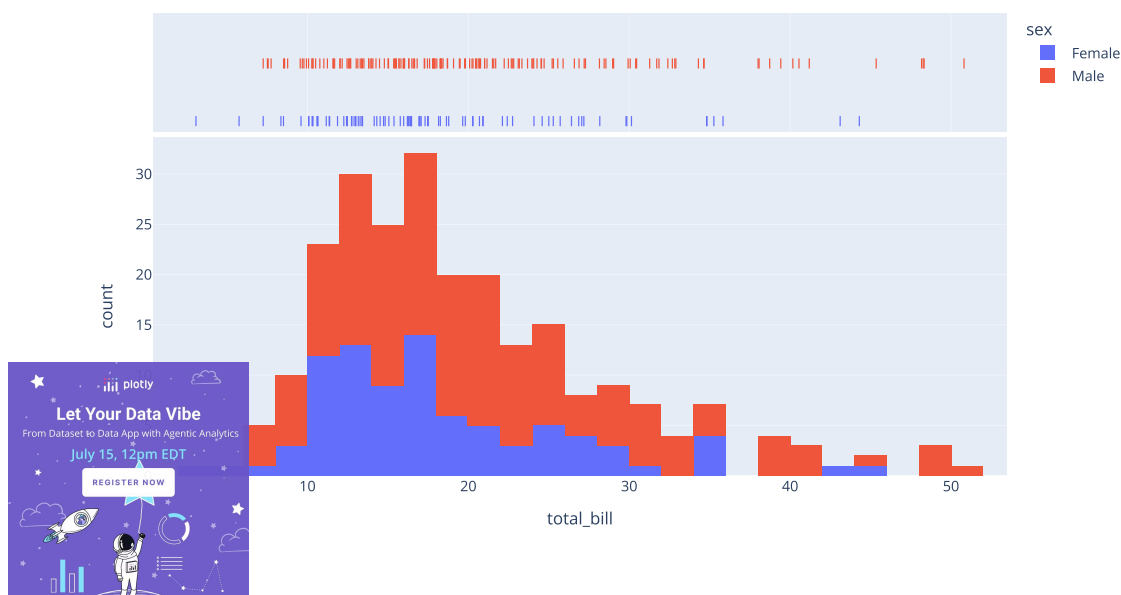
df = px.data.tips()
fig = px.histogram(df, x="sex", y="total_bill", color="sex", pattern_shape="smoker")
fig.show()
```



## Visualizing the distribution

With the `marginal` keyword, a [marginal](https://plotly.com/python/marginal-plots/) (<https://plotly.com/python/marginal-plots/>) is drawn alongside the histogram, visualizing the distribution. See [the distplot page](https://plotly.com/python/distplot/) (<https://plotly.com/python/distplot/>) for more examples of combined statistical representations.

```
import plotly.express as px
df = px.data.tips()
fig = px.histogram(df, x="total_bill", color="sex", marginal="rug", # can be `box`, `violin`
                  hover_data=df.columns)
fig.show()
```



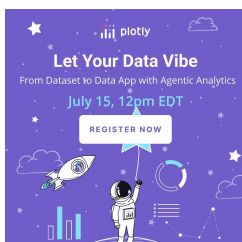
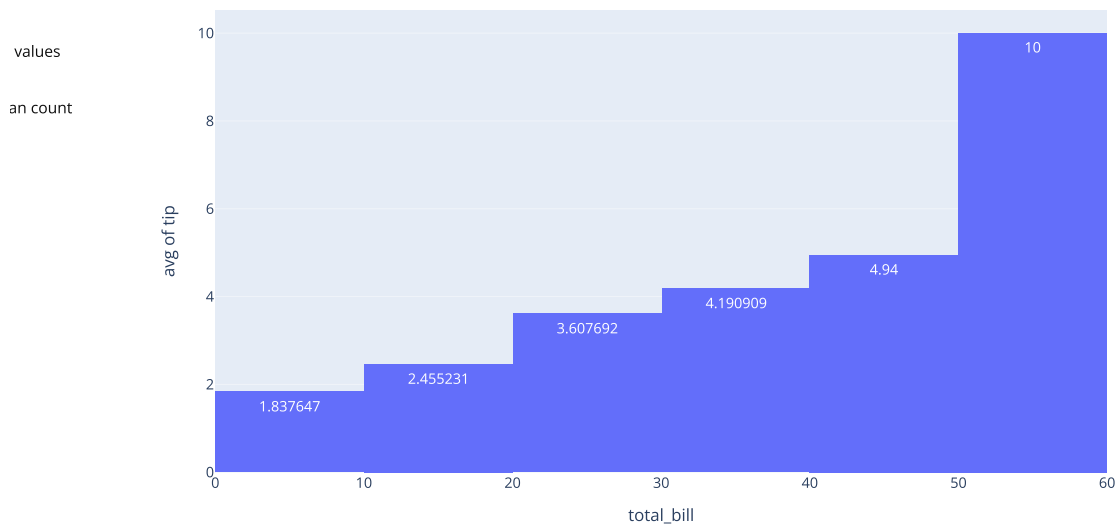


## Adding text labels

*New in v5.5*

You can add text to histogram bars using the `text_auto` argument. Setting it to `True` will display the values on the bars, and setting it to a d3-format formatting string will control the output format.

