TOOL

Plotting Tip Sheet: Matplotlib, Pandas, and Seaborn

Plotting in Python is supported by many different packages. This tip sheet focuses on the core functionality provided by Matplotlib, as well as the use of Matplotlib by both Pandas and Seaborn to provide convenient interfaces for plotting data from DataFrames and visually characterizing statistical properties of data. Links to documentation for each of these packages include:

- Matplotlib documentation
- Seaborn documentation
- Pandas plotting documentation

Matplotlib

- General information

```
import matplotlib.pyplot as plt       # import the pyplot interface and name it plt
```

  - Multiple function calls using **`plt`** can be made to build up a figure in stages (e.g., plot multiple data sets in the same figure) or customize the plot attributes (axis labels, tick marks, etc.).
  - At any given time, a particular figure is active, meaning that new **`plt`** commands are directed to that figure. If one is generating multiple figures at once, which figure is active can be switched with the **`plt.figure`** function as described below.
  - Some code examples are given below, but many more possibilities exist, so consult the available documentation for more information.

- **plt.figure:** Create a new figure.

```
plt.figure()                 # create a new figure
plt.figure(figsize=(10,10))  # create a new figure with specified size (width, height in inches)
plt.figure(2)                # either create a new figure numbered 2, or make figure 2 active if it
                               already exists
fig, ax = plt.subplots(2,2)  # create grid of subplots with specified number of rows and columns (e.g.,
                               (2,2))
ax[0,0].plot(x, y)           # make plot in subplot by indexing into array of axes produced by plt.
                               subplots
```

- **plt.plot:** Create a line plot.

```
plt.plot(y)                # plot data in list or array y; x-axis defaults to range(len(y))
plt.plot(x, y)             # plot data in y against data in x; requires x and y to have the same length
plt.plot(x, y, ‘bo’)       # plot data in y against data in x, using format string to plot blue (b) circles (o)
plt.plot(x, y, ‘rs-’)      # plot data in y against data in x, using format string to plot red (r) squares (s) connected by lines (-)
plt.plot(a[:,0], a[:,1])   # plot data in column 1 of 2-dimensional array a against data in column 0
plt.errorbar(x, y, yerr)   # similar to plt.plot, but with error bars around data in y as specified in yerr
```

- **plt.scatter:** Create a scatter plot.

```
plt.scatter(x, y)          # scatter plot data y against the data in x
plt.scatter(x, y, s=5,     # scatter plot data y against the data in x, with markers of size 5 and
c=‘b’)                       color blue (“b”)
plt.scatter(x, y,          # scatter plot data y against the data in x, using marker style “^”
marker=‘^’)                  (triangle pointing up)
```

- **plt.bar / plt.barh:** Create a bar chart.

```
plt.bar(x, height)         # vertical bar chart of data in height, with bars positioned according to
plt.bar(x, y, color=‘r’)     data in x
plt.barh(y, width)         # vertical bar chart of data in height, with bars positioned according to
                             data in x, coloring the bars red (“r”)
                           # horizontal bar chart of data in width, with bars positioned according to
                             data in y
```

- `plt.hist:` Create a histogram .

`plt.hist(x)`	`# plot a histogram (with vertical bars), putting data from x into default set of bins`
`plt.hist(x, bins=50)`	`# plot a histogram, putting data from x into specified number of bins (e.g., 50)`
`plt.hist(x, bins=range(0,100))`	`# plot a histogram, putting data from x into bins with specified bin edges`

- Customizing figures (providing additional commands to modify figures already generated with a plotting command):

```
plt.xlim(0, 100)               # set the limits of the x-axis to specified range (e.g., (0, 100))
plt.ylim(0.1, 0.9)             # set the limits of the y-axis to specified range (e.g., (0.1, 0.9))
plt.xlabel(‘year’)             # set the label of the x-axis to specified text (e.g., “year”)
plt.ylabel(‘cost’)             # set the label of the y-axis to specified text (e.g., “cost”)
plt.semilogx()                 # make the x-axis logarithmic
plt.semilogy()                 # make the y-axis logarithmic
plt.loglog()                   # make both axes logarithmic
plt.savefig(‘mydata.png’)      # save the current active figure to specified file; file format is detected from file suffix (e.g., “.png”)
plt.tight_layout()             # automatically adjust plot parameters, typically to reduce plot margins
```

Pandas

- General information

```
import pandas as pd                    # import pandas with shorthand pd
```

  - Pandas uses Matplotlib to make plots of data in DataFrames and Series by calling plot methods on those objects rather than using the underlying plt interface directly.
  - The plt interface can be used directly, however, to further customize plots generated by Pandas.
  - The code examples below assume that there exists a DataFrame named ‘df.’

