Import libraries

```
import numpy as np
import scipy as sp
from scipy import stats
import pandas as pd
from random import shuffle
```

Import time series data

```
import pandas as pd
df = pd.read_csv(r"/content/drive/MyDrive/covid_data_kerala.csv")
df
```

	Date	Confirmed	Recovered	Deceased
0	2020-01-31	0.0	NaN	0.0
1	2020-02-01	0.0	NaN	0.0
2	2020-02-02	1.0	NaN	0.0
3	2020-02-03	1.0	NaN	0.0
4	2020-02-04	0.0	NaN	0.0
...	...	...	...	...
836	2022-05-16	324.0	364.0	31.0
837	2022-05-17	596.0	366.0	6.0
838	2022-05-18	501.0	460.0	17.0
839	2022-05-19	556.0	362.0	23.0
840	2022-05-20	558.0	446.0	63.0

841 rows × 4 columns

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Create numpy array

```
Confirmed = df.Confirmed
values = np.array(Confirmed)
sample = shuffle(Confirmed)
```

```
/usr/lib/python3.8/random.py:307: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-c
  x[i], x[j] = x[j], x[i]
```

```
sample = np.array(Confirmed)
sample = sample[0:250]
```

```
sample,values
```

```
            7.8230e+03, 1.1079e+04, 9.2460e+03, 8.8670e+03, 7.9550e+03,
            7.5550e+03, 6.6760e+03, 7.6430e+03, 1.1150e+04, 8.7330e+03,
            9.3610e+03, 8.9090e+03, 8.5380e+03, 6.6640e+03, 7.1630e+03,
            9.4450e+03, 7.7380e+03, 7.7220e+03, 7.4270e+03, 7.1670e+03,
            5.2970e+03, 6.4440e+03, 7.3120e+03, 7.5450e+03, 6.5800e+03,
            6.5460e+03, 7.1240e+03, 5.4040e+03, 6.4090e+03, 7.5400e+03,
            7.2240e+03, 6.6740e+03, 6.4680e+03, 5.8480e+03, 4.5470e+03,
            5.5160e+03, 6.8490e+03, 6.1110e+03, 5.7540e+03, 6.0750e+03,
            5.0800e+03, 3.6980e+03, 4.9720e+03, 4.2800e+03, 5.9870e+03,
            4.6770e+03, 4.7410e+03, 4.3500e+03, 3.3820e+03, 4.7230e+03,
            5.4050e+03, 4.7000e+03, 4.9950e+03, 4.5570e+03, 4.4500e+03,
            3.2770e+03, 4.6560e+03, 5.0380e+03, 4.1690e+03, 3.9720e+03,
            3.7950e+03, 3.7770e+03, 2.4340e+03, 3.3770e+03, 4.0060e+03,
            3.4040e+03, 3.4710e+03, 3.2970e+03, 2.9950e+03, 2.2300e+03,
            2.7480e+03, 3.2050e+03, 2.5140e+03, 2.6050e+03, 2.4070e+03,
            1.8240e+03, 1.6360e+03, 2.4740e+03, 2.8460e+03, 2.4230e+03,
            2.6760e+03, 2.4350e+03, 2.8020e+03, 2.5600e+03, 3.6400e+03,
            4.8010e+03, 4.6490e+03, 5.2960e+03, 5.9440e+03, 6.2380e+03,
            5.7970e+03, 9.0660e+03, 1.2742e+04, 1.3468e+04, 1.6338e+04,
            1.7755e+04, 1.8123e+04, 2.2946e+04, 2.8481e+04, 3.4199e+04,
            4.6387e+04, 4.1668e+04, 4.5136e+04, 4.5449e+04, 2.6514e+04,
            5.5475e+04, 4.9771e+04, 5.1739e+04, 5.4537e+04, 5.0812e+04,
            5.1570e+04, 4.2154e+04, 5.1887e+04, 5.2199e+04, 4.2677e+04,
            3.8684e+04, 3.3538e+04, 2.6729e+04, 2.2524e+04, 2.9471e+04,
            2.3253e+04, 1.8420e+04, 1.6012e+04, 1.5184e+04, 1.1136e+04,
            8.9890e+03, 1.1776e+04, 1.2223e+04, 8.6550e+03, 7.7800e+03,
            6.7570e+03, 5.4270e+03, 4.0690e+03, 5.6910e+03, 5.0230e+03,
            4.0640e+03, 3.5810e+03, 3.2620e+03, 2.5240e+03, 2.0100e+03,
            2.8460e+03, 2.3730e+03, 2.2220e+03, 2.1900e+03, 1.8360e+03,
            1.4080e+03, 1.2230e+03, 1.7910e+03, 1.4210e+03, 1.4260e+03,
            1.1750e+03, 1.0880e+03, 8.8500e+02, 8.0900e+02, 1.1930e+03,
            9.6600e+02, 9.2200e+02, 8.4700e+02, 7.1900e+02, 5.9600e+02,
            4.9500e+02, 7.0200e+02, 7.0200e+02, 5.5800e+02, 5.4300e+02,
            4.9600e+02, 4.0000e+02, 3.4600e+02, 4.2400e+02, 4.3800e+02,
            4.2900e+02, 4.1800e+02, 3.3100e+02, 3.1000e+02, 2.5600e+02,
            3.5400e+02, 3.6100e+02, 2.9100e+02, 3.5300e+02, 3.4700e+02,
            2.2300e+02, 1.9600e+02, 2.9800e+02, 3.0100e+02, 2.3300e+02,
            1.9200e+02, 2.1400e+02, 2.7900e+02, 2.0900e+02, 3.5500e+02,
            3.3200e+02, 3.1500e+02, 3.0000e+02, 2.8100e+02, 2.9000e+02,
            2.5500e+02, 3.4100e+02, 3.4700e+02, 4.1200e+02, 3.3700e+02,
            3.1400e+02, 2.5000e+02, 2.9600e+02, 3.8600e+02, 3.4200e+02,
            4.0000e+02, 4.6100e+02, 3.8100e+02, 3.2400e+02, 3.4600e+02,
            4.8900e+02, 4.1300e+02, 4.1900e+02, 5.2300e+02, 4.2800e+02,
            3.2100e+02, 3.2400e+02, 5.9600e+02, 5.0100e+02, 5.5600e+02,
            5.5800e+02]))

# Create random samples

norm_a = values
norm_b = sample


norm_a
```

```
3.8684e+04, 3.3538e+04, 2.6729e+04, 2.2524e+04, 2.9471e+04,
2.3253e+04, 1.8420e+04, 1.6012e+04, 1.5184e+04, 1.1136e+04,
8.9890e+03, 1.1776e+04, 1.2223e+04, 8.6550e+03, 7.7800e+03,
6.7570e+03, 5.4270e+03, 4.0690e+03, 5.6910e+03, 5.0230e+03,
4.0640e+03, 3.5810e+03, 3.2620e+03, 2.5240e+03, 2.0100e+03,
2.8460e+03, 2.3730e+03, 2.2220e+03, 2.1900e+03, 1.8360e+03,
1.4080e+03, 1.2230e+03, 1.7910e+03, 1.4210e+03, 1.4260e+03,
1.1750e+03, 1.0880e+03, 8.8500e+02, 8.0900e+02, 1.1930e+03,
9.6600e+02, 9.2200e+02, 8.4700e+02, 7.1900e+02, 5.9600e+02,
4.9500e+02, 7.0200e+02, 7.0200e+02, 5.5800e+02, 5.4300e+02,
4.9600e+02, 4.0000e+02, 3.4600e+02, 4.2400e+02, 4.3800e+02,
4.2900e+02, 4.1800e+02, 3.3100e+02, 3.1000e+02, 2.5600e+02,
3.5400e+02, 3.6100e+02, 2.9100e+02, 3.5300e+02, 3.4700e+02,
2.2300e+02, 1.9600e+02, 2.9800e+02, 3.0100e+02, 2.3300e+02,
1.9200e+02, 2.1400e+02, 2.7900e+02, 2.0900e+02, 3.5500e+02,
3.3200e+02, 3.1500e+02, 3.0000e+02, 2.8100e+02, 2.9000e+02,
2.5500e+02, 3.4100e+02, 3.4700e+02, 4.1200e+02, 3.3700e+02,
3.1400e+02, 2.5000e+02, 2.9600e+02, 3.8600e+02, 3.4200e+02,
4.0000e+02, 4.6100e+02, 3.8100e+02, 3.2400e+02, 3.4600e+02,
4.8900e+02, 4.1300e+02, 4.1900e+02, 5.2300e+02, 4.2800e+02,
3.2100e+02, 3.2400e+02, 5.9600e+02, 5.0100e+02, 5.5600e+02,
5.5800e+02])
```

Create Python Script for K-S Test

```
def ks_2samp(sample1, sample2):
    # Gets all observations
    observations = np.concatenate((sample1, sample2))
    observations.sort()
    # Sorts the samples
    sample1.sort()
    sample2.sort()
    # Evaluates the KS statistic
    D_ks = [] # KS Statistic list
    for x in observations:
        cdf_sample1 = cdf(sample = sample1, x  = x)
        cdf_sample2 = cdf(sample = sample2, x  = x)
        D_ks.append(abs(cdf_sample1 - cdf_sample2))
    ks_stat = max(D_ks)
    # Calculates the P-Value based on the two-sided test
    # The P-Value comes from the KS Distribution Survival Function (SF = 1-CDF)
    m, n = float(len(sample1)), float(len(sample2))
    en = m * n / (m + n)
    p_value = stats.kstwo.sf(ks_stat, np.round(en))
    return {"ks_stat": ks_stat, "p_value" : p_value}
```

```
def cdf(sample, x, sort = False):
    # Sorts the sample, if unsorted
    if sort:
        sample.sort()
    # Counts how many observations are below x
    cdf = sum(sample <= x)
    # Divides by the total number of observations
    cdf = cdf / len(sample)
    return cdf


sets = [norm_a, norm_b, ]
names = ['norm_a', 'norm_b']
ks_scores = {}
for _ in range(len(names)):
    name1 = names.pop(0)
    sample1 = sets.pop(0)
    for name2, sample2 in zip(names, sets):
        key1 = name1 + "_" +  name2
        key2 = name2 + "_" +  name1
        ks = ks_2samp(sample1, sample2)
        ks_scores[key1] = ks
        ks_scores[key2] = ks
# Prints the results
print(f"norm_a vs norm_b: ks = {ks_scores['norm_a_norm_b']['ks_stat']:.4f} (p-value = {ks_scores['norm_a_norm_b']['p_value']:.3e}, are equal
```

```
norm_a vs norm_b: ks = 0.0217 (p-value = 1.000e+00, are equal = True)
```

✓ 2s completed at 6:01 AM