```
import plotly.express as px
df = px.data.tips()
fig = px.histogram(df, x="total_bill", y="tip", histfunc="avg", nbins=8, text_auto=True)
fig.show()
```



## Histograms with go.Histogram

If Plotly Express does not provide a good starting point, it is also possible to use [the more generic go.Histogram class from plotly.graph\\_objects \(/python/graph-objects/\)](https://plotly.com/python/reference#histogram). All of the available histogram options are described in the histogram section of the reference page: <https://plotly.com/python/reference#histogram> (<https://plotly.com/python/reference#histogram>).

### Basic Histogram

```
import plotly.graph_objects as go

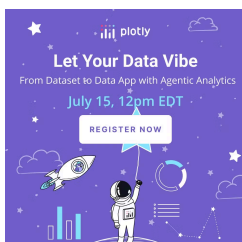
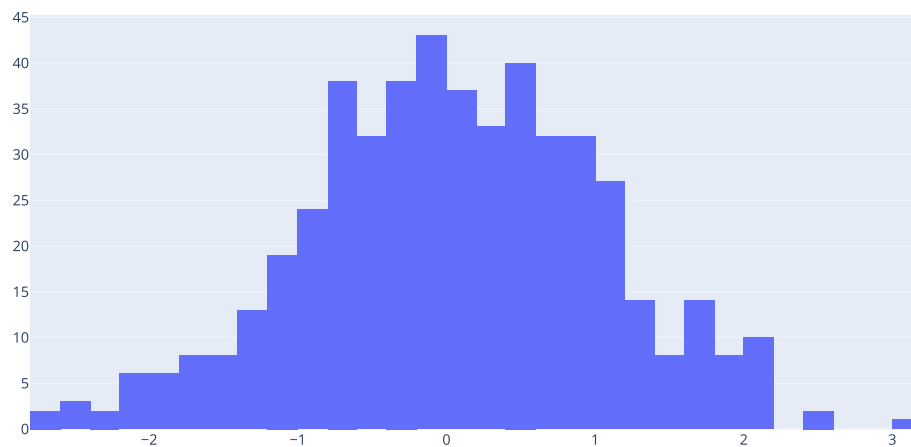
import numpy as np
np.random.seed(1)

x = np.random.randn(500)

fig = go.Figure(data=[go.Histogram(x=x)])
fig.show()
```

values

an count



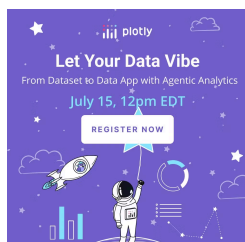
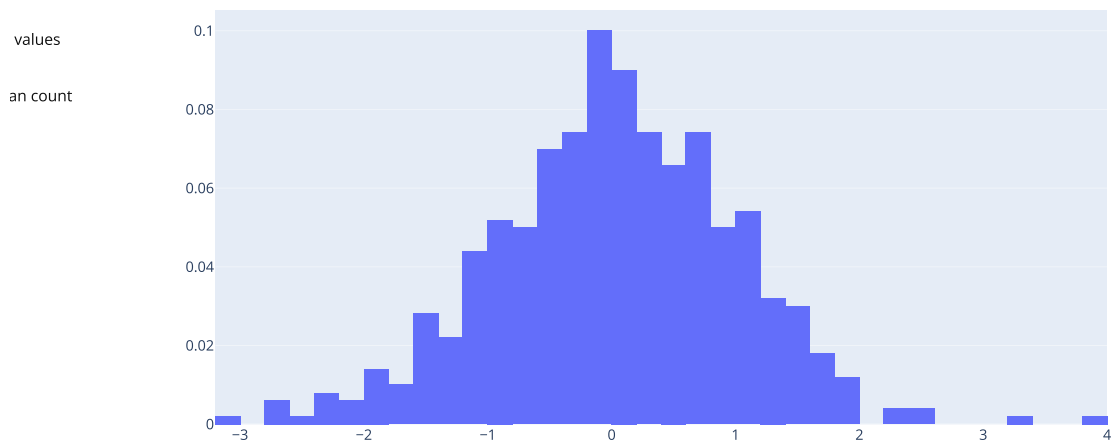
## Normalized Histogram

```
import plotly.graph_objects as go

import numpy as np

x = np.random.randn(500)
fig = go.Figure(data=[go.Histogram(x=x, histnorm='probability')])

fig.show()
```



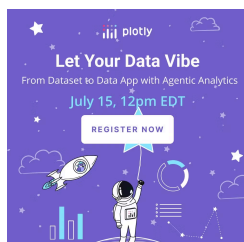
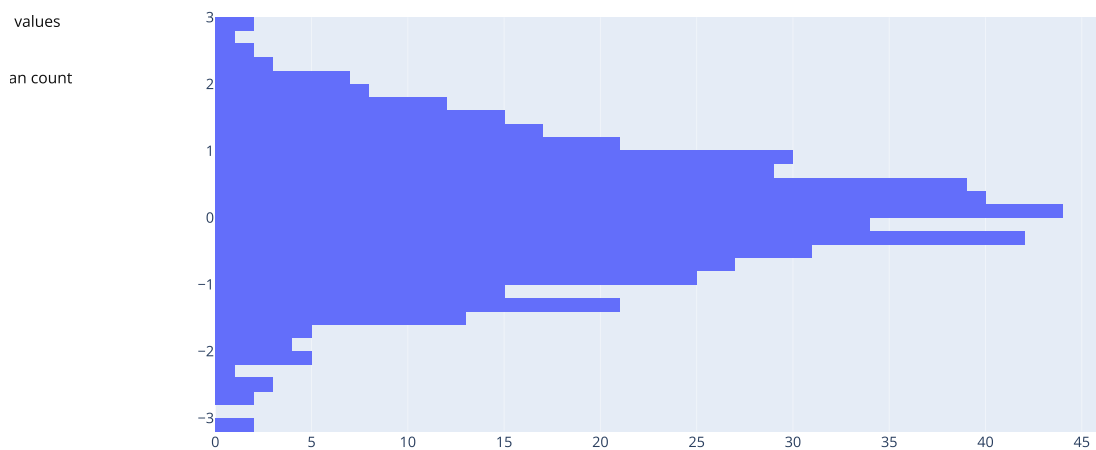
## Horizontal Histogram

```
import plotly.graph_objects as go

import numpy as np

y = np.random.randn(500)
# Use `y` argument instead of `x` for horizontal histogram

fig = go.Figure(data=[go.Histogram(y=y)])
fig.show()
```



## Overlaid Histogram

```
import plotly.graph_objects as go

import numpy as np

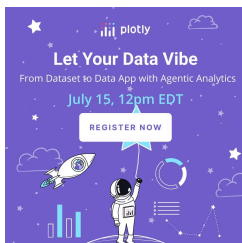
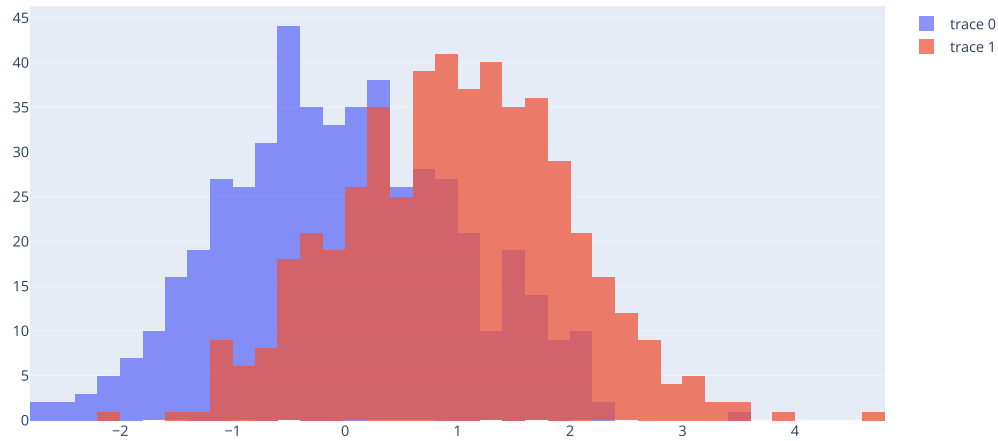
x0 = np.random.randn(500)
# Add 1 to shift the mean of the Gaussian distribution
x1 = np.random.randn(500) + 1

fig = go.Figure()
fig.add_trace(go.Histogram(x=x0))
fig.add_trace(go.Histogram(x=x1))

# Overlay both histograms
fig.update_layout(barmode='overlay')
# Reduce opacity to see both histograms
fig.update_traces(opacity=0.75)
fig.show()
```

values

an count



## Stacked Histograms

```
import plotly.graph_objects as go

import numpy as np

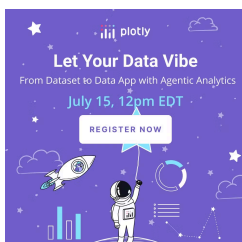
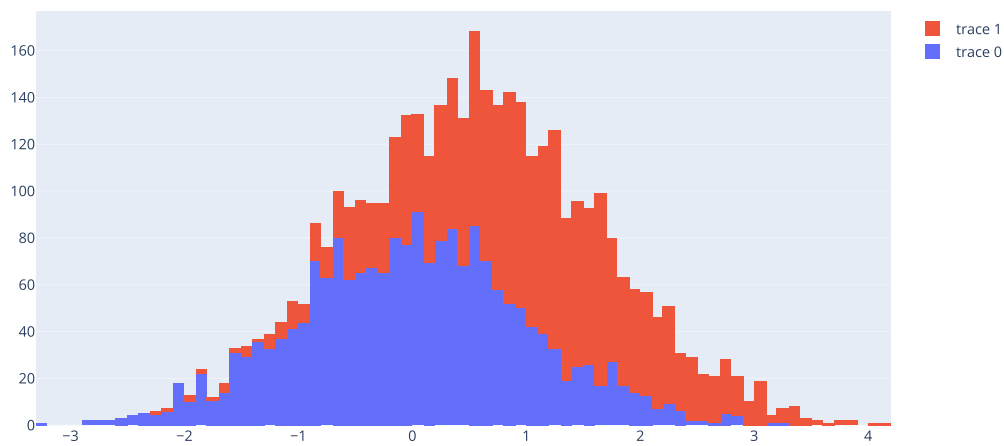
x0 = np.random.randn(2000)
x1 = np.random.randn(2000) + 1

fig = go.Figure()
fig.add_trace(go.Histogram(x=x0))
fig.add_trace(go.Histogram(x=x1))

# The two histograms are drawn on top of another
fig.update_layout(barmode='stack')
fig.show()
```

values

an count



## Styled Histogram

```
import plotly.graph_objects as go

import numpy as np
x0 = np.random.randn(500)
x1 = np.random.randn(500) + 1

fig = go.Figure()
fig.add_trace(go.Histogram(
    x=x0,
    histnorm='percent',
    name='control', # name used in Legend and hover Labels
    xbins=dict( # bins used for histogram
        start=-4.0,
        end=3.0,
        size=0.5
    ),
    marker_color='#EB89B5',
    opacity=0.75
))
fig.add_trace(go.Histogram(
    x=x1,
    histnorm='percent',
    name='experimental',
    xbins=dict(
        start=-3.0,
        end=4,
        size=0.5
    ),
    marker_color='#330C73',
    opacity=0.75
))

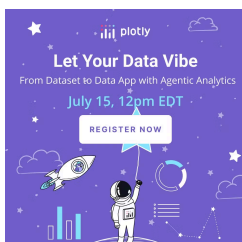
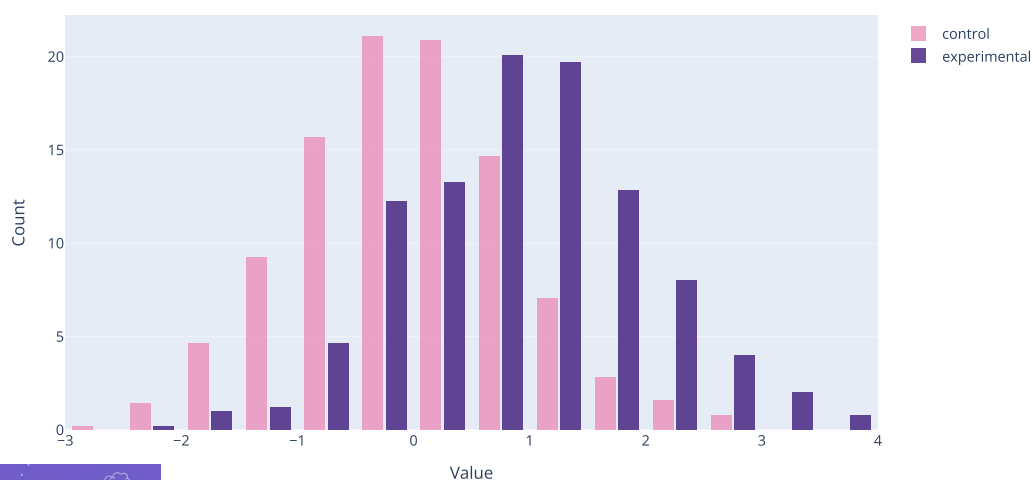
fig.update_layout(
    title_text='Sampled Results', # title of plot
    xaxis_title_text='Value', # xaxis Label
    yaxis_title_text='Count', # yaxis Label
    bargap=0.2, # gap between bars of adjacent Location coordinates
    bargroupgap=0.1 # gap between bars of the same Location coordinates
)

fig.show()
```

values

an count

Sampled Results



## Histogram Bar Text

You can add text to histogram bars using the `texttemplate` argument. In this example we add the x-axis values as text following the format `%{x}`. We also adjust the size of the text using `textfont_size`.

```
import plotly.graph_objects as go

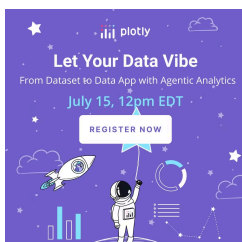
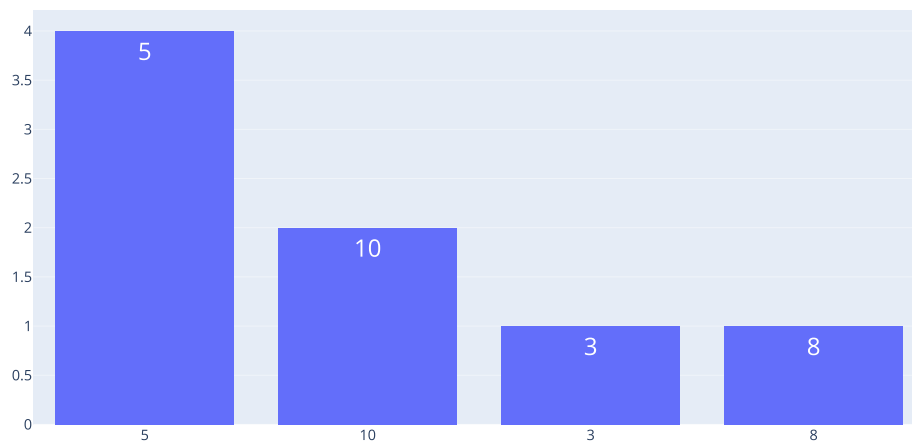
numbers = ["5", "10", "3", "10", "5", "8", "5", "5"]

fig = go.Figure()
fig.add_trace(go.Histogram(x=numbers, name="count", texttemplate="%{x}", textfont_size=20))

fig.show()
```

values

an count





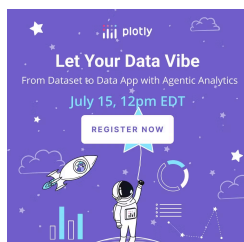
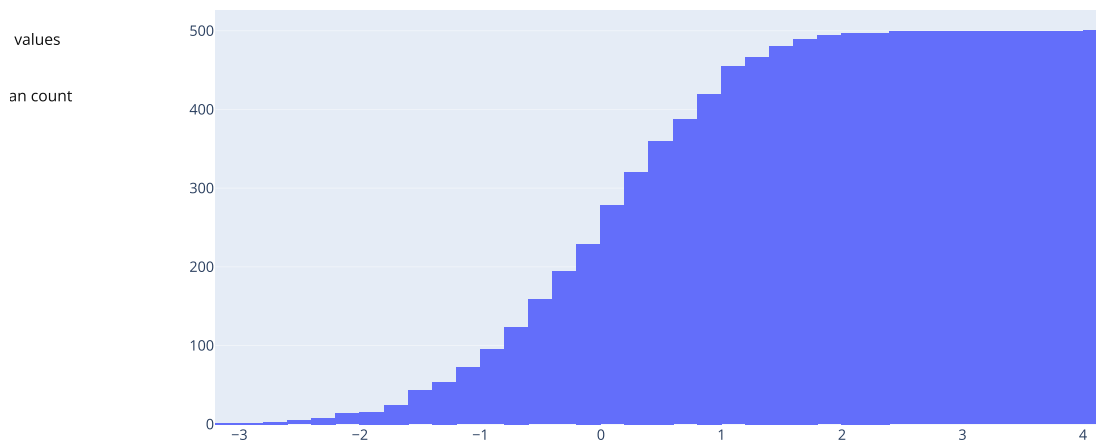
## Cumulative Histogram

```
import plotly.graph_objects as go

import numpy as np

x = np.random.randn(500)
fig = go.Figure(data=[go.Histogram(x=x, cumulative_enabled=True)])

fig.show()
```



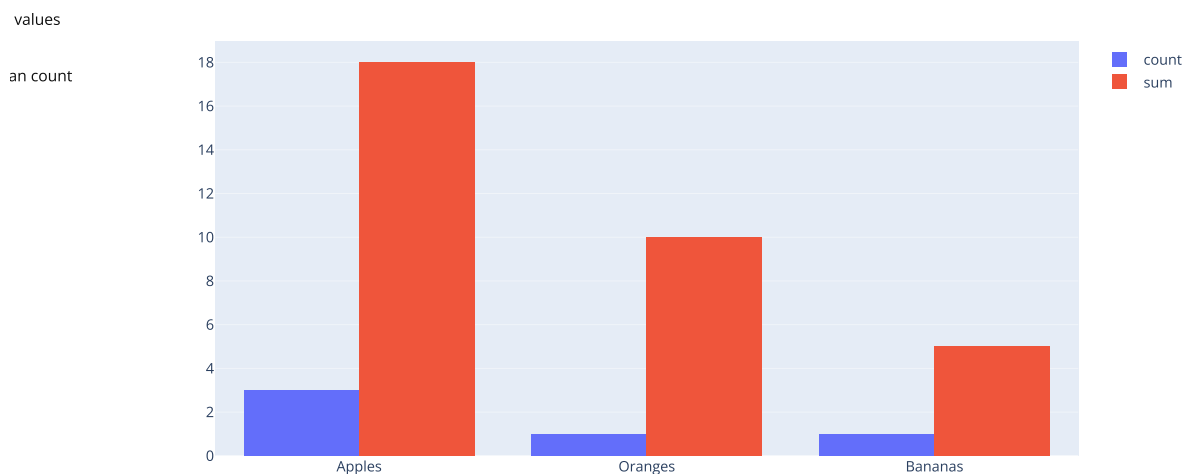
## Specify Aggregation Function

```
import plotly.graph_objects as go

x = ["Apples", "Apples", "Apples", "Oranges", "Bananas"]
y = ["5", "10", "3", "10", "5"]

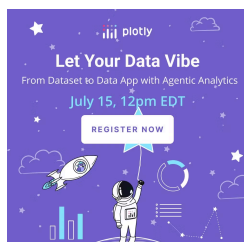
fig = go.Figure()
fig.add_trace(go.Histogram(histfunc="count", y=y, x=x, name="count"))
fig.add_trace(go.Histogram(histfunc="sum", y=y, x=x, name="sum"))

fig.show()
```



## Custom Binning

For custom binning along x-axis, use the attribute `nbinx` (<https://plotly.com/python/reference/histogram/#histogram-nbinx>). Please note that the autobin algorithm will choose a 'nice' round bin size that may result in somewhat fewer than `nbinx` total bins. Alternatively, you can set the exact values for `xbins` (<https://plotly.com/python/reference/histogram/#histogram-xbins>) along with `autobinx = False`.



