Advanced Graphics in Python

A Jupyter Notebook Example

Introduction

This presentation serves as a walk-through and detailed explanation of the provided Example Jupyter Notebook covering advanced graphics in Python.
This Notebook loads pre-generated datasets based on the cereals dataset used in the previous examples and generates visualizations based on this data.
Here, we will step through the example code and explain each block’s function.

Part 1 - Mosaic Plots

```
import pandas
import matplotlib.pyplot as plt
from statsmodels.graphics.
    mosaicplot import mosaic
```

These three lines import the libraries required to construct and display the mosaic plot. Note the the *statsmodels* library has built-in mosaic plotting capability. Most types of plots have already been implemented in Python, so needing to construct your own graphing function from scratch is unusual.

Mosaic Plots Cont.

```
df = pandas.read_csv(
     'cereals_conf.csv',
      sep=",")
```

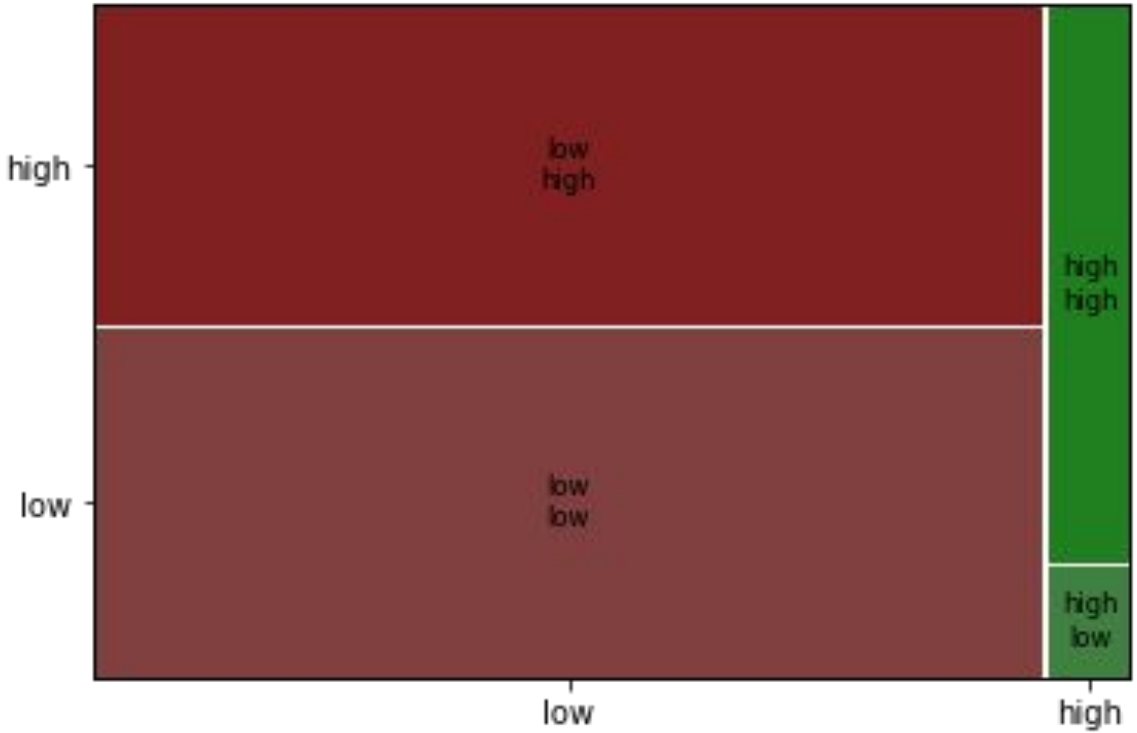
This line reads in the data we provided in a pre-generated file. This file splits the Fat and Protein levels of each cereal into High and Low categories, since a Mosaic Plot treats every distinct value of a feature as a category. Reducing to High and Low ensures that the plot will be easily readable.

Mosaic Plots Cont.

```
mosaic(df, ['Fat', 'Protein'])
plt.show()
```

These two lines construct and show the plot, respectively. The second argument of the *mosaic* function call determines which columns should be plotted and in which order they should be considered. Here, we use the Fat and Protein columns in that order since they are the only two columns in the dataset.

Resultant Plot



Part 2 - Parallel Coordinate Plots

```
import pandas
import matplotlib.pyplot as plt
from pandas.plotting
    import mosaic
```

These three lines serve the same purpose for the Parallel Coordinate Plot as they did for the Mosaic plot. In this case, the Parallel Coordinate Plot is built in to the *pandas* library instead of the *statsmodels* library.

Parallel Coordinate Plots Cont.

```
df = pandas.read_csv(
     'cereals_num.csv',
      sep=",")
```

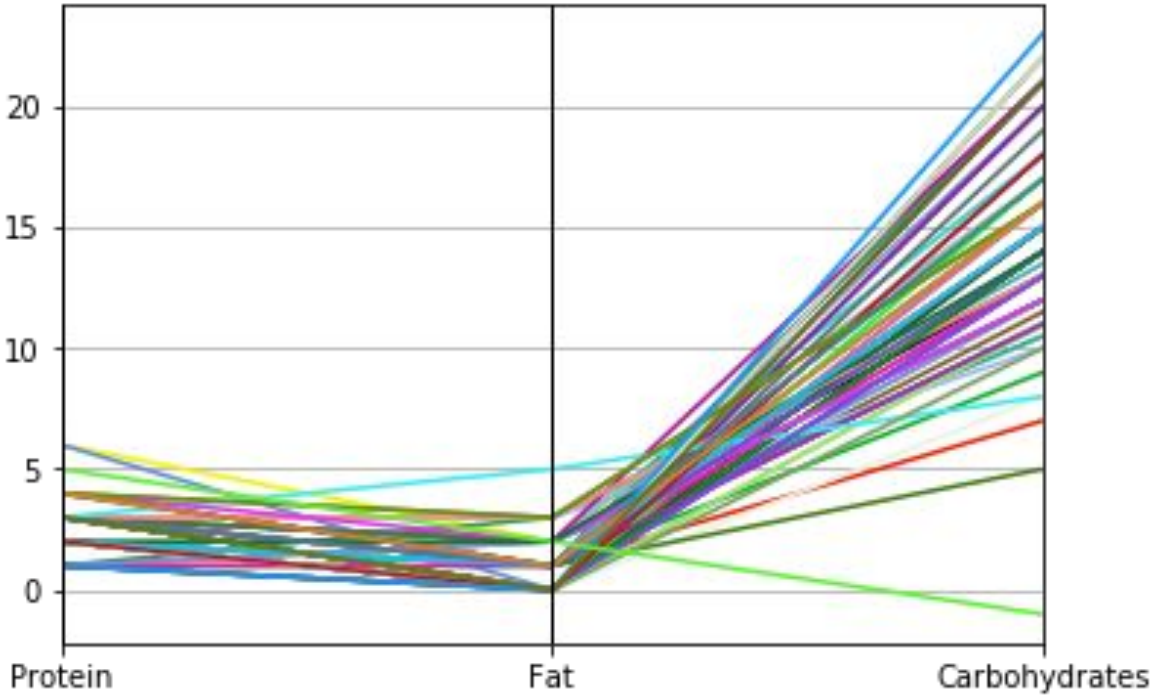
This line of code serves the same function as in the Mosaic plot, loading data from a file. In this case, we have filtered the cereal data down to three features: Protein, Fat, and Carbohydrates so that the resultant plot is maximally readable. Using more than these three features would result in a crowded plot that would be difficult to read.

Parallel Coordinate Plots Cont.

```
parallel_coordinates(data, 'Cereal')
plt.gca().legend_.remove()
plt.show()
```

Again, the graph is constructed in the first line and shown in the notebook in the last line. The second line removes the legend from the graph, since a legend displaying all cereal types would be meaningless. Omitting the legend is not necessary for PCPs, it may be useful if a small number of data instances are plotted. Instead, this graph shows us trends in cereal composition as a whole.

Resultant Plot



Part 3 - Pixel Plots

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cm as cm
```

As with the other two plots, we import libraries to help with the graph. In this case, the colormaps (cm) inform the color scheme of the plot. Here, we also use numpy instead of pandas for handling data.

Pixel Plots Cont.

```
inData = np.loadtxt(
    'cereals_normed.csv',
    delimiter=',',
    dtype=float,
    skiprows=1)
tData =
    inData.transpose()
```

As before, the first line loads data. Unlike before, this line loads data into a numpy matrix rather than a pandas dataframe. The second line transposes the data so that the matrix is oriented column-wise rather than row-wise.

Pixel Plots Cont.

```
plt.imshow(tData,
    interpolation='nearest',
    cmap=cm.inferno)
plt.axis('off')
plt.show()
```

These lines plot and show the pixel plot. The imshow function of matplotlib treated the matrix loaded previously as a matrix of pixel values. The *cmap* parameter specifies the color scheme to be used in plotting the data. Here, the plot's axis are turned off in order to make the plot readable.

Resultant Plot

Part 4 - Scatter Plots

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

As before, import the libraries needed to construct the plot. In this case, the scatter plot is built in to the pandas library, so no additional libraries are needed.

Scatter Plots Cont.

```
df = pandas.read_csv(
    'cereals_num.csv',
     sep=",")
```

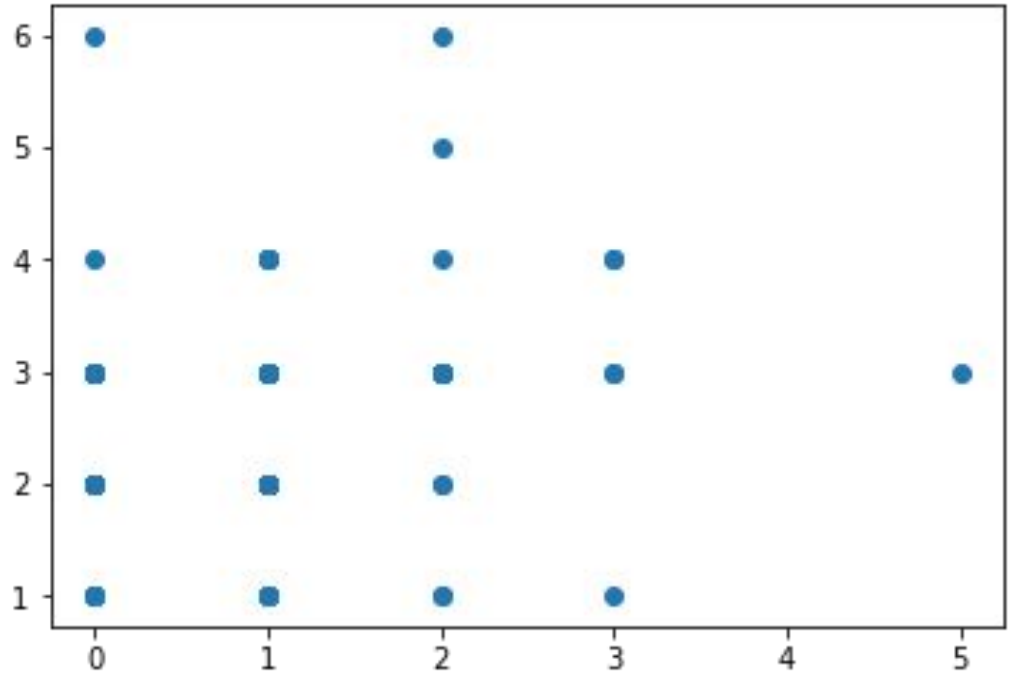
This line imports the same data file as used in the Parallel Coordinate Plot. While this data set contains three features and a scatter plot handles only two features, we will be able to select the features later.

Scatter Plots Cont.

```
plt.scatter(df['Fat'],
    df['Protein'])
plt.show()
```

These final lines construct and show the plot. The call to the scatter function specifies that the Fat column of the dataframe should be the X values and the Protein column should be the Y values.

Resultant Plot



Part 5 - WordCloud/Wordle

```
from wordcloud import
    WordCloud
import matplotlib.pyplot
    as plt
```

As before, these lines import required libraries. Since Wordles use text data instead of numerical data, the pandas library is not helpful here. The wordcloud library, on the other hand contains many of the functions needed to successfully construct a Wordle.

Wordle Cont.

```
text = open('kjb.txt').read()
```

This line reads in the text of the King James Bible, which is available for free online. The King James Bible is commonly used as sample text for text analysis techniques.

Wordle Cont.

```
wordcloud = WordCloud(
    background_color=
        'white')
wordcloud.generate(text)
```

Here, we initialize and generate the Wordle/WordCloud. Since the Jupyter Notebook background is white, we explicity set the background color to white, otherwise the Wordle would appear on a black background.

Wordle Cont.

```
plt.imshow(wordcloud,
    interpolation='bilinear')
plt.axis('off')
plt.show()
```

Again, we show the wordle image using the *imshow* function. Since the *imshow* function treats the wordle as a matrix of value, we must explicitly turn off the plot's axis and instruct *matplotlib* to use interpolation to scale the image appropriately.

Resultant Plot