Statistics_basics-fitting_distributions

April 7, 2020

1 Statistics basics

1.1 Introduction

In this tutorial you are going to learn how to use Python tools to visualize and analyze randomly generated data. The purpose is to get acquainted with statistical concepts such as **probability density functions** and **cumulative distribution functions**.

As usual, we start by importing Python packages that we will use later on. In this tutorial, we are using *NumPy* to create random numbers, the set of statistical tools in *SciPy*, and *Matplotlib* and *Seaborn* to plot graphs.

[1]:
```
import numpy as np
from scipy import stats
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
sns.set_context('notebook')
```

1.2 Example: normal distribution

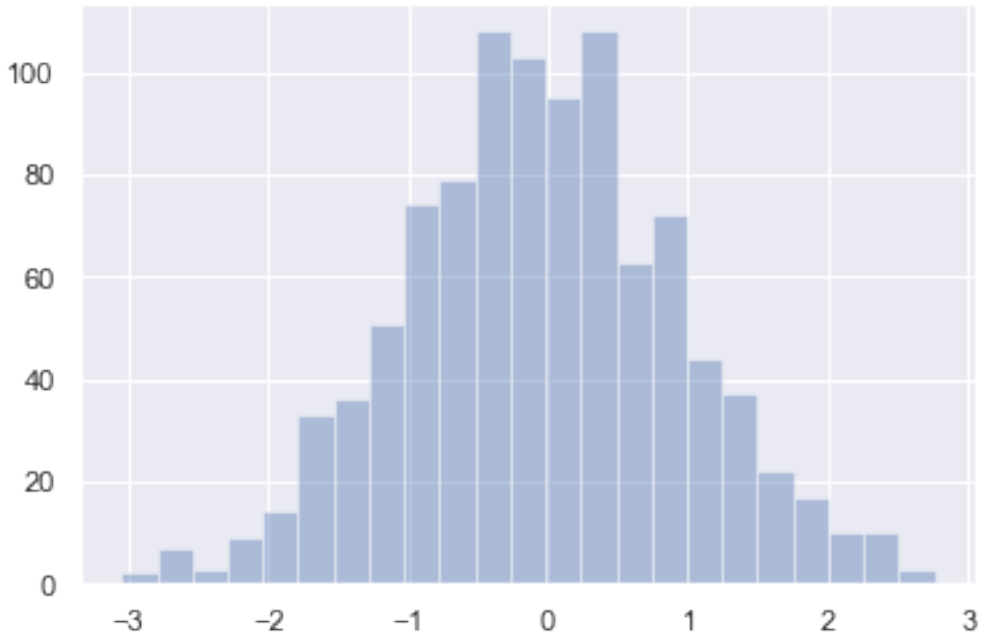
As an example, we create an array of random numbers extracted from a normal distribution of mean (`loc`) 0 and standard deviation (`scale`) 1.

[2]:
```
# create an array of random numbers taken from a normal distribution
np.random.seed(0)
x_rand = np.random.normal(loc=0, scale=1.0, size=1000)
```

1.2.1 Visualization

Let's plot the histogram of the random data with the *Seaborn* function `distplot`.