```

import plotly.graph_objects as go
from plotly.subplots import make_subplots

x = ['1970-01-01', '1970-01-01', '1970-02-01', '1970-04-01', '1970-01-02',
     '1972-01-31', '1970-02-13', '1971-04-19']

fig = make_subplots(rows=3, cols=2)

trace0 = go.Histogram(x=x, nbinsx=4)
trace1 = go.Histogram(x=x, nbinsx=8)
trace2 = go.Histogram(x=x, nbinsx=10)
trace3 = go.Histogram(x=x,
                      xbins=dict(
                          start='1969-11-15',
                          end='1972-03-31',
                          size='M18'), # M18 stands for 18 months
                      autobinx=False
                      )
trace4 = go.Histogram(x=x,
                      xbins=dict(
                          start='1969-11-15',
                          end='1972-03-31',
                          size='M4'), # 4 months bin size
                      autobinx=False
                      )
trace5 = go.Histogram(x=x,
                      xbins=dict(
                          start='1969-11-15',
                          end='1972-03-31',
                          size='M2'), # 2 months
                      autobinx=False
                      )

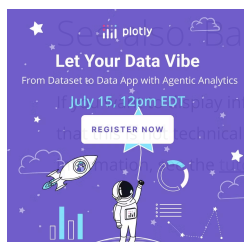
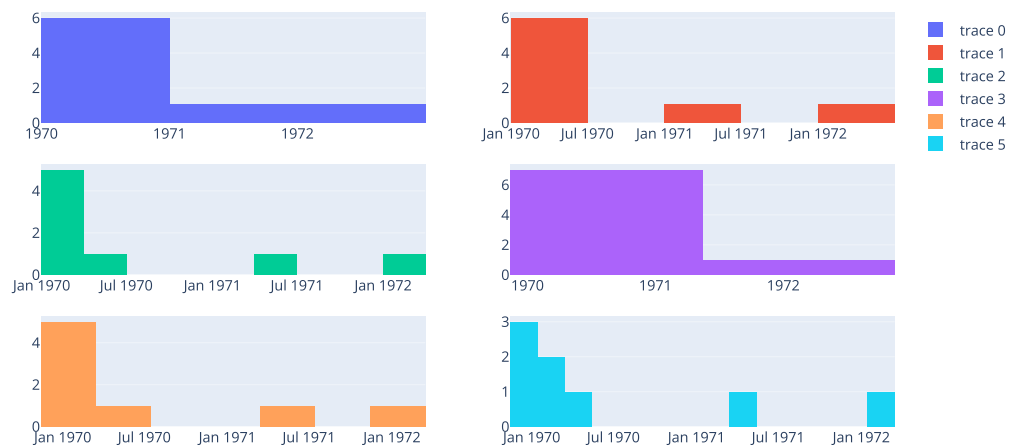
fig.add_trace(trace0, 1, 1)
fig.add_trace(trace1, 1, 2)
fig.add_trace(trace2, 2, 1)
fig.add_trace(trace3, 2, 2)
fig.add_trace(trace4, 3, 1)
fig.add_trace(trace5, 3, 2)

fig.show()

```

values

an count



## Selfico Bar Charts

From Dataset to Data App with Agentic Analytics

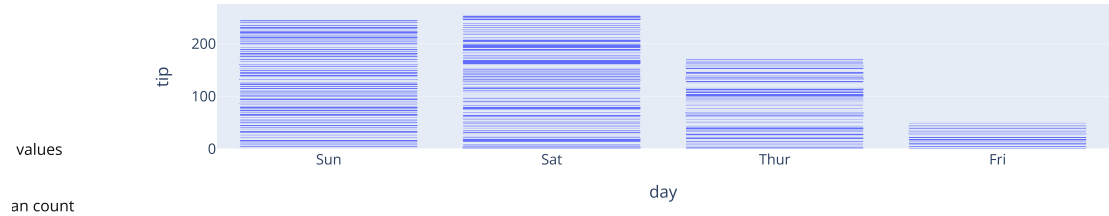
July 15, 12pm EDT

REGISTER NOW

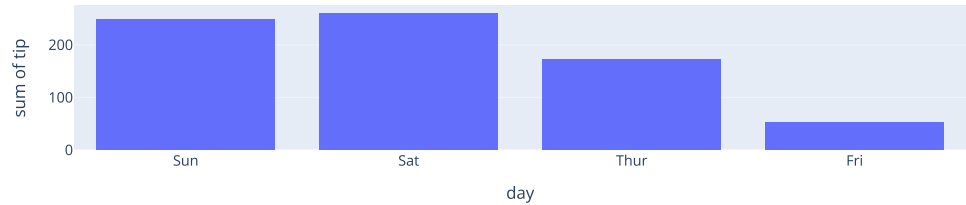
the more information about the individual items within each histogram bar, then create a stacked bar chart with hover information as shown below. Note that this is not technically the histogram chart type, but it will have a similar effect as shown below by comparing the output of `px.histogram` and `px.bar`. For more information, see the [tutorial on bar charts \(\(python/bar-charts/\)\)](https://plotly.com/python/bar-charts/).

