Based on the instructions and the lectures of Dr. Danielle Albers Szafir, I visualize the the house data which derived from data in the cities of the state of Washington USA from May1' to July 9', 2014, was downloaded from https://www.kaggle.com/shree1992/housedata

Getting started with Altair

In [1]:

```
# Import our data processing library
import pandas as pd
import altair as alt
data= pd.read_csv("data.csv")
data.head()
```

Out[1]:

	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	co
0	2014-05-02 00:00:00	313000.0	3.0	1.50	1340	7912	1.5	0	0	
1	2014-05-02 00:00:00	2384000.0	5.0	2.50	3650	9050	2.0	0	4	
2	2014-05-02 00:00:00	342000.0	3.0	2.00	1930	11947	1.0	0	0	
3	2014-05-02 00:00:00	420000.0	3.0	2.25	2000	8030	1.0	0	0	
4	2014-05-02 00:00:00	550000.0	4.0	2.50	1940	10500	1.0	0	0	

In [2]:

```
len(data.index)
```

Out[2]:

```
4600
```

In [3]:

```
data.describe()
```

Out[3]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	v
count	4.600000e+03	4600.000000	4600.000000	4600.000000	4.600000e+03	4600.000000	46
mean	5.519630e+05	3.400870	2.160815	2139.346957	1.485252e+04	1.512065	
std	5.638347e+05	0.908848	0.783781	963.206916	3.588444e+04	0.538288	
min	0.000000e+00	0.000000	0.000000	370.000000	6.380000e+02	1.000000	
25%	3.228750e+05	3.000000	1.750000	1460.000000	5.000750e+03	1.000000	

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors
50%	4.609435e+05	3.000000	2.250000	1980.000000	7.683000e+03	1.500000
75%	6.549625e+05	4.000000	2.500000	2620.000000	1.100125e+04	2.000000
max	2.659000e+07	9.000000	8.000000	13540.000000	1.074218e+06	3.500000

Because I see that there are some missing values with price equal 0, so I'm going to remove those rows from the data. And my target data points are the houses have price equal or below $10,000,000, I removed 2 rows that have price were over 10,000,000.

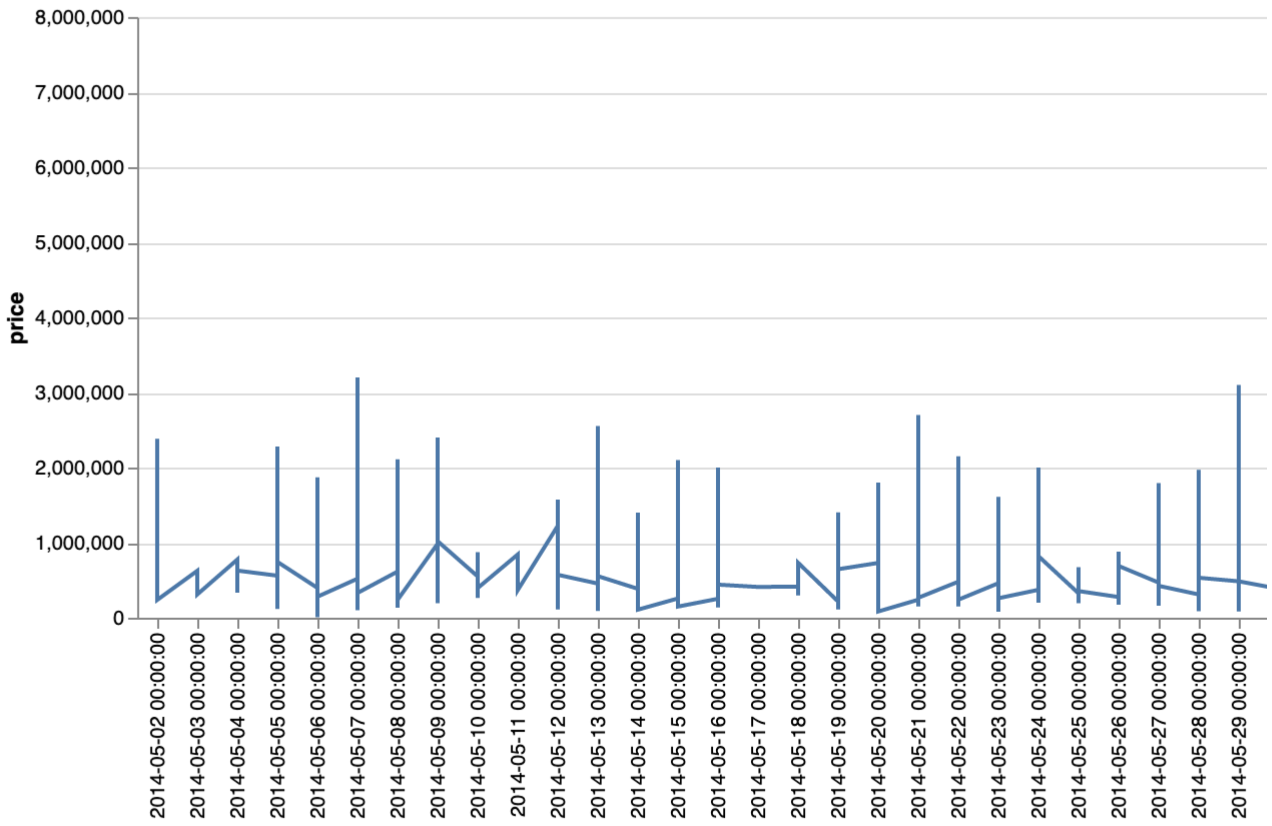
In [4]:
```
data = data.loc[data["price"] < 10000000]
data = data.loc[data["price"] > 0]
```

In [5]:
```
len(data.index)
```

Out[5]:
```
4549
```

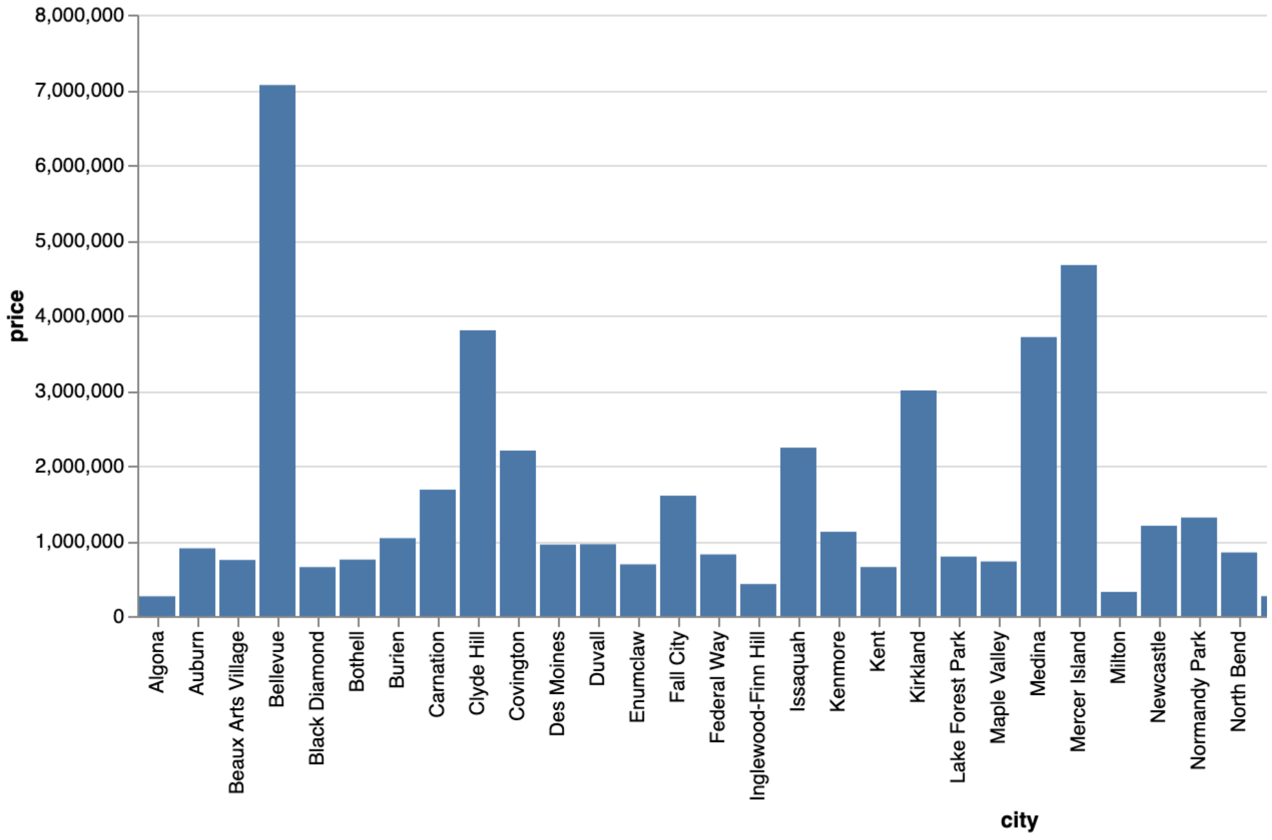
In [6]:
```
# plot the house price over date
alt.Chart(data).mark_line().encode(
    x='date',
    y='price'
)
```

Out[6]:



In [7]:
```
# visualize the price by city

alt.Chart(data).mark_bar().encode(x="city", y="price")
```

