Topic for the class-text and annotation

Unit _2 : Title-Digital data – an Imprint

Date & Time : 22.8.24 10.00 AM – 10.50 AM

Dr. Bhramaramba Ravi

Professor

Department of Computer Science and Engineering

GITAM School of Technology (GST)

Visakhapatnam – 530045

Email: **bravi@gitam.edu**

Unit2-syllabus

- **UNIT 2 Digital Data-An Imprint 9 hours, P - 2 hours** Type of data analytics (Descriptive, diagnostic, perspective, predictive, Prescriptive.) Exploratory Data Analysis (EDA), EDA-Quantitative Technique, EDA - Graphical Technique. Data Types for Plotting, Data Types and Plotting, Simple Line Plots, Simple Scatter Plots, Visualizing Errors, Density and Contour Plots, Histograms, Binnings, and Density, Customizing Plot Legends, Customizing Color bars, Multiple Subplots, Text and Annotation, Customizing Ticks.
- https://www.coursera.org/learn/data-visualization-r

Text and Annotation

- Creating a good visualization involves guiding the reader so that the figure tells a story.
- In some cases, this story can be told in an entirely visual manner, without the need for added text, but in others, small textual cues and labels are necessary.
- Perhaps the most basic types of annotations you will use are axes labels and titles, but the options go beyond this.
- Let's take a look at some data and how we might visualize and annotate it to help convey interesting information..
- We'll start by setting up the notebook for plotting and importing the functions we will use:
- In[1]: %matplotlib inline
- **import** matplotlib.pyplot **as** plt
- **import** matplotlib **as** mpl
- plt.style.use('seaborn-whitegrid')
- **import** numpy **as** np
- **import** pandas **as** pd