[3]:
```
# histogram
sns.distplot(x_rand, kde=False);
```

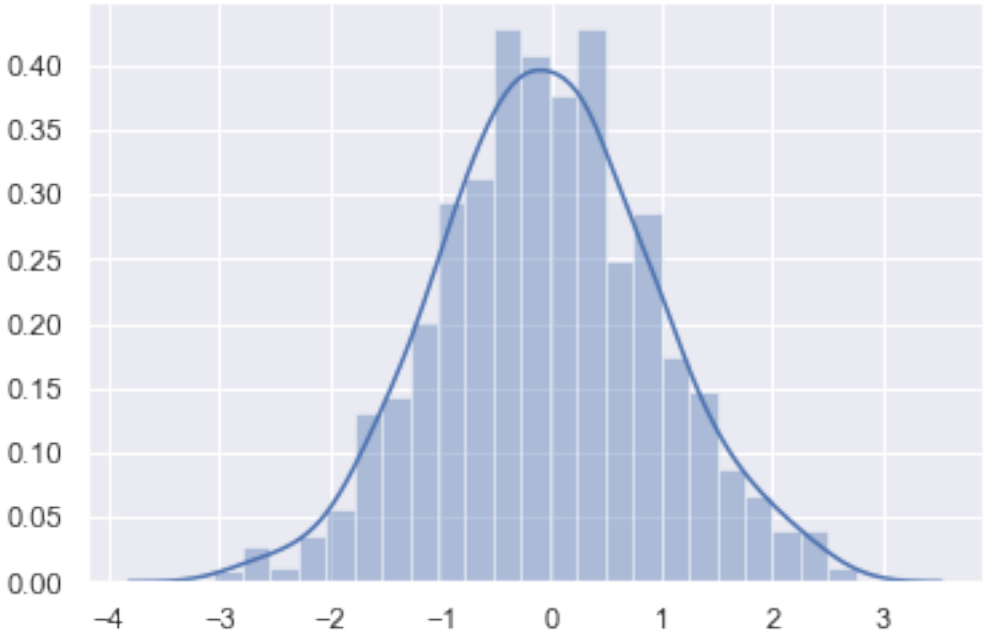


The histogram visualization is sensitive to the number of bins in which the variable range is divided. `distplot` calculates by default the number of bins that it estimates best, but you can set a fixed number of bins and check how the histagram changes.

[4]:
```
# histogram with different number of bins
sns.distplot(x_rand, bins=100, kde=False);
```

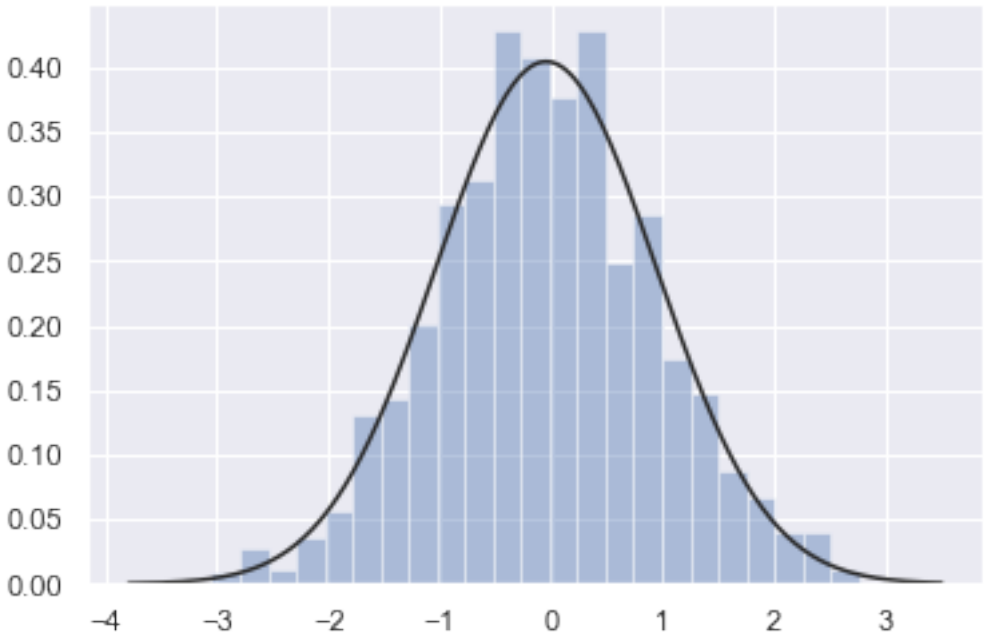
To avoid this limitation, `distplot` uses a function called kernel density estimation (KDE) to create a softened line that represents the shape of the data distribution independent of the number of bins.

[5]:
```
# histogram + kernel density estimate (default)
sns.distplot(x_rand);
```



The kernel density estimate resembles what we expect our distribution to look like, a Gaussian bell. Actually, if we know the distribution of our data, we can fit and plot the distribution with `distplot`.

