Interaction

“A graphic is not ‘drawn’ once and for all; it is ‘constructed’ and reconstructed until it reveals all the relationships constituted by the interplay of the data. The best graphic operations are those carried out by the decision-maker themself.” — Jacques Bertin

Visualization provides a powerful means of making sense of data. A single image, however, typically provides answers to, at best, a handful of questions. Through interaction we can transform static images into tools for exploration: highlighting points of interest, zooming in to reveal finer-grained patterns, and linking across multiple views to reason about multi-dimensional relationships.

At the core of interaction is the notion of a selection: a means of indicating to the computer which elements or regions we are interested in. For example, we might hover the mouse over a point, click multiple marks, or draw a bounding box around a region to highlight subsets of the data for further scrutiny.

Alongside visual encodings and data transformations, Altair provides a selection abstraction for authoring interactions. These selections encompass three aspects:

Input event handling to select points or regions of interest, such as mouse hover, click, drag, scroll, and touch events. Generalizing from the input to form a selection rule (or predicate) that determines whether or not a given data record lies within the selection. Using the selection predicate to dynamically configure a visualization by driving conditional encodings, filter transforms, or scale domains. This notebook introduces interactive selections and explores how to use them to author a variety of interaction techniques, such as dynamic queries, panning & zooming, details-on-demand, and brushing & linking.

This notebook is part of the data visualization curriculum.

In [1]:
```
import pandas as pd
import altair as alt
```

Datasets

We will visualize a variety of datasets from the vega-datasets collection:

A dataset of cars from the 1970s and early 1980s, A dataset of movies, previously used in the Data Transformation notebook, A dataset containing ten years of S&P 500 (sp500) stock prices, A dataset of technology company stocks, and A dataset of flights, including departure time, distance, and arrival delay.!

In [2]:
```
cars = 'https://cdn.jsdelivr.net/npm/vega-datasets@1/data/cars.json'
movies = 'https://cdn.jsdelivr.net/npm/vega-datasets@1/data/movies.json
sp500 = 'https://cdn.jsdelivr.net/npm/vega-datasets@1/data/sp500.csv'
stocks = 'https://cdn.jsdelivr.net/npm/vega-datasets@1/data/stocks.csv'
flights = 'https://cdn.jsdelivr.net/npm/vega-datasets@1/data/flights-5k
```

In [17]:
```
df_cars = pd.read_json(cars) # load cars data #testing
df_cars
```

Out[17]:

	Name	Miles_per_Gallon	Cylinders	Displacement	Horsepower	Weight_in_lbs	Acceleration
0	chevrolet chevelle malibu	18.0	8	307.0	130.0	3504	12.0
1	buick skylark 320	15.0	8	350.0	165.0	3693	11.5
2	plymouth satellite	18.0	8	318.0	150.0	3436	11.0
3	amc rebel sst	16.0	8	304.0	150.0	3433	12.0
4	ford torino	17.0	8	302.0	140.0	3449	10.5
...	...	...	...	...	...	...	...
401	ford mustang gl	27.0	4	140.0	86.0	2790	15.6
402	vw pickup	44.0	4	97.0	52.0	2130	24.6
403	dodge rampage	32.0	4	135.0	84.0	2295	11.6
404	ford ranger	28.0	4	120.0	79.0	2625	18.6
405	chevy s-10	31.0	4	119.0	82.0	2720	19.4

406 rows × 9 columns

Introducing Selections

Let's start with a basic selection: simply clicking a point to highlight it. Using the cars dataset, we'll start with a scatter plot of horsepower versus miles per gallon, with a color encoding for the number cylinders in the car engine.

In addition, we'll create a selection instance by calling alt.selection_single(), indicating we want a selection defined over a single value. By default, the selection uses a mouse click to determine the selected value. To register a selection with a chart, we must add it using the .add_selection() method.

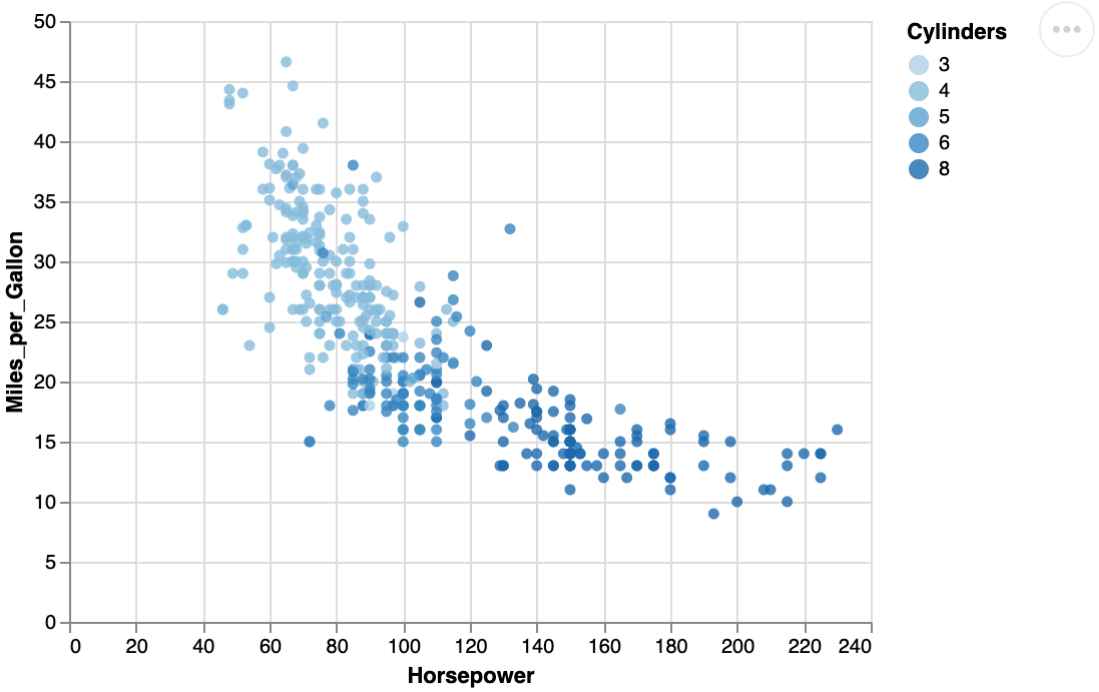
Once our selection has been defined, we can use it as a parameter for conditional encodings, which apply a different encoding depending on whether a data record lies in or out of the selection. For example, consider the following code:

color=alt.condition(selection, 'Cylinders:O', alt.value('grey')) This encoding definition states that data points contained within the selection should be colored according to the Cylinder field, while non-selected data points should use a default grey. An empty selection includes all data points, and so initially all points will be colored.

Try clicking different points in the chart below. What happens? (Click the background to clear the selection state and return to an "empty" selection.)

In [6]:
```
selection = alt.selection_single();

alt.Chart(cars).mark_circle().add_selection(
    selection
).encode(
    x='Horsepower:Q',
    y='Miles_per_Gallon:Q',
    color=alt.condition(selection, 'Cylinders:O', alt.value('grey')),
    opacity=alt.condition(selection, alt.value(0.8), alt.value(0.1))
)
```

Out[6]:



Of course, highlighting individual data points one-at-a-time is not particularly exciting! As we'll see, however, single value selections provide a useful building block for more powerful interactions. Moreover, single value selections are just one of the three selection types provided by Altair:

selection_single - select a single discrete value, by default on click events. selection_multi - select multiple discrete values. The first value is selected on mouse click and additional values toggled using shift-click. selection_interval - select a continuous range of values, initiated by mouse drag.

Let's compare each of these selection types side-by-side. To keep our code tidy we'll first define a function (plot) that generates a scatter plot specification just like the one above. We can pass a selection to the plot function to have it applied to the chart

In [9]:
```
def plot(selection):
    return alt.Chart(cars).mark_circle().add_selection(
        selection
    ).encode(
        x='Horsepower:Q',
        y='Miles_per_Gallon:Q',
        color=alt.condition(selection, 'Cylinders:O', alt.value('grey')),
        opacity=alt.condition(selection, alt.value(0.8), alt.value(0.1))
    ).properties(
        width=240,
        height=180
    )
```

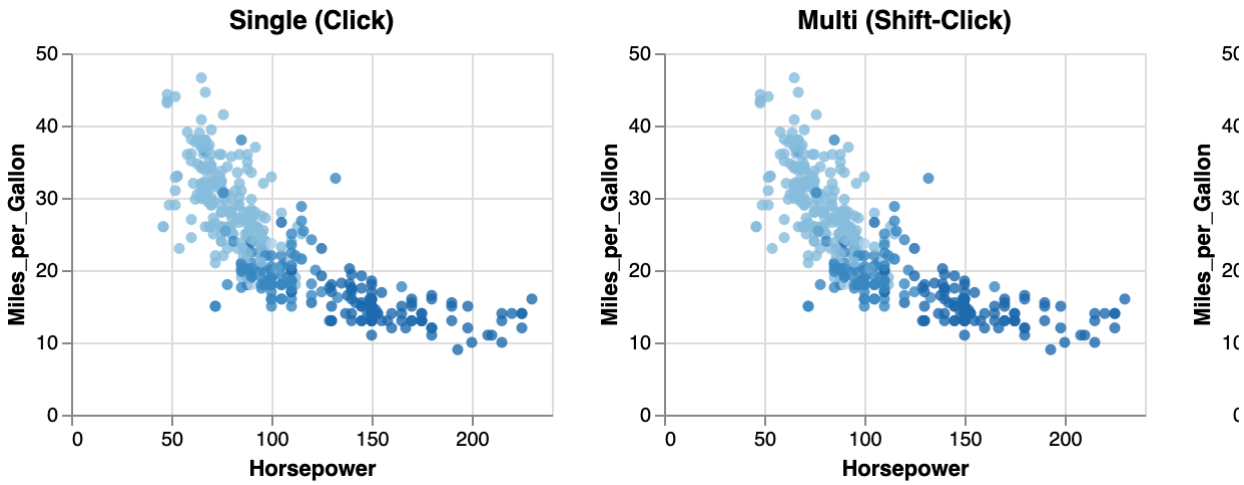
Let's use our plot function to create three chart variants, one per selection type.

