AMIS|Conclusion SIG Workshop
Data Wrangling with Jupyter Notebooks

21st February 2019

Data Wrangling is a crucial stage in the data science workflow. Or in any workflow that starts from raw data and hopes to achieve business insight – and perhaps ready to run well trained machine learning models. Data wrangling encompasses various steps and activities -from gathering raw data, exploring, validating, filtering, filling in missing values, joining, enriching, aggregating, shape shifting, unifying. No clear break may be apparent between gathering, preparing, exploring, visualizing and modelling the data. These are iterative steps – with data wrangler moving back and forth in the process.

One of the most used data wrangling workbenches is the Jupyter Notebook – most commonly powered by a Python engine and leveraging the many libraries and frameworks that make the Python ecosystem such an attractive place for data professionals. In this workshop, you will get an introduction to Jupyter Notebook and Python – a platform for Data Analytics (data wrangling, visualization, reporting and machine learning). We will start with getting going with your very own Jupyter Notebook environment, running in a Docker container either locally on your laptop or on a cloud environment. Then you will get acquainted with a simple introduction notebook and subsequently work through a predefined notebook for a somewhat more complicated business case. Finally, you are on your own. We will give some suggestions for next steps – creating your own Notebook and finding answers to business questions on your own.

Of course, the time we have is limited and the scope is potentially vast. So we will barely scratch the surface of both the area of data wrangling as well as the use of Jupyter Notebooks and Python as workbench. However, you will probably get a good inkling of what they have to offer and hopefully get you enthusiastic and inspired to take it from here and move forward.

In this workshop:

- Get access to a Jupyter Notebook environment; you can choose between two options
    - Run Jupyter Notebook in a Docker container on your own laptop
    - Make use of the declarative, cloud based Katacoda playground that does not require any local installation (in fact, only requires you to have a browser and internet access)
- Get acquainted with Jupyter Notebook in an introductory ("Hello World" style) notebook
- Explore the case of the Oracle OpenWorld & CodeOne session catalog using predefined, ready to run notebooks
    - And perhaps tweak and extend these notebooks a little
- On your own: get started on your very own Jupyter Notebook.

Resources

Sources for this workshop: https://github.com/lucasjellema/DataAnalytics--IntroductionDataWrangling-JupyterNotebooks

1. Prepare your Jupyter Notebook environment

As stated before, you can choose between two options. One requires some experience with Docker and a fairly powerful laptop on which you can install software and run virtual machines. The other one requires nothing from you besides a browser – and leads you through a number of declarative, very low code steps. However, the environment you create in this scenario will exist only for a few hours, whereas the environment you install locally is yours for as long as you want.

For the declarative, browser-based environment (leveraging the Katacoda platform) – go to *Appendix C*. Next, continue with Chapter 2.

Local installation of Jupyter Notebook on Docker

The sources for this workshop are in a GitHub Repo: https://github.com/lucasjellema/DataAnalytics--IntroductionDataWrangling-JupyterNotebooks . Among these sources are some scripts we will use for preparing the local Docker container. As a first step, please perform a git clone of this repository to your local file system – from the command line on your laptop in a suitable directory of your choosing.

```
git clone https://github.com/lucasjellema/DataAnalytics--IntroductionDataWrangling-JupyterNotebooks
```

This will download a number of sources to your laptop. We will be using some of them shortly.

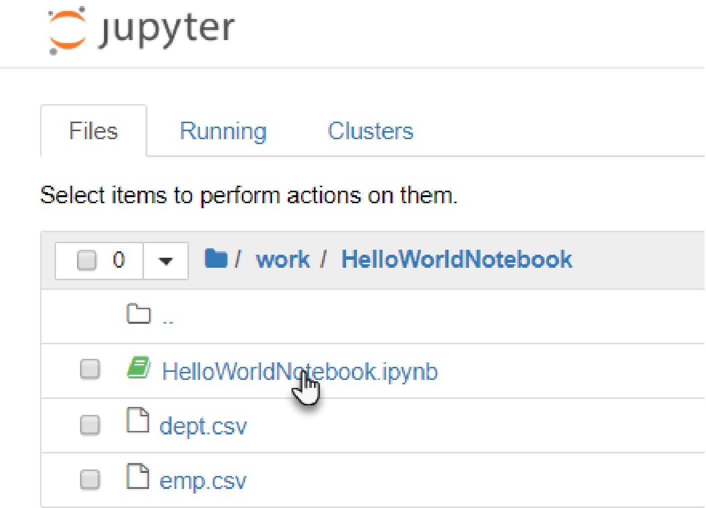
This local scenario comes in at least two flavors – depending on how you want to run Docker containers. We will discern between two situations:

- Use Vagrant and VirtualBox to create a Linux VM that acts as Docker engine – see *Appendix A* for further instructions
- Run Docker containers natively on Windows using Docker for Windows and leveraging Hyper-V (not compatible with Virtual Box) – see *Appendix B* for instructions

If you do not currently use Docker for Windows or Windows Hyper-V, it is recommended to go for the first option – using Virtual Box and Vagrant.

2. Get acquainted with Jupyter Notebook - Hello World notebook

From the Jupyter Notebook client running in your browser, open the file *HelloWorldNotebook.ipynb* notebook, from the */work/HelloWorldNotebook* folder.



Select items to perform actions on them.

/ work / HelloWorldNotebook

..

HelloWorldNotebook.ipynb

dept.csv

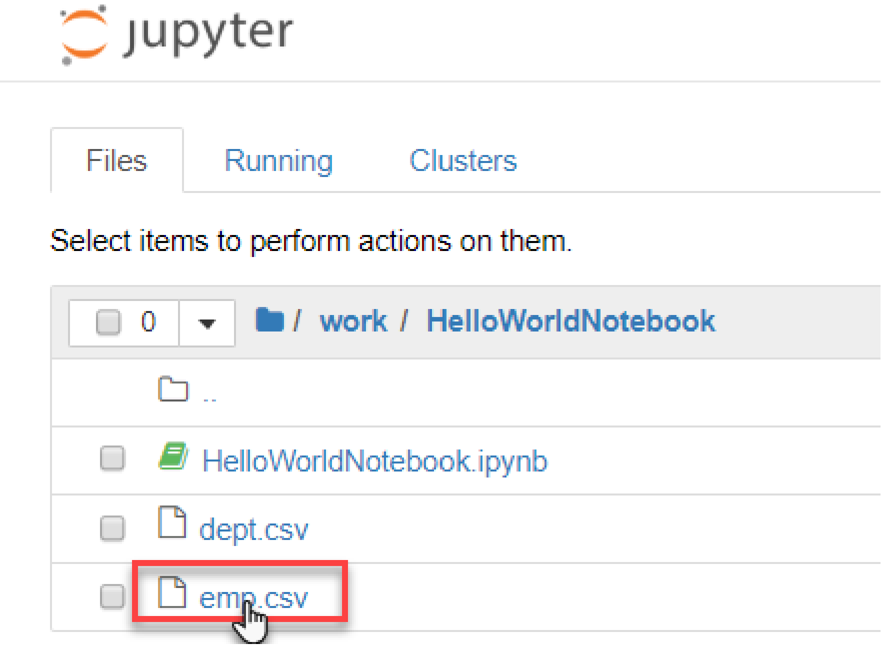
emp.csv

This notebook introduces you first of all to notebooks: how a notebook contains cells of two types (markdown for text and code for Python code) and how each cell can be executed (CTRL+Enter, the Run icon in the toolbar). You can create a new cell by clicking the + icon in the toolbar.

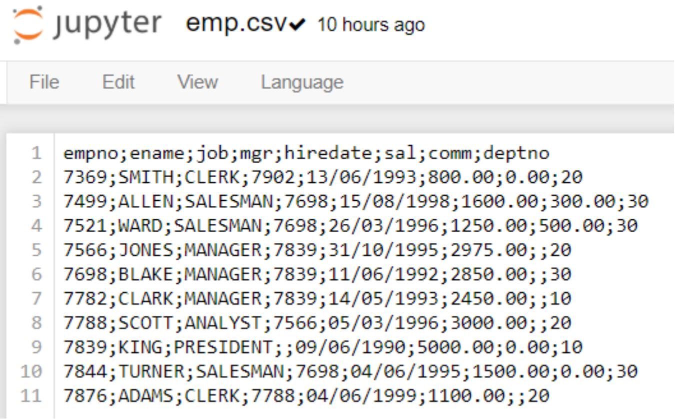
Please follow the instructions in the notebook: create new cells, copy the indicated code to the cell and execute the cell. Feel free to experiment: make changes to the code or simply type your own.

After the first set of cells, you will hit the section *Loading Data into a Pandas Dataframe*. This section is an introduction to the Pandas library (Paneled Data) – a library for data wrangling, data exploration, manipulation, visualization and analysis.

Most of the notebook will work with the employees data set (based on the EMP and DEPT tables in the SCOTT demo schema in early Oracle Database versions). Before you start working with this data set in the notebook, you can inspect it by opening the emp.csv file in your browser:

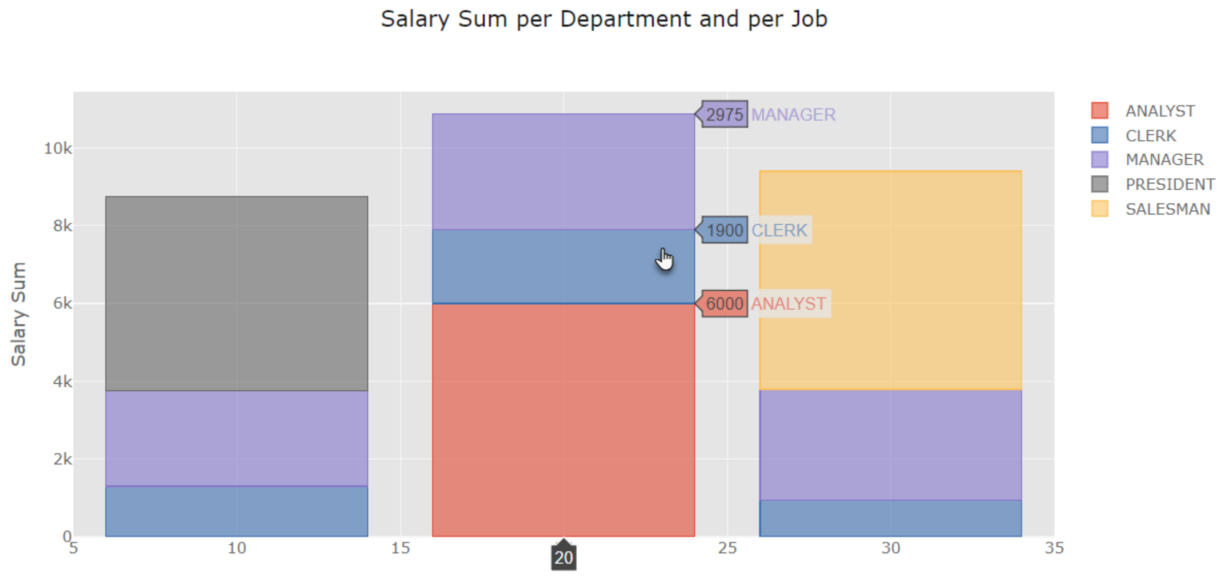


The file’s content is not very special and easy to interpret: 14 employees with an identifying number, a name, a job and manager, a hiredate and salary, for some a commission and for all a department number.



```
empno;ename;job;mgr;hiredate;sal;comm;deptno
7369;SMITH;CLERK;7902;13/06/1993;800.00;0.00;20
7499;ALLEN;SALESMAN;7698;15/08/1998;1600.00;300.00;30
7521;WARD;SALESMAN;7698;26/03/1996;1250.00;500.00;30
7566;JONES;MANAGER;7839;31/10/1995;2975.00;;20
7698;BLAKE;MANAGER;7839;11/06/1992;2850.00;;30
7782;CLARK;MANAGER;7839;14/05/1993;2450.00;;10
7788;SCOTT;ANALYST;7566;05/03/1996;3000.00;;20
7839;KING;PRESIDENT;;09/06/1990;5000.00;0.00;10
7844;TURNER;SALESMAN;7698;04/06/1995;1500.00;0.00;30
7876;ADAMS;CLERK;7788;04/06/1999;1100.00;;20
```

Follow the instructions in the notebook – from loading the dataset into a Pandas data frame to the wrangling and visualization, producing for example this chart (in a single line of code):



The notebook as a final Resources section with some references to documentation and tutorials on Notebooks and Pandas.

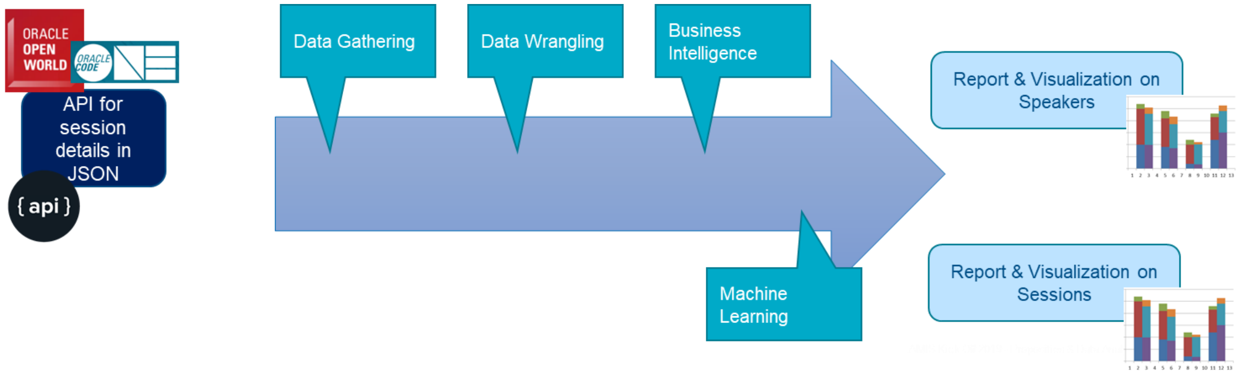
3. Explore the case of the Oracle OpenWorld & CodeOne session catalog

Oracle OpenWorld 2018 was a conference that took place in October 2018 in San Francisco. Over 30,000 attendees participated and visited some 2000 sessions. In this section, we will explore some of the data around this event. We will take a closer look at the speakers and their origin and at the sessions: what are they about, how are they tagged, what are their target audiences and how are they scheduled logistically?

Raw data from the session catalog is available from an API – a REST service that is consumed by the Session Catalog Web UI. This API does not seem intended for data analysts such as we. The data is not in great shape.

The next figure illustrates the steps we will go through in order to get from raw data to answers to burning questions about Oracle OpenWorld 2018 and the collocated CodeOne conference.

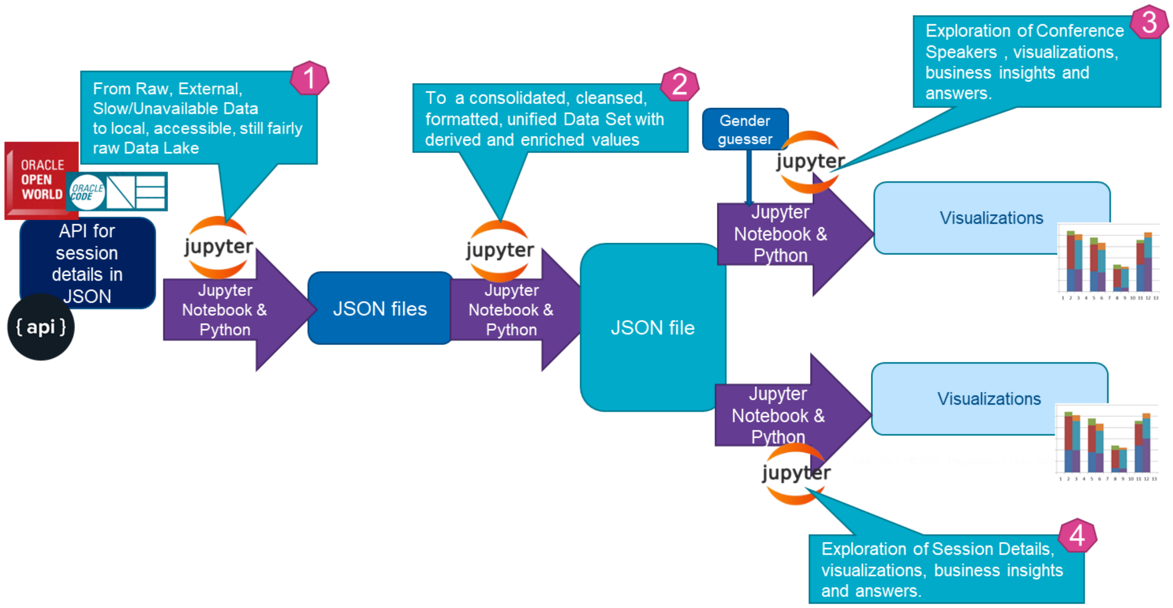
From the raw source, we will gather the data into our local datalake. Next, we will wrangle this data – whip it into shape, filter, cleanse, enrich it so it is easily usable for business intelligence and data science tasks such as machine learning.



We will use various Jupyter Notebooks – powered by Python – to go through these four stages:

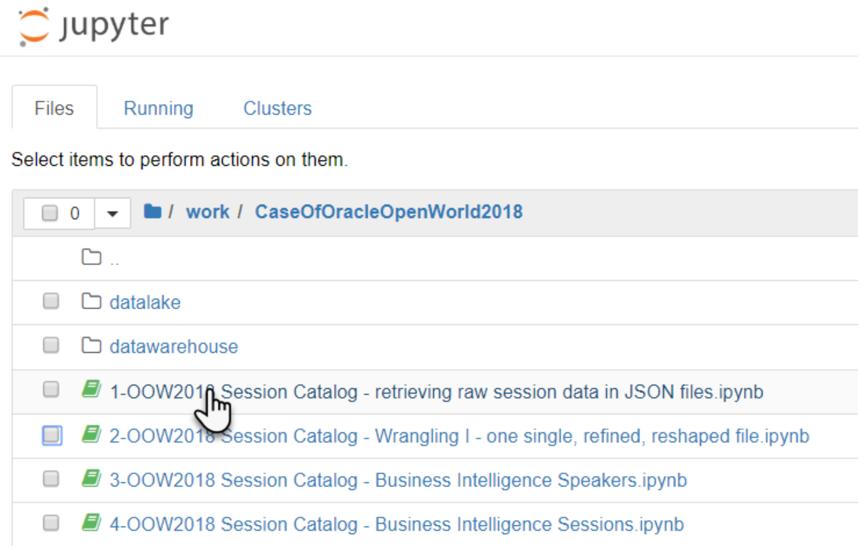
- 1-OOW2018 Session Catalog - retrieving raw session data in JSON files
- 2-OOW2018 Session Catalog - Wrangling I - one single, refined, reshaped file
- 3-OOW2018 Session Catalog - Business Intelligence Speakers
- 4-OOW2018 Session Catalog - Business Intelligence Sessions

The next figure gives a high level overview of what each of these notebooks does:


From Raw, External, Slow/Unavailable Data to local, accessible, still fairly raw Data Lake
1
ORACLE OPEN WORLD
ORACLE CODE
API for session details in JSON
{ api }
jupyter
Jupyter Notebook & Python
JSON files
To a consolidated, cleansed, formatted, unified Data Set with derived and enriched values
2
jupyter
Jupyter Notebook & Python
JSON file
Gender guesser
jupyter
Jupyter Notebook & Python
Exploration of Conference Speakers , visualizations, business insights and answers.
3
Visualizations
Jupyter Notebook & Python
jupyter
Visualizations
Exploration of Session Details, visualizations, business insights and answers.
4

1-OOW2018 Session Catalog - retrieving raw session data in JSON files

From the Jupyter Notebook client running in your browser, open the file *1-OOW2018 Session Catalog - retrieving raw session data in JSON files.ipynb* notebook, from the */work/CaseOfOracleOpenWorld2018* folder.



Select items to perform actions on them.

☐ 0 ▾ / work / CaseOfOracleOpenWorld2018

..

