Practice - sampling techniques

reg no 2148059

In [1]:

```python
# Import required libraries
import numpy as np
import pandas as pd

# Set random seed
np.random.seed(42)

# Define total number of products
number_of_products = 10

# Create data dictionary
data = {'product_id':np.arange(1, number_of_products+1).tolist(),
       'measure':np.round(np.random.normal(loc=10, scale=0.5, size=

# Transform dictionary into a data frame
df = pd.DataFrame(data)

# Store the real mean in a separate variable
real_mean = round(df['measure'].mean(),3)

# View data frame
df
```

Out[1]:

	product_id	measure
0	1	10.248
1	2	9.931
2	3	10.324
3	4	10.762
4	5	9.883
5	6	9.883
6	7	10.790
7	8	10.384
8	9	9.765
9	10	10.271

SIMPLE RANDOM SAMPLING

In [2]:
```python
# Obtain simple random sample
simple_random_sample = df.sample(n=4).sort_values(by='product_id')

# Save the sample mean in a separate variable
simple_random_mean = round(simple_random_sample['measure'].mean(),3

# View sampled data frame
simple_random_sample
```

Out[2]:

	product_id	measure
2	3	10.324
6	7	10.790
7	8	10.384
8	9	9.765

SYSTEMATIC SAMPLING

In [3]:
```python
# Define systematic sampling function
def systematic_sampling(df, step):

    indexes = np.arange(0,len(df),step=step)
    systematic_sample = df.iloc[indexes]
    return systematic_sample

# Obtain a systematic sample and save it in a new variable
systematic_sample = systematic_sampling(df, 3)

# Save the sample mean in a separate variable
systematic_mean = round(systematic_sample['measure'].mean(),3)

# View sampled data frame
systematic_sample
```

Out[3]:

	product_id	measure
0	1	10.248
3	4	10.762
6	7	10.790
9	10	10.271

CLUSTER SAMPLING

In [4]:

```
def cluster_sampling(df, number_of_clusters):

    try:
        # Divide the units into cluster of equal size
        df['cluster_id'] = np.repeat([range(1,number_of_clusters+1)

        # Create an empty list
        indexes = []

        # Append the indexes from the clusters that meet the criter
        # For this formula, clusters id must be an even number
        for i in range(0,len(df)):
            if df['cluster_id'].iloc[i]%2 == 0:
                indexes.append(i)
        cluster_sample = df.iloc[indexes]
        return(cluster_sample)

    except:
        print("The population cannot be divided into clusters of eq

# Obtain a cluster sample and save it in a new variable
cluster_sample = cluster_sampling(df,5)

# Save the sample mean in a separate variable
cluster_mean = round(cluster_sample['measure'].mean(),3)

# View sampled data frame
cluster_sample
```

Out[4]:

	product_id	measure	cluster_id
2	3	10.324	2
3	4	10.762	2
6	7	10.790	4
7	8	10.384	4

STRATIFIED SAMPLING

In [5]:

```python
# Create data dictionary
data = {'product_id':np.arange(1, number_of_products+1).tolist(),
        'product_strata':np.repeat([1,2], number_of_products/2).toli
        'measure':np.round(np.random.normal(loc=10, scale=0.5, size=

# Transform dictionary into a data frame
df = pd.DataFrame(data)

# View data frame
df
```

Out[5]:

	product_id	product_strata	measure
0	1	1	8.780
1	2	1	10.302
2	3	1	9.874
3	4	1	9.918
4	5	1	9.262
5	6	2	10.743
6	7	2	9.988
7	8	2	10.178
8	9	2	10.209
9	10	2	10.416

In [6]:

```python
# Import StratifiedShuffleSplit
from sklearn.model_selection import StratifiedShuffleSplit

# Set the split criteria
split = StratifiedShuffleSplit(n_splits=1, test_size=4)

# Perform data frame split
for x, y in split.split(df, df['product_strata']):
    stratified_random_sample = df.iloc[y].sort_values(by='product_i

# View sampled data frame
stratified_random_sample

# Obtain the sample mean for each group
stratified_random_sample.groupby('product_strata').mean().drop(['pr
```

Out[6]:

	measure
product_strata	
1	9.327
2	10.476

OVERSAMPLING AND UNDERSAMPLING