Text and annotation

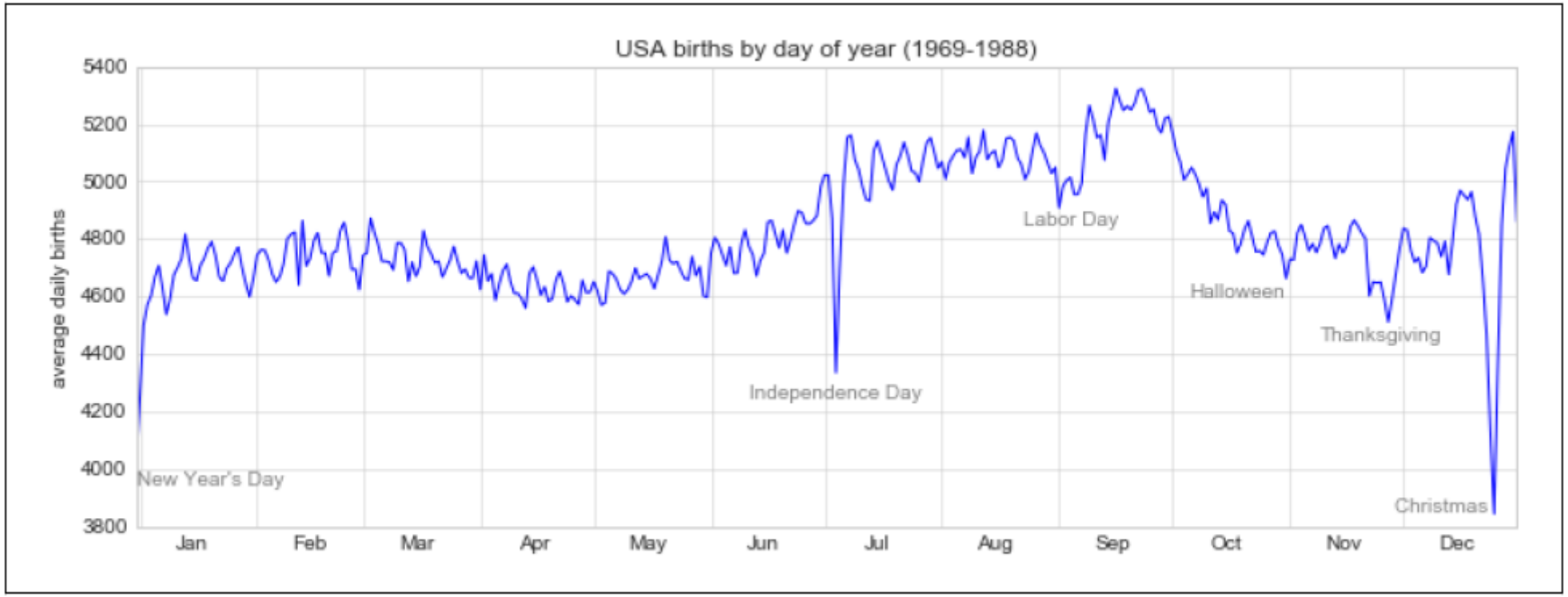

Figure 4-68. Annotated average daily births by date

Text and annotation

- The ax.text method takes an *x* position, a *y* position, a string, and then optional keywords specifying the color, size, style, alignment, and other properties of the text.
- Here we used ha='right' and ha='center', where ha is short for *horizonal alignment*.
- **Transforms and Text Position**
- In the previous example, we anchored our text annotations to data locations. Sometimes it’s preferable to anchor the text to a position on the axes or figure, independent of the data.
- In Matplotlib, we do this by modifying the *transform*.
- Any graphics display framework needs some scheme for translating between coordinate systems.
- For example, a data point at *x*, *y* = 1, 1 needs to somehow be represented at a certain location on the figure, which in turn needs to be represented in pixels on the screen.
- Mathematically, such coordinate transformations are relatively straightforward, and Matplotlib has a well-developed set of tools that it uses internally to perform them (the tools can be explored in the matplotlib.transforms submodule).

Text and annotation

- Any graphics display framework needs some scheme for translating between coordinate systems.
- For example, a data point at *x*, *y* = 1, 1 needs to somehow be represented at a certain location on the figure, which in turn needs to be represented in pixels on the screen.
- Mathematically, such coordinate transformations are relatively straightforward, and Matplotlib has a well-developed set of tools that it uses internally to perform them (the tools can be explored in the matplotlib.transforms submodule).
- The average user rarely needs to worry about the details of these transforms, but it is helpful knowledge to have when considering the placement of text on a figure.
- There are three predefined transforms that can be useful in this situation:
- ax.transData
- Transform associated with data coordinates
- ax.transAxes
- Transform associated with the axes (in units of axes dimensions)
- fig.transFigure
- Transform associated with the figure (in units of figure dimensions)
- Here let’s look at an example of drawing text at various locations using these transforms(Fig.4-69)

Text and annotation

- In[5]: fig, ax = plt.subplots(facecolor='lightgray')
- ax.axis([0, 10, 0, 10])
- *# transform=ax.transData is the default, but we'll specify it anyway*
- ax.text(1, 5, ". Data: (1, 5)", transform=ax.transData)
- ax.text(0.5, 0.1, ". Axes: (0.5, 0.1)", transform=ax.transAxes)
- ax.text(0.2, 0.2, ". Figure: (0.2, 0.2)", transform=fig.transFigure);

