Fundamentals of Data Analysis - Project

Abstract

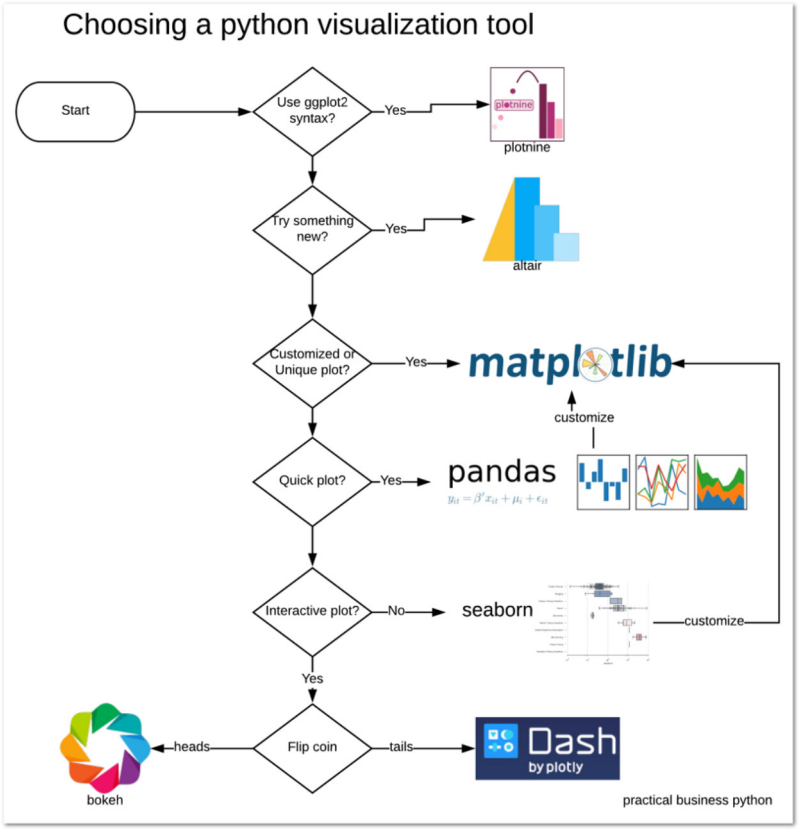
Box plot is one of the many tools used during exploratory data analysis, as it conveys a multitude of information visually. Its history stems from the work by John W. Tukey in 1969 that was further popularized by his formal publication in 1977. In this Jupyter Python notebook, the Road Accident Deaths in US States dataset is examined by box plots. Finally, a comparison of box plot to several other types of plots is also made to illustrate the advantages and disadvantages of box plot. All the plots herein are fully interactive, allowing the user to zoom in/out, rescale the axis and edit them with the Plotly Chart Studio.

Table of Contents

1. Introduction - History of Box Plot and Its Uses
2. Exploratory Data Analysis (EDA)
3. Alternatives to Box Plots
4. Discussion
5. Conclusion

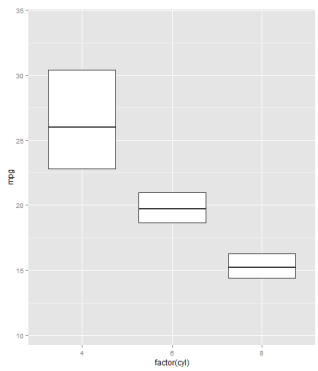
Introduction - History of Box Plot and Its Uses

Prior to performing more computationally-expensive analysis such as machine learning, it is important to visually and numerically explore the dataset in question. There are several approaches that can be taken, including boxplots (https://en.wikipedia.org/wiki/Box_plot), violin plots, correlation matrices, histograms and kernel density estimates - these are not mutually exclusive and can be achieved by simple commands. These approaches can be further augmented by interactive plots, enabled by Plotly and Bokeh libraries. The figure below summarizes the different Python plotting libraries.



Taken from PbPython (http://pbpython.com/python-vis-flowchart.html)

As the name suggests, box plots consist of a rectangle with a band within, as shown by the figure below.



Taken from a StackOverflow post (https://stackoverflow.com/questions/18459287/remove-whiskers-in-box-whisker-plot)

More often that not, this rectangle is also extended by lines ('whiskers'), leading these plots to be termed box and whisker plots. The anatomy of a box

In [1]:
```
import pandas as pd
df = pd.read_csv('road.csv')

# to make interactive plots with plotly
import plotly.figure_factory as ff
import plotly.graph_objs as go
from plotly.offline import download_plotlyjs, init_notebook_mode, iplot
init_notebook_mode(connected=True)
from plotly import tools
```

Exploratory Data Analysis (EDA)

We will first look at the data structure and make modifications as required.

In [2]:
```
df.head() # check to ensure reading the csv file worked as intended
```

Out[2]:

	Unnamed: 0	deaths	drivers	popden	rural	temp	fuel
0	Alabama	968	158	64.0	66.0	62	119.0
1	Alaska	43	11	0.4	5.9	30	6.2
2	Arizona	588	91	12.0	33.0	64	65.0
3	Arkanas	640	92	34.0	73.0	51	74.0
4	Calif	4743	952	100.0	118.0	65	105.0

From above, we can determine that we have 7 columns, with the first being a character/text column and the rest are numeric.

For some reason, the first column, which contains the names of the states in the United States of America, is not labelled. The other column headers are not really descriptive, so we will rename these first. Additionally, the state 'Arkanas' is a mispelling of Arkansas - probably there are other mistakes as well.

