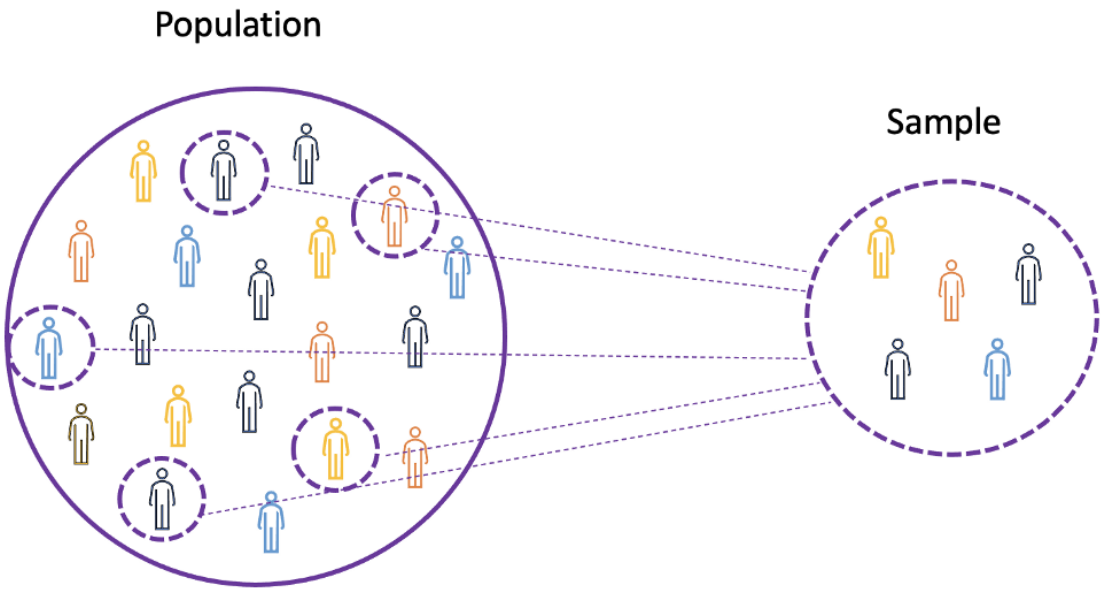
DATA SAMPLING METHODS IN PYTHON

Data Sampling forms the essential part of the majority of research, scientific and data experiments. It is one of the most important factors which determines the accuracy of your research or survey result. If your sample has not been accurately sampled then this might impact significantly the final results and conclusions. There are many sampling techniques that can be used to gather a data sample depending upon the need and situation. In this blog post, I will cover the following data sampling techniques:

```
- Terminology: Population and Sampling
- Random Sampling
- Systematic Sampling
- Cluster Sampling
- Weighted Sampling
- Stratified Sampling
```

INTRODUCTION TO POPULATION AND SAMPLE :

To start with, let's have a look at some basic terminology. It is important to learn the concepts of ***population*** and ***sample.*** The *population* is the set of all observations (individuals, objects, events, or procedures) and is usually very large and diverse, whereas a *sample* is a subset of observations from the population that ideally is a true representation of the population.

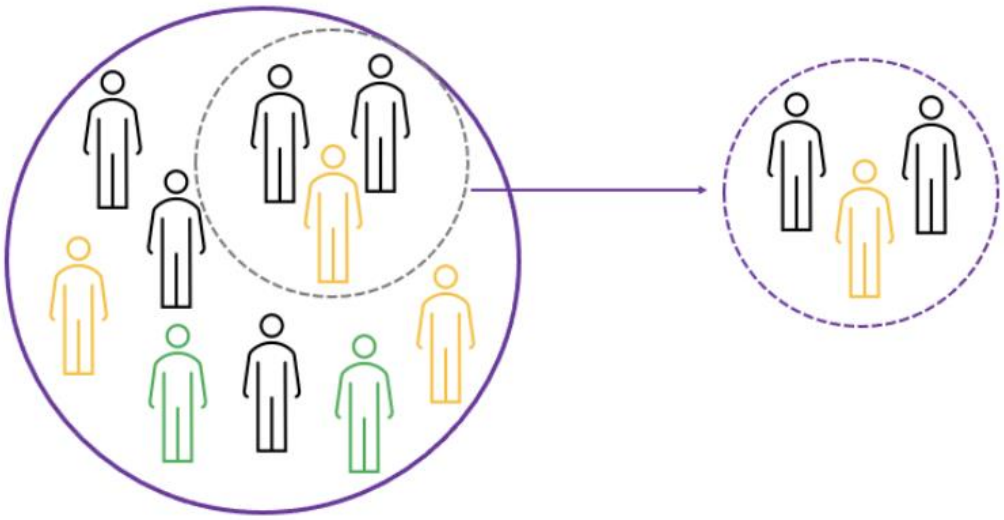


Given that experimenting with an entire population is either impossible or simply too expensive, researchers or analysts use samples rather than the entire population in their experiments or trials. To make sure that the experimental results are reliable and hold for the entire population, the sample needs to be a true representation of the population. That is, the sample needs to be unbiased.

RANDOM SAMPLING

The simplest data sampling technique that creates a random sample from the original population is Random Sampling. In this approach, every sampled observation has the same probability of getting selected during the sample generation process. Random Sampling is usually used when we don't have any kind of prior information about the target population.

For example, random selection of 3 individuals from a population of 10 individuals. Here, each individual has an equal chance of getting selected to the sample with a probability of selection of 1/10.



RANDOM SAMPLING: PYTHON IMPLEMENTATION:

First, we generate random data that will serve as population data. We will, therefore, randomly sample 10K data points from Normal distribution with mean mu = 10 and standard deviation std = 2. After this, we create a Python function called **random_sampling**() that takes population data and desired sample size and produces as output a random sample.

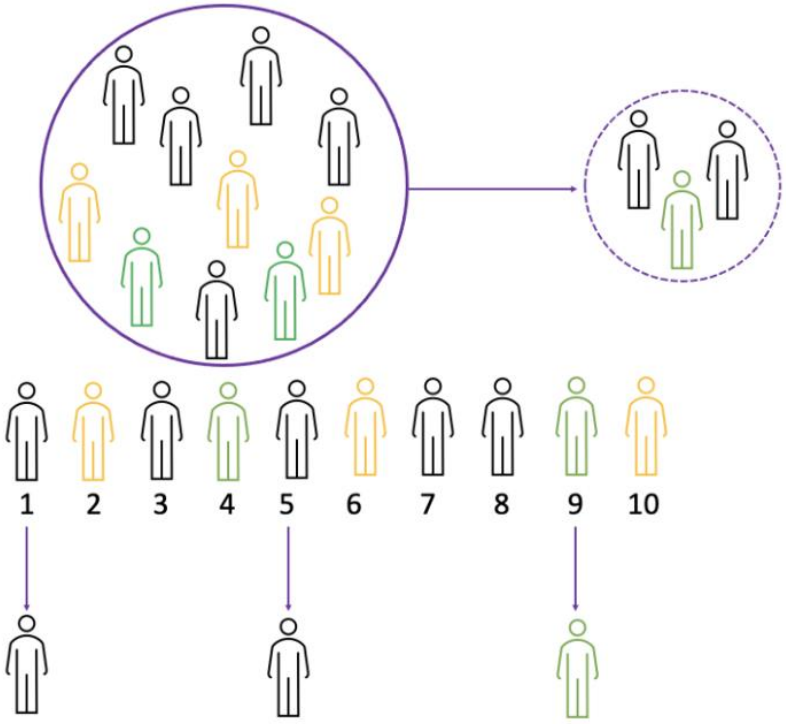
```
import numpy as np

# generating population data following Normal Distribution
N = 10000
mu = 10
std = 2
population_df = np.random.normal(mu,std,N)

# function that creates random sample
def random_sampling(df, n):
    random_sample = np.random.choice(df,replace = False, size = n)
    return(random_sample)
randomSample = random_sampling(population_df, N)
randomSample
```

SYSTEMATIC SAMPLING

Systematic sampling is defined as a probability sampling approach where the elements from a target population are selected from a random starting point and after a fixed *sampling interval*.

