Analysis of unstructured data

Lecture 9 - data visualization

Janusz Szwabiński

Outlook:

- Bokeh
- yhat/ggplot
- Seaborn
- Vincent
- Plotly
- Folium
- gmplot
- pygal
- NetworkX
- Graphviz

In [1]:

```
%matplotlib inline
import matplotlib.pyplot as plt
```

Bokeh

- http://bokeh.pydata.org/en/latest/ (http://bokeh.pydata.org/en/latest/)
- an interactive visualization library
- targets modern web browsers for presentation
- elegant graphics in the style of D3.js
- designed for very large or streaming datasets
- useful for interactive plots, dashboards, and data applications
- main features:
  - HTML5 Canvas (faster than SVG)
  - processing of data streams
  - real time updates of data
  - Google Maps integration
  - no JavaScript knowledge required

First steps

In [2]:

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

# save result to a static html file
output_file("line.html")

p = figure(plot_width=400, plot_height=400)

# add a circle
p.circle([1, 2, 3, 4, 5], [6, 7, 2, 4, 5], size=20, color="navy", alpha=0.5)

# show result (in a separate browser tab)
show(p)
```

Showing results in jupyter notebook

In [3]:

```
from bokeh.io import output_notebook, reset_output
output_notebook()
```

(https://bokeh.pydata.org) BokehJS 0.12.10 successfully loaded.

In [4]:

```
p = figure(plot_width=400, plot_height=400)
p.circle([1, 2, 3, 4, 5], [6, 7, 2, 4, 5], size=20, color="navy", alpha=0.5)
show(p)
```

In [5]:

```
reset_output() # switch off notebook output
output_file("plot.html")
```

Changing markers

In [6]:

```
p = figure(plot_width=400, plot_height=400)

# squares instead of circles
p.square([1, 2, 3, 4, 5], [6, 7, 2, 4, 5], size=[10, 15, 20, 25, 30], color="fir
ebrick", alpha=0.6)

show(p)
```

Linear plots

In [7]:

```
# new plot
p = figure(plot_width=600, plot_height=400, title="My first linear plot")

# line renderer
p.line([1, 2, 3, 4, 5], [6, 7, 2, 4, 5], line_width=2)

show(p)
```

Connecting symbols

In [8]:

```
x = [1, 2, 3, 4, 5]
y = [6, 7, 8, 7, 3]

p = figure(plot_width=400, plot_height=400)
p.line(x, y, line_width=2)
p.circle(x, y, fill_color="red", size=8)

show(p)
```

Legend

In [9]:

```
import numpy as np

x = np.linspace(0, 4*np.pi, 100)
y = np.sin(x)

p = figure()

p.circle(x, y, legend="sin(x)")
p.line(x, y, legend="sin(x)")

p.line(x, 2*y, legend="2*sin(x)", line_dash=[4, 4], line_color="orange", line_wi
dth=2)

p.line(x, 3*y, legend="3*sin(x)", line_color="green")
p.square(x, 3*y, legend="3*sin(x)", fill_color="white", line_color="green")

p.legend.location = "bottom_left"

show(p)
```

Multiple plots

In [10]:

```
from bokeh.layouts import gridplot

x = list(range(11))
y0, y1, y2 = x, [10-i for i in x], [abs(i-5) for i in x]

# new figure
s1 = figure(width=250, plot_height=250)
s1.circle(x, y0, size=10, color="navy", alpha=0.5)

# another new figure
s2 = figure(width=250, height=250)
s2.triangle(x, y1, size=10, color="firebrick", alpha=0.5)

# and another one
s3 = figure(width=250, height=250)
s3.square(x, y2, size=10, color="olive", alpha=0.5)

# put them next to each other
p = gridplot([[s1, s2, s3]], toolbar_location=None)

show(p)
```

Linked panning

In [11]:

```
plot_options = dict(width=250, plot_height=250, title=None, tools='pan')

s1 = figure(**plot_options)
s1.circle(x, y0, size=10, color="navy")

# synchronize x axes
s2 = figure(x_range=s1.x_range, y_range=s1.y_range, **plot_options)
s2.triangle(x, y1, size=10, color="firebrick")

# synchronize y axes
s3 = figure(x_range=s1.x_range, **plot_options)
s3.square(x, y2, size=10, color="olive")

p = gridplot([[s1, s2, s3]])

show(p)
```

Linked selections

In [12]:

```
from bokeh.models import ColumnDataSource

x = list(range(-20, 21))
y0, y1 = [abs(xx) for xx in x], [xx**2 for xx in x]

# build a data source for plots
source = ColumnDataSource(data=dict(x=x, y0=y0, y1=y1))

# selection tools available
TOOLS = "box_select,lasso_select,help"

# new plot
left = figure(tools=TOOLS, width=300, height=300)
left.circle('x', 'y0', source=source)

# another one
right = figure(tools=TOOLS, width=300, height=300)
right.circle('x', 'y1', source=source)

p = gridplot([[left, right]])

show(p)
```

Hovering the mouse over the plot

In [13]:

```
from bokeh.models import HoverTool

source = ColumnDataSource(
        data=dict(
            x=[1, 2, 3, 4, 5],
            y=[2, 5, 8, 2, 7],
            desc=['A', 'b', 'C', 'd', 'E'],
        )
    )

hover = HoverTool(
        tooltips=[
            ("index", "$index"),
            ("(x,y)", "($x, $y)"),
            ("desc", "@desc"),
        ]
    )

p = figure(plot_width=600, plot_height=600, tools=[hover], title="Mouse over the
 dots")

p.circle('x', 'y', size=40, source=source)

show(p)
```

Callback functions

In [14]:

```
from bokeh.models import TapTool, CustomJS, ColumnDataSource

callback = CustomJS(code="alert('hello world')")
tap = TapTool(callback=callback)

p = figure(plot_width=600, plot_height=300, tools=[tap])

p.circle('x', 'y', size=20, source=ColumnDataSource(data=dict(x=[1, 2, 3, 4, 5],
 y=[2, 5, 8, 2, 7])))

show(p)
```

Widgets

In [15]:

```
from bokeh.layouts import layout
from bokeh.models import CustomJS, ColumnDataSource, Slider

x = [x*0.005 for x in range(0, 200)]
y = x

source = ColumnDataSource(data=dict(x=x, y=y))

plot = figure(plot_width=400, plot_height=400)
plot.line('x', 'y', source=source, line_width=3, line_alpha=0.6)

callback = CustomJS(args=dict(source=source), code="""
    var data = source.get('data');
    var f = cb_obj.get('value')
    x = data['x']
    y = data['y']
    for (i = 0; i < x.length; i++) {
        y[i] = Math.pow(x[i], f)
    }
    source.trigger('change');
""")

slider = Slider(start=0.1, end=4, value=1, step=.1, title="Power", callback=call
back)

show(layout([[slider],[plot]]))
```

Example - more than 100k points

In [2]:

```
from bokeh.sampledata.world_cities import data

data.head()
```

Out[2]:

	name	lat	lng
0	Ordino	42.55623	1.53319
1	les Escaldes	42.50729	1.53414
2	la Massana	42.54499	1.51483
3	Encamp	42.53474	1.58014
4	Canillo	42.56760	1.59756

In [3]:

```
data.tail()
```

Out[3]:

	name	lat	lng
45060	Bindura	-17.30192	31.33056
45061	Beitbridge	-22.21667	30.00000
45062	Banket	-17.38333	30.40000
45063	Epworth	-17.89000	31.14750
45064	Chitungwiza	-18.01274	31.07555

In [6]:

```
#Google Maps API requires an API key.
#See https://developers.google.com/maps/documentation/javascript/get-api-key for
 more information

api_key = 'your_key'


from bokeh.models import (
    GMapOptions, GMapPlot, ColumnDataSource, PanTool, WheelZoomTool, Circle, Ran
ge1d
)

p = GMapPlot(
    x_range=Range1d(-160, 160), y_range=Range1d(-80, 80),
    plot_width=1000,plot_height=500,
    api_key=api_key,
    map_options=GMapOptions(lat=48.77, lng=9.18, zoom=4))

circle = Circle(x="lng", y="lat", size=5, line_color=None, fill_color='firebric
k', fill_alpha=0.3)
p.add_glyph(ColumnDataSource(data), circle)
p.add_tools(PanTool(), WheelZoomTool())
p.title.text = "Cities with more than 5k people"
output_file("cities.html", title="Cities example")
show(p)
```

Example - Usain Bolt and the rest of the world

In [7]:

```
from bokeh.sampledata.sprint import sprint
sprint[:10]
```

Out[7]:

	Name	Country	Medal	Time	Year
0	Usain Bolt	JAM	GOLD	9.63	2012
1	Yohan Blake	JAM	SILVER	9.75	2012
2	Justin Gatlin	USA	BRONZE	9.79	2012
3	Usain Bolt	JAM	GOLD	9.69	2008
4	Richard Thompson	TRI	SILVER	9.89	2008
5	Walter Dix	USA	BRONZE	9.91	2008
6	Justin Gatlin	USA	GOLD	9.85	2004
7	Francis Obikwelu	POR	SILVER	9.86	2004
8	Maurice Greene	USA	BRONZE	9.87	2004
9	Maurice Greene	USA	GOLD	9.87	2000

First, we import all models required for drawing the plot:

In [8]:

```
from bokeh.io import output_notebook, show
from bokeh.models.glyphs import Circle, Text
from bokeh.models import ColumnDataSource, Range1d, DataRange1d, Plot
```

Now, we prepare some auxiliary data:

In [9]:

```
abbrev_to_country = {
    "USA": "United States",
    "GBR": "Britain",
    "JAM": "Jamaica",
    "CAN": "Canada",
    "TRI": "Trinidad and Tobago",
    "AUS": "Australia",
    "GER": "Germany",
    "CUB": "Cuba",
    "NAM": "Namibia",
    "URS": "Soviet Union",
    "BAR": "Barbados",
    "BUL": "Bulgaria",
    "HUN": "Hungary",
    "NED": "Netherlands",
    "NZL": "New Zealand",
    "PAN": "Panama",
    "POR": "Portugal",
    "RSA": "South Africa",
    "EUA": "United Team of Germany",
}

gold_fill   = "#efcf6d"
gold_line   = "#c8a850"
silver_fill = "#cccccc"
silver_line = "#b0b0b1"
bronze_fill = "#c59e8a"
bronze_line = "#98715d"

fill_color = { "gold": gold_fill, "silver": silver_fill, "bronze": bronze_fill }
line_color = { "gold": gold_line, "silver": silver_line, "bronze": bronze_line }

def selected_name(name, medal, year):
    return name if medal == "gold" and year in [1988, 1968, 1936, 1896] else ''

t0 = sprint.Time[0]

sprint["Abbrev"]       = sprint.Country
sprint["Country"]      = sprint.Abbrev.map(lambda abbr: abbrev_to_country[abbr])
sprint["Medal"]        = sprint.Medal.map(lambda medal: medal.lower())
sprint["Speed"]        = 100.0/sprint.Time
sprint["MetersBack"]   = 100.0*(1.0 - t0/sprint.Time)
sprint["MedalFill"]    = sprint.Medal.map(lambda medal: fill_color[medal])
sprint["MedalLine"]    = sprint.Medal.map(lambda medal: line_color[medal])
sprint["SelectedName"] = sprint[["Name", "Medal", "Year"]].apply(tuple,
axis=1).map(lambda args: selected_name(*args))
source = ColumnDataSource(sprint)
```

In [7]:

```
sprint
```

Out[7]:

	Name	Country	Medal	Time	Year	Abbrev	Speed	MetersBack	MedalF
0	Usain Bolt	Jamaica	gold	9.63	2012	JAM	10.384216	0.000000	#efcf6d
1	Yohan Blake	Jamaica	silver	9.75	2012	JAM	10.256410	1.230769	#cccccc
2	Justin Gatlin	United States	bronze	9.79	2012	USA	10.214505	1.634321	#c59e8
3	Usain Bolt	Jamaica	gold	9.69	2008	JAM	10.319917	0.619195	#efcf6d
4	Richard Thompson	Trinidad and Tobago	silver	9.89	2008	TRI	10.111223	2.628918	#cccccc
5	Walter Dix	United States	bronze	9.91	2008	USA	10.090817	2.825429	#c59e8
6	Justin Gatlin	United States	gold	9.85	2004	USA	10.152284	2.233503	#efcf6d
7	Francis Obikwelu	Portugal	silver	9.86	2004	POR	10.141988	2.332657	#cccccc
8	Maurice Greene	United States	bronze	9.87	2004	USA	10.131712	2.431611	#c59e8
9	Maurice Greene	United States	gold	9.87	2000	USA	10.131712	2.431611	#efcf6d
10	Ato Boldon	Trinidad and Tobago	silver	9.99	2000	TRI	10.010010	3.603604	#cccccc
11	Obadele Thompson	Barbados	bronze	10.04	2000	BAR	9.960159	4.083665	#c59e8
12	Donovan Bailey	Canada	gold	9.84	1996	CAN	10.162602	2.134146	#efcf6d
13	Frankie Fredericks	Namibia	silver	9.89	1996	NAM	10.111223	2.628918	#cccccc
14	Ato Boldon	Trinidad and Tobago	bronze	9.90	1996	TRI	10.101010	2.727273	#c59e8
15	Linford Christie	Britain	gold	9.96	1992	GBR	10.040161	3.313253	#efcf6d
16	Frankie Fredericks	Namibia	silver	10.02	1992	NAM	9.980040	3.892216	#cccccc
17	Dennis Mitchell	United States	bronze	10.04	1992	USA	9.960159	4.083665	#c59e8
18	Carl Lewis	United States	gold	9.92	1988	USA	10.080645	2.923387	#efcf6d

	Name	Country	Medal	Time	Year	Abbrev	Speed	MetersBack	MedalF
19	Linford Christie	Britain	silver	9.97	1988	GBR	10.030090	3.410231	#cccccc
20	Calvin Smith	United States	bronze	9.99	1988	USA	10.010010	3.603604	#c59e8
21	Carl Lewis	United States	gold	9.99	1984	USA	10.010010	3.603604	#efcf6d
22	Sam Graddy	United States	silver	10.19	1984	USA	9.813543	5.495584	#cccccc
23	Ben Johnson	Canada	bronze	10.22	1984	CAN	9.784736	5.772994	#c59e8
24	Allan Wells	Britain	gold	10.25	1980	GBR	9.756098	6.048780	#efcf6d
25	Silvio Leonard Tartabull	Cuba	silver	10.25	1980	CUB	9.756098	6.048780	#cccccc
26	Petar Petrov	Bulgaria	bronze	10.39	1980	BUL	9.624639	7.314726	#c59e8
27	Hasely Crawford	Trinidad and Tobago	gold	10.06	1976	TRI	9.940358	4.274354	#efcf6d
28	Donald Quarrie	Jamaica	silver	10.08	1976	JAM	9.920635	4.464286	#cccccc
29	Valery Borzov	Soviet Union	bronze	10.14	1976	URS	9.861933	5.029586	#c59e8
...	...	...	...	...	...	...	...	...	...
55	Ralph Metcalfe	United States	silver	10.30	1932	USA	9.708738	6.504854	#cccccc
56	Arthur Jonath	Germany	bronze	10.40	1932	GER	9.615385	7.403846	#c59e8
57	Percy Williams	Canada	gold	10.80	1928	CAN	9.259259	10.833333	#efcf6d
58	John "Jack" London	Britain	silver	10.90	1928	GBR	9.174312	11.651376	#cccccc
59	Georg Lammers	Germany	bronze	10.90	1928	GER	9.174312	11.651376	#c59e8
60	Harold Abrahams	Britain	gold	10.60	1924	GBR	9.433962	9.150943	#efcf6d
61	Jackson Scholz	United States	silver	10.80	1924	USA	9.259259	10.833333	#cccccc

	Name	Country	Medal	Time	Year	Abbrev	Speed	MetersBack	MedalF
62	Arthur Porritt	New Zealand	bronze	10.90	1924	NZL	9.174312	11.651376	#c59e8
63	Charles Paddock	United States	gold	10.80	1920	USA	9.259259	10.833333	#efcf6d
64	Morris Kirksey	United States	silver	10.90	1920	USA	9.174312	11.651376	#ccccc
65	Harry Edward	Britain	bronze	10.90	1920	GBR	9.174312	11.651376	#c59e8
66	Ralph Craig	United States	gold	10.80	1912	USA	9.259259	10.833333	#efcf6d
67	Alvah Meyer	United States	silver	10.90	1912	USA	9.174312	11.651376	#ccccc
68	Donald Lippincott	United States	bronze	10.90	1912	USA	9.174312	11.651376	#c59e8
69	Reginald Walker	South Africa	gold	10.80	1908	RSA	9.259259	10.833333	#efcf6d
70	James Rector	United States	silver	11.00	1908	USA	9.090909	12.454545	#ccccc
71	Robert Kerr	Canada	bronze	11.00	1908	CAN	9.090909	12.454545	#c59e8
72	Charles "Archie" Hahn	United States	gold	11.20	1906	USA	8.928571	14.017857	#efcf6d
73	Fay Moulton	United States	silver	11.30	1906	USA	8.849558	14.778761	#ccccc
74	Nigel Barker	Australia	bronze	11.30	1906	AUS	8.849558	14.778761	#c59e8
75	Charles "Archie" Hahn	United States	gold	11.00	1904	USA	9.090909	12.454545	#efcf6d
76	Nathaniel Cartmell	United States	silver	11.20	1904	USA	8.928571	14.017857	#ccccc
77	Bill Hogenson	United States	bronze	11.20	1904	USA	8.928571	14.017857	#c59e8
78	Frank Jarvis	United States	gold	11.00	1900	USA	9.090909	12.454545	#efcf6d
79	J. Walter Tewksbury	United States	silver	11.10	1900	USA	9.009009	13.243243	#ccccc
80	Stanley Rowley	Australia	bronze	11.20	1900	AUS	8.928571	14.017857	#c59e8

	Name	Country	Medal	Time	Year	Abbrev	Speed	MetersBack	MedalF
81	Thomas Burke	United States	gold	12.00	1896	USA	8.333333	19.750000	#efcf6d
82	Fritz Hofmann	Germany	silver	12.20	1896	GER	8.196721	21.065574	#cccccc
83	Alojz Sokol	Hungary	bronze	12.60	1896	HUN	7.936508	23.571429	#c59e8
84	Francis Lane	United States	bronze	12.60	1896	USA	7.936508	23.571429	#c59e8

85 rows × 11 columns

The basic plot is simple:

In [10]:

```
plot_options = dict(plot_width=800, plot_height=480, toolbar_location=None,
                    outline_line_color=None)

radius = dict(value=5, units="screen")
medal_glyph = Circle(x="MetersBack", y="Year", radius=radius, fill_color="MedalF
ill",
                     line_color="MedalLine", fill_alpha=0.5)

athlete_glyph = Text(x="MetersBack", y="Year", x_offset=10, text="SelectedName",
    text_align="left", text_baseline="middle", text_font_size="9pt")

no_olympics_glyph = Text(x=7.5, y=1942, text=["No Olympics in 1940 or 1944"],
    text_align="center", text_baseline="middle",
    text_font_size="9pt", text_font_style="italic", text_color="silver")
```

In [11]:

```
xdr = Range1d(start=sprint.MetersBack.max()+2, end=0)  # +2 is for padding
ydr = DataRange1d(range_padding=0.05)

plot = Plot(x_range=xdr, y_range=ydr, **plot_options)
plot.add_glyph(source, medal_glyph)
plot.add_glyph(source, athlete_glyph)
plot.add_glyph(no_olympics_glyph)
```

Out[11]:

GlyphRenderer(id = '9e150a00-05cc-4460-8be8-9637c10d54d7', …)

In [12]:

```
show(plot)
```

We add axes and the grid:

In [13]:

```
from bokeh.models import Grid, LinearAxis, SingleIntervalTicker

xdr = Range1d(start=sprint.MetersBack.max()+2, end=0)  # +2 is for padding
ydr = DataRange1d(range_padding=0.05)

plot = Plot(x_range=xdr, y_range=ydr, **plot_options)
plot.add_glyph(source, medal_glyph)
plot.add_glyph(source, athlete_glyph)
plot.add_glyph(no_olympics_glyph)

xticker = SingleIntervalTicker(interval=5, num_minor_ticks=0)
xaxis = LinearAxis(ticker=xticker, axis_line_color=None, major_tick_line_color=N
one,
                   axis_label="Meters behind 2012 Bolt", axis_label_text_font_si
ze="10pt",
                   axis_label_text_font_style="bold")
plot.add_layout(xaxis, "below")

xgrid = Grid(dimension=0, ticker=xaxis.ticker, grid_line_dash="dashed")
plot.add_layout(xgrid)

yticker = SingleIntervalTicker(interval=12, num_minor_ticks=0)
yaxis = LinearAxis(ticker=yticker, major_tick_in=-5, major_tick_out=10)
plot.add_layout(yaxis, "right")

show(plot)
```

Now, we add some additional information about each sprinter:

In [14]:

```
from bokeh.models import HoverTool

tooltips = """
<div>
    <span style="font-size: 15px;">@Name</span> 
    <span style="font-size: 10px; color: #666;">(@Abbrev)</span>
</div>
<div>
    <span style="font-size: 17px; font-weight: bold;">@Time{0.00}</span> 
    <span style="font-size: 10px; color: #666;">@Year</span>
</div>
<div style="font-size: 11px; color: #666;">@{MetersBack}{0.00} meters behind</di
v>
"""

xdr = Range1d(start=sprint.MetersBack.max()+2, end=0)  # +2 is for padding
ydr = DataRange1d(range_padding=0.05)

plot = Plot(x_range=xdr, y_range=ydr, **plot_options)
medal = plot.add_glyph(source, medal_glyph)  # we need this renderer to configur
e the hover tool
plot.add_glyph(source, athlete_glyph)
plot.add_glyph(no_olympics_glyph)

xticker = SingleIntervalTicker(interval=5, num_minor_ticks=0)
xaxis = LinearAxis(ticker=xticker, axis_line_color=None, major_tick_line_color=N
one,
                   axis_label="Meters behind 2012 Bolt", axis_label_text_font_si
ze="10pt",
                   axis_label_text_font_style="bold")
plot.add_layout(xaxis, "below")

xgrid = Grid(dimension=0, ticker=xaxis.ticker, grid_line_dash="dashed")
plot.add_layout(xgrid)

yticker = SingleIntervalTicker(interval=12, num_minor_ticks=0)
yaxis = LinearAxis(ticker=yticker, major_tick_in=-5, major_tick_out=10)
plot.add_layout(yaxis, "right")

hover = HoverTool(tooltips=tooltips, renderers=[medal])
plot.add_tools(hover)

show(plot)
```

yhat/ggplot

- `ggplot` (https://github.com/yhat/ggplot (https://github.com/yhat/ggplot)) - a Python implementation of the The Grammar of Graphics (http://www.amazon.com/Grammar-Graphics-Statistics-Computing/dp/0387245448)
- a port of ggplot2 for R (but not a feature-for-feature one)
- from Python programmer's perspective the interface may seem weird (but to be honest, R is a little weird too :))

Our first plot

In [7]:

