Lab: Introduction Data Science Experience

July 19, 2018

Author: Elena Lowery elowery@us.ibm.com

Table of contents

Contents

Overview

In this lab you will learn the basics of working in **IBM Data Science Experience** (DSX)

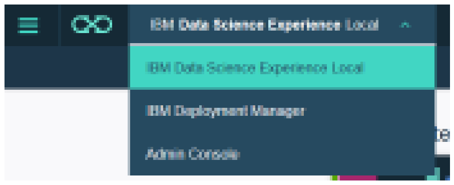
Required software, access, and files

- To complete this lab, you will need access to a DSX Local cluster.
- You will also need to download and unzip this GitHub repository: https://github.com/elenalowery/DSX_Local_Workshop_12
- After unzipping the downloaded file, navigate to the *Projects* folder, and rename the zip file to a unique name, for example, *DSX_Demos_el*. Do not unzip this file.

DSX Views

DSX has three views for three separate activities that can be performed in DSX:

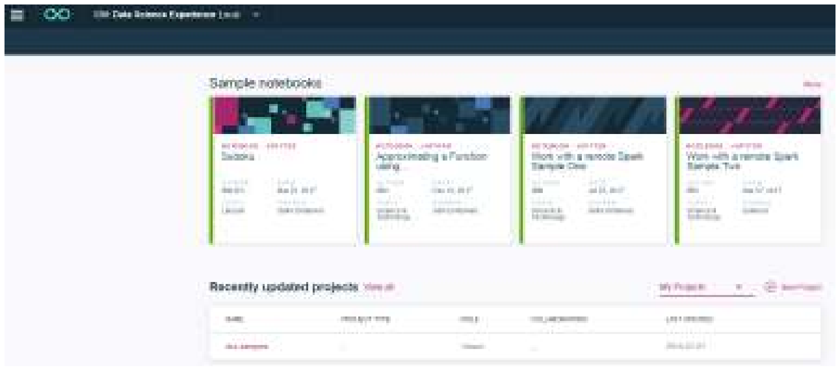
- **Data Science Experience** view is the main development environment for data scientists.
- **Deployment Manager** view is used for deployment.
- **Admin Console** is used by an IT admin to monitor and manage DSX.


IBM Data Science Experience Local
IBM Data Science Experience Local
IBM Deployment Manager
Admin Console

In this lab we will focus on the development environment.

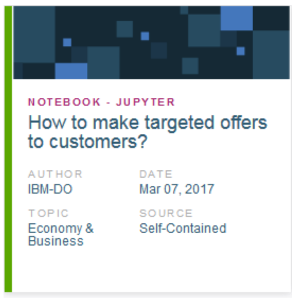
Projects

1. Log in to DSX Local.
2. The default view is the development environment. First, let's review the samples that are included with DSX.


IBM Data Science Experience Local
Sample notebooks
Sudoku
Approximating a Function using...
Work with a remote Spark Sample One
Work with a remote Spark Sample Two
Recently updated projects
View all

3. If you click the **More** option for the sample notebooks, you can view all the sample *Jupyter* notebooks that are packaged with DSX. Each notebook can be opened or downloaded.

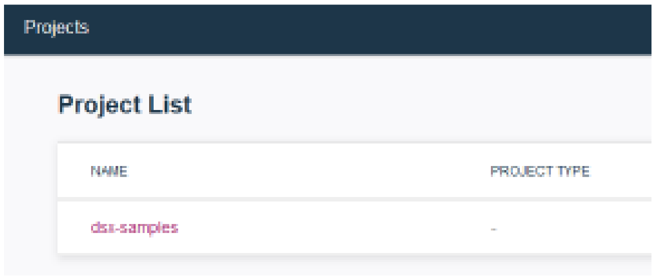
4. Open one of the notebooks, for example, *How to make targeted offers to customers*. This notebook is an example of using **Decision Optimization APIs** to solve a marketing use case.



