Bokeh

Mercer 2018

What is Bokeh?

- a library for interactive visualisations,
- it transalets Python code to java script,
- you can not only create plots with it, but also mini-applications,
- it can output html documents that you can easily share.

How to start?

Bokeh has a few basic functions that you need to import to be able to use it:

- `figure` from `bokeh.plotting` that allows to create a new plot object,
- `output_file` or `output_notebook` from `bokeh.io` that control how Bokeh outputs plots,
- `save` and `show` from `bokeh.io` to save and display plots.

In [34]:
```
from bokeh.io import output_notebook, show
from bokeh.plotting import figure
output_notebook() # you need to call it once per session to display plots in a given not
ebook
```

Bokeh plots in 4 steps

1. Create a figure object.
2. Add glyphs.
3. Customize the plot.
4. Save or display it.

1. Create a figure object

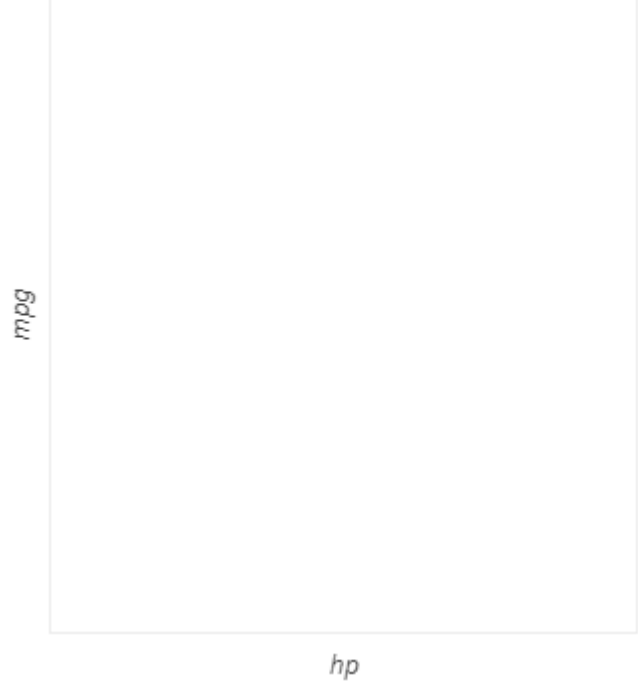
`figure()` function creates an empty Figure object, which is an empty plotting area. You can add layers of graphics to it.

In [35]:
```
plot = figure(width = 250, height = 250); show(plot)
```

`figure()` parameters

When calling `figure()` function you can specify x and y axis labels, their appearance, interactive tools available, tootltips and many other options.

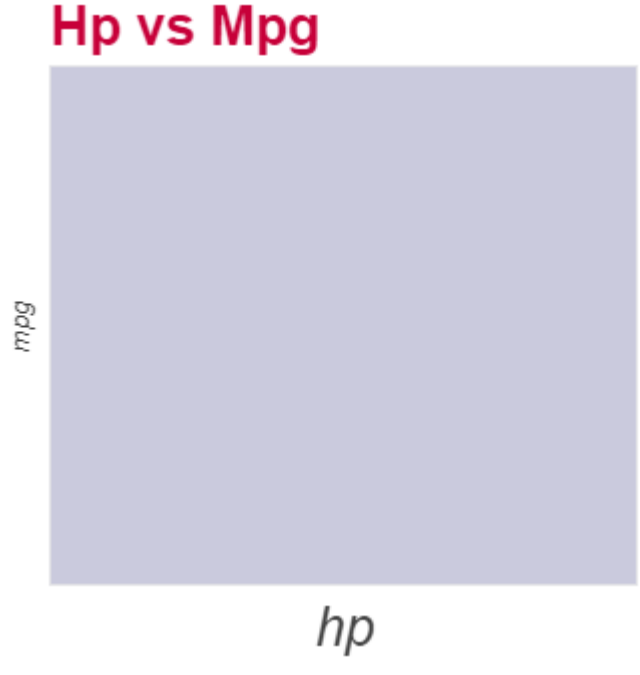
In [36]:
```
plot = figure(x_axis_label = 'hp', y_axis_label = 'mpg', plot_width = 350, plot_height =
 350,
              tools = [])
show(plot)
```



Changing visual properties of the plot

You can also change visual properties of the plot later by setting specific attributes of the Figure object.

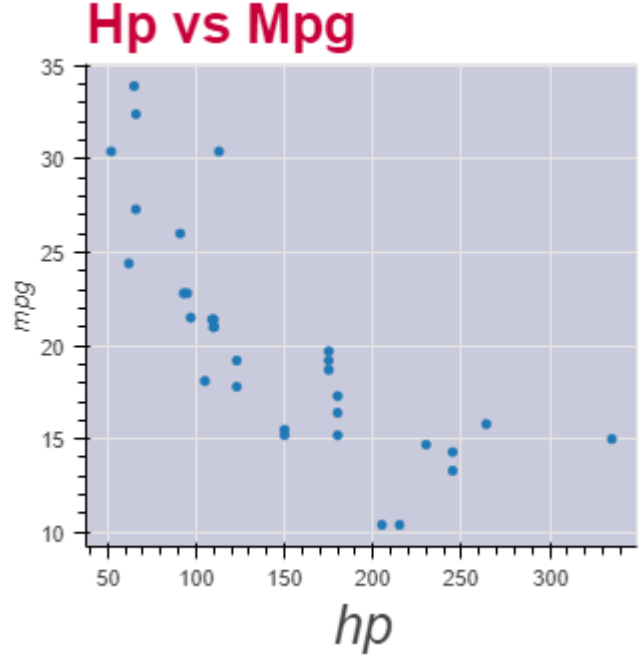
In [37]:
```
plot.background_fill_color = '#CACADD'; plot.title.text = 'Hp vs Mpg'
plot.title.text_font = 'arial'; plot.title.text_font_size = '20pt'
plot.title.text_color = '#C70039'; plot.xaxis.axis_label_text_font_size = '20pt';
show(plot)
```



2. Add glyphs

Glyphs are visual building blocks for your plots. They are methods, which you can use to draw dots, lines, squares and other shapes on your plotting area.

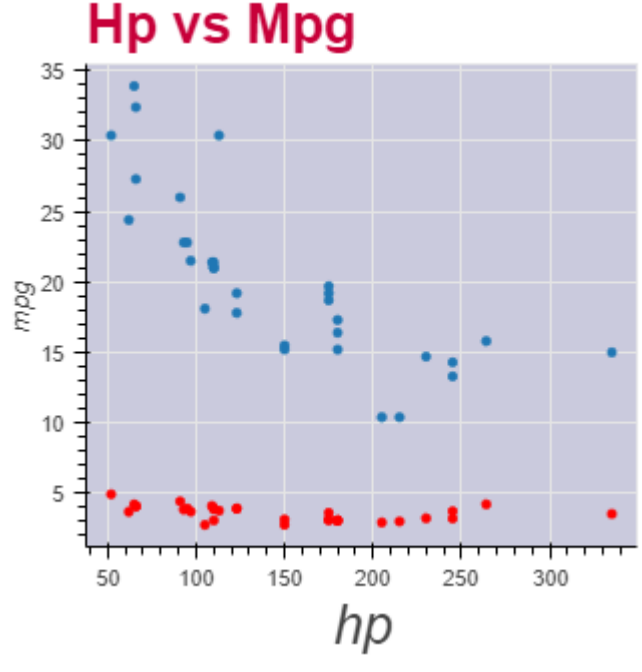
In [38]:
```
import pandas as pd
mtcars = pd.read_csv('../data/mtcars.csv')
plot.circle(x = 'hp', y = 'mpg', source = mtcars)

show(plot)
```



Calling glyph methods

Every time you call a glyph method on the figure object you are adding another layer of graphics onto it.

