Introduction

As data scientists, we love to do our job efficiently without reinventing the wheel. Tips-and-tricks articles provide snippets of code for common tasks in the data science world. In this article, we'll cover mainly Python and R, as well as other tips in Unix, Excel, Git, Docker, Google Spreadsheets, etc. Here, we will gather 10 tips and trips from our Tips Section. Stay tuned!

Python

1. How to sort a list of tuples by element

Let's say I have the following list:

```
l = [(1,2), (4,6), (5,1), (1,0)]
l
```

```
[(1, 2), (4, 6), (5, 1), (1, 0)]
```

And I want to sort it by the **second** element of the tuple:

```
sorted(l, key=lambda t: t[1])
```

```
[(1, 0), (5, 1), (1, 2), (4, 6)]
```

2. How to flatten a list of lists

Assume that our list is:

```
l = [[1, 2, 3], [4, 5, 6], [7], [8, 9]]
```

We want to flatten it into one list. We can use list comprehensions, as follows:

```
[item for sublist in l for item in sublist]
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

3. 'elif' in list comprehension

Scenario: You're dealing with product reviews that take values from `1` to `5`, and you want to create three categories:

- **Good,** if the review is greater or equal to `4`
- **Neutral**, if the review is a `3`
- **Negative**, if the review is less than `3`

```
x = [1,2,3,4,5,4,3,2,1]
["Good" if i>=4 else "Neutral" if i==3 else "Bad" for i in x]
```

```
['Bad', 'Bad', 'Neutral', 'Good', 'Good', 'Good', 'Neutral', 'Bad',
'Bad']
```

4. A shebang line: #!/usr/bin/python3

In many `.py` files, we may see the shebang line at the top of the script. Its purpose is to define the location of the interpreter. By adding the line `#!/usr/bin/python3` at the top of the script, we can run the `file.py` on a Unix system, and it'll automatically understand that this is a Python script. Alternatively, you could run the script as `python3 file.py`. For example, assume the `file.py` is:

```
#!/usr/bin/python3
print("Hello shebang line")
```

And we can run on Unix as:

```
$ ./file.py
```

R

6. Joining with 'dplyr' on multiple columns

`dplyr` allows us to join two data frames on more than a single column. All you have to do is to add the columns within the `by` like `by = c("x1" = "x2", "y1" = "y2")`. For example:

```
library(dplyr)
set.seed(5)
df1 <- tibble(
    x1 = letters[1:10],
    y1 = LETTERS[11:20],
    a = rnorm(10)
)
df2 <- tibble(
    x2 = letters[1:10],
    y2 = LETTERS[11:20],
    b = rnorm(10)
)
df<-df1%>%inner_join(df2, df2, by = c("x1" = "x2", "y1" = "y2"))
df
```

```
# A tibble: 10 x 4
   x1    y1          a      b

 1 a     K     -0.841   1.23
 2 b     L      1.38   -0.802
 3 c     M     -1.26   -1.08
 4 d     N      0.0701 -0.158
 5 e     O      1.71   -1.07
 6 f     P     -0.603  -0.139
 7 g     Q     -0.472  -0.597
 8 h     R     -0.635  -2.18
 9 i     S     -0.286   0.241
10 j     T      0.138  -0.259
```

7. How to store models in R with a for loop

Let’s say that we want to run a different regression model for each `Species` in the `iris` data set. We can do it in two different ways, as follows:

Store the models in a list:

```
my_models<-list()
for (s in unique(iris$Species)) {
    tmp<-iris[iris$Species==s,]
    my_models[[s]]<-lm(Sepal.Length~Sepal.Width+Petal.
Length+Petal.Width, data=tmp)
}
# get the 'setosa' model
my_models[['setosa']]



Call:
lm(formula = Sepal.Length ~ Sepal.Width + Petal.Length + Petal.Width, 
    data = tmp)

Coefficients:
 (Intercept)   Sepal.Width  Petal.Length   Petal.Width  
      2.3519        0.6548        0.2376        0.2521
```

Store the models by name using `assign`:

```
for (s in unique(iris$Species)) {
    tmp<-iris[iris$Species==s,]
    assign(s,lm(Sepal.Length~Sepal.Width+Petal.Length+Petal.Width, 
data=tmp))
}
# get the 'setosa' model
get("setosa")



Call:
lm(formula = Sepal.Length ~ Sepal.Width + Petal.Length + Petal.Width, 
    data = tmp)

Coefficients:
 (Intercept)   Sepal.Width  Petal.Length   Petal.Width  
      2.3519        0.6548        0.2376        0.2521
```

8. How to pass multiple parameters to ‘sapply’

Assume that we want to run an `sapply` or `lapply` in R, and the function has multiple parameters. Then we can define the parameter in which we want to apply the `sapply` and assign fixed values to the rest:

```
# this is the function like a linear equation
# of the form y= a + b * x
my_func<- function(a,b,c) {
  a+b*c
}
# the values of the x
x = c(1,5,10)
# we set a=1 and b=2
sapply(x,my_func,a=1, b=2)
```

```
[1]  3 11 21
```

9. How to get the column of the max value by row

Assume that our DataFrame is:

```
set.seed(5)
df<-as.data.frame(matrix(sample(1:100,12),ncol=3))
df
```

```
  V1 V2 V3
1 66 41 19
2 57 85  3
3 79 94 38
4 75 71 58
```

We can get the index and the name of the max column by row, as follows:

```
colnames(df)[max.col(df,ties.method="random")]
```

```
[1] "V1" "V2" "V2" "V1"
```

10. How to generate random dates

We can generate random dates from a specific range of Unix time stamps using the uniform distribution. For example, let’s generate 10 random dates:

```
library(lubridate)

lubridate::as_datetime( runif(10, 1546290000, 1577739600))
```

```
[1] "2019-12-09 15:45:26 UTC" "2019-08-31 19:28:03 UTC" "2019-01-13
12:15:13 UTC" "2019-11-15 00:13:25 UTC"
[5] "2019-01-19 06:31:10 UTC" "2019-11-02 12:46:34 UTC" "2019-09-04
19:16:31 UTC" "2019-07-29 11:53:43 UTC"
[9] "2019-01-25 23:08:20 UTC" "2019-02-03 02:30:21 UTC"…
```