Module 6 Part 1: Descriptive Statistics and Visualization

Introduction

This module explores the following topics:

- A review of descriptive statistics and types of measure
- How to calculate descriptive statistics using `pandas` functions
- Visualization techniques and Python commonly used libraries for visualization

This module consists of 2 parts:

- **Part 1** - Descriptive Statistics
- **Part 2** - Visualization

Each part is provided in a separate notebook file. It is recommended that you follow the order of the notebooks.

Learning Outcomes

For the topic of descriptive statistics, you will:

- Review the main concepts of descriptive statistics, including mean, median and standard deviation
- Review correlation
- Learn how to calculate measures of descriptive statistics using `pandas`

For the topic of visualization, you will:

- Learn from and re-create some classic examples
- Apply design principles to creating effective visualizations
- Describe some of the new technologies behind rich, interactive Web visualizations
- Use `matplotlib` to create data plots and charts

Readings and Resources

The majority of the notebook content draws from the recommended readings. We invite you to further supplement this notebook with the following recommended texts:

McKinney, W. (2017). Python for Data Analysis, 2nd Ed., Boston: O'Reilly Media

Diez, D., Barr, C. & Çetinkaya-Rundel, M. (2017). Chapter 1: Introduction to Data In OpenIntro Statistics (3rd Ed.). Available free online or paper copy for purchase.

Table of Contents

Descriptive Statistics

Creating the Model: Understanding the Main Characteristics of Data

> This image shows the stages of creating a model from start to finish. (Course Authors, 2018)
> *This image shows the stages of creating a model from start to finish.*

We have reached phase 3 of the analytics methodology. Once your data has been prepared (i.e., formatted and cleaned), you can start creating your model. The first step to this phase is using descriptive statistics to understand the characteristics of your data, including how it is distributed. This will help you select the correct modeling approach. In the previous modules we learned the basics of NumPy and Pandas. In this module we will learn to apply the functions within those libraries to a clean data set, and in Part 2 we will introduce MatPlotLib, which is a library that will help you visually represent your data.

Types of data

Descriptive statistics are measures that provide simple summaries of data in numeric form. For more information, see the Wikipedia page Descriptive statistics.

It is always a good idea to look at basic descriptive statistics after data cleaning and preparation tasks are complete. These measures provide summaries of the features and will help in understanding the main characteristics of the data.

Before we discuss descriptive statistics, let's talk about different types of data. The data type is a fundamental feature of any variable, it defines how the data can be described and, as you will see later in the program, what algorithms can be used if we need to build a predictive model for the variable in question.

There are two main statistical data types:

1. **Numerical** or **quantitative** data
2. **Categorical** or **qualitative** data

Numerical data can be:

- **Discrete**: these are numerical values that are obtained by **counting**. For example, it can be the number of petals on an iris flower, or a number of experiments, or the number of students in a class. It is any numerical value which represents quantities.
- **Continuous**: these are data points that are obtained by **measuring**. For example, the height of students in a class or the temperature of the water in a lake. It also includes values in a given interval of numbers, e.g. federal spending.

Categorical data can be:

- **Ordinal**: these are values that can be ordered or ranked. An example would be a survey question where you are asked how often you do certain actions: "always", "usually", "sometimes", "rarely", "never". Another example with a natural order: "hot", "medium", "cold".
- **Nominal**: any categorical data that doesn't have any order, e.g. "blue", "red", "green", or a list of Canadian provinces.

Summary statistics

In this section we will review the measures of **summary statistics**. Summary statistics describe data with the following main measures:

- a measure of **location**, or *a measure of center*, where the values are mostly located. This measure includes: mean, median, mode.
- a measure of **spread**, or *statistical dispersion*. This measure includes: standard deviation, variance, range, interquartile range. We will review the first two measures.
- a measure of the **shape** of the data distribution. This measure includes skewness or kurtosis.
- for more than one variable, a measure of **correlation**, statistical dependence of two variables. This measure is called a correlation coefficient.

Let's look at these measures in more detail.

Measures of location

The most frequently used measures of location for numerical data are:

- **Mean**, or arithmetic mean, or average. It is simply the sum of all values divided by the total number of values.

If we have the following dataset: `21 22 15 12 21 16`, the mean value is calculated as follows:

$\bar{X}$ = (21 + 22 + 15 + 12 + 21 + 16) / 6 = 17.83

- **Median** is the middle value in a sorted list of values. Median splits the data in the "middle" so that 50% of the data points are above this point and 50% - below.

Let's use the same simple dataset above. To get the median, first, sort the list of values:

`12 15 16 21 21 22`

In the "middle" of this dataset there are two values, `16` and `21`. If there is an even number of values in the middle of the sorted list, we need to calculate an average of these values. Hence, the median value for this dataset is:

`(16 + 21) / 2 = 18.5`

The median is often not as affected by **outliers** within a dataset as the mean. We will talk about outliers and their impact later in the course.

Let's review Python functions to calculate the measures discussed above. For this exercise, we will use the **iris dataset** which we have worked with in the `pandas` module.