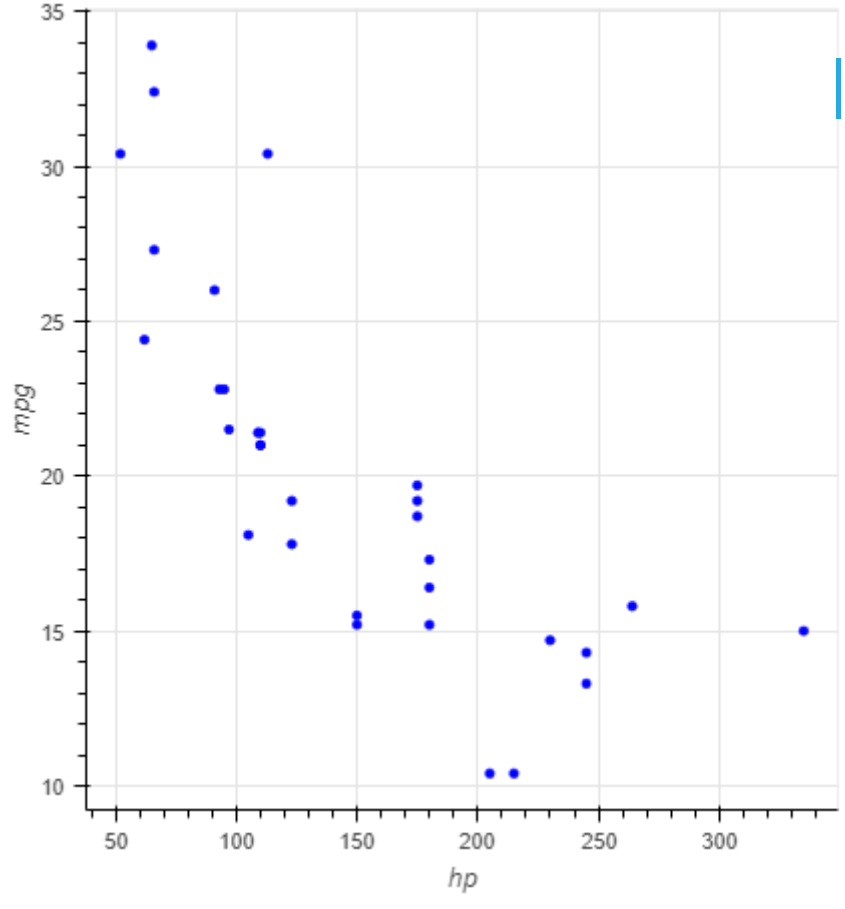
In [39]:
```
plot.circle(x = 'hp', y = 'drat', color = 'red', source = mtcars)
show(plot)
```



Resetting the figure

If you want reset your graph you need to create a new figure object.

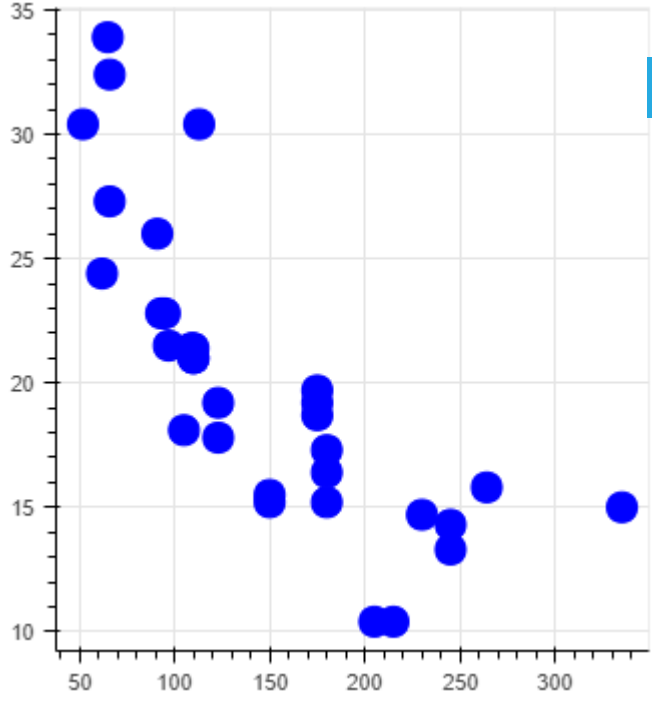
In [40]:
```
plot = figure(x_axis_label = 'hp', y_axis_label = 'mpg', plot_width = 450, plot_height =
 450)
plot.circle(x = 'hp', y = 'mpg', color = 'blue', source = mtcars)
show(plot)
```



Changing a glyph by resetting a figure

It is especially important if you decide to change visual aspects of the same glyph. Let's imagine you create the plot with large blue dots.

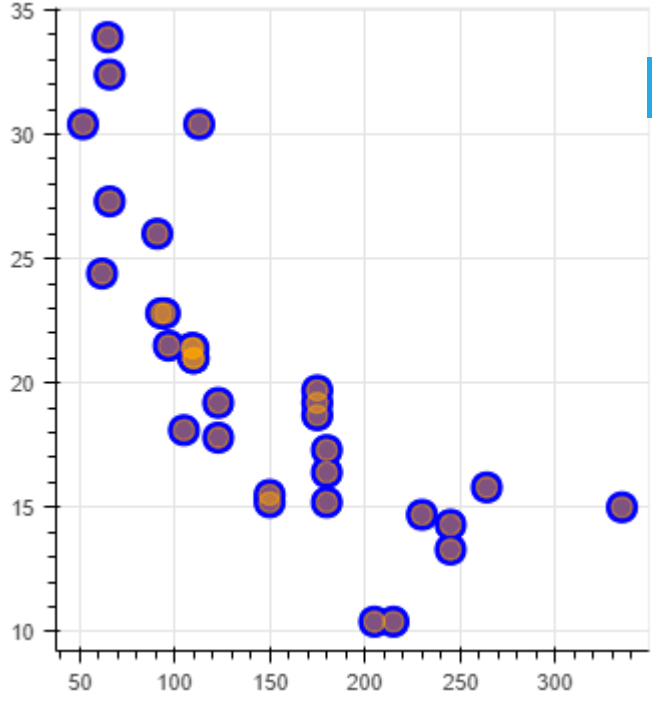
In [41]:
```
plot = figure(plot_width = 350, plot_height = 350)
plot.circle(x = 'hp', y = 'mpg', size = 15, color = 'blue', source = mtcars)
show(plot)
```



Changing a glyph by resetting a figure

But after a while you come to the conclusion that it looks pretty horrible, and decide that points should be orange, smaller, and more transparent. So you run the following command. What will happen?

