Exploratory Visualizations part-1 - matplotlib vs base R

Shankar Muthuswamy — 2014-12-16 04:30 — Comments (exploringviz1.html#disqus_thread)

In these series of posts, I will try to visually compare and contrast visualization tools focusing specifically on Python and R. We will look at a wide array of tools such as matplotlib, base graphics in R, lattice, ggplot2 and visually pit them against each other by creating some simple visualizations. Later we will turn our attention to matplotlib and implement some ideas to make its visualizations more impressive. This isn't meant to be a tutorial (there's plenty out there) but hopefully there's a trick or two along the way that's helpful.

In this first post, we will begin by comparing matplotlib with the base graphics package in R.

To start with we will use a simple dataset that provides specifications for 428 new vehicles for the year 2004. This dataset has been used for a number of statistics courses since the results are easier for everyone to relate to. We however will use the data to focus on the tools rather than focus on the data itself.

Let's go!

Extract the data with Pandas

We will read in the dataset using Pandas, the super-cool data analysis library of Python and add a couple of new features that'll make plotting easier. The dataset is fairly clean, so we can focus on the good stuff.

Pandas also comes with wrappers around several matplotlib routines to allow for quick and easy plotting of dataframes.

In [5]:
```
#Import required Python modules
from matplotlib import pyplot as plt
from matplotlib import colors
from pylab import cm
import numpy as np
import pandas as pd

# make graphics inline
%matplotlib inline
```

Let's read in the csv dataset with Pandas.

In [6]:
```
#Read in the dataset from visualizing.org
cars=pd.read_excel('04cars data.xls', na_values=[None, '*'])
```

In [7]:
```
#Let's take a quick peek at the dataset
cars.head(5)
```

Out[7]:

	Vehicle Name	Small/Sporty/ Compact/Large Sedan	Sports Car	SUV	Wagon	Minivan	Pickup	AWD	RWD	Retail Price	Dealer Cost	Engine Size (l)	Cyl	HP	City MPG	Hwy MPG	Weight
0	Acura 3.5 RL 4dr	1	0	0	0	0	0	0	0	43755	39014	3.5	6	225	18	24	3880
1	Acura 3.5 RL w/Navigation 4dr	1	0	0	0	0	0	0	0	46100	41100	3.5	6	225	18	24	3893
2	Acura MDX	0	0	1	0	0	0	1	0	36945	33337	3.5	6	265	17	23	4451
3	Acura NSX coupe 2dr manual S	0	1	0	0	0	0	0	1	89765	79978	3.2	6	290	17	24	3153
4	Acura RSX Type S 2dr	1	0	0	0	0	0	0	0	23820	21761	2.0	4	200	24	31	2778

That is pretty self-explanatory. It's a list of different types of vehicles and their specifications such as MPG, Retail price and so on. The one thing I would like to add to this dataset though is the vehicle family. E.g., Acura, Mercedes. Let's quickly extract this as a seperate column.

In [8]:
```
#Generate Brand for each Car
cars['Brand']=pd.DataFrame(cars['Vehicle Name'].str.split().tolist()).ix[:,0]
```

Let's also extract those labels as target values so we can use them easily in plots.

In [9]:
```
from sklearn.preprocessing import LabelEncoder
encode=LabelEncoder()
cars['Brand_label']=encode.fit_transform(cars.values[:,-1:].ravel())
```

In [10]:
```
#Save the df to disk for future executions
cars.to_pickle('cars.pkl')
```

matplotlib

Let's begin with matplotlib, the highly versatile but not-so-pretty out-of-the-box graphics package for Python. I am a huge fan of Python but never quite liked how old-school matplotlib was by default, unlike R's ggplot or for that matter R's default graphics package. But what I quickly learnt was that matplotlib is super-customizable. If you're willing to devote time, you can really turn it into something so much more pleasing to the eye.

Let's start with some basics - Line, Scatter, Histo and Bar, one at a time.

