

TAP Hands-On Workshop

Module 1: Performing Analytics on Your Data

Overview

This module uses the Analytics Toolkit (ATK) 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.

Steps	Time Required
Step 1	Varies based on your current environment
Step 2	15 minutes
Step 3	5 minutes
Step 4	5 minutes
Step 5	30 minutes
Step 6	5 minutes
Total Workshop Time	60 minutes

Setup

The data scientist needs to prepare his or her local environment/computer with the tools required to edit, assemble and deploy the module. **Important:** You need to have access to an instance or to set up an instance of ATK.

- The software you need includes:
 - **iPython Notebook** (also known as Jupyter) - a tool for exploratory computation and data analysis that stores inputs/outputs in notebook documents. <http://ipython.org/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 ATK on your system.
- Used Python or iPython Notebook.
- Accessed the help functions for editing a script.
- Shown how to attach a TAP server to your project to input data.
- Executed an ATK 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 ATK 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 iPython Notebook, review the help functions and edit the ATK.server.uri field in the file.
5. Execute the Netflow demo to deploy the ATK 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 Titan graph for computing statistics, computes bytes in and bytes out, 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

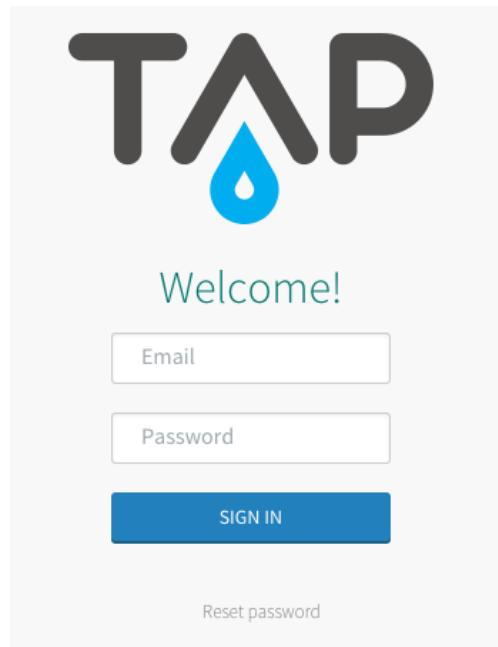
This section provides step-by-step instructions and their results.

1. Install all necessary developer tools listed in Setup

- 1.1. No additional instructions are provided for this step.

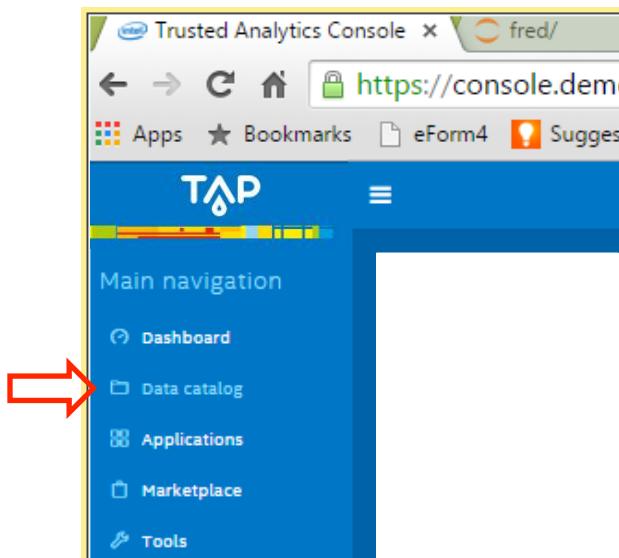
2. TAP Marketplace

- 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:
 - You can see your datasets and all the datasets marked public.
 - In this example, we are in the ATK-Demo group in Marketplace. The ATK-Demo group is only available in the TAP instance. The group name may be different if you use your own instance.
 - You can limit the number of datasets you see by either using the Advanced search option or choosing a specific category.
 - **Note:** Make sure you select the correct group when you look for your dataset in Marketplace.

The screenshot shows the 'Data sets' tab selected in the top navigation bar. Below it, a message says 'Found 18 datasets'. The interface includes search and filter options: 'Advanced search' (with a red arrow pointing to it), 'Format' (all, csv), 'Creation time' (from: [] to: []), and 'Tool for visualization' (Arcadia, Hue). A 'Choose category:' section has five buttons: 'all' (highlighted with a blue box and red arrow), 'other', 'business', 'finance', and 'science'. Below this is a grid of 18 dataset cards, each with a preview icon, name, and a 'View' button. The first few cards are: 'test-transfer-1437427519.6781388', 'transfer906099', 'cities', 'testest'; '1119', 'transfer1', 'transfer534484', 'test-transfer-143747000.9196236'; 'test145', 'movies2', 'xxxxxx', 'test'.

- Click the **View** button on the Data sets tab to display dataset information, such as:
 - Title, size, category ...
 - **TargetUri** is a link to a dataset. This particular dataset link may not be the one needed in this exercise. You need to consult the workshop instructor or your DevOps team to obtain the proper information
 - You can copy the link to use in other apps.

NF - week2

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,contFragmen
recordCount	23258685
isPublic	true
targetUri	hdfs://nameservice1/cf/broker/instances/b570125c-d786-48d6-b388-3c0e678eb005/39fe148a-ecc3-4962-8a75-21b61df75b73/000000_1
category	science
format	CSV
creationTime	2015-05-23T21:46:53.445469

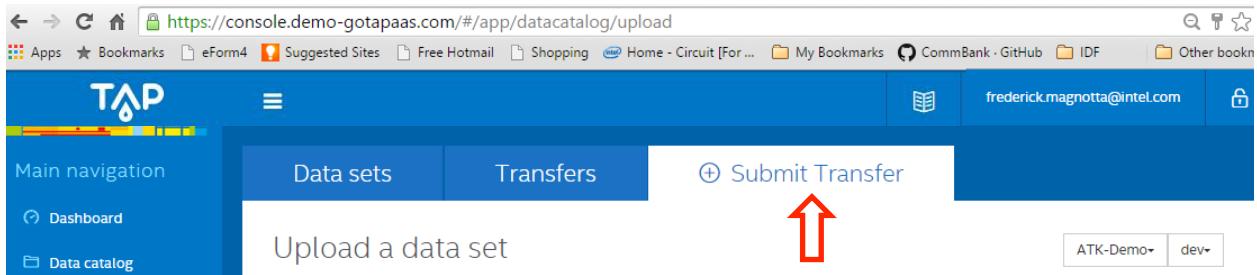
- Click on the **Transfers** tab to see what files have been uploaded to TAP and to get information on when a file was added to the system.

Click on the column header to sort the files by creation date, state ...

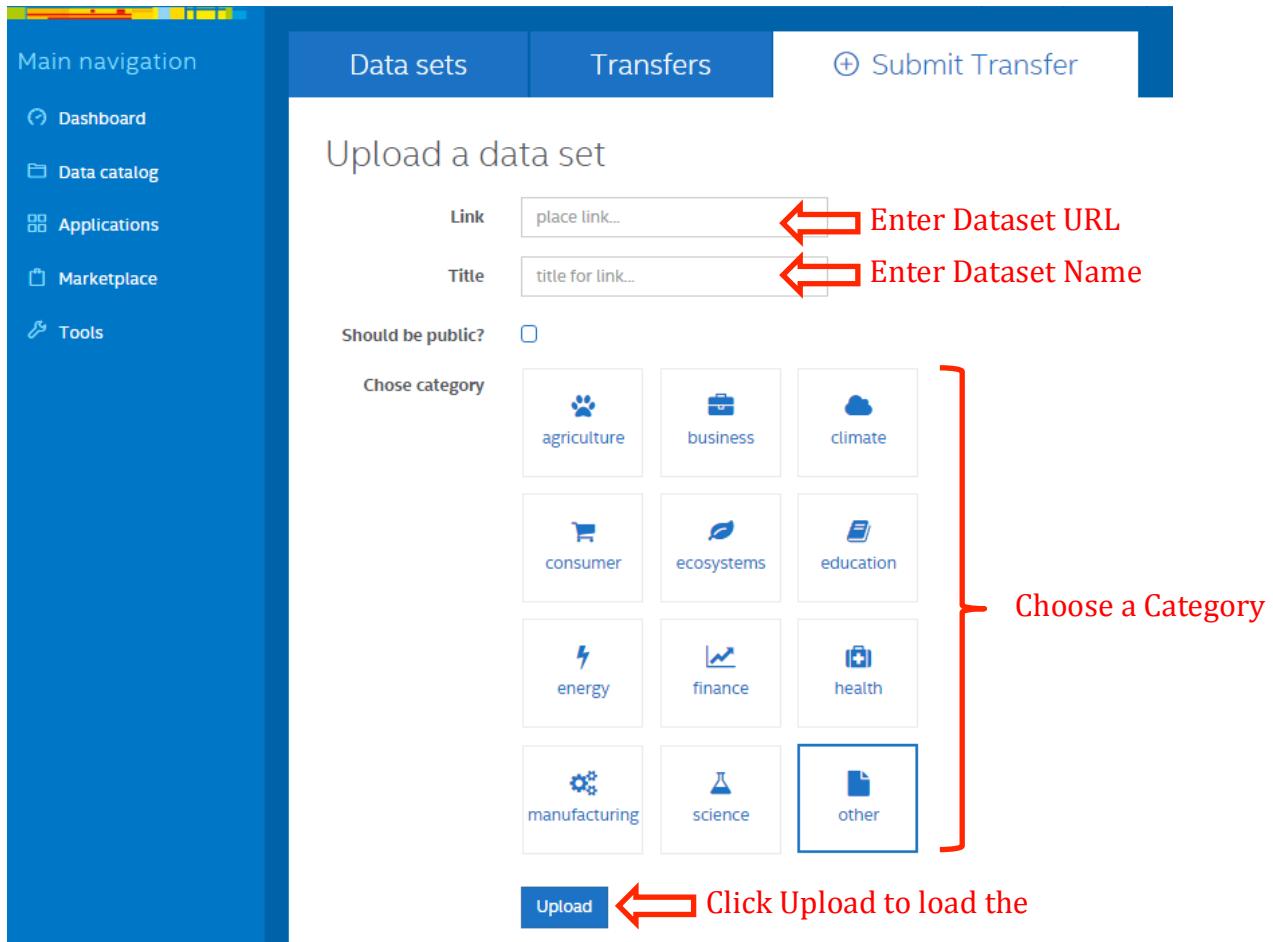
Source	Created	Last state	Category	Title	Item
http://www.vacommunity.org/VAST+Challenge+2013%3A+Mini-Challenge+3	25.06.2015 08:15:16	NEW 25.06.2015 08:15:16	other	VAST_mini_challange	
http://www.vacommunity.org/VAST+Challenge+2013%3A+Mini-Challenge+3	24.06.2015 08:21:27	NEW 24.06.2015 08:21:27	other	VAST_mini_challange	
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.