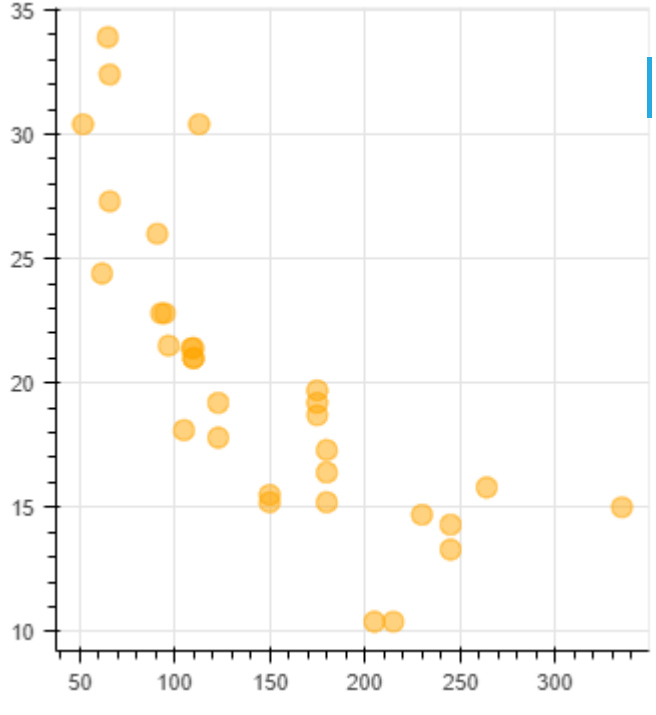
In [42]:
```
plot.circle(x = 'hp', y = 'mpg', color = 'orange', size = 10, alpha = 0.5, source = mtca
rs)
show(plot)
```



Changing a glyph by resetting a figure

To avoid this kind of situation you need to create a figure object every time you want to change a Glyph using a method.

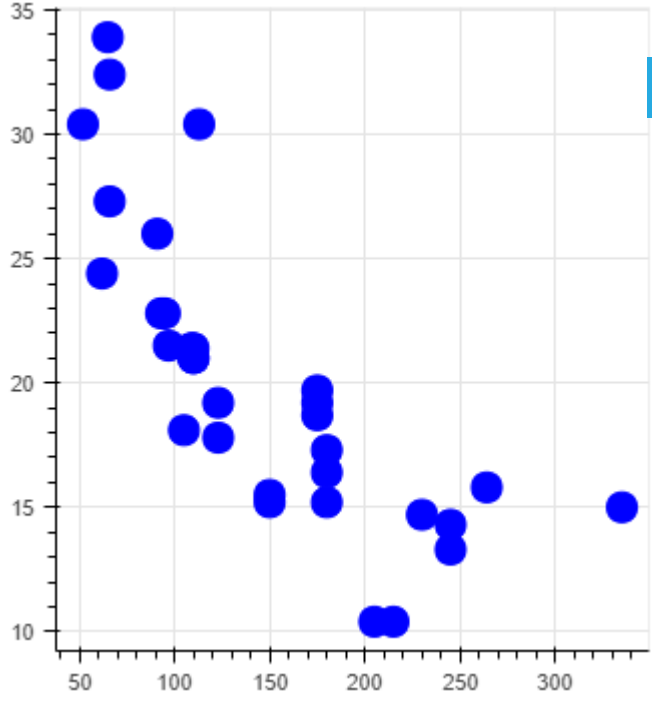
In [43]:
```
plot = figure(plot_width = 350, plot_height = 350)
plot.circle(x = 'hp', y = 'mpg', color = 'orange', size = 10, alpha = 0.5, source = mtca
rs)
show(plot)
```



Changing glyph properties

Alternatively, with the Glyph already put in place...

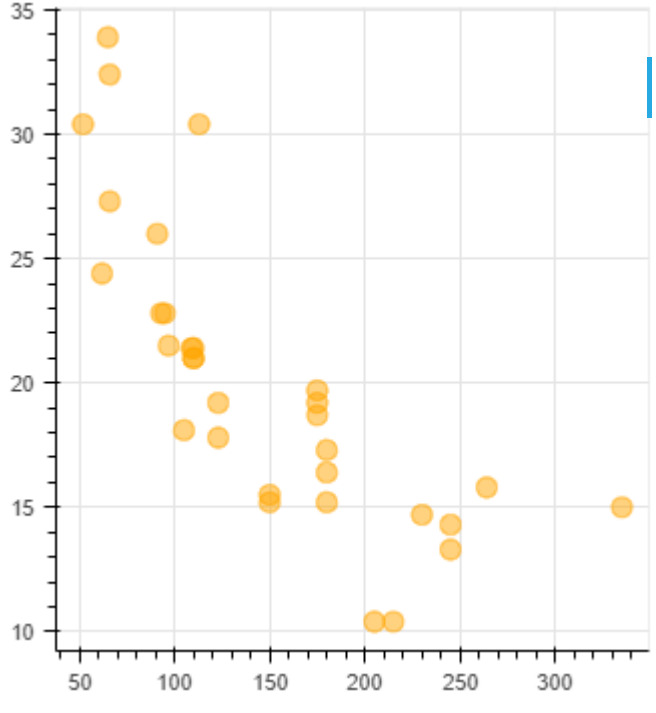
In [44]:
```
plot = figure(plot_width = 350, plot_height = 350)
r = plot.circle(x = 'hp', y = 'mpg', size = 15, color = 'blue', source = mtcars)
show(plot)
```



Changing glyph properties

... you could change Glyph properties.

In [45]:
```
r.glyph.fill_color = 'orange'
r.glyph.line_color = 'orange'
r.glyph.fill_alpha = .5
r.glyph.line_alpha = .5
r.glyph.size = 10
show(plot)
```



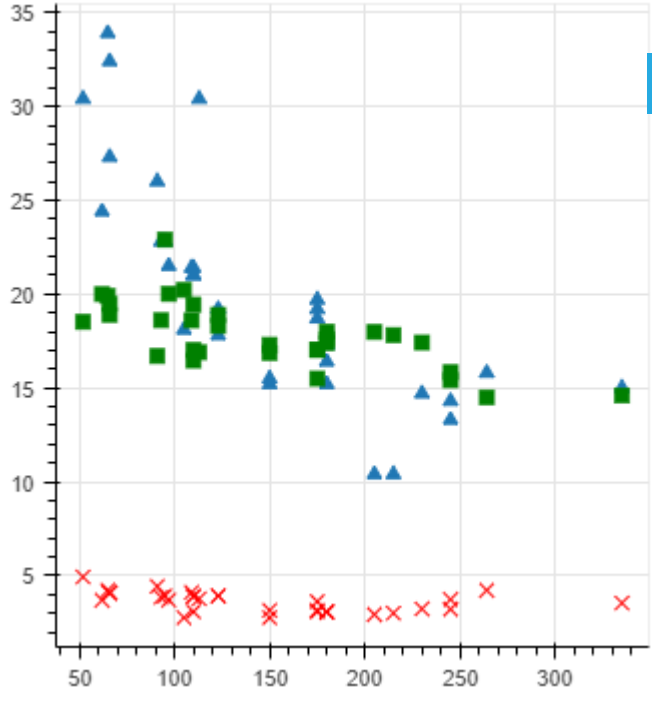
Types of Glyphs

Plotting markers

We have already seen that you can create a scatter plot with `circles()` method. It is just one of the option, as Bokeh offers several types of markers that you can choose from. Every available market type has it's own method like `asterisk()`, `diamond()`, `square()`, `traingle()`,or `x()`.

