Salesforce Data Masking for Sandbox with Python

Why Python?

Python is a very powerful scripting tool, with very fast learning curve and easy to develop on it. There are plenty libraries that provide manipulation for CSV files, connection to SF, generation of random data, etc.

One disadvantage of python is that it needs to be installed in laptop for the code to run and be able to edit and improve solution.

Using Python we can speed up the process of masking data in UAT after refresh. The process is as follows:

1. Extract data from System
2. Manipulate CSV files to create new files that will mask data in target Sandbox (using python)
3. Update data using bulk api

Technically the extract of data and the update can be done using Data Loader. Although using data loader to upload millions of records gives timeout issues.

Getting started with Python and the implementation

Learn Python

I used this very good guide to get started into Python https://www.youtube.com/watch?v=f79MRyMsjrQ

The solution

The solution is divided into three parts: extract data, manipulate data, update data

Extracting CSV from SF

- dataExtractorQueries.py: contains queries to extract data from a SF instance
- dataExtractorSF.py: contains all logic to extract data from SF using queries from the *'dataExtractorQueries.py'* file.

The dataExtractorSF script:

1. creates a Bulk Query job in SF
2. waits for the result CSV to be created in SF
3. downloads CSV file

```
def main():
    isTestMode = False
    query = queryCaseUATRefresh
    fileName = 'case.csv'

    if isTestMode:
        query += ' LIMIT 2000'

    extractDataFromQuery(query, fileName)
```

Sets the query to be used

Sets the filename where data will be stored

Function that extracts data from SF with Bulk API

To set org credentials look for function `extractDataFromQuery` in the *dataExtractorSF.py* file

Manipulating and creating new CSV

- csvTemplates.py: contains the templates for csv manipulation based on data that needs to be masked after full copy sandbox refresh.
- csvGenerator.py: script that processes input file and creates new csv file based on a given template

```
def main():

    fileName = 'case.csv'

    template = sampleTemplate

    createCSV(fileName, template, True)
```

Input file downloaded from SF

Template to create new file

Function that creates new file to be imported

The templates are used to map data into a new csv file, the new CSV will contain all the columns specified in the JSON template object

```
#The templates are used to create a new csv file
#New CSV will contain all the columns specified in the JSON template obj
ect
sampleTemplate = {
    'Id': 'id', #Creates ID column. Id field ALWAYS must be in original
file to be copied to new csv
    'Sample_Word__c': 'firstname', #Creates column with header Sample_Wo
rd__c, fills it with a random word
    'Sample_Blank__c': 'blank',  #Creates column with header Sample_Blan
k__c, fills it with #N/A
    'Sample_Email__c': 'email'   #Creates column with header Sample_Emai
l__c, fills it with random email address
}
```

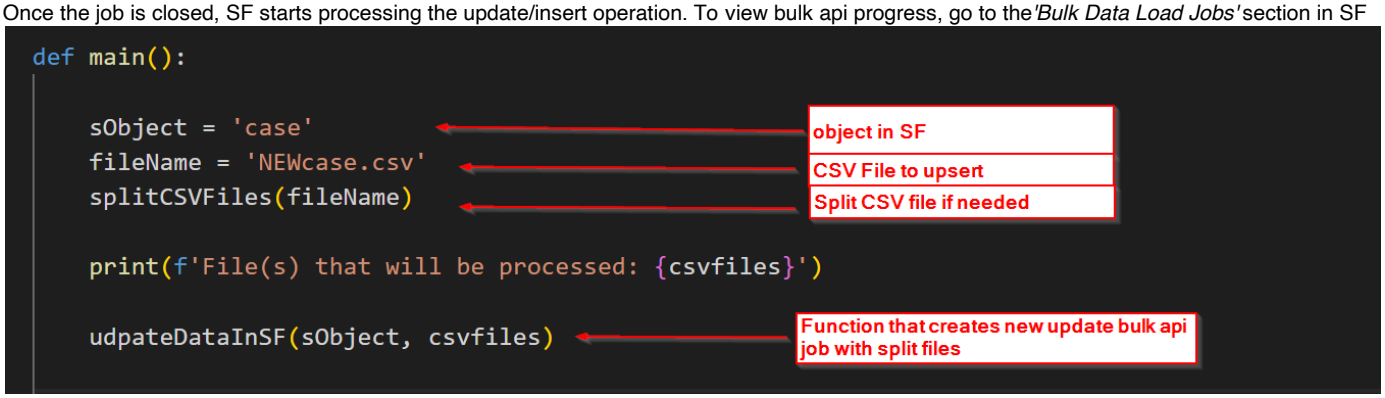
This sample file will generate a new file called NEWcase.csv:

	A	B	C	D	E
1	Id	Sample_Word__c	Sample_Blank__c	Sample_Email__c	
2	5002O000004PLpjQAG	nwmk	#N/A	test.43712650@smartsalary.com.au	
3	5002O000004PMYdQAO	bkae	#N/A	test.06142789@smartsalary.com.au	
4	5002O000004PinGQAS	wjov	#N/A	test.46071982@smartsalary.com.au	
5	5002O000004Q0VbQAK	kzqd	#N/A	test.85479320@smartsalary.com.au	
6	5002O000004Q9CLQA0	kuib	#N/A	test.91436582@smartsalary.com.au	
7	5002O000004QWEzQAO	ngax	#N/A	test.32071465@smartsalary.com.au	
8	5002O000004SC3qQAG	ijhg	#N/A	test.26589703@smartsalary.com.au	
9	5002O000004SC3rQAG	grbt	#N/A	test.65403971@smartsalary.com.au	
10	5002O000004SC3sQAG	ihac	#N/A	test.43029651@smartsalary.com.au	

Upserting new CSV into SF

- dataUploaderSF.py: script that does the following
    1. create a new bulk api 2.0 job
    2. split files (if necessary to comply with SF limits onmax file sizes)
    3. uploads file(s)
    4. closes the job

Once the job is closed, SF starts processing the update/insert operation. To view bulk api progress, go to the *'Bulk Data Load Jobs'* section in SF



To set org credentials look for function `getSFconnection`in the *dataUploaderSF.py* file

Templates Reference guide

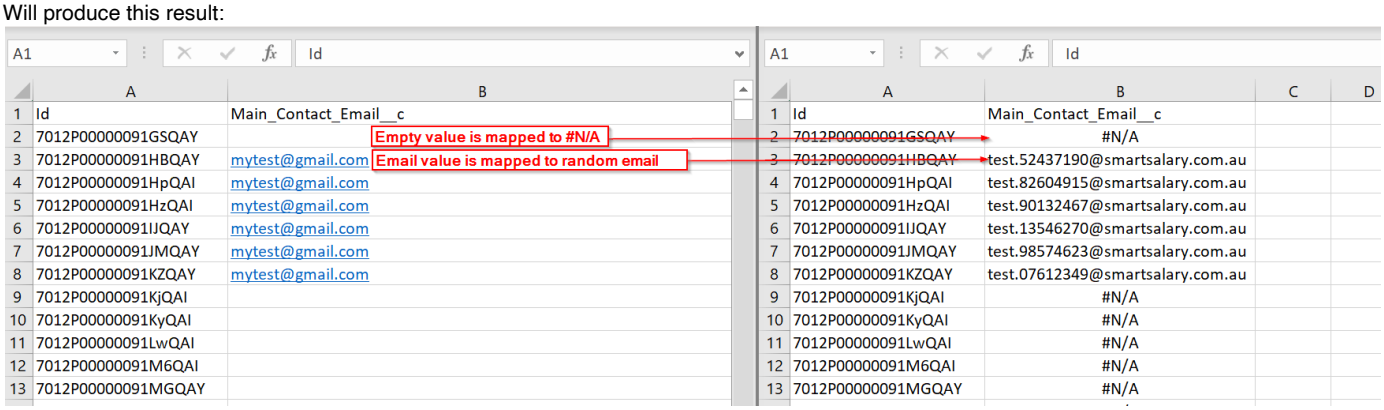
NOTE these template pairings are case sensitive.

- id : The Id of the original record that needs to be updated in SF. This column **MUST** exist in the input CSV file
- blank: #N/A. The bulk api interprets this as a blank/null value
- firstname: random word
- fullname: two random words
- phone: random number of 10 digits
- sentence: a sentence of random words
- emailAccount: creates email addresses for Person Account object. It expects field `AccountNumber__c` to be in the input CSV file. It will generate an email like this: test.<`AccountNumber__c`>@companydomain.com
- email: random email address with following format: test.<`random number`>@companydomain.com
- street: hardcoded to `133 Castlereagh Street`
- city: hardcoded to `Sydney`
- postcode: hardcoded to `2000`
- state: : hardcoded to `NSW`
- dob-ifexists-PersonBirthdate: hardcodes date to `1980-01-01`
- setphrase: hardcoded to `Lorem ipsum dolor sit amet`
- `-ifexists`: when `-ifexists` is added to the mapping, python expects the column to be in the original file. It will check in the original file if there is a value on that particular cell. If there is no value, it will put a blank, if not it will put what the mapping sets it to be.

For example, the following sample template

```
sampleIfExistsTemplate = {
    'Id': 'id',
    'Main_Contact_Email__c': 'email-ifexists'
}
```

Will produce this result:



	A	B
1	Id	Main_Contact_Email__c
2	7012P00000091GSQAY	
3	7012P00000091HBQAY	mytest@gmail.com
4	7012P00000091HpQAI	mytest@gmail.com
5	7012P00000091HzQAI	mytest@gmail.com
6	7012P00000091IJQAY	mytest@gmail.com
7	7012P00000091JMQAY	mytest@gmail.com
8	7012P00000091KZQAY	mytest@gmail.com
9	7012P00000091KjQAI	
10	7012P00000091KyQAI	
11	7012P00000091LwQAI	
12	7012P00000091M6QAI	
13	7012P00000091MGQAY	

	A	B	C	D
1	Id	Main_Contact_Email__c		
2	7012P00000091GSQAY	#N/A		
3	7012P00000091HBQAY	test.52437190@smartsalary.com.au		
4	7012P00000091HpQAI	test.82604915@smartsalary.com.au		
5	7012P00000091HzQAI	test.90132467@smartsalary.com.au		
6	7012P00000091IJQAY	test.13546270@smartsalary.com.au		
7	7012P00000091JMQAY	test.98574623@smartsalary.com.au		
8	7012P00000091KZQAY	test.07612349@smartsalary.com.au		
9	7012P00000091KjQAI	#N/A		
10	7012P00000091KyQAI	#N/A		
11	7012P00000091LwQAI	#N/A		
12	7012P00000091M6QAI	#N/A		
13	7012P00000091MGQAY	#N/A		

Caveats

- Data extracted from SF needs to be deleted from local machines after it has been manipulated, to comply with company's policies
- Manipulating millions of records can take time depending on the mappings used.
  - Cases file takes around 12 minutes to generate as it has a few mapping that generate random data
  - To speed up the process, opt to set hard coded data instead of random data

What's next to do on the python Script

- Perform checks and user entry validations when the data is updated/inserted into Production
-

- Encrypt the password on the local machine
- Create an executable file to avoid having to install all the python libraries
- Create new mappings that have more readable random sentences and readable names
- Create new mapping so that it masks days and months but not years for date fields
- Create new email mapping to add custom domains, at the moment all emails are hardcoded to *'@mycompany.com*
- Adapt the code so files are read and created into subfolders and not on the same level as the python files