

TAP Hands-On Workshop

Module 1: Performing Analytics on Your Data

Overview

This is module 1 of a two part workshop. This module uses the TAP Analytics Toolkit (TAP Analytics Toolkit) within the Trusted Analytics Platform (TAP) to execute a script that reads data from a TAP server and creates and outputs an amended dataset. The amended dataset becomes the input for Module 2: Visualizing Your Data with an App.

Estimated Length

This module will take approximately 1 hour to complete.

Setup

You will need to prepare your local environment/computer by installing software tools. The software you need includes:

- The software you need includes:
 - **Jupyter Notebook** (also known as Jupyter) - a tool for exploratory computation and data analysis that stores inputs/outputs in notebook documents.
<http://jupyter.readthedocs.io/en/latest/install.html>
 - **Python (2.7)** - a widely used general-purpose, high-level programming language.
<https://www.python.org/downloads/>

Objectives

After completing this module, you will have:

- Accessed Marketplace and TAP.
- Demonstrated the ability to create an instance of TAP Analytics Toolkit on your system.
- Used Python or Jupyter Notebook.
- Accessed the help functions for editing a script.
- Shown how to attach a TAP server to your project to input data.
- Executed an TAP Analytics Toolkit script that exports data.
- Demonstrated the ability to install multiple software tools needed to build and operate domain-specific applications.

Module Steps

1. Install all necessary developer tools listed in Setup.
2. Sign into Marketplace, view the Data Catalog functions and add a file into TAP.
3. Access Marketplace to create a new instance of TAP Analytics Toolkit on your system, view the running instance and copy the instance uri to use in the Netflow demo.
4. Netflow demo? Open the Netflow demo file in Python or Jupyter Notebook, review the help functions and edit the TAP Analytics Toolkit.server.uri field in the file.
5. Execute the Netflow demo to deploy the TAP Analytics Toolkit model. The demo first clears out old frames and graphs. Then it reads in the data, formats and prints the date/time data in columns, builds a graph for computing statistics, computes weighted and unweighted degree counts, computes a summary frame and downloads it to Pandas, computes and plots histograms, makes scatter plots, creates and trains an Support Vector Machine (SVM) model and downloads a scatter frame to Pandas.
6. Export the data to Hadoop Distributed File System (HDFS) as a comma separated value (CSV) file.

Important: The URLs and input commands provided in the workshop will change based on the instance used and the setup of your network. Check with the workshop instructor or consult with your development operations team (DevOps) to obtain the proper information.

Results

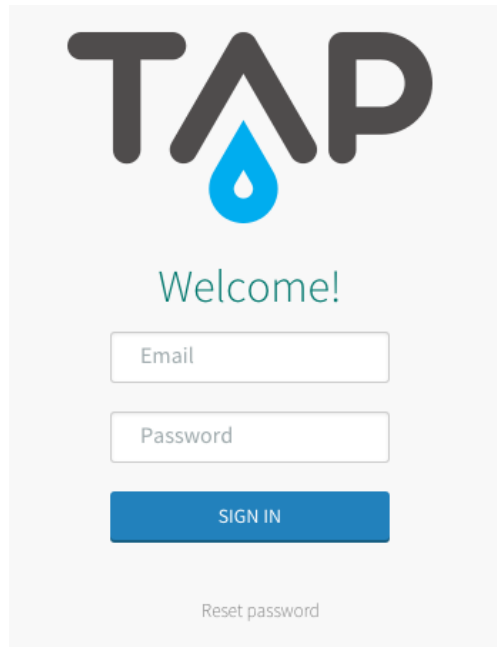
This section provides step-by-step instructions and their results.

1. Install all necessary developer tools listed in Setup

1.1. No additional instructions are provided for this step.

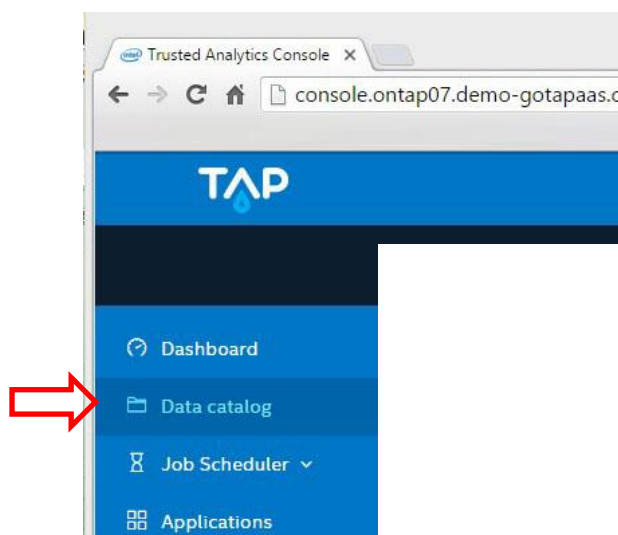
2. TAP Marketplace

2.1. Sign into TAP with your User ID and Password:



2.2. View the Data Catalog functions:

- Click on Data Catalog under Main navigation to access the three Data Catalog tabs – Data sets, Transfers and Submit Transfer.



- Click on the **Data sets** tab to see the datasets on TAP:

- If you have any data sets, you will see them listed on the first tab along with any data sets marked public. Use 'Advanced Search' or a category to limit the number of data sets you see.
- In this example, we are in the Demo organization in Marketplace. The Demo organization is only available in this TAP instance. The organization name may be different if you use your own instance.
- **Note:** Make sure you select the correct organization when you look for your dataset in Marketplace.

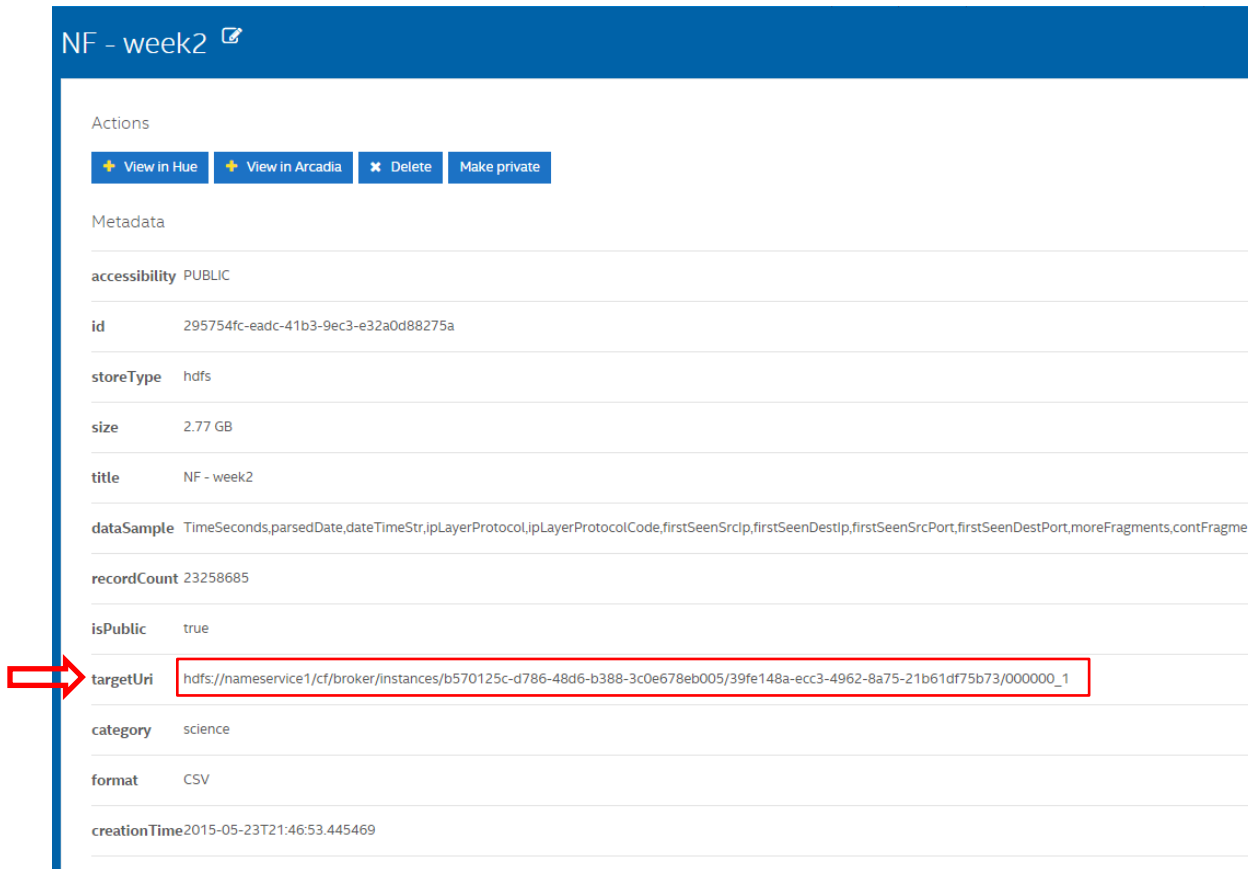
The screenshot shows the 'Data sets' tab in the Marketplace interface. At the top, there are tabs for 'Data sets', 'Transfers', and 'Submit Transfer'. Below the tabs, it says 'Found 18 datasets' and 'ATK-Demo'. The 'Advanced search' section includes a 'Format' dropdown (all, csv) and a 'Creation time' range selector. Below this is a 'Choose category:' section with buttons for 'all', 'other', 'business', 'finance', and 'science'. A grid of 12 dataset cards is displayed, each with a title, a 'View' button, and a lock icon. Red arrows point to the 'Advanced search' and 'science' category buttons with explanatory text.

Click Advanced search to locate a dataset based on its creation date

Click on a category to see datasets specific to that category

Dataset Name	View Button	Lock Icon
test-transfer-1437427519.6781388	+ View	
transfer906099	+ View	
cities	+ View	
testtest	+ View	
1119	+ View	
transfer1	+ View	
transfer534484	+ View	
test-transfer-1437470000.9196236	+ View	
test145	+ View	
movies2	+ View	
xxxxxxx	+ View	
test	+ View	

- Click the **Dataset Title** on any data set to display dataset information, such as:
 - Title, size, category ...
 - **TargetUri** is a link to a dataset. The dataset link shown may not be the one needed in this exercise.
 - You can copy this link to use in other apps.



The screenshot shows the Hue interface for a dataset titled "NF - week2". The interface includes an "Actions" bar with buttons for "View in Hue", "View in Arcadia", "Delete", and "Make private". Below this is a "Metadata" section displaying various dataset properties. A red arrow points to the "targetUri" field, which contains a long HDFS path: `hdfs://nameservice1/ct/broker/instances/b570125c-d786-48d6-b388-3c0e678eb005/39fe148a-ecc3-4962-8a75-21b61df75b73/000000_1`.

Actions	
View in Hue	View in Arcadia
Delete	Make private

Metadata

accessibility	PUBLIC
id	295754fc-eadc-41b3-9ec3-e32a0d88275a
storeType	hdfs
size	2.77 GB
title	NF - week2
dataSample	TimeSeconds,parsedDate,dateTimeStr,ipLayerProtocol,ipLayerProtocolCode,firstSeenSrcIp,firstSeenDestIp,firstSeenSrcPort,firstSeenDestPort,moreFragments,contFragments
recordCount	23258685
isPublic	true
targetUri	<code>hdfs://nameservice1/ct/broker/instances/b570125c-d786-48d6-b388-3c0e678eb005/39fe148a-ecc3-4962-8a75-21b61df75b73/000000_1</code>
category	science
format	CSV
creationTime	2015-05-23T21:46:53.445469

Note - Clicking the 'View' button will prompt the data set to be published in Hue through a pop up.

- Click on the **Data Catalog** tab.
- Click on the **Transfers** tab to see what files have been uploaded to TAP and to get information on when a file was added to the system.

Data sets

Transfers

+ Submit Transfer

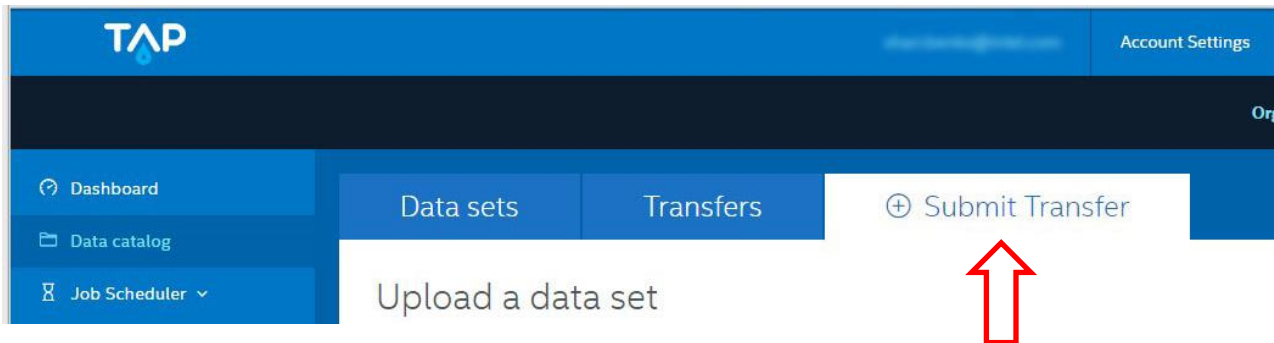
Transfers

ATK-Demo dev

Refresh

Source	Created	Last state	Category	Title	Item
<input type="text"/>			<input type="text"/>	<input type="text"/>	
http://www.vacommunity.org/VAST+Challenge+2013%3A+Mini-Challenge+3	25.06.2015 08:15:16	NEW 25.06.2015 08:15:16	other	VAST_mini_challenge	
http://www.vacommunity.org/VAST+Challenge+2013%3A+Mini-Challenge+3	24.06.2015 08:21:27	NEW 24.06.2015 08:21:27	other	VAST_mini_challenge	
http://console.demo-gotapaas.com/app/views/dashboard/dashboard.html	18.06.2015 07:41:33	FINISHED 18.06.2015 07:41:34	science	lda_a2	
http://console.demo-gotapaas.com/app/views/dashboard/dashboard.html	18.06.2015 07:28:01	FINISHED 18.06.2015 07:28:06	science	lda_a2	
https://s3-us-west-2.amazonaws.com/dp2-atk/lda.csv?X-Amz-Date=20150615T103426Z&X-Amz-Expir	15.06.2015 03:35:28	FINISHED 15.06.2015 03:35:29	science	lda_a2	
https://s3-us-west-2.amazonaws.com/dp2-atk/lda.csv?X-Amz-Date=20150615T101731Z&X-Amz-Expir	15.06.2015 03:20:13	FINISHED 15.06.2015 03:20:14	science	lda_a1	
https://s3-us-west-2.amazonaws.com/dp2-atk/lda.csv?X-Amz-Date=20150615T093311Z&X-Amz-Expir	15.06.2015 03:16:41	ERROR 15.06.2015 03:16:41	science	lda_a	
https://s3-eu-west-1.amazonaws.com/rbiegacz-temp/movie_data_with_names.csv?X-Amz-Date=2015	03.06.2015 12:35:32	FINISHED 03.06.2015 12:35:36	consumer	Movie Data (prod)	

- 2.3. To add a file into TAP:
- Click the **Submit Transfer** tab.