Stated differently, systematic sampling is an extended version of probability sampling techniques in which each member of the group is selected at regular periods to form a sample. We calculate the sampling interval by dividing the entire population size by the desired sample size.

Note that, Systematic Sampling usually produces a random sample but is not addressing the bias in the created sample.



SYSTEMATIC SAMPLING: PYTHON IMPLEMENTATION

We generate data that serve as population data as in the previous case. We then create a Python function called **systematic_sample**() that takes population data and interval for the sampling and produces as output a systematic sample.

```
import numpy as np
import pandas as pd
# generating population data following Normal Distribution
N = 10000
mu = 10
std = 2
population_df = np.random.normal(mu,std,N)

# function that creates random sample using Systematic Sampling
def systematic_sampling(df, step):
    id = pd.Series(np.arange(1,len(df),1))
    df = pd.Series(df)
    df_pd = pd.concat([id, df], axis = 1)
    df_pd.columns = ["id", "data"]
    # these indices will increase with the step amount not 1
    selected_index = np.arange(1,len(df),step)
    # using iloc for getting thee data with selected indices
    systematic_sampling = df_pd.iloc[selected_index]
    return(systematic_sampling)

n = 10
step = int(N/n)
sample = systematic_sampling(population_df, step)
sample
```

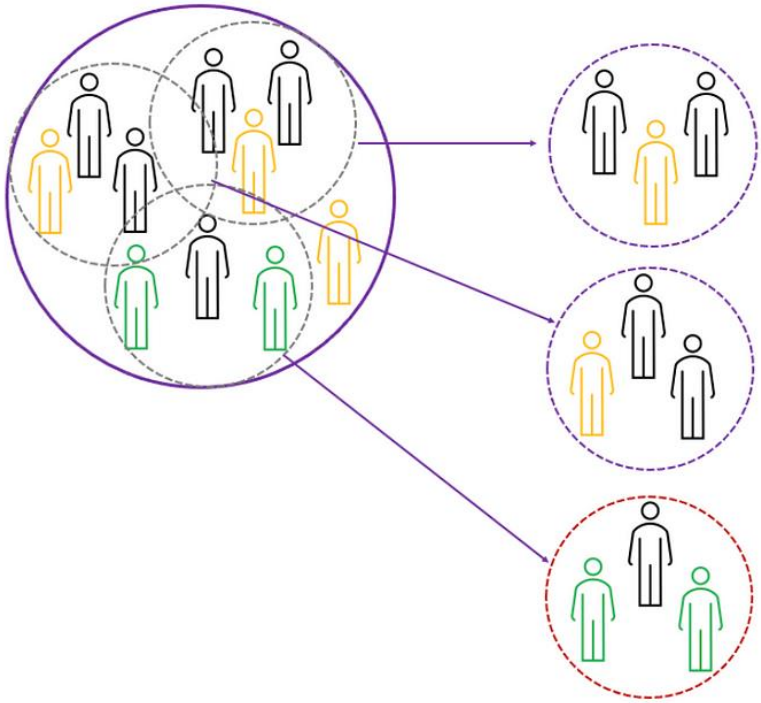
Note that, Systematic Sampling usually produces a random sample but is not addressing the bias in the created sample.

CLUSTER SAMPLING

Cluster sampling is a probability sampling technique where we divide the population into multiple clusters(groups) based on certain clustering criteria. Then we select a random cluster(s) with simple random or systematic sampling techniques. So, in cluster sampling, the entire population is divided into clusters or segments and then cluster(s) are randomly selected.

For example, if you want to conduct an experience evaluating the performance of sophomores in business education across Europe. It is impossible to conduct an experiment that involves a student in every university across the EU. Instead, by using Cluster Sampling, we can group the universities from each country into one cluster. These clusters then define all the sophomore student population in the EU. Next, you can use simple random sampling or systematic sampling and randomly select cluster(s) for the purposes of your research study.

Note that, Systematic Sampling usually produces a random sample but is not addressing the bias in the created sample.



CLUSTER SAMPLING: PYTHON IMPLEMENTATION:

First, we generate data that will serve as population data with 10K observations, and this data consists of the following 4 variables:

- ***Price****:* generated using Uniform distribution,
- ***Id***
- ***event_type****: which is a categorical variable with 3 possible values {type1, type2, type3}*
- ***click***: binary variable taking values {0: no click, 1: click}

```python
import numpy as np
import pandas as pd

# Generating Population data
price_vb = pd.Series(np.random.uniform(1,4,size = N))
id = pd.Series(np.arange(0,len(price_vb),1))
event_type = pd.Series(np.random.choice(["type1","type2","type3"],size = len(price_vb)))
click = pd.Series(np.random.choice([0,1],size = len(price_vb)))
df = pd.concat([id,price_vb,event_type, click],axis = 1)
df.columns = ["id","price","event_type", "click"]
df
```

```
          id     price event_type  click
0         0  1.767794      type2      0
1         1  2.974360      type2      0
2         2  2.903518      type2      0
3         3  3.699454      type2      1
4         4  1.416739      type1      0
...     ...       ...        ...    ...
9995   9995  3.689656      type2      1
9996   9996  1.929186      type3      0
9997   9997  2.393509      type3      1
9998   9998  1.276473      type2      1
9999   9999  3.959585      type2      1

[10000 rows x 4 columns]
```

Then the function **get_clustered_Sample**() takes as inputs the original data, the amount of observations per cluster, and a number of clusters you want to select, and produces as output a clustered sample.

```python
def get_clustered_Sample(df, n_per_cluster, num_select_clusters):
    N = len(df)
    K = int(N/n_per_cluster)
    data = None
    for k in range(K):
        sample_k = df.sample(n_per_cluster)
        sample_k["cluster"] = np.repeat(k,len(sample_k))
        df = df.drop(index = sample_k.index)
        data = pd.concat([data,sample_k],axis = 0)

    random_chosen_clusters = np.random.randint(0,K,size = num_select_clusters)
    samples = data[data.cluster.isin(random_chosen_clusters)]
    return(samples)

sample = get_clustered_Sample(df = df, n_per_cluster = 100, num_select_clusters = 2)
sample
```

```
        id     price   event_type  click  cluster
4847  4847  3.813680      type3      0       17
567    567  1.642347      type2      0       17
8982  8982  3.741744      type3      1       17
2321  2321  2.192724      type3      0       17
5045  5045  3.645671      type2      0       17
...    ...       ...         ...    ...      ...
5681  5681  3.175308      type1      0       90
882    882  2.676477      type2      1       90
2090  2090  3.861775      type3      1       90
907    907  1.947100      type3      0       90
2723  2723  2.557626      type1      0       90

[200 rows x 5 columns]
```

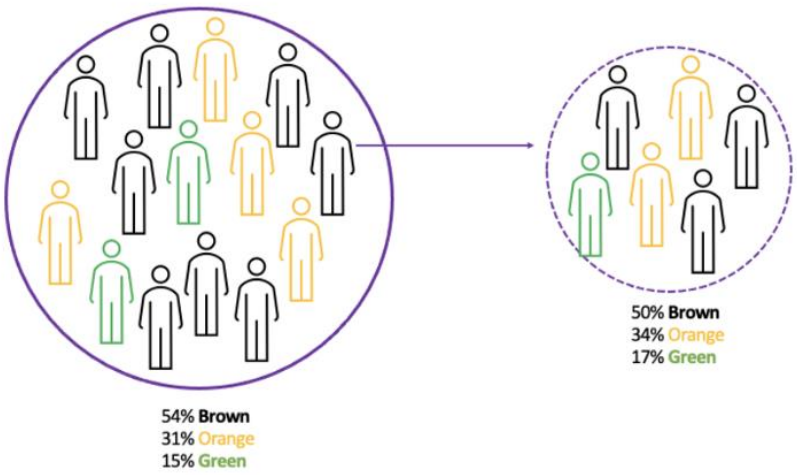
Note that, Cluster Sampling usually produces a random sample but is not addressing the bias in the created sample.

WEIGHTED SAMPLING

In some experiments, you might need items sampling probabilities to be according to weights associated with each item, that's when the proportions of the type of observations should be taken into account. For example, you might need a sample of queries in a search engine with weight as a number of times these queries have been performed so that the sample can be analysed for overall impact on the user experience. In this case, Weighted Sampling is much more preferred compared to Random Sampling or Systematic Sampling.

Weighted Sampling is a data sampling method with weights, that intends to compensate for the selection of specific observations with unequal probabilities (oversampling), non-coverage, non-responses, and other types of bias. If a biased data set is not adjusted and a simple random sampling type of approach is used instead, then the population descriptors (e.g., mean, median) will be skewed and they will fail to correctly represent the population's proportion to the population.

Weighted Sampling addresses the bias in the sample, by creating a sample that takes into account the proportions of the type of observations in the population. Hence, Weighted Sampling usually produces a random and unbiased sample.



Then the function **get_clustered_Sample**() takes as inputs the original data, the amount of observations per cluster, and a number of clusters you want to select, and produces as output a clustered sample.

WEIGHTED SAMPLING: PYTHON IMPLEMENTATION:

The function **get_weighted_sample**() takes as inputs the original data, and the desired sample size, and produces as output a weighted sample. Note that, the proportions, in this case, are defined based on the click event. That is, we compute the proportion of data points that had click events of 1 (let's say X%) and 0 (Y%, where Y% = 100-X%), then we generate a random sample such that, the sample will also contain X% observations with click = 1 and Y% observations with click = 0.

```
def get_weighted_sample(df,n):
    def get_class_prob(x):
        weight_x = int(np.rint(n * len(x[x.click != 0]) / len(df[df.click != 0])))
        sampled_x = x.sample(weight_x).reset_index(drop=True)
        return (sampled_x)
        # we are grouping by the target class we use for the proportions

    weighted_sample = df.groupby('event_type').apply(get_class_prob)
    print(weighted_sample["event_type"].value_counts())
    return (weighted_sample)

sample = get_weighted_sample(df,100)
sample
```

```
                  id     price    event_type  click
event_type
type1       0   6780  1.200188       type1      1
            1   8830  2.990630       type1      1
            2   8997  3.483728       type1      0
            3   7541  2.402993       type1      1
            4   4460  2.959203       type1      0
...              ...       ...         ...    ...
type3       29  5058  3.426289       type3      1
            30  5855  3.852197       type3      0
            31  6295  2.679898       type3      0
            32  8978  1.115072       type3      1
            33  7730  1.208441       type3      1

[100 rows x 4 columns]
```

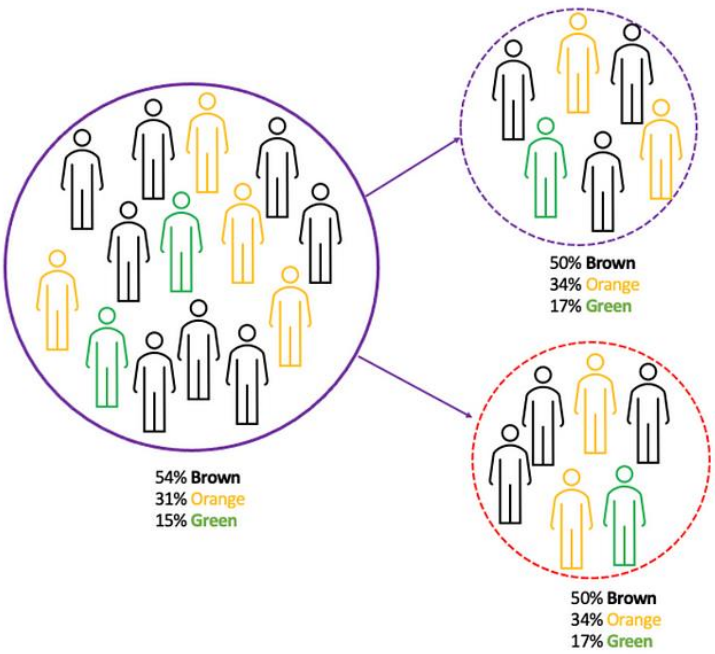
Weighted Sampling usually produces a random and unbiased sample.

STRATIFIED SAMPLING

Stratified Sampling is a data sampling approach, where we divide a population into homogeneous subpopulations called *strata* based on specific characteristics (e.g., age, race, gender identity, location, event type etc.).

Every member of the population studied should be in exactly one stratum. Each stratum is then sampled using Cluster Sampling, allowing data scientists to estimate statistical measures for each sub-population. We rely on Stratified Sampling when the populations' characteristics are diverse and we want to ensure that every characteristic is properly represented in the sample.

So, Stratified Sampling, is simply, the combination of Clustered Sampling and Weighted Sampling.



STRATIFIED SAMPLING: PYTHON IMPLEMENTATION:

The function **get_stratified_sample**() takes as inputs the original data, the desired sample size, the number of clusters needed, and it produces as output a stratified sample. Note that, this function, firstly performs weighted sampling using the **click** event. Secondly, it performs clustered sampling using the **event_type.**

```
def get_startified_sample(df,n,num_clusters_needed):
    N = len(df)
    num_obs_per_cluster = int(N/n)
    K = int(N/num_obs_per_cluster)

    def get_weighted_sample(df,num_obs_per_cluster):
        def get_sample_per_class(x):
            n_x = int(np.rint(num_obs_per_cluster*len(x[x.click !=0])/len(df[df.click !=0])))
            sample_x = x.sample(n_x)
            return(sample_x)
        weighted_sample = df.groupby("event_type").apply(get_sample_per_class)
        return(weighted_sample)

    stratas = None
    for k in range(K):
        weighted_sample_k = get_weighted_sample(df,num_obs_per_cluster).reset_index(drop = True)
        weighted_sample_k["cluster"] = np.repeat(k,len(weighted_sample_k))
        stratas = pd.concat([stratas, weighted_sample_k],axis = 0)
        df.drop(index = weighted_sample_k.index)
    selected_strata_clusters = np.random.randint(0,K,size = num_clusters_needed)
    stratified_samples = stratas[stratas.cluster.isin(selected_strata_clusters)]
    return(stratified_samples)

sample = get_startified_sample(df = df,n = 100,num_clusters_needed = 2)
sample
```

```
       id     price event_type  click  cluster
0    5131  2.707995      type1      0       45
1    5102  1.677190      type1      0       45
2    7370  1.893061      type1      0       45
3    4207  2.491246      type1      0       45
4    8909  3.252655      type1      1       45
..    ...       ...        ...    ...      ...
96   3254  2.637625      type3      0       85
97   1555  1.196040      type3      1       85
98   7627  3.240507      type3      1       85
99   6405  1.607379      type3      0       85
100  1075  2.471806      type3      0       85

[202 rows x 5 columns]
```

Stratified Sampling, is basically, the combination of Clustered Sampling and Weighted Sampling.