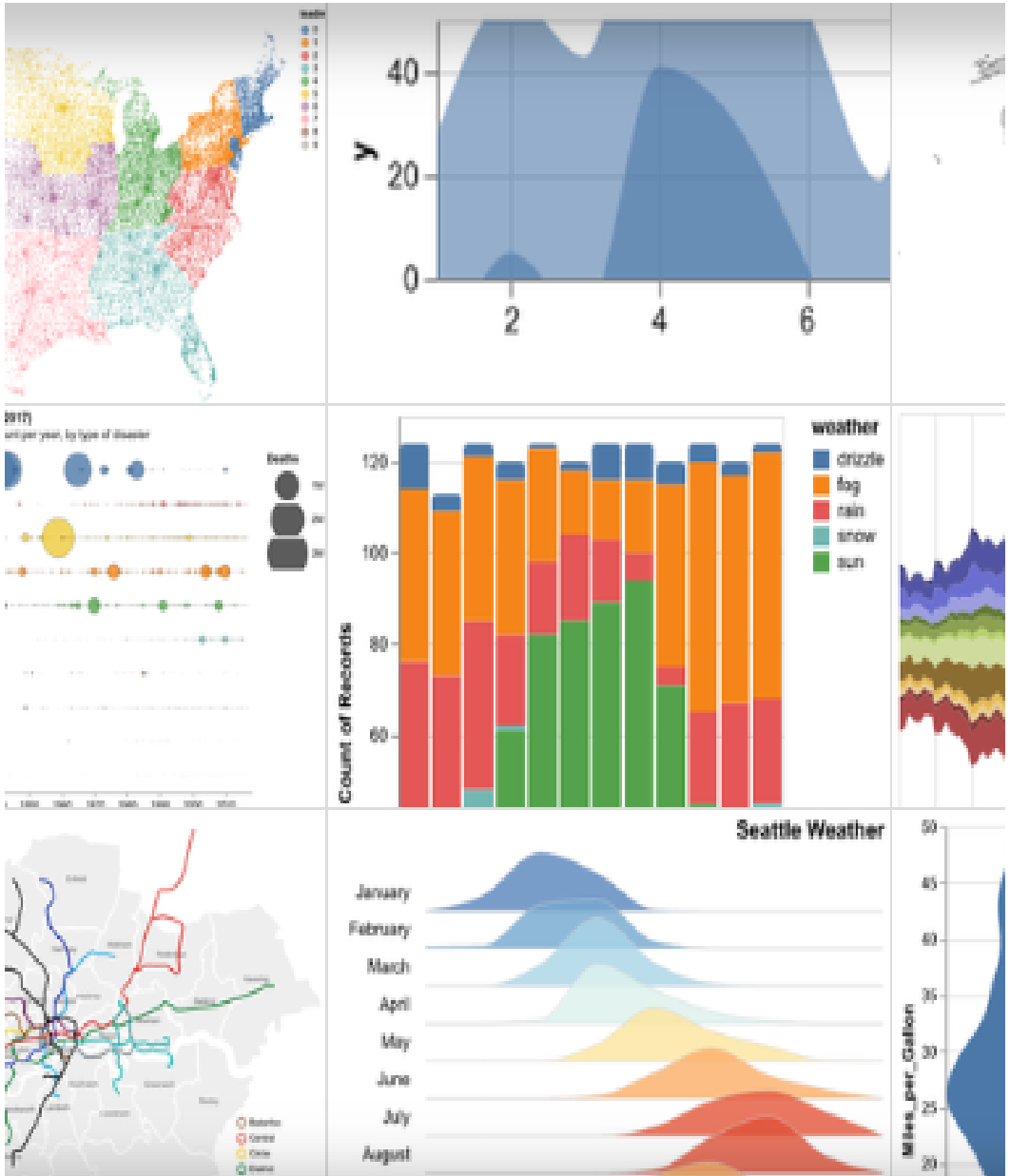
4 • VEGA ALTAIR-VIZ

TEACHER
Salvo Rinzivillo

University of Pisa
Department of Computer Science

Course: Visual Analytics (602AA)
Academic Year: 2024/2025

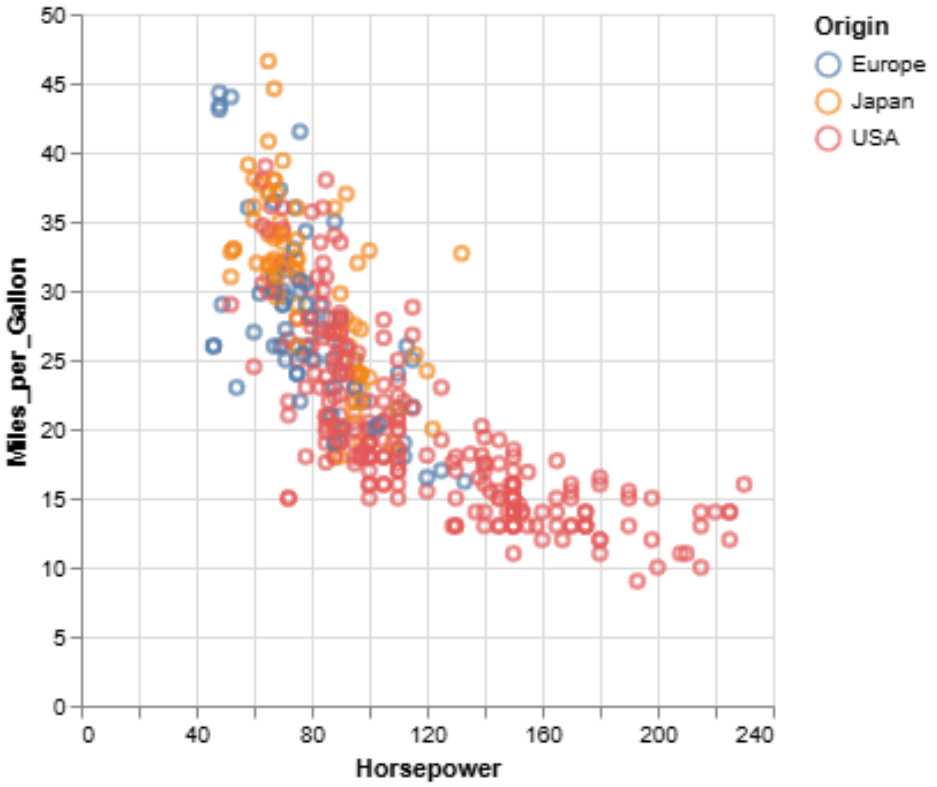


Vega-Altair and Vega-Lite

- **Vega** is a visualization grammar, a declarative format for creating, saving, and sharing interactive visualization designs.
- **Vega-Lite** is a high-level grammar of interactive graphics. It provides a concise JSON syntax for rapidly generating visualizations to support analysis.
- **Altair** is a declarative statistical visualization library for Python, based on Vega and Vega-Lite.

First Example in Altair

```
# import altair with an abbreviated alias
import altair as alt

# load a sample dataset as a pandas DataFrame
from vega_datasets import data
cars = data.cars()

# make the chart
alt.Chart(cars).mark_point().encode(
    x='Horsepower',
    y='Miles_per_Gallon',
    color='Origin',
).interactive()
```



Altair Basic Concepts

- **The Data**: The data to be visualized is passed to the Chart object as a Pandas DataFrame.
- **Encodings and Marks**: The visual appearance of the data is specified using encodings and marks.
  - Encodings map data fields to visual properties, such as position, color, size, and shape.
  - Marks are the basic building blocks of a visualization, such as points, lines, bars, and areas.
- **Data Transformation**: Altair provides a number of methods for transforming the data before it is visualized.
- **Interaction**: Altair provides a number of methods for adding interactivity to the visualization.