The first (single) chart replicates our earlier example. The second (multi) chart supports shift-click interactions to toggle inclusion of multiple points within the selection. The third (interval) chart generates a selection region (or brush) upon mouse drag. Once created, you can drag the brush around to select different points, or scroll when the cursor is inside the brush to scale (zoom) the brush size.

Try interacting with each of the charts below!

In [10]:
```
alt.hconcat(
  plot(alt.selection_single()).properties(title='Single (Click)'),
  plot(alt.selection_multi()).properties(title='Multi (Shift-Click)'),
  plot(alt.selection_interval()).properties(title='Interval (Drag)')
)
```

Out[10]:

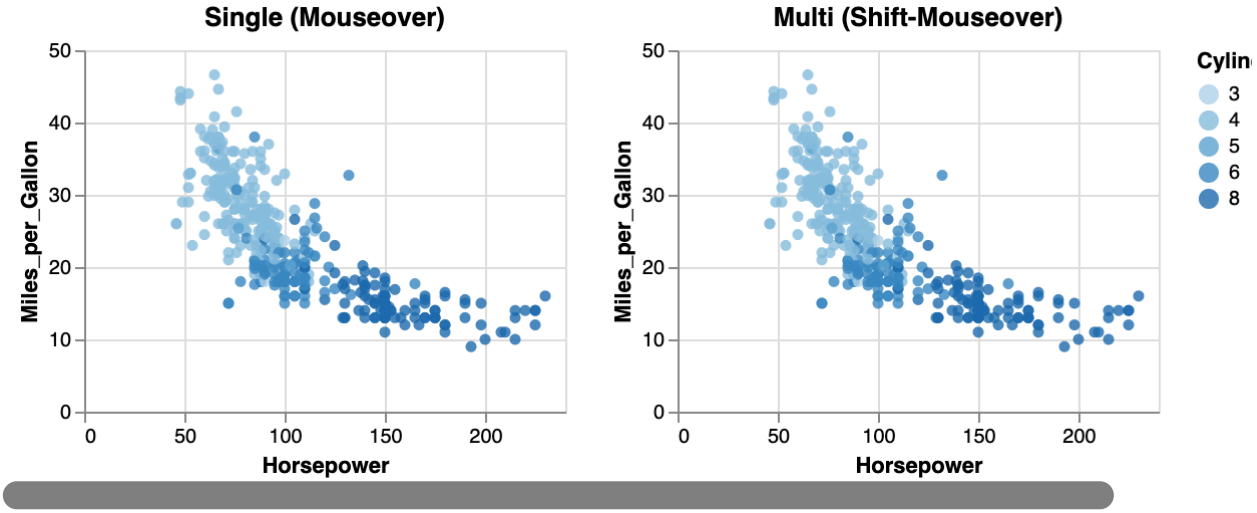


The examples above use default interactions (click, shift-click, drag) for each selection type. We can further customize the interactions by providing input event specifications using Vega event selector syntax. For example, we can modify our single and multi charts to trigger upon mouseover events instead of click events.

Hold down the shift key in the second chart to "paint" with data!

In [11]:
```
alt.hconcat(
  plot(alt.selection_single(on='mouseover')).properties(title='Single (
  plot(alt.selection_multi(on='mouseover')).properties(title='Multi (Sh
)
```

Out[11]:



Now that we've covered the basics of Altair selections, let's take a tour through the various interaction techniques they enable!

Dynamic Queries

Dynamic queries enables rapid, reversible exploration of data to isolate patterns of interest. As defined by Ahlberg, Williamson, & Shneiderman, a dynamic query:

*represents a query graphically, *provides visible limits on the query range, *provides a graphical representation of the data and query result, *gives immediate feedback of the result after every query adjustment, *and allows novice users to begin working with little training.

A common approach is to manipulate query parameters using standard user interface widgets such as sliders, radio buttons, and drop-down menus. To generate dynamic query widgets, we can apply a selection's bind operation to one or more data fields we wish to query.

Let's build an interactive scatter plot that uses a dynamic query to filter the display. Given a scatter plot of movie ratings (from Rotten Tomates and IMDB), we can add a selection over the Major_Genre field to enable interactive filtering by film genre.

To start, let's extract the unique (non-null) genres from the movies data:

In [12]:
```
pd.read_json(movies) # load movies data
es = df['Major_Genre'].unique() # get unique field values
es = list(filter(lambda d: d is not None, genres)) # filter out None values
es.sort() # sort alphabetically
```

In [16]:
```
1 df
```

Out[16]:

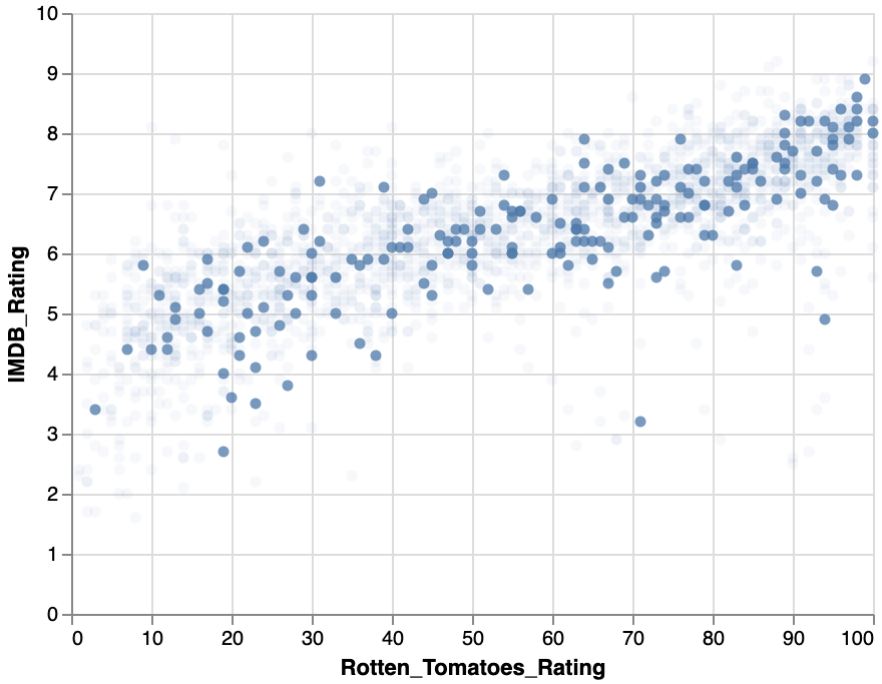
lease_Date	MPAA_Rating	Running_Time_min	Distributor	Source	Major_Genre	Creative_Ty
Jun 12 1998	R	NaN	Gramercy	None	None	No
Aug 07 1998	R	NaN	Strand	None	Drama	No
Aug 28 1998	None	NaN	Lionsgate	None	Comedy	No
Sep 11 1998	None	NaN	Fine Line	None	Comedy	No
Oct 09 1998	R	NaN	Trimark	Original Screenplay	Drama	Contempora Ficti
...	...	...	...	...	...	
Oct 31 2008	R	101.0	Weinstein Co.	Original Screenplay	Comedy	Contempora Ficti
Mar 02 2007	R	157.0	Paramount Pictures	Based on Book/Short Story	Thriller/Suspense	Dramatizati
Aug 11 2006	PG	NaN	Sony Pictures	Based on Comic/Graphic Novel	Adventure	Super He
Oct 28 2005	PG	129.0	Sony Pictures	Remake	Adventure	Histori Ficti
Jul 17 1998	PG-13	136.0	Sony Pictures	Remake	Adventure	Histori Ficti

Now let's create a single selection bound to a drop-down menu.