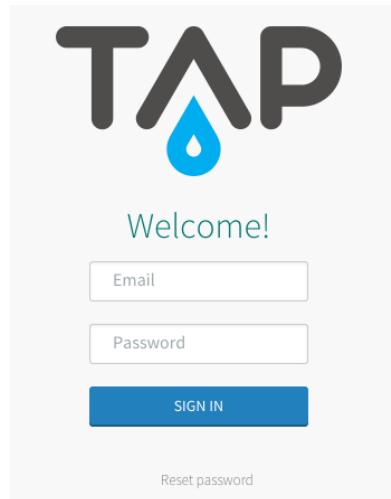


- Complete the Submit Transfer fields:
 - **Category** – Click on a category, based on your needs.
 - **Link field** – Enter the URL of the dataset you want to upload.
 - **Title field** – Enter a file name for your dataset.
- Click on Upload after you have entered the necessary information.



3. TAP Marketplace

- If you are not already logged into Marketplace, sign in with your username and password.



3.1. Create a new instance of ATK in TAP:

- Click on Marketplace under Main navigation.
- Click on ATK.

A screenshot of the TAP Marketplace page. On the left is a vertical 'Main navigation' sidebar with options: Dashboard, Data catalog, Applications, Marketplace (which has a red arrow pointing to it), and Tools. The main area is titled 'Marketplace' and displays several service cards. One card for 'ArangoDB 2.2' is visible. A second card for 'atk' (Intel Analytics Toolkit) is highlighted with a red border. This card includes an icon of a gear, the name 'atk', the description 'Intel Analytics Toolkit', and three tags: 'atk', 'analytics', and 'intel'. Other cards shown include 'Consul 0.3.1' and 'CouchDB 1.6'.

- In the ATK 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 ATK instances in this window.
- The ATK window displays all the running instances on TAP.

The screenshot shows the TAP (Tooling Application Platform) interface. On the left, there's a main navigation bar with links to Dashboard, Data catalog, Applications, Marketplace, and Tools. The central area is titled 'ATK' and contains the 'Analytics Toolkit' application. Under 'Available plans', there's a 'Simple' plan entry. A red arrow points from the text 'In the ATK window, enter a name for your instance and click Create new instance.' to the 'Web-Workshop' input field in the 'Create new instance' button. Below this, a table titled 'Running instances in space IDF/IDF-demo' lists four instances:

Name	Plan	Apps bound	Delete
atk-rr	simple		X
atk-rr-1	simple		X
atk-rr-2	simple		X
Fred-ATK-Instance	simple		X

3.2. View the running instance:

- Refresh your window to see the instance you created.

The screenshot shows the TAP (Tool for Analytics and Predictions) interface. On the left, there's a main navigation sidebar with options like Dashboard, Data catalog, Applications, Marketplace, and Tools. The main content area has two sections: 'Available plans' and 'Running instances in space IDF/IDF-demo'. In the 'Available plans' section, there's a 'Simple' plan with a blue placeholder image, labeled 'Simple' and 'Price:unknown', with a 'Create new instance' button. Below this, the 'Running instances' section lists several instances with their names, plans, and delete options. A red arrow points to the 'Web-Workshop' instance, which is highlighted in the list.

Name	Plan	Apps bound	Delete
atk-rr	simple		X
atk-rr-1	simple		X
atk-rr-2	simple		X
Fred-ATK-Instance	simple		X
jacek0901_1	simple		X
roadrunner	simple		X
Web-Workshop	simple		X

3.3. Copy the instance uri 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 uri of the instance. You will need the URL to run the Netflow demo in the next part of the workshop.
 - The red box 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 TAP (Tata Paas) application interface. On the left, there is a blue sidebar with the TAP logo at the top. Below it, the 'Main navigation' section contains links: Dashboard, Data catalog, Applications (which is highlighted with a red arrow), Marketplace, and Tools. The main content area is titled 'Applications'. It features a table with columns: Name, Status, Instances, and URLs. The table lists several services: platform-context, roadrunner, router-metrics-provider, service-catalog, smtp-broker, user-management, Web-Workshop, and zookeeper-wssb-broker. The 'Web-Workshop' row is highlighted with a red arrow and a red border around its URL cell. The URL is 'atk-d572dd00-be78-438a-960d.gotapaas.com'. There are also 'See details' links for each row. At the bottom of the table, there are navigation buttons for page 1 of 3.

Name	Status	Instances	URLs
platform-context	✓	1	platform-context.gotapaas.com See details »
roadrunner	✓	1	atk-3f07b63a-99ae-422f-8010.gotapaas.com See details »
router-metrics-provider	✓	1	router-metrics-provider.gotapaas.com See details »
service-catalog	✓	1	service-catalog.gotapaas.com See details »
smtp-broker	✓	1	smtp-broker.gotapaas.com See details »
user-management	✓	1	user-management.gotapaas.com See details »
Web-Workshop	■	0	atk-d572dd00-be78-438a-960d.gotapaas.com See details »
zookeeper-wssb-broker	✓	1	zookeeper-wssb-broker.gotapaas.com See details »

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 in Python or iPython Notebook. iPython Notebook is also known as Jupyter.
- You can find the file at <https://s3-us-west-2.amazonaws.com/analytics-tool-kit/public/datasets/latest/nf-hour.csv>.

The screenshot shows a Jupyter Notebook interface. At the top, there's a header with the Jupyter logo, a 'Logout' button, and navigation tabs for 'Files', 'Running', and 'Clusters'. Below the header is a toolbar with 'Upload', 'New', and other icons. The main area displays a file list under the path '/ fred'. The files listed are: 'Kmeans.ipynb' (Running), 'Movie_demo.ipynb', 'Netflow_Demo.ipynb' (Running), 'System Comands scratchpad.ipynb', 'atk-fred.cred', and 'atk-workshop.creds'. A red arrow points to the 'Netflow_Demo.ipynb' file.

File	Status
Kmeans.ipynb	Running
Movie_demo.ipynb	
Netflow_Demo.ipynb	Running
System Comands scratchpad.ipynb	
atk-fred.cred	
atk-workshop.creds	

4.3. Review the help functions:

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

A screenshot of a Jupyter Notebook cell. The input field contains "atk.KMeansModel". A dropdown menu is open, listing various command completions. The word "atk" is highlighted with a red box. The dropdown menu also has a red box around it. The text "Help window" is written vertically to the left of the input field.

```
scatter_matrix(df, figsize=[20,20],marker='x',
                c=df.predicted_label.apply(lambda x: (colors[1] if
atk.CollaborativeFilteringMod
atk.CsvFile
atk.EdgeFrame
atk.EdgeRule
atk.Frame
atk.Graph
atk.HiveQuery
atk.JsonFile
In [ ]: atk.KMeansModel
red_frame_%s" %(MY_SUFFIX),",")
```

In []: atk.|

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

A screenshot of a Jupyter Notebook cell. The input field contains "atk.CsvFile()". A tooltip-like box is displayed, containing the command signature and a docstring. The text "Help window" is written vertically to the right of the tooltip.

```
In [ ]: scored_frame.export_to_csv("scored_frame_%s" %(MY_SUFFIX),",")
```

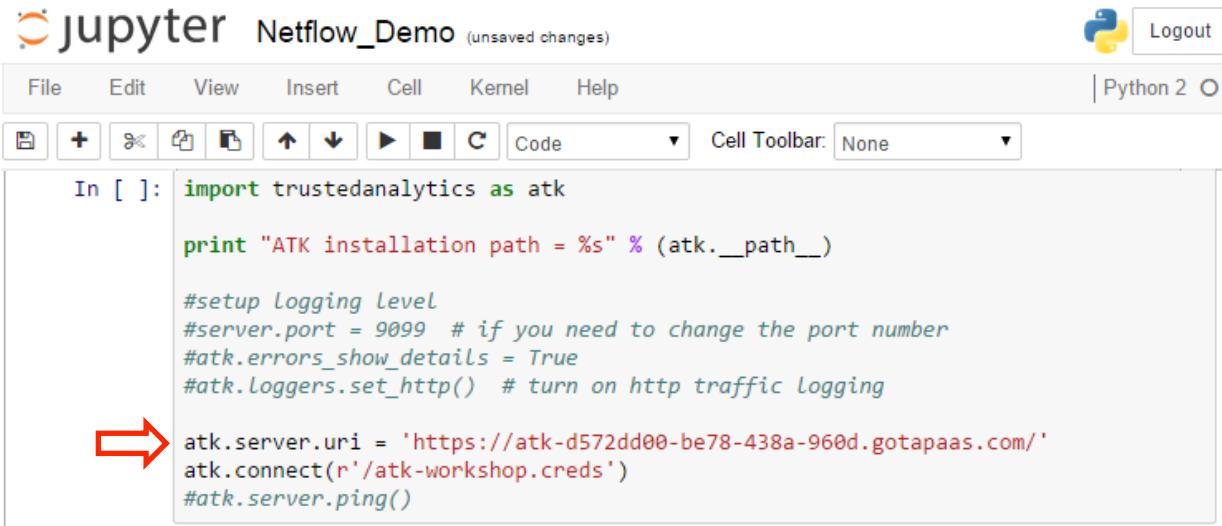
```
In [ ]: atk.CsvFile()
```

```
Init signature: atk.CsvFile(self, file_name, schema, delimiter=',', skip_header_lines=0)
Docstring:
Define a CSV file.
```

Help window

4.4. Edit the Netflow demo file:

- Change the ATK.server.uri field in the demo script to access the ATK instance you created in TAP.
- Copy the ATK.server.uri you created in TAP (Step 3.3).
- Paste the uri 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 uri and the “/” at the end, before you enter the uri information.
- You should already have created a credentials file, when you installed Python or iPython.



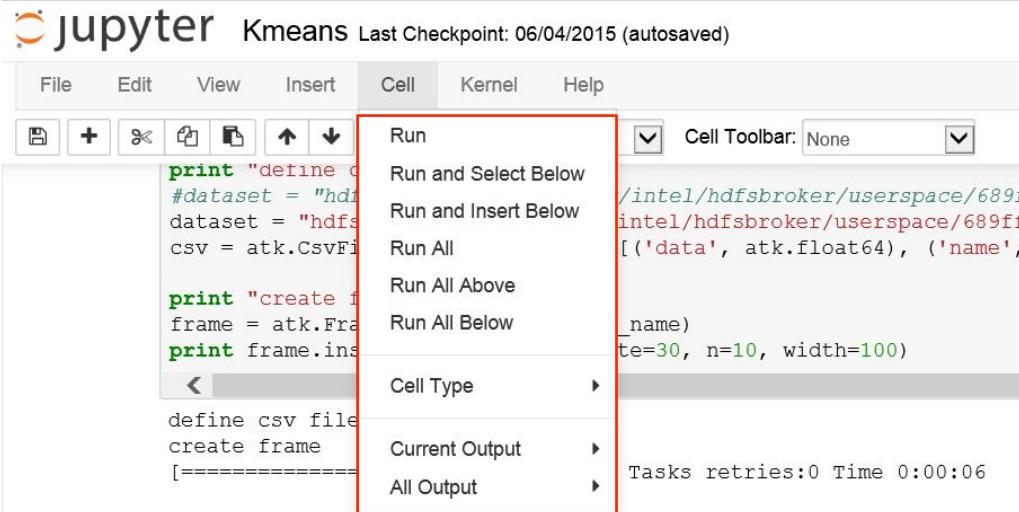
Jupyter Netflow_Demo (unsaved changes) Logout

File Edit View Insert Cell Kernel Help Python 2

```
In [ ]: import trustedanalytics as atk
         print "ATK installation path = %s" % (atk.__path__)
         #setup Logging Level
         #server.port = 9099 # if you need to change the port number
         #atk.errors_show_details = True
         #atk.Loggers.set_http() # turn on http traffic Logging
         atk.server.uri = 'https://atk-d572dd00-be78-438a-960d.gotapaas.com/'
         atk.connect(r'/atk-workshop.creds')
         #atk.server.ping()
```

5. Deploy the ATK Model

- Execute the Netflow demo, which deploys the ATK.
- 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.

A screenshot of a Jupyter Notebook interface titled "Netflow_Demo (unsaved changes)". The menu bar includes File, Edit, View, Insert, Cell (selected), Kernel, and Help. The toolbar includes standard icons. A Python logo icon and "Logout" button are in the top right. The code cell contains the following Python script:

```
#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('network_graph_titan_'+reference)
    drop('bytes_out_'+score)
    drop('bytes_in_'+score)
    drop('bytes_in_out_'+score)
    drop('ip_summary_frame_'+score)
    drop('svmModel1_ '+ reference)
```

5.2. Read in the data:

- This step reads in the data file that was loaded onto the ATK 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 interface with the title "Netflow_Demo (unsaved changes)". The menu bar includes File, Edit, View, Insert, Cell, Kernel, Help, and a Python 2 tab. The toolbar has icons for file operations like Open, Save, and Cell. The code cell contains Python code for reading a Netflow dataset from HDFS and inspecting it. The output cell shows two progress bars indicating task completion: one at 100.00% for tasks and another at 100.00% for time. The final output displays a table of Netflow data with columns: #, TimeSeconds, tstart, dateTimeStr, protocol, proto, sip, dip, sport, dport, flag, fwd, tdur, firstSeenSrcPayloadBytes, firstSeenDestPayloadBytes, ibyt, obyt, ipkt, opkt, recordForceOut. The data shows four entries for the same source IP 172.20.2.19.

```
dataset = "hdfs://nameservice1/org/intel/hdfsbroker/userspace/28ff2403-eb1d-4ace-b1d4-624c1d9f1f4f/6635acc0-71ff-40d5-9a2c-788dc3564349/000000_1"
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),
                 ('obyt', 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

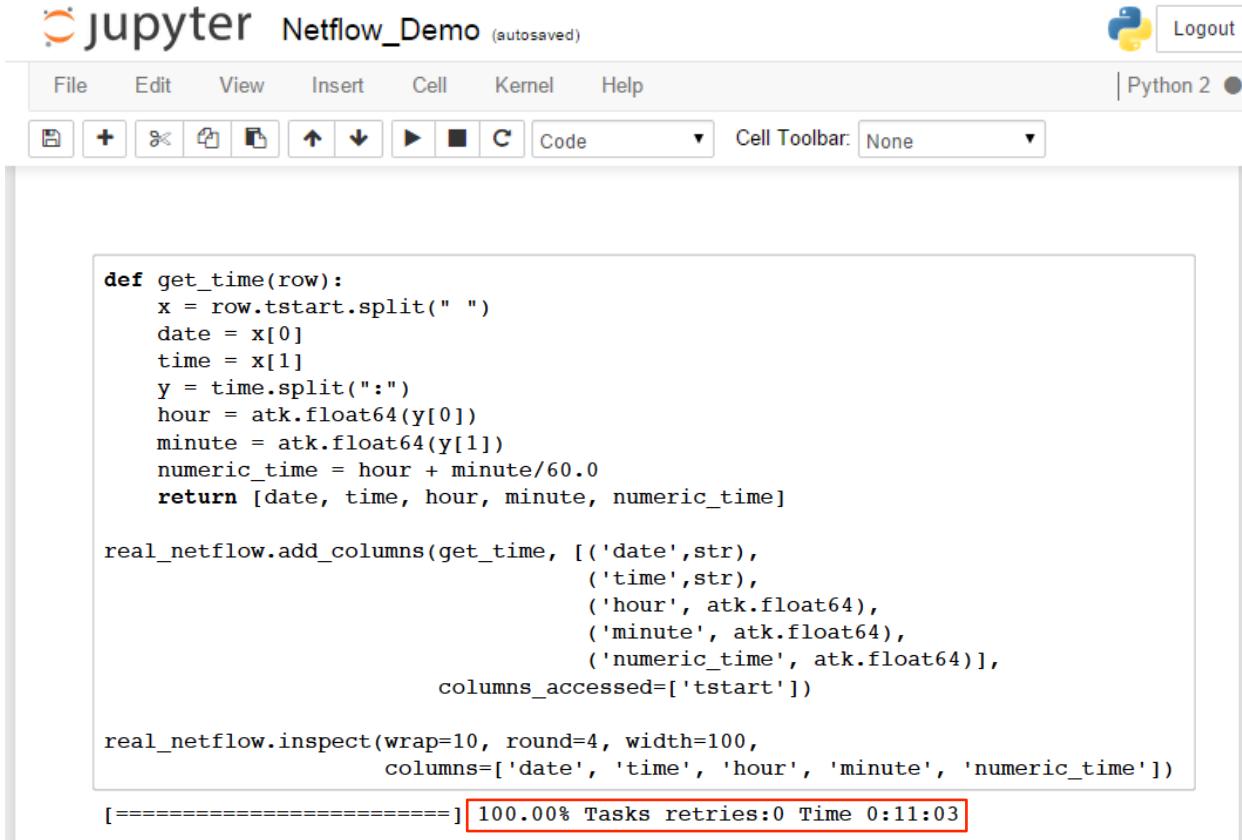
real_netflow.inspect(wrap=10, round=4, width=100)
```

[=====] 100.00% Tasks retries:0 Time 0:06:22
[=====] 100.00% Time 0:00:00

#	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	

5.3. Format and print the date/time data in columns:

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



The screenshot shows the Jupyter Notebook interface with a Python 2 kernel selected. A code cell contains the following script:

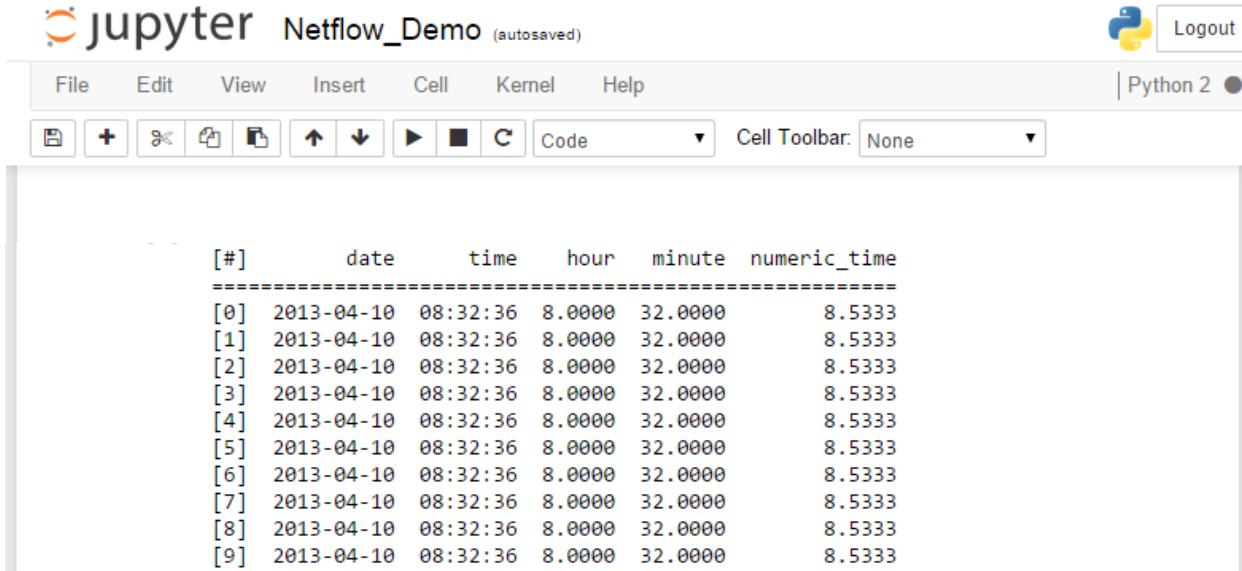
```
def get_time(row):
    x = row.tstart.split(" ")
    date = x[0]
    time = x[1]
    y = time.split(":")
    hour = atk.float64(y[0])
    minute = atk.float64(y[1])
    numeric_time = hour + minute/60.0
    return [date, time, hour, minute, numeric_time]

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'])
```

The output cell shows the progress: [=====] 100.00% Tasks retries:0 Time 0:11:03.

- The output from the date/time format script should look like this:



The screenshot shows the Jupyter Notebook interface with a Python 2 kernel selected. The output cell displays the following table:

#	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 Titan graph for computing statistics:

- This is the longest running script in the workshop. It may take four to five minutes to complete.
- Titan is a scalable graph database optimized for storing and querying graphs containing hundreds of billions of vertices and edges distributed across a multi-machine cluster.

The screenshot shows a Jupyter Notebook interface with the title "jupyter Netflow_Demo (autosaved)". The top menu bar includes File, Edit, View, Insert, Cell, Kernel, and Help. On the right, there's a Python 2 logo and a Logout button. Below the menu is a toolbar with various icons for file operations like Open, Save, and Print, along with navigation and cell execution buttons. A dropdown menu for "Cell Toolbar" is set to "None".

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

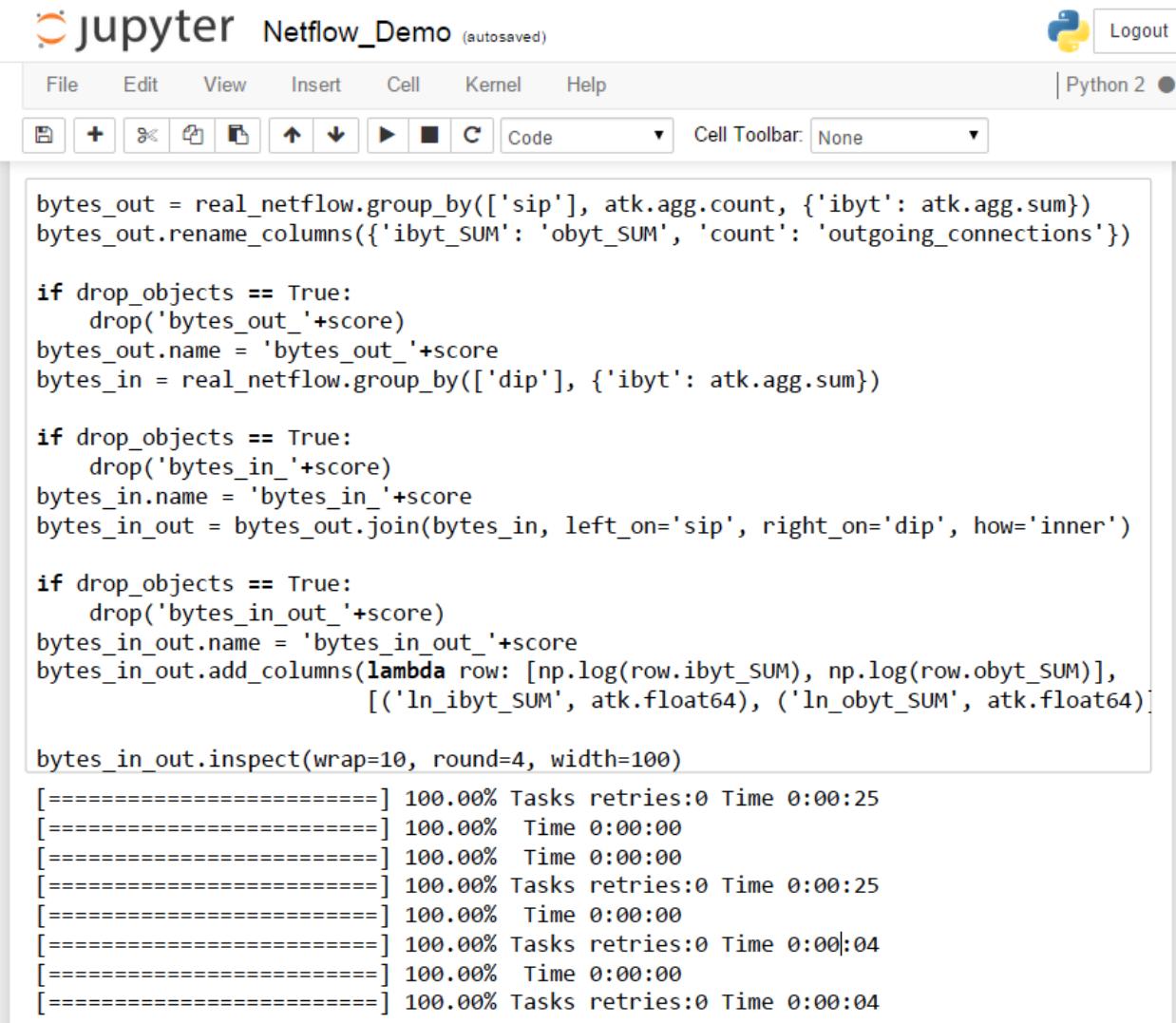
network_graph = atk.Graph()
if drop_objects == True:
    drop('network_graph_'+reference)
network_graph.name = 'network_graph_'+reference
network_graph.define_vertex_type('IP')
network_graph.define_edge_type('Connection', 'IP', 'IP', directed=False)
network_graph.edges['Connection'].add_edges(network_edges,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

network_graph_titan = network_graph.export_to_titan()
if drop_objects == True:
    drop('network_graph_titan_'+reference)
network_graph_titan.name = 'network_graph_titan_'+reference

[=====] 100.00% Tasks retries:0 Time 0:01:32
[=====] 100.00% Time 0:00:00
[=====] 100.00% Time 0:00:00
[=====] 100.00% Time 0:00:00
[=====] 100.00% Time 0:00:00
[=====] 100.00% Tasks retries:0 Time 0:00:28
[=====] 100.00% Tasks retries:0 Time 0:00:04
[=====] 100.00% Tasks retries:0 Time 0:02:04
[=====] 100.00% Time 0:00:00
```

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.



The screenshot shows a Jupyter Notebook interface titled "Netflow_Demo (autosaved)". The top bar includes a Python 2 kernel icon and a "Logout" button. The toolbar below has icons for file operations, cell execution, and cell toolbar settings. The main code cell contains the following Python code:

```
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)
```

The output of the code cell shows the results of the inspect() function:

```
[=====] 100.00% Tasks retries:0 Time 0:00:25
[=====] 100.00% Time 0:00:00
[=====] 100.00% Time 0:00:00
[=====] 100.00% Tasks retries:0 Time 0:00:25
[=====] 100.00% Time 0:00:00
[=====] 100.00% Tasks retries:0 Time 0:00:04
[=====] 100.00% Time 0:00:00
[=====] 100.00% Tasks retries:0 Time 0:00:04
```

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 interface titled "jupyter Netflow_Demo (autosaved)". The top bar includes a Python logo icon, "Logout", and a "Python 2" kernel selection. The menu bar has options: File, Edit, View, Insert, Cell, Kernel, Help. Below the menu is a toolbar with icons for file operations (New, Open, Save, etc.) and cell execution (Run, Stop, Cell). A dropdown "Cell Toolbar" is set to "None".

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

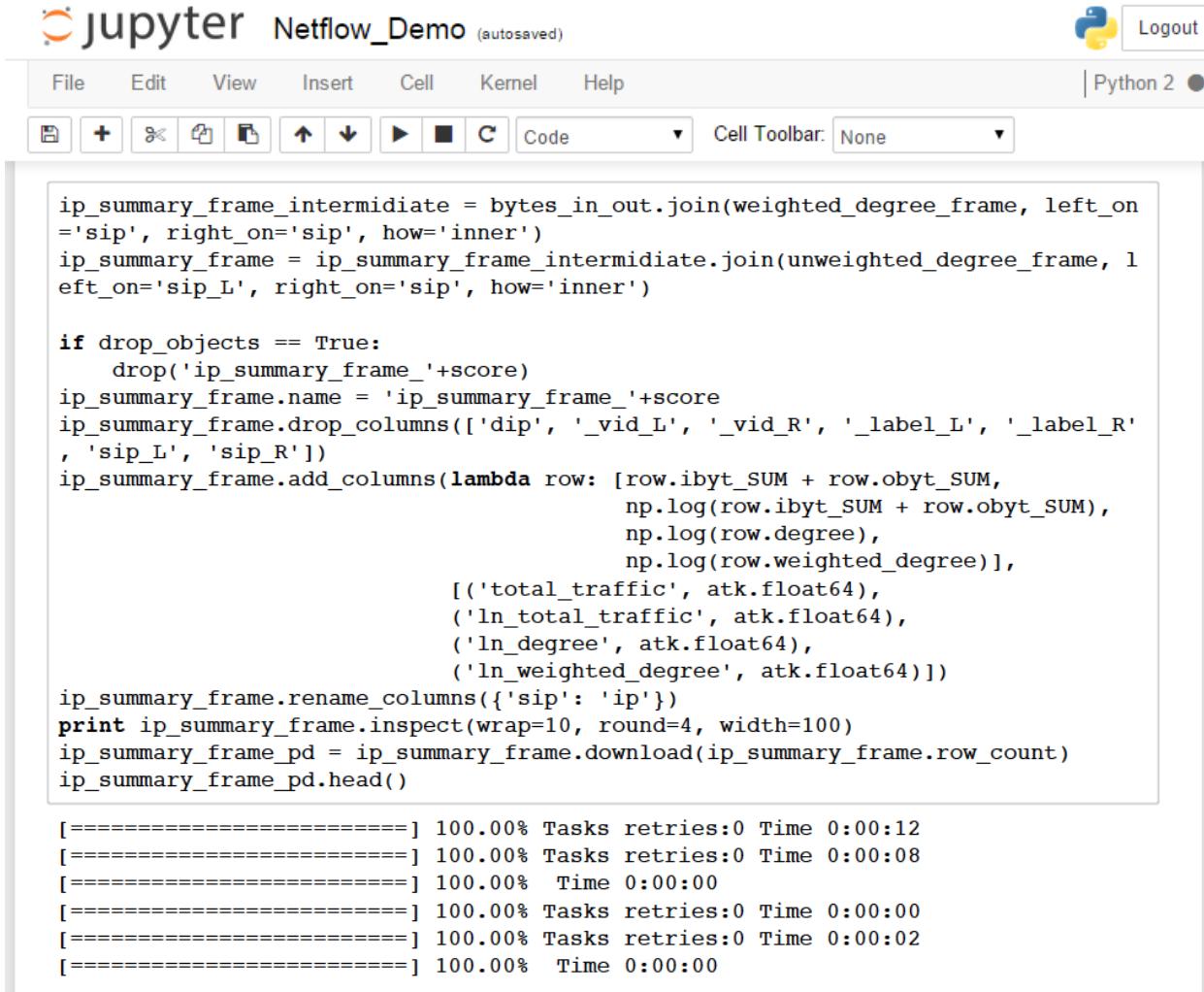
print 'Compute weighted degree count:'
weighted_degree_frame = network_graph_titan.annotate_weighted_degrees('weighted_degree', 'undirected',
edge_weight_property = 'count')['IP']

Compute degree count:
[=====] 100.00% Tasks retries:0 Time 0:00:09
Compute weighted degree count:
[=====] 100.00% Tasks retries:0 Time 0:00:09
```

The code cell contains two print statements followed by variable assignments. The first print statement is highlighted with a red box. The output below the code cell shows the printed messages followed by progress bars indicating task completion at 100.00% with a time of 0:00:09 and zero retries for both tasks.

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.



The screenshot shows a Jupyter Notebook interface with the title "jupyter Netflow_Demo (autosaved)". The menu bar includes File, Edit, View, Insert, Cell, Kernel, Help, and a Python 2 kernel selection. The toolbar below the menu has icons for file operations like Open, Save, and Print, along with navigation and cell execution buttons. The main code cell contains Python code for summarizing network traffic data:

```

ip_summary_frame_intermediate = bytes_in_out.join(weighted_degree_frame, left_on='sip', right_on='sip', how='inner')
ip_summary_frame = ip_summary_frame_intermediate.join(unweighted_degree_frame, left_on='sip_L', right_on='sip', how='inner')

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()

```

Below the code cell, the output shows the progress of the download task:

```

[=====] 100.00% Tasks retries:0 Time 0:00:12
[=====] 100.00% Tasks retries:0 Time 0:00:08
[=====] 100.00% Time 0:00:00
[=====] 100.00% Tasks retries:0 Time 0:00:00
[=====] 100.00% Tasks retries:0 Time 0:00:02
[=====] 100.00% Time 0:00:00

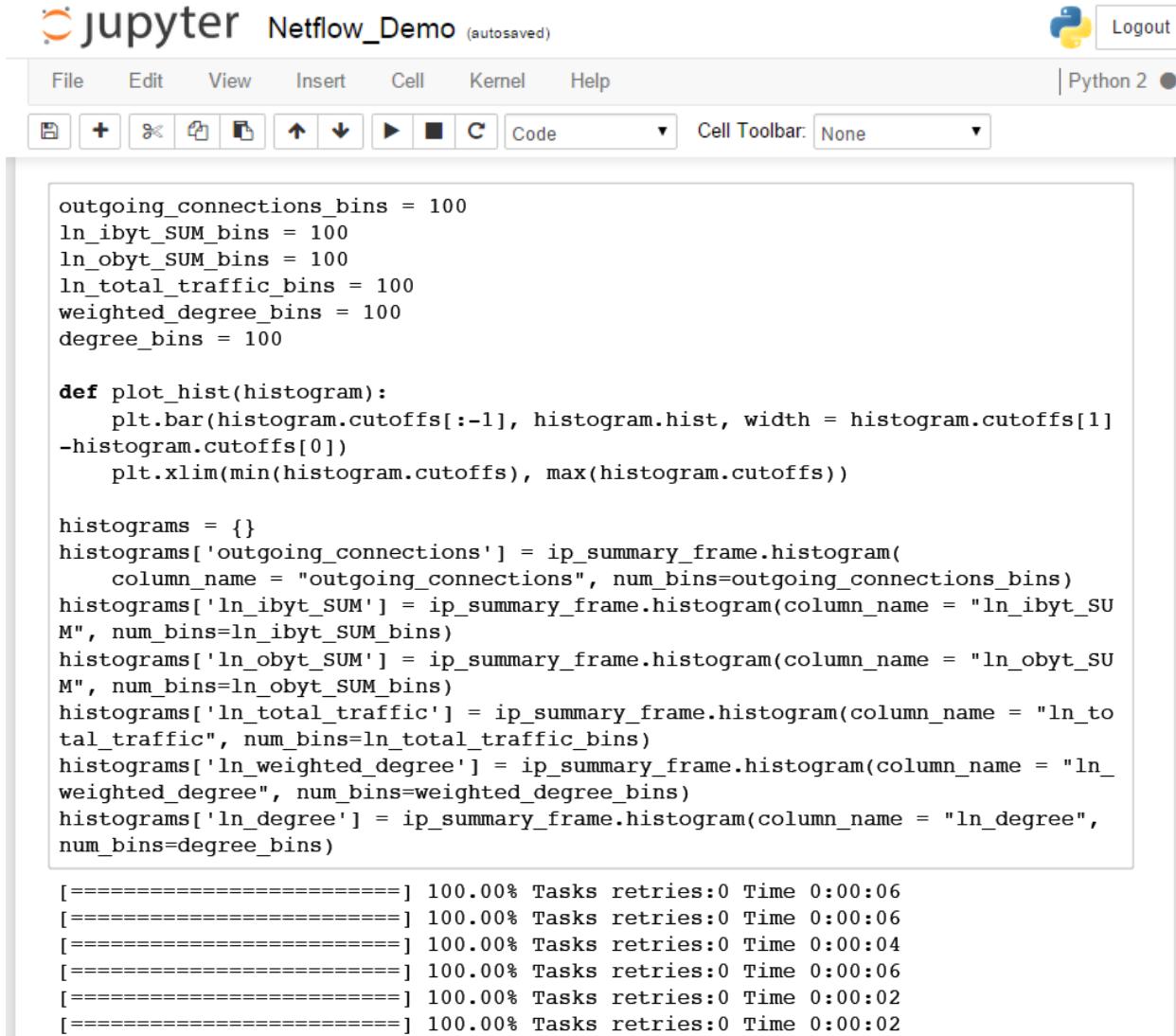
```

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

	outgoing_connections	obyt_SUM	ibyt_SUM	In_ibyt_SUM	In_obyt_SUM	weighted_
0	1176	17332354	22267710	16.918648	16.668085	1206
1	842	4819025	8652932	15.973409	15.388082	860
2	1105	9816363	23508230	16.972861	16.099561	1138
3	712	1578007	4087014	15.223325	14.271673	734
4	722	2469206	11834152	16.286500	14.719407	745

5.8. Compute and plot histogram:

- 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.



The screenshot shows a Jupyter Notebook interface titled "Netflow_Demo (autosaved)". The top navigation bar includes File, Edit, View, Insert, Cell, Kernel, Help, and a Python 2 kernel selector. Below the toolbar is a "Cell Toolbar" dropdown set to "None". The main code cell contains Python code for defining histogram bins and plotting functions, followed by a series of histogram plots.

```
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

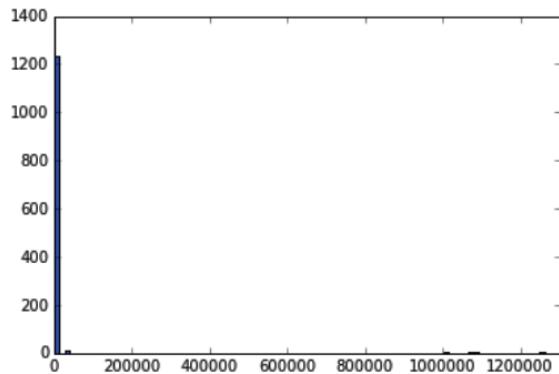
def plot_hist(histogram):
    plt.bar(histogram.cutoffs[:-1], histogram.hist, width = histogram.cutoffs[1]
-histogram.cutoffs[0])
    plt.xlim(min(histogram.cutoffs), max(histogram.cutoffs))

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_SU
M", num_bins=ln_ibyt_SUM_bins)
histograms['ln_obyt_SUM'] = ip_summary_frame.histogram(column_name = "ln_obyt_SU
M", num_bins=ln_obyt_SUM_bins)
histograms['ln_total_traffic'] = ip_summary_frame.histogram(column_name = "ln_to
tal_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)

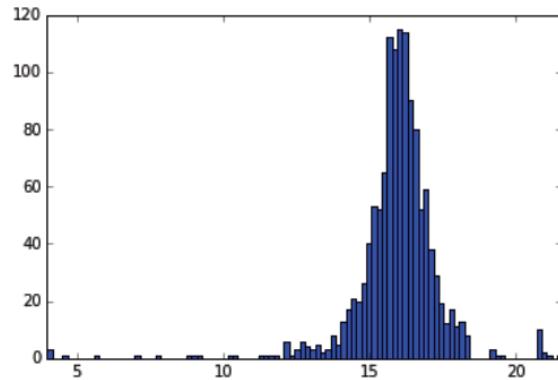
[=====] 100.00% Tasks retries:0 Time 0:00:06
[=====] 100.00% Tasks retries:0 Time 0:00:06
[=====] 100.00% Tasks retries:0 Time 0:00:04
[=====] 100.00% Tasks retries:0 Time 0:00:06
[=====] 100.00% Tasks retries:0 Time 0:00:02
[=====] 100.00% Tasks retries:0 Time 0:00:02
```

- Six histograms:

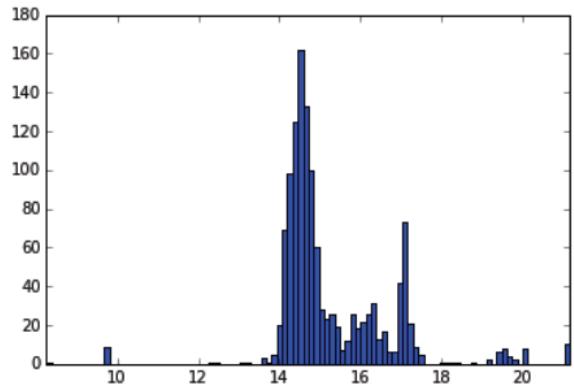
```
plot_hist(histograms['outgoing_connections'])
```



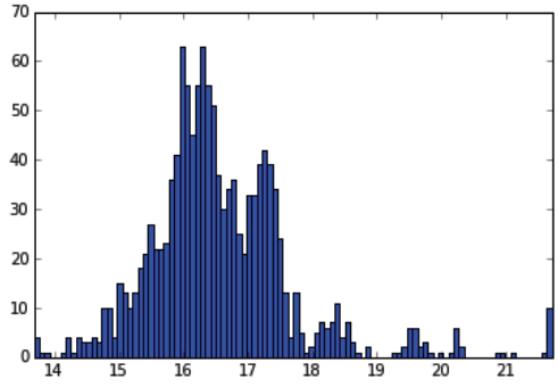
```
plot_hist(histograms['ln_ibyt_SUM'])
```



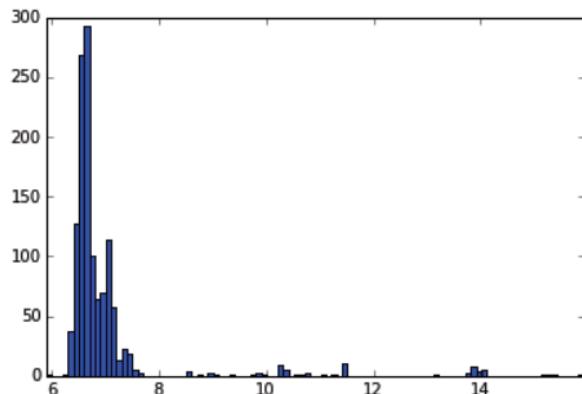
```
plot_hist(histograms['ln_obyt_SUM'])
```



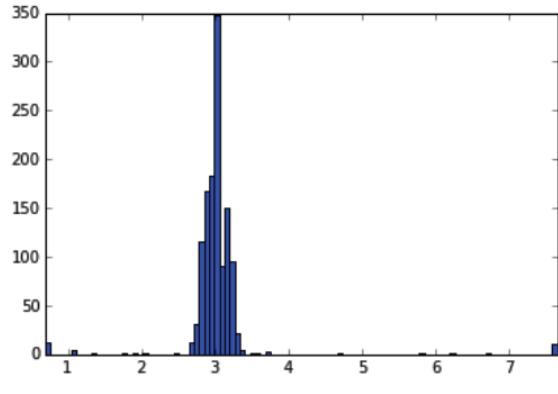
```
plot_hist(histograms['ln_total_traffic'])
```



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

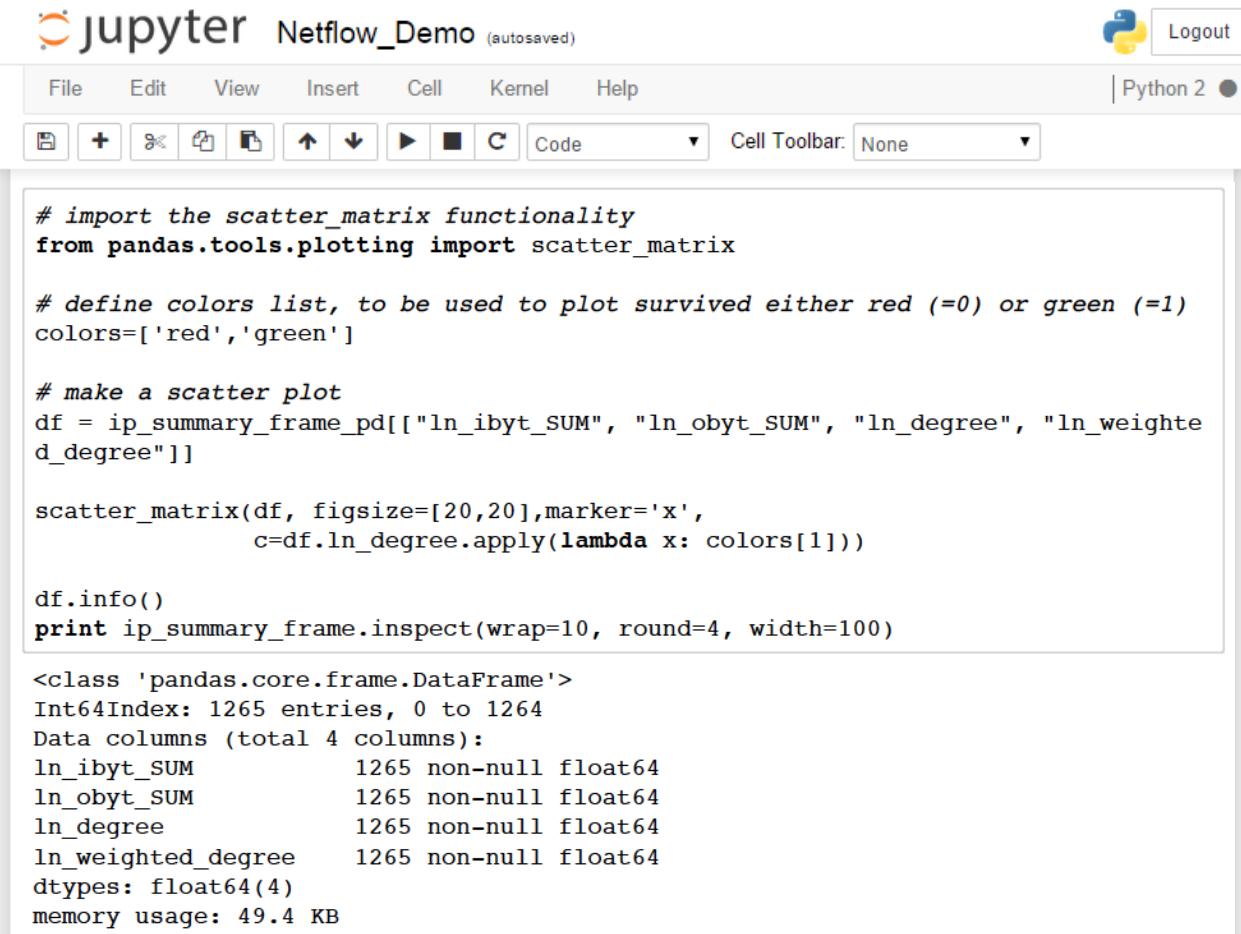


```
plot_hist(histograms['ln_degree'])
```



5.9. Create scatter plots:

- 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 the title "Netflow_Demo" and status "(autosaved)". The toolbar includes File, Edit, View, Insert, Cell, Kernel, Help, and a Python 2 kernel icon. The Cell Toolbar dropdown is set to "None".

```
# 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']

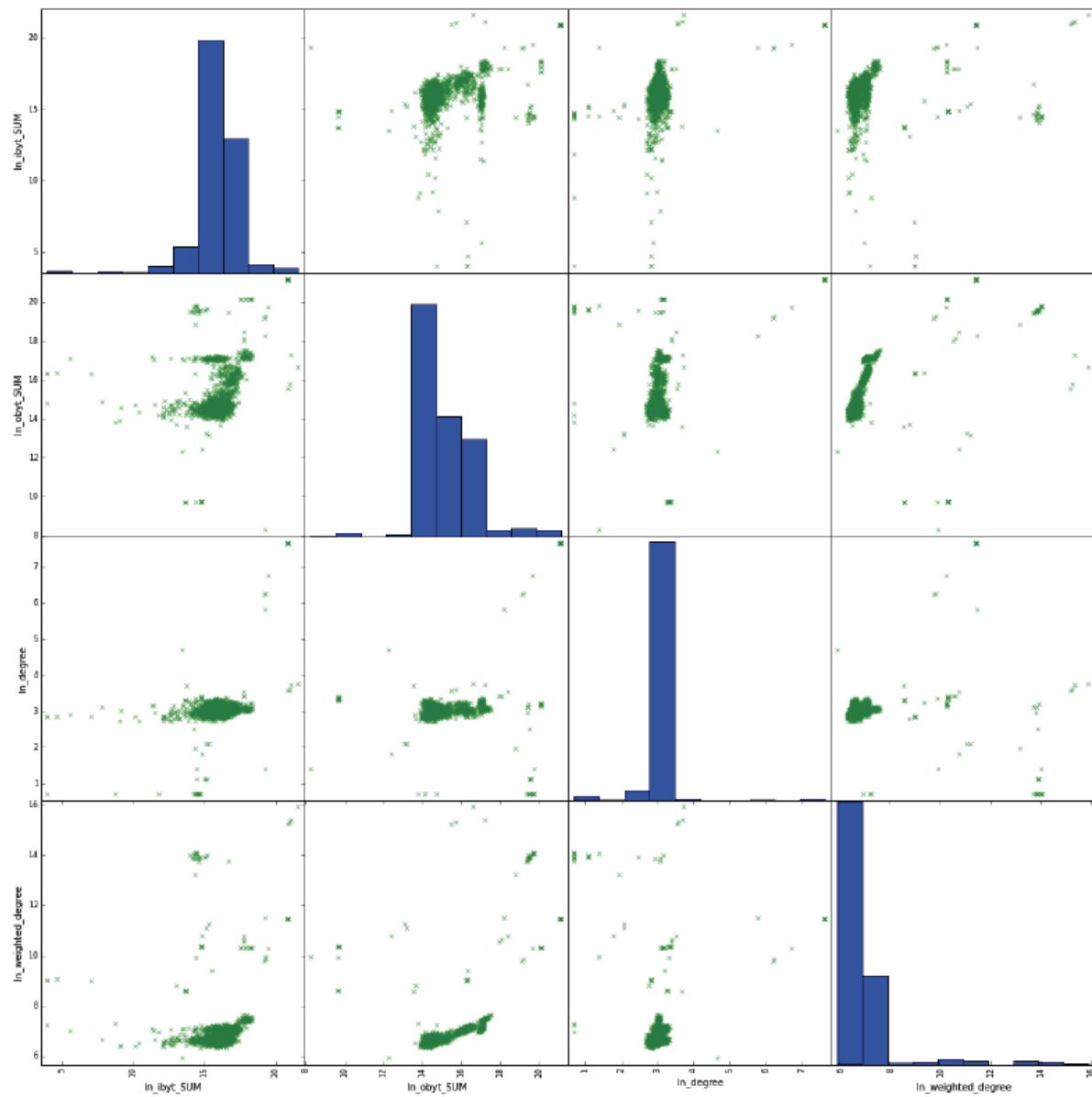
# make a scatter plot
df = ip_summary_frame_pd[['ln_ibyt_SUM', 'ln_obyt_SUM', 'ln_degree', 'ln_weighted_degree']]

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)

<class 'pandas.core.frame.DataFrame'>
Int64Index: 1265 entries, 0 to 1264
Data columns (total 4 columns):
ln_ibyt_SUM      1265 non-null float64
ln_obyt_SUM      1265 non-null float64
ln_degree        1265 non-null float64
ln_weighted_degree 1265 non-null float64
dtypes: float64(4)
memory usage: 49.4 KB
```

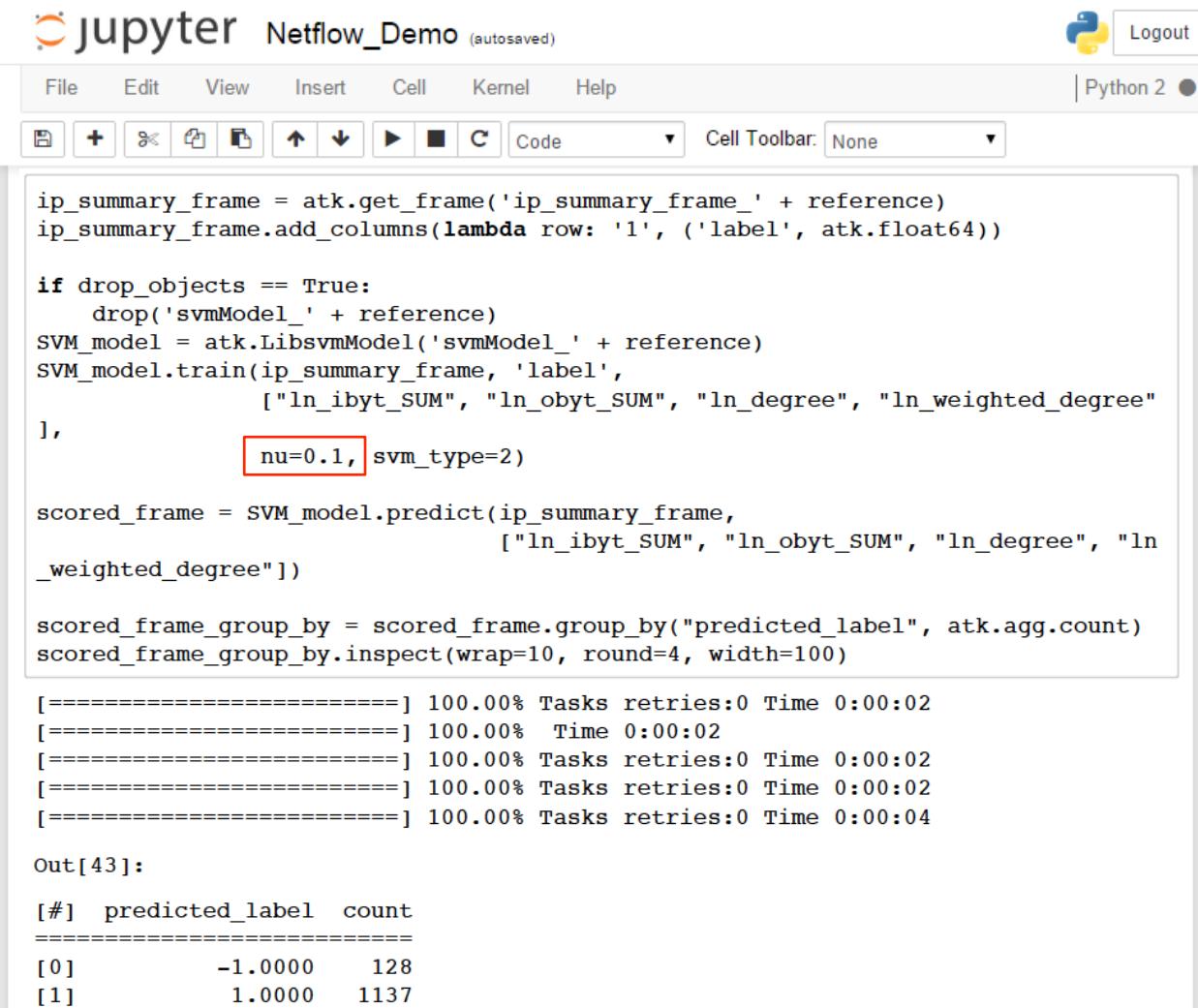
- Scatter-plot matrix:



- 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.



The screenshot shows a Jupyter Notebook interface titled "Netflow_Demo (autosaved)". The menu bar includes File, Edit, View, Insert, Cell, Kernel, Help, and a Python 2 kernel selector. The toolbar below the menu has icons for file operations like Open, Save, and Cell execution. A dropdown menu "Cell Toolbar" is set to "None". The code cell contains Python code for training an SVM model. The line `nu=0.1, svm_type=2` is highlighted with a red box. The output cell (Out[43]) shows the command run and its results, including a table of predicted labels and counts.

```
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"])

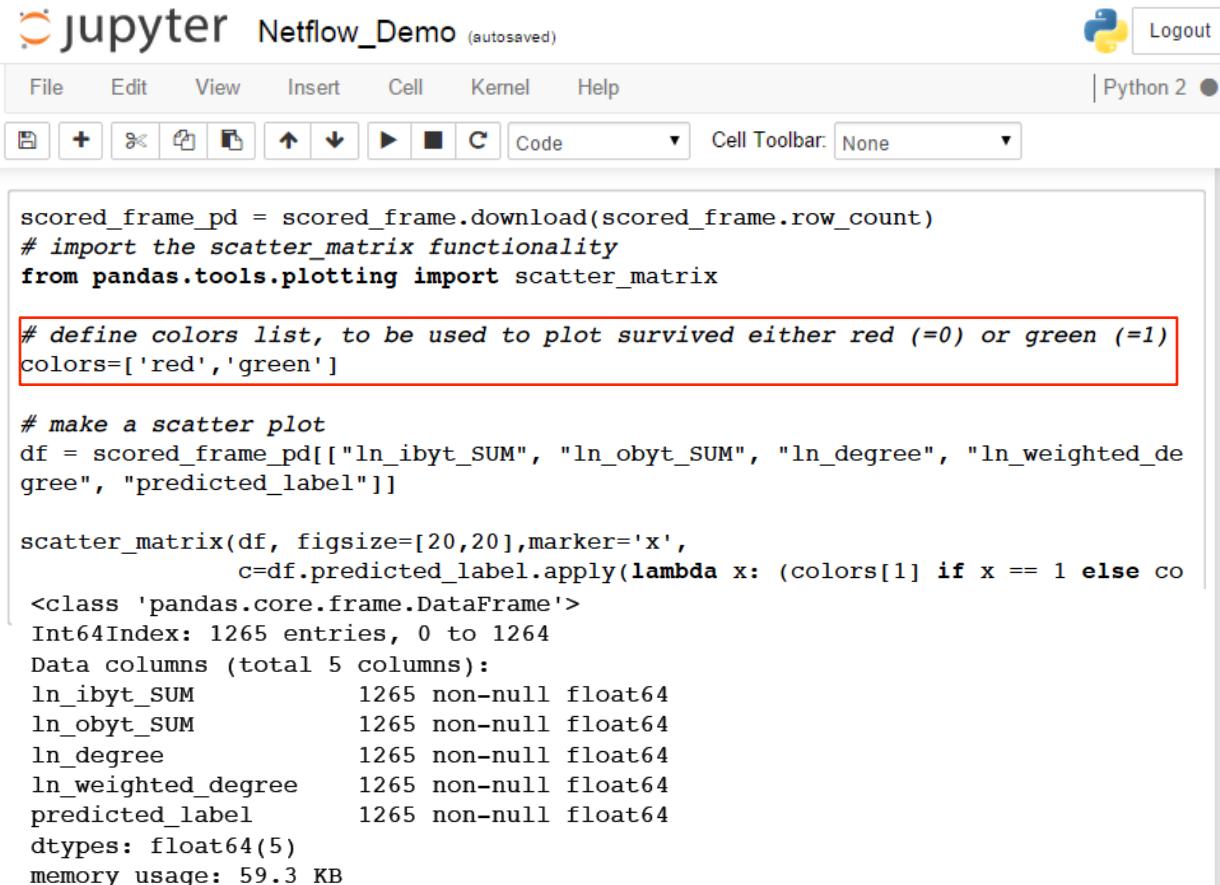
scored_frame_group_by = scored_frame.groupby("predicted_label", atk.agg.count)
scored_frame_group_by.inspect(wrap=10, round=4, width=100)
```

```
[=====] 100.00% Tasks retries:0 Time 0:00:02
[=====] 100.00% Time 0:00:02
[=====] 100.00% Tasks retries:0 Time 0:00:02
[=====] 100.00% Tasks retries:0 Time 0:00:02
[=====] 100.00% Tasks retries:0 Time 0:00:04

Out[43]:
[#] predicted_label count
=====
[0]      -1.0000    128
[1]       1.0000   1137
```

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.



The screenshot shows a Jupyter Notebook interface titled "Netflow_Demo (autosaved)". The top navigation bar includes File, Edit, View, Insert, Cell, Kernel, Help, and a Python 2 kernel selector. Below the toolbar is a cell editor with various icons for file operations. The main code cell contains Python code for generating a scatter matrix plot. A specific line of code, `# define colors list, to be used to plot survived either red (=0) or green (=1)`, is highlighted with a red border. The code also includes a color definition line: `colors=['red','green']`. The output of the code is displayed below the code cell, showing the creation of a scatter matrix with columns for `ln_ibyt_SUM`, `ln_obyt_SUM`, `ln_degree`, `ln_weighted_degree`, and `predicted_label`.

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

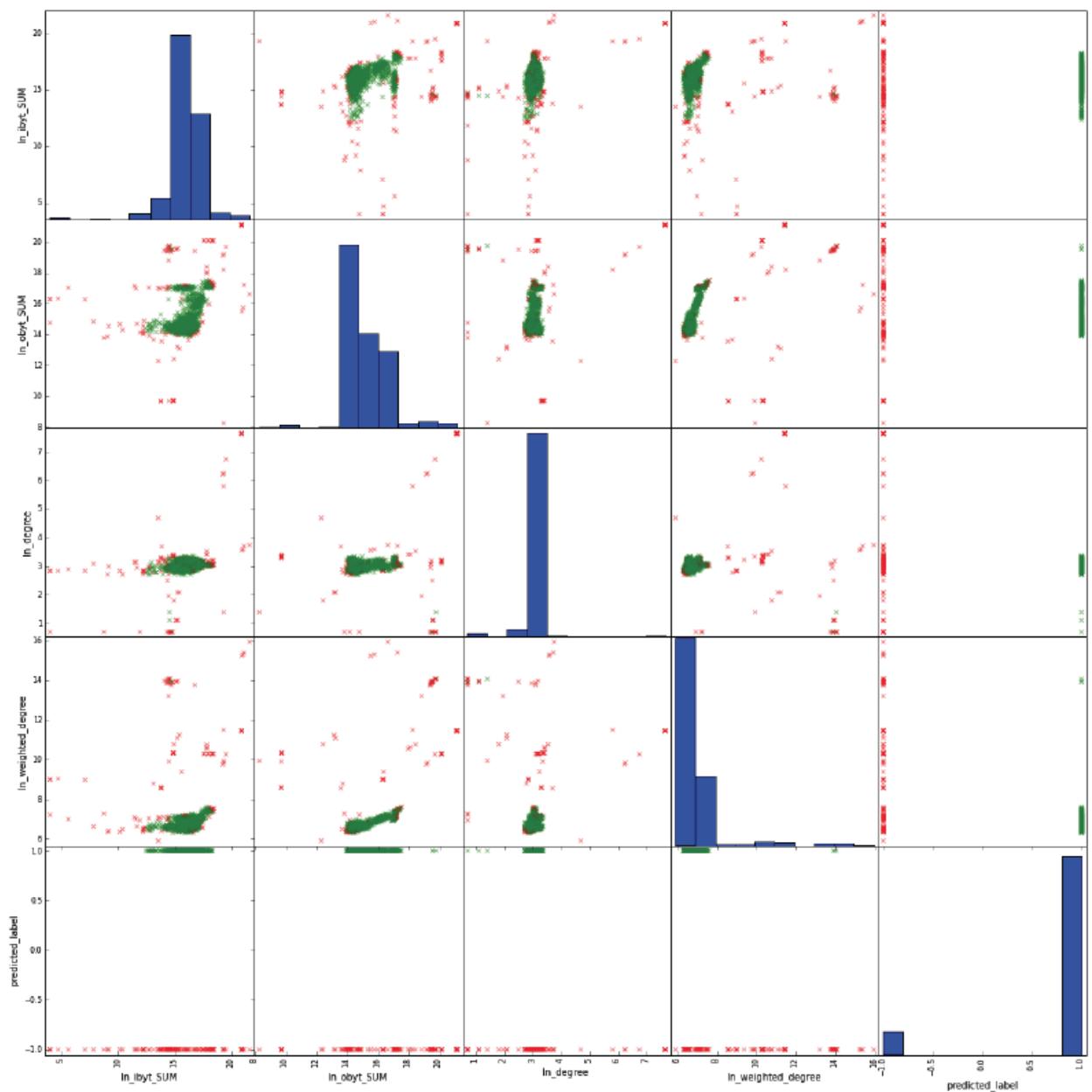
# define colors list, to be used to plot survived either red (=0) or green (=1)
colors=['red','green']

# make a scatter plot
df = scored_frame_pd[["ln_ibyt_SUM", "ln_obyt_SUM", "ln_degree", "ln_weighted_degree", "predicted_label"]]

scatter_matrix(df, figsize=[20,20],marker='x',
               c=df.predicted_label.apply(lambda x: (colors[1] if x == 1 else co
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1265 entries, 0 to 1264
Data columns (total 5 columns):
ln_ibyt_SUM           1265 non-null float64
ln_obyt_SUM           1265 non-null float64
ln_degree             1265 non-null float64
ln_weighted_degree   1265 non-null float64
predicted_label       1265 non-null float64
dtypes: float64(5)
memory usage: 59.3 KB
```

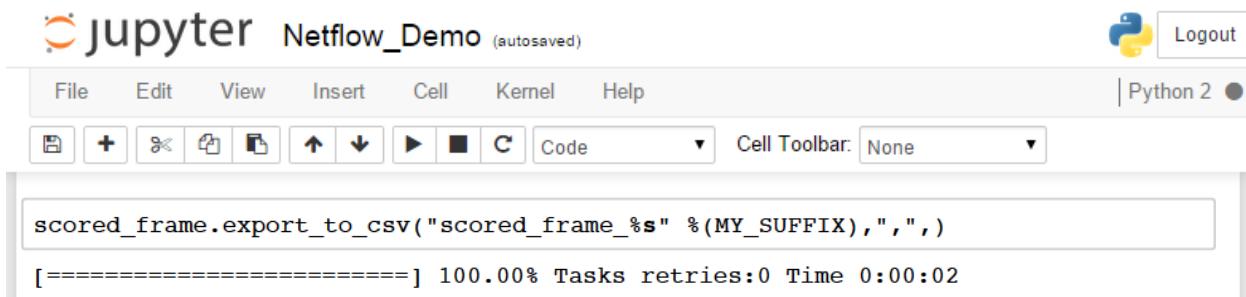
- See the scatter-plot graphs on the next page.

- 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 interface titled "jupyter Netflow_Demo (autosaved)". The top menu bar includes File, Edit, View, Insert, Cell, Kernel, and Help. On the right, there is a Python 2 kernel icon and a Logout button. Below the menu is a toolbar with various icons for file operations like opening, saving, and cell execution. A dropdown menu for "Cell Toolbar" is set to "None". The main code cell contains the following Python code:

```
scored_frame.export_to_csv("scored_frame_%s" %(MY_SUFFIX),",",)
```

Below the code cell, a progress bar indicates "100.00% Tasks retries:0 Time 0:00:02".

CONGRATULATIONS!
You have successfully completed the workshop.