In [3]:
```
# rename columns based on information from the source webpage (not all needs renaming)
# Columns are:
#
# state
# deaths - number of deaths.
# drivers - number of drivers (in 10,000s).
# popden - population density in people per square mile.
# rural - length of rural roads, in 1000s of miles.
# temp - average daily maximum temperature in January.
# fuel - fuel consumption in 10,000,000 US gallons per year.

df.rename(columns={'Unnamed: 0': 'State',
                   'deaths': 'Deaths',
                   'drivers': 'Number of drivers (10,000)',
                   'popden': 'Population density (people/mile**2)',
                   'rural': 'Length of rural roads (1000 miles)',
                   'temp': 'Average daily max temperature in Jan',
                   'fuel': 'Fuel consumption (10**7 US gal/year)'
                  }, inplace = True)
```

Looking at the State column, we can see a mixture of full and abbreviated state names. As not everyone is familiar with the abbreviations, these states will also be renamed. For example, Miss could be interpreted as either Mississippi or Missouri. Fortunately, standardized abbreviations are available (https://en.wikipedia.org/wiki/List_of_U.S._state_abbreviations).

In [4]:
```
# https://stackoverflow.com/questions/44218091/
df.loc[df['State'] == 'Calif', 'State'] = 'California'
df.loc[df['State'] == 'Colo', 'State'] = 'Colorado'
df.loc[df['State'] == 'Conn', 'State'] = 'Connecticut'

df.loc[df['State'] == 'Dela', 'State'] = 'Delaware'
df.loc[df['State'] == 'DC', 'State'] = 'Washington DC'
df.loc[df['State'] == 'Ill', 'State'] = 'Illinois'
df.loc[df['State'] == 'Ind', 'State'] = 'Indiana'

df.loc[df['State'] == 'Kent', 'State'] = 'Kentucky'
df.loc[df['State'] == 'Louis', 'State'] = 'Louisiana'
df.loc[df['State'] == 'Maryl', 'State'] = 'Maryland'
df.loc[df['State'] == 'Mass', 'State'] = 'Massachusetts'

df.loc[df['State'] == 'Mich', 'State'] = 'Michigan'
df.loc[df['State'] == 'Minn', 'State'] = 'Minnesota'
df.loc[df['State'] == 'Miss', 'State'] = 'Mississippi'
df.loc[df['State'] == 'Mo', 'State'] = 'Missouri'
df.loc[df['State'] == 'Mont', 'State'] = 'Montana'

# State Arkanas is spelled incorrectly
df.loc[df['State'] == 'Arkanas', 'State'] = 'Arkansas'
```

Before proceeding further, we should inspect the entire dataset first to ensure all the mistakes and column renaming operations above were completed correctly and that there are no missing values.

In [5]:
```
df
```

Out[5]:

	State	Deaths	Number of drivers (10,000)	Population density (people/mile**2)	Length of rural roads (1000 miles)	Average daily max temperature in Jan	Fuel consumption (10**7 US gal/year)
0	Alabama	968	158	64.0	66.0	62	119.0
1	Alaska	43	11	0.4	5.9	30	6.2
2	Arizona	588	91	12.0	33.0	64	65.0
3	Arkansas	640	92	34.0	73.0	51	74.0
4	California	4743	952	100.0	118.0	65	105.0
5	Colorado	566	109	17.0	73.0	42	78.0
6	Connecticut	325	167	518.0	5.1	37	95.0
7	Delaware	118	30	226.0	3.4	41	20.0
8	Washington DC	115	35	12524.0	0.0	44	23.0
9	Florida	1545	298	91.0	57.0	67	216.0
10	Georgia	1302	203	68.0	83.0	54	162.0
11	Idaho	262	41	8.1	40.0	36	29.0
12	Illinois	2207	544	180.0	102.0	33	350.0
13	Indiana	1410	254	129.0	89.0	37	196.0
14	Iowa	833	150	49.0	100.0	30	109.0
15	Kansas	669	136	27.0	124.0	42	94.0
16	Kentucky	911	147	76.0	65.0	44	104.0
17	Louisiana	1037	146	72.0	40.0	65	109.0
18	Maine	1196	46	31.0	19.0	30	37.0
19	Maryland	616	157	314.0	29.0	44	113.0
20	Massachusetts	766	255	655.0	17.0	37	166.0
21	Michigan	2120	403	137.0	95.0	33	306.0
22	Minnesota	841	189	43.0	110.0	22	132.0
23	Mississippi	648	85	46.0	59.0	57	77.0
24	Missouri	1289	234	63.0	100.0	40	180.0
25	Montana	259	38	4.6	72.0	29	31.0

As we can see above, the column labels are more descriptive. Because the dataset is quite small, no memory optimization/downcast operations are needed.

We can now start building a simple box plot.

In [6]:

```
# all traces are labelled by the State column for easier understanding
# we'll also draw mean lines (dashed)
trace1 = go.Box(y=df.iloc[:,1], marker = dict(size=3), text = df['State'], boxpoints='all', boxmean = T
rue)
trace2 = go.Box(y=df.iloc[:,2], marker = dict(size=3), text = df['State'], boxpoints='all', boxmean = T
rue )
trace3 = go.Box(y=df.iloc[:,3], marker = dict(size=3), text = df['State'], boxpoints='all', boxmean = T
rue )
trace4 = go.Box(y=df.iloc[:,4], marker = dict(size=3), text = df['State'], boxpoints='all', boxmean = T
rue )
trace5 = go.Box(y=df.iloc[:,5], marker = dict(size=3), text = df['State'], boxpoints='all', boxmean = T
rue )
trace6 = go.Box(y=df.iloc[:,6], marker = dict(size=3), text = df['State'], boxpoints='all', boxmean = T
rue )

# we have to make subplots because the scales are different for the variables
fig = tools.make_subplots(rows=2, cols=3, print_grid=False,
                          subplot_titles= (df.columns[1], df.columns[2], df.columns[3],
                                           df.columns[4], df.columns[5], df.columns[6]
                                                                         ))

fig.append_trace(trace1, 1, 1)
fig.append_trace(trace2, 1, 2)
fig.append_trace(trace3, 1, 3)
fig.append_trace(trace4, 2, 1)
fig.append_trace(trace5, 2, 2)
fig.append_trace(trace6, 2, 3)

fig['layout'].update(height=800, width=800, title='<b>Figure 1: Box Plot of the Road Accidents dataset<
/b>'
                     )
fig['layout'].update(showlegend=False)

# https://github.com/plotly/plotly.py/issues/985
for i in fig['layout']['annotations']:
    i['font'] = dict(size=10)

for prop in fig.layout:
    if prop.startswith('xaxis'):
        # don't really need both xaxis label and subplot title, so turn xaxis label off
        fig.layout[prop].showticklabels = False

iplot(fig)
```

Figure 1: Box Plot of the Road Accidents dataset

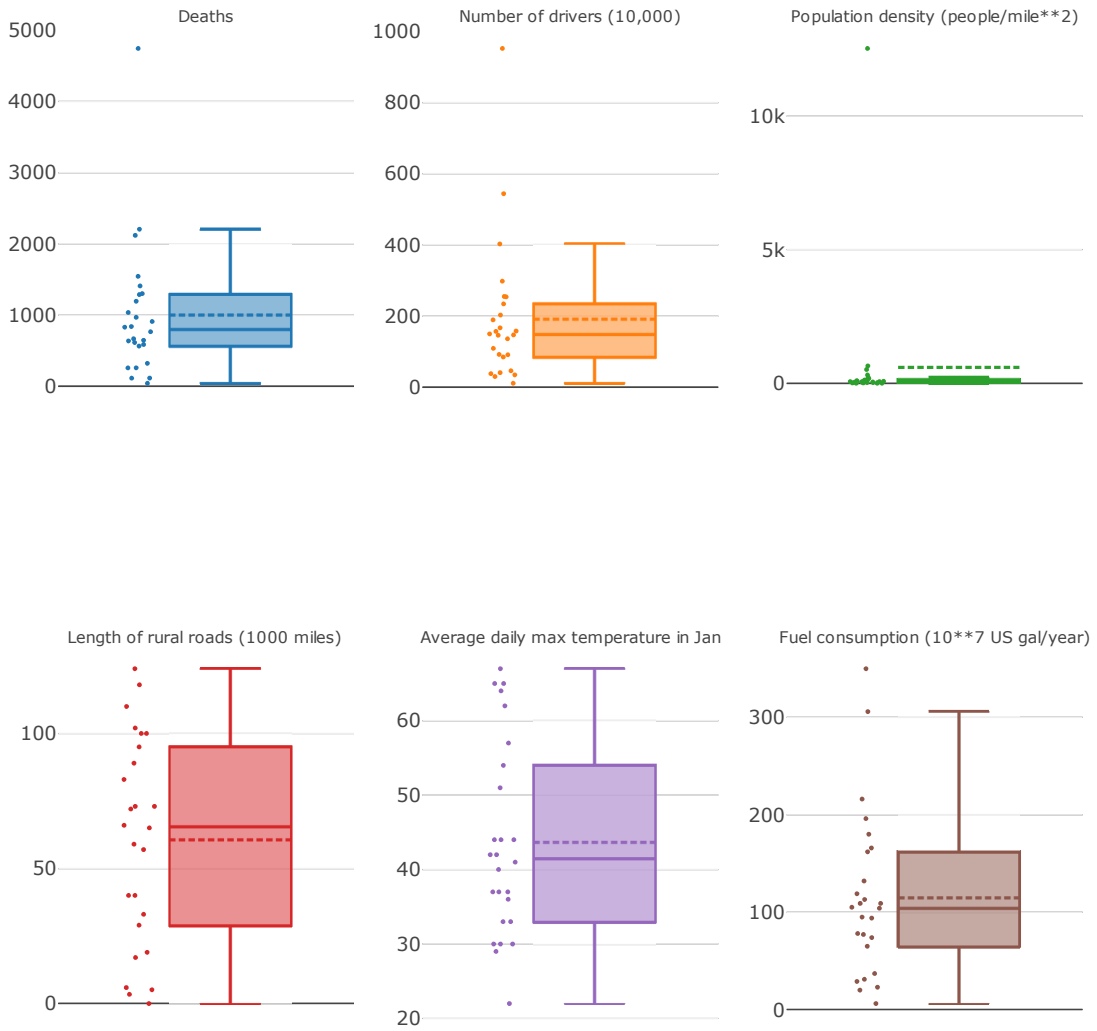


Figure 1 shows that for most of the variables, there is at least one point in each that extends beyond the whiskers (outliers). Interestingly, however, the length of rural roads and the average daily max temperature in January are evenly spread (probably normally distributed).

From this, we can make the following observations:

1. The state of California has an extremely high number of deaths compared to the other states.
2. There are significantly more drivers in California and Illinois than the other states, which could partially explain the higher number of deaths.
3. Washington, DC, which is a federal district and the capital of the US, has a significantly high population density. This is probably caused by the small land area with a concentration of foreign diplomats living in the capital.
4. The fuel consumption in Illinois is at the higher extreme end for unknown reasons.
5. With the exception of the population density, all the variables have their average values (mean, dashed line) close to their medians.

The same information can also be tabulated through the Pandas `describe()` function, which can be useful for programmatic access to the descriptive statistics.