Use the dynamic query menu below to explore the data. How do ratings vary by genre? How would you revise the code to filter MPAA_Rating (G, PG, PG-13, etc.) instead of Major_Genre?

In [13]:
```
nre = alt.selection_single(
='Select', # name the selection 'Select'
ds=['Major_Genre'], # limit selection to the Major_Genre field
={'Major_Genre': genres[0]}, # use first genre entry as initial value
=alt.binding_select(options=genres) # bind to a menu of unique genre values


t(movies).mark_circle().add_selection(
ctGenre
(
otten_Tomatoes_Rating:Q',
MDB_Rating:Q',
tip='Title:N',
ity=alt.condition(selectGenre, alt.value(0.75), alt.value(0.05))
```

Out[13]:



Select_Major_Genre Adventure

Our construction above leverages multiple aspects of selections:

*We give the selection a name ('Select'). This name is not required, but allows us to influence the label text of the generated dynamic query menu. (What happens if you remove the name? Try it!)

*We constrain the selection to a specific data field (Major_Genre). Earlier when we used a single selection, the selection mapped to individual data points. By limiting the selection to a specific field, we can select all data points whose Major_Genre field value matches the single selected value.

*We initialize init=... the selection to a starting value.

*We bind the selection to an interface widget, in this case a drop-down menu via binding_select.

*As before, we then use a conditional encoding to control the opacity channel.

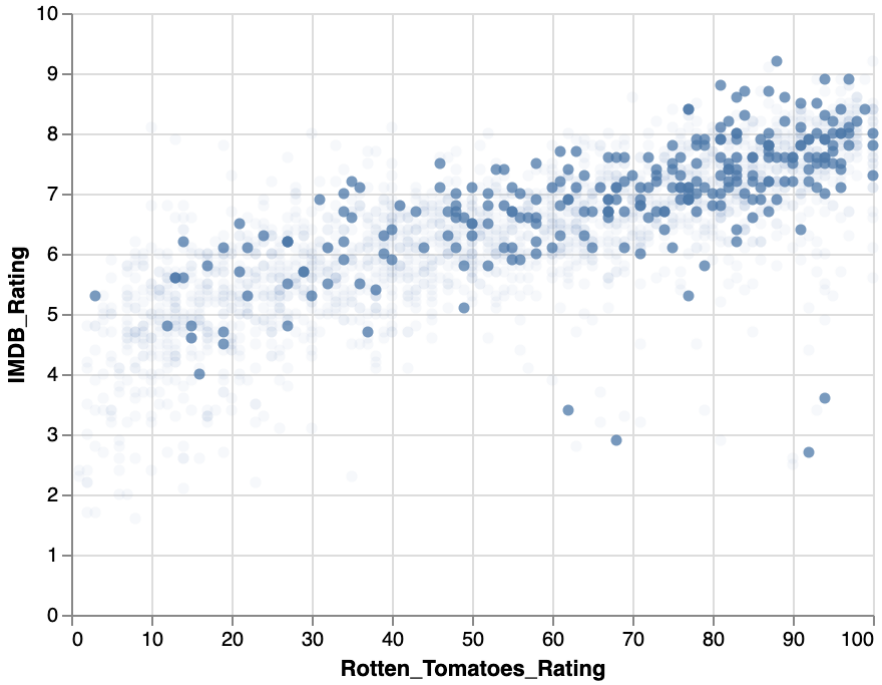
Binding Selections to Multiple Inputs

One selection instance can be bound to multiple dynamic query widgets. Let's modify the example above to provide filters for both Major_Genre and MPAA_Rating, using radio buttons instead of a menu. Our single selection is now defined over a single pair of genre and MPAA rating values

Look for surprising conjunctions of genre and rating. Are there any G or PG-rated horror films?

In [23]:
```
jor_Genre, MPAA_Rating] pairs
as the initial selected values
 # get unique field values
 not None, genres)) # filter out None values
ly



Rating'],
 'MPAA_Rating': 'R'},
ing_select(options=genres), 'MPAA_Rating': alt.binding_radio(options=mpaa)}


sed on selection
dd_selection(




on, alt.value(0.75), alt.value(0.05))
```

Out[23]:



Select_MPAA_Rating ○Action ○Adventure ○Black Comedy ○Comedy ○Concert/Performance ○Documentary ○Drama ○Horror ○Musical ○Romantic Comedy ○Thriller/Suspense ○Western

Select_Major_Genre Drama

Fun facts: The PG-13 rating didn't exist when the movies Jaws and Jaws 2 were released. The first film to receive a PG-13 rating was 1984's Red Dawn.

Using Visualizations as Dynamic Queries

Though standard interface widgets show the possible query parameter values, they do not visualize the distribution of those values. We might also wish to use richer interactions, such as multi-value or interval selections, rather than input widgets that select only a single value at a time.

To address these issues, we can author additional charts to both visualize data and support dynamic queries. Let's add a histogram of the count of films per year and use an interval selection to dynamically highlight films over selected time periods.

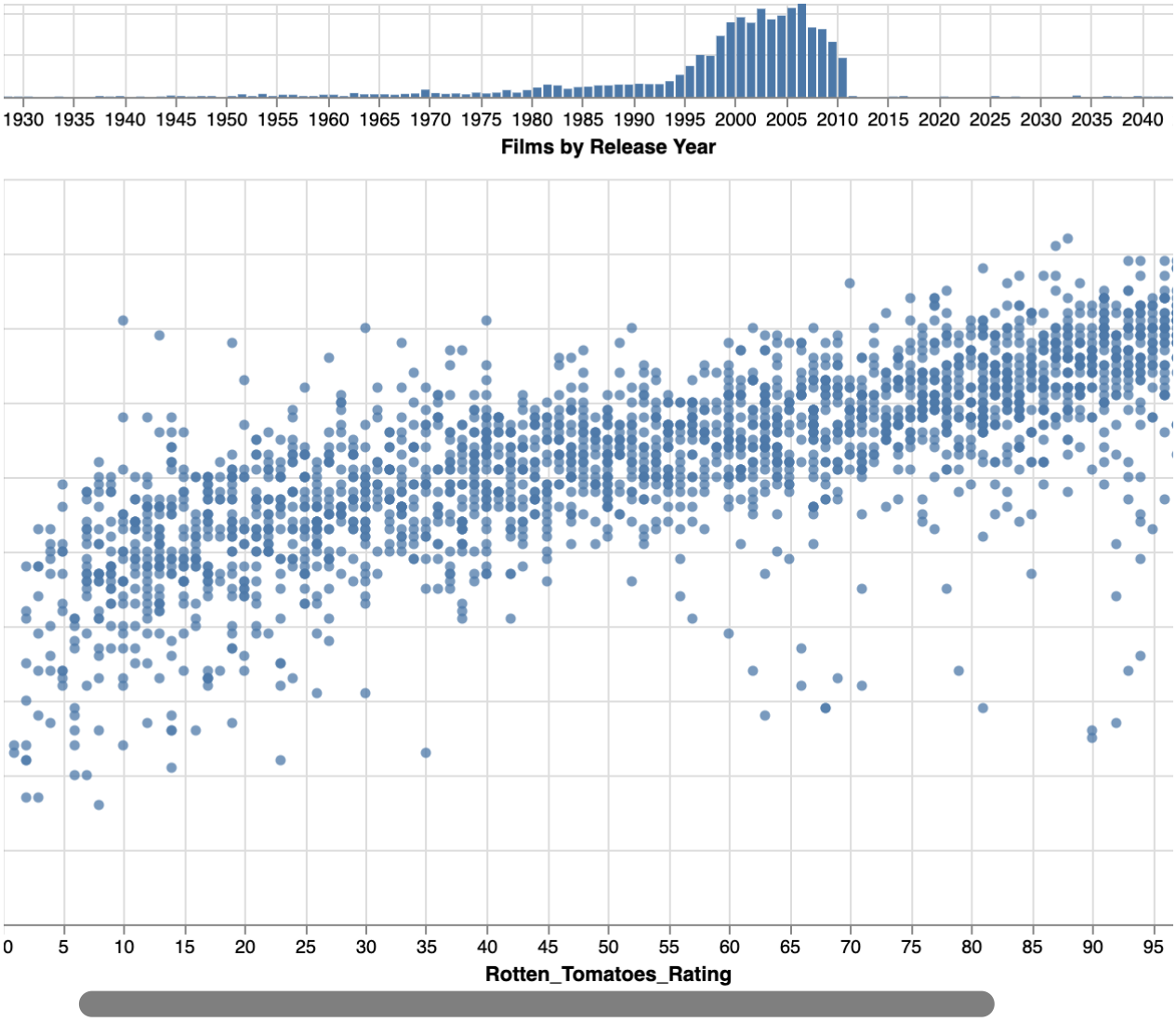
Interact with the year histogram to explore films from different time periods. Do you seen any evidence of sampling bias across the years? (How do year and critics' ratings relate?)

