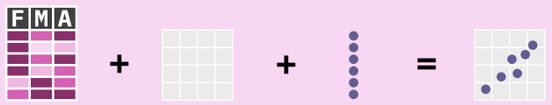
Data visualization with Plotnine *# cheatsheet*

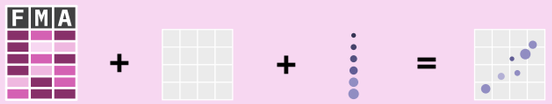
Basics

Plotnine is based on the **grammar of graphics**, the idea that you can build every plot from the same components: a **data frame**, a **coordinate system**, and **geoms**—visual marks that represent data points.



data frame + **coordinate system** + **geoms x, y** = **plot**

To display values, map columns in the data to visual properties of the geom (**aesthetics**) like **size**, **color**, and **x** and **y** locations.



data frame + **coordinate system** + **geoms x, y, size, color** = **plot**

Complete the template below to build a plot:

required
optional

```
from plotnine import *
from plotnine.data import *

(
    ggplot(data=data_frame)
    + geom_function(
        mapping=aes(**mappings),
        stat=stat,
        position=position)
    + coord_function()
    + facet_function()
    + scale_function()
    + theme_function()
    + theme(**settings)
)
```

For example:

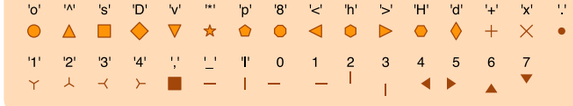
```
p = ggplot(mpg, aes("cty", "hwy")) + geom_point()

p.save("plot.png", width=6, height=4, dpi=200)
```

Aesthetics

Common properties and values

color and **fill**: color name or "#RRGGBB"
linetype: str one of ["solid", "dashed", "dashdot", "dotted"] or tuple of the form (offset, (on, off, on, off, ...))
size: int (in points)
linewidth: int (in points)
shape: character or int



Geoms

Use a geom function to represent data points, use the geom's aesthetic properties to represent variables. Each function returns a layer.

Graphical primitives

```
a = ggplot(economics, aes("date", "unemploy"))
b = ggplot(seals, aes(x="long", y="lat"))
```


a + **geom_blank()**
 # Ensure limits include values across all plots

a + **geom_path**(lineend="butt", linejoin="round", linemitre=1) *# x, y, alpha, color, group, linetype, size*

a + **geom_polygon**(aes(alpha=50)) *# color, fill, group, subgroup, linetype, size*

b + **geom_rect**(aes(xmin="long", ymin="lat", xmax="long+1", ymax="lat+1")) *# xmin, xmax, ymin, ymax, alpha, color, fill, linetype, size*

a + **geom_ribbon**(aes(ymin="unemploy-900", ymax="unemploy+900")) *# x, ymin, ymax, alpha, color, fill, group, linetype, size*

Line segments

Common aesthetics: x, y, alpha, color, linetype, size

b + **geom_abline**(aes(intercept=0, slope=1))
b + **geom_hline**(aes(yintercept="lat"))
b + **geom_vline**(aes(xintercept="long"))

b + **geom_segment**(aes(yend="lat+1", xend="long+1"))
b + **geom_spoke**(aes(angle=1:1155, radius=1))

One continuous variable

```
c = ggplot(mpg, aes(x="hwy")); c2 = ggplot(mpg)
```

c + **geom_area**(stat="bin") *# x, y, alpha, color, fill, linetype, size*

c + **geom_density**(kernel="gaussian") *# x, y, alpha, color, fill, group, linetype, size, weight*

c + **geom_dotplot**() *# x, y, alpha, color, fill*

c + **geom_freqpoly**() *# x, y, alpha, color, group, linetype, size*

c + **geom_histogram**(binwidth=5) *# x, y, alpha, color, fill, linetype, size, weight*

c2 + **geom_qq**(aes(sample="hwy")) *# x, y, alpha, color, fill, linetype, size, weight*

One discrete variable

```
d = ggplot(mpg, aes("fl"))
```

d + **geom_bar**() *# x, alpha, color, fill, linetype, size, weight*

Two continuous variables

```
e = ggplot(mpg, aes(x="cty", y="hwy"))
```

e + **geom_label**(aes(label="cty"), nudge_x=1, nudge_y=1) *# alpha, angle, color, family, fontface, hjust, lineheight, size, vjust*

e + **geom_point**() *# x, y, alpha, color, fill, shape, size, stroke*

e + **geom_quantile**() *# x, y, alpha, color, group, linetype, size, weight*

e + **geom_rug**(sides="bl") *# x, y, alpha, color, linetype, size*

e + **geom_smooth**(method="lm") *# x, y, alpha, color, fill, group, linetype, size, weight*

e + **geom_text**(aes(label="cty"), nudge_x=1, nudge_y=1) *# alpha, angle, color, family, fontface, hjust, lineheight, size, vjust*

One discrete, one continuous variable

```
f = ggplot(mpg, aes(x="class", y="hwy"))
```

f + **geom_col**() *# x, y, alpha, color, fill, group, linetype, size*

f + **geom_boxplot**() *# x, y, lower, middle, upper, ymax, ymin, alpha, color, fill, group, linetype, shape, size, weight*

f + **geom_dotplot**(binaxis="y", stackdir="center") *# x, y, alpha, color, fill, group*

f + **geom_violin**(scale="area") *# x, y, alpha, color, fill, group, linetype, size, weight*

Two discrete variables

```
g = ggplot(diamonds, aes(x="cut", y="color"))
```

g + **geom_count**() *# x, y, alpha, color, fill, shape, size, stroke*

e + **geom_jitter**(height=2, width=2) *# x, y, alpha, color, fill, shape, size*

Three continuous variables

```
import polars as pl

seals = pl.DataFrame(seals).with_columns(z=(pl.col("delta_long")**2 + pl.col("delta_lat")**2).sqrt())
l = ggplot(seals, aes(long, lat))
```

l + **geom_contour**(aes(z="z")) *# x, y, z, alpha, color, group, linetype, size, weight*

l + **geom_contour_filled**(aes(fill="z")) *# x, y, alpha, color, fill, group, linetype, size, subgroup*

Continuous function

```
h = ggplot(economics, aes("date", "unemploy"))
```

h + **geom_area**() *# x, y, alpha, color, fill, linetype, size*

h + **geom_line**() *# x, y, alpha, color, group, linetype, size*

h + **geom_step**(direction="hv") *# x, y, alpha, color, group, linetype, size*

Continuous bivariate distribution

```
i = ggplot(diamonds, aes("carat", "price"))
```

i + **geom_bin_2d**(binwidth=[0.25, 500]) *# x, y, alpha, color, fill, linetype, size, weight*

i + **geom_density_2d**() *# x, y, alpha, color, group, linetype, size*

Visualizing error

```
df = pl.DataFrame({"grp": ["A", "B", "C"], "fit": [4, 5, 6], "se": [1, 2, 1]}); j = ggplot(df, aes("grp", "fit", ymin="fit-se", ymax="fit+se"))
```

j + **geom_crossbar**(fatten=2) *# x, y, ymax, ymin, alpha, color, fill, group, linetype, size*

j + **geom_errorbar**(); j + **geom_errorbarh**() *# x, ymax, ymin, alpha, color, group, linetype, size, width*

j + **geom_linerange**() *# x, ymin, ymax, alpha, color, group, linetype, size*

j + **geom_pointrange**() *# x, y, ymin, ymax, alpha, color, fill, group, linetype, shape, size*

Maps

```
import geopandas as gp
import geodatasets as gd

np = gp.read_file(gd.get_path("geoda.nepal"))
```


ggplot(np) + **geom_map**(aes(fill="population"))

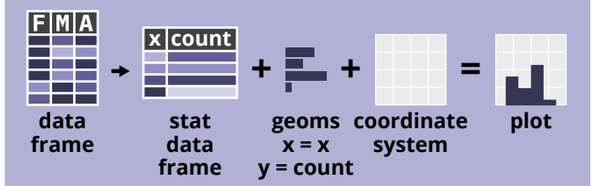
l + **geom_raster**(aes(fill="z"), hjust=0.5, vjust=0.5, interpolate=False) *# x, y, alpha, fill*

l + **geom_tile**(aes(fill = "z")) *# x, y, alpha, color, fill, linetype, size, width*

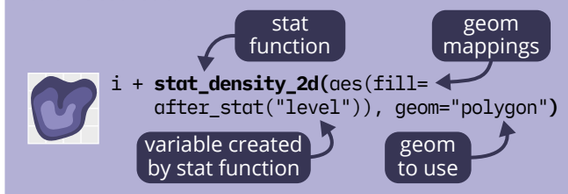
Stats

An alternative way to build a layer.

