Published in Towards Data Science

Ahmed Yahya Khaled Follow

Oct 23, 2020 · 8 min read · Listen

Data Transformation in R

With the tidyverse package

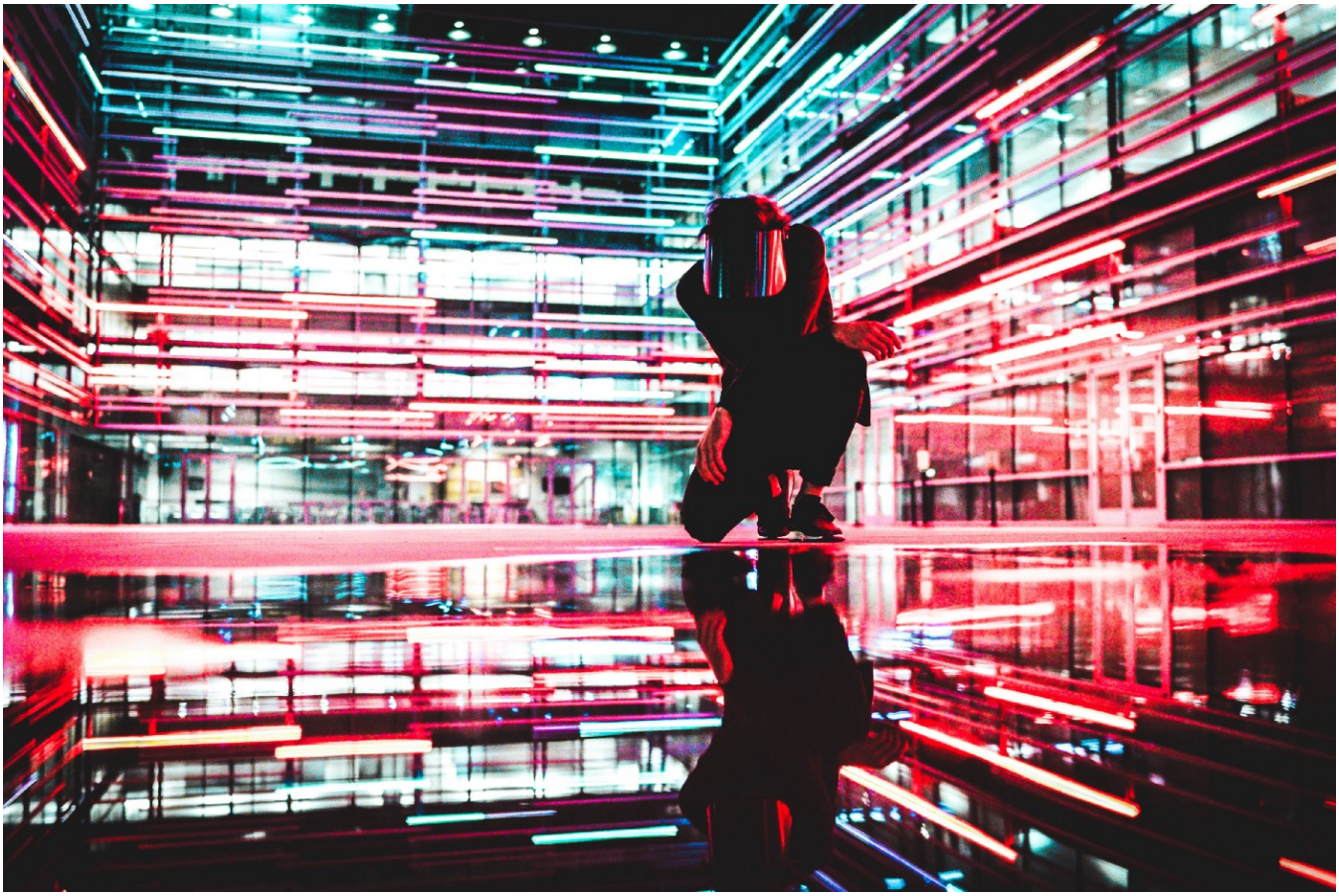

Photo by Drew Graham on Unsplash

Data Transformation is one of the key aspects of working for business data analysis, data science or even for the pre-work of artificial intelligence. In this exercise we will see how to transform data in R. R is an open-sourced programming language for statistical computing and machine learning supported by the R Foundation for

> *The **tidyverse** is an opinionated collection of R packages designed for data science. All packages share an underlying design philosophy, grammar, and data structures.*

Functions from **dplyr** & **tidyr** packages of **tidyverse** mostly do the work of data transformation.

Let's install and load the package first

```
install.packages("tidyverse")
library(tidyverse)
```

Note : you will need to install the package only once but need to load the package every time you start your environment.

These are the functions that we will work on in this article —

- **arrange()** : to order the observations
- **select()** : to select variables or columns
- **filter()** : to filter observations by their values
- **gather()** : to shift observations from columns to rows
- **spread()** : to shift variables from rows to columns
- **group_by() & summarize()** : to summarize data into groups
- **mutate()** : to create new variables from existing variables

arrange()

Arrange rows by variables

Photo by Efe Kurnaz on Unsplash

This function arranges the observations in order. It takes a column name or a set of column names as arguments. For a single column name, it sorts the column's data with other columns following that column and for multiple column names, each additional column breaks ties in the values of preceding columns.

To show this we will load the **mpg** dataset. it has fuel economy data from 1999 to 2008 for 38 popular car models.

```
?mpg
data(mpg)
View(mpg)
```

	manufacturer	model	displ	year	cyl	trans	drv	cty	hwy	fl	class
1	audi	a4	1.8	1999	4	auto(l5)	f	18	29	p	compact
2	audi	a4	1.8	1999	4	manual(m5)	f	21	29	p	compact
3	audi	a4	2.0	2008	4	manual(m6)	f	20	31	p	compact
4	audi	a4	2.0	2008	4	auto(av)	f	21	30	p	compact
5	audi	a4	2.8	1999	6	auto(l5)	f	16	26	p	compact
6	audi	a4	2.8	1999	6	manual(m5)	f	18	26	p	compact
7	audi	a4	3.1	2008	6	auto(av)	f	18	27	p	compact
8	audi	a4 quattro	1.8	1999	4	manual(m5)	4	18	26	p	compact
9	audi	a4 quattro	1.8	1999	4	auto(l5)	4	16	25	p	compact
10	audi	a4 quattro	2.0	2008	4	manual(m6)	4	20	28	p	compact
11	audi	a4 quattro	2.0	2008	4	auto(s6)	4	19	27	p	compact
12	audi	a4 quattro	2.8	1999	6	auto(l5)	4	15	25	p	compact
13	audi	a4 quattro	2.8	1999	6	manual(m5)	4	17	25	p	compact
14	audi	a4 quattro	3.1	2008	6	auto(s6)	4	17	25	p	compact
15	audi	a4 quattro	3.1	2008	6	manual(m6)	4	15	25	p	compact

Image by Author

First, we will order the observations as per the **displ** column. Values of this column will be arranged in ascending order by default and other columns will follow the order of the **displ** column.

```
mpg_arrange1 <- mpg %>% arrange(displ)
View(mpg_arrange1)
```

	manufacturer	model	displ	year	cyl	trans	drv	cty	hwy	fl	class
1	honda	civic	1.6	1999	4	manual(m5)	f	28	33	r	subcompact
2	honda	civic	1.6	1999	4	auto(l4)	f	24	32	r	subcompact
3	honda	civic	1.6	1999	4	manual(m5)	f	25	32	r	subcompact
4	honda	civic	1.6	1999	4	manual(m5)	f	23	29	p	subcompact
5	honda	civic	1.6	1999	4	auto(l4)	f	24	32	r	subcompact
6	audi	a4	1.8	1999	4	auto(l5)	f	18	29	p	compact
7	audi	a4	1.8	1999	4	manual(m5)	f	21	29	p	compact
8	audi	a4 quattro	1.8	1999	4	manual(m5)	4	18	26	p	compact
9	audi	a4 quattro	1.8	1999	4	auto(l5)	4	16	25	p	compact
10	honda	civic	1.8	2008	4	manual(m5)	f	26	34	r	subcompact
11	honda	civic	1.8	2008	4	auto(l5)	f	25	36	r	subcompact

Image by Author

Then we will add more two column names — **cty** & **hwy** in the arguments.

```
mpg_arrange2 <- mpg %>% arrange(displ, cty, hwy)
View(mpg_arrange2)
```

	manufacturer	model	displ	year	cyl	trans	drv	cty	hwy	fl	class
1	honda	civic	1.6	1999	4	manual(m5)	f	23	29	p	subcompact
2	honda	civic	1.6	1999	4	auto(l4)	f	24	32	r	subcompact
3	honda	civic	1.6	1999	4	auto(l4)	f	24	32	r	subcompact
4	honda	civic	1.6	1999	4	manual(m5)	f	25	32	r	subcompact
5	honda	civic	1.6	1999	4	manual(m5)	f	28	33	r	subcompact
6	audi	a4 quattro	1.8	1999	4	auto(l5)	4	16	25	p	compact
7	audi	a4 quattro	1.8	1999	4	manual(m5)	4	18	26	p	compact
8	audi	a4	1.8	1999	4	auto(l5)	f	18	29	p	compact
9	volkswagen	passat	1.8	1999	4	auto(l5)	f	18	29	p	midsize
10	audi	a4	1.8	1999	4	manual(m5)	f	21	29	p	compact
11	volkswagen	passat	1.8	1999	4	manual(m5)	f	21	29	p	midsize
12	toyota	corolla	1.8	1999	4	auto(l3)	f	24	30	r	compact
13	toyota	corolla	1.8	1999	4	auto(l4)	f	24	33	r	compact
14	honda	civic	1.8	2008	4	auto(l5)	f	24	36	c	subcompact
15	honda	civic	1.8	2008	4	auto(l5)	f	25	36	r	subcompact

Image by Author

We can see the second(**cty**) and the third(**hwy**) columns are breaking the ties in the values of the first(**displ**) and second(**cty**) columns respectively.

To order the observations in descending order, the **desc()** function will be used

```
mpg_arrange3 <- mpg %>% arrange(desc(displ))
View(mpg_arrange3)
```

	manufacturer	model	displ	year	cyl	trans	drv	cty	hwy	fl	class
1	chevrolet	corvette	7.0	2008	8	manual(m6)	r	15	24	p	2seater
2	chevrolet	k1500 tahoe 4wd	6.5	1999	8	auto(l4)	4	14	17	d	suv
3	chevrolet	corvette	6.2	2008	8	manual(m6)	r	16	26	p	2seater
4	chevrolet	corvette	6.2	2008	8	auto(s6)	r	15	25	p	2seater
5	jeep	grand cherokee 4wd	6.1	2008	8	auto(l5)	4	11	14	p	suv
6	chevrolet	c1500 suburban 2wd	6.0	2008	8	auto(l4)	r	12	17	r	suv
7	dodge	durango 4wd	5.9	1999	8	auto(l4)	4	11	15	r	suv
8	dodge	ram 1500 pickup 4wd	5.9	1999	8	auto(l4)	4	11	15	r	pickup
9	chevrolet	c1500 suburban 2wd	5.7	1999	8	auto(l4)	r	13	17	r	suv
10	chevrolet	corvette	5.7	1999	8	manual(m6)	r	16	26	p	2seater
11	chevrolet	corvette	5.7	1999	8	auto(l4)	r	15	23	p	2seater
12	chevrolet	k1500 tahoe 4wd	5.7	1999	8	auto(l4)	4	11	15	r	suv
13	dodge	durango 4wd	5.7	2008	8	auto(l5)	4	13	18	r	suv
14	dodge	ram 1500 pickup 4wd	5.7	2008	8	auto(l5)	4	13	17	r	pickup
15	jeep	grand cherokee 4wd	5.7	2008	8	auto(l5)	4	13	18	r	suv

Image by Author

Note : if there were any missing values then these would always be sorted at the end

select()

Select variables by name


Photo by Alexander Schimmeck on Unsplash

We will use the same **mpg** dataset here

```
?mpg
data(mpg)
View(mpg)
```

	manufacturer	model	displ	year	cyl	trans	drv	cty	hwy	fl	class
1	audi	a4	1.8	1999	4	auto(l5)	f	18	29	p	compact
2	audi	a4	1.8	1999	4	manual(m5)	f	21	29	p	compact
3	audi	a4	2.0	2008	4	manual(m6)	f	20	31	p	compact
4	audi	a4	2.0	2008	4	auto(av)	f	21	30	p	compact
5	audi	a4	2.8	1999	6	auto(l5)	f	16	26	p	compact
6	audi	a4	2.8	1999	6	manual(m5)	f	18	26	p	compact
7	audi	a4	3.1	2008	6	auto(av)	f	18	27	p	compact
8	audi	a4 quattro	1.8	1999	4	manual(m5)	4	18	26	p	compact
9	audi	a4 quattro	1.8	1999	4	auto(l5)	4	16	25	p	compact
10	audi	a4 quattro	2.0	2008	4	manual(m6)	4	20	28	p	compact
11	audi	a4 quattro	2.0	2008	4	auto(s6)	4	19	27	p	compact
12	audi	a4 quattro	2.8	1999	6	auto(l5)	4	15	25	p	compact
13	audi	a4 quattro	2.8	1999	6	manual(m5)	4	17	25	p	compact
14	audi	a4 quattro	3.1	2008	6	auto(s6)	4	17	25	p	compact
15	audi	a4 quattro	3.1	2008	6	manual(m6)	4	15	25	p	compact

Image by Author

Now we will select three columns — **displ**, **cty** & **hwy** to a new data frame

```
mpg_select1 <- mpg %>% select(displ, cty, hwy)
View(mpg_select1)
```

	displ	cty	hwy
1	1.8	18	29
2	1.8	21	29
3	2.0	20	31
4	2.0	21	30
5	2.8	16	26
6	2.8	18	26
7	3.1	18	27
8	1.8	18	26
9	1.8	16	25
10	2.0	20	28
11	2.0	19	27
12	2.8	15	25

If we want to select all columns from **displ *to* hwy** , we can write below

```
mpg_select2 <- mpg %>% select(displ : hwy)
View(mpg_select2)
```

	displ	year	cyl	trans	drv	cty	hwy
1	1.8	1999	4	auto(l5)	f	18	29
2	1.8	1999	4	manual(m5)	f	21	29
3	2.0	2008	4	manual(m6)	f	20	31
4	2.0	2008	4	auto(av)	f	21	30
5	2.8	1999	6	auto(l5)	f	16	26
6	2.8	1999	6	manual(m5)	f	18	26
7	3.1	2008	6	auto(av)	f	18	27
8	1.8	1999	4	manual(m5)	4	18	26
9	1.8	1999	4	auto(l5)	4	16	25
10	2.0	2008	4	manual(m6)	4	20	28
11	2.0	2008	4	auto(s6)	4	19	27
12	2.8	1999	6	auto(l5)	4	15	25
13	2.8	1999	6	manual(m5)	4	17	25
14	3.1	2008	6	auto(s6)	4	17	25
15	3.1	2008	6	manual(m6)	4	15	25

Image by Author from RStudio

filter()

Filter observations by conditions

Photo by Bill Oxford on Unsplash

For this we will use the **diamonds** dataset and see how to select observations based on conditions.

```
?diamonds
data(diamonds)
View(diamonds)
```

	carat	cut	color	clarity	depth	table	price	x	y	z
1	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
2	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
3	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
4	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
5	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75
6	0.24	Very Good	J	VVS2	62.8	57.0	336	3.94	3.96	2.48
7	0.24	Very Good	I	VVS1	62.3	57.0	336	3.95	3.98	2.47
8	0.26	Very Good	H	SI1	61.9	55.0	337	4.07	4.11	2.53
9	0.22	Fair	E	VS2	65.1	61.0	337	3.87	3.78	2.49
10	0.23	Very Good	H	VS1	59.4	61.0	338	4.00	4.05	2.39
11	0.30	Good	J	SI1	64.0	55.0	339	4.25	4.28	2.73
12	0.23	Ideal	J	VS1	62.8	56.0	340	3.93	3.90	2.46
13	0.22	Premium	F	SI1	60.4	61.0	342	3.88	3.84	2.33
14	0.31	Ideal	J	SI2	62.2	54.0	344	4.35	4.37	2.71
15	0.20	Premium	E	SI2	60.2	62.0	345	3.79	3.75	2.27

Image by Author from RStudio

We can check summary of the columns values

```
> summary(diamonds$carat)
Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
0.2000  0.4000  0.7000  0.7979  1.0400  5.0100

> summary(diamonds$price)
Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 326     950    2401    3933    5324   18823

> table(diamonds$cut)
     Fair      Good Very Good   Premium     Ideal
     1610      4906     12082     13791     21551
```


Let's filter only the ***Ideal*** **cut** diamonds

```
diamonds_filter1 <- diamonds %>% filter(cut == 'Ideal')
View(diamonds_filter1)
```

	carat	cut	color	clarity	depth	table	price	x	y	z
1	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
2	0.23	Ideal	J	VS1	62.8	56.0	340	3.93	3.90	2.46
3	0.31	Ideal	J	SI2	62.2	54.0	344	4.35	4.37	2.71
4	0.30	Ideal	I	SI2	62.0	54.0	348	4.31	4.34	2.68
5	0.33	Ideal	I	SI2	61.8	55.0	403	4.49	4.51	2.78
6	0.33	Ideal	I	SI2	61.2	56.0	403	4.49	4.50	2.75
7	0.33	Ideal	J	SI1	61.1	56.0	403	4.49	4.55	2.76
8	0.23	Ideal	G	VS1	61.9	54.0	404	3.93	3.95	2.44
9	0.32	Ideal	I	SI1	60.9	55.0	404	4.45	4.48	2.72
10	0.30	Ideal	I	SI2	61.0	59.0	405	4.30	4.33	2.63
11	0.35	Ideal	I	VS1	60.9	57.0	552	4.54	4.59	2.78
12	0.30	Ideal	D	SI1	62.5	57.0	552	4.29	4.32	2.69
13	0.30	Ideal	D	SI1	62.1	56.0	552	4.30	4.33	2.68
14	0.28	Ideal	G	VVS2	61.4	56.0	553	4.19	4.22	2.58
15	0.32	Ideal	I	VVS1	62.0	55.3	553	4.39	4.42	2.73

Image by Author

Now let's filter ***Ideal*** & ***Premium*** **cut** diamonds

```
diamonds_filter2 <- diamonds %>%
  filter(cut == c('Ideal', 'Premium'))
View(diamonds_filter2)
```

	carat	cut	color	clarity	depth	table	price	x	y	z
1	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
2	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
3	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	0.32	Premium	E	I1	60.9	58.0	345	4.38	4.42	2.68
5	0.30	Ideal	I	SI2	62.0	54.0	348	4.31	4.34	2.68
6	0.33	Ideal	I	SI2	61.2	56.0	403	4.49	4.50	2.75
7	0.29	Premium	F	SI1	62.4	58.0	403	4.24	4.26	2.65
8	0.32	Ideal	I	SI1	60.9	55.0	404	4.45	4.48	2.72
9	0.22	Premium	E	VS2	61.6	58.0	404	3.93	3.89	2.41
10	0.35	Ideal	I	VS1	60.9	57.0	552	4.54	4.59	2.78
11	0.30	Premium	D	SI1	62.6	59.0	552	4.23	4.27	2.66

Image by Author

This is how we will filter diamonds which **price** is ***greater than 2500***

```
diamonds_filter3 <- diamonds %>% filter(price > 2500)
View(diamonds_filter3)
```

	carat	cut	color	clarity	depth	table	price	x	y	z
1	0.70	Ideal	E	SI1	62.5	57.0	2757	5.70	5.72	3.57
2	0.86	Fair	E	SI2	55.1	69.0	2757	6.45	6.33	3.52
3	0.70	Ideal	G	VS2	61.6	56.0	2757	5.70	5.67	3.50
4	0.71	Very Good	E	VS2	62.4	57.0	2759	5.68	5.73	3.56
5	0.78	Very Good	G	SI2	63.8	56.0	2759	5.81	5.85	3.72
6	0.70	Good	E	VS2	57.5	58.0	2759	5.85	5.90	3.38
7	0.70	Good	F	VS1	59.4	62.0	2759	5.71	5.76	3.40
8	0.96	Fair	F	SI2	66.3	62.0	2759	6.27	5.95	4.07
9	0.73	Very Good	E	SI1	61.6	59.0	2760	5.77	5.78	3.56
10	0.80	Premium	H	SI1	61.5	58.0	2760	5.97	5.93	3.66
11	0.75	Very Good	D	SI1	63.2	56.0	2760	5.80	5.75	3.65
12	0.75	Premium	E	SI1	59.9	54.0	2760	6.00	5.96	3.58
13	0.74	Ideal	G	SI1	61.6	55.0	2760	5.80	5.85	3.59
14	0.75	Premium	G	VS2	61.7	58.0	2760	5.85	5.79	3.59
15	0.80	Ideal	I	VS1	62.9	56.0	2760	5.94	5.87	3.72

Image by Author

Now we will apply filter in three columns : ***Ideal* cut *0.54* carat** diamonds with **price *1266***

```
diamonds_filter4 <- diamonds %>%
  filter(cut == 'Ideal', carat == 0.54, price == 1266)
View(diamonds_filter4)
```

	carat	cut	color	clarity	depth	table	price	x	y	z

Let's see a more complex filtering criteria — ***Ideal* cut**, ***between 0.4 and(&) 1.0* carat** with **price *less than 580 or(|) greater than 10000***. Appropriate logical operators will be used to express these conditions.

```
diamonds_filter5 <- diamonds %>%
  filter(cut == 'Ideal' ,
         carat >= 0.4 & carat <= 1.0 ,
         price < 580 | price > 10000)
View(diamonds_filter5)
```

	carat	cut	color	clarity	depth	table	price	x	y	z
1	0.41	Ideal	I	SI1	61.7	55	561	4.77	4.80	2.95
2	0.40	Ideal	G	SI2	62.6	54	573	4.73	4.76	2.97
3	0.40	Ideal	G	SI2	62.4	56	573	4.71	4.75	2.95
4	0.40	Ideal	G	SI2	62.1	53	573	4.75	4.78	2.96
5	0.40	Ideal	I	SI1	61.8	55	573	4.74	4.78	2.94
6	1.00	Ideal	F	VVS1	62.3	53	10058	6.37	6.43	3.99
7	1.00	Ideal	F	VVS2	61.3	53	10577	6.44	6.48	3.96
8	1.00	Ideal	D	VVS1	60.7	56	14498	6.47	6.54	3.95
9	1.00	Ideal	D	IF	60.7	57	16469	6.44	6.48	3.92
10	0.40	Ideal	I	SI2	61.0	58	484	4.74	4.77	2.90
11	0.40	Ideal	H	SI2	61.6	56	516	4.73	4.75	2.92
12	0.40	Ideal	H	SI2	62.6	55	540	4.70	4.72	2.95
13	0.40	Ideal	H	SI2	62.2	55	540	4.75	4.77	2.96
14	0.40	Ideal	J	SI1	62.2	57	552	4.71	4.74	2.94

Image by Author

The ***not(!)*** operator needs to be used very carefully. We will need to keep in mind the *De Morgan's law* while using it. This law states that ***!(x & y)*** is the same as ***!x | !y***, and ***!(x | y)*** is the same as ***!x & !y***.

```
diamonds_filter6 <- diamonds %>%
  filter(cut != 'Ideal',
         !(carat >= 0.4 & carat <= 1.0) ,
         !(price < 580 | price > 10000))
View(diamonds_filter6)
```


	carat	cut	color	clarity	depth	table	price	x	y	z
1	1.17	Very Good	J	I1	60.2	61	2774	6.83	6.90	4.13
2	1.01	Premium	F	I1	61.8	60	2781	6.39	6.36	3.94
3	1.01	Fair	E	I1	64.5	58	2788	6.29	6.21	4.03
4	1.01	Premium	H	SI2	62.7	59	2788	6.31	6.22	3.93
5	1.05	Very Good	J	SI2	63.2	56	2789	6.49	6.45	4.09
6	1.05	Fair	J	SI2	65.8	59	2789	6.41	6.27	4.18
7	1.01	Fair	E	SI2	67.4	60	2797	6.19	6.05	4.13
8	1.04	Premium	G	I1	62.2	58	2801	6.46	6.41	4.00
9	1.20	Fair	F	I1	64.6	56	2809	6.73	6.66	4.33
10	1.02	Premium	G	I1	60.3	58	2815	6.55	6.50	3.94
11	1.17	Premium	J	I1	60.2	61	2825	6.90	6.83	4.13
12	1.01	Premium	H	SI2	61.6	61	2828	6.39	6.31	3.91
13	1.01	Good	I	I1	63.1	57	2844	6.35	6.39	4.02
14	1.27	Premium	H	SI2	59.3	61	2845	7.12	7.05	4.20
15	1.01	Fair	H	SI2	65.4	59	2846	6.30	6.26	4.11

Image by Author

gather()

Gather columns into key-value pairs

Photo by Phil Coffman on Unsplash

Sometimes we can have a dataset where the observations are found in column names that need to be gathered under a variable with a new column name. Let's build the dataset first.

```
Country_Name <- c('Qatar', 'United States', 'Germany',
                  'Canada', 'United Kingdom')
Y2009 <- c(59094, 47099, 41732, 40773, 38454)
Y2010 <- c(67403, 48466, 41785, 47450, 39080)
Y2011 <- c(82409, 49883, 46810, 52101, 41652)

gdp <- data.frame(Country_Name, Y2009, Y2010, Y2011)
View(gdp)
```

		2	3	4
	Country_Name	Y2009	Y2010	Y2011
1	Qatar	59094	67403	82409

Image By Author

This dataset can be considered as **GDP** data of five countries for 2009, 2010 & 2011. It is clearly seen that Y2009, Y2010 & Y2011 these column names are themselves observations and should be under a single variable or column name- '**Year**'. We will do it by the **gather()** function.

```
gdp_gather <- gdp %>% gather("Year", "GDP" , 2:4)
View(gdp_gather)
```

	Country_Name	Year	GDP
1	Qatar	Y2009	59094
2	United States	Y2009	47099
3	Germany	Y2009	41732
4	Canada	Y2009	40773
5	United Kingdom	Y2009	38454
6	Qatar	Y2010	67403
7	United States	Y2010	48466
8	Germany	Y2010	41785
9	Canada	Y2010	47450
10	United Kingdom	Y2010	39080
11	Qatar	Y2011	82409
12	United States	Y2011	49883
13	Germany	Y2011	46810
14	Canada	Y2011	52101
15	United Kingdom	Y2011	41652

Image by Author

To make this dataset ready for numerical analysis we need to remove the character "*Y*"(or any character) from the **Year** values and convert it from *character* to *integer*.

```
gdp_gather$Year <- gsub("[a-zA-Z]", "", gdp_gather$Year)
gdp_gather$Year <- as.integer(gdp_gather$Year)
View(gdp_gather)
```

	Country_Name	Year	GDP
1	Qatar	2009	59094
2	United States	2009	47099
3	Germany	2009	41732
4	Canada	2009	40773
5	United Kingdom	2009	38454
6	Qatar	2010	67403
7	United States	2010	48466
8	Germany	2010	41785
9	Canada	2010	47450
10	United Kingdom	2010	39080
11	Qatar	2011	82409
12	United States	2011	49883
13	Germany	2011	46810
14	Canada	2011	52101
15	United Kingdom	2011	41652

Image by Author

```
> glimpse(gdp_gather)
Observations: 15
Variables: 3
$ Country_Name <fct> Qatar, United States, Germany, Canada, Un...
$ Year         <int> 2009, 2009, 2009, 2009, 2009, 2010, 2010,...
$ GDP          <dbl> 59094, 47099, 41732, 40773, 38454, 67403,...
```

Image by Author

spread()

Spread a key-value pair across multiple columns

Photo by Zachary Nelson on Unsplash

Sometimes we can see variables are distributed in observations in a dataset. In this case we need to spread it to column names. Lets build a dataset with key-values.

```
Student <- c('Jack', 'Jack', 'Jack', 'Jack', 'Jack', 'Jack',
             'Rose', 'Rose', 'Rose', 'Rose', 'Rose', 'Rose')
Subject <- c('English', 'Biology', 'Chemistry', 'Maths', 'Physics',
             'Social Science', 'English', 'Biology', 'Chemistry',
             'Maths', 'Physics', 'Social Science')
Marks <- c(80, 70, 87, 75, 90, 82, 65, 70, 88, 92, 79, 93)

reportCard <- data.frame(Student, Subject, Marks)
View(reportCard)
```

	Student	Subject	Marks
1	Jack	English	80
2	Jack	Biology	70
3	Jack	Chemistry	87
4	Jack	Maths	75
5	Jack	Physics	90
6	Jack	Social Science	82
7	Rose	English	65
8	Rose	Biology	70
9	Rose	Chemistry	88
10	Rose	Maths	92
11	Rose	Physics	79
12	Rose	Social Science	93

Image by Author

In this report card dataset, if we consider the **Subject** names as variables then it need to spread out to the column names. We will do it by the **spread()** function.

```
reportCard_spread1 <- reportCard %>% spread(Subject, Marks)
View(reportCard_spread1)
```

	Student	Biology	Chemistry	English	Maths	Physics	Social Science
1	Jack	70	87	80	75	90	82
2	Rose	70	88	65	92	79	93

Image by Author

If we consider **Student** names as variables -

```
reportCard_spread2 <- reportCard %>% spread(Student, Marks)
View(reportCard_spread2)
```