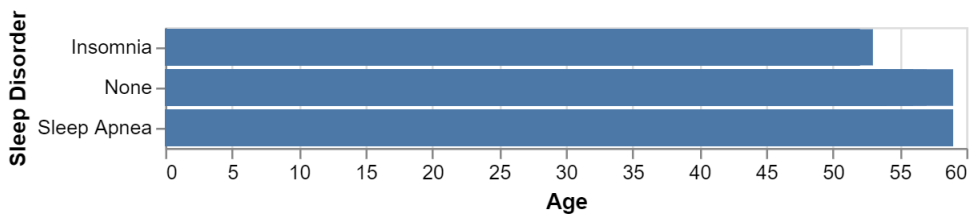
```
# Import our data processing library (note: you may have to install this!)
import pandas as pd

# Let's use this to upload a sample dataset and show the start of the dataset
# Note that you need to download the dataset and make sure it's in the same
# directory as your notebook
data= pd.read_csv("/content/Sleep_health_and_lifestyle_dataset.csv")
data.head()
```

	Person ID	Gender	Age	Occupation	Sleep Duration	Quality of Sleep	Physical Activity Level	Stress Level	BMI Category	P
0	1	Male	27	Software Engineer	6.1	6	42	6	Overweight	
1	2	Male	28	Doctor	6.2	6	60	8	Normal	
2	3	Male	28	Doctor	6.2	6	60	8	Normal	
3	4	Male	28	Sales Representative	5.9	4	30	8	Obese	

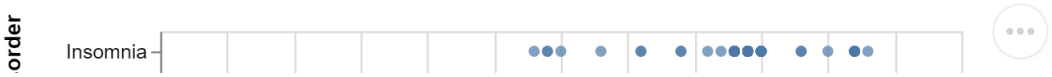
Now let's look at a bit different relationship in our data. Let's map see if Sleep Disorder correlates with Age. In other words, do we see evidence that with age sleep disorder occurs?

```
# Now let's visualize the data
import altair as alt

alt.Chart(data).mark_bar().encode(x="Age", y="Sleep Disorder")
```



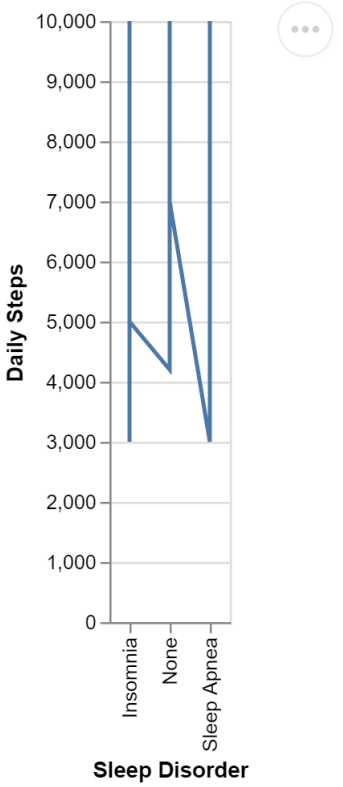
Let's try to use the same variables above but to create a line graph

```
alt.Chart(data).mark_circle().encode(x="Age", y="Sleep Disorder")
```



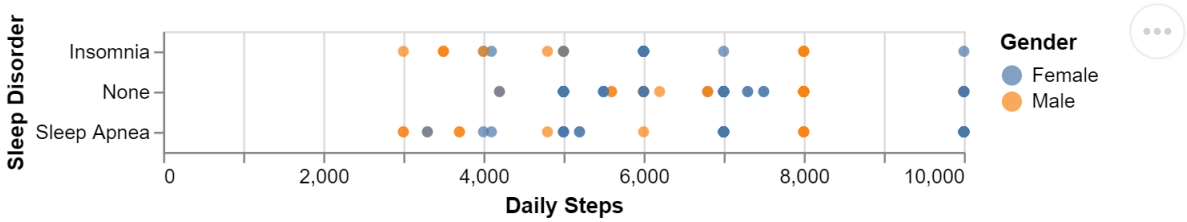
Now Let's try to use the other variables how it's influance our Sleep Disorder.Here I am trying to see corelation between daily Steps and Sleep Disorder.

```
alt.Chart(data).mark_line().encode(
    x='Sleep Disorder',
    y='Daily Steps'
)
```



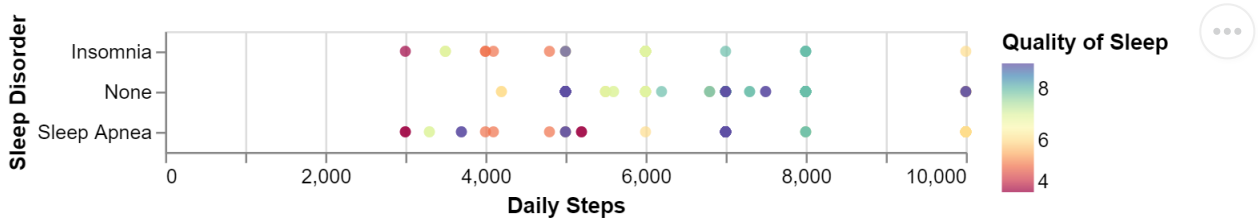
We can also add in a little more information to this plot. Let's add in some color to differentiate the data with Gender.

```
alt.Chart(data).mark_circle().encode(
    x = "Daily Steps",
    y = "Sleep Disorder",
    color="Gender"
)
```