A stat builds new variables to plot (e.g., count, prop).



Visualize a stat by changing the default stat of a geom function, `geom_bar(stat="count")` or by using a stat function, `stat_count(geom="bar")`, which calls a default geom to make a layer (equivalent to a function). Use `after_stat(name)` syntax to map the stat variable name to an aesthetic.




```
c + stat_bin(binwidth=1, boundary=10)  # x, y | count, ncount, density, ndensity
c + stat_count(width=1)  # x, y | count, prop
c + stat_density(adjust=1, kernel="gaussian")
    # x, y | count, density, scaled
e + stat_bin_2d(bins=30, drop=True)  # x, y, fill | count, density
e + stat_density_2d(contour=True, n=100)  # x, y, color, size | level
e + stat_ellipse(level=0.95, segments=51, type="t")
l + stat_contour(aes(z="z"))  # x, y, z, order | level
l + stat_summary_hex(aes(z="z"), bins=30, fun=max)  # x, y, z, fill | value
l + stat_summary_2d(aes(z="z"), bins=30, fun=mean)  # x, y, z, fill | value
f + stat_boxplot(coef=1.5)  # x, y | lower, middle, upper, width , ymin, ymax
f + stat_ydensity(kernel="gaussian", scale="area")  # x, y | density, scaled, count, n, violinwidth, width
e + stat_ecdf(n=40)  # x, y
e + stat_quantile(quantiles=(0.1, 0.9), formula="y ~ np.log(x)")  # x, y | quantile
e + stat_smooth(method="lm", formula="y ~ x", se=True, level=0.95)  # x, y | se, ymin, ymax
import scipy.stats as stats

ggplot() + lims(x=(-5, 5)) +
    stat_function(fun=stats.norm.pdf, n=20, geom="point")  # x | y
ggplot() + stat_qq(aes(sample=range(100)))  # x, y | sample, theoretical
e + stat_sum()  # x, y, size | n, prop
e + stat_summary(fun_data="mean_cl_boot")
i + stat_summary_bin(fun="mean", geom="bar")
e + stat_identity()
e + stat_unique()
```

Scales

Override default mappings.

Scales map data values to the visual values of an aesthetic. To change a mapping, add a new scale.

```
n = d + geom_bar(aes(fill="fl"))
```

```
n + scale_fill_manual(values=["palegreen", "green", "forestgreen", "darkgreen"],
    limits=["d", "e", "p", "r"],
    breaks=["d", "e", "p", "r"],
    labels=["D", "E", "P", "R"],
    name="fuel")
```

aesthetic to adjust · scale to use · scale-specific arguments · labels to use in legend/axis · title to use in legend/axis · breaks to use in legend/axis

General purpose scales

Use with most aesthetics

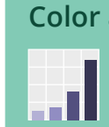
```
scale_*_continuous()  # Map continuous values
scale_*_discrete()  # Map discrete values
scale_*_binned()  # Map continuous values to bins
scale_*_identity()  # Use data values literally
scale_*_manual(values=[])  # Map discrete values
    to manually chosen visual ones
scale_*_date(date_labels="%d/%m"), date_breaks=
    "2 weeks")  # Treat data values as dates
scale_*_datetime()  # Treat values as date times
```

X and Y location scales

Use with x or y aesthetics (x shown here)

```
scale_x_log10()  # Plot on log10 scale
scale_x_reverse()  # Reverse direction of axis
scale_x_sqrt()  # Plot on square root scale
```

Color and fill scales, discrete



```
n + scale_fill_brewer(palette="Blues")
```

```
n + scale_fill_grey(start=0.2, end=0.8, na_value="red")
```

Color and fill scales, continuous

```
o = c + geom_dotplot(aes(fill="x"))
```

```
o + scale_fill_distiller(palette="Blues")
```

```
o + scale_fill_gradient(low="red", high="yellow")
```

```
o + scale_fill_gradient2(low="red", high="blue", mid="white", midpoint=25)
```

```
o + scale_fill_gradientn(colors=["green", "purple", "papayawhip"])
```

Shape and size scales

```
p = e + geom_point(aes(shape="fl", size="cyl"))
```

```
p + scale_shape() + scale_size()
p + scale_shape_manual(values=["o", "^", "s", "D", "v"])  # see page 1 for shapes
```

```
p + scale_radius(range=range(1, 7))
p + scale_size_area(max_size=6)
```