In [1]:
```python
import pandas as pd
import numpy as np
```

In [2]:
```python
iris = pd.read_csv('iris.data', sep=',',
                   header=None,  # the data file does not contain a header
                   names=['sepal length','sepal width','petal length','petal width'
                   )

iris.head()
```

Out[2]:

	sepal length	sepal width	petal length	petal width	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

Let's group the data by the class of the flower and calculate the measures of location for each class and for each variable, such as sepal length and width, and petal length and width.

In [3]:
```
# grouping data by class
iris_grouped = iris.groupby('class')

# calculating mean (average) for each class:
iris_grouped.mean()
```

Out[3]:

	sepal length	sepal width	petal length	petal width
class				
Iris-setosa	5.006	3.418	1.464	0.244
Iris-versicolor	5.936	2.770	4.260	1.326
Iris-virginica	6.588	2.974	5.552	2.026

In [4]:
```
# calculating median for each class:
iris_grouped.median()
```

Out[4]:

	sepal length	sepal width	petal length	petal width
class				
Iris-setosa	5.0	3.4	1.50	0.2
Iris-versicolor	5.9	2.8	4.35	1.3
Iris-virginica	6.5	3.0	5.55	2.0

We can see that in our example, the mean and the median values for all of the iris classes are very close to each other. This means that the data is evenly divided and distributed around the mean. We will talk about data distribution shortly.

Measures of spread

In this section, we will practice calculating **variance** and **standard deviation**, which are **measures of spread**, using the simple dataset of integers we used above:

`21 22 15 12 21 16`

Definitions of the main measures of spread are:

- **Variance** is a measure of the variability in the data, it measures how far values are spread out from the mean. Variance is roughly the average squared distance from the mean.
- **Standard deviation** is the square root of the variance and can be used to describe how close the typical data point is to the mean. Standard deviation is a descriptor of variability in the data. Usually 70% of the data will be within one standard deviation of the mean and about 95% will be within two standard deviations, if the data is normally distributed. The distance of any value from its mean is called **deviation**.

Here is how we would calculate deviations for each element of the data sequence (you remember that we calculated $\bar{x}$ as 17.83 above):

$\begin{align*} x_1 - \bar{x} &= 21 - 17.83 = 3.17 \\ x_2 - \bar{x} &= 22 - 17.83 = 4.17 \\ x_3 - \bar{x} &= 15 - 17.83 = -2.83 \\ x_4 - \bar{x} &= 12 - 17.83 = -5.83 \\ x_5 - \bar{x} &= 21 - 17.83 = 3.17 \\ x_6 - \bar{x} &= 16 - 17.83 = -1.83, \end{align*}$

where $ x_1 $ ... $ x_6 $ are data points of the sequence of integers we are investigating.

If we square these deviations and then take an average, the result will be the **variance** which is usually denoted by s^2:

The formula for variance is:

$s^2 = \dfrac{\displaystyle\sum_{i=1}^{n} (x_i - \bar{x})^2}{n - 1} $

Squaring the deviations achieves two goals: (1) it makes large values much larger, and (2) it gets rid of any negative signs. Let's calculate the variance for our dataset:

$s^2 = \dfrac{3.17^2 + 4.17^2 + (-2.83)^2 + (-5.83)^2 + 3.17^2 + (-1.83)^2}{6-1} = 16.57$

The **standard deviation** is the square root of the variance:

$ s = \sqrt{s^2} = \sqrt{16.57} = 4.07$

The standard deviation is a good indicator of how close the data points are to the mean.

NOTE: You probably noticed that in the formula for variance, we used `n-1` instead of `n`. This is called Bessel's correction, it corrects the bias in the estimation of the population variance. You can read more about this correction in this Wikipedia article. We will also discuss it in more detail in one of the subsequent courses of the program.

Moving back to the iris dataset, let's calculate the variance and standard deviation for each class of flower.

In [5]:
```
# Variance:

iris_grouped.var()
```

Out[5]:

	sepal length	sepal width	petal length	petal width
class				
Iris-setosa	0.124249	0.145180	0.030106	0.011494
Iris-versicolor	0.266433	0.098469	0.220816	0.039106
Iris-virginica	0.404343	0.104004	0.304588	0.075433

In [6]:
```
# Standard deviation:

iris_grouped.std()
```

Out[6]:

	sepal length	sepal width	petal length	petal width
class				
Iris-setosa	0.352490	0.381024	0.173511	0.107210
Iris-versicolor	0.516171	0.313798	0.469911	0.197753
Iris-virginica	0.635880	0.322497	0.551895	0.274650

Before we discuss the last two measures of summary statistics (shape and correlation), we need to talk about data distribution: what it shows us and its main characteristics.

Data distributions

In statistics, a **frequency distribution** is usually represented as a list, table or graph that displays the frequency of data values within a dataset. If the frequency distribution is presented as a table, it will display the number of values that fall within certain data intervals. This table summarizes the distribution of values in the dataset.

Let's consider an example. Let's say we are researching house prices in a certain part of the city. We collected data for the houses on the market, and placed it in the table below. In order to make an analysis easier, we decided to group the prices into bins and count how many houses fall into each bin based on their price. We might end up with the following table (the numbers are arbitrary and used for illustration purposes only):

Price range, $	Number of houses
0-200,000	2
200,001 - 300,000	10
300,001 - 400,000	15
400,001 - 500,000	25
500,001 - 600,000	30
600,001 - 700,000	25
700,001 - 800,000	15
800,001 - 900,000	10
900,001 - 1,000,000	2

We created a **frequency table**. If we take the data from the table and create a bar graph with it, we have a **frequency distribution**. Also called a histogram, frequency distributions provide a view of **data density** and are very convenient for describing the shape of the data distribution.