- `df.plot:` Plot data in a DataFrame.

```
df.plot(x=‘year’, y=‘cost’)                          # make a line plot of the DataFrame column “cost” against the column “year” (assuming they exist)
df.plot(x=‘year’, y=‘cost’, kind=‘bar’)              # make a bar plot of “cost” against “year”
df.plot.bar((x=‘year’, y=‘cost’)                     # same as previous example: bar plot of “cost” against “year”
df.plot(x=‘year’, y=‘cost’, kind=‘scatter’)          # make a scatter plot of “cost” against “year”
df.plot.scatter(x=‘year’, y=‘cost’)                  # same as previous example
df.plot.hist(‘cost’)                                 # plot a histogram of the data in column “cost”
df.plot.box()                                        # make a box plot (box and whisker) of all the columns in the DataFrame
df.groupby(‘category’).mean().plot(x=‘year’, y=‘cost’)    # make a plot for a DataFrame derived through other operations such as groupby
```

Seaborn

- General information

```
import seaborn as sns                # import seaborn with shorthand sns
```

  - Seaborn uses Matplotlib to make plots of data, usually in DataFrames and Series, by calling plot methods on those objects rather than using the underlying plt interface directly.
  - The plt interface can be used directly, however, to further customize plots generated by Seaborn.
  - Since Seaborn is focused largely on characterizing statistical properties of data, its functionality is broadly divided into subareas for: (1) visualizing statistical relationships; (2) plotting with categorical data; (3) visualizing the distribution of a data set; and (4) visualizing linear relationships.
  - Seaborn contains some predefined data sets that can be loaded using the `sns.load_dataset` function; for example:
    - `tips = sns.load_dataset('tips')`
    - `mpg = sns.load_dataset('mpg')`
    - `fmri = sns.load_dataset('fmri')`
    - `titanic = sns.load_dataset('titanic')`
    - `iris = sns.load_dataset('iris')`
  - The code examples below assume that these sample DataFrames have been loaded and are drawn from the material in the Official Seaborn Tutorial.
- `sns.relplot:` Plot relationship between variables in a DataFrame.

```
sns.relplot(x='total_bill', y='tip', data=tips)                 # for tips, make a scatter plot between "total_bill" and "tip" (scatter plot: default kind)
sns.relplot(x='total_bill', y='tip', hue='smoker', data=tips)   # same as above, but color each point by categorical data in "smoker"
sns.relplot(x='total_bill', y='tip', kind='line', data=tips)    # for tips, make a line plot between "total_bill" and "tip" (kind="line")
sns.relplot(x='timepoint', y='signal', kind='line', ci=95, data=fmri)   # for fmri, make a line plot with 95% confidence interval about mean
```

- `sns.catplot:` Plot with categorical data.

```
sns.catplot(x='day', y='total_bill', data=tips)                         # for each category in x ("day"), make scatter plot of data in y ("total_bill"), with jitter
sns.catplot(x='day', y='total_bill', kind='swarm', data=tips)           # as above, but splay points as in a "beeswarm" to prevent them from overlapping
sns.catplot(x='day', y='total_bill', kind='box', data=tips)             # for each category in x, make a box plot for data in y
sns.catplot(x='sex', y='survived', hue='class', kind='bar', data=titanic)   # make a bar chart, coloring each bar by "class" category
sns.catplot(x='deck', kind='count', palette='ch:.25', data=titanic)     # make "count" plot (i.e., histogram) for categories in x (with a specified color palette)
sns.catplot(x='day', y='total_bill', col='smoker', data=tips)           # make multiple catplots in a FacetGrid, in columns for each category in col="smoker"
```

- `sns.distplot:` Plot a univariate distribution.

```
sns.distplot(tips['total_bill'])                        # plot a histogram, along with a line representing kernel density estimation (kde=True by default)
sns.distplot(tips['total_bill'], kde=False, rug=True)   # plot a histogram, along with a "rug" plot showing individual values, but no kde
sns.distplot(tips['total_bill'], bins=20, rug=True)     # change the number of bins
sns.distplot(tips['total_bill'], hist=False, rug=True)  # no histogram, but kde and rug plot
```

- `sns.jointplot:` Plot bivariate distributions.

```
sns.jointplot(x='total_bill', y='tip', data=tips)               # make a scatter plot of x and y, with histograms for each variable along each axis
sns.jointplot(x='total_bill', y='tip', kind='hex', data=tips)   # make a "hexbin" plot of x and y, with histograms for each variable along each axis
sns.jointplot(x='total_bill', y='tip', kind='kde', data=tips)   # make a kde-based contour plot of x and y, with kde plots for each variable along each axis
```

- `sns.pairplot:` Plot pairwise relationships among variables.

```
sns.pairplot(iris)                                      # make a pairwise scatter plot grid of columns in "iris" data set
sns.pairplot(iris, hue='species')                       # color each category of "species" by a distinct color
```

- `sns.regplot and sns.lmplot:` Plot regression models among variables.

```
sns.regplot(x='total_bill', y='tip', data=tips)               # scatter plot of x and y, along with best-fit linear regression and 95% confidence interval
sns.lmplot(x='total_bill', y='tip', hue='smoker', data=tips)  # multiple scatter plots and regression lines, for each category in hue="smoker"
```