The years range from 1930 to 2040! Are future films in pre-production, or are there "off-by-one century" errors? Also, depending on which time zone you're in, you may see a small bump in either 1969 or 1970. Why might that be? (See the end of the notebook for an explanation!)

In [25]:

```
brush = alt.selection_interval(
    encodings=['x'] # limit selection to x-axis (year) values
)

# dynamic query histogram
years = alt.Chart(movies).mark_bar().add_selection(
    brush
).encode(
    alt.X('year(Release_Date):T', title='Films by Release Year'),
    alt.Y('count():Q', title=None)
).properties(
    width=650,
    height=50
)

# scatter plot, modify opacity based on selection
ratings = alt.Chart(movies).mark_circle().encode(
    x='Rotten_Tomatoes_Rating:Q',
    y='IMDB_Rating:Q',
    tooltip='Title:N',
    opacity=alt.condition(brush, alt.value(0.75), alt.value(0.05))
).properties(
    width=650,
    height=400
)

alt.vconcat(years, ratings).properties(spacing=5)

```

Out[25]:



The example above provides dynamic queries using a linked selection between charts:

We create an interval selection (brush), and set encodings=['x'] to limit the selection to the x-axis only, resulting in a one-dimensional selection interval. We register brush with our histogram of films per year via .add_selection(brush). We use brush in a conditional encoding to adjust the scatter plot opacity. This interaction technique of selecting elements in one chart and seeing linked highlights in one or more other charts is known as brushing & linking.

Panning & Zooming

The movie rating scatter plot is a bit cluttered in places, making it hard to examine points in denser regions. Using the interaction techniques of panning and zooming, we can inspect dense regions more closely.

Let's start by thinking about how we might express panning and zooming using Altair selections. What defines the "viewport" of a chart? Axis scale domains!

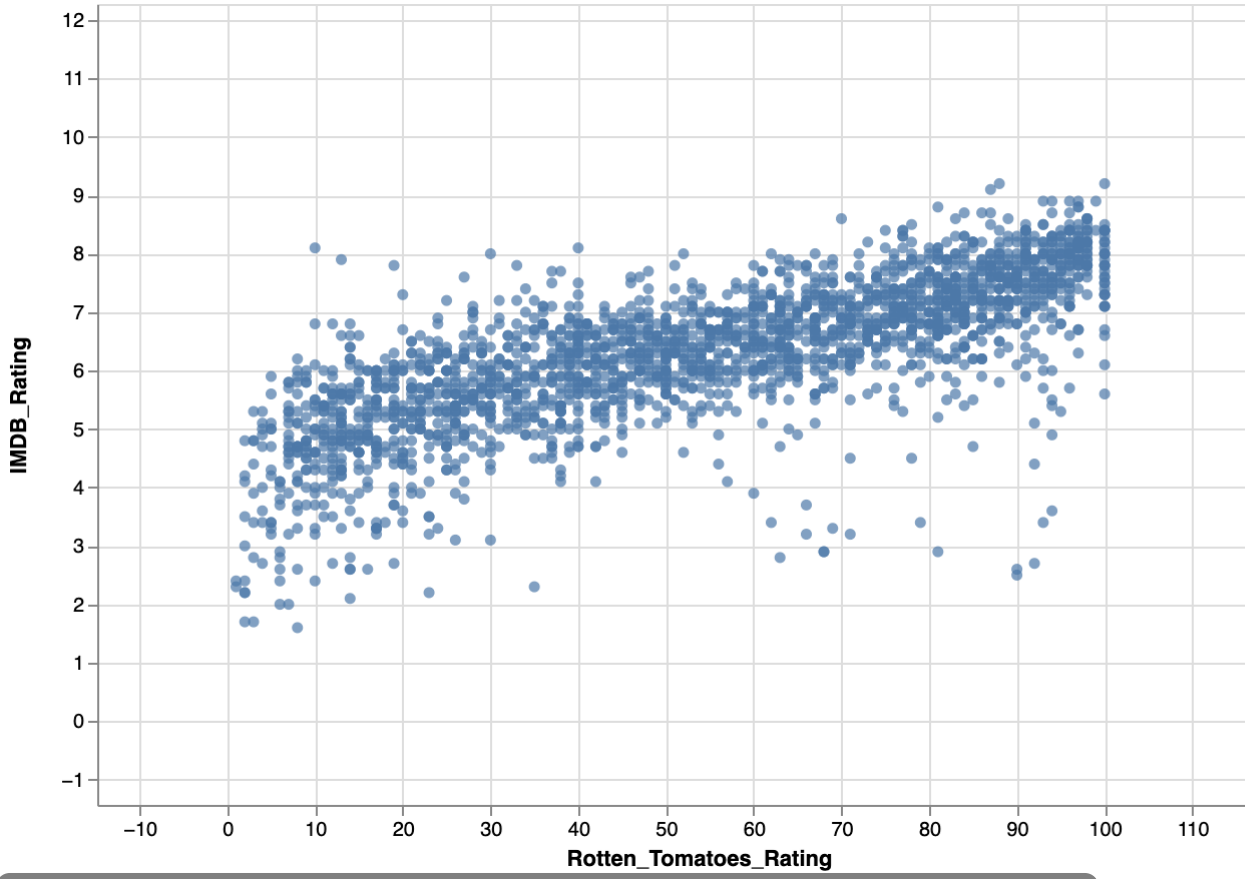
We can change the scale domains to modify the visualized range of data values. To do so interactively, we can bind an interval selection to scale domains with the code bind='scales'. The result is that instead of an interval brush that we can drag and zoom, we instead can drag and

zoom the entire plotting area!

In the chart below, click and drag to pan (translate) the view, or scroll to zoom (scale) the view. What can you discover about the precision of the provided rating values?

In [26]:
```
alt.Chart(movies).mark_circle().add_selection(
    alt.selection_interval(bind='scales')
).encode(
    x='Rotten_Tomatoes_Rating:Q',
    y=alt.Y('IMDB_Rating:Q', axis=alt.Axis(minExtent=30)), # use min ex
    tooltip=['Title:N', 'Release_Date:N', 'IMDB_Rating:Q', 'Rotten_Toma
).properties(
    width=600,
    height=400
)
```

Out[26]:



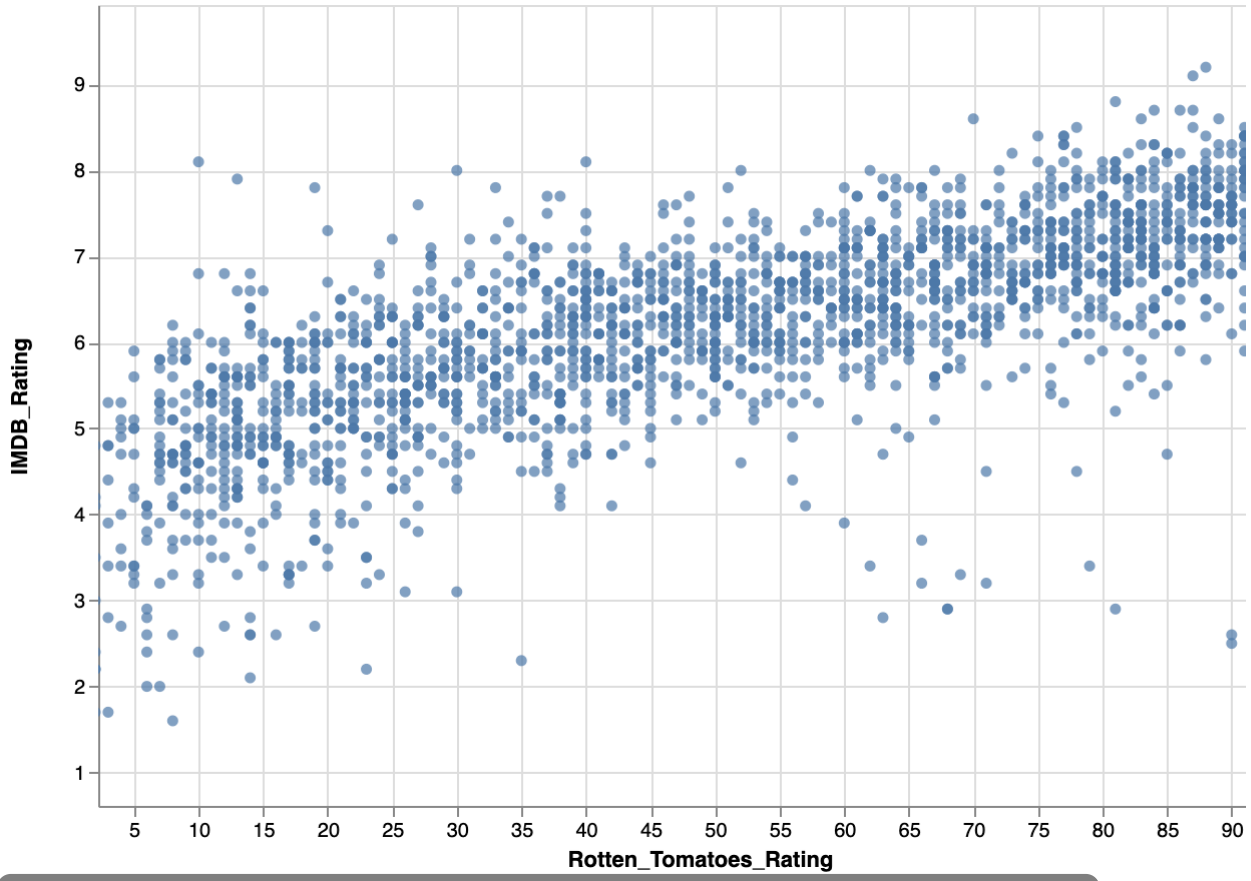
Zooming in, we can see that the rating values have limited precision! The Rotten Tomatoes ratings are integers, while the IMDB ratings are truncated to tenths. As a result, there is overplotting even when we zoom, with multiple movies sharing the same rating values.

Reading the code above, you may notice the code alt.Axis(minExtent=30) in the y encoding channel. The minExtent parameter ensures a minimum amount of space is reserved for axis ticks and labels. Why do this? When we pan and zoom, the axis labels may change and cause the axis title position to shift. By setting a minimum extent we can reduce distracting movements in the plot. Try changing the minExtent value, for example setting it to zero, and then zoom out to see what happens when longer axis labels enter the view.

Altair also includes a shorthand for adding panning and zooming to a plot. Instead of directly creating a selection, you can call .interactive() to have Altair automatically generate an interval selection bound to the chart's scales:

In [27]:
```
hart(movies).mark_circle().encode(
='Rotten_Tomatoes_Rating:Q',
=alt.Y('IMDB_Rating:Q', axis=alt.Axis(minExtent=30)), # use min extent to st
ooltip=['Title:N', 'Release_Date:N', 'IMDB_Rating:Q', 'Rotten_Tomatoes_Ratir
perties(
idth=600,
eight=400
eractive()
```

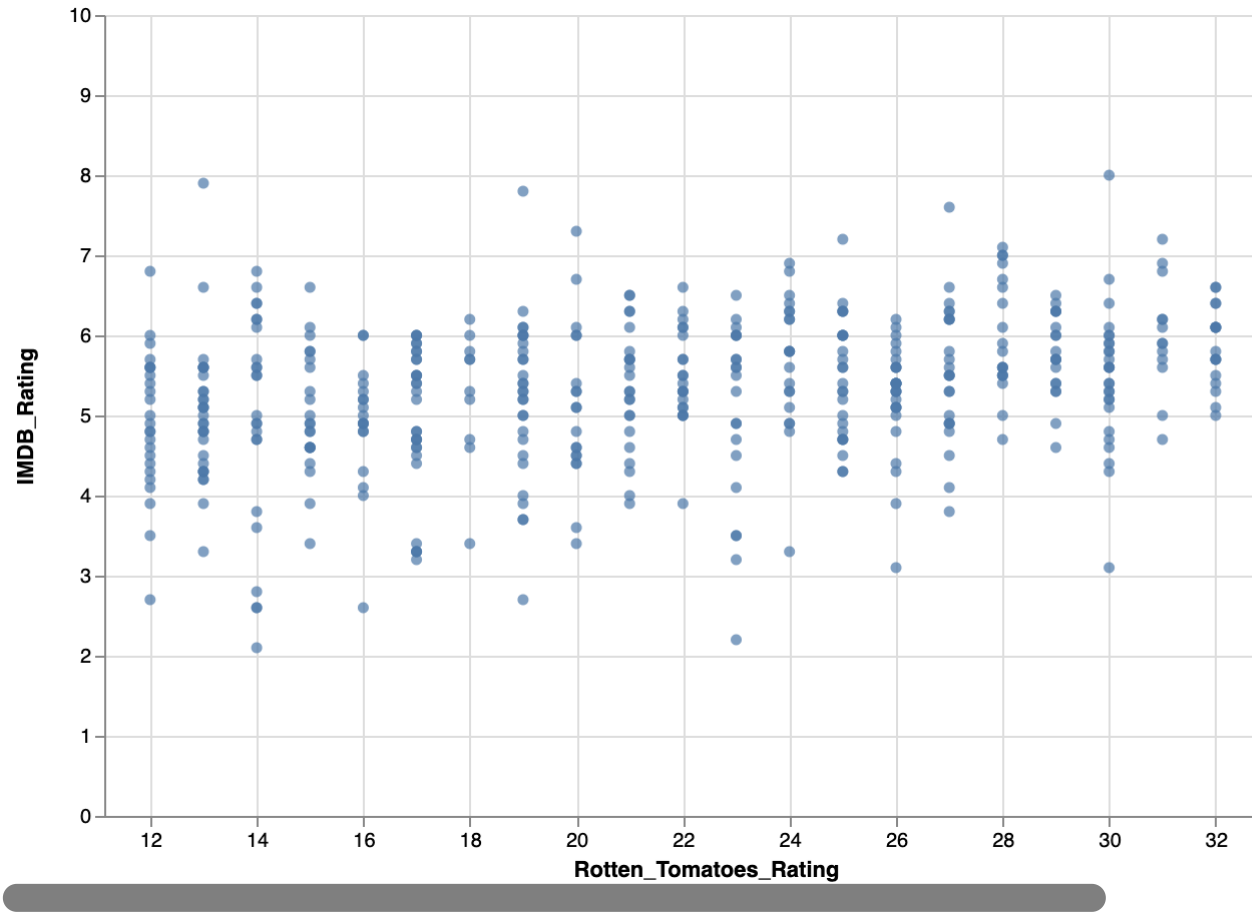
Out[27]:



By default, scale bindings for selections include both the x and y encoding channels. What if we want to limit panning and zooming along a single dimension? We can invoke encodings=['x'] to constrain the selection to the x channel only:

In [28]:
```
hart(movies).mark_circle().add_selection(
lt.selection_interval(bind='scales', encodings=['x'])
ode(
='Rotten_Tomatoes_Rating:Q',
=alt.Y('IMDB_Rating:Q', axis=alt.Axis(minExtent=30)), # use min extent to st
ooltip=['Title:N', 'Release_Date:N', 'IMDB_Rating:Q', 'Rotten_Tomatoes_Ratir
perties(
idth=600,
eight=400
```

Out[28]:



When zooming along a single axis only, the shape of the visualized data can change, potentially affecting our perception of relationships in the data. Choosing an appropriate aspect ratio is an important visualization design concern!

