1. What does the *number* in parentheses in np.random.seed(*number*) represents? Explain.

Algorithms that generate numbers that "look randomize" are called pseudo-random number generators (PRNGs). To get our iterative PRNG started, we need an *initial input*. This initial input is called a "seed". Since the PRNG is deterministic, for a given seed, it will generate an identical sequence of numbers. Usually, there is a default seed that is, itself, sort of randomized. The most common one is the current time. However, the current time isn't a very good random number, so this behavior is known to cause problems sometimes. If you want your program to run identically each time you run it, you can *provide* a seed (0 is a popular option, but is entirely arbitrary). Then, you get a sequence of randomized numbers, but if you give your code to someone they can entirely recreate the runtime of the program as you witnessed it when you ran it.

Reference,

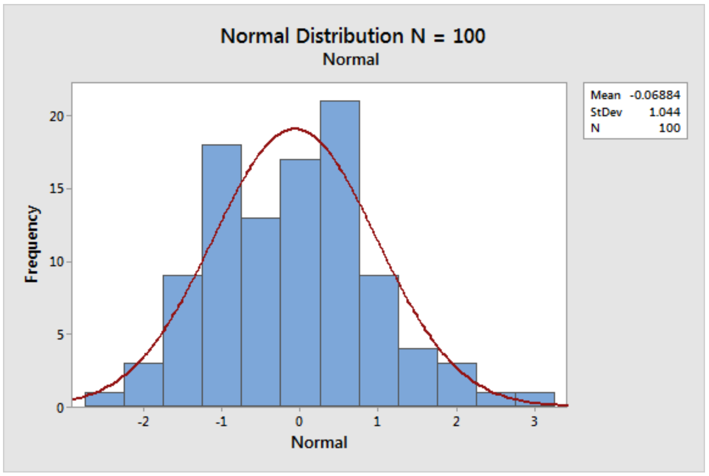
https://stackoverflow.com/questions/55477551/what-does-the-number-in-parentheses-in-np-random-seednumber-means

2. Which graphs can be used to check normality? Explain with examples.

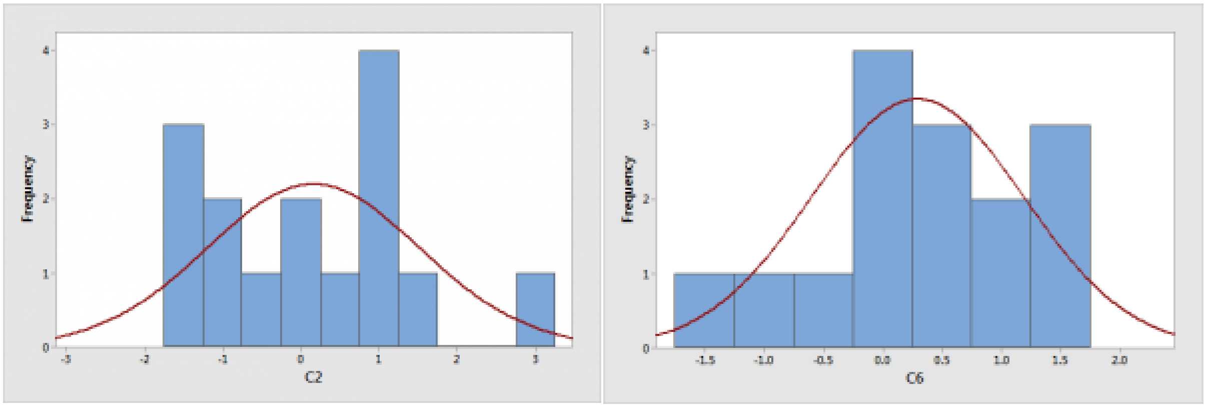
Three different graphs can use to determine a data has normal distribution or not,