☐ datalake

☐ datawarehouse

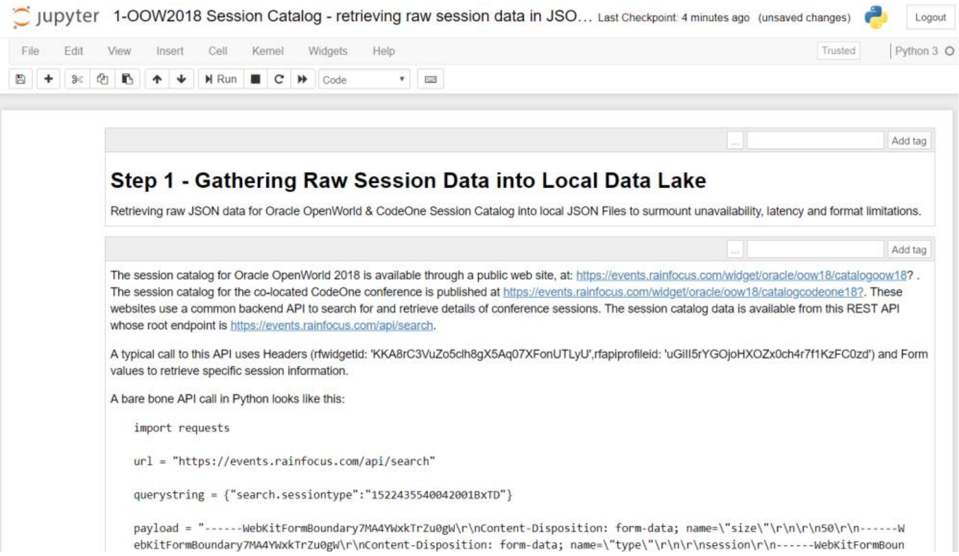
☐ 1-OOW2018 Session Catalog - retrieving raw session data in JSON files.ipynb

☐ 2-OOW2018 Session Catalog - Wrangling I - one single, refined, reshaped file.ipynb

☐ 3-OOW2018 Session Catalog - Business Intelligence Speakers.ipynb

☐ 4-OOW2018 Session Catalog - Business Intelligence Sessions.ipynb

Note: you do not actually need to run this notebook, although you can. Its final output is already available in the */datalake* folder. You can simply browse through the notebook to see how the data retrieval and local storage is done.


jupyter
1-OOW2018 Session Catalog - retrieving raw session data in JSO… Last Checkpoint: 4 minutes ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
Add tag

Step 1 - Gathering Raw Session Data into Local Data Lake

Retrieving raw JSON data for Oracle OpenWorld & CodeOne Session Catalog into local JSON Files to surmount unavailability, latency and format limitations.

The session catalog for Oracle OpenWorld 2018 is available through a public web site, at: https://events.rainfocus.com/widget/oracle/oow18/catalogoow18? . The session catalog for the co-located CodeOne conference is published at https://events.rainfocus.com/widget/oracle/oow18/catalogcodeone18?. These websites use a common backend API to search for and retrieve details of conference sessions. The session catalog data is available from this REST API whose root endpoint is https://events.rainfocus.com/api/search.

A typical call to this API uses Headers (rfwidgetid: 'KKA8rC3VuZo5clh8gX5Aq07XFonUTLyU',rfapiprofileid: 'uGiII5rYGOjoHXOZx0ch4r7f1KzFC0zd') and Form values to retrieve specific session information.

A bare bone API call in Python looks like this:

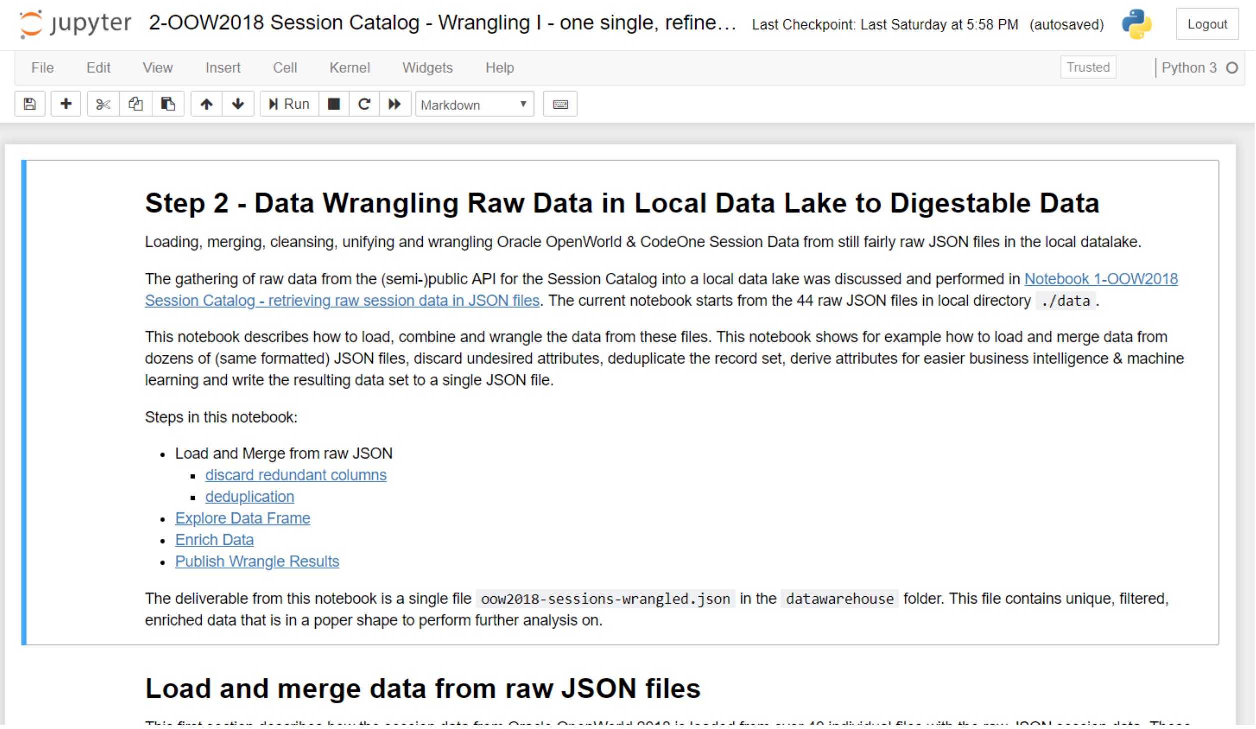
```
import requests

url = "https://events.rainfocus.com/api/search"

querystring = {"search.sessiontype":"1522435540042001BxTD"}

payload = "------WebKitFormBoundary7MA4YWxkTrZu0gW\r\nContent-Disposition: form-data; name=\"size\"\r\n\r\n50\r\n------W
ebKitFormBoundary7MA4YWxkTrZu0gW\r\nContent-Disposition: form-data; name=\"type\"\r\n\r\nsession\r\n------WebKitFormBoun
```

This notebook retrieves the raw data from the remote API – using subsequent HTTP REST requests. The data is gathered using calls per session type (22 different types), per event (two events – Oracle OpenWorld and CodeOne) and per batch of 50. This means that close to 60 HTTP calls are made. The resulting data is in JSON format. This data is written to local files per event and session type (44 in total) in the folder */datalake*.

By perusing the notebook, you will get an idea of how to perform REST calls, how to process the response body and JSON data content, how to turn JSON content in a Pandas Data Frame and how to write the content of such a Data Frame to a JSON file on disk. You will get a glimpse of the raw data. In the next section, we will discuss the Jupyter Notebook that does the main data wrangling, to turn this raw, not very accessible set of data files into a single, refined data warehouse that will be to the delight of data analysts.

2-OOW2018 Session Catalog - Wrangling I - one single, refined, reshaped file

Open the notebook *2-OOW2018 Session Catalog - Wrangling I - one single, refined, reshaped file.ipynb* into your Jupyter Notebook client.


jupyter
2-OOW2018 Session Catalog - Wrangling I - one single, refine…
Last Checkpoint: Last Saturday at 5:58 PM (autosaved)
Logout
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Python 3
Run
Markdown

Step 2 - Data Wrangling Raw Data in Local Data Lake to Digestable Data

Loading, merging, cleansing, unifying and wrangling Oracle OpenWorld & CodeOne Session Data from still fairly raw JSON files in the local datalake.

The gathering of raw data from the (semi-)public API for the Session Catalog into a local data lake was discussed and performed in Notebook 1-OOW2018 Session Catalog - retrieving raw session data in JSON files. The current notebook starts from the 44 raw JSON files in local directory `./data`.

This notebook describes how to load, combine and wrangle the data from these files. This notebook shows for example how to load and merge data from dozens of (same formatted) JSON files, discard undesired attributes, deduplicate the record set, derive attributes for easier business intelligence & machine learning and write the resulting data set to a single JSON file.

Steps in this notebook:

- Load and Merge from raw JSON
    - discard redundant columns
    - deduplication
- Explore Data Frame
- Enrich Data
- Publish Wrangle Results

The deliverable from this notebook is a single file `oow2018-sessions-wrangled.json` in the `datawarehouse` folder. This file contains unique, filtered, enriched data that is in a poper shape to perform further analysis on.

Load and merge data from raw JSON files

This first section describes how the session data from Oracle OpenWorld 2018 is loaded from over 40 individual files with the raw JSON session data. These

This notebook takes 44 separate JSON files in the Data Lake as its input and produces a single consolidated, cleansed and enriched JSON file as its output. The output set has been rid of unnecessary attributes and has newly engineered features that expose attributes for easy data analytics downstream in the data analytics pipeline.

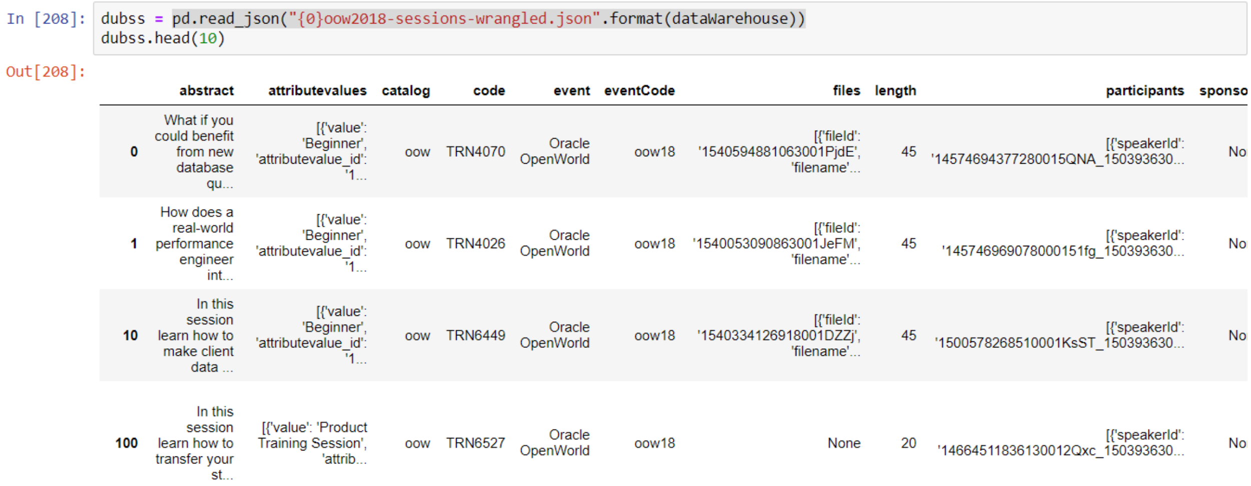
By browsing or stepping through the notebook, you will some important steps in the Data Wrangling process. The 44 JSON files are loaded into Pandas Data Frames that are subsequently merged into one big Data Frame. This joined Data Frame is explored a little – what are the columns, what are the values in those columns. Then unneeded columns are removed – and the set of records [representing conference sessions] is deduplicated by *session code*. This takes the set down to just over 1700 sessions.

After a little digging around some of the columns in the Data Frame, we look closer at the attributeValues, participants and files columns. Each of these contains a nested JSON object that contains relevant data – albeit in a fairly inaccessible way. We expose these data elements by engineering new features, for example

- a flag indicating whether the session is (co-)presented by an Oracle employee,
- another flag indicating if a file (with the presentation) slides has been uploaded,
- flags for specials speaker designations (such as Oracle ACE Director and Java Champion)
- features for each of three audience levels (beginner, intermediate, advanced) and one for *all* in case all three are marked
- a feature for track(s) to which the session is assigned
- a feature for the number of speakers associated with a session
- a session instance count for the number of times the session is scheduled to take place

- a feature to indicate the room capacity of the (first) room in which the session is scheduled
- features for day, time(slot) and timestamp

Finally, with all these features added to the data set, the data is saved to disk in folder */datawarehouse* as single, consolidated file that is just waiting for data analysts to get their hands on.



```
In [208]: dubss = pd.read_json("{0}oow2018-sessions-wrangled.json".format(dataWarehouse))
          dubss.head(10)
```

Out[208]:

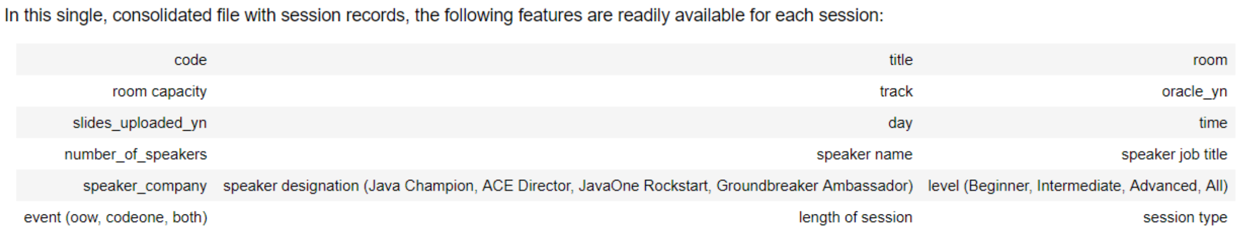
	abstract	attributevalues	catalog	code	event	eventCode	files	length	participants	sponso
0	What if you could benefit from new database qu...	[{'value': 'Beginner', 'attributevalue_id': '1...	oow	TRN4070	Oracle OpenWorld	oow18	[{'fileId': '1540594881063001PjdE', 'filename'...	45	[{'speakerId': '1457469437728001 5QNA_150393630...	No
1	How does a real-world performance engineer int...	[{'value': 'Beginner', 'attributevalue_id': '1...	oow	TRN4026	Oracle OpenWorld	oow18	[{'fileId': '1540053090863001JeFM', 'filename'...	45	[{'speakerId': '1457469690780001 51fg_150393630...	No
10	In this session learn how to make client data ...	[{'value': 'Beginner', 'attributevalue_id': '1...	oow	TRN6449	Oracle OpenWorld	oow18	[{'fileId': '1540334126918001DZZj', 'filename'...	45	[{'speakerId': '150057826851000 1KsST_150393630...	No
100	In this session learn how to transfer your st...	[{'value': 'Product Training Session', 'attrib...	oow	TRN6527	Oracle OpenWorld	oow18	None	20	[{'speakerId': '146645118361300 12Qxc_150393630...	No

3-OOW2018 Session Catalog - Business Intelligence Speakers

The third notebook is the first to leverage the wrangled data, do some additional wrangling and then produce findings, answers on business questions and show some data visualizations.

Load *3-OOW2018 Session Catalog - Business Intelligence Speakers.ipynb* into the Jupyter Notebook client.

Here is a list of all features that our data set contains at this stage



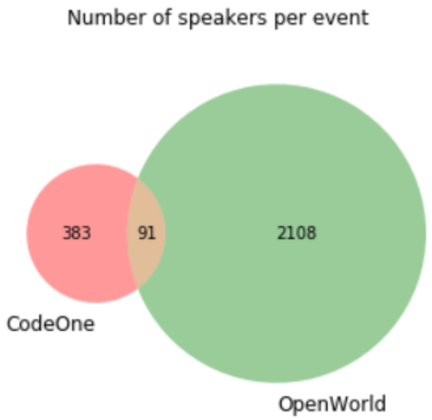
In this single, consolidated file with session records, the following features are readily available for each session:

code	title	room
room capacity	track	oracle_yn
slides_uploaded_yn	day	time
number_of_speakers	speaker name	speaker job title
speaker_company	speaker designation (Java Champion, ACE Director, JavaOne Rockstart, Groundbreaker Ambassador)	level (Beginner, Intermediate, Advanced, All)
event (oow, codeone, both)	length of session	session type

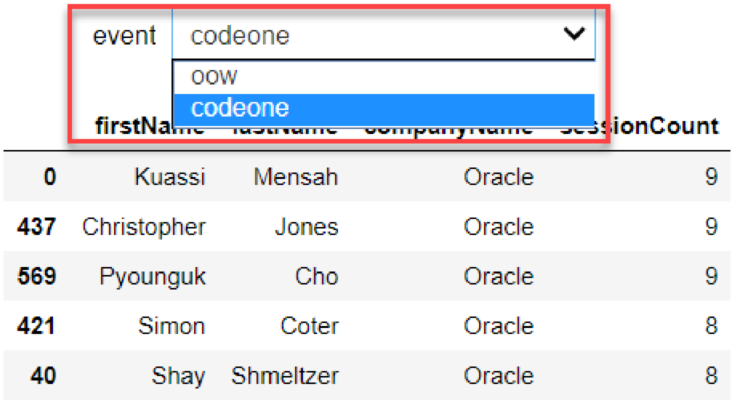
The first step after loading the wrangled dataset from the */datawarehouse* folder is further data preparation: a new Pandas Data Frame *sp* is created with speaker records. These records are produced from the *participants* feature in the sessions data set.

The *sp* data frame has the following features: firstName, lastName, jobTitle, companyName, bio, oracle_employee (special Y/N flag), photoURL, twitter, sessionCount, oow (Y/N flag to indicate if the speaker performs at the Oracle OpenWorld event), codeone (Y/N flag to indicate that the speaker presents a session at CodeOne) and finally four Y/N flag columns for the special designations that speakers can have: 'JavaOne Rockstar', 'Oracle ACE Director', 'Oracle Java Champion', 'Groundbreaker Ambassador'.

The first section in the notebook after the creation of the *sp* Data Frame for speakers is about simple counting: the number of speakers in general and per event. Here we also see te first visualization: a Venn diagram as alternative representation for data earlier presented in a Cross Table.

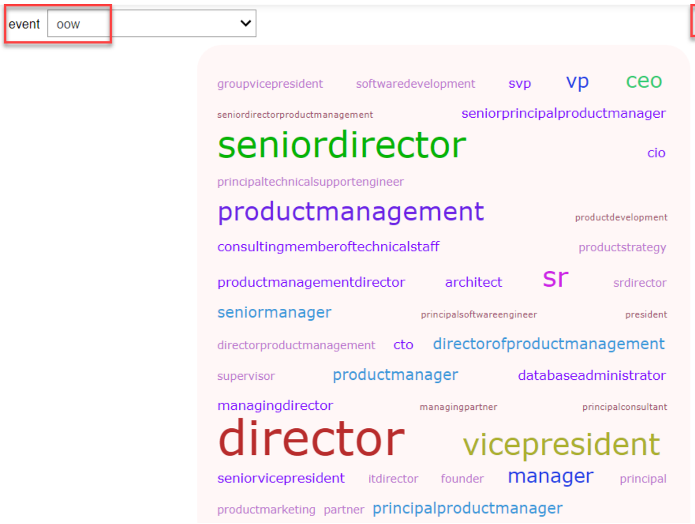


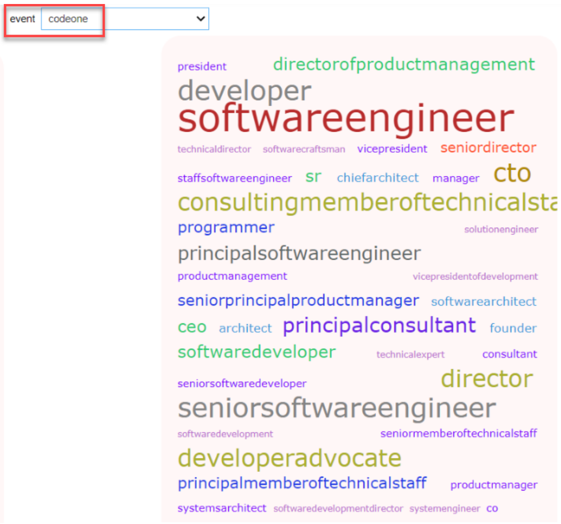
The notebook continues with an analysis of very active speakers. It turns out that one speaker takes part in now fewer than 12 sessions. It turns out that the top 10 of active speakers only has speakers working for Oracle. We then have a cell in the notebook that allows us to look at active speakers per event – for oow and codone respectively. This cell uses an interactive widget – in this case a dropdown list – that allows us to interact with the notebook:



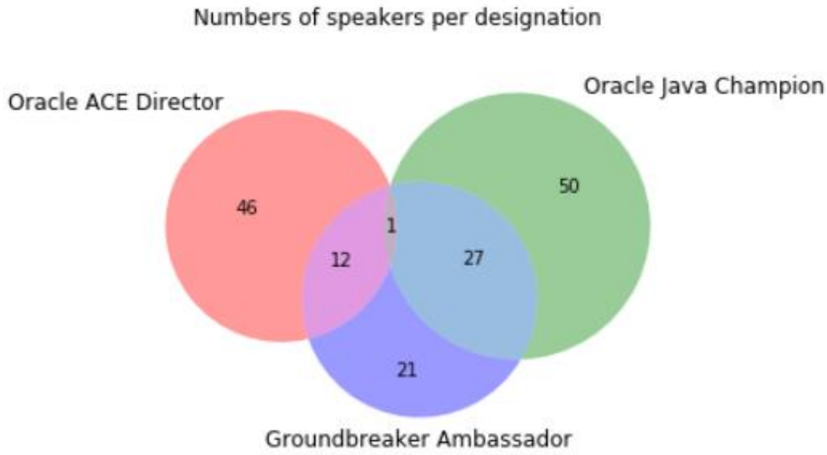
	firstName	lastName	companyName	sessionCount
0	Kuassi	Mensah	Oracle	9
437	Christopher	Jones	Oracle	9
569	Pyounguk	Cho	Oracle	9
421	Simon	Coter	Oracle	8
40	Shay	Shmeltzer	Oracle	8

The next section looks at Job Titles. The final visualization in this section uses a Word Cloud to show the most frequently used job titles – for the Oracle OpenWorld and CodeOne events side by side. The difference is striking:


event oow

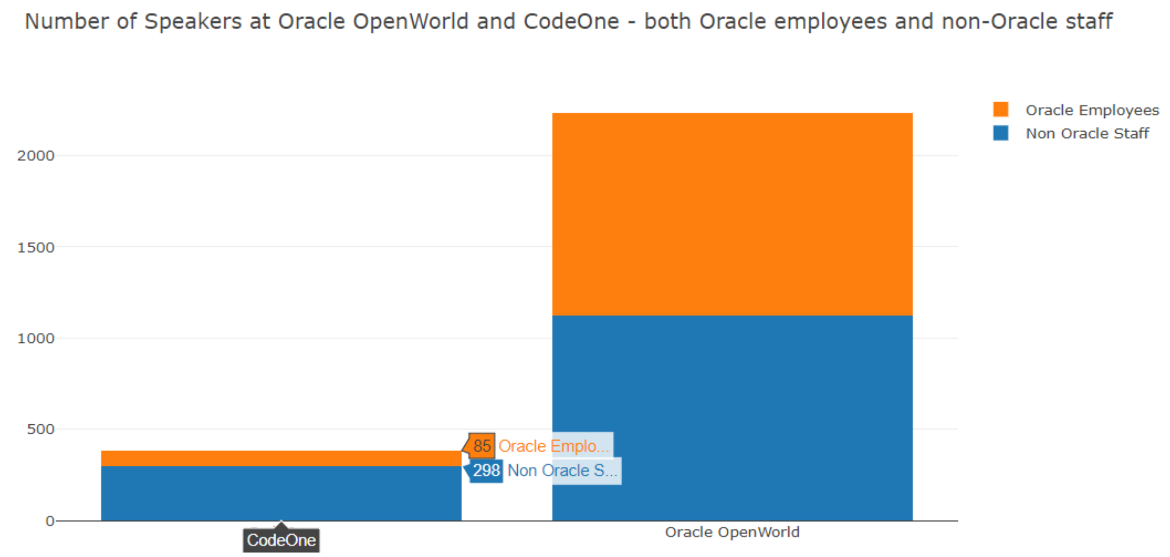

event codeone

The notebook does a brief analysis of the designations or titles of speakers. It counts the number of speakers per designation and looks at how many speakers have specific combinations of titles. For some people in the community, this is a very important topic.



The companies that the speakers are working for is looked at next. It quickly turns out that in the top 10 of companies with the highest numbers of speakers, we find mainly sponsors of the events. Wells Fargo is the first non-sponsor company on the list.

The number of speakers employed by Oracle is very substantial, especially at Oracle OpenWorld:



A brief analysis of the incidence of first names quickly reveals that David, John and Michael are the top 3 first names among speakers.

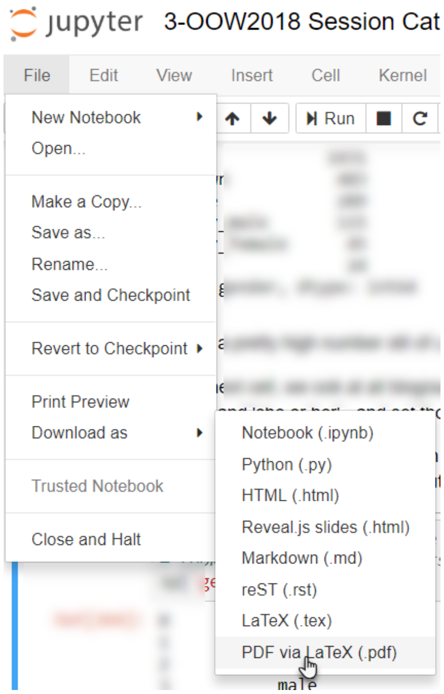
Gender Guessing

It is interesting to know if women appear as speakers at the two events - given our strive for more *women in tech*. We may be able to stimulate the participation of women if we better understand what type of event attracts female speakers and which tracks perhaps have a higher than average percentage of female speakers. However: our data set does not contain and straightforward gender information. Gender is not a feature of our speaker records.

Does our exploration end before it event started - for lack of data? Or can we be creative? Perhaps we can guess the gender from the first name feature that we do have for all speakers. And check the biographies or gender specific terms such as he and she, his, hum and her. Using these tricks – which turns out to be not very difficult at all - we get a fair idea about the gender distribution and whether there is a significant distinction between Oracle OpenWorld and CodeOne in this regard.

Create PDF Report from Notebook

It is quite simple to create a PDF report from the notebook. Use File | Download as | PDF via LaTeX (.pdf) to have a PDF rendition downloaded:



The generated PDF document opens in your browser (or downloads to your browser's local file system.

Note that many other formats are available as well.

Running Jupyter Notebooks in the background from the command line

Jupyter Notebooks can be run in the background – for example scheduled to run every day to process the latest available input into output – both the output produced in the cells in the notebook and the representation in the selected format of the executed notebook itself.

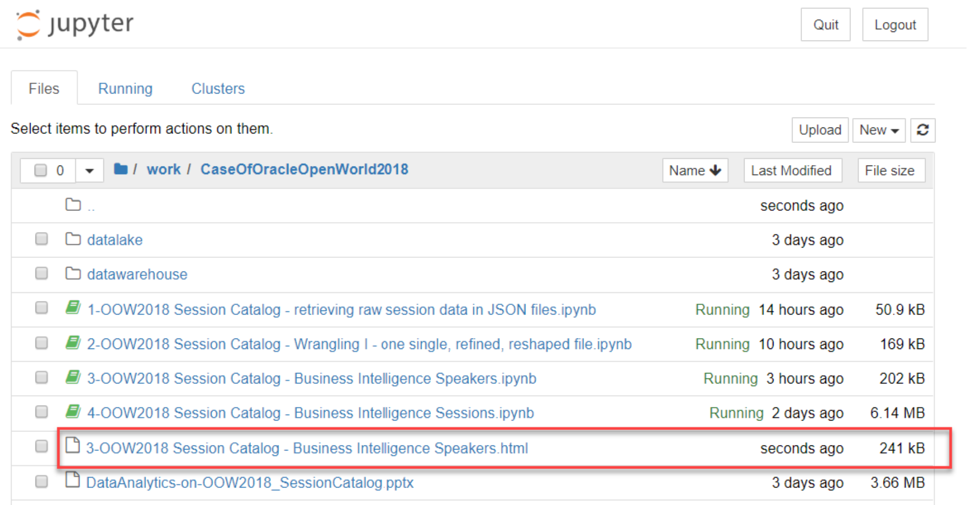
On the command line of the Jupyter Notebook server , you can use a command such as the following to run (and convert) a notebook:

```
jupyter nbconvert --to html --template basic  mynotebook.ipynb
```

In our environment, you can use the following command on the command line terminal – either in the Katacoda environment or locally on your Docker host – to run the step 3 notebook discussed in this section and produce HTML output:

```
jupyter nbconvert --to html --template basic  ~/work/CaseOfOracleOpenWorld2018/'3-
OOW2018 Session Catalog - Business Intelligence Speakers.ipynb'
```

This results in an HTML document being generated; any other side effect caused by the notebook is also created by running the notebook with nbconvert (for example files, API calls, database updates):



Select items to perform actions on them.

0 / work / CaseOfOracleOpenWorld2018

Name		Last Modified	File size
..		seconds ago	
datalake		3 days ago	
datawarehouse		3 days ago	
1-OOW2018 Session Catalog - retrieving raw session data in JSON files.ipynb	Running	14 hours ago	50.9 kB
2-OOW2018 Session Catalog - Wrangling I - one single, refined, reshaped file.ipynb	Running	10 hours ago	169 kB
3-OOW2018 Session Catalog - Business Intelligence Speakers.ipynb	Running	3 hours ago	202 kB
4-OOW2018 Session Catalog - Business Intelligence Sessions.ipynb	Running	2 days ago	6.14 MB
3-OOW2018 Session Catalog - Business Intelligence Speakers.html		seconds ago	241 kB
DataAnalytics-on-OOW2018_SessionCatalog.pptx		3 days ago	3.66 MB

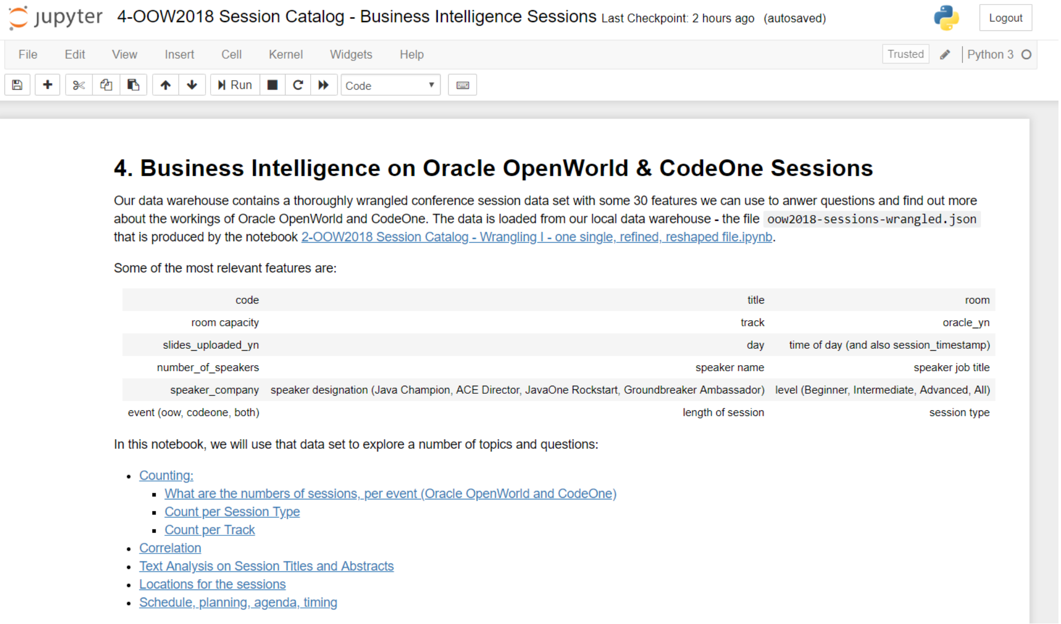
The documentation for nbconvert can be found here: https://nbconvert.readthedocs.io/en/latest/usage.html

A notebook can also be downloaded as or converted to executable script – taking only the Code cells and discarding all Markdown cells.

4-OOW2018 Session Catalog - Business Intelligence Sessions

In this section, we will analyze data on conference sessions. How many sessions, on which topics, for which levels, and more.

Load *4-OOW2018 Session Catalog - Business Intelligence Sessions* into the Jupyter Notebook client.


jupyter
4-OOW2018 Session Catalog - Business Intelligence Sessions Last Checkpoint: 2 hours ago (autosaved)
Logout
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Python 3
Run
Code

4. Business Intelligence on Oracle OpenWorld & CodeOne Sessions

Our data warehouse contains a thoroughly wrangled conference session data set with some 30 features we can use to anwer questions and find out more about the workings of Oracle OpenWorld and CodeOne. The data is loaded from our local data warehouse - the file `oow2018-sessions-wrangled.json` that is produced by the notebook 2-OOW2018 Session Catalog - Wrangling I - one single, refined, reshaped file.ipynb.

Some of the most relevant features are:

code	title	room
room capacity	track	oracle_yn
slides_uploaded_yn	day	time of day (and also session_timestamp)
number_of_speakers	speaker name	speaker job title
speaker_company	speaker designation (Java Champion, ACE Director, JavaOne Rockstart, Groundbreaker Ambassador)	level (Beginner, Intermediate, Advanced, All)
event (oow, codeone, both)	length of session	session type

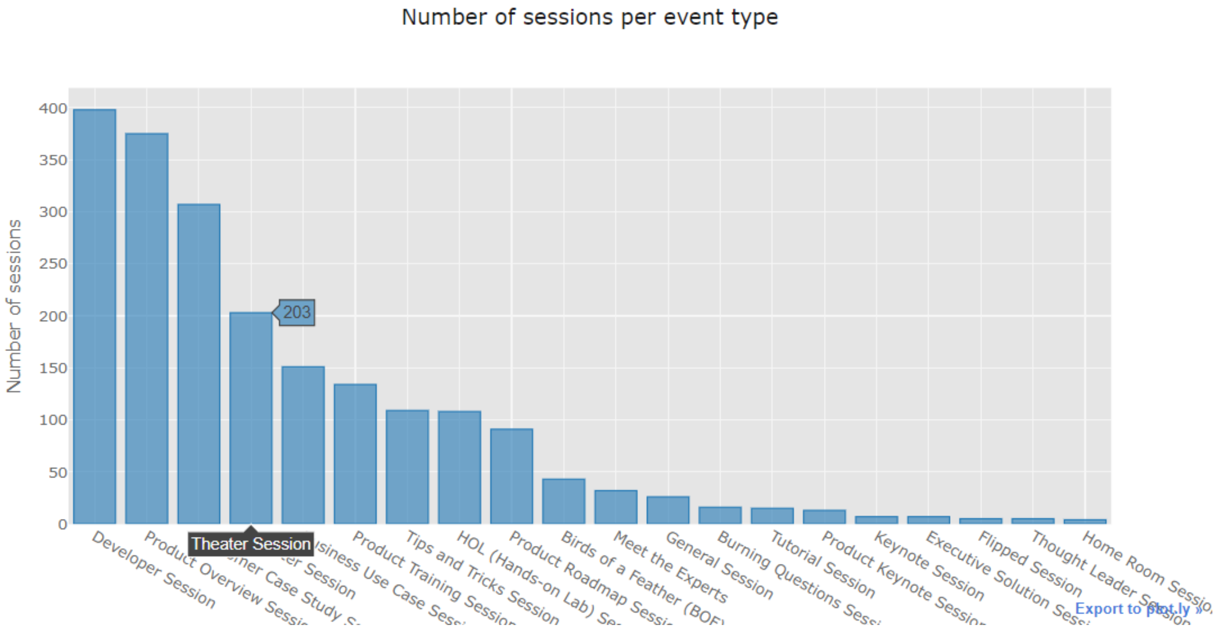
In this notebook, we will use that data set to explore a number of topics and questions:

- Counting:
  - What are the numbers of sessions, per event (Oracle OpenWorld and CodeOne)
  - Count per Session Type
  - Count per Track
- Correlation
- Text Analysis on Session Titles and Abstracts
- Locations for the sessions
- Schedule, planning, agenda, timing

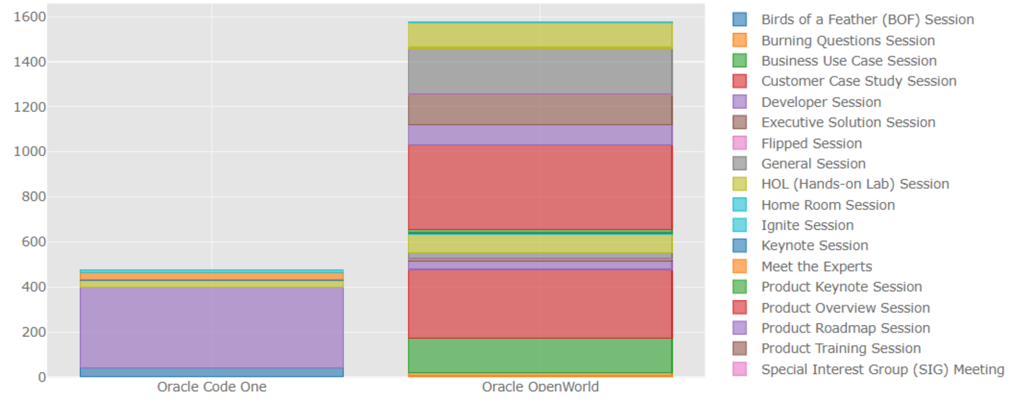
The first part of the notebook loads the wrangled data from file in the *datawarehouse* folder. It then does a little additional wrangling: the session_datetime is constructed as a datetime object from the session_timestamp feature that was read from the file; this feature expresses a timestamp as a number of milliseconds since 1970 – a value that is difficult to work with in data analysis. New features are derived as well: primary_track and secondary_track (from the *track* column) and the *track_count*. A quick describe() on the *ss* dataframe helps us get insight in the numerical columns.

Navigate to the next section on *Session Counting*. There are multiple ways to skin this cat. Using *value_counts()* is the easiest. Using *groupBy()* and aggregation operators such as *size()* – as well as sum, min, max, … - for more advanced ways of counting.

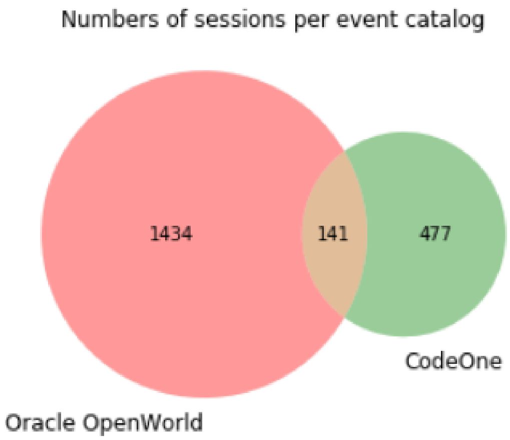
The *plotly* library is integrated into Pandas to allow visualization of data. This histogram shows the numbers of sessions per *session type*.


Number of sessions per event type

The next cell renders a stacked bar chart that shows the number of sessions per event and per type. The subsequent figure switches event and session type:

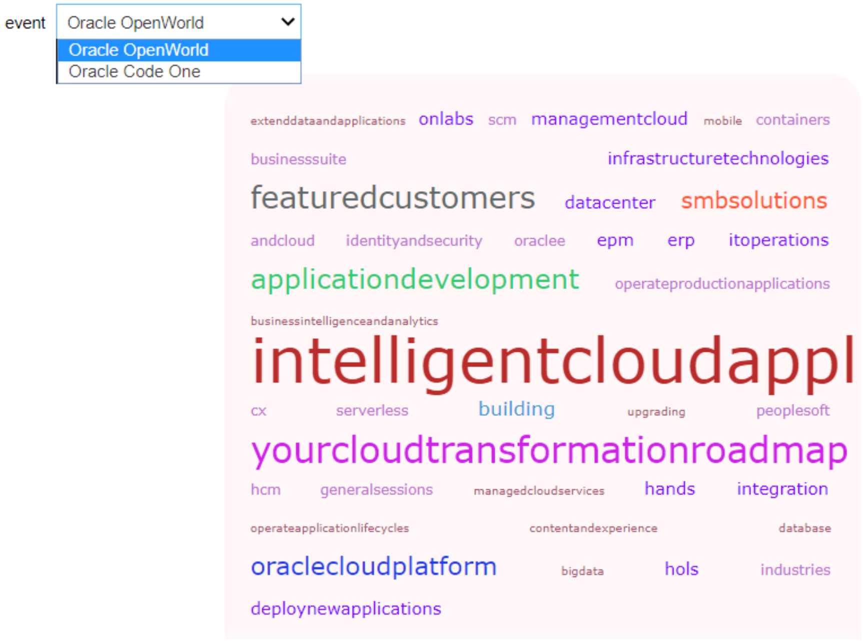


The notebook then shows the number of sessions per catalog: sessions exclusively for Oracle OpenWorld or for CodeOne and a number of sessions listed in both catalogs:



The session counting continues with a brief look at sessions with at least one Oracle speaker – this turns out to be the case very frequently at Oracle OpenWorld. Next, we check out per track: sessions can be associated with tracks and this supposedly helps the attendees find the sessions they like. One may wonder if that works out so well – given the fact that some tracks have one in three sessions associated with them, and others just two or three sessions.

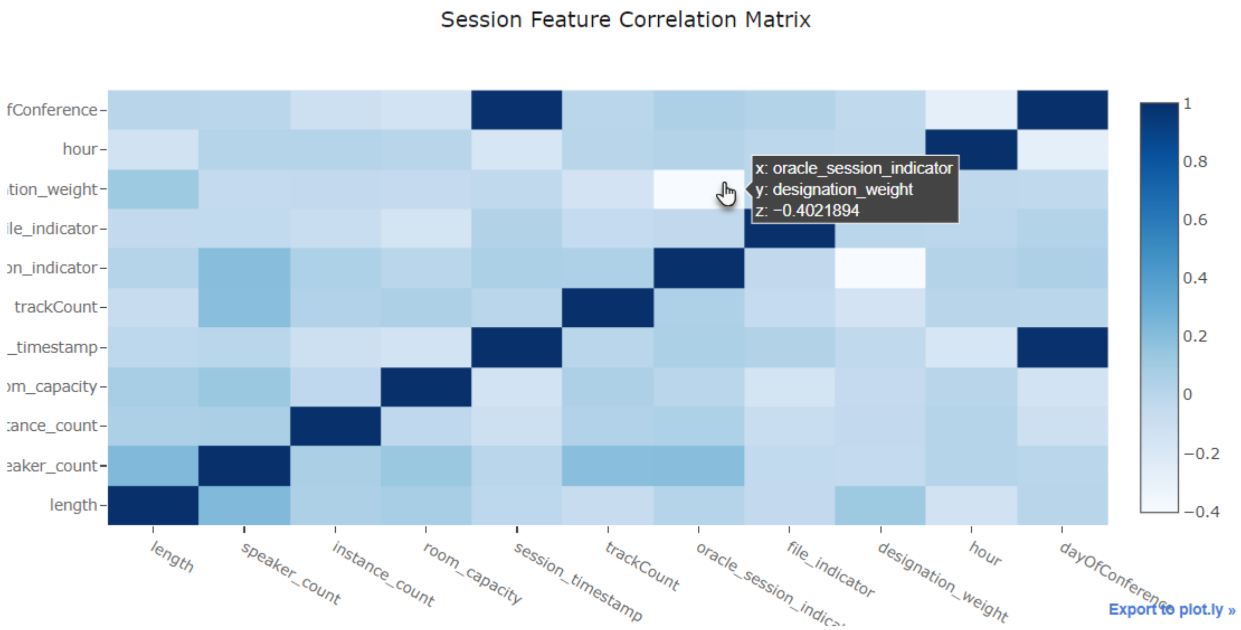
Using a word cloud, we can quickly get a feel for the important tracks (in terms of the numbers of sessions) for the two events.



event Oracle OpenWorld

Correlation

With a Pandas, Python and our Jupyter Notebook, we can easily explore some of the correlations that may hide in our data. We do not say anything about causality- just the somewhat more than average coincidence of values. The notebook demonstrates how to turn some categorical features into numerical values and uses those for a very simple correlation analysis:



The darker the cell, the higher the correlation between the attributes defining the row and the column. In this figure, we see for example that there is a negative correlation between *designation_weight* and *oracle_session_indicator*. In words: sessions with speakers working for Oracle have fewer speakers with special titles and designations than average sessions do. Which is no surprise, given that Oracle employees cannot have such titles and designations.

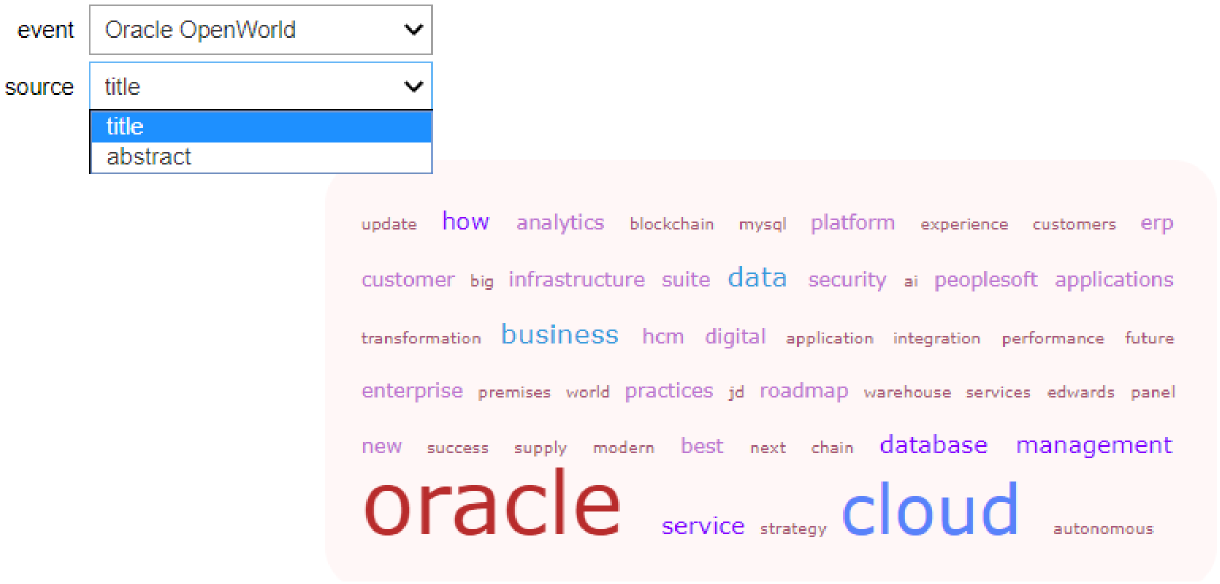
You could add these two feature derivations:

```
ss['titleLength']= ss['title'].apply(len) # the number of characters used in the title (max 80)

ss['abstractLength']= ss['abstract'].apply(len) # the number of characters used in the title (max 750)
```

And check for correlations. Note the correlation between *designation_weight* and *abstract_length*. Speakers with titles and designations tend to write longer abstracts it would seem. The longer the abstract, the longer the session is another finding.

Text Analysis of Title and Abstract

A simple text analysis is done of either session titles or session abstracts and presented in a tag cloud. Which terms are frequently used in the titles and abstracts for either Oracle OpenWorld or CodeOne:



event: Oracle OpenWorld

source: title

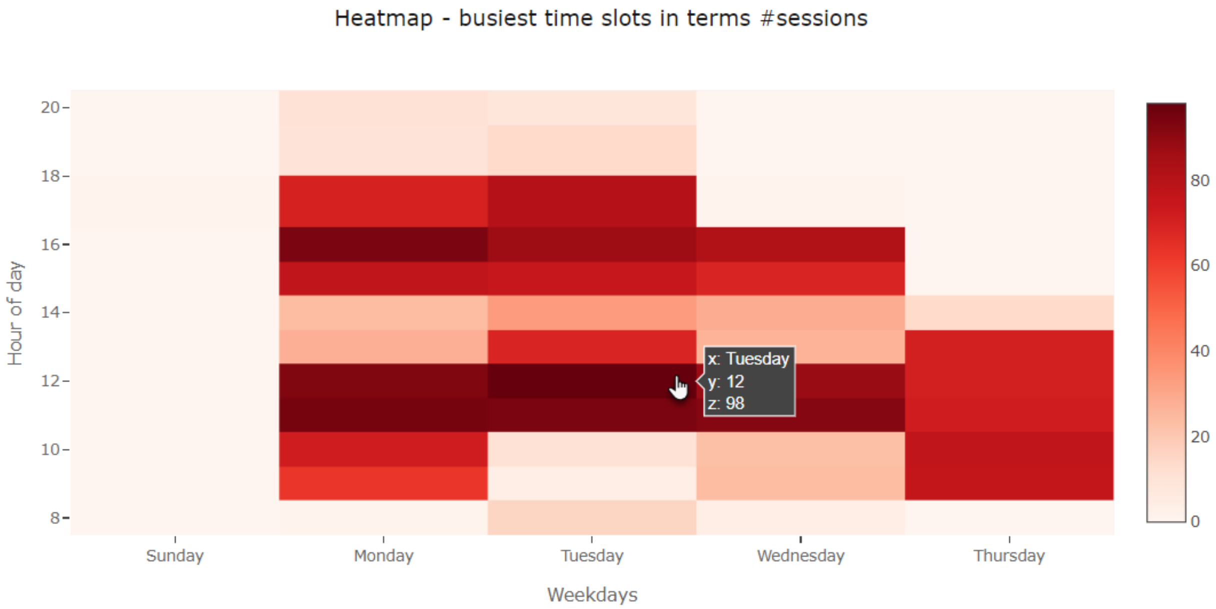
Venues and Session Locations

From the *room* feature that describes in which room a session takes place, we can derive the seven venues used for the conferences. A new *venue* feature is engineerd.

Subsequently, we can find out about the busiest venues, in terms of rooms and sessions and total room capacity.

Time is of the essence

The final section in the notebook – before the data frame in its engineered state is saved to the *datamart* – takes a closer look at time: how are the sessions distributed over the days and what are the busy timeslots. It turns out to be quite simple to produce this heatmap to identify the busy slots:

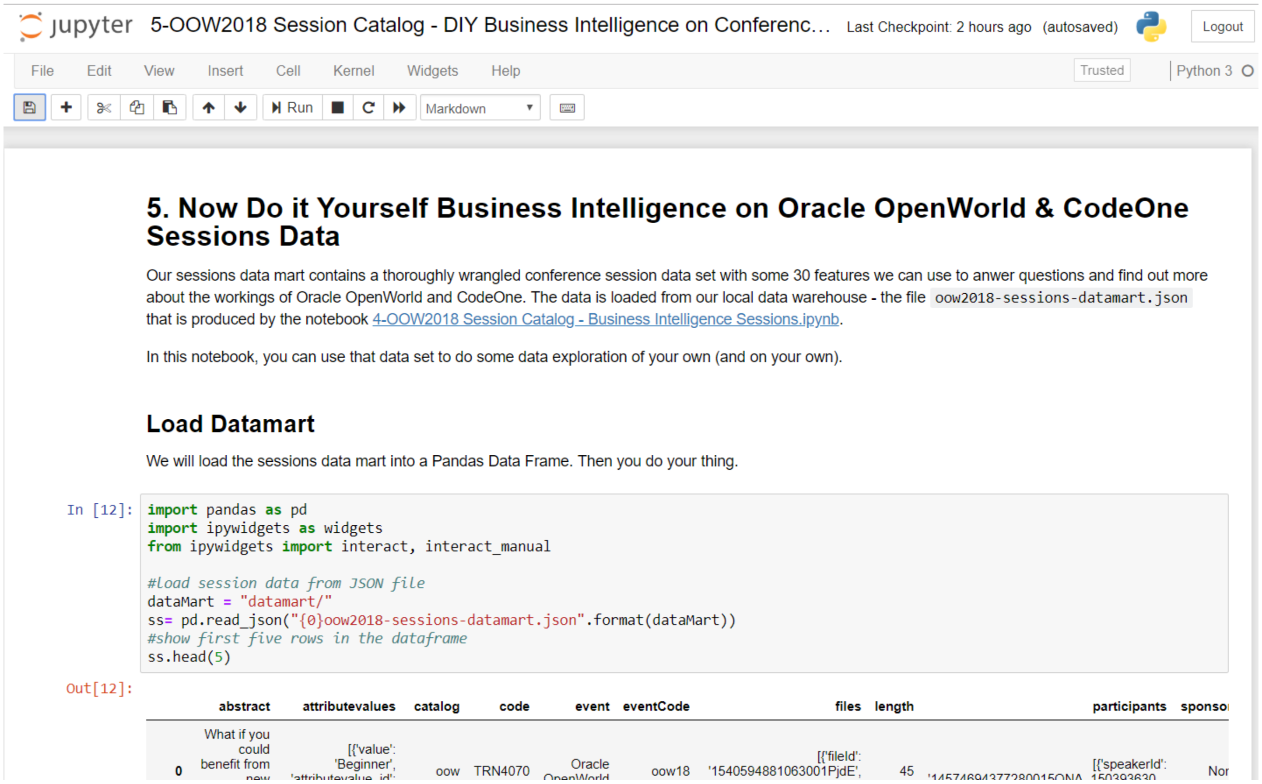


A little bit more complex is the Gantt Chart – a common way of presenting the usage of resources (the rooms) through time (for sessions).

5-OOW2018 Session Catalog - DIY Business Intelligence on Conference Sessions

The fifth notebook in this series is almost empty. It loads the enriched session data that were created in the fourth notebook from the datamart file into a Pandas Data Frame. At this point – it is up to you. Try and get a little data analysis going on the sessions.

Load *5-OOW2018 Session Catalog - DIY Business Intelligence on Conference Sessions* into the Jupyter Notebook client.



jupyter 5-OOW2018 Session Catalog - DIY Business Intelligence on Conferenc… Last Checkpoint: 2 hours ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Markdown

5. Now Do it Yourself Business Intelligence on Oracle OpenWorld & CodeOne Sessions Data

Our sessions data mart contains a thoroughly wrangled conference session data set with some 30 features we can use to anwer questions and find out more about the workings of Oracle OpenWorld and CodeOne. The data is loaded from our local data warehouse - the file `oow2018-sessions-datamart.json` that is produced by the notebook 4-OOW2018 Session Catalog - Business Intelligence Sessions.ipynb.

In this notebook, you can use that data set to do some data exploration of your own (and on your own).

Load Datamart

We will load the sessions data mart into a Pandas Data Frame. Then you do your thing.

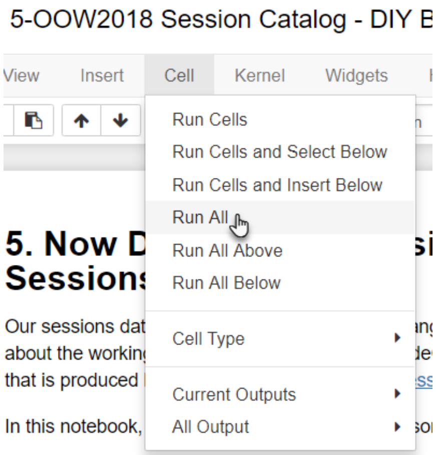
In [12]:

```
import pandas as pd
import ipywidgets as widgets
from ipywidgets import interact, interact_manual

#load session data from JSON file
dataMart = "datamart/"
ss= pd.read_json("{0}oow2018-sessions-datamart.json".format(dataMart))
#show first five rows in the dataframe
ss.head(5)
```

Out[12]:

	abstract	attributevalues	catalog	code	event	eventCode	files	length		participants	sponso
0	What if you could benefit from new	[{'value': 'Beginner', 'attributevalue_id':	oow	TRN4070	Oracle OpenWorld	oow18	[{'fileId': '1540594881063001PjdE',	45	'1457469437728001QNA	[{'speakerId': 150393630	Nor

Run the notebook:



Scroll to the end, to section *Your Data Exploration, Analysis and Visualization*. Now create a new cell. Of type Code (the default).

Then type into the cell:

```
ss.columns
```

Then press CTRL+Enter (on Windows) or whatever the shortcut is for running a cell on your platform.

Create a new cell. Type:

```
ss['type'].value_counts()
```

and execute the cell. What does this tell you?

Now in the same or another new cell, type:

```
pd.crosstab(index=ss['catalog'], columns=ss['type'])
```

and execute the cell. Which Session Type may not make an appearance on Oracle OpenWorld 2019?

4. On your own with Titanic: get started on your very own Jupyter Notebook.

Now it is time to create your very own notebook. The Titanic dataset you are going to use is stored in the *Titanic* folder. Create a new Notebook in the same folder by clicking *New --> Python 3.* This will create a new Untitled notebook which you can rename in the notebook itself. Remember that if you are working in Katacoda, the notebook is in the cloud and your notebook won't be stored. If you log out or the container is destroyed, your notebook will be lost!

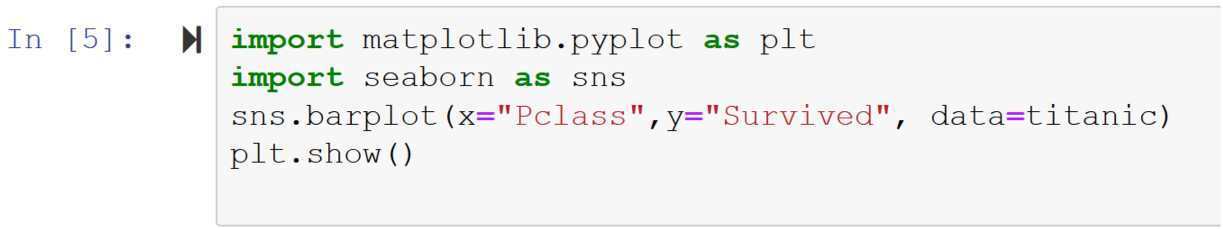
Inspect the data

Load the Titanic.csv dataset into your notebook and start investigating the data (use the read_csv function of pandas for this, data is separated by a comma). You will notice the following columns:

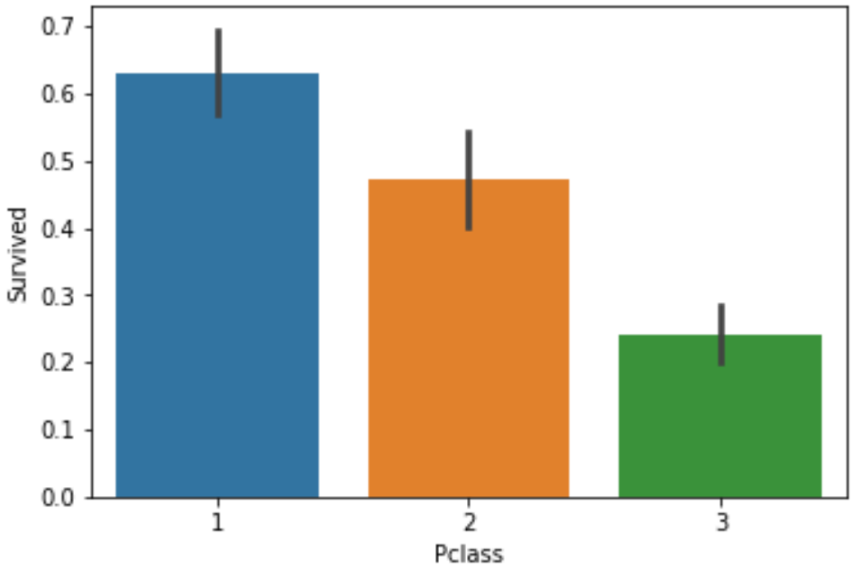
COLUMN	DESCRIPTION	TYPE
PassengerId	ID of passenger	Integer
Survived	If passenger survived or not. This will be the classifier to train a model on	Boolean
Pclass	Class of travel	Integer
Name	Name of passenger	String
Sex	Gender	String
Age	Age of passenger	Int
SibSp	Number of Sibling/Spouse aboard	Int
Parch	Number of Parent/Child aboard	Int
Ticket	Ticket number	String
Fare	Price of ticket	Float
Cabin	Cabin number of passengers	String
Embarked	The port in which a passenger has embarked. C = Cherbourg, S = Southampton, Q = Queenstown	String

You can use pyplot in combination with the seaborn library to visually show the data. For example, to show the correlation between class and survival rate use:

```
import matplotlib.pyplot as plt
import seaborn as sns
sns.barplot(x=“Pclass”,y=“Survived”, data=titanic)
plt.show()
```



```
In [5]:  import matplotlib.pyplot as plt
         import seaborn as sns
         sns.barplot(x="Pclass",y="Survived", data=titanic)
         plt.show()
```



Use this to also look at the correlation between other attributes and see if you can find out something useful.

Wrangle the dataset

First, check if the data contains missing values. Use the following function to count missing values and see which columns have missing values:

```
titanic.apply(lambda x: sum(x.isnull()), axis=0)
```

You will have to deal with missing values. The following common used methods can be used to deal with missing values:

- Drop records
- Drop attributes
- Insert attribute mean for missing values
- Insert attribute median for missing values
- Use regression for insert missing values
- (Or any other statistical method)

Determine yourself which method is suitable for which missing values or try different methods. See section handy functions to transform your data.

Most of the machine learning algorithms (such as SVM, neural networks) only allow numbers as input values. Therefore, all attributes with Strings should be transformed to integer values. Tip, categorize String attributes into int groups, i.e. put all passengers embarked from C into group 1, embarks from S into group 2, and embarks from Q into group 3.

Finally, determine which attributes you think are not useful. These attributes can be dropped. Note that you will have to look carefully at the dataset, since some attributes may not look useful at first but if you look closer it may contain some information. Hint, it may look like that you cannot use the Name attribute since you cannot easily transform it to an integer, but if you look closer at the data it contains some information about the status of the person, ie some persons have a higher status/title (Dr, Capt, Sir, etc). This might have a correlation with the survival rate.

Test your data

After you have cleaned up your data, it is time to create a model and test the accuracy. Since this is not a workshop on creating models, use the following code to apply a Logistic Regression and a Support Vector Machine on your dataset, adjust the feature_names array to your needs. A value between 0 and 1 is outputted, where a higher number means a better result. Try different modifications on your data and see the effects of it.

```
from sklearn.svm import SVC

###### PUT ALL ATTRIBUTES YOU WANT TO USE TO CREATE A MODEL IN THIS ARRAY #####
feature_names = ['Sex', 'Age', 'SibSp', 'Parch']
X = titanic[feature_names]
y = titanic['Survived']

# This will split up your dataset into a test and training set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

# Apply logistic regression on your dataset
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression()
logreg.fit(X_train, y_train)
print('Accuracy of Logistic regression classifier on training set: {:.3f}'
     .format(logreg.score(X_train, y_train)))
print('Accuracy of Logistic regression classifier on test set: {:.3f}'
     .format(logreg.score(X_test, y_test)))

# Apply support vector machine on your dataset
from sklearn.svm import SVC
svm = SVC()
svm.fit(X_train, y_train)
print('Accuracy of SVM classifier on training set: {:.3f}'
     .format(svm.score(X_train, y_train)))
print('Accuracy of SVM classifier on test set: {:.3f}'
      .format(svm.score(X_test, y_test)))
```

Some handy Panda functions

All functions which you can apply on your titanic dataset can be found here:

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html

Some handy functions you can use, do not forget to assign the result of the functions to the correct column of your dataset or the dataset itself if the function changes the data (assuming you loaded your dataset into the variable titanic):

```
titanic.shape
```

dimensions of the dataset

```
titanic.dtypes
```

shows the type of each column

```
titanic["column"].value_counts()
```

counts the number of records for each unique value in the column

```
titanic["column"]= titanic["column"].fillna("somevalue")
```

sets empty values of the column to specified value

```
titanic["column"]= titanic["column1"].fillna(titanic["column2"].mean())
```

sets empty values to mean of column

```
titanic["Sex"] = titanic["Sex"].map({"female": 0, "male": 1})
```

maps values to other values

```
titanic = titanic.drop(["column1", "column2"], axis=1)
```

drops columns

```
titanic.dropna(subset=["column"], inplace=True)
```

drop records with no value in column

```
titanic["Name"].apply(lambda x: x.split(",")[0])
```

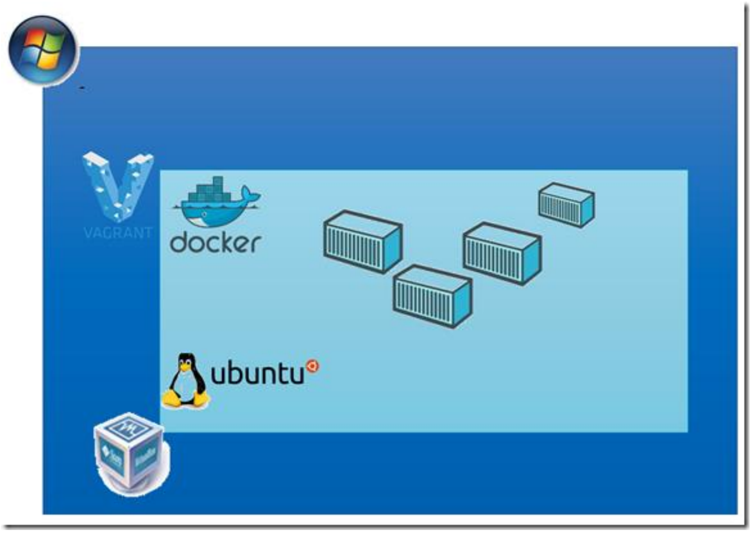
apply a function to each record of the table, this for example gets first name of each person

Appendix A - Run a Linux VM locally as Docker engine

You may already have a Linux VM with Docker engine running on your laptop. If that is the case, you skip the next section and jump to *Running Jupyter on Docker*. If you currently do not have the capability to run Docker containers, then continue with the next section that will provide pointers for installing Virtual Box, Vagrant and using these two a spinning up a Linux Ubuntu Virtual Machine with Docker engine.

Installing Virtual Box and Vagrant

Here we will use the combination of VirtualBox and Vagrant to manage one (or more) Linux Ubuntu VM(s) with Docker inside and keep your Windows or Mac OS environment very uncluttered, as visualized in the next figure.



If you currently do not have Oracle Virtual Box installed, please download and install from https://www.virtualbox.org/wiki/Downloads .

Next, if you currently do not have Vagrant installed, please download and install from https://www.vagrantup.com/downloads.html.

After installing Vagrant, you need to install two vagrant plugins – in order to provide docker-compose into the VM and to allocate a greater than default disk size. Execute these two statements from the command line on your laptop:

```
vagrant plugin install vagrant-docker-compose
```

```
vagrant plugin install vagrant-disksize
```

Creating an Ubuntu VM with Docker Host using Vagrant

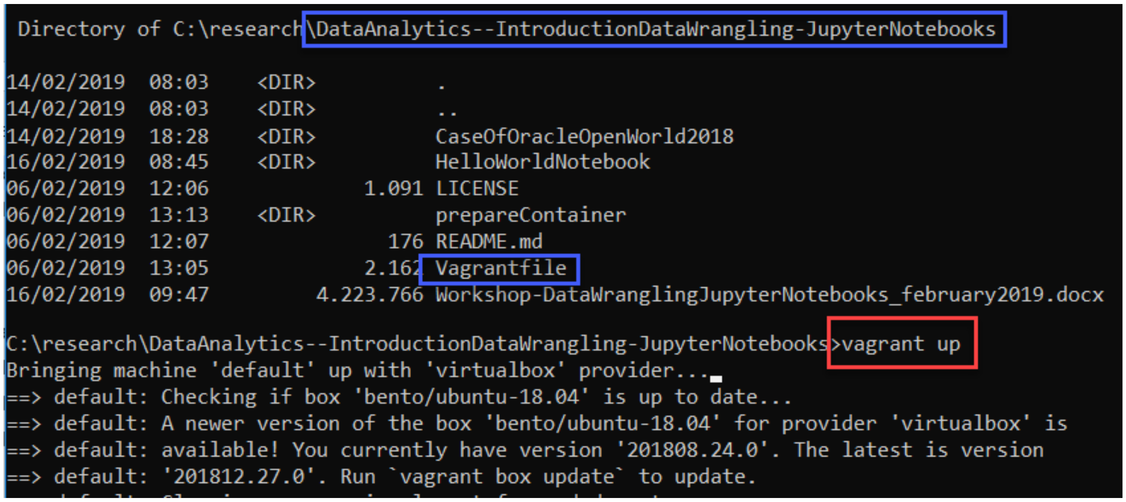
Open a command line window on your laptop, navigate to the local copy root directory of the GitHub repository you have cloned earlier on; this directory should contain the file Vagrantfile.

Now execute

```
vagrant up
```

This statement will launch Vagrant and bring up a new Virtual Machine according to the definition in the Vagrantfile. Feel free to inspect this file.

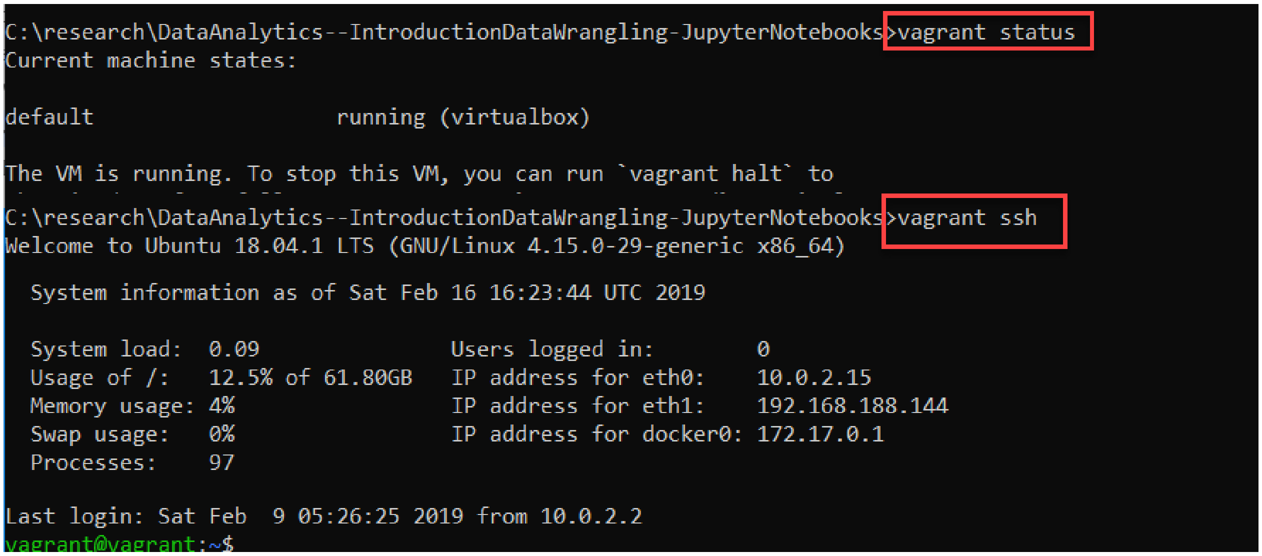
This step will take quite some time – up to 10 minutes or even longer, depending on how much needs to be downloaded and how fast your network connection is. The good thing is: you do not need to do anything. Just sit back and relax.



```
 Directory of C:\research\DataAnalytics--IntroductionDataWrangling-JupyterNotebooks

14/02/2019  08:03    <DIR>          .
14/02/2019  08:03    <DIR>          ..
14/02/2019  18:28    <DIR>          CaseOfOracleOpenWorld2018
16/02/2019  08:45    <DIR>          HelloWorldNotebook
06/02/2019  12:06             1.091 LICENSE
06/02/2019  13:13    <DIR>          prepareContainer
06/02/2019  12:07               176 README.md
06/02/2019  13:05             2.162 Vagrantfile
16/02/2019  09:47         4.223.766 Workshop-DataWranglingJupyterNotebooks_february2019.docx

C:\research\DataAnalytics--IntroductionDataWrangling-JupyterNotebooks>vagrant up
Bringing machine 'default' up with 'virtualbox' provider...
==> default: Checking if box 'bento/ubuntu-18.04' is up to date...
==> default: A newer version of the box 'bento/ubuntu-18.04' for provider 'virtualbox' is
==> default: available! You currently have version '201808.24.0'. The latest is version
==> default: '201812.27.0'. Run `vagrant box update` to update.
```

When this command is complete, the Virtual Machine has been provisioned and is running. You can verify this using *vagrant status*. To open a terminal window in the VM, execute – at the same command line as before or at least from within the same directory that contains the Vagrantfile – this command:

```
vagrant ssh
```

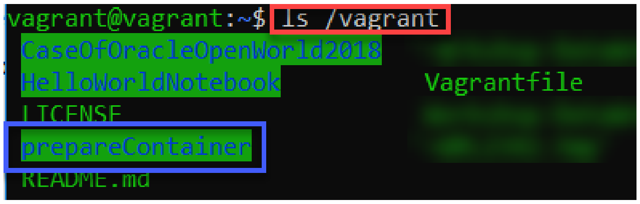


```
C:\research\DataAnalytics--IntroductionDataWrangling-JupyterNotebooks>vagrant status
Current machine states:

default                   running (virtualbox)

The VM is running. To stop this VM, you can run `vagrant halt` to
C:\research\DataAnalytics--IntroductionDataWrangling-JupyterNotebooks>vagrant ssh
Welcome to Ubuntu 18.04.1 LTS (GNU/Linux 4.15.0-29-generic x86_64)

  System information as of Sat Feb 16 16:23:44 UTC 2019

  System load:  0.09              Users logged in:        0
  Usage of /:   12.5% of 61.80GB  IP address for eth0:    10.0.2.15
  Memory usage: 4%                IP address for eth1:    192.168.188.144
  Swap usage:   0%                IP address for docker0: 172.17.0.1
  Processes:    97

Last login: Sat Feb  9 05:26:25 2019 from 10.0.2.2
vagrant@vagrant:~$
```

More details on these steps are shown in this article: https://technology.amis.nl/2018/05/21/rapidly-spinning-up-a-vm-with-ubuntu-and-docker-on-my-windows-machine-using-vagrant-and-virtualbox/.

The directory on your laptop from which you ran the two vagrant commands has been mapped into the VM automatically, mounted at /vagrant. When you check the contents of /vagrant, you should find several files and directories – especially the directory *prepareContainer*:

```
ls /vagrant
```



```
vagrant@vagrant:~$ ls /vagrant
 CaseOfOracleOpenWorld2018          [redacted]
 HelloWorldNotebook                 Vagrantfile
 LICENSE                            [redacted]
 prepareContainer
 README.md
```

Running Jupyter on Docker

Here we are. You can run Docker containers. But nothing in your environment is yet Jupyter Notebook or even Python specific. Good. Let's change that.

I will assume that you are at the command line with Linux at your fingertips and Docker running in the backgroud. docker ps needs to return nothing at this point.

Many container images are available that contain Jupyter Notebooks in some form or shape. I will use the jupyter/scipy -notebook image from Jupyter Docker Stacks.The jupyter/scipy -notebook image is fairly rich image. It contains:

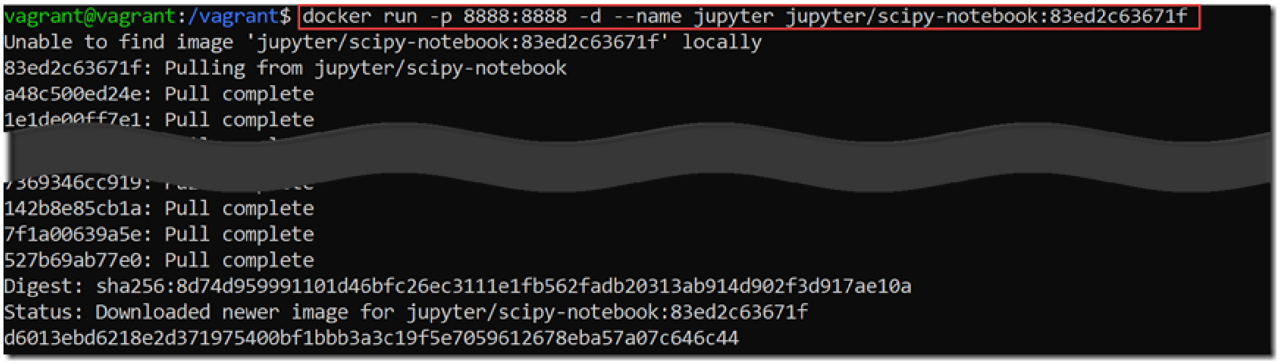
- A minimally-functional Jupyter Notebook server
- Miniconda Python 3.6
- Pandoc and TeX Live for notebook document conversion
- git, emacs, jed, nano, and unzip
- pandas, numexpr, matplotlib, scipy, seaborn, scikit-learn, scikit-image, sympy, cython, patsy, statsmodel, cloudpickle, dill, numba, bokeh, sqlalchemy, hdf5, vincent, beautifulsoup, protobuf, and xlrd packages
- ipywidgets for interactive visualizations in Python notebooks
- Facets for visualizing machine learning datasets

We will add a few other libraries on this 'base' image.

As first step: run a Docker container based on the image

```
docker run -p 8888:8888 -d --name jupyter jupyter/scipy-notebook:83ed2c63671f
```

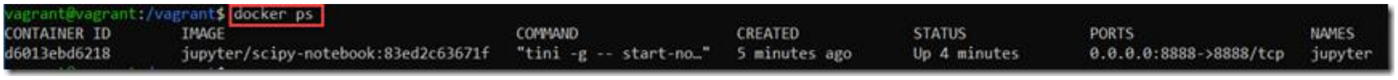
Note: the Docker image tag (id) is no strictly necessary; if you strip it off (jupyter/scipy-notebook) you will get the latest – which may do everything you need. This particular id is from early February 2019 and it seems to work for me. See all Docker image tags: https://hub.docker.com/r/jupyter/scipy-notebook/tags/ .



```
vagrant@vagrant:/vagrant$ docker run -p 8888:8888 -d --name jupyter jupyter/scipy-notebook:83ed2c63671f
Unable to find image 'jupyter/scipy-notebook:83ed2c63671f' locally
83ed2c63671f: Pulling from jupyter/scipy-notebook
a48c500ed24e: Pull complete
1e1de00ff7e1: Pull complete
7369346cc919:
142b8e85cb1a: Pull complete
7f1a00639a5e: Pull complete
527b69ab77e0: Pull complete
Digest: sha256:8d74d959991101d46bfc26ec3111e1fb562fadb20313ab914d902f3d917ae10a
Status: Downloaded newer image for jupyter/scipy-notebook:83ed2c63671f
d6013ebd6218e2d371975400bf1bbb3a3c19f5e7059612678eba57a07c646c44
```

The container image is quite sizable – close to 2 GB. Downloading is bound to take a while – depending on the network capacity you can leverage.

When downloading and extracting is complete for all layers, the container will be running. It exposes port 8888. The Jupyter server is accessible at that port.

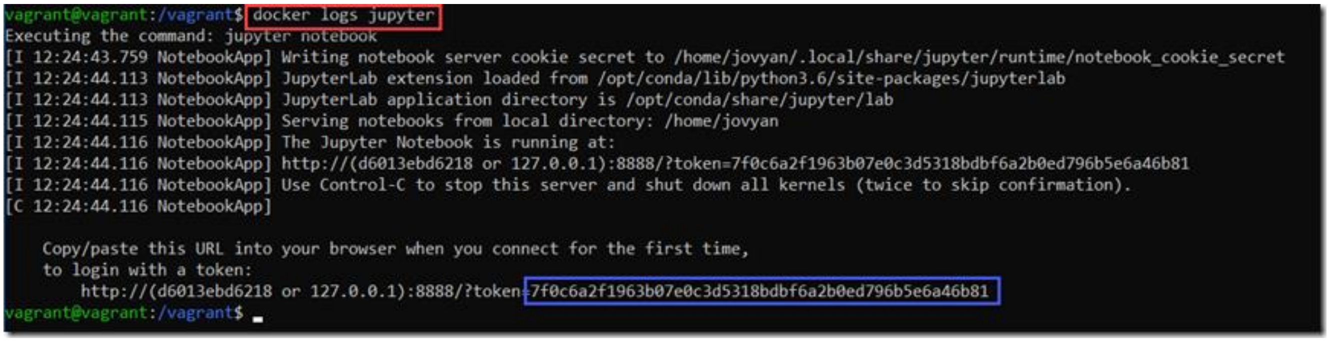


```
vagrant@vagrant:/vagrant$ docker ps
CONTAINER ID        IMAGE                                 COMMAND                  CREATED             STATUS              PORTS                    NAMES
d6013ebd6218        jupyter/scipy-notebook:83ed2c63671f   "tini -g -- start-no…"   5 minutes ago       Up 4 minutes        0.0.0.0:8888->8888/tcp   jupyter
```

Access the Jupyter Notebook environment from a browser on your laptop; the endpoint depends on the IP address of the host running the Docker container. In my case, using the Vagrant file in the GitHup repo associated with this article, I will access the Jupyter Notebook at: http://192.168.188.144:8888 .

The Jupyter server will prompt you for a token – to ensure not just anyone can access the environment. Back on the command line, execute this statement:

```
docker logs jupyter
```

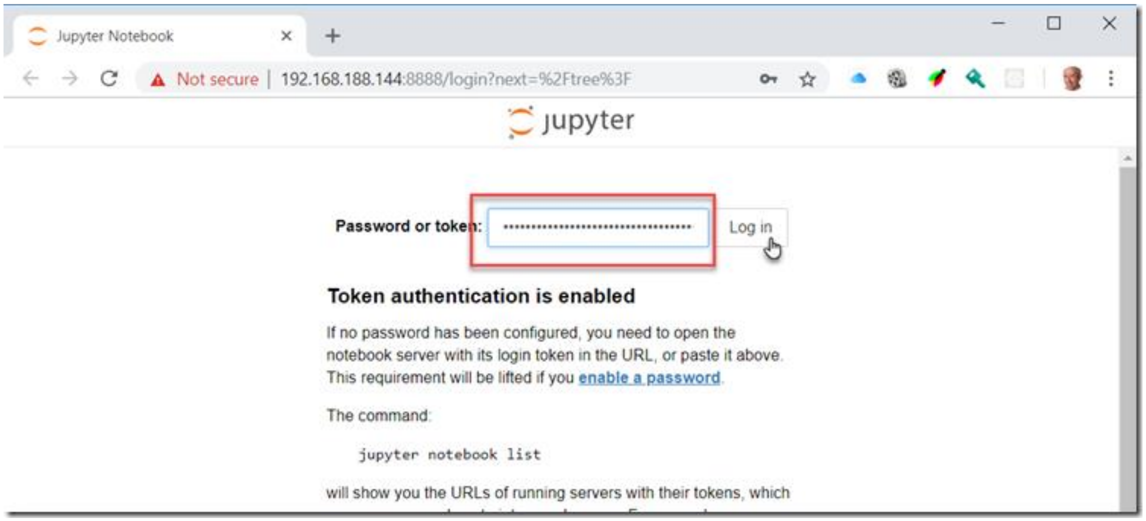


```
vagrant@vagrant:/vagrant$ docker logs jupyter
Executing the command: jupyter notebook
[I 12:24:43.759 NotebookApp] Writing notebook server cookie secret to /home/jovyan/.local/share/jupyter/runtime/notebook_cookie_secret
[I 12:24:44.113 NotebookApp] JupyterLab extension loaded from /opt/conda/lib/python3.6/site-packages/jupyterlab
[I 12:24:44.113 NotebookApp] JupyterLab application directory is /opt/conda/share/jupyter/lab
[I 12:24:44.115 NotebookApp] Serving notebooks from local directory: /home/jovyan
[I 12:24:44.116 NotebookApp] The Jupyter Notebook is running at:
[I 12:24:44.116 NotebookApp] http://(d6013ebd6218 or 127.0.0.1):8888/?token=7f0c6a2f1963b07e0c3d5318bdbf6a2b0ed796b5e6a46b81
[I 12:24:44.116 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 12:24:44.116 NotebookApp]

    Copy/paste this URL into your browser when you connect for the first time,
    to login with a token:
        http://(d6013ebd6218 or 127.0.0.1):8888/?token=7f0c6a2f1963b07e0c3d5318bdbf6a2b0ed796b5e6a46b81
vagrant@vagrant:/vagrant$ _
```

This will show something like:

```
Copy/paste this URL into your browser when you connect for the first time,

to login with a token:

http://localhost:8888/?token=f89b02dd78479d52470b3c3a797408b20cc5a11e067e94b8
```

The token is the value behind `/?token=`. You need that for logging in.



Password or token: ••

Token authentication is enabled

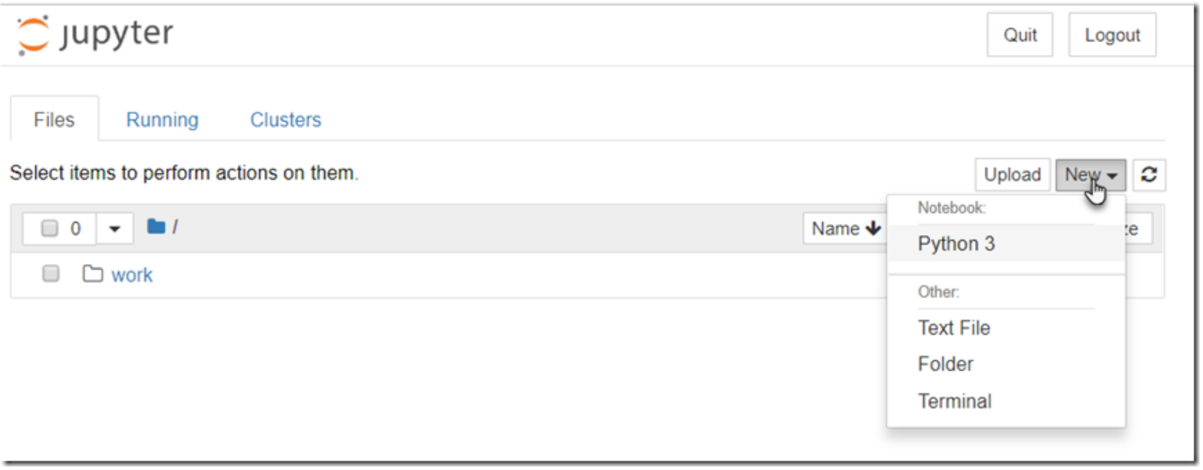
If no password has been configured, you need to open the notebook server with its login token in the URL, or paste it above. This requirement will be lifted if you enable a password.

The command:

```
jupyter notebook list
```

will show you the URLs of running servers with their tokens, which

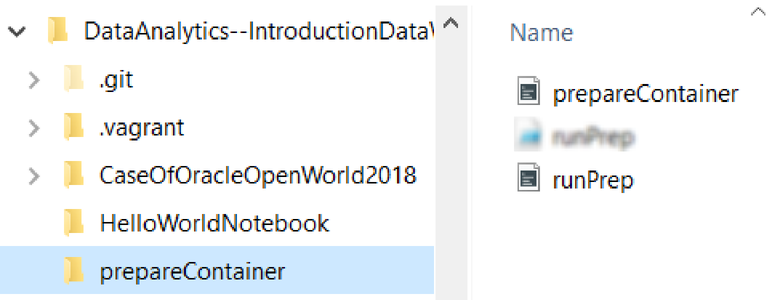
After pasting the token, click on the Log In button:



Select items to perform actions on them.

At this point, you can start creating your own notebook or upload a notebook from your laptop's file system. The container currently does not contain any Jupyter Notebooks that we can open and run. We will change this now.

The GitHub repo for this handson that you cloned a little while back has a folder *prepareContainer* that contains two *sh* scripts.



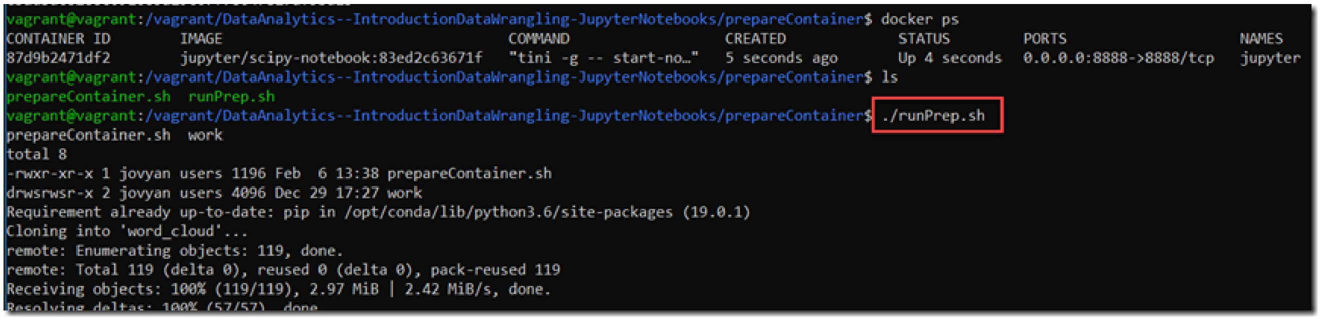
The script *runPrep.sh* copies the script *prepareContainer.sh* into the container and executes it – to install some packages and git clone a few notebooks.

Execute these steps – on the command line of your Docker host – likely from directory /vagrant :

```
cd prepareContainer
```

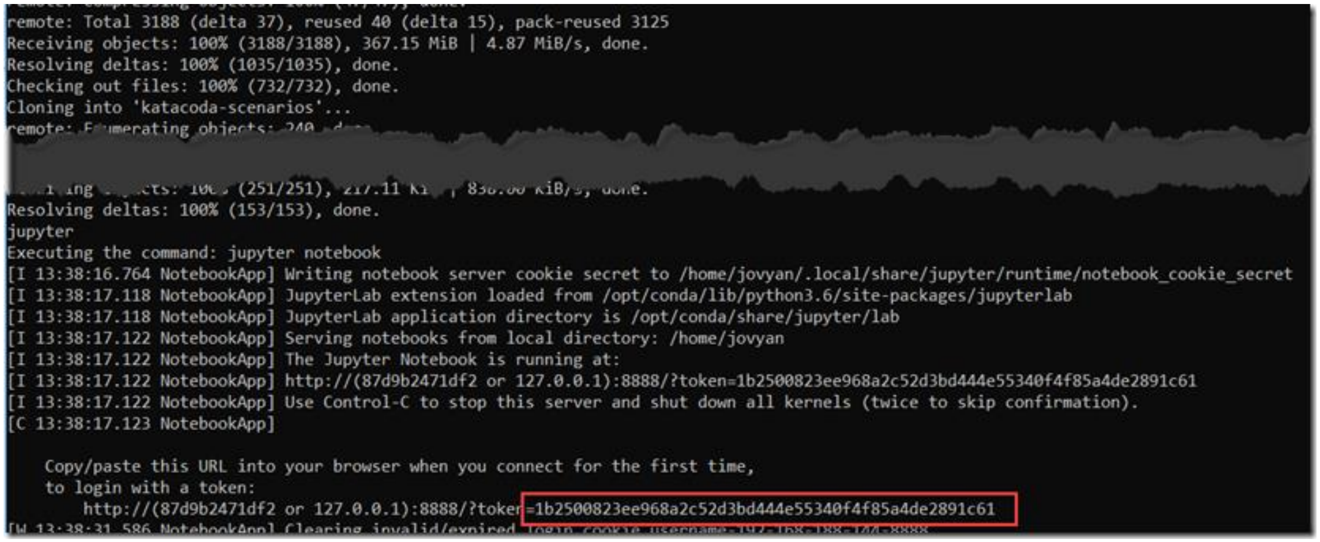
```
./runPrep.sh
```

The script prepareContainer.sh is copied into the container and made executable. Then it is executed. It installs various Python packages using *pip* and git clones Jupyter notebooks into the container.



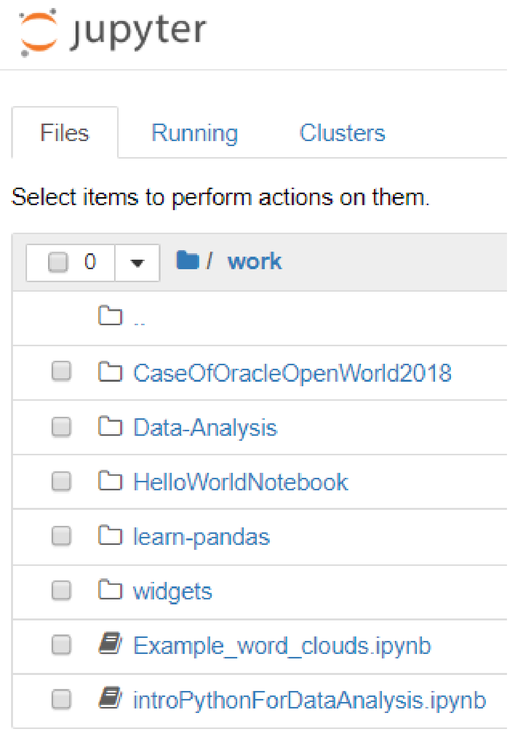
```
vagrant@vagrant:/vagrant/DataAnalytics--IntroductionDataWrangling-JupyterNotebooks/prepareContainer$ docker ps
CONTAINER ID        IMAGE                                COMMAND                  CREATED             STATUS          PORTS                    NAMES
87d9b2471df2        jupyter/scipy-notebook:83ed2c63671f  "tini -g -- start-no…"   5 seconds ago       Up 4 seconds    0.0.0.0:8888->8888/tcp   jupyter
vagrant@vagrant:/vagrant/DataAnalytics--IntroductionDataWrangling-JupyterNotebooks/prepareContainer$ ls
prepareContainer.sh  runPrep.sh
vagrant@vagrant:/vagrant/DataAnalytics--IntroductionDataWrangling-JupyterNotebooks/prepareContainer$ ./runPrep.sh
prepareContainer.sh  work
total 8
-rwxr-xr-x 1 jovyan users 1196 Feb  6 13:38 prepareContainer.sh
drwsrwsr-x 2 jovyan users 4096 Dec 29 17:27 work
Requirement already up-to-date: pip in /opt/conda/lib/python3.6/site-packages (19.0.1)
Cloning into 'word_cloud'...
remote: Enumerating objects: 119, done.
remote: Total 119 (delta 0), reused 0 (delta 0), pack-reused 119
Receiving objects: 100% (119/119), 2.97 MiB | 2.42 MiB/s, done.
Resolving deltas: 100% (57/57), done.
```

When the actions inside the container are done – note: this can take a few minutes – the container is restarted to have the Jupyter Notebook server pick up all changes. You may need a new token from the restarted server to login to the Jupyter Notebook environment in the browser.



```
remote: Total 3188 (delta 37), reused 40 (delta 15), pack-reused 3125
Receiving objects: 100% (3188/3188), 367.15 MiB | 4.87 MiB/s, done.
Resolving deltas: 100% (1035/1035), done.
Checking out files: 100% (732/732), done.
Cloning into 'katacoda-scenarios'...
remote: Enumerating objects: 240, done.

Receiving objects: 100% (251/251), 217.11 KiB | 836.00 KiB/s, done.
Resolving deltas: 100% (153/153), done.
jupyter
Executing the command: jupyter notebook
[I 13:38:16.764 NotebookApp] Writing notebook server cookie secret to /home/jovyan/.local/share/jupyter/runtime/notebook_cookie_secret
[I 13:38:17.118 NotebookApp] JupyterLab extension loaded from /opt/conda/lib/python3.6/site-packages/jupyterlab
[I 13:38:17.118 NotebookApp] JupyterLab application directory is /opt/conda/share/jupyter/lab
[I 13:38:17.122 NotebookApp] Serving notebooks from local directory: /home/jovyan
[I 13:38:17.122 NotebookApp] The Jupyter Notebook is running at:
[I 13:38:17.122 NotebookApp] http://(87d9b2471df2 or 127.0.0.1):8888/?token=1b2500823ee968a2c52d3bd444e55340f4f85a4de2891c61
[I 13:38:17.122 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 13:38:17.123 NotebookApp]

    Copy/paste this URL into your browser when you connect for the first time,
    to login with a token:
        http://(87d9b2471df2 or 127.0.0.1):8888/?token=1b2500823ee968a2c52d3bd444e55340f4f85a4de2891c61
[W 13:38:31.586 NotebookApp] Clearing invalid/expired login cookie username-192-168-188-144-8888
```

When you next enter the Jupyter Notebook environment in the browser, you will see a number of notebooks that were not there before.



Select items to perform actions on them.

□ 0 ▾ / work

- ..
- □ CaseOfOracleOpenWorld2018
- □ Data-Analysis
- □ HelloWorldNotebook
- □ learn-pandas
- □ widgets
- □ Example_word_clouds.ipynb
- □ introPythonForDataAnalysis.ipynb

For example: open and run introPythonForDataAnalysis.ipynb in the work folder. Or open and run Example_word_clouds.ipynb.

Appendix B - Run Docker using Docker for Windows

This instruction assumes that you are able to run Docker container from the Windows command line. And that you have opened a Powershell command line terminal. Navigate to the root of the GitHub repository clone – the directory that contains the file README.md and a subdirectory *prepareContainer*.

Nothing in your environment is yet Jupyter Notebook or even Python specific. Good. Let's change that.

Many container images are available that contain Jupyter Notebooks in some form or shape. I will use the jupyter/scipy -notebook image from Jupyter Docker Stacks.The jupyter/scipy -notebook image is fairly rich image. It contains:

- A minimally-functional Jupyter Notebook server
- Miniconda Python 3.6
- Pandoc and TeX Live for notebook document conversion
- git, emacs, jed, nano, and unzip
- pandas, numexpr, matplotlib, scipy, seaborn, scikit-learn, scikit-image, sympy, cython, patsy, statsmodel, cloudpickle, dill, numba, bokeh, sqlalchemy, hdf5, vincent, beautifulsoup, protobuf, and xlrd packages
- ipywidgets for interactive visualizations in Python notebooks
- Facets for visualizing machine learning datasets

We will add a few other libraries on this 'base' image.

As first step: run a Docker container based on the image

```
docker run -p 8888:8888 -d --name jupyter jupyter/scipy-notebook:83ed2c63671f
```

Note: the Docker image tag (id) is no strictly necessary; if you strip it off (jupyter/scipy-notebook) you will get the latest – which may do everything you need. This particular id is from early February 2019 and it seems to work for me. See all Docker image tags: https://hub.docker.com/r/jupyter/scipy-notebook/tags/ .

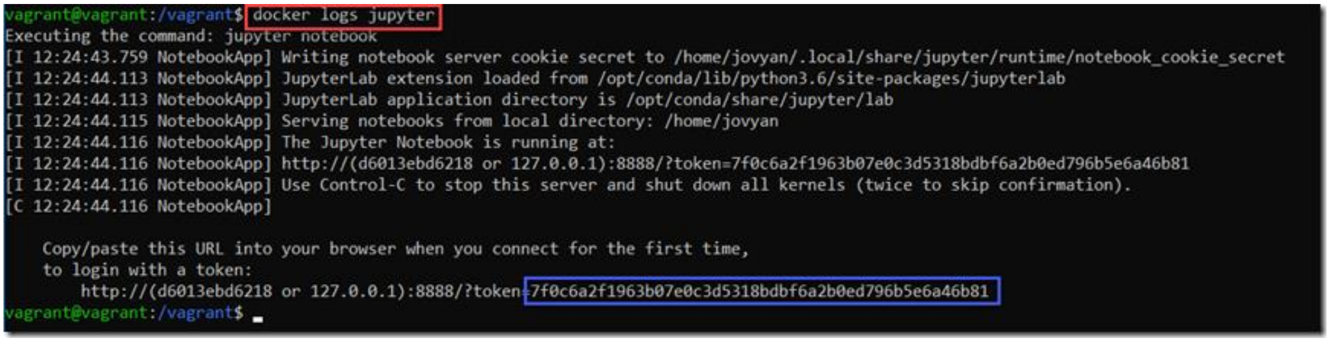
The container image is quite sizable – close to 2 GB. Downloading is bound to take a while – depending on the network capacity you can leverage.

When downloading and extracting is complete for all layers, the container will be running. It exposes port 8888. The Jupyter server is accessible at that port.

Access the Jupyter Notebook environment from a browser; with Docker for Windows, you can probably just point your browser a http://localhost:8888 .

The Jupyter server will prompt you for a token – to ensure not just anyone can access the environment. Back on the command line, execute this statement:

```
docker logs jupyter
```

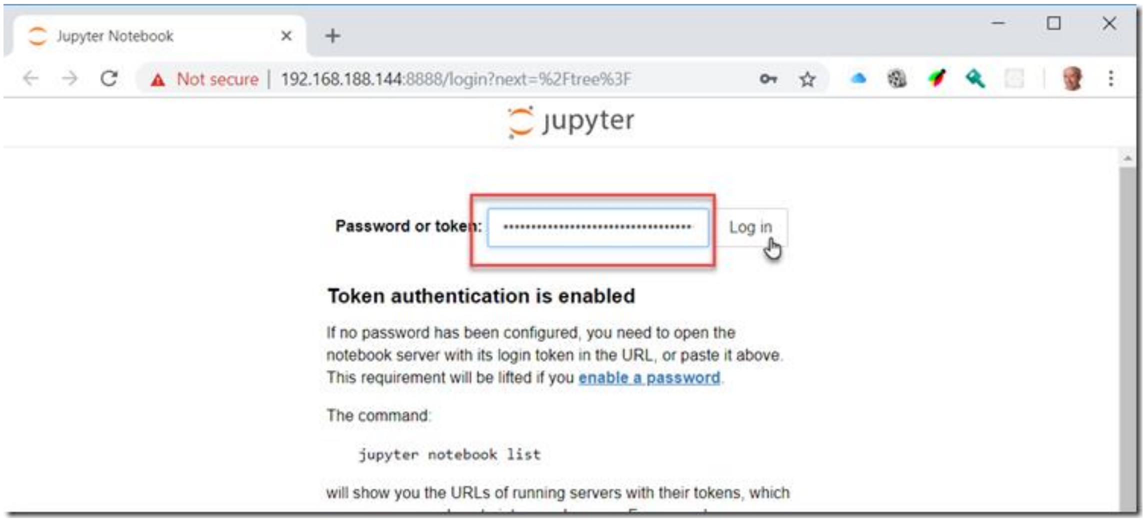


```
vagrant@vagrant:/vagrant$ docker logs jupyter
Executing the command: jupyter notebook
[I 12:24:43.759 NotebookApp] Writing notebook server cookie secret to /home/jovyan/.local/share/jupyter/runtime/notebook_cookie_secret
[I 12:24:44.113 NotebookApp] JupyterLab extension loaded from /opt/conda/lib/python3.6/site-packages/jupyterlab
[I 12:24:44.113 NotebookApp] JupyterLab application directory is /opt/conda/share/jupyter/lab
[I 12:24:44.115 NotebookApp] Serving notebooks from local directory: /home/jovyan
[I 12:24:44.116 NotebookApp] The Jupyter Notebook is running at:
[I 12:24:44.116 NotebookApp] http://(d6013ebd6218 or 127.0.0.1):8888/?token=7f0c6a2f1963b07e0c3d5318bdbf6a2b0ed796b5e6a46b81
[I 12:24:44.116 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 12:24:44.116 NotebookApp]

    Copy/paste this URL into your browser when you connect for the first time,
    to login with a token:
        http://(d6013ebd6218 or 127.0.0.1):8888/?token=7f0c6a2f1963b07e0c3d5318bdbf6a2b0ed796b5e6a46b81
vagrant@vagrant:/vagrant$ _
```

This will show something like:

```
        Copy/paste this URL into your browser when you connect for the first time,

        to login with a token:

        http://localhost:8888/?token=f89b02dd78479d52470b3c3a797408b20cc5a11e067e94b8
```

The token is the value behind `/?token=`. You need that for logging in.



Token authentication is enabled

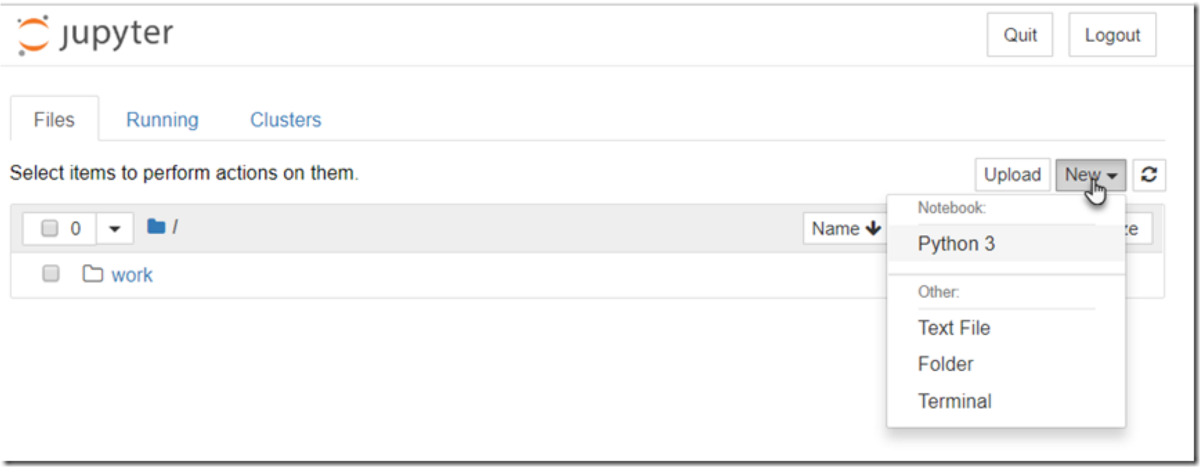
If no password has been configured, you need to open the notebook server with its login token in the URL, or paste it above. This requirement will be lifted if you enable a password.

The command:

```
jupyter notebook list
```

will show you the URLs of running servers with their tokens, which

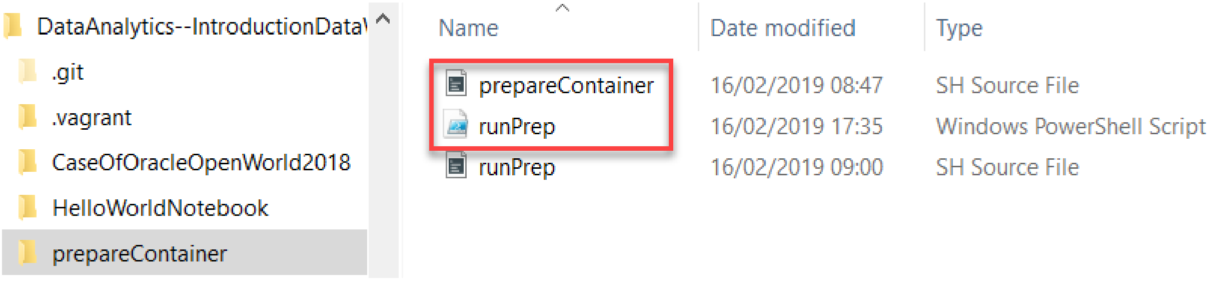
After pasting the token, click on the Log In button:



Select items to perform actions on them.

At this point, you can start creating your own notebook or upload a notebook from your laptop's file system. The container currently does not contain any Jupyter Notebooks that we can open and run. We will change this now.

The GitHub repo for this handson that you cloned a little while back has a folder *prepareContainer* that contains two *sh* scripts and a Powershell script.



Name	Date modified	Type
prepareContainer	16/02/2019 08:47	SH Source File
runPrep	16/02/2019 17:35	Windows PowerShell Script
runPrep	16/02/2019 09:00	SH Source File

The script *runPrep.ps1* copies the script *prepareContainer.sh* into the container and executes it – to install some packages and git clone a few notebooks.

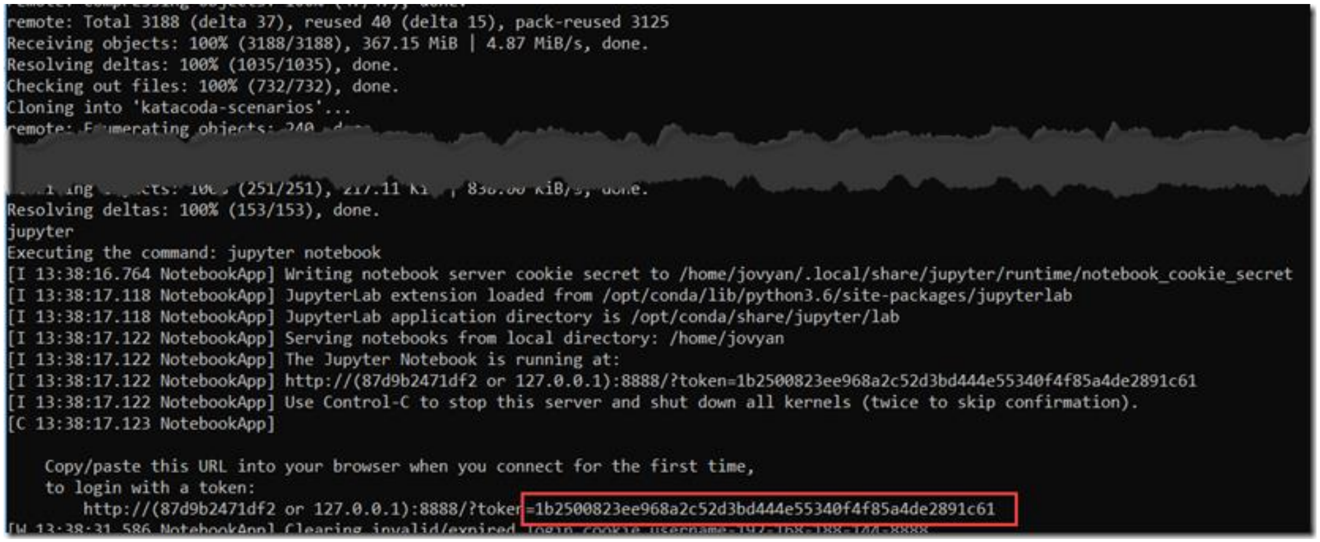
Execute these steps:

```
cd prepareContainer
```

```
./runPrep.ps1
```

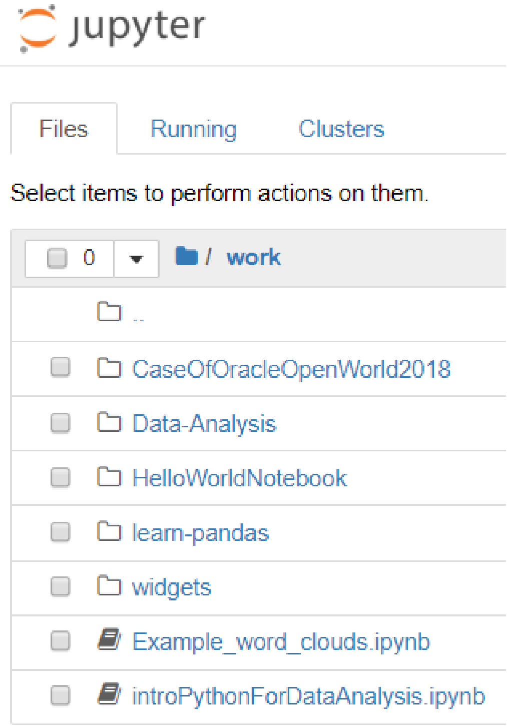
The script prepareContainer.sh is copied into the container and made executable. Then it is executed. It installs various Python packages using *pip* and git clones Jupyter notebooks into the container.

When the actions inside the container are done – note: this can take a few minutes – the container is restarted to have the Jupyter Notebook server pick up all changes. You may need a new token from the restarted server to login to the Jupyter Notebook environment in the browser.



```
remote: Compressing objects: 100% (47/47), done.
remote: Total 3188 (delta 37), reused 40 (delta 15), pack-reused 3125
Receiving objects: 100% (3188/3188), 367.15 MiB | 4.87 MiB/s, done.
Resolving deltas: 100% (1035/1035), done.
Checking out files: 100% (732/732), done.
Cloning into 'katacoda-scenarios'...
remote: Enumerating objects: 240, done.

Receiving objects: 100% (251/251), 217.11 KiB | 838.00 KiB/s, done.
Resolving deltas: 100% (153/153), done.
jupyter
Executing the command: jupyter notebook
[I 13:38:16.764 NotebookApp] Writing notebook server cookie secret to /home/jovyan/.local/share/jupyter/runtime/notebook_cookie_secret
[I 13:38:17.118 NotebookApp] JupyterLab extension loaded from /opt/conda/lib/python3.6/site-packages/jupyterlab
[I 13:38:17.118 NotebookApp] JupyterLab application directory is /opt/conda/share/jupyter/lab
[I 13:38:17.122 NotebookApp] Serving notebooks from local directory: /home/jovyan
[I 13:38:17.122 NotebookApp] The Jupyter Notebook is running at:
[I 13:38:17.122 NotebookApp] http://(87d9b2471df2 or 127.0.0.1):8888/?token=1b2500823ee968a2c52d3bd444e55340f4f85a4de2891c61
[I 13:38:17.122 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 13:38:17.123 NotebookApp]

    Copy/paste this URL into your browser when you connect for the first time,
    to login with a token:
        http://(87d9b2471df2 or 127.0.0.1):8888/?token=1b2500823ee968a2c52d3bd444e55340f4f85a4de2891c61
[W 13:38:31.586 NotebookApp] Clearing invalid/expired login cookie username-192-168-188-144-8888
```

When you next enter the Jupyter Notebook environment in the browser, you will see a number of notebooks that were not there before.



Select items to perform actions on them.

- □ 0 / work
- ..
- □ CaseOfOracleOpenWorld2018
- □ Data-Analysis
- □ HelloWorldNotebook
- □ learn-pandas
- □ widgets
- □ Example_word_clouds.ipynb
- □ introPythonForDataAnalysis.ipynb

For example: open and run introPythonForDataAnalysis.ipynb in the work folder. Or open and run Example_word_clouds.ipynb.

Appendix C - Cloud based Jupyter Notebook Playground using Katacoda

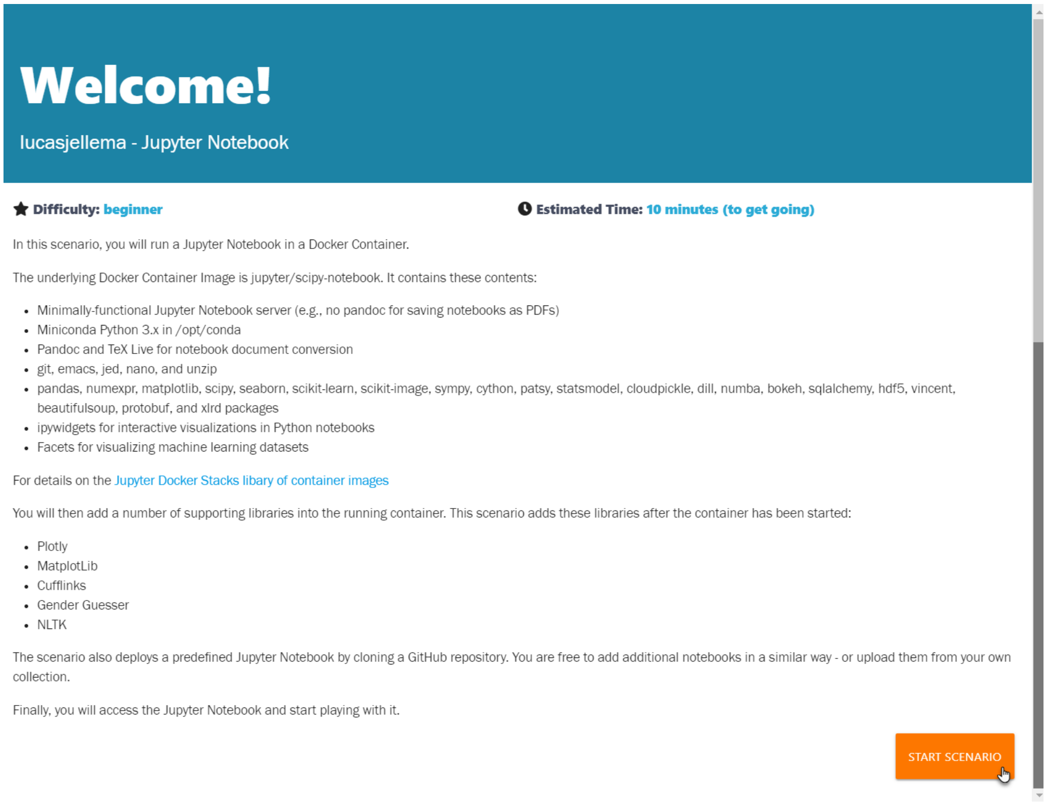
Katacoda is an online platform that offers hundreds of scenarios and sandbox environments to learn about and play with different kinds of technologies. Katacoda is special in that it not only offers the handson instructions – it also provides the runtime environment in which these steps can be executed instantly. Examples of such environments: Linux server, Docker engine, Kubernetes cluster and almost anything that can be run as container. I have created a Katacoda scenario for Jupyter Notebook – with a number of handpicked Python libraries and a Notebook git cloned from GitHub; with all the resources we need for our workshop.

In order to get your running Jupyter Notebook environment, you have to take these steps:

1. Go to https://www.katacoda.com and sign up – using your GitHub account.

2. Open your browser at https://www.katacoda.com/lucasjellema/scenarios/jupyter-notebook

You will come to a page that looks like this figure:



Welcome!

lucasjellema - Jupyter Notebook

★ **Difficulty:** **beginner**

🕒 **Estimated Time:** **10 minutes (to get going)**

In this scenario, you will run a Jupyter Notebook in a Docker Container.

The underlying Docker Container Image is jupyter/scipy-notebook. It contains these contents:

- Minimally-functional Jupyter Notebook server (e.g., no pandoc for saving notebooks as PDFs)
- Miniconda Python 3.x in /opt/conda
- Pandoc and TeX Live for notebook document conversion
- git, emacs, jed, nano, and unzip
- pandas, numexpr, matplotlib, scipy, seaborn, scikit-learn, scikit-image, sympy, cython, patsy, statsmodel, cloudpickle, dill, numba, bokeh, sqlalchemy, hdf5, vincent, beautifulsoup, protobuf, and xlrd packages
- ipywidgets for interactive visualizations in Python notebooks
- Facets for visualizing machine learning datasets

For details on the Jupyter Docker Stacks libary of container images

You will then add a number of supporting libraries into the running container. This scenario adds these libraries after the container has been started:

- Plotly
- MatplotLib
- Cufflinks
- Gender Guesser
- NLTK

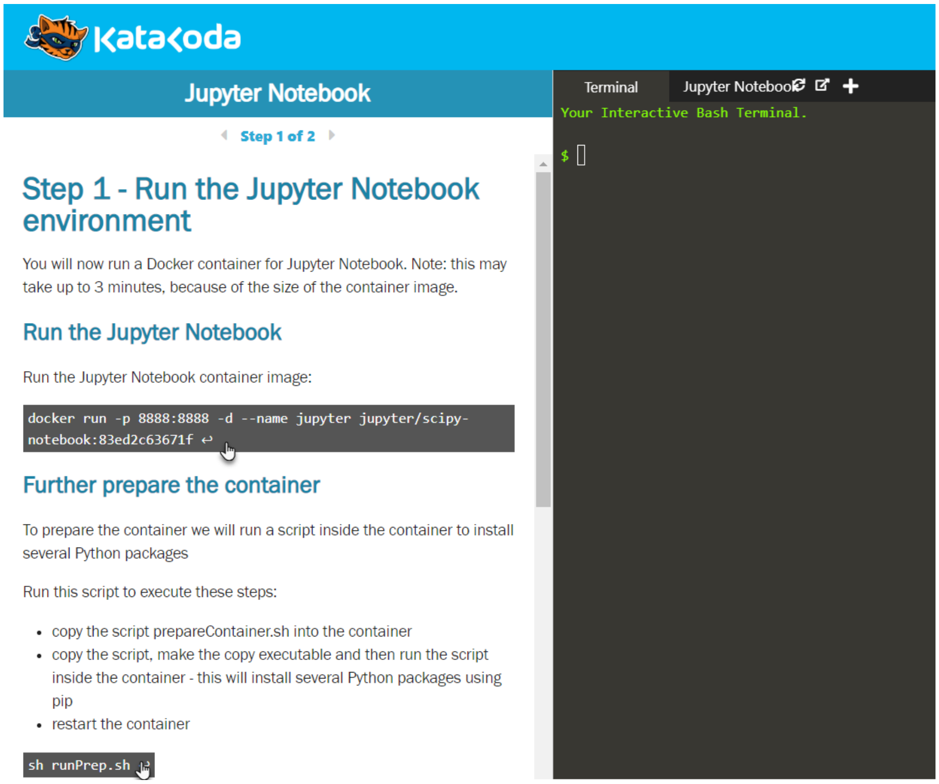
The scenario also deploys a predefined Jupyter Notebook by cloning a GitHub repository. You are free to add additional notebooks in a similar way - or upload them from your own collection.

Finally, you will access the Jupyter Notebook and start playing with it.

3. Click on the button *Start Scenario*

4. Follow the instructions in Step 1

The first step in the scenario looks as shown in the next figure.



Jupyter Notebook

Step 1 - Run the Jupyter Notebook environment

You will now run a Docker container for Jupyter Notebook. Note: this may take up to 3 minutes, because of the size of the container image.

Run the Jupyter Notebook

Run the Jupyter Notebook container image:

```
docker run -p 8888:8888 -d --name jupyter jupyter/scipy-notebook:83ed2c63671f ↩
```

Further prepare the container

To prepare the container we will run a script inside the container to install several Python packages

Run this script to execute these steps:

- copy the script prepareContainer.sh into the container
- copy the script, make the copy executable and then run the script inside the container - this will install several Python packages using pip
- restart the container

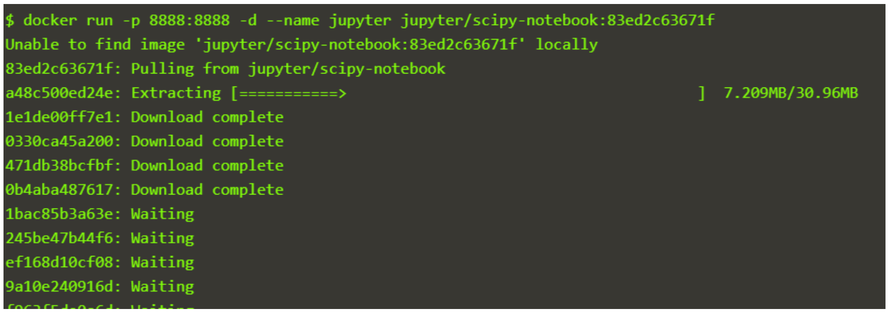
```
sh runPrep.sh ↩
```

```
Your Interactive Bash Terminal.

$
```

You have to click twice – to have two key commands executed that will configure your environment:

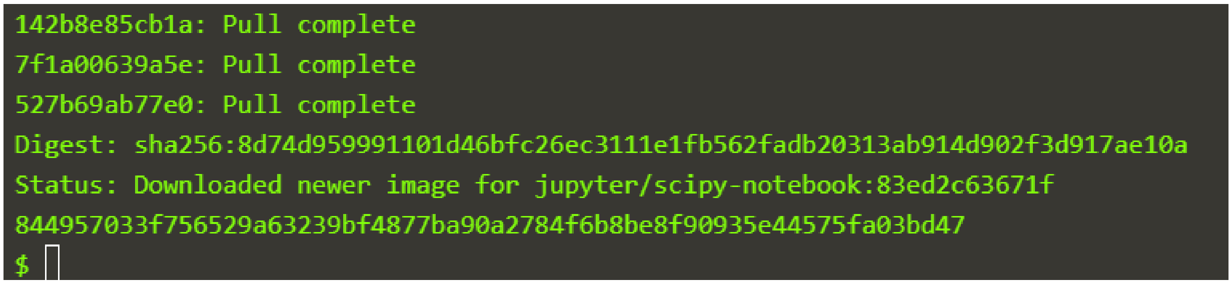
1. Click on the first dark gray area that contains a command starting with *docker run*. By clicking on this command, you cause it to be copied to the terminal window shown on the right where it is immediately executed.

   This command instructs the Docker container engine to start the indicated container image jupyter/scipy-notebook. Because that image is not available yet, it needs to be downloaded. This will take some time (the download size is around 2 GB) – typically 2-4 minutes.

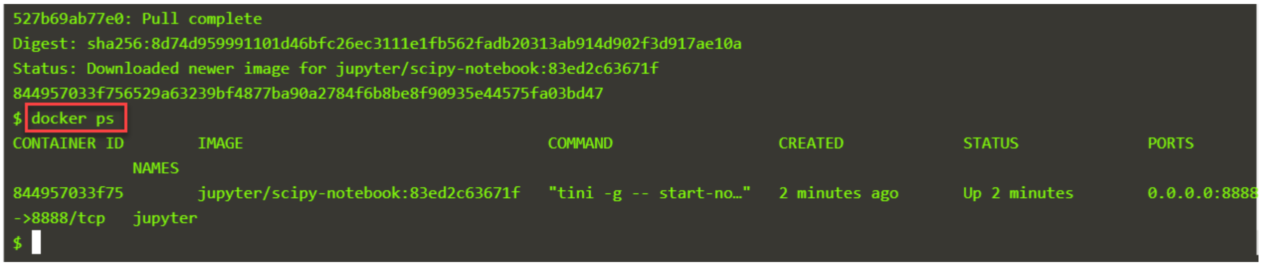
   ```
   $ docker run -p 8888:8888 -d --name jupyter jupyter/scipy-notebook:83ed2c63671f
   Unable to find image 'jupyter/scipy-notebook:83ed2c63671f' locally
   83ed2c63671f: Pulling from jupyter/scipy-notebook
   a48c500ed24e: Extracting [===========>                                       ]  7.209MB/30.96MB
   1e1de00ff7e1: Download complete
   0330ca45a200: Download complete
   471db38bcfbf: Download complete
   0b4aba487617: Download complete
   1bac85b3a63e: Waiting
   245be47b44f6: Waiting
   ef168d10cf08: Waiting
   9a10e240916d: Waiting
   ```

   When the container is running – the terminal will show something like this:



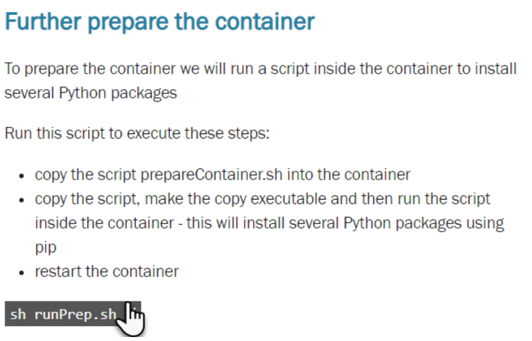
```
142b8e85cb1a: Pull complete
7f1a00639a5e: Pull complete
527b69ab77e0: Pull complete
Digest: sha256:8d74d959991101d46bfc26ec3111e1fb562fadb20313ab914d902f3d917ae10a
Status: Downloaded newer image for jupyter/scipy-notebook:83ed2c63671f
844957033f756529a63239bf4877ba90a2784f6b8be8f90935e44575fa03bd47
$ 
```

If you feel brave, you could type a little something into the terminal window, for example: *docker ps*. This requests the Docker engine to show a list of all running containers:



```
527b69ab77e0: Pull complete
Digest: sha256:8d74d959991101d46bfc26ec3111e1fb562fadb20313ab914d902f3d917ae10a
Status: Downloaded newer image for jupyter/scipy-notebook:83ed2c63671f
844957033f756529a63239bf4877ba90a2784f6b8be8f90935e44575fa03bd47
$ docker ps
CONTAINER ID        IMAGE                                 COMMAND                  CREATED             STATUS              PORTS
            NAMES
844957033f75        jupyter/scipy-notebook:83ed2c63671f   "tini -g -- start-no…"   2 minutes ago       Up 2 minutes        0.0.0.0:8888
->8888/tcp   jupyter
$ 
```

2. Click on the second gray area – containing a command *sh runPrep.sh*. This command will copy and execute a script. This script will do a number of things. This all contributes to making sure the container contains all required resources. Note: running this script will also take considerable time – another 2-4 minutes is typical.



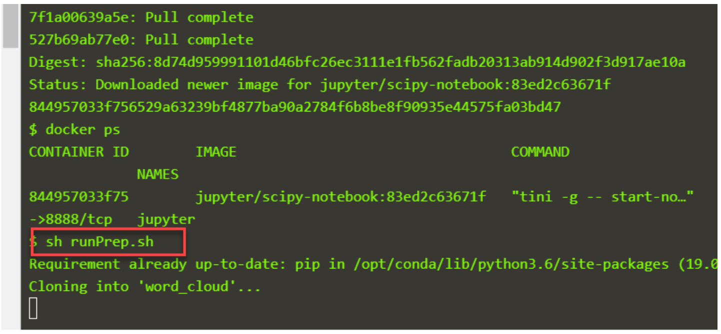
Further prepare the container

To prepare the container we will run a script inside the container to install several Python packages

Run this script to execute these steps:

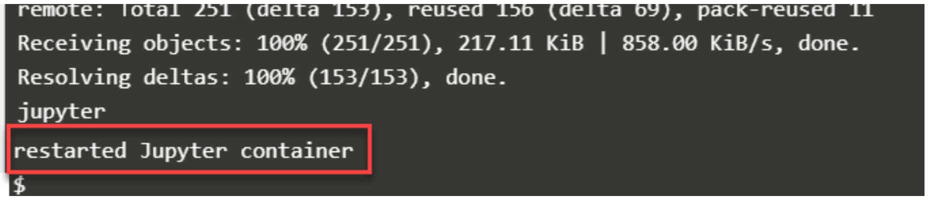
- copy the script prepareContainer.sh into the container
- copy the script, make the copy executable and then run the script inside the container - this will install several Python packages using pip
- restart the container

```
sh runPrep.sh
```



```
7f1a00639a5e: Pull complete
527b69ab77e0: Pull complete
Digest: sha256:8d74d959991101d46bfc26ec3111e1fb562fadb20313ab914d902f3d917ae10a
Status: Downloaded newer image for jupyter/scipy-notebook:83ed2c63671f
844957033f756529a63239bf4877ba90a2784f6b8be8f90935e44575fa03bd47
$ docker ps
CONTAINER ID        IMAGE                                 COMMAND
            NAMES
844957033f75        jupyter/scipy-notebook:83ed2c63671f   "tini -g -- start-no…"
->8888/tcp   jupyter
$ sh runPrep.sh
Requirement already up-to-date: pip in /opt/conda/lib/python3.6/site-packages (19.0
Cloning into 'word_cloud'...
```

When the script is complete – it will restart the Docker container to ensure all changes take effect.

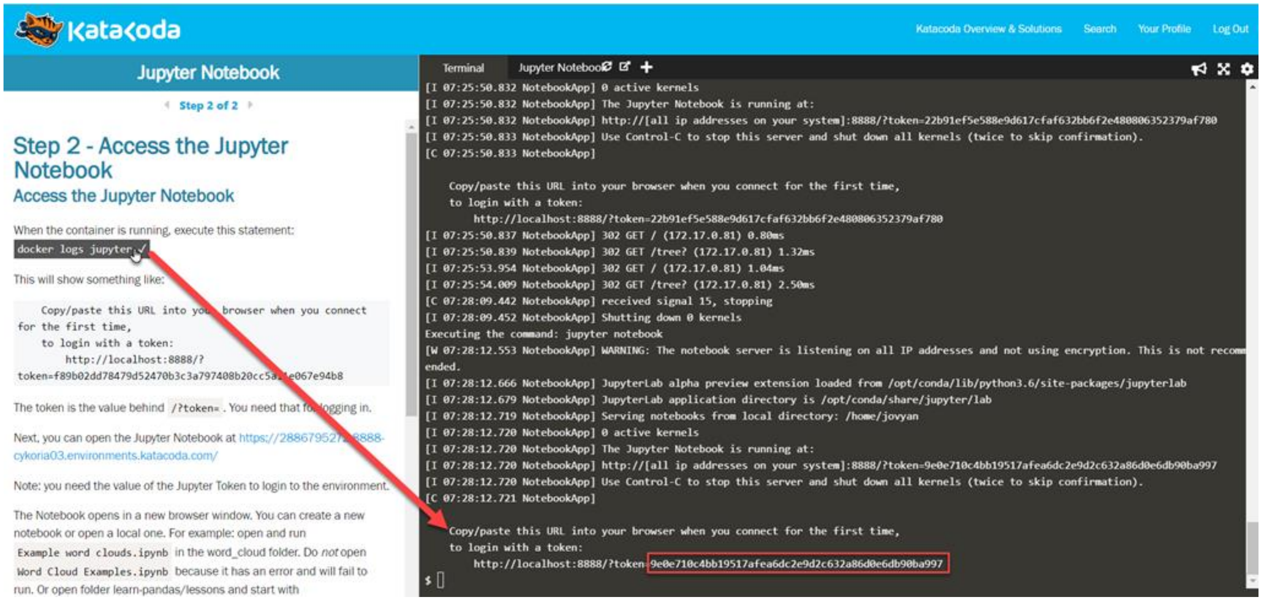


```
remote: Total 251 (delta 153), reused 156 (delta 69), pack-reused 11
Receiving objects: 100% (251/251), 217.11 KiB | 858.00 KiB/s, done.
Resolving deltas: 100% (153/153), done.
jupyter
restarted Jupyter container
$
```

Now you can proceed to ***step 2*** in the scenario.

5. Perform the tasks in Step 2 in the Katacoda scenario

In Step 2, you will access the Jupyter Notebook server that is running at this point. In order to access the environment in a browser, we need a token. We can read the token from the log-files produced in the container. Click on the next code block – that contains *docker logs jupyter* -- to retrieve these logs. The logs are shown in the terminal and they will contain the token we need:



Step 2 - Access the Jupyter Notebook

Access the Jupyter Notebook

When the container is running, execute this statement:

```
docker logs jupyter
```

This will show something like:

```
    Copy/paste this URL into your browser when you connect
for the first time,
    to login with a token:
        http://localhost:8888/?
token=f89b02dd78479d52470b3c3a797408b20cc5a1e067e94b8
```

The token is the value behind `/?token=`. You need that for logging in.

Next, you can open the Jupyter Notebook at https://2886795271-8888-cykoria03.environments.katacoda.com/

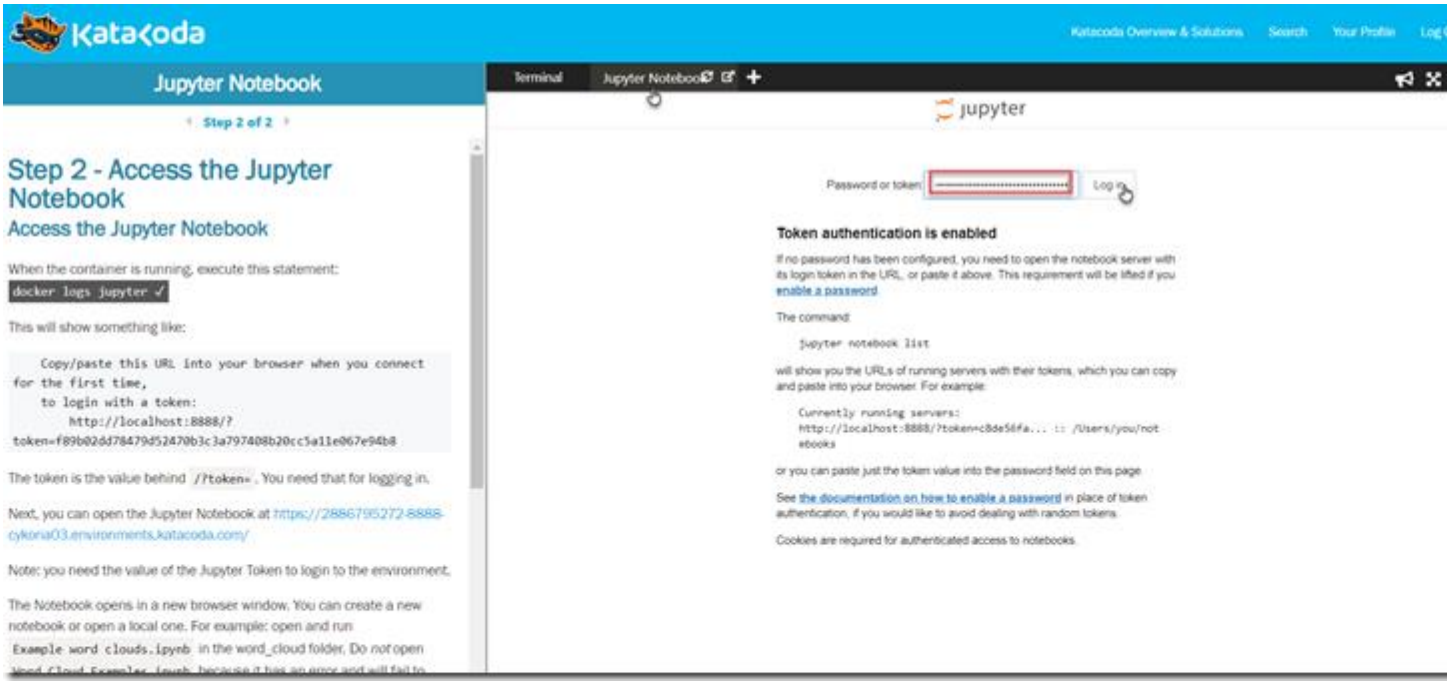
Note: you need the value of the Jupyter Token to login to the environment.

The Notebook opens in a new browser window. You can create a new notebook or open a local one. For example: open and run `Example word clouds.ipynb` in the word_cloud folder. Do *not* open `Word Cloud Examples.ipynb` because it has an error and will fail to run. Or open folder learn-pandas/lessons and start with

```
[I 07:25:50.832 NotebookApp] 0 active kernels
[I 07:25:50.832 NotebookApp] The Jupyter Notebook is running at:
[I 07:25:50.832 NotebookApp] http://[all ip addresses on your system]:8888/?token=22b91ef5e588e9d617cfaf632bb6f2e480806352379af780
[I 07:25:50.833 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 07:25:50.833 NotebookApp]

    Copy/paste this URL into your browser when you connect for the first time,
    to login with a token:
        http://localhost:8888/?token=22b91ef5e588e9d617cfaf632bb6f2e480806352379af780
[I 07:25:50.837 NotebookApp] 302 GET / (172.17.0.81) 0.80ms
[I 07:25:50.839 NotebookApp] 302 GET /tree? (172.17.0.81) 1.32ms
[I 07:25:53.954 NotebookApp] 302 GET / (172.17.0.81) 1.04ms
[I 07:25:54.009 NotebookApp] 302 GET /tree? (172.17.0.81) 2.50ms
[C 07:28:09.442 NotebookApp] received signal 15, stopping
[I 07:28:09.452 NotebookApp] Shutting down 0 kernels
Executing the command: jupyter notebook
[W 07:28:12.553 NotebookApp] WARNING: The notebook server is listening on all IP addresses and not using encryption. This is not recomm
ended.
[I 07:28:12.666 NotebookApp] JupyterLab alpha preview extension loaded from /opt/conda/lib/python3.6/site-packages/jupyterlab
[I 07:28:12.679 NotebookApp] JupyterLab application directory is /opt/conda/share/jupyter/lab
[I 07:28:12.719 NotebookApp] Serving notebooks from local directory: /home/jovyan
[I 07:28:12.720 NotebookApp] 0 active kernels
[I 07:28:12.720 NotebookApp] The Jupyter Notebook is running at:
[I 07:28:12.720 NotebookApp] http://[all ip addresses on your system]:8888/?token=9e0e710c4bb19517afea6dc2e9d2c632a86d0e6db90ba997
[I 07:28:12.720 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 07:28:12.721 NotebookApp]

    Copy/paste this URL into your browser when you connect for the first time,
    to login with a token:
        http://localhost:8888/?token=9e0e710c4bb19517afea6dc2e9d2c632a86d0e6db90ba997
$ 
```

Copy the token shown in the terminal to the clipboard.

Click on the tab Jupyter Notebook – or the link in the text on the left hand side – to open the Jupyter Notebook web console. Paste in the token and press Login:



Step 2 - Access the Jupyter Notebook

Access the Jupyter Notebook

When the container is running, execute this statement:

```
docker logs jupyter ✓
```

This will show something like:

```
    Copy/paste this URL into your browser when you connect
for the first time,
    to login with a token:
        http://localhost:8888/?
token=f89b02dd78479d52470b3c3a797408b20cc5a11e067e94b8
```

The token is the value behind `/?token=` . You need that for logging in.

Next, you can open the Jupyter Notebook at https://2886795272-8888-cykoria03.environments.katacoda.com/

Note: you need the value of the Jupyter Token to login to the environment.

The Notebook opens in a new browser window. You can create a new notebook or open a local one. For example: open and run `Example word clouds.ipynb` in the word_cloud folder. Do *not* open `Word Cloud Examples.ipynb` because it has an error and will fail to

Password or token: ••••••••••••••••••••••••••••

Token authentication is enabled

If no password has been configured, you need to open the notebook server with its login token in the URL, or paste it above. This requirement will be lifted if you enable a password.

The command:

```
jupyter notebook list
```

will show you the URLs of running servers with their tokens, which you can copy and paste into your browser. For example:

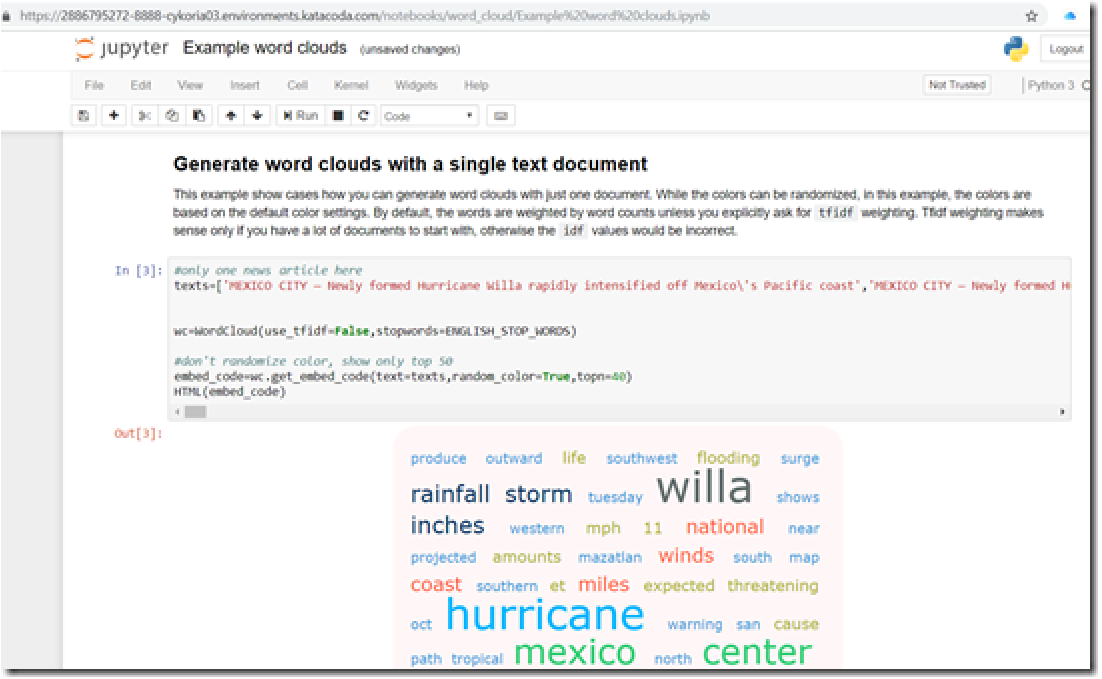
```
Currently running servers:
http://localhost:8888/?token=c8de56fa... :: /Users/you/not
ebooks
```

or you can paste just the token value into the password field on this page.

See the documentation on how to enable a password in place of token authentication, if you would like to avoid dealing with random tokens.

Cookies are required for authenticated access to notebooks.

This will open the Jupyter web environment. Open for example the preloaded notebook Example Word Cloud (from File | Open), click on Cell | Run All and see the result:



Generate word clouds with a single text document

This example show cases how you can generate word clouds with just one document. While the colors can be randomized, in this example, the colors are based on the default color settings. By default, the words are weighted by word counts unless you explicitly ask for `tfidf` weighting. Tfidf weighting makes sense only if you have a lot of documents to start with, otherwise the `idf` values would be incorrect.

```
In [3]: #only one news article here
        texts=['MEXICO CITY — Newly formed Hurricane Willa rapidly intensified off Mexico\'s Pacific coast','MEXICO CITY — Newly formed H


        wc=WordCloud(use_tfidf=False,stopwords=ENGLISH_STOP_WORDS)

        #don't randomize color, show only top 50
        embed_code=wc.get_embed_code(text=texts,random_color=True,topn=40)
        HTML(embed_code)

Out[3]:
```

Now you are ready for the next assignment.

Note: this blog article explains in detail how this playground was set up on Katacoda as a scenario. You will find that the scenario contains almost the same steps as are used for the local installation. You can also create your own Katacoda scenarios, for Jupyter Notebook, other data analytics activities as well as virtually anything you can run on a standard Linux environment.