Introduction to Data Visualization with Python for Statisticians

Stephen Elston, Quantia Analytics, LLC

Presented at Symposium on Statistics and Data Science, Bellevue WA, May 2019

Many statisticians are familiar with the data visualization tools in R, particularly the popular ggplot2 package. However, there is a general lack of awareness of the growing tool set available in Python. This notebook gives statisticians a quick start with performing data visualization with the Python language. These tools along with the Pandas dataframe package are suitable for exploration of complex datasets

More specifically, this notebook provides an overview of the following topics:

1. Basic of scientific plotting using the low-level **Matplotlib** package. Matplot lib is the basis for the other packages discussed herein.
2. Quick plotting with the **Pandas** data frame package.
3. Creating statistical charts for complex data sets with the **Seaborn** package.

About Me

These lessons have been prepared by:

Stephen Elston

- Co-founder and principal consultant Quantia Analytics, LLC
- Instructor, University of Washington and Harvard Extension School
- Creator of seven edX data science and machine learning courses
- Decades of experience in predictive analytics and machine learning
- Experience in multiple industries: payment, telecom, capital markets, logistics, etc.
- PhD, MS in Geophysics from Princeton University

About this Jupyter Notebook

This notebook contains material to help you learn how to explore data visually. This notebook and the data set can be downloaded from GitHub:

https://github.com/Quantia-Analytics/DyDataSF2016Visualization (https://github.com/Quantia-Analytics/DyDataSF2016Visualization)

This notebook was constructed using the Anaconda 3.7 Python distribution. Anaconda provides a convenient way to install a scientific Python tool stack. Once you have installed Anaconda you will be able to use many powerful Python data analysis packages. If you are not running version Anaconda 3.5 or higher, we suggest you update your Anaconda distribution now. You can download the Python 3 Anaconda distribution for your operating system from the Continuum Analytics web site (https://www.continuum.io/downloads).

If you have an older version of the Python 3.X Anaconda stack, you can update by executing the following command from a console command prompt:

```
conda update --all
```

To run this notebook you need to install the Seaborn graphics packages, as it is not in the Anaconda distribution as of now. From a console command prompt on your computer type and execute the following command. If no errors occur, you will have installed Seaborn.

```
pip install seaborn
```

Or you can un-comment and run the code in the cell below. Make sure you leave the shell escape character `!`.

In [ ]:
```
#!pip install seaborn
```

About the dataset

The tips dataset (https://www.kaggle.com/ranjeetjain3/seaborn-tips-dataset) is widely used to demonstrate statistical graphics software including the popular R ggplot2 package. We will use this dataset here, since it is simple and can be compared to some examples in the documentation.

Resources

This tutorial only provides an introduction to the powerful Python plotting packages, Matplotlib, Pandas and Seaborn. These packages have extensive online documentation. There is an extensive tutorial on **Visualization with Pandas** (http://pandas.pydata.org/pandas-docs/version/0.18.0/visualization.html). The **Seaborn tutorial** (https://stanford.edu/~mwaskom/software/seaborn/tutorial.html) contains many examples of statistical data visualization. The matplotlib web site has addition **resources for learning plotting with Python tools** (http://matplotlib.org/resources/index.html).

Load and examine the data set

Let's get started. The code in the cell below imports all of the packages required for this notebook. Two points to notice:

1. Aliases are assigned for some packages with long names using the `as` directive.
2. The IPython 'magic' command `%matplotlib inline` instructs the interpreter to display charts in line in the notebook.

Execute this code.

In [ ]:
```
## Import some packages
import matplotlib.pyplot as plt
import numpy
import numpy.random as nr
import pandas
import seaborn as sns
%matplotlib inline
```

Execute the code in the cell below to load the dataset and examine the head of the **Pandas dataframe**.

In [ ]:
```
tips = sns.load_dataset("tips")
tips.head(10)
```

There are both numeric and categorical columns in this dataframe. As Pandas does not have a categorical data type, categorical data is represented by strings.

The Pandas dataframe package has a great many methods for manipulation, aggregation and computing summaries of dataframes. Here, we will only use one basic summary method, `describe`, which computes summary statistics of numeric columns.

Further exploration of the many capabilities of the Pandas package is beyond the scope of this tutorial. The Pandas project web site contains complete documentation (https://pandas.pydata.org/pandas-docs/stable/) and tutorials (https://pandas.pydata.org/pandas-docs/stable/getting_started/tutorials.html).