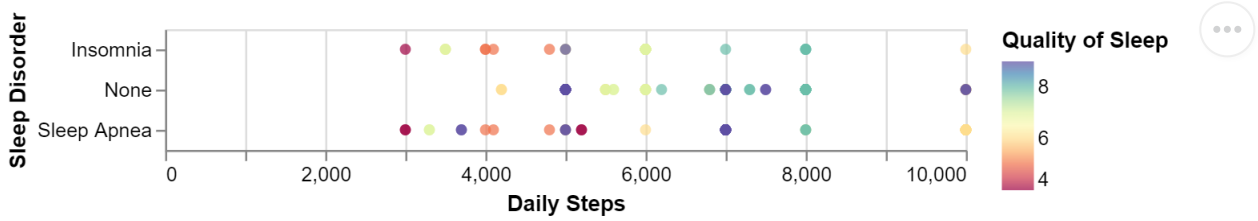
Let's choose a little bit different color scheme to make Quality of Sleep pop a bit more based on Daily Steps

```
alt.Chart(data).mark_circle().encode(
    x = "Daily Steps",
    y = "Sleep Disorder",
    color=alt.Color('Quality of Sleep', scale=alt.Scale(scheme='spectral'))
)
```



Let's start to integrate some additional information into the visualization, starting with a basic interaction: adding a tooltip.With mouse over it will reflect the data.

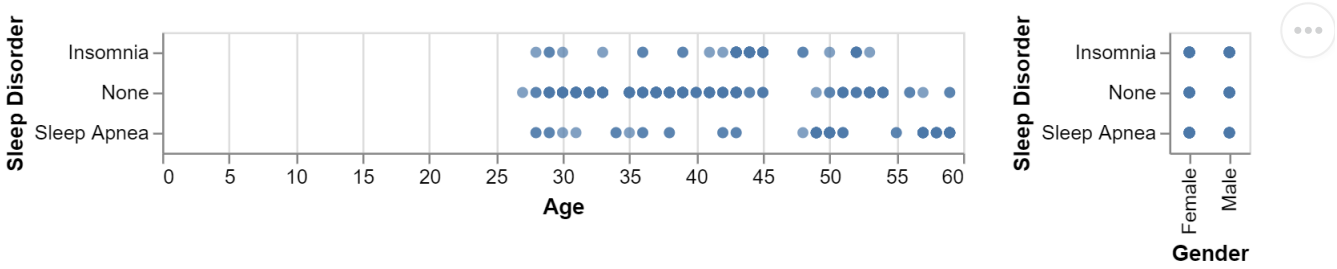
```
alt.Chart(data).mark_circle().encode(
    x = "Daily Steps",
    y = "Sleep Disorder",
    color=alt.Color('Quality of Sleep', scale=alt.Scale(scheme='spectral')),
    tooltip=["Gender", "Sleep Disorder"]
)
```



Now we can experiment to show different charts in a single go. We can apply various approaches here. let's first try to plot 2 different charts togather .

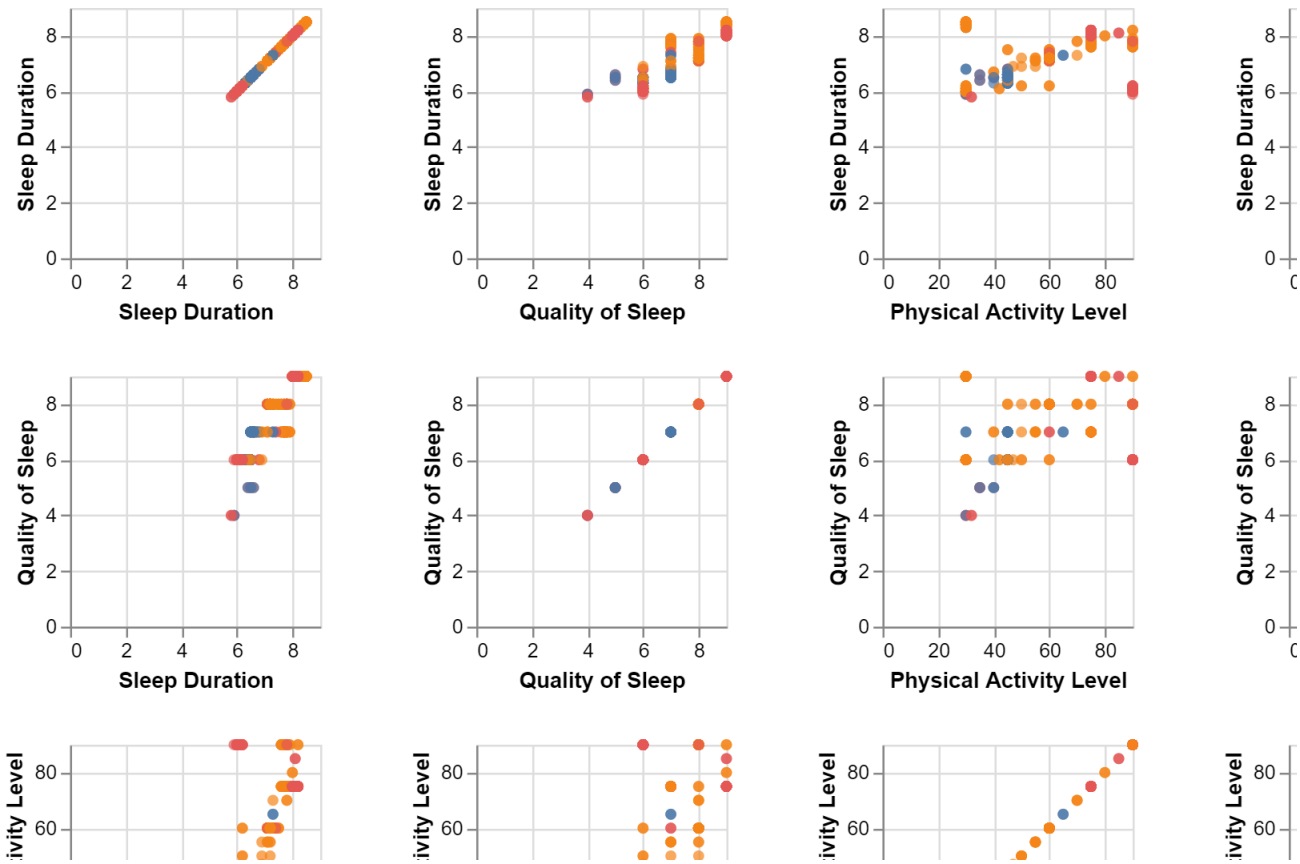
```
c1 = alt.Chart(data).mark_circle().encode(
    x = "Age",
    y = "Sleep Disorder",
)

c2 = alt.Chart(data).mark_circle().encode(
    x = "Gender",
    y = "Sleep Disorder",
```

```
)

c1|c2
```



