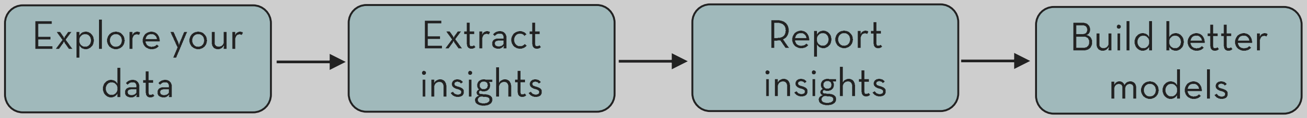
seaborn basics

Data visualization (in Python)

recap

- Data visualization is an essential part in the data analysis process:



Explore your data → Extract insights → Report insights → Build better models

- Python's plotting modules and packages enable customized graphs and more:

Module/Package	Logo	Description
Matplotlib	matplotlib	- Open Source plotting library - Interactive plotting - Syntax fimiliar to Matlab
Ipython	IP[y]: IPython Interactive Computing	- "pylab" mode: designed for interactive plotting with matplotlib
Plotly	plotly	- Collaborative browser-based plotting and analytics platform
ggplot	ggplot	- Based on R‘s ggplot2 - „Grammar of Graphics“: build your plot from various layers
Seaborn		- Visualization library based on matplotlib with simple functions - Provides good default values and integration with Pandas

Source: https://wiki.python.org/moin/NumericAndScientific/Plotting

How to plot
Steps

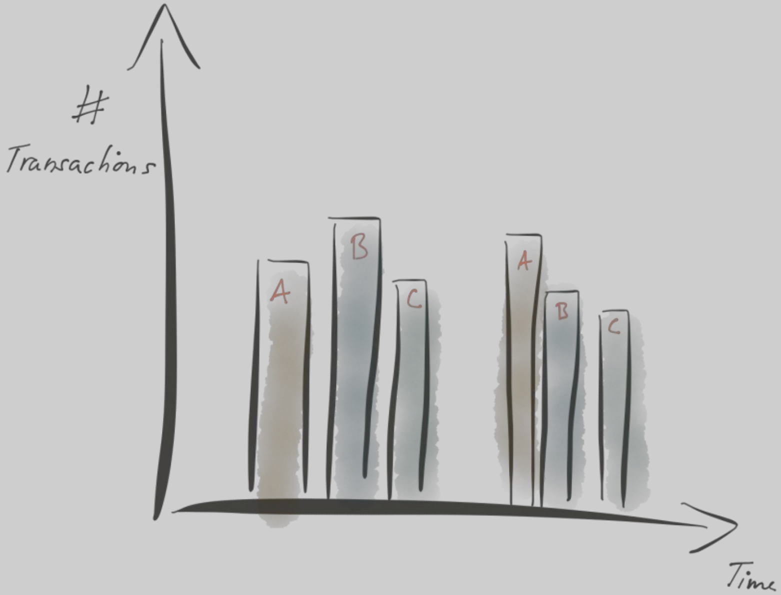
1. Choose a plot type.
2. Find the Python function.
3. Transforming data.
4. Create the plot.
5. Improve aesthetic features of the plot
6. Save plot.

same as last unit

Step 1: Choose the plot type
Decide the best way to convey the information

- What do you want to show?
  - A single variable.
  - The relationship between multiple variables.
- Are your data continuous or discrete?

same as last unit



Why use seaborn instead of Matplotlib with pyplot

Why use seaborn instead of Matplotlib with pyplot?

`Seaborn` provides a convenient API on top of `matplotlib` enabling nice plots with simple commands:

Disadvantages Matplotlib (pyplot)	Seaborn Functionality
Defaults are not the best choices (based on MatLab in 1999).	Sane *plot and color defaults.*
Relatively low-level and customization of plots require a lot of code.	*Simple functions* for statistical plot types.
Not designed for use with pandas dataframes: Extraction and Concatenation of series to the right format is often necessary.	Integration with functionality provided by *Pandas dataframes.*

Remember: `Seaborn` is based on the `Matplotlib` library and plots with its functionality.

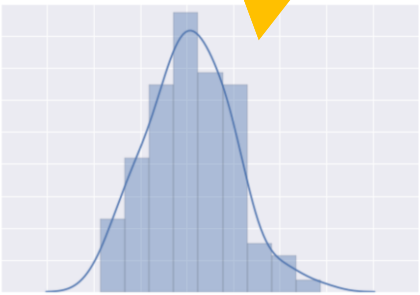
Use `import seaborn as sbn` to load the package.

https://www.oreilly.com/learning/data-visualization-with-seaborn

Step 2: Find the function
Plotting a single variable

Continuous

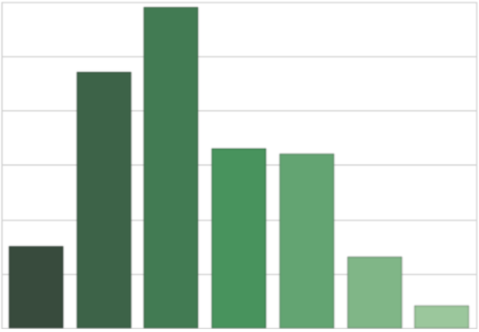
Kernel density estimator and histogram.



`sbn.distplot(x)`

Note that we imported `seaborn` as **`sbn`**.

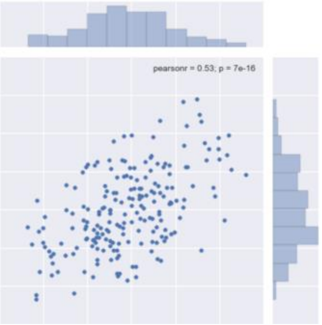
Discrete



`sbn.distplot(x)`

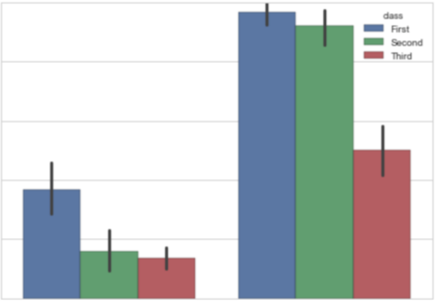
Step 2: Find the function
Plotting two variables