[6]:
```
# histogram + fitted normal distribution
sns.distplot(x_rand, fit=stats.norm, kde=False);
```



1.2.2 Distribution fitting

Now were are going to use thes statistical tools in *SciPy* (`stats`) to fit a distribution to our data and to learn what are the **probability density function** and the **cumulative distribution function**.

To fit a distribution function in `Scipy.stats` we only need to call the function `fit` to the specific distribution.

[7]:
```
# fit a normal distribution
loc, scale = stats.norm.fit(x_rand)
print('loc = {0}\tscale = {1}'.format(loc, scale))
```

```
loc = -0.045256707490195384     scale = 0.9870331586690257
```

We check that the fit is not perfect, since we generated the random values from a normal distribution $N(0, 1)$, i.e., $loc = 0$ and $scale = 1$. This is due to a limited amount of data, but the fit is good enough.

1.2.3 Probability density function (PDF)

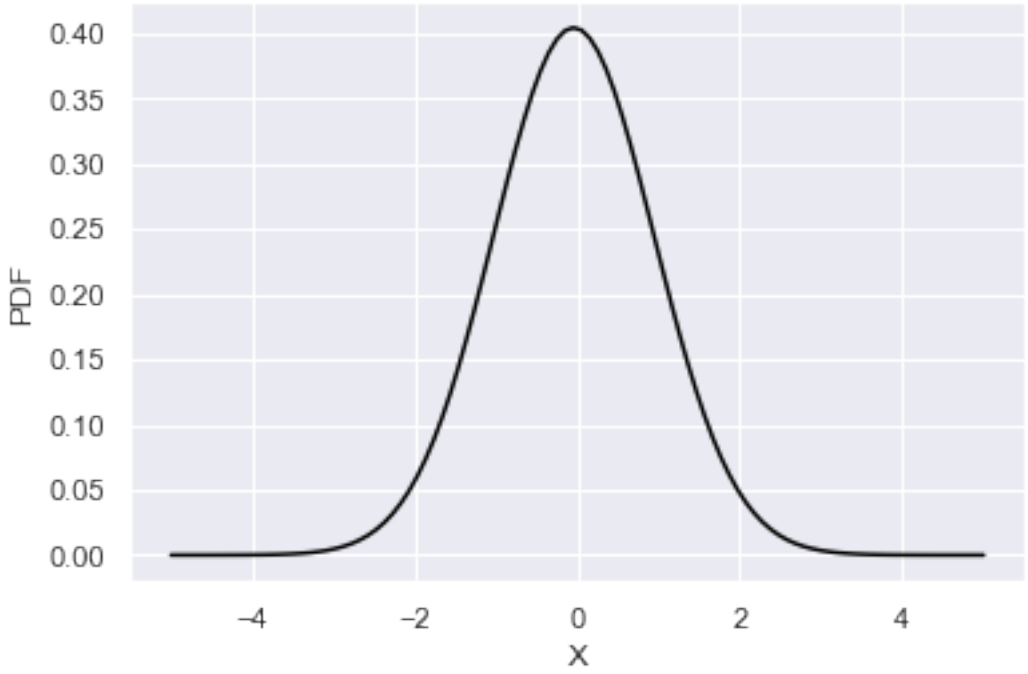
The probability density function represents the *relative likelihood* that the variable takes a specific value. In practical terms, values with a higher PDF are more likely to happen. In the case of the normal distribution, the probability density function is the well-known Gaussian bell. Let's plot it for the distribution we have fitted above.

In `scipy.stats`, we will use the function `pdf`.

[8]:
```
# linearly spaced values of X
x = np.linspace(start=-5, stop=5, num=100)
```

[9]:
```
# PDF (probability density function)
pdf = stats.norm.pdf(x, loc=loc, scale=scale)
```

[10]:
```
# line plot of the PDF
plt.plot(x, pdf, color='black')
plt.xlabel('X')
plt.ylabel('PDF');
```



1.2.4 Cumulative distribution function (CDF)

The cumulative distribution function is the probability that the variable X takes values lower or equal than x; for that reason it is also called the non-exceedance probability. The cumulative distribution function (CDF) is the integral of the probability density function (PDF).

