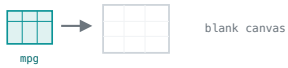
plotnine

A tour of the grammar of graphics in Python, building plots one layer at a time with plotnine the way ggplot2 does in R.

Foundations: the grammar

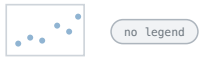
Data + aesthetics + a geom, added with +.

PURPOSE	COMMAND	RESULT
Start a plot from a data frame.	`ggplot(mpg)`	
Map columns to x and y.	`ggplot(mpg,aes(x="displ", y="hwy"))`	
Add a layer that draws (a geom).	`ggplot(mpg, aes(...))+ geom_point()`	
Stack layers with +.	`...+ geom_point()` `  + geom_smooth(method="lm")`	
Save the object, build later.	`p =ggplot(...)+ geom_point()`	
Display in a notebook / script.	`p.show() print(p)`	

until you add a geom there is nothing to draw; the plot is just a Python object you keep extending

Aesthetics: mapping data to channels

aes() maps columns to visual channels; constants live outside it.

PURPOSE	COMMAND	RESULT
Color points by a category.	`aes("displ", "hwy", color="drv")`	
Fill by a category (bars/areas).	`aes("class", fill="drv")`	
Size by a numeric column.	`aes("displ", "hwy", size="cyl")`	
Vary shape by a category.	`aes("displ", "hwy", shape="drv")`	
Set a FIXED look (outside aes).	`geom_point(color="steelblue", alpha=0.5)`	
Use a computed stat as an aes.	`geom_histogram(aes(y=` `  after_stat("density")))`	

mapped (inside aes) builds a legend; a fixed constant look goes outside aes() on the geom

Geoms: the chart vocabulary

Pick the geom that matches your question and data shape.

PURPOSE	COMMAND	RESULT
Scatter points.	`geom_point()`	 relationship
Line / time series.	`geom_line()`	 trend
Bars from counts.	`geom_bar()`	 counts rows
Bars from values you supply.	`geom_col()`	 y = value
Histogram of one variable.	`geom_histogram(bins=20)`	 distribution
Box / violin by group.	`geom_boxplot()geom_violin()`	 spread

geom_bar counts rows for you; geom_col uses a y you already computed (the classic bar gotcha)

Stats and position

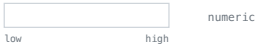
Geoms run a stat under the hood; position settles overlaps.

PURPOSE	COMMAND	RESULT
Fit a trend line + ribbon.	`geom_smooth(method="lm", se=True)`	 fit + band
Smooth without a model.	`geom_smooth(method="lowess")`	 local curve
Side-by-side grouped bars.	`geom_bar(position="dodge")`	 apart
Proportions that sum to 1.	`geom_bar(position="fill")`	 100% 0%
Jitter to reveal overplotting.	`geom_jitter(width=0.2, alpha=0.3)`	
Plot a summary statistic.	`stat_summary(fun_y=np.mean, geom="point")`	 fun_y= (not fun=) mean dot

dodge puts grouped bars side by side; fill makes proportions; jitter nudges coincident points apart

Scales: axes, colors, breaks

Scales decide how data values become pixels, colors, and tick labels.

PURPOSE	COMMAND	RESULT
Set axis limits and ticks.	`scale_x_continuous(limits=(1,7), breaks=...)`	
Log-transform an axis.	`scale_y_log10()`	
Hand-pick group colors.	`scale_color_manual(values=["#1b9e77", ...])`	
Use a ColorBrewer palette.	`scale_color_brewer(palette="Set1")`	
Continuous color gradient.	`scale_color_gradient(low=..., high=...)`	
Rename one axis label.	`scale_x_continuous(name="Displacement")`	

setting limits on a scale drops out-of-range data, unlike a coord_* zoom

Facets and coordinates

Split one plot into a grid; reshape the coordinate space.

PURPOSE	COMMAND	RESULT
One panel per category.	`facet_wrap("~class")`	
Grid of rows x columns.	`facet_grid("drv ~ cyl")`	
Free per-panel axis ranges.	`facet_wrap("~class", scales="free")`	
Flip x and y.	`coord_flip()`	
Lock the aspect ratio.	`coord_fixed(ratio=1)`	
Zoom without dropping data.	`coord_cartesian(xlim=(2, 5))`	

coord_cartesian() zooms the view without discarding data; that is the safe way to zoom

Themes and labels

Titles and theme control the non-data ink.

PURPOSE	COMMAND	RESULT
Add titles and axis labels.	`labs(title="Fuel economy", x=..., y=...)`	
Apply a premade theme.	`theme_minimal()`	
Tweak individual theme parts.	`theme(axis_text_x=element_text(rotation=45))`	
Move or hide the legend.	`theme(legend_position="bottom")`	
Set a default theme for all plots.	`theme_set(theme_bw())`	
Set the figure size (inches).	`theme(figure_size=(8, 5))`	

labs() fills titles and labels; the theme controls fonts, gridlines, background, legend, and size

Save and export

Write the figure to disk at the size and resolution you want.

PURPOSE	COMMAND	RESULT
Save with object method.	`p.save("plot.png")`	
Save with the function form.	`ggsave(p, "plot.png")`	
Control size + resolution.	`p.save("plot.png",` `  width=8, height=6, dpi=300)`	
Pick a vector format.	`p.save("plot.svg")` `p.save("plot.pdf")`	
Write into a folder.	`p.save("fig.png", path="figures/")`	
Quiet the save messages.	`p.save("plot.png", verbose=False)`	

p.save() and ggsave(p, ...) are the same method; .svg / .pdf give crisp vector output