Objective:

To understand about different data sampling techniques through Pythonic implementation.

Lets first start with clearning some basic concepts that would be needed for this implementation.

1. Population Vs Sample:

- Population is a super set. Whereas, sample is subset of Population.
- For Example: During elections if we consider data of people polling their votes in a particular state then, the data set thus formed will be treated as Population dataset. Now, to understand Sample lets's consider example of exit polls performed by different media platforms in an attempt to predict the outcome of elections held. In exit poll a sample of people is selected from different regions and based on their feedback, conclusion for whole population is made which is called exit poll. However, we cannot deny the fact that most of these exit polls come out to be false and the reason behind this is that sampling/selection of people done was not appropriate. Selection should cater diversity which not doing so can make our samping biased.

1. Why we need to sample data?

Ans: 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.

We are now good to start discussing different data sampling techniques. We will be discussing following sampling techniques:

- Random Sampling
- Systematic Sampling
- Cluster Sampling
- Stratified Sampling

In [106…

```
#Importing Pythonic libraries that we be required during implementation.
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
```

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 implementation using Python:

In [8]:

```
# Generating population data following Normal Distribution
N = 100
mu = 10
std = 2
population_df = np.random.normal(mu,std,N)
#Above line of code will generate 100 random values following normal distribution with mean=10 and std dev=2
print("Population dataset: \n\n",population_df)
```

```
Population dataset: 

 [11.87201794  9.19410729 10.13480995 10.80907848  9.17771309  6.35283347
 10.63237929  9.1210806  11.12359807  9.43290443  9.76791458 10.91684694
 12.4963343  10.46178451 12.56664108 11.07472206  7.4540457  10.92919438
  9.72040803  5.71890128 11.19850436  9.95538486  9.91927533 12.35089683
 11.4448732   8.45208418 11.40254646  7.07219775 10.07182938  8.24256263
 10.18924272 12.20119338  8.26324406 12.79346388 11.03798394  9.2412469
 10.29070179 10.45945076 10.20900602  8.79734861 10.08414052  8.8095375
  7.57270965 11.12694343  7.82879517  7.93579714  5.6710653  10.24516194
 11.00732927  7.06969169 10.42314058 10.18678414 10.33158434  9.1497616
  9.09489977  7.07996543 10.18882841  9.46098938 10.98996733  9.94900945
 10.30006276  5.49721959 10.55576768  9.95733914  8.92628428 11.6730851
 10.93369093 10.64389695  9.69448563 10.39171933 10.82633658 12.54020367
 10.39049423 10.11674238  7.32647289 10.2050764  10.64710426  7.99310922
 10.06392775  7.69615701 11.87301229  9.50271905 11.03182927 13.0628293
  8.6878569   6.82815884  7.64474515  9.23431871  9.16254145 10.53469048
  9.93290127  4.78405266  8.14657929  9.91647938 10.02837626  9.36565911
 11.88540474 11.04279177  7.88230273 10.11617635]
```

In [10]:

```
# Below line of code generates sample data by randomly choosing 15 data points from population dataset. Note that size=15
# represents sample size.
sample_df = np.random.choice(population_df, size = 15)
print("Sample dataset generated from Population dataset: \n\n",sample_df)
```

```
Sample dataset generated from Population dataset:

 [ 9.72040803 10.11674238  5.6710653  11.03798394 11.07472206 10.39049423
 10.20900602 10.63237929 10.45945076 10.33158434  8.24256263 10.42314058
  8.79734861  7.64474515 11.4448732 ]
```

Systematic sampling implementation using Python:

- Probability sampling approach where the elements from a target population are selected from a random starting point and after a fixed sampling interval.
- It's a sampling technique in which each member of the group is selected at regular periods to form a sample.
- Sampling interval is calculated 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.

