Lecture12_Bokeh

April 10, 2024

1 Bokeh

Today, we will introduce Bokeh.

Matplotlib and seaborn create static graphics that are useful for quick and simple visualizations, or for creating publication quality images.

Bokeh creates visualisations for display on the web (whether locally or embedded in a webpage) and most importantly, the visualisations are meant to be highly interactive. Matplotlib does not offer either of these features.

If would you like to visually interact with your data in an exploratory manner or you would like to distribute interactive visual data to a web audience, Bokeh is the library for you!

If your main interest is producing finalised visualisations for publication, matplotlib or seaborn may be better, although Bokeh does offer a way to create static graphics.

Set up the session by importing the modules needed and the data needed (Phoenix Park weather data).

Notice that we import figure, output_file, show, output_notebook from bokeh.plotting.

[1]:
```python
import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import numpy as np

import os

from bokeh.plotting import figure, output_file, show, output_notebook

import bokeh.io

os.getcwd()

os.chdir("C:/Users/DKITStaff/OneDrive - Dundalk Institute of Technology/DKIT/
↪Programming for Data Analytics/Programming_2024_25/Datasets")
```

```
pp_weather = pd.read_csv("phoenix_park_weather.csv", skiprows = 13)

pp_weather.date = pd.to_datetime(pp_weather.date, format = '%d-%b-%y')

pp_weather.set_index(['date'], inplace = True)

pp_weather.sort_index(inplace = True)

print(pp_weather.head())
```

```
            ind  rain  ind.1  maxt  ind.2  mint  gmin soil
date
2009-01-01    0   0.0      0   4.5      1  -0.7  -4.7
2009-01-02    0   0.0      0   5.8      0   2.8   0.3
2009-01-03    0   0.0      0   4.7      0   3.1   1.5
2009-01-04    0   1.0      0   3.2      0   0.9  -2.1
2009-01-05    0   0.0      0   5.4      1  -4.1  -6.8
```

2 Basic rules for creating plots with the bokeh.plotting module

2.0.1 As per the Bokeh quickstart guide, the basic steps to creating plots with the bokeh.plotting module are:

1. Prepare some data

We have done this for the Phoenix Park data.

2. Tell Bokeh where to generate output

Using output_file(), with a filename eg "max_temp.html", or output_notebook() to view in Jupyter notebook.

3. Call figure()

This creates a plot with typical default options and easy customization of title, tools, and axes labels.

4. Add renderers

In this case, we use line() for our data, specifying visual customizations like colors, legends, and widths.

5. Ask Bokeh to show() or save() the results

These functions save the plot to an HTML file and optionally display it in a browser.

In Bokeh, the methods are called glyph methods.

3 Step 2: Output to a HTML or notebook file

I am outputting to the notebook here using output_notebook().

I am also outputting to a HTML file. For a HTML file, specify output_file() with a filename.

```
[2]: output_notebook()

     output_file("max_temp.html")
```

4 Step 3: Call figure()

4.0.1 Create a new plot called max_temp_plot with a title and axis labels.

```
[6]: max_temp_plot = figure(title="February and March 2011 maximum temperature in␣
      ↪Phoenix Park",
                           x_axis_label='Index', y_axis_label='Max temp (°C)')
```

5 Step 4: Add renderers

5.0.1 In this case, we use line() for our data, specifying a legend, and a line width.

5.0.2 Just use indexing from 1 to 59 for now. The date is not automatically formatted here.

```
[7]: #list(range(59))
```

```
[8]: max_temp_plot.line(x = list(range(59)), y = pp_weather['2011-02':'2011-03'].
      ↪maxt,
                       legend_label="Max temp", line_width = 1.5)
```

```
[8]: GlyphRenderer(id='1040', …)
```

6 Step 5: Ask Bokeh to show() or save() the results

6.0.1 Display the results. This will automatically open a browser window with the output if the output file is a HTML file.

```
[9]: show(max_temp_plot)
```

6.0.2 The legend slightly covers the final value. Although it is opaque and the graph can be panned, will change y limits. The y limits are changed using the y_range argument when specifying the figure.

6.0.3 The same process is followed but is now shown in one cell of code:

```
[10]: output_file("max_temp2.html")

      output_notebook()

      # create a new plot called p with a title and axis labels
      max_temp_plot = figure(y_range = [5, 18],
          title="February and March 2011 maximum temperature in Phoenix Park",
                             x_axis_label='Index', y_axis_label='Max temp (°C)')

      ### Add a line with legend and line width
      ### Just use indexing from 1 to 59 for now. The date is not automatically␣
      ↪formatted here.

      max_temp_plot.line(x = list(range(59)), y = pp_weather['2011-02':'2011-03'].
      ↪maxt, legend_label="Max temp", line_width = 1.5)

      ### Display the results. This will automatically open a browser window with the␣
      ↪output

      show(max_temp_plot)
```

6.0.4 We can include multiple lines in a Bokeh plot.

[12]:
```
output_file("max_min_temp.html")
```

7 Create a new plot called temp_plot with a title and axis labels.

7.0.1 You can also specify the tools you want included with this plot window within the figure argument:

[13]:
```
temp_plot = figure(tools="pan,box_zoom,reset,save",
                   title="February and March 2011 max and min temperature in␣
↪Phoenix Park",
                   x_axis_label='Day number', y_axis_label='Temperature (°C)')
```

7.0.2 Now we add two line glyph methods, one for max temp, one for min temp (colored red):

[14]:
```
temp_plot.line(x = list(range(59)), y = pp_weather['2011-02':'2011-03'].maxt,
               legend_label="Max temp", line_width = 2.5)

temp_plot.line(x = list(range(59)), y = pp_weather['2011-02':'2011-03'].mint,
               legend_label="Min temp", line_width = 2.5, line_color = 'red')
```

[14]:
```
GlyphRenderer(id='1551', …)
```

7.0.3 Add circles to show where the data points are on the lines:

[15]:
```
temp_plot.circle(x = list(range(59)), y = pp_weather['2011-02':'2011-03'].maxt,
                 legend_label="Max temp", fill_color = 'white', size = 6)

temp_plot.circle(x = list(range(59)), y = pp_weather['2011-02':'2011-03'].mint,
                 legend_label="Min temp", fill_color = 'red', size = 6)
```

[15]:
```
GlyphRenderer(id='1592', …)
```

7.0.4 Display the results. This will automatically open a browser window with the output if you are using HTML output.

[16]:
```
show(temp_plot)
```

7.0.5 Again, we must change the y_range.

The only thing that changes below is the y_range argument is now specified in figure.

[15]:
```
output_file("max_min_temp.html")

temp_plot = figure(tools="pan,box_zoom,reset,save", y_range = [-5, 20],
                   title="February and March 2011 max and min temperature in␣
 ↪Phoenix Park",
                   x_axis_label='Day number', y_axis_label='Temperature (°C)')

temp_plot.line(x = list(range(59)), y = pp_weather['2011-02':'2011-03'].maxt,
               legend_label="Max temp", line_width = 1.5)

temp_plot.line(x = list(range(59)), y = pp_weather['2011-02':'2011-03'].mint,
               legend_label="Min temp", line_width = 1.5, line_color = 'red')

temp_plot.circle(x = list(range(59)), y = pp_weather['2011-02':'2011-03'].maxt,
                 legend_label="Max temp", fill_color = 'white', size = 6)

temp_plot.circle(x = list(range(59)), y = pp_weather['2011-02':'2011-03'].mint,
                 legend_label="Min temp", fill_color = 'red', size = 6)

show(temp_plot)
```

8 ColumnDataSource

8.0.1 The Bokeh object ColumnDataSource links Pandas DataFrames with Bokeh.

8.0.2 ColumnDataSource accepts a Pandas DataFrame as an argument.

8.0.3 After it is created, the ColumnDataSource can then be passed to glyph methods (eg. circle, line) via the source parameter and other parameters, such as our x and y data, can then reference column names within our source.

[17]:
```
from bokeh.models import ColumnDataSource

output_file("max_min_temp_scatter.html")

source = ColumnDataSource(pp_weather)
```

8.0.4 Plot the max temp versus the min temp:

Here we do not include any arguments in figure(). We don't necessarily have to specify the axis titles when we initialise the figure. They can be specified later.

```
[19]: p = figure()

      p.circle(x='mint', y='maxt', source=source, size=10, color='blue')
```

```
[19]: GlyphRenderer(id='2028', …)
```

8.0.5 Now specify the title, axis labels etc:

```
[20]: p.title.text = 'Maximum temperature v minimum temperature at Phoenix Park'
      p.xaxis.axis_label = 'Minimum temperature (°C)'
      p.yaxis.axis_label = 'Maximum temperature (°C)'

      show(p)
```

8.0.6 How does this deal with dates?

Plot maximum temperature at Phoenix Park in February and March 2011 with date on the x axis.

```
[21]: output_file("pandas_example_dates.html")

      source = ColumnDataSource(pp_weather['2011-02':'2011-03'])

      p = figure()

      p.line(x = 'date', y = 'maxt', source=source, color='blue')

      p.title.text = 'Maximum temperature at Phoenix Park in February and March 2011'
      p.xaxis.axis_label = 'Dates'
      p.yaxis.axis_label = 'Maximum temperature (°C)'

      show(p)
```

8.0.7 We have to specify the x_axis_type to be dates so it is formatted correctly.

We add x_axis_type = 'datetime' as an argument when specifying the figure.

```
[22]: output_file("pandas_example_dates.html")

      source = ColumnDataSource(pp_weather['2011-02':'2011-03'])

      p = figure(x_axis_type = 'datetime')

      p.line(x = 'date', y = 'maxt', source=source, color='blue')
```

```
### We don't necessarily have to specify the axis titles when we initialise the␣
↪figure:

p.title.text = 'Maximum temperature at Phoenix Park in February and March 2011'
p.xaxis.axis_label = 'Dates'
p.yaxis.axis_label = 'Maximum temperature (°C)'

show(p)
```

8.0.8 Add the minimum temperature to this plot and add circles to both of the lines.

[23]:
```
output_file("pandas_example_dates.html")

source = ColumnDataSource(pp_weather['2011-02':'2011-03'])

p = figure(x_axis_type = 'datetime')

p.line(x = 'date', y = 'maxt', source=source, color='red')
p.line(x = 'date', y = 'mint', source=source, color='blue')
p.circle(x = 'date', y = 'maxt', source=source, color = 'red',␣
↪fill_color='white', size = 8)
p.circle(x = 'date', y = 'mint', source=source, color='blue',␣
↪fill_color='white', size = 8)

p.title.text = 'Maximum and minimum temperature at Phoenix Park in February and␣
↪March 2011'
p.xaxis.axis_label = 'Dates'
p.yaxis.axis_label = 'Temperature (°C)'

show(p)
```

9 The HoverTool

9.0.1 Add a hover Tool to a plot

9.0.2 HoverTool allows you to set a tooltips property which takes a list of tuples. The first part of the tuple is a display name and the second is a column name from your ColumnDataSource prefaced with @. Once we've set up the tool, we add it to the plot using the add_tool method.

9.0.3 Must import HoverTool first:

[24]:
```
from bokeh.models.tools import HoverTool
```

9.0.4 Now reproduce the last plot, but with a HoverTool included:

Below is the same code as the last plot, but we do not display the plot yet because we must specify the HoverTool.

[25]:
```
output_file("pandas_example_dates.html")

source = ColumnDataSource(pp_weather['2011-02':'2011-03'])

p = figure(x_axis_type = 'datetime')

p.line(x = 'date', y = 'maxt', source=source, color='red')
p.line(x = 'date', y = 'mint', source=source, color='blue')
p.circle(x = 'date', y = 'maxt', source=source, color = 'red',␣
  ↪fill_color='white', size = 12)
p.circle(x = 'date', y = 'mint', source=source, color='blue',␣
  ↪fill_color='white', size = 12)

p.title.text = 'Maximum and minimum temperature at Phoenix Park in February and␣
  ↪March 2011'
p.xaxis.axis_label = 'Dates'
p.yaxis.axis_label = 'Temperature (°C)'
```

9.0.5 First, call hover = HoverTool().

9.0.6 Then specify the tooltips element of hover as a list of tuples. The first part of the tuple is a display name and the second is a column name from your ColumnDataSource prefaced with @.

9.0.7 Finally, add the HoverTool to your plot: p.add_tools(hover)

p is the name of our plot, and hover is the name of the HoverTool we have set up.

[27]:
```
hover = HoverTool()
hover.tooltips=[
    ('Date', '@date'),
    ('Minimum temperature', '@mint'),
    ('Maximum temperature', '@maxt'),
    ('Rainfall', '@rain')
]

p.add_tools(hover)

show(p)
```

9.0.8 Again, the date is causing problems. It is not formatted correctly in the HoverTool.

9.0.9 A quick Google told me to use this: First, specify the date format inside curly brackets after giving the name of the date column: {%Y-%m-%d}. Then specify the formatters in hover.formatters. This is a dictionary that formats the keys (column names) as the values (in this case 'datetime').

Now the Date tuple in tooltips is: ('Date', '@date{%Y-%m-%d}'), and the formatters tell Bokeh that the date column ('@date') should be formatted as a datetime object.

[28]:
```
output_file("pandas_example_dates.html")

source = ColumnDataSource(pp_weather['2011-02':'2011-03'])

p = figure(x_axis_type = 'datetime')

p.line(x = 'date', y = 'maxt', source=source, color='red')
p.line(x = 'date', y = 'mint', source=source, color='blue')
p.circle(x = 'date', y = 'maxt', source=source, color = 'red',␣
 ↪fill_color='white', size = 8)
p.circle(x = 'date', y = 'mint', source=source, color='blue',␣
 ↪fill_color='white', size = 8)

p.title.text = 'Maximum temperature at Phoenix Park in February and March 2011'
p.xaxis.axis_label = 'Dates'
p.yaxis.axis_label = 'Maximum temperature (°C)'

hover = HoverTool()
hover.tooltips=[
    ('Date', '@date{%Y-%m-%d}'),
    ('Minimum temperature', '@mint'),
    ('Maximum temperature', '@maxt'),
    ('Rainfall', '@rain')
]
hover.formatters = {'@date': 'datetime'}

p.add_tools(hover)

show(p)
```

10 Sizing points in a scatterplot based on a variable in the dataset

10.0.1 Now for something a bit different. The next plot is our scatterplot of maximum temperature v minimum temperature at Phoenix Park. However, here the points will be sized based on the rainfall values: a larger rainfall value means a larger point.

Specify size = 'rain' in the circle glyph.

We include a HoverTool again.

[29]:
```
output_file("pandas_example_dates.html")

source = ColumnDataSource(pp_weather)

p = figure()

p.circle(x = 'mint', y = 'maxt', source=source, color = 'red', size = 'rain')

p.title.text = 'Maximum temperature v minimum temperature at Phoenix Park'
p.xaxis.axis_label = 'Minimum temperature (°C)'
p.yaxis.axis_label = 'Maximum temperature (°C)'

hover = HoverTool()
hover.tooltips=[
    ('Date', '@date{%Y-%m-%d}'),
    ('Minimum temperature', '@mint'),
    ('Maximum temperature', '@maxt'),
    ('Rainfall', '@rain')
]
hover.formatters = {'@date': 'datetime'}

p.add_tools(hover)

show(p)
```

10.0.2 I create a new column 'rain_colour', which is rain + 5 to size by. Why might I have done this?

The only change in the code is that size now equals the rain_colour column in the circle glyph.

[30]:
```
pp_weather['rain_size'] = pp_weather['rain'] + 5

output_file("pandas_example_dates.html")

source = ColumnDataSource(pp_weather)

p = figure()

p.circle(x = 'mint', y = 'maxt', source=source, color = 'red', size =␣
 ↪'rain_size')

p.title.text = 'Maximum temperature v minimum temperature at Phoenix Park'
p.xaxis.axis_label = 'Minimum temperature (°C)'
p.yaxis.axis_label = 'Maximum temperature (°C)'

hover = HoverTool()
```

```python
hover.tooltips=[
    ('Date', '@date{%Y-%m-%d}'),
    ('Minimum temperature', '@mint'),
    ('Maximum temperature', '@maxt'),
    ('Rainfall', '@rain')
]
hover.formatters = {'@date': 'datetime'}

p.add_tools(hover)

show(p)
```

[60]:
```python
from bokeh.models import LinearColorMapper, ColorBar

pp_weather['rain_size'] = pp_weather['rain'] + 5

output_file("pandas_example_dates.html")

source = ColumnDataSource(pp_weather)

color_mapper = LinearColorMapper(palette = 'Viridis256',
                                 low = min(pp_weather['rain']),
                                 high = max(pp_weather['rain']))

color_bar = ColorBar(color_mapper = color_mapper, location = (0,0))

p = figure()

p.circle(x = 'mint', y = 'maxt', color = {'field': 'rain', 'transform':␣
 ↪color_mapper},
         source=source, size = 'rain_size')

p.title.text = 'Maximum temperature v minimum temperature at Phoenix Park'
p.xaxis.axis_label = 'Minimum temperature (°C)'
p.yaxis.axis_label = 'Maximum temperature (°C)'

hover = HoverTool()
hover.tooltips=[
    ('Date', '@date{%Y-%m-%d}'),
    ('Minimum temperature', '@mint'),
    ('Maximum temperature', '@maxt'),
    ('Rainfall', '@rain')
]
hover.formatters = {'@date': 'datetime'}

p.add_tools(hover)
```

```
p.add_layout(color_bar, 'right')

show(p)
```

11 Categorical data in Bokeh

11.0.1 So far we have looked at line plots and scatterplots for continuous data.

11.0.2 Here we will look at representing categorical data in Bokeh.

11.0.3 Will use the pl_top6 data:

[31]:
```
pl = pd.read_csv("pl_2seasons.csv")

pl.Date = pd.to_datetime(pl.Date, format = '%d/%m/%Y')

pl.head()

pl_top6 = pl.loc[pl.HomeTeam.isin(['Arsenal', 'Chelsea', 'Liverpool', 'Man␣
↪United', 'Man City', 'Tottenham'])]

print(pl_top6)
```

```
       Season        Date    HomeTeam        AwayTeam  FTHG  FTAG FTR  HTHG  \
0    20172018  2017-08-11     Arsenal       Leicester     4     3   H     2
2    20172018  2017-08-12     Chelsea         Burnley     2     3   A     0
8    20172018  2017-08-13  Man United        West Ham     4     0   H     1
13   20172018  2017-08-19   Liverpool  Crystal Palace     1     0   H     0
18   20172018  2017-08-20   Tottenham         Chelsea     1     2   A     0
..        …         …          …               …     …     …   ..    …
747  20182019  2019-05-05     Chelsea         Watford     3     0   H     0
749  20182019  2019-05-06    Man City       Leicester     1     0   H     0
755  20182019  2019-05-12   Liverpool          Wolves     2     0   H     1
756  20182019  2019-05-12  Man United         Cardiff     0     2   A     0
758  20182019  2019-05-12   Tottenham         Everton     2     2   D     1

     HTAG HTR  … HST  AST  HF  AF  HC  AC  HY  AY  HR  AR
0       2   D  …  10    3   9  12   9   4   0   1   0   0
2       3   A  …   6    5  16  11   8   5   3   3   2   0
8       0   H  …   6    1  19   7  11   1   2   2   0   0
13      0   D  …  13    1  12  13   4   2   1   3   0   0
18      1   A  …   6    2  14  21  14   3   3   3   0   0
..    …  ..  … ..  …  ..  ..  ..  ..  ..  ..  ..  ..
747     0   D  …   9    3   6  12   6   6   0   1   0   0
749     0   D  …   5    2  12   5  11   0   3   2   0   0
755     0   H  …   5    2   3  11   4   1   0   2   0   0
756     1   A  …  10    4   9   6  11   2   3   3   0   0
758     0   H  …   3    9  10  13   7   4   0   2   0   0
```

```
[228 rows x 23 columns]
```

11.0.4 Find the sum of some columns for each team eg. the total number of goals scored by each team at home (FTHG), and the total number of goals conceded at home (FTAG).

11.0.5 groupby('HomeTeam') sets the column that we are grouping on.

11.0.6 Then we specify a list of columns we want to aggregate.

11.0.7 We then use sum() to tell Python how to aggregate.

[32]:
```python
top6sums = pl_top6.groupby('HomeTeam')[['FTHG', 'FTAG', 'HF', 'AF']].sum()

print(top6sums)
```

```
            FTHG  FTAG   HF   AF
HomeTeam
Arsenal       96    36  404  451
Chelsea       69    28  337  424
Liverpool    100    20  315  336
Man City     118    26  331  335
Man United    71    34  417  428
Tottenham     74    32  351  379
```

11.0.8 Use the vbar glyph method to create a vertical bar chart (hbar glyph method to create a horizontal bar chart).

11.0.9 First, specify the output_file:

[33]:
```python
output_file("bar_chart1.html")
```

11.0.10 Next, specify the source and initialise the figure.

11.0.11 However, in this case we need to tell Bokeh how to handle the x-axis since it is categorical.

To do that we use the following code: source = ColumnDataSource(top6sums) teams = source.data['HomeTeam'].tolist() p = figure(x_range = teams)

This takes the HomeTeam values from the source data and converts them to a list which is then specified as the x_range of the figure.

[34]:
```python
source = ColumnDataSource(top6sums)
teams = source.data['HomeTeam'].tolist()
p = figure(x_range = teams)
```

11.0.12 The vbar method takes a top parameter rather than a y value:

[35]:
```
p.vbar(x='HomeTeam', top='FTHG', source=source, width=0.7)

p.title.text ='Number of goals scored at home over two seasons by the top 6'
p.xaxis.axis_label = 'Team'
p.yaxis.axis_label = 'Number of goals'

show(p)
```

11.0.13 Add some colouring

11.0.14 Import Spectral6 from bokeh.palettes. It is a built in palette for 6 different colourings.

11.0.15 If we had 5 categories we could import Spectral5

11.0.16 factor_cmap helps to map colors to bars in bar-charts

11.0.17 Initialise the output_file, the source data and the figure:

[36]:
```
from bokeh.palettes import Spectral6

from bokeh.transform import factor_cmap


output_file("bar_chart2.html")

source = ColumnDataSource(top6sums)
teams = source.data['HomeTeam'].tolist()
p = figure(x_range = teams)
```

11.0.18 Specify the color_map using factor_cmap

11.0.19 factor_cmap creates a color map that matches an individual color to each category (what Bokeh calls a factor).

11.0.20 Don't forget to add this as the color argument of vbar for the plot itself!

[37]:
```
color_map = factor_cmap(field_name='HomeTeam',
                        palette = Spectral6, factors = teams)

p.vbar(x='HomeTeam', top='FTHG', source=source, width=0.7, color = color_map)

p.title.text ='Number of goals scored at home over two seasons by the top 6'
p.xaxis.axis_label = 'Team'
p.yaxis.axis_label = 'Number of goals'

show(p)
```

11.0.21 Try a different color palette.

11.0.22 Importing Set3 and including it as the palette gives an error.

11.0.23 For some reason we have to specify the number of categories again: Set3_6

11.0.24 If we had 5 categories we could import Set3_5

[38]:
```
from bokeh.palettes import Set3_6

output_file("bar_chart3.html")

source = ColumnDataSource(top6sums)
teams = source.data['HomeTeam'].tolist()
p = figure(x_range = teams)

color_map = factor_cmap(field_name='HomeTeam', palette = Set3_6, factors =␣
 ↪teams)

p.vbar(x='HomeTeam', top='FTHG', source=source, width=0.7, color = color_map)

p.title.text ='Number of goals scored at home over two seasons by the top 6'
p.xaxis.axis_label = 'Team'
p.yaxis.axis_label = 'Number of goals'

show(p)
```

11.0.25 There are lots of other color palettes. Try Inferno6 for example, or Google it and find your favourite one!

12 Stacked bar charts

12.0.1 Suppose we want to see how many goals were scored and conceded by each team at home.

We can still use the top6sums data because we summed the FTAG column for each team.

[36]:
```
print(top6sums)
```

```
            FTHG  FTAG   HF   AF
HomeTeam
Arsenal       96    36  404  451
Chelsea       69    28  337  424
Liverpool    100    20  315  336
Man City     118    26  331  335
Man United    71    34  417  428
Tottenham     74    32  351  379
```

12.0.2 Initialise the output_file, the source data and the figure:

[39]:
```
output_file("stacked_bar_chart.html")

source = ColumnDataSource(top6sums)
teams = source.data['HomeTeam'].tolist()
p = figure(x_range = teams)
```

12.0.3 We use the vbar_stack glyph method for a stacked bar chart. It takes the arguments stackers (containing the variables to be stacked) and x (the categorical variable on the x axis).

We also give the legend_label.

[40]:
```
p.vbar_stack(stackers = ['FTHG', 'FTAG'], x = 'HomeTeam',
             legend_label = ['Goals scored', 'Goals conceded'],
             source=source, width=0.7)

p.title.text ='Number of goals scored at home over two seasons by the top 6'
p.xaxis.axis_label = 'Team'
p.yaxis.axis_label = 'Number of goals'

show(p)
```

12.0.4 What is the problem there?

[41]:
```
output_file("stacked_bar_chart.html")

source = ColumnDataSource(top6sums)
teams = source.data['HomeTeam'].tolist()
p = figure(x_range = teams)

p.vbar_stack(stackers = ['FTHG', 'FTAG'], x = 'HomeTeam',
             legend_label = ['Goals scored', 'Goals conceded'],
             source=source, width=0.7, color = ['blue', 'red'])

p.title.text ='Number of goals scored at home over two seasons by the top 6'
p.xaxis.axis_label = 'Team'
p.yaxis.axis_label = 'Number of goals'

show(p)
```

13 Gridplot

13.0.1 Next, put two bar charts side by side. We will look at a bar chart of goals scored by each of the top6 teams at home, and another of goals conceded by each of the top6 teams at home

We create two individual figures (left and right), and then put them into one figure using gridplot([[left, right]]).

First, import everything from bokeh.plotting:

[42]:
```
from bokeh.plotting import *
```

13.0.2 Set up the output file, source, and teams list to be used as the x_range on both figures.

[43]:
```
output_file("grid_bar_chart.html")

source = ColumnDataSource(top6sums)
teams = source.data['HomeTeam'].tolist()
```

13.0.3 Create the first figure (called left) which will be a bar chart of goals scored by each of the top6 teams at home:

[44]:
```
left = figure(x_range = teams)

color_map = factor_cmap(field_name='HomeTeam', palette = Set3_6, factors =␣
  ↪teams)

left.vbar(x='HomeTeam', top='FTHG', source=source, width=0.7, color = color_map)

left.title.text ='Number of goals scored at home over two seasons by the top 6'
left.xaxis.axis_label = 'Team'
left.yaxis.axis_label = 'Number of goals'
```

13.0.4 Next, create the second figure (called right) which will be a bar chart of goals conceded by each of the top6 teams at home:

[45]:
```
right = figure(x_range = teams)

color_map = factor_cmap(field_name='HomeTeam', palette = Set3_6, factors =␣
  ↪teams)

right.vbar(x='HomeTeam', top='FTAG', source=source, width=0.7, color =␣
  ↪color_map)

right.title.text ='Number of goals conceded at home over two seasons by the top␣
  ↪6'
```

```
right.xaxis.axis_label = 'Team'
right.yaxis.axis_label = 'Number of goals'
```

13.0.5 Put the two barcharts left and right into one plot called p, and display p:

[46]:
```
p = gridplot([[left, right]])

show(p)

save(p)
```

[46]:
```
'C:\\Users\\DKITStaff\\OneDrive - Dundalk Institute of
Technology\\DKIT\\Programming for Data
Analytics\\Programming_2024_25\\Datasets\\grid_bar_chart.html'
```

14 An example from the Bokeh Quickstart page

14.0.1 This is from the Bokeh Quickstart page. Run the code yourself, have a play around with it and we will discuss what it is doing.

[47]:
```
# prepare some data
N = 300
x = np.linspace(0, 4*np.pi, N)
y0 = np.sin(x)
y1 = np.cos(x)

# output to static HTML file
output_file("linked_brushing.html")

# NEW: create a column data source for the plots to share
source = ColumnDataSource(data=dict(x=x, y0=y0, y1=y1))

TOOLS = "pan,wheel_zoom,box_zoom,reset,save,box_select,lasso_select"

# create a new plot and add a renderer
left = figure(tools=TOOLS, width=350, height=350, title=None)
left.circle('x', 'y0', source=source)

# create another new plot and add a renderer
right = figure(tools=TOOLS, width=350, height=350, title=None)
right.circle('x', 'y1', source=source)

# put the subplots in a gridplot
p = gridplot([[left, right]])

# show the results
show(p)
```

14.1 Links on Bokeh

To revise Bokeh and get another perspective on it, we will follow these links:

https://realpython.com/python-data-visualization-bokeh/

https://www.geeksforgeeks.org/python-bokeh-tutorial-interactive-data-visualization-with-bokeh/

15 Exercise

15.0.1 Create your own interactive Bokeh plot and present it at the end of the lab.

[ ]:

```

```