WA2905 Data Engineering with Python

Student Labs

Web Age Solutions Inc.

Table of Contents

Lab 1 - A/B Testing Data Engineering Tasks Project

A/B testing involves two versions (A and B) of the same entity, e.g two designs of the same web page, two different contents of a service offer e-mail, two different medication treatment options for the same disease, etc. Version A is usually the one that is currently in use (commonly referred to as the control version) and version B is a modified version of the same entity introduced as an improved over A version that is expected to generate more traffic to the web site, increase client response rates, result in better treatment outcomes, etc.

In this small project, you will be required to answer a series of questions related to common data engineering activities that happen when working on a typical A/B testing project on a web site.

Part 1 - The Reference Web Site

The hypothetical web site we will be using as a reference point for you data engineering project in support of A/B testing activities is designed as a common three-tier web application consisting of:

1. The web server

2. The application server

3. The database

Your company's web site sells a number of things and the on-line sales of one of the products – the gadget X – have started falling off. The business wants to introduce an improved version of the product, called Y and you, as a Data Engineer, are requested to conduct an A/B testing to find if Y would sell better.

Your company has existing customers but you also want to reach out to new customers.

Part 2 - The Touch Points

Essentially, you will have to deal with the following technical aspects of the solution:

1. Y's data in the database (technical specs, price, etc)

2. All the aspects related to rendering Y's data on the web site

3. Capturing user interest in the new product

4. Providing information to the business about the outcomes of the A/B testing

Part 3 - The Questions

Now you are invited to answer a number of questions. Basically, the questions are designed to help you get a better idea of what a Data Engineer may be tasked with at work. Your answers may be as generic / detailed as you want them to be. Related questions are grouped together. There are no "ready-to-use" solutions and the answers

may vary significantly based on your background, technology preferences, number of coffees you have had today, etc.

Q1: How would you support the separation of data points related to product X and Y in the database? Would it be a separate database or would you rather have a smaller data footprint in the form of new records in the existing tables ("Specs", "Prices", etc)?

Q2: How do you envision showing the new product Y: side-by-side with the existing product X, or on a separate page? Do you think changing the design of the page may also increase the Y sales? If so, would you have to update accordingly the design of X info page to match?

Q3: How would you drive traffic to the new product page? Would it be by email to select or all of your existing customers? Are you going to broadcast the arrival of the new product through social media networks to reach out to new prospect customers? Are you going to track the sites where user interest in the new product was triggered, and, if so, then how? How are you going to correlate the social media sites with a user coming to view / purchase product Y?

Q4: How are you going to track user activities on your web site?

Q5: How would you gauge the effectiveness of the way product Y is advertised? Would you just capture some average hits using your web / application server log files, or would you expect some dynamics richer in the form of trends, patterns, etc. that you need to be able to capture and communicate to the business?

Lab 2 - Data Availability and Consistency

In this lab, you will work through a practical system design case that involves finding a tradeoff between data availability and consistency, which is critical from the data engineering point of view.

Part 1 - Data Availability and Consistency

Suppose you are tasked with developing a solution where some information (say, price quotes for your company's products) is valid for some time, say, about 6 hours, which may fluctuate a bit. An important business requirement is to comply with the effective date/time. The actual prices are updated within the same period (6 hours, give or take) in the back-end Prices database which your service accesses for prices. The Prices database's update time is the actual effective (valid from) date/time.

A part of the customer-facing page is shown below.

Note: € is the Euro currency symbol.

You need to implement a solution offering your customers the up-to-date "valid from" information without overloading the Prices database.

Select the appropriate solution(s) for this Prices service from the list below. Give you rationale for selecting / rejecting a particular solution. If none is satisfactory, in your opinion, offer your own.

1. You need to build a highly performance service with locally deployed database that gets updated using the Fan-out pattern along with the Prices database.

2. Hey, you just need to deploy a caching service and integrate it with your service. Let

someone else figure out the details.

3. Same as #2 solution, plus an added read-through cache path from the Prices database.

4. You implement solution #3 adding the cache expiry time (TTL) that equals the *offer valid for* time (about 6 hours).

5. You add to solution #3 back-end functionality for synchronizing Prices database updates with Prices service's cache purge. You then comment on solution #4 that it breaks consistency with the actual data, which, in turn, breaks the business rule for the expected published effective date / time. Why would you say so?

6. We just need to lift and shift our IT assets to the cloud and, for sure, there will be a ready-to-use solution we can leverage.

6. Your solution, if any.

One practical solution (there may be other solutions as well) is provided on the next page.

Solution

The practical solution is #5 :

5. You add to solution #3 back-end functionality for synchronizing Prices database updates with Prices service's cache purge ...

Solution explanation

You need a cache which would fulfill the "*overloading the Prices database*" requirement (data will be retrieved from the cache rather than having clients hit the Prices database all the time). Solutions #2 and #3 cover this (caching) part of the requirements.

Solution #4 has a problem as a TTL period for the cache will be set to 6 hours; however, the actual back-end update of the Prices DB might happen earlier, say in 5 hours (e.g. in an attempt to match up competition). So you may end up having stale data in the cache -- that's what the clients will see, not the actual valid from date/time.

Solution #5 addresses the inconsistency between the cache and the actual data in the Prices DB by introducing back-end functionality to purge the cache which is synchronized with the DB updates (it might be an update/insert trigger, or an intermediate service that does a fan-out update: one for the DB, the other for cache (purge)).

The DB gets updated any time the business deems necessary and this update operation triggers the cache purge, so the next client call will find no prices in the cache (it was purged) and your service will have to make a call to the Prices DB, retrieve the data, and cache it (the read-through cache logic); then the data will be returned to the client. The client sees the actual offer with the valid from date/time stamp.

There are a couple of small details you should attend to, though. You need to ensure that the cache purge happens right before the new prices are published (inserted or updated in the Prices DB); also, while the cache is clean (after having been purged), the clients should not be able to access the database that is being updated. All these details are quite easy to deal with.

Lab 3 - Using Jupyter Notebook

Jupyter (note the spelling of the word) is a browser-based development environment that represents an evolution of the original IPython command-line REPL. It is driven by the built-in web server that can be securely accessed remotely as well.

Your Jupyter workspace is a collection of notebooks which are basically dynamic interactive HTML5 web pages containing a mix of your code, embedded visualizations, comments, and other such artifacts. You can export your code from Jupyter as a regular Python file that you can then execute from command line; you can also load external scripts.

You create notebooks with the Jupyter Notebook App, which is a client-server application, that you usually invoke from command line. The web server is based on the Python-based Tornado framework [https://www.tornadoweb.org/en/stable/] that is well suited for long polling, WebSockets, and other applications that require long-lived connections from each client.

Jupyter uses the concept of *kernel* which is, essentially, a specific language run-time. In addition to Python, Jupyter supports kernels for R, Go, C#, and other (https://github.com/jupyter/jupyter/wiki/Jupyter-kernels).

Jupyter development environment can be augmented with additional plugins, like spell checkers.

Like in IPython, you can get immediate help on commands and/or functions using the **?** operator. For example:

```
import numpy as np
np.st*?
```

will print a list of NumPy functions that start with **st**

To get the Docstring help of a function, use the full function name with the **?** next to it, e.g.

```
np.std?
```

Note: Just quickly review the following steps (do **not** execute them for now as we will do it a bit later on) needed to get started with Jupyter.

To start a Jupyter work session you need to launch **Jupyter Notebook App.** Use a command prompt to navigate to the directory on your local file system from which you want to run it and enter this command:

```
# Do not execute this command for now this is just for illustration!
jupyter notebook
```

The above command will go ahead and start the Notebook Dashboard – a Jupyter interactive editor session that gets opened in your web browser.

The Notebook Dashboard is like a file manager and it is mainly used to create new or open existing notebook documents (notebooks) as well as to manage the running kernels.

Each notebook you create in your Jupyter session will be backed up by an Interactive Python Notebook file with extension *.ipynb* saved in the current working directory (from which you started Jupyter).

In this lab, you will learn the basic Jupyter interactive session commands.

For this course we will be using Chrome for the web browser.

Note

If you have a problem while working on a lab in the virtual class delivery environment, let the instructor know the following three pieces of information:

1. The lab number (or the lab title)

2. The lab part (printed in the lab as **Part # - <Part Name>**, e.g. **Part 3 - Read the Input File)**

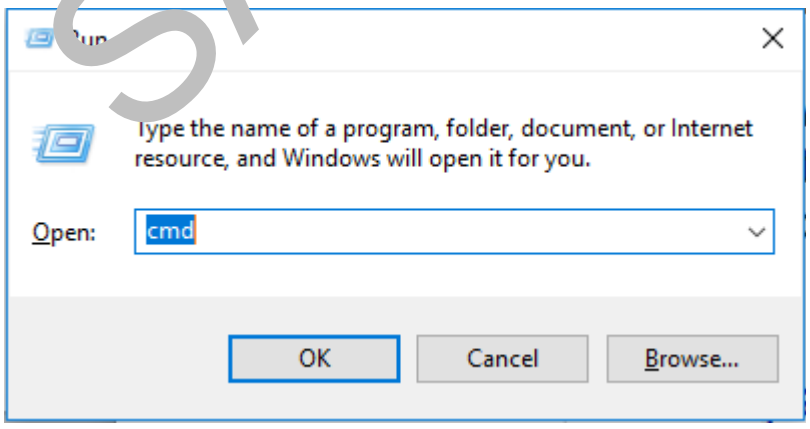
3. The lab step number (lab step numbers are prefixed with two underscores, e.g. **__2**)

Part 1 - Set up the Environment

Note: The instructions listed below apply to Windows ®.

__1. Start a command prompt.

A fast way to start a command prompt in any modern version of Windows ® is to press the **Win** + **R** key combination and then type **cmd** in the open window and press **Enter**



__2. In the command prompt window that opens, create a new working directory **c:\Works** and then change directory to it using this commands:

```
mkdir c:\Works
cd c:\Works
```

> **Note**: All the subsequent labs, unless specified otherwise, will be done in the c:\Works directory.

Part 2 - Check the Versions of Key Modules

__1. Enter the following command in the command prompt window:

```
conda list
```

You should see a listing of installed packages.

__2. Scroll up until you see the seaborn package line, it should be like below or higher:

```
seaborn                   0.9.0                    py37_0
```

> **Note:** Conda is an open source package management system for dependency and environment management for a variety of languages, including Python, R, Ruby, Lua, Scala, Java, JavaScript, C/ C++, FORTRAN [https://conda.io/docs/]. Conda supports the following commands (use them without quotes):
>
> 'clean', 'config', 'create', 'help', 'info', 'install', 'list', 'package', 'remove', 'uninstall', 'search', 'update', 'upgrade'.

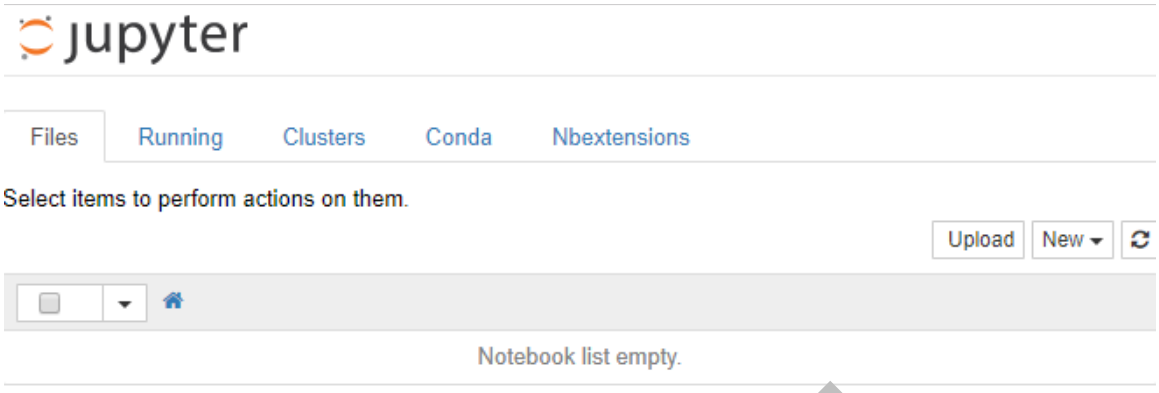
Part 3 - Start and Work in a New Jupyter Notebook