```
import plotly.express as px
df = px.data.tips()
fig1 = px.bar(df, x='day', y='tip', height=300,
              title='Stacked Bar Chart - Hover on individual items')
fig2 = px.histogram(df, x='day', y='tip', histfunc='sum', height=300,
                   title='Histogram Chart')
fig1.show()
fig2.show()
```

Stacked Bar Chart - Hover on individual items

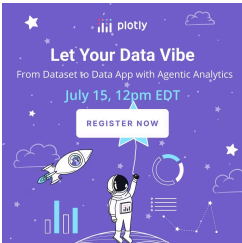


Histogram Chart



# Share bins between histograms

In this example both histograms have a compatible bin settings using `bingroup` (<https://plotly.com/python/reference/histogram/#histogram-bingroup>) attribute. Note that traces on the same subplot, and with the same barmode ("stack", "relative", "group") are forced into the same bingroup, however traces with barmode = "overlay" and on different axes (of the same axis type) can have compatible bin settings. Histogram and `histogram2d` (<https://plotly.com/python/2D-Histogram/>) trace can share the same bingroup.



```
import plotly.graph_objects as go
import numpy as np

fig = go.Figure(go.Histogram(
    x=np.random.randint(7, size=100),
    bingroup=1))

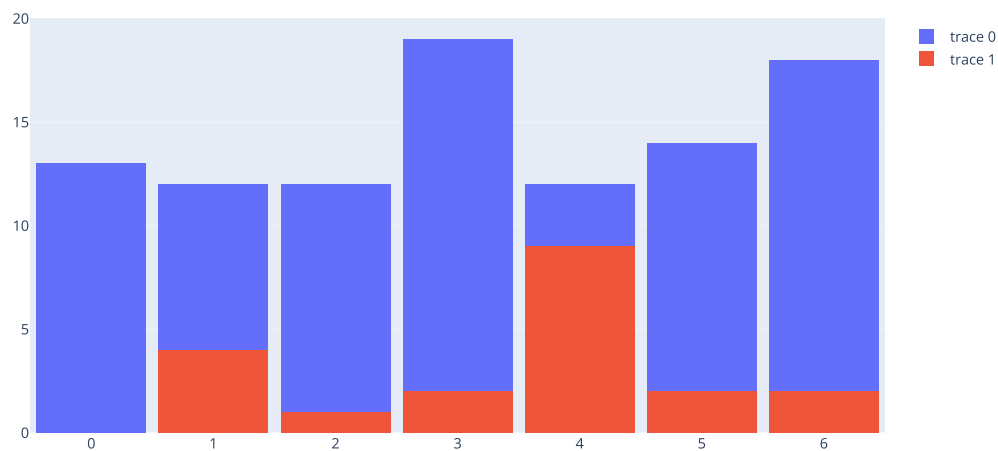
fig.add_trace(go.Histogram(
    x=np.random.randint(7, size=20),
    bingroup=1))

fig.update_layout(
    barmode="overlay",
    bargap=0.1)

