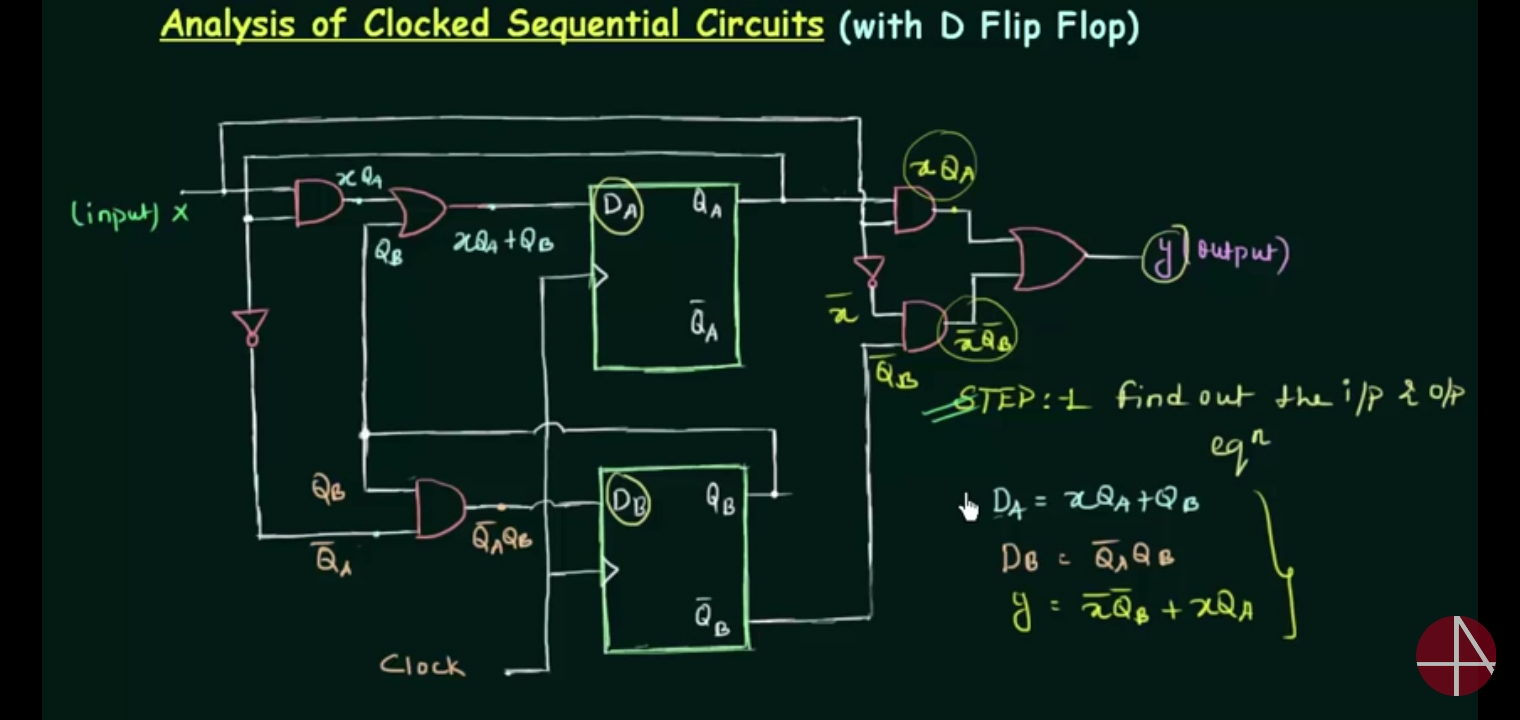