Coordinate Systems

```
r = d + geom_bar()
```

```
r + coord_cartesian(xlim=[0, 5])  # xlim, ylim. The default cartesian coordinate system
```

```
r + coord_fixed(ratio=1/2)  # ratio, xlim, ylim. Cartesian coordinates with fixed aspect ratio between x and y units
```

```
r + coord_flip()  # Flip cartesian coordinates by switching x and y aesthetic mappings.
```

```
r + coord_trans(y="sqrt")  # x, y, xlim, ylim. Transformed cartesian coordinates.
```

Position Adjustments

Position adjustments determine how to arrange geoms that would otherwise occupy the same space.

```
s = ggplot(mpg, aes("fl", fill="drv"))
```

```
s + geom_bar(position="dodge")  # Arrange elements side by side.
```

```
s + geom_bar(position="fill")  # Stack elements on top of one another, normalize height.
```

```
e + geom_point(position="jitter")  # Add random noise to X and Y position of each element to avoid overplotting.
```


```
e + geom_label(position="nudge")  # Nudge labels away from points.
```

```
s + geom_bar(position="stack")  # Stack elements on top of one another.
```

Each position adjustment can be recast as a function with manual width and height arguments:

```
s + geom_bar(position=position_dodge(width=1))
```

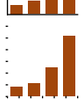
Themes

```
r + theme_bw()  # White background with grid
```

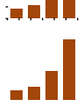
```
r + theme_gray()  # Grey background (default)
```

```
r + theme_dark()  # Dark for contrast
```

```
r + theme_classic()
r + theme_light()
```



```
r + theme_linedraw()
r + theme_minimal()
```



```
r + theme_void()  # Empty theme
```

```
r + theme()  # Customize aspects of the theme such as axis, legend, panel, and facet properties.
r + theme(plot_title_position="plot")
r + theme(panel_background=element_rect(fill="blue"))
```

Faceting

Facets divide a plot into subplots based on the values of one or more discrete variables.

```
t = ggplot(mpg, aes("cty", "hwy")) + geom_point()
```

```
t + facet_grid(cols="fl")
# Facet into columns based on fl
```

```
t + facet_grid("year")
# Facet into rows based on year
```

```
t + facet_grid("year", "fl")
# Facet into both rows and columns
```

```
t + facet_wrap("fl", ncol=4)
# Wrap facets into a rectangular layout
```

Set **scales** to let axis limits vary across facets:

```
t + facet_grid("drv", "fl", scales="free")
```

x and y axis limits adjust to individual facets:

```
"free_x"  # x axis limits adjust
"free_y"  # y axis limits adjust
```

Set **labeller** to adjust facet label:

```
t + facet_grid(cols="fl", labeller="label_both")
```

fl: c	fl: d	fl: e	fl: p	fl: r

Labels and Legends

Use **labs()** to label the elements of your plot.

```
t + labs(x="New x axis label", y="New y axis label",
    title="Add a title above the plot",
    subtitle="Add a subtitle below title",
    caption="Add a caption below plot",
    tag="Add a tag to the plot",
    aes="New aes legend title")
```

```
t + annotate(geom="text", x=8, y=9, label="A")
# Places a geom with manually selected aesthetics
```

```
p + guides(x=guide_axis(n_dodge=2))  # Avoid crowded or overlapping labels with n_dodge or angle
```

```
n + guides(fill="none")  # Set legend type for each aesthetic: colorbar, legend, or none (no legend)
```

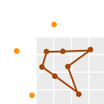
```
n + theme(legend_position="bottom")  # Place legend at "bottom", "top", "left", or "right"
```

```
n + scale_fill_discrete(name="Title", labels=["A", "B", "C", "D", "E"])  # Set legend title and labels with a scale function
```

Zooming

Without clipping (preferred):

```
t + coord_cartesian(xlim=[10, 100], ylim=[0, 20])
```



With clipping (removes unseen data points):

```
t + lims(x=(10, 100), y=(0, 20))  # option 1
t + scale_x_continuous(limits=[10, 100]) +
scale_y_continuous(limits=[0, 20])  # option 2
```

 • info@posit.co • posit.co • Learn more at plotnine.org • HTML cheatsheets at pos.it/cheatsheets • plotnine 0.15 • Updated: 2025-08