Project - Building a Custom Visualization of Bar using Color

The challenges users face when trying to make judgements about probabilistic data generated through samples. As an example, they look at a bar chart of four years of data. Each year has a y-axis value, which is derived from a sample of a larger dataset. For instance, the first value might be the number votes in a given district or riding for 1992, with the average being around 33,000. On top of this is plotted the 95% confidence interval for the mean (see the boxplot lectures for more information, and the yerr parameter of barcharts).

A challenge that users face is that, for a given y-axis value (e.g. 42,000), it is difficult to know which x-axis values are most likely to be representative, because the confidence levels overlap and their distributions are different (the lengths of the confidence interval bars are unequal). One of the solutions the authors propose for this problem (Figure 2c) is to allow users to indicate the y-axis value of interest (e.g. 42,000) and then draw a horizontal line and color bars based on this value. So bars might be colored red if they are definitely above this value (given the confidence interval), blue if they are definitely below this value, or white if they contain this value.

Easiest option: Implement the bar coloring as described above - a color scale with only three colors, (e.g. blue, white, and red). Assume the user provides the y axis value of interest as a parameter or variable.

Harder option: Implement the bar coloring as described in the paper, where the color of the bar is actually based on the amount of data covered (e.g. a gradient ranging from dark blue for the distribution being certainly below this y-axis, to white if the value is certainly contained, to dark red if the value is certainly not contained as the distribution is above the axis).

Even Harder option: Add interactivity to the above, which allows the user to click on the y axis to set the value of interest. The bar colors should change with respect to what value the user has selected.

Hardest option: Allow the user to interactively set a range of y values they are interested in, and recolor based on this (e.g. a y-axis band, see the paper for more details).

In [3]:

```
import matplotlib.pyplot as plt
from matplotlib.colors import Normalize
from matplotlib.cm import get_cmap
%matplotlib notebook
import pandas as pd
import numpy as np

np.random.seed(12345)

df = pd.DataFrame([np.random.normal(32000,200000,3650),
                   np.random.normal(43000,100000,3650),
                   np.random.normal(43500,140000,3650),
                   np.random.normal(48000,70000,3650)],
                  index=[1992,1993,1994,1995])
df
```

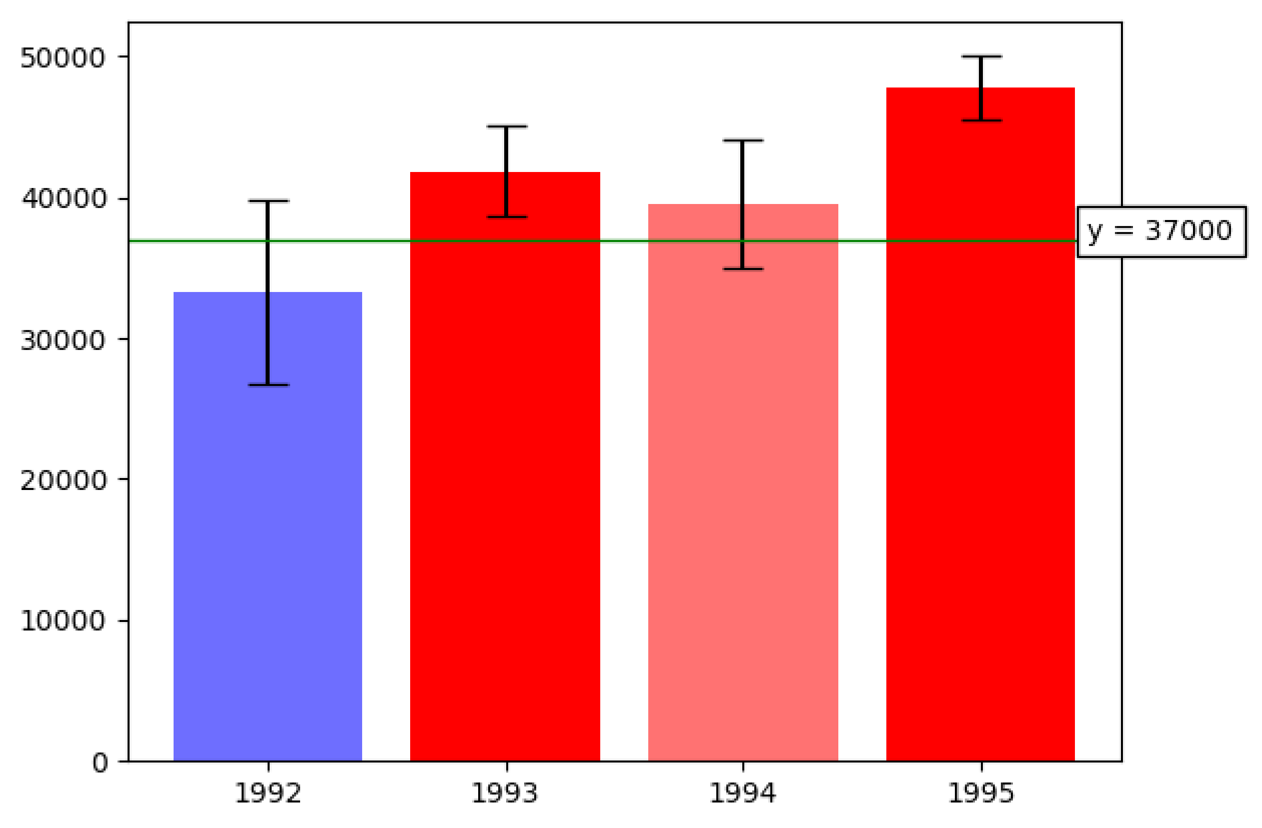
Out[3]:

	0	1	2	3	4	5	6	7	
1992	-8941.531897	127788.667612	-71887.743011	-79146.060869	425156.114501	310681.166595	50581.575349	88349.230566	18580
1993	-51896.094813	198350.518755	-123518.252821	-129916.759685	216119.147314	49845.883728	149135.648505	62807.672113	2336
1994	152336.932066	192947.128056	389950.263156	-93006.152024	100818.575896	5529.230706	-32989.370488	223942.967178	-6672
1995	-69708.439062	-13289.977022	-30178.390991	55052.181256	152883.621657	12930.835194	63700.461932	64148.489835	-2931

4 rows × 3650 columns

In [4]:

```
df_mean = df.mean(axis=1) # Averaging column entries.

# df.shape[1] gives the number of columns = 3650.
# df.std only does numerator calculation of standard deviation formula.
df_std = df.std(axis=1)/np.sqrt(df.shape[1])

# Choice of y value:
y = 37000

# In probability and statistics, 1.96 is the approximate value of the 97.5 percentile point of the norm
al distribution.
# 95% of the area under a normal curve lies within roughly 1.96 standard deviations of the mean, and du
e to the central limit theorem, this number is therefore used in the construction of approximate 95% co
nfidence intervals.
norm = Normalize(vmin=-1.96, vmax=1.96)

# 'seismic', 'coolwarm', etc. are examples of available colour palettes.
cmap = get_cmap('bwr')

df_colors = pd.DataFrame([])
df_colors['intensity'] = norm((df_mean-y)/df_std) # Usual normalising formula.
df_colors['color'] = [cmap(x) for x in df_colors['intensity']] # Assign colour depending on norm value.

# Remember we normalised df_std for assigning colour intensity earlier. Therefore the actual error will
 be scaled by 1.96.
# capsize sets thw whiskers for the error on the barplot.
bar_plot = plt.bar(df.index, df_mean, yerr=df_std*1.96, color=df_colors['color'], capsize=7);

# axhline = Horizontal line.
hoz_line = plt.axhline(y=y, color='g', linewidth=1, linestyle='-');

# Text box for chosen value. 1995.5 gives the x axis location for positioning the box.
# ec is the colour of the box border. fc is the colour of the box filling.
y_text = plt.text(1995.45, y, 'y = %d' %y, bbox=dict(fc='white',ec='k'));

# Add xticks
plt.xticks(df.index, ('1992', '1993', '1994', '1995'));

# Add interactivity
def onclick(event):
    for i in range(4):
        shade = cmap(norm((df_mean.values[i]-event.ydata)/df_std.values[i]))
        bar_plot[i].set_color(shade)
    hoz_line.set_ydata(event.ydata)
    y_text.set_text('y = %d' %event.ydata);
    y_text.set_position((1995.45, event.ydata));

plt.gcf().canvas.mpl_connect('button_press_event', onclick);
```



In [ ]: