Statistical data visualizations (interactive)

BIOF 440

Abhijit Dasgupta

Why dynamic/interactive data visualizations

- Traditional data visualizations have been meant for print
  - Journals
  - Magazines
  - Reports
  - Billboards

Why dynamic/interactive data visualizations

- Media for data reporting has changed to the web
  - websites
  - blogs
  - digital paper
  - electronic billboards
- The web is fundamentally an interactive medium
  - Point and click
  - Explore through links
  - Movement

Why dynamic/interactive data visualizations

- In 1984, Apple introduced a personal computer operating via graphical interaction and display
- This innovation led to widespread availability of personal affordable computers on which we can view and interact with graphics
- Ability to create data visualizations became easy, but was commonly done poorly

Stephen Few, *Data Visualization: Past, Present and Future*, 2007

Visual analytics

- The science of analytical reasoning facilitated by interactive visual interfaces (Thomas & Cook, 2005)
- Helps with visualizing and analyzing data
  - Summarize data
  - Dig deeper
  - Provide analytic insights, cause-and-effect
  - Compressing information from large corpuses of data into small spaces

Visual analytics

- Represent data graphically
- Interact with those visual representations
  - change the nature of the display,
  - filter out what's not relevant,
  - drill into lower levels of detail,
  - highlight subsets of data across multiple graphs simultaneously.

Tableau, *Why Visual Analytics?*

Visual analytics

We will still use principles of good visual encodings

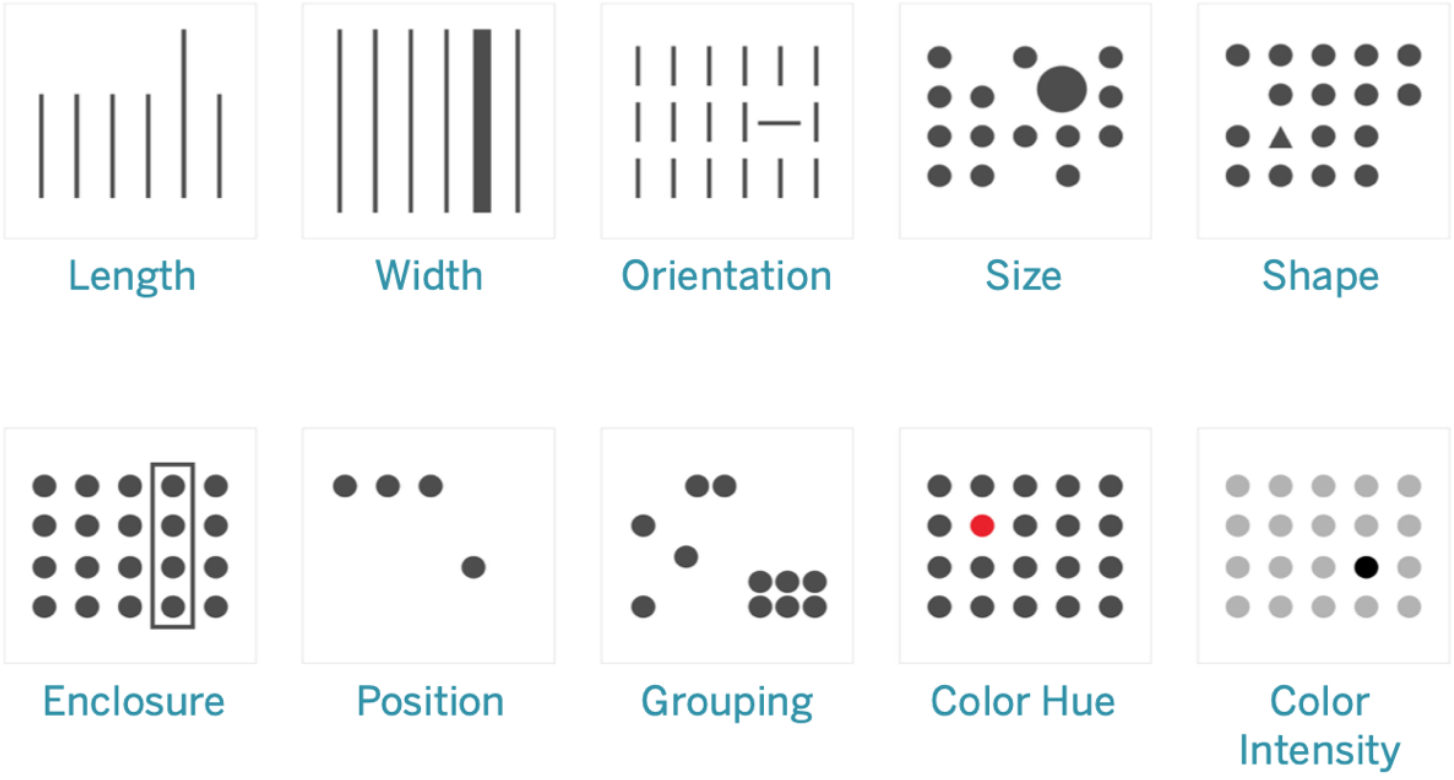
- colors
- shapes/markers
- ink ratio
- size

We can add

- tooltips
- on/off mechanisms
- facets

Visual analytics

As a reminder, we'll still use the usual geometries to encode data



Visual analytics

Kinds of interactions

1. Scroll and pan
2. Zoom
3. Open and close
4. Sort and re-arrange
5. Search and filter

Visual analytics

Ben Schneiderman proposed principles for interactive/dynamic graphics

1. Overview first
2. Zoom and filter
3. Details on demand

This can be translated as

1. Give an overview of the data in a single plot
2. Zoom, select, and move around in the space of the plot
3. Add information (meta-data), usually through tooltips

Visual analytics

The treemap is one of the first interactive tools moving from research to business (Ben Schneiderman)

- display up to two different quantitative variables at different levels of a hierarchy

Visual analytics

The Gapminder data, made famous by Hans Rosling, provides an opportunity to show an example of several aspects of interactive data visualization

Visual analytics

The most popular application for dynamic data visualization is Tableau.

However, using Python, or R, or other programming languages to create these visualizations allows you to unify the data science pipeline that includes description, analysis and visualization

Interactive/dynamic data visualizations in Python

A caveat

In this module we won't talk about animations, i.e., a sequence of graphics that show a flow of data over time. That is dynamic, but not necessarily interactive.

We will look briefly at animations later in the term

Python packages for interactive visualizations

The main packages for interactive visualizations in Python are

1. plotly
2. bokeh
3. altair

In addition, there are several others, including mpld3 (using d3.js), pygal, and holoviews.

Python packages for interactive graphics

We will explore plotly and altair in this class. In week 6 we'll provide resources for bokeh and using holoviews to create graphics using matplotlib, bokeh and plotly

Both plotly and altair have a coding schema (API) that makes the mappings from the data to the visualization explicit, leading to an easier mental model for creating interactive graphics

Setting up Python

```
import altair as alt
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import plotly
import plotly.express as px
import plotly.figure_factory as ff
import plotly.graph_objects as go
import seaborn as sns
```

plotly

plotly

plotly.js is a popular Javascript-based interactive visualization library based on d3.js

The company behind plotly.js developed both Python and R interfaces to create interactive graphics using plotly.js

We'll concentrate here on statistical visualizations using plotly, but the documentation will show many other kinds of graphics that can be generated.

plotly

The Python interface to plotly includes two tracks

- a granular interface using `plotly.graph_objects`
- a higher-level interface using `plotly.express`

Often it's easier to start a graphic with `plotly.express`, and then customize it with elements from `plotly.graph_objects`.

plotly

We'll start with examples using the panguins data, which we will grab from the seaborn package as a pandas `DataFrame`

```
penguins = sns.load_dataset("penguins")
penguins.head()
```

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
0	Adelie	Torgersen	39.1	18.7	181.0	3750.0	Male
1	Adelie	Torgersen	39.5	17.4	186.0	3800.0	Female
2	Adelie	Torgersen	40.3	18.0	195.0	3250.0	Female
3	Adelie	Torgersen	NaN	NaN	NaN	NaN	NaN
4	Adelie	Torgersen	36.7	19.3	193.0	3450.0	Female

plotly

Let's start with a basic scatterplot

```
fig = px.scatter(
    data_frame=penguins,
    x="bill_length_mm",
    y="body_mass_g",
    template="simple_white"
)

show_fig(fig, "week4_files/scatter1.html")
```

plotly

We'll now add *species* encoded as color

```
fig = px.scatter(
    data_frame=penguins,
    x="bill_length_mm",
    y="body_mass_g",
    color="species",  # <<
    template="simple_white",
)
show_fig(fig, "week4_files/scatter2.html")
```

plotly

We can also add marginal plots with additional arguments

```
fig = px.scatter(
    penguins,
    x="bill_length_mm",
    y="body_mass_g",
    color="species",
    marginal_x="box",  # <<
    marginal_y="violin",  # <<
    template="simple_white",
)
show_fig(fig, "week4_files/scatter3.html")
```

plotly

Add regression lines

```
fig = px.scatter(
    data_frame=penguins,
    x="bill_length_mm",
    y="body_mass_g",
    color="species",
    marginal_x="box",
    marginal_y="violin",
    trendline="lowess",  # <<
    template="simple_white",
)
show_fig(fig, "week4_files/scatter4.html")
```

plotly

Add regression lines

```
fig = px.scatter(
    data_frame=penguins,
    x="bill_length_mm",
    y="body_mass_g",
    color="species",
    marginal_x="box",
    marginal_y="violin",
    trendline="ols",  # <<
    template="simple_white",
)
show_fig(fig, filename="week4_files/scatter5.ht
```

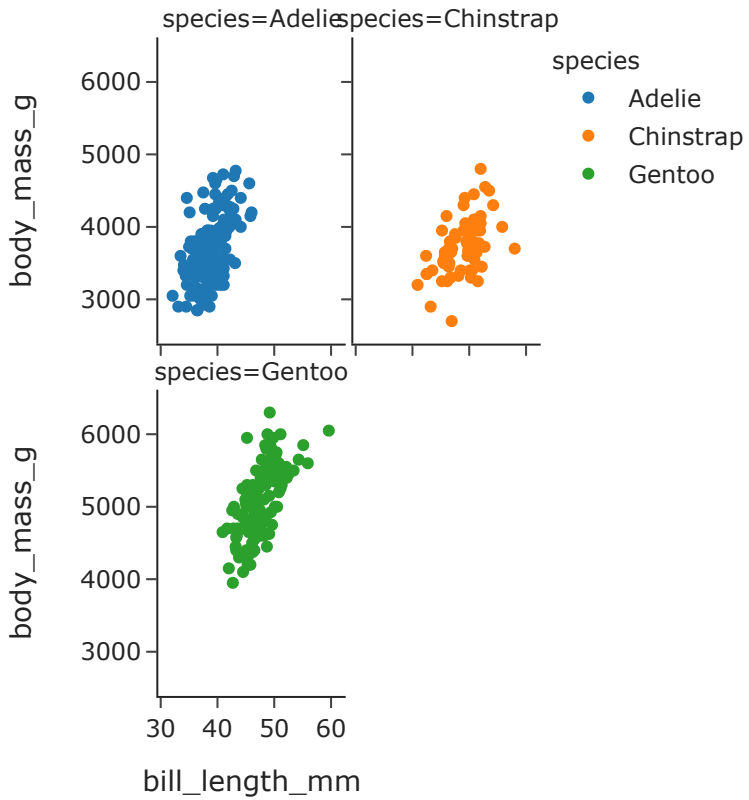
plotly

We can also do trellis graphics pretty easily using plotly

```
fig = px.scatter(
    data_frame=penguins,
    x="bill_length_mm",
    y="body_mass_g",
    color="species",
    facet_col="species",  # <<
    template="simple_white",
)
show_fig(fig, filename="week4_files/scatter6.ht
```

plotly

```
fig = px.scatter(
    data_frame=penguins,
    x="bill_length_mm",
    y="body_mass_g",
    color="species",
    facet_col="species",  # <<
    facet_col_wrap=2,  # <<
    template="simple_white",
)
show_fig(fig, filename="week4_files/scatter7.ht
```



plotly

We can clean up the plot

```
fig = px.scatter(
    data_frame=penguins,
    x="bill_length_mm",
    y="body_mass_g",
    color="species",
    facet_col="species",
    facet_col_wrap=2,  # <<b
    template="simple_white",
    labels={
        "body_mass_g": "Body mass (g)",
        "bill_length_mm": "Bill length (mm)",
        "species": "Species",
    },
)
show_fig(fig, filename="week4_files/scatter8.ht
```