__1. In the command prompt window, enter the following command to launch Jupyter Notebook App:

```
jupyter notebook
```

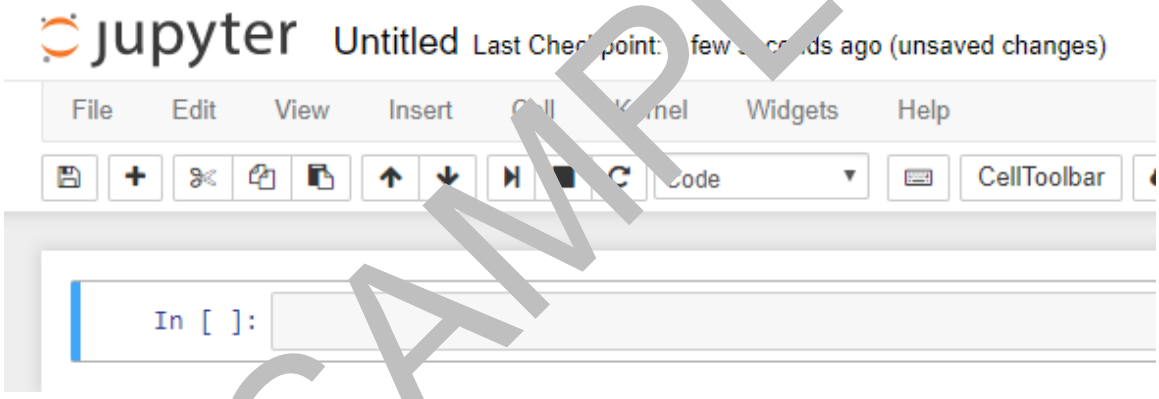
Wait for the Jupyter web server to start.

A new browser window should open displaying the Notebook Dashboard with no notebooks in it (yet!).



__2. In the **New** drop-down, select **Python 3** to create a new notebook:

The new *Untitled* notebook should initialize.



Note: If you have multiple versions of Python installed, you can specify the one you want to use, the lab instructions however, apply to Python 3.

__3. In the current input cell with a blue border on the left, enter the following command:

```
print("Hi Jupyter!")
```

Notice that as you started entering the command, the color of the border changed to green.

A Jupyter notebook has two different keyboard input modes: *Edit mode* and *command mode*.

Edit mode allows you to type code/text into a cell and is indicated by a **green** cell border.

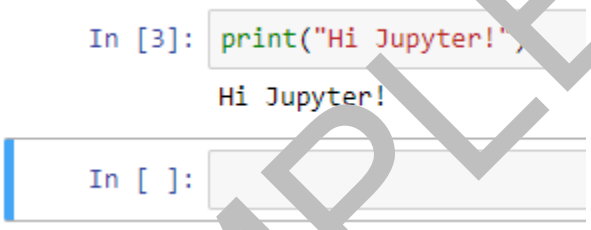
Command mode binds the keyboard to notebook-level actions and is indicated by a grey cell border with the **blue** left margin.

You enable **command mode** by pressing **Esc**.

Edit mode gets enabled when you press **Enter** or click inside the cell.

__4. Press **Shift+Enter**

You should get the *print's* argument, **Hi Jupyter!** printed just below the cell and a new command cell opened below.

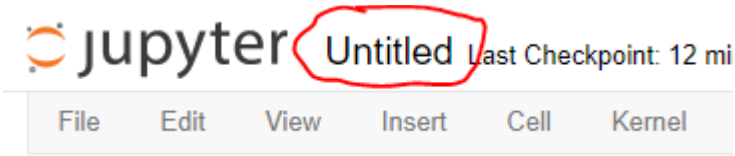


Note: When you read in subsequent lab instructions *enter* or *submit command(s)*, you should enter the provided command(s) in a new cell and use the **Shift+Enter** key combination to submit the command(s) for execution.

To get help on Jupyter shortcuts select the **Help > Keyboard Shortcuts** in the menu bar.

Let's rename the notebook.

__5. Click the **Untitled** label in the top left hand-side corner.



__6. In the **Rename Notebook** dialog that pops up, enter **Getting started with Jupyter**

__7. Press **Rename**.

Spend some time navigating the Jupyter browser-based development environment.

Let's see how you can save the current notebook as a Python file.

