Assignment3

January 15, 2019

1 Assignment 3 - Building a Custom Visualization

In this assignment you must choose one of the options presented below and submit a visual as well as your source code for peer grading. The details of how you solve the assignment are up to you, although your assignment must use matplotlib so that your peers can evaluate your work. The options differ in challenge level, but there are no grades associated with the challenge level you chose. However, your peers will be asked to ensure you at least met a minimum quality for a given technique in order to pass. Implement the technique fully (or exceed it!) and you should be able to earn full grades for the assignment.

Ferreira, N., Fisher, D., & Konig, A. C. (2014, April). Sample-oriented task-driven visualizations: allowing users to make better, more confident decisions. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (pp. 571-580). ACM. (video)

In this paper the authors describe 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 (replicated below in Figure 1). 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).

Figure 1 from (Ferreira et al, 2014).

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.

Figure 2c from (Ferreira et al. 2014). Note that the colorbar legend at the bottom as well as the arrows are not required in the assignment descriptions below.

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).

Note: The data given for this assignment is not the same as the data used in the article and as a result the visualizations may look a little different.

```
In [1]: # Use the following data for this assignment:

        import pandas as pd
        import numpy as np
        from scipy import stats
        import matplotlib.pyplot as plt
        import matplotlib.colors as mcol
        import matplotlib.cm as cm



        np.random.seed(12345)

        # Create data frame with random normal distributions.  Then transpose the 
        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]).T
        #df.describe()
```

2 Define functions

```
In [2]: def prepare_dataframe(df, std_err, confidence, n):
            df_description = df.describe(percentiles = [0.025, 0.975])
            df_description = df_description.T
            df_description['std_err'] = std_err
            df_description['yerr'] = stats.norm.ppf(confidence) * std_err
            return df_description


        def massage_data(df):
            data_for_graph = df
            threshold = np.mean(data_for_graph['mean'])
            lower_bound = data_for_graph['mean'] - data_for_graph['yerr']
```

```
    upper_bound = data_for_graph['mean'] + data_for_graph['yerr']
    data_for_graph['percentage'] = ((upper_bound - threshold) / (upper_bour
    f = lambda x: 0 if x['percentage'] < 0 else 1 if x['percentage'] > 1 el
    data_for_graph['percentage1'] = data_for_graph.apply(f, axis=1)
    return data_for_graph

def mygraph():
    threshold = np.mean(data_for_graph['mean'])
    percentages = list(data_for_graph['percentage1'])

    # Setup the colormap
    cmap = mcol.LinearSegmentedColormap.from_list("Cmap",["blue", "white",
    cpick = cm.ScalarMappable(cmap=cmap)
    cpick.set_array([])

    # Create figure and subplot
    fig = plt.figure(figsize = (13, 10))
    ax = fig.add_subplot(111)

    # Define range values for X axis
    xvals = range(len(data_for_graph.index))

    # Create a graph bar and add threshold to it
    bar_graph = plt.bar(xvals, data_for_graph['mean'], width = 0.8, alpha =
                        , yerr =  data_for_graph['yerr'], error_kw = {'capsize'
    ax.axhline(y=threshold, color="red")

    # Create categories in X axis
    plt.xticks( xvals, list(data_for_graph.index))

    #Create titles, labels and colorbar
    plt.xlabel('Years',fontsize= 15)
    plt.ylabel('Means', fontsize= 15)
    plt.title('Random Fake Data \n 1992 - 1995', fontsize=25)
    plt.rc('xtick', labelsize= 15)
    plt.rc('ytick', labelsize= 15)
    plt.colorbar(cpick, orientation='horizontal', boundaries=np.linspace(0,

    # Set the borders of the graph not visible
    for i in list(['top', 'right', 'bottom', 'left']):
        plt.gca().spines[i].set_visible(False)

    plt.show()

    return
```

3 Define variables

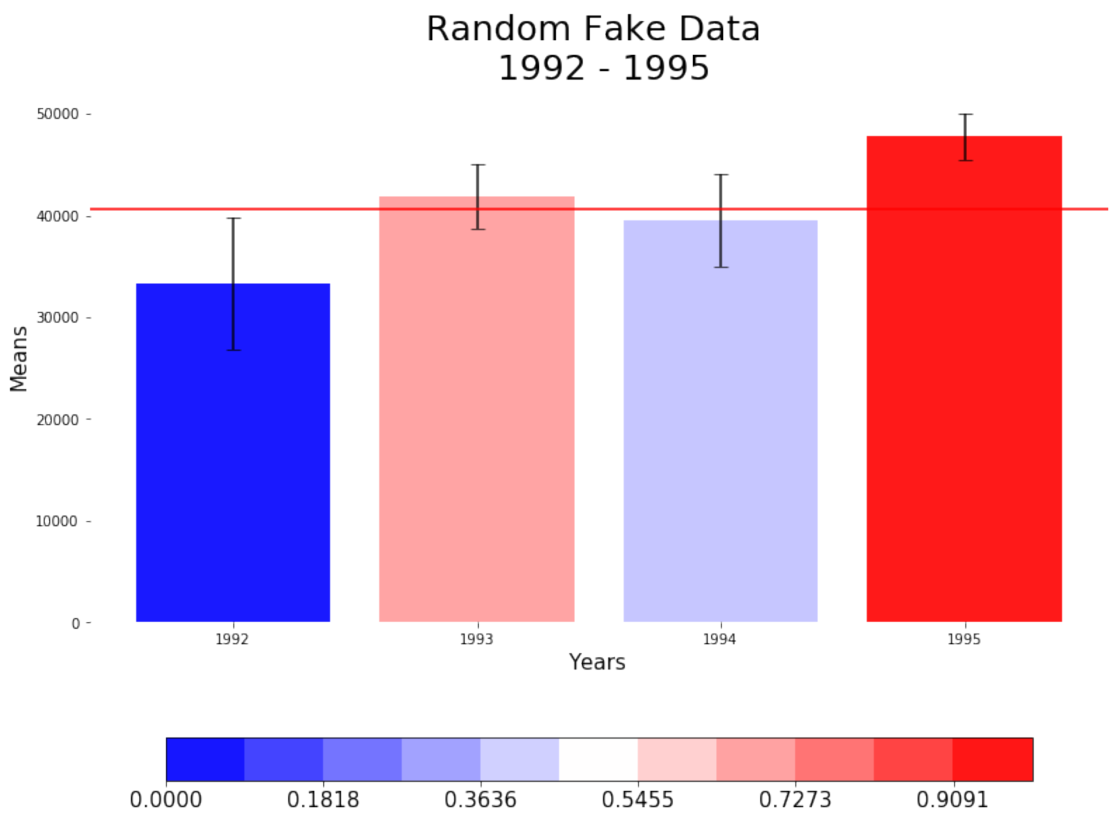
```
In [3]: #def data_for_graph(df)
        data = df
        std_err = stats.sem(df)
        confidence = 1 - 0.05 / 2
        n = len(df)
```

4 Set dataframes

```
In [4]: data_for_graph = prepare_dataframe(data, std_err, confidence, n)

        data_for_graph = data_for_graph.loc[:, :]

        data_for_graph = massage_data(data_for_graph)

        #data_for_graph
```

5 Create graph

```
In [5]: mygraph()
```


Random Fake Data
Means
Years