Tooltips

Tooltips

You can add data from column(s) of the DataFrame as tooltips quite easily

```
fig = px.scatter(
    data_frame=penguins,
    x="bill_length_mm",
    y="body_mass_g",
    color="species",
    facet_col="species",
    facet_col_wrap=2,  # <<b
    template="simple_white",
    labels={
        "body_mass_g": "Body mass (g)",
        "bill_length_mm": "Bill length (mm)",
        "species": "Species",
    },
    hover_name="island",  # <<
)
show_fig(fig, filename="week4_files/scatter9.ht
```

Tooltips

Choose which variables go into the tooltip

```
fig = px.scatter(
    data_frame=penguins,
    x="bill_length_mm",
    y="body_mass_g",
    color="species",
    facet_col="species",
    facet_col_wrap=2,  # <<b
    template="simple_white",
    labels={
        "body_mass_g": "Body mass (g)",
        "bill_length_mm": "Bill length (mm)",
        "species": "Species",
    },
    hover_data={
        "island": True,
        "species": False,
        "bill_length_mm": False,
        "body_mass_g": False,
    },  # <<
)
show_fig(fig, filename="week4_files/scatter10.
```

Tooltips

You can change the appearance of the tooltip

```
fig.update_layout(
  hoverlabel={
    "bgcolor": "red",
    "font_family": "Futura"
    })
show_fig(fig,
filename="week4_files/scatter11.html")
```

Tool tips

You can also provide a template for the tooltips

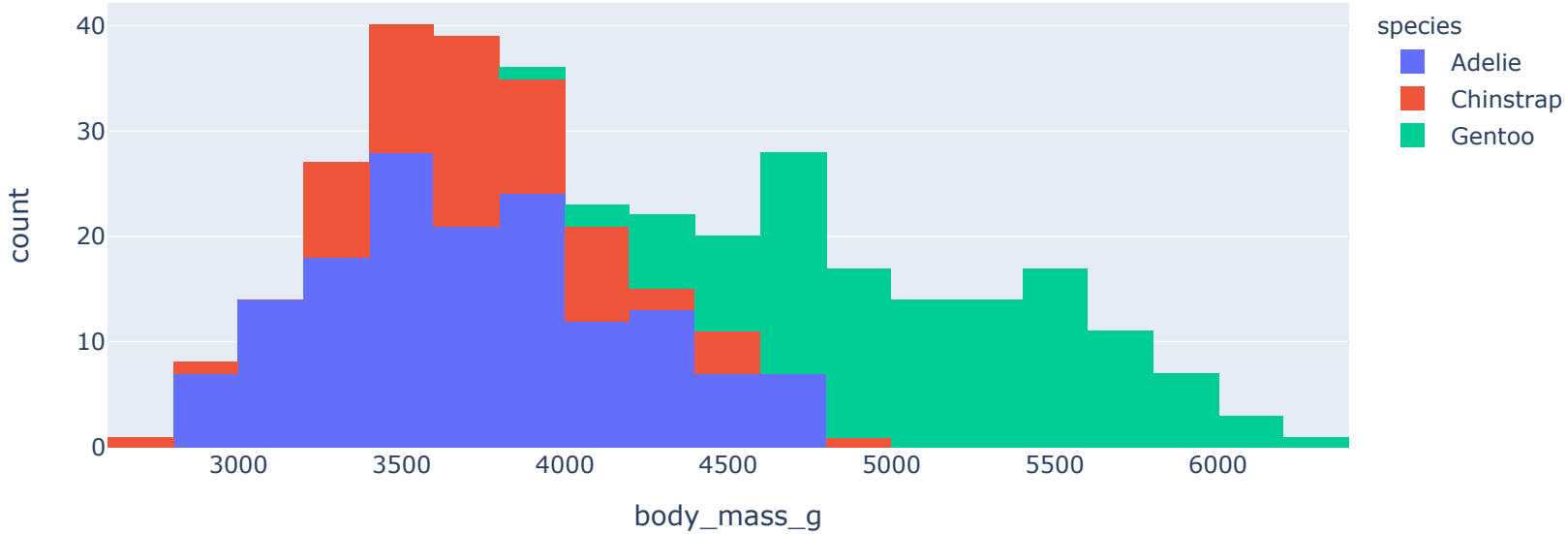
```
import plotly.express as px

df_2007 = (px.data.gapminder()
  .query("year==2007"))

fig = px.scatter(df_2007,
  x="gdpPercap",
  y="lifeExp",
  log_x=True,
  color="continent")

fig.update_traces(
  hovertemplate="GDP: %{x} <br>Life Expectancy
fig.update_traces(
    hovertemplate=None, selector={"name": "Eur
)  # revert to default hover
show_fig(fig, filename="week4_files/scatter12.
```

Univariate plots

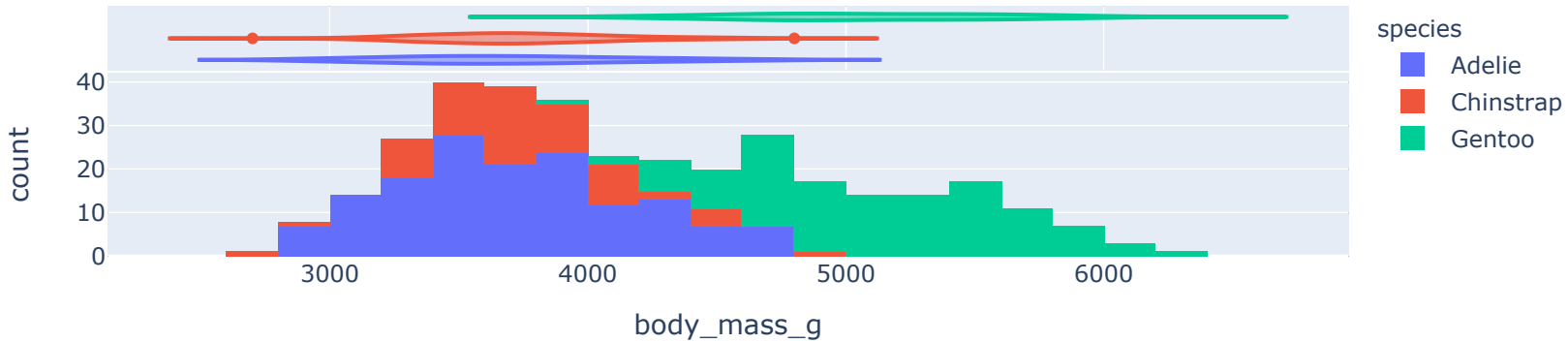
Distributional plots

```
fig = px.histogram(
    data_frame=penguins,
    x="body_mass_g",
)
fig.update_xaxes(title="Body mass (g)")
show_fig(fig, filename="week4_files/hist1.html")
```

```
fig = px.histogram(data_frame=penguins, x="body_mass_g", color="species")
show_fig(fig, filename="week4_files/hist2.html")
```



```
fig = px.histogram(
    data_frame=penguins, x="body_mass_g", color="species", marginal="violin"
)
show_fig(fig, filename="week4_files/hist3.html")
```

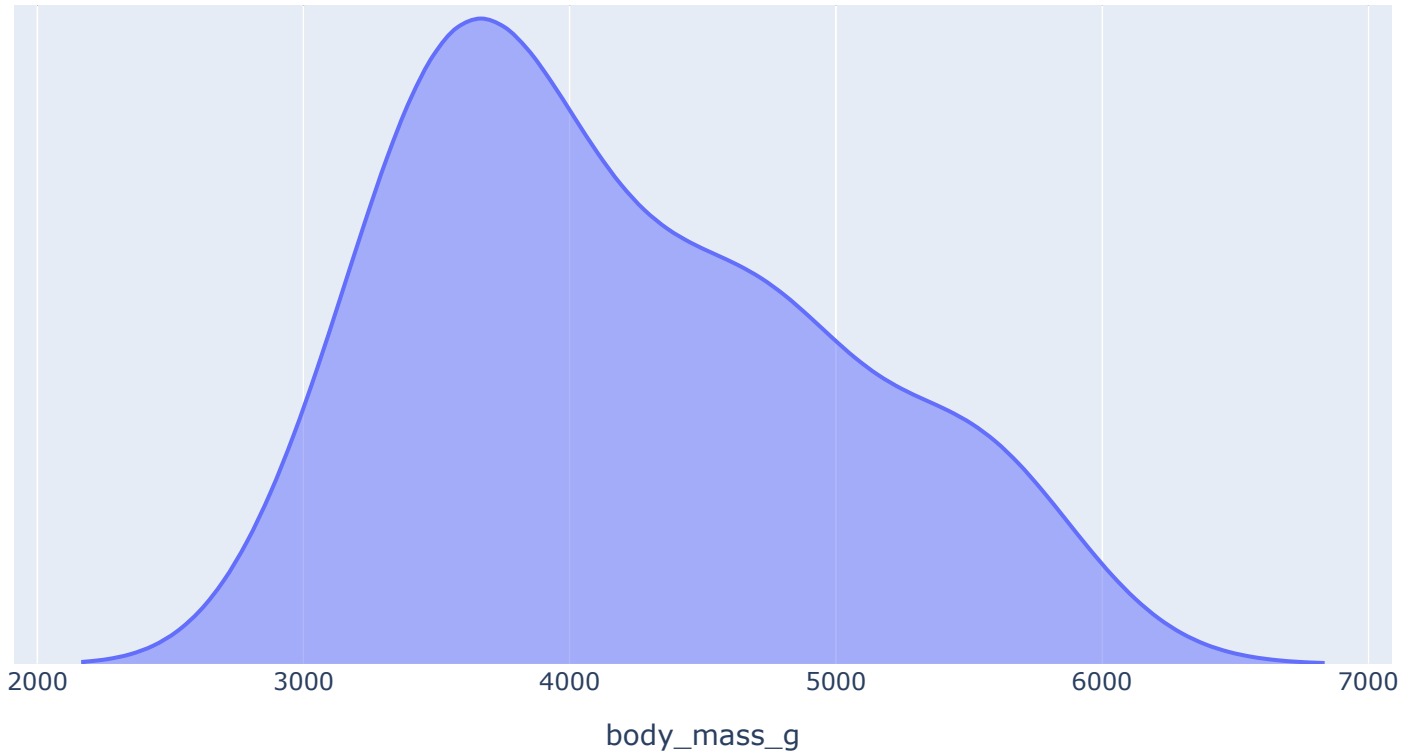


Density plots

We use a trick to create a density plot from a violin plot

```
fig = px.violin(data_frame=penguins, x="body_mass_g")
fig.update_traces(orientation="h", side="positive")  # <<
show_fig(fig, filename="week4_files/density1.html")
```

Density plots

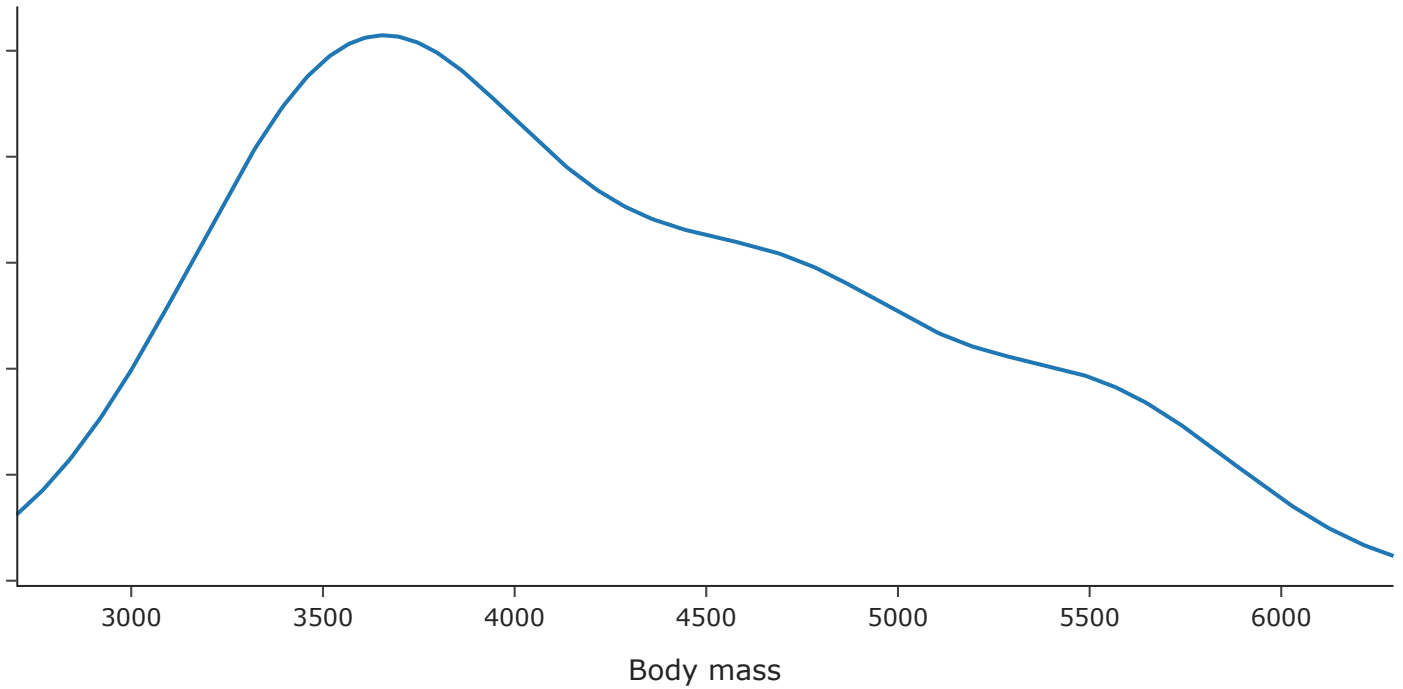


Density plots

The `plotly.express` and `plotly.graphical_object` paradigms don't currently create density plots. There is another function, `figure_factory`, that allows you to create density plots. These `figure_factory` are generally deprecated, but are kept to fill in gaps in other paradigms

```
import plotly.figure_factory as ff

fig = ff.create_distplot(
    [penguins["body_mass_g"].dropna().to_list()],  # Need to get rid of missing data
    group_labels=["Body mass"],
    bin_size=100,
    show_hist=False,
    show_rug=False,
)
fig.update_yaxes(showticklabels=False)
fig.update_xaxes(title="Body mass")
fig.update_layout(showlegend=False, template="simple_white")
show_fig(fig, filename="week4_files/density2.html")
```

Density plots



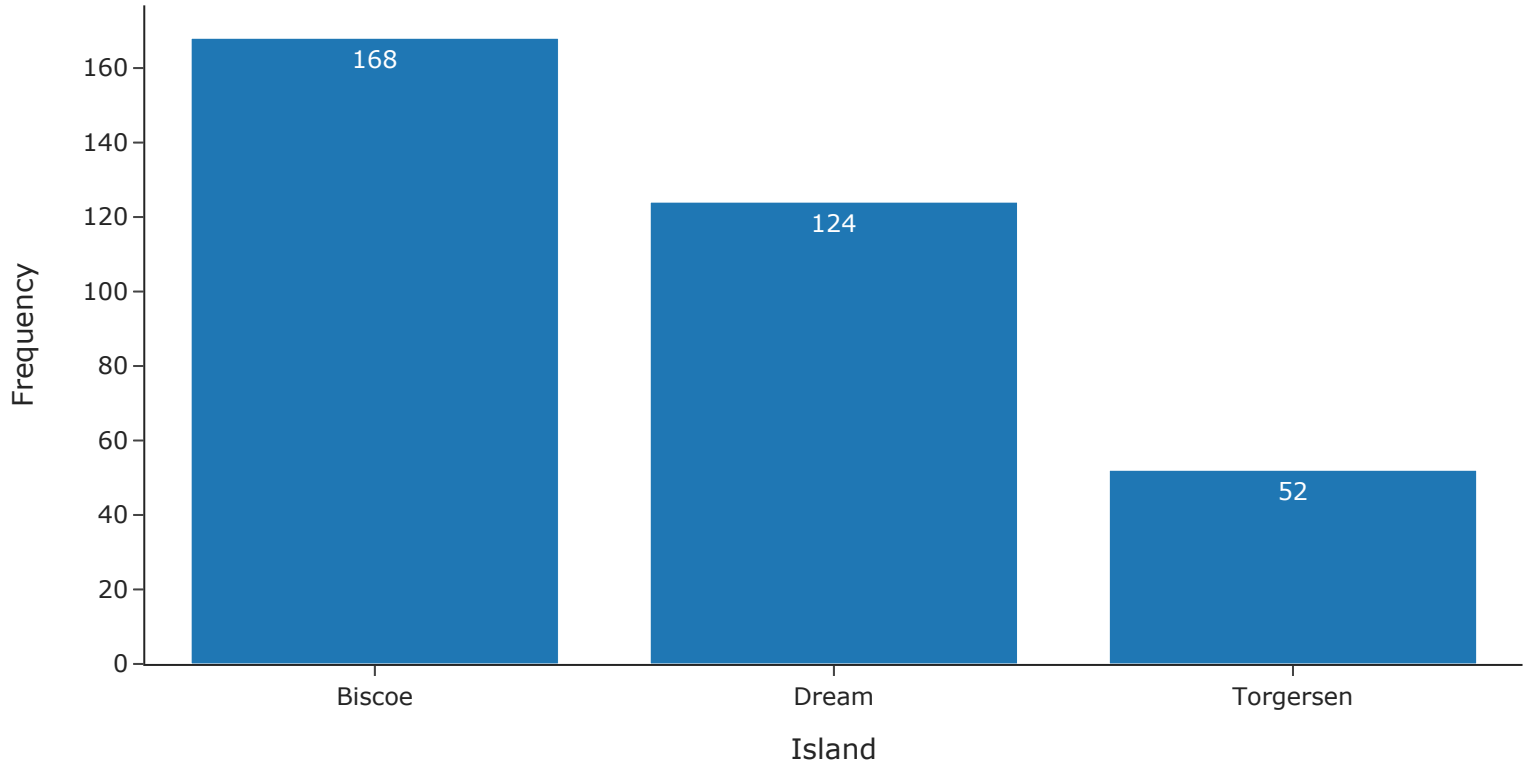
Frequency bar plots

```
d = penguins.island.value_counts().reset_index()

print(d)
```

```
       index  island
0     Biscoe     168
1      Dream     124
2  Torgersen      52
```

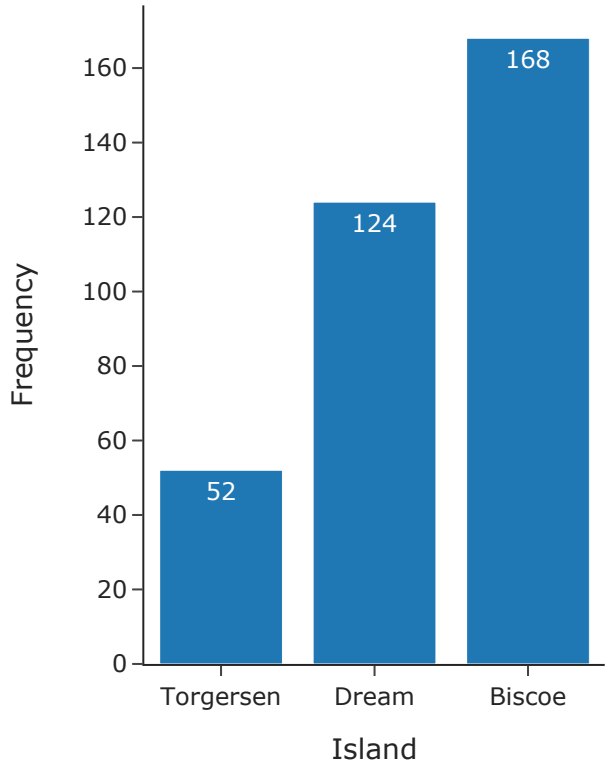
```
fig = px.bar(
    d,
    x="index",
    y="island",
    text="island",  # <<
    labels={"island": "Frequency", "index": "Island"},
    template="simple_white",
)
show_fig(fig, filename="week4_files/bar1.html")
```

Frequency bar plots



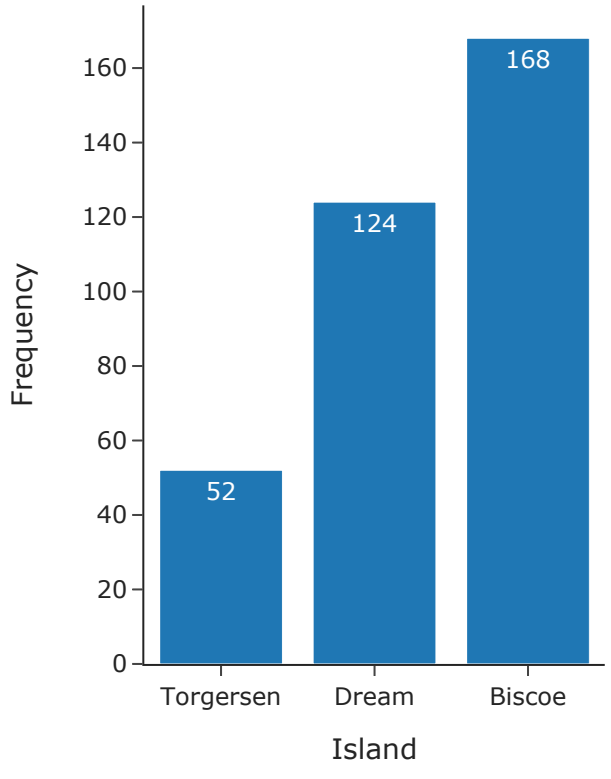
Frequency bar plots

```
d = penguins.island.value_counts().reset_index(

fig = px.bar(
    d,
    x="index",
    y="island",
    text="island",  # <<
    labels={"island": "Frequency",
      "index": "Island"},
    template="simple_white",
).update_layout(
    xaxis={"categoryorder": "category descendi
)
show_fig(fig, filename="week4_files/bar2.html"
```



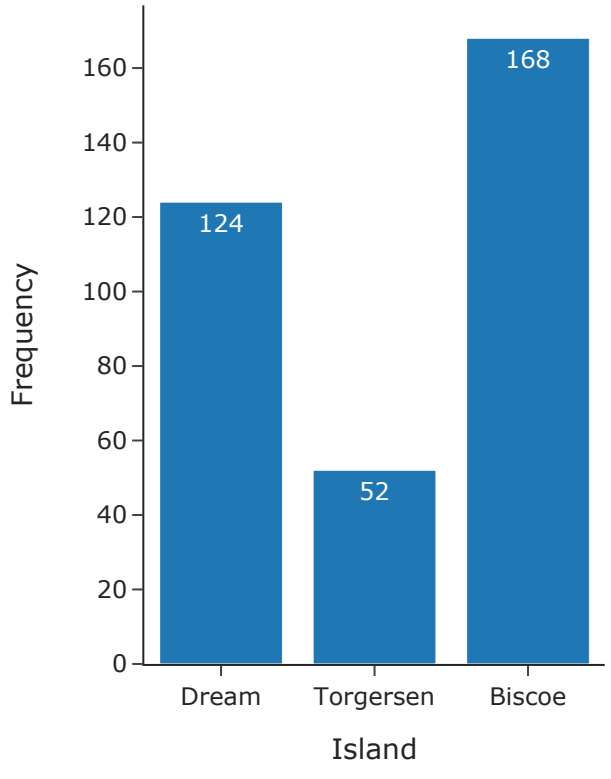
Frequency bar plots

```
d = penguins.island.value_counts().reset_index(

fig = px.bar(
    d,
    x="index",
    y="island",
    text="island",  # <<
    labels={"island": "Frequency",
    "index": "Island"},
    template="simple_white",
).update_layout(
    xaxis={"categoryorder":
      "total ascending"},  # <<  value order
)
show_fig(fig, filename="week4_files/bar3.html"
```



Frequency bar plots

```
d = penguins.island.value_counts().reset_index(

fig = px.bar(
    d,
    x="index",
    y="island",
    text="island",  # <<
    labels={"island": "Frequency", "index": "Is
    template="simple_white",
).update_layout(
    xaxis={
        "categoryorder": "array",
        "categoryarray": ["Dream", "Torgersen"
    },  # Specify the order
)
show_fig(fig, filename="week4_files/bar4.html"
```



Grouped bar charts

Stacked bar charts

```
tips = px.data.tips()

fig = px.bar(data_frame=tips, x="day", y="total_bill", color="sex")
fig.update_layout(showlegend=False)
show_fig(fig, filename="week4_files/bar5.html")
```

