

## Python

### 1. How To COALESCE In Pandas

This function returns the first non-null value between 2 columns.

```
import pandas as pd  
import numpy as np
```

```
df=pd.DataFrame({"A": [1, 2, np.nan, 4, np.nan], "B": ['A', 'B', 'C', 'D', 'E'] })  
  
df
```

	A	B
0	1.0	A
1	2.0	B
2	NaN	C
3	4.0	D
4	NaN	E

In the following example, it will return the values of column A and if they are null, it will return the corresponding value of column B.

```
df['combined'] = df['A'].combine_first(df['B'])  
df
```

	A	B	combined
0	1.0	A	1
1	2.0	B	2
2	NaN	C	C
3	4.0	D	4
4	NaN	E	E

### 2. How To Disable All Warnings In Python

You can disable all python warnings by running the following code block.

```
import sys
if not sys.warnoptions:
    import warnings
    warnings.simplefilter("ignore")
```

### 3.How to Convert A Pandas DataFrame To XML

There is no Pandas function to convert a dataframe to XML but we will show you how to do it with a simple custom function. This is very useful, especially when working with flask and you want your API to have as output an XML file.

```
#lets create a dataframe
df=pd.DataFrame({'A':[1,2,3,4,5], 'B':['a','b','c','d','e']})
df
```

	A	B
0	1	a
1	2	b
2	3	c
3	4	d
4	5	e

Let's build the `to_xml` function:

```
def to_xml(df):
    def row_xml(row):
        xml = []
        for i, col_name in enumerate(row.index):
            xml.append(' <{0}>{1}'.format(col_name, row.iloc[i]))
        xml.append('')
        return '\n'.join(xml)
    res = '\n'.join(df.apply(row_xml, axis=1))
    return(res)

print(to_xml(df))
```

```
<item>
    <A>1</A>
    <B>a</B>
</item>
<item>
    <A>2</A>
    <B>b</B>
</item>
<item>
    <A>3</A>
    <B>c</B>
</item>
<item>
    <A>4</A>
    <B>d</B>
</item>
<item>
    <A>5</A>
    <B>e</B>
</item>
```

Now, if you want to save it as XML:

```
with open('df.XML', 'w') as f:
    f.write(to_xml(df))
```

#### 4.Replace a List of Strings with another List of Strings

There isn't any common way to replace a list of strings with another list of strings within a text without applying a for loop or multiple regular expression calls. With this quick and easy hack, we can do it in one line of code using Pandas DataFrames.

```
import pandas as pd
df = pd.DataFrame({'text': ['Billy is going to visit Rome in November', 'I was
born in 10/10/2010', 'I will be there at 20:00']})
df
```

text	
0	Billy is going to visit Rome in November
1	I was born in 10/10/2010
2	I will be there at 20:00

```
to_replace=['Billy', 'Rome', 'January|February|March|April|May|June|July|August|September|October|November|December', '\d{2}:\d{2}', '\d{2}/\d{2}/\d{4}']
```

```
replace_with=['name', 'city', 'month', 'time', 'date']
```

```
df['modified'] = df.text.replace(to_replace, replace_with, regex=True)
```

```
df
```

	text	modified
0	Billy is going to visit Rome in November	name is going to visit city in month
1	I was born in 10/10/2010	I was born in date
2	I will be there at 20:00	I will be there at time

Note that the replacements on the text are done in the order they appear in the lists.

## R

### 5. Use of `select_if` | `rename_if` in Tidyverse

The `select_if` function belongs to `dplyr` and is very useful where we want to choose some columns based on some conditions. We can also add a function that applies to column names.

**Example:** Let's say that I want to choose only the numeric variables and to add the prefix "numeric\_" to their column names.

```
library(tidyverse)
```

```
iris %>% select_if(is.numeric, list(~ paste0("numeric_", .))) %>% head()
```

	numeric_Sepal.Length	numeric_Sepal.Width	numeric_Petal.Length	numeric_Petal.Width
1	5.1	3.5	1.4	0.2
2	4.9	3.0	1.4	0.2
3	4.7	3.2	1.3	0.2
4	4.6	3.1	1.5	0.2
5	5.0	3.6	1.4	0.2
6	5.4	3.9	1.7	0.4

Note that we can also use the `rename_if` in the same way. An important note is that the `rename_if()`, `rename_at()`, and `rename_all()` have been superseded by `rename_with()`. The matching `select` statements have been superseded by the combination of a `select()` + `rename_with()`.

These functions were superseded because `mutate_if()` and friends were superseded by `across()`. `select_if()` and `rename_if()` already use tidy selection so they can't be replaced by `across()` and instead we need a new function.

### 6. Use of `where` in Tidyverse

We can `select` or `rename` columns using the `where` by selecting the variables for which a function returns TRUE. We will work with the same examples as above.

**Example:** Let's say that I want to choose only the numeric variables and to add the prefix "numeric\_" to their column names.

```
library(tidyverse)

iris %>% rename_with(~ paste0("numeric_", .), where(is.numeric)) %>%
  select(where(is.numeric)) %>% head()

  numeric_Sepal.Length numeric_Sepal.Width numeric_Petal.Length numeric_Petal.Width
1           5.1            3.5          1.4            0.2
2           4.9            3.0          1.4            0.2
3           4.7            3.2          1.3            0.2
4           4.6            3.1          1.5            0.2
5           5.0            3.6          1.4            0.2
6           5.4            3.9          1.7            0.4
```

## 7.Use of everything in Tidyverse

In many Data Science projects, we want one particular column (usually the dependent variable y) to appear first or last in the dataset. We can achieve this using the `everything()` from `dplyr` package.

**Example:** Let's say that I want the column Species to appear **first** in my dataset.

```
library(tidyverse)

mydataset <- iris %>% select(Species, everything())
mydataset %>% head()

  Species Sepal.Length Sepal.Width Petal.Length Petal.Width
1  setosa      5.1       3.5        1.4       0.2
2  setosa      4.9       3.0        1.4       0.2
3  setosa      4.7       3.2        1.3       0.2
4  setosa      4.6       3.1        1.5       0.2
5  setosa      5.0       3.6        1.4       0.2
6  setosa      5.4       3.9        1.7       0.4
```

## 8.Use of relocate in Tidyverse

The `relocate()` is a new addition in `dplyr 1.0.0`. You can specify exactly where to put the columns with `.before` or `.after`

**Example:** Let's say that I want the Petal.Width column to appear next to Sepal.Width

```
library(tidyverse)

iris %>% relocate(Petal.Width, .after=Sepal.Width) %>% head()

  Sepal.Length Sepal.Width Petal.Width Petal.Length Species
1           5.1       3.5       0.2        1.4  setosa
2           4.9       3.0       0.2        1.4  setosa
3           4.7       3.2       0.2        1.3  setosa
4           4.6       3.1       0.2        1.5  setosa
5           5.0       3.6       0.2        1.4  setosa
6           5.4       3.9       0.4        1.7  setosa
```

Notice that we can also set to appear after the last column.

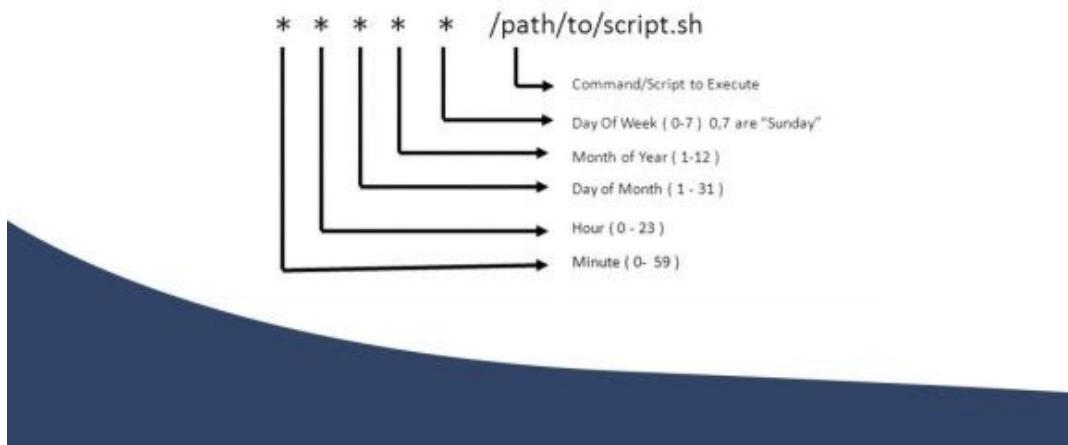
**Example:** Let's say that I want the Petal.Width to be the last column

```
iris %>% relocate(Petal.Width, .after=last_col()) %>% head()

  Sepal.Length Sepal.Width Petal.Length Species Petal.Width
1           5.1       3.5        1.4  setosa       0.2
2           4.9       3.0        1.4  setosa       0.2
3           4.7       3.2        1.3  setosa       0.2
4           4.6       3.1        1.5  setosa       0.2
5           5.0       3.6        1.4  setosa       0.2
6           5.4       3.9        1.7  setosa       0.4
```

# Linux

## 9. How To Schedule A Cron Job In Linux



Many times there is a need to run a script in our server periodically. We can schedule our work using [cron](#). The first thing that we need to do is to go to the **terminal** and to open the crontab by typing the command:

```
crontab -e
```

The first time it will ask you to choose your editor, which can be then **nano**, **vim** etc. Personally, I prefer the **nano** which is the simplest one. Once we open the editor we are ready to schedule our cron job.

There are 5 placeholders which are referred to **minute** (0-59), **hour** (0-23), **day of month** (1-31), **month** (1-12) and **day of week** (0-6) starting from Sunday=0.

Let's say that I have a python script called **mytest.py** and I want to run it **every minute**. Then we have to write within crontab:

```
* * * * * python /path/to/mytest.py
```

Let's say that I want to run it **every ten minutes**. Notice that we can use the "/" to define steps:

```
# it runs the script every 10 minutes
*/10 * * * * python /path/to/mytest.py
```

Let's give some other examples. Look at the comments.

```
# it runs the script every Friday at 3am
0 3 * * 5 python /path/to/mytest.py
```

```
# it runs the script every week
0 0 */7 * * python /path/to/mytest.py
```

```
# it runs the script every 1st and 15th of the month
0 0 1,15 * * python /path/to/mytest.py
```

You can see which cron jobs have been scheduled with the command:

```
crontab -l
```

You can open the text editor and erase the particular line of the corresponding cron job or you can remove all the cron jobs without opening the editor by running:

```
crontab -r
```

If you get confused and you are not sure about what you have scheduled, there is a nice [crontab-generator](#) which can write the command of crontab for you! Then you just need to copy-paste it at the crontab after typing `crontab -e`.

## 10. How to Select Columns

Assume that we are dealing with the following CSV file called **eg1.csv**.

```
ID,Name,Dept,Gender  
1,George,DS,M  
2,Billy,DS,M  
3,Nick,IT,M  
4,George,IT,M  
5,Nikki,HR,F  
6,Claudia,HR,F  
7,Maria,Sales,F  
8,Jimmy,Sales,M  
9,Jane,Marketing,F  
10,George,DS,M
```

If you want to select columns, you can use the command **cut**. It has several options (use **man cut** to explore them), but the most common is something like:

```
cut -f 1-2,4 -d , eg1.csv
```

This means “select columns 1 through 2 and columns 4, using comma as the separator”. **cut** uses **-f** (meaning “fields”) to specify columns and **-d** (meaning “delimiter”) to specify the separator. The above command returns:

```
ID,Name,Gender  
1,George,M  
2,Billy,M  
3,Nick,M  
4,George,M  
5,Nikki,F  
6,Claudia,F  
7,Maria,F  
8,Jimmy,M  
9,Jane,F  
10,George,M
```

In order to **exclude** a column or columns we do the opposite of selecting columns by adding the **--complement**. For instance let's say that we want to exclude the second column.

```
cut --complement -f 2 -d , eg1.csv ...
```