In our last chart, we had experimented with faceting our data to visualize different charts for different combinations of data. Now, we'll use the repeat function to look at Sleep Disorder(mapped to color) related to age and across Sleep Duration, Quality of Sleep, Physical Activity Level,Daily Steps data.

```
# Build a SPLOM
alt.Chart(data).mark_circle().encode(
    alt.X(alt.repeat("column"), type="quantitative"),
    alt.Y(alt.repeat("row"), type="quantitative"),
    color="Sleep Disorder",
    tooltip=["Age", "Sleep Disorder"]
).properties(
    width=125,
    height=125
).repeat(
    row=["Sleep Duration","Quality of Sleep","Physical Activity Level","Daily Steps"],
    column=["Sleep Duration","Quality of Sleep","Physical Activity Level","Daily Steps"]
)
```



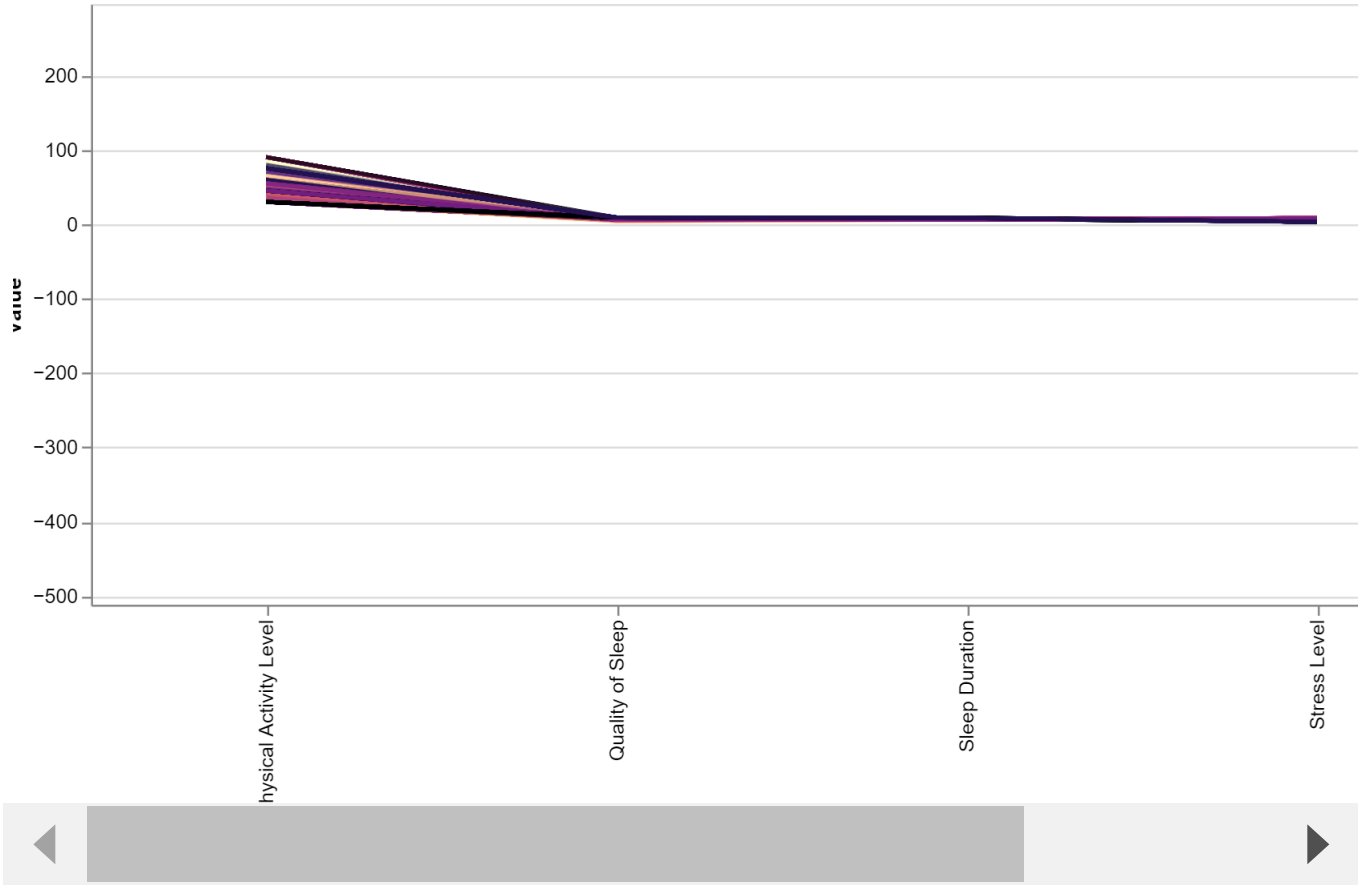
Altair enables us to create a broad variety of traditional and non-traditional visualizations. For example, we can also create line graphs to look at data over time or across variables like rankings or parallel coordinate plots to explore a variety of dimensions. Here's a line graph that explores helth pattern as a function of Heart Rate.

We can explore multiple variables at once using a parallel coordinates plot. In a parallel coordinates plot, we have multiple axes (one for each dimension) that are arrayed horizontally. Each line represents one datapoint. The position of a line on each axis corresponds to its value at that axis. We can look at lines that move together to see correlations between different data values.

Implementing this graph is a bit more complicated than some of the others, but will hopefully give you a sense of the power of Altair as a platform. The plot below uses the transform_window and transform_fold operations to specify the axes (which we can use the key variable to reflect). The value variable will give us the value of the point for a given dimension and the index variable gives us our dimension. Finally, we'll reduce the opacity of all lines to 50% to see our data a bit better and use a multihue sequential colorscheme to make the differences a little easier to see.

```
# Build a parallel coordinates plot
alt.Chart(data).transform_window(
    index="count()"
).transform_fold(
    ["Sleep Duration","Quality of Sleep","Physical Activity Level","Stress Level"]
).mark_line().encode(
    x="key:N",
```

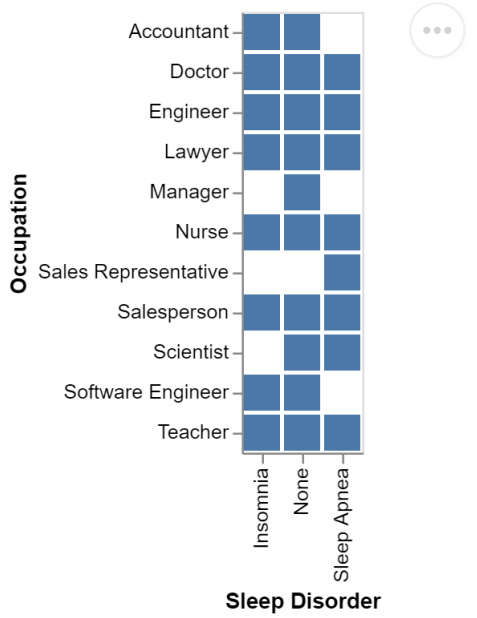
```
    x="key:N",
    y="value:Q",
    detail="index:N",
    opacity=alt.value(0.5),
    color=alt.Color("Heart Rate", scale=alt.Scale(scheme="Magma")),
    tooltip=["Sleep Disorder"]
).properties(width=700).interactive()
```



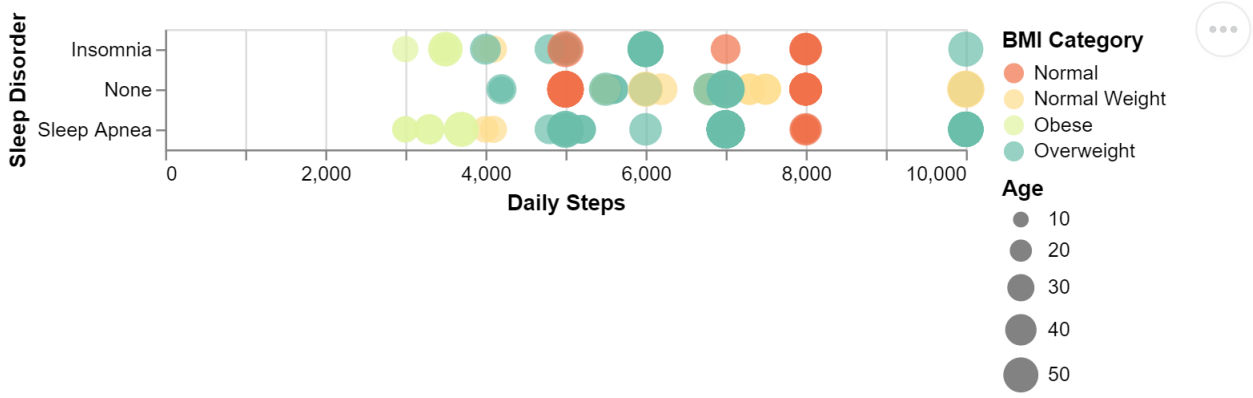
We can save any plot to export it as an image using the "..." icon in the upper right of the chart. Alternatively, you can programmatically save your visualizations as interactive Javascript charts embeddable in web pages. You simply need to assign your chart to a variable (chart = alt.Chart(...)) and 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

```
# 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="Sleep Disorder",
    tooltip=["Occupation", "Sleep Disorder"]
).properties(
    width=125,
    height=125
).repeat(
```

```
    row=["Sleep Duration","Quality of Sleep","Physical Activity Level","Stress Level"],
    column=["Sleep Duration","Quality of Sleep","Physical Activity Level","Stress Level"]
).interactive()

chart.save('/content/webchart.html', embed_options={'renderer':'svg'})


alt.Chart(data).mark_bar().encode(x="Sleep Disorder", y="Occupation")
```



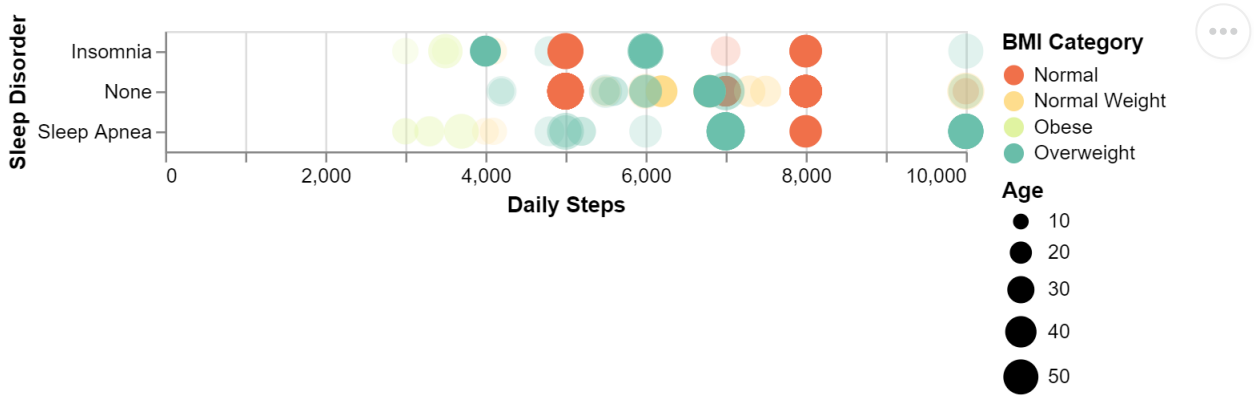

```
alt.Chart(data).mark_circle().encode(
    x = "Daily Steps",
    y = "Sleep Disorder",
    color=alt.Color('BMI Category', scale=alt.Scale(scheme='spectral')),
    size="Age",
    tooltip=["Occupation","Sleep Disorder"]
)
```



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

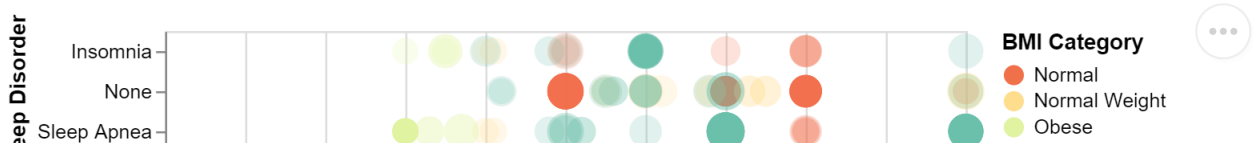
people select a ocupation and Sleep disorder by clicking a point. We'll reduce the opacity of any of the selected points using the alt.condition function.

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

alt.Chart(data).mark_circle().encode(
    x = "Daily Steps",
    y = "Sleep Disorder",
    color=alt.Color('BMI Category', scale=alt.Scale(scheme='spectral')),
    size="Age",
    tooltip=["Occupation", "Sleep Disorder"],
    opacity=alt.condition(selection,alt.value(1),alt.value(.2))
).add_selection(selection)
```



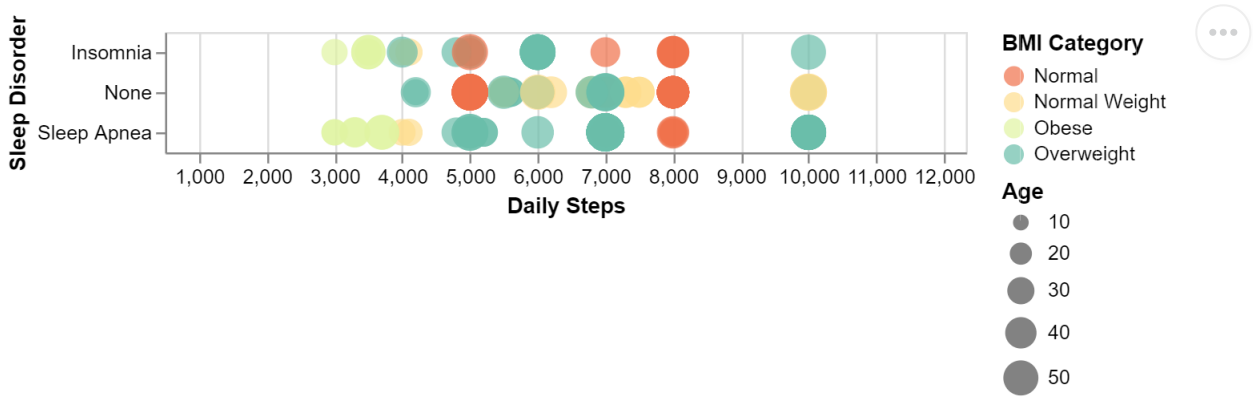
Clicking can be a bit cumbersome, so can switch to a mouseover to be faster.

```
selection = alt.selection(type='multi', fields=['Occupation'], on='mouseover', nearest=True)

alt.Chart(data).mark_circle().encode(
    x = "Daily Steps",
    y = "Sleep Disorder",
    color=alt.Color('BMI Category', scale=alt.Scale(scheme='spectral')),
    size="Age",
    tooltip=["Occupation", "Sleep Disorder"],
    opacity=alt.condition(selection,alt.value(1),alt.value(.2))
).add_selection(selection)
```



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()

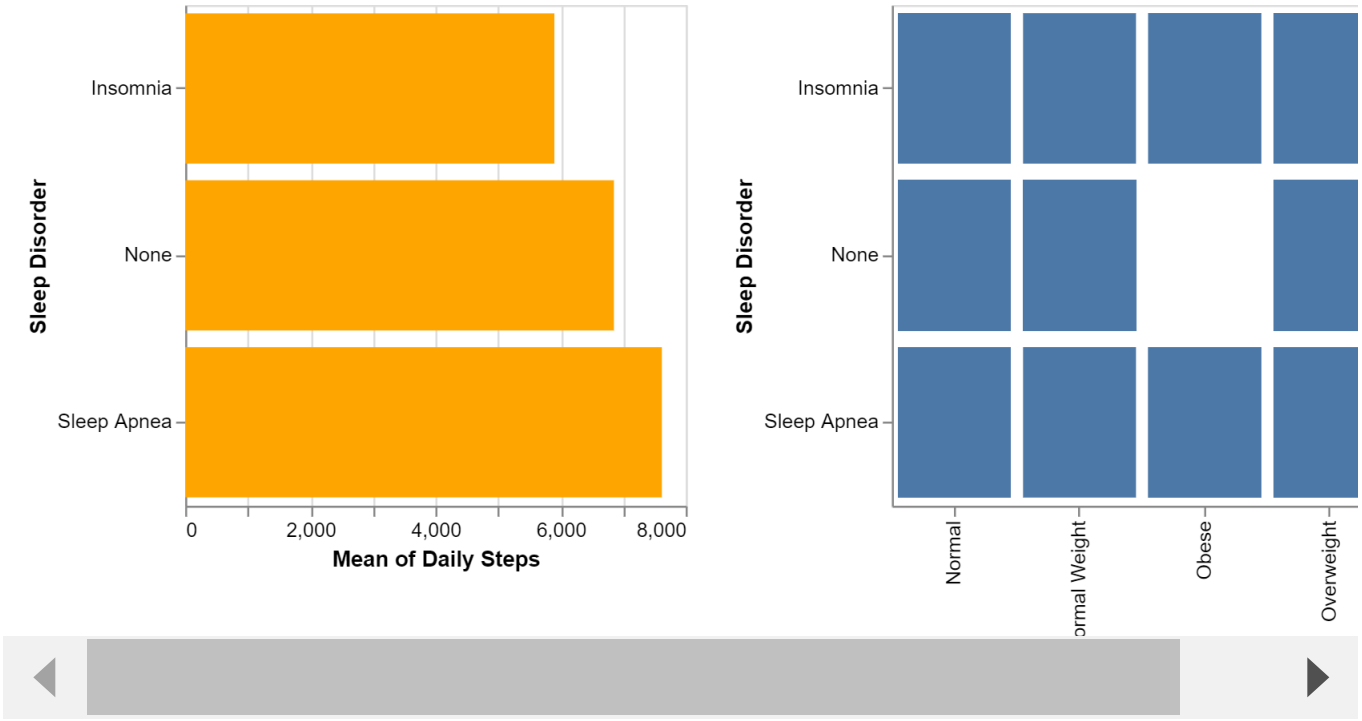
```
alt.Chart(data).mark_circle().encode(
    x = "Daily Steps",
    y = "Sleep Disorder",
    color=alt.Color('BMI Category', scale=alt.Scale(scheme='spectral')),
    size="Age",
    tooltip=["Occupation", "Sleep Disorder"]
).interactive()
```



We can change the level of detail we explore in a visualization using abstract and elaborate interactions. Semantic zooming is a common way to do this. In the chart below, we'll aggregate or disaggregate mean(Daily Steps) based on their BMI Category.

```
# Let's implement filtering using dynamic queries.
selection = alt.selection(type="multi", fields=["Region"])

# Create a container for our two different views
base =  alt.Chart(data).properties(width=500, height=250)

# Let's specify our overview chart
overview = alt.Chart(data).mark_bar().encode(
    y = "Sleep Disorder",
    x = "mean(Daily Steps)",
    color=alt.condition(selection, alt.value("orange"), alt.value("lightgrey"))
).add_selection(selection).properties(height=250, width=250)

# Create a detail chart
detail = hist = base.mark_bar().encode(
    y = "Sleep Disorder",
```

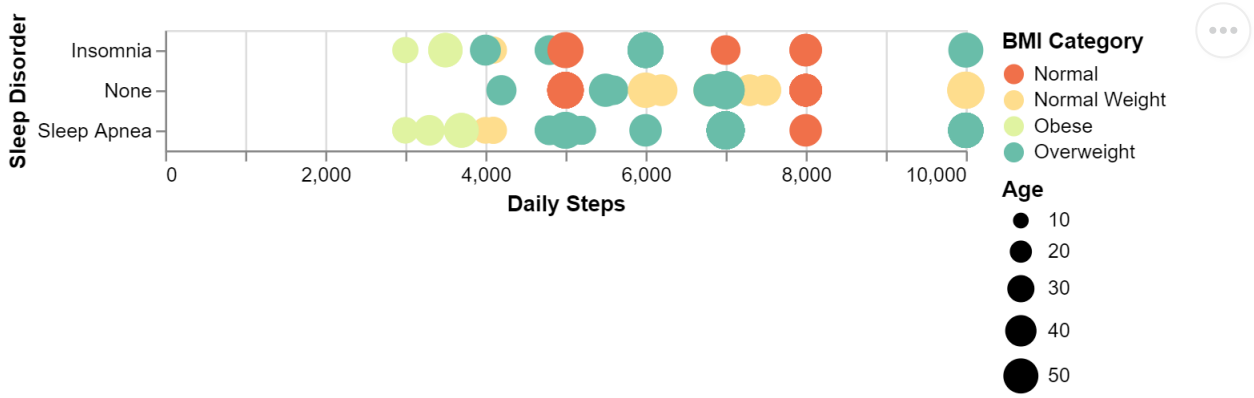
```
    x = "BMI Category"
).transform_filter(selection).properties(height=250, width=250)

overview | detail
```



While our selection implementation allowed us to filter our data based on the points we selected, we can also use other UI elements to achieve the same filtering without selecting individual values. For example, we can filter by BMI Category by binding the selection to the legend. In this case, we can select a BMI Category of interest by clicking on that BMI Category in the legend

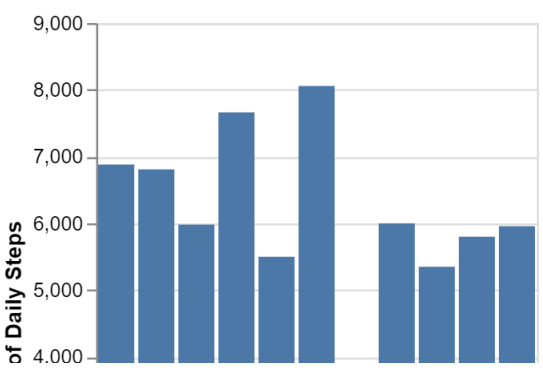
```
selection = alt.selection(type='multi', fields=['BMI Category'], bind='legend')
alt.Chart(data).mark_circle().encode(
    x = "Daily Steps",
    y = "Sleep Disorder",
    color=alt.Color('BMI Category', scale=alt.Scale(scheme='spectral')),
    size="Age",
    tooltip=["Occupation", "Sleep Disorder"],
    opacity=alt.condition(selection,alt.value(1),alt.value(.2))
).add_selection(selection)
```


```
# dropdown = alt.binding_select (options=data["Occupation"].unique(),name="Select a BMI Categ

selection = alt.selection(type='multi', fields=['BMI Category'], bind='legend')

alt.Chart(data).mark_circle().encode(
    x = "Daily Steps",
    y = "Sleep Disorder",
    color=alt.Color('BMI Category', scale=alt.Scale(scheme='spectral')),
    size="Age",
    tooltip=["Occupation", "Sleep Disorder"],
    opacity=alt.condition(selection,alt.value(1),alt.value(.2))
).add_selection(selection)
```



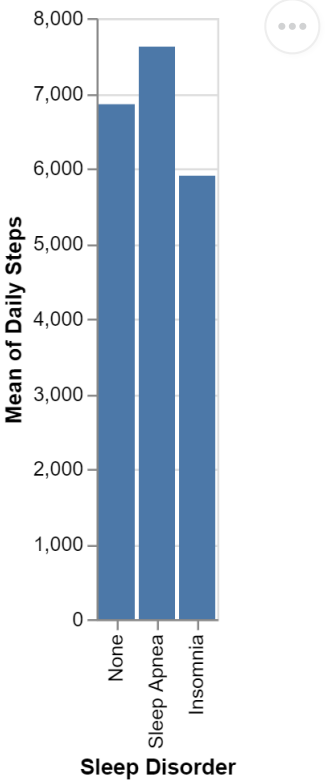
To reconfigure data in Altair, we'll just use Altair's basic sort parameter. Let's start with a standard bar chart. You can see for categorical attributes, Altair will sort the data alphabetically by default.

```
alt.Chart(data).mark_bar().encode(
    y = "mean(Daily Steps)",
    x = "Occupation"
)
```

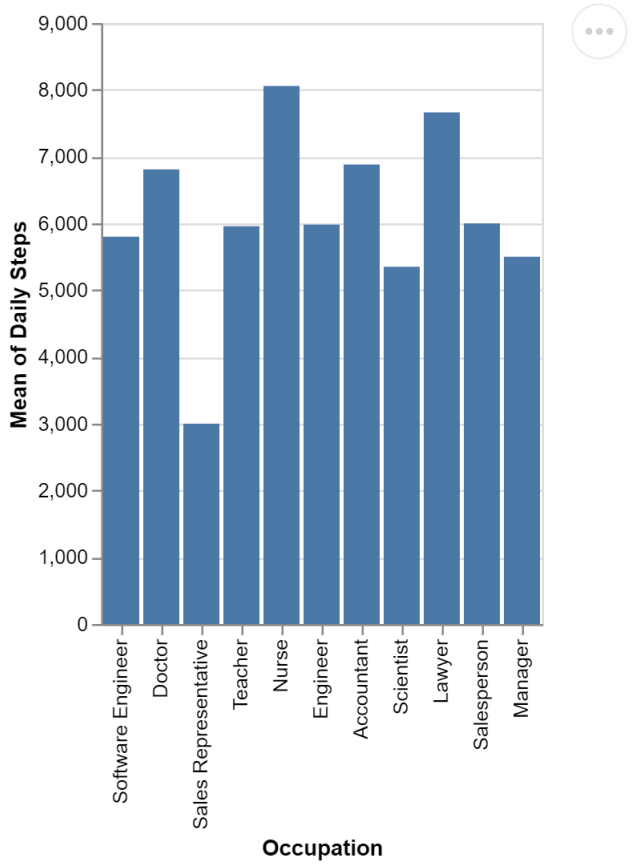


Now let's take same other data and reorder it according to some prespecified aspect of our data. For example, let's say that we want to look at how the mean(Daily Steps) change from the lowest to highest with Sleep Disorder. We can sort the data by the mean(Daily Steps) by Sleep Disorder using the EncodingSortField function. You can try changing the sorting order from ascending to descending using the order parameter of EncodingSortField.

```
alt.Chart(data).mark_bar().encode(
    y = "mean(Daily Steps)",
    x = alt.X(field='Sleep Disorder', type='nominal', sort=alt.EncodingSortField(field='BMI C
)
```



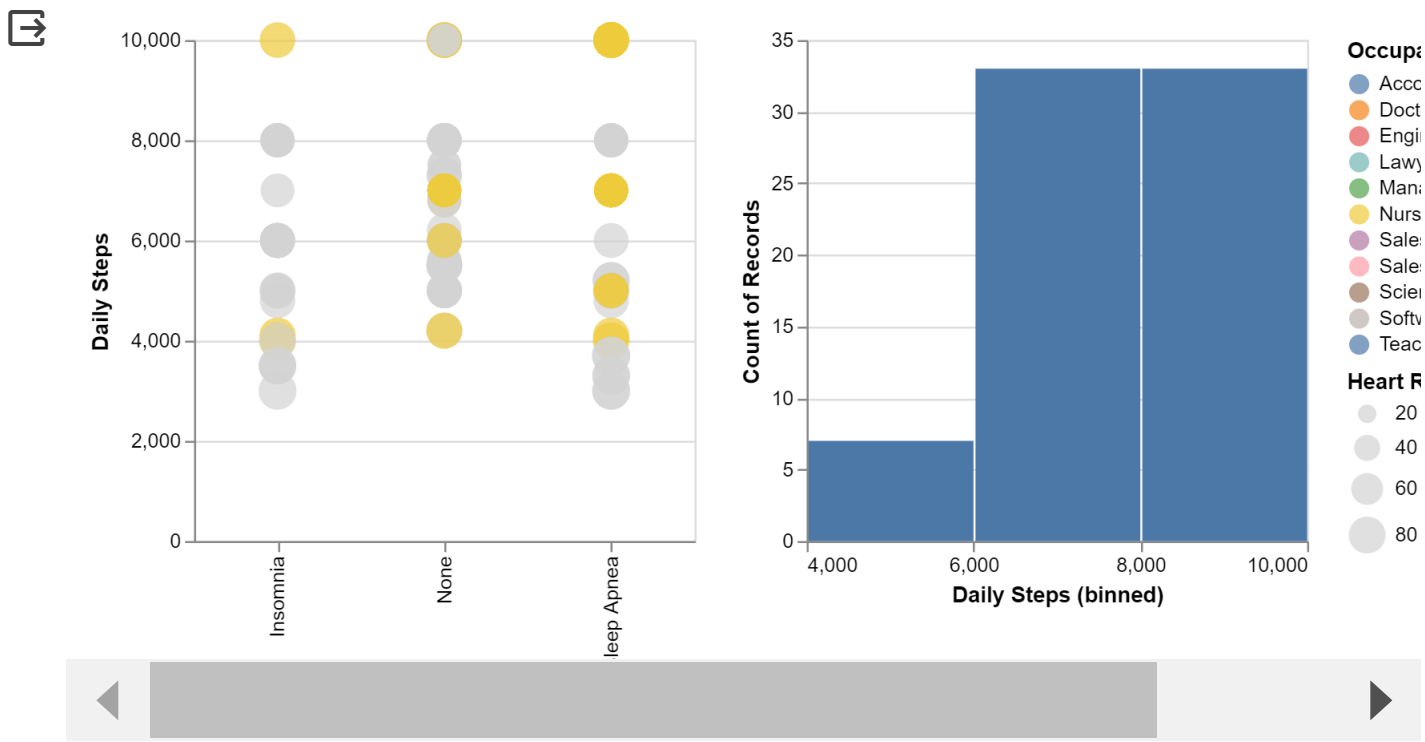

```
alt.Chart(data).mark_bar().encode(
    y = "mean(Daily Steps)",
    x = alt.X(field='Occupation', type='nominal', sort=alt.EncodingSortField(field='BMI Categ
)
```



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 Occupation. We can use the transform_filter function to filter based on the value of the selection.

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

# 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 = 'Sleep Disorder',
    y = 'Daily Steps',
    size = "Heart Rate",
    color = alt.condition(selection, "Occupation", alt.value('lightgray'))
).add_selection(selection)

# Create a histogram
hist = base.mark_bar().encode(
    x = alt.X("Daily Steps", bin=alt.Bin(maxbins=5)),
    y = "count()"
```

```
).transform_filter(selection)

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



We can make the selection a little more specific to filter for sets of Occupation 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.

```
# 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 = 'Sleep Disorder',
    y = 'Daily Steps',
    size = "Heart Rate",
    color = alt.condition(selection, "Occupation", 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("Daily Steps", 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
```