In [12]:

```python
# Generating population data following Normal Distribution
N = 100
mu = 10
std = 2
population_df = np.random.normal(mu,std,N)
#Above line of code will generate 100 random values following normal distribution with mean=10 and std dev=2
print("Population dataset: \n\n",population_df)
```

```
Population dataset:

 [ 7.73046212 13.53534289  9.82209079 11.24937157 10.11541466  9.48420511
  8.68820962  8.69098096  9.34811924  8.96001701 11.20892174 11.52138591
 12.02766125 13.07476548 10.77326312  8.34139627 10.91790223  8.44643424
 11.77657739  9.06669507 14.62155786  8.71701942  8.41317915  8.4550526
 11.16654562 10.62468591  5.67312983 12.44272423 12.68496501 10.54311696
  9.69886362 11.23620827 11.63825538 13.67188866 10.44557911  9.33109983
 10.06822814 10.60048608  7.85583702  9.70058034 10.93420836 10.05978754
 11.03081284  9.79297793 11.25809435 11.47799052 10.40903942  7.96040268
  7.67875673 16.17034952 13.18626635  8.48672796 12.53037318 11.92936543
  8.23052911  9.60695157  8.96765409  8.59767551  9.96984678  8.42741936
  9.77189593 11.78400764 10.70069716 12.48055828  9.68842104 11.87648734
  8.08186167 11.07027796 13.51828574  7.60675945 12.13149915 11.46574127
 12.34859896 10.01947431 11.99057886 11.08353044  8.61592796 11.97117253
 13.18805412  7.97172139  8.68587879  7.13066183 11.17278703  9.04809312
 12.4872701  10.36605971  8.64710078  9.25363189  6.69712707  9.46953091
 10.9363169  11.10802659  9.22695701  8.79890468 11.98593287  7.27294637
  8.41321662  9.91937603 11.30329542 10.40820749]
```

In [29]:

```python
#Generating sample data
```

```python
n = 20  #Size of sample data to be extracted from population dataset
N = len(population_df) #Size of population dataset
step = int(N/n) #step represents sampling interval
print(type(population_df))
print("\n"+"="*35)

#Now we will convert our population data into series/dataframe

#print(np.arange(1,len(population_df)+1,1)) --> this will generate a array starting from 1 untill 100 with jump/step of 1
#In below line of code we are coverting array into series datatype. This will act as ID that can be used to uniquely identify
#values
ID = pd.Series(np.arange(1,len(population_df)+1,1))
print("IDs for uniquely identifying values:\n",ID)
print("\n"+"="*35)

df = pd.Series(population_df) #Converting population data in series
#print(df)

df_pd = pd.concat([ID, df], axis = 1) #Concatenating ID and df along the column
df_pd.columns = ["id", "data"]        #Changing default indexes(0,1,2,3,...) along the column to user defined named indexes
print("Population data in array type get converted into DATAFRAME type:\n",df_pd)
print(type(df_pd))   #this proves that on combining 2 or more series it gets converted into dataframe.
print("\n"+"="*35)

# these indices will increase with the step amount not 1
selected_index = np.arange(0,len(population_df),step)   #defining indexes to be selected based on sampling interval
print("Selected indexes based on which sampling in certain intervals will occur: \n",selected_index)
print("\n"+"="*35)

# using iloc for getting the data with selected indices
systematic_sampling = df_pd.iloc[selected_index] #note that this by default does indexing based on row indexes

print("Dataset produced after systematic sampling with sample size 20: \n",systematic_sampling)
```

```
<class 'numpy.ndarray'>

===================================
IDs for uniquely identifying values:
 0       1
1       2
2       3
3       4
4       5
     ...
95     96
96     97
97     98
98     99
99    100
```

```
Length: 100, dtype: int32

===================================
Population data in array type get converted into DATAFRAME type:
      id       data
0      1   7.730462
1      2  13.535343
2      3   9.822091
3      4  11.249372
4      5  10.115415
..   ...        ...
95    96   7.272946
96    97   8.413217
97    98   9.919376
98    99  11.303295
99   100  10.408207

[100 rows x 2 columns]
<class 'pandas.core.frame.DataFrame'>

===================================
Selected indexes based on which sampling in certain intervals will occur:
 [ 0  5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95]

===================================
Dataset produced after systematic sampling with sample size 20:
     id       data
0     1   7.730462
5     6   9.484205
10   11  11.208922
15   16   8.341396
20   21  14.621558
25   26  10.624686
30   31   9.698864
35   36   9.331100
40   41  10.934208
45   46  11.477991
50   51  13.186266
55   56   9.606952
60   61   9.771896
65   66  11.876487
70   71  12.131499
75   76  11.083530
80   81   8.685879
85   86  10.366060
90   91  10.936317
95   96   7.272946
```

Note:

In above cell we proved that dataframe (which is a pandas object) is a combination of 2 or more series.

