2
Inferential Statistics

Before getting understanding the inferential statistics, let's look at what descriptive statistics is about.

Descriptive statistics is a term given to data analysis that summarizes data in a meaningful way such that patterns emerge from it. It is a simple way to describe data, but it does not help us to reach a conclusion on the hypothesis that we have made. Let's say you have collected the height of 1,000 people living in Hong Kong. The mean of their height would be descriptive statistics, but their mean height does not indicate that it's the average height of whole of Hong Kong. Here, inferential statistics will help us in determining what the average height of whole of Hong Kong would be, which is described in depth in this chapter.

Inferential statistics is all about describing the larger picture of the analysis with a limited set of data and deriving conclusions from it.

In this chapter, we will cover the following topics:

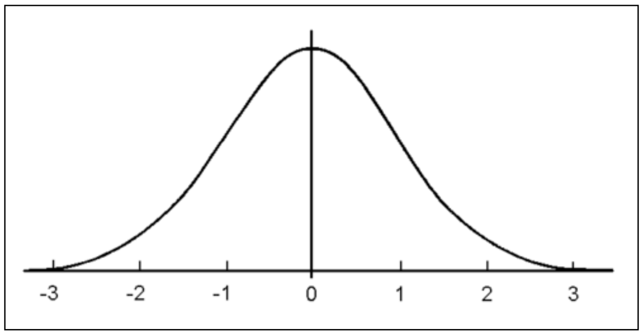
- The different kinds of distributions
- Different statistical tests that can be utilized to test a hypothesis
- How to make inferences about the population of a sample from the data given
- Different kinds of errors that can occur during hypothesis testing
- Defining the confidence interval at which the population mean lies
- The significance of p-value and how it can be utilized to interpret results

Various forms of distribution

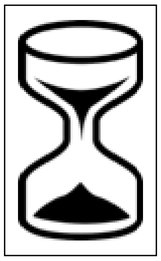
There are various kinds of probability distributions, and each distribution shows the probability of different outcomes for a random experiment. In this section, we'll explore the various kinds of probability distributions.

A normal distribution

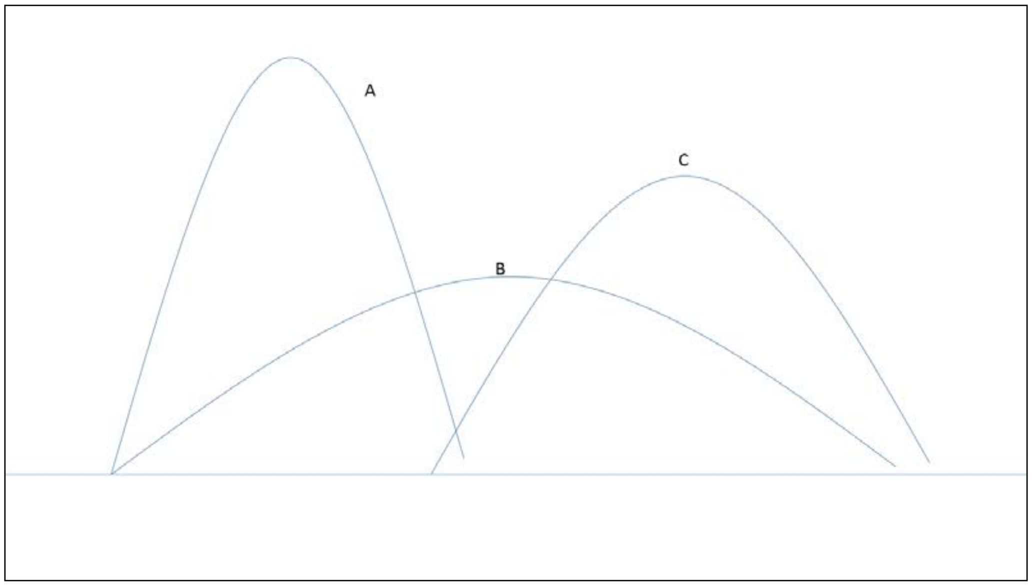
A normal distribution is the most common and widely used distribution in statistics. It is also called a "bell curve" and "Gaussian curve" after the mathematician Karl Friedrich Gauss. A normal distribution occurs commonly in nature. Let's take the height example we saw previously. If you have data for the height of all the people of a particular gender in Hong Kong city, and you plot a bar chart where each bar represents the number of people at this particular height, then the curve that is obtained will look very similar to the following graph. The numbers in the plot are the standard deviation numbers from the mean, which is zero. The concept will become clearer as we proceed through the chapter.



Also, if you take an hourglass and observe the way sand stacks up when the hour glass is inverted, it forms a normal distribution. This is a good example that shows how normal distribution exists in nature.



Take the following figure: it shows three curves with normal distribution. The curve **A** has a standard deviation of 1, curve **C** has a standard deviation of 2, and curve **B** has a standard deviation of 3, which means that the curve **B** has the maximum spread of values, whereas curve **A** has the least spread of values. One more way of looking at it is if curve **B** represented the height of people of a country, then this country has a lot of people with diverse heights, whereas the country with the curve **A** distribution will have people whose heights are similar to each other.



A normal distribution from a binomial distribution

Let's take a coin and flip it. The probability of getting a head or a tail is 50%. If you take the same coin and flip it six times, the probability of getting a head three times can be computed using the following formula:

$$P(x) = \frac{n!}{x!(n-x)!} p^x q^{n-x}$$

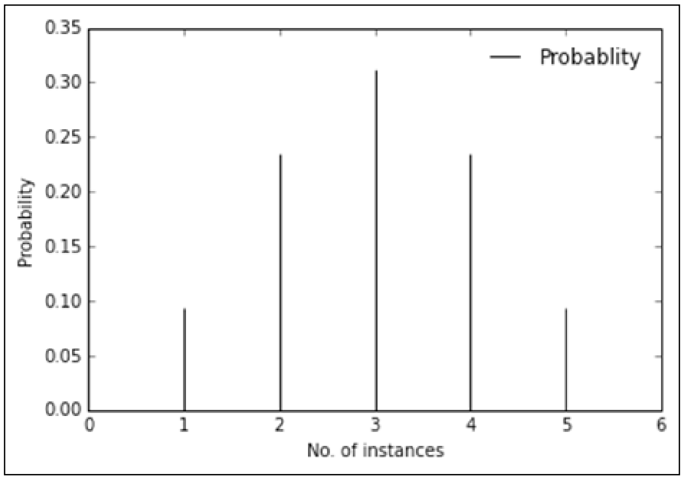
$$\textit{and } x \text{ is the number of successes desired}$$

In the preceding formula, *n* is the number of times the coin is flipped, *p* is the probability of success, and *q* is (*1– p*), which is the probability of failure.

The SciPy package of Python provides useful functions to perform statistical computations. You can install it from `http://www.scipy.org/`. The following commands helps in plotting the binomial distribution:

```
>>> from scipy.stats import binom
>>> import matplotlib.pyplot as plt


>>> fig, ax = plt.subplots(1, 1)
>>> x = [0, 1, 2, 3, 4, 5, 6]
>>> n, p = 6, 0.5
>>> rv = binom(n, p)
>>> ax.vlines(x, 0, rv.pmf(x), colors='k', linestyles='-', lw=1,
         label='Probablity')

>>> ax.legend(loc='best', frameon=False)
>>> plt.show()
```

The `binom` function in the SciPy package helps generate binomial distributions and the necessary statistics related to it. If you observe the preceding commands, there are parts of it that are from the matplotlib, which we'll use right now to plot the binomial distribution. The matplotlib library will be covered in detail in later chapters. The `plt.subplots` function helps in generating multiple plots on a screen. The `binom` function takes in the number of attempts and the probability of success. The `ax.vlines` function is used to plot vertical lines and `rv.pmf` within it helps in calculating the probability at various values of `x`. The `ax.legend` function adds a legend to the graph, and finally, `plt.show` displays the graph. The result is as follows:

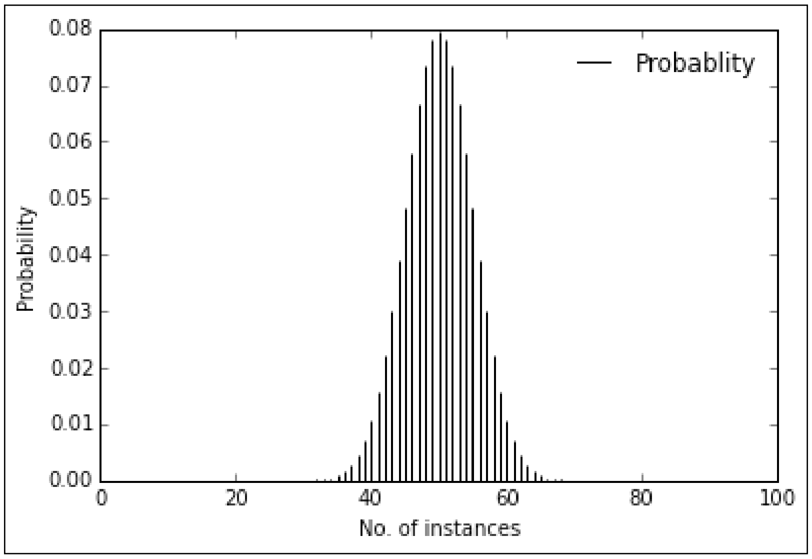


As you can see in the graph, if the coin is flipped six times, then getting three heads has the maximum probability, whereas getting a single head or five heads has the least probability.

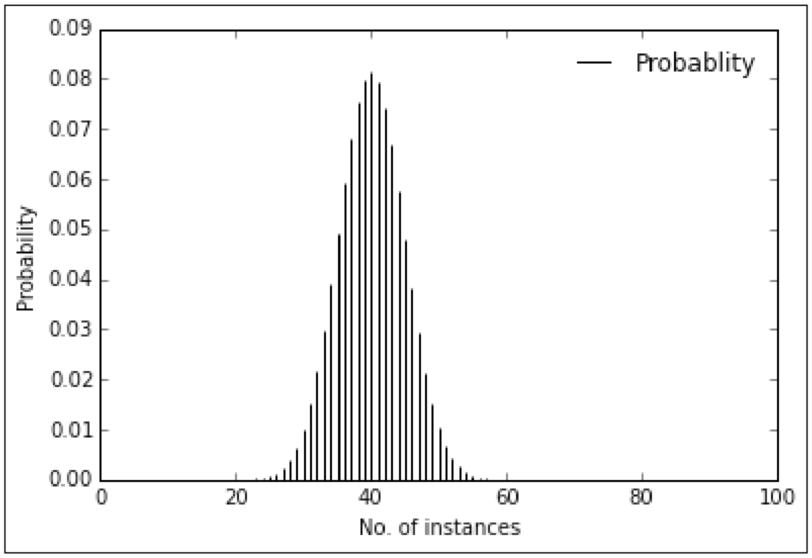
Now, let's increase the number of attempts and see the distribution:

```
>>> fig, ax = plt.subplots(1, 1)
>>> x = range(101)
>>> n, p = 100, 0.5
>>> rv = binom(n, p)
>>> ax.vlines(x, 0, rv.pmf(x), colors='k', linestyles='-', lw=1,
          label='Probablity')

>>> ax.legend(loc='best', frameon=False)
>>> plt.show()
```

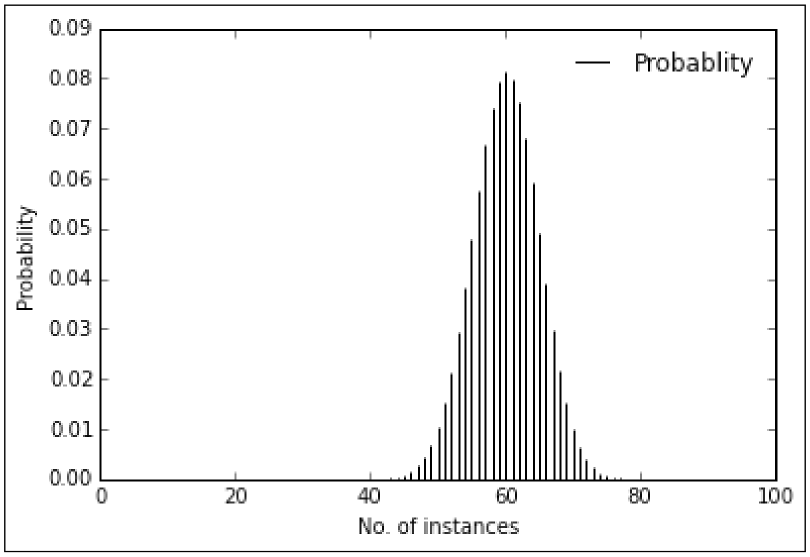
Here, we try to flip the coin 100 times and see the distribution:



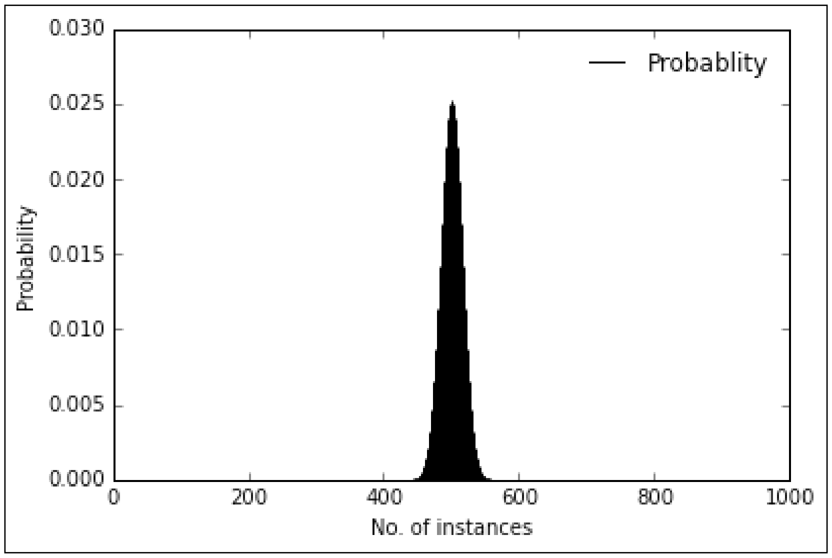
When the probability of success is changed to `0.4`, this is what you see:



When the probability is `0.6`, this is what you see:



When you flip the coin 1000 times at `0.5` probability:



As you can see, the binomial distribution has started to resemble a normal distribution.

A Poisson distribution

A Poisson distribution is the probability distribution of independent interval occurrences in an interval. A binomial distribution is used to determine the probability of binary occurrences, whereas, a Poisson distribution is used for count-based distributions. If lambda is the mean occurrence of the events per interval, then the probability of having a *k* occurrence within a given interval is given by the following formula:

$$f(k;\lambda) = \Pr(X = k) = \frac{\lambda^k e^{-\lambda}}{k!}$$

Here, *e* is the Euler's number, *k* is the number of occurrences for which the probability is going to be determined, and lambda is the mean number of occurrences.

Let's understand this with an example. The number of cars that pass through a bridge in an hour is 20. What would be the probability of 23 cars passing through the bridge in an hour?

For this, we'll use the poisson function from SciPy:

```
>>> from scipy.stats import poisson
>>> rv = poisson(20)
>>> rv.pmf(23)

0.066881473662401172
```

With the Poisson function, we define the mean value, which is 20 cars. The `rv.pmf` function gives the probability, which is around 6%, that 23 cars will pass the bridge.

