Motivation

In Python we can use this simple one line of code `data.isnull().sum` to list down missing data in a data set. But the problem lies in how this function produces a whole list containing all the feature names whether they have any missing value or not. Which can look very messy in a Notebook when the data set in question is a large one with lots of features in it.

To overcome that in this tutorial we will see how we can write a simple utility function that will calculate missing values for only the features with missing observations and store it in a data frame which can later be used to report or visualize.

Preparation

Tools and Libraries

In this tutorial I will be using RStudio as the IDE. Thus I will use R package Reticulate to run Python codes.

I will be using a *Mini Conda* virtual environment, *curious-joe* in the back-end as the Python environment. To be able to reproduce this tutorial you may want to create your own virtual environment in Conda and use the name of that in the `reticulte::use_condaenv()` function. To learn detail about creating and managing Conda environment you can visit this document.

```
# loading libraries

library(reticulate)
library(dplyr)
library(kableExtra)
library(knitr)

# setting up virtual python environment
reticulate::use_condaenv("curious-joe")
```

Python Function

The function we will see depends on the Python library *Pandas*, a library commonly used for data wrangling and analysis.

The function simple enough to understand as you will read through it. But for people brand new to Python programming here is a breakdown of how the functionality flows inside the function:

1. Creates a list, *colNames* of string values to store column names,
2. Creates a blank data frame *df* with the values from *colNames* as column names,
3. Runs a for loop to iterate over each column of the input data frame and performs following series of tasks:

- Calculates percentage of missing values in a column and saves the output in an object called *p*,
- Calculates total count of missing values in a column and saves the output in an object called *q*,
- Runs a check if *p*, percent of missing value, is larger than zero and if it is populates the

empty data frame *df* with the column name and its corresponding count and percentage of missing values.
- Sorts the *df*, the result data frame on descending order,

4. Returns *df*, the data frame with names and missing count of the features with missing values.

```
# pyhton library
import pandas as pd

# @ countMissing
# Fetches columns from the spefied dataset that contains missing values
# @param dataFrame Name of the dataframe object

def countMissing(dataFrame):
    # colNames = ['colNames', 'missingValue', 'missingValuePerc']
    colNames = ['Featuers', 'Missing_Value', 'Percentage_Missing']
    df = pd.DataFrame(columns = colNames)
    for i in dataFrame.columns:
        p = round((dataFrame[i].isnull().sum()/dataFrame.shape[0]) *
100, 2)
        q = round(dataFrame[i].isnull().sum(), 0)
        if p > 0:
            df.loc[len(df)] = [i, q, p]
    # creating data frame with the missing value columns and values
    df = df.sort_values(['Percentage_Missing'], ascending =
False).reset_index(drop=True)
    return(df)
```

Demo

Data

To demonstrate how the function will work I will use *iris* data set and introduce some *NA* values (missing values in R's language) in the data.

```
# preparing data
data <- iris
data = data %>% mutate(Sepal.Width = ifelse(Sepal.Length >7, NA,
Sepal.Width))
data = data %>% mutate(Sepal.Length = ifelse(Sepal.Length >7, NA,
Sepal.Length))
```

In the code we have removed values for Sepal.Width and Sepal.Length features when Sepal.Length value is larger than 7. Which result in 24 rows with missing values.

Application

The following code chunk applies *countMissing()*, the function that we have just created and prints out the output data frame.

```
# calculating missing value using countMissing()
table = countMissing(r.data)
table
```

```
##        Featuers Missing_Value  Percentage_Missing
## 0  Sepal.Length            12                 8.0
## 1   Sepal.Width            12                 8.0
```

Let's use some R markdown packages to make the output look nicer!

```
knitr::kable(py$table, caption = "Missing Values") %>%
  kable_classic(full_width = F, html_font = "Cambria")
```

Table 1: Missing Values

Featuers	Missing_Value	Percentage_Missing
Sepal.Length	12	8
Sepal.Width	12	8

What Did We Improve?

If you look inside the *countMissing()* function you will see that we are using *isnull().sum()* inside, the same function that we could use to get the missing count. The only reason we created *countMissing()* was to make sure that the missing count is produced in a more presentable and usable way. Though the difference is more obvious when they are run on wider data set, the following code chunk shows how the outputs from these two approaches differ.

```
r.data.isnull().sum()
## Sepal.Length    12
## Sepal.Width     12
## Petal.Length     0
## Petal.Width      0
## Species          0
## dtype: int64
```

VS

```
countMissing(r.data)
##        Featuers Missing_Value  Percentage_Missing
## 0  Sepal.Length            12                 8.0
## 1   Sepal.Width            12                 8.0
```

Or Even Better

```
knitr::kable(py$table, caption = "Missing Values") %>%
  kable_classic(full_width = F, html_font = "Cambria")
```

Table 2: Missing Values

Featuers	Missing_Value	Percentage_Missing
Sepal.Length	12	8
Sepal.Width	12	8