In [7]:
```
# http://www.datasciencemadesimple.com/format-integer-column-of-dataframe-in-python-pandas/
pd.options.display.float_format = '{:,.2f}'.format # round to the 2 decimal places
df.describe()
```

Out[7]:

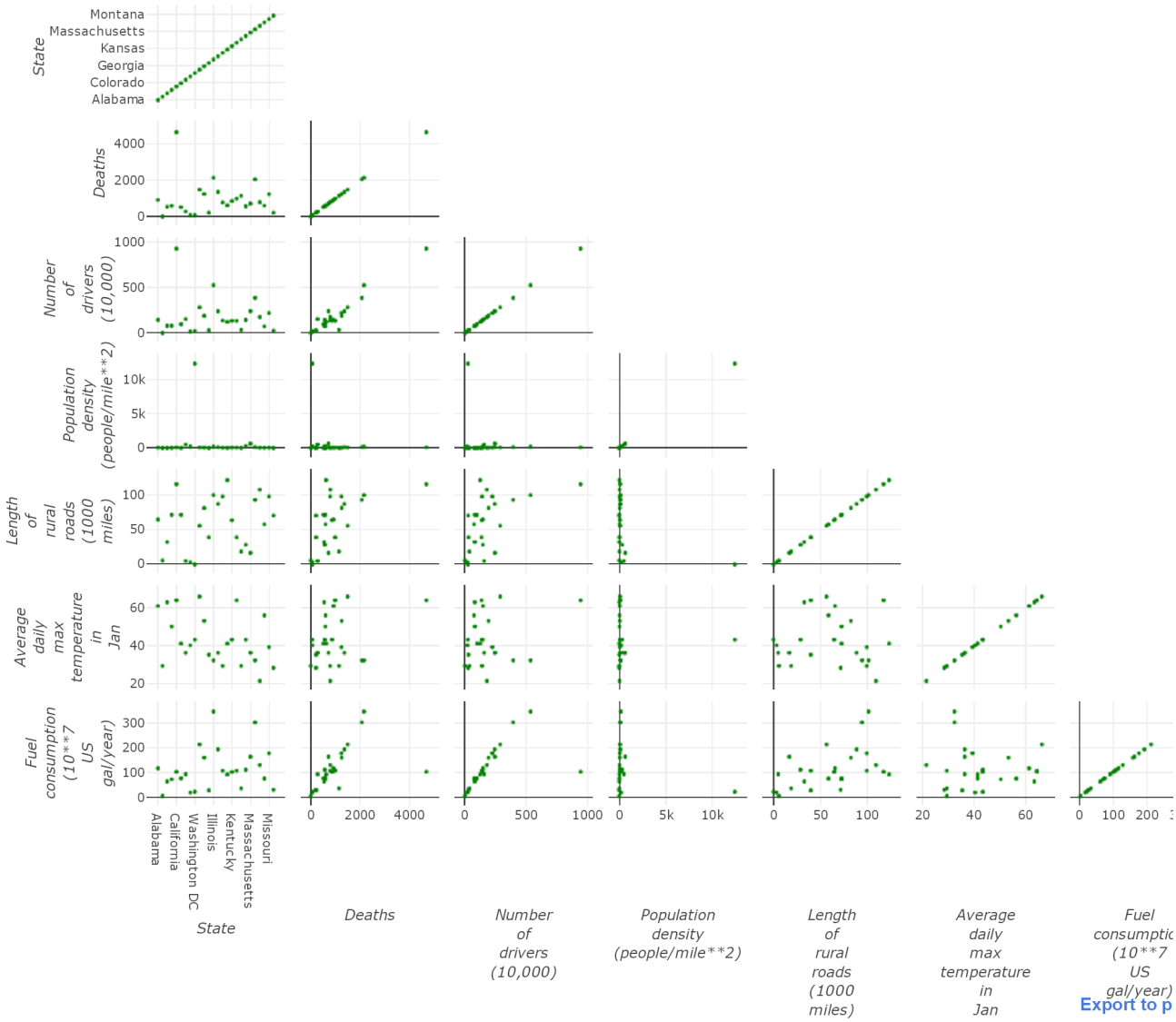
	Deaths	Number of drivers (10,000)	Population density (people/mile**2)	Length of rural roads (1000 miles)	Average daily max temperature in Jan	Fuel consumption (10**7 US gal/year)
count	26.00	26.00	26.00	26.00	26.00	26.00
mean	1,000.65	191.19	595.73	60.71	43.69	115.24
std	946.84	196.88	2,437.95	38.38	13.01	83.86
min	43.00	11.00	0.40	0.00	22.00	6.20
25%	571.50	86.50	31.75	30.00	33.75	67.25
50%	799.50	148.50	66.00	65.50	41.50	104.50
75%	1,265.75	226.25	135.00	93.50	53.25	154.50
max	4,743.00	952.00	12,524.00	124.00	67.00	350.00

To determine the correlations between these variables, a scatterplot matrix can be drawn.

In [8]:

```
# https://community.plot.ly/t/splom-scatter-matrix-changing-styles-of-all-axes-in-one-go/16636/2
# https://plot.ly/python/splom/

# initiate placeholders
dataPanda = []
label = ''

for column in list(df.columns):

    columnSplit = column.split(' ') # create a list of string from the column name

    for i in columnSplit:
        label = label + i + '<br>' # plotly uses JavaScript, we can use the html <br> tag to split to n
ext line

    # inspired by: https://stackoverflow.com/questions/43229013/
    for i in range(3):
        label = '<br>' + label + '<br>'

    label = '<i>' + label + '</i>'
    trace = dict(label = label, values = df[column])

    dataPanda.append(trace)
    label = '' # reset the variable for next for iteration

trace1 = go.Splom(dimensions=dataPanda, showupperhalf=False, marker=dict(size=3, color='green'),
                  text = df['State'])
data = [trace1]

layout = go.Layout(title = '<b>Figure 2: Scatterplot Matrix of the Road Accidents dataset</b>',
                font=dict(size=8.5), # global font
                titlefont=dict(size=20), # but we want only title to have different size
                # layout control- left, right, bottom, top (t) and padding (pad )
                margin = dict(l = 150, r = 20, b = 150)
                )

fig = go.Figure(data=data, layout=layout)
fig['layout'].update(height=800, width=900)

iplot(fig)
```

Figure 2: Scatterplot Matrix of the Road Accidents dataset

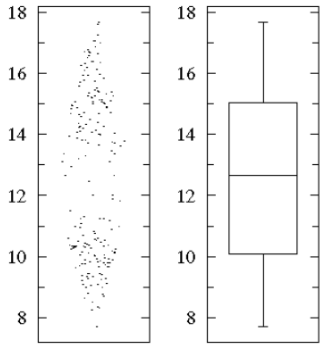


As shown in Figure 2, there are strong positive correlations between the number of deaths and the number of drivers, length of rural roads (presumably these have lower quality surfacing and lighting), average daily maximum temperature in January and fuel consumption. However, it should be noted that strong correlations do not mean causation.

Alternatives to Box Plots

Although it seems the box plots are useful for summarizing a given dataset, it is by itself insufficient to capture the full content of the dataset.

For instance, a bimodal dataset with most values being 10 and 15 will not be represented as such in a box plot:



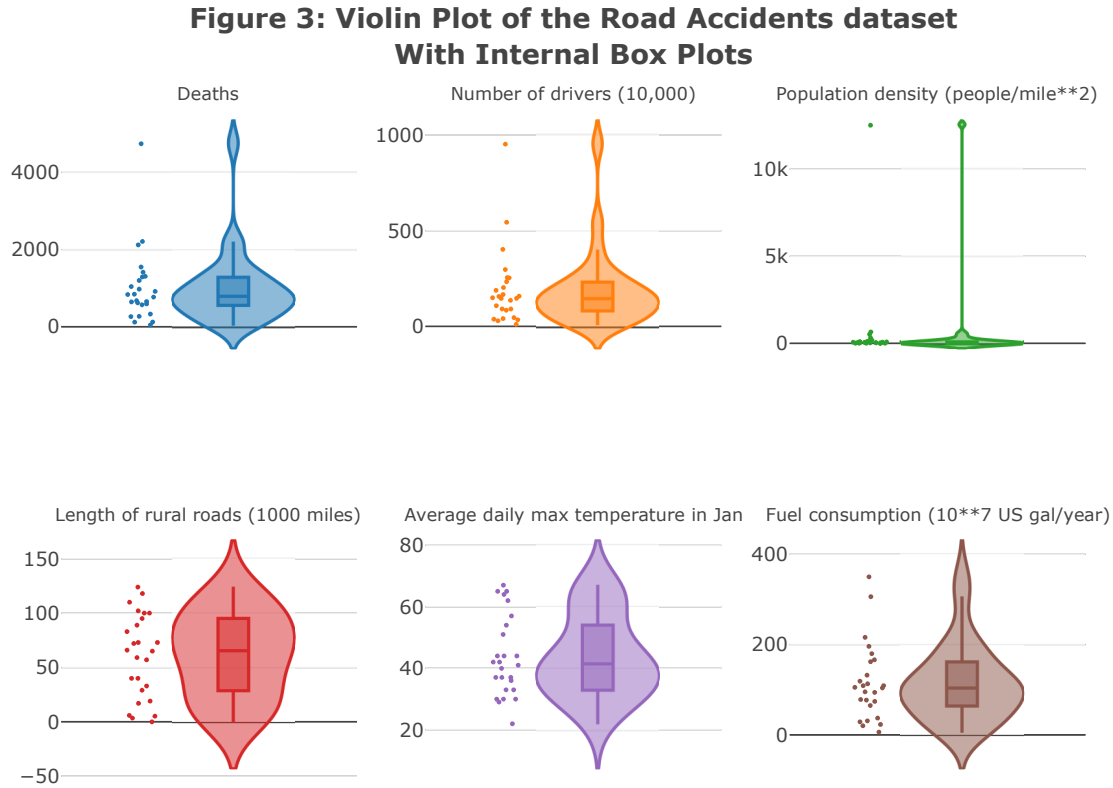
Taken from Statistics to Use (http://www.physics.csbsju.edu/stats/box2.html)

As a box plot does not show the data distribution, modifications to the classical box plots have been proposed. However, their usage within scientific publications is still rather limited.

Violin Plot and Histogram

The most directly comparable alternative to box plot is the violin plot (https://en.wikipedia.org/wiki/Violin_plot). Although rarely used, it is actually more informative than box plots but require some non-intuitive interpretation. Plotly provides a wrapper function to create violin plots (https://plot.ly/python/violin/), which is useful in explaining the differences with respect to the Road Accidents dataset.

In [9]:
```
data = []

for i in range(1, len(df.columns)):
    trace = go.Violin(y=df.iloc[:,i], text = df['State'], points = 'all', marker = dict(size=3),
                      box=dict(visible=True))
    data.append(trace)

# we have to make subplots because the scales are different for the variables
fig = tools.make_subplots(rows=2, cols=3, print_grid=False,
                          subplot_titles= (df.columns[1], df.columns[2], df.columns[3],
                                           df.columns[4], df.columns[5], df.columns[6]
                                                                         ))

fig.append_trace(data[0], 1, 1)
fig.append_trace(data[1], 1, 2)
fig.append_trace(data[2], 1, 3)
fig.append_trace(data[3], 2, 1)
fig.append_trace(data[4], 2, 2)
fig.append_trace(data[5], 2, 3)

fig['layout'].update(height=600, width=800, title='<b>Figure 3: Violin Plot of the Road Accidents datas
et<br>' +
                     'With Internal Box Plots</b>')
fig['layout'].update(showlegend=False)

# https://github.com/plotly/plotly.py/issues/985
for i in fig['layout']['annotations']:
    i['font'] = dict(size=11)

for prop in fig.layout:
    if prop.startswith('xaxis'):
        # don't really need both xaxis label and subplot title, so turn xaxis label off
        fig.layout[prop].showticklabels = False

iplot(fig)
```



Export to plot.ly »

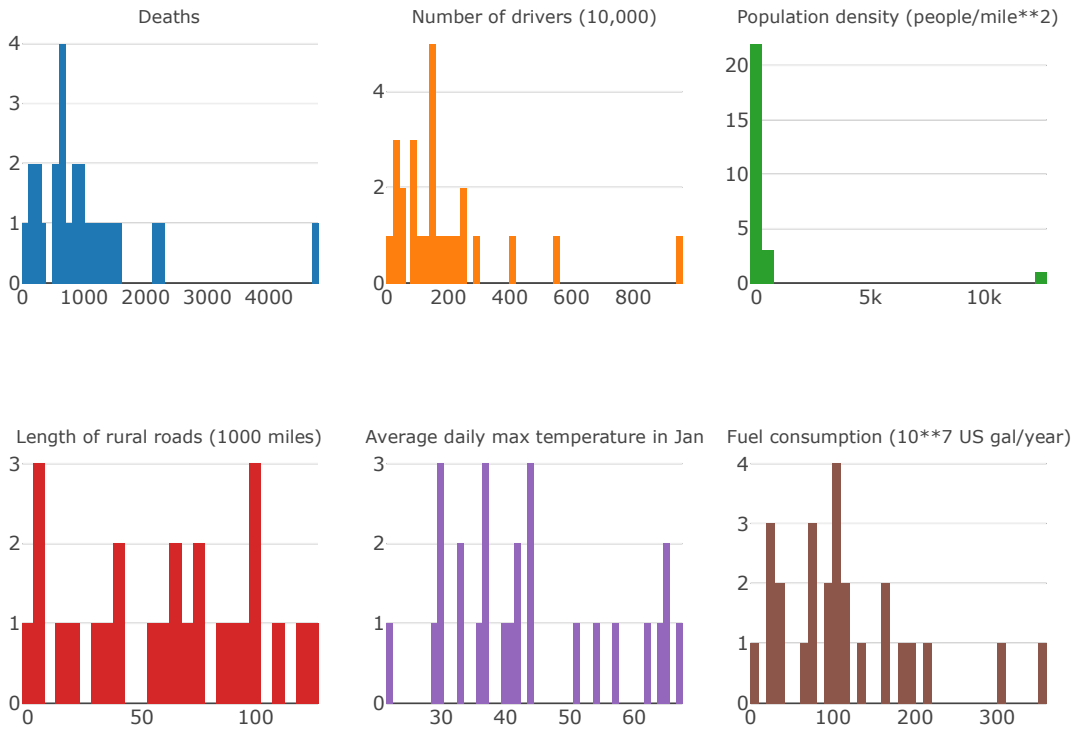
Compared to the box plot, in addition to the five-number summary (median, the two quartiles, minimum/maximum, whiskers and outliers), we also obtain the kernal density estimate (kde) in a violin plot. This means that the thickest section corresponds to the data mode (the x value with the highest occurence/frequency). In this case, most states have about 700 deaths caused by road accidents and about 1,000,000 (100 x 10000) drivers - note that no two states have the exactly same values, so these numbers are midpoints of where most of the values cluster together. Similarly, the length of rural roads are distributed evenly among the states.

Violin plots are actually similar to vase plots, except the former use all the data to create the density curve whereas vase plots only use the central 50 %, leading to the presence of whiskers and outliers (reference) (https://pdfs.semanticscholar.org/d315/05d5b6d61ad75a5ae6ded8fa5b7202e66372.pdf). Secondly, violin plots use kde while vase plots employ non-parametric density estimation.

KDE is a much better alternative to the classical histogram, mainly as the latter requires binning leading to loss of subtle features in the dataset. In fact, in the example below, the parameter `nbinsx` had to be manually determined for the best display of the data because the default estimation algorithm is not great. An online interactive tool demonstrating the pitfalls of binning is available here (https://mglerner.github.io/posts/histograms-and-kernel-density-estimation-kde-2.html).

In [10]:
```
data = []

for i in range(1, len(df.columns)):
    trace = go.Histogram(x=df.iloc[:,i], text = df.State, nbinsx = 50, name=df.columns[i]
                         )
    data.append(trace)

# we have to make subplots because the scales are different for the variables
fig = tools.make_subplots(rows=2, cols=3, print_grid=False,
                          subplot_titles= (df.columns[1], df.columns[2], df.columns[3],
                                           df.columns[4], df.columns[5], df.columns[6]
                                                                         ))

fig.append_trace(data[0], 1, 1)
fig.append_trace(data[1], 1, 2)
fig.append_trace(data[2], 1, 3)
fig.append_trace(data[3], 2, 1)
fig.append_trace(data[4], 2, 2)
fig.append_trace(data[5], 2, 3)

fig['layout'].update(height=600, width=800, title='<b>Figure 4: Histogram of the Road Accidents dataset
<br>')
fig['layout'].update(showlegend=False)

# https://github.com/plotly/plotly.py/issues/985
for i in fig['layout']['annotations']:
    i['font'] = dict(size=11)

iplot(fig)
```

Figure 4: Histogram of the Road Accidents dataset



Export to plot.ly »

Other Variants of Box Plots

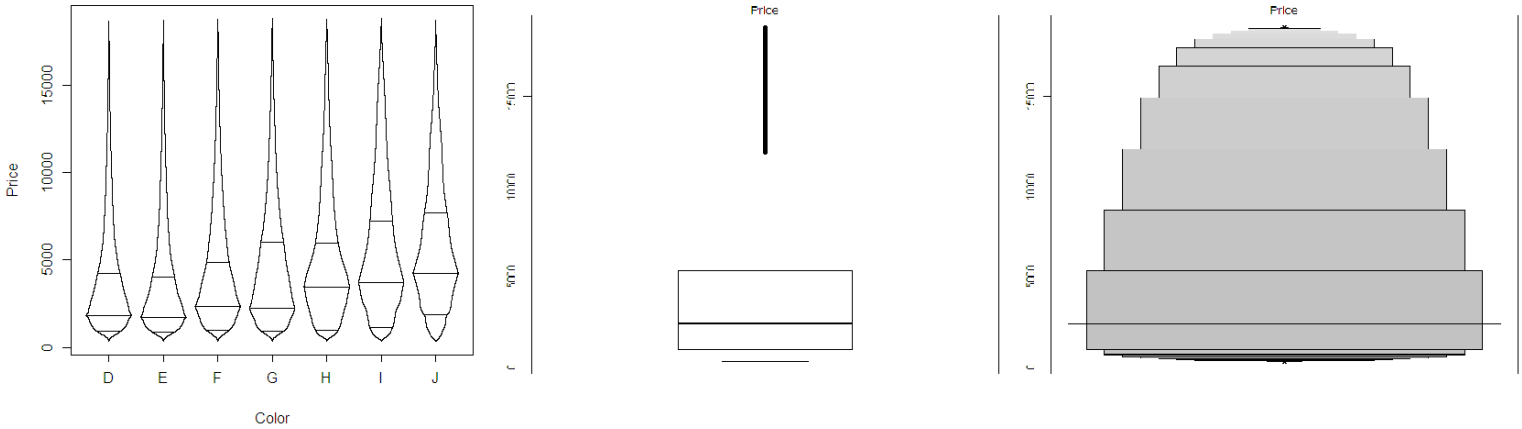
When the box plot was first published in 1970, there was no sophisticated graphical software or libraries for creating plots with non-rectangular shapes. In this era, however, there is no reason to not include additional shapes (http://datavizcatalogue.com/blog/box-plot-variations/). On the other hand, widespread use of multiple types of plots might confuse non-statistics educated readers of scientific articles who will to contend with the exact information each plot is trying to convey.

Box-percentile plot

This type of plot does not use kde to plot the density curve.

Letter-value plot

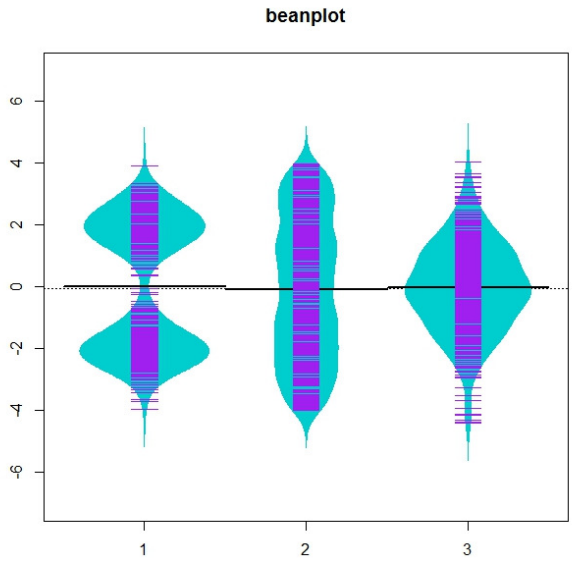
More useful for large datasets as the behavior in the tails are unreliable for moderate ($n < 1000$) datasets.



Left: Box-percentile plots of color vs. price. **Middle:**The original boxplots of color vs. price. **Right:** Letter-value boxplots showing the extreme values (> 10000) more clearly than boxplots. Figures taken from Lisa Stryjewski (2010). (https://www.semanticscholar.org/paper/40-years-of-boxplots-Stryjewski/d31505d5b6d61ad75a5ae6ded8fa5b7202e66372)

Bean plot (https://www.semanticscholar.org/paper/40-years-of-boxplots-Stryjewski/d31505d5b6d61ad75a5ae6ded8fa5b7202e66372)

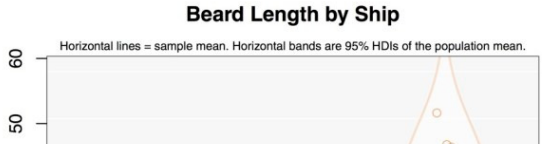
Similar to a violin plot, with the addition of individual data points.



Taken from the R Graph Gallery. (http://rgraphgallery.blogspot.com/2013/04/rg-beanplot.html)

Pirate plot (http://rgraphgallery.blogspot.com/2013/04/rg-beanplot.html)

This plot is also similar to a bean plot with a central line indicating the mean and horizontal bands representing the 95 % highest density intervals (HDIs) of the population mean. HDIs are similar to confidence intervals with the added advantage of allowing side-by-side comparison of means when there are multiple groups within a dataset. In the statistics programming language R, the Pirate Plot uses another package [BEST (Bayesian Estimation Supersedes the T-Test)] to generate the HDIs.

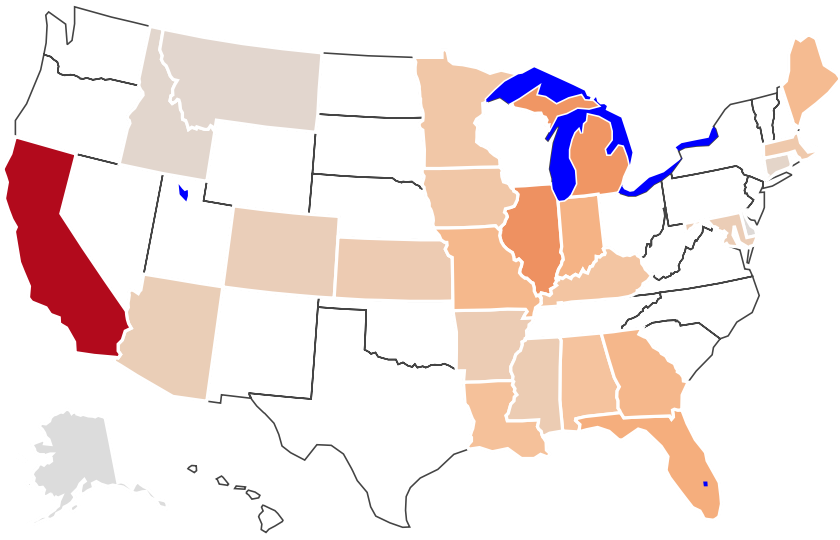


In [11]:

```
# modified from https://stackoverflow.com/questions/48979352/choropleth-map-in-python-using-plotly-without-state-codes
state_codes = {
    'Washington DC' : 'dc','Mississippi': 'MS', 'Oklahoma': 'OK',
    'Delaware': 'DE', 'Minnesota': 'MN', 'Illinois': 'IL', 'Arkansas': 'AR',
    'New Mexico': 'NM', 'Indiana': 'IN', 'Maryland': 'MD', 'Louisiana': 'LA',
    'Idaho': 'ID', 'Wyoming': 'WY', 'Tennessee': 'TN', 'Arizona': 'AZ',
    'Iowa': 'IA', 'Michigan': 'MI', 'Kansas': 'KS', 'Utah': 'UT',
    'Virginia': 'VA', 'Oregon': 'OR', 'Connecticut': 'CT', 'Montana': 'MT',
    'California': 'CA', 'Massachusetts': 'MA', 'West Virginia': 'WV',
    'South Carolina': 'SC', 'New Hampshire': 'NH', 'Wisconsin': 'WI',
    'Vermont': 'VT', 'Georgia': 'GA', 'North Dakota': 'ND',
    'Pennsylvania': 'PA', 'Florida': 'FL', 'Alaska': 'AK', 'Kentucky': 'KY',
    'Hawaii': 'HI', 'Nebraska': 'NE', 'Missouri': 'MO', 'Ohio': 'OH',
    'Alabama': 'AL', 'Rhode Island': 'RI', 'South Dakota': 'SD',
    'Colorado': 'CO', 'New Jersey': 'NJ', 'Washington': 'WA',
    'North Carolina': 'NC', 'New York': 'NY', 'Texas': 'TX',
    'Nevada': 'NV', 'Maine': 'ME'}

df['state_code'] = df['State'].apply(lambda x : state_codes[x])
```

In [12]:
```
# concatenate string with numeric columns:
# https://stackoverflow.com/questions/11858472/pandas-combine-string-and-int-columns
# https://stackoverflow.com/questions/50782934/what-is-difference-between-mapstr-and-astypestr-in-dataf
rame
# we use PEP8 recommendation to link multi-line codes in parentheses

df['text'] = (
            df['State'] + '<br>' + df.columns[1] + ': ' + df.iloc[:, 1].astype(str) +
                          '<br>' + df.columns[2] + ': ' + df.iloc[:, 2].astype(str) +
                          '<br>' + df.columns[3] + ': ' + df.iloc[:, 3].astype(str) +
                          '<br>' + df.columns[4] + ': ' + df.iloc[:, 4].astype(str) +
                          '<br>' + df.columns[5] + ': ' + df.iloc[:, 5].astype(str) +
                          '<br>' + df.columns[6] + ': ' + df.iloc[:, 6].astype(str)
            )

data = [ dict(
        type='choropleth',
       # colorscale = 'Reds',
        autocolorscale = True,
        hoverinfo = 'text', # change from default 'all' so that we don't display to z values on top
        locations = df['state_code'],
        z = df['Deaths'],
        locationmode = 'USA-states',
        text = df['text'],
        marker = dict(
            line = dict (
                color = 'rgb(255,255,255)',
                width = 2
            ) ),
        colorbar = dict(
            title = '<b>Deaths</b>')
        ) ]

layout = dict(
        title = '<b>Figure 5: Choropleth Map of Road Accidents in the United States</b><br><i>(Hover fo
r breakdown)</i>',
        geo = dict(
            scope='usa',
            projection=dict( type='albers usa' ),
            showlakes = True,
            lakecolor = 'rgb(0, 0, 255)'), # lakes are in blue
             )

fig = dict(data=data, layout=layout)
iplot(fig)
```

Figure 5: Choropleth Map of Road Accidents in the United States
(Hover for breakdown)



From the map, we can see that California is quite a large state, a piece of information that could not have been gleaned by the more generic box or violin plots. Perhaps due to this, there is a high number of drivers and deaths. Secondly, we note that there are more eastern states in the dataset than the western states. The large blue sections on the northeastern region are the Great Lakes (https://en.wikipedia.org/wiki/Great_Lakes).

Discussion

By using box and violin plots, we have been able to understand the distribution of the Road Accidents data and relationship among the variables in a visual way. From these two plots, the state of California is determined to have a significantly higher number of deaths and number of drivers than any other states. However, a more general conclusion associated any one of the variables cannot be made without other pieces of information. For example, many of the Californian drivers could be truck drivers (https://www.randallreilly.com/where-are-the-drivers-a-state-by-state-heat-map-analysis/) who are legally more experienced than younger car drivers and would be expected to be more careful on the road. Moreover, although the size of California is similar to the combined area of Louisiana, Arkansas and Missouri, the total deaths are 4743 *vs.* 2966, respectively. The length of rural roads and the average daily maximum temperature in January seems to have no correlation with the number of deaths. Taken together, these observations suggest the potential involvement of other (unknown) factors.

Conclusion

We have seen how useful the box plot is, how it can be augmented by a violin plot and in certain cases, even a geographical map. Good and correct visualization is important to convey information in a quick manner especially to an audience consisting of people not trained in data science. At the same, presenting the actual data points (untainted by 'fancy' manipulations) can lend more credence to the conclusions drawn from a set of plots.