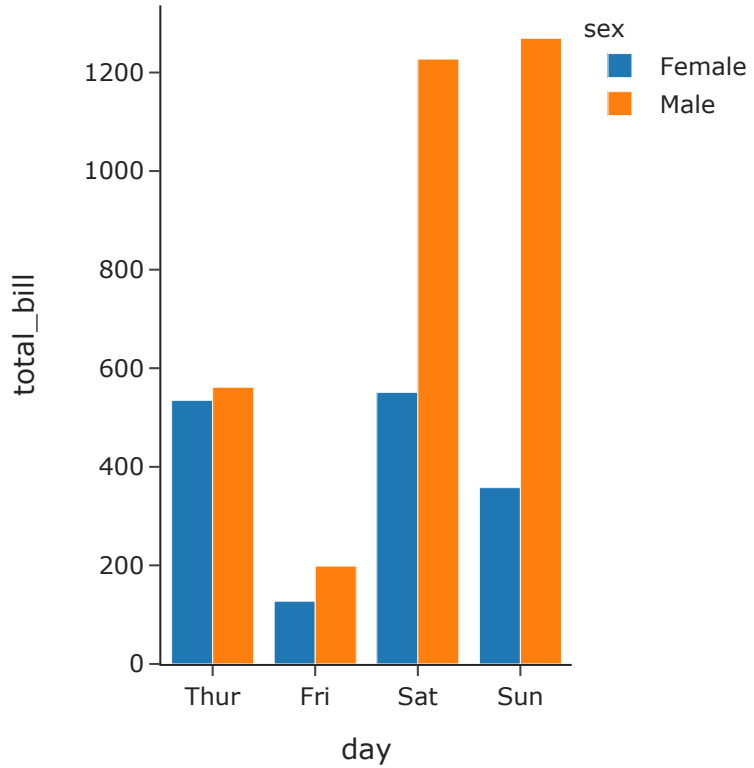
Stacked bar charts

```
tips_summary = tips.groupby(["day", "sex"])["total_bill"].sum().reset_index()
```

```
fig = px.bar(
    data_frame=tips_summary,
    x="day",
    y="total_bill",
    color="sex",
    category_orders={"day": ["Thur", "Fri", "Sa
    template="simple_white",
)
show_fig(fig, filename="week4_files/bar6.html"
```

Grouped bar chart

```
fig = px.bar(
    data_frame=tips_summary,
    x="day",
    y="total_bill",
    color="sex",
    category_orders={"day": ["Thur", "Fri", "Sa
    barmode="group",
    template="simple_white",
)
show_fig(fig, filename="week4_files/bar7.html"
```



Percent bar chart

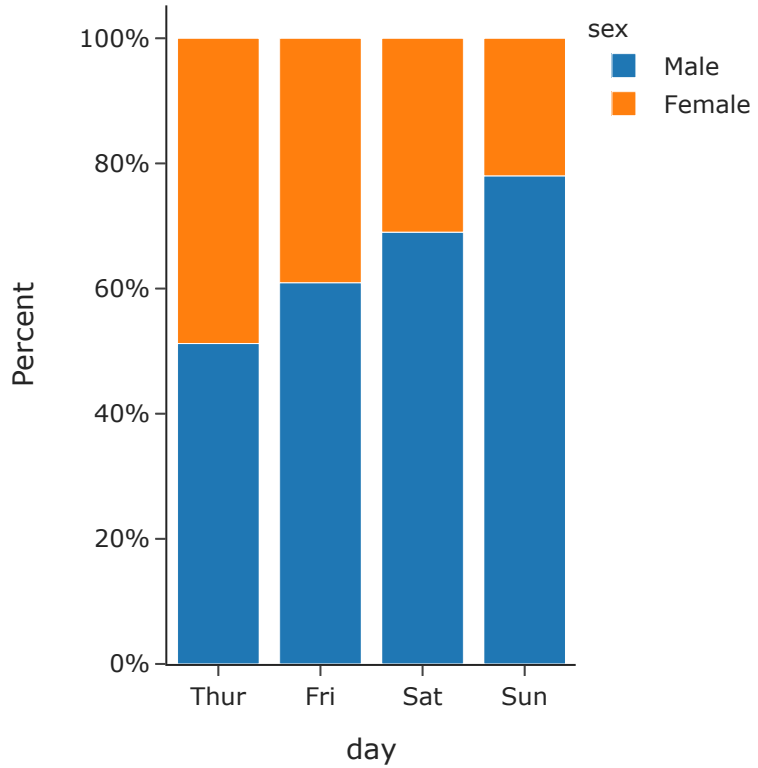
For the percent bar chart you have to compute the percentages first before creating the bar charts.

```
tips_summary["Percent"] = tips_summary.groupby(["day"])["total_bill"].apply(
    lambda x: x / float(x.sum())
)

tips_summary.head()
```

	day	sex	total_bill	Percent
0	Fri	Female	127.31	0.390665
1	Fri	Male	198.57	0.609335
2	Sat	Female	551.05	0.309857
3	Sat	Male	1227.35	0.690143
4	Sun	Female	357.70	0.219831

Percent bar chart

```
fig = px.bar(
    data_frame=tips_summary,
    x="day",
    y="Percent",
    color="sex",
    template="simple_white",
    category_orders={"day": ["Thur", "Fri", "Sa
      "sex": ["Male", "Female"]},
)
fig.update_layout(yaxis_tickformat="%")
show_fig(fig, filename="week4_files/bar8.html"
```

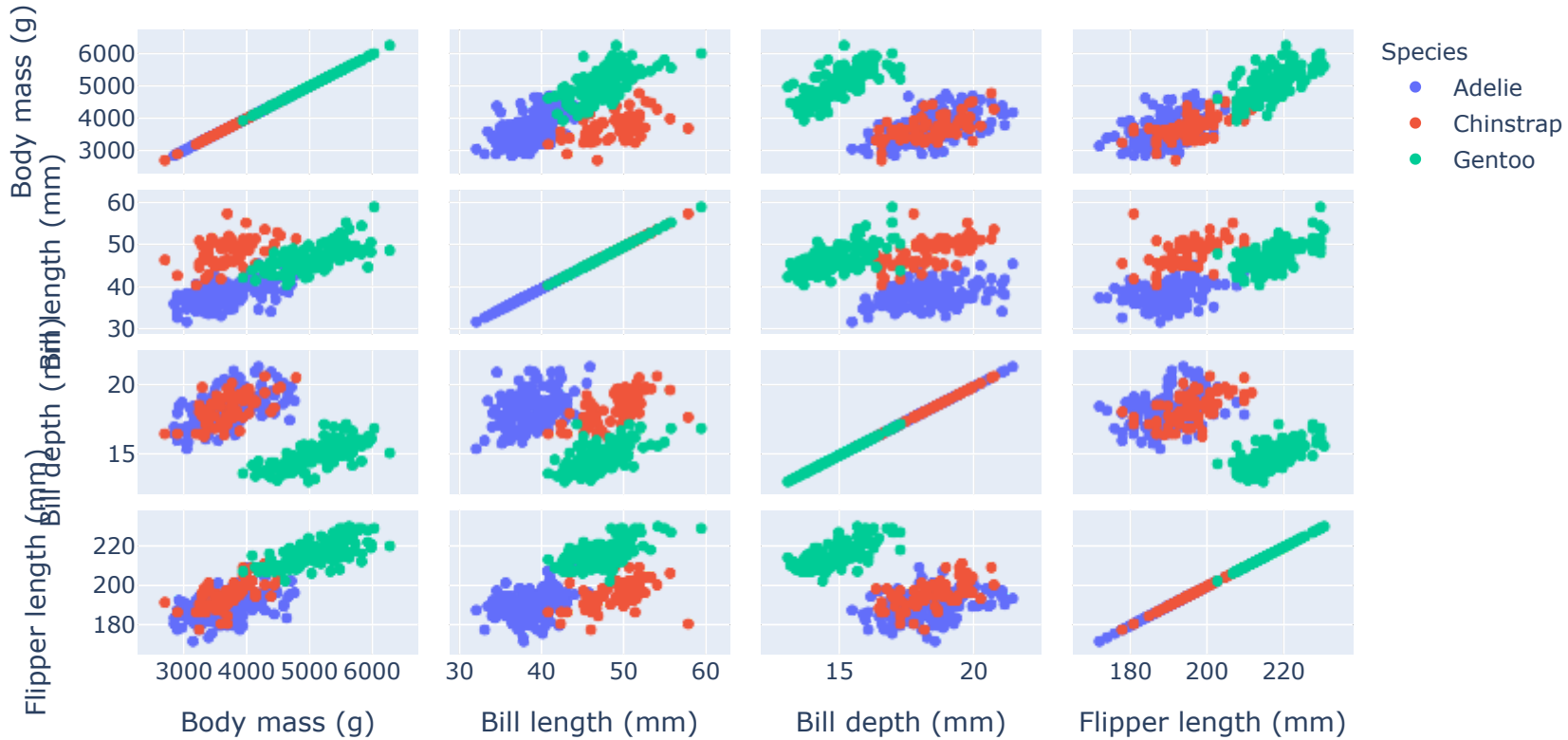


plotly

Scatterplot matrices

```
penguins_labels = {
    "body_mass_g": "Body mass (g)",
    "bill_length_mm": "Bill length (mm)",
    "bill_depth_mm": "Bill depth (mm)",
    "flipper_length_mm": "Flipper length (mm)",
    "species": "Species",
    "island": "Island",
}

fig = px.scatter_matrix(
    data_frame=penguins,
    dimensions=["body_mass_g", "bill_length_mm", "bill_depth_mm", "flipper_length_mm"],
    color="species",
    labels=penguins_labels,
)
show_fig(fig, filename="week4_files/splom.html")
```

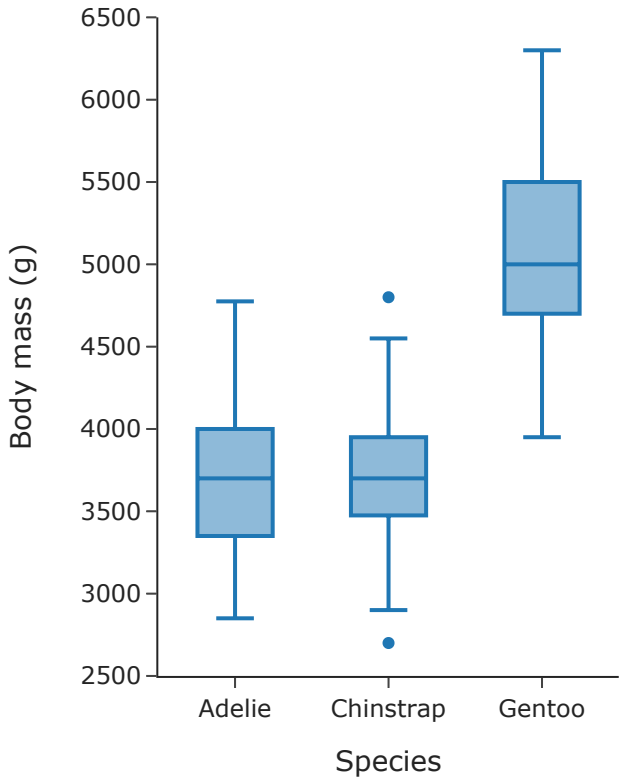
Scatterplot matrices



Continuous vs categorical

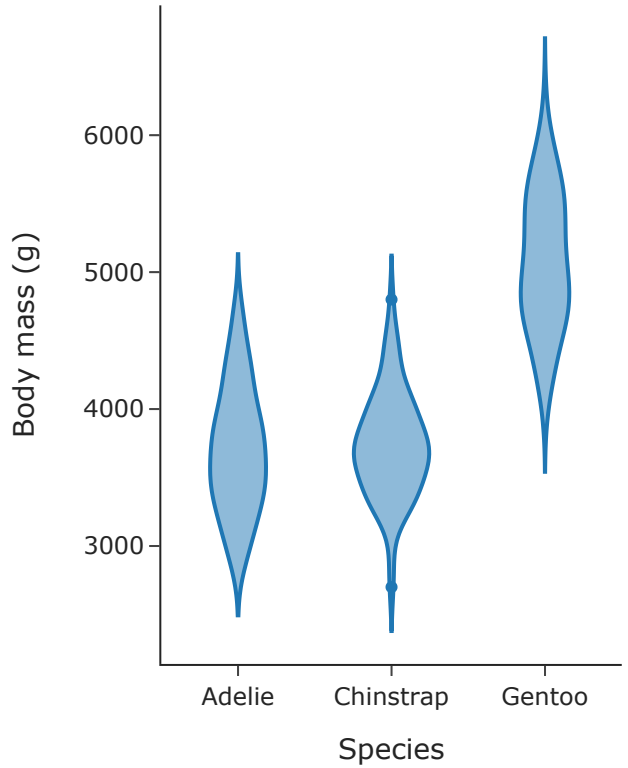
Boxplots

```
fig = px.box(
    data_frame=penguins,
    x="species",
    y="body_mass_g",
    template="simple_white",
    labels=penguins_labels,
)
show_fig(fig, filename="week4_files/box1.html"
```



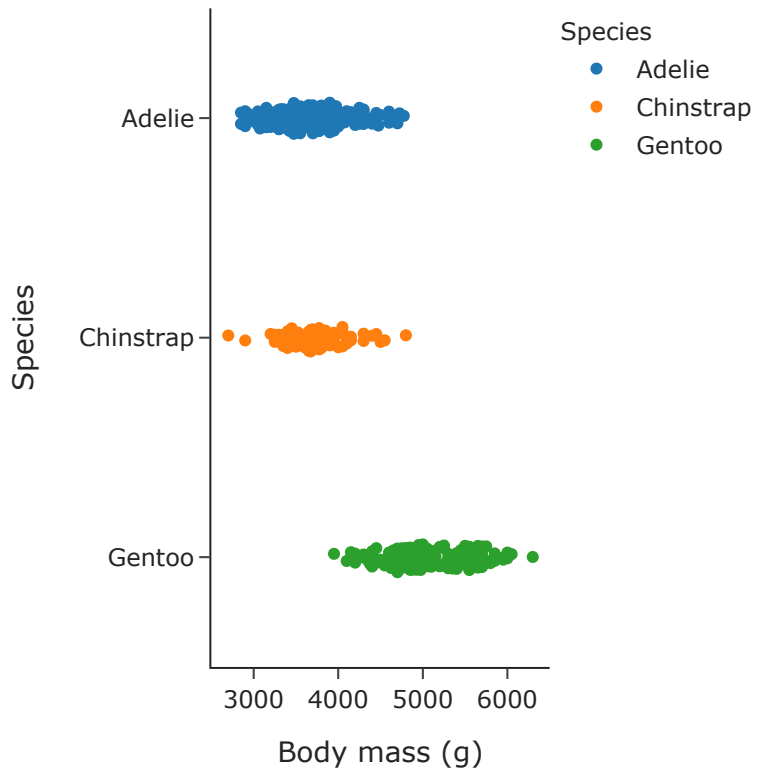
Violin plot

```
fig = px.violin(
    data_frame=penguins,
    x="species",
    y="body_mass_g",
    template="simple_white",
    labels=penguins_labels,
)
show_fig(fig, filename="week4_files/violin1.htm
```



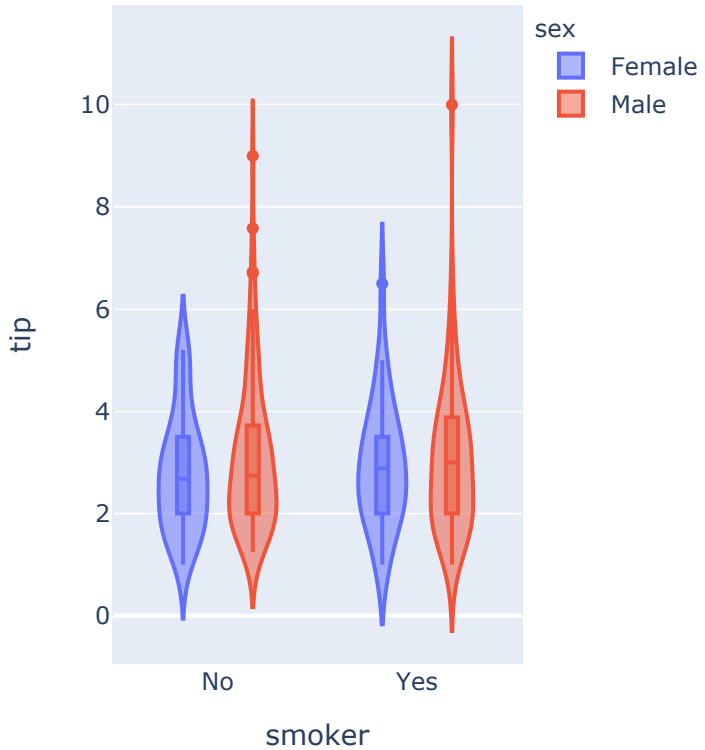
Strip plot

```
fig = px.strip(
    data_frame=penguins,
    x="body_mass_g",
    y="species",
    color="species",
    template="simple_white",
    labels=penguins_labels,
)
show_fig(fig, filename="week4_files/strip1.htm
```



Grouped violin plots

```
fig = px.violin(data_frame=tips,
  y="tip",
  x="smoker",
  color="sex",
  box=True)
show_fig(fig, filename="week4_files/violin2.htm
```



Parallel coordinates plot

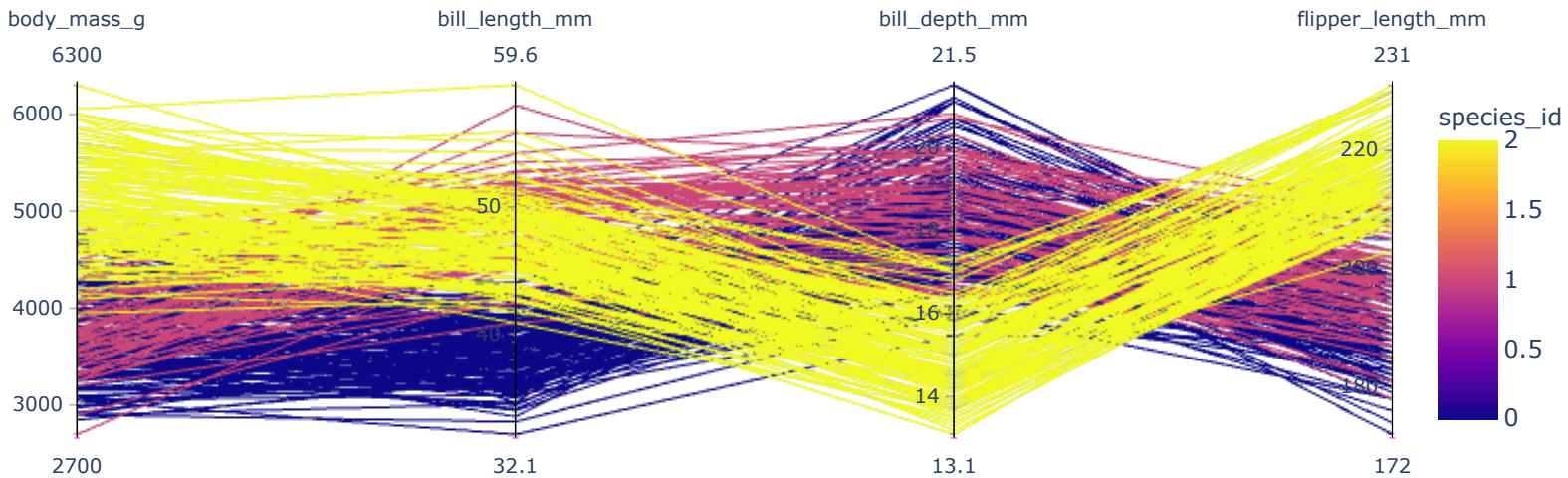
For parallel coordinate plots, the categorical variable values must be transformed to numeric codes

```
penguins.species.astype("category").cat.codes.value_counts(sort=False)
```

```
0    152
1     68
2    124
dtype: int64
```

Parallel coordinates plot

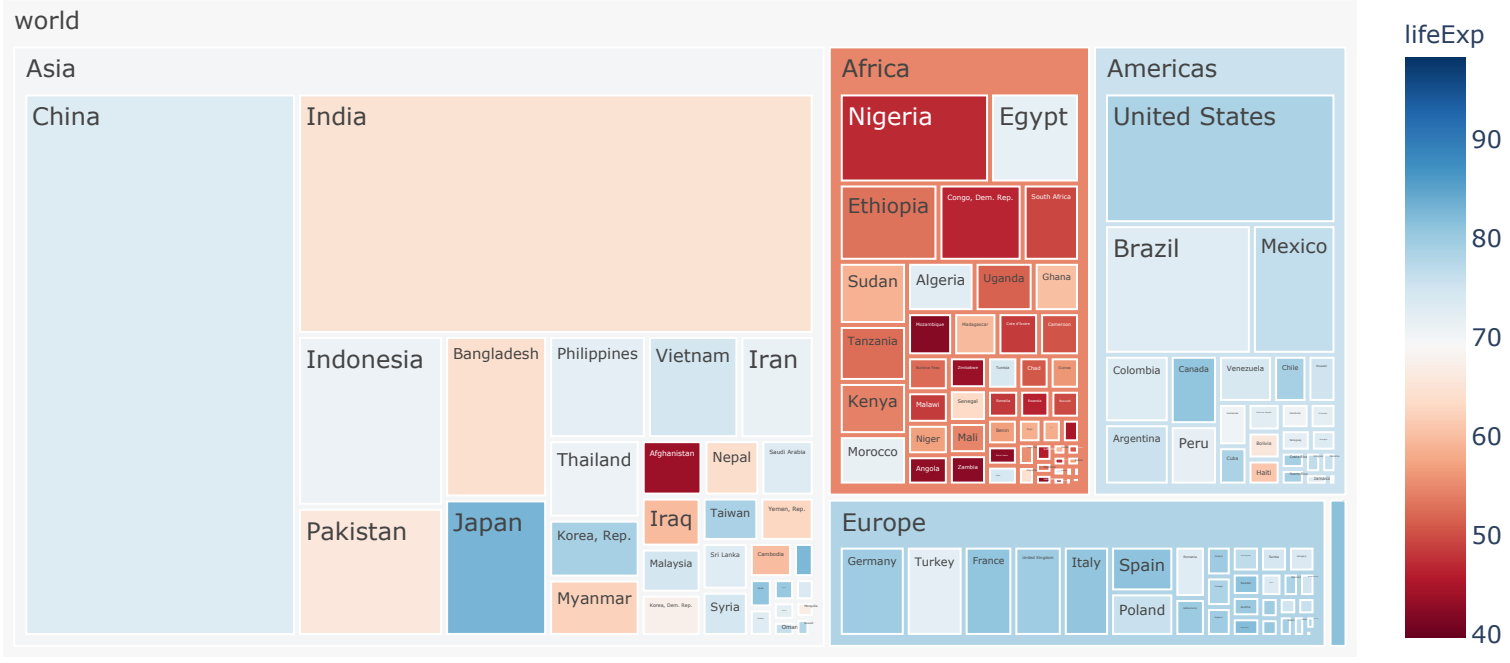
```
penguins["species_id"] = penguins.species.astype("category").cat.codes  # <<
fig = px.parallel_coordinates(
    penguins,
    dimensions=["body_mass_g", "bill_length_mm", "bill_depth_mm", "flipper_length_mm"],
    color="species_id",
)
show_fig(fig, filename="week4_files/parallel.html")
```



Treemap

```
df = px.data.gapminder().query("year == 2007")
df["world"] = "world"  # in order to have a single root node
fig = px.treemap(
    df,
    path=["world", "continent", "country"],  # << sets hierarchy
    values="pop",
    color="lifeExp",
    hover_data=["iso_alpha"],
    color_continuous_scale="RdBu",
    color_continuous_midpoint=np.average(df["lifeExp"], weights=df["pop"]),
)
show_fig(fig, filename="week4_files/treemap1.html")
```

Treemap


world
Asia
China
India
Indonesia
Bangladesh
Philippines
Vietnam
Iran
Pakistan
Japan
Thailand
Korea, Rep.
Myanmar
Afghanistan
Nepal
Iraq
Taiwan
Malaysia
Sri Lanka
Syria
Cambodia
Africa
Nigeria
Egypt
Ethiopia
Congo, Dem. Rep.
South Africa
Sudan
Algeria
Uganda
Ghana
Tanzania
Kenya
Morocco
Niger
Mali
Angola
Zambia
Malawi
Senegal
Americas
United States
Brazil
Mexico
Colombia
Canada
Venezuela
Chile
Argentina
Peru
Haiti
Europe
Germany
Turkey
France
Italy
Spain
Poland
lifeExp
90
80
70
60
50
40

More dynamism

Sliders

```
gapm = px.data.gapminder()
fig = px.scatter(
    gapm,
    x="gdpPercap",
    y="lifeExp",
    animation_frame="year",  # <<
    animation_group="country",  # <<
    size="pop",
    color="continent",
    hover_name="country",
    log_x=True,
    size_max=55,
    range_x=[100, 100000],
    range_y=[25, 90],
)
show_fig(fig, filename="week4_files/gap1.html")
```

Sliders


90
80
70
60
50
40
30
lifeExp
100
2
5
1000
2
5
10k
2
5
100k
gdpPercap
continent
Asia
Europe
Africa
Americas
Oceania
year=1952
1952
1957
1962
1967
1972
1977
1982
1987
1992
1997
2002
2007

Altair

Altair

Altair provides a wrapper around the Vega-Lite Javascript library, which is based on the famous d3.js.

It provides a syntax that explicitly describes the visual encodings that will be put on a plot.

This syntax is different from plotly, but it is clear in its own way.

Altair

```
import altair as alt

mpg = sns.load_dataset("mpg")
mpg.head()
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	name
0	18.0	8	307.0	130.0	3504	12.0	70	usa	chevrolet chevelle malibu
1	15.0	8	350.0	165.0	3693	11.5	70	usa	buick skylark 320
2	18.0	8	318.0	150.0	3436	11.0	70	usa	plymouth satellite
3	16.0	8	304.0	150.0	3433	12.0	70	usa	amc rebel sst
4	17.0	8	302.0	140.0	3449	10.5	70	usa	ford torino

```
(
    alt.Chart(mpg)  # data set
    .mark_point()  # geometry
    .encode(x="horsepower", y="mpg", color="origin")  # encodings
    .save('week4_files/alt_scatter1.html')
)
show_fig2('week4_files/alt_scatter1.html')
```

Automatic aggregations

```
alt.Chart(mpg).mark_point().encode(x="cylinders", y="average(mpg)").save('week4_files/alt_agg1.html
show_fig2('week4_files/alt_agg1.html')
```

Aggregation

More explicitly,

```
alt.Chart(mpg).mark_bar().encode(
    alt.X("cylinders", type="quantitative"),
    alt.Y("mpg", type="quantitative", aggregate="average"),
).save('week4_files/alt_agg3.html')
show_fig2('week4_files/alt_agg3.html')
```

Univariate plots

Histograms

```
alt.Chart(mpg).mark_bar().encode(
    x=alt.X("mpg:Q", bin=True),
    y="count()",
).save('week4_files/alt_hist1.html')
show_fig2('week4_files/alt_hist1.html')
```

Density plots

```
(
    alt.Chart(mpg)
    .transform_density(density="mpg", as_=["mpg", "density"])
    .mark_area()
    .encode(alt.X("mpg:Q"), alt.Y("density:Q"))
).save('week4_files/alt_density1.html')
show_fig2('week4_files/alt_density1.html')
```

Frequency bar plots

```
d = gapm.query('country=="France"')
d.head()
```

Frequency bar plots

	country	continent	year	lifeExp	pop	gdpPercap	iso_alpha	iso_num
528	France	Europe	1952	67.41	42459667	7029.809327	FRA	250
529	France	Europe	1957	68.93	44310863	8662.834898	FRA	250
530	France	Europe	1962	70.51	47124000	10560.485530	FRA	250
531	France	Europe	1967	71.55	49569000	12999.917660	FRA	250
532	France	Europe	1972	72.38	51732000	16107.191710	FRA	250

```
(
    alt.Chart(d)
    .mark_bar()
    .encode(
        y="year:O",
        x=alt.X("lifeExp:Q", title="Life expectancy"),
    )
    .save('week4_files/alt_bar1.html')
)
show_fig2('week4_files/alt_bar1.html')
```

Bivariate plots

Scatter plots

```
(
    alt.Chart(penguins)
    .mark_point()
    .encode(  # Points
        x=alt.X(
            "bill_length_mm", title="Bill length (mm)", scale=alt.Scale(zero=False)
        ),
        y=alt.Y("body_mass_g", title="Body mass (g)", scale=alt.Scale(zero=False)),
    )
    .save('week4_files/alt_points1.html')
)
show_fig2('week4_files/alt_points1.html')
```

Boxplots

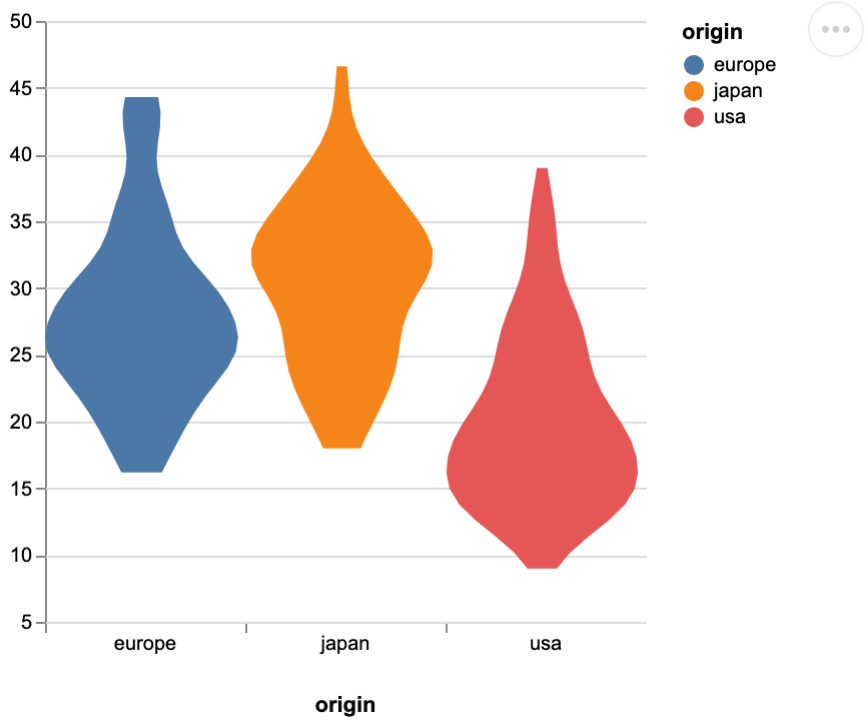
```
(
    alt.Chart(penguins)
    .mark_boxplot()
    .encode(x="species:O", y="bill_length_mm:Q")
    .properties(width=500, height=250)
    .save('week4_files/alt_box1.html')
)
show_fig2('week4_files/alt_box1.html')
```

Violin plots

Violin plots, like density plots, are a little trickier, since you have to manually compute the density using the `transform_density` function

```
(
    alt.Chart(mpg)
    .transform_density("mpg", as_=["mpg", "density"], groupby=["origin"])
    .mark_area(orient="horizontal")
    .encode(
        y=alt.Y("mpg:Q", title=""),
        x=alt.X(
            "density:Q",
            stack="center",
            impute=None,
            title=None,
            axis=alt.Axis(labels=False, values=[0], grid=False, ticks=True),
        ),
        color="origin:N",
        column=alt.Column(
            "origin:N",
            header=alt.Header(
                titleOrient="bottom",
                labelOrient="bottom",
                labelPadding=0,
            ),
        ),
    )
```

Violin plots



Strip plots

```
alt.Chart(mpg).mark_tick().encode(x="horsepower:Q", y="cylinders:O").save('week4_files/alt_strip1.ht
show_fig2('week4_files/alt_strip1.html')
```

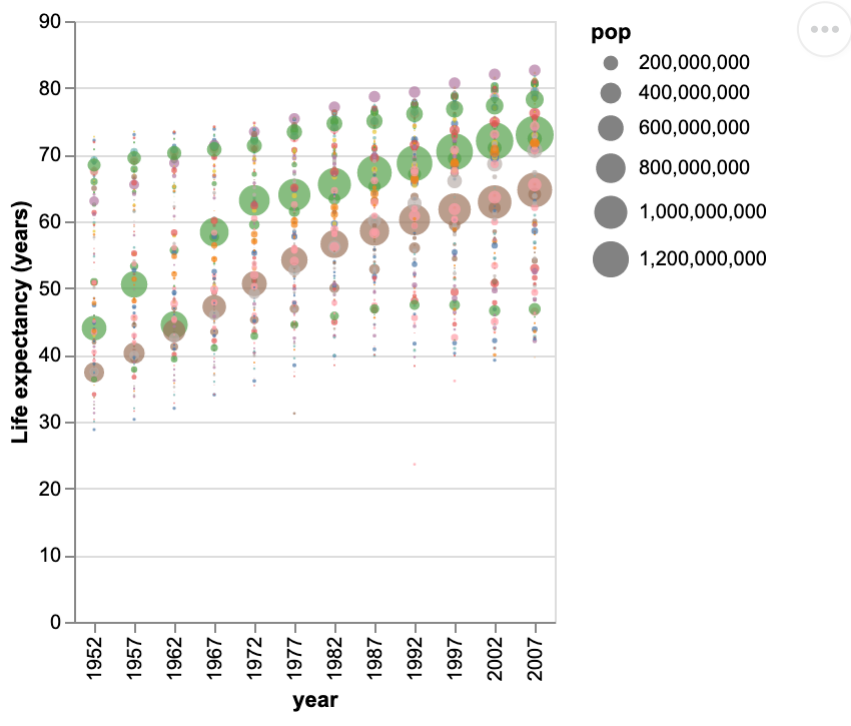
Adding layers

Scatter plots

```
alt.Chart(mpg).mark_point().encode(x="horsepower:Q", y="mpg:Q", color="origin:N").save('week4_files/
show_fig2('week4_files/alt_points_layers1.html')
```

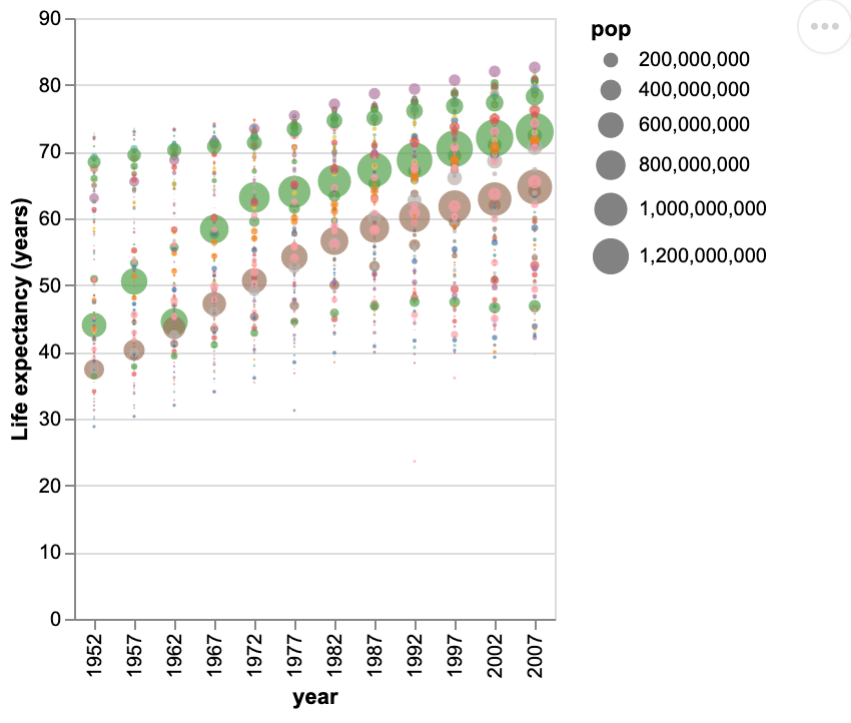
Scatter plots

```
alt.Chart(gapm).mark_circle().encode(
    x=alt.X(
        "year:O", scale=alt.Scale(zero=False)
    ),  # Don't start from 0, make year ordina
    y=alt.Y("lifeExp", title="Life expectancy
    color=alt.Color("country", legend=None),
    size="pop:Q",
).save('week4_files/alt_gap1.html')
show_fig2('week4_files/alt_gap1.html')
```



Scatter plots + tooltip

```
alt.Chart(gapm).mark_circle().encode(
    x=alt.X(
        "year:O", scale=alt.Scale(zero=False)
    ),  # Don't start from 0, make year ordina
    y=alt.Y("lifeExp", title="Life expectancy
    color=alt.Color("country", legend=None),
    size="pop:Q",
    tooltip=["country:N", "year:O", "pop", "li
).save('week4_files/alt_gap2.html')
show_fig2('week4_files/alt_gap2.html')
```



Scatter plots + tooltip

pull-left[

```
alt.Chart(gapm).mark_circle().encode(
    x=alt.X(
        "year:O", scale=alt.Scale(zero=False)
    ),  # Don't start from 0, make year ordinal
    y=alt.Y("lifeExp", title="Life expectancy (years)"),
    color=alt.Color("country", legend=None),
    size="pop:Q",
    tooltip=[
        alt.Tooltip("country", type="nominal"),
        alt.Tooltip("year", title="Year"),
        alt.Tooltip("pop:Q", title="Population", format=",.2s"),  # SI units
        alt.Tooltip("lifeExp", title="Life expectancy", format=".2f"),
    ],
).save('week4_files/alt_gap3.html')
show_fig2('week4_files/alt_gap3.html')
```

]

Bar plots