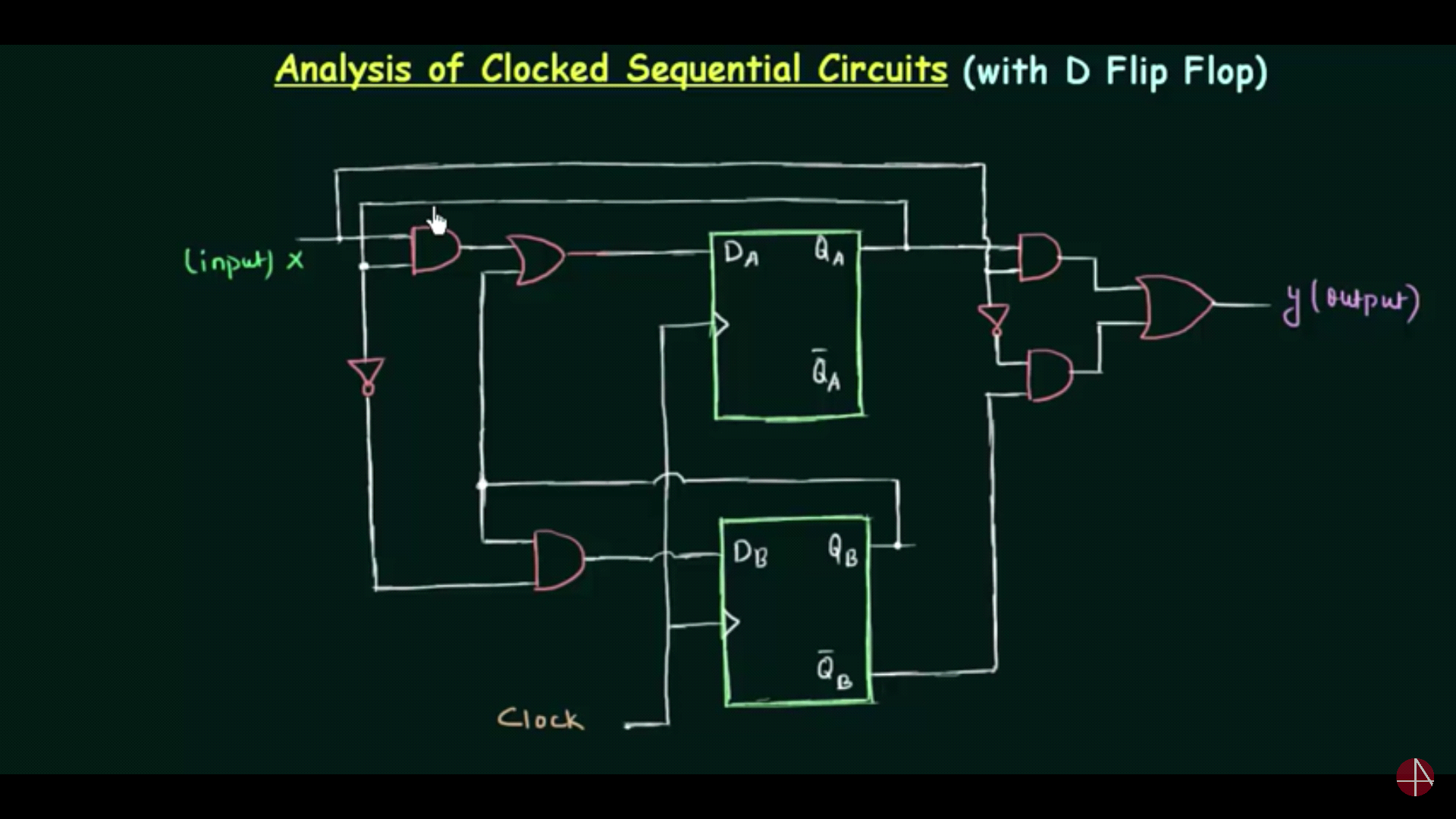