```
from ggplot import * #for the sake of convenience we import everything

mtcars
```

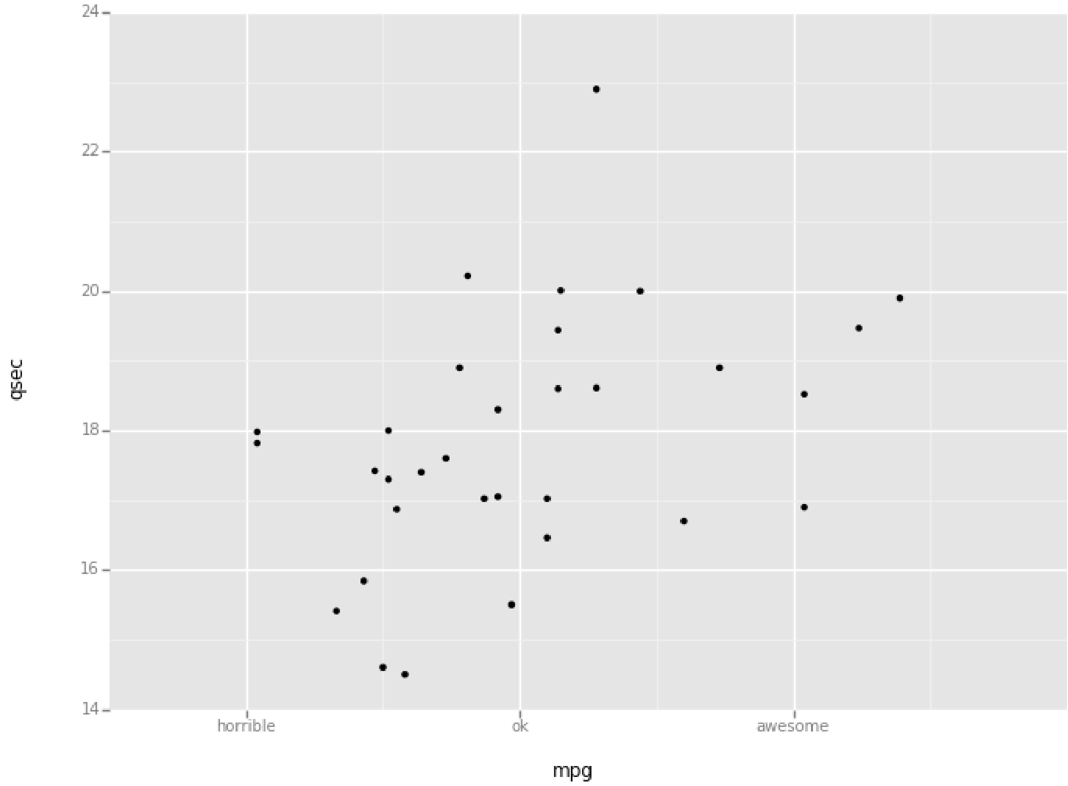
```
/usr/local/lib/python3.5/dist-packages/ggplot/utils.py:81: FutureWar
ning: pandas.tslib is deprecated and will be removed in a future ver
sion.
You can access Timestamp as pandas.Timestamp
  pd.tslib.Timestamp,
/usr/local/lib/python3.5/dist-packages/ggplot/stats/smoothers.py:4:
 FutureWarning: The pandas.lib module is deprecated and will be remo
ved in a future version. These are private functions and can be acce
ssed from pandas._libs.lib instead
  from pandas.lib import Timestamp
/usr/local/lib/python3.5/dist-packages/statsmodels/compat/pandas.py:
56: FutureWarning: The pandas.core.datetools module is deprecated an
d will be removed in a future version. Please use the pandas.tseries
module instead.
  from pandas.core import datetools
```

Out[7]:

	name	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
1	Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
2	Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
3	Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
4	Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
5	Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
6	Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
7	Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
8	Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
9	Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4
10	Merc 280C	17.8	6	167.6	123	3.92	3.440	18.90	1	0	4	4
11	Merc 450SE	16.4	8	275.8	180	3.07	4.070	17.40	0	0	3	3
12	Merc 450SL	17.3	8	275.8	180	3.07	3.730	17.60	0	0	3	3
13	Merc 450SLC	15.2	8	275.8	180	3.07	3.780	18.00	0	0	3	3
14	Cadillac Fleetwood	10.4	8	472.0	205	2.93	5.250	17.98	0	0	3	4
15	Lincoln Continental	10.4	8	460.0	215	3.00	5.424	17.82	0	0	3	4
16	Chrysler Imperial	14.7	8	440.0	230	3.23	5.345	17.42	0	0	3	4
17	Fiat 128	32.4	4	78.7	66	4.08	2.200	19.47	1	1	4	1
18	Honda Civic	30.4	4	75.7	52	4.93	1.615	18.52	1	1	4	2
19	Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.90	1	1	4	1
20	Toyota Corona	21.5	4	120.1	97	3.70	2.465	20.01	1	0	3	1
21	Dodge Challenger	15.5	8	318.0	150	2.76	3.520	16.87	0	0	3	2
22	AMC Javelin	15.2	8	304.0	150	3.15	3.435	17.30	0	0	3	2
23	Camaro Z28	13.3	8	350.0	245	3.73	3.840	15.41	0	0	3	4
24	Pontiac Firebird	19.2	8	400.0	175	3.08	3.845	17.05	0	0	3	2
25	Fiat X1-9	27.3	4	79.0	66	4.08	1.935	18.90	1	1	4	1
26	Porsche 914-2	26.0	4	120.3	91	4.43	2.140	16.70	0	1	5	2
27	Lotus Europa	30.4	4	95.1	113	3.77	1.513	16.90	1	1	5	2
28	Ford Pantera L	15.8	8	351.0	264	4.22	3.170	14.50	0	1	5	4
29	Ferrari Dino	19.7	6	145.0	175	3.62	2.770	15.50	0	1	5	6
30	Maserati Bora	15.0	8	301.0	335	3.54	3.570	14.60	0	1	5	8
31	Volvo 142E	21.4	4	121.0	109	4.11	2.780	18.60	1	1	4	2

In [8]:

```
# aes (i.e. aesthetics) describe how data will relate to the plot
print(ggplot(mtcars, aes('mpg', 'qsec')) + \
  geom_point(colour='steelblue') + \
  scale_x_continuous(breaks=[10,20,30],  \
                     labels=["horrible", "ok", "awesome"]))
```



<ggplot: (8762164053161)>

Multiple datasets on one plot

In [9]:

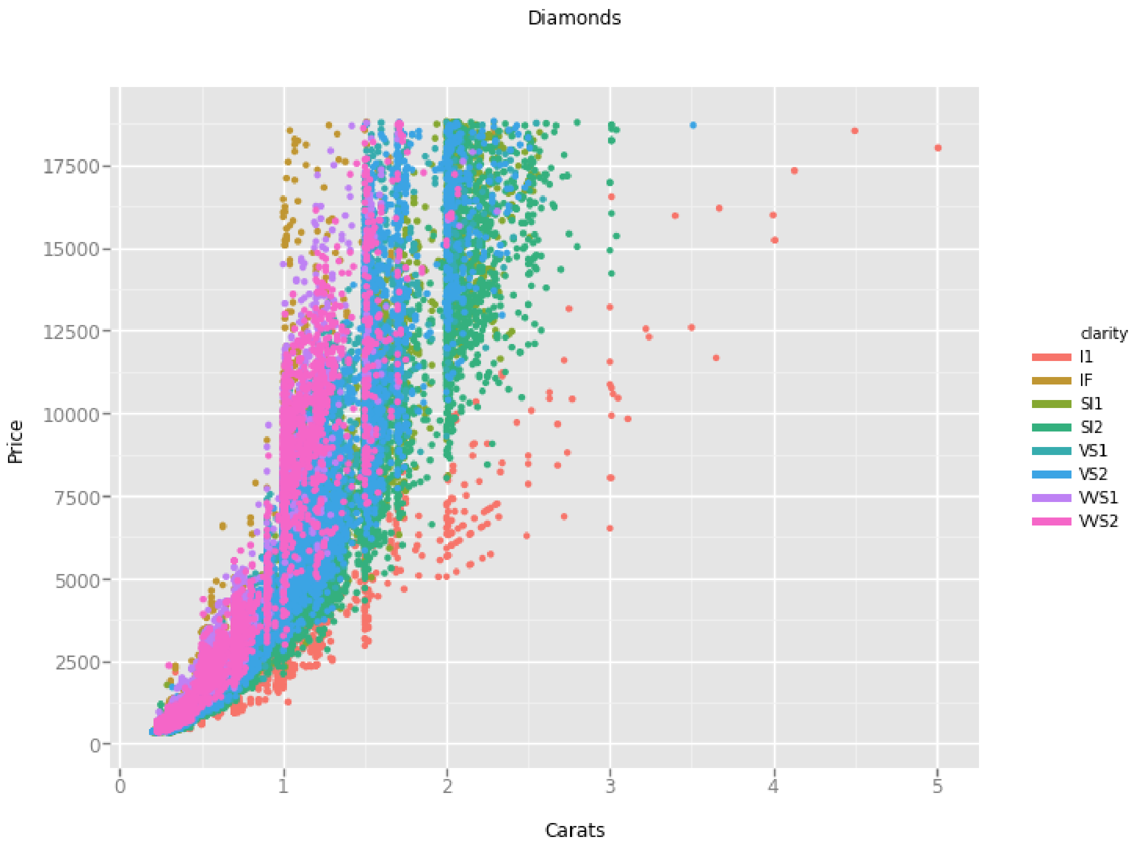
```
diamonds.head()
```

Out[9]:

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75

In [4]:

```
ggplot(diamonds, aes(x='carat', y='price', color='clarity')) + geom_point() +\
    xlab("Carats") + ylab("Price") + ggtitle("Diamonds")
```



Out[4]:

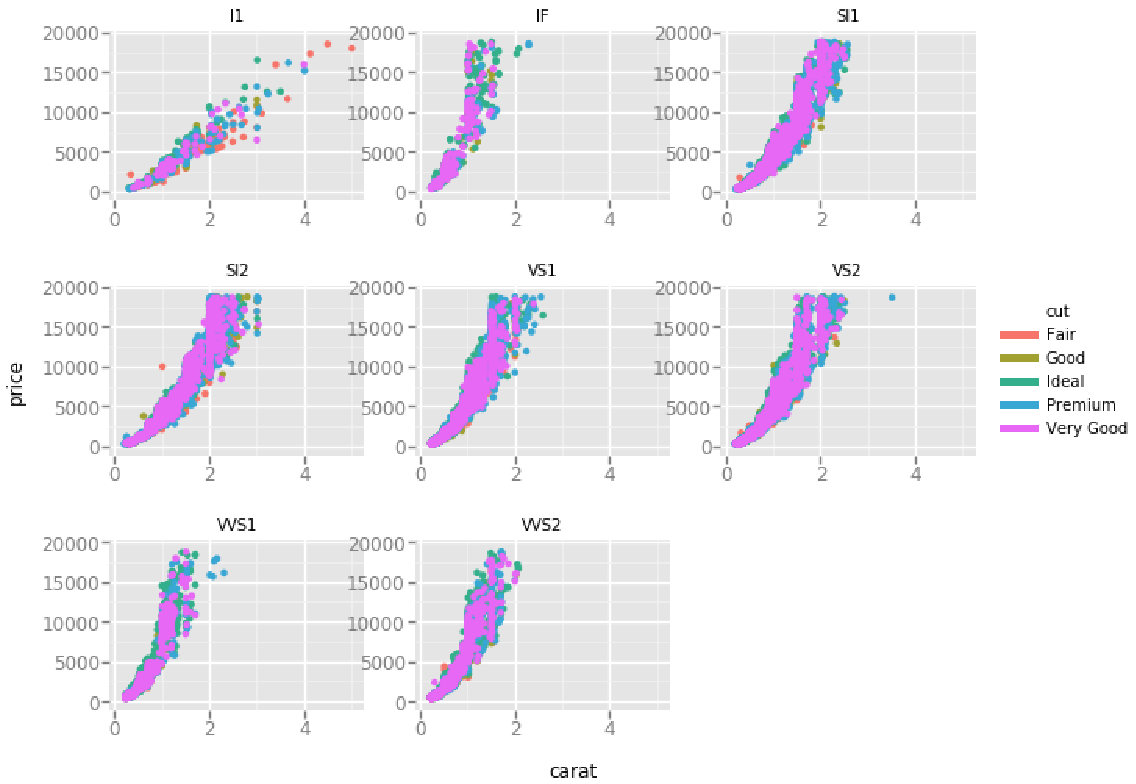
<ggplot: (-9223363297745261133)>

Data facets

We can split the above plot into a matrix of plots corresponding to different categories of data:

In [5]:

```
ggplot(diamonds, aes(x='carat', y='price', color='cut')) + geom_point() + facet_
wrap('clarity')
```



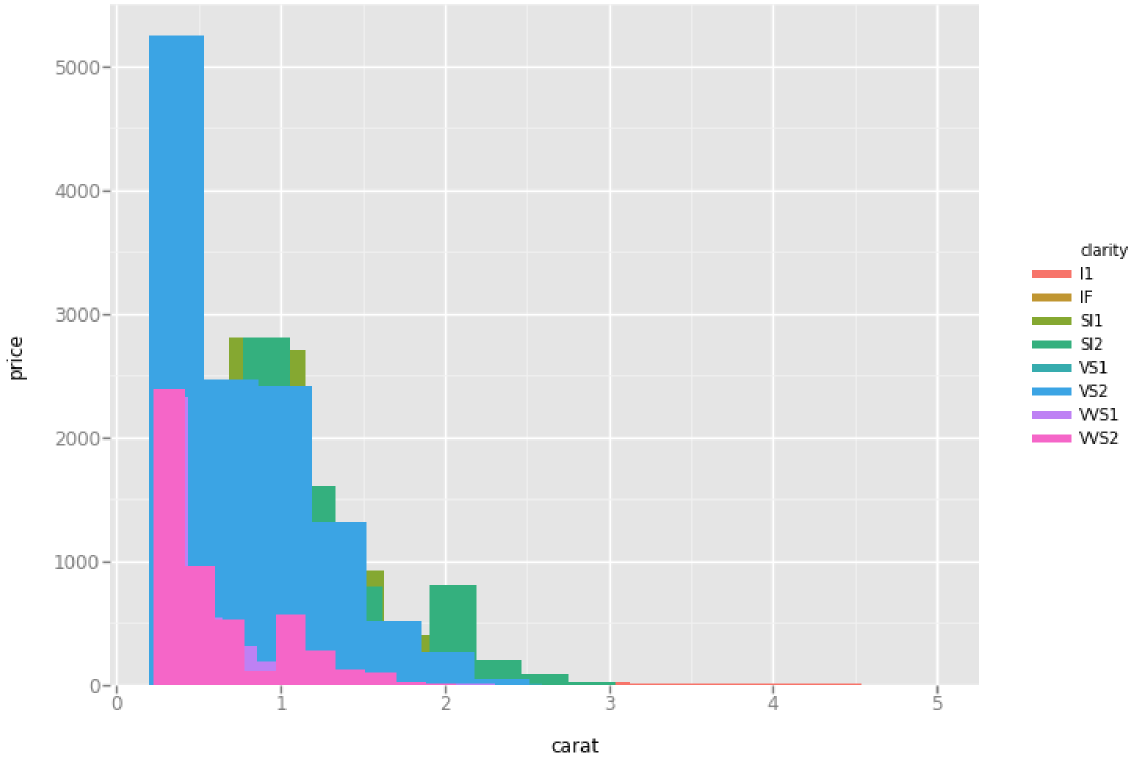
Out[5]:

<ggplot: (8739109453695)>

Histograms

In [6]:

```
ggplot(diamonds, aes(x='carat', y='price', fill='clarity')) + geom_histogram()
```



Out[6]:

```
<ggplot: (-9223363297747978373)>
```

Seaborn

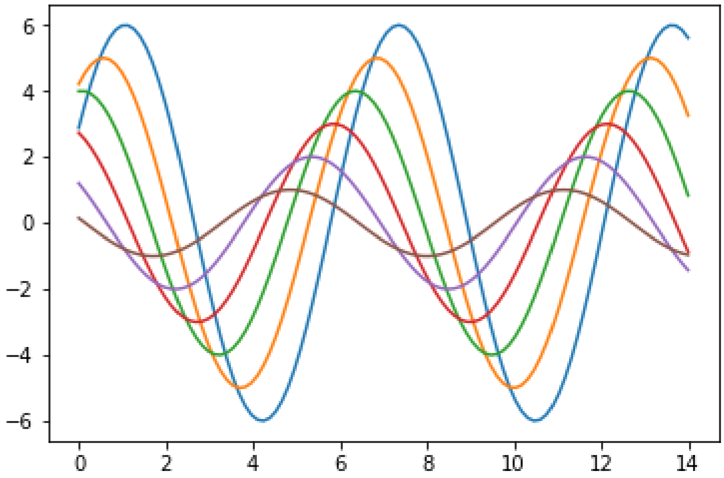
- https://seaborn.pydata.org/ (https://seaborn.pydata.org/)
- a Python visualization library based on matplotlib
- a high-level interface for drawing attractive statistical graphics

Controlling figure aesthetics

Consider first the following matplotlib example:

In [16]:

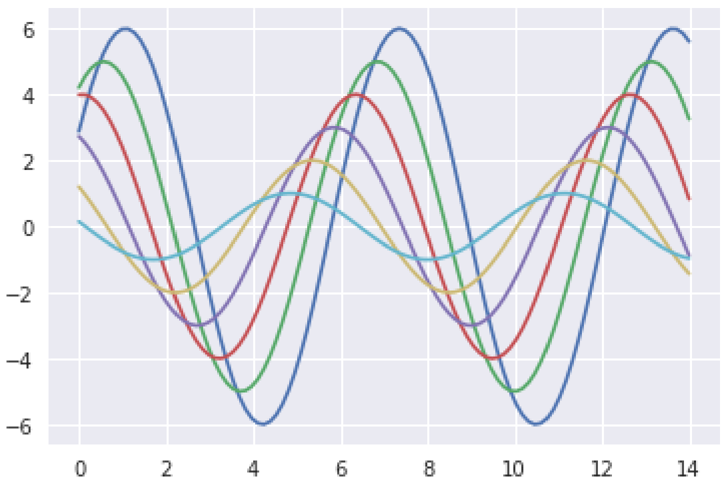
```
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
np.random.seed(sum(map(ord, "aesthetics")))

def sinplot(flip=1):
    x = np.linspace(0, 14, 100)
    for i in range(1, 7):
        plt.plot(x, np.sin(x + i * .5) * (7 - i) * flip)

sinplot()
```



To switch to seaborn defaults, simply call the `set()` function.

In [17]:

```
import seaborn as sns
sns.set()
sinplot()
```

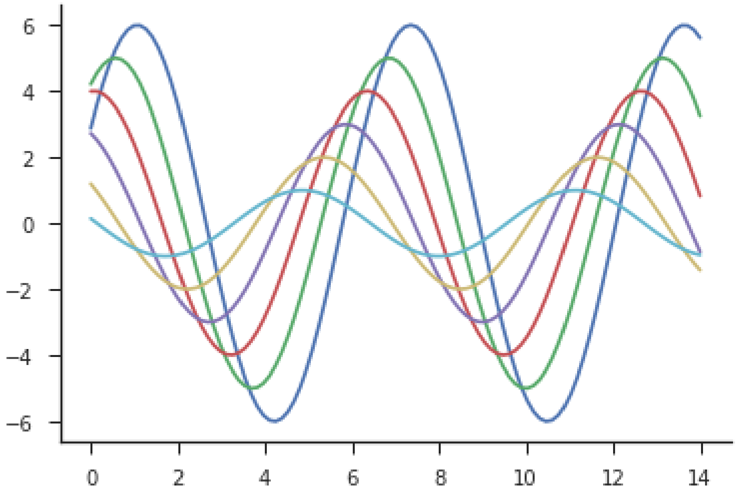


Seaborn splits matplotlib parameters into two independent groups:

- first group sets the aesthetic style of the plot
- the second one scales various elements of the figure so that it can be easily incorporated into different contexts

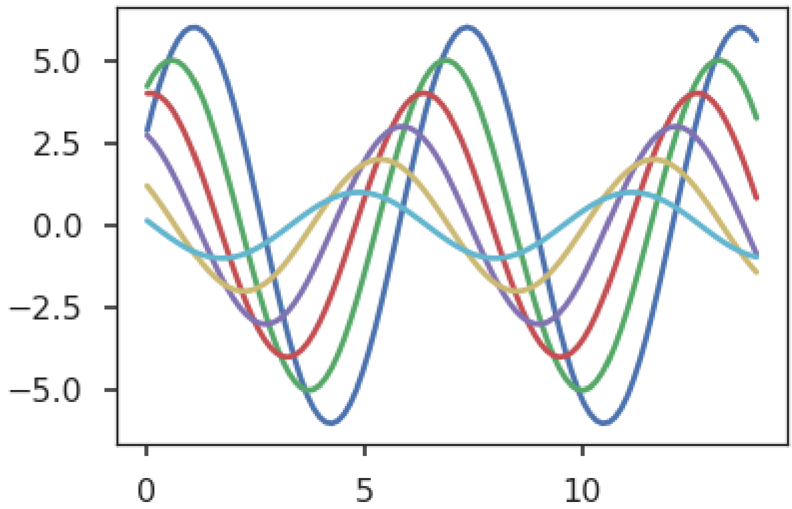
In [12]:

```
sns.set_style("ticks")
sinplot()
sns.despine()
```



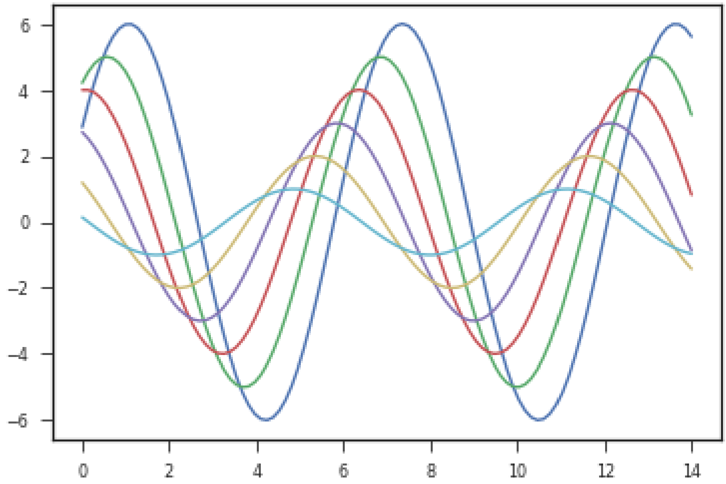
In [13]:

```
sns.set_context("poster")
sinplot()
```



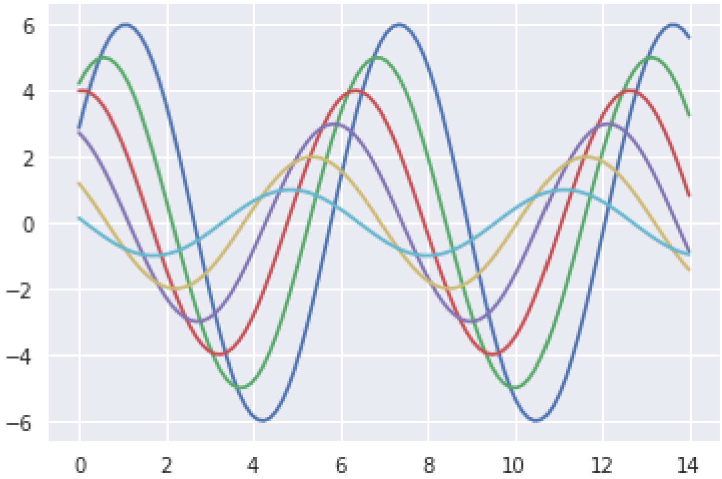
In [14]:

```
sns.set_context("paper")
sinplot()
```



In [15]:

```
sns.set()
sinplot()
```

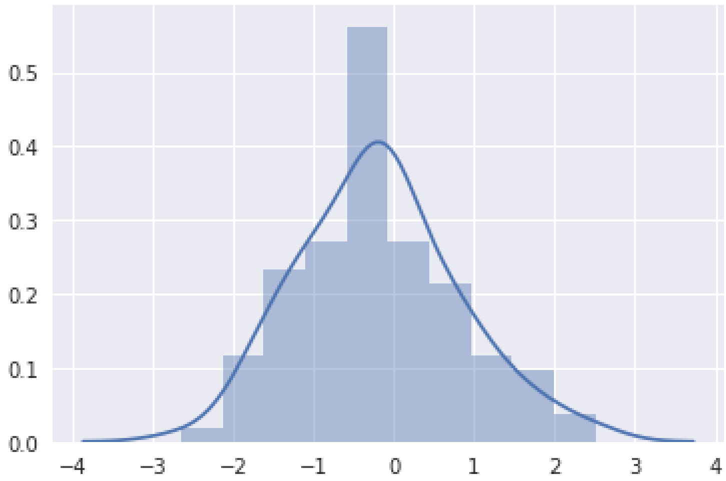


Useful types of plots

Distribution plots

In [16]:

```
sns.set(color_codes=True)
x = np.random.normal(size=100)
sns.distplot(x);
```



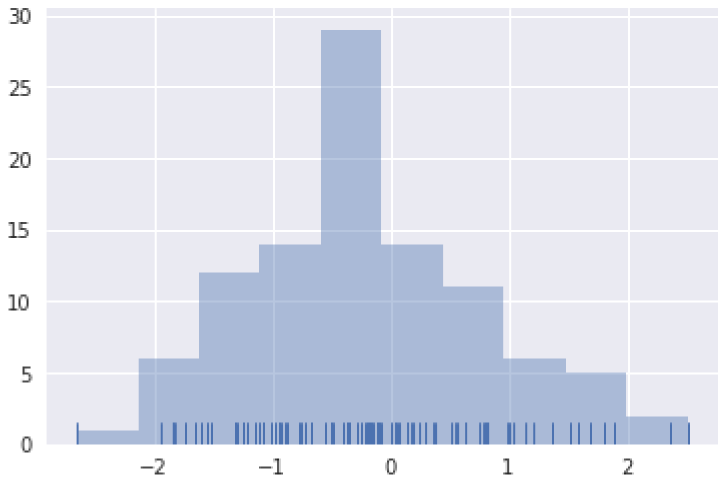
Warning In case you get the error `TypeError: slice indices must be integers or None or have an __index__ method` at this point, you have to update `statsmodels` module to version 0.8.0:

```
pip3 install -U statsmodels==0.8.0
```

Histograms

In [17]:

```
sns.distplot(x, kde=False, rug=True); #kde - kernel density estimate,
                                       #rug=True draws a small vertical tick at e
ach observation
```

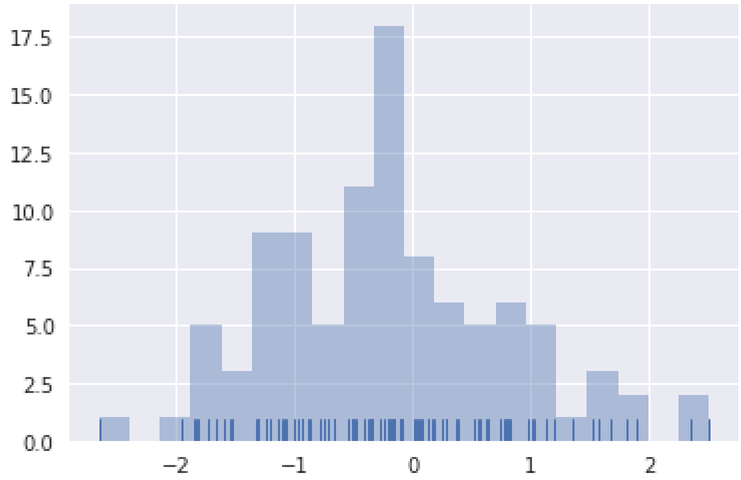


In [18]:

```
sns.distplot(x, bins=20, kde=False, rug=True)
```

Out[18]:

<matplotlib.axes._subplots.AxesSubplot at 0x7f2baf45ea90>

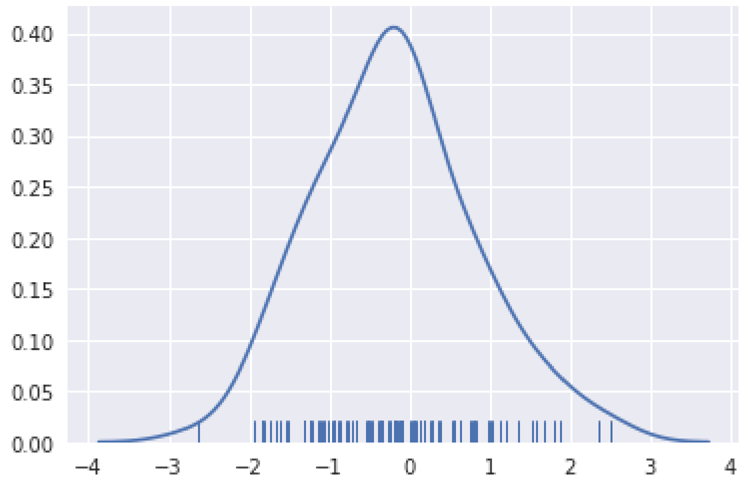


In [19]:

```
sns.distplot(x, hist=False, rug=True)
```

Out[19]:

<matplotlib.axes._subplots.AxesSubplot at 0x7f2bb84a1c88>

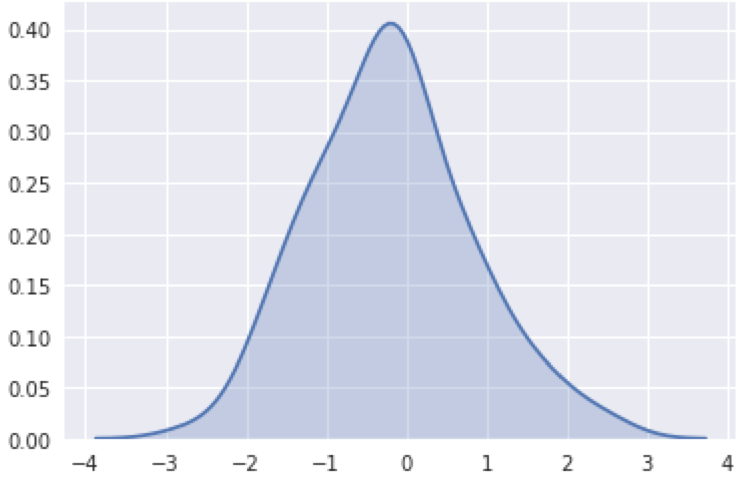


In [20]:

```
sns.kdeplot(x, shade=True)
```

Out[20]:

<matplotlib.axes._subplots.AxesSubplot at 0x7f2baf58d0f0>



Plotting bivariate distributions

In [21]:

```
import pandas as pd
mean, cov = [0, 1], [(1, .5), (.5, 1)]
data = np.random.multivariate_normal(mean, cov, 200)
df = pd.DataFrame(data, columns=["x", "y"])
df.head()
```

Out[21]:

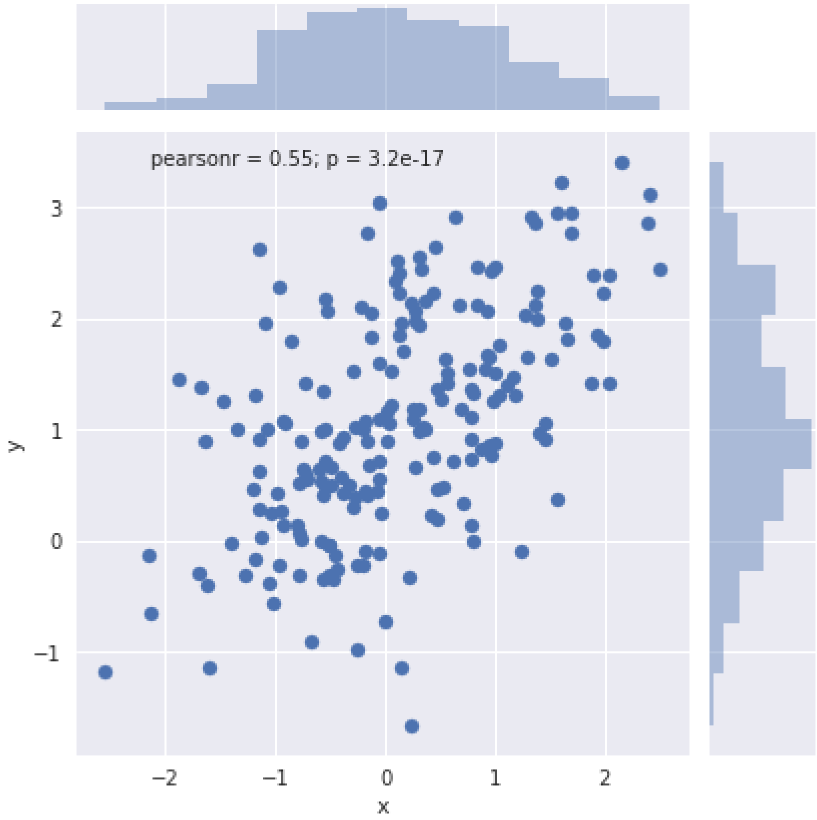
	x	y
0	1.003367	2.469839
1	-0.541031	2.178963
2	1.462172	0.907851
3	0.782053	1.362310
4	0.961787	0.773157

In [22]:

```
sns.jointplot(x="x", y="y", data=df)
```

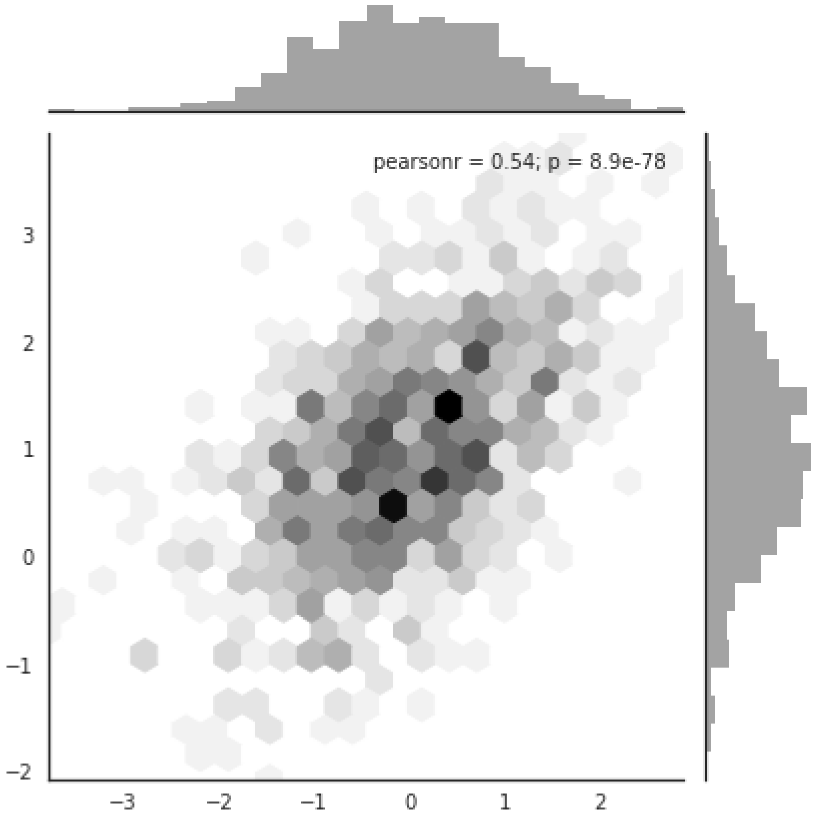
Out[22]:

```
<seaborn.axisgrid.JointGrid at 0x7f2baf6414e0>
```



In [23]:

```
x, y = np.random.multivariate_normal(mean, cov, 1000).T
with sns.axes_style("white"):
    sns.jointplot(x=x, y=y, kind="hex", color="k")
```

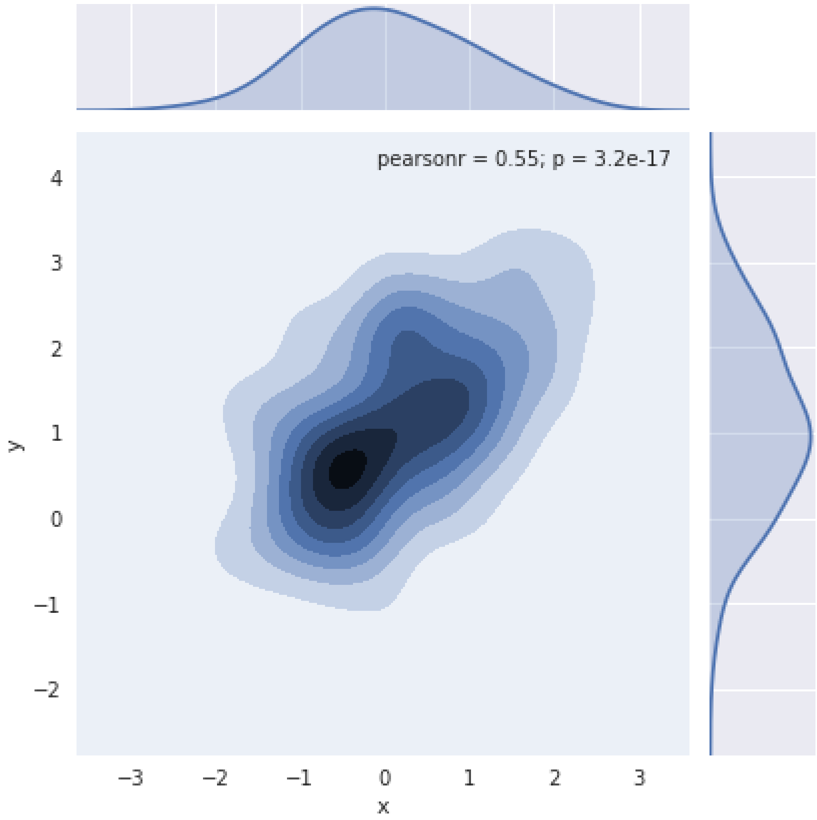


In [24]:

```
sns.jointplot(x="x", y="y", data=df, kind="kde")
```

Out[24]:

```
<seaborn.axisgrid.JointGrid at 0x7f2baf3a3eb8>
```



Pairwise relationships in a dataset

In [25]:

```
iris = sns.load_dataset("iris")
iris.head()
```

Out[25]:

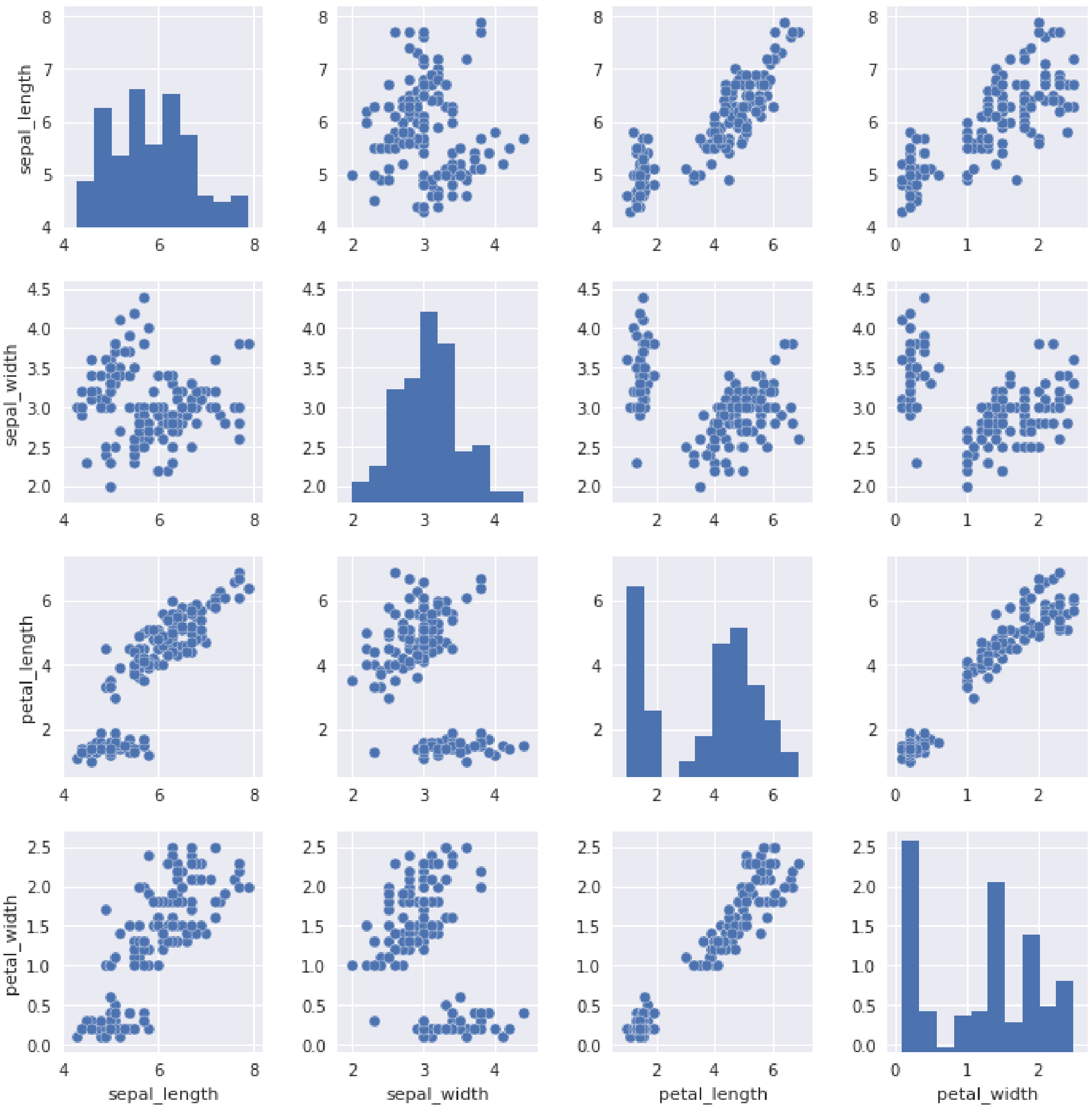
	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

In [28]:

```
sns.pairplot(iris)
```

Out[28]:

<seaborn.axisgrid.PairGrid at 0x7f2baf241d30>



Regression models

In [29]:

```
tips = pd.read_csv('tips.csv')
tips.head()
```

Out[29]:

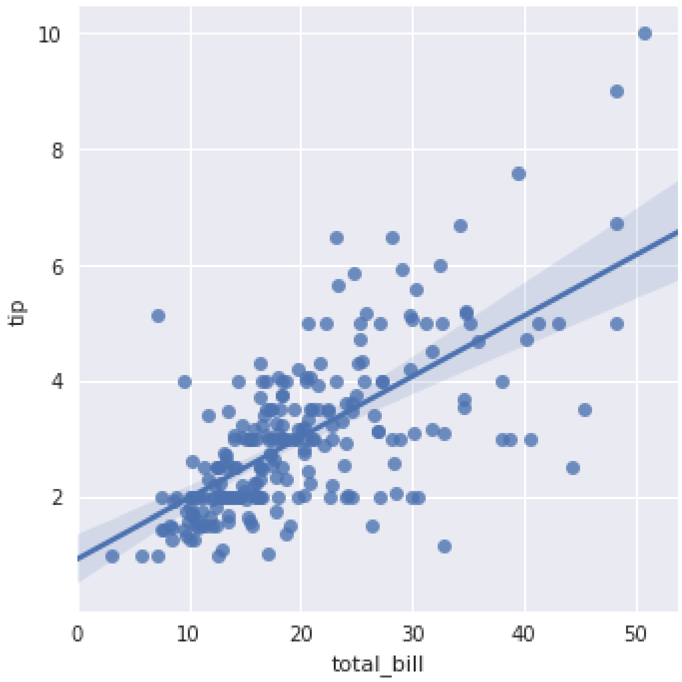
	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

In [30]:

```
sns.lmplot(x="total_bill", y="tip", data=tips)
```

Out[30]:

<seaborn.axisgrid.FacetGrid at 0x7f2bae494f98>

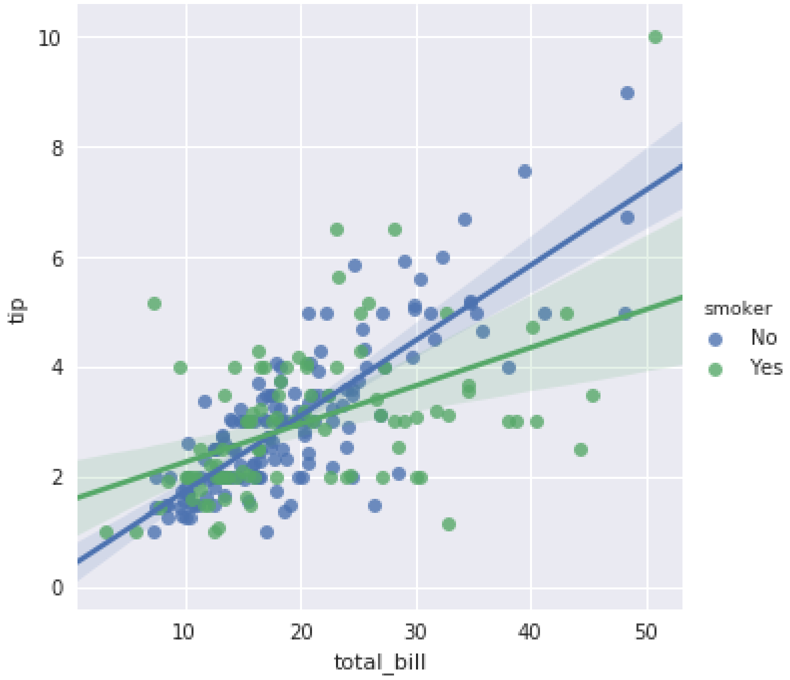


In [31]:

```
sns.lmplot(x="total_bill", y="tip", hue="smoker", data=tips)
```

Out[31]:

<seaborn.axisgrid.FacetGrid at 0x7f2bae419208>

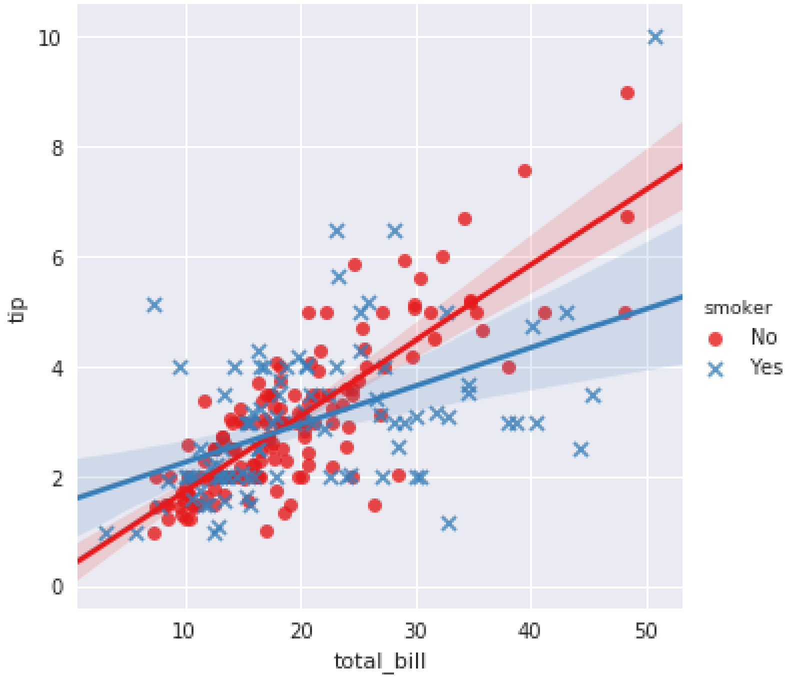


In [32]:

```
sns.lmplot(x="total_bill", y="tip", hue="smoker", data=tips,
           markers=["o", "x"], palette="Set1")
```

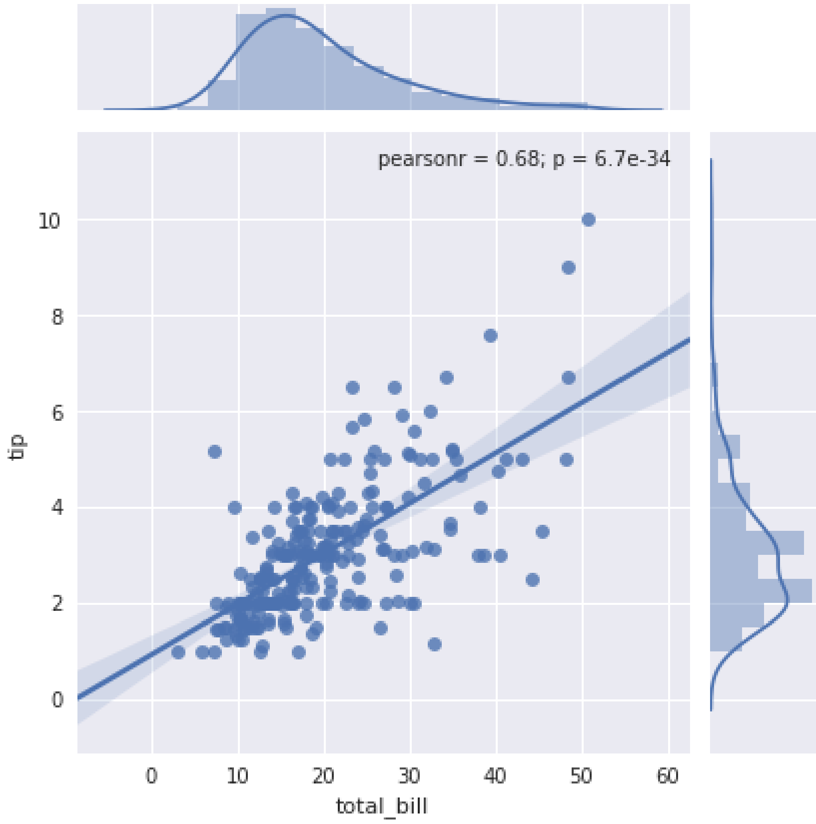
Out[32]:

<seaborn.axisgrid.FacetGrid at 0x7f2bae252438>



In [33]:

```
sns.jointplot(x="total_bill", y="tip", data=tips, kind="reg");
```



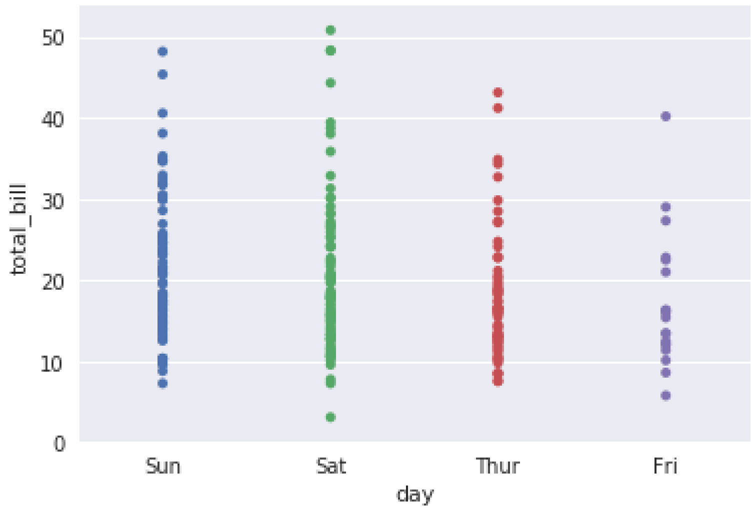
Discrete data

In [34]:

```
sns.stripplot(x="day", y="total_bill", data=tips)
```

Out[34]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2bae054588>
```

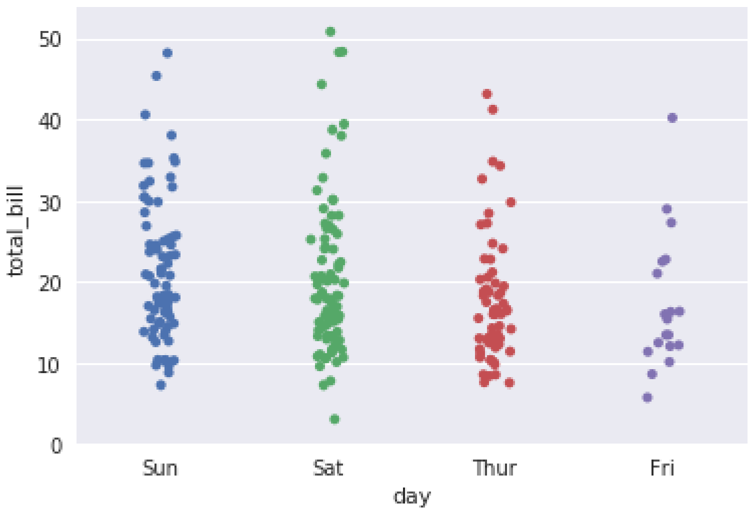


In [35]:

```
sns.stripplot(x="day", y="total_bill", data=tips, jitter=True)
```

Out[35]:

<matplotlib.axes._subplots.AxesSubplot at 0x7f2bae0da550>

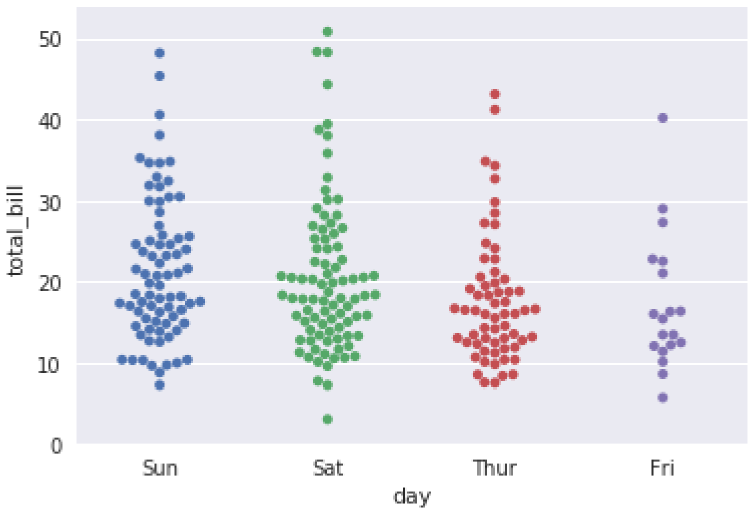


In [36]:

```
sns.swarmplot(x="day", y="total_bill", data=tips)
```

Out[36]:

<matplotlib.axes._subplots.AxesSubplot at 0x7f2bae06fe10>

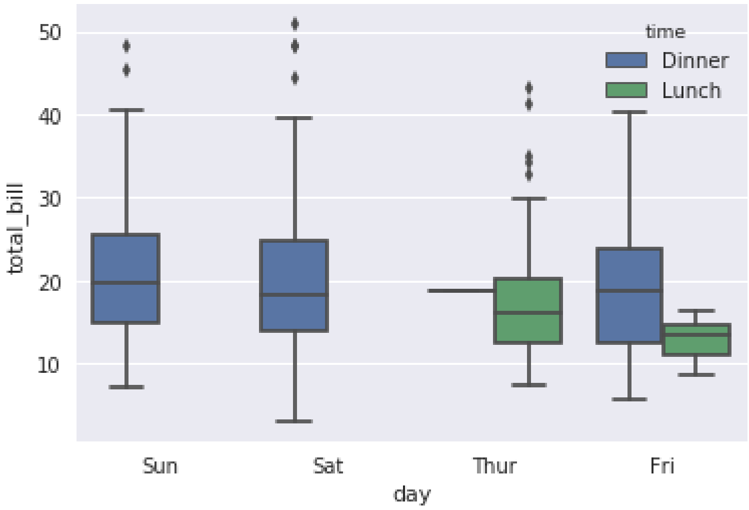


In [37]:

```
sns.boxplot(x="day", y="total_bill", hue="time", data=tips)
```

Out[37]:

<matplotlib.axes._subplots.AxesSubplot at 0x7f2bae027a20>

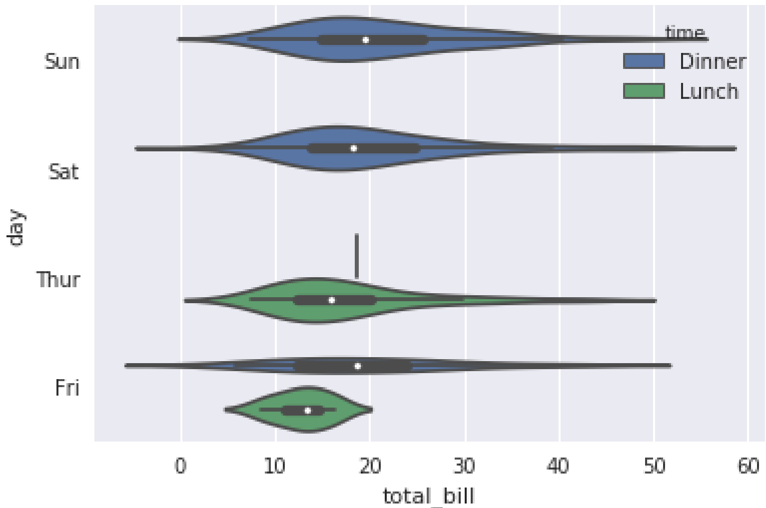


In [38]:

```
sns.violinplot(x="total_bill", y="day", hue="time", data=tips) #combination of b
oxplot and kde
```

Out[38]:

<matplotlib.axes._subplots.AxesSubplot at 0x7f2badeaa898>

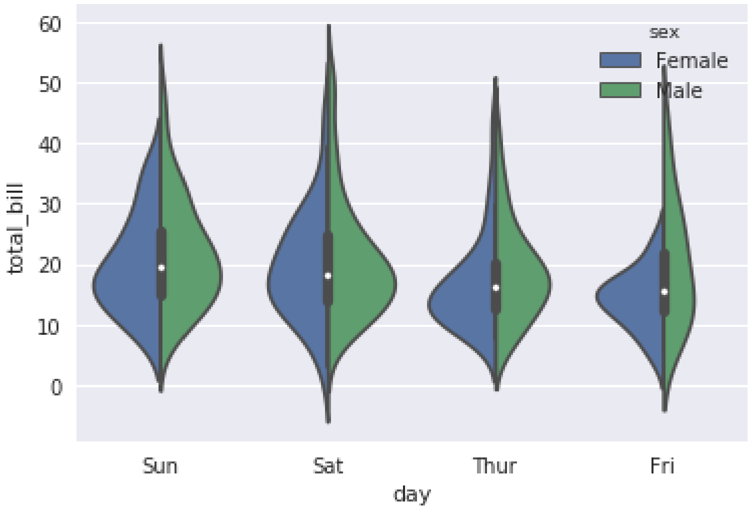


In [39]:

```
sns.violinplot(x="day", y="total_bill", hue="sex", data=tips, split=True)
```

Out[39]:

<matplotlib.axes._subplots.AxesSubplot at 0x7f2bade33128>

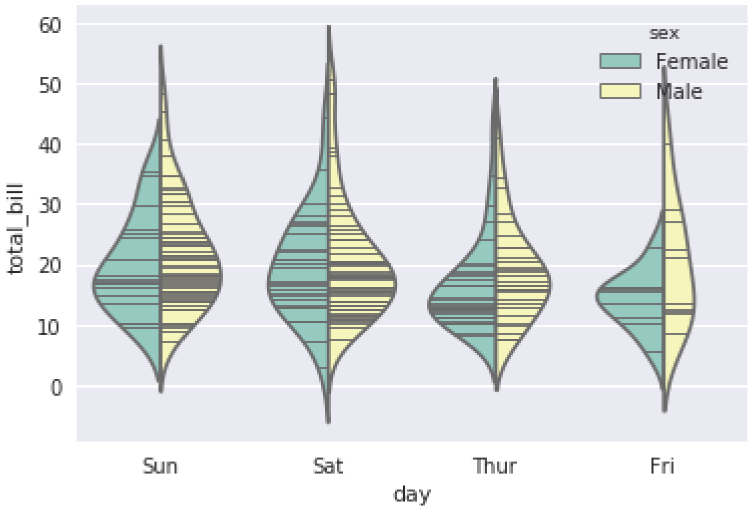


In [40]:

```
sns.violinplot(x="day", y="total_bill", hue="sex", data=tips,
               split=True, inner="stick", palette="Set3")
```

Out[40]:

<matplotlib.axes._subplots.AxesSubplot at 0x7f2bade4a630>

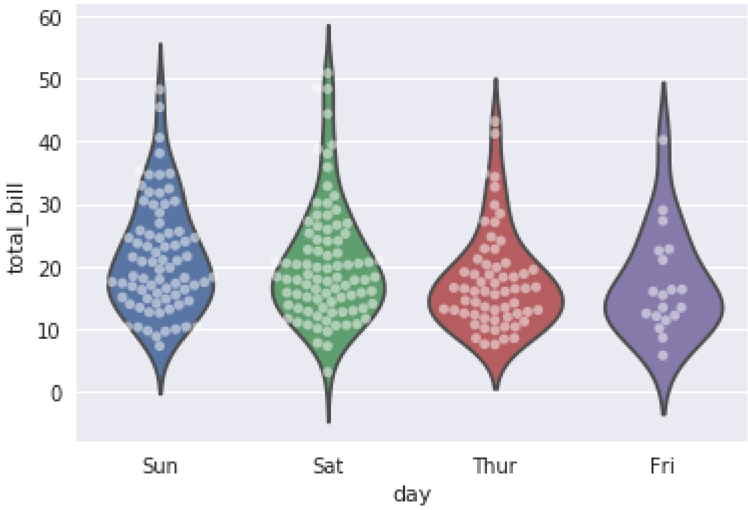


In [41]:

```
sns.violinplot(x="day", y="total_bill", data=tips, inner=None)
sns.swarmplot(x="day", y="total_bill", data=tips, color="w", alpha=.5)
```

Out[41]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2badabbef0>
```



Vincent

From Vega website (https://vega.github.io/vega/ (https://vega.github.io/vega/)):

> Vega is a visualization grammar, a declarative language for creating, saving, and sharing interactive visualization designs. With Vega, you can describe the visual appearance and interactive behavior of a visualization in a JSON format, and generate web-based views using Canvas or SVG.

- https://vincent.readthedocs.io/en/latest/ (https://vincent.readthedocs.io/en/latest/)
- a Python to Vega translator
- combines the data capabilities of Python and the visualization capabilities of JavaScript
- integrates very well with `pandas`

First steps

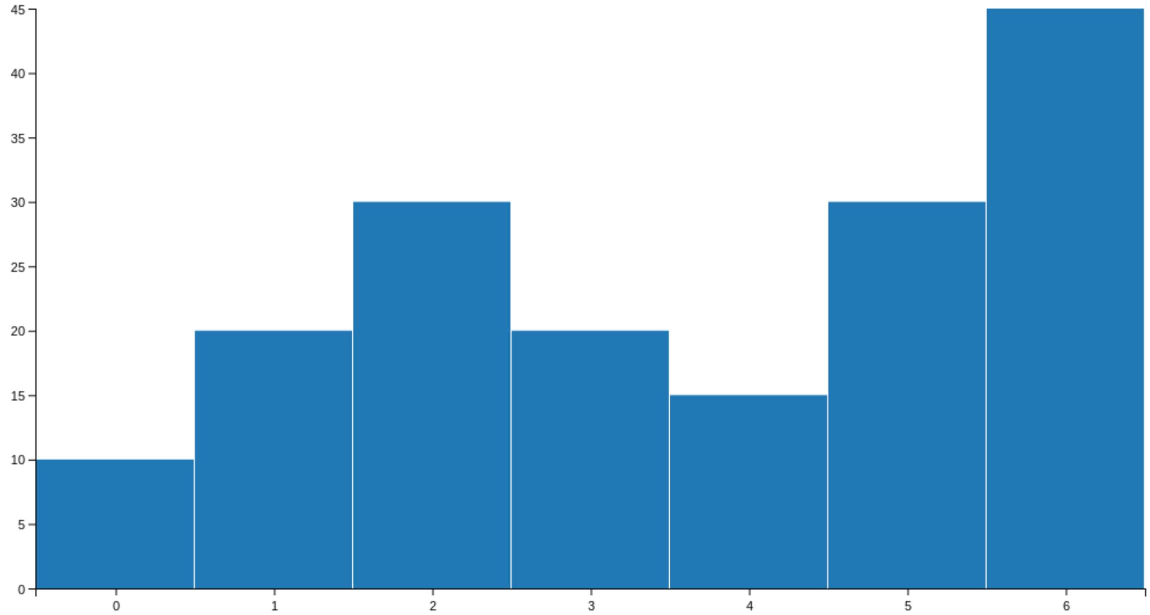
In [10]:

```
import vincent
vincent.core.initialize_notebook() #to show the results in the notebook
```

In [11]:

```
list_data = [10, 20, 30, 20, 15, 30, 45]
bar = vincent.Bar(list_data)
bar
```

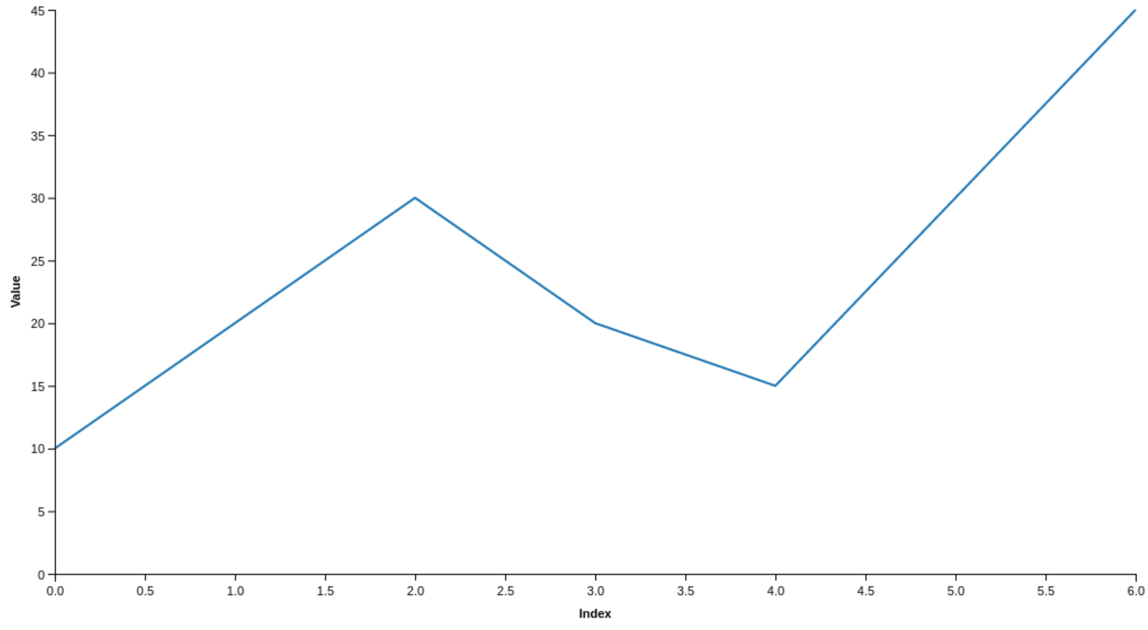
Out[11]:



In [12]:

```
line = vincent.Line(list_data)
line.axis_titles(x='Index', y='Value')
```

Out[12]:



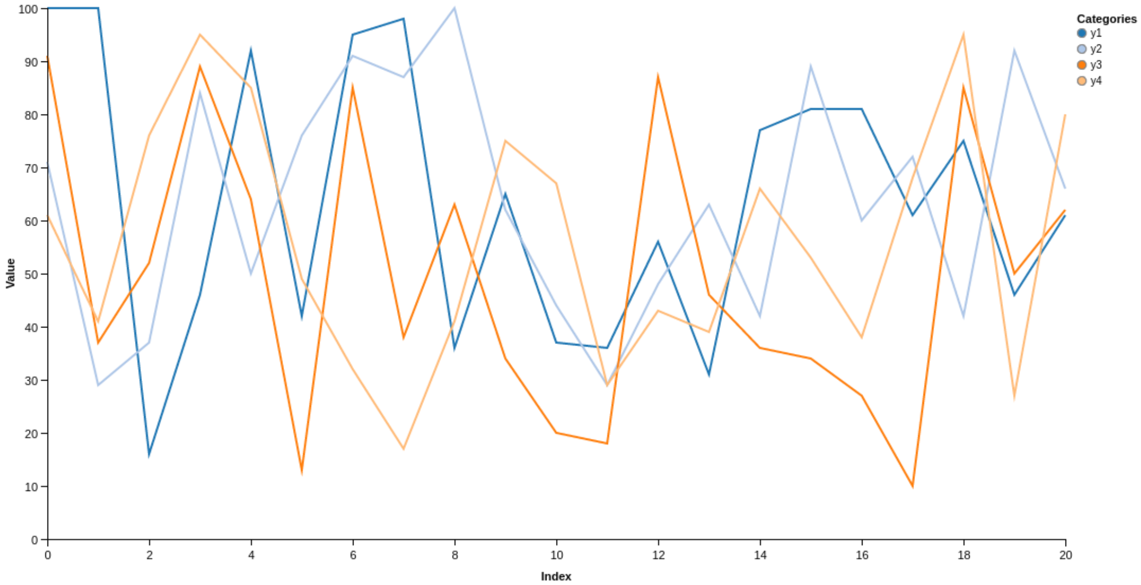
In [13]:

```
import random
cat_1 = ['y1', 'y2', 'y3', 'y4']
index_1 = range(0, 21, 1)
multi_iter1 = {'index': index_1}
for cat in cat_1:
    multi_iter1[cat] = [random.randint(10, 100) for x in index_1]
```

In [14]:

```
line = vincent.Line(multi_iter1, iter_idx='index')
line.axis_titles(x='Index', y='Value')
line.legend(title='Categories')
```

Out[14]:



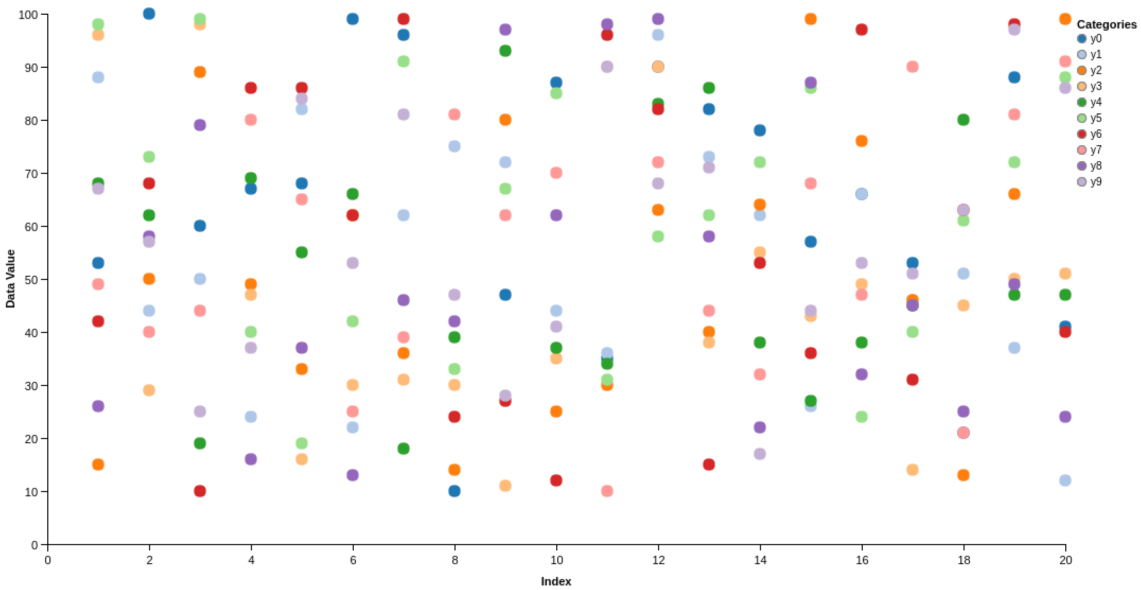
In [15]:

```
cat_2 = ['y' + str(x) for x in range(0, 10, 1)]
index_2 = range(1, 21, 1)
multi_iter2 = {'index': index_2}
for cat in cat_2:
    multi_iter2[cat] = [random.randint(10, 100) for x in index_2]
```

In [16]:

```
scatter = vincent.Scatter(multi_iter2, iter_idx='index')
scatter.axis_titles(x='Index', y='Data Value')
scatter.legend(title='Categories')
```

Out[16]:



In [17]:

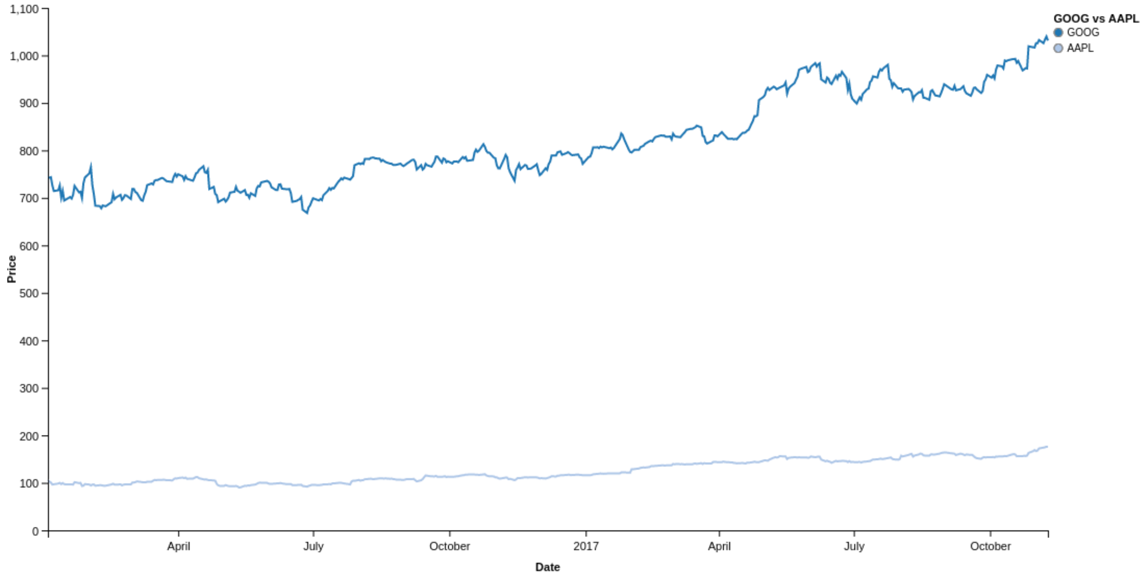
```
import pandas_datareader.data as web
import datetime
import pandas as pd

start = datetime.datetime(2016,1,1)
end = datetime.datetime(2017,11,9)
all_data = {}
for ticker in ['AAPL', 'GOOG','IBM', 'YHOO', 'MSFT']:
    all_data[ticker] = web.DataReader(ticker,'google', start, end)
price = pd.DataFrame({tic: data['Close']
                      for tic, data in all_data.items()})
```

Warning! In case you get a `UnicodeDecodeError` at this point, check if `GoogleDailyReader.url()` in `pandas_datareader/google/daily.py` returns 'http://www.google.com/finance/historical (http://www.google.com/finance/historical)'. If so, change it to 'http://finance.google.com/finance/historical (http://finance.google.com/finance/historical)'.

In [18]:

```
line = vincent.Line(price[['GOOG', 'AAPL']])
line.axis_titles(x='Date', y='Price')
line.legend(title='GOOG vs AAPL')
```

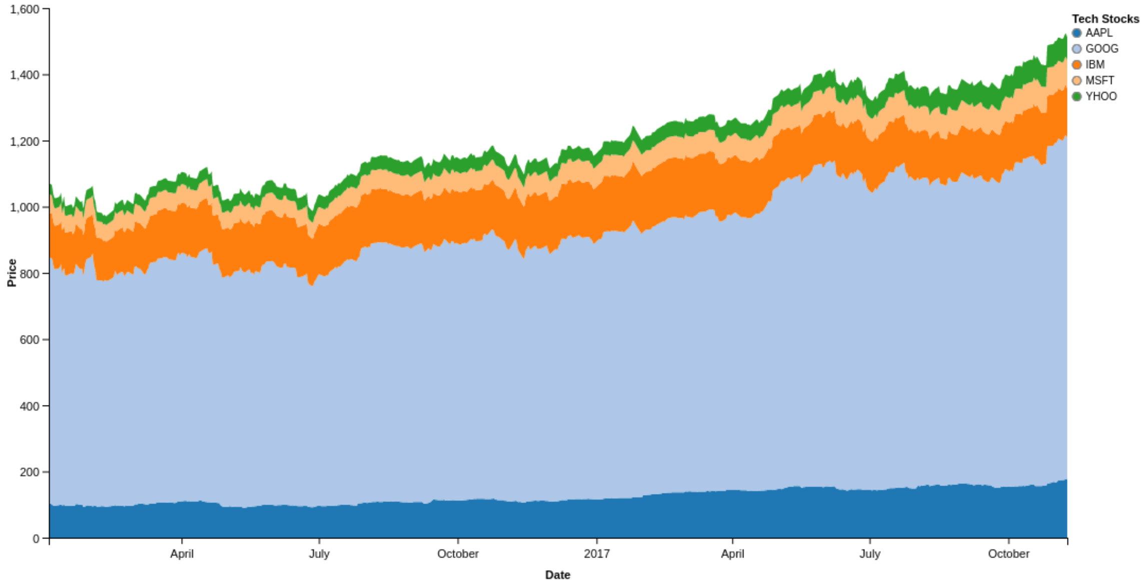
Out[18]:



In [19]:

```
stacked = vincent.StackedArea(price)
stacked.axis_titles(x='Date', y='Price')
stacked.legend(title='Tech Stocks')
```

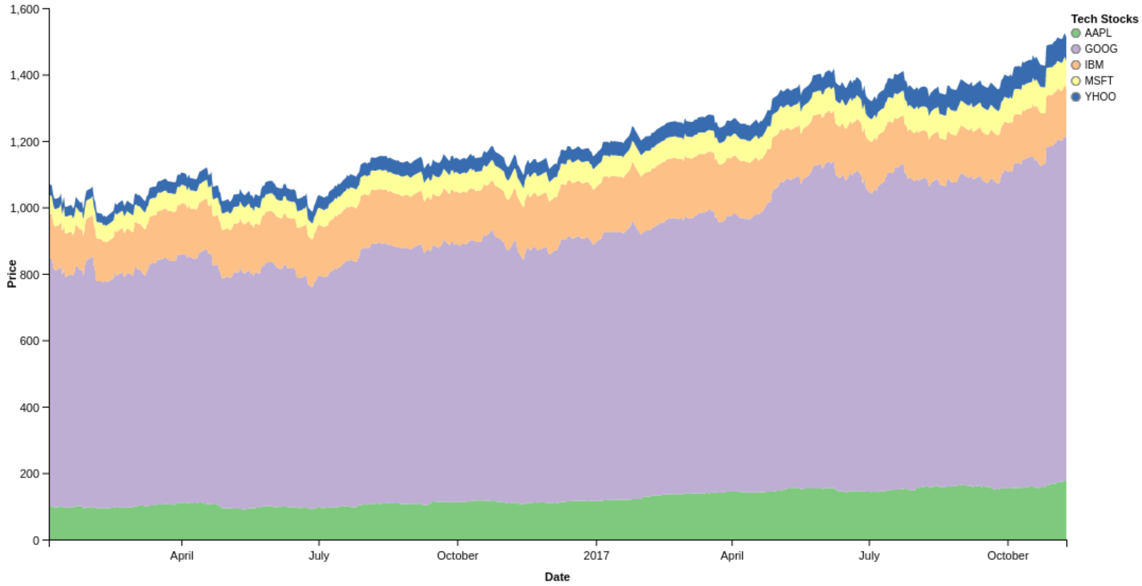
Out[19]:



In [20]:

```
stacked = vincent.StackedArea(price)
stacked.axis_titles(x='Date', y='Price')
stacked.legend(title='Tech Stocks')
stacked.colors(brew='Accent')
```

Out[20]:



In [21]:

```
cat_4 = ['Metric_' + str(x) for x in range(0, 10, 1)]
index_4 = ['Data 1', 'Data 2', 'Data 3', 'Data 4']
data_3 = {}
for cat in cat_4:
    data_3[cat] = [random.randint(10, 100) for x in index_4]
df_2 = pd.DataFrame(data_3, index=index_4)
```

In [22]:

```
df_2.head()
```

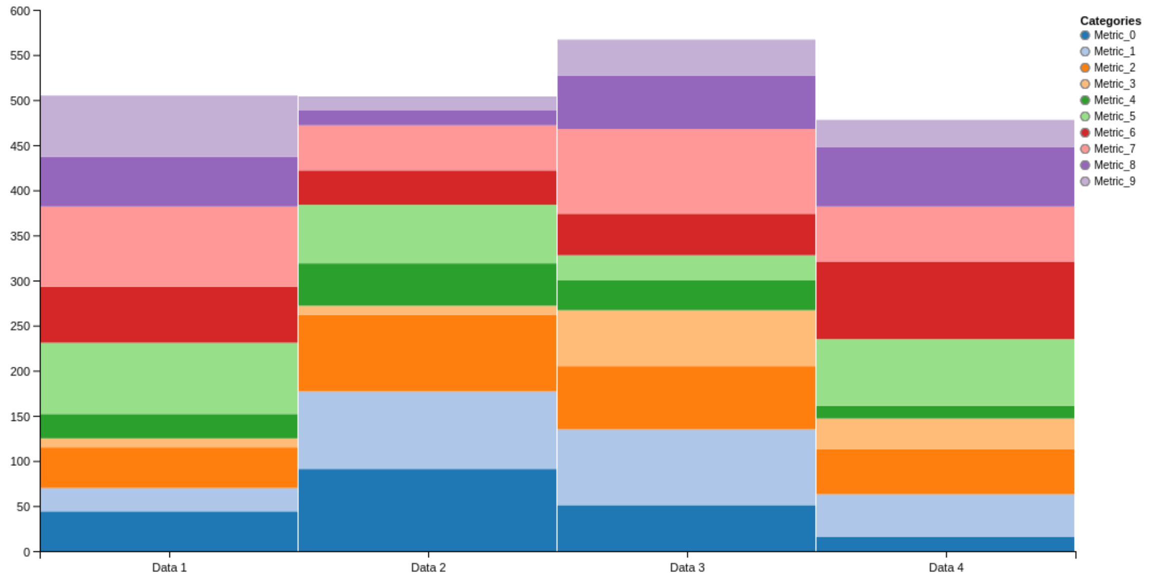
Out[22]:

	Metric_0	Metric_1	Metric_2	Metric_3	Metric_4	Metric_5	Metric_6	Metric_7	M
Data 1	44	26	45	10	27	79	62	89	5
Data 2	91	86	85	10	47	65	38	50	1
Data 3	51	84	70	62	33	28	46	94	5
Data 4	16	47	50	34	14	74	86	61	6

In [23]:

```
stack = vincent.StackedBar(df_2)
stack.legend(title='Categories')
```

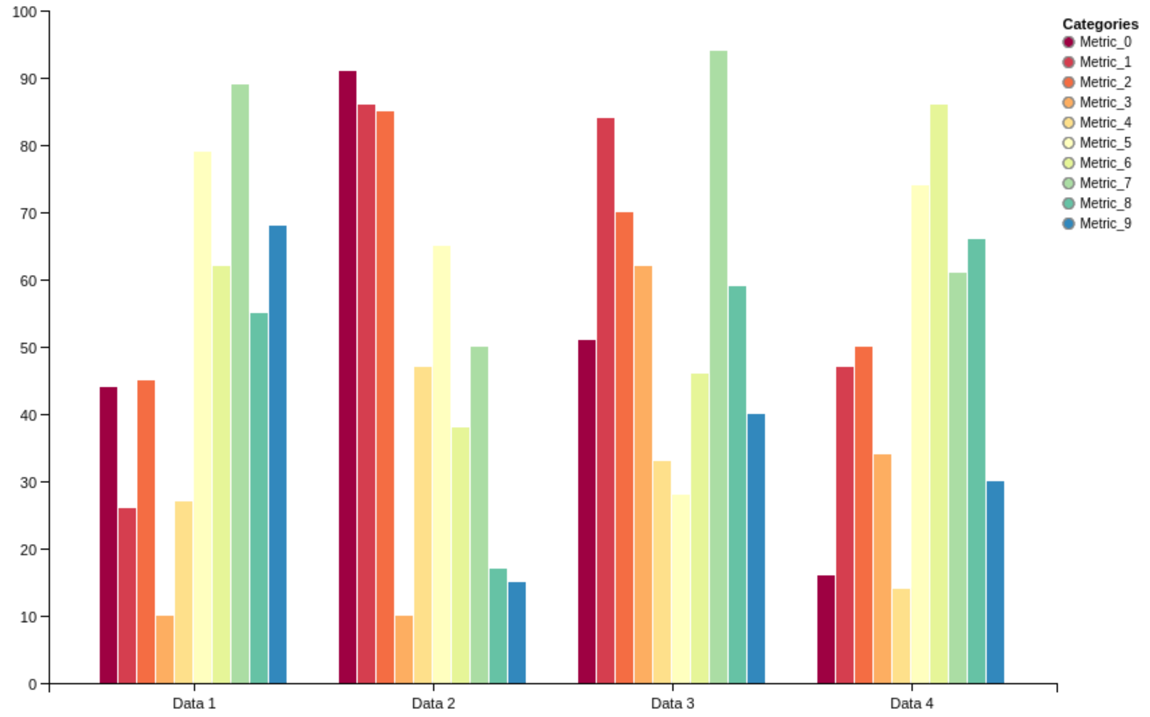
Out[23]:



In [24]:

```
group = vincent.GroupedBar(df_2)
group.legend(title='Categories')
group.colors(brew='Spectral')
group.width=750
group
```

Out[24]:



Plotly for Python

- Plotly (https://plot.ly (https://plot.ly))
- interactive charting libraries for R, Python, and JavaScript
- charts may be hosted online (good for collaboration)
- offline plotting possible

In [25]:

```
import plotly as py
import plotly.graph_objs as go

py.offline.plot({
    "data": [go.Scatter(x=[1, 2, 3, 4], y=[4, 3, 2, 1])],
    "layout": go.Layout(title="My first plotly plot")
})
```

Out[25]:

```
'file:///home/szwabin/Dropbox/Zajecia/UnstructuredData/9_visualizati
on/temp-plot.html'
```

In [26]:

```
import numpy as np

N = 1000
random_x = np.random.randn(N)
random_y = np.random.randn(N)

# Create a trace
trace = go.Scatter(
    x = random_x,
    y = random_y,
    mode = 'markers'
)

data = [trace]

py.offline.plot(data, filename='basic-scatter.html')
```

Out[26]:

```
'file:///home/szwabin/Dropbox/Zajecia/UnstructuredData/9_visualizati
on/basic-scatter.html'
```

In [35]:

```
labels = ['Oxygen','Hydrogen','Carbon_Dioxide','Nitrogen']
values = [4500,2500,1053,500]

trace = go.Pie(labels=labels, values=values)

py.offline.plot([trace], filename='basic_pie_chart.html')
```

Out[35]:

```
'file:///home/szwabin/Dropbox/Zajecia/UnstructuredData/9_visualizati
on/basic_pie_chart.html'
```

In [36]:

```
y0 = np.random.randn(50)-1
y1 = np.random.randn(50)+1

trace0 = go.Box(
    y=y0
)
trace1 = go.Box(
    y=y1
)
data = [trace0, trace1]
py.offline.plot(data)
```

Out[36]:

```
'file:///home/szwabin/Dropbox/Zajecia/UnstructuredData/9_visualizati
on/temp-plot.html'
```

In [37]:

```
trace0 = go.Box(
    y=[2.37, 2.16, 4.82, 1.73, 1.04, 0.23, 1.32, 2.91, 0.11, 4.51, 0.51, 3.75,
1.35, 2.98, 4.50, 0.18, 4.66, 1.30, 2.06, 1.19],
    name='Only Mean',
    marker=dict(
        color='rgb(8, 81, 156)',
    ),
    boxmean=True
)
trace1 = go.Box(
    y=[2.37, 2.16, 4.82, 1.73, 1.04, 0.23, 1.32, 2.91, 0.11, 4.51, 0.51, 3.75,
1.35, 2.98, 4.50, 0.18, 4.66, 1.30, 2.06, 1.19],
    name='Mean & SD',
    marker=dict(
        color='rgb(10, 140, 208)',
    ),
    boxmean='sd'
)
data = [trace0, trace1]
py.offline.plot(data)
```

Out[37]:

```
'file:///home/szwabin/Dropbox/Zajecia/UnstructuredData/9_visualizati
on/temp-plot.html'
```

In [38]:

```
data = go.Data([
    go.Contour(
        z=[[10, 10.625, 12.5, 15.625, 20],
           [5.625, 6.25, 8.125, 11.25, 15.625],
           [2.5, 3.125, 5., 8.125, 12.5],
           [0.625, 1.25, 3.125, 6.25, 10.625],
           [0, 0.625, 2.5, 5.625, 10]]
    )
])
py.offline.plot(data)
```

Out[38]:

```
'file:///home/szwabin/Dropbox/Zajecia/UnstructuredData/9_visualizati
on/temp-plot.html'
```

Folium

- https://github.com/python-visualization/folium (https://github.com/python-visualization/folium)
- Python data, leaflet.js maps

From Leaflet.js website (http://leafletjs.com/ (http://leafletjs.com/)):

> Leaflet is the leading open-source JavaScript library for mobile-friendly interactive maps. Weighing just about 38 KB of JS, it has all the mapping features most developers ever need. Leaflet is designed with simplicity, performance and usability in mind. It works efficiently across all major desktop and mobile platforms, can be extended with lots of plugins, has a beautiful, easy to use and well-documented API and a simple, readable source code that is a joy to contribute to.

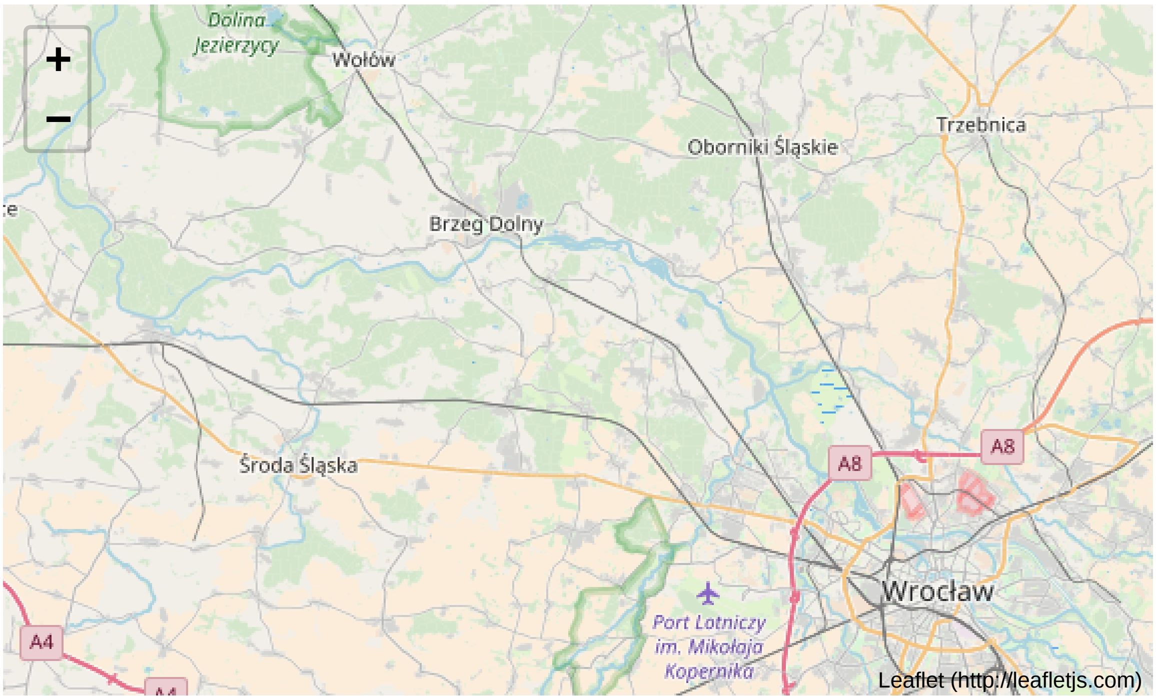
- Folium allows to visualize data on a Leaflet map without the knowledge of JavaScript
- it enables both the binding of data to a map for choropleth visualizations as well as passing rich vector/raster/HTML visualizations as markers on the map
- a number of built-in tilesets from OpenStreetMap, Mapbox, and Stamen
- supports custom tilesets with Mapbox or Cloudmade API keys
- supports Image, Video, GeoJSON and TopoJSON overlays

First steps

In [27]:

```
import folium
map_osm = folium.Map(location=[51.109495,17.061096399999997])
map_osm
```

Out[27]:



Markers

In [28]:

```
map_1 = folium.Map(location=[51.109495,17.061096399999997],zoom_start=16)
folium.Marker([51.109495,17.061096399999997], popup='WMat').add_to(map_1)
folium.Marker([51.1090988,17.060515], popup='Weka').add_to(map_1)
map_1
```

Out[28]:

In [29]:

```
map_1 = folium.Map(location=[51.109495,17.061096399999997],zoom_start=16)


folium.Marker([51.109495,17.061096399999997],
              popup='WMat',
              icon=folium.Icon(color='red')).add_to(map_1)
folium.Marker([51.1090988,17.060515],
              popup='Weka',
              icon=folium.Icon(color='green')).add_to(map_1)
map_1
```

Out[29]:

Map and marker style

In [30]:

```
map_2 = folium.Map(location=[51.109495,17.061096399999997],
                   zoom_start=16,
                   tiles='Stamen Toner')

folium.CircleMarker([51.109495,17.061096399999997],
                    popup='WMat',
                    radius=10,
                    color='#3186cc',
                    fill_color='#3186cc').add_to(map_2)
folium.Marker([51.1090988,17.060515], popup='Weka').add_to(map_2)

map_2
```

Out[30]:

Reading geographical coordinates

In [31]:

```
map_3 = folium.Map(
    location=[51.109495,17.061096399999997],
    zoom_start=16)
map_3.add_child(folium.LatLngPopup())
map_3
```

Out[31]:

Interactive marker placement

In [32]:

```
map_4 = folium.Map(location=[51.109495,17.061096399999997],
                    zoom_start=16)
folium.Marker([51.109495,17.061096399999997],
               popup='WMat',
               icon=folium.Icon(color='red')).add_to(map_4)
map_4.add_child(folium.ClickForMarker(popup="My Marker"))
map_4
```

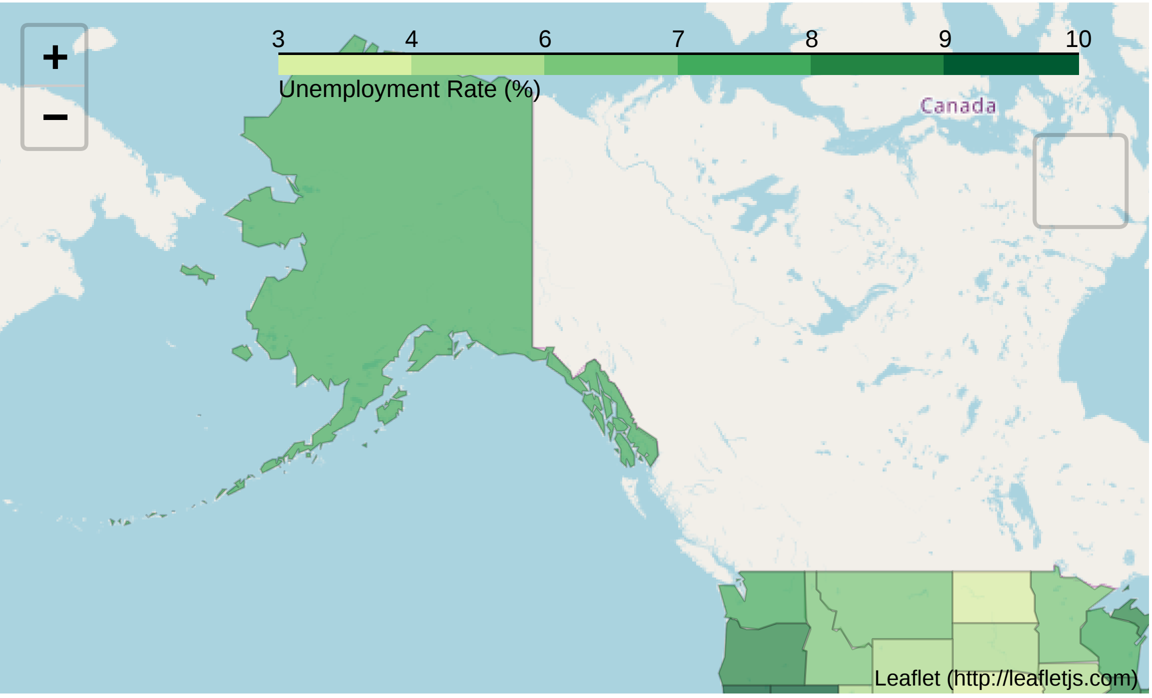
Out[32]:

Choropleth maps

In [33]:

```
import pandas as pd
import os

state_geo = os.path.join('data', 'us-states.json')

state_unemployment = os.path.join('data', 'US_Unemployment_Oct2012.csv')
state_data = pd.read_csv(state_unemployment)

m = folium.Map(location=[48, -102], zoom_start=3)

m.choropleth(
    geo_data=state_geo,
    name='choropleth',
    data=state_data,
    columns=['State', 'Unemployment'],
    key_on='feature.id',
    fill_color='YlGn',
    fill_opacity=0.7,
    line_opacity=0.2,
    legend_name='Unemployment Rate (%)'
)


folium.LayerControl().add_to(m)

m
```

Out[33]:



GeoJSON/TopoJSON Overlays

In [34]:

```
antarctic_ice_edge = os.path.join('data', 'antarctic_ice_edge.json')
antarctic_ice_shelf_topo = os.path.join('data', 'antarctic_ice_shelf_topo.json')

m = folium.Map(
    location=[-59.1759, -11.6016],
    tiles='Mapbox Bright',
    zoom_start=2
)

folium.GeoJson(
    antarctic_ice_edge,
    name='geojson'
).add_to(m)

folium.TopoJson(
    open(antarctic_ice_shelf_topo),
    'objects.antarctic_ice_shelf',
    name='topojson'
).add_to(m)

folium.LayerControl().add_to(m)

m
```

Out[34]:

Combining Folium with Vincent

In [35]:

```
#generate data
import numpy as np
import pandas as pd
import numpy.ma as ma


def make_data():
    x = np.linspace(-np.pi, np.pi, 101)
    sin = np.sin(x)
    cos = np.cos(x)
    cos[20:50] = np.NaN
    return pd.DataFrame(np.asanyarray([sin, cos]).T, columns=['sin', 'cos'], ind
ex=x)

df = make_data()
resolution, width, height = 75, 7, 3
```

In [36]:

```
#instantiate map
station = '42'
lon, lat = -42, -21
mapa = folium.Map(location=[lat, lon], zoom_start=5)
```

In [37]:

```
import json
import vincent

df.fillna(value='null', inplace=True)  # Does not handle missing values.
vis = vincent.Line(df, width=width*resolution, height=height*resolution)
vis.legend(title='Vega');
```

In [38]:

```
vega = folium.Vega(json.loads(vis.to_json()), width="100%", height="100%")
popup = folium.Popup(max_width=vis.width+75).add_child(vega)

icon = folium.Icon(color="green", icon="ok")
marker = folium.Marker(location=[lat-1, lon+1], popup=popup, icon=icon)

mapa.add_child(marker);
```

In [39]:

```
mapa
```

Out[39]:

gmplot

- https://github.com/vgm64/gmplot (https://github.com/vgm64/gmplot)
- plotting data on Google Maps
- a matplotlib-like interface to generate the HTML and javascript to render all the data
- plot types:
  - polygons with fills
  - drop pins
  - scatter points
  - grid lines
  - heatmaps
- poor documentation :(

In [40]:

```
import gmplot
import numpy as np
gmap = gmplot.GoogleMapPlotter(51.109495, 17.061096399999997, 16)

latitudes=np.linspace(51.108,51.11,2)
longitudes=17.06*np.ones(2)

gmap.plot(latitudes, longitudes, 'red', edge_width=10)

gmap.draw("mymap.html")
```

In [41]:

```
import gmplot
import numpy as np
import random as rd
gmap = gmplot.GoogleMapPlotter(51.109495, 17.061096399999997, 16)

latitudes=[rd.uniform(51.108, 51.11) for i in range(10)]
longitudes=[rd.uniform(17.05, 17.07) for i in range(10)]

print(latitudes)
print(longitudes)

gmap.scatter(latitudes, longitudes, 'red', size=10, marker=False)

gmap.draw("mymap.html")
```

```
[51.10860958087966, 51.10936940460042, 51.10913947036361, 51.1084782
26747664, 51.10842810677007, 51.10853425486182, 51.10845223924469, 5
1.10888060996333, 51.10841560411439, 51.109533482429576]
[17.06460002950345, 17.0627760849363, 17.06087301539322, 17.06277738
713114, 17.06825909869605, 17.056284046218746, 17.06413954133867, 1
7.051792381365647, 17.06781141096382, 17.051076761628046]
```

pygal

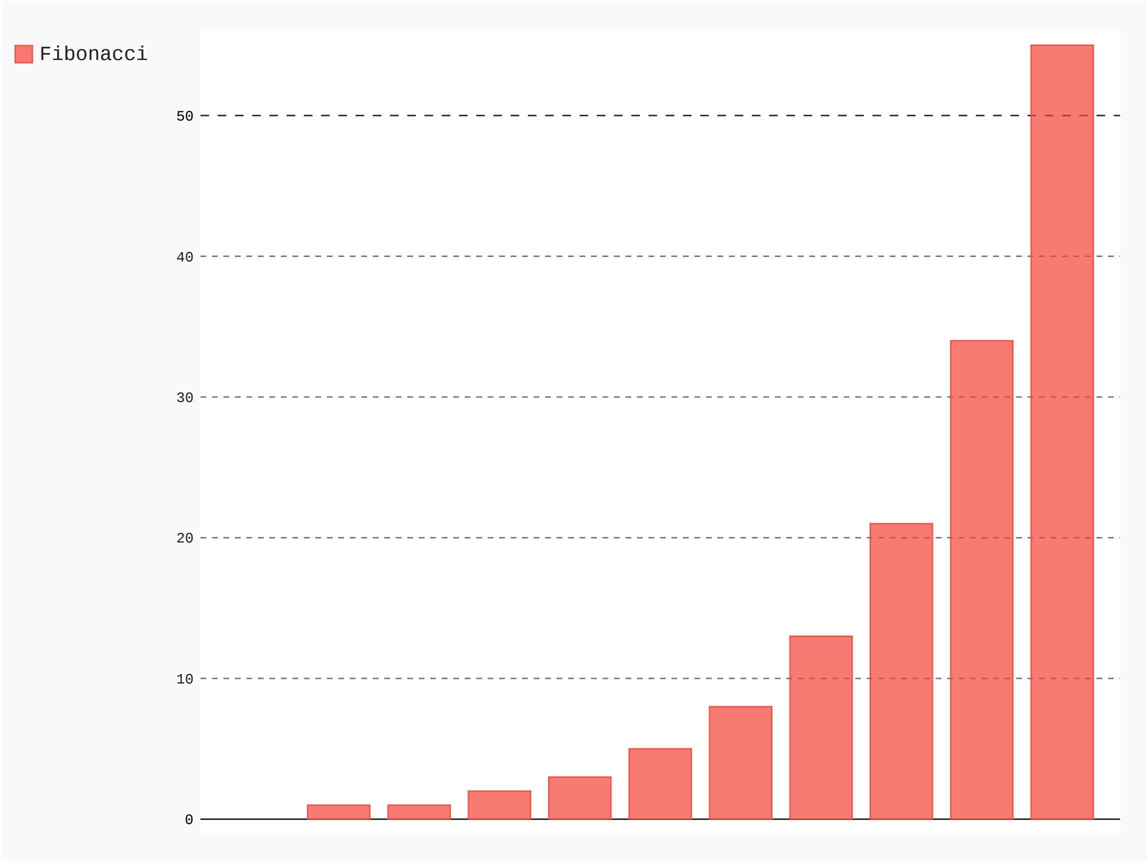
- http://www.pygal.org/ (http://www.pygal.org/)
- a SVG graph plotting library

In [42]:

```
import pygal                                                    # First impor
t pygal
bar_chart = pygal.Bar()                                         # Then create
 a bar graph object
bar_chart.add('Fibonacci', [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55])  # Add some va
lues
bar_chart.render_to_file('bar_chart.svg')                       # Save the sv
g to a file
```

In [43]:

```
from IPython.display import SVG,HTML,display
display(SVG('bar_chart.svg'))
```

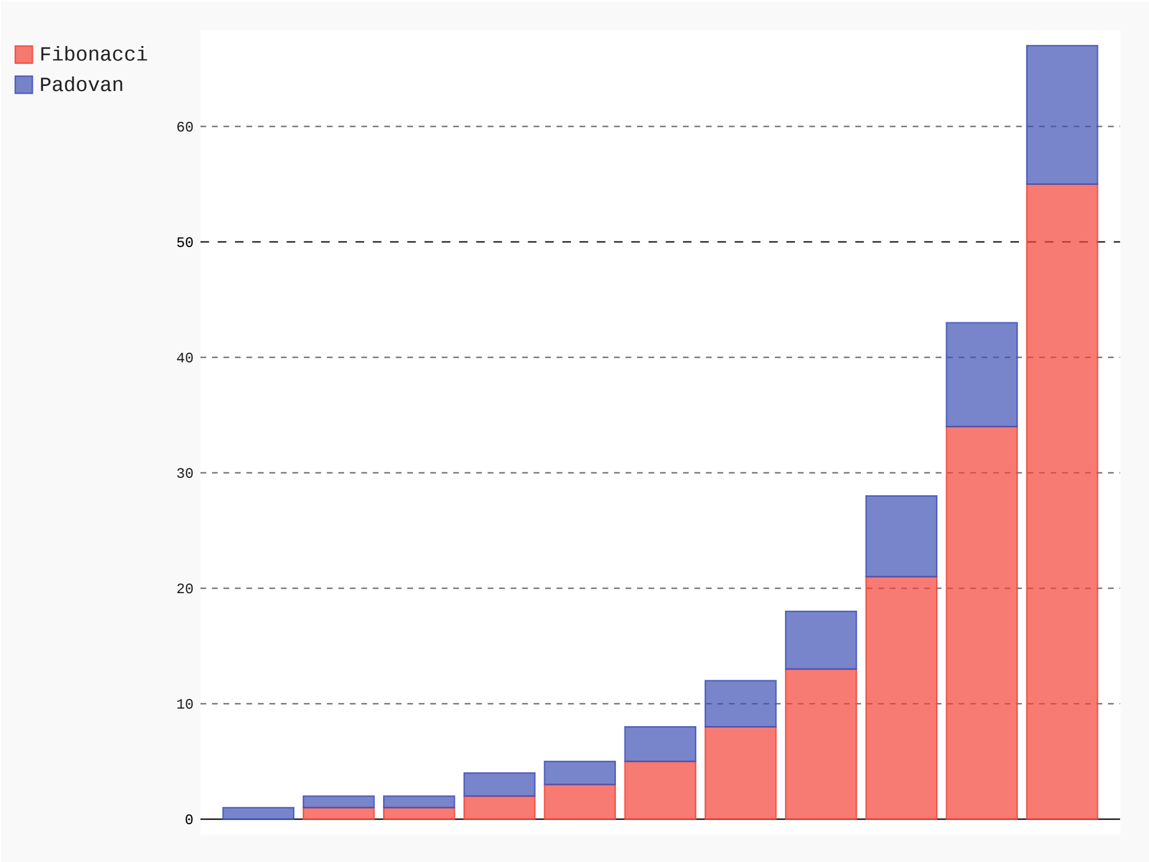


In [44]:

```
bar_chart = pygal.StackedBar()
bar_chart.add('Fibonacci', [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55])
bar_chart.add('Padovan', [1, 1, 1, 2, 2, 3, 4, 5, 7, 9, 12])
bar_chart.render_to_file('bar_chart2.svg')
```

In [45]:

```
display(SVG('bar_chart2.svg'))
```

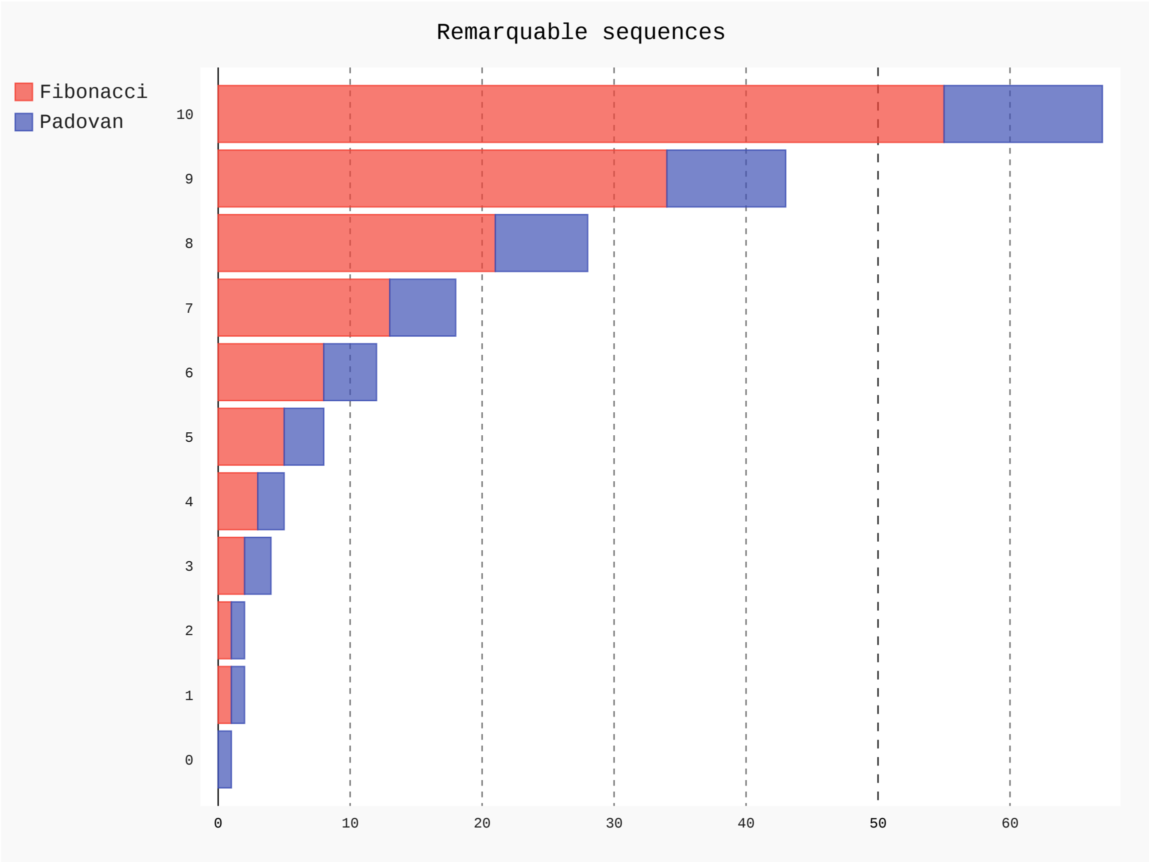


In [46]:

```
bar_chart = pygal.HorizontalStackedBar()
bar_chart.title = "Remarquable sequences"
bar_chart.x_labels = map(str, range(11))
bar_chart.add('Fibonacci', [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55])
bar_chart.add('Padovan', [1, 1, 1, 2, 2, 3, 4, 5, 7, 9, 12])
bar_chart.render_to_file('bar_chart3.svg')
```

In [47]:

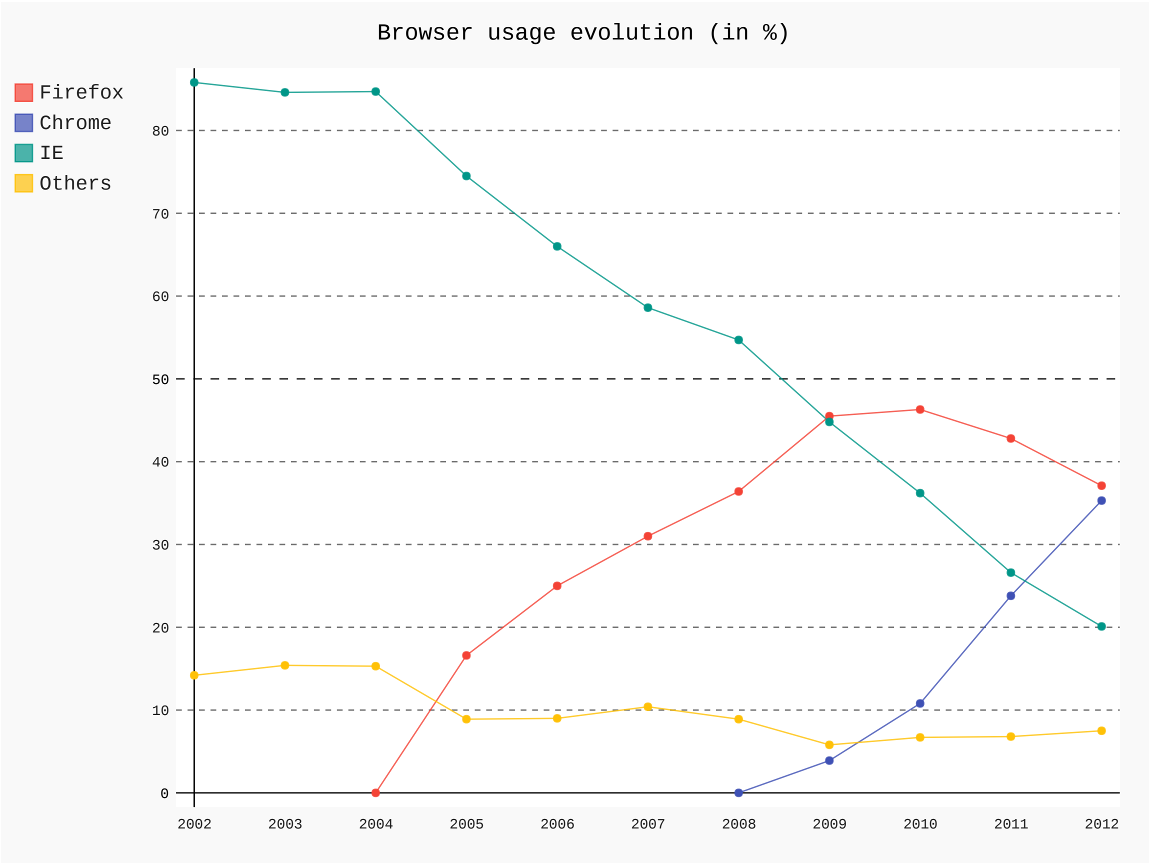
```
display(SVG('bar_chart3.svg'))
```



Displaying the plots directly in the notebook:

In [48]:

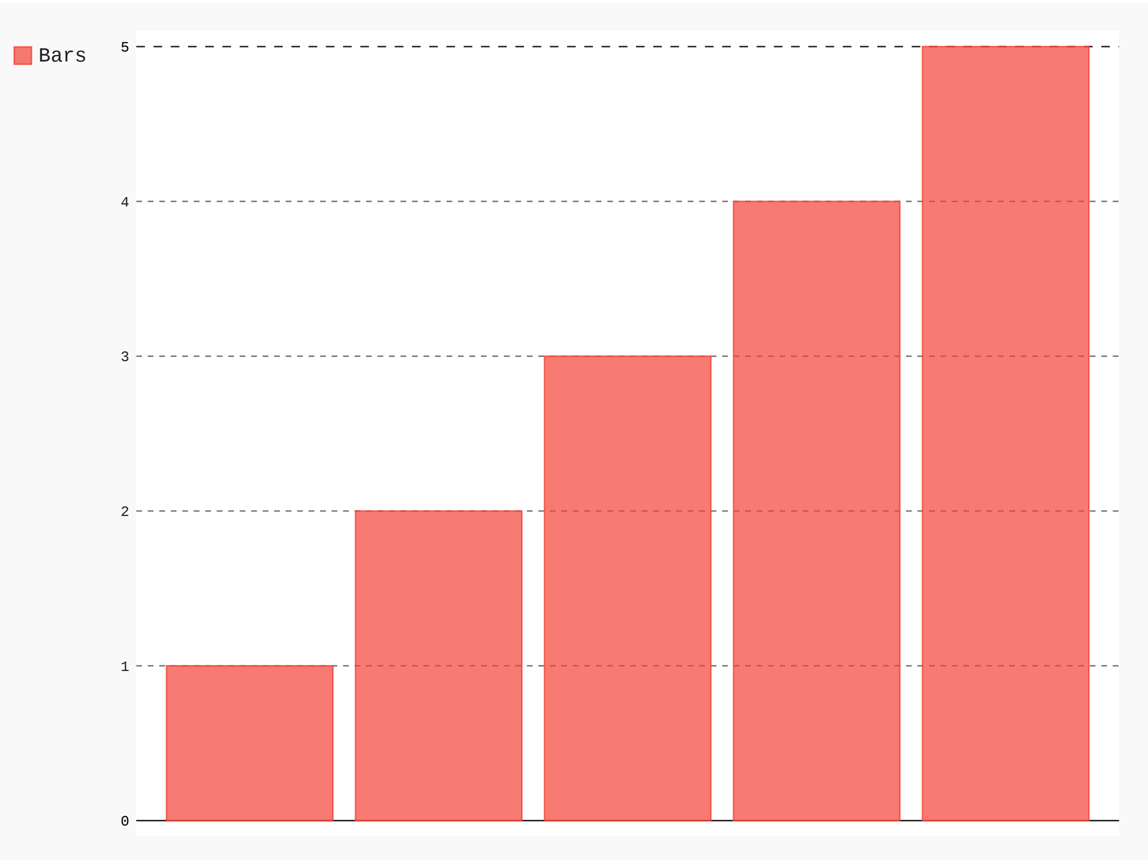
```
line_chart = pygal.Line()
line_chart.title = 'Browser usage evolution (in %)'
line_chart.x_labels = map(str, range(2002, 2013))
line_chart.add('Firefox', [None, None,    0, 16.6,   25,   31, 36.4, 45.5, 46.3,
 42.8, 37.1])
line_chart.add('Chrome',  [None, None, None, None, None, None,    0,  3.9, 10.8,
 23.8, 35.3])
line_chart.add('IE',      [85.8, 84.6, 84.7, 74.5,   66, 58.6, 54.7, 44.8, 36.2,
 26.6, 20.1])
line_chart.add('Others',  [14.2, 15.4, 15.3,  8.9,    9, 10.4,  8.9,  5.8,  6.7,
  6.8,  7.5])
display({'image/svg+xml': line_chart.render()}, raw=True)
```



If you want to see the tooltips in the notebok:

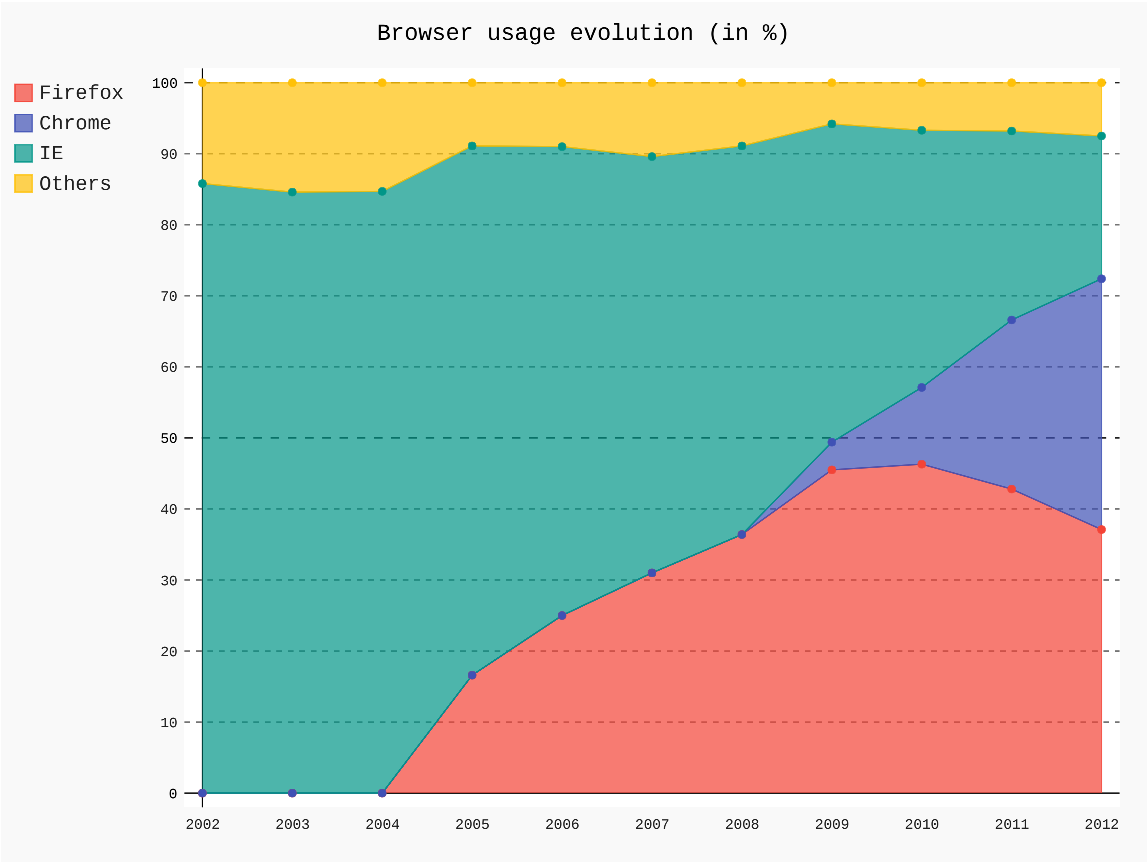
In [49]:

```
base_html = """
<!DOCTYPE html>
<html>
  <head>
  <script type="text/javascript" src="http://kozea.github.com/pygal.js/javascrip
ts/svg.jquery.js"></script>
  <script type="text/javascript" src="https://kozea.github.io/pygal.js/2.0.x/pyg
al-tooltips.min.js""></script>
  </head>
  <body>
    <figure>{rendered_chart}

    </figure>
  </body>
</html>
"""

def galplot(chart):
    rendered_chart = chart.render(is_unicode=True)
    plot_html = base_html.format(rendered_chart=rendered_chart)
    display(HTML(plot_html))

bar_chart = pygal.StackedBar()
bar_chart.add('Bars', [1,2,3,4,5])

galplot(bar_chart)
```



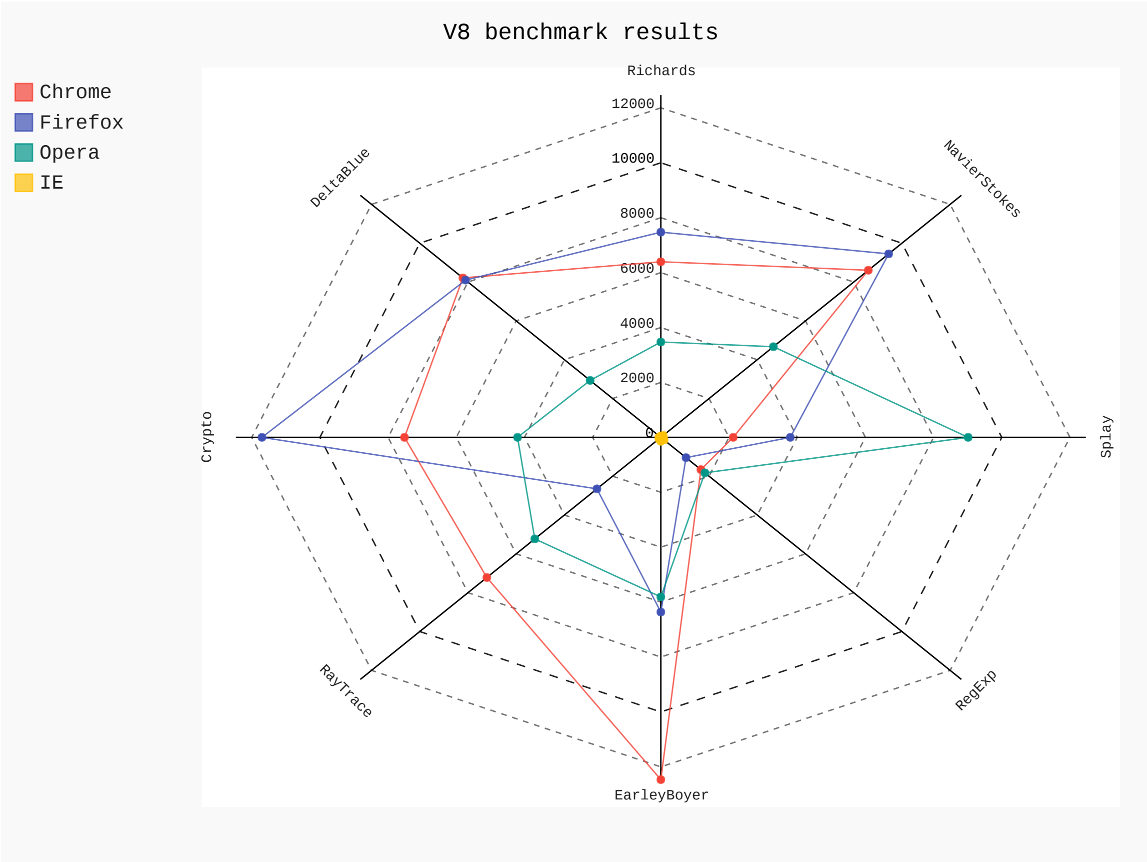
In [50]:

```
stackedline_chart = pygal.StackedLine(fill=True)
stackedline_chart.title = 'Browser usage evolution (in %)'
stackedline_chart.x_labels = map(str, range(2002, 2013))
stackedline_chart.add('Firefox', [None, None, 0, 16.6,   25,   31, 36.4, 45.5, 4
6.3, 42.8, 37.1])
stackedline_chart.add('Chrome',  [None, None, None, None, None, None,    0,
3.9, 10.8, 23.8, 35.3])
stackedline_chart.add('IE',      [85.8, 84.6, 84.7, 74.5,   66, 58.6, 54.7,
44.8, 36.2, 26.6, 20.1])
stackedline_chart.add('Others',  [14.2, 15.4, 15.3,  8.9,    9, 10.4,  8.9,
5.8,  6.7,  6.8,  7.5])
galplot(stackedline_chart)
```



In [51]:

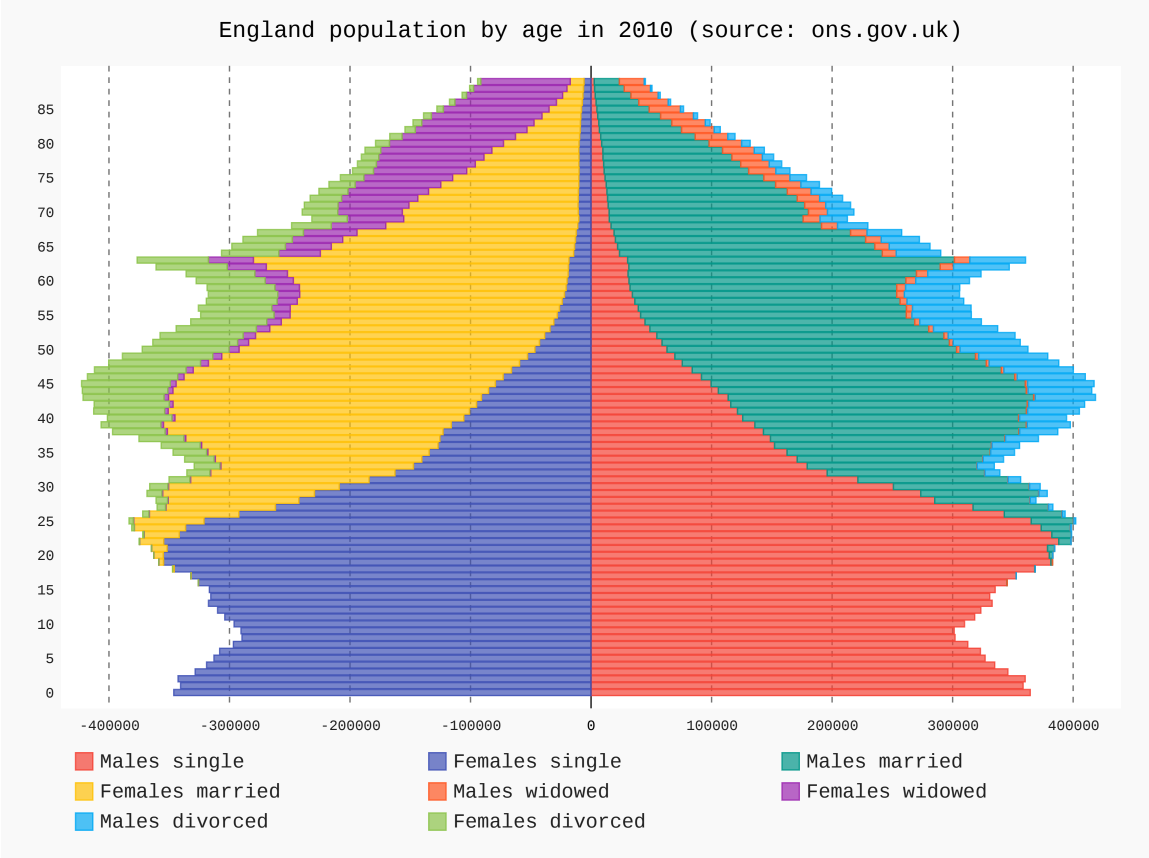
```
radar_chart = pygal.Radar()
radar_chart.title = 'V8 benchmark results'
radar_chart.x_labels = ['Richards', 'DeltaBlue', 'Crypto', 'RayTrace', 'EarleyBo
yer', 'RegExp', 'Splay', 'NavierStokes']
radar_chart.add('Chrome', [6395, 8212, 7520, 7218, 12464, 1660, 2123, 8607])
radar_chart.add('Firefox', [7473, 8099, 11700, 2651, 6361, 1044, 3797, 9450])
radar_chart.add('Opera', [3472, 2933, 4203, 5229, 5810, 1828, 9013, 4669])
radar_chart.add('IE', [43, 41, 59, 79, 144, 136, 34, 102])
galplot(radar_chart)
```



In [52]:

```
ages = [(364381, 358443, 360172, 345848, 334895, 326914, 323053, 312576, 302015,
 301277, 309874, 318295, 323396, 332736, 330759, 335267, 345096, 352685, 368067,
 381521, 380145, 378724, 388045, 382303, 373469, 365184, 342869, 316928, 285137,
 273553, 250861, 221358, 195884, 179321, 171010, 162594, 152221, 148843, 143013,
 135887, 125824, 121493, 115913, 113738, 105612, 99596, 91609, 83917, 75688, 695
38, 62999, 58864, 54593, 48818, 44739, 41096, 39169, 36321, 34284, 32330, 31437,
 30661, 31332, 30334, 23600, 21999, 20187, 19075, 16574, 15091, 14977, 14171, 13
687, 13155, 12558, 11600, 10827, 10436, 9851, 9794, 8787, 7993, 6901, 6422,
5506, 4839, 4144, 3433, 2936, 2615),
   (346205, 340570, 342668, 328475, 319010, 312898, 308153, 296752, 289639, 2904
66, 296190, 303871, 309886, 317436, 315487, 316696, 325772, 331694, 345815, 3546
96, 354899, 351727, 354579, 341702, 336421, 321116, 292261, 261874, 242407, 2294
88, 208939, 184147, 162662, 147361, 140424, 134336, 126929, 125404, 122764, 1160
04, 105590, 100813, 95021, 90950, 85036, 79391, 72952, 66022, 59326, 52716, 4658
2, 42772, 38509, 34048, 30887, 28053, 26152, 23931, 22039, 20677, 19869, 19026,
18757, 18308, 14458, 13685, 12942, 12323, 11033, 10183, 10628, 10803, 10655, 104
82, 10202, 10166, 9939, 10138, 10007, 10174, 9997, 9465, 9028, 8806, 8450, 7941,
 7253, 6698, 6267, 5773),
   (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 23, 91, 412, 1319, 2984, 581
6, 10053, 16045, 24240, 35066, 47828, 62384, 78916, 97822, 112738, 124414, 13065
```

```
8, 140789, 153951, 168560, 179996, 194471, 212006, 225209, 228886, 239690, 24597
4, 253459, 255455, 260715, 259980, 256481, 252222, 249467, 240268, 238465, 23816
7, 231361, 223832, 220459, 222512, 220099, 219301, 221322, 229783, 239336, 25836
0, 271151, 218063, 213461, 207617, 196227, 174615, 160855, 165410, 163070, 15737
9, 149698, 140570, 131785, 119936, 113751, 106989, 99294, 89097, 78413, 68174, 6
0592, 52189, 43375, 35469, 29648, 24575, 20863),
   (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 74, 392, 1351, 3906, 7847, 1
2857, 19913, 29108, 42475, 58287, 74163, 90724, 108375, 125886, 141559, 148061,
152871, 159725, 171298, 183536, 196136, 210831, 228757, 238731, 239616, 250036,
251759, 259593, 261832, 264864, 264702, 264070, 258117, 253678, 245440, 241342,
239843, 232493, 226118, 221644, 223440, 219833, 219659, 221271, 227123, 232865,
250646, 261796, 210136, 201824, 193109, 181831, 159280, 145235, 145929, 140266,
133082, 124350, 114441, 104655, 93223, 85899, 78800, 72081, 62645, 53214, 44086,
 38481, 32219, 26867, 21443, 16899, 13680, 11508),
   (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 7, 5, 17, 15, 31, 34, 38,
35, 45, 299, 295, 218, 247, 252, 254, 222, 307, 316, 385, 416, 463, 557, 670, 83
0, 889, 1025, 1149, 1356, 1488, 1835, 1929, 2130, 2362, 2494, 2884, 3160, 3487,
3916, 4196, 4619, 5032, 5709, 6347, 7288, 8139, 9344, 11002, 12809, 11504,
11918, 12927, 13642, 13298, 14015, 15751, 17445, 18591, 19682, 20969, 21629, 225
49, 23619, 25288, 26293, 27038, 27039, 27070, 27750, 27244, 25905, 24357, 22561,
 21794, 20595),
   (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 6, 8, 0, 8, 21, 34, 49, 84,
97, 368, 401, 414, 557, 654, 631, 689, 698, 858, 1031, 1120, 1263, 1614, 1882, 2
137, 2516, 2923, 3132, 3741, 4259, 4930, 5320, 5948, 6548, 7463, 8309, 9142, 103
21, 11167, 12062, 13317, 15238, 16706, 18236, 20336, 23407, 27024, 32502, 37334,
 34454, 38080, 41811, 44490, 45247, 46830, 53616, 58798, 63224, 66841, 71086, 73
654, 77334, 82062, 87314, 92207, 94603, 94113, 92753, 93174, 91812, 87757,
84255, 79723, 77536, 74173),
   (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 5, 0, 11, 35, 137,
331, 803, 1580, 2361, 3632, 4866, 6849, 8754, 10422, 12316, 14152, 16911, 19788,
 22822, 27329, 31547, 35711, 38932, 42956, 46466, 49983, 52885, 55178, 56549, 57
632, 57770, 57427, 56348, 55593, 55554, 53266, 51084, 49342, 48555, 47067,
45789, 44988, 44624, 44238, 46267, 46203, 36964, 33866, 31701, 28770, 25174, 227
02, 21934, 20638, 19051, 17073, 15381, 13736, 11690, 10368, 9350, 8375, 7063, 60
06, 5044, 4030, 3420, 2612, 2006, 1709, 1264, 1018),
   (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 6, 11, 20, 68, 179, 480,
1077, 2094, 3581, 5151, 7047, 9590, 12434, 15039, 17257, 19098, 21324, 24453, 27
813, 32316, 37281, 43597, 49647, 53559, 58888, 62375, 67219, 70956, 73547,
74904, 75994, 76224, 74979, 72064, 70330, 68944, 66527, 63073, 60899, 60968, 587
56, 57647, 56301, 57246, 57068, 59027, 59187, 47549, 44425, 40976, 38077, 32904,
 29431, 29491, 28020, 26086, 24069, 21742, 19498, 17400, 15738, 14451, 13107, 11
568, 10171, 8530, 7273, 6488, 5372, 4499, 3691, 3259, 2657)]

types = ['Males single', 'Females single',
         'Males married', 'Females married',
         'Males widowed', 'Females widowed',
         'Males divorced', 'Females divorced']

pyramid_chart = pygal.Pyramid(human_readable=True, legend_at_bottom=True)
pyramid_chart.title = 'England population by age in 2010 (source: ons.gov.uk)'
pyramid_chart.x_labels = map(lambda x: str(x) if not x % 5 else '', range(90))
for type, age in zip(types, ages):
    pyramid_chart.add(type, age)
galplot(pyramid_chart)
```



NetworkX

- https://networkx.github.io/ (https://networkx.github.io/)
- a Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks
- not primarily a graph drawing package but basic drawing with Matplotlib

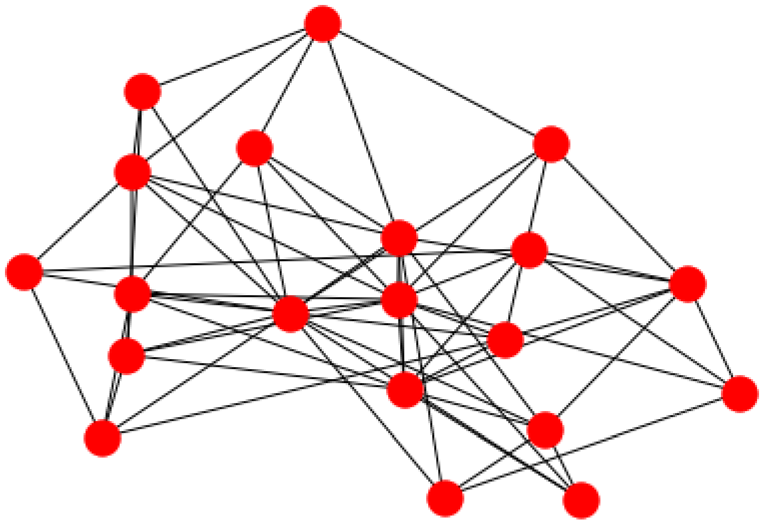
First steps

In [53]:

```
import networkx as nx
```

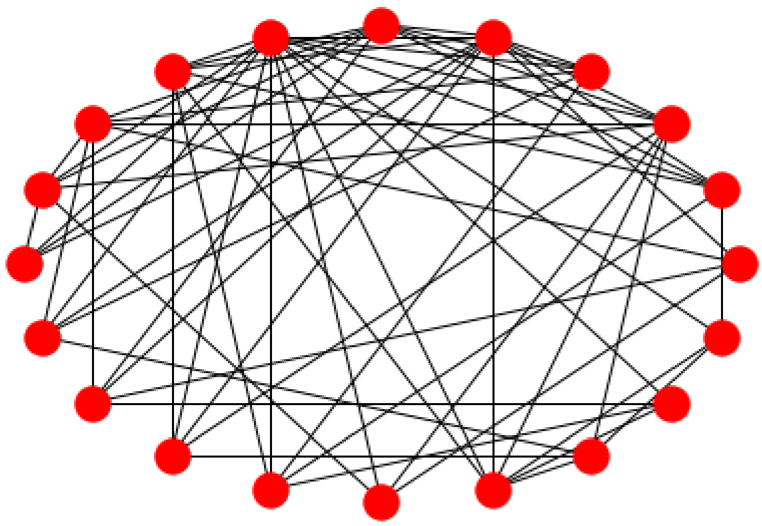
In [20]:

```
G = nx.barabasi_albert_graph(20,4)
nx.draw(G)
```



In [21]:

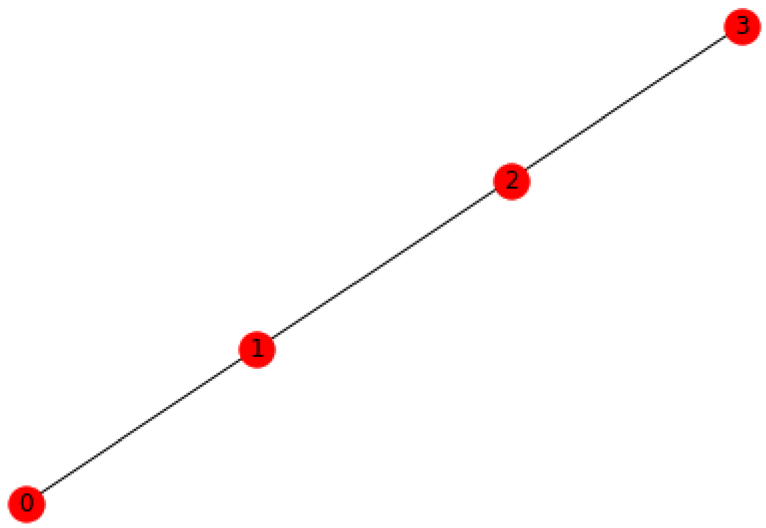
```
nx.draw_circular(G)
```



Node labels

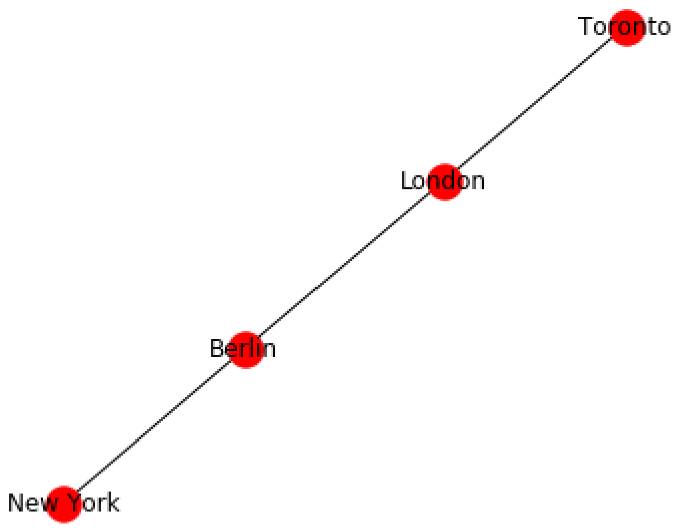
In [22]:

```
G=nx.path_graph(4)
nx.draw(G,with_labels=True)
```



In [23]:

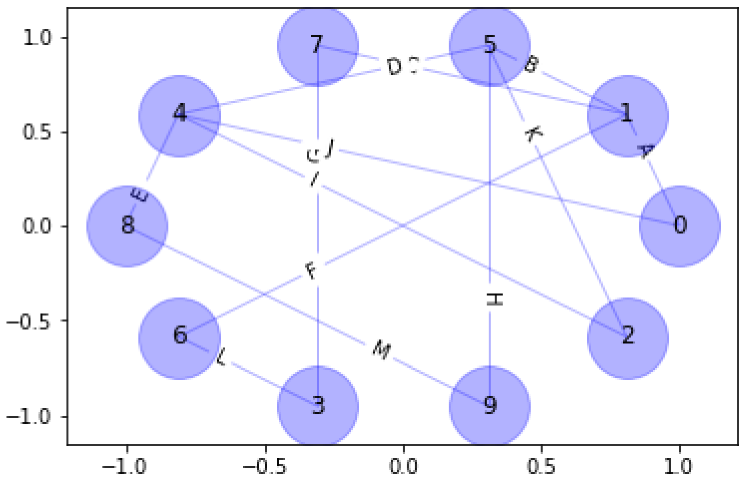
```
cities = {0:"Toronto",1:"London",2:"Berlin",3:"New York"}
H=nx.relabel_nodes(G,cities)
nx.draw(H,with_labels=True)
```



Style

In [24]:

```
def draw_graph(graph, labels=None, graph_layout='shell',
               node_size=1600, node_color='blue', node_alpha=0.3,
               node_text_size=12,
               edge_color='blue', edge_alpha=0.3, edge_tickness=1,
               edge_text_pos=0.3,
               text_font='sans-serif'):
    """ Draw graph given as list of edges"""

    # create networkx graph
    G=nx.Graph()

    # add edges
    for edge in graph:
        G.add_edge(edge[0], edge[1])

    # these are different layouts for the network you may try
    # shell seems to work best
    if graph_layout == 'spring':
        graph_pos=nx.spring_layout(G)
    elif graph_layout == 'spectral':
        graph_pos=nx.spectral_layout(G)
    elif graph_layout == 'random':
        graph_pos=nx.random_layout(G)
    else:
        graph_pos=nx.shell_layout(G)

    # draw graph
    nx.draw_networkx_nodes(G,graph_pos,node_size=node_size,
                           alpha=node_alpha, node_color=node_color)
    nx.draw_networkx_edges(G,graph_pos,width=edge_tickness,
                           alpha=edge_alpha,edge_color=edge_color)
    nx.draw_networkx_labels(G, graph_pos,font_size=node_text_size,
                            font_family=text_font)

    if labels is None:
        labels = range(len(graph))

    edge_labels = dict(zip(graph, labels))
    nx.draw_networkx_edge_labels(G, graph_pos, edge_labels=edge_labels,
                                 label_pos=edge_text_pos)

    # show graph
    plt.show()

graph = [(0, 1), (1, 5), (1, 7), (4, 5), (4, 8), (1, 6), (3, 7), (5, 9),
         (2, 4), (0, 4), (2, 5), (3, 6), (8, 9)]

# you may name your edge labels
labels = map(chr, range(65, 65+len(graph)))
draw_graph(graph, labels)
```

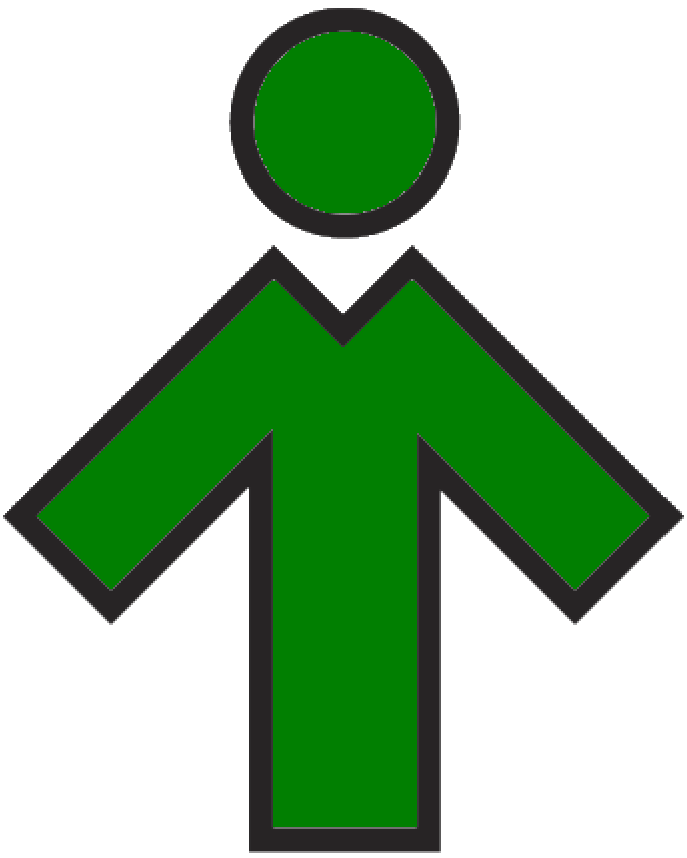


Custom node markers

In [25]:

```
from IPython.display import Image
Image("spinson_up_green.png")
```

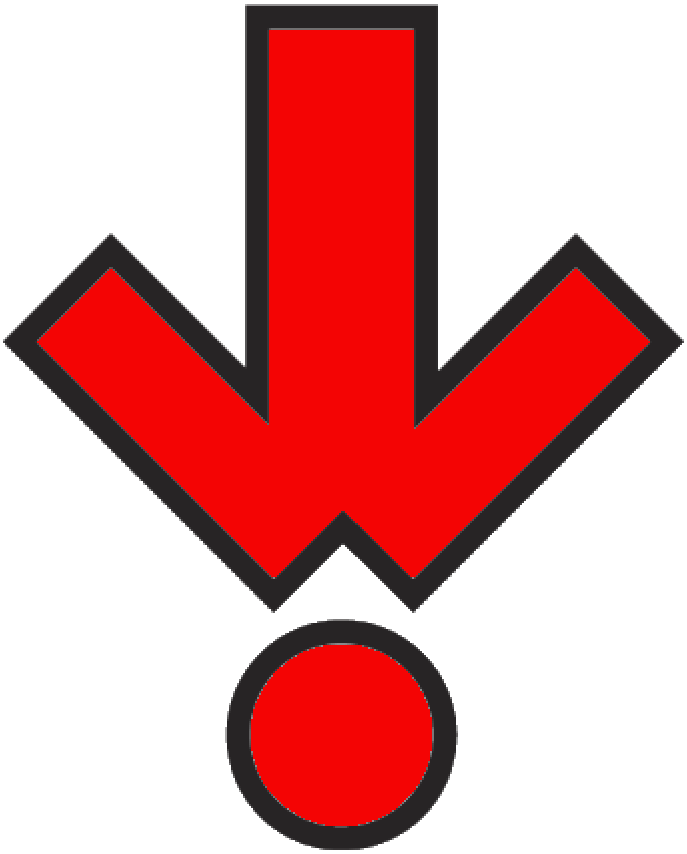
Out[25]:



In [26]:

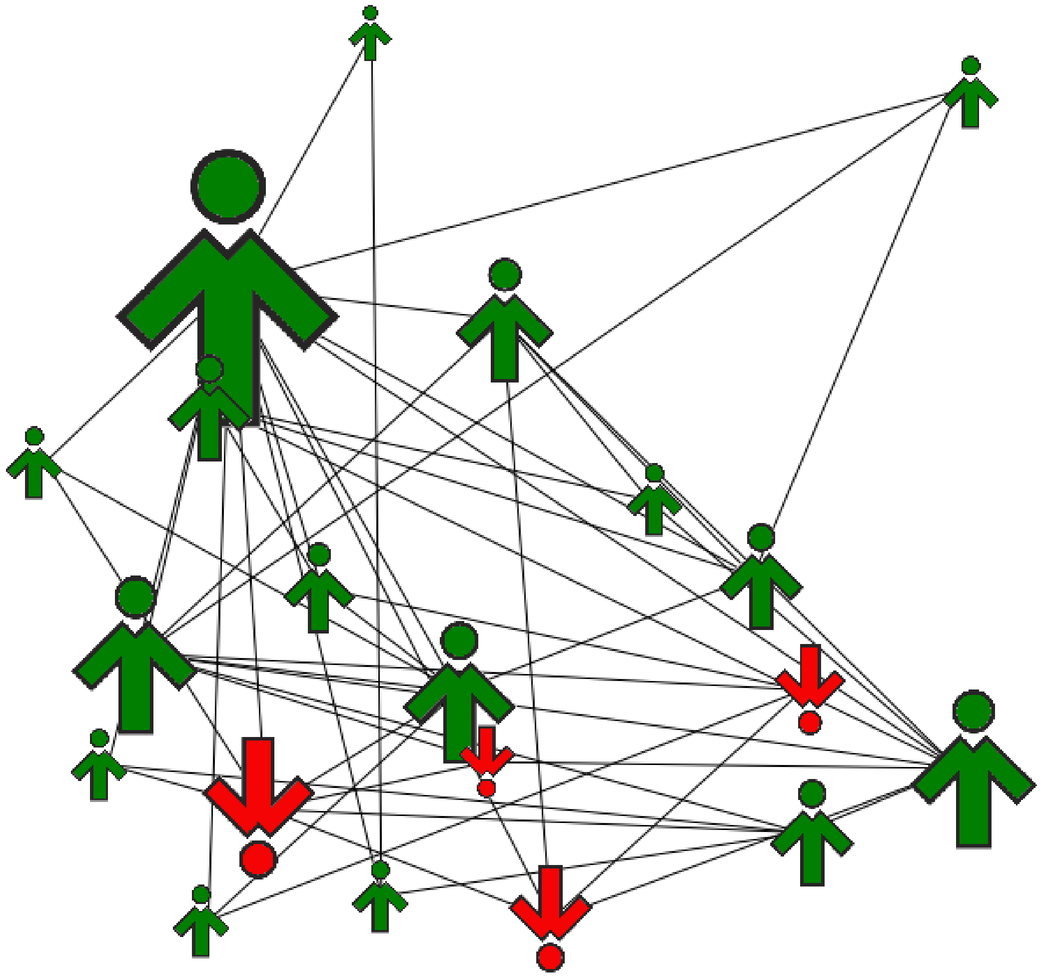
```
from IPython.display import Image
Image("spinson_down_red.png")
```

Out[26]:



In [27]:

```
import networkx as nx
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import random as rd

img1=mpimg.imread('spinson_up_green.png')
img2=mpimg.imread('spinson_down_red.png')

G=nx.barabasi_albert_graph(20,3)

imsize = 0.02 # this is the basic image size
imscal = 0.8

for nd in G.nodes():
    if rd.random() < 0.7:
        G.node[nd]['image'] = img1
        G.node[nd]['size'] = imsize*(1+imscal*G.degree(nd))
    else:
        G.node[nd]['image'] = img2
        G.node[nd]['size'] = imsize*(1+imscal*G.degree(nd))

#pos=nx.spring_layout(G)
pos=nx.random_layout(G)

fig=plt.figure(figsize=(10,10))
ax=plt.subplot(111)
ax.set_aspect('equal')
ax.axis('off')
nx.draw_networkx_edges(G,pos,ax=ax)

plt.xlim(-0,1)
plt.ylim(-0,1)

trans=ax.transData.transform
trans2=fig.transFigure.inverted().transform

piesize=0.05 # this is the image size
p2=piesize/2.0
for n in G:
   xx,yy=trans(pos[n]) # figure coordinates
   xa,ya=trans2((xx,yy)) # axes coordinates
   imsize=G.node[n]['size']
   i2 = imsize/2.0
   a = plt.axes([xa-i2,ya-i2, imsize, imsize])
   a.set_aspect('equal')
   a.imshow(G.node[n]['image'])
   a.axis('off')

plt.show()
```



Graphviz

- http://www.graphviz.org/ (http://www.graphviz.org/)
- open source graph visualization software
- descriptions of graphs in a simple text language (dot language)
- may be used together with NetworkX

First steps

In [54]:

```
import graphviz as gv
g1 = gv.Graph(format='svg')
g1.node('A')
g1.node('B')
g1.edge('A', 'B')
```

The `DOT` representation of the graph is stored in the `source` attribute:

In [55]:

```
print(g1.source)
```

```
graph {
	A
	B
		A -- B
}
```

Graph may be saved:

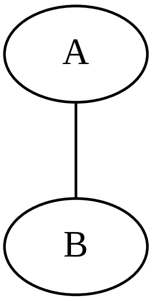
In [56]:

```
filename = g1.render('g1')
print(filename)
```

```
g1.svg
```

In [57]:

```
from IPython.display import SVG, display

display(SVG(filename))
```

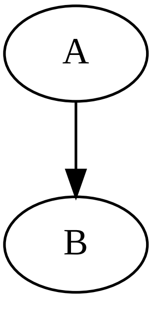


Directed graphs

In [58]:

```
g2 = gv.Digraph()
g2.node('A')
g2.node('B')
g2.edge('A', 'B')
g2
```

Out[58]:



Useful helpers

In [59]:

```
import functools
graph = functools.partial(gv.Graph, format='svg')
digraph = functools.partial(gv.Digraph, format='svg')
```

The creation of graphs is much easier now:

In [60]:

```
g3 = graph()
```

Let us write some custom functions for adding nodes and edges from lists:

In [61]:

```
def add_nodes(graph, nodes):
    for n in nodes:
        if isinstance(n, tuple):
            graph.node(n[0], **n[1])
        else:
            graph.node(n)
    return graph

def add_edges(graph, edges):
    for e in edges:
        if isinstance(e[0], tuple):
            graph.edge(*e[0], **e[1])
        else:
            graph.edge(*e)
    return graph
```

Simple check:

In [62]:

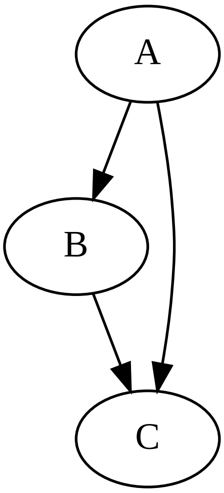
```
add_edges(add_nodes(digraph(), ['A', 'B', 'C']), [('A', 'B'), ('A', 'C'), ('B',
'C')]).render('g4')
```

Out[62]:

```
'g4.svg'
```

In [63]:

```
display(SVG('g4.svg'))
```



Labels

In [64]:

```
add_edges(
    add_nodes(digraph(), [
        ('A', {'label': 'Node A'}),
        ('B', {'label': 'Node B'}),
        'C'
    ]),
    [
        (('A', 'B'), {'label': 'Edge 1'}),
        (('A', 'C'), {'label': 'Edge 2'}),
        ('B', 'C')
    ]
).render('g5')
```

Out[64]:

```
'g5.svg'
```