1- Histograms
2- Normal probability plots
3- Boxplot

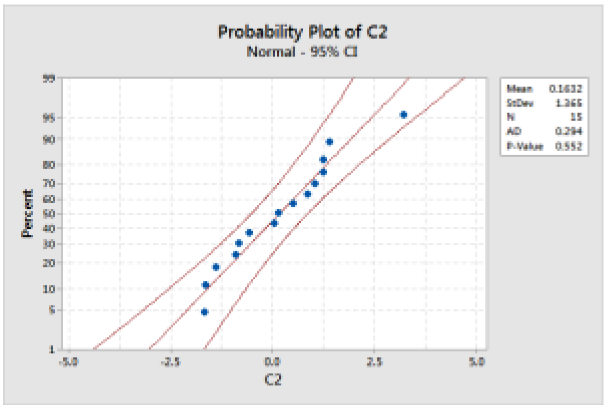
You can see the sample for histogram below and it is simply determining it has a normal distribution.

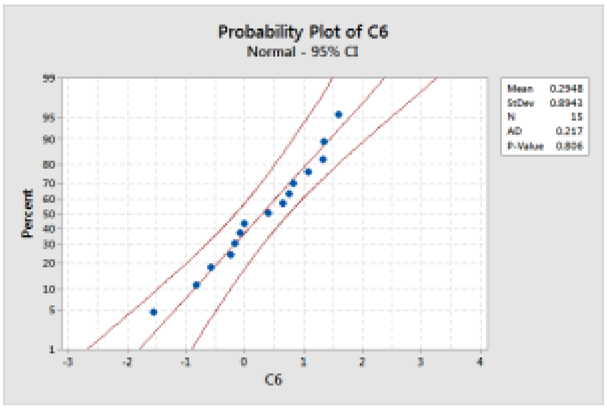


But the histogram plot is not useful for low observation data and you can't understand the correct distribution of data when you are using it, for example, the below plots are created for normally distributed data but it is impossible to understand.



The best plot to determine the normal distribution is Normal probability plots also known as quantile-quantile plots, or Q-Q Plots for short. If the points track the straight line, your data follow the normal distribution. It's very straightforward. To understand different this plot you can see below graph for above histogram,





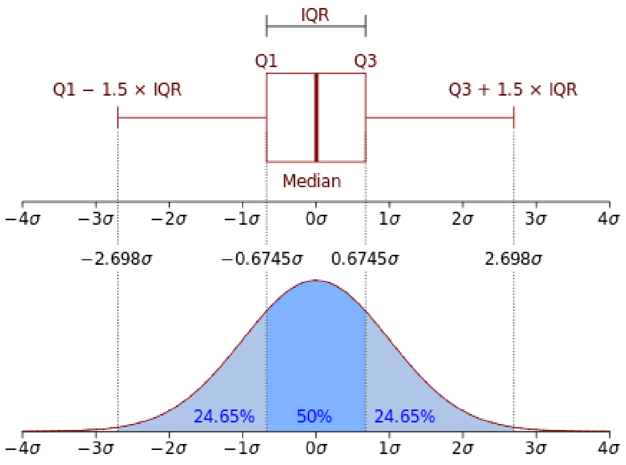
As you can see the distribution of observation stay around a straight line which shows they have a normal distribution.

The last graph can discuss here is a boxplot that one of the most famous graphs to work with data. It uses for finding different aspects of one data such as outliers, quartiles, median, mean, IQR, etc.

For checking the distribution of data, we use the standard deviation and sample mean. Input these values into the formulas for Q1 and Q3 below.

- $Q1 = \bar{x} - (s * 0.67)$
- $Q3 = \bar{x} + (s * 0.67)$

Compare these calculated values to your data's actual Q1 and Q3 values. If they are notably different, your data might not follow the normal distribution.



Reference,

https://statisticsbyjim.com/basics/assessing-normality-histograms-probability-plots/
https://statisticsbyjim.com/?s=box+plot+normality
https://www.statisticshowto.com/assumption-of-normality-test/

3. Which statistical tests can be used to check normality? Explain with examples.

There is different statistics test exist for answering the big concern that a distribution is a normal distribution or not, here we explain some of them and say the positive and negative points of each one,

- Chi-square Test for Normality

It is the simplest test you can do in your data to understand your data has a normal distribution or not, but it has some limitations,

1- It works well for a small sample of data (less or equal to 20)
2- It is designed for the districts' data
3- Your data is randomly sampled

It is worked with the calculation of each observation's expected value and using the below formula for chi-square

$$X^2 = \sum \frac{(observed - expected)^2}{(expected)}$$

- Kolmogorov-Smirnov Goodness of Fit Test (K-S test)

This compares your data with a known distribution. Although the test is nonparametric — it doesn't assume any particular underlying distribution — it is commonly used as a test for normality to see if your data is normally distributed. It's also used to check the assumption of normality in the Analysis of Variance.

