In many cases, there is a need to split a userbase into 2 or more buckets. For example:

- **UCG:** Many companies that run promotional campaigns, in order to quantify and evaluate the performance of the campaigns, create a Universal Control Group (UCG) which is a random sample of the userbase and does not receive any offer or message.
- **Bucketize**: For testing purposes, it is common to split the userbase into buckets so that to be able to compare them in a long term.
- **Samples for Machine Learning**: A userbase can become too large for a machine learning model to run and for that reason, it is common to get random samples.

The requirements

For the cases that we mentioned above, the splitting algorithm must satisfy the following two requirements:

1. There should be a mapping function so that every time we encounter an **existing user** to be assigned to the same group. For instance, if the **UserID** `152514` was initially assigned to UCG, then it will always be to UCG group.
2. There should be a mapping function so that every **new user** to be assigned to a group.

We can fulfil the requirements above by applying the modulo operation.

Example of Splitting the Userbase with Modulo

Let's see how we can split the Userbase into two buckets. Let's say that we want the **20%** of the users to be in **UCG** and the rest **80%** to be **Control**. Usually the UserIDs will be hashed, according to GDPR compliance. Below we generate some random data:

```
library(tidyverse)
library(digest)
library(Rmpfr)
set.seed(5)


df<-tibble(Row_Number = seq(1,100000))


df<-df%>%rowwise%>%mutate(Hash_Name = digest(paste(sample(LETTERS, 10,
replace = TRUE), collapse = ""),
                                                algo="md5", serialize=F),
                          Event_Date = lubridate::as_datetime( runif(1,
1546290000, 1577739600)))


head(df)
```

Output:

```
# A tibble: 6 x 3
# Rowwise:
  Row_Number Hash_Name                         Event_Date
```

```
1           1 275db34231203750f10adb24c76b9619 2019-06-10 06:15:33
2           2 9a449c58ac6baed3b3648f0f3b5f8084 2019-03-27 21:38:34
3           3 e28e89ab554739a982c862cccf024464 2019-12-02 15:43:48
4           4 45b9aea890d3b98419cae72bb497e94b 2019-10-18 18:58:23
5           5 c4ce7434621d08f5195fbd1bfc1c20c2 2019-08-09 06:14:45
6           6 0b8a304be1015cacfcf31dd40ef6a381 2019-04-10 08:07:28
```

In order to generate random numbers, it is better to choose prime number for the modulo operation. For this example we will take the **997** which is a prime number. The other thing that we need to do, is to convert the MD5 Hashed to numeric. We can do it with the `Rmpfr` library in R. To sum up:

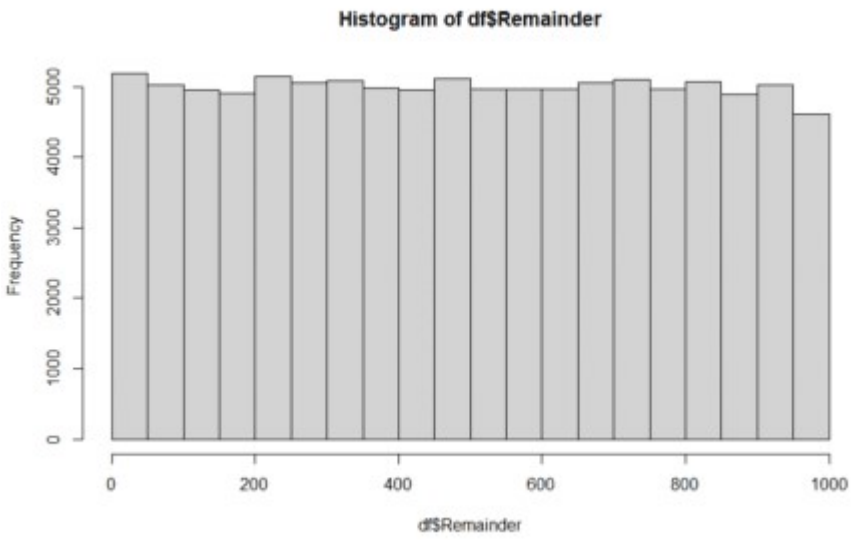
- We will convert the MD5 to numeric
- We will divide the above number by **997** and we will keep store the remainder

```
df$Remainder <- as.numeric(mpfr(df$Hash_Name, base=16) %% 997)
```

Is it Random

This approach generates pseudo-random numbers. Let's see if the distribution of the numbers (from 0 to 996) is random.

```
hist(df$Remainder)
```



We can apply a Chi-Square test too.

```
chisq.test(table(df$Remainder))
```

Output:

```
        Chi-squared test for given probabilities

data:  table(df$Remainder)
X-squared = 995.2, df = 996, p-value = 0.5012
```

The **P-value is 0.5012** which implies that the generated numbers can be considered random.

Now, we can split our UB into **UCG** and **Control** as follows:

If the remainder is less than 200 then UCG else Control

```
df$Group <- ifelse(df$Remainder<200, 'UCG', 'Control')
```

```
df
```

```
# A tibble: 100,000 x 5
# Rowwise:
   Row_Number Hash_Name                       Event_Date          Remainder Group
        <int> <chr>                           <dttm>                  <dbl> <chr>
 1          1 4f4bc108e2bc6ecd7b0dfc8ee4c289b3 2019-02-21 18:26:41      404 Control
 2          2 108d546754281be1df0f99a292afcc96 2019-08-22 15:58:58       38 UCG
 3          3 d05a0d27937e66c1b937264ed82e0efc 2019-12-04 04:55:41      506 Control
 4          4 fa4cd3629a67731ac1c43c32f0176557 2019-11-13 15:45:50      576 Control
 5          5 734c160291f6aa8a94967b0adb06cd7b 2019-03-08 12:20:29      794 Control
 6          6 5f956d4a8deb46aef66be9d73f196690 2019-09-25 18:42:50      258 Control
 7          7 3af5528168f3a76c38168c0339a7f435 2019-11-02 21:42:51      358 Control
 8          8 e09abc7536e69d9f38cd32a68c19941e 2019-11-04 17:06:11      202 Control
 9          9 5cea04d57e328928e77a39730074a862 2019-12-13 09:11:18      140 UCG
10         10 437436faa794ae04be7b5c6eab561065 2019-10-07 22:57:28      528 Control
# ... with 99,990 more rows
```

Check the Proportions

Finally, we want to make sure that the proportion is 80% vs 20% for Control and UCG respectively.

```
prop.table(table(df$Group))
```

Output:

```
Control     UCG
0.80002 0.19998
```

Conclusion

We can use the modulo function to split a userbase in a reproducible and efficient way.