Line Charts

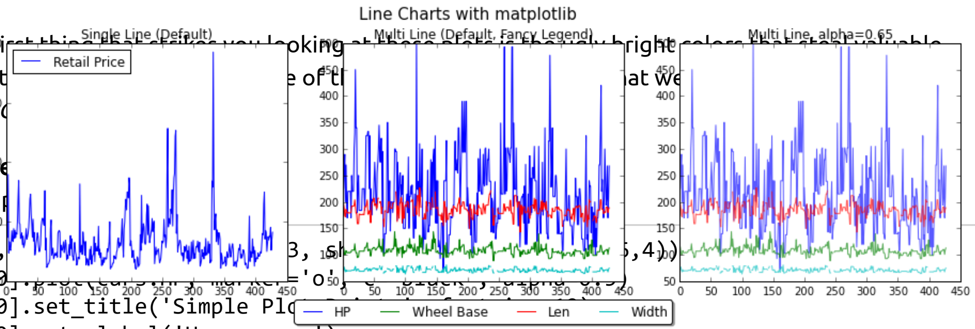
In [6]:

```
fig, ax = plt.subplots(1, 3, sharex=True, figsize=(16,4))
ax[0].plot(cars['Retail Price'])
ax[0].legend(['Retail Price'], loc='best')
ax[0].set_title('Single Line (Default)', fontsize=12)



ax[1].plot(cars['HP'])
ax[1].plot(cars['Wheel Base'])
ax[1].plot(cars['Len'])
ax[1].plot(cars['Width'])
ax[1].set_title('Multi Line (Default, Fancy Legend)', fontsize=12)
ax[1].legend(['HP','Wheel Base', 'Len', 'Width'], loc='upper center', bbox_to_anchor=(0.5, -0.05),
          fancybox=True, shadow=True, ncol=5)

ax[2].plot(cars['HP'], alpha=0.65)
ax[2].plot(cars['Wheel Base'], alpha=0.65)
ax[2].plot(cars['Len'], alpha=0.65)
ax[2].plot(cars['Width'], alpha=0.65)
ax[2].set_title('Multi Line, alpha=0.65', fontsize=12)
# ax[2].legend(['HP','Wheel Base', 'Len', 'Width'], loc='upper center', bbox_to_anchor=(0.5, -0.05),
#           fancybox=True, ncol=5)
plt.suptitle('Line Charts with matplotlib', fontsize=15, y=1.02)
plt.show()
```



The first thing that strikes you looking at these plots is the ugly bright colors that steal valuable data t e of tl at we (*RGBC*

Scatte

Let's p

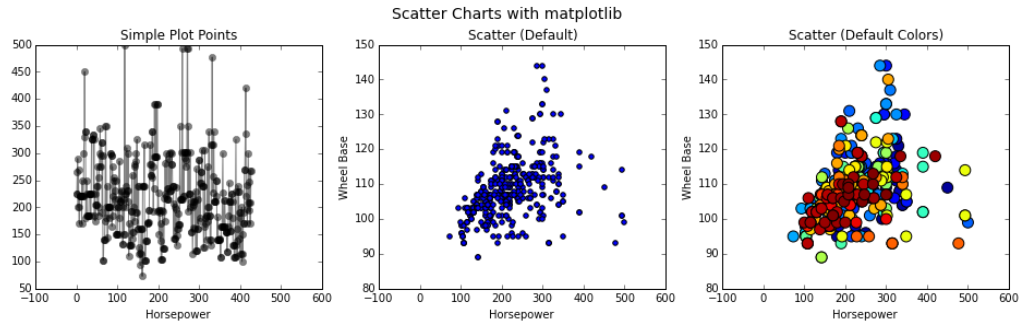
In [7]:

```
fig, 3, sh 5,4))
ax[0].plot(cars.HP, marker='o', c='black', alpha=0.5)
ax[0].set_title('Simple Plo
ax[0].set_xlabel('Horsepower')

ax[1].scatter(cars['HP'], cars['Wheel Base'])
ax[1].set_xlabel('Horsepower')
ax[1].set_ylabel('Wheel Base')
ax[1].set_title('Scatter (Default)', fontsize=12)

ax[2].scatter(cars['HP'], cars['Wheel Base'], s=100, c=cars.Brand_label)
ax[2].set_xlabel('Horsepower')
ax[2].set_ylabel('Wheel Base')
ax[2].set_title('Scatter (Default Colors)', fontsize=12)

plt.suptitle('Scatter Charts with matplotlib', fontsize=14, y=1.02)
plt.show()
```



These look better than line charts but there's a lot we can do to make it aesthetically more appealing as well as turn more focus to the data itself. The latter isn't specific to matplotlib, some level of customization will always be required to allow for better presentation.

Histos & Bar

How about Bar Charts and Histograms?

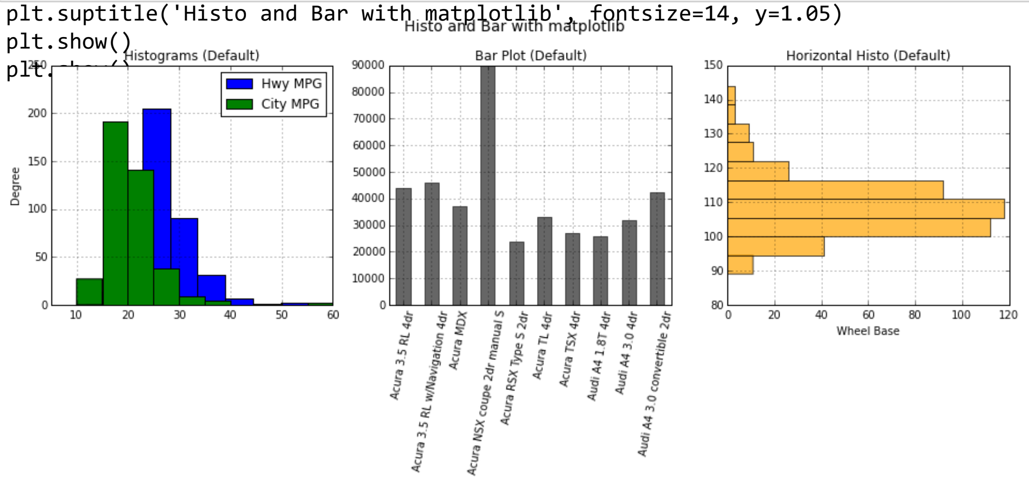
In [61]:

```
fig, ax = plt.subplots(1, 3, figsize=(16,4))
cars['Hwy MPG'].plot(kind='hist', ax=ax[0])
cars['City MPG'].plot(kind='hist', ax=ax[0])
ax[0].set_xlim(5,60)
ax[0].set_title('Histograms (Default)')
ax[0].legend(loc='best')

cars[:10].plot('Vehicle Name','Retail Price', kind='bar', legend=False, ax=ax[1], color='black', alpha=.6, rot=80)
ax[1].set_title('Bar Plot (Default)')
ax[1].set_xlabel('')

cars[['Wheel Base']].plot(kind='hist', orientation='horizontal', ax=ax[2], color='Orange', alpha=.7, legend=False)
ax[2].set_title('Horizontal Histo (Default)')
ax[2].set_xlabel('Wheel Base')
```