Cluster Sampling implementation using Python:

- In cluster sampling, the entire population is divided into clusters or segments using some criteria and then cluster(s) are randomly selected using simple random or systematic sampling techniques.
- For example, if you want to conduct an experience evaluating the performance of high school students in business education across Europe. It is impossible to conduct an experiment that involves a student in every university across the Europe. Instead, by using Cluster Sampling, we can group the universities from each country into one cluster. These clusters then define all the high school student population in the Europe. 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.

In [69]:

```
# Creating Population data of size 100 where price_vb represents uniformally distributed prices, ID represents unique
# identifier, type representing type of item and click represents binary values where 0 means no click and 1 means click

# Generating price of uniform values with lower limit 1 and upper limit 4(excluded) and size 100
price_vb = pd.Series(np.random.uniform(1,4,size = 100))
print("Price:\n",price_vb)
print("\n"+35*"=")

# Generating 100 IDs to uniquely identify enteries
ID = pd.Series(np.arange(0,len(price_vb),1))

# Generating Event type of size 100 containing values: "type1","type2" and "type3"
'''
Eg: Generate a uniform random sample from np.arange(5) of size 3:
>>>np.random.choice(5, 3)
O/P: array([0, 3, 4])
'''
#In below line of code we are randomly choosing 100 values from list containing "type1","type2" and "type3" string values
event_type = pd.Series(np.random.choice(["type1","type2","type3"],size = len(price_vb)))
print("Event Type:\n",event_type)
print("\n"+35*"=")

# Generating click of size 100 containing randomly selected values 0 or 1
click = pd.Series(np.random.choice([0,1],size = len(price_vb)))
print("Click:\n",click)
print("\n"+35*"=")

#Concatenating all the features of series type along the column to form population dataset of size 100 and of type: DataFrame
df = pd.concat([ID,price_vb,event_type, click],axis = 1)
```

```python
df.columns = ["ID","price","event_type", "click"]
print("Population dataset:\n",df)
```

```
Price:
 0     3.316366
1     3.129744
2     1.089180
3     2.177873
4     2.523790
        ...
95    1.815636
96    2.112071
97    3.759438
98    1.013373
99    1.511597
Length: 100, dtype: float64

===================================
Event Type:
 0     type2
1     type3
2     type1
3     type1
4     type2
      ...
95    type3
96    type2
97    type2
98    type3
99    type2
Length: 100, dtype: object

===================================
Click:
 0     1
1     1
2     1
3     0
4     1
     ..
95    1
96    1
97    1
98    0
99    0
Length: 100, dtype: int32

===================================
Population dataset:
     ID     price event_type  click
0    0  3.316366      type2      1
```

```
1    1  3.129744      type3      1
2    2  1.089180      type1      1
3    3  2.177873      type1      0
4    4  2.523790      type2      1
..  ..       ...        ...    ...
95  95  1.815636      type3      1
96  96  2.112071      type2      1
97  97  3.759438      type2      1
98  98  1.013373      type3      0
99  99  1.511597      type2      0

[100 rows x 4 columns]
```