Let's go through the steps to create a frequency distribution for the dataset above. First, we create a dataframe:

In [7]:
```
houses = pd.DataFrame({
    'price, thousands': [200,300,400,500,600,700,800,900,1000],
    'num of houses':[2,10,15,25,30,25,15,10,2]
})
houses.head()
```

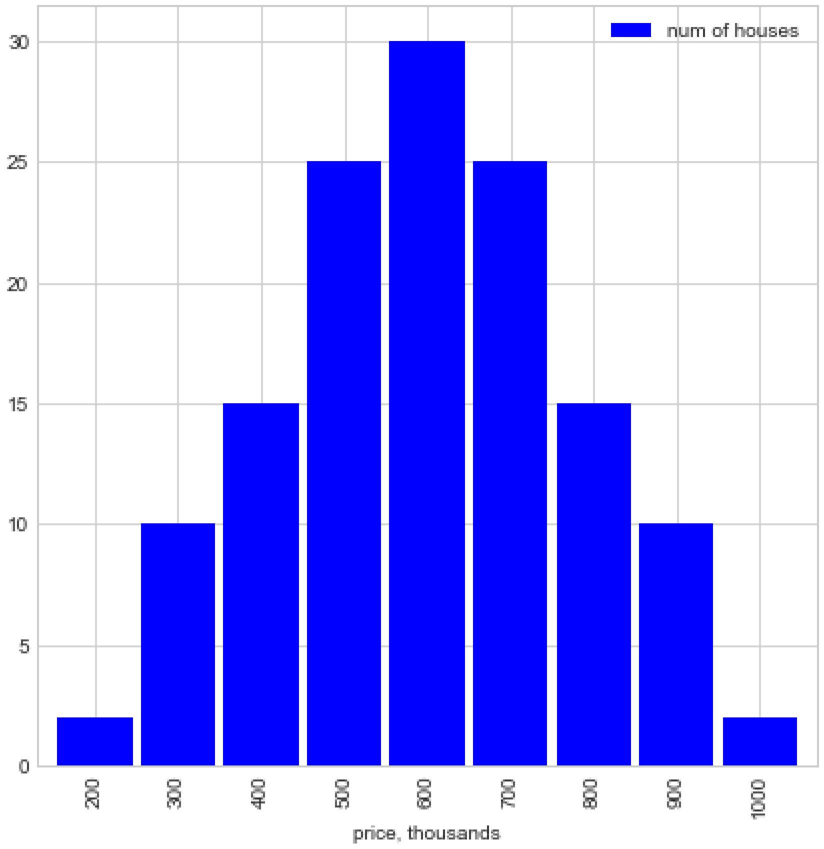
Out[7]:

	price, thousands	num of houses
0	200	2
1	300	10
2	400	15
3	500	25
4	600	30

We can now plot the frequency distribution graph. Usually, we would use Python's `hist()` function to create a histogram based on the raw data. This function calculates frequencies and create bins which is what we did above, but because in this particular example we already have bins and frequencies, we can simply draw a bar plot. You will see examples of the `hist()` function later in this module.

NOTE: For plotting graphs, we will use Python's visualization library `matplotlib`. We are not going into the details of this library here, this is a topic for Part 2 of this module.

In [8]:
```
from matplotlib import pyplot as plt
%matplotlib inline
plt.style.use('seaborn-whitegrid')
plt.rcParams["figure.figsize"] = (7,7)

houses.plot(x = 'price, thousands', y = 'num of houses', kind='bar', color = 'blue'
plt.show()
```



You can see from the plot that the distribution appears to be symmetric and bell-shaped. This is an example of what we call a **normal distribution**. Half of the data will fall to the left of the mean; half will fall to the right. Mean and median values for normally distributed data should be roughly the same.

Let's explore the mean, median and standard deviation for this data set:

In [9]:
```
houses['num of houses'].mean()
```

Out[9]:
```
14.88888888888889
```

In [10]:
```
houses['num of houses'].median()
```

Out[10]:
```
15.0
```

```
In [11]: houses['num of houses'].std()
```

```
Out[11]: 10.080233683358294
```

If the mean and median were different from one another, the shape of the curve would shift to have a left or right skew.

If the standard deviation were larger, the curve would be wider, and if the standard deviation were smaller, the curve would be narrower.