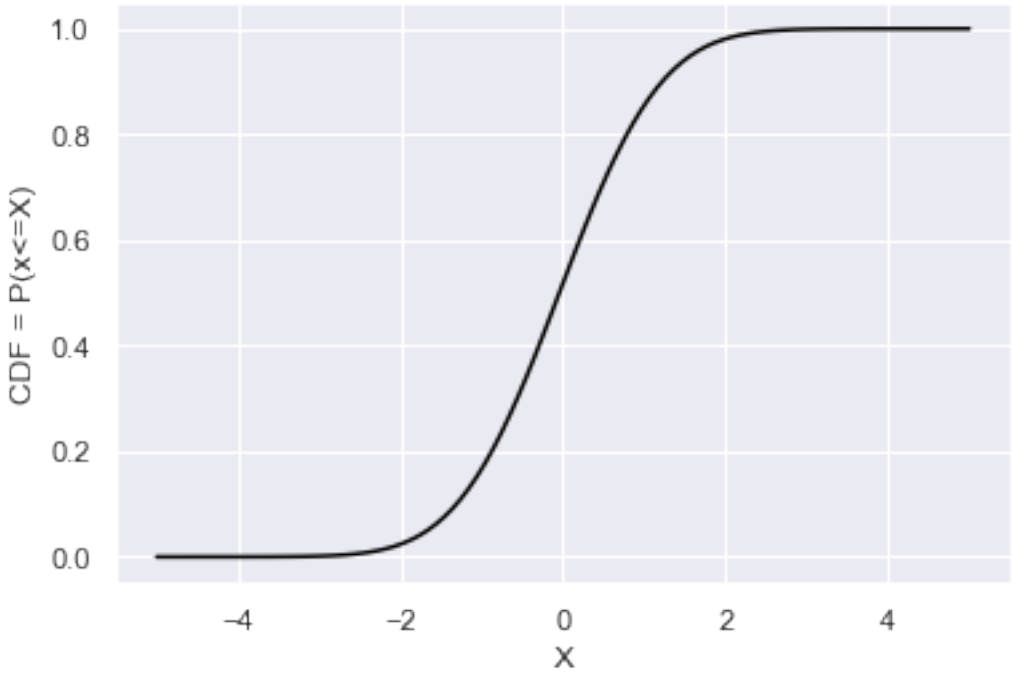
$$CDF = P(x \leq X) = \int PDF$$

The CDF is a monotonic incresing function that ranges between 0 and 1.

In `scipy.stats`, we will use the function `cdf`.

[11]:
```
# CDF (cumulative distribution function)
cdf = stats.norm.cdf(x, loc=loc, scale=scale)
```

[12]:
```
# line plot of the CDF
plt.plot(x, cdf, color='black')
plt.xlabel('X')
plt.ylabel('CDF = P(x<=X)');
```