__8. In the menu bar, select **File > Download as > Python (.py)**

You will be prompted to confirm the download and save command (the prompt depends on your browser).

__9. Confirm your intended operation (by clicking the **Keep**, or the **OK** button, depending on the browser).

The file named **Getting+started+with+Jupyter.py** should be saved in the current user's *Downloads* directory.

If you open the file in your text editor, you should see the following content (yours may be slightly different):

```
# coding: utf-8

# In[3]:

print("Hi Jupyter!")


# In[ ]:
```

You can view the actual dynamic HTML page that is rendered by the Jupyter web server by selecting the *Download as html* option.

Part 4 - Jupyter Command Commands

Basic edit mode (**EM**) commands:

- Shift+Enter - run code in the current cell and add a new cell below for the next command
- Ctrl+Enter - run code in the current cell and switch to CM; if you have multiple selected cells (you can do it in CM), code in all the selected cells is executed

Basic **CM** commands (press **Esc** to switch from **EM**):

- **a** - add a cell above the current cell
- **b** - add a cell below the current cell
- **c** - copy a cell (Ctrl-v to paste it)
- **d** - delete the current cell

If you need to re-execute commands in your notebook (All, All Above, or All Below) use the **Cell** menu option in the menu bar.

Part 5 - Leveraging IPython's Functionality

In 2014, the original author, Fernando Pérez, announced a spin-off project from IPython called Project Jupyter with IPython acting as Jupyter's processing engine.

Jupyter Notebook App leverages some of the functionality provided by IPython. In addition to OS-level commands, IPython offers a set of predefined 'magic commands' that are prefixed with the **%** character. Those commands work much like OS command-line calls: they get as an argument the rest of the command line (that is passed without parentheses or quotes) and run it.

Note 1: In most cases, you can drop the **%** character.

Note 2: There are also variants of these magic commands which are prefixed with %%.

Those are called *Cell Magics*. These commands treat the first line in the cell as setup code – the setup code gets executed but not timed. It is the rest of the code in the cell that gets timed.

Here is a list of some of the commands that you can try out:

IPython (Jupyter system) Command	What it does
pwd	Print Working Directory
ls	File Listing
history	Command history
cd	Change directory, e.g. **cd "c:\\Works"**, or simply **cd c:\Works**
!	Execute OS command, e.g. **! python -V** **! dir c:\works**
%quickref	Quick Reference Cards (help)
%env	Get, Set, or List environment variables, e.g. **%env PATH**
%timeit EXPRESSION	Average time of executing the EXPRESSION, e.g **import math** **%timeit math.exp(111)**

IPython (Jupyter system) Command	What it does
%debug	Invoke an instance of the Interactive Debugger ipdb after an exception occurs to examine the problem.
%who	Print session variables visible to the Python run-time (*who?* to get more details)

Note: You can get help on IPython's 'magic' functions by running the **%magic** command.

For more information on IPython's functionality (most of which has been migrated to Jupyter Notebooks), visit https://ipython.org/ipython-doc/3/interactive/tutorial.html

Part 6 - Debugging Code with ipdb

When you run the *%debug* magic command, you will be placed at the ipdb's command prompt **ipdb>**

You finish your debugging session by entering q at the command prompt.

You can get help on the supported commands by typing in **?**

The list of the supported commands is listed below.

```
EOF     cl         disable    next    psource   rv         unt
a       clear      display    p       q         s          until
alias   commands   down       pdef    quit      source     up
args    condition  enable     pdoc    r         step       w
b       cont       exit       pfile   restart   tbreak     whatis
break   continue   h          pinfo   return    u          where
bt      d          help       pinfo2  retval    unalias
c       debug      ignore     pp      run       undisplay
```

You can get help on a command by entering ? <command>, e.g. **? n**

You can perform step-by-step debugging of a Python program by running this command:

%run -d path_to_your_python_app.py

Part 7 - Clean Up

__1. In the menu bar, select **File > Close and Halt**

The command will shutdown the notebook's kernel (the sandboxed Python session) and close the interactive edit session.

Keep the browser window open as we are going to use it later.

This is the last step in the lab.

Part 8 - Review

In this lab, you learned the basics of the Jupyter development environment.