```
# IMPORTANT: RUN THIS CELL IN ORDER TO IMPORT YOUR KAGGLE DATA SOURCES,
# THEN FEEL FREE TO DELETE THIS CELL.
# NOTE: THIS NOTEBOOK ENVIRONMENT DIFFERS FROM KAGGLE'S PYTHON
# ENVIRONMENT SO THERE MAY BE MISSING LIBRARIES USED BY YOUR
# NOTEBOOK.
import kagglehub
rupakroy_credit_data_path = kagglehub.dataset_download('rupakroy/credit-data')

print('Data source import complete.')
```

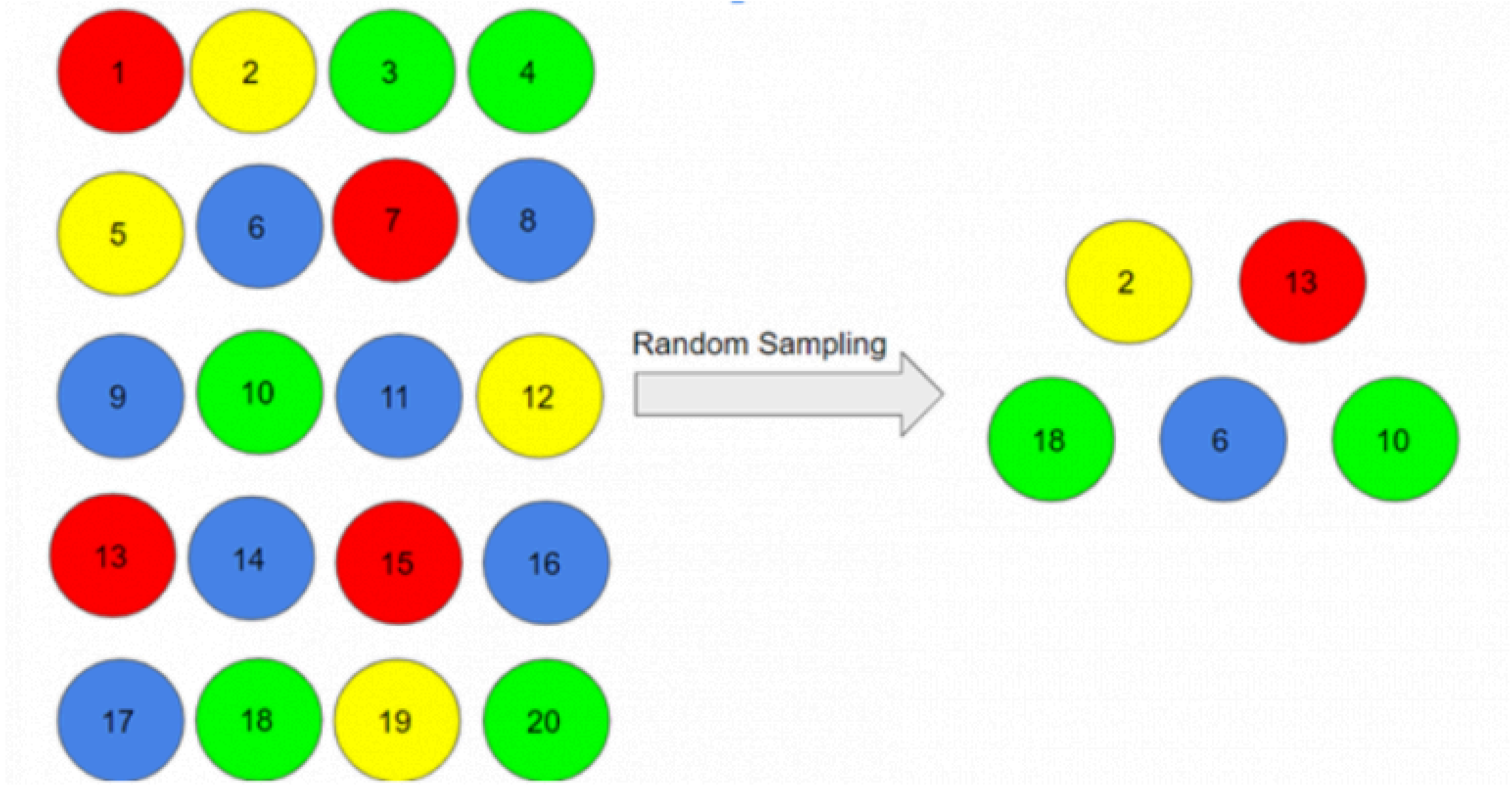
simple random sampling

what is simple random sampling?

simple random sampling is the most elementary and frequently used process to extract elements (data) from a given set (dataset).

how this works?

using this technique we can extract a sample of elements at random from a population of elements, that is, each item is chosen completely at random and each item in the population has the same probability of being included in the sample.



it is really very simple.

how to use it?

a set of random numbers is generated and the units that present these numbers will be included in the sample. for example, let's say you have a population of 1000 people and you would like to choose a simple random sample of 50 people.

there is a library in python called pandas which has a ready-made function for extracting a simple random sample.

in this notebook, I will show you how to use this function and create another function to facilitate its use in any tabulated dataset.

let's code

```
# libraries
```

```
import numpy as np
import pandas as pd
import random as rd
import os


# example dataset

for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

print('\n\n')
df = pd.read_csv('/kaggle/input/credit-data/credit_data.csv')

display(df.head(3))
print('\n\n')

display(df.info())
print('\n\n')

display(df.shape)
print('\n\n')
```

first impressions

in our first analysis we can see that we have a dataset with 2000 rows and, although we have some null data, we are able to extract some simple random samples.

```
# exmplo using "sample" function from pandas librarie

df_sample_random = df.sample(n = 100, replace = False, random_state = None)
```

```
display(df_sample_random.head(3))
display(df_sample_random.shape)
```

understanding

what can we perceive from the function?

1. it is as simple as the concept of random sampling itself.
2. the sample size is defined by the parameter n.