In [70]:

```
#Generating clusters



N = len(df)  #Population size
n_per_cluster = 25   #Size of each cluster/sample
K = int(N/n_per_cluster) #Number of clusters/samples in entire population
data = None
for k in range(K):
    sample_k = df.sample(n_per_cluster)  #taking a random sample from popluation of size = n_per_cluster to form a cluster
                            #(Simple random sampling technique used for selecting data after dividing into clusters.
                            #Here, criteria of cluster division is simply dividing population size by sample size per cluster
    print("Cluster no,:",k+1)
    print("\n", sample_k)
    '''
    Eg: Obtain an array by repeating 3, four time
        >>>  np.repeat(3, 4)
        O/P: array([3, 3, 3, 3])
    '''
    #In the below line of code we are adding a new column with name "cluster". Records that will have "cluster" value as 1
    #will represent that, that particular record belongs to 1st cluster. Similarly if 2, represents record belong to 2nd clus
    #And similarly, 3 represents 3rd cluster whereas, 4 represents 4th cluster.
    sample_k["cluster"] = np.repeat(k+1,len(sample_k))
    #df = df.drop(index = sample_k.index)
    data = pd.concat([data,sample_k],axis = 0)  #concatenating data along the columns

print(data)
```

```
Cluster no,: 1

     ID     price event_type  click
89  89  2.135595      type2      1
52  52  3.074140      type1      1
95  95  1.815636      type3      1
81  81  1.760794      type2      0
5    5  3.521768      type2      0
```

```
36  36  1.599077      type3      0
78  78  1.724693      type2      0
32  32  2.522665      type2      0
86  86  3.836545      type2      0
4    4  2.523790      type2      1
70  70  3.364422      type1      1
3    3  2.177873      type1      0
84  84  2.594029      type3      1
99  99  1.511597      type2      0
43  43  2.154949      type1      0
37  37  2.455719      type3      0
85  85  3.823000      type3      1
47  47  1.627933      type2      0
26  26  1.726931      type3      1
33  33  2.731100      type1      1
66  66  1.812807      type2      0
18  18  2.603091      type1      0
27  27  1.863253      type2      1
30  30  1.389713      type1      1
63  63  2.318162      type1      0
Cluster no,: 2

     ID     price event_type  click
91  91  2.206243      type2      0
79  79  3.455065      type2      1
38  38  2.882295      type3      1
13  13  2.296898      type1      1
96  96  2.112071      type2      1
63  63  2.318162      type1      0
78  78  1.724693      type2      0
14  14  3.982859      type2      0
29  29  3.345522      type2      1
74  74  1.187533      type2      1
51  51  3.848946      type2      0
11  11  3.930611      type2      1
4    4  2.523790      type2      1
9    9  2.892479      type2      0
90  90  2.930388      type2      1
15  15  2.859826      type2      1
94  94  3.708997      type2      1
97  97  3.759438      type2      1
99  99  1.511597      type2      0
95  95  1.815636      type3      1
98  98  1.013373      type3      0
20  20  1.948146      type2      0
41  41  1.561570      type2      1
46  46  1.204428      type3      0
64  64  3.108955      type3      1
Cluster no,: 3

     ID     price event_type  click
```

```
74  74  1.187533      type2      1
92  92  3.559090      type1      0
7    7  3.807257      type3      1
33  33  2.731100      type1      1
96  96  2.112071      type2      1
73  73  2.237065      type1      0
31  31  1.070862      type3      0
22  22  3.825528      type2      0
76  76  3.229898      type3      1
69  69  3.462679      type1      0
78  78  1.724693      type2      0
50  50  2.713694      type2      0
98  98  1.013373      type3      0
68  68  1.544435      type1      1
48  48  1.828603      type2      0
9    9  2.892479      type2      0
19  19  2.720405      type3      1
84  84  2.594029      type3      1
72  72  2.766854      type1      1
64  64  3.108955      type3      1
16  16  2.489658      type3      1
24  24  2.973800      type3      1
30  30  1.389713      type1      1
26  26  1.726931      type3      1
71  71  3.922616      type1      0
Cluster no,: 4

     ID     price event_type  click
77  77  3.409212      type3      0
55  55  3.924044      type3      1
94  94  3.708997      type2      1
64  64  3.108955      type3      1
14  14  3.982859      type2      0
29  29  3.345522      type2      1
93  93  2.455821      type3      1
15  15  2.859826      type2      1
23  23  1.926942      type1      0
75  75  2.767531      type3      0
31  31  1.070862      type3      0
25  25  2.375453      type1      1
98  98  1.013373      type3      0
61  61  2.352222      type3      1
92  92  3.559090      type1      0
46  46  1.204428      type3      0
65  65  1.515188      type1      1
9    9  2.892479      type2      0
84  84  2.594029      type3      1
36  36  1.599077      type3      0
42  42  2.450618      type3      0
48  48  1.828603      type2      0
16  16  2.489658      type3      1
```

```
50  50  2.713694      type2      0
33  33  2.731100      type1      1
    ID     price event_type  click  cluster
89  89  2.135595      type2      1        1
52  52  3.074140      type1      1        1
95  95  1.815636      type3      1        1
81  81  1.760794      type2      0        1
5    5  3.521768      type2      0        1
..  ..       ...        ...    ...      ...
42  42  2.450618      type3      0        4
48  48  1.828603      type2      0        4
16  16  2.489658      type3      1        4
50  50  2.713694      type2      0        4
33  33  2.731100      type1      1        4

[100 rows x 5 columns]
```

In [71]:

```
data['cluster'].value_counts()
## this represents that each cluster is containing equal number of data points.
```

Out[71]:

```
1    25
2    25
3    25
4    25
Name: cluster, dtype: int64
```

In [72]:

```
num_select_clusters=2  #this represents the number of clusters that user wishes to choose to carry out their experiment/resea
random_chosen_clusters = np.random.randint(1,K,size = num_select_clusters)
print("Randomly choosen cluster to perform research or experiment:\n",random_chosen_clusters)
samples = data[data.cluster.isin(random_chosen_clusters)]
#In above line of code we are simply matching cluster column of data with random_chosen_clusters using isin operation followe
#by returing all the data thus obtained.
```

```
Randomly choosen cluster to perform research or experiment:
 [3 1]
```

In [73]:

```
#Finally obtained sample data using Clustering sample data technique
samples
```

Out[73]:

	ID	price	event_type	click	cluster
89	89	2.135595	type2	1	1
52	52	3.074140	type1	1	1
95	95	1.815636	type3	1	1
81	81	1.760794	type2	0	1

5	5	3.521768	type2	0	1
36	36	1.599077	type3	0	1
78	78	1.724693	type2	0	1
32	32	2.522665	type2	0	1
86	86	3.836545	type2	0	1
4	4	2.523790	type2	1	1
70	70	3.364422	type1	1	1
3	3	2.177873	type1	0	1
84	84	2.594029	type3	1	1
99	99	1.511597	type2	0	1
43	43	2.154949	type1	0	1
37	37	2.455719	type3	0	1
85	85	3.823000	type3	1	1
47	47	1.627933	type2	0	1
26	26	1.726931	type3	1	1
33	33	2.731100	type1	1	1
66	66	1.812807	type2	0	1
18	18	2.603091	type1	0	1
27	27	1.863253	type2	1	1
30	30	1.389713	type1	1	1
63	63	2.318162	type1	0	1
74	74	1.187533	type2	1	3
92	92	3.559090	type1	0	3
7	7	3.807257	type3	1	3
33	33	2.731100	type1	1	3
96	96	2.112071	type2	1	3
73	73	2.237065	type1	0	3
31	31	1.070862	type3	0	3
22	22	3.825528	type2	0	3
76	76	3.229898	type3	1	3

69	69	3.462679	type1	0	3
78	78	1.724693	type2	0	3
50	50	2.713694	type2	0	3
98	98	1.013373	type3	0	3
68	68	1.544435	type1	1	3
48	48	1.828603	type2	0	3
9	9	2.892479	type2	0	3
19	19	2.720405	type3	1	3
84	84	2.594029	type3	1	3
72	72	2.766854	type1	1	3
64	64	3.108955	type3	1	3
16	16	2.489658	type3	1	3
24	24	2.973800	type3	1	3
30	30	1.389713	type1	1	3
26	26	1.726931	type3	1	3
71	71	3.922616	type1	0	3

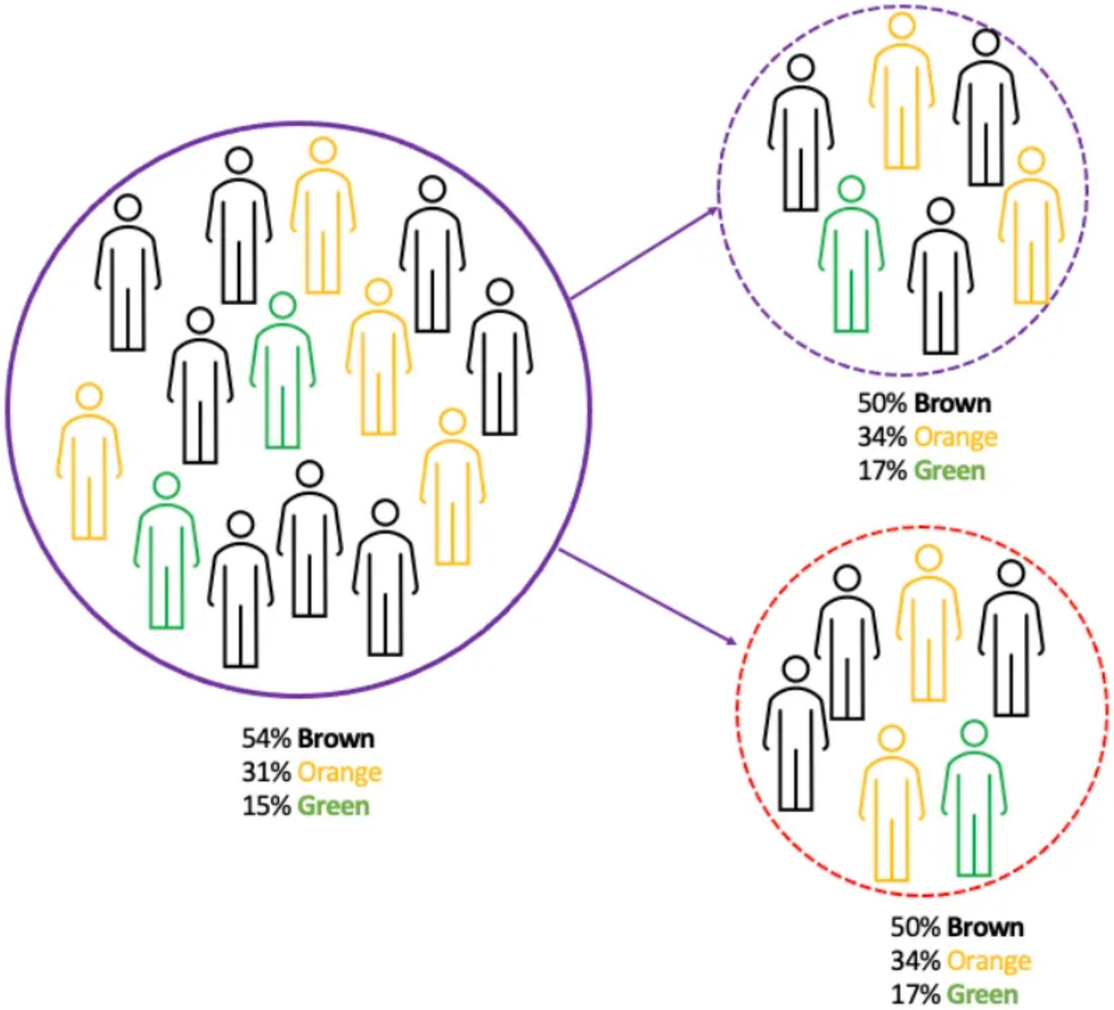
In [75]:

```
print(len(samples))
#Observer that in our current execution cluster 3 and 1 get randomlly choosen. Each one of these clusters is having 25
#data points. This is the reason why we are getting length as 50.
```

```
50
```

Stratified Sampling implementation using Python:

- Stratified sampling is weighted clustering sampling.
- It is the sampling technique 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.
- 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.



In [104…

```
# To understand stratified sampling we use pima dataset that based on various independent features predicts where a person
# is having diabetes(Outcome=1) or not having diabetes(represented by Outcome with binary 0)

pima_df = pd.read_csv("https://raw.githubusercontent.com/npradaschnor/Pima-Indians-Diabetes-Dataset/master/diabetes.csv")
pima_df.head()
```

Out[104…

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1

3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

In [105…

```
# Here, we are determing portion/proportions of 0 and 1 in population dataset

pima_df["Outcome"].value_counts(normalize=True)*100
```

Out[105…

```
0    65.104167
1    34.895833
Name: Outcome, dtype: float64
```

In [108…

```
#Checking whether null value is present or not

pima_df.isnull().sum()
```

Out[108…

```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

In [109…

```
# Grouping independent features

X = pima_df[["Pregnancies","Glucose","BloodPressure","SkinThickness","Insulin","BMI","DiabetesPedigreeFunction","Age"]]
```

In [110…

```
X
```

Out[110…

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	6	148	72	35	0	33.6	0.627	50
1	1	85	66	29	0	26.6	0.351	31
2	8	183	64	0	0	23.3	0.672	32
3	1	89	66	23	94	28.1	0.167	21
4	0	137	40	35	168	43.1	2.288	33

...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63
764	2	122	70	27	0	36.8	0.340	27
765	5	121	72	23	112	26.2	0.245	30
766	1	126	60	0	0	30.1	0.349	47
767	1	93	70	31	0	30.4	0.315	23

768 rows × 8 columns

In [112…

```
# Intializing dependent feature

y = pima_df[["Outcome"]]
y
```

Out[112…

	Outcome
0	1
1	0
2	1
3	0
4	1
...	...
763	0
764	0
765	0
766	1
767	0

768 rows × 1 columns

Sampling popluation dataset without weights or stratification

In [157…

```
# Spliting Population dataset into train and test dataset. Note that train and test here can be regarded as Samples of
# population
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.35, random_state=1)
```

In [158…

```
X_train
```

Out[158…

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
487	0	173	78	32	265	46.5	1.159	58
495	6	166	74	0	0	26.6	0.304	66
723	5	117	86	30	105	39.1	0.251	42
554	1	84	64	23	115	36.9	0.471	28
623	0	94	70	27	115	43.5	0.347	21
...	...	...	...	...	...	...	...	...
645	2	157	74	35	440	39.4	0.134	30
715	7	187	50	33	392	33.9	0.826	34
72	13	126	90	0	0	43.4	0.583	42
235	4	171	72	0	0	43.6	0.479	26
37	9	102	76	37	0	32.9	0.665	46

499 rows × 8 columns

In [159…

```
X_test
```

Out[159…

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
285	7	136	74	26	135	26.0	0.647	51
101	1	151	60	0	0	26.1	0.179	22
581	6	109	60	27	0	25.0	0.206	27
352	3	61	82	28	0	34.4	0.243	46
726	1	116	78	29	180	36.1	0.496	25
...	...	...	...	...	...	...	...	...
429	1	95	82	25	180	35.0	0.233	43
368	3	81	86	16	66	27.5	0.306	22
62	5	44	62	0	0	25.0	0.587	36

447	0	95	80	45	92	36.5	0.330	26
528	0	117	66	31	188	30.8	0.493	22

269 rows × 8 columns

In [160…

```
y_train
```

Out[160…

	Outcome
487	0
495	0
723	0
554	0
623	0
...	...
645	0
715	1
72	1
235	1
37	1

499 rows × 1 columns

In [161…

```
y_test
```

Out[161…

	Outcome
285	0
101	0
581	0
352	0
726	0
...	...

429	1
368	0
62	0
447	0
528	0

269 rows × 1 columns

In [165…

```
print("Proportions of 0 and 1 in train split before stratification: \n",y_train["Outcome"].value_counts(normalize=True)*100)
print("\n====================================================\n")
print("Proportions of 0 and 1 in test split before stratification: \n",y_test["Outcome"].value_counts(normalize=True)*100)
```

```
Proportions of 0 and 1 in train split before stratification:
 0    65.330661
1    34.669339
Name: Outcome, dtype: float64

====================================================

Proportions of 0 and 1 in test split before stratification:
 0    64.684015
1    35.315985
Name: Outcome, dtype: float64
```

Note:

In population dataset proportions of 0 and 1 were 65.104167 and 34.895833 respectively. According to stratification sampling if we are splitting our population dataset in train and test datasets(samples) these samples (with each other) should also have similar proportions of 0 and 1.

But, above we can see that 0 and 1 for train and test are are not in proportion. For eg: For training our model we are provided with less portion of 1s (34.669339) as compared 1s in test dataset (35.315985). In such a case where there is unequal division (this is called oversampling), when model is trained it might miss the trends due to which model will start acting as a biased learning model.

In order to overcome above stated problem stratification is needed during spliting the data from population dataset.

Sampling popluation dataset with weights or stratification

In [163…

```
# By introducing an additional parameter "stratify=y" we are enabling stratification during train_test_split.
# stratify=y means, yes perform stratification.
```

```python
Xnew_train, Xnew_test, ynew_train, ynew_test = train_test_split(X, y, test_size=0.35, random_state=42, stratify=y)
```

In [166…

```python
print("Proportions of 0 and 1 in train split after stratification: \n",ynew_train["Outcome"].value_counts(normalize=True)*10
print("\n====================================================\n")
print("Proportions of 0 and 1 in test split after stratification: \n",ynew_test["Outcome"].value_counts(normalize=True)*100)
```

```
Proportions of 0 and 1 in train split after stratification: 
 0    65.130261
1    34.869739
Name: Outcome, dtype: float64

====================================================

Proportions of 0 and 1 in test split after stratification: 
 0    65.055762
1    34.944238
Name: Outcome, dtype: float64
```

Note:

Now, observe that proportions of 0 and 1 in train and test data splits are almost similar once stratification sampling technique is utilized. This will ensure that during model training model learns all the trends. Also, this will reduce over sampling problem and will make our learning model unbaised.

Therefore, we can conclude that we should set stratify=y during train_test_split

References:

1. https://towardsdatascience.com/data-sampling-methods-in-python-a4400628ea1b
2. https://www.youtube.com/watch?v=ixBbAZDS7TU

In [ ]: