Altair

The essentials of Altair, the declarative library where you map data to visual channels and let it draw, layer, and add interaction.

Grammar: Chart, mark, encode

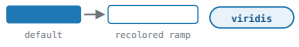
Three chained calls build every Altair chart: data in, geometry, channel mapping.

PURPOSE	COMMAND	RESULT
Wrap a DataFrame in a chart.	`alt.Chart(df)`	
Choose the geometry (mark).	`alt.Chart(df).mark_point()`	
Map columns to x and y.	`alt.Chart(df).mark_point()` `.encode(x="Horsepower", y="Miles_per_Gallon")`	
The full one-liner.	`alt.Chart(df).mark_bar()` `.encode(x="Origin", y="count()")`	 one chained capsule
Render it (notebook auto-displays).	`chart # or chart.show()`	
Inspect the compiled spec.	`chart.to_dict() # or .to_json()`	

`alt.Chart(df).mark_*().encode(...)` is the whole grammar; Altair compiles it to a Vega-Lite JSON spec the browser renders

Encoding Channels & Types

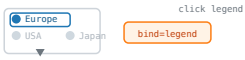
Map columns to x/y/color/size/shape, each tagged with its measurement type.

PURPOSE	COMMAND	RESULT
Encode with explicit types.	`.encode(x="Horsepower:Q", y="MPG:Q",` `color="Origin:N")`	
Add more visual channels.	`.encode(x="Horsepower:Q", y="MPG:Q",` `size="Weight:Q", shape="Origin:N")`	
Use the object form for control.	`x=alt.X("Horsepower:Q", title="HP",` `scale=alt.Scale(zero=False))`	
Aggregate inside the channel.	`.encode(x="mean(Horsepower):Q", y="Origin:N")`	
Bin a quantitative field.	`x=alt.X("Horsepower:Q",bin=True), y="count()"`	
Tune the color scale.	`color=alt.Color("Acceleration:Q",` `scale=alt.Scale(scheme="viridis"))`	

The "col:type" suffix (:Q :N :O :T) drives default scale, axis, and legend; swap the string for alt.X/alt.Color to set title, scale, bin, or aggregate

Interactive

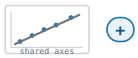
add_params plus a selection makes charts pan, zoom, highlight, and link.

PURPOSE	COMMAND	RESULT
Free pan and zoom.	`chart.interactive()`	
Drag-select a region (brush).	`brush = alt.selection_interval()` `chart.add_params(brush)`	
Color by what is selected.	`color=alt.condition(brush,` `"Origin:N", alt.value("lightgray"))`	
Click-select categories.	`sel = alt.selection_point(` `fields=["Origin"],bind="legend")`	
Linked brushing across charts.	`top:add_params(brush)` `bottom.transform_filter(brush)`	
Drive a chart with a widget.	`alt.binding_select(options=[...],` `name="Origin: ") into selection_point`	

selection_interval drags, selection_point clicks; attach with add_params, then feed it to condition or another chart's transform_filter

Compose

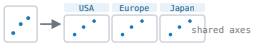
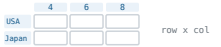
Overlay marks on shared axes, or place separate charts side by side.

PURPOSE	COMMAND	RESULT
Overlay marks (layer).	`points + line ≡ alt.layer(...)`	
Reuse a base spec for layers.	`base= alt.Chart(df).encode(x=..., y=...)` `base.mark_point() +base.mark_line()`	
Side by side (horizontal).	`chartA | chartB ≡ hconcat`	
Stacked (vertical).	`chartA & chartB ≡ vconcat`	
Grid of charts.	`alt.concat(c1, c2, c3, c4,columns=2)`	
Add a shared title.	`(points + line)` `.properties(title="HP vs MPG")`	

+ / alt.layer overlays on shared axes; | & concat place independent charts; .properties(title=) titles the whole composition

Facet (small multiples)

Split one chart into a grid of subplots, one per category, on shared scales.

PURPOSE	COMMAND	RESULT
Facet into wrapped columns.	`chart.facet(facet="Origin:N", columns=3)`	
Facet as an encoding channel.	`.encode(x="x:Q", y="y:Q", facet=alt.Facet("Origin:N", columns=2))`	
Grid by row and column.	`.encode(x="x:Q", y="y:Q", row="Origin:N", column="Cylinders:O")`	
Independent y-scales per facet.	`facet(...).resolve_scale(y="independent")`	
Repeat a chart over fields.	`.encode(x=alt.X(alt.repeat("column"))) .repeat(column=["Horsepower", "Weight"])`	

facet / row / column split one chart into a shared-scale grid; resolve_scale frees an axis; repeat stamps a chart across fields

Transform

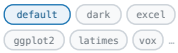
Reshape data inside the spec so Vega does the work, not pandas.

PURPOSE	COMMAND	RESULT
Aggregate into new fields.	`.transform_aggregate(mean_hp= "mean(Horsepower)", groupby=["Origin"])`	
Filter rows by a predicate.	`.transform_filter(alt.datum.Horsepower > 100)`	
Compute a derived column.	`.transform_calculate(kw= "datum.Horsepower * 0.7457")`	
Bin into a named field.	`.transform_bin("hp_bin", "Horsepower", bin=alt.Bin(maxbins=20))`	
Running / windowed total.	`.transform_window(cum="sum(Acceleration)", sort=[{"field": "Year"}])`	
Pivot long to wide.	`.transform_pivot("Origin", value="count", groupby=["Year"])`	

transform_* reshapes data at render time inside the Vega-Lite spec, keeping the chart self-contained and the source table tidy

Theme, Configure & Size

Switch the global look, override config, and set chart width and height.

PURPOSE	COMMAND	RESULT
See the active theme.	`alt.theme.active`	
Switch the global theme.	`alt.theme.enable("dark")`	
List available themes.	`alt.theme.names()`	
Register a custom theme.	`@alt.theme.register("mytheme", enable=True)`	
Set size and title.	`chart.properties(width=400, height=300, title="HP vs MPG")`	
Tweak a single config group.	`chart.configure_axis( grid=False, labelFontSize=12)`	

alt.theme.enable(...) sets the global look for the session; .properties(...) and .configure_*(...) override one chart only

Save & Export

Write the chart to interactive HTML or static PNG/SVG/PDF; share the spec as JSON.

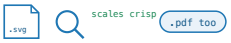
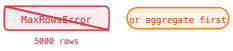
PURPOSE	COMMAND	RESULT
Save interactive HTML.	`chart.save("chart.html")`	
Save a static PNG.	`chart.save("chart.png")`	
Save vector SVG or PDF.	`chart.save("chart.svg")`	
Set resolution / scale factor.	`chart.save("chart.png",scale_factor=2.0)`	
Export just the spec.	`chart.save("chart.json")`	
Lift the row limit for big data.	`alt.data_transformers.disable_max_rows()`	

chart.save("name.ext") writes by extension: .html is interactive, .png/.svg/.pdf are static via vl-convert, .json is the portable spec