Text and annotation

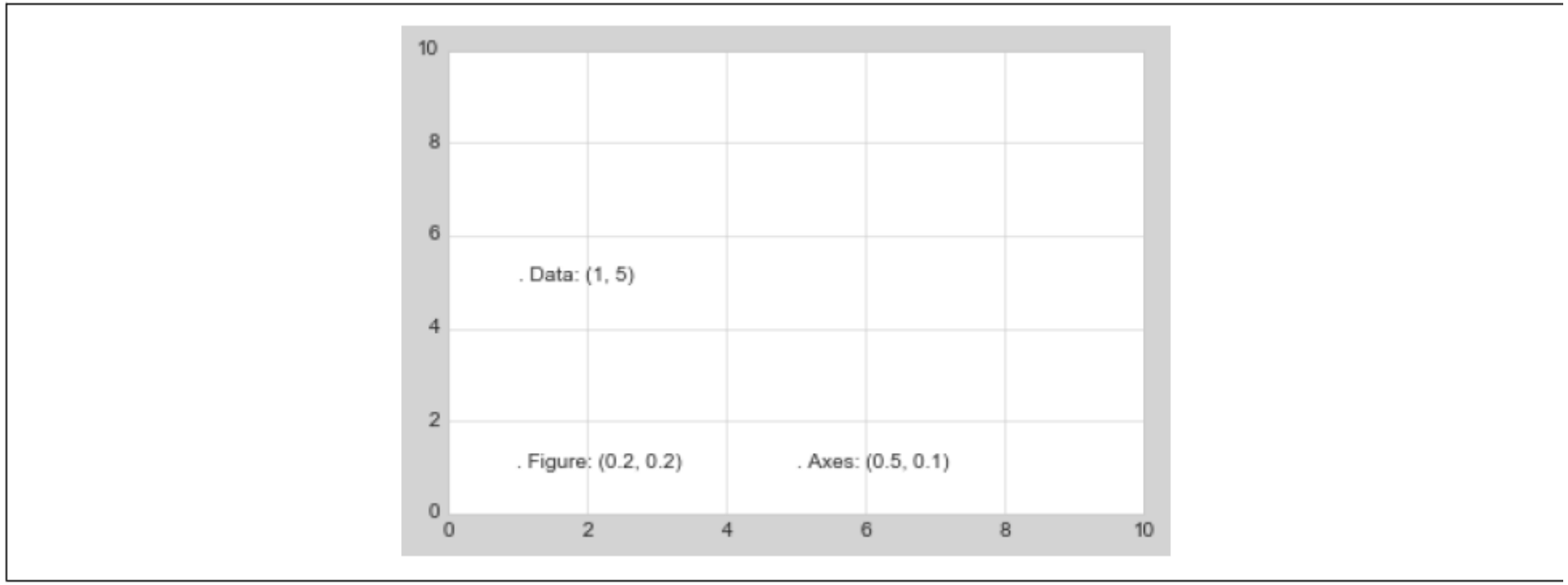

Figure 4-69. Comparing Matplotlib's coordinate systems

Text and annotation

- Note that by default, the text is aligned above and to the left of the specified coordinates;
- here the "." at the beginning of each string will approximately mark the given coordinate location.
- The transData coordinates give the usual data coordinates associated with the x- and y-axis labels.
- The transAxes coordinates give the location from the bottom-left corner of the axes (here the white box) as a fraction of the axes size.
- The transfigure coordinates are similar, but specify the position from the bottom left of the figure
- (here the gray box) as a fraction of the figure size.
- Notice now that if we change the axes limits, it is only the transData coordinates that will be affected, while the others remain stationary (Figure 4-70):
- In[6]: ax.set_xlim(0, 2)
- ax.set_ylim(-6, 6)
- fig