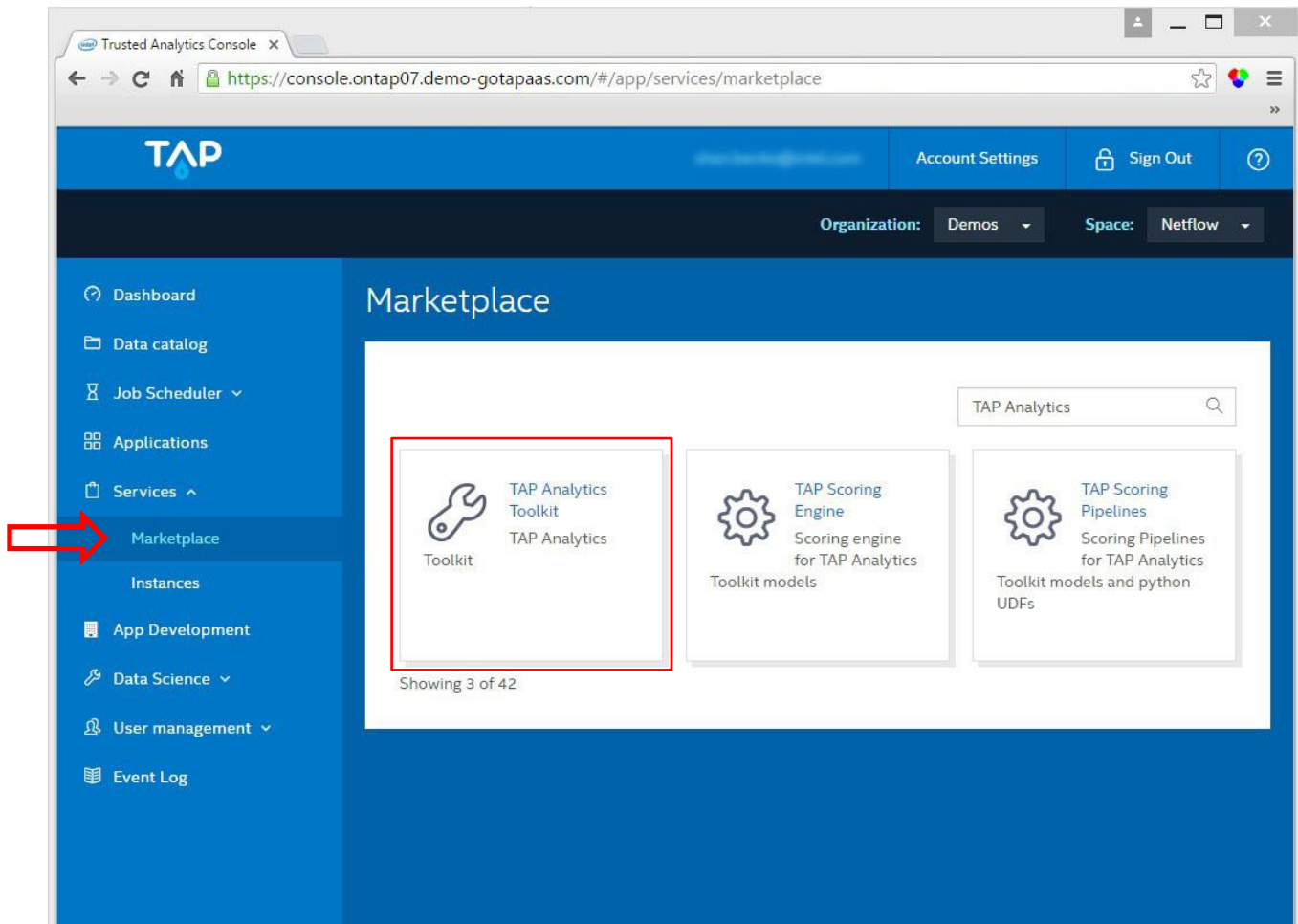
- Complete the Submit Transfer fields:
 - **Category** – Click on a category, based on your needs.
 - **Link field** – Enter the URL of the dataset you want to upload.
 - **Title field** – Enter a file name for your dataset.
- Click on Upload after you have entered the necessary information.

This screenshot shows the 'Submit Transfer' form in the TAP interface. The form is titled 'Upload a data set'. It has two radio buttons for 'Choose input type': 'Link' (selected) and 'Local path'. Below these are two text input fields: 'Link' with a placeholder 'place link...' and 'Title' with a placeholder 'title for link...'. Red arrows point to these fields with the text 'Enter Dataset'. Below the input fields is a section 'Choose category' with a grid of 12 category icons: agriculture, business, climate, consumer, ecosystems, education, energy, finance, health, manufacturing, science, and other. A red bracket on the right side of the grid is labeled 'Choose a'. The 'other' category is highlighted with a blue border. At the bottom of the form is an 'Upload' button, with a red arrow pointing to it and the text 'Click Upload to load the'.

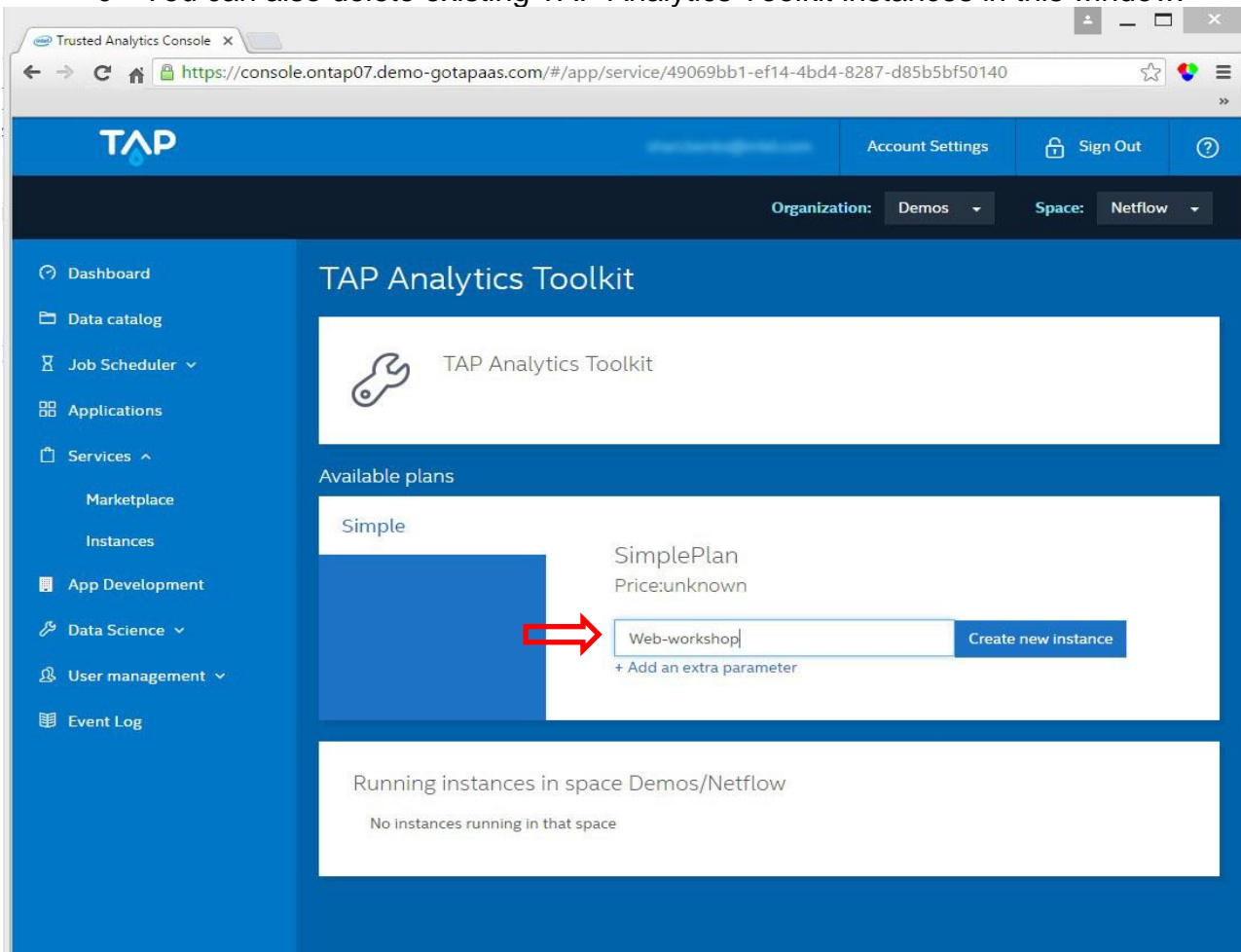
3. TAP Marketplace

3.1. Create a new instance of TAP Analytics Toolkit in TAP:

- Click on **Services > Marketplace** under Main navigation.
- Search for then click on TAP Analytics Toolkit.

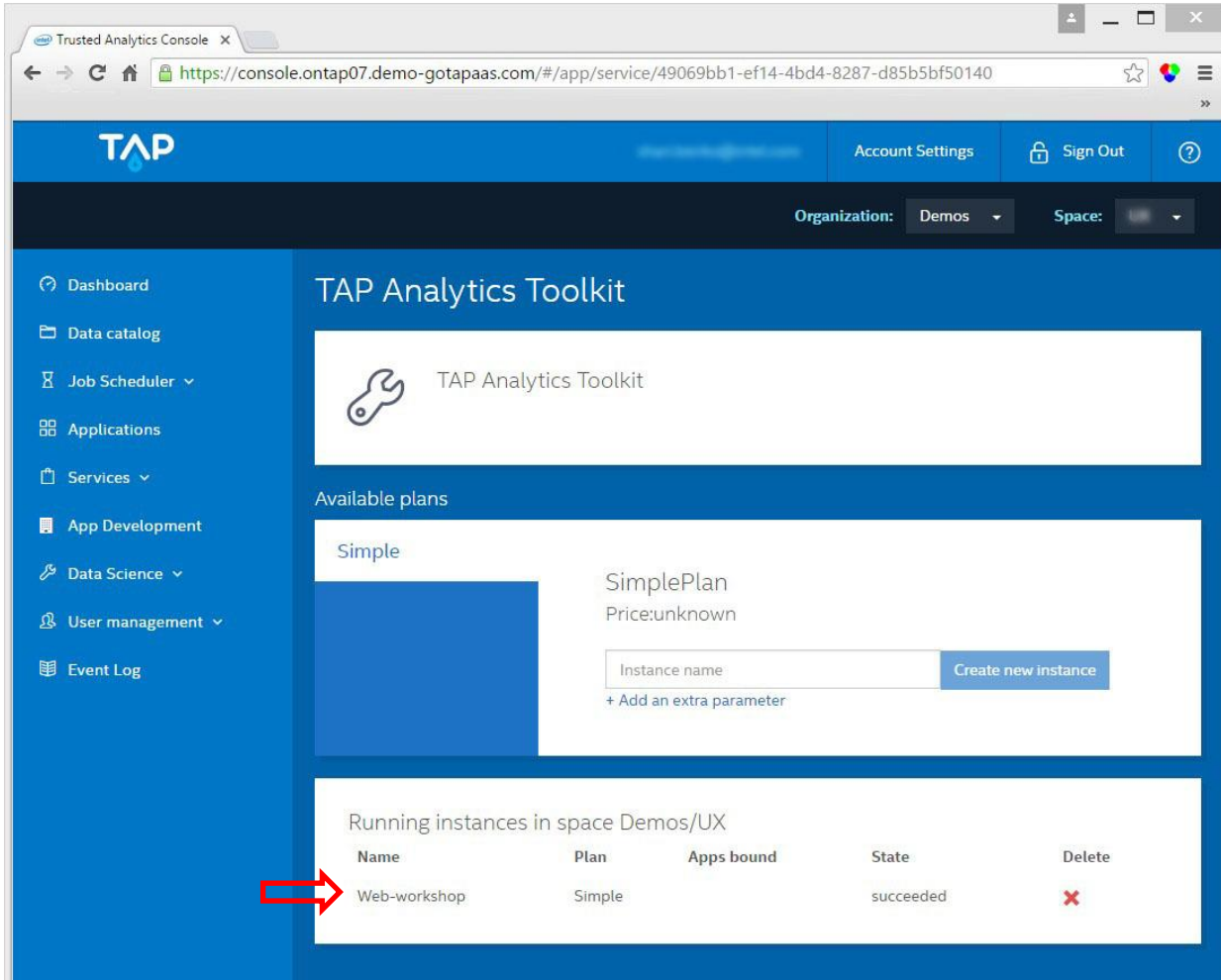


- In the TAP Analytics Toolkit window, enter a name for your instance and click Create new instance. In this example, we created an instance called Web-Workshop.
- You can also delete existing TAP Analytics Toolkit instances in this window.




3.2. View the running instance:

- Refresh your window to see the instance you created.



The screenshot shows the TAP Analytics Toolkit web interface. The top navigation bar includes the TAP logo, user information, and links for Account Settings, Sign Out, and help. Below this, the 'Organization' is set to 'Demos' and the 'Space' is set to 'UX'. The left sidebar contains a menu with options: Dashboard, Data catalog, Job Scheduler, Applications, Services, App Development, Data Science, User management, and Event Log. The main content area is titled 'TAP Analytics Toolkit' and features a wrench icon. Under 'Available plans', the 'Simple' plan is selected, showing 'SimplePlan' with a price of 'unknown'. A form for creating a new instance includes an 'Instance name' field and a 'Create new instance' button. Below this, a table titled 'Running instances in space Demos/UX' displays one instance: 'Web-workshop' on the 'Simple' plan, with a state of 'succeeded'. A red arrow points to the 'Web-workshop' entry in the table.

Name	Plan	Apps bound	State	Delete
Web-workshop	Simple		succeeded	

3.3. Copy the instance URL to use in the Netflow demo:

- Click Applications under Main navigation to view the status of the new instance in the Applications window.
- Copy the URL of the instance. You will need the URL to run the Netflow demo in the next part of the workshop.
 - The red box in the Status column indicates the instance is still initiating and not yet available. The red box is replaced by a green check when the instance is running.
 - Click on the instance's URL to see if the instance is up and running.

The screenshot shows the 'Applications' page in the Trusted Analytics Console. The left sidebar has a red arrow pointing to the 'Applications' menu item. The main content area shows a table with the following data:

Name	Status	Instances	URLs
Web-workshop-6845fffd	✓	1	Web-workshop-6845fffd.ontap07.demo-gotapaas.com

The URL 'Web-workshop-6845fffd.ontap07.demo-gotapaas.com' is highlighted with a red box. A 'See details' link is also present. At the bottom right, there are pagination controls for 10, 25, 50, and 100 items.