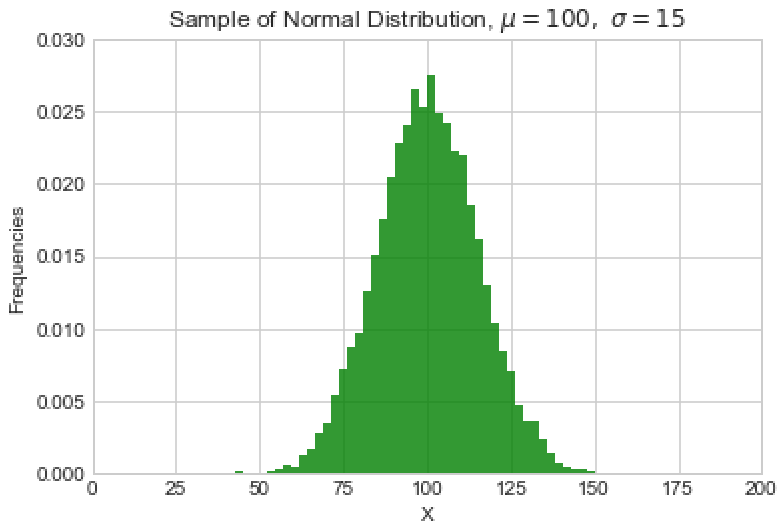
Let's create a normal distribution of random numbers and plot a histogram of this sequence. We will use the function `randn()` from the `numpy.random` library which will create an array of random normally distributed floating point numbers. For details regarding the `numpy.random` library, see the SciPy.org documentation for the `numpy.random.randn` function.

The array created using this function will have mean 0 and variance 1. The normal distribution with mean 0 and standard deviation 1 is called the **standard normal distribution**.

Based on this standard normal distribution, we are going to create another array of random normally distributed floating point numbers, `x`, with the `mean = 100` and `standard deviation = 15`. Then we will create a histogram with 50 bins and plot it.

NOTE: We chose 50 bins for this example, but this is not an ideal or required number. You might want to experiment and change this number later.

```
In [12]: mu = 100       # mean of distribution
         sigma = 15     # standard deviation of distribution

         # Creating an array of 10,000 random numbers with 0 mean and St Dev of 1
         # x is a random variable with mean value of 'mu' and St Dev of 'sigma'
         x = mu + sigma * np.random.randn(10000)

         num_bins = 50   # number of bins to create
         # the histogram of the data
         n, bins, patches = plt.hist(x, num_bins, density=1, facecolor='green', alpha=0.8)
         plt.xlabel('X')
         plt.axis([0, 200, 0, 0.03])
         plt.ylabel('Frequencies')
         plt.title("Sample of Normal Distribution, " + "$\mu=100,\ \sigma=15$")
         plt.show()
```



In [13]:
```
"""
We can check the standard deviation of the distribution we drew above.
NOTE: The value will be slightly different from the original number, 15, due to rou
If you re-run the code above, you might get slightly different outputs for std() an
"""
x.std()
```

Out[13]:
```
15.051548838188845
```

In [14]:
```
x.mean()
```

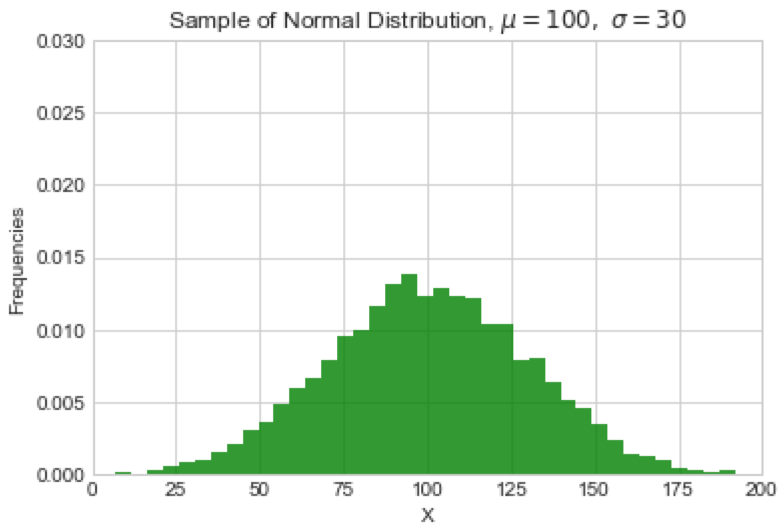
Out[14]:
```
100.08464488990754
```

NOTE: Every time you re-run the code above, the shape of the distribution will change slightly because on each execution the random number generator will create a new array of random numbers.

We discussed above that standard deviation is a measure used to quantify the amount of variability in the data. A low standard deviation indicates that there is little variability in the data and the data points are tightly clustered around the mean of the distribution, while a high value of standard deviation indicates that the values are spread out over a wider range of values. This will also be reflected in the histogram.

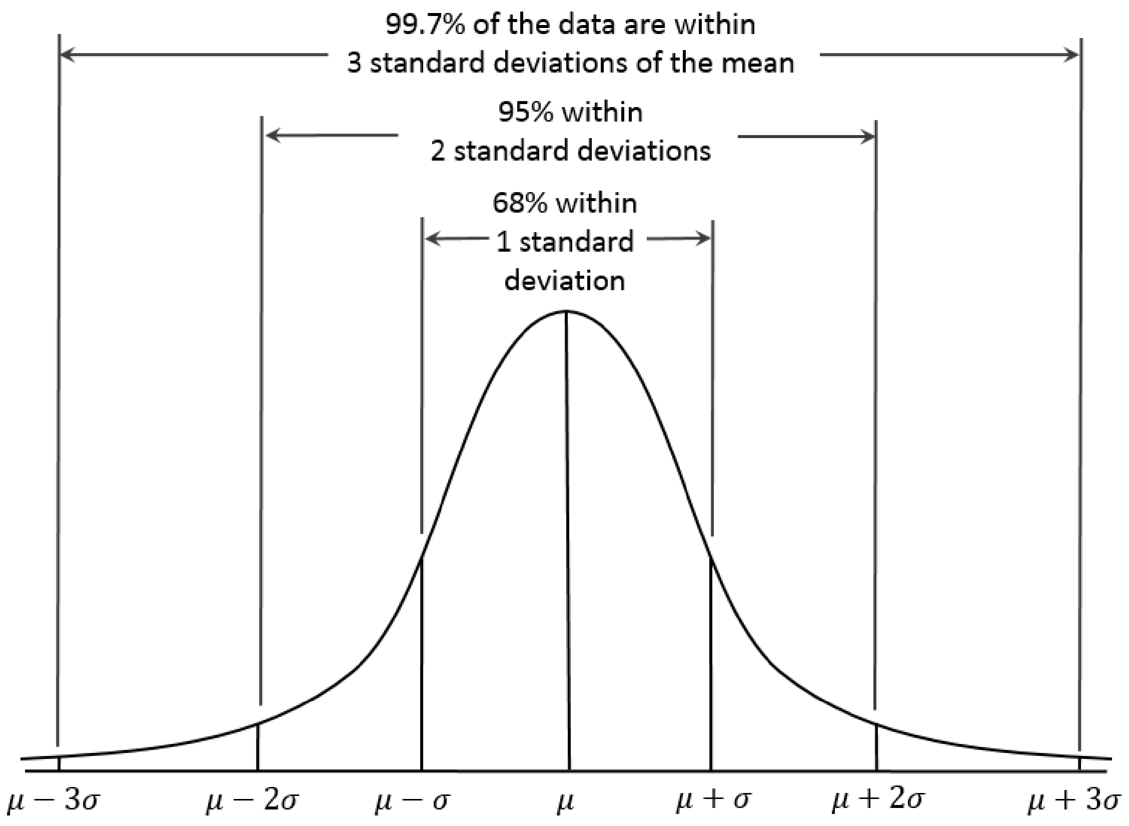
Let's take the histogram above and change only one parameter, standard deviation, from 15 to 30. We will copy the code from above and make the change:

In [15]:
```
mu = 100
sigma = 30     # changed standard deviation from 15 to 30

x = mu + sigma * np.random.randn(10000)

num_bins = 50   # number of bins to create
n, bins, patches = plt.hist(x, num_bins, density=1, facecolor='green', alpha=0.8)
plt.xlabel('X')
plt.axis([0, 200, 0, 0.03])
plt.ylabel('Frequencies')
plt.title("Sample of Normal Distribution, " + "$\mu=100,\ \sigma=30$")
plt.show()
```



You can see that the distribution is much wider than the previous one, which indicates more variability in the data. There is a wider range of values within the data and some data points are farther away from the mean.

The graphic below summarizes measures of the normal distribution. This histogram demonstrates the **empirical rule**, also known as the 68–95–99.7 rule.


99.7% of the data are within 3 standard deviations of the mean
95% within 2 standard deviations
68% within 1 standard deviation
$\mu-3\sigma$
$\mu-2\sigma$
$\mu-\sigma$
μ
$\mu+\sigma$
$\mu+2\sigma$
$\mu+3\sigma$

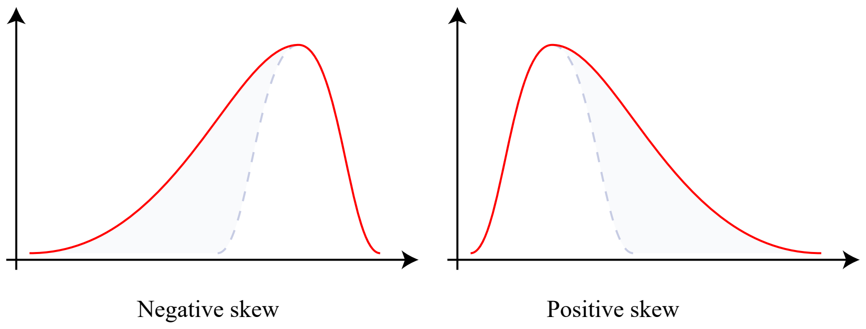
This histogram shows that for the normal distribution, 99.7% of the data is within 3 standard deviations of the mean; 95% of the data is within two standard deviations from the mean, and the values less than one standard deviation away from the mean account for 68.27% of the dataset.

Measures of shape

Histograms are very useful for describing the shape of data distributions. There are two measures that describe the shape of a distribution:

1. **Skewness** is a measure of the asymmetry of a distribution. When a distribution trails off to the right, appearing as a tail on the right-hand side of the graph, the shape is said to be **right skewed**, or has a *positive skew*. Similarly, when a distribution trails off to the left, we say that the data is **left skewed**, or has a *negative skew*.

The normal distribution has a skewness of 0. Therefore, the skewness will tell us whether data is concentrated to the left or to the right of the mean (Wikipedia, 2018).