Continuous Continuous

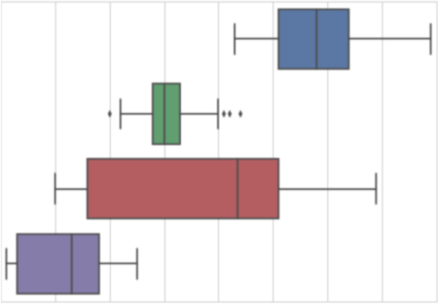


`sbn.jointplot(x,y)`

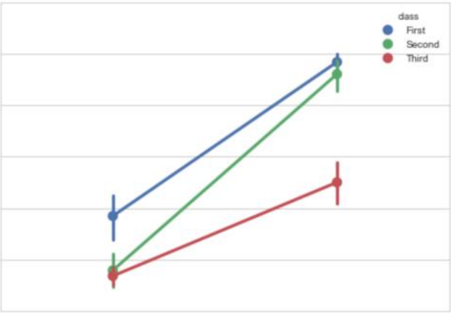
Continuous Discrete



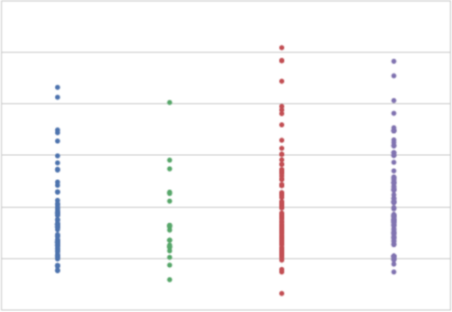
`sbn.barplot()`



`sbn.boxplot()`



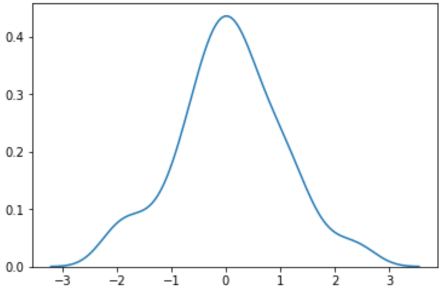
`sbn.pointplot()`



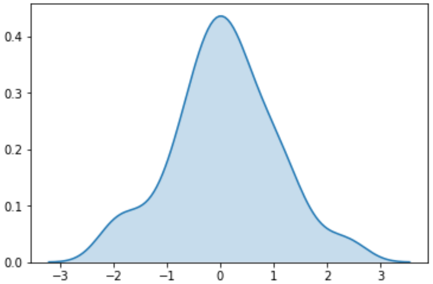
`sbn.stripplot()`

Step 2: Find the function
Functions and maps

Functions

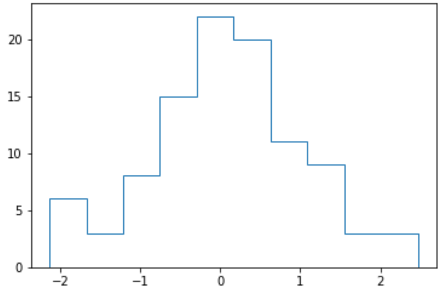


```
sbn.distplot(x,
 hist=False)
```



```
sbn.distplot(x, hist=False,
 kde_kws={"shade":True}))
```

Specify the options of the Gaussian kernel density estimate.



```
sbn.distplot(x,
     sbn.distplot(d,
     hist=False,
     kde_kws={"shade":True,
            "alpha":1}))
```

Apply a Gaussian kernel density estimate.

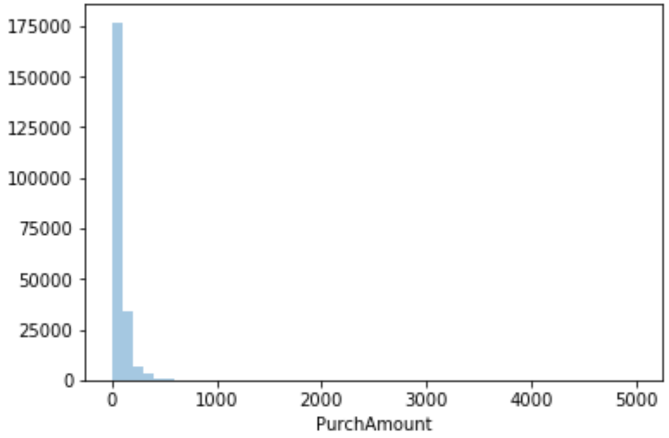
Step 3: Transforming data
Some graphs might require transformed data input

- It is quite rare that you get data that is ready to go for plots or calculations.
- In most cases it is necessary to transform your data before plotting it.
- For example:
  - Transform times and dates (Lecture 2) for aggregation (Lecture 6) of month or years.
  - Group data for better overview.
  - Logarithmic transformations for nicer distributions.

same as last unit

Step 4: Create the plot
Example 1: Create a histogram

Customer	TransDate	Quantity	PurchAmount	Cost	TransID
149332	15/11/05	1	199.95	107.00	127998739
172951	29/08/08	1	199.95	108.00	128888288
120621	19/10/07	1	99.95	49.00	125375247
149236	14/11/05	1	39.95	18.95	127996226
149236	12/06/07	1	79.95	35.00	128670302
...	...	...	...	...	...

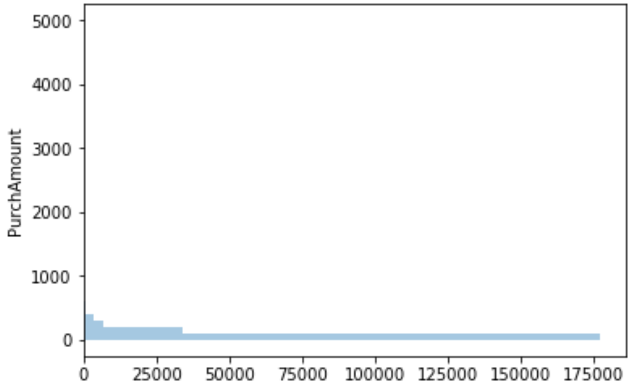


```
sbn.distplot(myData["PurchAmount"], kde=False)
plt.show()
```

Only Histogram.

Step 4: Create the plot
Example 1: Flip the coordinates of the histogram

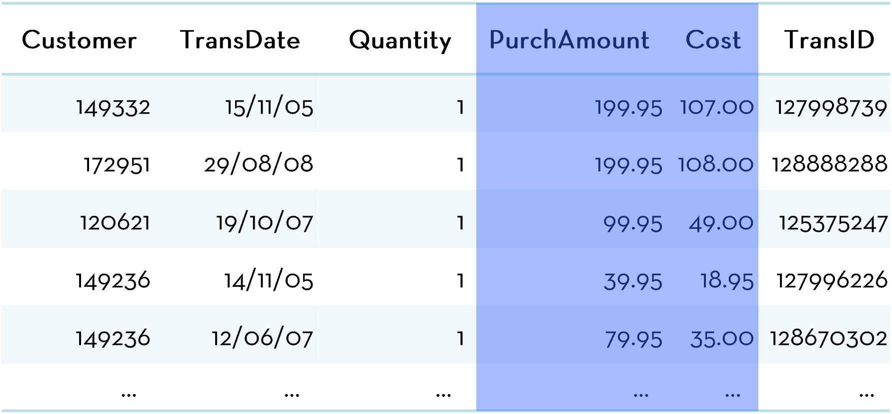
Customer	TransDate	Quantity	PurchAmount	Cost	TransID
149332	15/11/05	1	199.95	107.00	127998739
172951	29/08/08	1	199.95	108.00	128888288
120621	19/10/07	1	99.95	49.00	125375247
149236	14/11/05	1	39.95	18.95	127996226
149236	12/06/07	1	79.95	35.00	128670302
...	...	...	...	...	...



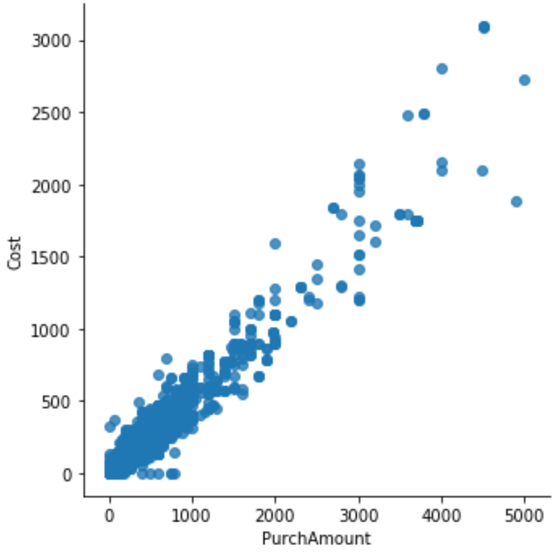
```
sbn.distplot(myData["PurchAmount"], kde=False,
vertical=True)
plt.show()
```

Flips coordinates.