Execute the code in the cell below and examine the summary statistics for the numeric columns in the tips dataframe.

In [ ]:
```
tips.describe()
```

Basic Plots with Matplotlib

The base Python plotting package is called **Matplotlib**. In principle, you can create most any plot using the low-level plotting capabilities of Matplotlib. In practice, most people use Python graphics packages which abstract many of the details of Matplotlib. We will discuss two such packages, Pandas plotting and Seaborn in this tutorial.

Nonetheless, it is quite useful to know a bit about Matplotlib. You can use Matplotlib functions to customize plots created by other packages based on this library. You may also wish to create highly specialized plots types directly in Matplotlib.

For a first chart, lets create a scatter plot of the tip amount vs. the total bill. This basic plot is easily created with the Matplotlib scatter (https://matplotlib.org/api/_as_gen/matplotlib.pyplot.scatter.html) method, as shown in the cell below. The scatter plot method operates on the `total_bill` and the `tip` column. Execute this code and examine the result.

In [ ]:
```
plt.scatter(x = tips.total_bill, y = tips.tip)
```

The result is as expected. A very basic scatter plot.

However, this basic plot lacks many essential components, such as axis labels and a title. Fortunately, these elements are easy to add with Matplotlib. The code in the cell below uses several methods to add some missing elements to the chart. Notice how a method for each plot attribute added to the plot is called.

Execute the code and examine the result.

In [ ]:
```
plt.scatter(x = tips.total_bill, y = tips.tip)
plt.xlabel('Bill amount')
plt.ylabel('tip')
plt.title('Tips vs. bill amount')
```

The plot now has reasonable annotation.

In many cases it is important to control the figure size. Correct figure sizing can improve presentation, and more importantly, control aspect ratio.

A Matplotlib figure of the desired size is created using the following calls:

1. A figure with specified dimensions is defined.
2. An axis is then defined for a plot on this figure.
3. Limits are set for the x and y axes.
4. The scatter plot method includes specification of marker attributes, color, transparency and size.
5. Annotations are created by applying the appropriate methods on the axis object.

Execute this code and examine the result.

In [ ]:
```
fig = plt.figure(figsize=(8, 8)) # define plot area
ax = fig.gca() # define axis
ax.scatter(x = tips.total_bill, y = tips.tip, c = 'red', alpha = 0.3, s = 10.0)
ax.set_xlim(0.0, 60.0)
ax.set_ylim(0.0, 12.0)
ax.set_xlabel('Bill amount')
ax.set_ylabel('tip')
ax.set_title('Tips vs. bill amount')
```

There are a great many attributes you can set for an axis. Complete documentation for axis methods is available (https://matplotlib.org/api/_as_gen/matplotlib.pyplot.axes.html).

A Matplotlib figure can have many axes, defining subplots. The Matplotlib documentation includes a demonstration of using multiple axes (https://matplotlib.org/gallery/subplots_axes_and_figures/subplots_demo.html) to create subplots.

Pandas Plot Methods

The Pandas package contains a number of useful basic plot methods which operate on data frames. The simple recipe for plotting from Pandas data frames is:

- Use the plot method, along with the specific plot type method. Alternatively, you can use the `kind` argument of the plot method.
- Specify the columns with the values for the x and y axes.

The code in the cell below creates a basic plot using the plot.scatter (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.plot.scatter.html) method. Execute the code and examine the result.

In [ ]:
```
tips.plot.scatter(x = 'total_bill', y = 'tip')
```

Pandas has a number of useful plotting methods for dataframes. The Pandas documentation contains the details of these plot methods and the available options (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.plot.html).

Since Pandas plotting is based on Matplotlib you can use the same methods to customize a plot. The code in the cell below creates a plot identical to the one you previously created using Matplotlib alone. Notice that the definition of the axes and the plot attribute arguments and methods are identical. There is one difference. The `ax` argument is used to place the Pandas plot on the specified axes. Execute this code and examine the results.

In [ ]:
```
fig = plt.figure(figsize=(8, 8)) # define plot area
ax = fig.gca() # define axis
tips.plot.scatter(x = 'total_bill', y = 'tip', ax = ax, c = 'red', alpha = 0.3, s = 10.0)
ax.set_xlim(0.0, 60.0)
ax.set_ylim(0.0, 12.0)
ax.set_xlabel('Bill amount')
ax.set_ylabel('tip')
ax.set_title('Tips vs. bill amount')
```

You can see that the plot displayed is indeed identical to the one previously created.

