Effective business simulation: the core values for business simulations with alteryx MD5 hashing function

(Henry T.H. Tu, 07-nov-2019)

We will use alteryx hashing function MD5_ASCII() instead of rand() function to generate business codes, categories, metrics. This method provides better control and creativity over simulations.

1.1. Random numbers with MD5 hashing function

We make use of the uniformly-distribution feature of the MD5 hashing function to generate uniform values instead of using rand() function. This method gives us more degrees of freedom to derive and to control the random sequence than does the traditional seeding method (aka linear congruential method). If we hash or transform a text sequence with MD5, we can get one random hashed string as in column [md5]. If we remove the letters and take first 7 digits of the sequence to form a number, we will get a seven-digit number. And we divide the seven-digit number by 10000000, we will get a random number between 0 and 1. These random numbers are core values to form any business simulation.

text	md5	md5_digits	unif
abcd 123456 this is a long string hashing or seeding	MD5_ASCII([text])	REGEX_Replace([md5], "\D+", "")	tonumber(left([md5_digits], 7))/pow(10, 7)

The table below shows the step-by-step transformation of a text column (text sequence) into a uniform column (uniformly-distributed random variable). The hashed value in column [md5] is hexadecimal string. Technically, you can change this value into a big integer number. For simplicity, we just take only first 7 digits to form a random value.

	text	md5	md5_digits	unif
1	abcd	e2fc714c4727ee9395f324cd2e7f331f	27144727939532427331	0.271447
2	123456	e10adc3949ba59abbe56e057f20f883e	103949595605720883	0.10395
3	this is a long string	5734266cd023df9ad0a9533f75d29ce8	573426602390953375298	0.573427
4	hashing or seeding	3091262b87a690ed1ee6b3ac535ff914	309126287690163535914	0.309126

1.2. Identifiers and reference codes

When we want to refer to a row in a datatable, or when we want to link two datatables, we need a code or a reference number. You may use the row number, but if you add or remove rows, all the reference will change and you may refer to the wrong row. We normally use rc7 or rc16 to identify business entities. For example, when you open a bank account, the bank gives you a card with 16 digits (rc16).

Increasing sequence	code7	code16
1 2 3 4	"c" + left(regex_replace(MD5_ASCII([RowCount] + "/U1") , "\D+", ""), 7)	"c" + left(regex_replace(MD5_ASCII([RowCount] + "/part1") + MD5_ASCII([RowCount] + "/part2") , "\D+", ""), 16)

The table below shows the results of the code columns we generate from the column [RowCount]. This is the simplest way to generate the reference codes, which look like bank account code. The prefix string "c" is needed to make sure this is the string sequence. Therefore the zero digit in front is not truncated. In computer science and data science, the first three letters of the code is used to denote the object type (CST=customer, DPT=department, ENT=business entity, STO=store, EMP=employee).

	RowCount	code7	code16
1	1000	c9388359	c6313720333964430
2	1001	c4914644	c1000233721293192
3	1002	c1152291	c1541518957449367
4	1003	c8215832	c7101725836281566
5	1004	c8866987	c3936590167660518
6	1005	c7620798	c9032687604667049
7	1006	c4884787	c6700479230563283
8	1007	c6406210	c0644544037359883
9	1008	c3643431	c3165329442185730
10	1009	c6259846	c9984842284321256
11	1010	c6812110	c4736940209459875
12	1011	c1132482	c2512886347463268
13	1012	c1638647	c6677194321915747
14	1013	c9445205	c7153774474834859
15	1014	c3605673	c9406265686157977
16	1015	c7166617	c5835150590935120

1.3. Fair coins and fair dices

Random columns can be used to generate categorical columns. We will show how to generate a coin column (binomial categorical random variable with head and tail). A fair coin is the coin with a 50% chance of observing head another 50% chance of observing tail. The gender column, which is also binomial categorical variable with male and female value. And a fair dice column with values from 1 to 6 of the same chance of occurrence.

U1	coin	gender	dice
tonumber(left(regex_replace(MD5_ASCII([RowCount] + "/U1") , "\D+", ""), 7))/pow(10, 7)	if [U1]<0.5 then "head" else "tail" endif	if [U1]<0.4 then "female" else "male" endif	1 + FLOOR([U1]*6)

The following table shows the simulation results with 4000 rows. We can see in the statistics, around 2000 rows with head value (2017 to be exact) and around 2000 rows with tail value (1984 to be exact). You can read the similar numbers with gender. However, around 0.4*4000 rows or 1623 with female value and around (1-0.4)*4000 or 2378 rows with male values. We see the similar counts with different dice values, which shows that the dice is fair.

	RowCount	U1	coin1	gender	dice
1	1000	0.938836	tail	male	6
2	1001	0.491464	head	male	3
3	1002	0.115229	head	female	1
4	1003	0.821583	tail	male	5
5	1004	0.886699	tail	male	6
6	1005	0.76208	tail	male	5
7	1006	0.488479	head	male	3
8	1007	0.640621	tail	male	4
9	1008	0.364343	head	female	3
10	1009	0.625985	tail	male	4
11	1010	0.681211	tail	male	5
12	1011	0.113248	head	female	1
13	1012	0.163865	head	female	1
14	1013	0.94452	tail	male	6
15	1014	0.360567	head	female	3
16	1015	0.716662	tail	male	5
17	1016	0.509278	tail	male	4
18	1017	0.687471	tail	male	5
19	1018	0.2198	head	female	2

	coin1	Count
1	head	2017
2	tail	1984

	gender	Count
1	female	1623
2	male	2378

	dice	Count
1	1	675
2	2	697
3	3	645
4	4	638
5	5	683
6	6	663

1.4. Test of uniformity and test of independence

If you are skeptical about the uniformity and independence of the hashing method, you can perform the following experiment to generate two random variables (U1 and U2) and then derive the two categorical tests (test1 and test2), which are used for chi-square tests

U1	U2	test1	test2
tonumber(left(regex_replace(MD5_ASCII([RowCount] + "/U1") , "\D+", ""), 7))/pow(10, 7)	tonumber(left(regex_replace(MD5_ASCII([RowCount] + "/U2") , "\D+", ""), 7))/pow(10, 7)	if [U1]<0.5 then "a" else "b" endif	if [U2]<1/4 then "A" elseif [U2]<2/4 then "B" elseif [U2]<3/4 then "C" else "D" endif

We generate 4000 data rows and we have the following data tables as well as statistics for you to perform the chi-square test of homogeneity against the uniform distributions and the chi-square test of independence.

	RowCount	U1	U2	test1	test2
1	1	0.47936	0.339729	a	B
2	2	0.861055	0.216482	b	A
3	3	0.095011	0.26548	a	B
4	4	0.358994	0.194343	a	A
5	5	0.280123	0.47222	a	B
6	6	0.857436	0.303017	b	B
7	7	0.526594	0.60448	b	C
8	8	0.083605	0.268198	a	B
9	9	0.405345	0.095951	a	A
10	10	0.010469	0.187243	a	A
11	11	0.403969	0.256845	a	B
12	12	0.917978	0.537921	b	C
13	13	0.619612	0.019781	b	A
14	14	0.70821	0.192973	b	A
15	15	0.280615	0.67938	a	C
16	16	0.179196	0.049192	a	A
17	17	0.67357	0.13783	b	A
18	18	0.814985	0.740068	b	C

	test1	Count
1	a	2013
2	b	1987

	test2	Count
1	A	985
2	B	1013
3	C	1048
4	D	954

	test1	test2	Count
1	a	A	509
2	a	B	543
3	a	C	520
4	a	D	441
5	b	A	476
6	b	B	470
7	b	C	528
8	b	D	513

1.5. Summary

Hashing method is used to generate core values of business simulations instead of the seeding method. This method is more flexible and it gives us more degrees of freedom to control random sequences.

Reference codes, categorical columns, fair coin, fair dice can be generated / simulated with the uniform values by if-then rules or floor() function.