- Lilliefors Test for Normality

It is an improvement on the Kolmogorov-Smirnov (K-S) test — correcting the K-S for small values at the tails of probability distributions — and is therefore sometimes called the K-S D test. Unlike the K-S test, Lilliefors can be used when you don't know the population mean or standard deviation. Essentially, the Lilliefors test is a K-S test that allows you to estimate these parameters from your sample.

The important point about this type of test is, that it has low power.

- Shapiro-Wilk Test

It is known as the best test for understanding data distribution, and it for normality is one of three general normality tests designed to detect all departures from normality. It is comparable in power to the other two tests.

The test has limitations, most importantly that the test has a bias by sample size. The larger the sample, the more likely you'll get a statistically significant result.

$$W = \frac{\left(\sum_{i=1}^{n} a_i x_{(i)}\right)^2}{\sum_{i=1}^{n}(x_i - \overline{x})^2}$$

- xi are the ordered random sample values
- ai are constants generated from the covariances, variances, and means of the sample (size n) from a normally distributed sample.

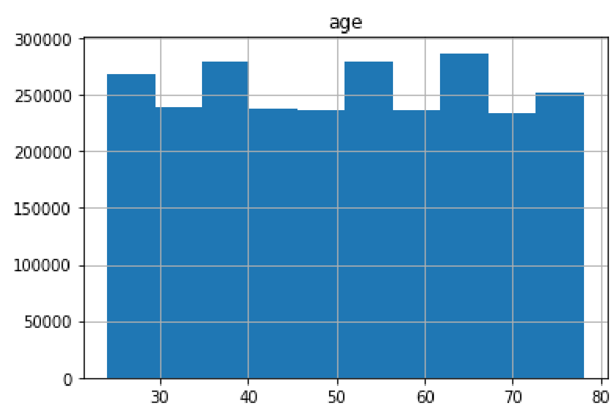
Reference,

https://www.statisticshowto.com/chi-square-test-normality/
https://www.statisticshowto.com/kolmogorov-smirnov-test/
https://www.statisticshowto.com/shapiro-wilk-test/

4. In **Python**, write code blocks to check the normality (one using a graph and one using a statistical test)

For the graph test, I have used my data population that you can see the population data below hist graph,

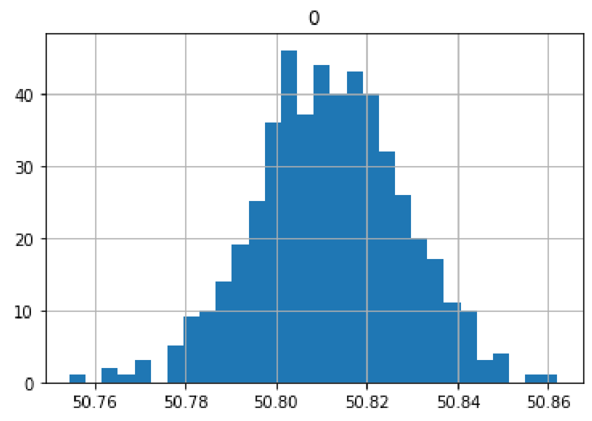
```
import pandas as pd
import numpy as np
import math
import scipy.stats as stats
import psycopg2
import statsmodels.api as sm

postgresConnection = psycopg2.connect(host='192.168.0.246', port='5432',
database="postgres", user="postgres", password="1qazXSW@")
cursor = postgresConnection.cursor()
data = pd.read_sql('select age from
Course2_ToolTechniquesforDataScience',postgresConnection)
cursor.close()

data.hist()
population_data = np.array(data)[:,0]
```

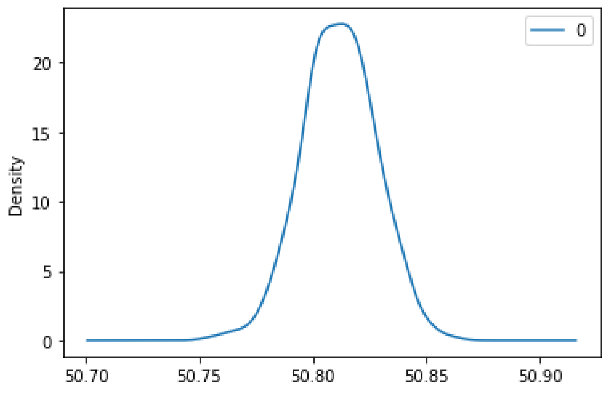


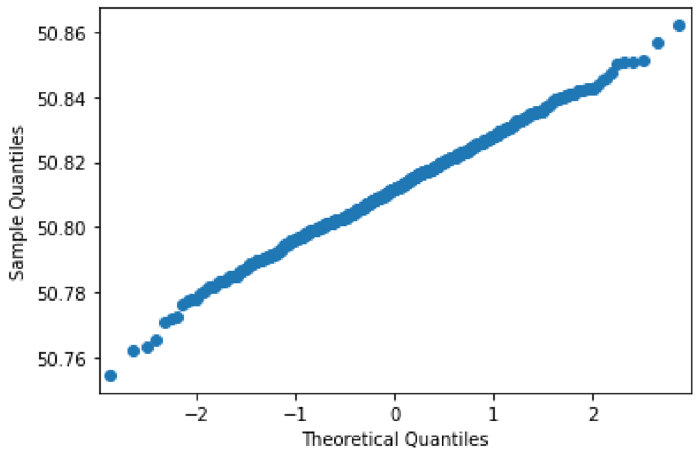
As you can see the hist plot shows that, it has near uniform distribution,

```
def mean_sampling(dataset, sample_number, popsamp):
    np.random.seed(1987)
    Mean_sample, propotion_sample = [], []
    for i in range(sample_number):
        sample_data = np.random.choice(a=dataset, size= popsamp)
        Mean_sample.append(sample_data.mean())
    return Mean_sample
```

```python
mean_sample = mean_sampling(population_data,500,1000000)
df_mean_sample = pd.DataFrame(mean_sample)
arr_sample_mean = np.array(mean_sample)

df_mean_sample.hist(bins = 30)
df_mean_sample.plot(kind='density')
sm.qqplot(arr_sample_mean)
```







As the above graph shows, the mean sampling distribution has a normal distribution

For testing normal distribution, I create a dummy information and check that,

```python
np.random.seed(1366)
first_distribution = stats.poisson.rvs(loc=21, mu=25, size= 1500)
second_distribution = np.random.normal(loc=21, scale=5, size = 1500)

# check different function of normal random distribution
alpha = 0.05
# Shapiro-Wilk Test
```

```
_ , Shapiro_first_p = stats.shapiro(first_distribution)
_ , Shapiro_second_p = stats.shapiro(second_distribution)
print('''Shapiro test p_value of first distribution: {0}
p_value of second distribution: {1}'''.format(Shapiro_first_p, Shapiro_second_p))
```

```
p_value of first distribution: 2.0422969555511372e-06
p_value of second distribution: 0.48060593008995056
```

```
# D'Agostino and Pearson's Test
_ , Agostino_first_p = stats.normaltest(first_distribution)
_ , Agostino_second_p = stats.normaltest(second_distribution)
print('''Agostino test p_value of first distribution: {0}
p_value of second distribution: {1}'''.format(Agostino_first_p,
Agostino_second_p))
```

```
p_value of first distribution: 0.0013702730297818495
p_value of second distribution: 0.18132790611097624
```

Reference,

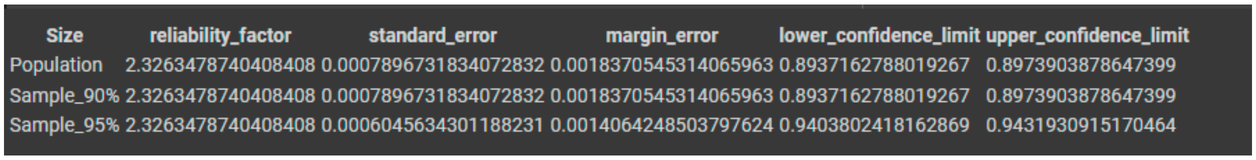
https://machinelearningmastery.com/a-gentle-introduction-to-normality-tests-in-python/

5. Construct a **Python document** to create a **99%** confidence interval for the population proportion *p* of successes (of your interest) with the following steps:
   a) obtain the reliability factor,
   b) obtain the standard error,
   c) obtain the margin of error,
   d) obtain the lower and upper confidence limits of the interval, and
   e) interpret your findings.

I used a normal distribution,

```
import scipy.stats as stats
import math
import numpy as np
import pandas as pd
from prettytable import PrettyTable

def Pro_estimation(df, CI, great_prcentage):
    reliability_factor = stats.norm.ppf(q= CI)
    sample_size = df.count()
    P_success = df[df < np.quantile(df,great_prcentage)].count() / sample_
size
    standard_error = math.sqrt(P_success*(1-P_success)/sample_size)
    margin_error = reliability_factor * standard_error
    lower_confidence_limit = P_success - margin_error
    upper_confidence_limit = P_success + margin_error
    return reliability_factor, standard_error, margin_error, lower_confide
nce_limit, upper_confidence_limit

np.random.seed(10)
population_ages1 = stats.poisson.rvs(loc=18, mu=35, size=150000)
df1=(pd.DataFrame(population_ages1))
reliability_factor_p, standard_error_p, margin_error_p, lower_confidence_l
imit_p, upper_confidence_limit_p = Pro_estimation(df1, 0.99, 0.9)

np.random.seed(20)
minnesota_ages1 = stats.poisson.rvs(loc=18, mu=35, size=30)
df2 = (pd.DataFrame(minnesota_ages1))
reliability_factor_s, standard_error_s, margin_error_s, lower_confidence_l
imit_s, upper_confidence_limit_s = Pro_estimation(df1, 0.99, 0.9)

minnesota_ages2 = stats.poisson.rvs(loc=18, mu=35, size=30)
df2 = (pd.DataFrame(minnesota_ages1))
reliability_factor_s2, standard_error_s2, margin_error_s2, lower_confidenc
e_limit_s2, upper_confidence_limit_s2 = Pro_estimation(df1, 0.99, 0.95)
```

```
population_sample = PrettyTable(['Size','reliability_factor', 'standard_error', 'margin_error', 'lower_confidence_limit','upper_confidence_limit'])
population_sample.add_row(['Population',reliability_factor_p, standard_error_p, margin_error_p, lower_confidence_limit_p[0], upper_confidence_limit_p[0]])
population_sample.add_row(['Sample_90%',reliability_factor_s, standard_error_s, margin_error_s, lower_confidence_limit_s[0], upper_confidence_limit_s[0]])
population_sample.add_row(['Sample_95%',reliability_factor_s2, standard_error_s2, margin_error_s2, lower_confidence_limit_s2[0], upper_confidence_limit_s2[0]])

population_sample
```



Size	reliability_factor	standard_error	margin_error	lower_confidence_limit	upper_confidence_limit
Population	2.3263478740408408	0.0007896731834072832	0.0018370545314065963	0.8937162788019267	0.8973903878647399
Sample_90%	2.3263478740408408	0.0007896731834072832	0.0018370545314065963	0.8937162788019267	0.8973903878647399
Sample_95%	2.3263478740408408	0.0006045634301188231	0.0014064248503797624	0.9403802418162869	0.9431930915170464

As you can see,

1- The changing the sample size doesn't affect the output value
2- Changing the used seed can't change it
3- With changing in the requested success percentage, we can see,

Reliability_factor not change
standard_error has opposite relation with the success percentage
margin_error has opposite relation with the success percentage
so the range of confidence interval has opposite relation