3. it is very important to pay attention to the replace parameter, because if it is equal to true, some elements of the set can be extracted and inserted in the sample more than once.
4. another parameter that needs attention is the random_state parameter, it defines whether the elements should change every time the function is executed. that is, we will get a different sample of elements with each run. to lock this selection in a single sample, just enter a number greater than or equal to zero in this parameter.

more information can be found in the pandas library documentation.

new useful function

to simplify the use of this function we can create another function to be used in the context of analyzing randomly generated samples.

in this new function we will include the parameters that must be informed to guarantee the random context.

```
# function for extracting random samples with useful parameters

def sample_random(df, number_samples, state = None):
    return df.sample(n = number_samples, replace = False, random_state = state)
```

```
# using the function

df_sample_random = sample_random(df, 100)
display(df_sample_random.head(3))
display(df_sample_random.shape)
```

analyzing the result of the function

we can see that, at each run, we get a different sample of 100 elements.

if we want to lock the selection of the element for reasons of specific analysis, we can change the state parameter informing the number 1.

note the example below where, on repetition, the sample elements do not change:

```
df_sample_random_1 = sample_random(df, 100, 1)
display(df_sample_random_1.head(3))
display(df_sample_random_1.shape)

print('\n\n')

df_sample_random_2 = sample_random(df, 100, 1)
display(df_sample_random_2.head(3))
display(df_sample_random_2.shape)
```

there are other ways we can generate and work with population samples.

i created other notebooks explaining the most used sampling techniques for random selection and, consequently, guaranteeing representativeness. are they:

- simple-random-sampling
- stratified-sampling
- systematic-sampling

- cluster-sampling
- reservoir-sampling

in each of them i try to explain the concepts and create functions to optimize the process of defining and selecting a sample in any dataset.

- topic about all issues

any questions, criticisms and suggestions i will be at your disposal!

thank you