A Bernoulli distribution

You can perform an experiment with two possible outcomes: success or failure. Success has a probability of *p*, and failure has a probability of *1 - p*. A random variable that takes a *1* value in case of a success and *0* in case of failure is called a Bernoulli distribution. The probability distribution function can be written as:

$$P(n) = \begin{cases} 1-p & \text{for } n = 0 \\ p & \text{for } n = 1 \end{cases}$$

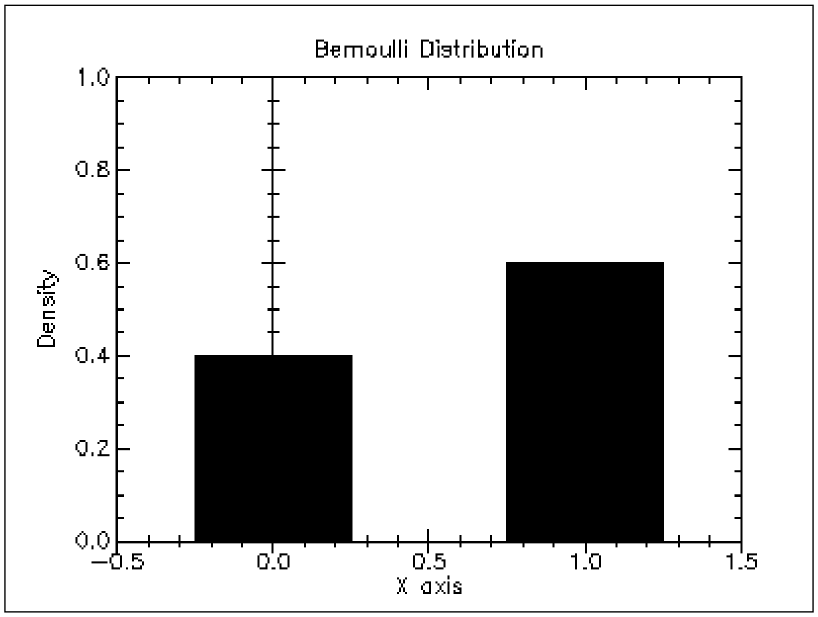
It can also be written like this:

$$P(n) = p^n (1-p)^{1-n}$$

The distribution function can be written like this:

$$D(n) = \begin{cases} 1-p & \text{for } n = 0 \\ 1 & \text{for } n = 1 \end{cases}$$

Following plot shows a Bernoulli distribution:



Voting in an election is a good example of the Bernoulli distribution.

A Bernoulli distribution can be generated using the `bernoulli.rvs()` function of the SciPy package. The following function generates a Bernoulli distribution with a probability of 0.7:

```
>>> from scipy import stats
>>> stats.bernoulli.rvs(0.7, size=100)
array([1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
         1, 0,
       1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1,
         1, 0,
       1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0,
         0, 0,
       1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1,
         1, 1,
       1, 0, 1, 1, 1, 0, 1, 1])])
```

If the preceding output is the number of votes for a candidate by people, then the candidate has 70% of the votes.

A z-score

A z-score, in simple terms, is a score that expresses the value of a distribution in standard deviation with respect to the mean. Let's take a look at the following formula that calculates the z-score:

$$z = (X - \mu) / \sigma$$

Here, *X* is the value in the distribution, μ is the mean of the distribution, and σ is the standard deviation of the distribution

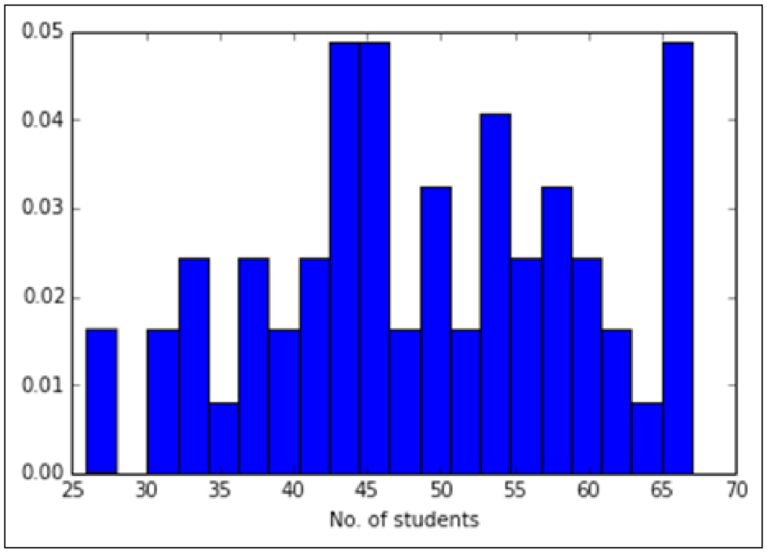
Let's try to understand this concept from the perspective of a school classroom.

A classroom has 60 students in it and they have just got their mathematics examination score. We simulate the score of these 60 students with a normal distribution using the following command:

```
>>> classscore
>>> classscore = np.random.normal(50, 10, 60).round()


[ 56.  52.  60.  65.  39.  49.  41.  51.  48.  52.  47.  41.  60.
    54.  41.
  46.  37.  50.  50.  55.  47.  53.  38.  42.  42.  57.  40.  45.
    35.  39.
  67.  56.  35.  45.  47.  52.  48.  53.  53.  50.  61.  60.  57.
    53.  56.
  68.  43.  35.  45.  42.  33.  43.  49.  54.  45.  54.  48.  55.
    56.  30.]
```

The NumPy package has a random module that has a normal function, where 50 is given as the mean of the distribution, 10 is the standard deviation of the distribution, and 60 is the number of values to be generated. You can plot the normal distribution with the following commands:

```
>>> plt.hist(classscore, 30, normed=True) #Number of breaks is 30
>>> plt.show()
```



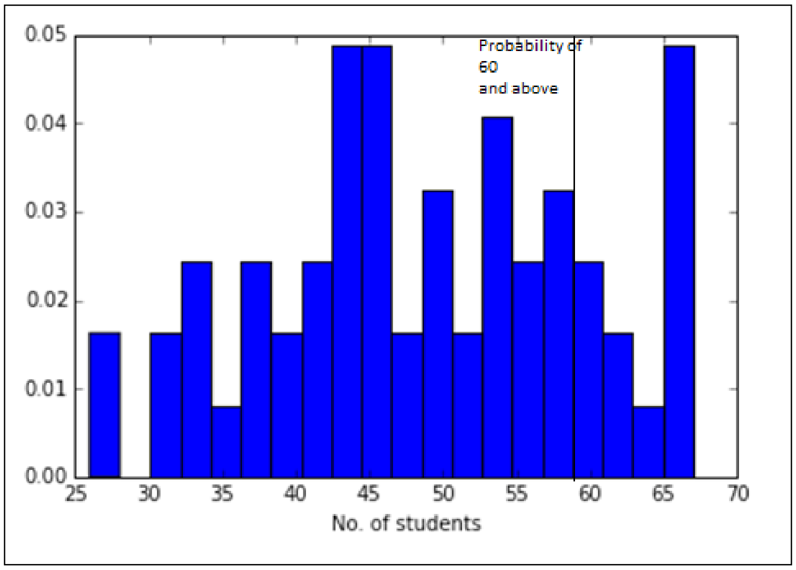
The score of each student can be converted to a z-score using the following functions:

```
>>> stats.zscore(classscore)


[ 0.86008868  0.38555699  1.33462036  1.92778497 -1.15667098
  0.02965823
 -0.91940514  0.26692407 -0.08897469  0.38555699 -0.20760761 -
  0.91940514
  1.33462036  0.62282284 -0.91940514 -0.32624053 -1.39393683
  0.14829115
  0.14829115  0.74145576 -0.20760761  0.50418992 -1.2753039  -
  0.80077222
 -0.80077222  0.9787216  -1.03803806 -0.44487345 -1.63120267 -
  1.15667098
  2.16505081  0.86008868 -1.63120267 -0.44487345 -0.20760761
  0.38555699
 -0.08897469  0.50418992  0.50418992  0.14829115  1.45325329
  1.33462036
  0.9787216   0.50418992  0.86008868  2.28368373 -0.6821393  -
  1.63120267
 -0.44487345 -0.80077222 -1.86846851 -0.6821393   0.02965823
  0.62282284
 -0.44487345  0.62282284 -0.08897469  0.74145576  0.86008868 -
  2.22436727]
```

So, a student with a score of 60 out of 100 has a z-score of 1.334. To make more sense of the z-score, we'll use the standard normal table.