Let's try another example, creating a histogram. The code in the cell below is similar to that used to create a scatter plot. The Pandas plot.hist (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.plot.hist.html) method is used in this example. Notice that the aspect ratio of this plot is different from previous examples. Execute the code and examine the result.

In [ ]:
```
fig = plt.figure(figsize=(6, 4)) # define plot area
ax = fig.gca() # define axis
tips['tip'].plot.hist(ax = ax, alpha = 0.5, bins = 15) # Use the plot.hist method on subset of the data frame
ax.set_title('Distribution of tips') # Give the plot a main title
ax.set_xlabel('Tip size') # Set text for the x axis
ax.set_ylabel('Count')# Set text for y axis
```

Statistical Plotting with Seaborn

We will now work with the Seaborn package. Seaborn is a newer Python statistical plotting package which abstracts lower level matplotlib charts. Seaborn also includes some additional cutting-edge statistical chart types suitable for exploration of complex datasets.

We will start with a basic statistical display, a box plot (https://seaborn.pydata.org/generated/seaborn.boxplot.html). Seaborn is built on top of Matplotlib. Therefore, the basic recipe for creating Seaborn plots is the same as other Matplotlib based packages. In the example below this recipe is:

1. Define the figure area and axes.
2. Set a style for the Seaborn plot. In this case, the commonly used `whitegrid` style is specified.
3. The `boxplot` method is called. Seaborn operates on Pandas dataframes and the `data` argument is used to specify the dataframe to use.
4. Other plot attributes are specified using the `set` method on the axes.

Execute this code and examine the results.

In [ ]:
```
fig = plt.figure(figsize=(8,8)) # define plot area
ax = fig.gca() # define axis
sns.set(style="whitegrid")
sns.boxplot(x = 'day', y = "total_bill", data = tips, ax = ax, color = "lightgray")
ax.set(ylim = (0.0, 60.0),
      title = 'Distribution of total bill by day of week',
      xlabel = 'Day of week',
      ylabel = 'Total bill')
```

Kernel density plots with Seaborn

Let's try an example of creating kernel density or kde plots (https://seaborn.pydata.org/generated/seaborn.kdeplot.html). We will start with a one-dimensional KDE plot example. The recipe for this plot is the same as for the boxplot you created with Seaborn.

There is one peculiarity with the `kdeplot` method. There is only a `data` argument which takes one or two columns of data. Whereas, other methods require specification of `x`, `y`, etc. variables.

Execute the code and examine the result.

In [ ]:
```
fig = plt.figure(figsize=(8,4)) # define plot area
ax = fig.gca() # define axis
sns.set_style("whitegrid")
sns.kdeplot(tips['tip'], ax = ax)
ax.set_title('KDE plot of tip') # Give the plot a main title
ax.set_xlabel('Tip') # Set text for the x axis
ax.set_ylabel('Density')# Set text for y axis
```

Next, let's try a two-dimensional KDE plot. As before, the basic recipe is the same. Two columns are specified, one for each axis of the plot. A few other arguments are used. The `bw` argument sets the bandwidth of the kernel. The `cmap` argument specifies a color palette. Execute this code and examine the result.

In [ ]:
```
fig = plt.figure(figsize=(10,10)) # define plot area
ax = fig.gca() # define axis
sns.set_style("whitegrid")
sns.kdeplot(tips[['total_bill','tip']], bw = 1.5, ax = ax, cmap="Blues_d")
sns.scatterplot('total_bill', 'tip', data = tips, ax = ax)
ax.set_xlim(0.0, 60.0)
ax.set_ylim(0.0, 12.0)
ax.set_title('KDE plot of total bill vs tip') # Give the plot a main title
ax.set_xlabel('Tip') # Set text for the x axis
ax.set_ylabel('Total bill')# Set text for y axis
```

There are many other options for the kdeplot method. As with all Seaborn plot methods there is extensive documentation (https://seaborn.pydata.org/generated/seaborn.kdeplot.html).

Violin plots in Seaborn

Violin plots (https://seaborn.pydata.org/generated/seaborn.violinplot.html) are another commonly used statistical display. Seaborn has good support for this plot type.

Additional dimensions of a dataset can be projected using the many Seaborn plot attributes. In the example below, the `hue` and `split` argument are used to split the violin plots by the `sex` variable.

Execute this code to examine the results.

In [ ]:
```
fig = plt.figure(figsize=(8,8)) # define plot area
ax = fig.gca() # define axis
sns.set(style="whitegrid")
sns.violinplot(x = 'day', y = "total_bill", hue = 'sex', split = True, data = tips, ax = ax)
ax.set_title('Distribution of total bill by day of week') # Give the plot a main title
ax.set_xlabel('Day of week') # Set text for the x axis
ax.set_ylabel('Total bill')# Set text for y axis
```

Plot attributes for projecting additional dimensions

Additional dimensions of complex datasets can be added to Seaborn plots. In this example we will use a scatter plot created with the lmplot (https://seaborn.pydata.org/generated/seaborn.lmplot.html) method, which plots a regression line onto the plot. Two attributes are used to show additional dimensions of the dataset:

- The categorical `time` of the meal is shown using the `hue` attribute.
- The numeric `size` of the party is shown as the marker size, using the `s` argument.

Two attributes of this plot are passed to the underlying Matplotlib in the form of a Python dictionary, `alpha` and `s`. Most any attribute of a Matplotlib (https://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html) plot can be set in this matter.

Execute the code in the cell below and examine the result.

In [ ]:
```
sns.lmplot(x = 'total_bill', y = 'tip', data = tips, hue = "time", palette="Set2", scatter_kws={'alpha':0.2, 's':40*tip
s['size']})
plt.title('Tip vs. total bill \nmarker size = party size') # Give the plot a main title
plt.xlabel('Total bill') # Set text for the x axis
plt.ylabel('Tip')# Set text for y axis
```

Facet plots

The Seaborn package has facilities for creating axes for displaying facet plots (https://seaborn.pydata.org/tutorial/axis_grids.html). Facet plots are defined in the following way:

- A `FacetGrid` is defined for the data set. In this case we have only one facet (group by variable), across columns and one across rows.
- The `map` method is applied to the facet grid with a plot type specified. In this case, the regplot (https://seaborn.pydata.org/generated/seaborn.regplot.html) method to create a scatter plot. Notice that some plot attributes are passed to Matplotlib in a Python dictionary.

Execute the code and examine the results.

In [ ]:
```
g = sns.FacetGrid(tips, col = "time", row = 'sex', margin_titles=True)
g.map(sns.regplot, "total_bill", "tip", scatter_kws={'alpha':0.2, 's':40*tips['size']})
```

Pair plots with Seaborn

Pairs plots, also known as scatter plot matrices, are a widely used and powerful exploratory chart type. Seaborn includes sophisticated pairs plotting (https://seaborn.pydata.org/generated/seaborn.pairplot.html) capability which offers many options.

The code in the example below does the following:

1. Creates a scatter plot matrix for the subset of numeric columns specified.
2. Kernel density plots, or `kde` plots, are shown on the diagonal of the matrix.
3. The hue of the markers indicate the value of the `time` variable using the `Set2` pallet.
4. The scatter plots in the upper triangular portion of the display include two-dimensions KDE contour plots.

Execute this code and examine the results.

In [ ]:
```
num_cols = ['total_bill', 'tip', 'size', 'time']
sns.pairplot(tips.loc[:, num_cols], hue='time', palette="Set2",
             diag_kind="kde").map_upper(sns.kdeplot, cmap="Blues", bw = 1.0)
```

Interactive Visualization

Interactive data visualization is beyond the scope of this introductory tutorial. Still, it is worth noting that there are several powerful interactive scientific visualization packages with Python APIs. Two leading packages are Bokeh (https://bokeh.pydata.org/en/latest/) and Plotly (https://plot.ly/python/).

Summary

We have covered a lot of ground in this tutorial. By now, you should have a feel for the types of visualization tools available with the Python language.

Specifically, in this tutorial we have:

- The Matplotlib package forms the base for most scientific visualization with Python. Using the facilities in Matplotlib allows you to customize the attributes of plots created with higher-level graphics packages, including Pandas plotting and Seaborn.
- The Pandas dataframe package includes many useful high-level plotting methods.
- The Seaborn package includes sophisticated statistical plotting capabilities which operate on Pandas dataframes.

Finally, **have fun exploring your data!**