fig.show()
```

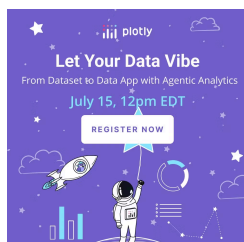
values

an count



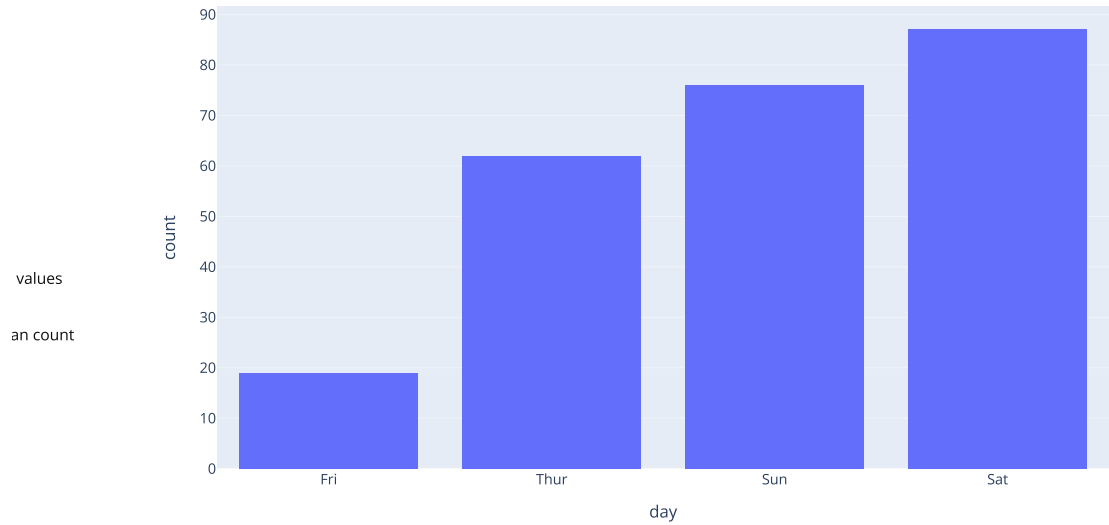
## Sort Histogram by Category Order

Histogram bars can also be sorted based on the ordering logic of the categorical values using the [categoryorder](https://plotly.com/python/reference/layout/xaxis/#layout-xaxis-categoryorder) (<https://plotly.com/python/reference/layout/xaxis/#layout-xaxis-categoryorder>) attribute of the x-axis. Sorting of histogram bars using `categoryorder` also works with multiple traces on the same x-axis. In the following examples, the histogram bars are sorted based on the total numerical values.



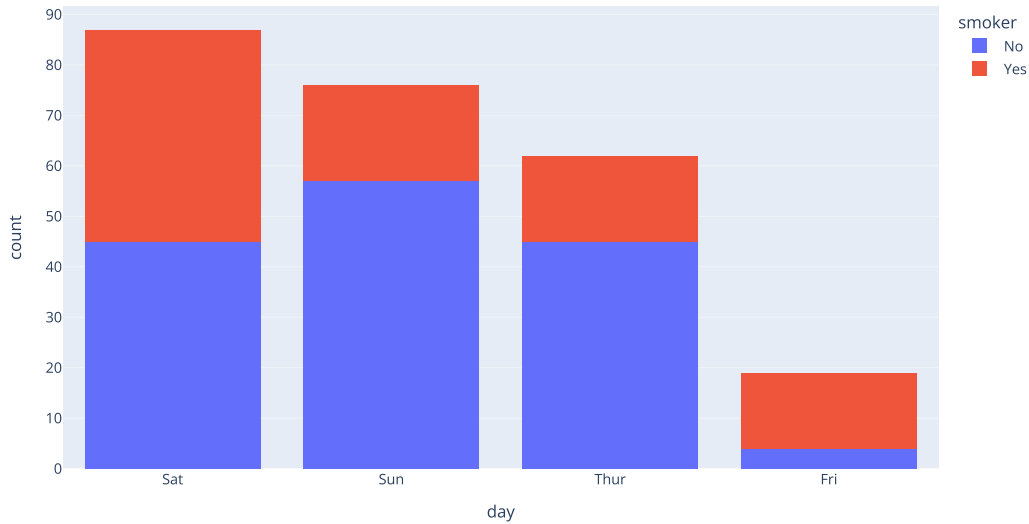
```
import plotly.express as px

df = px.data.tips()
fig = px.histogram(df, x="day").update_xaxes(categoryorder='total ascending')
fig.show()
```

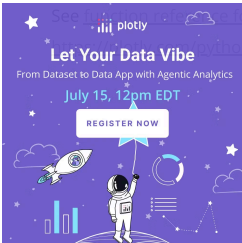


```
import plotly.express as px

df = px.data.tips()
fig = px.histogram(df, x="day", color="smoker").update_xaxes(categoryorder='total descending')
fig.show()
```



Reference



or [px.histogram\(\)](https://plotly.com/python-api-reference/generated/plotly.express.histogram) (<https://plotly.com/python-api-reference/generated/plotly.express.histogram>) or [python/reference/histogram/](https://plotly.com/python/reference/histogram/) (<https://plotly.com/python/reference/histogram/>) for more information and chart attribute options!

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:


values  
an count

```
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
```



# Dash your way to interactive web apps.

No JavaScript required!

GET STARTED NOW


### My First App with Data, Graph, and Controls

pop

lifeExp

gdpPerCap

country	pop	continent	lifeExp	gdpPerCap
Afghanistan	31889923	Asia	43.828	974.5883384
Albania	3600523	Europe	76.423	5937.829525999999
Algeria	33333216	Africa	72.381	6223.367465
Angola	12420476	Africa	42.731	4707.231267
Argentina	40301927	Americas	75.32	12779.37964
Australia	20434176	Oceania	81.235	34435.367439999995
Austria	8199783	Europe	79.829	36126.4927
Bahrain	706573	Asia	75.635	29796.04834
Bangladesh	150448339	Asia	64.062	1701.253792
Belgium	10391226	Europe	79.441	33062.04908
Benin	8878314	Africa	56.728	1441.284873
Bolivia	9119152	Americas	65.554	3821.137884



([https://dash.plotly.com/tutorial?utm\\_medium=graphing\\_libraries&utm\\_content=python\\_footer](https://dash.plotly.com/tutorial?utm_medium=graphing_libraries&utm_content=python_footer))

### JOIN OUR MAILING LIST

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

SUBSCRIBE  
(<https://go.plot.ly/subscription>)

### About Us

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

### Products

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

### Support

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

### Pricing

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

