Visualization with Distributions

This lesson introduces us with types of distributions used to visualize data , and how Python provides support with its libraries.

WE'LL COVER THE FOLLOWING

Introduction to distributions

A probability distribution is a mathematical function that provides the probabilities of the occurrence of different possible outcomes.

For example, you might have a program that returns **1** with a 50% probability and **0** with a 50% probability. Thus, 50% of your probability distribution would be assigned to 1 and 50% to 0.

If you were to plot this expected distribution, you would have two bars of equal height for 1 and 0.

Often, with data you don’t know the mathematical function which generated your data, so instead you observe the empirical distribution. You might sample 10 colored balls from a bag and get 2 red, 3 yellow, and 5 green. That would then be your empirical distribution and you could graphical represent it with 3 bars. One of height 2 for red, one of height 3 for yellow, and one of height 5 for green.

Seaborn has a few ways to plot distributions:

- Histograms

- Box plots
- Violin plots
- Joint plots

Types of distributions

Histogram

As explained by Wikipedia, a **histogram** is an estimate of the probability distribution of a continuous variable. To construct a histogram, the first step is to **bin** the range of values—that is, divide the entire range of values into a series of intervals—and then count how many values fall into each interval. The bins are usually specified as consecutive, non-overlapping intervals of a variable.

You create a histogram with the `distplot()` function is Seaborn. You only need to pass one argument which is the continuous variable for which you would like to construct a histogram.

Histograms, though, have a very important parameter - the number of bins. You specify the number of bins with the `bin` parameter. If unspecified, Seaborn tries to find a useful number of bins to use. It is important to remember, though that the more bins, the higher variance your plot will have; the fewer bins, the more bias. Be thoughtful when choosing the number of bins as it might change the way you view your data.

Let’s take a look:

```
from sklearn.datasets import load_boston
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
# Set the palette and style to be more minimal
sns.set(style='ticks', palette='Set2')

# Load data as explained in introductory lesson
boston_data = load_boston()
boston_df = pd.DataFrame(boston_data.data, columns=boston_data.feature_names)

# Create the histogram plot
sns.distplot(boston_df.NOX, kde=False)
# Remove excess chart lines and ticks for a nicer looking plot
sns.despine()
```

Here is our same histogram but with 100 bins:

```
from sklearn.datasets import load_boston
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
# Set the palette and style to be more minimal
sns.set(style='ticks', palette='Set2')

# Load data as explained in introductory lesson
boston_data = load_boston()
boston_df = pd.DataFrame(boston_data.data, columns=boston_data.feature_names)

# Create the histogram plot
sns.distplot(boston_df.NOX, bins=100, kde=False)
# Remove excess chart lines and ticks for a nicer looking plot
sns.despine()
```

You can see the increased variance by increasing the number of bins.

You will notice we have been setting `kde` =**False**. With this equal to True, it plots the shape of the distribution. We won't go into too much detail about this, but you can learn more here.

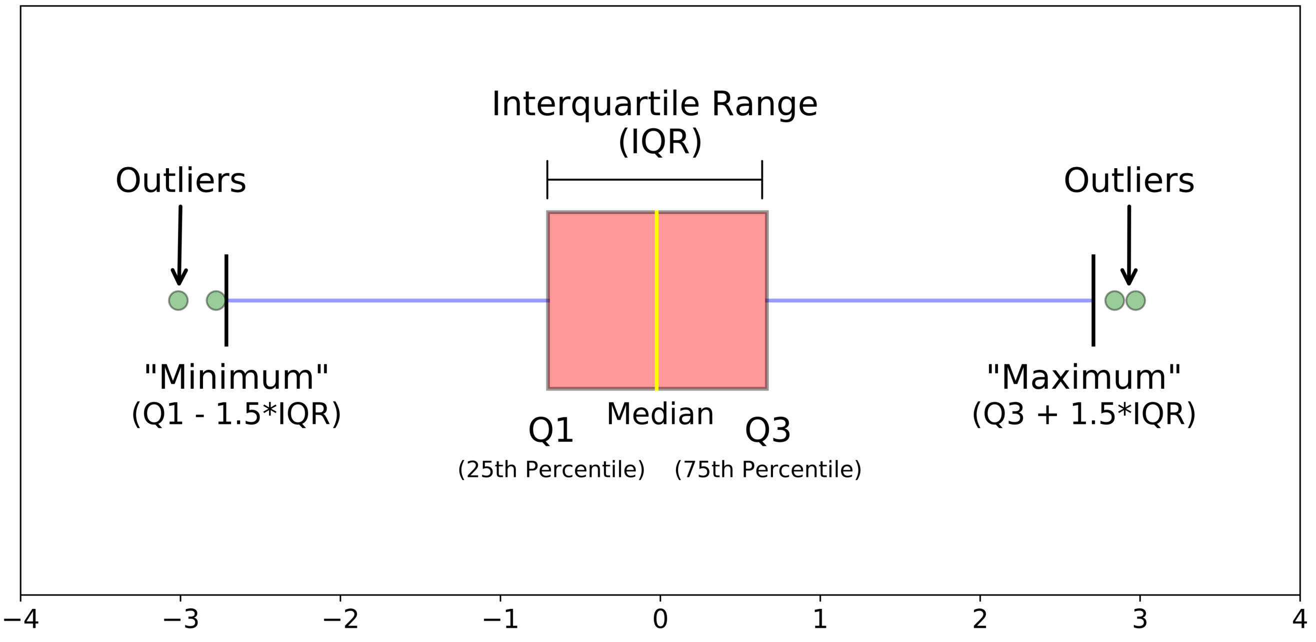
Here is an example:

```
from sklearn.datasets import load_boston
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
# Set the palette and style to be more minimal
sns.set(style='ticks', palette='Set2')

# Load data as explained in introductory lesson
boston_data = load_boston()
boston_df = pd.DataFrame(boston_data.data, columns=boston_data.feature_names)

# Create the histogram plot
sns.distplot(boston_df.NOX, kde=True)
# Remove excess chart lines and ticks for a nicer looking plot
sns.despine()
```

You can see all the parameters for the `distplot` here.

Box plot

Seaborn also has a box plot. A box plot is a graphical view of summary statistics from a distribution. Here is an example:



The middle line is the median, the right side of the box is the 75th percentile and the left side the 25th percentile. The whiskers are the lines that stick out from the box. Usually these extend out 1.5 times the interquartile range (IQR) where the IQR is the distance between the 75th and 25th percentiles. Any values outside the whiskers are usually shown as dots, and as we know, could be considered outliers.

You can create a box plot using the `boxplot` function from Seaborn. The only argument you usually have to pass is the first one which is the continuous variable you which to represent with a box plot.

Here is an example plotting **NOX**:

```
from sklearn.datasets import load_boston
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
# Set the palette and style to be more minimal
sns.set(style='ticks', palette='Set2')
```

```
# Load data as explained in introductory lesson
boston_data = load_boston()
boston_df = pd.DataFrame(boston_data.data, columns=boston_data.feature_names)


# Create the box plot
sns.boxplot(boston_df.NOX)
# Remove excess chart lines and ticks for a nicer looking plot
sns.despine()
```

Violin plot

Seaborn has a plot very similar to a box plot called a violin plot. With this plot you have removed the box part of the box plot and replaced it with the kde curve we saw from the `distplot()` function. This can be valuable as it allows you to see the modality of your distribution. For example, below we see that INDUS is bi-modal. In a box plot, we would not have seen that distinction.

> **Note**: kde curves can look smooth with small amounts of data.

You create a violin plot using the `violinplot()` function in Seaborn. You just pass it a dataframe and it will create a violin plot for each column in your dataframe.

I prefer my plots vertically oriented, so I often use the `orient` parameter with a value of **"v"** to do so.

Here is an example with the **INDUS** variable:

```
from sklearn.datasets import load_boston
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
# Set the palette and style to be more minimal
sns.set(style='ticks', palette='Set2')

# Load data as explained in introductory lesson
boston_data = load_boston()
boston_df = pd.DataFrame(boston_data.data, columns=boston_data.feature_names)

# Create the violin plot
sns.violinplot(boston_df['INDUS'], orient="v")
# Remove excess chart lines and ticks for a nicer looking plot
sns.despine()
```

Joint plot

Lastly, joint plots can plot the histogram of two variables on one plot and also show the scatter plot in the middle. In the top right, you also get the Pearson correlation coefficient between the variables as well as the p-value.

This is done with the `jointplot()` function from Seaborn.

The first parameter is the variable you would like on the x-axis. The second parameter is the variable you would like on the y-axis.

You can also use the `kind` parameter to change how the scatter points are visualized. I like passing a value of **"hex"**.

Here is an example using the **CRIM** and **NOX** variables:

```
from sklearn.datasets import load_boston
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
# Set the palette and style to be more minimal
sns.set(style='ticks', palette='Set2')

# Load data as explained in introductory lesson
boston_data = load_boston()
boston_df = pd.DataFrame(boston_data.data, columns=boston_data.feature_names)

# Create the violin plot
sns.jointplot(boston_df.CRIM, boston_df.NOX, kind="hex")
# Remove excess chart lines and ticks for a nicer looking plot
sns.despine()
```

That's how your data can be visualized through distributions. Next, we'll look at a few ways to visualize data with the line graphs.