This table helps in determining the probability of a score.

We would like to know what the probability of getting a score above 60 would be.



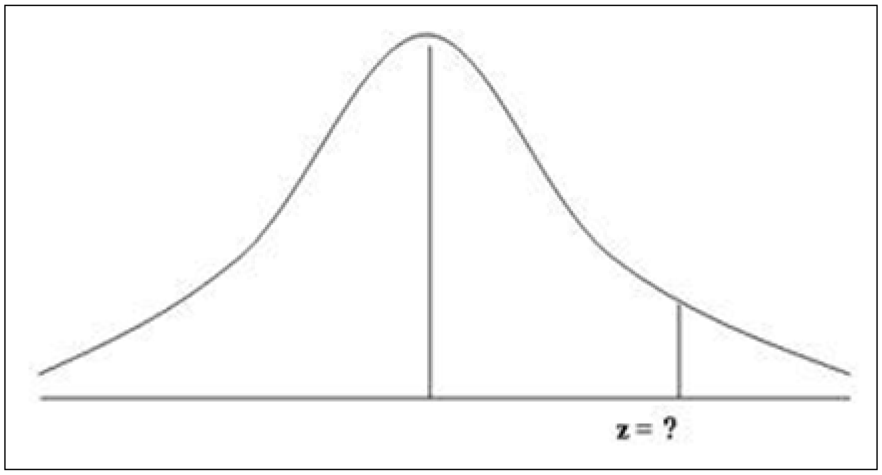
The standard normal table can help us in determining the probability of the occurrence of the score, but we do not have to perform the cumbersome task of finding the value by looking through the table and finding the probability. This task is made simple by the `cdf` function, which is the cumulative distribution function:

```
>>> prob = 1 - stats.norm.cdf(1.334)
>>> prob

0.091101928265359899
```

The `cdf` function gives the probability of getting values up to the z-score of `1.334`, and doing a minus one of it will give us the probability of getting a z-score, which is above it. In other words, 0.09 is the probability of getting marks above 60.

Let's ask another question, "how many students made it to the top 20% of the class?"

Here, we'll have to work backwards to determine the marks at which all the students above it are in the top 20% of the class:



Now, to get the z-score at which the top 20% score marks, we can use the `ppf` function in SciPy:

```
>>> stats.norm.ppf(0.80)


0.8416212335729143
```

The z-score for the preceding output that determines whether the top 20% marks are at 0.84 is as follows:

```
>>> (0.84 * classscore.std()) + classscore.mean()


55.942594176524267
```

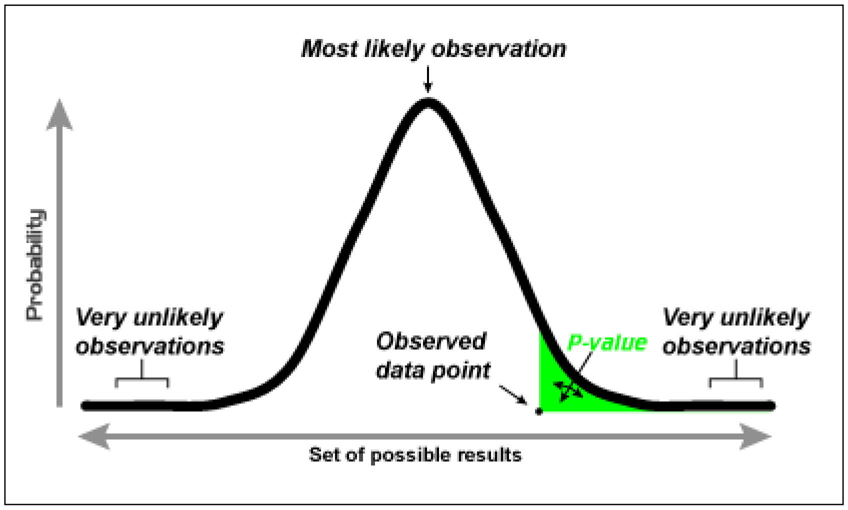
We multiply the z-score with the standard deviation and then add the result with the mean of the distribution. This helps in converting the z-score to a value in the distribution. The `55.83` marks means that students who have marks more than this are in the top 20% of the distribution.

The z-score is an essential concept in statistics, which is widely used. Now you can understand that it is basically used in standardizing any distribution so that it can be compared or inferences can be derived from it.

A p-value

A p-value is the probability of rejecting a null-hypothesis when the hypothesis is proven true. The null hypothesis is a statement that says that there is no difference between two measures. If the hypothesis is that people who clock in 4 hours of study everyday score more that 90 marks out of 100. The null hypothesis here would be that there is no relation between the number of hours clocked in and the marks scored.

If the p-value is equal to or less than the significance level (*a*), then the null hypothesis is inconsistent and it needs to be rejected.



Let's understand this concept with an example where the **null hypothesis** is that it is common for students to score 68 marks in mathematics.

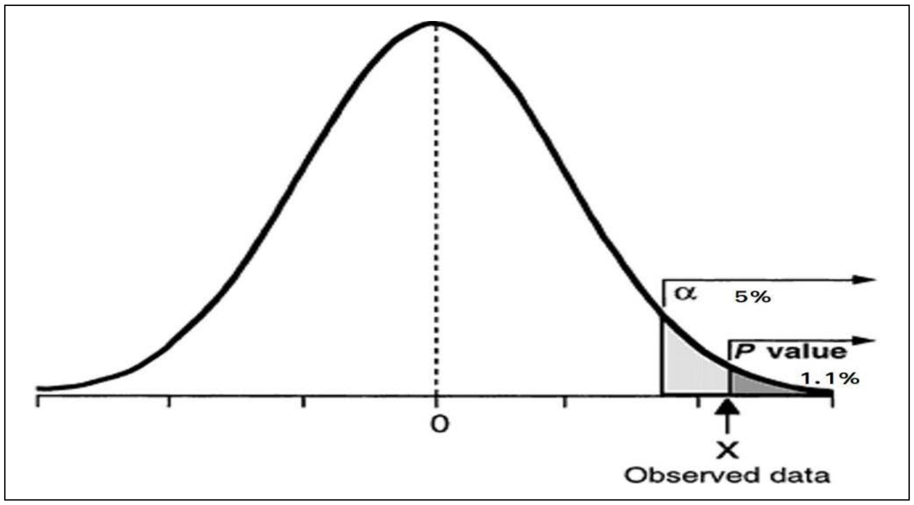
Let's define the significance level at 5%. If the p-value is less than 5%, then the null hypothesis is rejected and it is not common to score 68 marks in mathematics.

Let's get the z-score of 68 marks:

```
>>> zscore = ( 68 - classscore.mean() ) / classscore.std()
>>> zscore
2.283
```

Now, let's get the value:

```
>>> prob = 1 - stats.norm.cdf(zscore)
>>> prob
0.032835182628040638
```

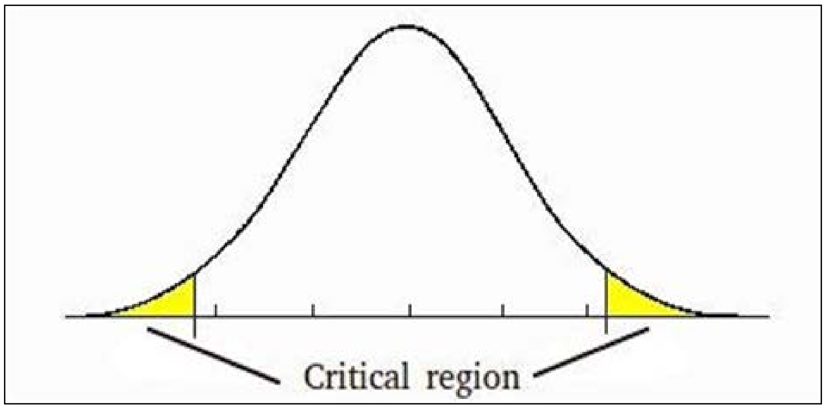


So, you can see that the p-value is at 3.2%, which is lower than the significance level. This means that the null hypothesis can be rejected, and it can be said that it's not common to get 68 marks in mathematics.

One-tailed and two-tailed tests

The example in the previous section was an instance of a one-tailed test where the null hypothesis is rejected or accepted based on one direction of the normal distribution.

In a two-tailed test, both the tails of the null hypothesis are used to test the hypothesis.



In a two-tailed test, when a significance level of 5% is used, then it is distributed equally in the both directions, that is, 2.5% of it in one direction and 2.5% in the other direction.

Let's understand this with an example. The mean score of the mathematics exam at a national level is 60 marks and the standard deviation is 3 marks.

The mean marks of a class are 53. The null hypothesis is that the mean marks of the class are similar to the national average. Let's test this hypothesis by first getting the z-score 60:

```
>>> zscore = ( 53 - 60 ) / 3.0
>>> zscore
-2.3333333333333335
```

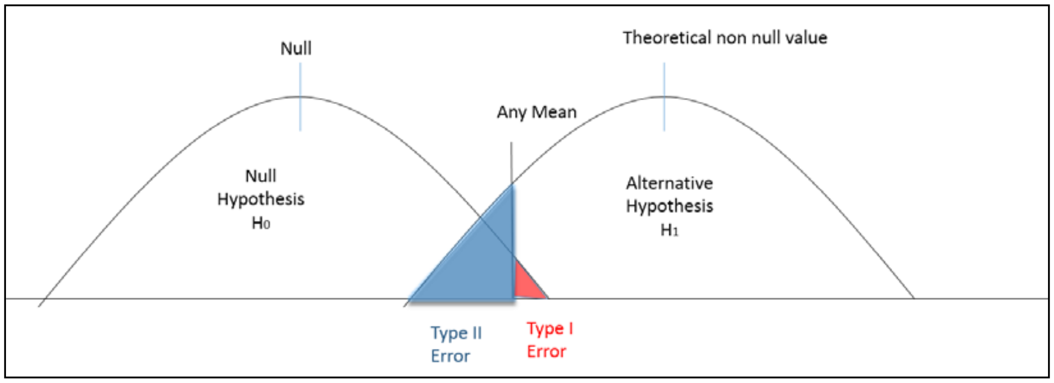
The p-value would be:

```
>>> prob = stats.norm.cdf(zscore)
>>> prob

0.0098153286286453336
```

So, the p-value is 0.98%. The null hypothesis is to be rejected, and the p-value should be less than 2.5% in either direction of the bell curve. Since the p-value is less than 2.5%, we can reject the null hypothesis and clearly state that the average marks of the class are significantly different from the national average.

Type 1 and Type 2 errors

Type 1 error is a type of error that occurs when there is a rejection of the null hypothesis when it is actually true. This kind of error is also called an error of the first kind and is equivalent to false positives.



Let's understand this concept using an example. There is a new drug that is being developed and it needs to be tested on whether it is effective in combating diseases. The null hypothesis is that it is not effective in combating diseases.

The significance level is kept at 5% so that the null hypothesis can be accepted confidently 95% of the time. However, 5% of the time, we'll accept the rejecttion of the hypothesis although it had to be accepted, which means that even though the drug is ineffective, it is assumed to be effective.

The Type 1 error is controlled by controlling the significance level, which is alpha. Alpha is the highest probability to have a Type 1 error. The lower the alpha, the lower will be the Type 1 error.

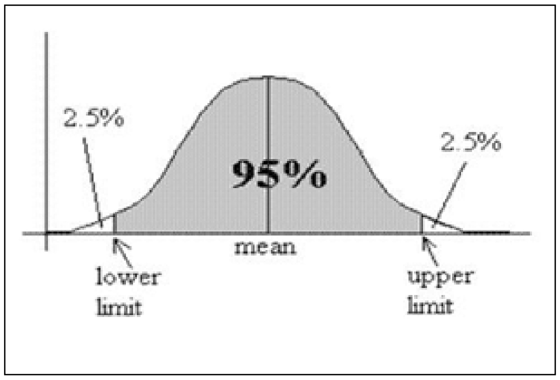
The Type 2 error is the kind of error that occurs when we do not reject a null hypothesis that is false. This error is also called the error of the second kind and is equivalent to a false negative.

This kind of error occurs in a drug scenario when the drug is assumed to be ineffective but is actually it is effective.

These errors can be controlled one at a time. If one of the errors is lowered, then the other one increases. It depends on the use case and the problem statement that the analysis is trying to address, and depending on it, the appropriate error should reduce. In the case of this drug scenario, typically, a Type 1 error should be lowered because it is better to ship a drug that is confidently effective.

A confidence interval

A confidence interval is a type of interval statistics for a population parameter. The confidence interval helps in determining the interval at which the population mean can be defined.



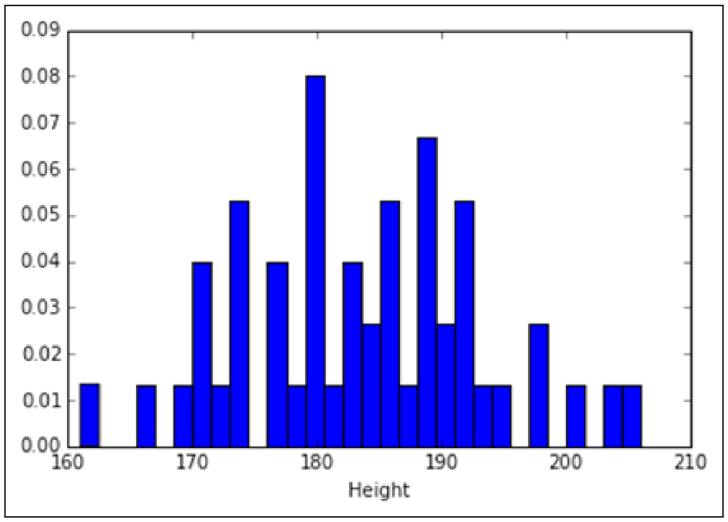
Let's try to understand this concept by using an example. Let's take the height of every man in Kenya and determine with 95% confidence interval the average of height of Kenyan men at a national level.

Let's take 50 men and their height in centimeters:

```
>>> height_data = np.array([ 186.0, 180.0, 195.0, 189.0, 191.0,
    177.0, 161.0, 177.0, 192.0, 182.0, 185.0, 192.0,

  173.0, 172.0, 191.0, 184.0, 193.0, 182.0, 190.0, 185.0, 181.0,
    188.0, 179.0, 188.0,

  170.0, 179.0, 180.0, 189.0, 188.0, 185.0, 170.0, 197.0, 187.0,
    182.0, 173.0, 179.0,

  184.0, 177.0, 190.0, 174.0, 203.0, 206.0, 173.0, 169.0, 178.0,
    201.0, 198.0, 166.0,

  171.0, 180.0])
```

Plotting the distribution, it has a normal distribution:

```
>>> plt.hist(height_data, 30, normed=True)
>>> plt.show()
```



The mean of the distribution is as follows:

```
>>> height_data.mean()

183.24000000000001
```

So, the average height of a man from the sample is 183.4 cm.

To determine the confidence interval, we'll now define the standard error of the mean.

The standard error of the mean is the deviation of the sample mean from the population mean. It is defined using the following formula:

$$\mathrm{SE}_{\bar{x}} = \frac{s}{\sqrt{n}}$$

Here, *s* is the standard deviation of the sample, and *n* is the number of elements of the sample.

This can be calculated using the `sem()` function of the SciPy package:

```
>>> stats.sem(height_data)


1.3787187190005252
```

So, there is a standard error of the mean of 1.38 cm. The lower and upper limit of the confidence interval can be determined by using the following formula:

*Upper/Lower limit = mean(height) + / - sigma * SEmean(x)*

For lower limit:

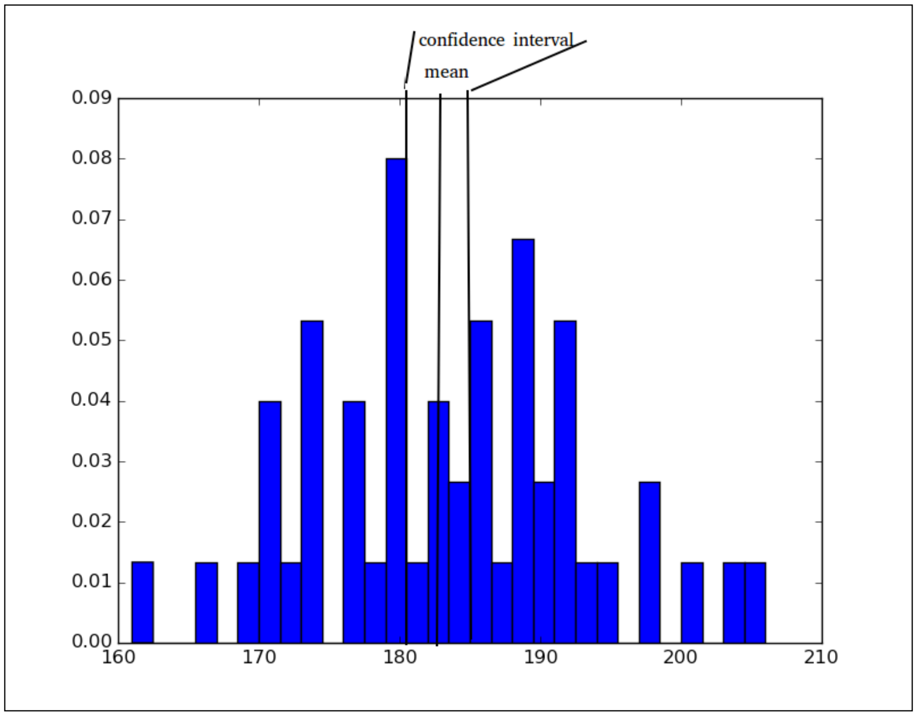
*183.24 + (1.96 * 1.38) = 185.94*

For upper limit:

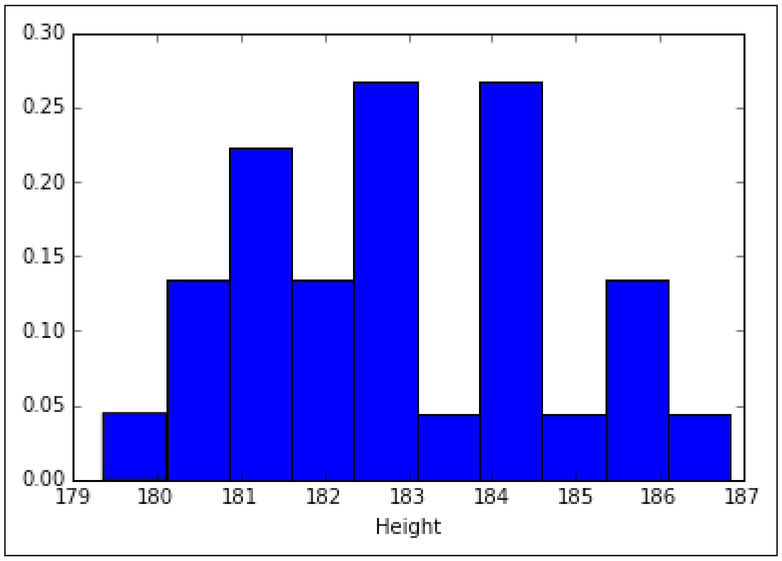
*183.24 - (1.96*1.38) = 180.53*

A 1.96 standard deviation covers 95% of area in the normal distribution.

We can confidently say that the population mean lies between 180.53 cm and 185.94 cm of height.



Let's assume we take a sample of 50 people, record their height, and then repeat this process 30 times. We can then plot the averages of each sample and observe the distribution.



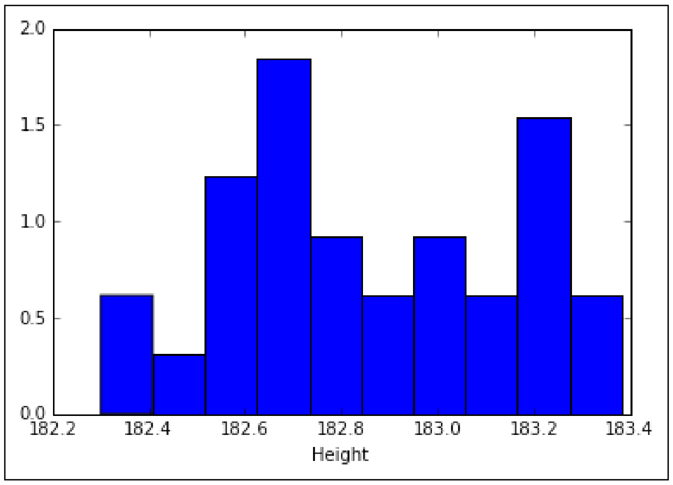
The commands that simulated the preceding plot is as follows:

```
>>> average_height = []
>>> for i in xrange(30):
>>>     sample50 = np.random.normal(183, 10, 50).round()
>>>     average_height.append(sample50.mean())


>>> plt.hist(average_height, 20, normed=True)
>>> plt.show()
```

You can observe that the mean ranges from 180 to 187 cm when we simulated the average height of 50 sample men, which was taken 30 times.

Let's see what happens when we sample 1000 men and repeat the process 30 times:

```
>>> average_height = []
>>> for i in xrange(30):
>>>     sample1000 = np.random.normal(183, 10, 1000).round()
>>>     average_height.append(sample1000.mean())


>>> plt.hist(average_height, 10, normed=True)
>>> plt.show()
```



As you can see, the height varies from 182.4 cm and to 183.4 cm. What does this mean?

It means that as the sample size increases, the standard error of the mean decreases, which also means that the confidence interval becomes narrower, and we can tell with certainty the interval that the population mean would lie on.

Correlation

In statistics, correlation defines the similarity between two random variables. The most commonly used correlation is the Pearson correlation and it is defined by the following:

$$\rho_{X,Y} = \frac{\operatorname{cov}(X,Y)}{\sigma_X \sigma_Y} = \frac{E\left[(X-\mu_X)(Y-\mu_Y)\right]}{\sigma_X \sigma_Y}$$

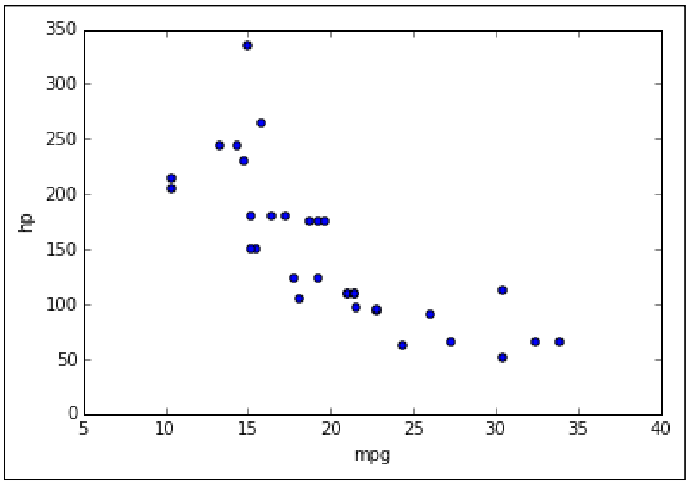
The preceding formula defines the Pearson correlation as the covariance between *X* and *Y*, which is divided by the standard deviation of *X* and *Y*, or it can also be defined as the expected mean of the sum of multiplied difference of random variables with respect to the mean divided by the standard deviation of X and Y. Let's understand this with an example. Let's take the mileage and horsepower of various cars and see if there is a relation between the two. This can be achieved using the `pearsonr` function in the SciPy package:

```
>>> mpg = [21.0, 21.0, 22.8, 21.4, 18.7, 18.1, 14.3, 24.4, 22.8,
       19.2, 17.8, 16.4, 17.3, 15.2, 10.4, 10.4, 14.7, 32.4, 30.4,

       33.9, 21.5, 15.5, 15.2, 13.3, 19.2, 27.3, 26.0, 30.4, 15.8,
       19.7, 15.0, 21.4]
>>> hp = [110, 110, 93, 110, 175, 105, 245, 62, 95, 123, 123, 180,
      180, 180, 205, 215, 230, 66, 52, 65, 97, 150, 150, 245,

      175, 66, 91, 113, 264, 175, 335, 109]


>>> stats.pearsonr(mpg,hp)


(-0.7761683718265864, 1.7878352541210661e-07)
```

The first value of the output gives the correlation between the horsepower and the mileage and the second value gives the p-value.

So, the first value tells us that it is highly negatively correlated and the p-value tells us that there is significant correlation between them:

```
>>> plt.scatter(mpg, hp)
>>> plt.show()
```



From the plot, we can see that as the mpg increases, the horsepower decreases.

Let's look into another correlation called the Spearman correlation. The Spearman correlation applies to the rank order of the values and so it provides a monotonic relation between the two distributions. It is useful for ordinal data (data that has an order, such as movie ratings or grades in class) and is not affected by outliers.

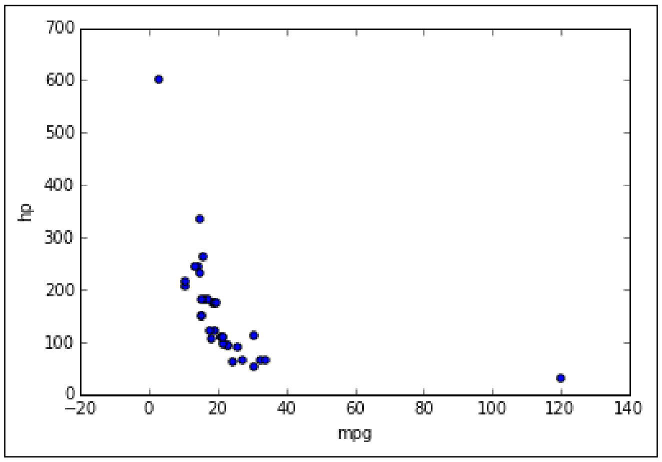
Let's get the Spearman correlation between the miles per gallon and horsepower. This can be achieved using the `spearmanr()` function in the SciPy package:

```
>>> stats.spearmanr(mpg,hp)


(-0.89466464574996252, 5.085969430924539e-12)
```

We can see that the Spearman correlation is -0.89 and the p-value is significant.

Let's do an experiment in which we introduce a few outlier values in the data and see how the Pearson and Spearman correlation gets affected:

```
>>> mpg = [21.0, 21.0, 22.8, 21.4, 18.7, 18.1, 14.3, 24.4, 22.8,
       19.2, 17.8, 16.4, 17.3, 15.2, 10.4, 10.4, 14.7, 32.4, 30.4,

       33.9, 21.5, 15.5, 15.2, 13.3, 19.2, 27.3, 26.0, 30.4, 15.8,
       19.7, 15.0, 21.4, 120, 3]
>>> hp = [110, 110, 93, 110, 175, 105, 245, 62, 95, 123, 123, 180,
      180, 180, 205, 215, 230, 66, 52, 65, 97, 150, 150, 245,

      175, 66, 91, 113, 264, 175, 335, 109, 30, 600]


>>> plt.scatter(mpg, hp)
>>> plt.show()
```



From the plot, you can clearly make out the outlier values. Lets see how the correlations get affected for both the Pearson and Spearman correlation

The following commands show you the Pearson correlation:

```
>>> stats.pearsonr(mpg, hp)
>>> (-0.47415304891435484, 0.0046122167947348462)
```

Here is the Spearman correlation:

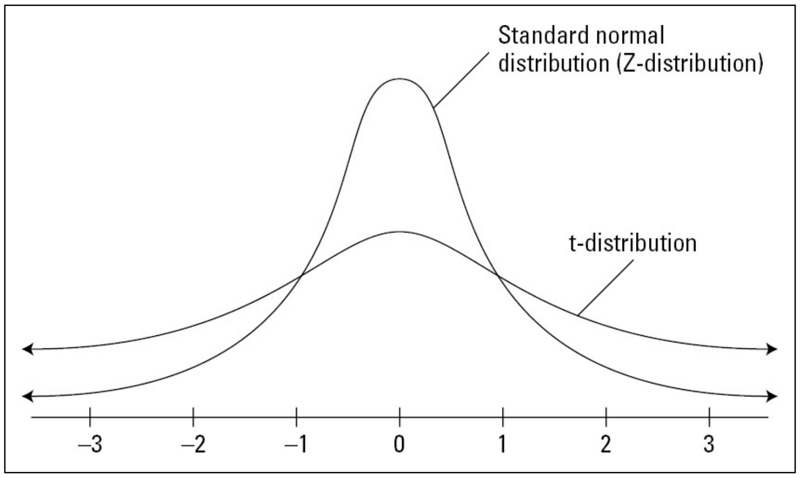
```
>>> stats.spearmanr(mpg, hp)
>>> (-0.91222184337265655, 6.0551681657984803e-14)
```

We can clearly see that the Pearson correlation has been drastically affected due to the outliers, which are from a correlation of 0.89 to 0.47.

The Spearman correlation did not get affected much as it is based on the order rather than the actual value in the data.

Z-test vs T-test

We have already done a few Z-tests before where we validated our null hypothesis.



A T-distribution is similar to a Z-distribution—it is centered at zero and has a basic bell shape, but its shorter and flatter around the center than the Z-distribution.

The T-distributions' standard deviation is usually proportionally larger than the Z, because of which you see the fatter tails on each side.

The t distribution is usually used to analyze the population when the sample is small.

The Z-test is used to compare the population mean against a sample or compare the population mean of two distributions with a sample size greater than 30. An example of a Z-test would be comparing the heights of men from different ethnicity groups.

The T-test is used to compare the population mean against a sample, or compare the population mean of two distributions with a sample size less than 30, and when you don't know the population's standard deviation.

Let's do a T-test on two classes that are given a mathematics test and have 10 students in each class:

```
>>> class1_score = np.array([45.0, 40.0, 49.0, 52.0, 54.0, 64.0,
                    36.0, 41.0, 42.0, 34.0])


>>> class2_score = np.array([75.0, 85.0, 53.0, 70.0, 72.0, 93.0,
                    61.0, 65.0, 65.0, 72.0])
```

To perform the T-test, we can use the `ttest_ind()` function in the SciPy package:

```
>>> stats.ttest_ind(class1_score,class2_score)


(array(-5.458195056848407), 3.4820722850153292e-05)
```

The first value in the output is the calculated t-statistics, whereas the second value is the p-value and p-value shows that the two distributions are not identical.

The F distribution

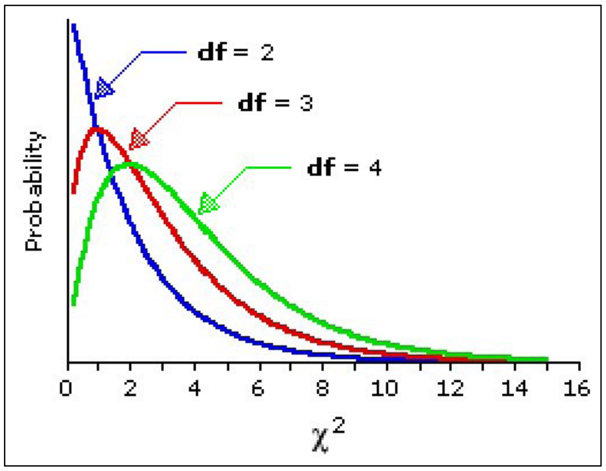
The F distribution is also known as Snedecor's F distribution or the Fisher–Snedecor distribution.