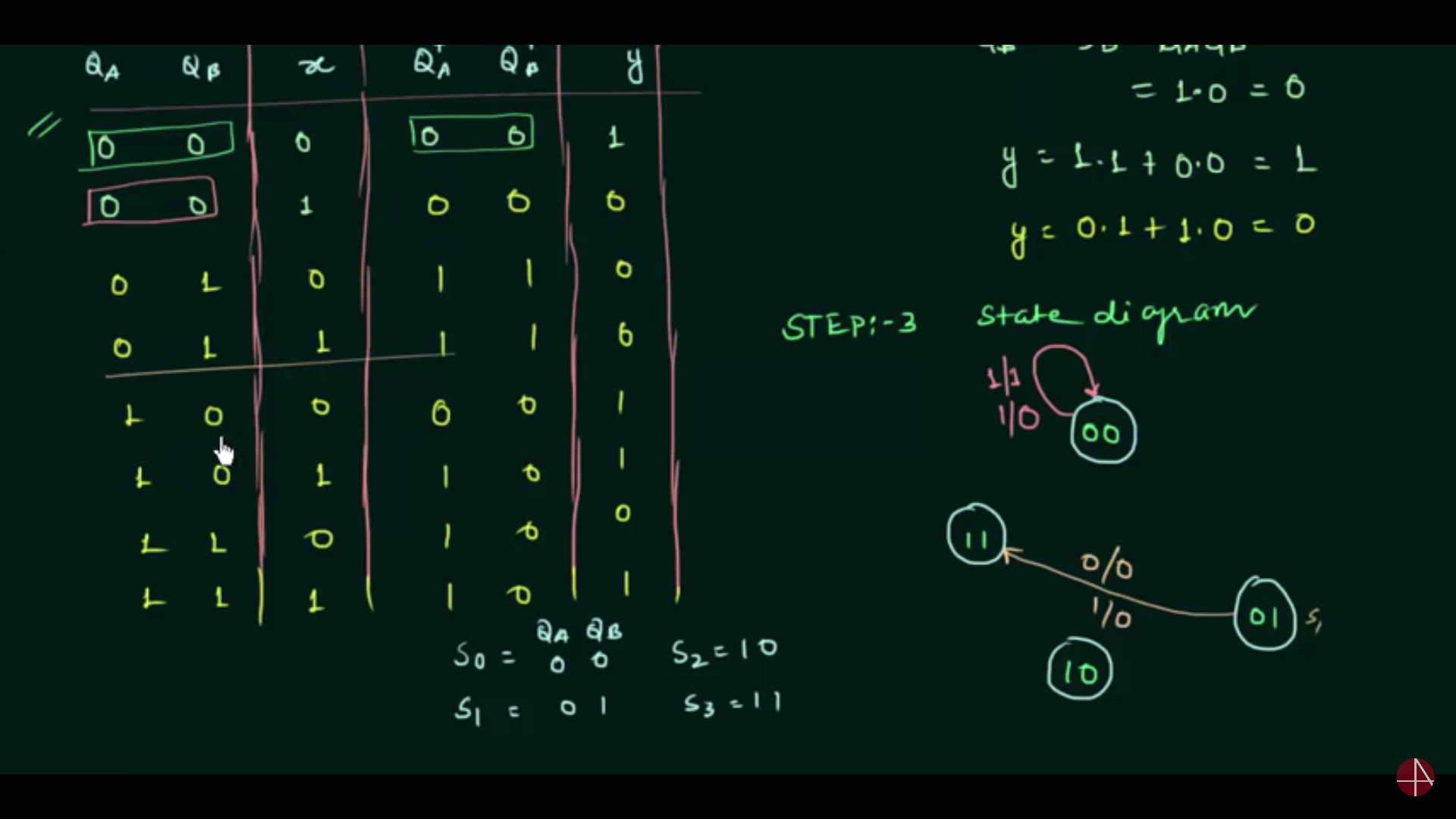