```
medals = px.data.medals_long()
alt.Chart(medals).mark_bar().encode(
    x="medal", y="sum(count):Q",
    color="medal:N",column="nation:N",
).properties(
    width=250,
).save('week4_files/alt_facet1.html')
show_fig2('week4_files/alt_facet1.html')
```

Bar plots

```
alt.Chart(medals).mark_bar().encode(
    x=alt.X("medal", sort=["gold", "silver", "bronze"]), y="sum(count):Q",
    color=alt.Color("medal:N", sort=["gold", "silver", "bronze"]), column="nation:N",
).properties(width=250).save('week4_files/alt_facet2.html')
show_fig2('week4_files/alt_facet2.html')
```

Stacked bar charts

```
alt.Chart(medals).mark_bar().encode(
    x="nation",
    y=alt.Y("count", sort="color"),
    color=alt.Color("medal:N", sort=["gold", "silver", "bronze"]),
).properties(
    width=200,
).save('week4_files/alt_stack1.html')
show_fig2('week4_files/alt_stack1.html')
```

Stacked bar charts

```
alt.Chart(medals).mark_bar().encode(
    x="nation",
    y=alt.Y("count", sort="color", stack="normalize", axis=alt.Axis(format="%")),
    color=alt.Color("medal:N", sort=["gold", "silver", "bronze"]),
).properties(
    width=200,
).save('week4_files/alt_stack2.html')
show_fig2('week4_files/alt_stack2.html')
```

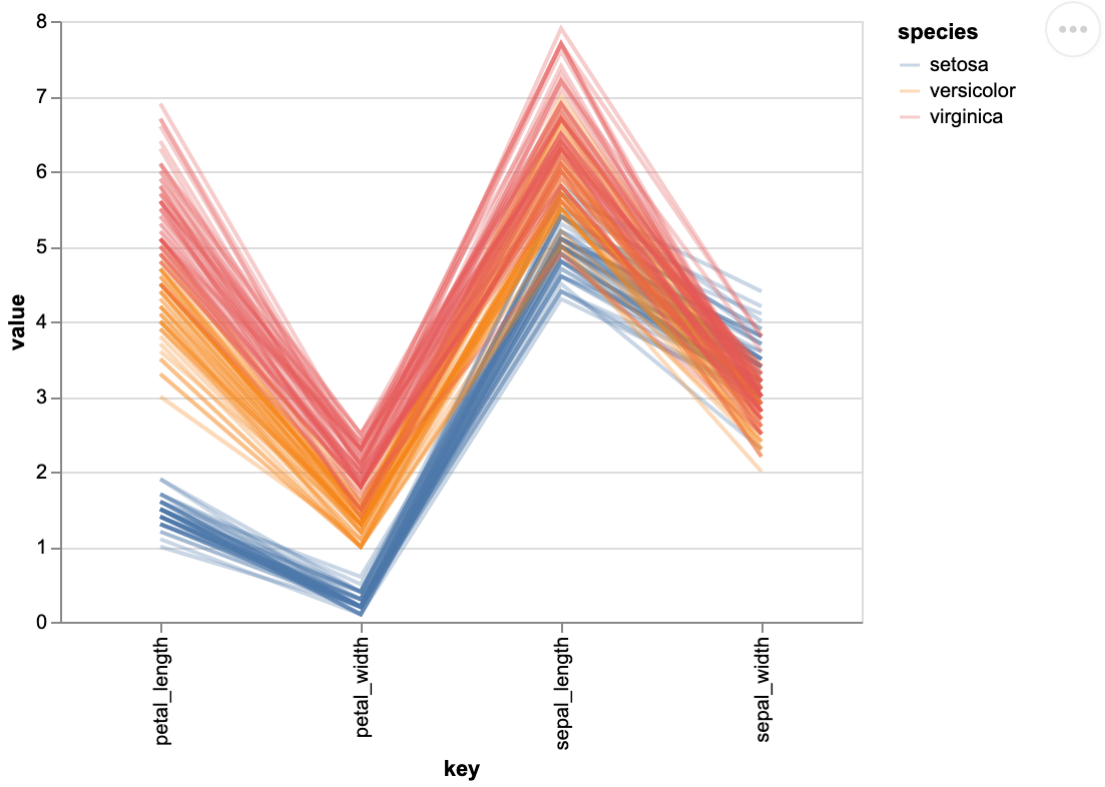
Stacked bar charts

```
medal_order = ["gold", "silver", "bronze"]  # <<
alt.Chart(medals).mark_bar().encode(
    x="nation",
    y=alt.Y("count", sort="color", stack="normalize", axis=alt.Axis(format="%")),
    color=alt.Color("medal:N", sort=["gold", "silver", "bronze"]),
    order=alt.Order("color_medal_sort_index:Q"),  # <<
).properties(
    width=200,
).save('week4_files/alt_stack3.html')
show_fig2('week4_files/alt_stack3.html')
```

Parallel coordinates

```
iris = px.data.iris()
alt.Chart(iris).transform_window(
    index="count()",
).transform_fold(  # convert wide data to long data
    ["sepal_length", "sepal_width", "petal_length", "petal_width"]
).mark_line(
    opacity=0.3
).encode(
    x="key:N",
    y="value:Q",
    color="species",
    detail="index:N",
).properties(
    width=400
).save('week4_files/alt_parallel1.html')
show_fig2('week4_files/alt_parallel1.html')
```

Parallel coordinates



Adding layers explicitly

```
from altair import datum

(
    alt.Chart(gapm)
    .transform_filter(datum.country == "Egypt")
    .mark_point()
    .encode(x="year:O", y="lifeExp:Q")
    .save('week4_files/alt_layer1.html')
)
show_fig2('week4_files/alt_layer1.html')
```

Adding layers explicitly

```
base = (
    alt.Chart(gapm)
    .transform_filter(datum.country == "Egypt")
    .encode(x="year:O", y="lifeExp:Q")
)

(base.mark_point() + base.mark_line()).save('week4_files/alt_layer2.html')
show_fig2('week4_files/alt_layer2.html')
```

Adding layers explicitly

```
base = (
    alt.Chart(gapm)
    .transform_filter(datum.country == "Egypt")
    .encode(x="year:O", y="lifeExp:Q")
)

alt.layer(base.mark_point(), base.mark_line()).save('week4_files/alt_layers3.html')
show_fig2('week4_files/alt_layers3.html')
```

Scatter plots + lines

```
base = alt.Chart(penguins).encode(
    x=alt.X("bill_length_mm:Q", scale=alt.Scale(zero=False)),
    y=alt.Y("body_mass_g:Q", scale=alt.Scale(zero=False)),
    color="species:N",
)

(base.mark_point() + base.transform_regression(
    "bill_length_mm", "body_mass_g", groupby=["species"]
).mark_line(size=4)).save('week4_files/alt_layers4.html')
show_fig2('week4_files/alt_layers4.html')
```

Scatter plots + lines

```
base = alt.Chart(penguins).encode(
    x=alt.X("bill_length_mm:Q", scale=alt.Scale(zero=False)),
    y=alt.Y("body_mass_g:Q", scale=alt.Scale(zero=False)),
    color="species:N",
)

(base.mark_point() + base.transform_loess(
    "bill_length_mm", "body_mass_g", groupby=["species"]
).mark_line(size=4)).save('week4_files/alt_layers5.html')
show_fig2('week4_files/alt_layers5.html')
```

Facets

```
alt.Chart(mpg).mark_point().encode(x="horsepower:Q", y="mpg:Q", column="origin:N").save('week4_files
show_fig2('week4_files/alt_facets4.html')
```

Facets

```
alt.Chart(penguins).mark_point().encode(
    x=alt.X("bill_length_mm:Q", title="Bill length (mm)", scale=alt.Scale(zero=False)),
    y=alt.Y("body_mass_g:Q", title="Body mass (g)", scale=alt.Scale(zero=False)),
    column=alt.Column("species:N", title=None),
    row=alt.Row("island:N", title=None),
).properties(width=300).save('week4_files/alt_facet5.html')
show_fig2('week4_files/alt_facet5.html')
```

Scatterplot matrix

```
alt.Chart(penguins).mark_circle().encode(
    x=alt.X(alt.repeat("column"), type="quantitative", scale=alt.Scale(zero=False)),
    y=alt.Y(alt.repeat("row"), type="quantitative", scale=alt.Scale(zero=False)),
    color="species:N",
).properties(width=200, height=200).repeat(
    row=["bill_length_mm", "bill_depth_mm", "flipper_length_mm"],
    column=["bill_length_mm", "bill_depth_mm", "flipper_length_mm"],
).save('week4_files/alt_splom.html')
show_fig2('week4_files/alt_splom.html')
```

Putting charts together

Side-by-side

```
plot1 = (
    alt.Chart(penguins)
    .mark_circle()
    .encode(x="bill_length_mm", y="body_mass_g", color="species")
)
plot2 = (
    alt.Chart(penguins)
    .mark_circle()
    .encode(x="bill_depth_mm", y="body_mass_g", color="species")
)

(plot1 | plot2).save('week4_files/alt_concat1.html')
show_fig2('week4_files/alt_concat1.html')
```

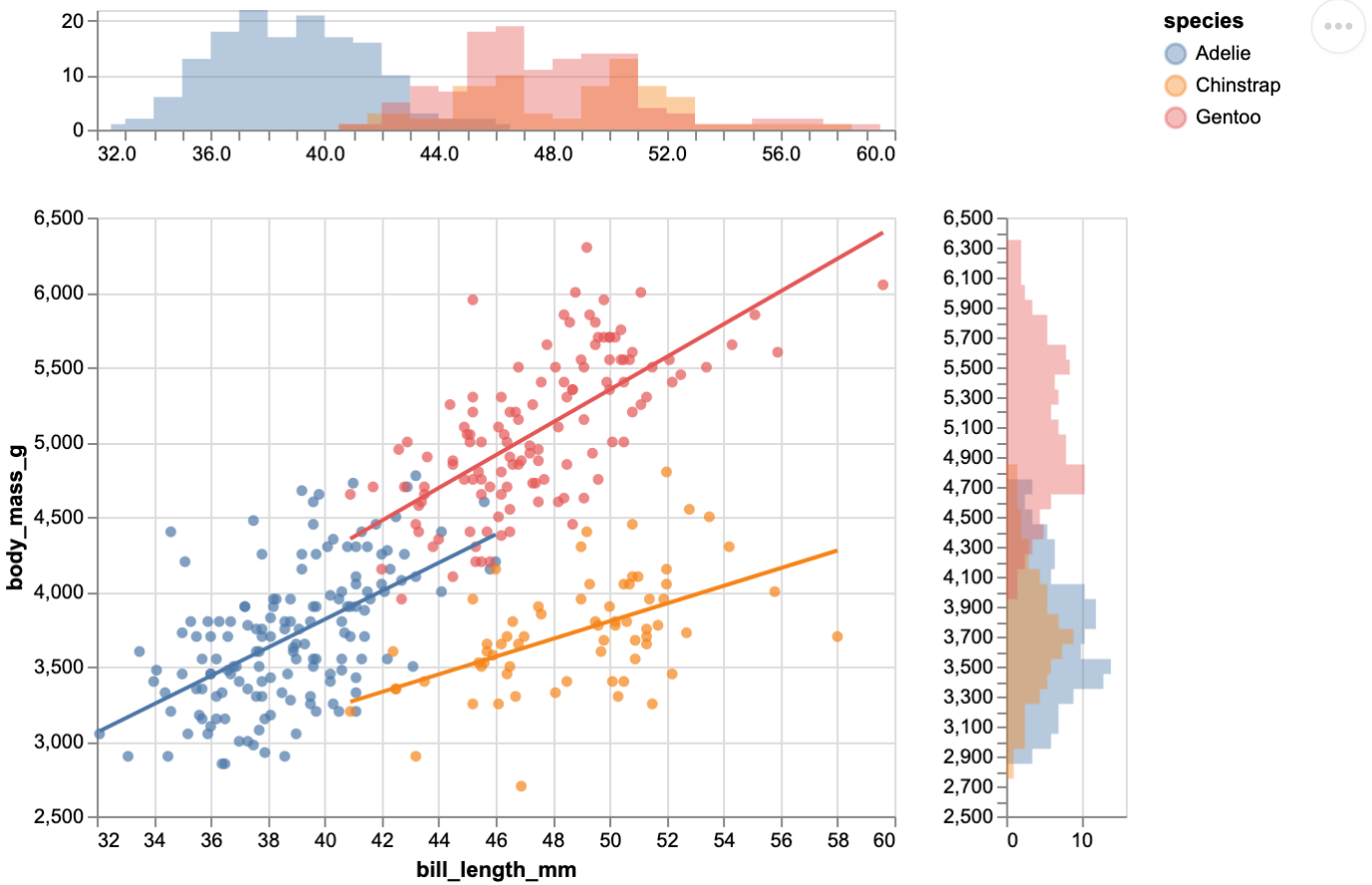
In a column