An f statistic is given by the following formula:

$$f = \left[s_1^2 / \sigma_1^2 \right] / \left[s_2^2 / \sigma_2^2 \right]$$

Here, s_1 is the standard deviation of a sample 1 with an n_1 size, s_2 is the standard deviation of a sample 2, where the size $n_2\sigma_1$ is the population standard deviation of a sample $1\sigma_2$ is the population standard deviation of a sample 12.

The distribution of all the possible values of **f** statistics is called F distribution. The `d1` and `d2` represent the degrees of freedom in the following chart:



The chi-square distribution

The chi-square statistics are defined by the following formula:

$$X^2 = \left[(n-1)*s^2\right]/\sigma^2$$

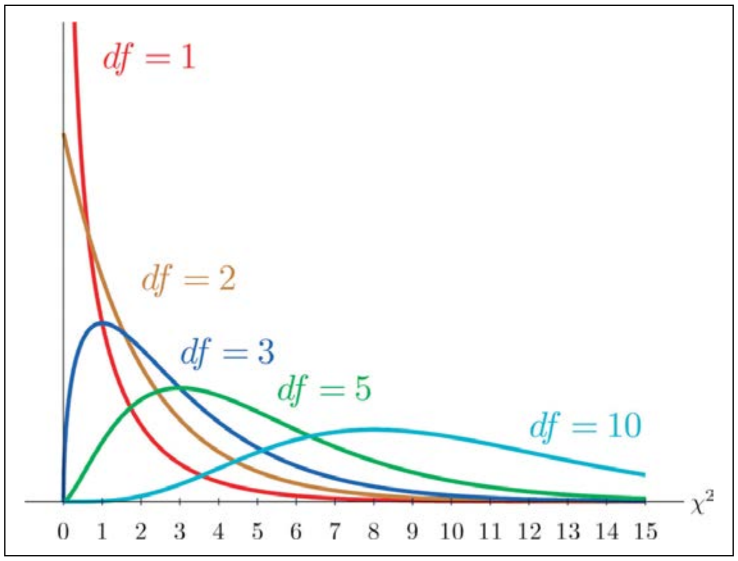
Here, *n* is the size of the sample, *s* is the standard deviation of the sample, and *σ* is the standard deviation of the population.

If we repeatedly take samples and define the chi-square statistics, then we can form a chi-square distribution, which is defined by the following probability density function:

$$Y = Y_0 * \left(X^2\right)^{(v/2-1)} * e^{-X2/2}$$

Here, Y_0 is a constant that depends on the number of degrees of freedom, X_2 is the chi-square statistic, *v* = *n - 1* is the number of degrees of freedom, and *e* is a constant equal to the base of the natural logarithm system.

Y_0 is defined so that the area under the chi-square curve is equal to one.



Chi-square for the goodness of fit

The Chi-square test can be used to test whether the observed data differs significantly from the expected data. Let's take the example of a dice. The dice is rolled 36 times and the probability that each face should turn upwards is 1/6. So, the expected distribution is as follows:

Expected Frequency	**Outcome**
6	1
6	2
6	3
6	4
6	5
6	6

```
>>> expected = np.array([6,6,6,6,6,6])
```

The observed distribution is as follows:

Observed Frequency	**Outcome**
7	1
5	2
3	3
9	4
6	5
6	6

```
>>> observed = observed = np.array([7, 5, 3, 9, 6, 6])
```

The null hypothesis in the chi-square test is that the observed value is similar to the expected value.

The chi-square can be performed using the `chisquare` function in the SciPy package:

```
>>> stats.chisquare(observed,expected)
(3.3333333333333333, 0.64874235866759344)
```

The first value is the chi-square value and the second value is the p-value, which is very high. This means that the null hypothesis is valid and the observed value is similar to the expected value.

The chi-square test of independence

The chi-square test of independence is a statistical test used to determine whether two categorical variables are independent of each other or not.

Let's take the following example to see whether there is a preference for a book based on the gender of people reading it:

Flavour				
Total	**Biography**	**Suspense**	**Romance**	**Gender**
280	60	120	100	Men
640	90	200	350	Women
920	150	320	450	

The Chi-Square test of independence can be performed using the `chi2_contingency` function in the SciPy package:

```
>>> men_women = np.array([[100, 120, 60],[350, 200, 90]])
>>> stats.chi2_contingency(men_women)
(28.362103174603167, 6.9382117170577439e-07, 2, array([[
       136.95652174,   97.39130435,   45.65217391],

       [ 313.04347826,  222.60869565,  104.34782609]]))
```

The first value is the chi-square value:

The second value is the p-value, which is very small, and means that there is an association between the gender of people and the genre of the book they read. The third value is the degrees of freedom. The fourth value, which is an array, is the expected frequencies.

ANOVA

Analysis of Variance (**ANOVA**) is a statistical method used to test differences between two or more means.

This test basically compares the means between groups and determines whether any of these means are significantly different from each other:

$$H_0 : \mu_1 = \mu_2 = \mu_3 = \cdots = \mu_k$$

ANOVA is a test that can tell you which group is significantly different from each other. Let's take the height of men who are from three different countries and see if their heights are significantly different from others:

```
>>> country1 = np.array([ 176.,  179.,  180.,  188.,  187.,  184.,  171.,
        201.,  172.,
        181.,  192.,  187.,  178.,  178.,  180.,  199.,  185.,  176.,
        207.,  177.,  160.,  174.,  176.,  192.,  189.,  187.,  183.,
        180.,  181.,  200.,  190.,  187.,  175.,  179.,  181.,  183.,
        171.,  181.,  190.,  186.,  185.,  188.,  201.,  192.,  188.,
        181.,  172.,  191.,  201.,  170.,  170.,  192.,  185.,  167.,
        178.,  179.,  167.,  183.,  200.,  185.])


>>> country2 = np.array([ 177.,  165.,  175.,  172.,  179.,  192.,  169.,
                        185.,  187.,
        167.,  162.,  165.,  188.,  194.,  187.,  175.,  163.,  178.,
        197.,  172.,  175.,  185.,  176.,  171.,  172.,  186.,  168.,
        178.,  191.,  192.,  175.,  189.,  178.,  181.,  170.,  182.,
        166.,  189.,  196.,  192.,  189.,  171.,  185.,  198.,  181.,
        167.,  184.,  179.,  178.,  193.,  179.,  177.,  181.,  174.,
        171.,  184.,  156.,  180.,  181.,  187.])


>>> country3 = np.array([ 191.,  190.,  191.,  185.,  190.,  184.,
        173.,  175.,  200.,
        190.,  191.,  184.,  167.,  194.,  195.,  174.,  171.,  191.,
        174.,  177.,  182.,  184.,  176.,  180.,  181.,  186.,  179.,
        176.,  186.,  176.,  184.,  194.,  179.,  171.,  174.,  174.,
        182.,  198.,  180.,  178.,  200.,  200.,  174.,  202.,  176.,
        180.,  163.,  159.,  194.,  192.,  163.,  194.,  183.,  190.,
        186.,  178.,  182.,  174.,  178.,  182.])
```

To perform the one-way ANOVA, we can use the `f_oneway()` function of the SciPy package:

```
>>> stats.f_oneway(country1,country2,country3)
(2.9852039682631375, 0.053079678812747652)
```

The first value of the output gives the F-value and the second value gives the p-value. Since the p-value is greater than 5% by a small margin, we can tell that the mean of the heights in the three countries is not significantly different from each other.

Summary

In this chapter, you learned about the various probability distributions. You also learned about how to use z-score, p-value, Type 1, and Type 2 errors. You gained an insight into the Z-test and T-test followed by the chi-square distribution and saw how it can be used to test a hypothesis.

In the next chapter, you'll learn about data mining and how to execute it.