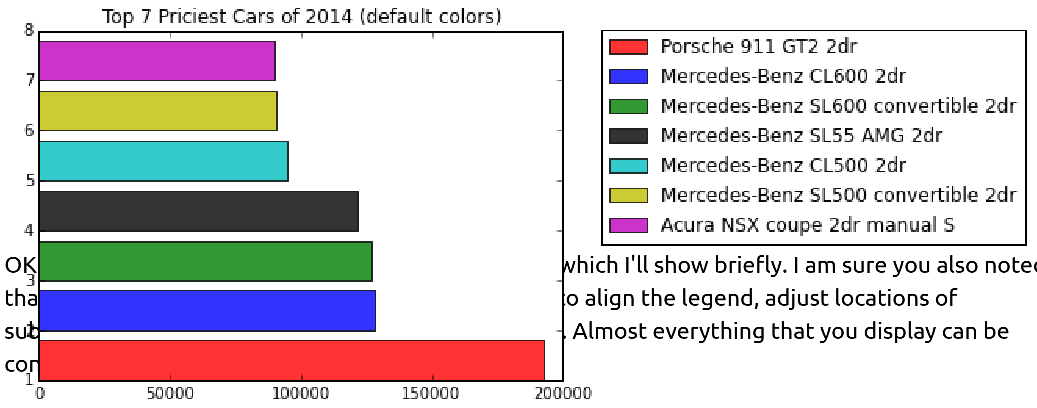
```
plt.suptitle('Histo and Bar with matplotlib', fontsize=14, y=1.05)
plt.show()
plt.show()
```



What if we wanted something customized? Like say the Top x pricey cars I can't afford?

```
In [65]:  pricey_cars=cars.sort('Retail Price', ascending=False, inplace=False)
          pricey_cars.reset_index(drop=True, inplace=True)
          colors=['r','b','g','black','c','y','m']
          for i in range(7):
              plt.barh(i+1, pricey_cars['Retail Price'].ix[i], label=pricey_cars['Vehicle Name'].ix[i], color=colors[i], alpha

          plt.legend(bbox_to_anchor=(1.9, 1.03))
          plt.title('Top 7 Priciest Cars of 2014 (default colors)', fontsize=12)
          plt.show()
```



OK which I'll show briefly. I am sure you also noted tha o align the legend, adjust locations of sub . Almost everything that you display can be con

Before we look at options of customization, let's take a short tour of R and see how the defaults fare in comparison.

Plotting in Base R

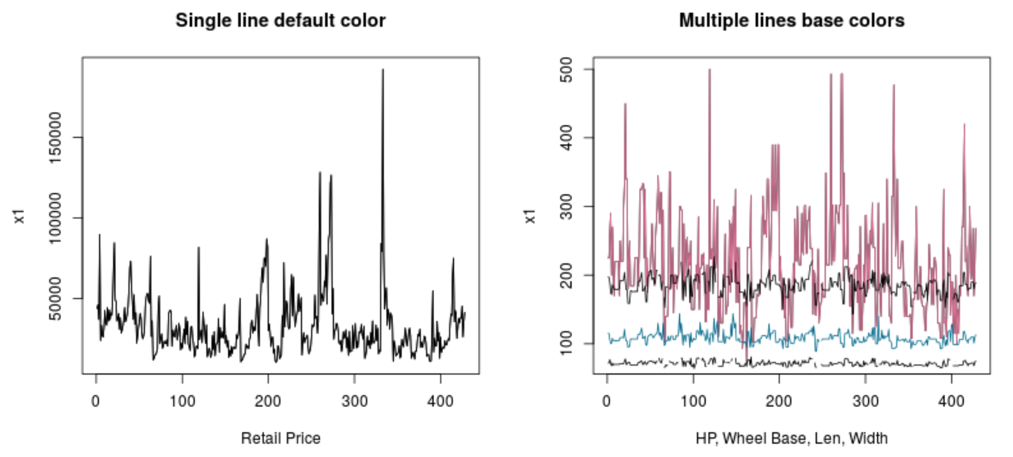
Let's create some basic plots, now in R. We're not going to be using the powerful ggplot2 yet. Let's stick with default R for the time being.

In [68]:
```
#Load the R Magic so we can execute R scripts within this notebook
%load_ext rmagic
```

Line Charts

In [69]:
```
%%R -i cars -w 800 -h 380 -u px
#Pass and receive dataframe from Python to R
cars<-data.frame(cars)
```

In [70]:
```
%%R -w 800 -h 380 -u px
mat<-layout(matrix(c(1,2),1,2))
plot(cars['Retail.Price'], type='l', main='Single line default color', xlab='Retail Price')
lines(cars['Retail.Price'])
plot(cars['HP'], type='l',main='Multiple lines base colors',xlab='HP, Wheel Base, Len, Width')
lines(cars['HP'], col='palevioletred')
lines(cars['Wheel.Base'], col='deepskyblue4')
lines(cars['Len'], color='paleturquoise4')
lines(cars['Width'], color='firebrick')
#legend('topright', inset=c(-.2,0),legend=c('HP','Wheel Base', 'Len', 'Width'),
#       col=c('palevioletred','deepskyblue4','paleturquoise4','firebrick'),xpd=TRUE)
```



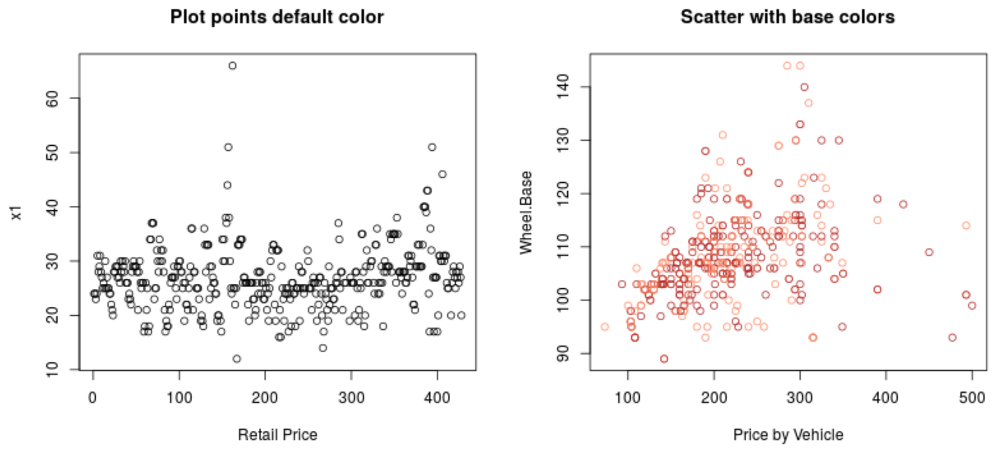
Few interesting things to note there.

- The default color that R throws at you is black, which is much better than the nasty blue.
- Out of the box R gives you a plethora of colors so you don't need to go elsewhere. As you can see the second plot with firebrick, turqoise and so on.
- Default R is more pleasing to the eye and less distracting than matplotlib

Note the lack of a legend and ugly use of labels in the x axis. That's partly my fault as I chose to put it there but the lack of legend is due to IPython and R not playing well together. If you try to move the legend outside the plot (it's too big to be inside) IPython will truncate it since we fix the plot size going in.

You can see the command to display legend though in the commented line. Let's finish up and look at scatter and bar plots as well.

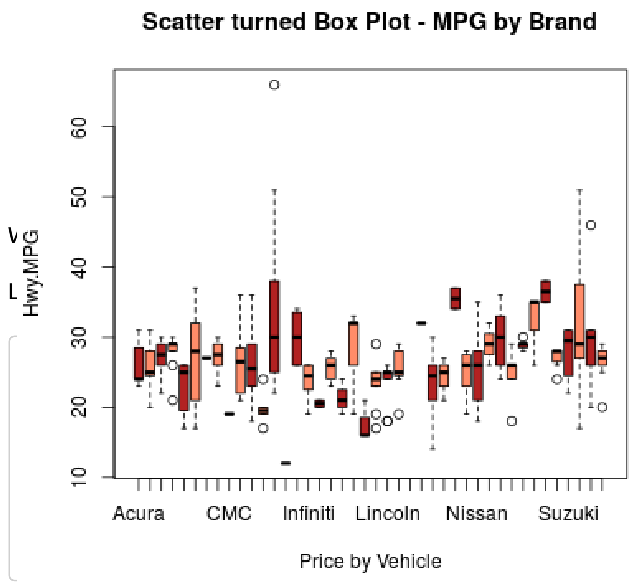
In [71]:
```
%%R -w 800 -h 380 -u px
#Pass and receive dataframe from Python to R
mat<-layout(matrix(c(1,2),1,2))
plot(cars['Hwy.MPG'], main='Plot points default color', xlab='Retail Price')
plot(cbind(cars['HP'], cars['Wheel.Base']), main='Scatter with base colors',xlab='Price by Vehicle',
         col=c('firebrick','salmon1'))
```



Again, R plots look more beautiful yet easy on the eye. It's the colors that make a huge difference here. Also note that the second scatter plot wasn't colored based on anything specific. I love the way we code labels in Python when the system wonderfully uses the colors to contrast different values, by default. We can do the same thing in R, its just a bit more involved.

R has other advantages too though. It automatically adjustd chart types based on the data we're plotting. For instance, if I chose to display the Highway MPG by Brand, R will automatically change the plot type to a mixed box and scatter plot.

In [72]:
```
%%R -w 400 -h 380 -u px
#Pass and receive dataframe from Python to R
plot(cbind(cars['Brand'], cars['Hwy.MPG']), main='Scatter turned Box Plot - MPG by Brand',xlab='Price by Vehicle',
         col=c('firebrick','salmon1'))
```



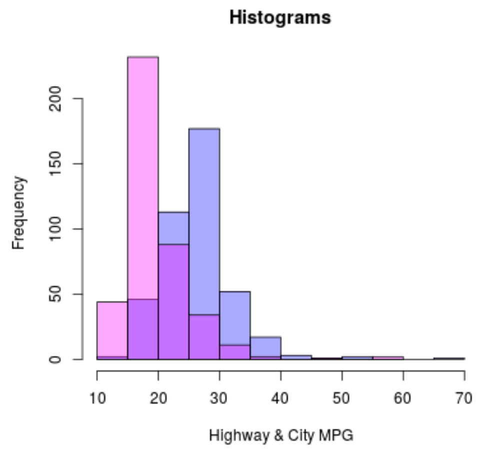
V

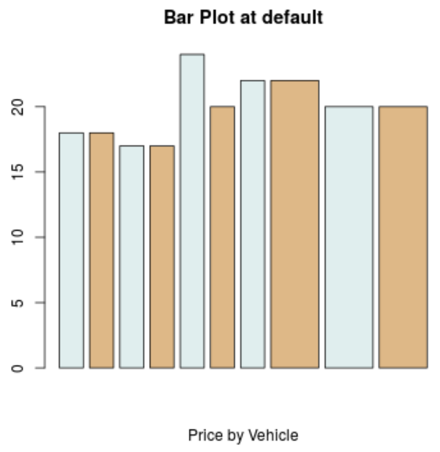
L

ta point for CMC.

In [179]:

```
0,1,1/3), main='Histograms', ylim=c(0,225), xlab="Highway & Cit
,0,1,1/3), add=T)
)))[1:10],  as.numeric(unlist(cars['Brand']))[1:10],
hicle',
```





In our brief tour of base graphics with R, we already see that the defaults in matplotlib aren't anywhere close aesthetically to the results in R. Although there is much more than just aesthetics to a visualization. The beauty of R is not its base graphics package but the awesome ggplot2.

We will explore the ggplot2 package in the next post.

Happy Reading!

Share more, Learn more!

0 1 0 0 Google + 1 0

R (../categories/r.html) **base R graphics (../categories/base-r-graphics.html)** **ggplot2 (../categories/ggplot2.html)** **matplotlib (../categories/matplotlib.html)** **pandas (../categories/pandas.html)** **python (../categories/python.html)** **visualization (../categories/visualization.html)**

Previous post (pca-vs-lr.html) Next post (exploringviz2.html)

Comments

Comments powered by Disqus (//disqus.com)