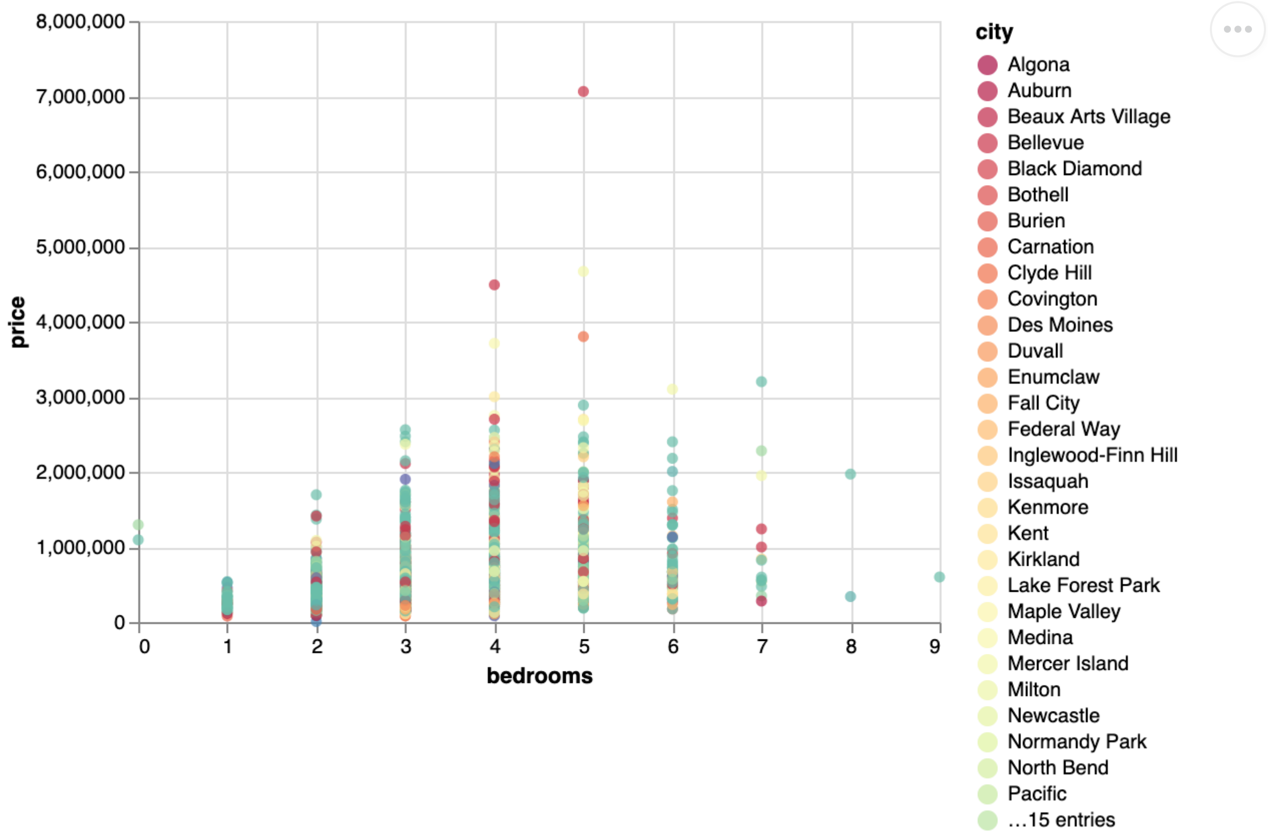
Out[7]:



Now let's look at a bit different relationship in our data. Let's map see if price correlates with the number of bedrooms. In other words, do we see evidence that the house has more bedrooms will have the higher price?

In [8]:
```
# visualize the data with price by the number of bedrooms
alt.Chart(data).mark_circle().encode(
    x = "bedrooms",
    y = "price",
    color=alt.Color('city', scale=alt.Scale(scheme='spectral')),
    tooltip=["city", "price"]
)
```

Out[8]:

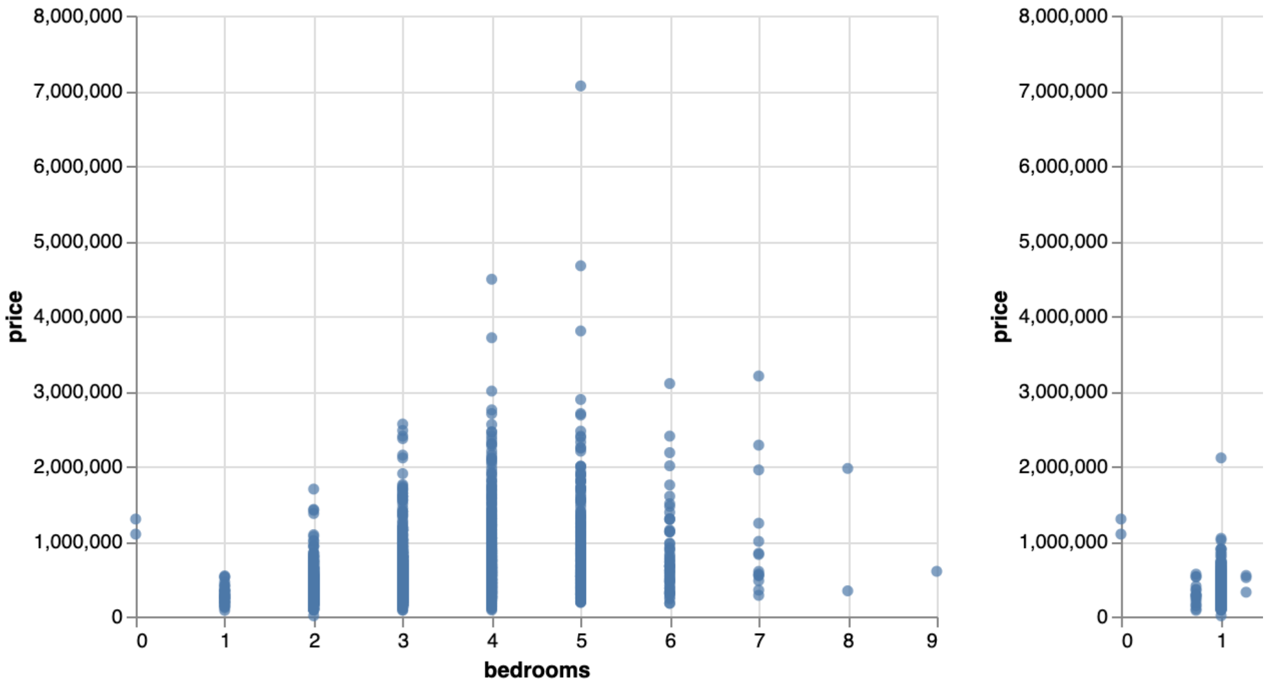


Let's add a little more information to our chart. Since we're already using position to encode two dimensions of our dataset, we can use other channels to represent new data. Try mapping bathrooms to the size of marks in scatterplot

In [9]:

```
c1 = alt.Chart(data).mark_circle().encode(
    x = "bedrooms",
    y = "price",
)

c2 = alt.Chart(data).mark_circle().encode(
    x = "bathrooms",
    y = "price",
)

c1|c2
```

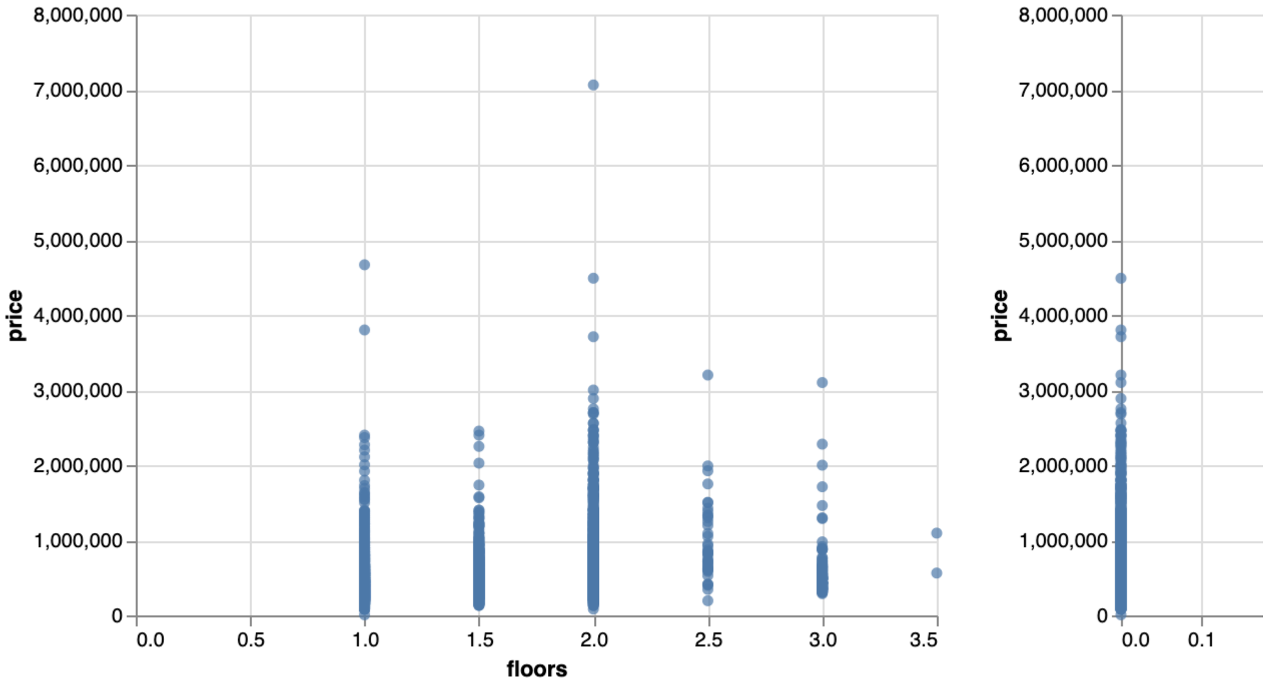
Out[9]:



In [10]:

```
c3 = alt.Chart(data).mark_circle().encode(
    x = "floors",
    y = "price",
)

c4 = alt.Chart(data).mark_circle().encode(
    x = "waterfront",
    y = "price",
)

c3|c4
```

Out[10]:

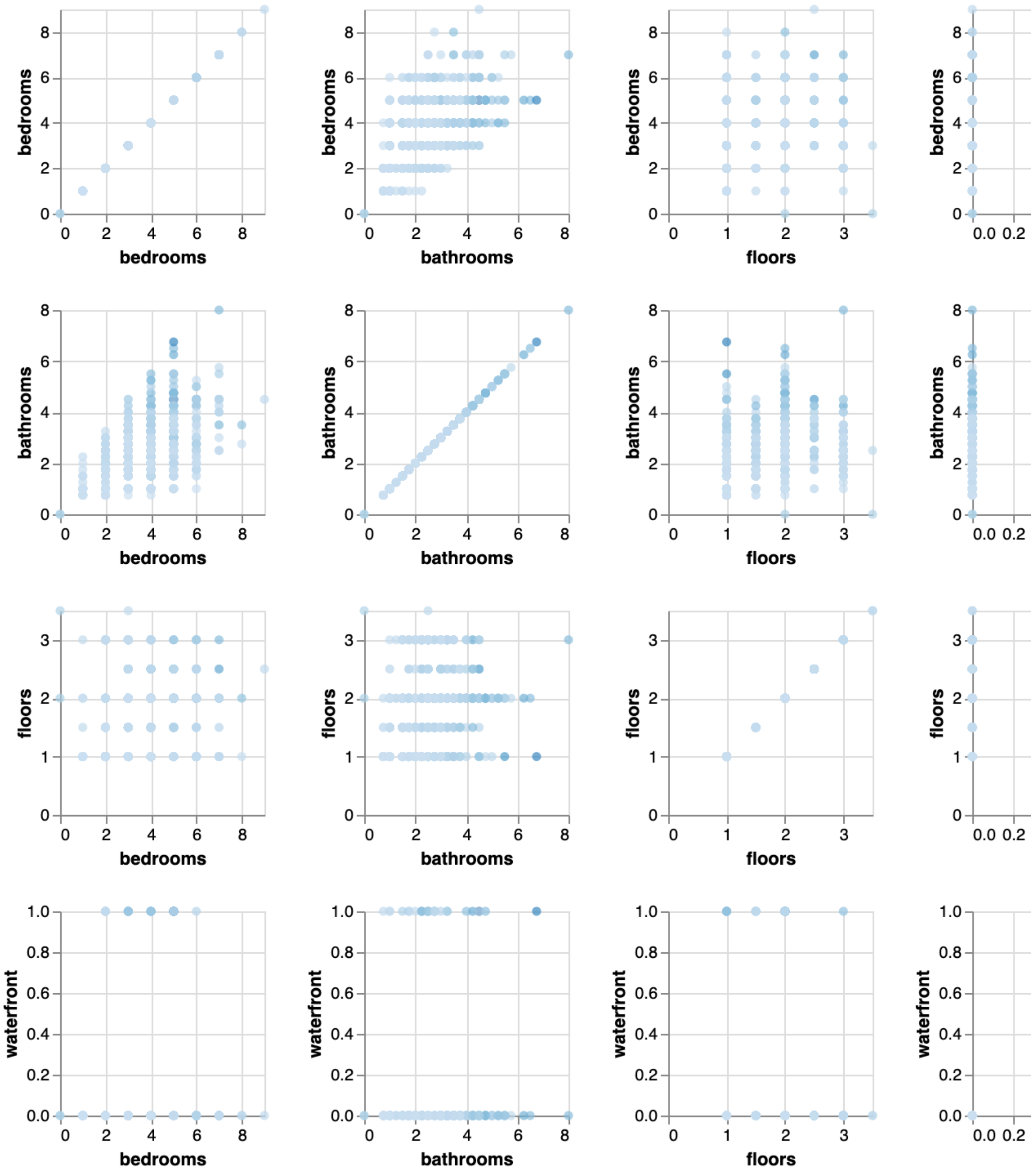


In our last chart, we'll experiment with faceting our data to visualize different charts for different combinations of dimensions. To do this, we'll use the `repeat` function to look at price (mapped to color) across the number of bedrooms, bathrooms, floors and waterfront. Which dimensions appear to correlate most to price?

In [11]:

```
# Build a SPLOM
alt.Chart(data).mark_circle().encode(
    alt.X(alt.repeat("column"), type="quantitative"),
    alt.Y(alt.repeat("row"), type="quantitative"),
    color="price",
    tooltip=["city", "price"]
).properties(
    width=125,
    height=125
).repeat(
    row=["bedrooms", "bathrooms", "floors", "waterfront"],
    column=["bedrooms", "bathrooms", "floors", "waterfront"]
)
```

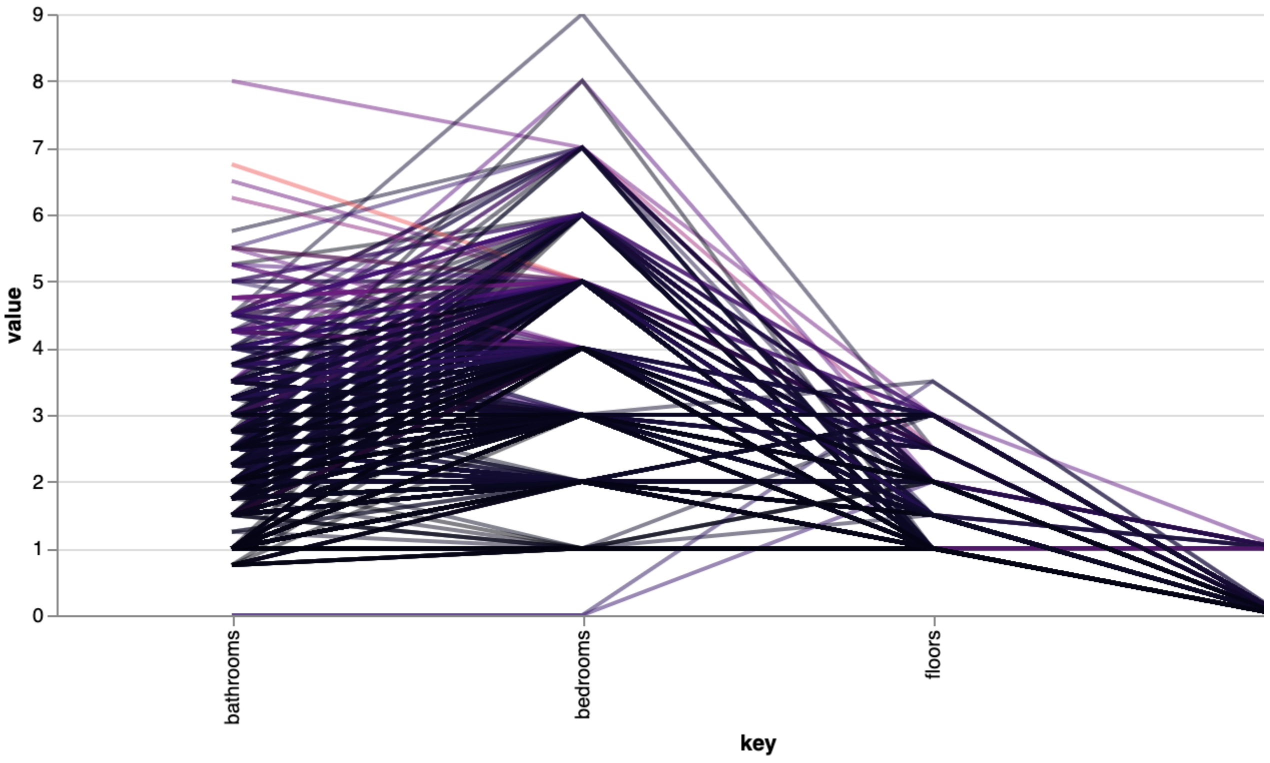
Out[11]:



In [12]:

```
# Build a parallel coordinates plot
alt.Chart(data).transform_window(
    index="count()"
).transform_fold(
    ["bedrooms", "bathrooms", "floors", "waterfront"]
).mark_line().encode(
    x="key:N",
    y="value:Q",
    detail="index:N",
    opacity=alt.value(0.5),
    color=alt.Color("price:Q", scale=alt.Scale(scheme="Magma")),
    tooltip=["city"]
).properties(width=700).interactive()
```

Out[12]:



We can save any plot to export it as an image using the "..." icon in the upper right of the chart. Alternatively, you can programmatically save your visualizations as interactive Javascript charts embeddable in web pages. You simply need to assign your chart to a variable (`chart = alt.Chart(...)`) and use `chart.save()` as in the following example. Note that the chart will not render to the notebook if you assign it to a variable. Instead, the following code will automatically write out an HTML document containing an interactive SVG of the visualization.

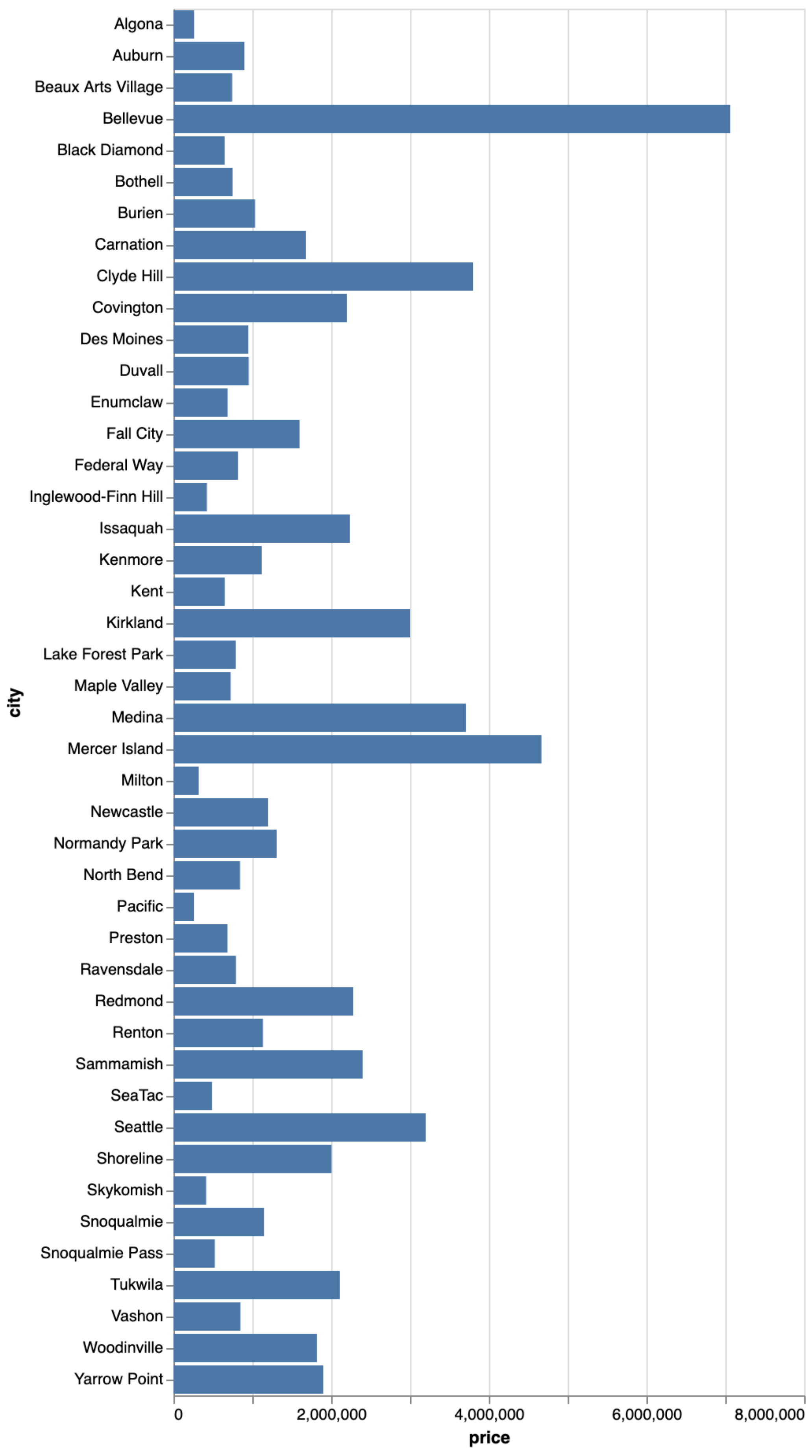
In [13]:

```
# Store the SPLOM
chart = alt.Chart(data).mark_circle().encode(
    alt.X(alt.repeat("column"), type="quantitative"),
    alt.Y(alt.repeat("row"), type="quantitative"),
    color="price",
    tooltip=["city", "price"]
).properties(
    width=125,
    height=125
).repeat(
    row=["bedrooms", "bathrooms", "floors", "waterfront"],
    column=["bedrooms", "bathrooms", "floors", "waterfront"]
).interactive()

chart.save('housechart.html', embed_options={'renderer':'svg'})
```

In [14]:

```
alt.Chart(data).mark_bar().encode(x="price", y="city")
```

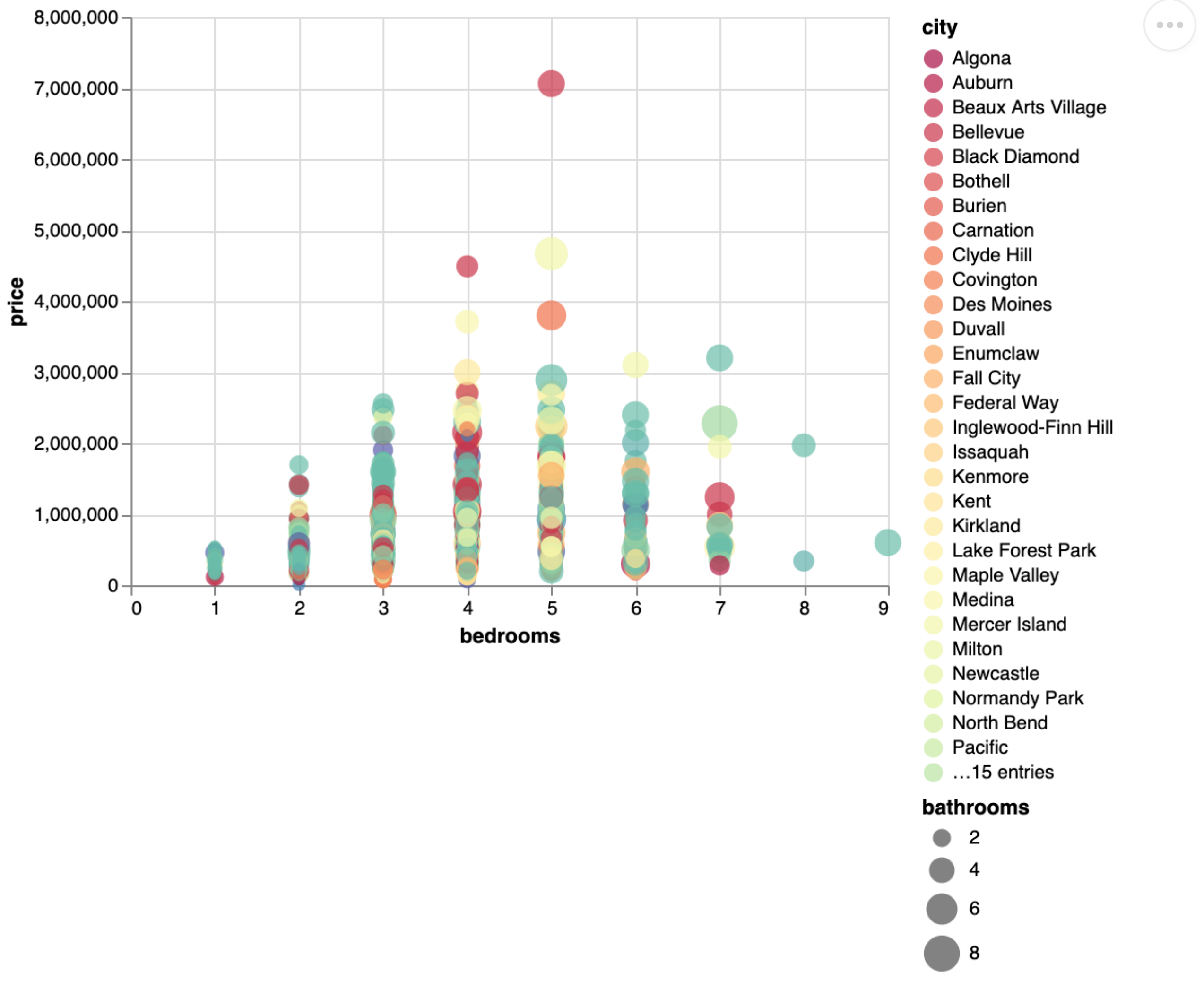
Out[14]:



In [15]:

```
alt.Chart(data).mark_circle().encode(
    x = "bedrooms",
    y = "price",
    color=alt.Color('city', scale=alt.Scale(scheme='spectral')),
    size="bathrooms",
    tooltip=["city", "price"]
)
```

Out[15]:



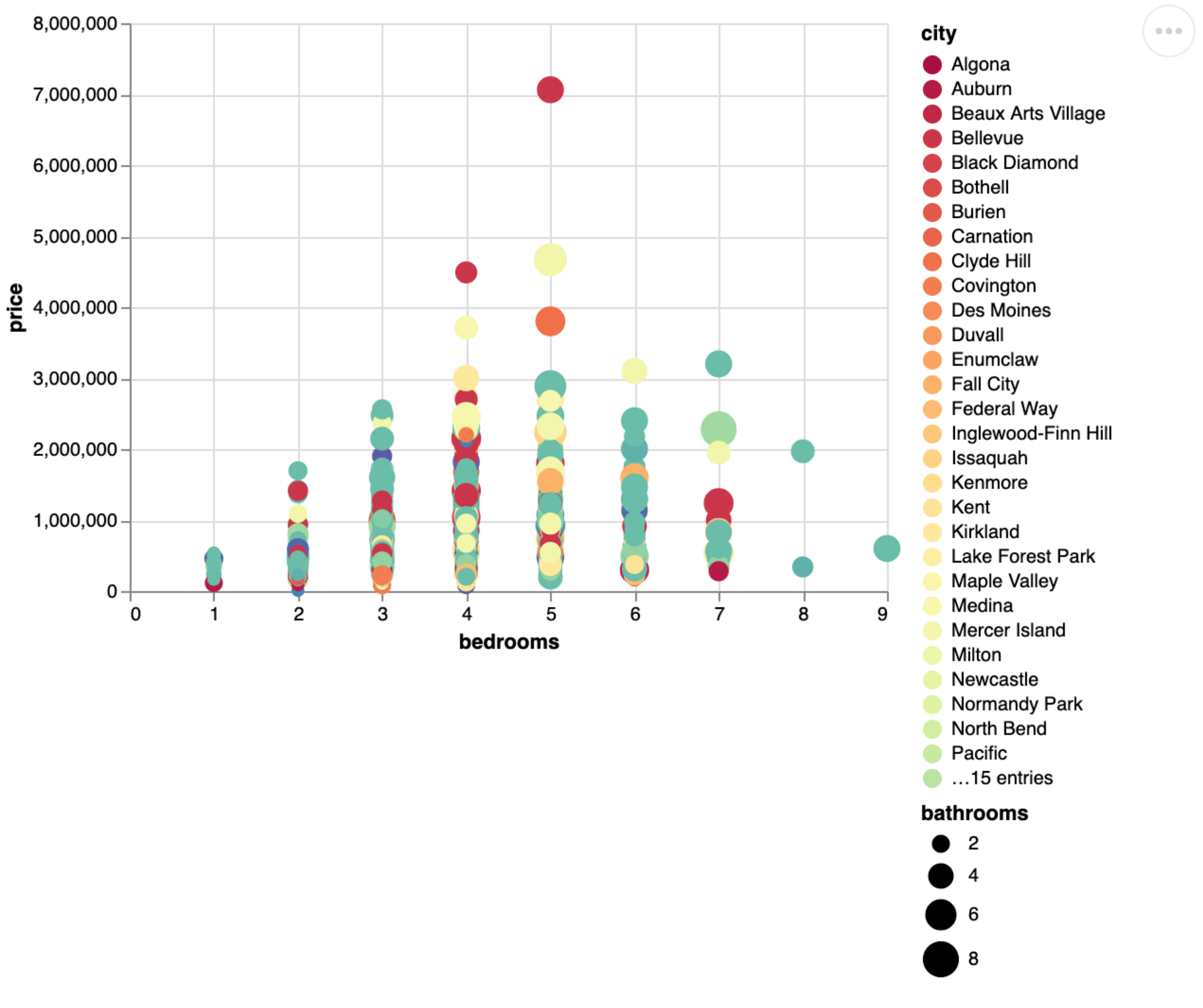
Selection

We can implement selection using Altair's `alt.selection` function. This will create a new type of selection action that we can bind to certain elements of the chart. For this first chart, we'll let people select a country and select countries of the same region by clicking a point. We'll reduce the opacity of any of the selected points using the `alt.condition` function.

In [16]:

```
# Implementing selection
selection = alt.selection(type='multi', fields=['city'])

alt.Chart(data).mark_circle().encode(
    x = "bedrooms",
    y    = "price",
    color=alt.Color('city', scale=alt.Scale(scheme='spectral')),
    size="bathrooms",
    tooltip=["city", "price"],
    opacity=alt.condition(selection,alt.value(1),alt.value(.2))
).add_selection(selection)
```

Out[16]:

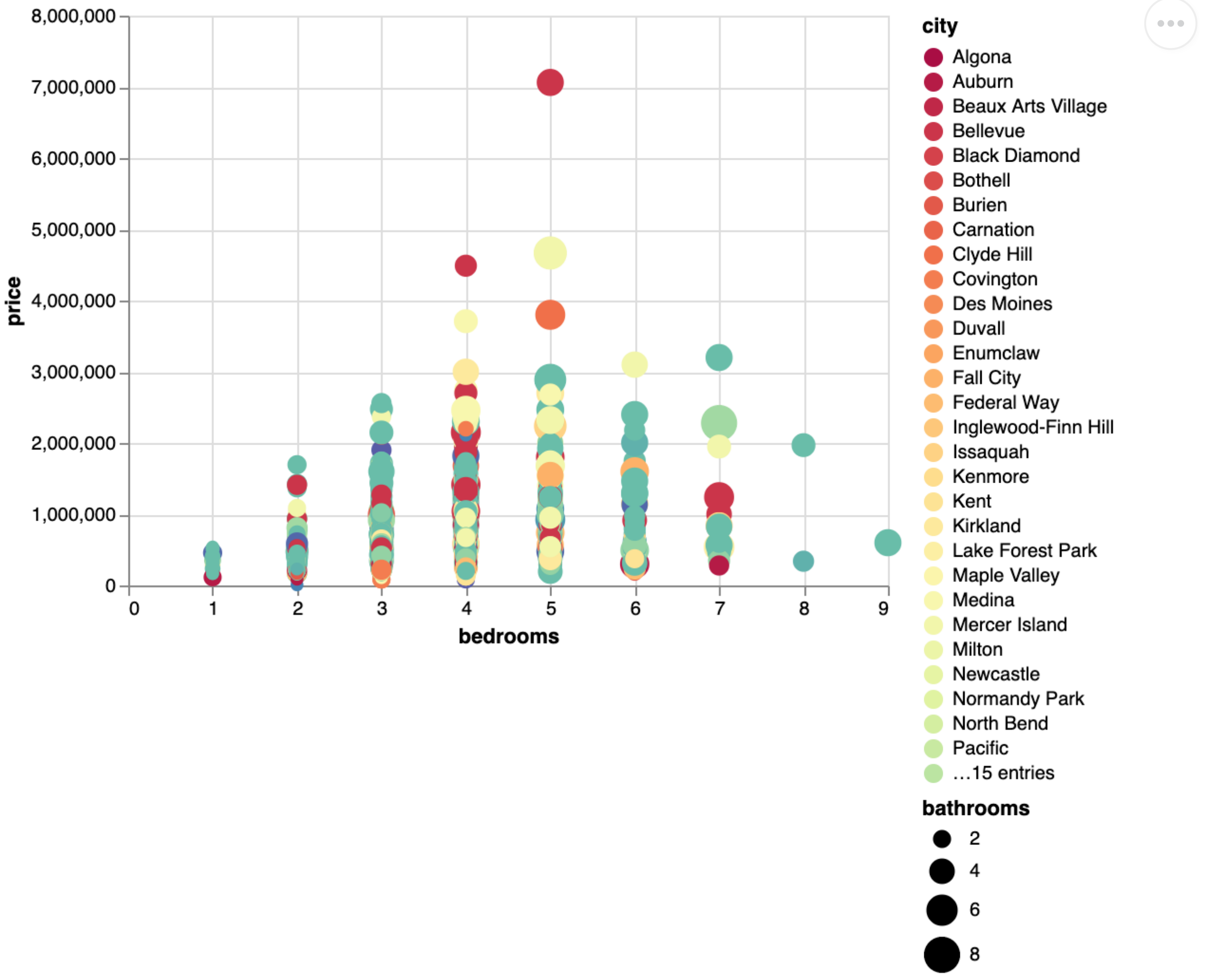


Clicking can be a bit cumbersome, so we'll switch to a mouseover to be faster.

In [17]:

```
# Implementing selection
selection = alt.selection(type='multi', fields=['Region'], on='mouseover', neare

alt.Chart(data).mark_circle().encode(
    x = "bedrooms",
    y    = "price",
    color=alt.Color('city', scale=alt.Scale(scheme='spectral')),
    size="bathrooms",
    tooltip=["city", "price"],
    opacity=alt.condition(selection,alt.value(1),alt.value(.2))
).add_selection(selection)
```

Out[17]:

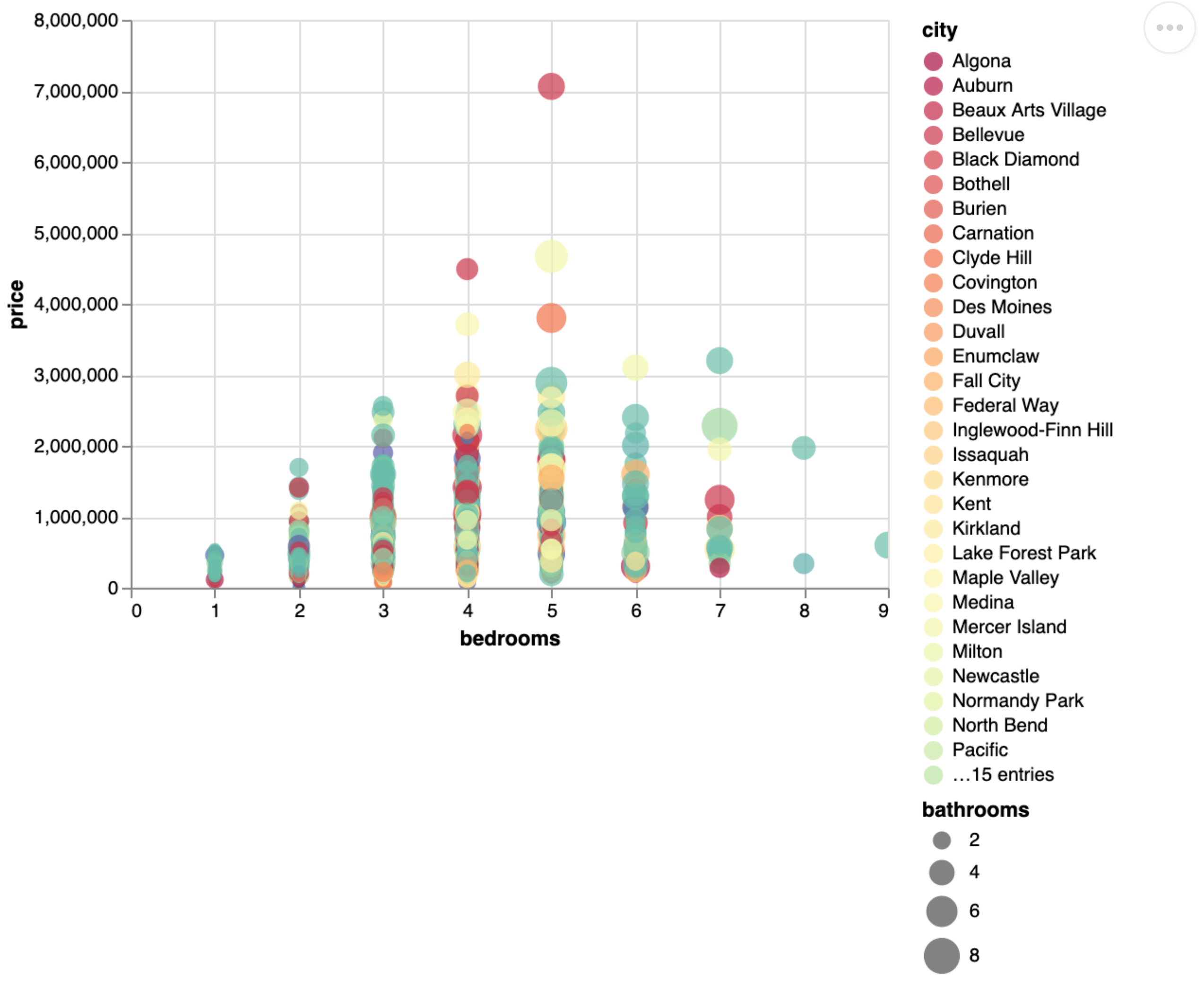


Exploration

We'll implement exploration through panning and zooming. Pan and zoom are such common operations that Altair lets you implement them using a single function: `interactive()`

In [18]:
```
alt.Chart(data).mark_circle().encode(
    x = "bedrooms",
    y    = "price",
    color=alt.Color('city', scale=alt.Scale(scheme='spectral')),
    size="bathrooms",
    tooltip=["city", "price"]
).interactive()
```

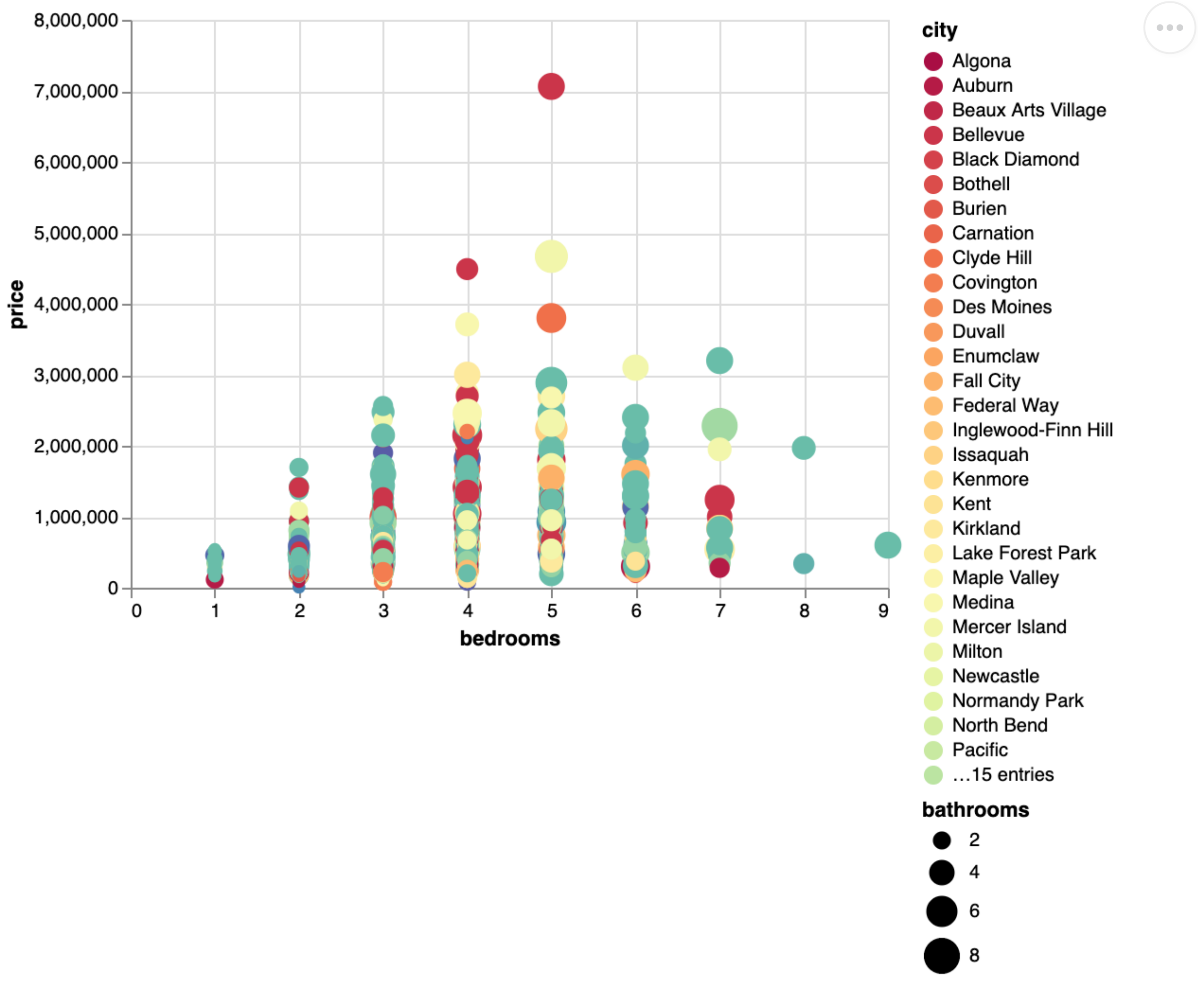
Out[18]:



In [19]:

```
# Let's implement filtering using dynamic queries.
dropdown = alt.binding_select (options=data["city"].unique(), name="Select a cit

# Create a new selection that uses my dynamic query widget
selection = alt.selection(type="single", fields=["city"], bind=dropdown)

# Let's specify our chart
alt.Chart(data).mark_circle().encode(
    x = "bedrooms",
    y = "price",
    color=alt.Color('city', scale=alt.Scale(scheme='spectral')),
    size="bathrooms",
    tooltip=["city", "price"],
    opacity=alt.condition(selection,alt.value(1),alt.value(.2))
).add_selection(selection)
```

Out[19]:



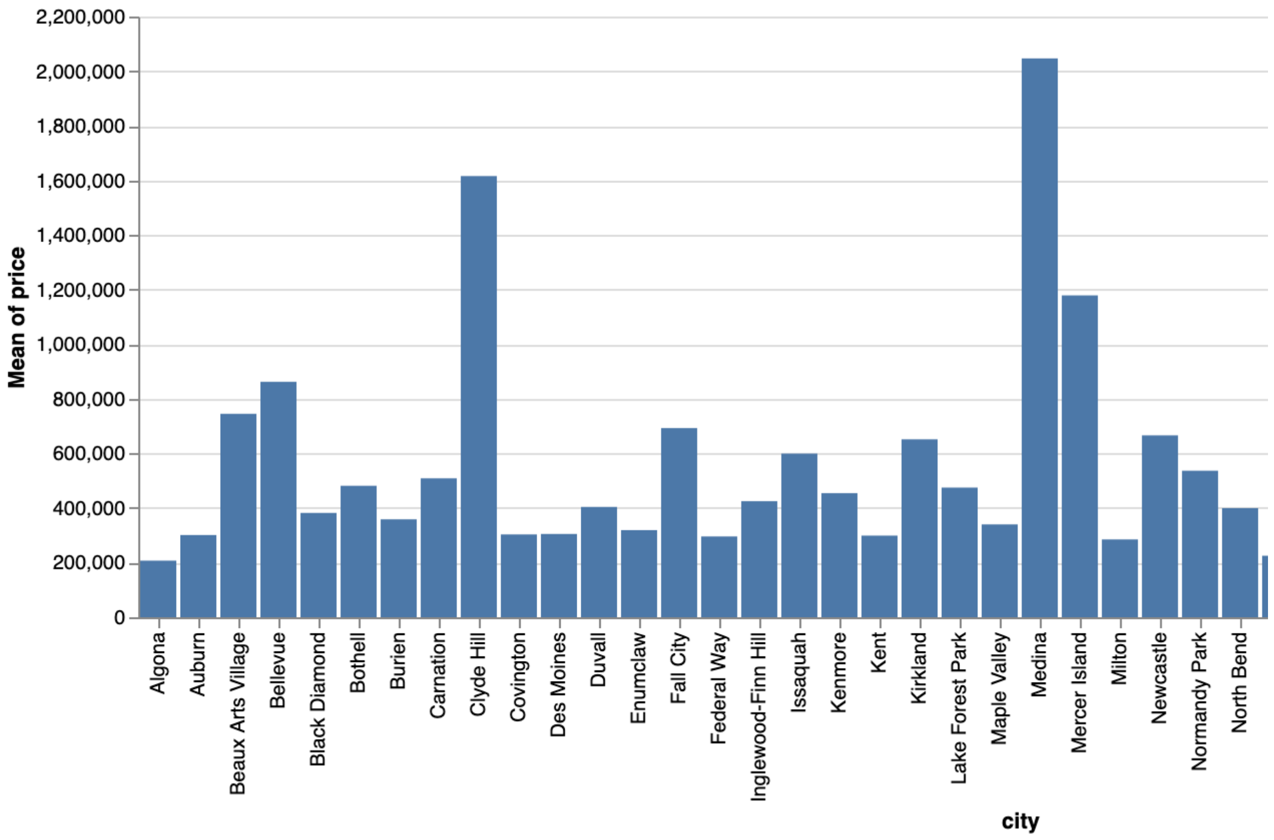
Select a city: Shoreline

Reconfigure

To reconfigure data in Altair, we'll just use Altair's basic `sort` parameter. Let's start with a standard bar chart. You can see for categorical attributes, Altair will sort the data alphabetically by default.

In [20]:
```
# Let's specify our chart
alt.Chart(data).mark_bar().encode(
    y = "mean(price)",
    x = "city"
)
```

Out[20]:

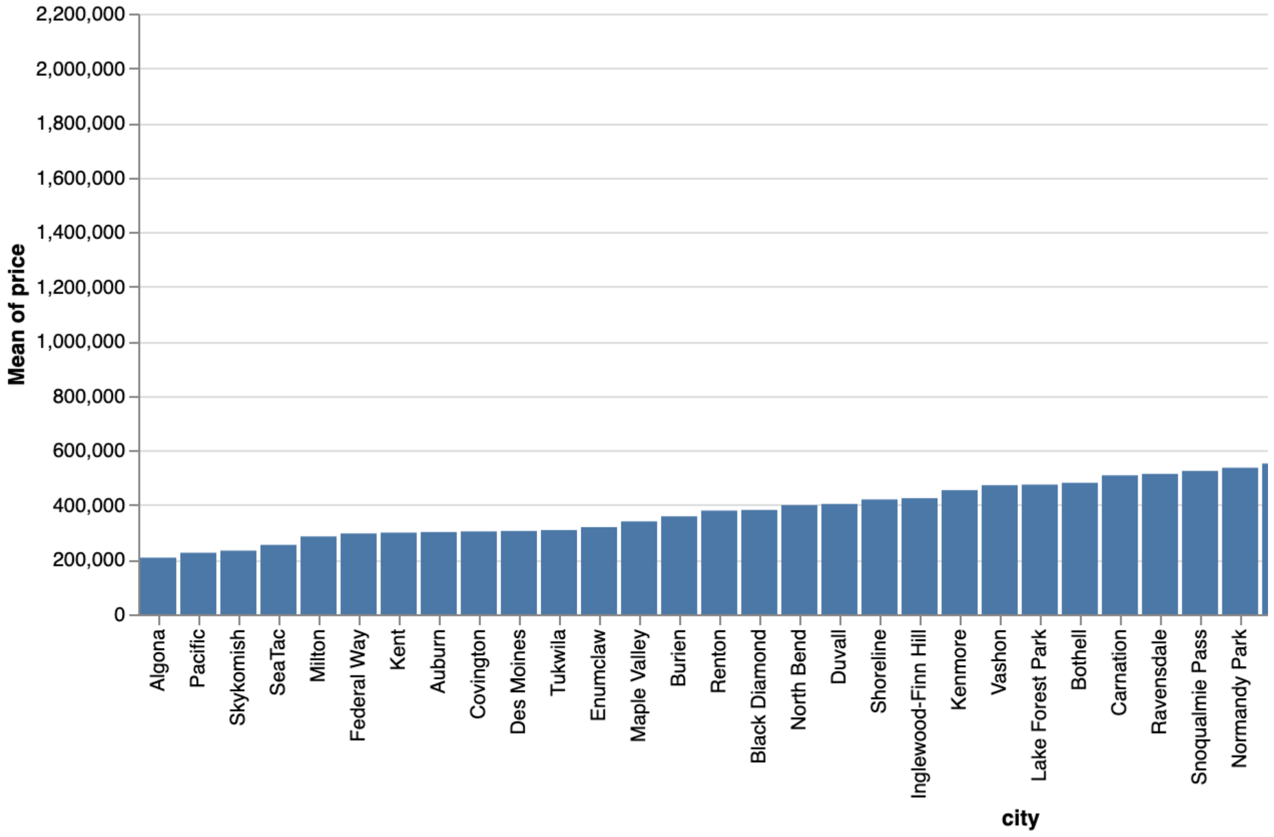


Now let's take that same data and reorder it according to some prespecified aspect of our data. For example, let's say that we want to look at how the price change from the lowest to highest cities. We can sort the data by the mean price per city using the `EncodingSortField` function. We can try changing the sorting order from ascending to descending using the `order` parameter of `EncodingSortField`.

In [21]:

```
# Let's specify our chart
alt.Chart(data).mark_bar().encode(
    y = "mean(price)",
    x = alt.X(field='city', type='nominal', sort=alt.EncodingSortField(field='pr
)
```

Out[21]:

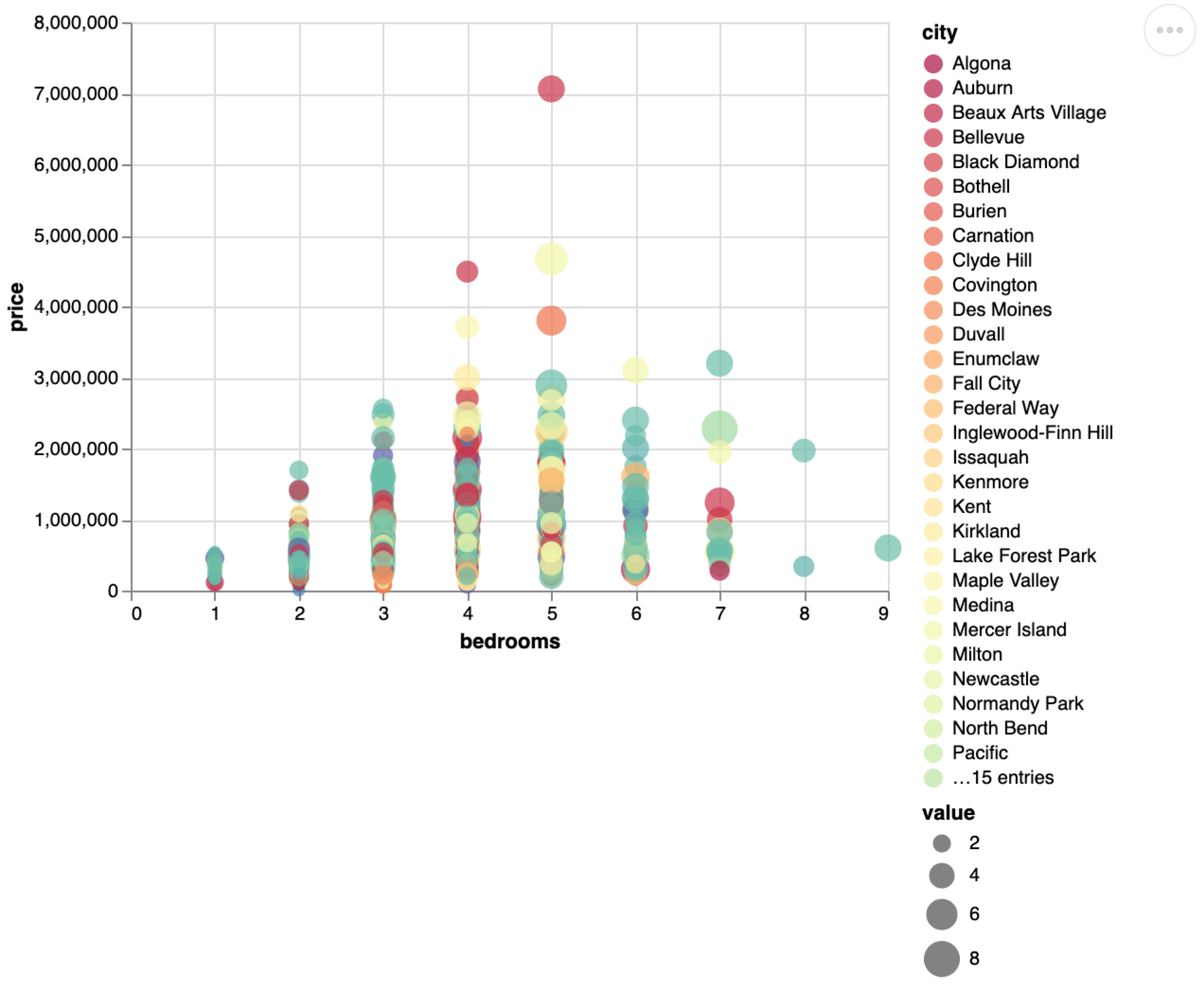


Encode

We can change the dimensions or data mappings used in a visualization to surface different patterns and properties of the data. One way to do this is through UI widgets that let us specify what attributes a given dimension maps to. We can use the `transform_fold` function we learned about in the Intro to Altair notebook to bind a list of attributes to a dropdown list. We can then specify what elements of that new `column, value` pair we create using the `selection` value.

In [22]:
```
# Let's implement filtering using dynamic queries.
dropdown = alt.binding_select (options=["bathrooms", "floors", "waterfront"], na

# Create a new selection that uses my dynamic query widget
selection = alt.selection(type="single", fields=['column'], bind=dropdown, init=

# Let's specify our chart
alt.Chart(data).transform_fold(
    ["bathrooms", "floors", "waterfront"],
    as_=['column', 'value']
).transform_filter(
    selection
).mark_circle().encode(
    x = "bedrooms",
    y = "price",
    color=alt.Color('city', scale=alt.Scale(scheme='spectral')),
    size="value:Q",
    tooltip=["city", "price"],
).add_selection(selection)
```

Out[22]:



Select a size variable: bathrooms

Connect

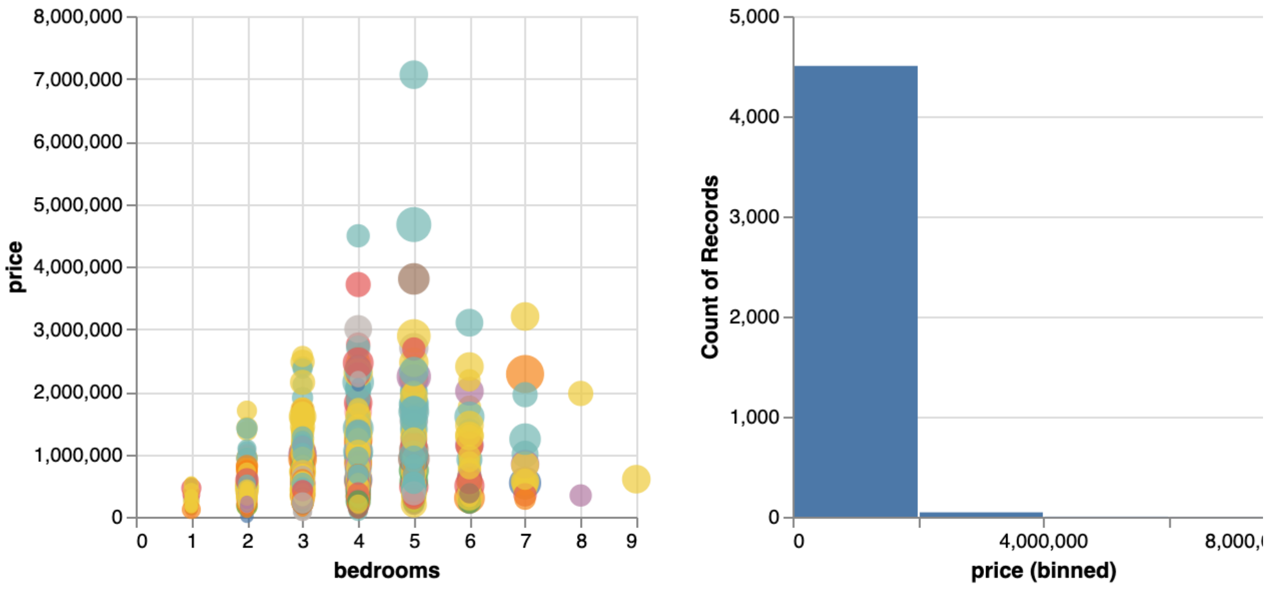
Connection interactions pair actions in one visualization with corresponding actions in another. For example, selecting a set of points in one visualization may change the corresponding data visualized in a second. In this example, we'll pair a bubblechart with a histogram using two different forms of selection. In the first form, clicking on a point will filter the histogram for the region of the selected country. We can use the `transform_filter` function to filter based on the value of the `selection` .

In [23]:

```
# Linked views
# Creating a selection:
selection = alt.selection(type="multi", fields=["city"])

# Create a container for our two different views
base =  alt.Chart(data).properties(width=250, height=250)
```

```
# Create our scatterplot
scatterplot = base.mark_circle().encode(
    x = 'bedrooms',
    y = 'price',
    size = "bathrooms",
    color = alt.condition(selection, "city", alt.value('lightgray'))
).add_selection(selection)

# Create a histogram
hist = base.mark_bar().encode(
    x = alt.X("price", bin=alt.Bin(maxbins=5)),
    y = "count()"
).transform_filter(selection)

# Connect our charts using the pipe operation
scatterplot | hist
```

Out[23]:

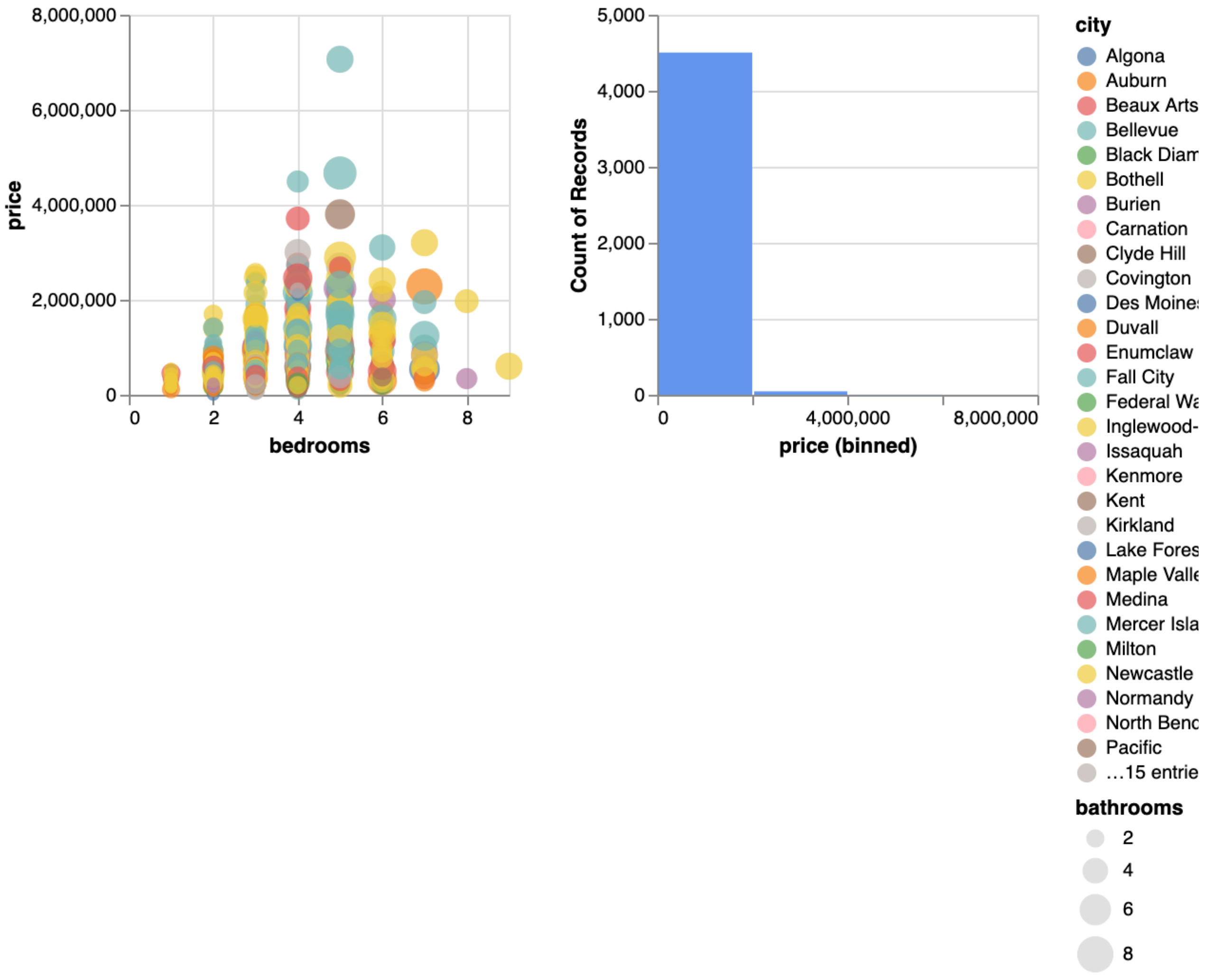


We can make the selection a little more specific to filter for sets of countries using a lasso selection. In a lasso selection, we can click and drag on a chart to select a set of points. We can do this in Altair by using an `interval` selection on the `x` and `y` attributes of the visualization. In other words, we set up the selection to select for the interval in x and y between the x value we first click on and the y value we release the mouse button on. We'll also add in a little extra context by layering the revised histogram overtop of the original data distribution.

In [24]:

```
# This selection is going to be an interval selection
selection = alt.selection(type="interval", encodings=["x", "y"])

# Create our scatterplot
scatterplot = alt.Chart(data).mark_circle().encode(
    x = 'bedrooms',
    y = 'price',
    size = "bathrooms",
    color = alt.condition(selection, "city", alt.value('lightgray'))
).properties(
    width = 200,
    height = 200
).add_selection(selection)

# Define our background chart
base = alt.Chart().mark_bar(color="cornflowerblue").encode(
    x = alt.X("price", bin=alt.Bin(maxbins=5)),
    y = "count()"
).properties (
    width=200,
    height = 200
)

# Grey background to show the selection range in the scatterplot
background = base.encode(color=alt.value('lightgray')).add_selection(selection)

# Blue highlights to show the transformed (brushed) data
highlight = base.transform_filter(selection)

# Layer the two charts
layers = alt.layer(background, highlight, data = data)

scatterplot | layers
```

Out[24]:


8,000,000
6,000,000
4,000,000
2,000,000
0
price
0
2
4
6
8
bedrooms
5,000
4,000
3,000
2,000
1,000
0
Count of Records
0
4,000,000
8,000,000
price (binned)
city
Algona
Auburn
Beaux Arts
Bellevue
Black Diam
Bothell
Burien
Carnation
Clyde Hill
Covington
Des Moines
Duvall
Enumclaw
Fall City
Federal Wa
Inglewood-
Issaquah
Kenmore
Kent
Kirkland
Lake Fores
Maple Valle
Medina
Mercer Isla
Milton
Newcastle
Normandy
North Benc
Pacific
…15 entrie
bathrooms
2
4
6
8