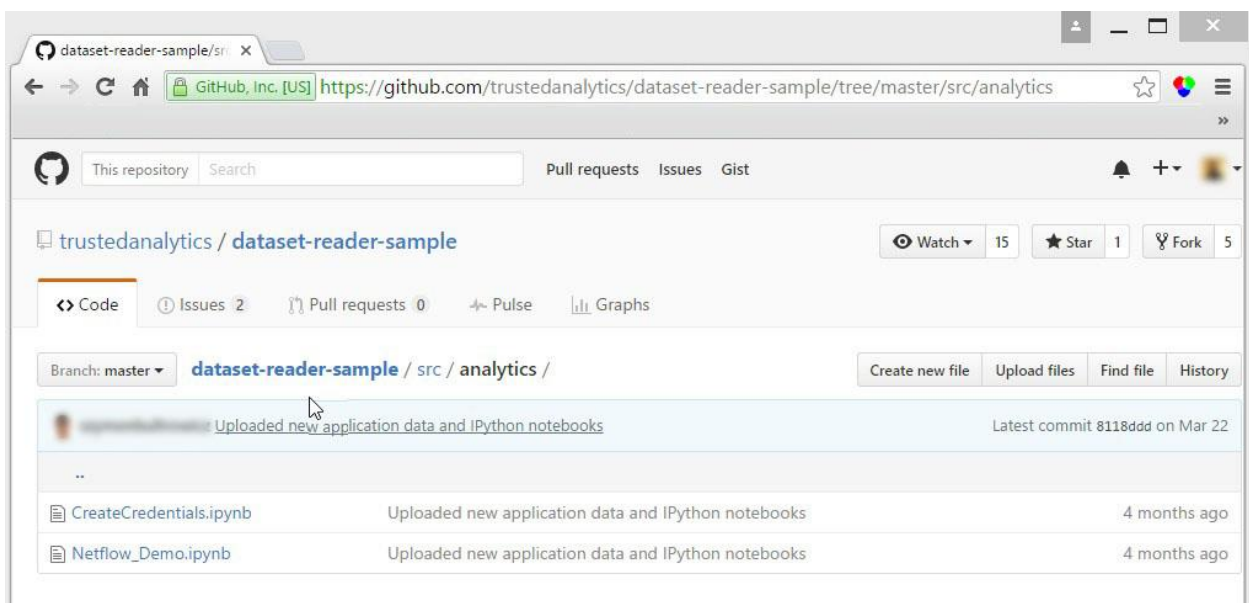
4. Netflow demo

4.1. What is the Netflow demo?

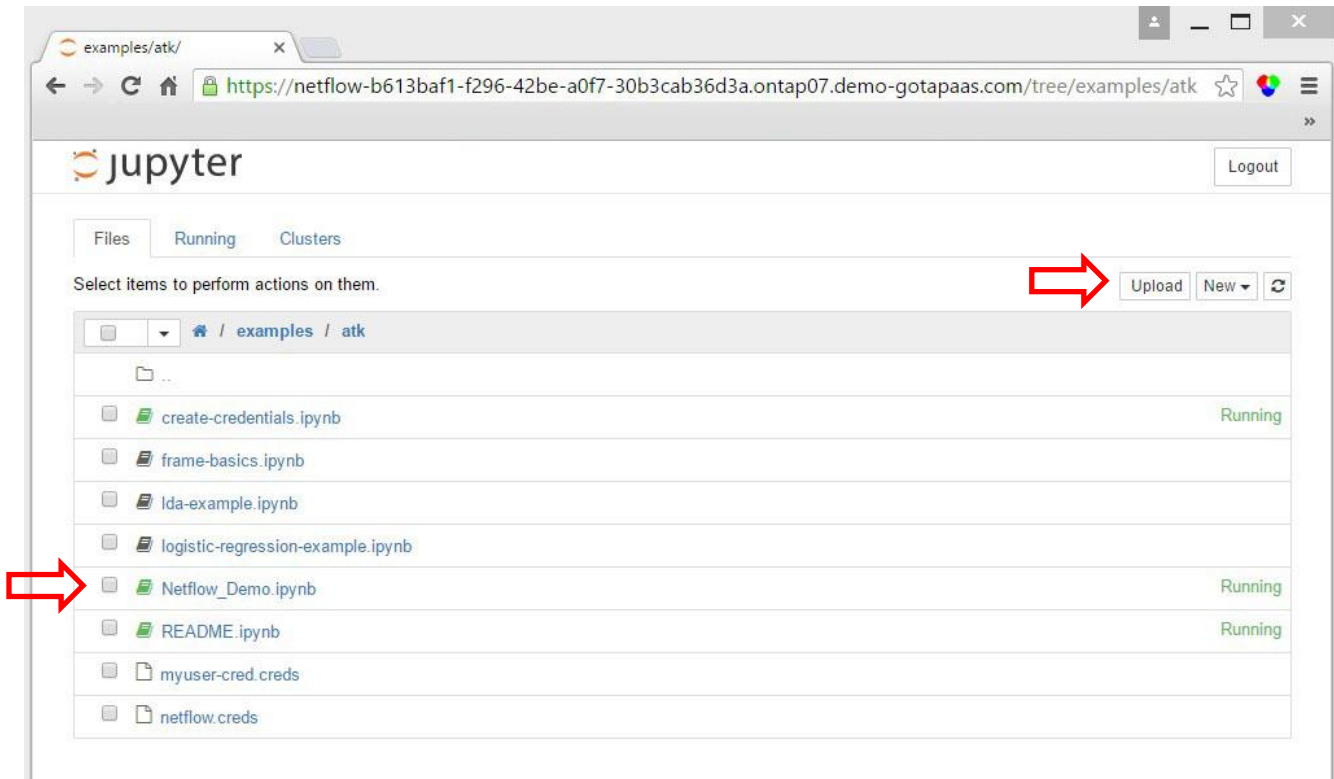
- The Netflow dataset contains information from web servers. The dataset contains information on tracked network traffic, which is incoming and outgoing internet traffic.
- The demo reads and trains the dataset. The demo then establishes a normal traffic pattern by tying the network edges and nodes together.
- It builds a graph showing all the connections based on the nodes' IP addresses.
- Finally, it produces a number of charts and graphs to show the network traffic flow and display outliers, which are outside the normal traffic pattern.

4.2. Open the Netflow demo file:

- Open the Netflow demo file using a Jupyter Notebook instance on TAP using the following steps:
 - Using the left side navigation, click 'Data Science'. Three additional options appear, click 'Jupyter'.
 - Specify an instance name for your new Jupyter instance and then click 'Create new Jupyter instance.' A notification will appear saying your new instance is created but may take a while. Refresh the page and you should see a new Jupyter instance on the page.
 - If you are using the Jupyter Notebook instance for the first time, you'll need to create a new set of credentials to connect to TAP Analytics Toolkit. You can use the file provided here: <https://github.com/trustedanalytics/dataset-reader-sample/tree/master/src/analytics>.
 - Download the Netflow demo script file (Netflow_demo.ipynb) located in same repository <https://github.com/trustedanalytics/dataset-reader-sample/tree/master/src/analytics>.

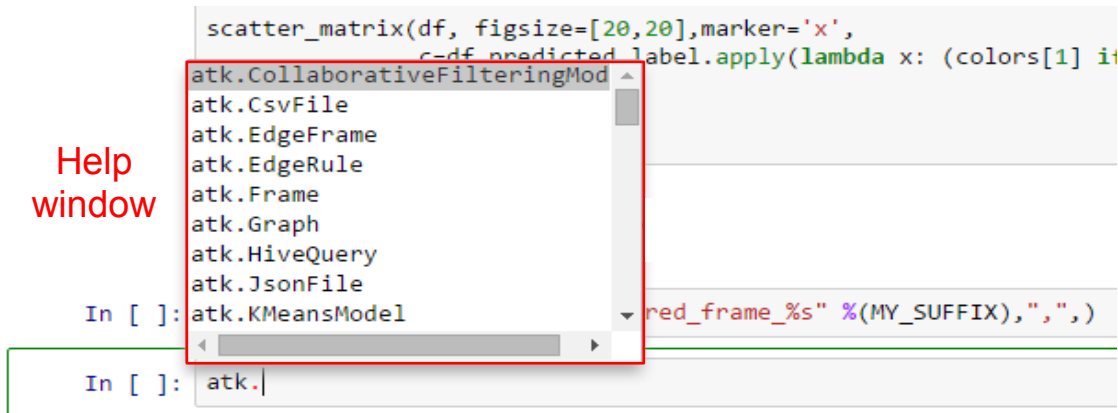


- Upload the demo script file using the upload button in the upper right of your Jupyter Instance, then click on the file to open the demo.
- You can find the training dataset located on the Trusted Analytics Community site: <https://community.trustedanalytics.org/docs/DOC-1058>



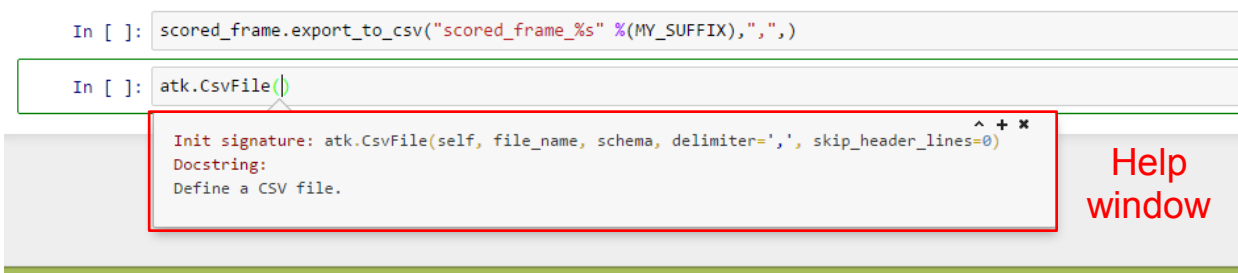
4.3. Review the help functions:

- If you need help knowing what commands are available:
 - Type the command into the In [] field; for example, “**TAP Analytics Toolkit.**”
 - Then press the **tab key** to see the available command options.



A screenshot of a Jupyter Notebook interface. The top part shows a code cell with the following code: `scatter_matrix(df, figsize=[20,20],marker='x',` followed by a line that is partially obscured by a help window. The help window is a dropdown menu showing a list of available commands: `atk.CollaborativeFilteringMod`, `atk.CsvFile`, `atk.EdgeFrame`, `atk.EdgeRule`, `atk.Frame`, `atk.Graph`, `atk.HiveQuery`, `atk.JsonFile`, and `atk.KMeansModel`. The text "Help window" is written in red to the left of the dropdown. Below the code cell, the input field shows `In []: atk.KMeansModel` and `In []: atk.` with a red box highlighting the `atk.` part.

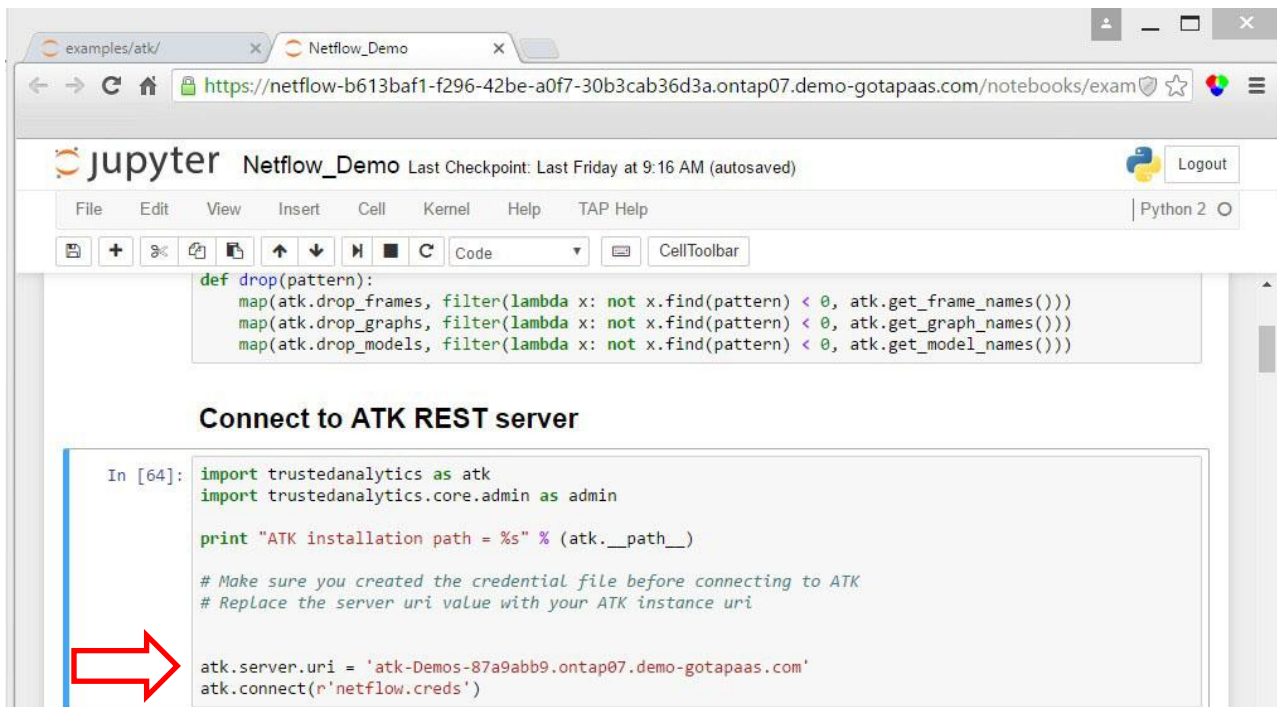
- If you need help knowing what the default parameters are for a command:
 - Type the command into the In field. For example, “**TAP Analytics Toolkit.CsvFile()**”.
 - Then press the **shift** and **tab keys** to see the default parameters for that command.



A screenshot of a Jupyter Notebook interface. The top part shows a code cell with the following code: `In []: scored_frame.export_to_csv("scored_frame_%s" %(MY_SUFFIX),",",)`. Below this, the input field shows `In []: atk.CsvFile()` with a red box highlighting the `atk.CsvFile()` part. A help window is open, showing the following text: `Init signature: atk.CsvFile(self, file_name, schema, delimiter=',', skip_header_lines=0)`, `Docstring:`, and `Define a CSV file.` The text "Help window" is written in red to the right of the help window.

4.4. Connect to ATK REST Server

- Change the `atk.server.uri` in the demo script to access the TAP Analytics Toolkit instance you created in TAP.
- Copy the URL for your Analytics Toolkit instance you created in Step 3.3.
- Paste the URL of the instance into the `atk.server.uri =` field of the Netflow demo script.
- **Important:** You must remove the “`http://`” at the beginning of the URL and the “`/`” at the end, before you enter the uri information.
- You should already have created a credentials file, when you installed Python or Jupyter.



The screenshot shows a Jupyter Notebook titled "Netflow_Demo" in a web browser. The notebook contains a Python script for connecting to an ATK REST server. A red arrow points to the line where `atk.server.uri` is assigned a value.

```
def drop(pattern):
    map(atk.drop_frames, filter(lambda x: not x.find(pattern) < 0, atk.get_frame_names()))
    map(atk.drop_graphs, filter(lambda x: not x.find(pattern) < 0, atk.get_graph_names()))
    map(atk.drop_models, filter(lambda x: not x.find(pattern) < 0, atk.get_model_names()))

Connect to ATK REST server

In [64]: import trustedanalytics as atk
import trustedanalytics.core.admin as admin

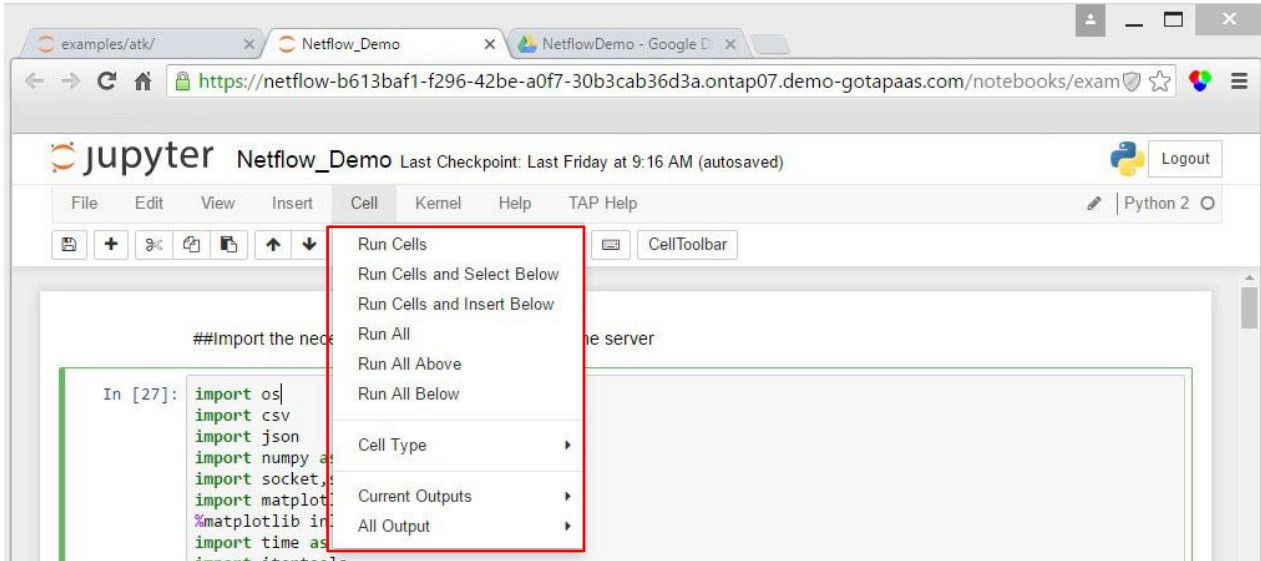
print "ATK installation path = %s" % (atk.__path__)

# Make sure you created the credential file before connecting to ATK
# Replace the server uri value with your ATK instance uri

atk.server.uri = 'atk-Demos-87a9abb9.ontap07.demo-gotapaas.com'
atk.connect(r'netflow.creds')
```

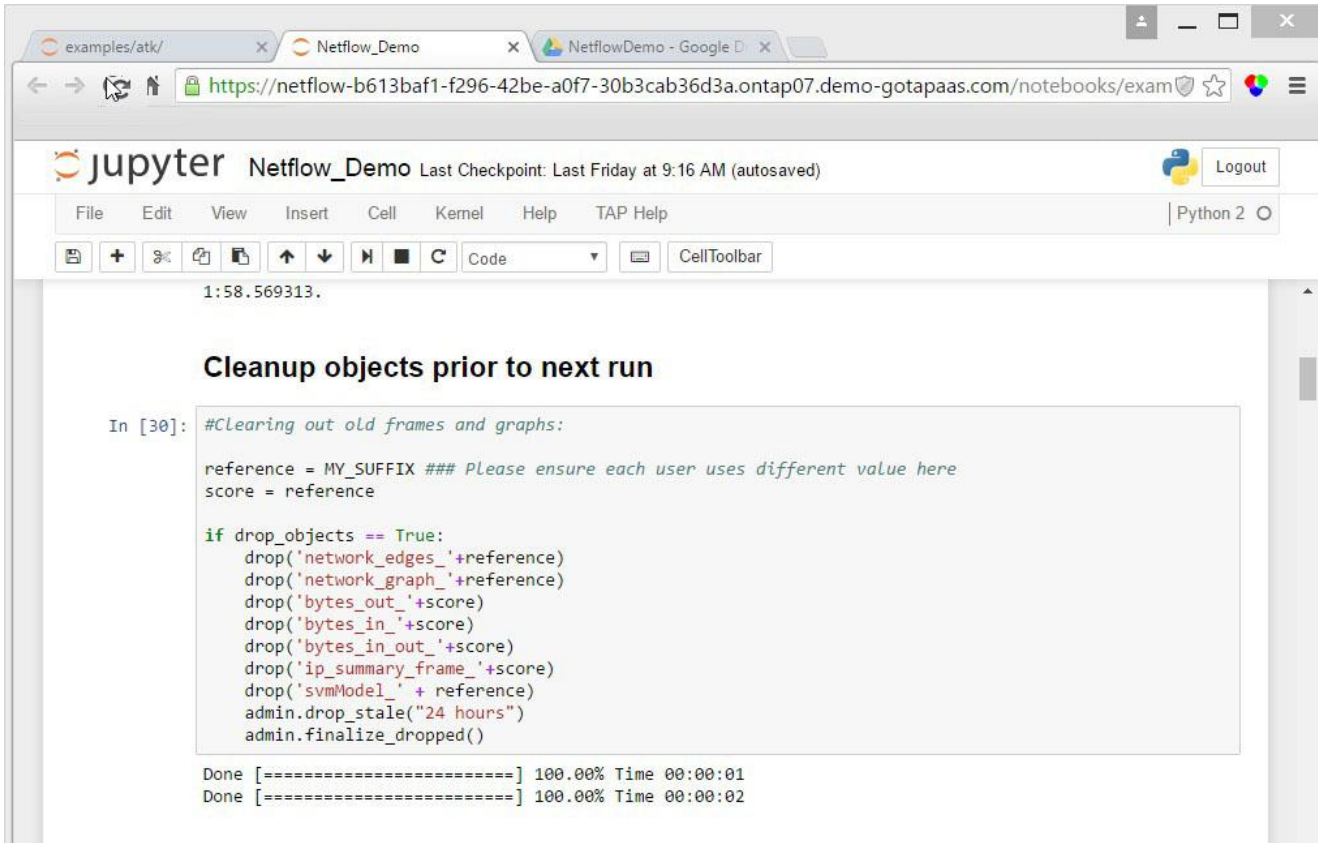
5. Deploy the TAP Analytics Toolkit Model

- Execute the Netflow demo, which deploys the TAP Analytics Toolkit.
- To begin, click Cell and select Run All. This runs all the scripts in the demo.
- If you need to rerun a single task, choose Run to execute a specific script in the demo.
- Or, if you need to run a group of scripts, choose Run All Below.



5.1. Clear out the old frames and graphs:

- The first step in the script cleans up the objects to ensure the code can be rerun multiple times.
- This script resets all the counters to zero.



The screenshot shows a Jupyter Notebook interface in a web browser. The browser tabs include 'examples/atk/', 'Netflow_Demo', and 'NetflowDemo - Google D...'. The address bar shows the URL 'https://netflow-b613baf1-f296-42be-a0f7-30b3cab36d3a.ontap07.demo-gotapaas.com/notebooks/exam'. The Jupyter interface has a top bar with the 'jupyter' logo, the title 'Netflow_Demo', and a 'Last Checkpoint: Last Friday at 9:16 AM (autosaved)' message. A 'Logout' button is on the right. Below the top bar is a menu bar with 'File', 'Edit', 'View', 'Insert', 'Cell', 'Kernel', 'Help', and 'TAP Help'. A toolbar contains icons for file operations and a 'Code' dropdown menu. The main area shows a code cell with the following content:

```
1:58.569313.

Cleanup objects prior to next run

In [30]: #Clearing out old frames and graphs:

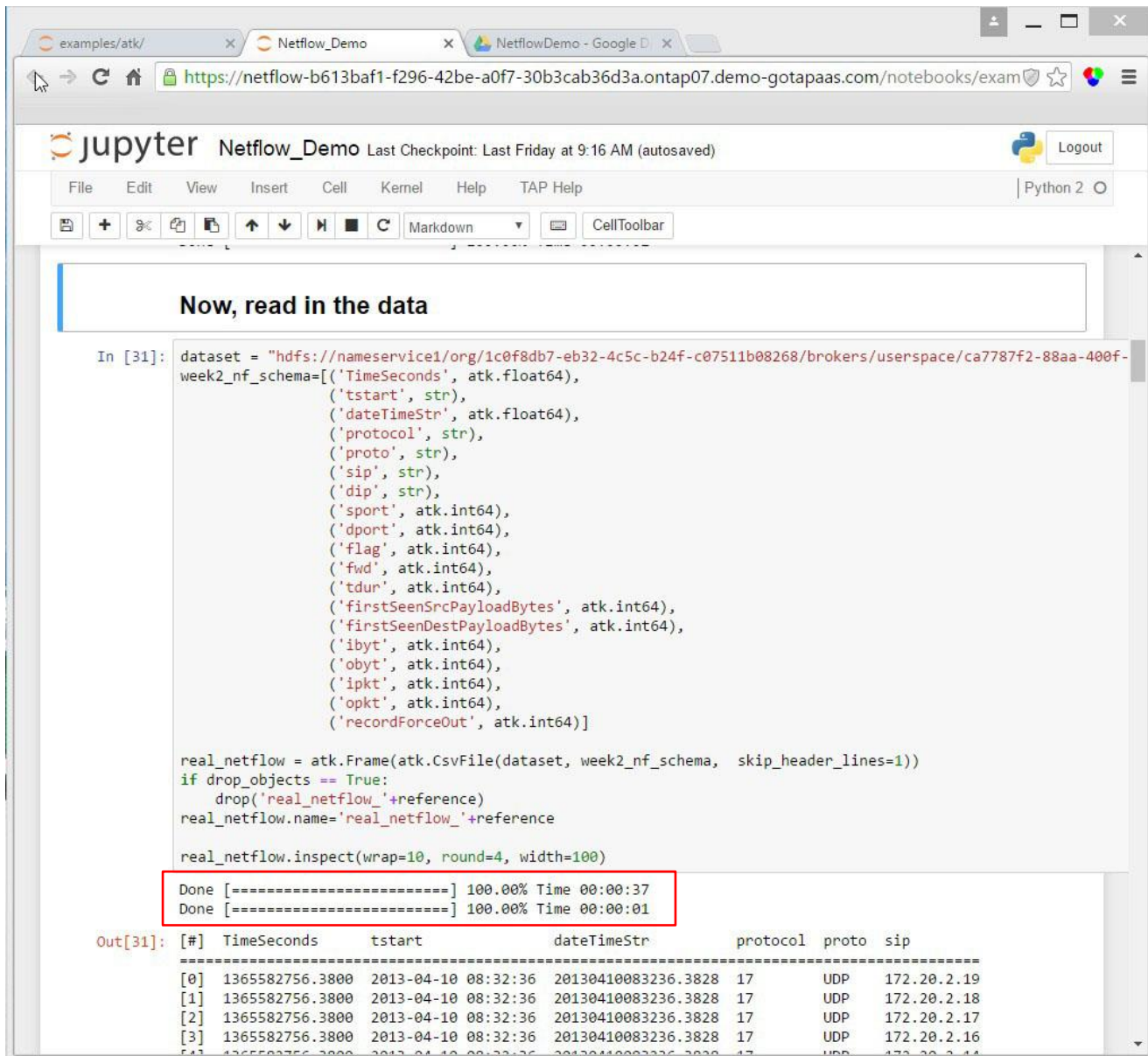
reference = MY_SUFFIX ### Please ensure each user uses different value here
score = reference

if drop_objects == True:
    drop('network_edges_'+reference)
    drop('network_graph_'+reference)
    drop('bytes_out_'+score)
    drop('bytes_in_'+score)
    drop('bytes_in_out_'+score)
    drop('ip_summary_frame_'+score)
    drop('svmModel_'+reference)
    admin.drop_stale("24 hours")
    admin.finalize_dropped()

Done [=====] 100.00% Time 00:00:01
Done [=====] 100.00% Time 00:00:02
```

5.2. Now, read in the data:

- This step reads in the data file that was loaded onto the TAP Analytics Toolkit server.
- The Netflow dataset includes: date and time information, node IP addresses, protocols, bytes in and out, packets in and out ...
- Confirm that the script completed 100% of the tasks and review the log for error messages. You may see several progress bars when multiple tasks run in the same section.



The screenshot shows a Jupyter Notebook titled "Netflow_Demo" in a web browser. The code cell contains the following Python code:

```
In [31]: dataset = "hdfs://nameservice1/org/1c0f8db7-eb32-4c5c-b24f-c07511b08268/brokers/userspace/ca7787f2-88aa-400f-week2_nf_schema=[('TimeSeconds', atk.float64),
                    ('tstart', str),
                    ('dateTimeStr', atk.float64),
                    ('protocol', str),
                    ('proto', str),
                    ('sip', str),
                    ('dip', str),
                    ('sport', atk.int64),
                    ('dport', atk.int64),
                    ('flag', atk.int64),
                    ('fwd', atk.int64),
                    ('tdur', atk.int64),
                    ('firstSeenSrcPayloadBytes', atk.int64),
                    ('firstSeenDestPayloadBytes', atk.int64),
                    ('ibyt', atk.int64),
                    ('obyte', atk.int64),
                    ('ipkt', atk.int64),
                    ('opkt', atk.int64),
                    ('recordForceOut', atk.int64)]

real_netflow = atk.Frame(atk.CsvFile(dataset, week2_nf_schema, skip_header_lines=1))
if drop_objects == True:
    drop('real_netflow_'+reference)
real_netflow.name='real_netflow_'+reference

real_netflow.inspect(wrap=10, round=4, width=100)
```

The output of the code cell is shown below the code, with a red box highlighting the progress bar:

```
Out[31]: Done [=====] 100.00% Time 00:00:37
         Done [=====] 100.00% Time 00:00:01
```

#	TimeSeconds	tstart	dateTimeStr	protocol	proto	sip
[0]	1365582756.3800	2013-04-10 08:32:36	20130410083236.3828	17	UDP	172.20.2.19
[1]	1365582756.3800	2013-04-10 08:32:36	20130410083236.3828	17	UDP	172.20.2.18
[2]	1365582756.3800	2013-04-10 08:32:36	20130410083236.3828	17	UDP	172.20.2.17
[3]	1365582756.3800	2013-04-10 08:32:36	20130410083236.3828	17	UDP	172.20.2.16

5.3. Add Columns for date/time

- This script formats the data into columns for date, time ...

The screenshot shows a Jupyter Notebook titled "Netflow_Demo" in a web browser. The notebook contains a Python script that defines a function `get_time` to parse a timestamp string into date, time, hour, minute, and numeric_time components. The script then uses `real_netflow.add_columns` to add these columns to the dataset and `real_netflow.inspect` to display the first 10 rows. The output shows a table with 6 columns: `date`, `time`, `hour`, `minute`, and `numeric_time`. The `numeric_time` column contains the value 8.5333 for all rows.

```
In [32]: def get_time(row):
          x = row.tstart.split(" ")
          date = x[0]
          time = x[1]
          y = time.split(":")
          hour = atk.float64(y[0])
          minute = atk.float64(y[1])
          numeric_time = hour + minute/60.0
          return [date, time, hour, minute, numeric_time]

          real_netflow.add_columns(get_time, [('date', str),
                                              ('time', str),
                                              ('hour', atk.float64),
                                              ('minute', atk.float64),
                                              ('numeric_time', atk.float64)],
                                  columns_accessed=['tstart'])

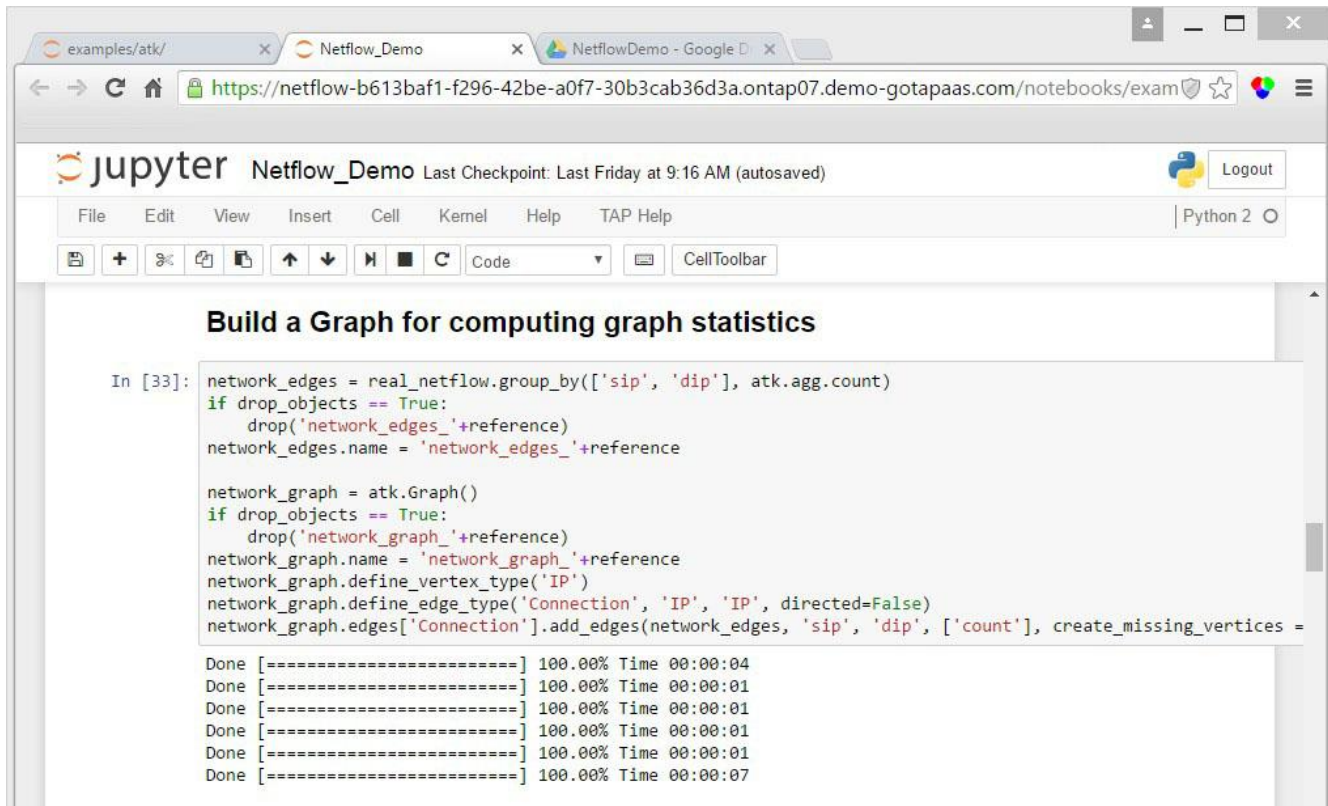
          real_netflow.inspect(wrap=10, round=4, width=100,
                               columns=['date', 'time', 'hour', 'minute', 'numeric_time'])

          Done [=====] 100.00% Time 00:00:07

Out[32]:
```