The Skewness coefficient can be calculated using the formula below which is called Pearson's second skewness coefficient (median skewness).

`Skewness = 3 * (Mean - Median) / standard deviation`

For more details about measures of skewness, see Skewness. (Wikipedia, 2018).

2. **Kurtosis** is a measure of the "tailedness" of the distribution — it is a good indicator of the outliers in the data.

The normal distribution has a kurtosis of 3. If the distribution has kurtosis less than 3, there are fewer outliers than the normal distribution and vice versa, kurtosis of greater than 3 indicates there are more outliers.

We won't go into more details on these two measures in this module, you will meet with them one more time in the Statistics course of the program.

However, let's take a quick look at the iris data and calculate skewness for this data:

In [16]:
```
iris_grouped.skew()
```

Out[16]:

class	sepal length	sepal width	petal length	petal width
Iris-setosa	0.120087	0.107053	0.071846	1.197243
Iris-versicolor	0.105378	-0.362845	-0.606508	-0.031180
Iris-virginica	0.118015	0.365949	0.549445	-0.129477

As you can see from the table above, data for the `Iris-setosa` class of iris flower is right skewed (positive skew) for all four parameters we are analyzing here, the data for `Iris-versicolor` is mostly left skewed (negative skew), and the data for `Iris-virginica` is mostly right skewed with `petal width` data being left skewed. We will discuss the significance of this measure in the next course of the program.

Correlation

Correlation is the last measure we will discuss in this section. Correlation, which always takes values between -1 and 1, describes the strength of the linear relationship between two variables. The correlation coefficient is usually denoted by `R`.

Two or more variables can vary with each other. This is called **covariance**. Covariance is a measure of the joint variability of two variables.

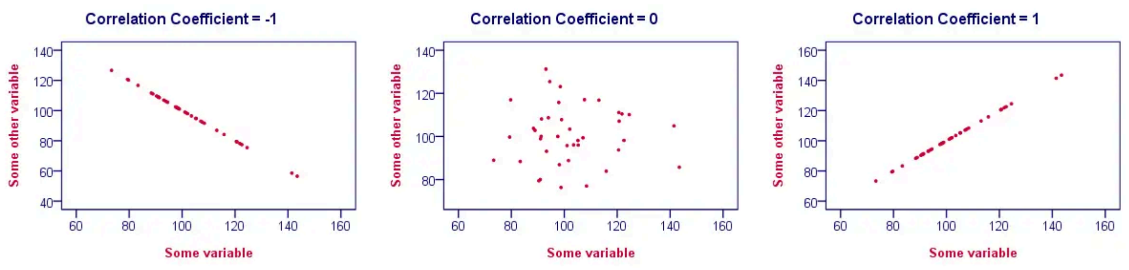
The sign of covariance shows the tendency in the linear relationship between the variables.

- **Positive correlation**: as one variable increases, the other variable also increases. (Low values for variable x associated with low values for variable y; high values for variable x associated with high values for variable y.)
- **Negative correlation**: as one variable increases, the other variable decreases. (Low values for variable x associated with high values for variable y; high values for variable x associated with low values for variable y)

However, the magnitude of the covariance is not easily interpretable. **Correlation coefficient** is used as a measure of covariance.

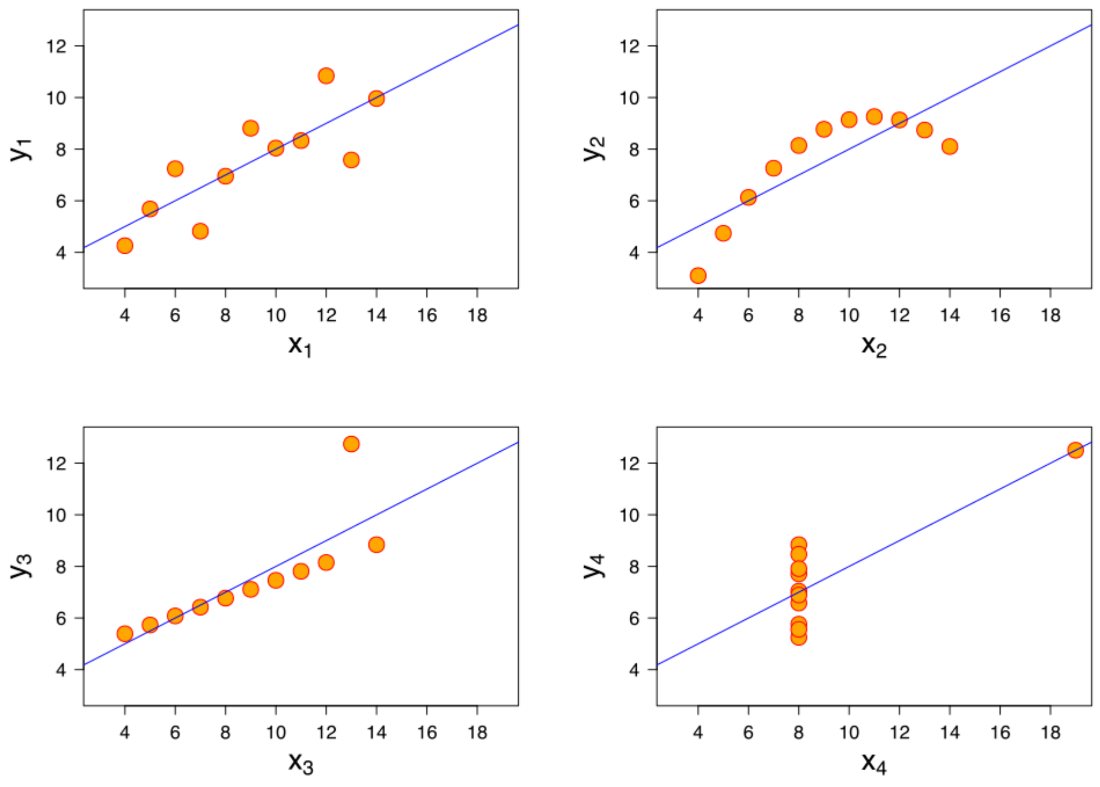
As seen in the set of images below:

- If the linear relationship is strong and negative, the correlation coefficient R will be near -1, but never less than -1.
- If there is no apparent linear relationship, the correlation coefficient R will be near 0.
- If the linear relationship is strong and positive, the correlation coefficient R will be near +1, but never greater than +1.



(SigmaPlus Consulting (2018))

The correlation coefficient indicates the strength of a linear relationship between two variables but it does not provide any insights into other characteristics of these relationships. The image below shows a set of famous scatter plots known as Anscombe's quartet. The four `y` variables have the same mean (7.5), variance (4.12), correlation (0.816) and the same regression line `(y = 3 + 0.5x)`. However, as you can see on the plots, the distribution of the variables is very different.



(Wikipedia, 2018))

Plot 1 (top left corner) shows data which appears scattered but demonstrates a consistent linear trend for the entire data set. Plot 2 (top right corner) shows the data plot that resembles an arc. Plot 3 (first in the second row) shows almost a perfect linear relationship for all data points, except one outlier, which sits far above the trendline of the rest of the data. Plot 4 shows all data points (except one) are centered on one value for `x`, and scattered across a range for `y`. The single data point that does not fit with this description has a much larger value for both `x` and `y` than all of the others.

These four datasets were constructed in 1973 by statistician Francis Anscombe to demonstrate both the importance of graphing data before analyzing it and the effect of outliers on statistical properties. He described the article as being intended to counter the impression among statisticians that "numerical calculations are exact, but graphs are rough."

In `pandas`, the correlation coefficient is calculated using `corr()` function. However, keep in mind that this function won't work on a grouped object. For example, we can calculate the correlation between sepal length and petal length in the iris data, across all classes of the flower.

In [17]:
```
iris['sepal length'].corr(iris['petal length'])
```

Out[17]:
```
0.8717541573048718
```

We can see from the number above that there is a pretty strong correlation between sepal and petal length across all three classes of flowers.

If we want to be able to calculate correlations within the groups, we can use a couple of approaches.

Approach #1: Use the `corrwith()` function which allows us to calculate pairwise correlation between the columns of two DataFrame objects:

In [18]:
```
correlations = (iris[['sepal length', 'class']]
                .groupby('class')
                .corrwith(iris['petal length'])
                .rename(columns={'sepal length' : 'Corr Coef'}))

correlations
```

Out[18]:

	Corr Coef
class	
Iris-setosa	0.263874
Iris-versicolor	0.754049
Iris-virginica	0.864225

Approach #2: Use `apply()` and `lambda` on the groupBy object:

In [19]:
```
iris_grouped.apply(lambda iris_grouped: iris_grouped['sepal length'].corr(iris_grou
```

Out[19]:
```
class
Iris-setosa        0.263874
Iris-versicolor    0.754049
Iris-virginica     0.864225
dtype: float64
```

As you can see, we cannot rely on the generalized calculation across multiple classes of the flowers. The correlation between sepal and petal lengths is quite strong for versicolor (R = 0.75) and virginica (R = 0.86) flowers, but is weak for setosa flowers with R = 0.26.

Summary Statistics in Pandas

We have already used several functions to calculate some of the descriptive statistics measures, such as standard deviation, skewness, mean and median. However, pandas has a very useful function, `describe()`, which calculates multiple measures and displays then in a table format. Let's look at this function. The dataset we are going to use for this exercise is the Wine Quality Data Set (Cortez et al, 2009). Please take a look at the description of the data set and what each attribute means. Then come back and we wiil analyze it.

First let's load the data:

In [20]:
```
# This will set data_url to the URL of the dataset so you can download it
data_url = ("http://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/

# Download the data
red_wine_data = pd.read_csv(data_url, header=0, sep=";")
red_wine_data.head()
```

Out[20]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	a
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	

In [21]:
```
# Using describe():

red_wine_data.describe()
```

Out[21]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	tota
count	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599
mean	8.319637	0.527821	0.270976	2.538806	0.087467	15.874922	46
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	32
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	62
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289

As you can see, for numeric data, the resulting output includes the following measures:

- count, which is a count of non-null values for each attribute
- mean
- standard deviation
- min, the lowest value
- max, the greatest value
- lower (25%), 50 (50%) and upper (75%) percentiles. The 50 percentile is the same as the median.

What we can see in this dataset? First of all, it seems that there are no null values because the counts for all attributes are equal to 1,599. We can validate this assumption:

In [22]:
```
red_wine_data.isnull().values.any()
```

Out[22]:
```
False
```

If required, we can calculate individual values for single columns. For example, here are mean, median and standard deviation for the attribute "density":

In [23]:
```
red_wine_data.density.mean()
```

Out[23]:
```
0.9967466791744833
```

In [24]:
```
red_wine_data.density.median()
```

Out[24]:
```
0.99675
```

In [25]:
```
red_wine_data.density.std()
```

Out[25]:
```
0.0018873339538425563
```

This data set contains attributes that are on different scales. To normalize the data so all attributes (columns) are on the same scale, we can scale each column so that the mean becomes zero and the data is in units of standard deviation. See the following article for different normalization methods https://en.wikipedia.org/wiki/Normalization_(statistics) (Wikipedia, 2018).

We will use Standard Score, which is calculated as follows:

${\dfrac {X-m }{s}}$,

where $ m $ is the mean and $ s $ is standard deviation.

In [26]:
```
normalized_red_wine_data = (red_wine_data - red_wine_data.mean()) / red_wine_data.s
normalized_red_wine_data[:5]
```

Out[26]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	
0	-0.528194	0.961576	-1.391037	-0.453077	-0.243630	-0.466047	-0.379014	0.558100	1.
1	-0.298454	1.966827	-1.391037	0.043403	0.223805	0.872365	0.624168	0.028252	-0.
2	-0.298454	1.296660	-1.185699	-0.169374	0.096323	-0.083643	0.228975	0.134222	-0.
3	1.654339	-1.384011	1.483689	-0.453077	-0.264878	0.107558	0.411372	0.664069	-0.
4	-0.528194	0.961576	-1.391037	-0.453077	-0.243630	-0.466047	-0.379014	0.558100	1.

Now we can calculate correlations for all attributes within the DataFrame:

In [27]:
```
normalized_red_wine_data.corr()
```

Out[27]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	de
fixed acidity	1.000000	-0.256131	0.671703	0.114777	0.093705	-0.153794	-0.113181	0.66
volatile acidity	-0.256131	1.000000	-0.552496	0.001918	0.061298	-0.010504	0.076470	0.02
citric acid	0.671703	-0.552496	1.000000	0.143577	0.203823	-0.060978	0.035533	0.36
residual sugar	0.114777	0.001918	0.143577	1.000000	0.055610	0.187049	0.203028	0.35
chlorides	0.093705	0.061298	0.203823	0.055610	1.000000	0.005562	0.047400	0.20
free sulfur dioxide	-0.153794	-0.010504	-0.060978	0.187049	0.005562	1.000000	0.667666	-0.02
total sulfur dioxide	-0.113181	0.076470	0.035533	0.203028	0.047400	0.667666	1.000000	0.07
density	0.668047	0.022026	0.364947	0.355283	0.200632	-0.021946	0.071269	1.00
pH	-0.682978	0.234937	-0.541904	-0.085652	-0.265026	0.070377	-0.066495	-0.34
sulphates	0.183006	-0.260987	0.312770	0.005527	0.371260	0.051658	0.042947	0.14
alcohol	-0.061668	-0.202288	0.109903	0.042075	-0.221141	-0.069408	-0.205654	-0.49
quality	0.124052	-0.390558	0.226373	0.013732	-0.128907	-0.050656	-0.185100	-0.17

In the next section, we will learn how to visualize this matrix and how to visualize the data.

End of Part 1

This notebook makes up one part of this module. Now that you have completed this part, please proceed to the next notebook in this module.

If you have any questions, please reach out to your peers using the discussion boards. If you and your peers are unable to come to a suitable conclusion, do not hesitate to reach out to your instructor on the designated discussion board.

References

Wikipedia (2018). Descriptive statistics. Retrieved from https://en.wikipedia.org/wiki/Descriptive_statistics

Wikipedia (2018). Bessel's correction. Retrieved from https://en.wikipedia.org/wiki/Bessel%27s_correction

SciPy Community (2018). numpy.random.randn. Retrieved from https://docs.scipy.org/doc/numpy/reference/generated/numpy.random.randn.html.

Wikipedia (2018). Normal Distribution. Empirical Rule diagram. Retrieved from https://en.wikipedia.org/wiki/Normal_distribution

Wikipedia (2018). Skewness. Retrieved from https://en.wikipedia.org/wiki/Skewness

SigmaPlus Consulting (2018). SPSS Tutorials. Pearson Correlations - Quick Introduction. Retrieved from https://www.spss-tutorials.com/pearson-correlation-coefficient/

Wikipedia (2018). Correlation and dependence. Retrieved from https://en.wikipedia.org/wiki/Correlation_and_dependence

Cortez, P., Cerdeira, A., Almeida, F., Matos, T., & Reis, J (2009). Modeling wine preferences by data mining from physicochemical properties. Decision Support Systems, Elsevier, 47(4):547-553. Retrieved from https://archive.ics.uci.edu/ml/datasets/Wine+Quality

Wikipedia, 2018. Normalization (statistics). Retrieved from https://en.wikipedia.org/wiki/Normalization_(statistics)

In [ ]: