Introduction to Statistical Graphics – Jupyter Notebook

Data Exploration and Visualization

Introduction

Welcome to Data Exploration and Visualization.
This presentation serves as a walk-through and detailed explanation of the provided Example 2 Jupyter Notebook.
This Notebook loads a database of cereals and various properties about the cereals and then generate graphs of these properties using the MatPlotLib library.
Here, we will step through the example code and explain each block’s function.

Requirements

In order to successfully run the code provided in this example, two additional packages need to be installed, `matplotlib`, a graphing package, and `pandas`, a data-processing package.

On OSX, these packages can be installed by entering `pip3 install matplotlib pandas` in a Terminal window. If you have already started Jupyter Notebook, you must restart the server for the packages to be picked up.

On Windows, these packages can be installed by opening the Anaconda Prompt application that was installed with Juypter Notebook and running the command: `conda install matplotlib pandas`

Graphs

The four graphs generated in this example are:

1. Pie Chart
2. Bar Chart
3. Histogram
4. Box-and-Whisker Plot

Cells

In the Notebook provided for this example, the code to generate each of the graphs is in its own *cell*. This allows you to generate one graph at a time and modify one graph’s code without affecting the others. In order to set up the graphing environment, you must run the code in the first cell each time you open the notebook and before attempting to run any graph code.

Part 1

```
import sqlite3
import pandas as pd
import matplotlib.pyplot as plt

db_filename = 'cereals.db'

conn = sqlite3.connect(
    db_filename)
c = conn.cursor()
```

This block of code includes the libraries we'll need for the rest of the example. In this example, we need the sqlite3 and pandas libraries and parts of the matplotlib library, which will be used for graphing. In addition, we load the database and create a cursor so we can extract the data to be graphed.

Part 2: Graph #1

```
c.execute("SELECT
    Manufacturer,count(*) FROM
   cereals GROUP BY Manufacturer")
counts = c.fetchall()
```

This cell generates and shows a Pie chart representing the distribution of cereals by manufacturer. These two lines request the data to be graphed by asking the database to count the number of records for each manufacturer.

Part 3: Graph #1

```
manuStats = pd.DataFrame.
    from_records(counts, columns
    =['manufacturer', 'value'])
```

In this line, the data to be graphed is converted to a DataFrame from the records returned by the database. This pseudo-tabular format is used by the bokeh graphing functions. Using `from_records()` is an easy way to construct this format from database records.

Part 4: Graph #1

```
plt.pie(manuStats['value'],
   labels='manufacturer',
   shadow=False)
plt.axis('equal')
plt.show()
```

In the first of these lines, the graph is created. Specifying the ‘value’ column from the DataFrame created earlier indicates which column of data to graph. Similarly, the ‘label’ column applies a label to each slice of the pie. Finally, the `plt.show()` function renders the graph in the notebook.

Part 5: Graph #2

```
c.execute("SELECT Cereal,
   Sugars FROM cereals")
sugars = c.fetchall()

sugarFrame = pd.DataFrame.
    from_records(sugars,
    columns=['Cereal',
    'Sugar'])
```

This cell generates a bar chart showing each cereal’s sugar content. These lines request pairs of sugar content and cereal name from the database and format the data for display in a bokeh graph.

Part 6: Graph #2

```
plt.bar(
    range(len(
    sugarFrame['Sugar']
    )),
    sugarFrame['Sugar'])
plt.xticks([])
plt.show()
```

These two lines again create and then render a graph in the notebook. In this case, the graph is a bar graph showing the values of the ‘Sugars’ field for each of the named cereals.

Part 7: Graph #3

```
c.execute("SELECT Sugars
   FROM cereals")
sugar = c.fetchall()

sugarFrame = pd.DataFrame.
    from_records(sugar,
    columns=['Sugar'])
```

This cell generates a histogram showing the distribution of sugar content over the dataset. In this case, we need only the sugars values from each record, so our query requests only those. As above, the last line formats the data for use with bokeh.

Part 8: Graph #3

```
plt.hist(
    sugarFrame['Sugar'],
    bins=9)
plt.show()
```

Here, we generate the actual sugar values Histogram. The bokeh package’s built-in histogram can perform all of the histogram computations, but here we specify the number of bins to comply with the √N rule. The final line renders the graph in the Notebook.

Part 9: Graph #4

```
c.execute("SELECT
    Manufacturer,Sugars FROM
   cereals")
sugarByMan = c.fetchall()

sugarBoxFrame =pd.DataFrame.
    from_records(sugarByMan,
    columns=['Manufacturer',
    'Sugar'])
```

This cell generates a box-and-whisker plot of sugar content by manufacturer. In this case, we require a list of all sugar contents from the data set with the addition of manufacturer labelling, so both are requested from each record. Again, the third line reformats the data for bokeh.

Part 10: Graph #4

```
plt.boxplot(
    sugarFrame['Sugar'])
plt.show()
```

Here, we use bokeh’s built-in BoxPlot function. In order for the function to provide accurate results, we must indicate the feature of the data used to group the sugars, manufacturer. The function then computes the quantiles. The final line renders the plot in the notebook.

Conclusion

This example demonstrates the process of loading a dataset and generating 4 graphs from the data in the dataset.