#	date	time	hour	minute	numeric_time
[0]	2013-04-10	08:32:36	8.0000	32.0000	8.5333
[1]	2013-04-10	08:32:36	8.0000	32.0000	8.5333
[2]	2013-04-10	08:32:36	8.0000	32.0000	8.5333
[3]	2013-04-10	08:32:36	8.0000	32.0000	8.5333
[4]	2013-04-10	08:32:36	8.0000	32.0000	8.5333
[5]	2013-04-10	08:32:36	8.0000	32.0000	8.5333
[6]	2013-04-10	08:32:36	8.0000	32.0000	8.5333
[7]	2013-04-10	08:32:36	8.0000	32.0000	8.5333
[8]	2013-04-10	08:32:36	8.0000	32.0000	8.5333
[9]	2013-04-10	08:32:36	8.0000	32.0000	8.5333

5.4. Build a Graph for computing graph statistics



The screenshot shows a Jupyter Notebook interface with the title "Netflow_Demo". The notebook contains a single code cell with the following Python code:

```
In [33]: network_edges = real_netflow.groupby(['sip', 'dip'], atk.agg.count)
if drop_objects == True:
    drop('network_edges_'+reference)
network_edges.name = 'network_edges_'+reference

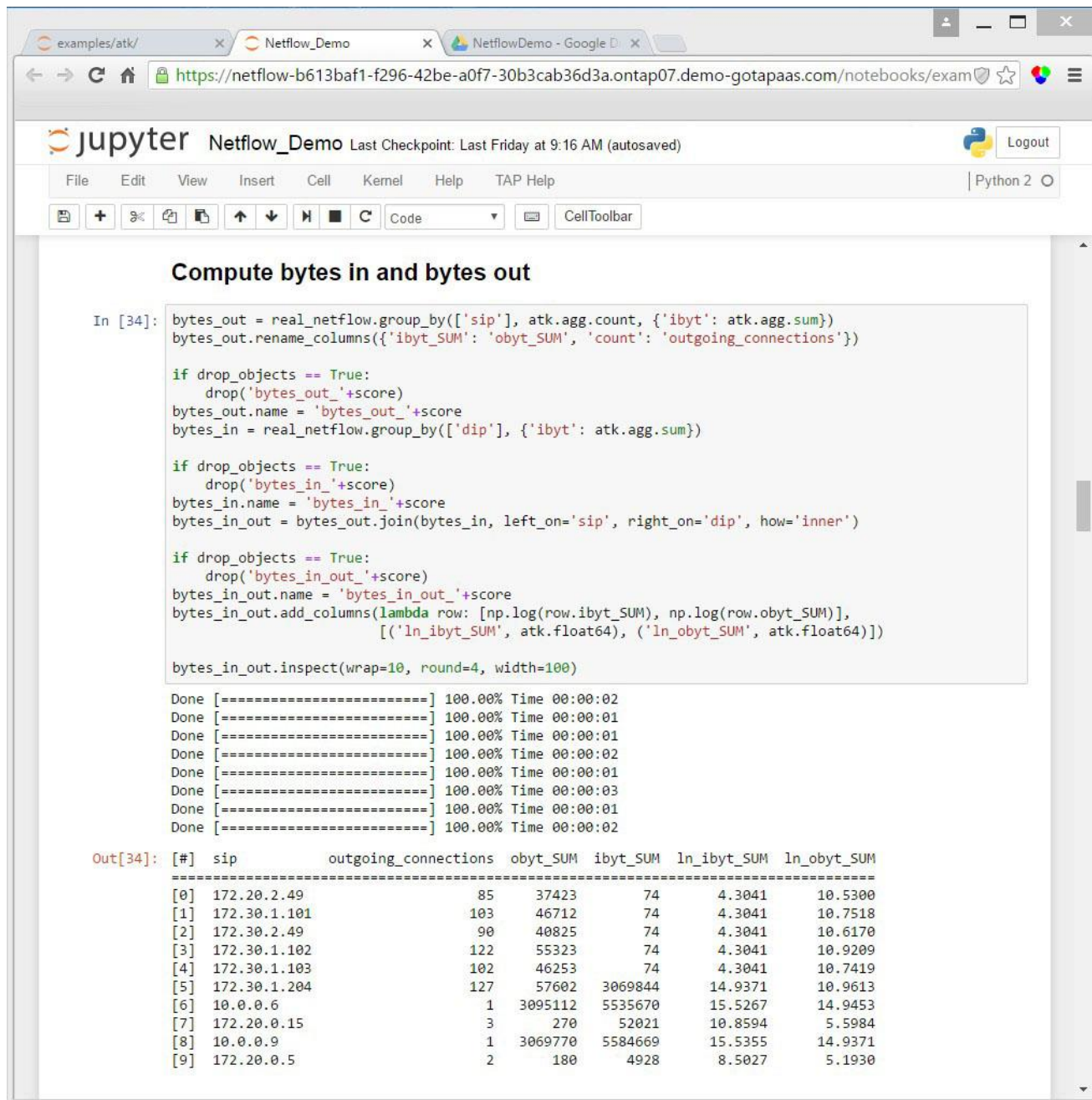
network_graph = atk.Graph()
if drop_objects == True:
    drop('network_graph_'+reference)
network_graph.name = 'network_graph_'+reference
network_graph.define_vertex_type('IP')
network_graph.define_edge_type('Connection', 'IP', 'IP', directed=False)
network_graph.edges['Connection'].add_edges(network_edges, 'sip', 'dip', ['count'], create_missing_vertices =
```

The output of the code cell shows progress bars for each step:

```
Done [=====] 100.00% Time 00:00:04
Done [=====] 100.00% Time 00:00:01
Done [=====] 100.00% Time 00:00:01
Done [=====] 100.00% Time 00:00:01
Done [=====] 100.00% Time 00:00:01
Done [=====] 100.00% Time 00:00:07
```


5.5. Computes bytes in and bytes out:

- This script computes the number of bytes in and bytes out for each node and creates total I/O counts for the nodes.



Compute bytes in and bytes out

```
In [34]: bytes_out = real_netflow.groupby(['sip'], atk.agg.count, {'ibyt': atk.agg.sum})
bytes_out.rename_columns({'ibyt_SUM': 'obyt_SUM', 'count': 'outgoing_connections'})

if drop_objects == True:
    drop('bytes_out_'+score)
bytes_out.name = 'bytes_out_'+score
bytes_in = real_netflow.groupby(['dip'], {'ibyt': atk.agg.sum})

if drop_objects == True:
    drop('bytes_in_'+score)
bytes_in.name = 'bytes_in_'+score
bytes_in_out = bytes_out.join(bytes_in, left_on='sip', right_on='dip', how='inner')

if drop_objects == True:
    drop('bytes_in_out_'+score)
bytes_in_out.name = 'bytes_in_out_'+score
bytes_in_out.add_columns(lambda row: [np.log(row.ibyt_SUM), np.log(row.obyt_SUM)],
                                [('ln_ibyt_SUM', atk.float64), ('ln_obyt_SUM', atk.float64)])

bytes_in_out.inspect(wrap=10, round=4, width=100)
```

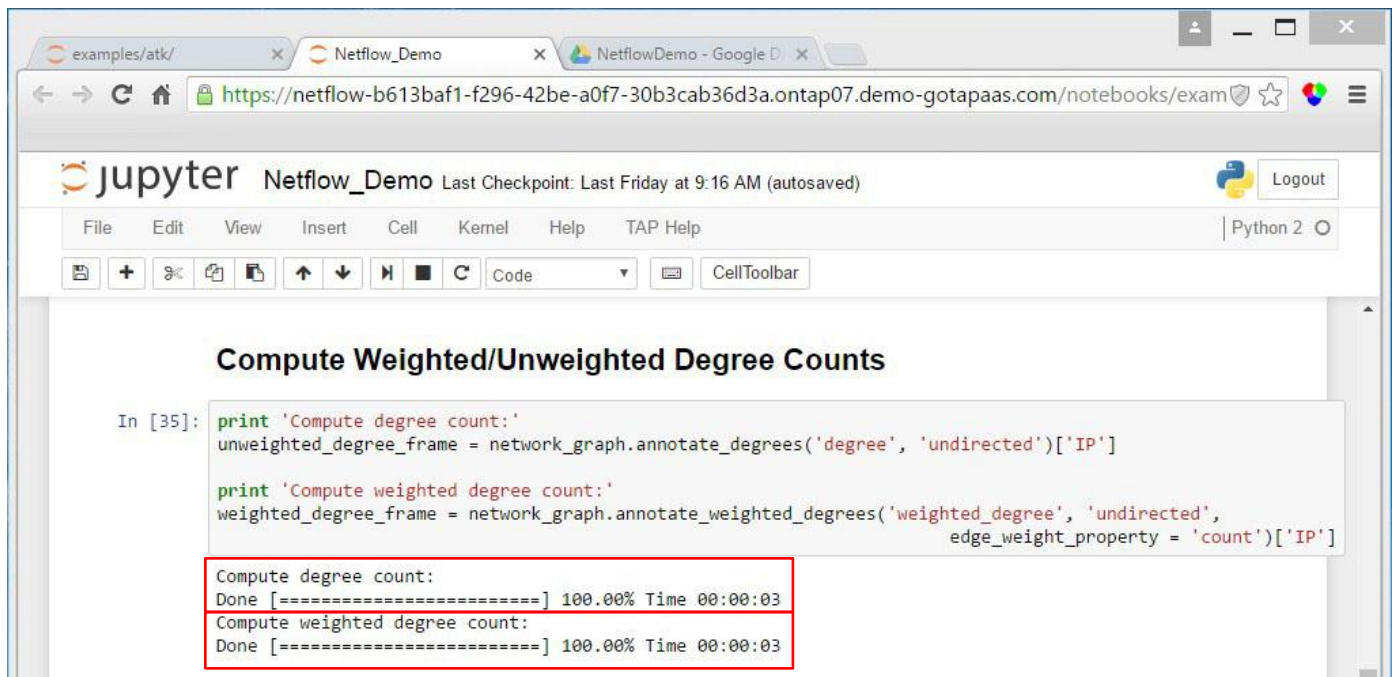
Done [=====] 100.00% Time 00:00:02
Done [=====] 100.00% Time 00:00:01
Done [=====] 100.00% Time 00:00:01
Done [=====] 100.00% Time 00:00:02
Done [=====] 100.00% Time 00:00:01
Done [=====] 100.00% Time 00:00:03
Done [=====] 100.00% Time 00:00:01
Done [=====] 100.00% Time 00:00:02

Out[34]:

#	sip	outgoing_connections	obyt_SUM	ibyt_SUM	ln_ibyt_SUM	ln_obyt_SUM
[0]	172.20.2.49	85	37423	74	4.3041	10.5300
[1]	172.30.1.101	103	46712	74	4.3041	10.7518
[2]	172.30.2.49	90	40825	74	4.3041	10.6170
[3]	172.30.1.102	122	55323	74	4.3041	10.9209
[4]	172.30.1.103	102	46253	74	4.3041	10.7419
[5]	172.30.1.204	127	57602	3069844	14.9371	10.9613
[6]	10.0.0.6	1	3095112	5535670	15.5267	14.9453
[7]	172.20.0.15	3	270	52021	10.8594	5.5984
[8]	10.0.0.9	1	3069770	5584669	15.5355	14.9371
[9]	172.20.0.5	2	180	4928	8.5027	5.1930

5.6. Compute weighted and unweighted degree counts:

- The degree of a node is the number of relations (edges) it has.
- Relations (edges) are borders between 2 IP nodes. If a node has a high degree, it means it has a lot of nodes as neighbors.
- The weighted degree of a node is like the degree. It's based on the number of edges for a node but is influenced by the weight of each edge.
- In addition to calculating the degree counts, this script identifies the 2 tasks it runs using print statements, e.g., print 'Compute degree count:' and print 'Compute weighted degree count:'. Notice the printout located just above each of the Tasks %.



The screenshot shows a Jupyter Notebook titled "Netflow_Demo" in a web browser. The notebook contains a code cell with the following Python code:

```
In [35]: print 'Compute degree count:'
unweighted_degree_frame = network_graph.annotate_degrees('degree', 'undirected')['IP']

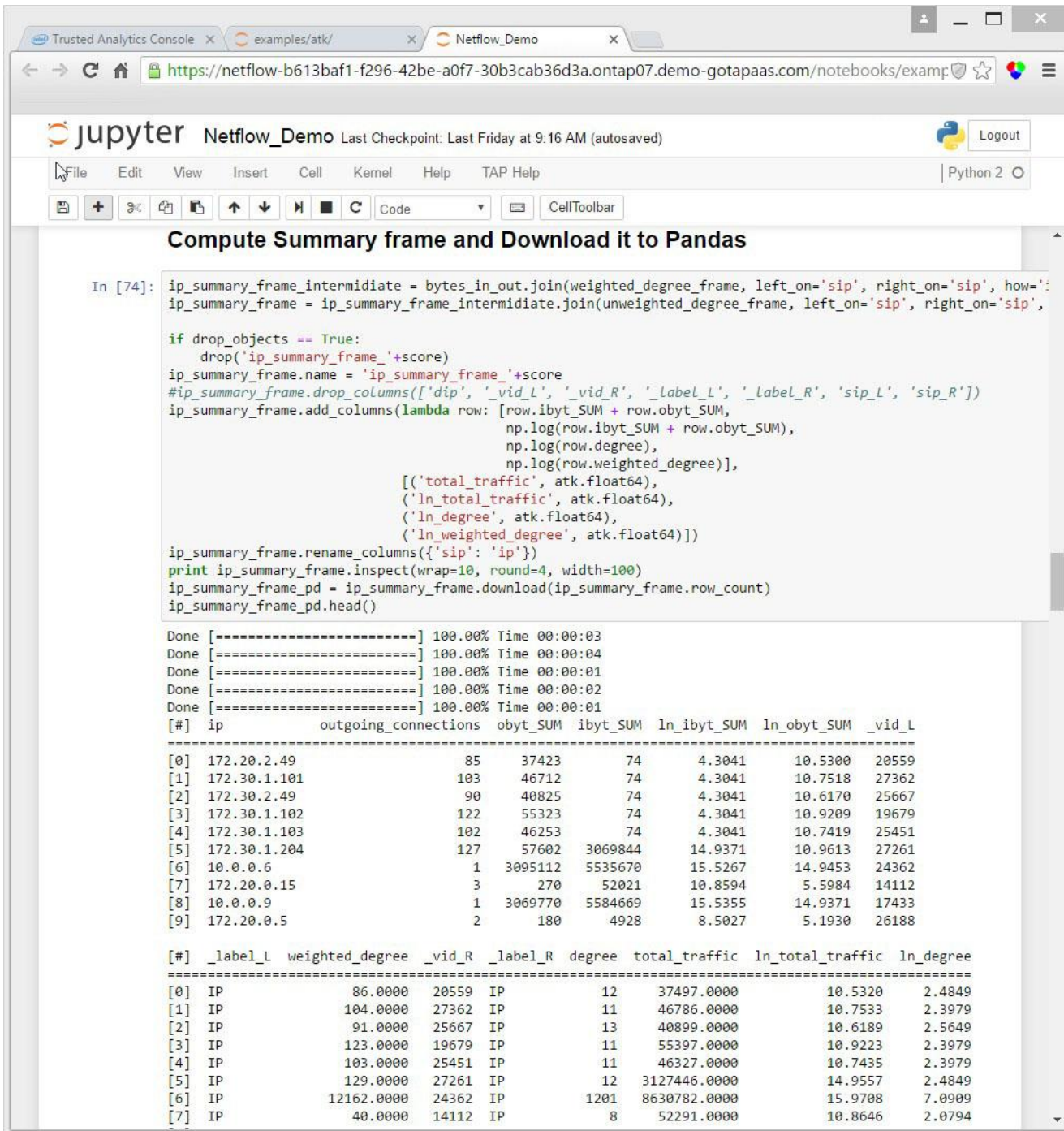
print 'Compute weighted degree count:'
weighted_degree_frame = network_graph.annotate_weighted_degrees('weighted_degree', 'undirected',
                                                                edge_weight_property = 'count')['IP']
```

The output of the code cell is displayed below the code, enclosed in a red box:

```
Compute degree count:
Done [=====] 100.00% Time 00:00:03
Compute weighted degree count:
Done [=====] 100.00% Time 00:00:03
```

5.7. Compute summary frame and download to Pandas:

- A data frame is a table, or two-dimensional array-like structure, in which each column contains measurements on one variable and each row contains one case. A summary frame is the sum of each row or column.
- This script prints the summary frame results in a tabular format.
- Additionally, this script downloads the data to your local machine. This allows you to execute commands and scripts against the sample data set.



```
In [74]: ip_summary_frame_intermediate = bytes_in_out.join(weighted_degree_frame, left_on='sip', right_on='sip', how=':')
ip_summary_frame = ip_summary_frame_intermediate.join(unweighted_degree_frame, left_on='sip', right_on='sip',
how=':')

if drop_objects == True:
    drop('ip_summary_frame_'+score)
ip_summary_frame.name = 'ip_summary_frame_'+score
#ip_summary_frame.drop_columns(['_dip', '_vid_L', '_vid_R', '_label_L', '_label_R', 'sip_L', 'sip_R'])
ip_summary_frame.add_columns(lambda row: [row.ibyt_SUM + row.obyt_SUM,
np.log(row.ibyt_SUM + row.obyt_SUM),
np.log(row.degree),
np.log(row.weighted_degree)],
[('total_traffic', atk.float64),
('ln_total_traffic', atk.float64),
('ln_degree', atk.float64),
('ln_weighted_degree', atk.float64)])
ip_summary_frame.rename_columns({'sip': 'ip'})
print ip_summary_frame.inspect(wrap=10, round=4, width=100)
ip_summary_frame_pd = ip_summary_frame.download(ip_summary_frame.row_count)
ip_summary_frame_pd.head()
```

```
Done [=====] 100.00% Time 00:00:03
Done [=====] 100.00% Time 00:00:04
Done [=====] 100.00% Time 00:00:01
Done [=====] 100.00% Time 00:00:02
Done [=====] 100.00% Time 00:00:01
```

#	ip	outgoing_connections	obyte_SUM	ibyte_SUM	ln_ibyte_SUM	ln_obyte_SUM	_vid_L
[0]	172.20.2.49	85	37423	74	4.3041	10.5300	20559
[1]	172.30.1.101	103	46712	74	4.3041	10.7518	27362
[2]	172.30.2.49	90	40825	74	4.3041	10.6170	25667
[3]	172.30.1.102	122	55323	74	4.3041	10.9209	19679
[4]	172.30.1.103	102	46253	74	4.3041	10.7419	25451
[5]	172.30.1.204	127	57602	3069844	14.9371	10.9613	27261
[6]	10.0.0.6	1	3095112	5535670	15.5267	14.9453	24362
[7]	172.20.0.15	3	270	52021	10.8594	5.5984	14112
[8]	10.0.0.9	1	3069770	5584669	15.5355	14.9371	17433
[9]	172.20.0.5	2	180	4928	8.5027	5.1930	26188

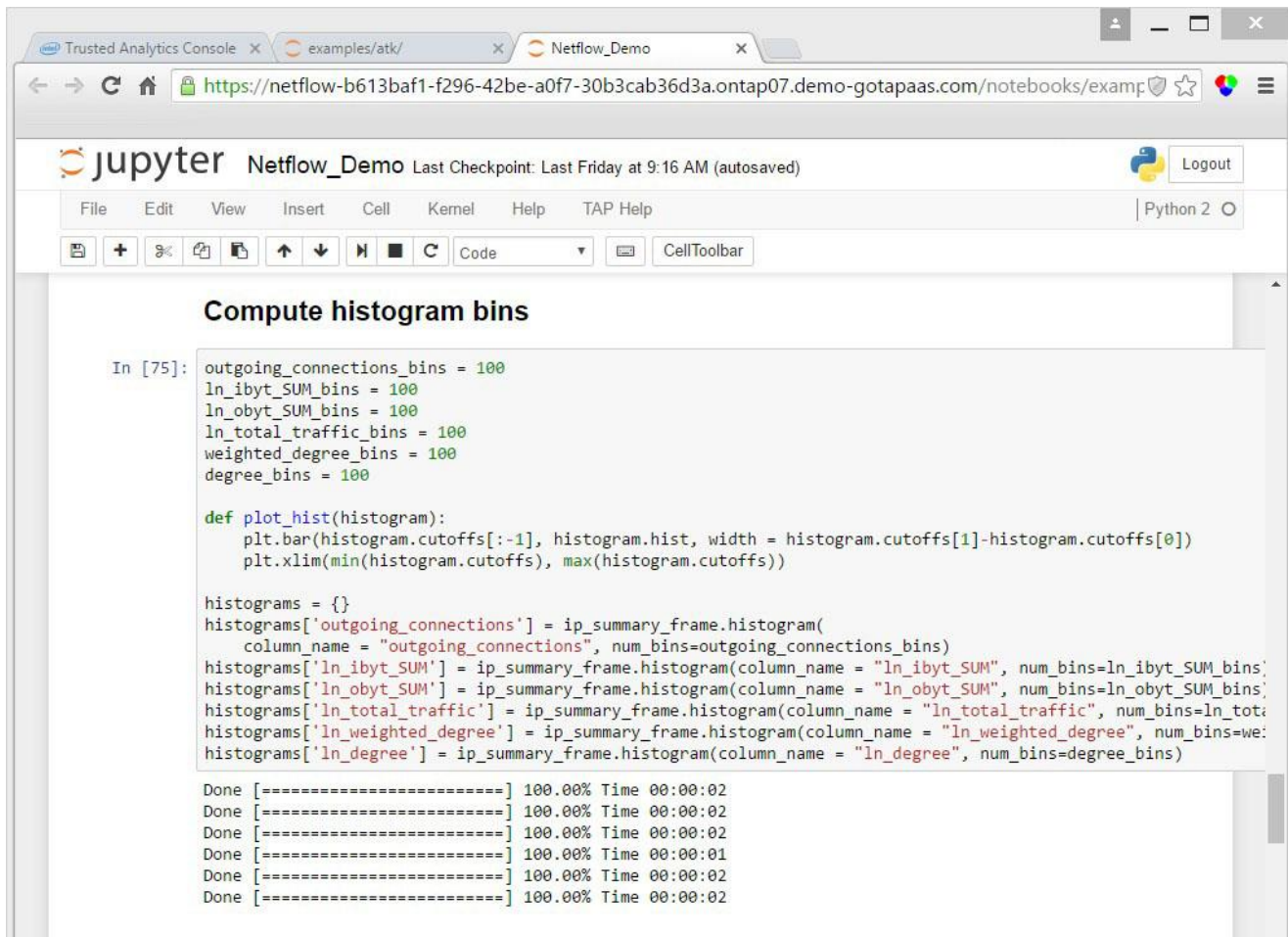
#	_label_L	weighted_degree	_vid_R	_label_R	degree	total_traffic	ln_total_traffic	ln_degree
[0]	IP	86.0000	20559	IP	12	37497.0000	10.5320	2.4849
[1]	IP	104.0000	27362	IP	11	46786.0000	10.7533	2.3979
[2]	IP	91.0000	25667	IP	13	40899.0000	10.6189	2.5649
[3]	IP	123.0000	19679	IP	11	55397.0000	10.9223	2.3979
[4]	IP	103.0000	25451	IP	11	46327.0000	10.7435	2.3979
[5]	IP	129.0000	27261	IP	12	3127446.0000	14.9557	2.4849
[6]	IP	12162.0000	24362	IP	1201	8630782.0000	15.9708	7.0909
[7]	IP	40.0000	14112	IP	8	52291.0000	10.8646	2.0794

- This is only a sample of the summary frame table.

	ip	outgoing_connections	obyte_SUM	ibyte_SUM	ln_ibyte_SUM	ln_obyte_SUM	_vid_L	_label_L	weigh
0	172.20.2.49	85	37423	74	4.304065	10.530041	20559	IP	86.0
1	172.30.1.101	103	46712	74	4.304065	10.751756	27362	IP	104.0
2	172.30.2.49	90	40825	74	4.304065	10.617050	25667	IP	91.0
3	172.30.1.102	122	55323	74	4.304065	10.920944	19679	IP	123.0
4	172.30.1.103	102	46253	74	4.304065	10.741882	25451	IP	103.0

5.8. Compute histogram bins:

- A histogram is a visual interpretation of numerical data indicating the number of data points that lie within a range of values, called a class or a bin. The frequency of the data in each class is depicted by the use of a bar.
- This script separates the data into six groups—outgoing connections, in byte sum, out byte sum, total traffic, weighted degree and degree.
- It displays the results in 6 histograms, seen on the next page.



```
In [75]: outgoing_connections_bins = 100
ln_ibyt_SUM_bins = 100
ln_obyt_SUM_bins = 100
ln_total_traffic_bins = 100
weighted_degree_bins = 100
degree_bins = 100

def plot_hist(histogram):
    plt.bar(histogram.cutoffs[:-1], histogram.hist, width = histogram.cutoffs[1]-histogram.cutoffs[0])
    plt.xlim(min(histogram.cutoffs), max(histogram.cutoffs))

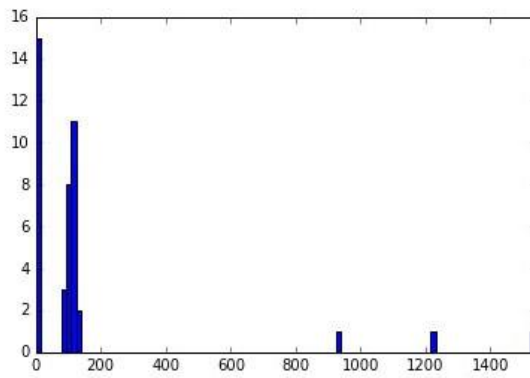
histograms = {}
histograms['outgoing_connections'] = ip_summary_frame.histogram(
    column_name = "outgoing_connections", num_bins=outgoing_connections_bins)
histograms['ln_ibyt_SUM'] = ip_summary_frame.histogram(column_name = "ln_ibyt_SUM", num_bins=ln_ibyt_SUM_bins)
histograms['ln_obyt_SUM'] = ip_summary_frame.histogram(column_name = "ln_obyt_SUM", num_bins=ln_obyt_SUM_bins)
histograms['ln_total_traffic'] = ip_summary_frame.histogram(column_name = "ln_total_traffic", num_bins=ln_total_traffic_bins)
histograms['ln_weighted_degree'] = ip_summary_frame.histogram(column_name = "ln_weighted_degree", num_bins=weighted_degree_bins)
histograms['ln_degree'] = ip_summary_frame.histogram(column_name = "ln_degree", num_bins=degree_bins)

Done [=====] 100.00% Time 00:00:02
Done [=====] 100.00% Time 00:00:02
Done [=====] 100.00% Time 00:00:02
Done [=====] 100.00% Time 00:00:01
Done [=====] 100.00% Time 00:00:02
Done [=====] 100.00% Time 00:00:02
```

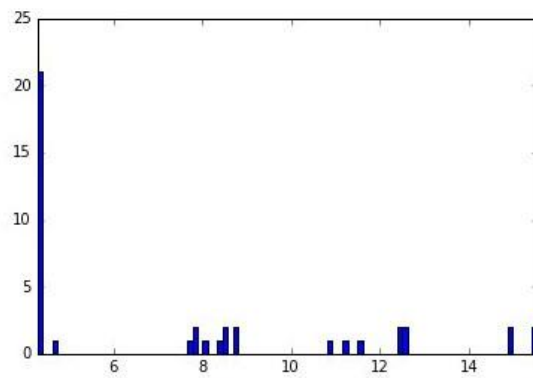
- Six histograms:

Plot histograms

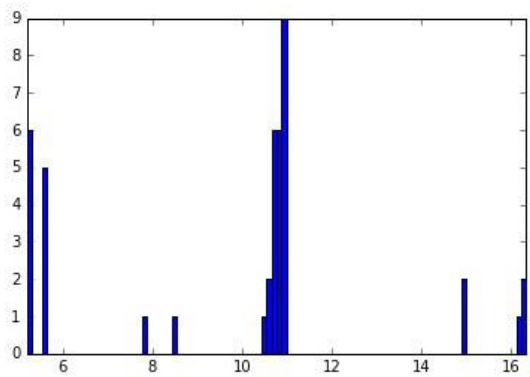
```
plot_hist(histograms['outgoing_connections'])
```



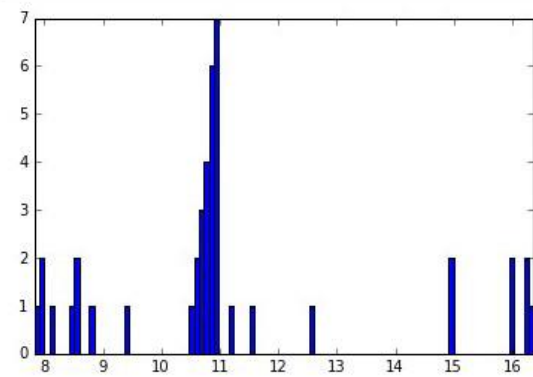
```
plot_hist(histograms['ln_ibyt_SUM'])
```



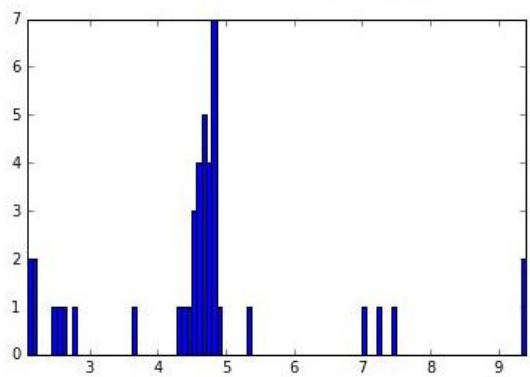
```
plot_hist(histograms['ln_obyt_SUM'])
```



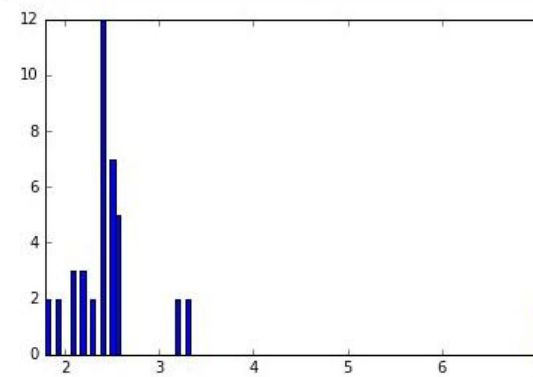
```
plot_hist(histograms['ln_total_traffic'])
```



```
plot_hist(histograms['ln_weighted_degree'])
```

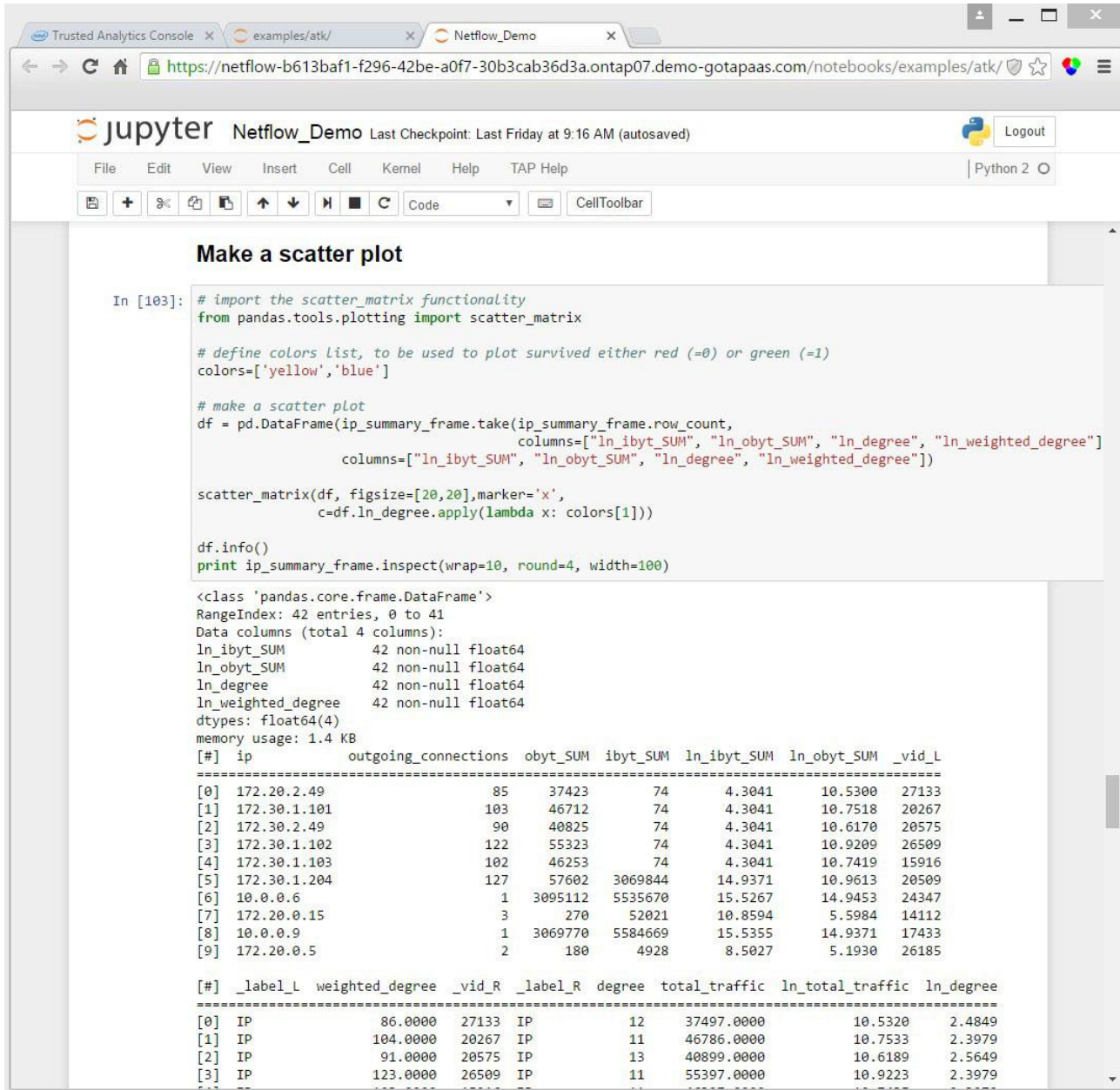


```
plot_hist(histograms['ln_degree'])
```



5.9. Make a Scatter Plot

- A scatter (XY) plot has points that show the relationship between 2 sets of data. The relationship between two variables is called a correlation.
- This script uses 4 groups of data—in byte sum, out byte sum, degree, and weighted degree.
- It creates a four-by-four scatter-plot matrix.



The screenshot shows a Jupyter Notebook interface with a browser window at the top. The notebook title is "Netflow_Demo" and it shows the last checkpoint at 9:16 AM. The code cell is titled "Make a scatter plot" and contains the following Python code:

```
In [103]: # import the scatter_matrix functionality
from pandas.tools.plotting import scatter_matrix

# define colors list, to be used to plot survived either red (=0) or green (=1)
colors=['yellow','blue']

# make a scatter plot
df = pd.DataFrame(ip_summary_frame.take(ip_summary_frame.row_count,
                                       columns=["ln_ibyt_SUM", "ln_obyt_SUM", "ln_degree", "ln_weighted_degree"]
                                       columns=["ln_ibyt_SUM", "ln_obyt_SUM", "ln_degree", "ln_weighted_degree"]))

scatter_matrix(df, figsize=[20,20],marker='x',
               c=df.ln_degree.apply(lambda x: colors[1]))

df.info()
print ip_summary_frame.inspect(wrap=10, round=4, width=100)
```

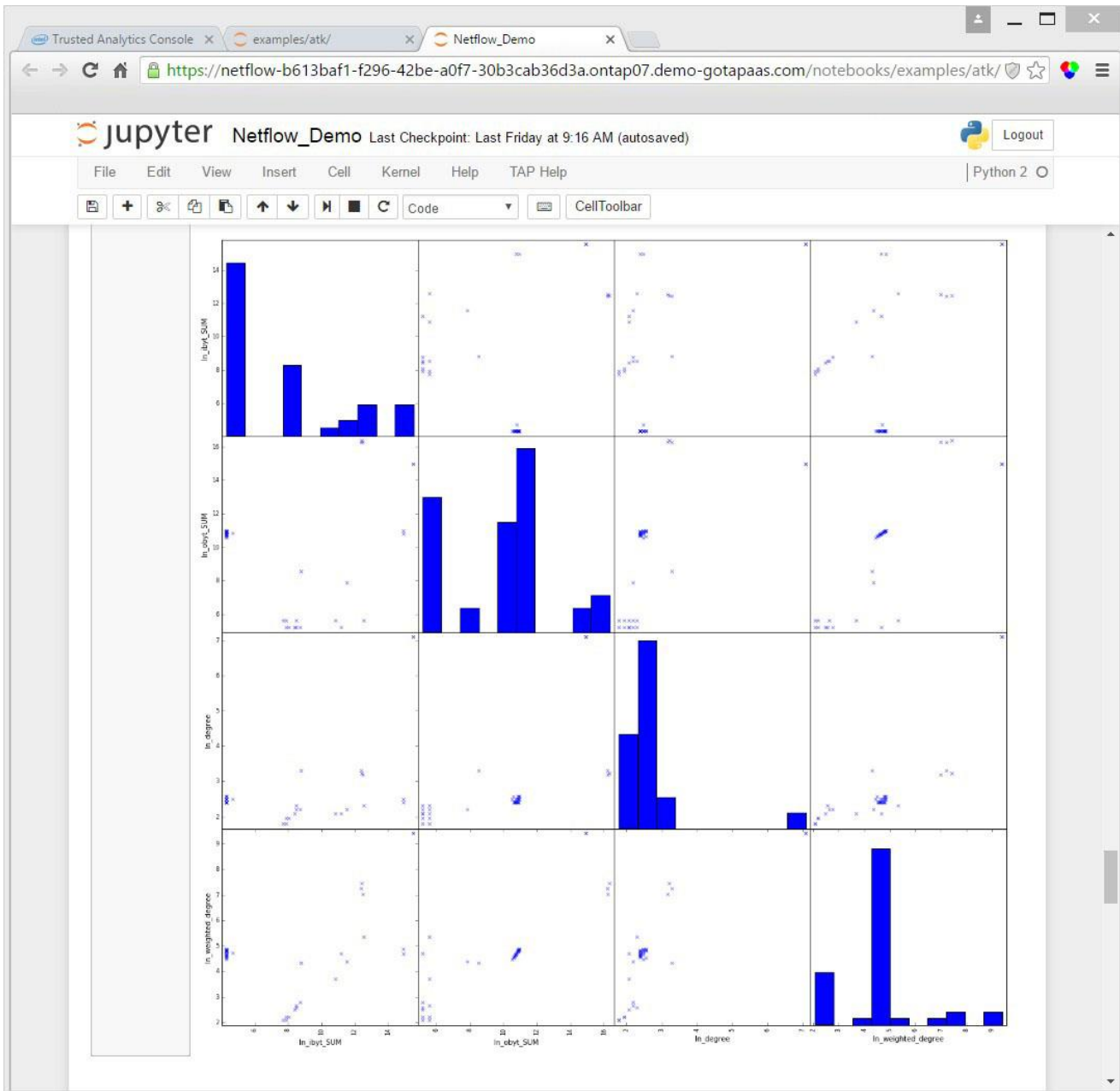
The output of the code is as follows:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 42 entries, 0 to 41
Data columns (total 4 columns):
ln_ibyt_SUM      42 non-null float64
ln_obyt_SUM      42 non-null float64
ln_degree        42 non-null float64
ln_weighted_degree 42 non-null float64
dtypes: float64(4)
memory usage: 1.4 KB
```

[#]	ip	outgoing_connections	obyt_SUM	ibyt_SUM	ln_ibyt_SUM	ln_obyt_SUM	_vid_L
[0]	172.20.2.49	85	37423	74	4.3041	10.5300	27133
[1]	172.30.1.101	103	46712	74	4.3041	10.7518	20267
[2]	172.30.2.49	90	40825	74	4.3041	10.6170	20575
[3]	172.30.1.102	122	55323	74	4.3041	10.9209	26509
[4]	172.30.1.103	102	46253	74	4.3041	10.7419	15916
[5]	172.30.1.204	127	57602	3069844	14.9371	10.9613	20509
[6]	10.0.0.6	1	3095112	5535670	15.5267	14.9453	24347
[7]	172.20.0.15	3	270	52021	10.8594	5.5984	14112
[8]	10.0.0.9	1	3069770	5584669	15.5355	14.9371	17433
[9]	172.20.0.5	2	180	4928	8.5027	5.1930	26185

[#]	_label_L	weighted_degree	_vid_R	_label_R	degree	total_traffic	ln_total_traffic	ln_degree
[0]	IP	86.0000	27133	IP	12	37497.0000	10.5320	2.4849
[1]	IP	104.0000	20267	IP	11	46786.0000	10.7533	2.3979
[2]	IP	91.0000	20575	IP	13	40899.0000	10.6189	2.5649
[3]	IP	123.0000	26509	IP	11	55397.0000	10.9223	2.3979

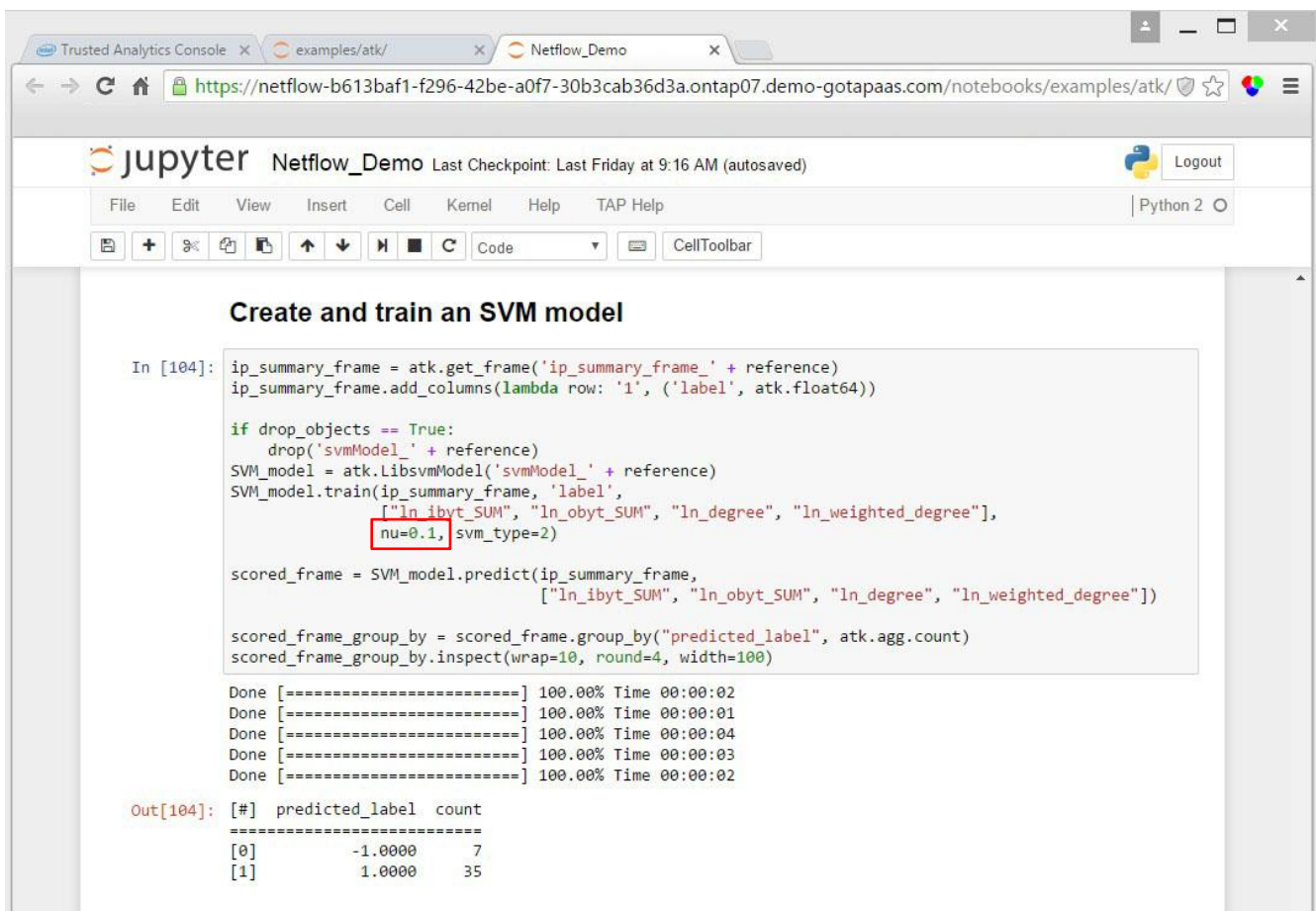
- Scatter-plot matrix:



- The scatter-plot matrix is replicated in Module 2: Visualizing Your Data with an App.

5.10. Create and train an SVM model:

- The script calculates the normal distribution of network traffic and the outliers. It detects anomalies on the network, such as DOS attack.
- The **nu** value determines the precision of our calculation. The value is the reciprocal of the dispersive power of a medium, known as constringency.
- After completing the remaining workshop steps, we encourage you to change the nu value in this script to see how it affects the data.
- Changing the nu value:
 - You **must** rerun step 5.7 “**Compute summary frame and download to Pandas**” each time you change the nu value.
 - Then run this step and the last step in the workshop: 5.11 “**Download the scatter frame to Pandas.**”
 - Change the nu value to 0.2 or higher and notice how the graphs change.
 - Locate the upper limit of the nu value.



```
In [104]: ip_summary_frame = atk.get_frame('ip_summary_frame_' + reference)
ip_summary_frame.add_columns(lambda row: '1', ('label', atk.float64))

if drop_objects == True:
    drop('svmModel_' + reference)
SVM_model = atk.LibsvmModel('svmModel_' + reference)
SVM_model.train(ip_summary_frame, 'label',
                ["ln_ibyt_SUM", "ln_obyt_SUM", "ln_degree", "ln_weighted_degree"],
                nu=0.1, svm_type=2)

scored_frame = SVM_model.predict(ip_summary_frame,
                                ["ln_ibyt_SUM", "ln_obyt_SUM", "ln_degree", "ln_weighted_degree"])

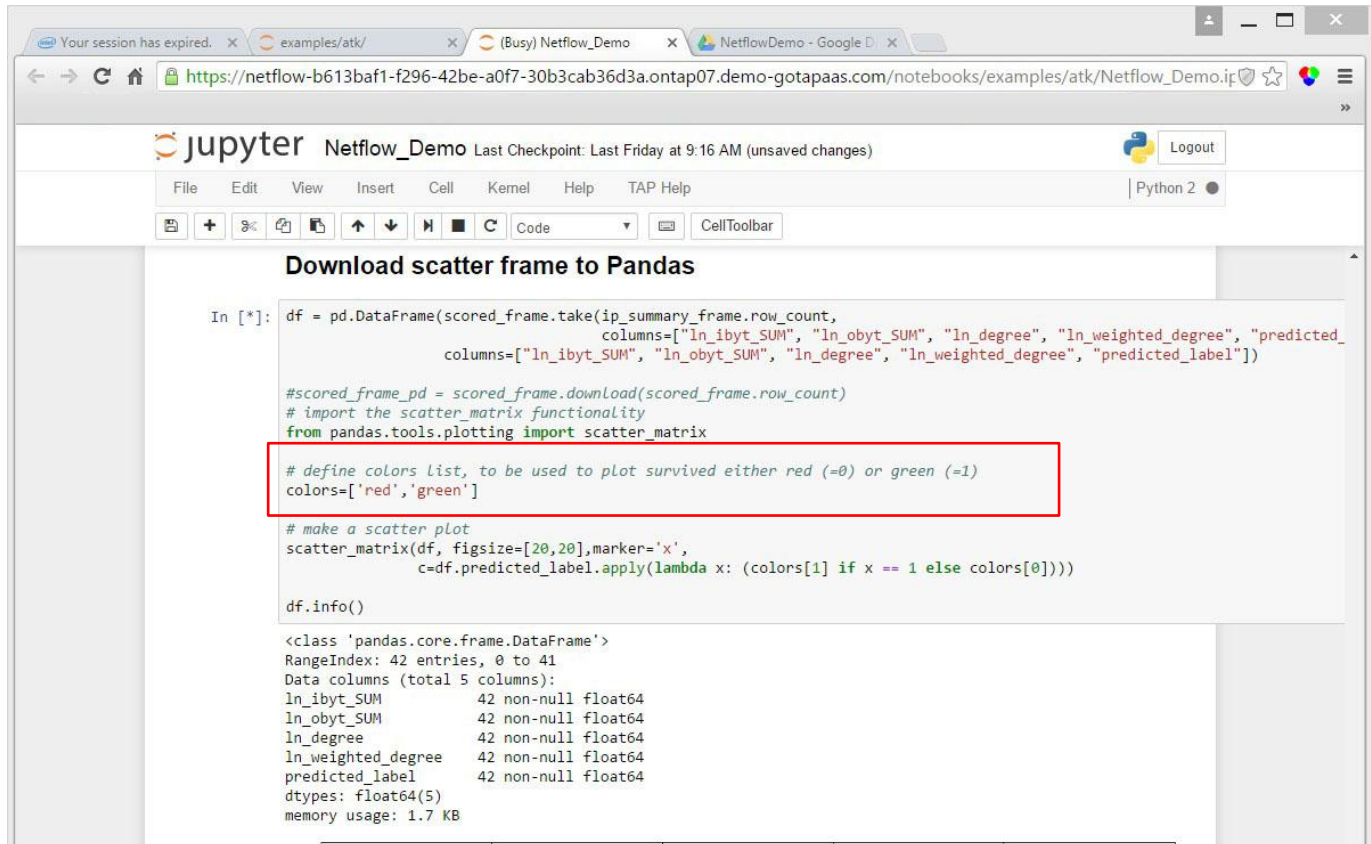
scored_frame_group_by = scored_frame.group_by("predicted_label", atk.agg.count)
scored_frame_group_by.inspect(wrap=10, round=4, width=100)

Done [=====] 100.00% Time 00:00:02
Done [=====] 100.00% Time 00:00:01
Done [=====] 100.00% Time 00:00:04
Done [=====] 100.00% Time 00:00:03
Done [=====] 100.00% Time 00:00:02

Out[104]: [#] predicted_label count
=====
[0]      -1.0000      7
[1]       1.0000     35
```


5.11. Download the scatter frame to Pandas:

- This script creates plot graphs from the results calculated in the “Create and train an SVM model” script in the previous step.
- The areas in **green** show the normal distribution. The points in **red** show the outliers.
- The points that are away from the “green cloud” in each frame should be investigated, as they fall outside the normal traffic patterns.
- To change the graph’s colors, modify the *# define colors list* in the script.



```
In [*]: df = pd.DataFrame(scored_frame.take(ip_summary_frame.row_count,
                                         columns=["ln_ibyt_SUM", "ln_obyt_SUM", "ln_degree", "ln_weighted_degree", "predicted_label"],
                                         columns=["ln_ibyt_SUM", "ln_obyt_SUM", "ln_degree", "ln_weighted_degree", "predicted_label"]))

#scored_frame_pd = scored_frame.download(scored_frame.row_count)
# import the scatter_matrix functionality
from pandas.tools.plotting import scatter_matrix

# define colors list, to be used to plot survived either red (=0) or green (=1)
colors=['red','green']

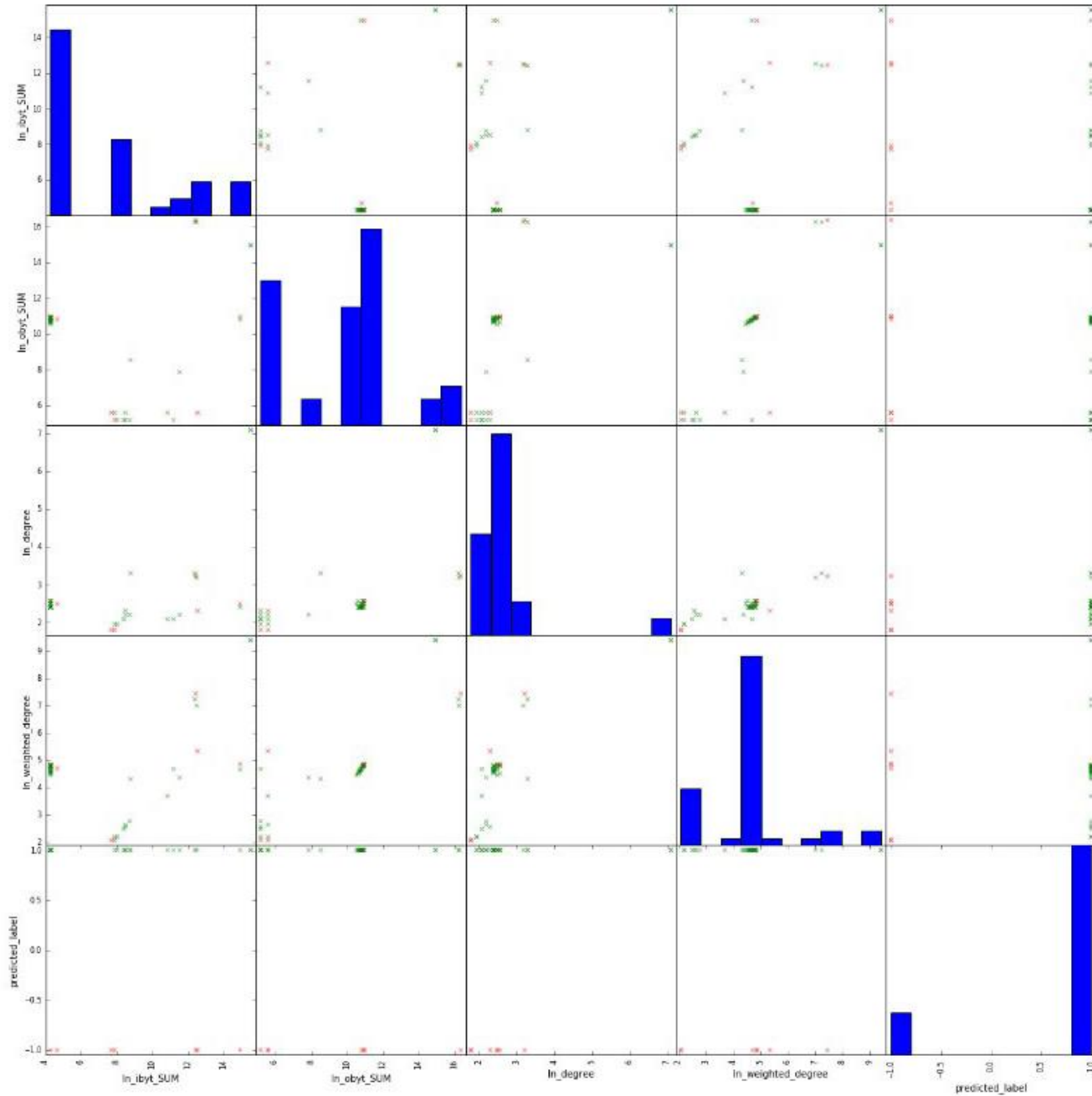
# make a scatter plot
scatter_matrix(df, figsize=[20,20],marker='x',
               c=df.predicted_label.apply(lambda x: (colors[1] if x == 1 else colors[0])))

df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 42 entries, 0 to 41
Data columns (total 5 columns):
ln_ibyt_SUM      42 non-null float64
ln_obyt_SUM      42 non-null float64
ln_degree        42 non-null float64
ln_weighted_degree 42 non-null float64
predicted_label   42 non-null float64
dtypes: float64(5)
memory usage: 1.7 KB
```

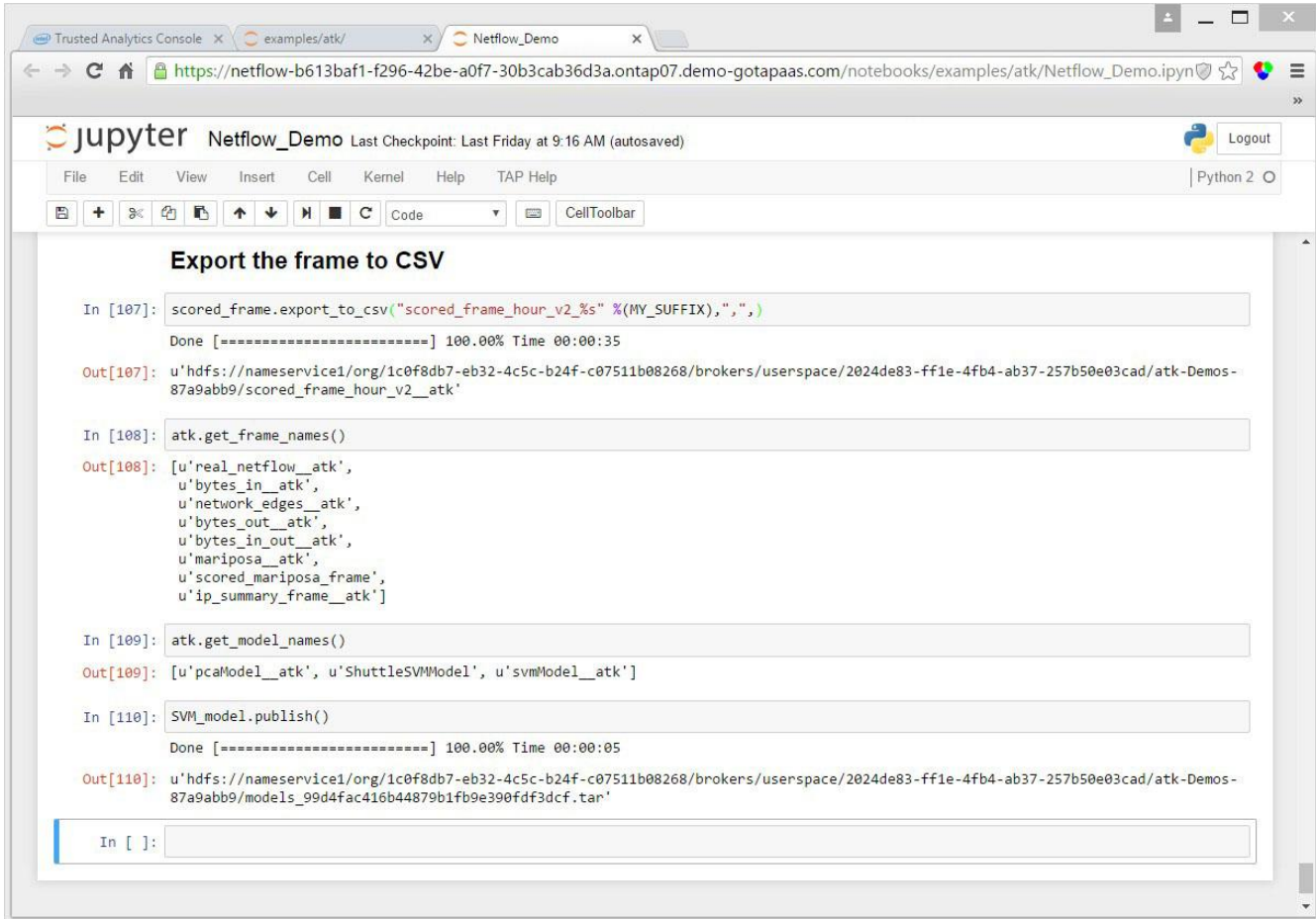
- See the scatter-plot graphs on the next page.

- Scatter-plot graphs:



6. Export the data to HDFS as a CSV file

- This script exports the data to CSV files, e.g., Microsoft Excel.



The screenshot shows a Jupyter Notebook titled "Netflow_Demo" in a web browser. The notebook contains several code cells and their outputs. The first cell exports a frame to CSV, the second lists frame names, the third lists model names, and the fourth publishes an SVM model. The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Help, TAP Help), a toolbar with icons for file operations and execution, and a status bar at the bottom.

```
Export the frame to CSV

In [107]: scored_frame.export_to_csv("scored_frame_hour_v2_%s" % (MY_SUFFIX), ",", ",")
Done [=====] 100.00% Time 00:00:35

Out[107]: u'hdfs://nameservice1/org/1c0f8db7-eb32-4c5c-b24f-c07511b08268/brokers/userspace/2024de83-ff1e-4fb4-ab37-257b50e03cad/atk-Demos-87a9abb9/scored_frame_hour_v2_atk'

In [108]: atk.get_frame_names()

Out[108]: [u'real_netflow_atk',
u'bytes_in_atk',
u'network_edges_atk',
u'bytes_out_atk',
u'bytes_in_out_atk',
u'mariposa_atk',
u'scored_mariposa_frame',
u'ip_summary_frame_atk']

In [109]: atk.get_model_names()

Out[109]: [u'pcaModel_atk', u'ShuttleSVMModel', u'svmModel_atk']

In [110]: SVM_model.publish()
Done [=====] 100.00% Time 00:00:05

Out[110]: u'hdfs://nameservice1/org/1c0f8db7-eb32-4c5c-b24f-c07511b08268/brokers/userspace/2024de83-ff1e-4fb4-ab37-257b50e03cad/atk-Demos-87a9abb9/models_99d4fac416b44879b1fb9e390fdf3dcf.tar'

In [ ]:
```


- The CSV file will not be available to view in your TAP environment located in the Data Catalog tab.

The screenshot displays the TAP Data Catalog interface. The left sidebar contains navigation links: Dashboard, Data catalog (highlighted with a red arrow), Job Scheduler, Applications, Services, App Development, Data Science, User management, and Event Log. The main content area is titled 'Data sets' and shows 'Found 2 datasets'. An 'Advanced search' section includes a 'Keyword' field with 'scored', a 'Format' dropdown set to 'all' (with 'csv' also visible), and a 'Creation time' range selector. Below this, a 'Tool for visualization' section has radio buttons for 'Arcadia' (selected) and 'Hue'. A 'Choose category' section shows two 'all' category buttons. The search results list two datasets: 'scored_frame_hour_atk' and 'scored_frame_hour_v2_...', each with a '+ View' button. A red arrow points to the second dataset. The bottom of the results area indicates 'Showing 2 of 2'.

CONGRATULATIONS!

You have successfully completed the workshop.