```
plot1 = (
    alt.Chart(penguins)
    .mark_circle()
    .encode(x="bill_length_mm",
    y="body_mass_g",
    color="species")
)
plot2 = (
    alt.Chart(penguins)
    .mark_circle()
    .encode(
      x="bill_depth_mm",
      y="body_mass_g",
      color="species")
)

(plot1 & plot2).save('week4_files/alt_concat2.
show_fig2('week4_files/alt_concat2.html', heig
```

Composite plots

```
base = alt.Chart(penguins)
xscale = alt.Scale(zero=False, domain=(32, 60))
yscale = alt.Scale(zero=False, domain=(2500, 6500))
area_args = {"opacity": 0.4, "interpolate": "step"}

scatter = base.mark_circle().encode(
    x=alt.X("bill_length_mm", scale=xscale),
    y=alt.Y("body_mass_g", scale=yscale),
    color="species",
)

top_hist = (
    base.mark_area(**area_args)
    .encode(
        x=alt.X(
            "bill_length_mm",
            bin=alt.Bin(maxbins=50, extent=xscale.domain),
            stack=None,
            title="",
        ),
        y=alt.Y("count()", stack=None, title=""),
        color="species",
    )
    .properties(height=60)
)
```

Composite plots

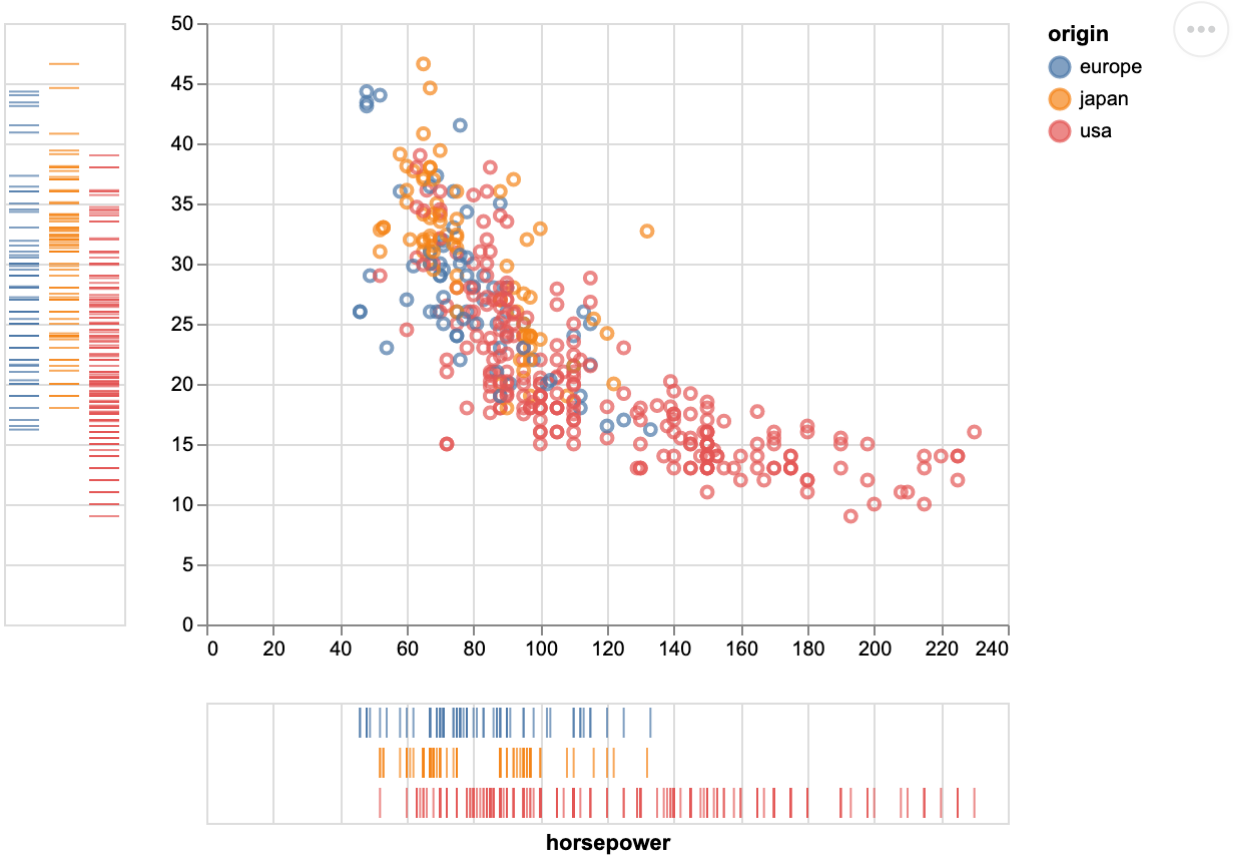


Interactivity

Brushing

```
brush = alt.selection(type="interval")
base = alt.Chart(mpg).add_selection(brush)

scatter = base.mark_point().encode(
    x=alt.X("horsepower:Q", title=""),
    y=alt.Y("mpg:Q", title=""),
    color=alt.condition(brush, "origin:N", alt.value("grey")),
)

tick_axis = alt.Axis(labels=False, ticks=False, domain=False)
x_ticks = base.mark_tick().encode(
    x=alt.X("horsepower", axis=tick_axis),
    y=alt.Y("origin", title="", axis=tick_axis),
    color=alt.condition(brush, "origin", alt.value("lightgrey")),
)
y_ticks = base.mark_tick().encode(
    x=alt.X("origin", title="", axis=tick_axis),
    y=alt.Y("mpg", title="", axis=tick_axis),
    color=alt.condition(brush, "origin", alt.value("lightgrey")),
)

(y_ticks | (scatter & x_ticks)).save('week4_files/alt_brush1.html')
show_fig2('week4_files/alt_brush1.html')
```

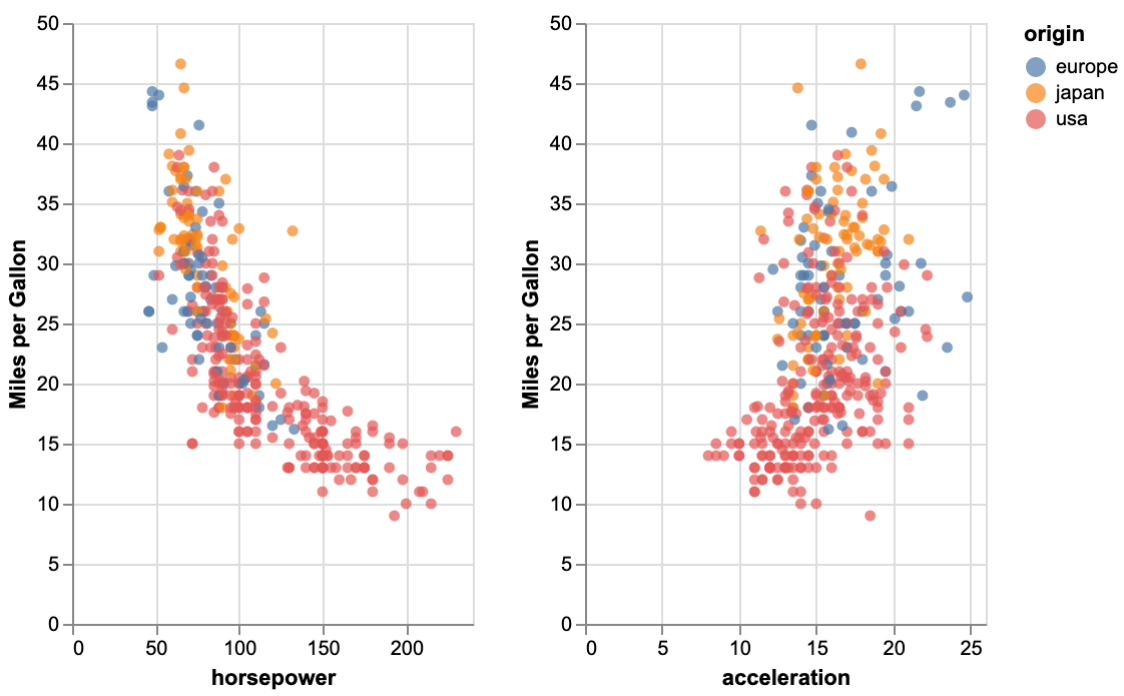
Brushing


origin
europe
japan
usa
50
45
40
35
30
25
20
15
10
5
0
0
20
40
60
80
100
120
140
160
180
200
220
240
horsepower

Brushing

```
brush = alt.selection(type="interval", resolve="global")

base = (
    alt.Chart(mpg)
    .mark_circle()
    .encode(
        y=alt.Y("mpg", title="Miles per Gallon"),
        color=alt.condition(brush, "origin", alt.value("gray")),
    )
    .add_selection(brush)
    .properties(width=200)
)

(base.encode(x="horsepower") | base.encode(x="acceleration")).save('week4_files/alt_brush2.html')
show_fig2('week4_files/alt_brush2.html')
```

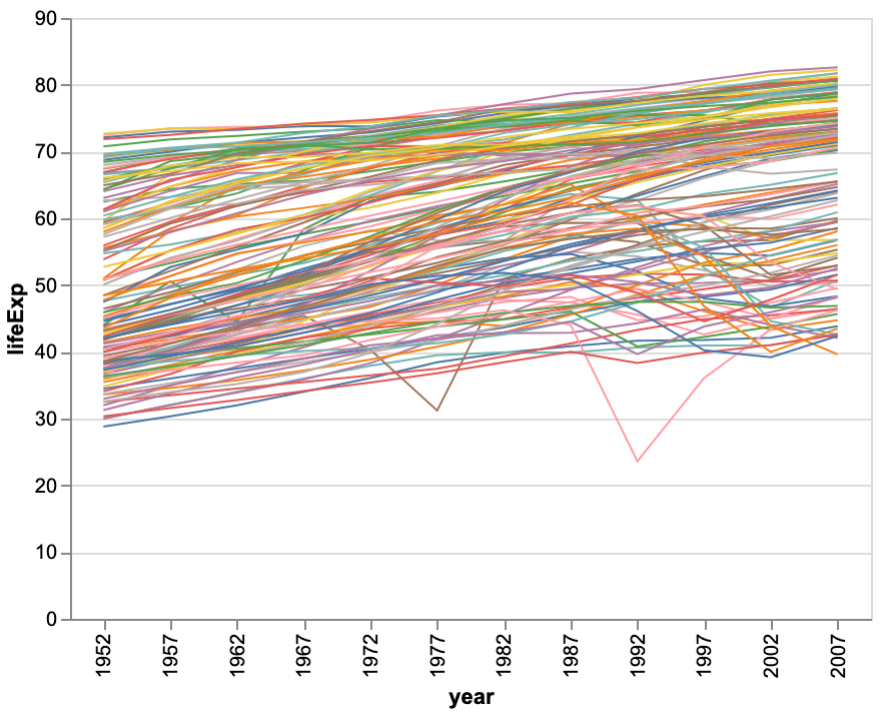
Brushing


Miles per Gallon
horsepower
acceleration
origin
europe
japan
usa

Multi-line highlight

```
highlight = alt.selection(
    type="single", on="mouseover", fields=["country"], nearest=True
)

base = alt.Chart(gapm).encode(
    x="year:O",
    y="lifeExp:Q",
)
points = (
    base.mark_circle()
    .encode(
        opacity=alt.value(0),
        tooltip=["country", "year", "lifeExp"],
    )
    .add_selection(highlight)
    .properties(width=400)
)

lines = base.mark_line().encode(
    size=alt.condition(~highlight, alt.value(1), alt.value(3)),
    color=alt.condition(highlight, "country", alt.value("lightgrey"), legend=None),
)

(points + lines).save('week4_files/alt_lines1.html')
show_fig2('week4_files/alt_lines1.html')
```

Multi-line highlight



Resources

1. Encodings
2. Geometries
3. Transformations