Text and annotation

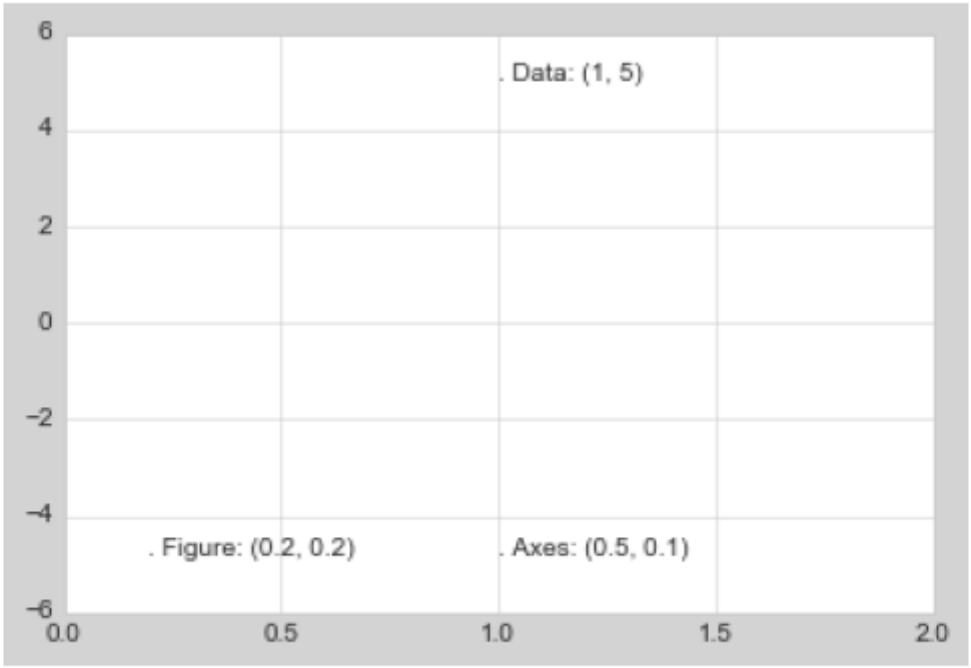

Figure 4-70. Comparing Matplotlib's coordinate systems

Text and annotation

- **Arrows and Annotation**
- Along with tick marks and text, another useful annotation mark is the simple arrow.
- Drawing arrows in Matplotlib is often much harder than you might hope.
- While there is a plt.arrow() function available, I wouldn’t suggest using it; the arrows it creates are SVG objects that will be subject to the varying aspect ratio of your plots, and the result is rarely what the user intended.
- Instead, I’d suggest using the plt.annotate() function.
- This function creates some text and an arrow, and the arrows can be very flexibly specified.

Text and annotation

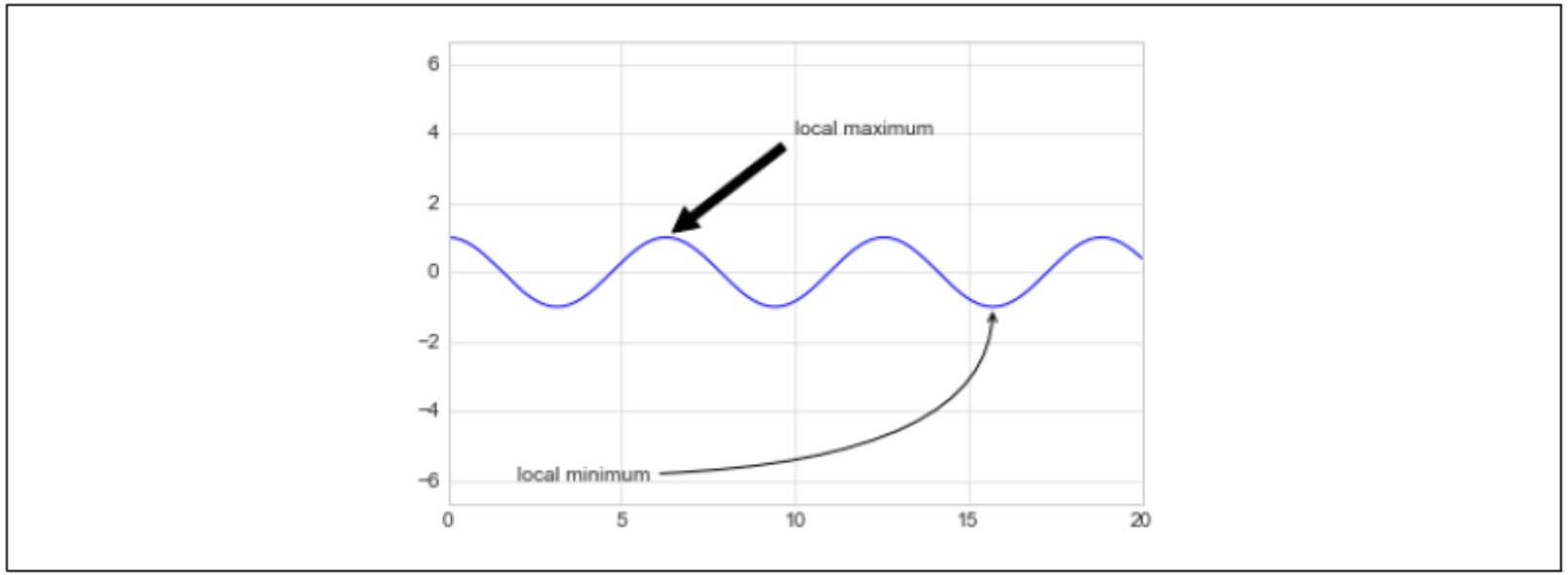

Figure 4-71. Annotation examples

Text and annotation

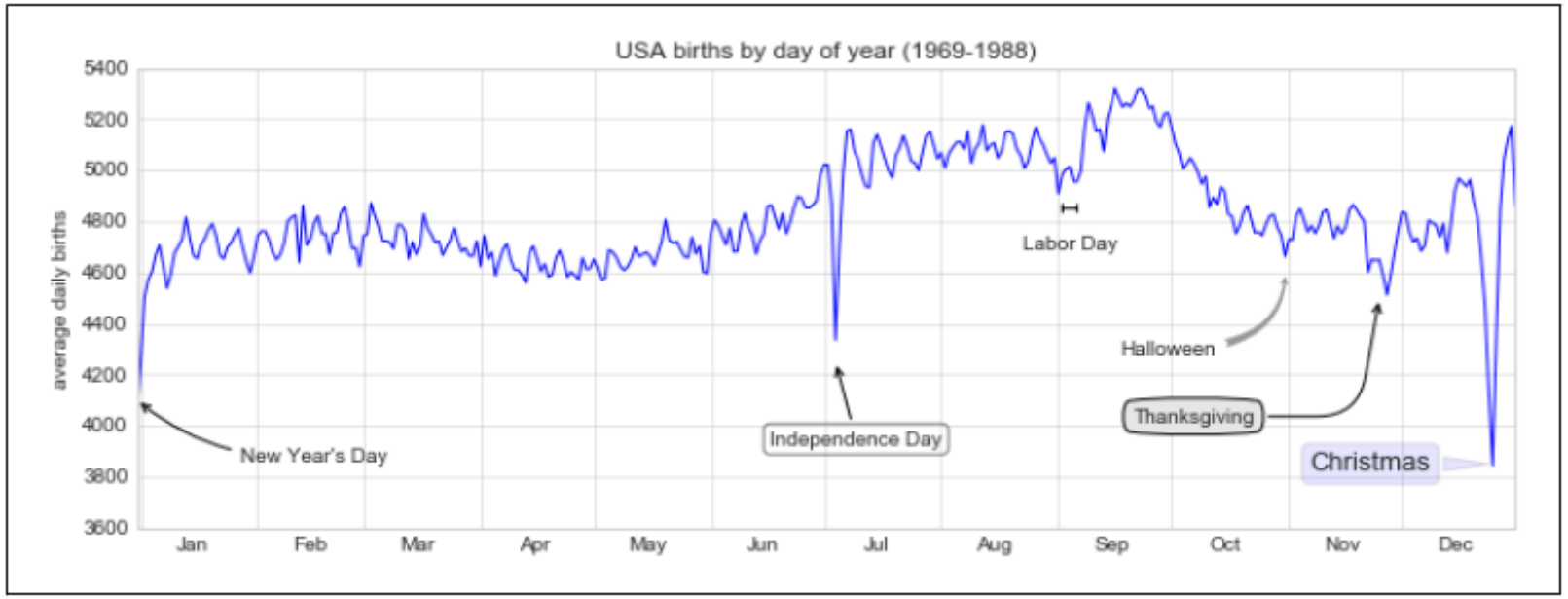

Figure 4-72. Annotated average birth rates by day

THANK YOU