Fall 2016 CSE4334/5334 Data Mining

Instructor: Chengkai Li
Department of CSE, University of Texas at Arlington

Programming Assignment 3 (P3)

Due: 11:59pm, December 9th (Friday), 2016

In this assignment, you will write MapReduce programs in Python. You will use Apache Hadoop, an open-source implementation of Google's proprietary MapReduce system. Since we don't have a cluster to use for this course, you will set up a single-node Hadoop environment on your own personal computer. The programs you write will work in a real cluster. It is just that in the single-node setup you won't be able to observe performance advantage against a centralized system. In fact, you will observe worse execution efficiency, since the overhead of the Hadoop environment cannot pay off in a single-node setup.

Setting up your own Hadoop environment can be non-trivial, even if it has only one node. To avoid the hassle, we will use a virtual machine. The virtualization software we will use is VMware Player. The Hadoop-ready virtual machine we will use is Hortonworks Sandbox. There are other virtualization software and Hadoop virtual machines. Our following discussion is based on the setup of VMware Player + Hortonworks Sandbox.

You need quite large free space on your hard drive. 11GB for a virtual machine file, 24GB for the virtual machine loaded from the file, and at least hundreds of MBs for the files produced while running your programs, as well as the VMware Player. You don't need a top-notch computer, but you need a reasonable one. For instance, it worked fine on one of my personal laptops which was purchased 2 years ago at $1000. However, on another laptop purchased 4 years ago at $500, it was very slow.

In following the instructions in this document, make sure you type in exactly what the instructions say. If something doesn't work as expected, look for typos first. If you copy & paste from this document, it will reduce typos.

1 What to Submit

Submit the following 8 files through Programming Assignment 3 (P3)'s entry in Blackboard.

2itemsets_mapper.py
2itemsets_reducer.py
3itemsets_step1_mapper.py
3itemsets_step1_reducer.py
3itemsets_step2_mapper.py
3itemsets_step2_reducer.py

3itemsets_step3_mapper.py
3itemsets_step3_reducer.py

2 Program and Data Files

In this assignment, we will use the following program and data files. They are provided in a ZIP file that can be downloaded from the Programming Assignment 3 (P3) entry in Blackboard.
* *debates*: This folder has 30 files that contain the transcripts of all general election presidential debates in history. Each file is for one debate.
* *wordcount*: This folder contains mapper.py and reducer.py--- Python files for the word count example.
* *retail*: This folder contains 1 data file, retail.dat. It contains a transaction table.
* *arm_output*: This folder contains the expected output files.

3 Setting up the Environment

I am assuming 64-bit Windows as the host operating system. Mac and Linux should work too.

3.1 Enable BIOS Support for Virtualization

Here is an example of how to do it: http://bit.ly/1ncbAqk

This step is necessary if you use a PC. It appears to be irrelevant to Mac, but I couldn't verify. If you encounter troubles using an Mac, this page might have some useful information for you: http://bit.ly/1BZoaKe.

3.2 Download and Install VMware Workstation Pro 12 from http://www.vmware.com/products/workstation/workstation-evaluation.html

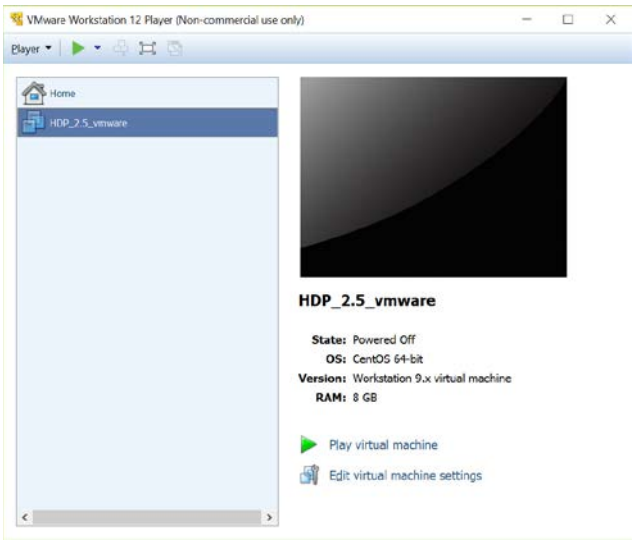
VMware Workstation Pro 12 is what I used and tested for this assignment. Other versions may work as well. For instance, if you have an Mac, it seems that you need to use VMware Fusion http://www.vmware.com/products/fusion/fusion-evaluation.html.

3.3 Download Hortonworks Sandbox HDP 2.5 from http://hortonworks.com/downloads/#sandbox

Under "Hortonworks Sandbox on a VM; HDP® 2.5 on Hortonworks Sandbox", click "Download for VMware".

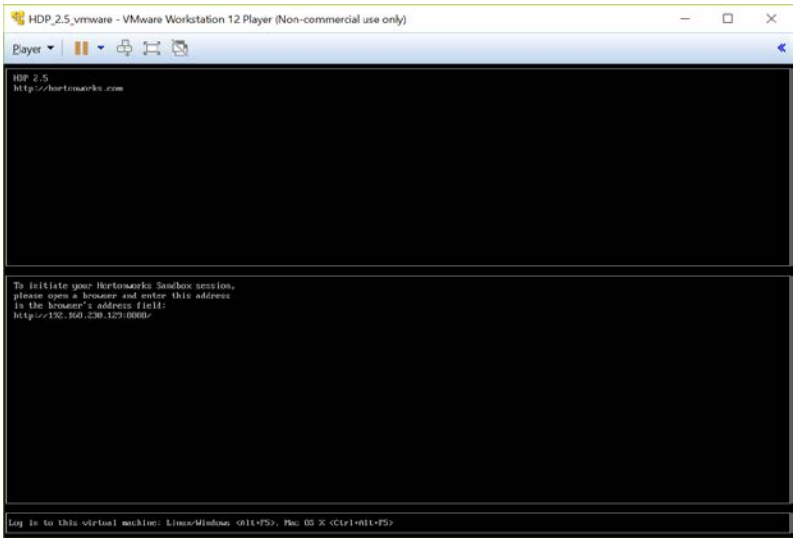
3.4 Install Hortonworks Sandbox HDP 2.5 in VMware

The guide for importing Hortonworks Sandbox on VMware is at http://hortonworks.com/hadoop-tutorial/hortonworks-sandbox-guide/. It is very simple. You don't really need a guide. Just "Open" the downloaded file HDP_2.5_vmware.ova in VMware. After a while, you should be able to see the following screenshot:



You might want to click “Edit virtual machine settings” to choose appropriate amount of memory and processors for the virtual machine.

The virtual machine with Hadoop environment is ready. Just click the Play (Power on) button or click “Play virtual machine” to turn on the guest OS. After a while (maybe 30 minutes), you should see the following screenshot:



As said in the above screen, you can log into the virtual machine by pressing Alt+F5 or Ctrl+Alt+F5 in the virtual machine. But we will remotely log in using SSH, so that you can log in through multiple consoles, which will make programming and debugging easier.

You can turn on and off this virtual machine, as if it is a real computer. And the files will not be lost. If it becomes necessary, you can delete the whole virtual machine, and load the Hortonworks Sandbox again to get a fresh virtual machine and start from scratch.

4 Test Example

To verify the environment is working, we will run the classic word count example.

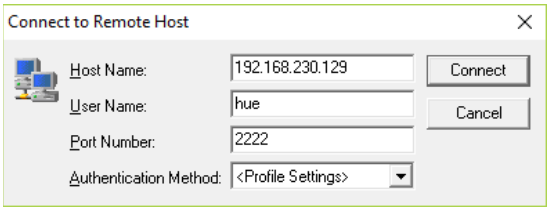
4.1 Transfer data files into the local file system of the guest OS

Use SSH and SFTP tools to upload all files mentioned in Section 2 into the guest OS with the Hadoop environment. Login information as follows:

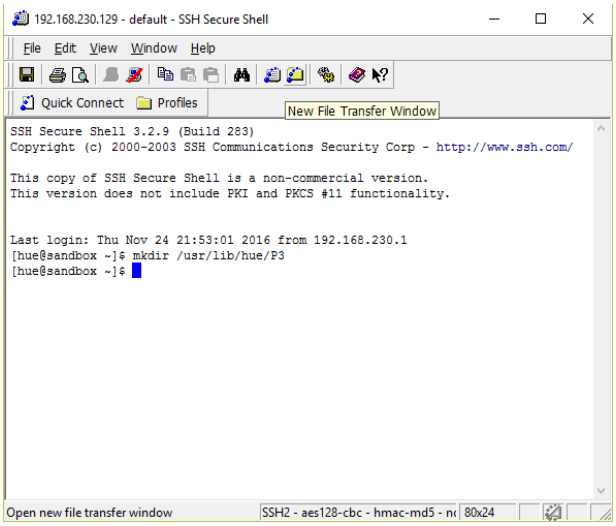
IP address: 192.168.230.129 (your virtual machine may have a different IP. Refer to the screenshot you get, similar to the above one.)
Port: 2222 (Don't use the default 22 in your SSH client. Don't use 8888 either, because 8888 is for the web client (as displayed in the above screenshot), which I will not mention.)
Username: hue (You can also login as "root". But I assume "hue" from now on.)
Password: hadoop

We now explain the detailed steps in Windows, using SSH Secure Shell Client and SSH Secure File Transfer which can be downloaded from http://www.uta.edu/oit/cs/unix/ssh/Secure-Shell-Client.php. There are similar tools on Mac and Unix/Linux. And you can also use commands.

(a) Login to the guest OS using SSH Secure Shell Client. Below is the screenshot of my SSH Secure Shell Client that is about to the make the connection.



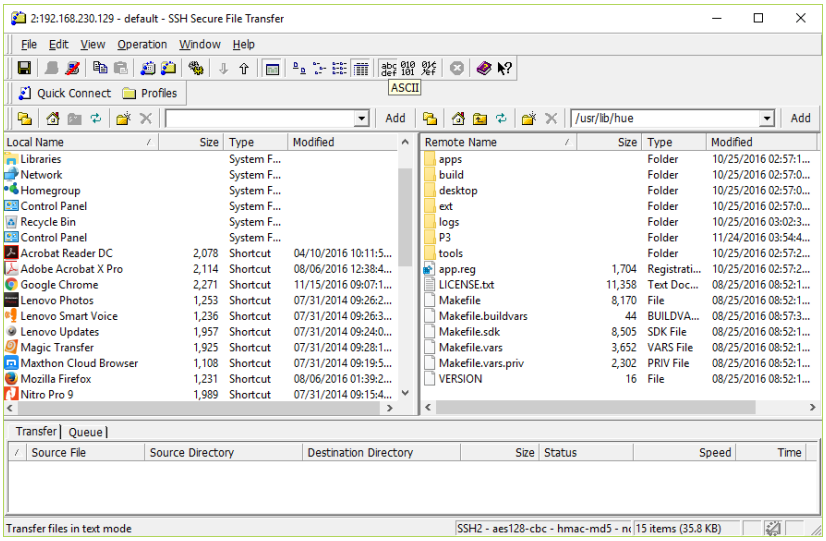
After I press "connect" and enter "hadoop" as the password, I get the following screenshot:



(b) Create a folder /usr/lib/hue/P3 by typing the following command: mkdir /usr/lib/hue/P3

From now on, I assume you are always in folder /usr/lib/hue/P3. If the commands in ensuing discussion do not work, make sure to go to the right folder by command cd /usr/lib/hue/P3.

(c) Open SSH Secure File Transfer.
Notice the "SSH Secure File Transfer" icon in the above screenshot? When you move your mouse over it, you see "New File Transfer Window". Click on it, you will get the following window:



Now you can figure out how to transfer files into the guest OS. From now on, I assume all the folders and files mentioned in Section 2 are uploaded to folder /usr/lib/hue/P3. Make sure to use ASCII mode to transfer all data files and source codes in this assignment. (Click the icon labelled as "ASCII" in the above screenshot, before you transfer the files.)

4.2 Create HDFS data folder and prepare data files

Go back to the SSH Secure Shell Client. Type the following commands:

hdfs dfs -mkdir debates (Create a folder named "debates" in the Hadoop Distributed File System (HDFS). The folder will be used to store input data files.)

hdfs dfs -put /usr/lib/hue/P3/debates/*.txt debates (Copy 30 debate files from the local file system to the created HDFS folder)

4.3 Run an Hadoop program

The guest OS already has a word count example implemented, in a JAR file hadoop-mapreduce-examples.jar. Type the following command:

hadoop jar /usr/hdp/current/hadoop-mapreduce-client/hadoop-mapreduce-examples.jar wordcount debates wordcount_output

The command will run word count on the 30 presidential debate files in HDFS folder "debates". The output will be stored in HDFS folder "wordcount_output". During its execution, you will see

various messages. When it is done, you will see "map 100% reduce 100%" and some statistics about its execution.

Now, let's see what output files were produced in the output folder:

```
hdfs dfs -ls wordcount_output
```

You should see a file named part-r-00000 in the folder. Let's see its content:

```
hdfs dfs -cat wordcount_output/part-r-00000
```

It is a long file. So you might want to see it page by page. Type the following command instead.

```
hdfs dfs -cat wordcount_output/part-r-00000 | more
```

Note: If you want to run the wordcount Hadoop program again, you need to first delete the output file folder "wordcount_output" from HDFS, by the following command. The same applies to our programs you are about to write.

```
hdfs dfs -rm -r -f wordcount_output
```

Even though the Hadoop environment allows you to run multiple Hadoop jobs at the same time, it is unnecessary in this assignment and may freeze your guest and host OS, due to limited resources. In case the virtual machine appears to be very slow, you can use the following commands to list current Hadoop jobs and kill some.

```
mapred job -list
```

```
mapred job –kill <jobID>
```

5 Hadoop Programs in Python

Hadoop was implemented for Java programs. However, it also has a way to support programs in Python, through Hadoop streaming. In this step, we will do Word Count, using Python. The following page explains Hadoop streaming. It can be handy.

https://hadoop.apache.org/docs/r1.2.1/streaming.html

(5.1) By following the instructions in Section 2, you should already have mapper.py and reducer.py in folder `/usr/lib/hue/P3/wordcount`.

(5.2) Execute the following commands:

```
chmod +x /usr/lib/hue/P3/wordcount/*.py
```

This makes sure you can run mapper.py by just command `mapper.py` (without the longer form `python mapper.py`). This is enabled by the shebang in the first line of `mapper.py` : `#!/usr/bin/env python`. The same for `reducer.py`.

Transferring the files in ASCII format from your host OS' file system into the guest OS' file system was crucial. If you didn't use ASCII, Unix line ending of the files might be destroyed, which will make the shebang not working properly.

(5.3) One nice thing about using Python for Hadoop is that we can even test it without a Hadoop environment. Run the following command. It uses Unix pipes to connect mapper.py and reducer.py with standard Linux commands cat and sort.

```
cat debates/*.txt | wordcount/mapper.py | sort -k1,1 | wordcount/reducer.py
```

What do you see in the output? Word count results!

If this doesn't work, it is likely due to shebang not working properly. You can use the following commands instead:

```
cat debates/*.txt | python wordcount/mapper.py | sort -k1,1 | python wordcount/reducer.py
```

(5.4) Now run the Python programs in Hadoop.

Execute the following command to remove the current HDFS folder wordcount_output:

```
hdfs dfs -rm -r -f wordcount_output
```

Then run the Python Hadoop program:

```
hadoop jar /usr/hdp/current/hadoop-mapreduce-client/hadoop-streaming.jar -D stream.map.output.field.separator="\t" -files wordcount/mapper.py,wordcount/reducer.py -input debates -output wordcount_output -mapper mapper.py -reducer reducer.py
```

(Again, make sure your command is identical to the above one. For instance, there shouldn't be any space before or after the comma. Otherwise, it will not work.)

(As the parameters in the command indicate, the input files are in HDFS folder debates and the output files will be placed in HDFS folder wordcount_output.)

Remember to use the following commands to find out the output file name and check its content:

```
hdfs dfs –ls wordcount_output
```

```
hdfs dfs –cat wordcount_output/part-00000
```
(Note that the file name is different from the one produced by the Java wordcount program in Section 3.)

6 Task: Frequent Itemset Mining

You will implement the Apriori algorithm for frequent itemset mining. You will break down the task into several steps. For each step, you will use Python to write a mapper and a reducer that will execute in Hadoop streaming.

The transactions are in file retail.dat inside folder retail. Each line is a transaction, i.e., a set of items purchased together in the transaction. More details about the data can be found at http://fimi.ua.ac.be/data/retail.pdf, if you are interested.

Use 1000 as the support count threshold, i.e., an itemset is frequent if it is contained in at least 1000 transactions. You can hard-code this in your programs.

6.1 Task 1: Find frequent 2-itemsets

Task: We will skip the steps of finding frequent items. Instead, let's directly find frequent 2-itemsets.

What to submit: the mapper and reducer should be in 2 files, named 2itemsets_mapper.py and 2itemsets_reducer.py, respectively.

Output file: Your program's output should be stored in HDFS file arm_output/2itemsets_output/part-00000, when you use Hadoop streaming with Python. When you use Linux shell commands, the output should be stored in local file arm_output/2itemsets_output. The correct output file is provided to you, in arm_output/2itemsets_output_reference.

(1) Composite Keys

For each transaction, the mapper can produce a key-value pair for each pair of items in the transaction, where the key is ***composite*** and has two fields <item1, item2>. Suppose a transaction has three items 'A B C'. Then the corresponding key-value pairs produced by the mapper would be (<A, B>, 1), (<A, C>, 1), (<B, C>, 1). The reducer also uses such composite keys.

More specifically, the mapper can produce a key-value pair by print('%s\t%s\t%s' % (item1, item2, 1)). To make sure <item1, item2> together (instead of item1 only) is treated as the key, you need to tell Hadoop, by running the program using the following command.

```
hadoop jar /usr/hdp/current/hadoop-mapreduce-client/hadoop-streaming.jar -D
stream.map.output.field.separator="\t" -D stream.num.map.output.key.fields=2 -D
mapreduce.partition.keypartitioner.options=-k1,2 -files
arm/2itemsets_mapper.py,arm/2itemsets_reducer.py -input retail/retail.dat -output
arm_output/2itemsets_output -mapper 2itemsets_mapper.py -reducer 2itemsets_reducer.py
```

Compared with the command in Section 5 for running the word count example, there are 2 new options. -D stream.num.map.output.key.fields=2 -D mapreduce.partition.keypartitioner.options=-k1,2 They tell Hadoop to use the first 2 fields of each line from the output of the mapper as the key, in preparing the input to the reducer.

(2) Other options

Also pay attention to the option -D stream.map.output.field.separator="\t" and make sure to use tab to separate the fields, i.e., print('%s\t%s\t%s' % (item1, item2, 1)).

Since we use the option -output arm_output/2itemsets_output, you can find the output in HDFS folder arm_output/2itemsets_output.

(3) Shell command

Before you run the Hadoop program, you can also use Linux shell commands to test your program:

```
cat retail/retail.dat | ./arm/2itemsets_mapper.py | sort -k1,2 | ./arm/2itemsets_reducer.py > arm_output/2itemsets_output
```

Note that you need to use sort -k1,2 instead of sort -k1,1. Also note that arm_output/2itemsets_output is a local file instead of an HDFS file.

6.2 Task 2 Step 1: Find candidate 3-itemsets by joining frequent 2-itemsets

Task: We generate candidate 3-itemsets by joining frequent 2-items. If A B and A C are frequent, a candidate A B C is generated.

What to submit: the mapper and reducer should be in 2 files, named 3itemsets_step1_mapper.py and 3itemsets_step1_reducer.py, respectively.

Output file: Your program's output should be stored in HDFS file arm_output/3itemsets_step1_output/part-00000, when you use Hadoop streaming with Python. When you use Linux shell commands, the output should be stored in local file arm_output/3itemsets_step1_output. The correct output file is provided to you, in file arm_output/3itemsets_step1_output_reference.

Commands: The expected commands are as follows:

```
hadoop jar /usr/hdp/current/hadoop-mapreduce-client/hadoop-streaming.jar -D stream.map.output.field.separator="\t" -files arm/3itemsets_step1_mapper.py,arm/3itemsets_step1_reducer.py -input arm_output/2itemsets_output -output arm_output/3itemsets_step1_output -mapper 3itemsets_step1_mapper.py -reducer 3itemsets_step1_reducer.py
```

```
cat arm_output/2itemsets_output | ./arm/3itemsets_step1_mapper.py | sort -k1,1 | ./arm/3itemsets_step1_reducer.py > arm_output/3itemsets_step1_output
```

6.3 Task 2 Step 2: Prune candidate 3-itemsets

Task: If A B and A C are frequent, a candidate A B C is generated. We further check if B C is frequent. If not, A B C is pruned, without scanning the transaction database. To do this, we need the output from both 6.1 and 6.2. If B C is a frequent 2-itemset in the output of section 6.1, the mapper for 6.3 (this task) will produce a key-value pair using a composite key <B, C> and some special value that can indicate B C is a frequent 2-itesemt. If A B C is generated as candidate 3-

itemset in the output of section 6.2, the mapper for this task will produce a key-value pair using a composite key <B,C> and value A.

What to submit: the mapper and reducer should be in 2 files, named 3itemsets_step2_mapper.py and 3itemsets_step2_reducer.py, respectively.

Output file: Your program's output should be stored in HDFS file arm_output/3itemsets_step2_output/part-00000, when you use Hadoop streaming with Python. When you use Linux shell commands, the output should be stored in local file arm_output/3itemsets_step2_output. The correct output file is provided to you, in file arm_output/3itemsets_step2_output_reference.

Commands: The expected commands are as follows. Note that we are using composite keys again. And we are using two inputs -input arm_output/3itemsets_step1_output -input arm_output/2itemsets_output.

```
hadoop jar /usr/hdp/current/hadoop-mapreduce-client/hadoop-streaming.jar -D
stream.map.output.field.separator="\t" -D stream.num.map.output.key.fields=2 -D
mapreduce.partition.keypartitioner.options=-k1,2 -files
arm/3itemsets_step2_mapper.py,arm/3itemsets_step2_reducer.py -input
arm_output/3itemsets_step1_output -input arm_output/2itemsets_output -output
arm_output/3itemsets_step2_output -mapper 3itemsets_step2_mapper.py -reducer
3itemsets_step2_reducer.py
```

```
cat arm_output/3itemsets_step1_output arm_output/2itemsets_output
| ./arm/3itemsets_step2_mapper.py | sort -k1,2 | ./arm/3itemsets_step2_reducer.py >
arm_output/3itemsets_step2_output
```

6.4 Task 2 Step 3: Report frequent 3-itsesets.

Task: Scan the transaction table to find the support count of remaining candidate 3-itemsets and report frequent 3-itsesets. The mapper will read through the transaction table. For each transaction, it will emit a key-value pair for every candidate 3-itemset contained in the transaction. In order to accomplish it, the mapper needs to have access to the output of 6.3. This can be done by **Distributed Cache.**

What to submit: the mapper and reducer should be in 2 files, named 3itemsets_step3_mapper.py and 3itemsets_step3_reducer.py, respectively.

Output file: Your program's output should be stored in HDFS file arm_output/3itemsets_step3_output/part-00000, when you use Hadoop streaming with Python. When you use Linux shell commands, the output should be stored in local file

arm_output/3itemsets_step3_output. The correct output file is provided to you, in file arm_output/3itemsets_step3_output_reference.

Commands: The expected commands are as follows. Note that we are using 3-field composite keys. The option -cacheFile hdfs:/user/hue/arm_output/3itemsets_step2_output/part-00000#candidates specifies that the HDFS file arm_output/3itemsets_step2_output/part-00000 should be copied to every node so that it can be directly accessed by the mapper using alias candidates. Note that this file will be directly accessed by the mapper. It is not in the -input option, otherwise the mapper will read the file and produce key-value pairs using its content. For more details about Distributed Cache, read https://hadoop.apache.org/docs/r1.2.1/streaming.html. (Note that we are using deprecated option –cacheFile. You can also use -files to specify HDFS file that can be accessed.)

```
hadoop jar /usr/hdp/current/hadoop-mapreduce-client/hadoop-streaming.jar -D
stream.map.output.field.separator="\t" -D stream.num.map.output.key.fields=3 -D
mapreduce.partition.keypartitioner.options=-k1,3 -files
arm/3itemsets_step3_mapper.py,arm/3itemsets_step3_reducer.py -cacheFile
hdfs:/user/hue/arm_output/3itemsets_step2_output/part-00000#candidates -input
retail/retail.dat -output arm_output/3itemsets_step3_output -mapper
3itemsets_step3_mapper.py -reducer 3itemsets_step3_reducer.py
```

```
cp arm_output/3itemsets_step2_output ./candidates
```

```
cat retail/retail.dat | ./arm/3itemsets_step3_mapper.py | sort -k1,3
| ./arm/3itemsets_step3_reducer.py > arm_output/3itemsets_step3_output
```