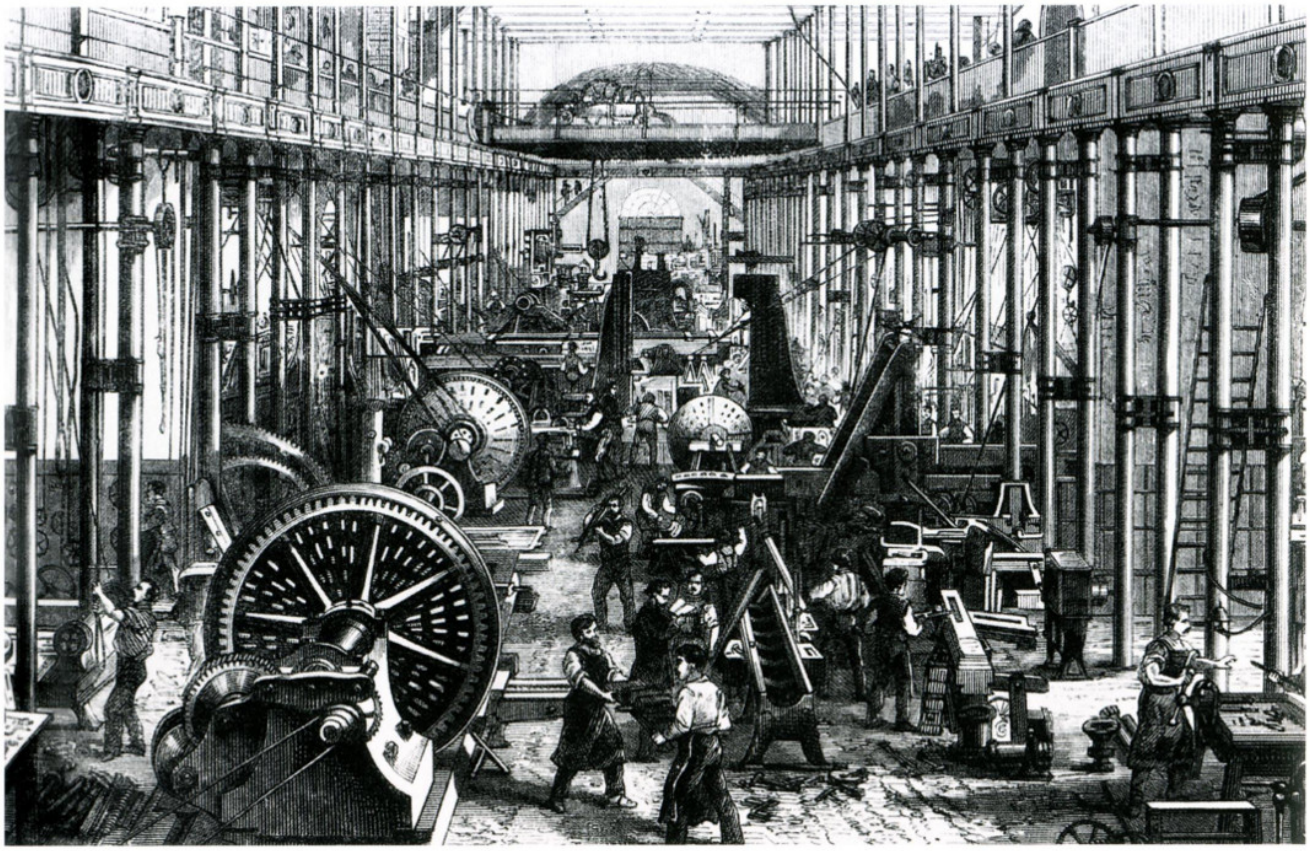
Practical Business Python (https://pbpython.com/)

Taking care of business, one python script at a time

Mon 22 July 2019

Automated Report Generation with Papermill: Part 1 (https://pbpython.com/drafts/papermil-rclone-report-1.html)

Posted by Chris Moffitt (https://pbpython.com/author/chris-moffitt.html) in articles (https://pbpython.com/category/articles.html)



Introduction

This guest post that walks through a great example of using python to automate a report generating process. I think PB Python readers will enjoy learning from this real world example using python, jupyter notebooks, papermill and several other tools.

Before we get started, I would like to introduce the author:

> My name is Duarte Carmo (https://duarteocarmo.com/) and I'm a product manager and digital consultant. Originally from Lisbon - Portugal, but currently living and working in Copenhagen - Denmark. Find more about my work and leisure in my website (https://duarteocarmo.com/).

Part 1 - Tool roundup

Welcome to part 1 of this two-part series post about automating report generation using python, jupyter, papermill, and a couple of other tools.

In the first part, we will cover 4 main important workflows that are part of the automation process. In the second and final part, we will bring everything together and build our own report automation system.

Note: This code was written in python 3.7. You might have to adapt the code for older versions of python.

All of the code for this article is available on github (https://github.com/duarteocarmo/automation-post).

Alright, let's get to work.

Automating report generation with Python - Why?

Not everyone can code. This might seem like an obvious statement, but once you start using python to automate or analyze things around you, you start to encounter a big problem: **reproducibility**. Not everyone knows how to run your scripts, use your tools, or even use a modern browser.

Let us say you built a killer script. How exactly do you make someone who has never heard the word "python" use it? You could teach them python, but that would take a long time.

In this series, we will teach you how you can automatically generate shareable Html reports from any excel file using a combination of tools, centered around python.

Creating a Jupyter Notebook reports from Excel files

Let us say you have an excel file `sales_january.xlsx` with a list of the sales generated by a group of employees. Just like this:

	A	B	C
1	Employee Name	Sales	
2	Paul	€2.000,00	
3	Oscar	€1.500,00	
4	Richard	€4.000,00	
5	Thomas	€500,00	
6	Ursula	€780,00	
7	Gabriela	€1.760,00	
8	Allison	€1.000,00	
9	Louis	€530,00	
10			
11			
12			

Let's start by using a jupyter notebook `sales_january.ipynb` to create a very simple analysis of that sales data.

We start by importing the pandas (https://pandas.pydata.org/) and maplotlib (https://matplotlib.org/) libraries. After that, we specify the name of our file using the `filename` variable. Finally, we use the `read_excel` function to read our data into a pandas DataFrame.

```
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline # so plots are printed automatically

filename = "sales_january.xlsx"
data = pd.read_excel(filename, index_col=0)
```

When printing the `data` dataframe, we get the following:

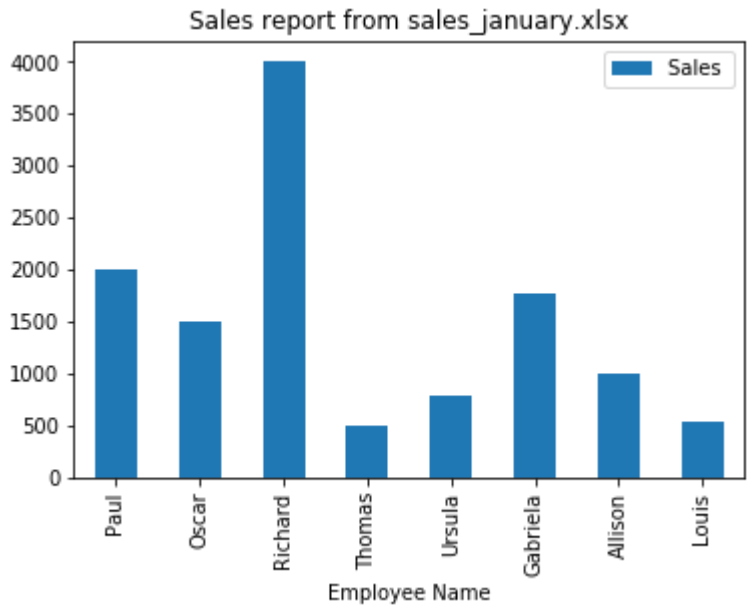
[72]:

Employee Name	Sales
Paul	2000
Oscar	1500
Richard	4000
Thomas	500
Ursula	780
Gabriela	1760
Allison	1000
Louis	530

After that, we plot the data using pandas:

```
data.plot(kind="bar", title=f"Sales report from {filename}")
```

And we get the following:



And that's it! We have a jupyter notebook (https://github.com/duarteocarmo/automation-post/blob/master/src/sales_january.ipynb) that analyzes (a very simple analysis let us say) a sales report in excel. Now let's say we want to share that report with other people in the organization, what do we do?

Generating Html reports from Jupyter Notebooks to share with colleagues

In my experience, the easiest way to share a report with colleagues is to use a little tool called nbconvert (https://nbconvert.readthedocs.io/en/latest/). Nbconvert allows you to generate an Html version of your notebook. To install it simply run `pip install nbconvert` .

To do this, start by navigating to the same directory where your notebook is and run the following from your terminal:

```
$ jupyter nbconvert sales_january.ipynb
```

You will see that a new file named `sales_january.html` was created. Html files are better than `ipynb` in the measure that they are easily shareable via email, message, or any other way. Just make sure the person receiving the file opens it via a relatively modern browser.

But lets us say that this sales report comes in every month, how can we automatically run this notebook with any excel file that has the same format?

Automating report generation using papermill

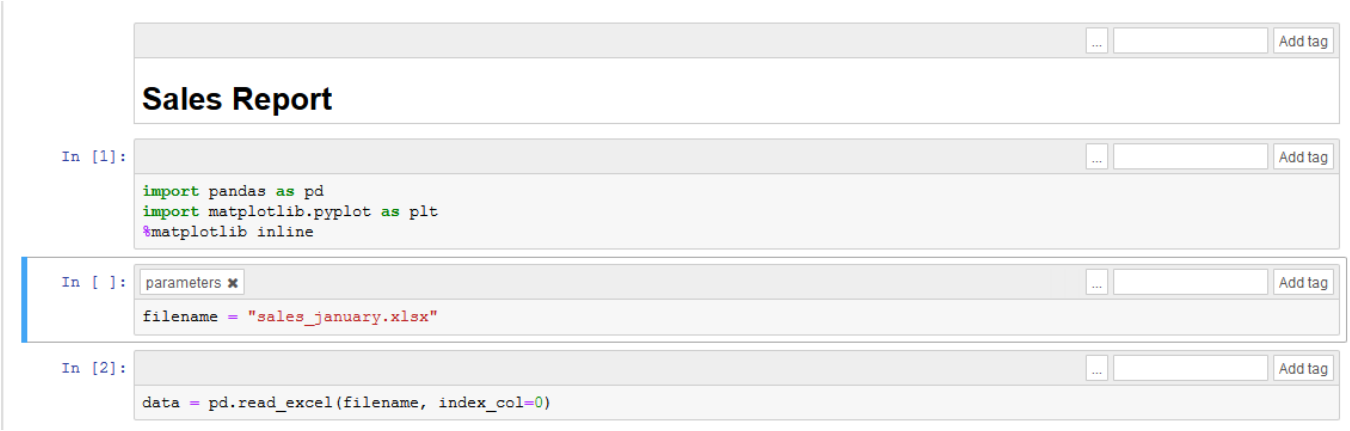
Papermill (https://papermill.readthedocs.io/en/latest/) is a handy tool that allows us to "parameterize and execute" Jupyter Notebooks. This basically means that papermill allows you to execute the same jupyter notebook, with different variables defined outside its context.

To install it, run `pip install papermill` , or follow the more complete installation instructions (https://papermill.readthedocs.io/en/latest/installation.html).

Let us say we want to generate the same report as above, but with another excel file: `sales_february.xlsx` . You should have in your directory, the following:

```
├── sales_february.xlsx
├── sales_january.html
├── sales_january.ipynb
└── sales_january.xlsx
```

The first step is to parameterize our notebook, to do this, let us create a `template.ipynb` file. This notebook is very similar to `sales_january.ipynb` but with a small difference: a new cell with a tag `parameters` . Just like this:



(If you have trouble adding a tag to your notebook, visit this link (https://papermill.readthedocs.io/en/latest/usage-parameterize.html#notebook))

The cell with the `parameters` tag, will allow you to run this notebook from another python script while feeding the `filename` variable, any value you would like.

Your directory should look like this:

```
├── sales_february.xlsx
├── sales_january.html
├── sales_january.ipynb
├── sales_january.xlsx
└── template.ipynb
```

You can always browse the code in the github repo (https://github.com/duarteocarmo/automation-post).

Now that we have everything in place, let's generate a report for a new `february_sales.xlsx` excel file.

To do it, in a new python file, or python console, run the following:

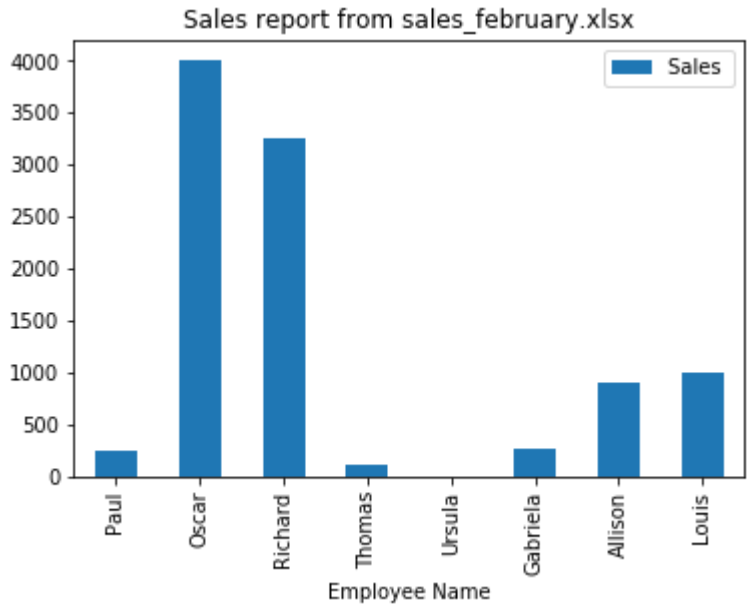
```
import papermill as pm

pm.execute_notebook(
   'template.ipynb',
   'sales_february.ipynb',
   parameters=dict(filename="sales_february.xlsx")
)
```

Let's break this down. The `pm.execute_notebook` function takes 3 arguments. The first, `template.ipynb` is the name of the file what we will use as a base to run our notebook, the one with the `parameters` tag. The second argument is the name of the new notebook that we will generate with the new arguments. Finally, `parameters` is a dictionary of the variables that we want to insert into our template, in this case, the `filename` variable, that will now point to our February sales report.

After running the above code, you will notice a new file in your directory:

```
├── sales_february.ipynb <- This one!
├── sales_february.xlsx
├── sales_january.html
├── sales_january.ipynb
├── sales_january.xlsx
└── template.ipynb
```

Which means, that Papermill has generated a new notebook for us, based on the `sales_february.xlsx` sales report. When openning this notebook, we see a new graph with the new february numbers:



This is pretty handy! We could have a continuous script that always runs this notebook with different sales reports from different months. But how can we automate the process even more? Stay tuned to learn how!

In the second part of this series, you will learn how to bring all of this together to build a full report automation workflow that your colleagues can use! Sign up to the mailing list (https://pbpython.com/pages/mailinglist.html) to make sure you are alerted when the next part comes out!

Tags pandas (https://pbpython.com/tag/pandas.html) jupyter (https://pbpython.com/tag/jupyter.html) excel (https://pbpython.com/tag/excel.html)