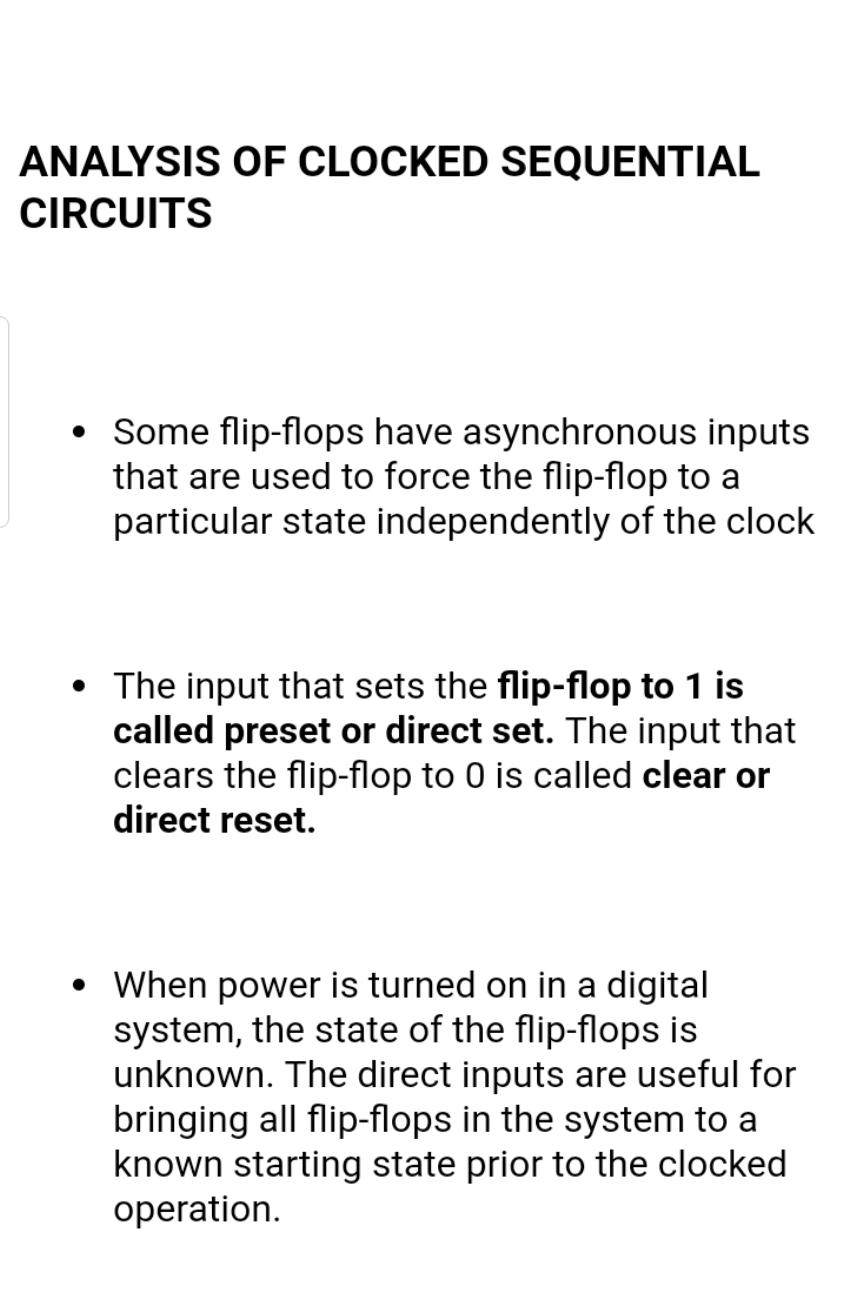
DAILY ASSESSMENT FORMAT

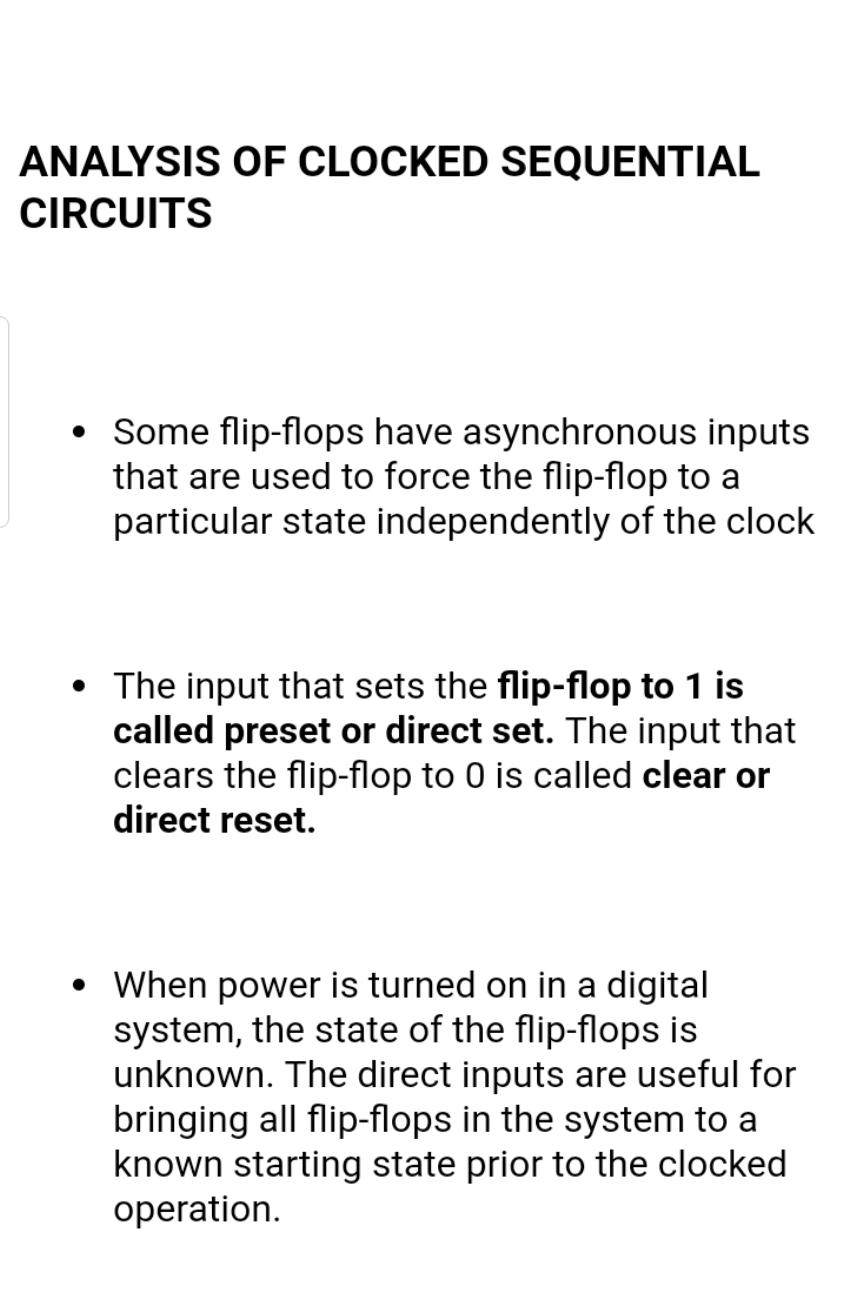
|  |  |  |  |
| --- | --- | --- | --- |
| Date | 29/05/2020 | Name: | Prajna |
| Course: | Logic design | USN: | 4AL16EC047 |
| Topic: | Analysis of clocked sequential circuits  Digital clock design | Semester &  Section: | 8 “A” |
| FORENOON SESSION DETAILS | | | |











|  |  |  |  |
| --- | --- | --- | --- |
| Date | 29/05/2020 | Name: | Prajna |
| Course: | PYTHON | USN: | 4AL16EC047 |
| Topic: | 1. Python Statistics.  2. Introduction to Machine learning.  3. Building Machine learning model. | Semester &  Section: | 8 “A” |
| AFTERNOON SESSION DETAILS | | | |

**Python Statistics Fundamentals:**

**How to Describe Data?**

* Box Plots.
* Histograms.
* Pie Charts.
* Bar Charts.
* X-Y Plots.
* Heatmaps.

## Descriptive Statistics:

**Descriptive statistics** is about describing and summarizing data. It uses two main approaches:

1. **The quantitative approach** describes and summarizes data numerically.
2. **The visual approach** illustrates data with charts, plots, histograms, and other graphs.

We can apply descriptive statistics to one or many datasets or variables. When you describe and summarize a single variable, you’re performing **univariate analysis**. When you search for statistical relationships among a pair of variables, you’re doing a **bivariate analysis**. Similarly, a **multivariate analysis** is concerned with multiple variables at once.

### Types of Measures:

In this tutorial, I learnt about the following types of measures in descriptive statistics:

* **Central tendency** tells you about the centers of the data. Useful measures include the mean, median, and mode.
* **Variability** tells you about the spread of the data. Useful measures include variance and standard deviation.
* **Correlation or joint variability** tells you about the relation between a pair of variables in a dataset. Useful measures include covariance and the correlation coefficient.

### Population and Samples:

In statistics, the **population** is a set of all elements or items that you’re interested in. Populations are often vast, which makes them inappropriate for collecting and analyzing data. That’s why statisticians usually try to make some conclusions about a population by choosing and examining a representative subset of that population.

This subset of a population is called a **sample**. Ideally, the sample should preserve the essential statistical features of the population to a satisfactory extent. That way, you’ll be able to use the sample to glean conclusions about the population.

### Outliers:

An **outlier** is a data point that differs significantly from the majority of the data taken from a sample or population. There are many possible causes of outliers, but here are a few to start you off:

* **Natural variation** in data
* **Change** in the behavior of the observed system
* **Errors** in data collection

Data collection errors are a particularly prominent cause of outliers. For example, the limitations of measurement instruments or procedures can mean that the correct data is simply not obtainable. Other errors can be caused by miscalculations, data contamination, human error, and more.

There isn’t a precise mathematical definition of outliers. You have to rely on experience, knowledge about the subject of interest, and common sense to determine if a data point is an outlier and how to handle it.

## Choosing Python Statistics Libraries:

There are many Python statistics libraries out there for you to work with, but in this tutorial, you’ll be learning about some of the most popular and widely used ones:

* **Python’s**[**statistics**](https://docs.python.org/3/library/statistics.html) is a built-in Python library for descriptive statistics. You can use it if your datasets are not too large or if you can’t rely on importing other libraries.
* [**NumPy**](https://docs.scipy.org/doc/numpy/user/index.html) is a third-party library for numerical computing, optimized for working with single- and multi-dimensional arrays. Its primary type is the array type called [ndarray](https://docs.scipy.org/doc/numpy/reference/arrays.ndarray.html). This library contains many [routines](https://docs.scipy.org/doc/numpy/reference/routines.statistics.html) for statistical analysis.
* [**SciPy**](https://www.scipy.org/getting-started.html) is a third-party library for scientific computing based on NumPy. It offers additional functionality compared to NumPy, including [scipy.stats](https://docs.scipy.org/doc/scipy/reference/stats.html) for statistical analysis.
* [**Pandas**](https://pandas.pydata.org/pandas-docs/stable/) is a third-party library for numerical computing based on NumPy. It excels in handling labeled one-dimensional (1D) data with [Series](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.html) objects and two-dimensional (2D) data with [DataFrame](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html) objects.
* [**Matplotlib**](https://matplotlib.org/) is a third-party library for data visualization. It works well in combination with NumPy, SciPy, and Pandas.

In many cases, Series and DataFrame objects can be used in place of NumPy arrays. Often, we might just pass them to a NumPy or SciPy statistical function. In addition, we can get the unlabeled data from a Series or DataFrame as a np.ndarray object by calling [.values](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.values.html) or [.to\_numpy()](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.to_numpy.html).

## Getting Started With Python Statistics Libraries

The built-in Python statistics library has a relatively small number of the most important statistics functions. The [official documentation](https://docs.python.org/3/library/statistics.html) is a valuable resource to find the details. If you’re limited to pure Python, then the Python statistics library might be the right choice.

## Calculating Descriptive Statistics

>>>

>>> import math

>>> import statistics

>>> import numpy as np

>>> import scipy.stats

>>> import pandas as pd

Let’s create some data to work with.

>>>

>>> x = [8.0, 1, 2.5, 4, 28.0]

>>> x\_with\_nan = [8.0, 1, 2.5, math.nan, 4, 28.0]

>>> x

[8.0, 1, 2.5, 4, 28.0]

>>> x\_with\_nan

[8.0, 1, 2.5, nan, 4, 28.0]

Now we have the lists x and x\_with\_nan. They’re almost the same, with the difference that x\_with\_nan contains a nan value. It’s important to understand the behavior of the Python statistics routines when they come across a [**not-a-number value (nan)**](https://en.wikipedia.org/wiki/NaN). In data science, missing values are common, and you’ll often replace them with nan.

### Measures of Central Tendency

The **measures of central tendency** show the central or middle values of datasets. There are several definitions of what’s considered to be the center of a dataset. In this tutorial, you’ll learn how to identify and calculate these measures of central tendency:

* Mean
* Weighted mean
* Geometric mean

#### Mean

The **sample mean**, also called the **sample arithmetic mean** or simply the **average**, is the arithmetic average of all the items in a dataset. The mean of a dataset 𝑥 is mathematically expressed as Σᵢ𝑥ᵢ/𝑛, where 𝑖 = 1, 2, …, 𝑛. In other words, it’s the sum of all the elements 𝑥ᵢ divided by the number of items in the dataset 𝑥.

#### Weighted Mean

The **weighted mean**, also called the **weighted arithmetic mean** or **weighted average**, is a generalization of the arithmetic mean that enables you to define the relative contribution of each data point to the result.

The weighted mean is very handy when you need the mean of a dataset containing items that occur with given relative frequencies. For example, say that you have a set in which 20% of all items are equal to 2, 50% of the items are equal to 4, and the remaining 30% of the items are equal to 8. You can calculate the mean of such a set like this:

>>>

>>> 0.2 \* 2 + 0.5 \* 4 + 0.3 \* 8

#### Harmonic Mean

The **harmonic mean** is the reciprocal of the mean of the reciprocals of all items in the dataset: 𝑛 / Σᵢ(1/𝑥ᵢ), where 𝑖 = 1, 2, …, 𝑛 and 𝑛 is the number of items in the dataset 𝑥. One variant of the pure Python implementation of the harmonic mean is this:

>>>

>>> hmean = len(x) / sum(1 / item for item in x)

>>> hmean

2.7613412228796843

>>>

#### Geometric Mean

The **geometric mean** is the 𝑛-th root of the product of all 𝑛 elements 𝑥ᵢ in a dataset 𝑥: ⁿ√(Πᵢ𝑥ᵢ), where 𝑖 = 1, 2, …, 𝑛. The following figure illustrates the arithmetic, harmonic, and geometric means of a dataset:

Introduction to Machine learning:

Machine learning has become an integral part of many commercial applications and research projects, but this field is not exclusive to large companies with extensive research teams. If you use Python, even as a beginner, this book will teach you practical ways to build your own machine learning solutions.

Building Machine learning model:

* Define adequately our problem (objective, desired outputs…).
* Gather data.
* Choose a measure of success.
* Set an evaluation protocol and the different protocols available.
* Prepare the data (dealing with missing values, with categorial values…).
* Spilt correctly the data.
* Differentiate between over and under fitting, defining what they are and explaining the best ways to avoid them.
* An overview of how a model learns.
* What is regularization and when is appropiate to use it.
* Develop a benchmark model.
* Choose an adequate model and tune it to get the best performance possible.

## 