Step 4: Create the plot
Example 2: Create a scatterplot



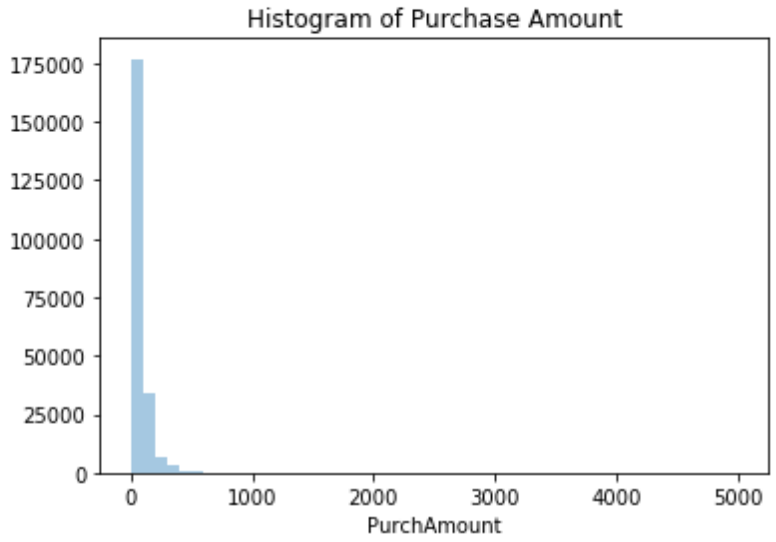
Customer	TransDate	Quantity	PurchAmount	Cost	TransID
149332	15/11/05	1	199.95	107.00	127998739
172951	29/08/08	1	199.95	108.00	128888288
120621	19/10/07	1	99.95	49.00	125375247
149236	14/11/05	1	39.95	18.95	127996226
149236	12/06/07	1	79.95	35.00	128670302
...	...	...	...	...	...

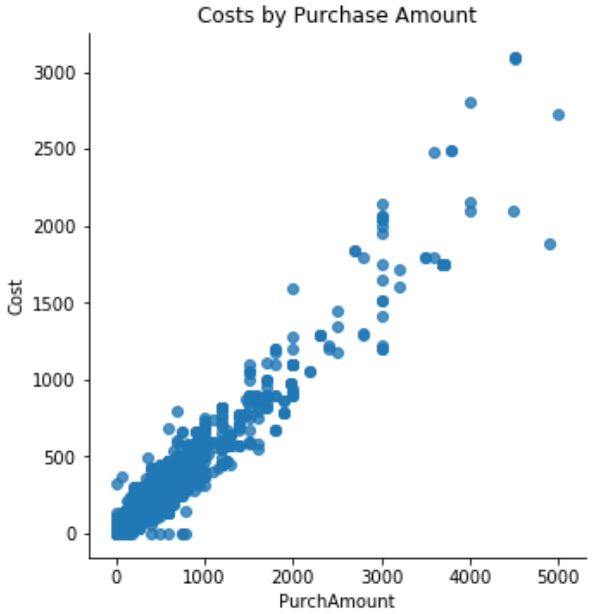


Scatterplot

```
sbn.lmplot(x="PurchAmount", y="Cost", data=myData, fit_reg=False)
plt.show()
```

Step 5: Fine tune the plot
Layer on the title



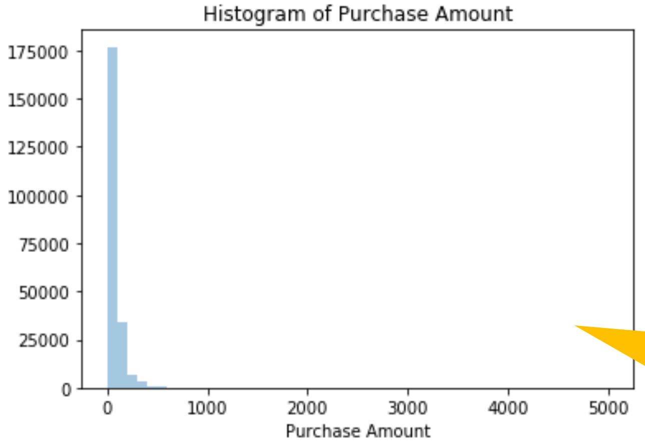


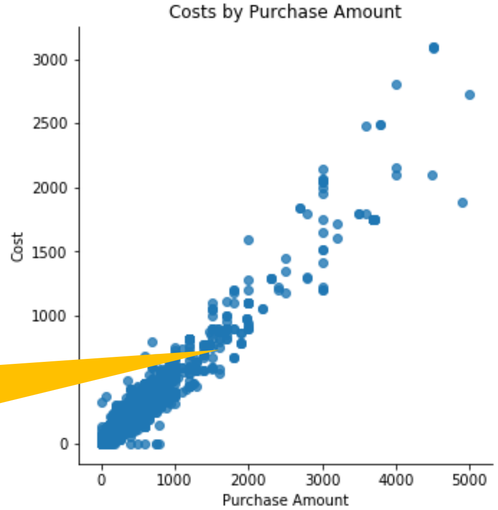
```
sbn.distplot(myData["PurchAmount"],
kde=False)
plt.title("Histogram of Purchase
Amount")
plt.show()
```

Add title.

```
sbn.lmplot(x="PurchAmount",
y="Cost", data=myData,
fit_reg=False)
plt.title("Costs by Purchase
Amount")
plt.show()
```

Step 5: Fine tune the plot
Layer on the axis labels





Notice the similarity and differences between the 2 plots.

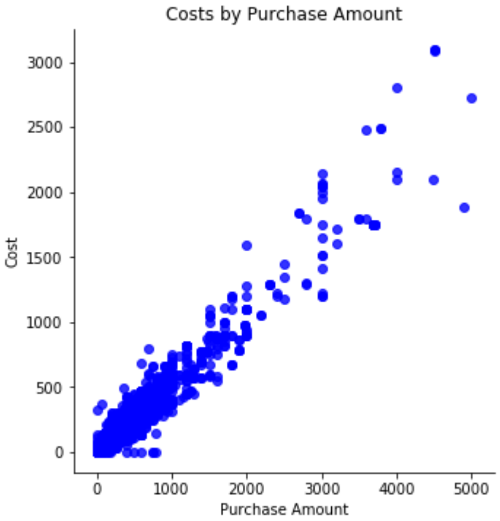
```
sbn.distplot(myData["PurchAmount"],
             kde=False)
 plt.title("Histogram of Purchase
             Amount")
  plt.xlabel("Purchase Amount")
             plt.show()
```

Add x axis label.

```
sbn.lmplot(x="PurchAmount",
   y="Cost", data=myData,
        fit_reg=False)
plt.title("Costs by Purchase
          Amount")
plt.xlabel("Purchase Amount")
     plt.ylabel("Cost")
          plt.show()
```

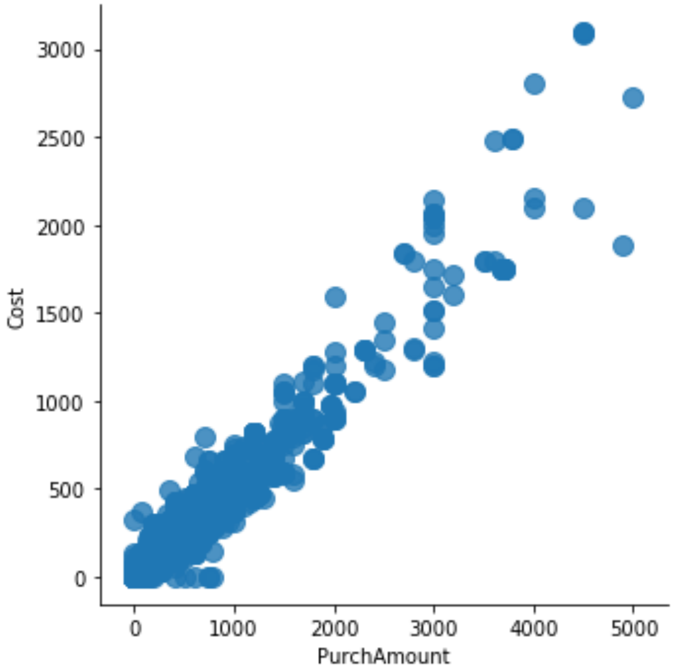
Add axes labels.

Step 5: Fine tune the plot
Change point color



```
sbn.lmplot(x="PurchAmount",
          y="Cost",
scatter_kws={“color“:“blue“},
 data=myData, fit_reg=False)
              ...
           plt.show()
```

Step 5: Fine tune the plot
Change point size



```
sbn.lmplot(x="PurchAmount", y="Cost", data=myData,
        fit_reg=False, scatter_kws={"s": 100})
                    plt.show()
```

`scatter_kws` are dictionaries which are passed to *`plt.plot()`* as additional keyword arguments.

Caution:
"markersize" is not an argument for `sbn.lmplot()`.

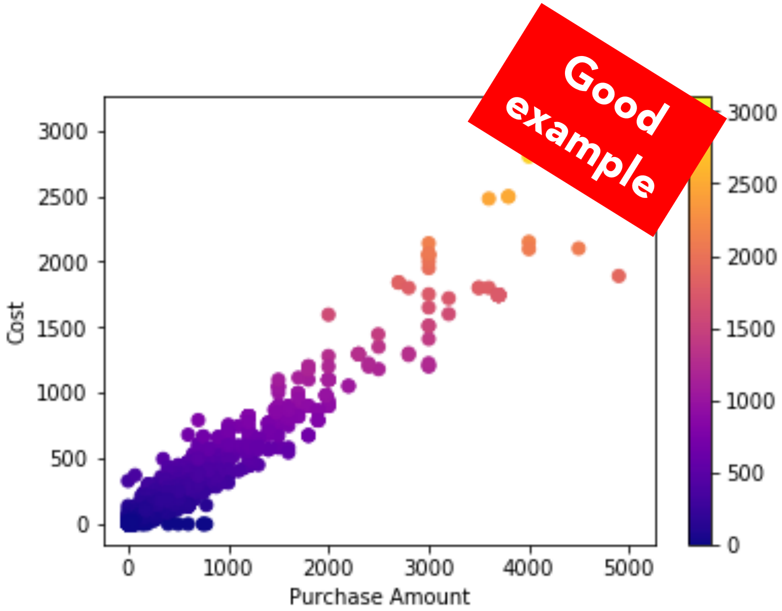
Exercise
seaborn basics

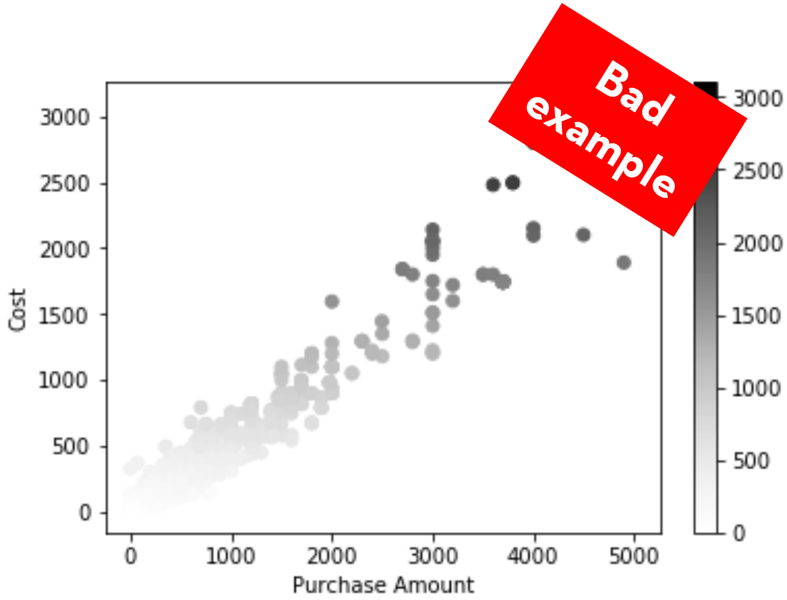
Hint:
Specify
„`fit_reg: True`“

1. Create a scatter plot with regression line of order two for the variables `PurchAmount` (x) and `Cost` (y) in seaborn.
2. Add a title to your plot, name your axes, change point size to “150” and point color to “green”.

Color palettes, themes, and style

Colors do matter!





```
points=plt.scatter(myData["PurchAmount"],
      myData["Cost"], c=myData["Cost"],
                 cmap="plasma")
            plt.colorbar(points)
        plt.xlabel("Purchase Amount")
              plt.ylabel("Cost")
                  plt.show()
```

Color depends on the value of variable **"Cost"**.

Add the colorbar legend to the plot.

Same as left hand-side with

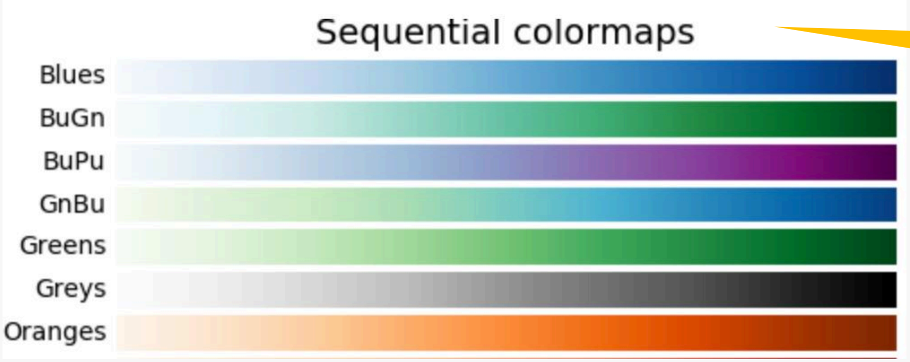
```
cmap= "greys"
```

Use the colormap "greys".

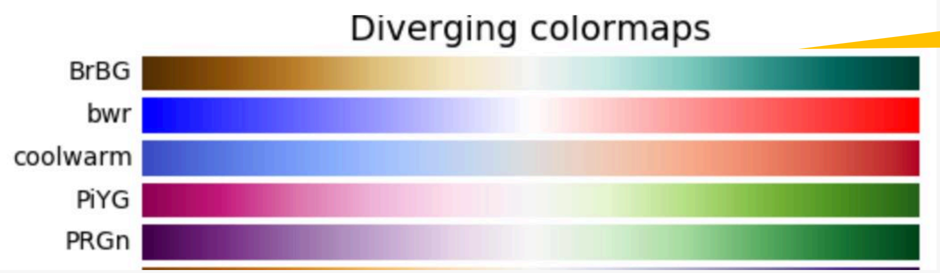
Using palettes saves time

- A **colormap** is a matched set of colors.
- Use palettes to coordinate plots:
  - within a projects.
  - across projects.

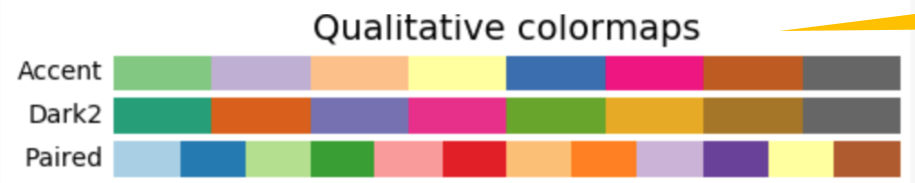
Sidenote: Overview over preexisting colormaps



Well suited to illustrate metric data.



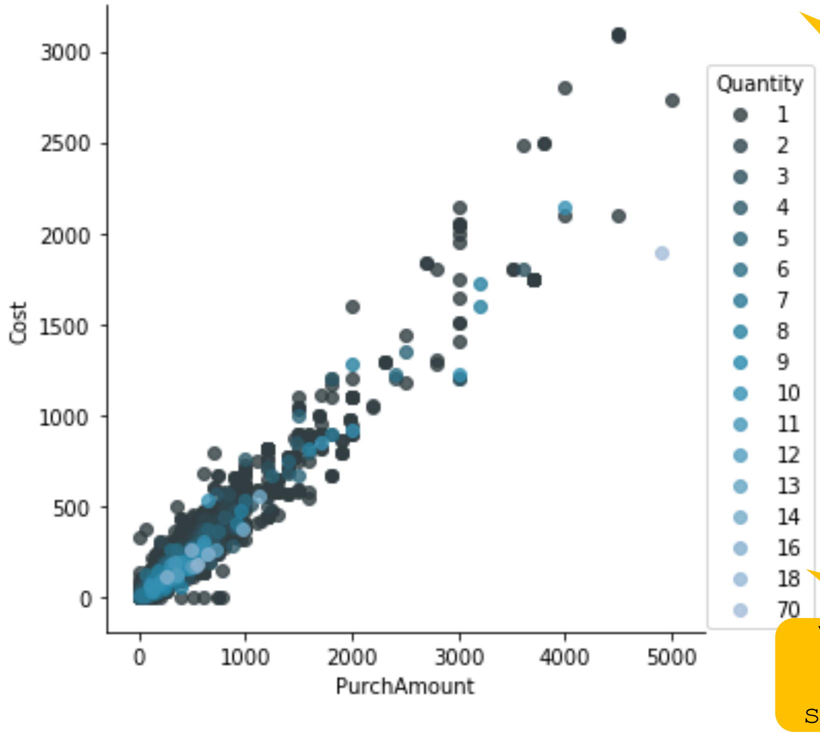
Illustrate contrasts.



Illustrate categorical data.

http://matplotlib.org/examples/color/colormaps_reference.html

Using color palettes to indicate a third dimension



Legend for the third dimension will be added automatically.

You can set a default color palette for all plots with `sbn.set_palette("Accent")`

`lmplot` the offers the parameter `hue` for factor levels.

```
sbn.lmplot("PurchAmount", "Cost", data=myData, fit_reg=False,
hue="Quantity", palette="PuBuGn_d")
plt.plot()
```

Specify color for different factor levels of a third dimension. In this case `hue = "Quantity"`

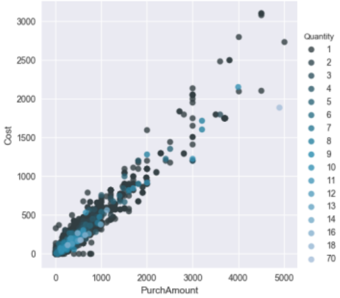
Use the “`PuBuGn_d`“ color palette .

In seaborn we can use pre-defined themes

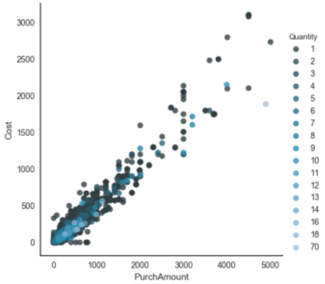
- Themes allow a consistent design. They can be shared within a company.
- They are easy to use and difficult to be “broken” and ”ruined” and save repetitive work.
- There are five seaborn themes:

```
sbn.set_style("dark")
```

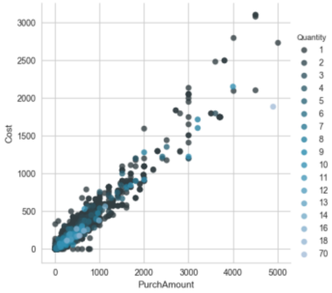
- darkgrid



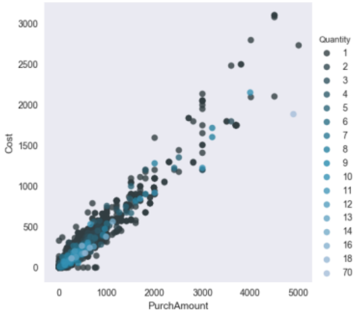
- white (default)



- whitegrid

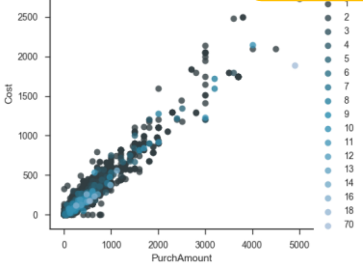


- dark



- ticks

Plots ticks at all boarders for certain functions.



You can adapt any theme to your requirements and personalize non-data elements of a plot

- At first, call the function which returns the current settings: **sbn.axes_style()**

- With **set_style()** you can then control themes and set non-data elements of a plot

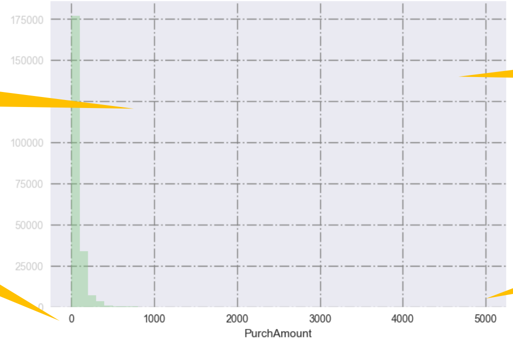
```
sbn.set_style("darkgrid", {"grid.color": "grey",
                           "grid.linestyle": u"-. ",
                           "ytick.color":".80" })
 sbn.distplot(myData["PurchAmount"], kde=False)
```

Personalize the grid.

Choose the theme “darkgrid”.

Choose colors of the y-axes labels.

- Further you can control:



Background color.

Grid lines.

Font.

Axis elements.

Exercise
Color palettes, themes, and style

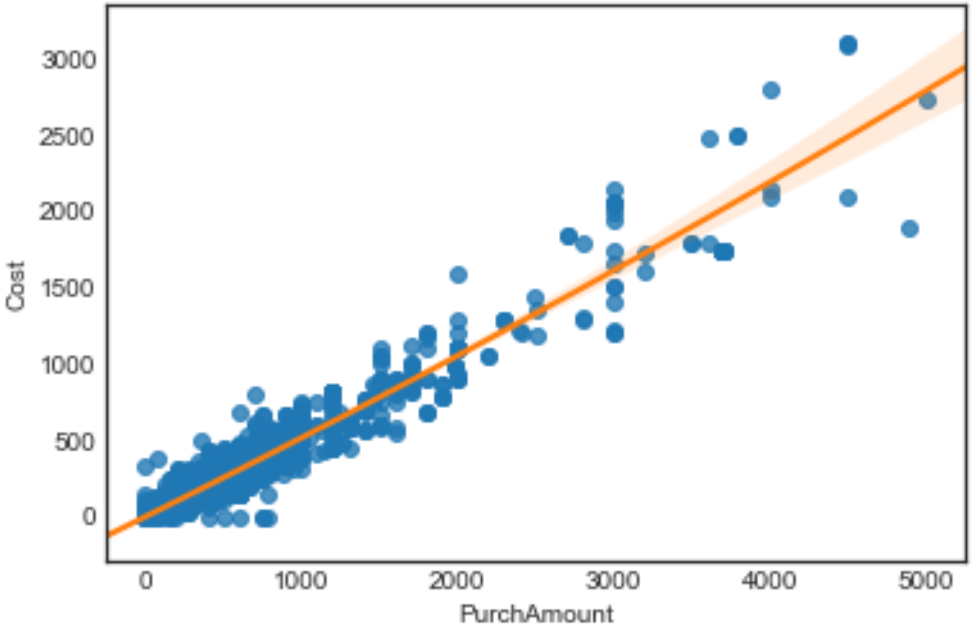
1. Apply theme `dark()` to your plot from Exercise 2.

2. Add the parameter ”`hue=Quantity`” and use the palette ”`prism`” to illustrate this dimension.

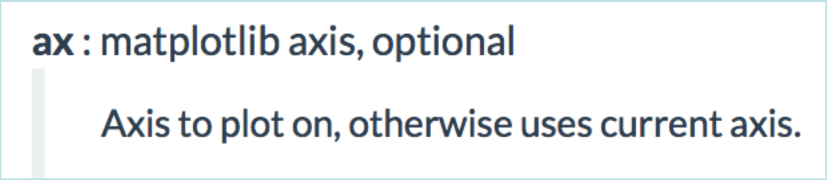
Caution: Make sure you use `lmplot()`.

Advanced plotting topics

Overlay plots on the same axes



Extract from the seaborn.kdeplot API.


ax : matplotlib axis, optional

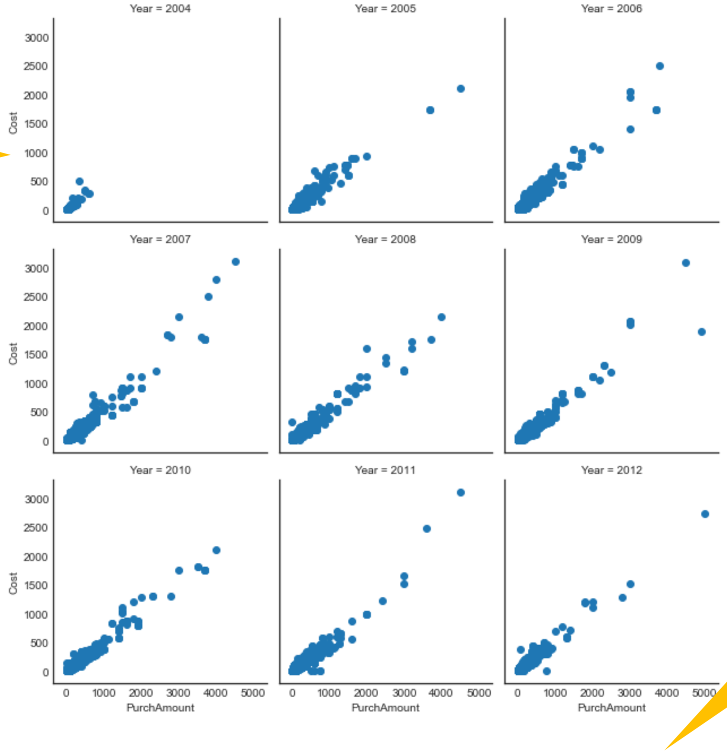
Axis to plot on, otherwise uses current axis.

```
               fig, ax = plt.subplots()
sbn.regplot(x="PurchAmount", y="Cost", data=myData,
                 fit_reg=False, ax=ax)
sbn.regplot(x="PurchAmount", y="Cost", data=myData,
              order=2, scatter=False, ax=ax)
```

Plot both plots on the same axes which was specified before.

Facets split up your data by one or more variables and plot the subsets of data together

Company started in December 2004, hence there are fewer transactions.



Initialize a FacetGrid object with a DataFrame (myData).

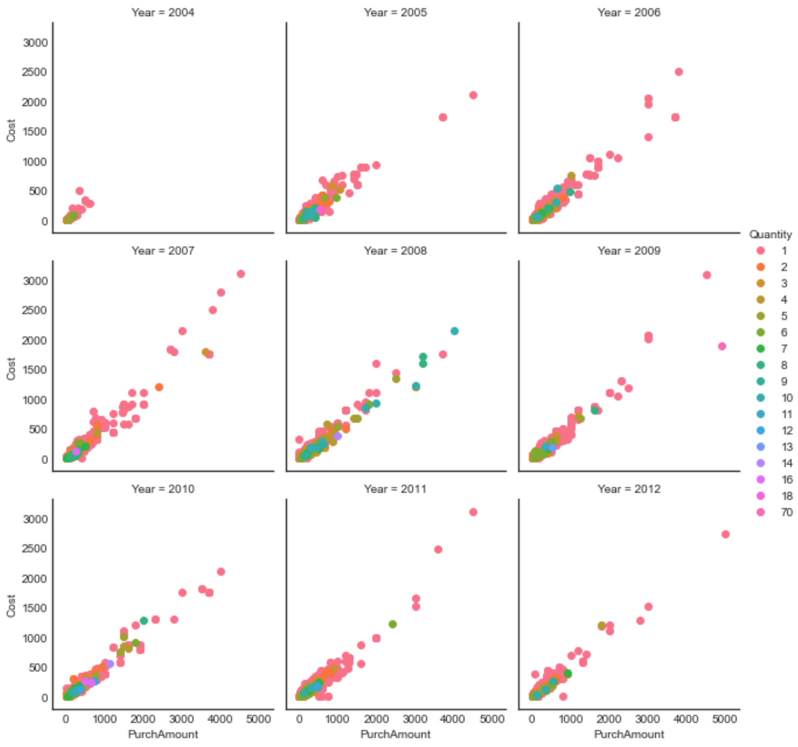
Variable that defines subsets of the data, which will be drawn on separate plots in the grid.

```
g = sbn.FacetGrid(myData, col="Year", col_wrap=3)
        g.map(plt.scatter, "PurchAmount", "Cost")
```

Visualize data on the grid with map() by providing a plotting function and variable names.

Define the number of columns of the grid.

With facets you can plot up to four dimensions in one single figure

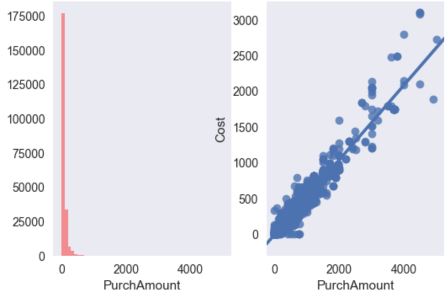


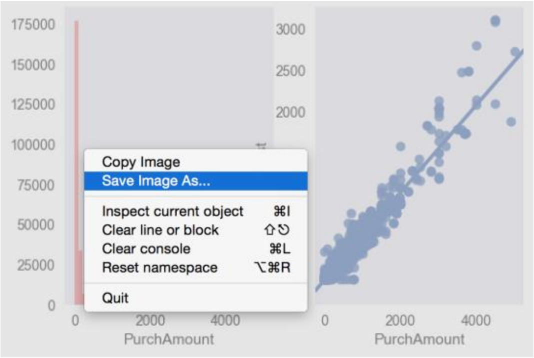
Add the fourth dimension by using the parameter `hue`.

```
g = sbn.FacetGrid(myData, col="Year", hue="Quantity“, col_wrap=3)
g.map(plt.scatter, "PurchAmount", "Cost")
g.add_legend();
```

Add the legend to the grid.

Save your plot with `savefig()` or via the point-and-click method

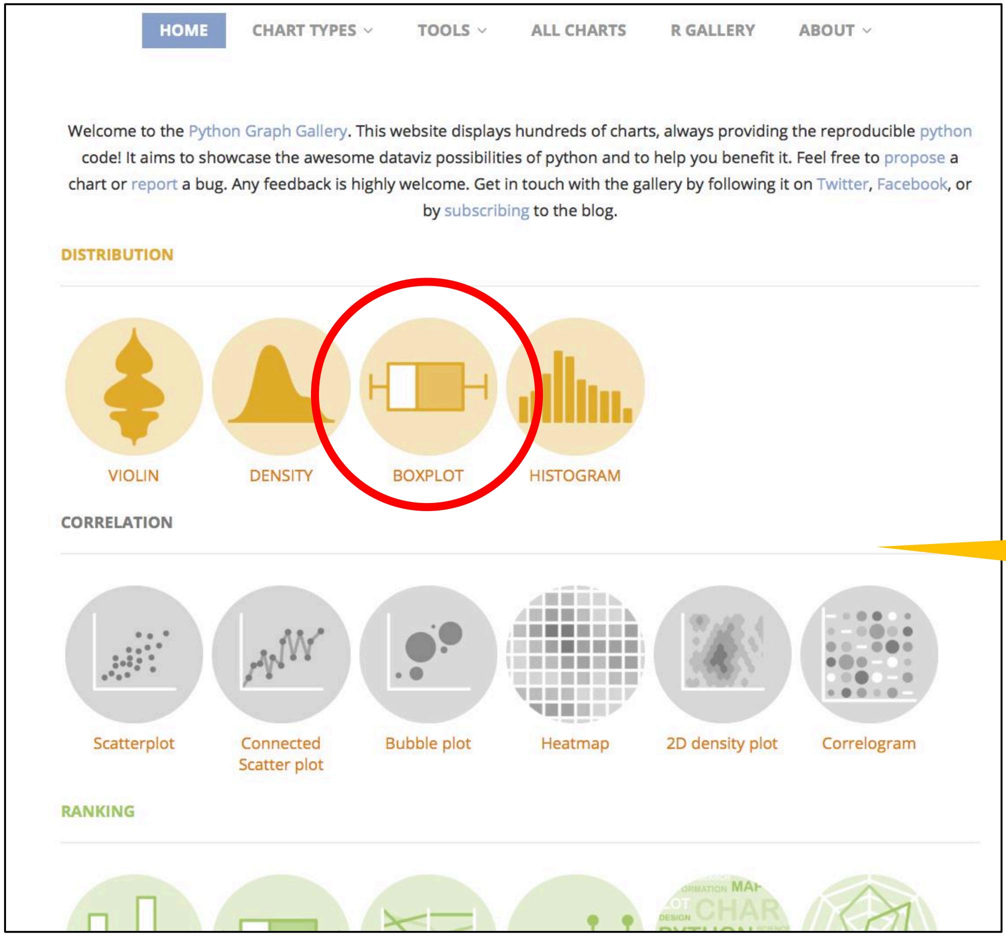




```
fig.savefig("myplot.png")
```

Saves last plot.

Visualize data with https://python-graph-gallery.com
Step 1: Explore functions

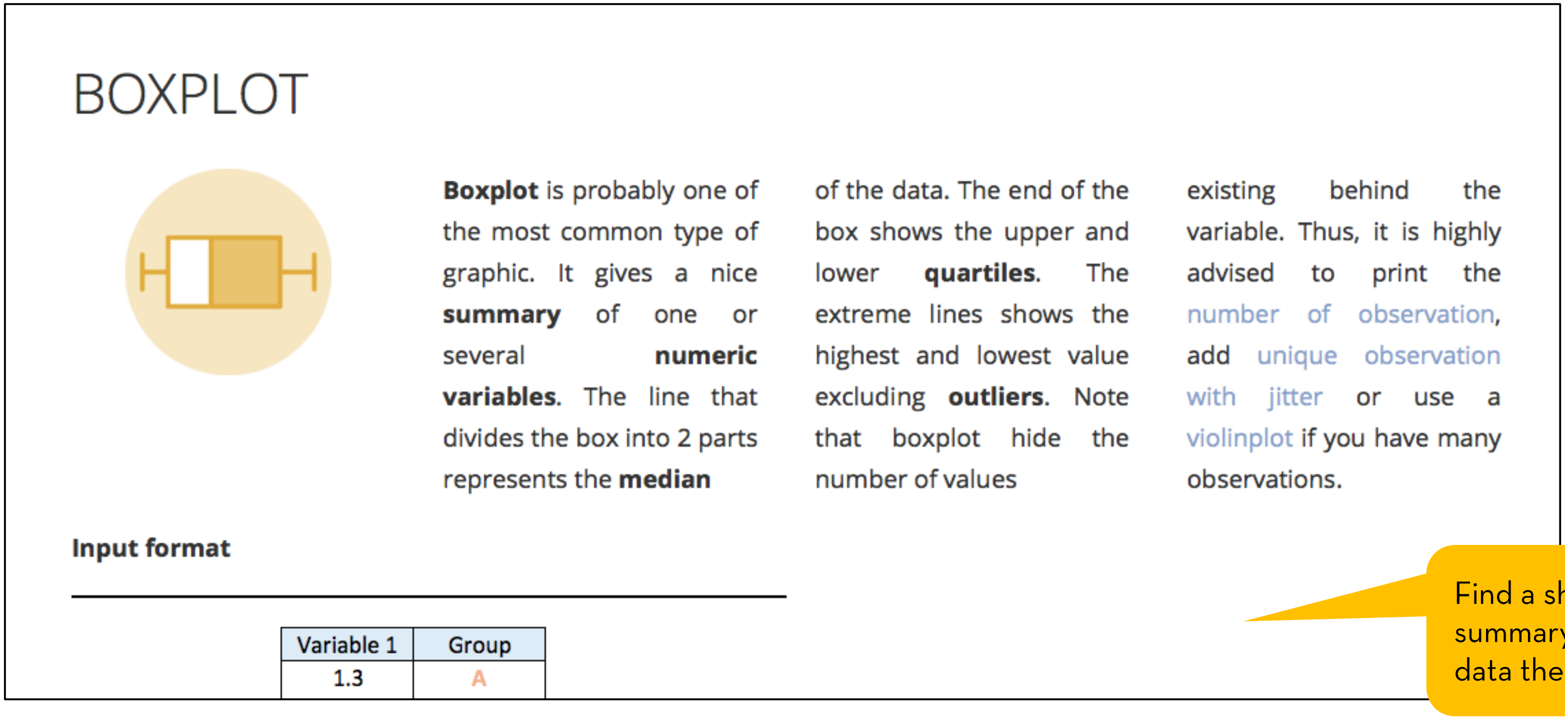


Welcome to the Python Graph Gallery. This website displays hundreds of charts, always providing the reproducible python code! It aims to showcase the awesome dataviz possibilities of python and to help you benefit it. Feel free to propose a chart or report a bug. Any feedback is highly welcome. Get in touch with the gallery by following it on Twitter, Facebook, or by subscribing to the blog.

Get inspired by the graphs on the ”home” page.

- Get a smart overview over possibilities to visualize data with reproducible Python code at https://python-graph-gallery.com

Visualize data with https://python-graph-gallery.com
Step 2: Understand your function



BOXPLOT

Boxplot is probably one of the most common type of graphic. It gives a nice **summary** of one or several **numeric variables**. The line that divides the box into 2 parts represents the **median** of the data. The end of the box shows the upper and lower **quartiles**. The extreme lines shows the highest and lowest value excluding **outliers**. Note that boxplot hide the number of values existing behind the variable. Thus, it is highly advised to print the number of observation, add unique observation with jitter or use a violinplot if you have many observations.

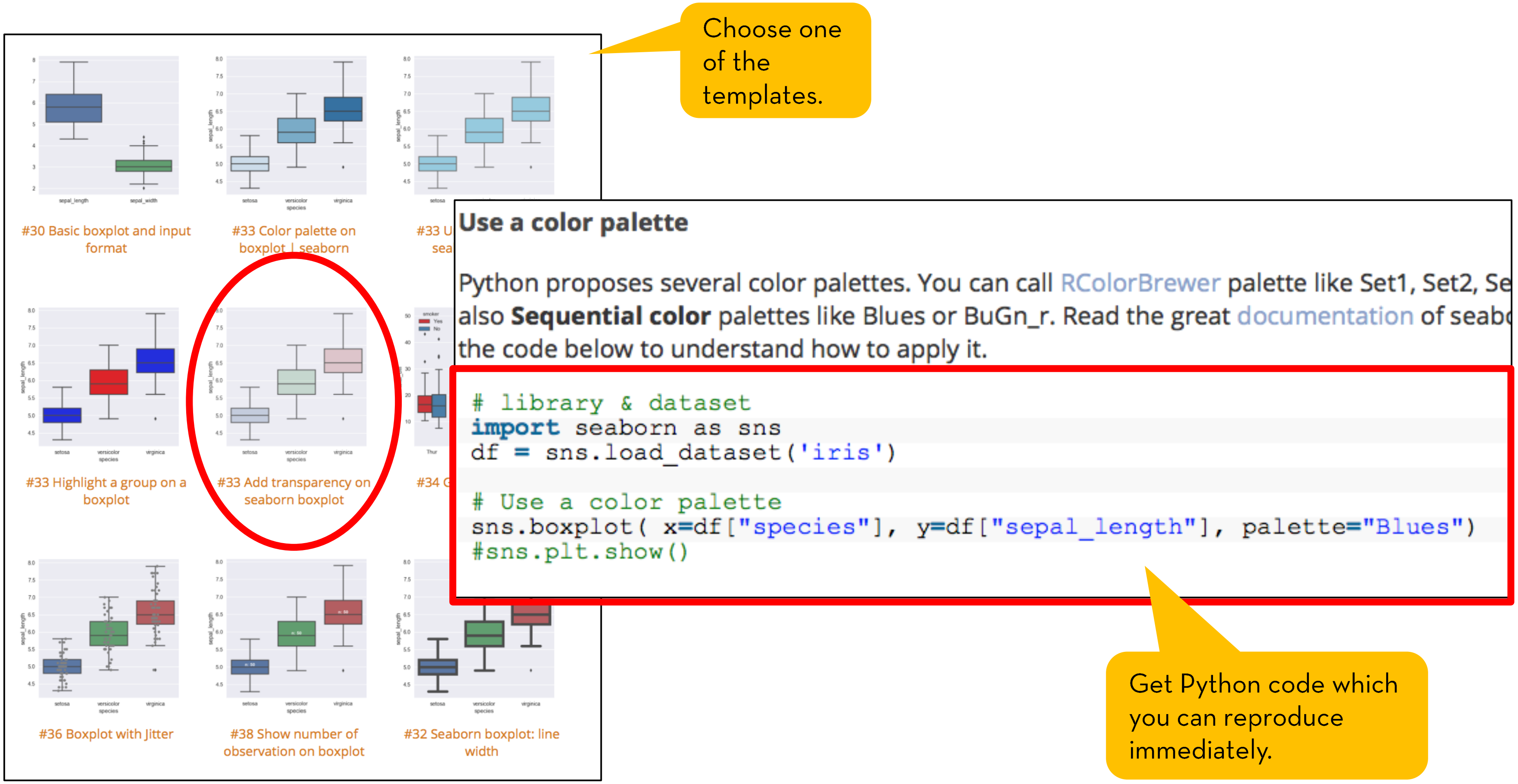
Input format

Variable 1	Group
1.3	A

Find a short function summary and for what data the plot is suited.

Visualize data with https://python-graph-gallery.com

Step 3: Choose your favorite and get the code



#30 Basic boxplot and input format
#33 Color palette on boxplot | seaborn
#33 U
sea
#33 Highlight a group on a boxplot
#33 Add transparency on seaborn boxplot
#34 C
#36 Boxplot with Jitter
#38 Show number of observation on boxplot
#32 Seaborn boxplot: line width

Choose one of the templates.

Use a color palette

Python proposes several color palettes. You can call RColorBrewer palette like Set1, Set2, Se also **Sequential color** palettes like Blues or BuGn_r. Read the great documentation of seab the code below to understand how to apply it.

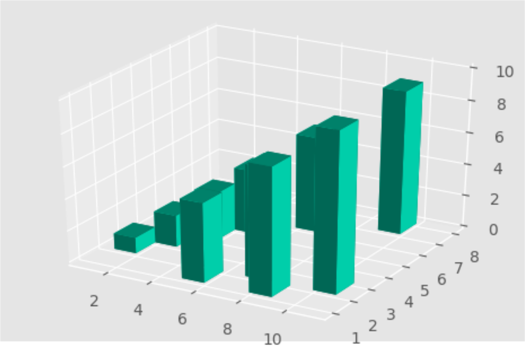
```
# library & dataset
import seaborn as sns
df = sns.load_dataset('iris')

# Use a color palette
sns.boxplot( x=df["species"], y=df["sepal_length"], palette="Blues")
#sns.plt.show()
```

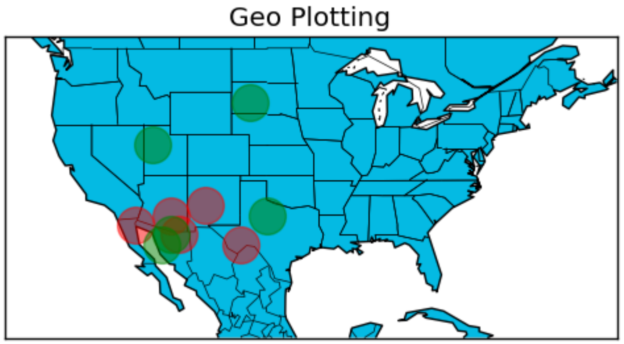
Get Python code which you can reproduce immediately.

Summing up: endless possiblities enable users to create professional data visualization

There are (almost) no restrictions in data visualization. Watch the examples and understand why and how to create even better plots and maps!



3D barplots.



“Basemap” functionality with adjusted markersizes.

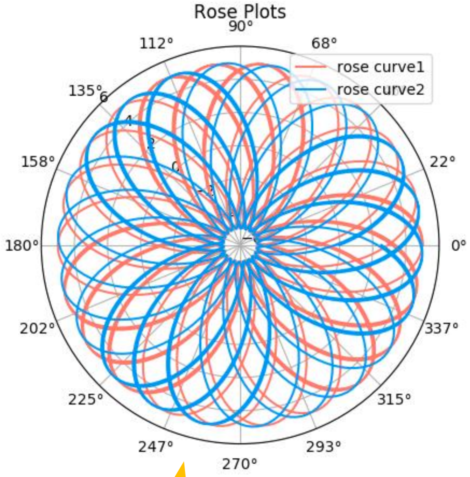


Illustration of polar coordinates.