Plotting markers

In [46]:
```
plot = figure(width = 350, height = 350)
plot.triangle(x = 'hp', y = 'mpg', source = mtcars, size = 7)
plot.x(x = 'hp', y = 'drat', source = mtcars, color = 'red', size = 7)
plot.square(x = 'hp', y = 'qsec', source = mtcars, color = 'green', size = 7)
show(plot)
```

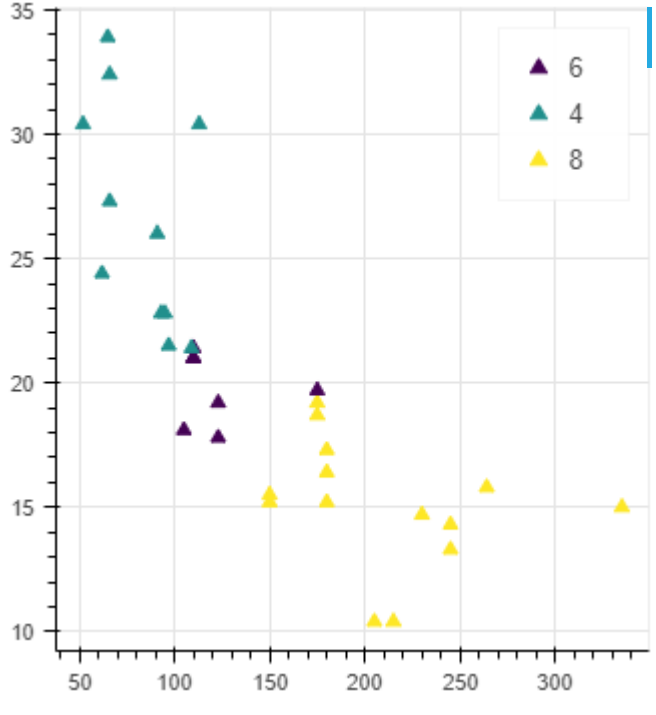


Mapping a categorical variable into a color

To map some categorical variable to markers' color, you need to create a `CategoricalColorMapper` first, and then pass it to `color` parameter as a part of a dictionary. A variable used for mapping must be a string.

Mapping a categorical variable into a color

In [14]:
```
from bokeh.palettes import Viridis
from bokeh.models import CategoricalColorMapper
mtcars['cyl_str'] = mtcars.cyl.apply(str)
col_map = CategoricalColorMapper(factors = mtcars.cyl_str.unique(), palette = Viridis[3
])
plot = figure(plot_width = 350, plot_height = 350)
plot.triangle(x = 'hp', y = 'mpg', color = {'field': 'cyl_str', 'transform': col_map},
              source = mtcars, size = 7, legend = 'cyl')
plot.toolbar.logo = None; show(plot)
```



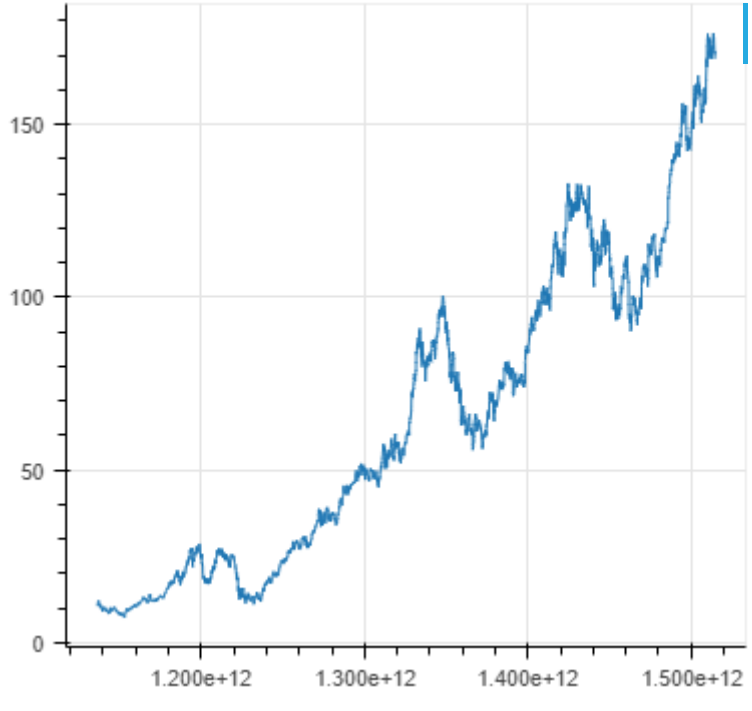
Exercise 1 (7 min)

Create a plot that will show the relation between the budget and the revenue (both in millions) of english movies released in years 2014, 2015, and 2016. Choose markers that you like the most. Color them according to the year. Set alpha to 0.5. Display the legend in the top left corner. Change it's orientation to horizontal.

Line plots

Line plots can be created similarily to the plots with markers but with `line()` method instead.

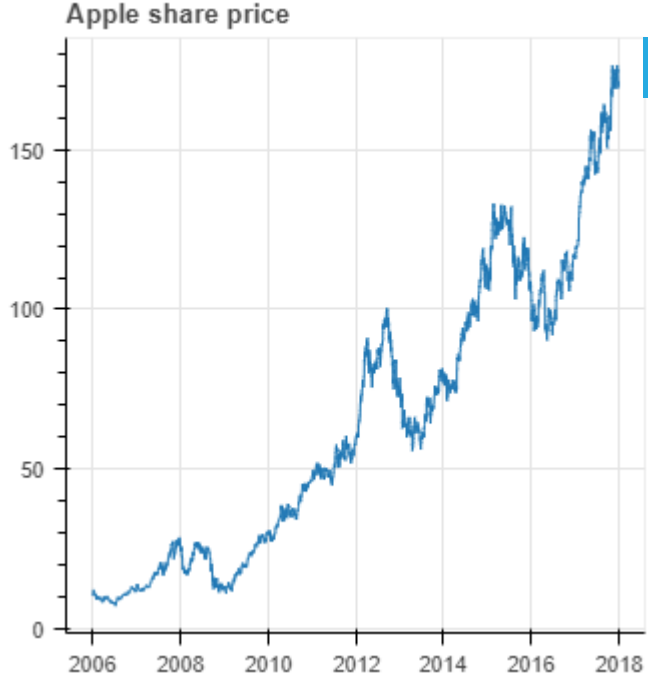
In [15]:
```
aapl = pd.read_csv('../data/AAPL_2006-01-01_to_2018-01-01.csv', parse_dates = [0])
aapl_plot = figure(plot_width=400, plot_height=350)
aapl_plot.line(x = 'Date', y = 'Close', source = aapl)
aapl_plot.toolbar.logo = None; show(aapl_plot)
```



Line plots

To propely display dates on x axis you need to set `x_axis_type` parameter in `figure()` function to 'datetime'.

In [16]:
```
aapl_plot = figure(x_axis_type="datetime", plot_width=350,
                   plot_height=350, title = 'Apple share price')
aapl_plot.line(x = 'Date', y = 'Close', source = aapl)
aapl_plot.toolbar.logo = None
show(aapl_plot)
```



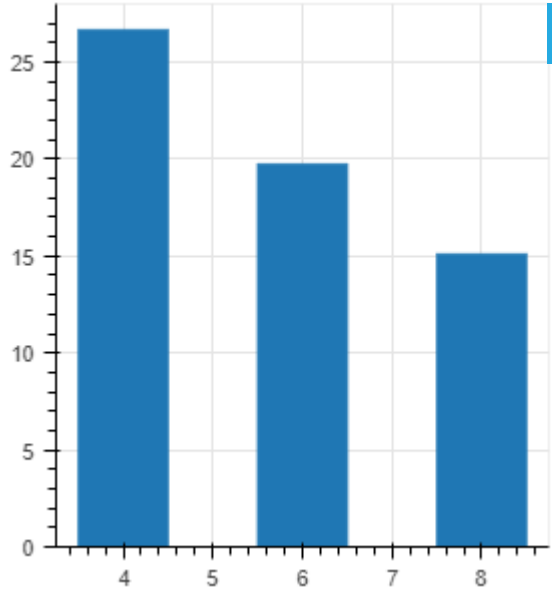
Exercise 2 (5 min)

Compare the number of French, German and Italian movies released in years 2010-2016. Plot the data for each country as a seperate line.

Bar plots

`vbar()` method creates a vertical bar plot. You need to specify `x`, `top` (bars' heights), and `width` parameters.

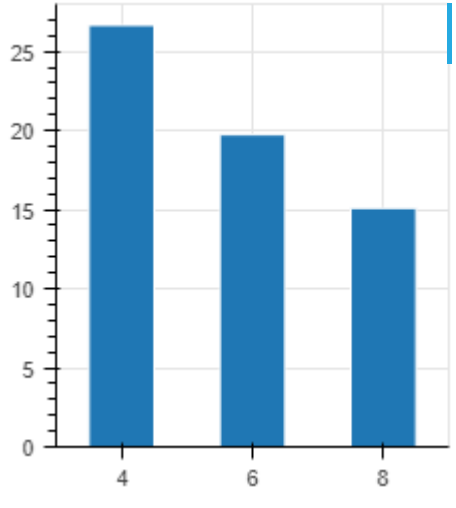
In [17]:
```
result = mtcars.groupby('cyl_str').mean().reset_index()
p = figure(plot_height = 300, plot_width = 300)
p.vbar(x='cyl', top='mpg', width = 1, source=result)
p.y_range.start = 0
p.toolbar.logo = None
show(p)
```



Bar plots

In the previous plot we have plotted cyl as a continuous variable, which adds to x-axis unnecessary ticks. We can plot it as categorical variable to make it look better, but we need to define `x_range` parameter during the creation of figure object to make it work.

In [18]:
```
p = figure(x_range = result['cyl_str'], plot_height = 250, plot_width = 250)
p.vbar(x = 'cyl_str', top = 'mpg', width = 0.5, source = result,
       line_color = "white", line_width = 1)
p.y_range.start = 0; p.toolbar.logo = None; show(p)
```

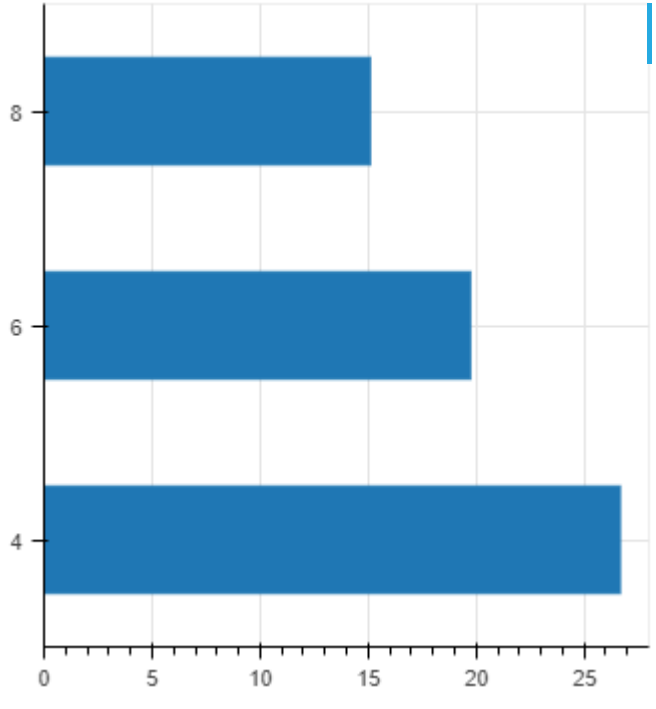


Horizontal bar plots

`hbar()` works similarly to `vbar()` but it creates a horizontal bar plot.

`y`, `right`, and `width` parameters are required.

In [19]:
```
ph = figure(y_range = result['cyl_str'], plot_height = 350, plot_width = 350)
ph.hbar(y='cyl_str', right='mpg', height = .5, source=result)
ph.x_range.start = 0; ph.toolbar.logo = None; show(ph)
```



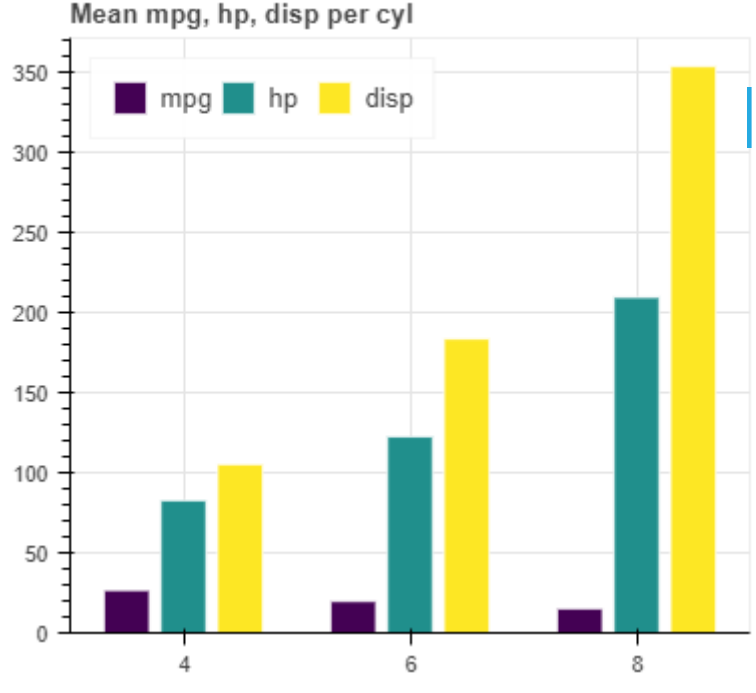
Multiple bars per category

It is also possible to plot more than one bar for each category, but you need to specify the relative position of each bar.

Multiple bars per category

In [53]:

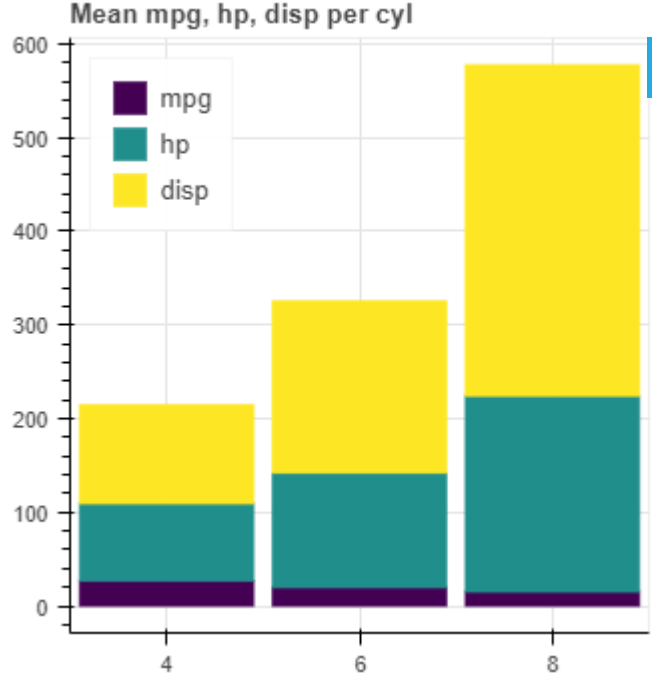
```
from bokeh.transform import dodge; from bokeh.core.properties import value
p = figure(x_range = result['cyl_str'], plot_height = 350, plot_width = 400,
           title = 'Mean mpg, hp, disp per cyl')
p.vbar(x=dodge('cyl_str', -0.25, range = p.x_range), top='mpg', color = Viridis[3][0],
       width = 0.2, source=result, line_color="white", line_width = 1, legend = value('m
pg'))
p.vbar(x=dodge('cyl_str', 0, range = p.x_range), top='hp', color = Viridis[3][1],
       width = 0.2, source=result, line_color="white", line_width = 1, legend = value('h
p'))
p.vbar(x=dodge('cyl_str', 0.25, range = p.x_range), top='disp', color = Viridis[3][2],
       width = 0.2, source=result, line_color="white", line_width = 1, legend = value('d
isp'))
p.y_range.start = 0; p.legend.location = "top_left"; p.legend.orientation = "horizontal"
; show(p)
```



Stacked bar chart

You can also stack bars on each other using `vbar_stack()` method.

In [21]:
```
p = figure(x_range = result['cyl_str'], plot_height = 350, plot_width = 350,
           title = 'Mean mpg, hp, disp per cyl')
p.vbar_stack(['mpg', 'hp', 'disp'], x='cyl_str', width=0.9, color=Viridis[3], source=res
ult,
             legend = list(map(value, ['mpg', 'hp', 'disp'])))
p.legend.location = "top_left"
p.toolbar.logo = None; show(p)
```



Exercise 3 (10 min)

Create a bar chart that will show the number of Action, Horror, and Romance movies released in 2014-2016 in Italian, French, and German as the original language. On x-axis there should be abbreviated name of the language, and each movie genre should have a separate bar.

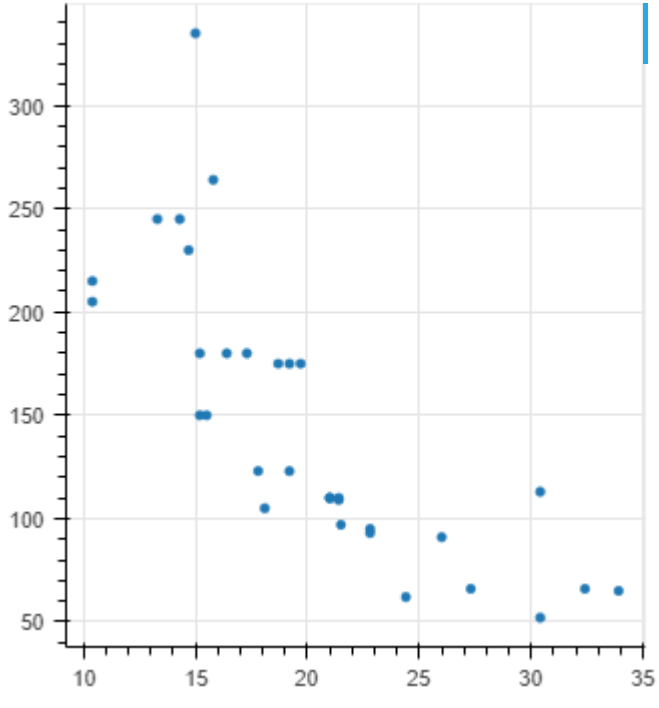
Tooltips

Tooltips

One of the most useful features of interactive plots are tooltips, that tell you the addtional information about a given observation. To add them to your Bokeh plot, you need to create a template for them in the form of list of tuples first, and then pass it to `tooltips` parameter in `figure` function.

Tooltips

In [22]:
```
tooltips = [('Miles per gallon', '@mpg'),
            ('Horsepower', '@hp'),
            ('Cylinders', '@cyl')]
p = figure(plot_width = 350, plot_height = 350, tooltips = tooltips, tools = [])
p.circle('mpg', 'hp', source = mtcars); p.toolbar.logo = None; show(p)
```



Exercise 4 (5 min)

Add the tooltip to the plot from exercise one that will show the information on the title, budget, revenue, and vote_average.

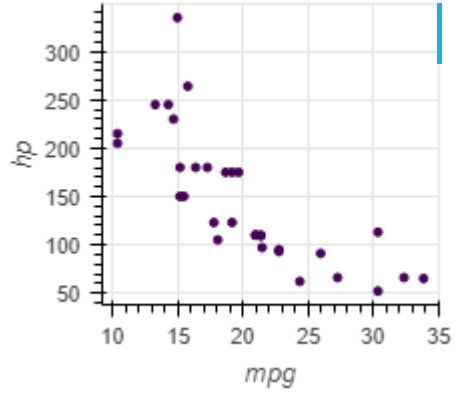
Layouts

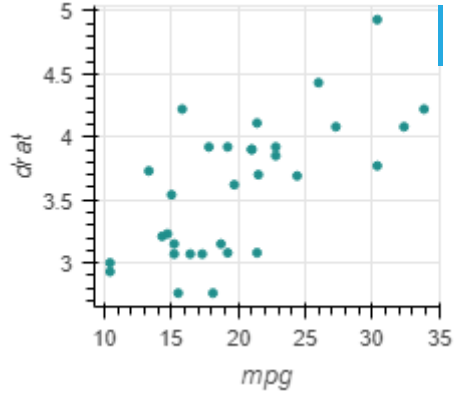
Layouts

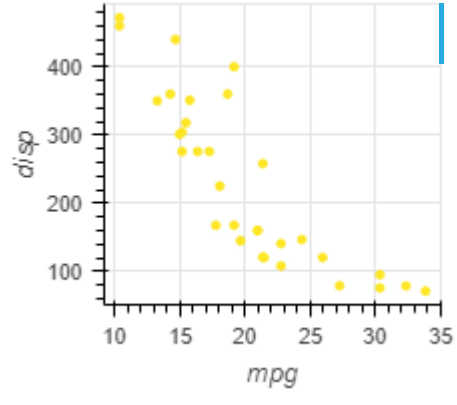
`row()` and `column()` functions can be used to combine many plots into one visualisation.

Layouts

In [23]:
```
from bokeh.layouts import row, column
p1 = figure(plot_width = 250, plot_height = 200, x_axis_label = 'mpg',
            y_axis_label = 'hp')
p1.circle('mpg', 'hp', color = Viridis[3][0],source = mtcars); p1.toolbar.logo = None
p2 = figure(plot_width = 250, plot_height = 200, x_axis_label = 'mpg',
            y_axis_label = 'drat')
p2.circle('mpg', 'drat', color = Viridis[3][1], source = mtcars); p2.toolbar.logo = None
p3 = figure(plot_width = 250, plot_height = 200, x_axis_label = 'mpg',
            y_axis_label = 'disp')
p3.circle('mpg', 'disp', color = Viridis[3][2], source = mtcars); p3.toolbar.logo = None
layout = row(p1, p2, p3); p.toolbar.logo = None; show(layout)
```



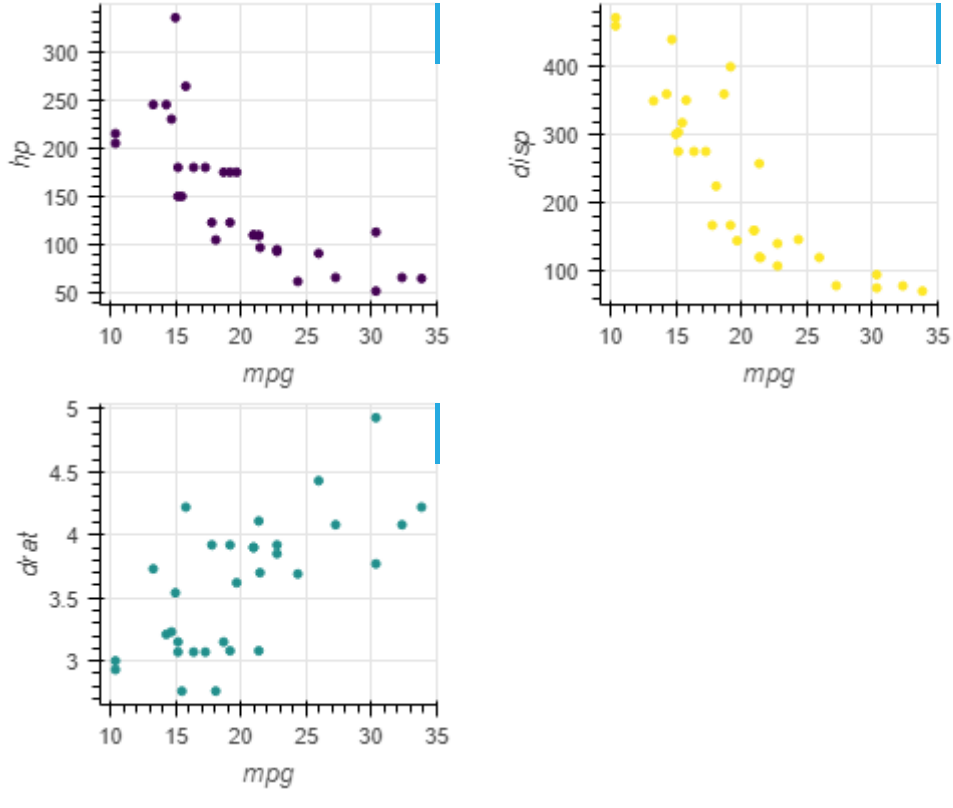




Layouts

You can nest `column()` and `row()` functions in one another to reach the desired layout.

In [24]:
```
layout = row(column(p1, p2), p3)
show(layout)
```

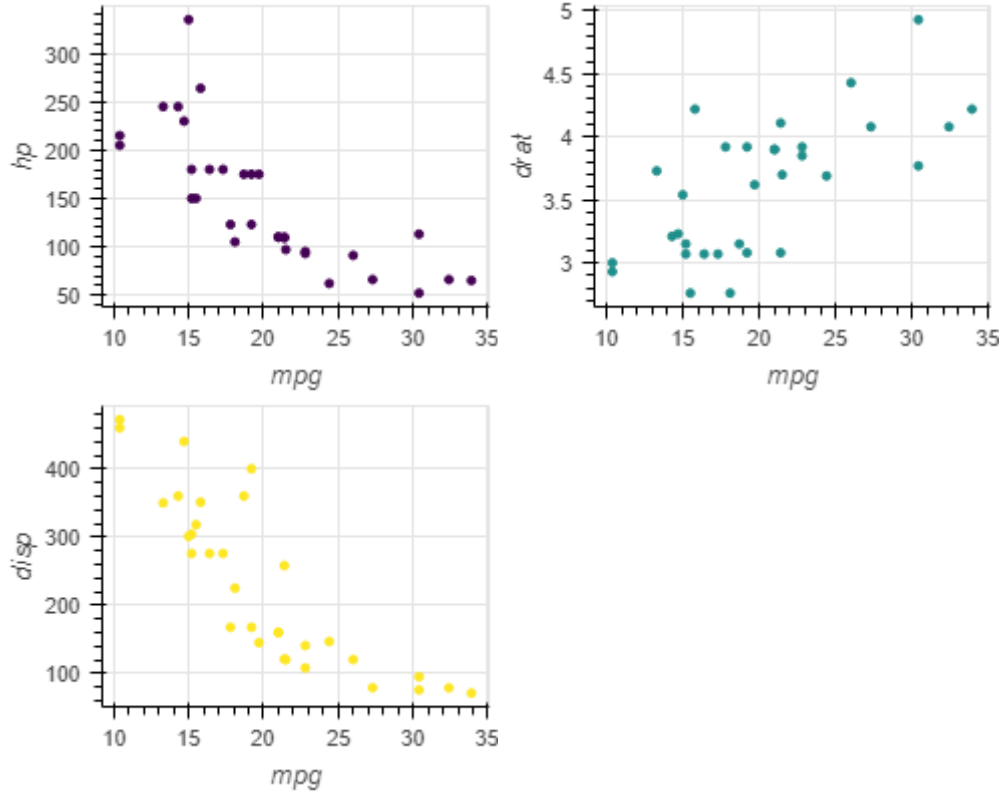


Creating layouts with `gridplot()`

The downside of using `row()` and `column()` functions is that separate toolbars are created for each plot. You can remove them, but if you want to keep them, it might be problematic. The better solution is to use `gridplot()`. You can also create a multiplot figure with it, and it handles the toolbar better.

Creating layouts with `gridplot()`

In [25]:
```
from bokeh.layouts import gridplot
grid = gridplot([[p1, p2], [p3, None]], toolbar_location = None); show(grid)
```



Tabs

Bokeh gives you the ability to divide you visualisations into tabs. You need to use `Panel()` function first to specify what will be in a given tab and then pass your Panels to `Tabs()` function.

Exercise 5 (5 min)

Put plots from Exercise 2 and 3 into separate tabs in one file.

ColumnDataSource

How Bokeh handles the data that is used in plots?

Bokeh can work well with Python lists, Numpy arrays and Pandas series. Under the hood, these inputs are converted into a special objects of class called **ColumnDataSource**. To use more sophisticated features of Bokeh (like apps) you need to work with ColumnDataSource directly.

How to create ColumnDataSource object?

You can create a ColumnDataSource object using the function of the same name.

In [26]:
```
from bokeh.models.sources import ColumnDataSource
data = ColumnDataSource(mtcars)
data
```

Out[26]: **ColumnDataSource**(id = '01644f5b-87b2-40b2-9338-cc1ffa633923', ...)

Let's take a closer look at ColumnDataSource object

ColumnDataSource object has a `column_name` attribute that can be used to check column names of the the data.

In [27]:
```
data.column_names
```

Out[27]:
```
['model',
 'mpg',
 'cyl',
 'disp',
 'hp',
 'drat',
 'wt',
 'qsec',
 'vs',
 'am',
 'gear',
 'carb',
 'cyl_str',
 'index']
```

Data in ColumnDataSource

Data inside the **ColumnDataSource** is stored as a dictionary with column names as keys and arrays as values.

In [28]:
```
data.data
```

Out[28]:
```
{'model': array(['Mazda RX4', 'Mazda RX4 Wag', 'Datsun 710', 'Hornet 4 Drive',
       'Hornet Sportabout', 'Valiant', 'Duster 360', 'Merc 240D',
       'Merc 230', 'Merc 280', 'Merc 280C', 'Merc 450SE', 'Merc 450SL',
       'Merc 450SLC', 'Cadillac Fleetwood', 'Lincoln Continental',
       'Chrysler Imperial', 'Fiat 128', 'Honda Civic', 'Toyota Corolla',
       'Toyota Corona', 'Dodge Challenger', 'AMC Javelin', 'Camaro Z28',
       'Pontiac Firebird', 'Fiat X1-9', 'Porsche 914-2', 'Lotus Europa',
       'Ford Pantera L', 'Ferrari Dino', 'Maserati Bora', 'Volvo 142E'],
      dtype=object),
 'mpg': array([21. , 21. , 22.8, 21.4, 18.7, 18.1, 14.3, 24.4, 22.8, 19.2, 17.8,
       16.4, 17.3, 15.2, 10.4, 10.4, 14.7, 32.4, 30.4, 33.9, 21.5, 15.5,
       15.2, 13.3, 19.2, 27.3, 26. , 30.4, 15.8, 19.7, 15. , 21.4]),
 'cyl': array([6, 6, 4, 6, 8, 6, 8, 4, 4, 6, 6, 8, 8, 8, 8, 8, 8, 4, 4, 4, 4, 8,
       8, 8, 8, 4, 4, 4, 8, 6, 8, 4], dtype=int64),
 'disp': array([160. , 160. , 108. , 258. , 360. , 225. , 360. , 146.7, 140.8,
       167.6, 167.6, 275.8, 275.8, 275.8, 472. , 460. , 440. ,  78.7,
        75.7,  71.1, 120.1, 318. , 304. , 350. , 400. ,  79. , 120.3,
        95.1, 351. , 145. , 301. , 121. ]),
 'hp': array([110, 110,  93, 110, 175, 105, 245,  62,  95, 123, 123, 180, 180,
       180, 205, 215, 230,  66,  52,  65,  97, 150, 150, 245, 175,  66,
        91, 113, 264, 175, 335, 109], dtype=int64),
 'drat': array([3.9 , 3.9 , 3.85, 3.08, 3.15, 2.76, 3.21, 3.69, 3.92, 3.92, 3.92,
       3.07, 3.07, 3.07, 2.93, 3.  , 3.23, 4.08, 4.93, 4.22, 3.7 , 2.76,
       3.15, 3.73, 3.08, 4.08, 4.43, 3.77, 4.22, 3.62, 3.54, 4.11]),
 'wt': array([2.62 , 2.875, 2.32 , 3.215, 3.44 , 3.46 , 3.57 , 3.19 , 3.15 ,
       3.44 , 3.44 , 4.07 , 3.73 , 3.78 , 5.25 , 5.424, 5.345, 2.2  ,
       1.615, 1.835, 2.465, 3.52 , 3.435, 3.84 , 3.845, 1.935, 2.14 ,
       1.513, 3.17 , 2.77 , 3.57 , 2.78 ]),
 'qsec': array([16.46, 17.02, 18.61, 19.44, 17.02, 20.22, 15.84, 20.  , 22.9 ,
       18.3 , 18.9 , 17.4 , 17.6 , 18.  , 17.98, 17.82, 17.42, 19.47,
       18.52, 19.9 , 20.01, 16.87, 17.3 , 15.41, 17.05, 18.9 , 16.7 ,
       16.9 , 14.5 , 15.5 , 14.6 , 18.6 ]),
 'vs': array([0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0,
       0, 0, 0, 1, 0, 1, 0, 0, 0, 1], dtype=int64),
 'am': array([1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0,
```

bokeh apps

How to create an app with bokeh?

Bokeh has the capability to create basic applications. To create such an application you just need to do a few things:

- import curdoc from bokeh.io,
- create plots and widgets that you want to show in your app,
- add callbacks that will update your plots,
- arrange plots and widgets into layouts,
- put `curdoc().add_root(layout)` with proper layout at the end of your code.

Running your Bokeh application

To start your Bokeh application you need to run a Bokeh server using `bokeh serve --show myapp.py` in your command line.

Exercise 6 (20 min)

Create a Bokeh application that will allow users to explore the data about the english movies, that meets the following criteria:

1. Users should be able to select one of four columns: popularity, vote_average, budget, revenue.
2. Show revenues and budget in millions.
3. Users should be able to choose a year from 2000 to 2017 (use slider). Show data for top 50 most popular movies in this year.
4. Add a tooltip that will show the title of a given movie.

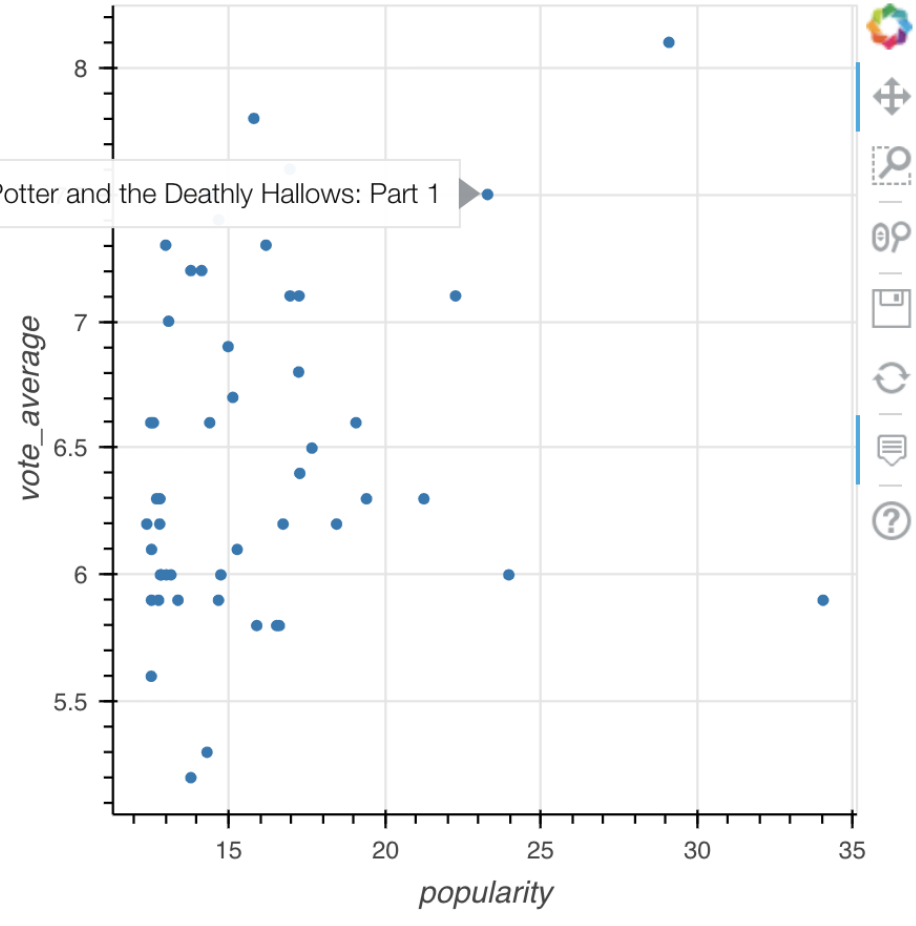
2010

Select x-axis data

popularity

Select y-axis data

vote_average



mb@idash.pl mo@idash.pl