DATA

Altair - Specifying Data

- **Data**: Altair uses tabular data as its basic data model.Individual rows represent individual data points, and columns represent attributes of the data points.
- **Data Source**: The data to be visualized is passed to the Chart object as a `Pandas DataFrame`.
- **Data Types**: Altair supports a number of data types, including quantitative, ordinal, nominal, and temporal data. These data types are inferred from the `DataFrame` columns.
- **`Data` Object**: Altair provides a dedicated class to represent the data, which can be used to specify the data types and formats.

```
import altair as alt

data = alt.Data(values=[{'x': 'A', 'y': 5},
                        {'x': 'B', 'y': 3},
                        {'x': 'C', 'y': 6},
                        {'x': 'D', 'y': 7},
                        {'x': 'E', 'y': 2}])
alt.Chart(data).mark_bar().encode(
    x='x:N',  # specify nominal data
    y='y:Q',  # specify quantitative data
)
```

Long-form data and Wide-form data

- **Long-form data**: Each row represents a single observation, and each column represents a variable.
- **Wide-form data**: it has one row per independent variable and metadata encoded in columns and rows names.

Wide-form data

	Date	AAPL	AMZN	GOOG
0	2007-10-01	189.95	89.15	707.00
1	2007-11-01	182.22	90.56	693.00
2	2007-12-01	198.08	92.64	691.48

Long-form data

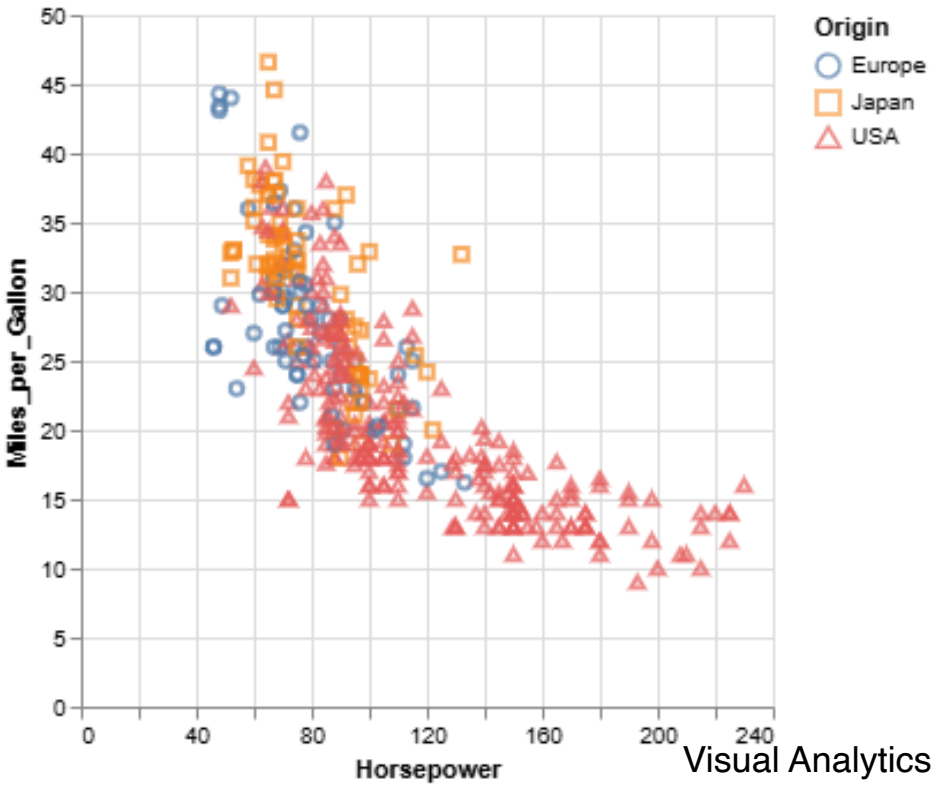
	Date	company	price
0	2007-10-01	AAPL	189.95
1	2007-11-01	AAPL	182.22
2	2007-12-01	AAPL	198.08
3	2007-10-01	AMZN	89.15
4	2007-11-01	AMZN	90.56
5	2007-12-01	AMZN	92.64
...			

ENCODINGS

Altair - Encodings

- **Encodings**: Encodings map data fields to visual properties, such as position, color, size, and shape.
- Encodings represent the visual properties of the marks in the visualization.

```
import altair as alt
from vega_datasets import data

cars = data.cars()

alt.Chart(cars).mark_point().encode(
    x='Horsepower',
    y='Miles_per_Gallon',
    color='Origin',
    shape='Origin'
)
```



Channel Options

The **channel options** are the parameters to customize the visual mappings. The new version introduces a new style for options, with the following syntax:

Method-based Syntax

```
alt.Chart(cars).mark_point().encode(
    alt.X('Horsepower').axis(tickMinStep=50),
    alt.Y('Miles_per_Gallon').title('Miles per Gallon'),
    color='Origin',
    shape='Origin'
)
```

Attribute-based Syntax

```
alt.Chart(cars).mark_point().encode(
    alt.X('Horsepower', axis=alt.Axis(tickMinStep=50)),
    alt.Y('Miles_per_Gallon', title="Miles per Gallon"),
    color='Origin',
    shape='Origin'
)
```

Encodings Data Types

Altair supports five main data types for encodings:

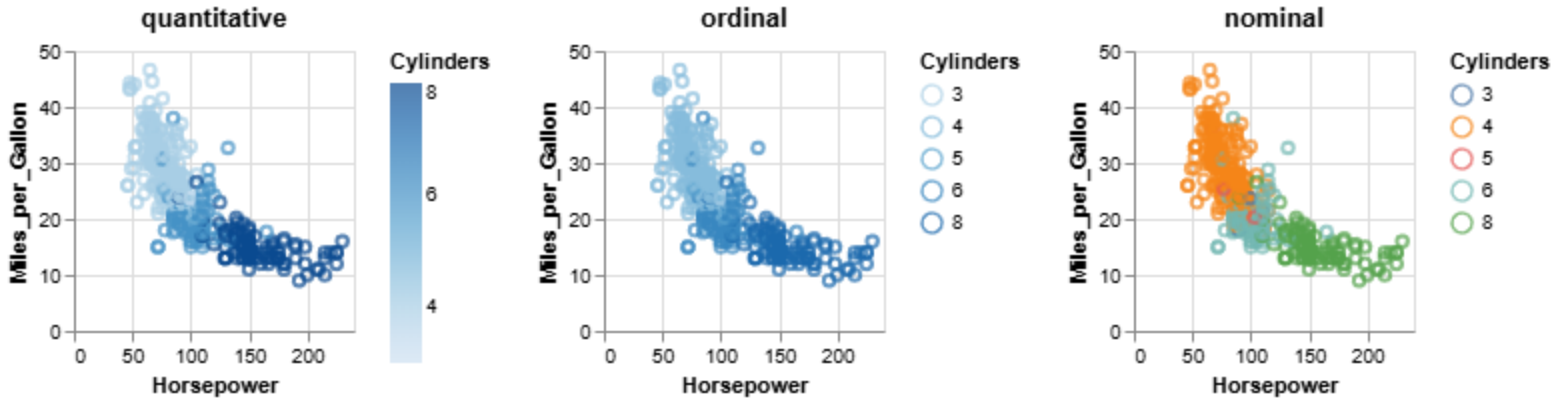
Data Type	Shorthand Code	Description
quantitative	Q	a continuous real-valued quantity
ordinal	O	a discrete ordered quantity
nominal	N	a discrete unordered category
temporal	T	a time or date value
geojson	G	a geographic shape

```
alt.Chart(cars).mark_point().encode(
    x='Acceleration:Q',
    y='Miles_per_Gallon:Q',
    color='Origin:N'
)
```

```
alt.Chart(cars).mark_point().encode(
    alt.X('Acceleration', type='quantitative'),
    alt.Y('Miles_per_Gallon', type='quantitative'),
    alt.Color('Origin', type='nominal')
)
```

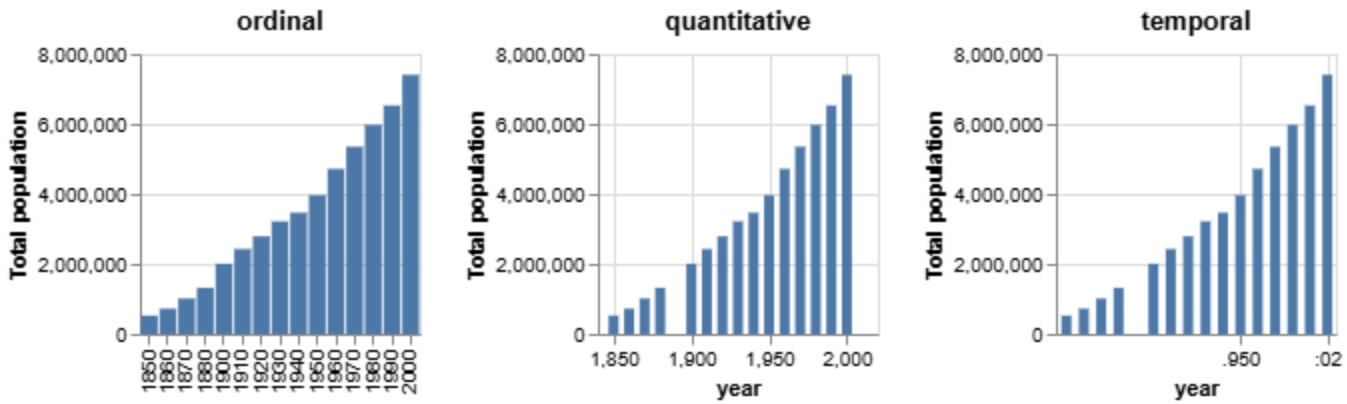
Choosing the correct data type (colors)

```
base = alt.Chart(cars).mark_point().encode(
    x='Horsepower:Q',
    y='Miles_per_Gallon:Q',
).properties(width=140, height=140)

alt.hconcat(
   base.encode(color='Cylinders:Q').properties(title='quantitative'),
   base.encode(color='Cylinders:O').properties(title='ordinal'),
   base.encode(color='Cylinders:N').properties(title='nominal'),
)
```



Choosing the correct data type (axis)

```
base = alt.Chart(pop).mark_bar().encode(
    alt.Y('mean(people):Q').title('Total population')
).properties(width=140, height=140)

alt.hconcat(
    base.encode(x='year:O').properties(title='ordinal'),
    base.encode(x='year:Q').properties(title='quantitative'),
    base.encode(x='year:T').properties(title='temporal')
)
```



Encodings Shorthands

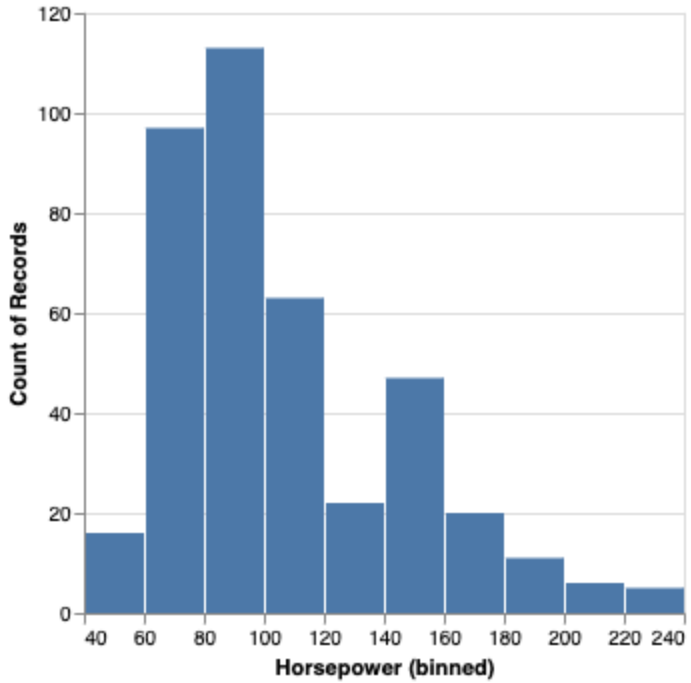
The shorthand syntax is a convenient way to specify encodings in Altair. The shorthand syntax uses a colon to separate the field name from the data type.

Shorthand Syntax	Equivalent long-form
`x='name'`	`alt.X('name')`
`x='name:Q'`	`alt.X('name', type='quantitative')`
`x='sum(name)'`	`alt.X('name', aggregate='sum')`
`x='sum(name):Q'`	`alt.X('name', aggregate='sum', type='quantitative')`
`x='count():Q'`	`alt.X(aggregate='count', type='quantitative')`

Binning and Aggregation - Histogram

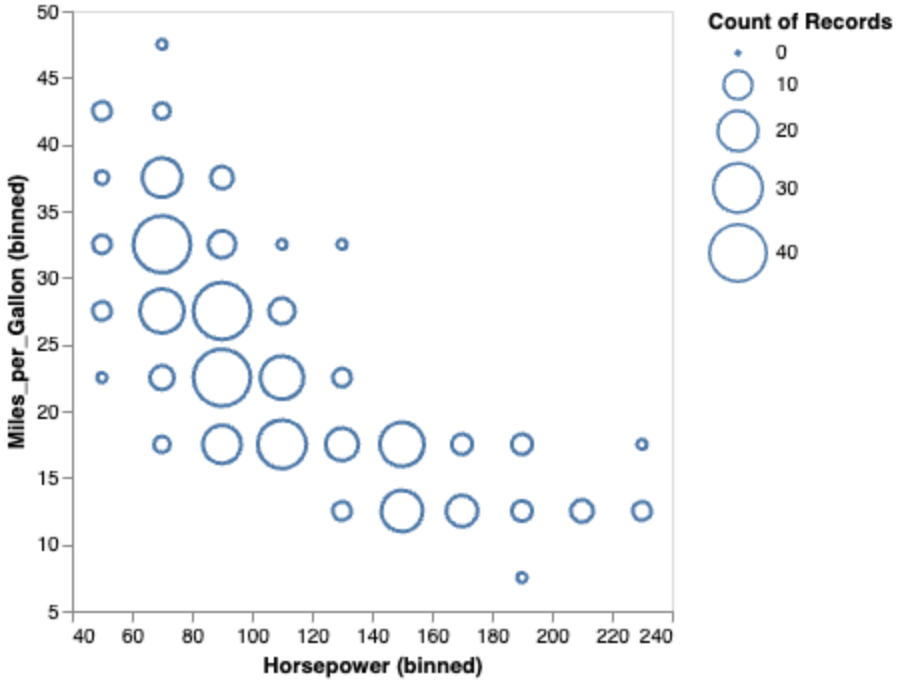
Altair's visualizations are built on the concept of the database-style grouping and aggregation. The split-apply-combine abstraction underpins many data analysis approaches.

```
alt.Chart(cars).mark_bar().encode(
    alt.X('Horsepower').bin(),
    y='count()'
    # could also use alt.Y(aggregate='count', type='quantitative')
)
```



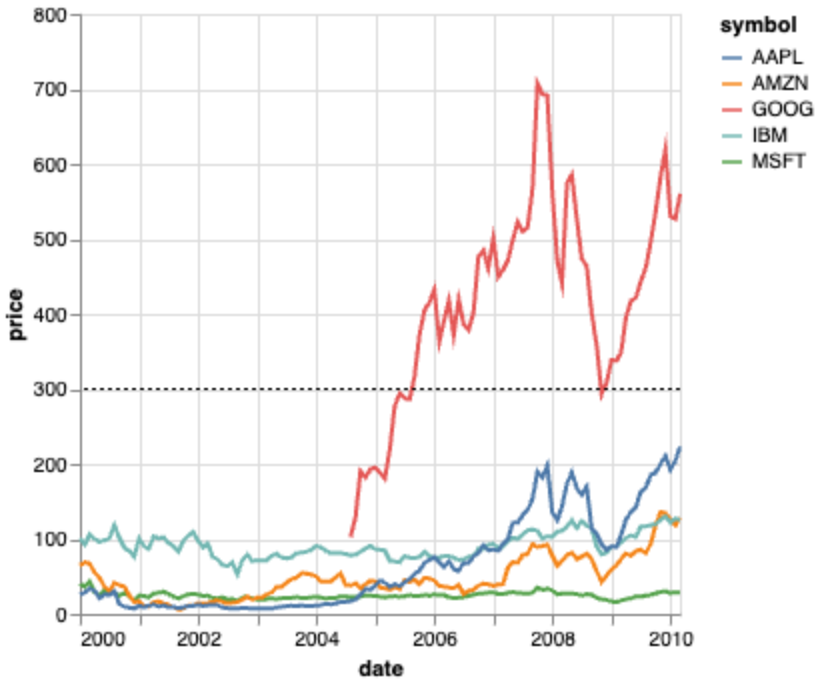
Binning and Aggregation - 2D Histogram

```
alt.Chart(cars).mark_point().encode(
    alt.X('Horsepower').bin(),
    alt.Y('Miles_per_Gallon').bin(),
    size='count()',
)
```



Datum

```
import altair as alt
from vega_datasets import data

source = data.stocks()
base = alt.Chart(source)
lines = base.mark_line().encode(
    x="date:T",
    y="price:Q",
    color="symbol:N"
)
rule = base.mark_rule(strokeDash=[2, 2]).encode(
    y=alt.datum(300)
)

lines + rule
```



Marks (1)

Marks are the basic building blocks of a visualization. The `mark` property is what specifies how exactly those attributes should be represented on the plot.

Mark	Method	Description
Arc	mark_arc()	A pie chart.
Area	mark_area()	A filled area plot.
Bar	mark_bar()	A bar plot.
Circle	mark_circle()	A scatter plot with filled circles.
Geoshape	mark_geoshape()	Visualization containing spatial data
Image	mark_image()	A scatter plot with image markers.
Line	mark_line()	A line plot.
Point	mark_point()	A scatter plot with configurable point shapes.

Marks (2)

Marks are the basic building blocks of a visualization. The `mark` property is what specifies how exactly those attributes should be represented on the plot.

Mark	Method	Description
Rect	mark_rect()	A filled rectangle, used for heatmaps
Rule	mark_rule()	A vertical or horizontal line spanning the axis.
Square	mark_square()	A scatter plot with filled squares.
Text	mark_text()	A scatter plot with points represented by text.
Tick	mark_tick()	A vertical or horizontal tick mark.
Trail	mark_trail()	A line with variable width.

Composite Marks

Composite marks are a combination of multiple marks that are used to create more complex visualizations.

Mark	Method	Description
Errorband	mark_errorband()	An error band around a line
Errorbar	mark_errorbar()	An error bar around a point
Boxplot	mark_boxplot()	A box plot.

DATA TRANSFORMATION

Data Transformation

It is often necessary to transform or filter data in the process of visualizing it. In Altair you can do this one of two ways:

- Before the chart definition, using standard pandas data transformations.
- Within the chart definition, using Vega-Lite's data transformation tools.

Data Transformation Operators (1)

Tranform	Method	Description
Aggregate	transform_aggregate()	Create a new data column by aggregating an existing column.
Bin	transform_bin()	Create a new data column by binning an existing column.
Calculate	transform_calculate()	Create a new data column using an arithmetic calculation on an existing column.
Density	transform_density()	Create a new data column with the kernel density estimate of the input.
Extent	transform_extent()	Find the extent of a field and store the result in a parameter.
Filter	transform_filter()	Select a subset of data based on a condition.

Data Transformation Operators (2)

Tranform	Method	Description
Flatten	transform_flatten()	Flatten array data into columns.
Fold	transform_fold()	Convert wide-form data into long-form data (opposite of pivot).
Impute	transform_impute()	Impute missing data.
Join Aggregate	transform_joinaggregate()	Aggregate transform joined to original data.
LOESS	transform_loess()	Create a new column with LOESS smoothing of data.
Lookup	transform_lookup()	One-sided join of two datasets based on a lookup key.
Pivot	transform_pivot()	Convert long-form data into wide-form data (opposite of fold).

Data Transformation Operators (3)

Tranform	Method	Description
Quantile	transform_quantile()	Compute empirical quantiles of a dataset.
Regression	transform_regression()	Fit a regression model to a dataset.
Sample	transform_sample()	Random sub-sample of the rows in the dataset.
Stack	transform_stack()	Compute stacked version of values.
TimeUnit	transform_timeunit()	Discretize/group a date by a time unit (day, month, year, etc.)
Window	transform_window()	Compute a windowed aggregation

INTERACTION

Interaction

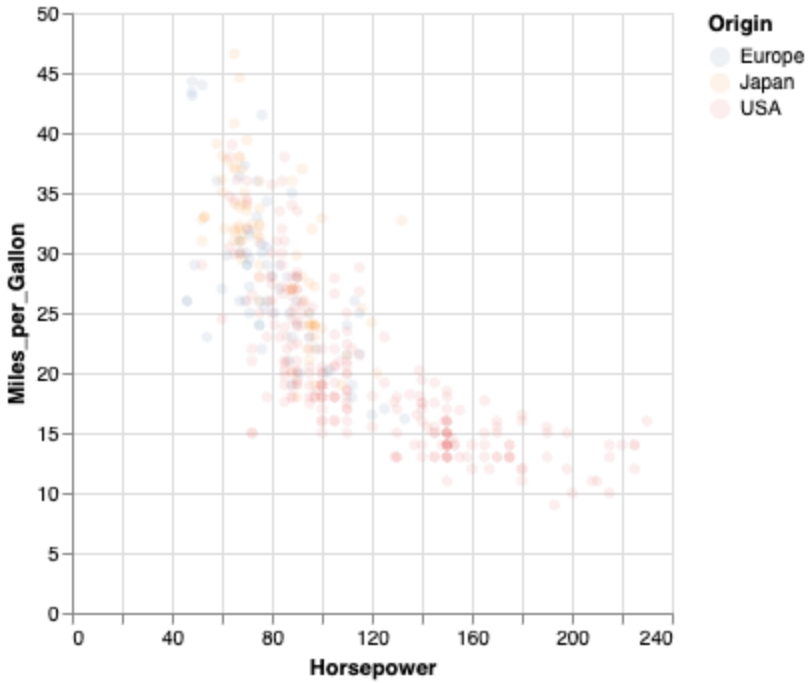
Altair provides a declarative grammar for specifying interactive visualizations. Interaction is based on three key concepts:

- **Parameters** are the basic building blocks in the grammar of interaction. They can either be simple variables or more complex selections that map user input (e.g., mouse clicks and drags) to data queries.
- **Conditions and filters** can respond to changes in parameter values and update chart elements based on that input.
- **Widgets** and other **chart input elements** can bind to parameters so that charts can be manipulated via drop-down menus, radio buttons, sliders, legends, etc.

Parameters - Variables

Variable parameters allow for a value to be defined once and then reused throughout the rest of the chart. This is useful for defining a value that is used in multiple places in the chart, such as a color or size.

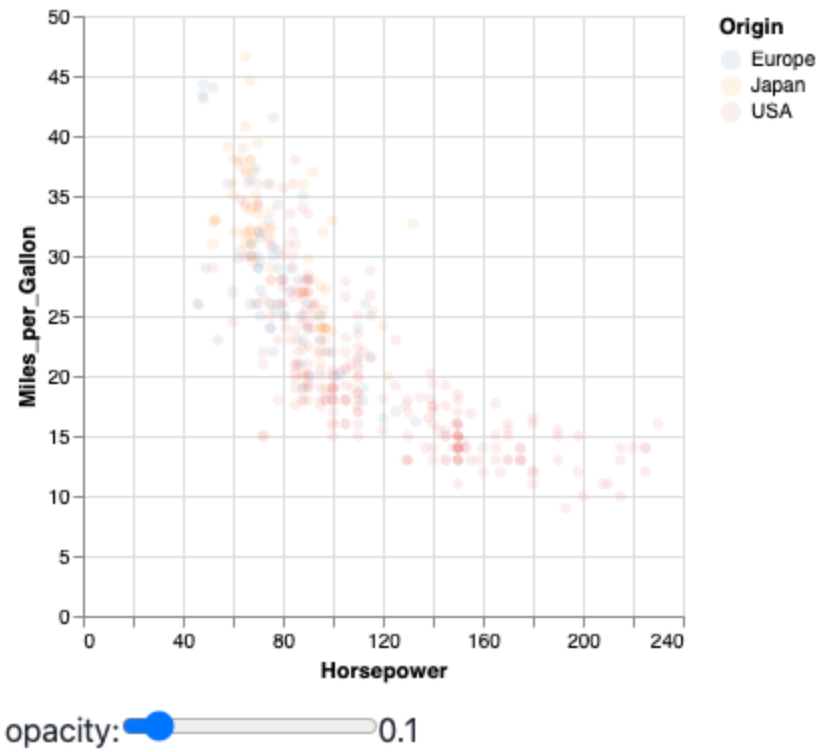
```
op_var = alt.param(value=0.1)

alt.Chart(cars).mark_circle(opacity=op_var).encode(
    x='Horsepower:Q',
    y='Miles_per_Gallon:Q',
    color='Origin:N'
).add_params(
    op_var
)
```



Parameters - Widgets

Widgets are interactive elements that allow users to manipulate the chart. Widgets can be bound to parameters, so that the chart updates in response to user input.

```
slider = alt.binding_range(min=0, max=1, step=0.05, name='opacity:')
op_var = alt.param(value=0.1, bind=slider)

alt.Chart(cars).mark_circle(opacity=op_var).encode(
    x='Horsepower:Q',
    y='Miles_per_Gallon:Q',
    color='Origin:N'
).add_params(
    op_var
)
```

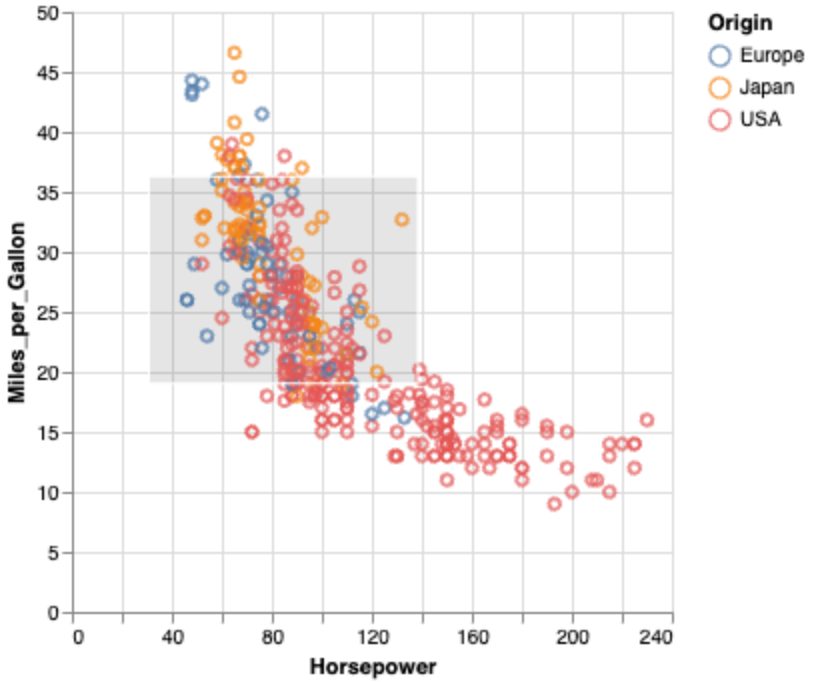


Selection and Charts Interaction

Selection parameters define data queries that are driven by interactive manipulation of the chart by the user (e.g., via mouse clicks or drags).

There are two types of selections: `selection_interval()` and `selection_point()`.

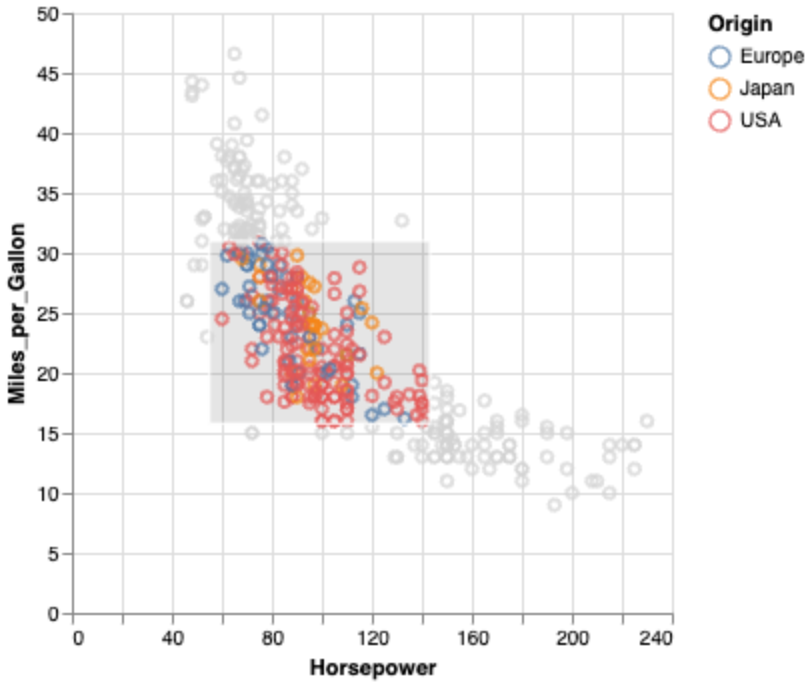
```
brush = alt.selection_interval()

alt.Chart(cars).mark_point().encode(
    x='Horsepower:Q',
    y='Miles_per_Gallon:Q',
    color='Origin:N'
).add_params(
    brush
)
```



Conditions

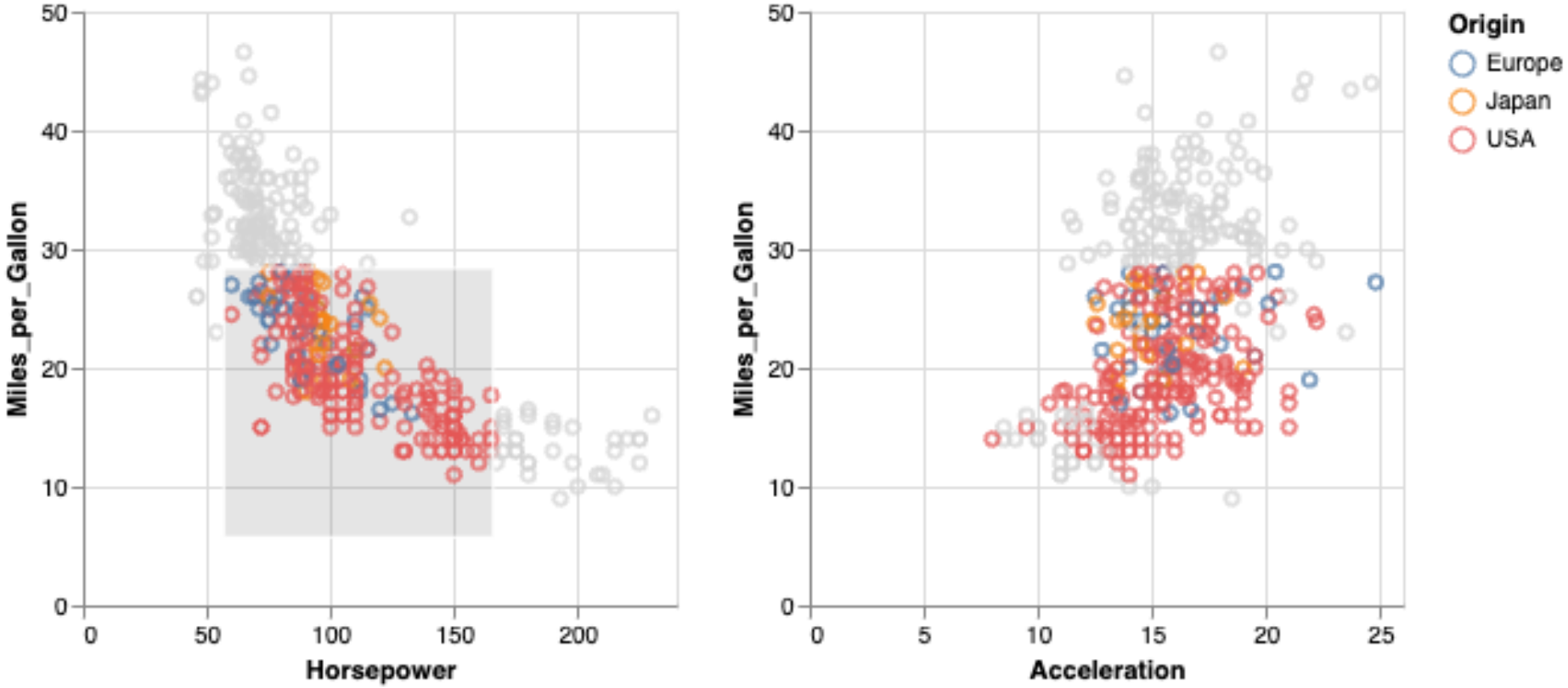
Conditions are used to update the chart based on the value of a parameter. Conditions can be used to change the appearance of the chart, filter the data, or update the chart in other ways.

```
conditional = alt.when(brush).then("Origin:N").otherwise(alt.value("lightgray"))

alt.Chart(cars).mark_point().encode(
    x="Horsepower:Q",
    y="Miles_per_Gallon:Q",
    color=conditional,
).add_params(
    brush
)
```



Linked Conditions (1)

Conditions can be linked together to create more complex interactions among charts. This allows for more sophisticated interactions between different parts of the visualization.



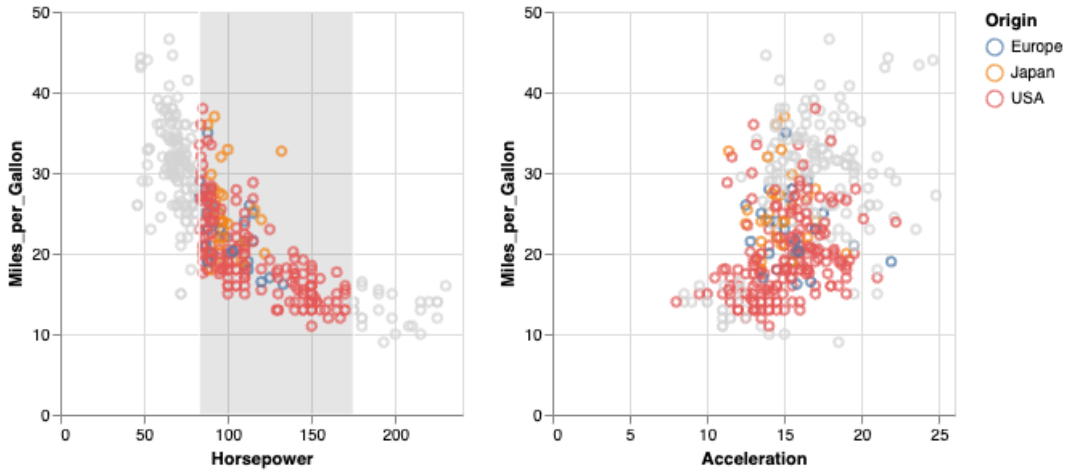
Linked Conditions (2)

```
chart = alt.Chart(cars).mark_point().encode(
    x='Horsepower:Q',
    y='Miles_per_Gallon:Q',
    color=alt.when(brush).then("Origin:N").otherwise(alt.value("lightgray")),
).properties(
    width=250,
    height=250
).add_params(
    brush
)

chart | chart.encode(x='Acceleration:Q')
```

Selection Types

Each selection type can be customized with additional properties to control its behavior.

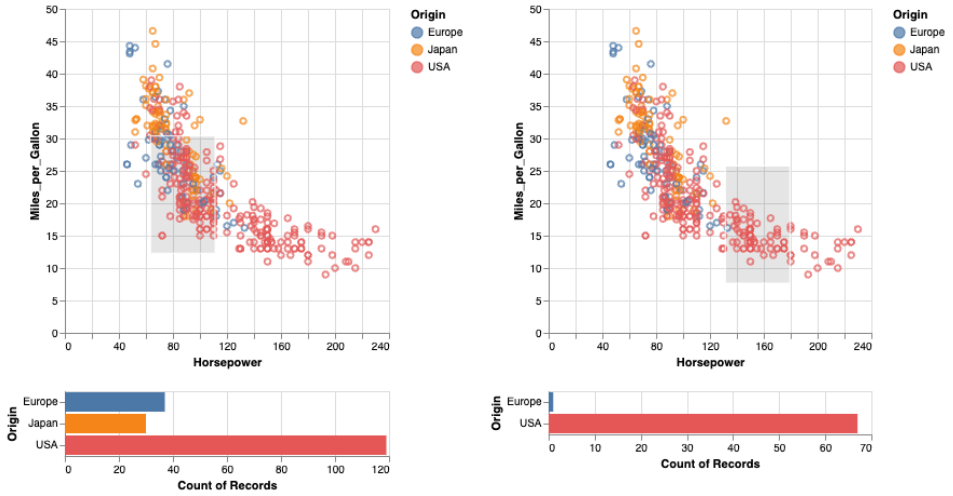
```
brush = alt.selection_interval(encodings=['x'])

chart = alt.Chart(cars).mark_point().encode(
    x='Horsepower:Q',
    y='Miles_per_Gallon:Q',
    color=alt.when(brush).then("Origin:N").otherwise(alt.value("lightgray")),
).properties(
    width=250,
    height=250
).add_params(
    brush
)

chart | chart.encode(x='Acceleration:Q')
```



Filters

A selection parameter can be used to filter the data in the chart. This allows for interactive filtering of the data based on user input.

```
brush = alt.selection_interval()

points = alt.Chart(cars).mark_point().encode(
    x='Horsepower:Q',
    y='Miles_per_Gallon:Q',
    color='Origin:N'
).add_params(
    brush
)

bars = alt.Chart(cars).mark_bar().encode(
    x='count()',
    y='Origin:N',
    color='Origin:N'
).transform_filter(
    brush
```



Takeaways

- **Data**: Altair uses tabular data as its basic data model. The data to be visualized is passed to the Chart object as a Pandas DataFrame.
- **Encodings and Marks**: The visual appearance of the data is specified using encodings and marks. Encodings map data fields to visual properties, such as position, color, size, and shape. Marks are the basic building blocks of a visualization, such as points, lines, bars, and areas.
- **Data Transformation**: Altair provides a number of methods for transforming the data before it is visualized. These methods can be used to filter, aggregate, and sort the data.
- **Interaction**: Altair provides a number of methods for adding interactivity to the visualization. These methods can be used to add tooltips, zooming, panning, and other interactive features to the visualization.