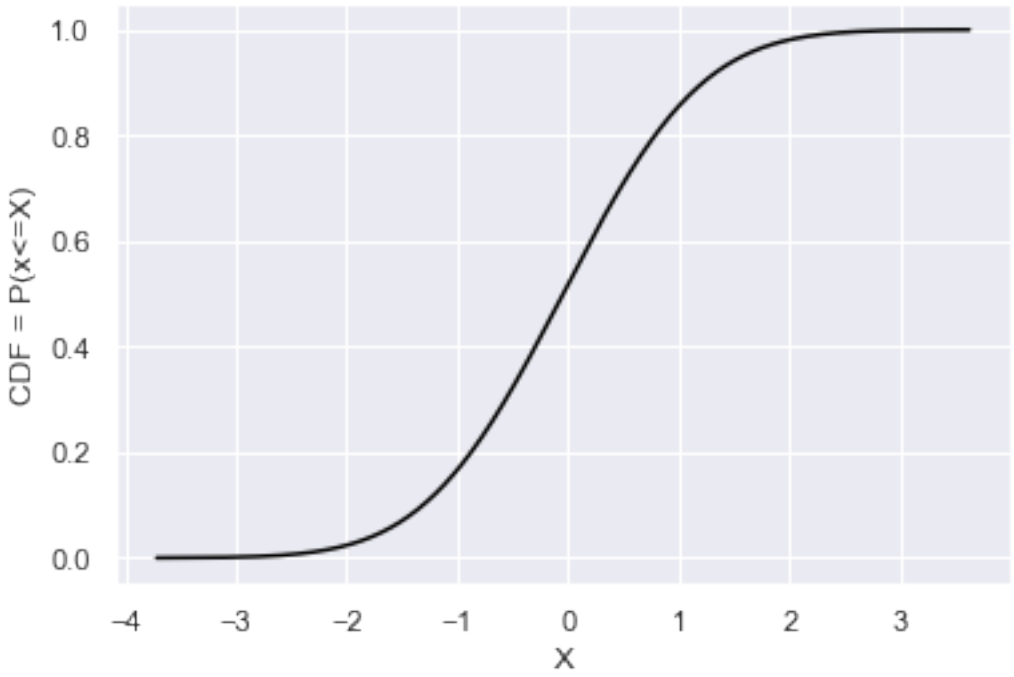
1.2.5 Percent point function (PPF)

`Scipy.stats` includes the function `ppf` (percent point function) that inverts the CDF, that is, it calculates the value of X for a given value of the CDF.

[13]:
```
# linearly spaced values CDF
cdf_ = np.linspace(start=0, stop=1, num=10000)
```

[14]:
```
# x using 'ppf'
x_ = stats.norm.ppf(cdf_, loc=loc, scale=scale)
```

[15]:
```
# line plot of the CDF
plt.plot(x_, cdf_, color='black')
plt.xlabel('X')
plt.ylabel('CDF = P(x<=X)');
```



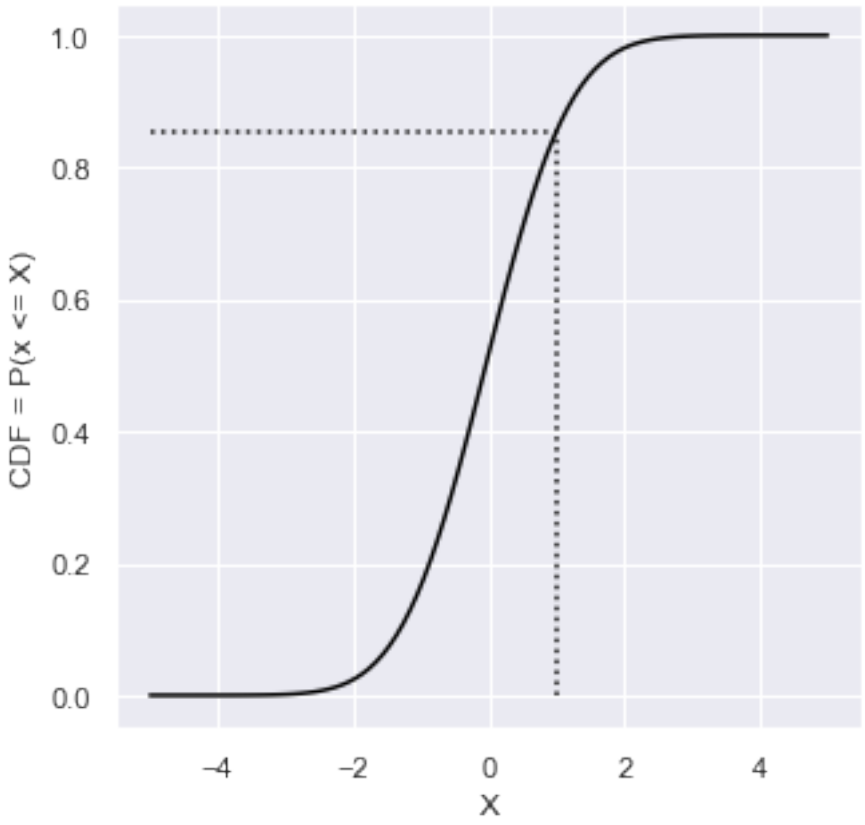
1.2.6 Use of the CDF

The fact that the CDF is a monotonic increasing function allows us to connect the values of the variable X with its probability and viceversa. For instance: 1. What is the probability that the variable X takes a value lower than $x = 1$? $\rightarrow$ **CDF** 2. What is the value x that is not exceeded 99% of the time? $\rightarrow$ **PPF**

[16]:
```
# probability that the variable  takes a value lower than  =1
cdf_1 = stats.norm.cdf(1, loc=loc, scale=scale)
cdf_1
```

[16]:
```
0.8551974783788131
```

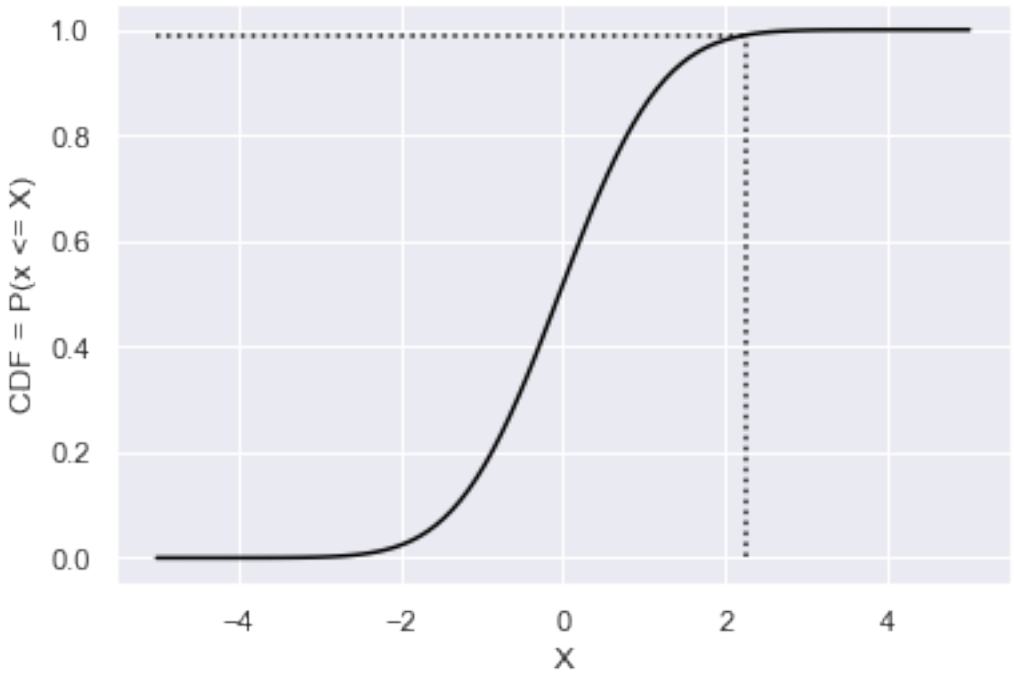
[17]:
```
plt.figure(figsize=(5, 5))
plt.plot(x, cdf, color='black')
plt.vlines(1, 0, cdf_1, linestyle=':')
plt.hlines(cdf_1, -5, 1, linestyle=':')
plt.xlabel('X')
plt.ylabel('CDF = P(x <= X)')
plt.savefig('../output/Statistics basics-CDF.png', dpi=300, tight_layout=True);
```



[18]:
```
# value  that is not exceeded 99% of the time
x_99 = stats.norm.ppf(0.99, loc=loc, scale=scale)
x_99
```

[18]: 2.2509257827873084

[19]:
```
plt.plot(x, cdf, color='black')
plt.hlines(.99, -5, x_99, linestyle=':')
plt.vlines(x_99, 0, .99, linestyle=':')
plt.xlabel('X')
plt.ylabel('CDF = P(x <= X)');
```



1.2.7 Useful link:

Seaborn: visualizing the distribution of a dataset