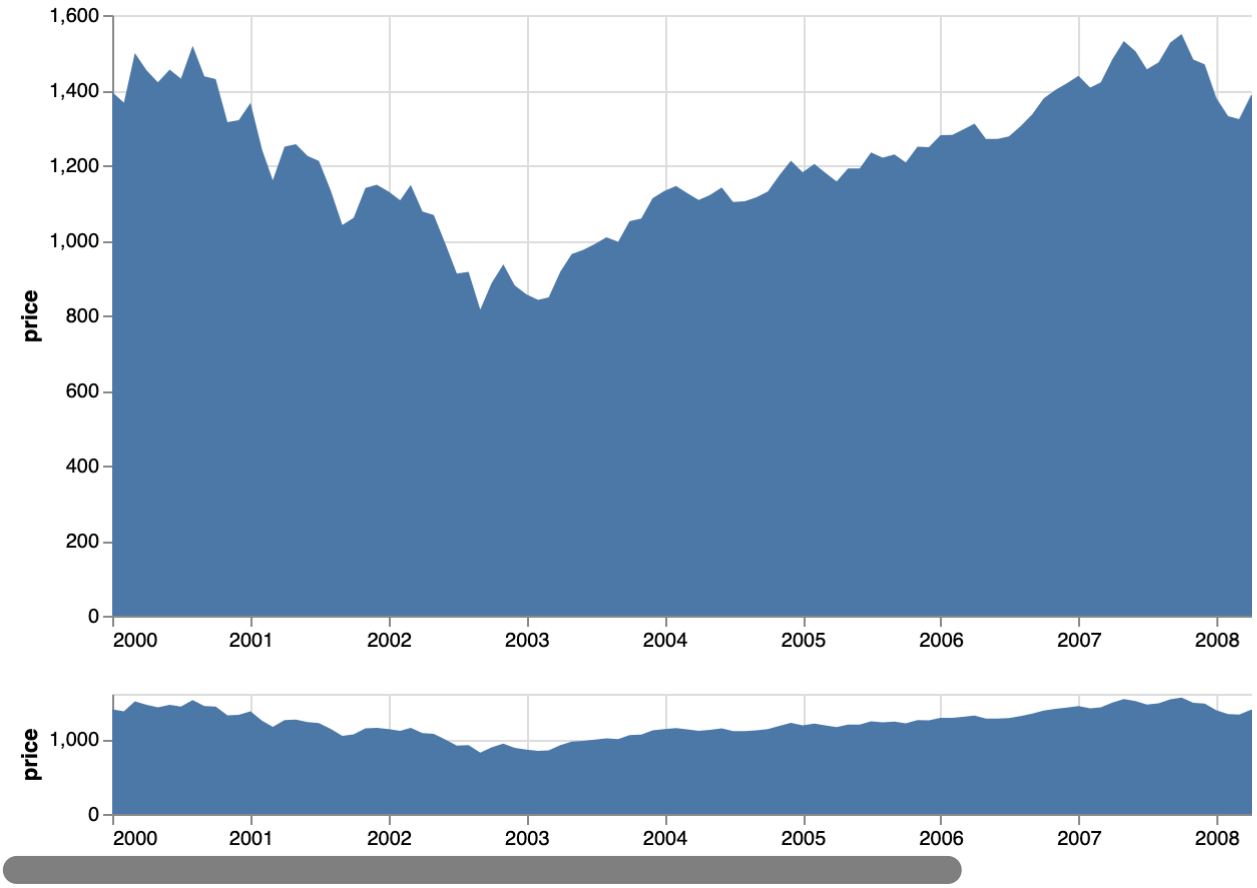
Navigation: Overview + Detail

When panning and zooming, we directly adjust the "viewport" of a chart. The related navigation strategy of overview + detail instead uses an overview display to show all of the data, while supporting selections that pan and zoom a separate focus display.

Below we have two area charts showing a decade of price fluctuations for the S&P 500 stock index. Initially both charts show the same data range. Click and drag in the bottom overview chart to update the focus display and examine specific time spans.

In [29]:
```
rush = alt.selection_interval(encodings=['x']);

ase = alt.Chart().mark_area().encode(
alt.X('date:T', title=None),
alt.Y('price:Q')
.properties(
width=700


lt.vconcat(
base.encode(alt.X('date:T', title=None, scale=alt.Scale(domain=brush))),
base.add_selection(brush).properties(height=60),
data=sp500


```

Out[29]:



Unlike our earlier panning & zooming case, here we don't want to bind a selection directly to the scales of a single interactive chart. Instead, we want to bind the selection to a scale domain in another chart. To do so, we update the x encoding channel for our focus chart, setting the scale domain property to reference our brush selection. If no interval is defined (the selection is empty), Altair ignores the brush and uses the underlying data to determine the domain. When a brush interval is created, Altair instead uses that as the scale domain for the focus chart.

Details on Demand

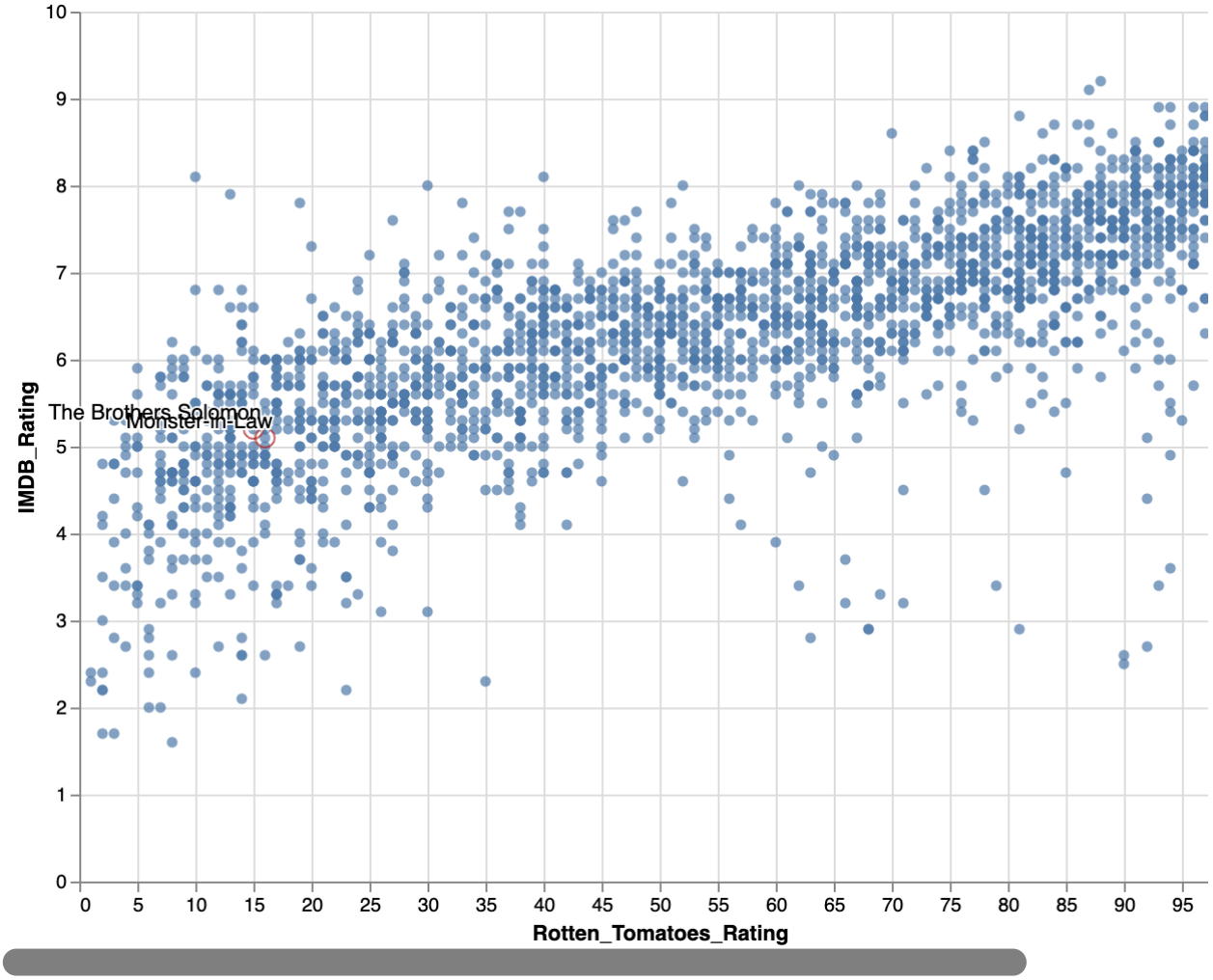
Once we spot points of interest within a visualization, we often want to know more about them. Details-on-demand refers to interactively querying for more information about selected values. Tooltips are one useful means of providing details on demand. However, tooltips typically only show information for one data point at a time. How might we show more?

The movie ratings scatterplot includes a number of potentially interesting outliers where the Rotten Tomatoes and IMDB ratings disagree. Let's create a plot that allows us to interactively select points and show their labels. To trigger the filter query on either the hover or click interaction, we will use the Altair composition operator | ("or").

Mouse over points in the scatter plot below to see a highlight and title label. Shift-click points to make annotations persistent and view multiple labels at once. Which movies are loved by Rotten Tomatoes critics, but not the general audience on IMDB (or vice versa)? See if you can find possible errors, where two different movies with the same name were accidentally combined!

In [30]:
```
.selection_single(
seover',  # select on mouseover
=True,    # select nearest point to mouse cursor
none'     # empty selection should match nothing


.selection_multi(
none' # empty selection matches no points


lot encodings shared by all marks
Chart().mark_circle().encode(
en_Tomatoes_Rating:Q',
_Rating:Q'


se for new layers
.transform_filter(
 click # filter to points in either selection


tter plot points, halo annotations, and title labels

d_selection(hover).add_selection(click),
rk_point(size=100, stroke='firebrick', strokeWidth=1),
rk_text(dx=4, dy=-8, align='right', stroke='white', strokeWidth=2).encode(te
rk_text(dx=4, dy=-8, align='right').encode(text='Title:N'),
vies
s(
00,
450
```

Out[30]:



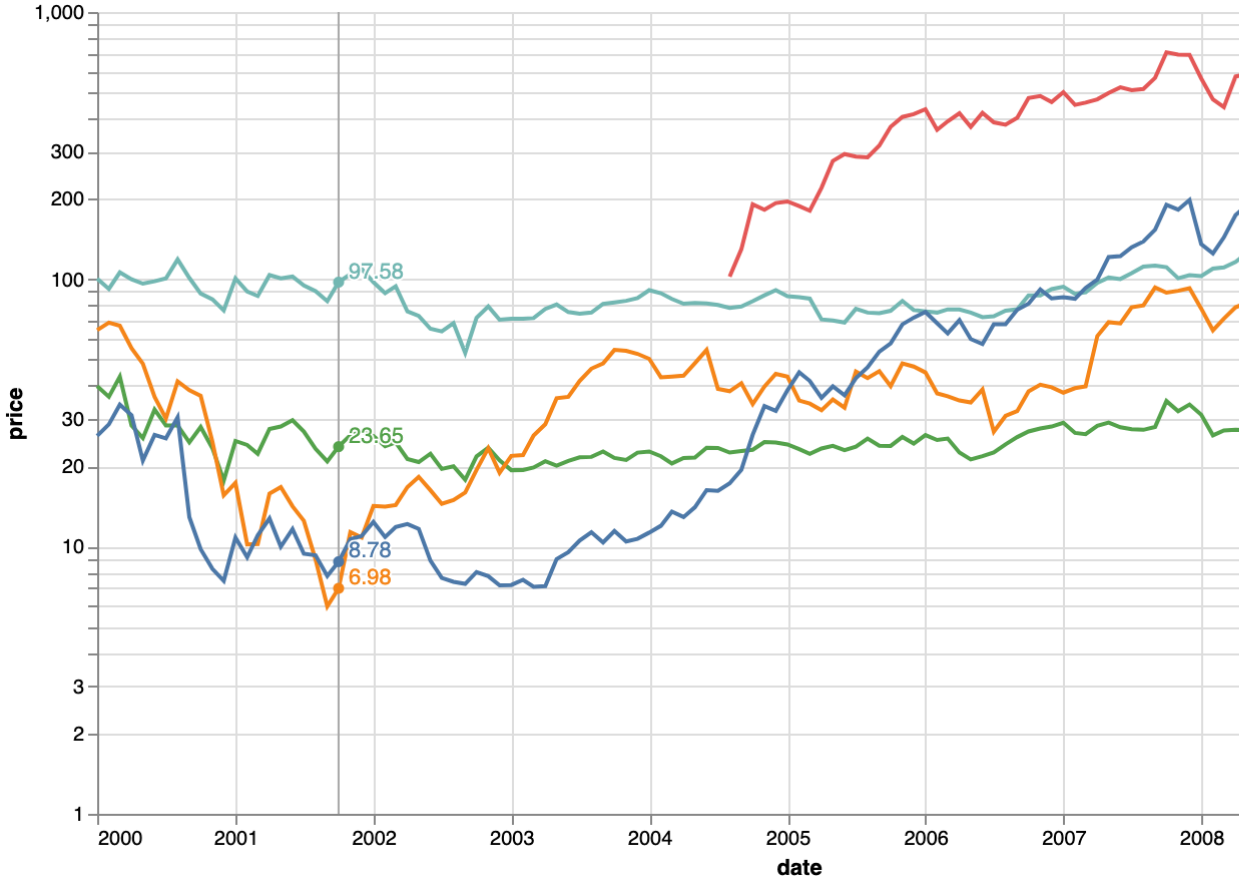
The example above adds three new layers to the scatter plot: a circular annotation, white text to provide a legible background, and black text showing a film title. In addition, this example uses two selections in tandem:

A single selection (hover) that includes nearest=True to automatically select the nearest data point as the mouse moves. A multi selection (click) to create persistent selections via shift-click. Both selections include the set empty='none' to indicate that no points should be included if a selection is empty. These selections are then combined into a single filter predicate — the logical or of hover and click — to include points that reside in either selection. We use this predicate to filter the new layers to show annotations and labels for selected points only.

Using selections and layers, we can realize a number of different designs for details on demand! For example, here is a log-scaled time series of technology stock prices, annotated with a guideline and labels for the date nearest the mouse cursor:

In [31]:
```
a point for which to provide details-on-demand
lt.selection_single(
ings=['x'], # limit selection to x-axis value
ouseover',  # select on mouseover events
st=True,    # select data point nearest the cursor
='none'     # empty selection includes no data points


our base line chart of stock prices
t.Chart().mark_line().encode(
('date:T'),
('price:Q', scale=alt.Scale(type='log')),
olor('symbol:N')


(
 # base line chart

 a rule mark to serve as a guide line
hart().mark_rule(color='#aaa').encode(
='date:T'
nsform_filter(label),

 circle marks for selected time points, hide unselected points
mark_circle().encode(
pacity=alt.condition(label, alt.value(1), alt.value(0))
_selection(label),

 white stroked text to provide a legible background for labels
mark_text(align='left', dx=5, dy=-5, stroke='white', strokeWidth=2).encode(
ext='price:Q'
nsform_filter(label),

 text labels for stock prices
mark_text(align='left', dx=5, dy=-5).encode(
ext='price:Q'
nsform_filter(label),

stocks
ies(
=700,
t=400
```

Out[31]:



Putting into action what we've learned so far: can you modify the movie scatter plot above (the one with the dynamic query over years) to include a rule mark that shows the average IMDB (or Rotten Tomatoes) rating for the data contained within the year interval selection?

Brushing & Linking, Revisited

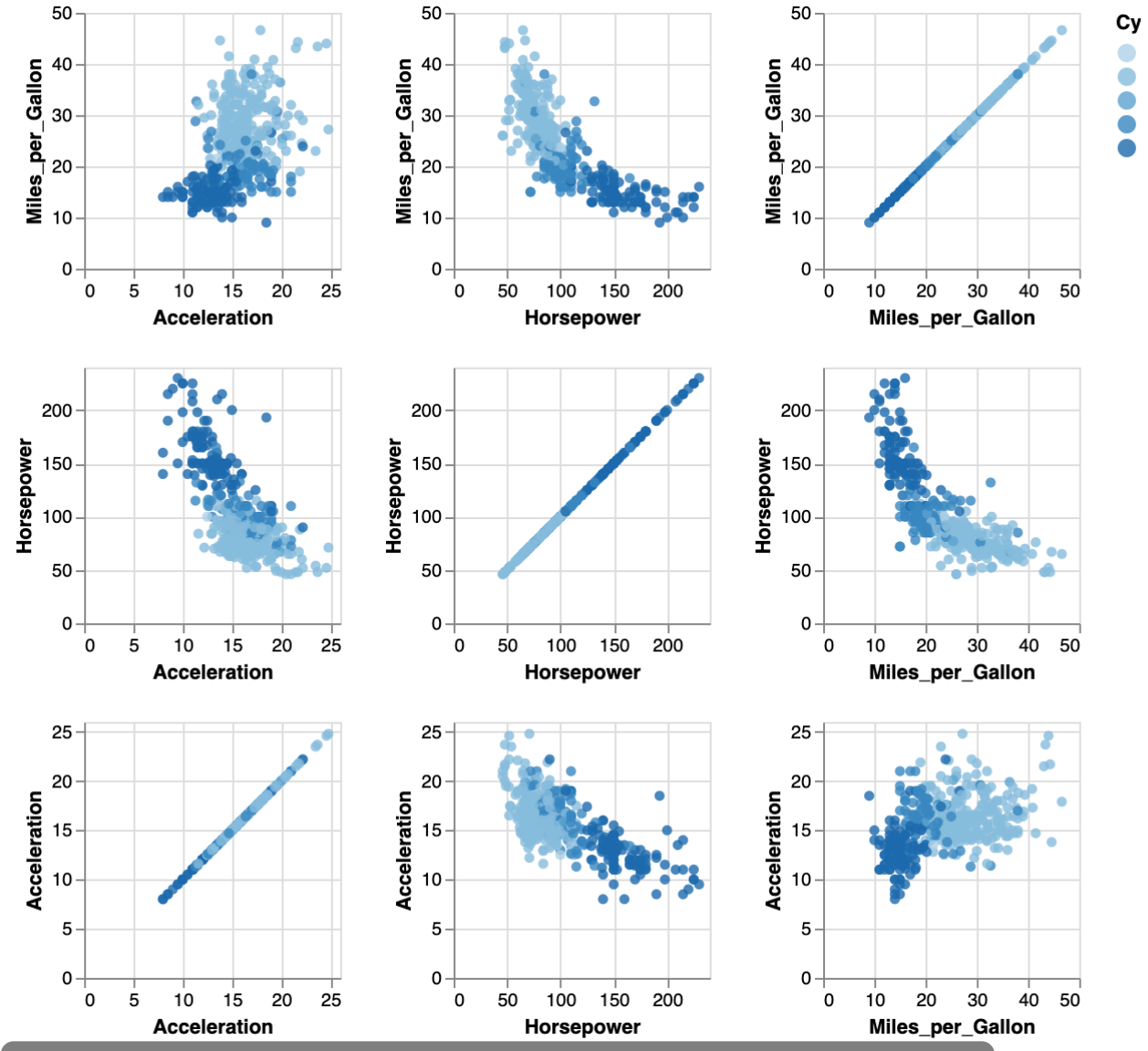
Earlier in this notebook we saw an example of brushing & linking: using a dynamic query histogram to highlight points in a movie rating scatter plot. Here, we'll visit some additional examples involving linked selections.

Returning to the cars dataset, we can use the repeat operator to build a scatter plot matrix (SPLOM) that shows associations between mileage, acceleration, and horsepower. We can define an interval selection and include it within our repeated scatter plot specification to enable linked selections among all the plots.

Click and drag in any of the plots below to perform brushing & linking!

In [32]:
```
brush = alt.selection_interval(
  resolve='global' # resolve all selections to a single global instance
)

alt.Chart(cars).mark_circle().add_selection(
  brush
).encode(
  alt.X(alt.repeat('column'), type='quantitative'),
  alt.Y(alt.repeat('row'), type='quantitative'),
  color=alt.condition(brush, 'Cylinders:O', alt.value('grey')),
  opacity=alt.condition(brush, alt.value(0.8), alt.value(0.1))
).properties(
  width=140,
  height=140
).repeat(
  column=['Acceleration', 'Horsepower', 'Miles_per_Gallon'],
  row=['Miles_per_Gallon', 'Horsepower', 'Acceleration']
)
```

Out[32]:



Note above the use of resolve='global' on the interval selection. The default setting of 'global'

indicates that across all plots only one brush can be active at a time. However, in some cases we might want to define brushes in multiple plots and combine the results. If we use resolve='union', the selection will be the union of all brushes: if a point resides within any brush it will be selected. Alternatively, if we use resolve='intersect', the selection will consist of the intersection of all brushes: only points that reside within all brushes will be selected.

Try setting the resolve parameter to 'union' and 'intersect' and see how it changes the resulting selection logic.

Cross-Filtering

The brushing & linking examples we've looked at all use conditional encodings, for example to change opacity values in response to a selection. Another option is to use a selection defined in one view to filter the content of another view.

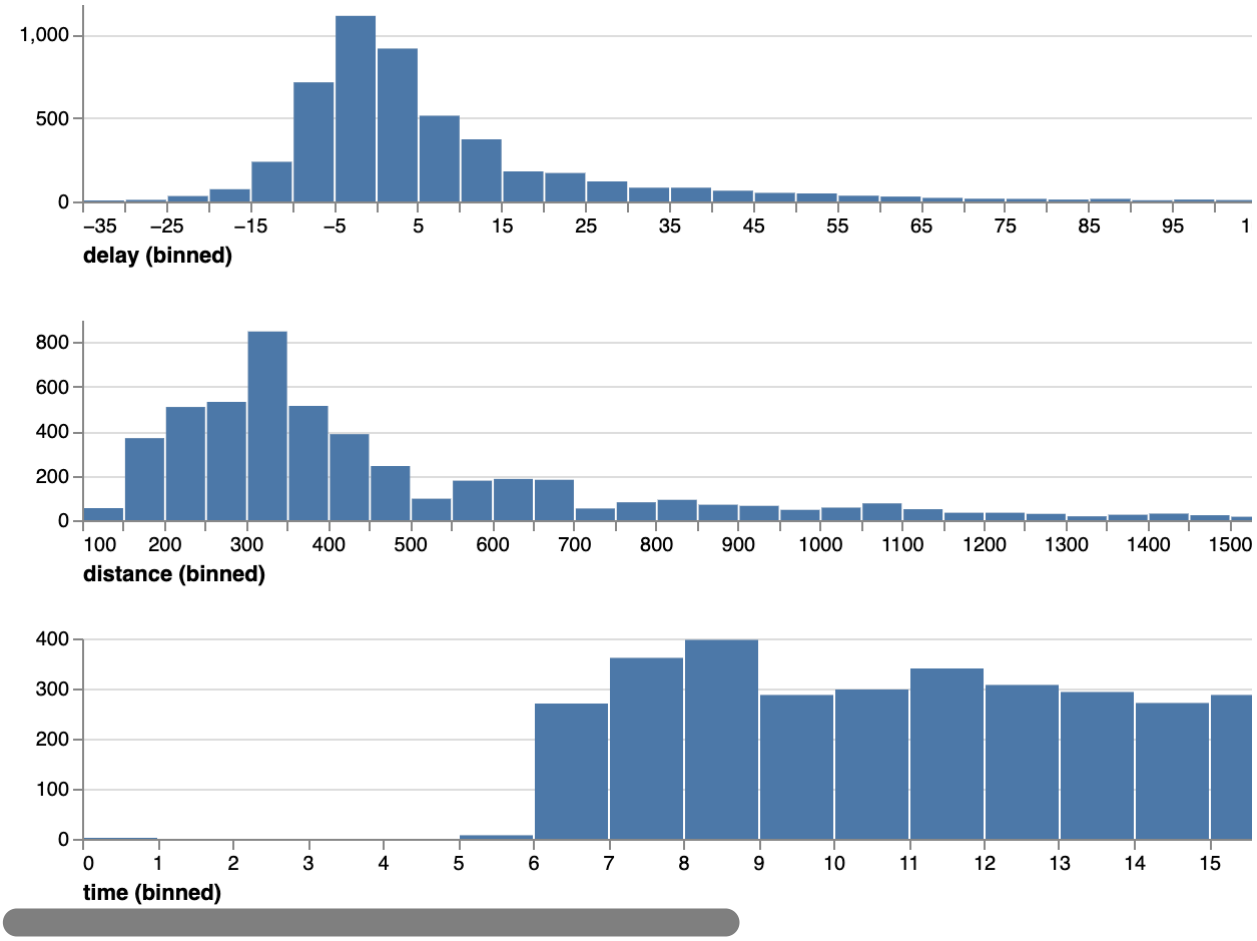
Let's build a collection of histograms for the flights dataset: arrival delay (how early or late a flight arrives, in minutes), distance flown (in miles), and time of departure (hour of the day). We'll use the repeat operator to create the histograms, and add an interval selection for the x axis with brushes resolved via intersection.

In particular, each histogram will consist of two layers: a gray background layer and a blue foreground layer, with the foreground layer filtered by our intersection of brush selections. The result is a cross-filtering interaction across the three charts!

Drag out brush intervals in the charts below. As you select flights with longer or shorter arrival delays, how do the distance and time distributions respond?

In [33]:
```
ction_interval(
x'],
ersect'


().mark_bar().encode(
peat('row'), type='quantitative',
Bin(maxbins=100, minstep=1), # up to 100 bins
.Axis(format='d', titleAnchor='start') # integer format, left-aligned title

():Q', title=None) # no y-axis title



ection(brush).encode(color=alt.value('lightgrey')),
rm_filter(brush)




, 'distance', 'time'],

ulate(
.delay < 180 ? datum.delay : 180', # clamp delays > 3 hours
datum.date) + minutes(datum.date) / 60' # fractional hours
'(
sparent' # no outline
```

Out[33]:



By cross-filtering you can observe that delayed flights are more likely to depart at later hours. This phenomenon is familiar to frequent fliers: a delay can propagate through the day, affecting subsequent travel by that plane. For the best odds of an on-time arrival, book an early flight!

The combination of multiple views and interactive selections can enable valuable forms of multi-dimensional reasoning, turning even basic histograms into powerful input devices for asking questions of a dataset!

Summary

For more information about the supported interaction options in Altair, please consult the Altair interactive selection documentation. For details about customizing event handlers, for example to compose multiple interaction techniques or support touch-based input on mobile devices, see the Vega-Lite selection documentation.

Interested in learning more?

The selection abstraction was introduced in the paper Vega-Lite: A Grammar of Interactive Graphics, by Satyanarayan, Moritz, Wongsuphasawat, & Heer. The PRIM-9 system (for projection, rotation, isolation, and masking in up to 9 dimensions) is one of the earliest interactive visualization tools, built in the early 1970s by Fisherkeller, Tukey, & Friedman. A retro demo video survives! The concept of brushing & linking was crystallized by Becker, Cleveland, & Wilks in their 1987 article Dynamic Graphics for Data Analysis. For a comprehensive summary of interaction techniques for

visualization, see Interactive Dynamics for Visual Analysis by Heer & Shneiderman. Finally, for a treatise on what makes interaction effective, read the classic Direct Manipulation Interfaces paper by Hutchins, Hollan, & Norman.

Appendix: On The Representation of Time

Earlier we observed a small bump in the number of movies in either 1969 and 1970. Where does that bump come from? And why 1969 or 1970? The answer stems from a combination of missing data and how your computer represents time.

Internally, dates and times are represented relative to the UNIX epoch, in which time "zero" corresponds to the stroke of midnight on January 1, 1970 in UTC time, which runs along the prime meridian. It turns out there are a few movies with missing (null) release dates. Those null values get interpreted as time 0, and thus map to January 1, 1970 in UTC time. If you live in the Americas – and thus in "earlier" time zones – this precise point in time corresponds to an earlier hour on December 31, 1969 in your local time zone. On the other hand, if you live near or east of the prime meridian, the date in your local time zone will be January 1, 1970.

The takeaway? Always be skeptical of your data, and be mindful that how data is represented (whether as date times, or floating point numbers, or latitudes and longitudes, etc.) can sometimes lead to artifacts that impact analysis!

In [ ]:
```
1
```

In [ ]:
```
1
```