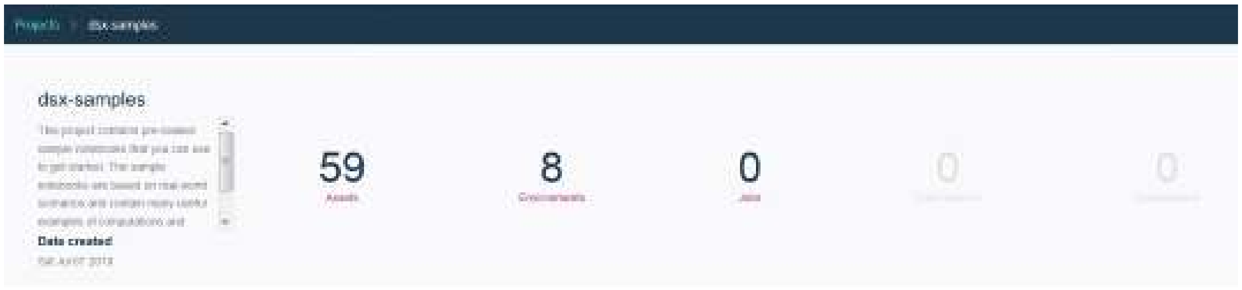
5. If you wish, you can clear the output in the notebook, review it, and run it.

   Because this notebook is a self-contained example, it generates input data and prints output to the notebook. In an actual implementation of the use case we would read and write data to different types of data sources.

6. Next, navigate to the **Projects** view and click on *dsx_samples* project.



NAME	PROJECT TYPE
dsx-samples	-

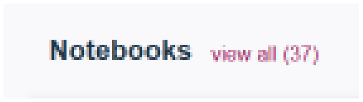


In DSX projects are used to organize all asset types: notebooks, RStudio applications, SPSS Modeler flows, scripts, and flat files.

Some of the options for the *dsx-sample* project are disabled because it's a special project type.

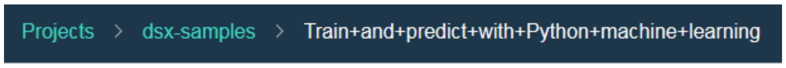
7. Click on **Assets**. This view shows all analytic assets that are included in the samples. Some of them are the *Jupyter* notebooks that are displayed in the other view, but there are also other assets, such as models and datasets.

8. Click **view all** next to **Notebooks**.



9. Open, review and run another notebook, for example, *Train+and+predict+with+Python+machine+learning*.

   This notebook shows a typical process for building a model and saving it in DSX repository.

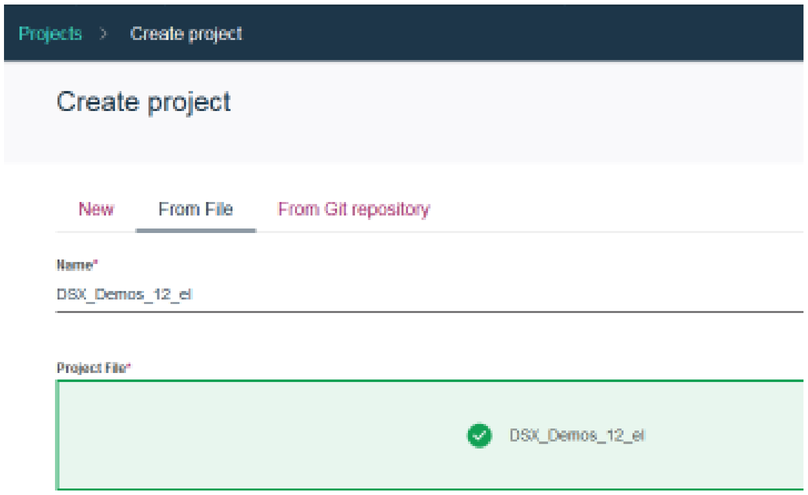


10. Next, we will create a new project. In the *Projects* view click **New project**.

    In our example we will be creating a project from file because we want to review existing assets, but you can also create a blank project or create a project that's connected to a Git repository.

11. Click on the **From File** tab and navigate to the *Projects* folder of the unzipped git repo. Select the zip file you renamed earlier in the lab. In our example it's *DSX_Demos_el.*

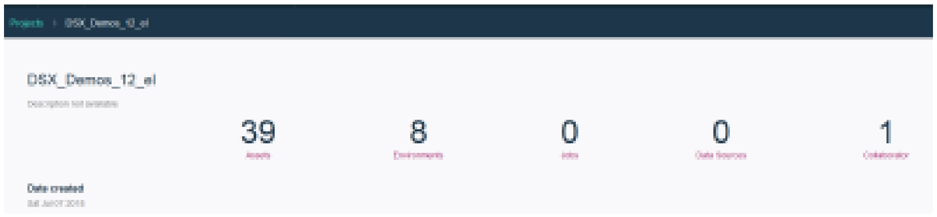
    Note: We have to rename the file because in DSX project name is inherited from the file, and all project names in the cluster must be unique.



Name*
DSX_Demos_12_el

Project File*
DSX_Demos_12_el

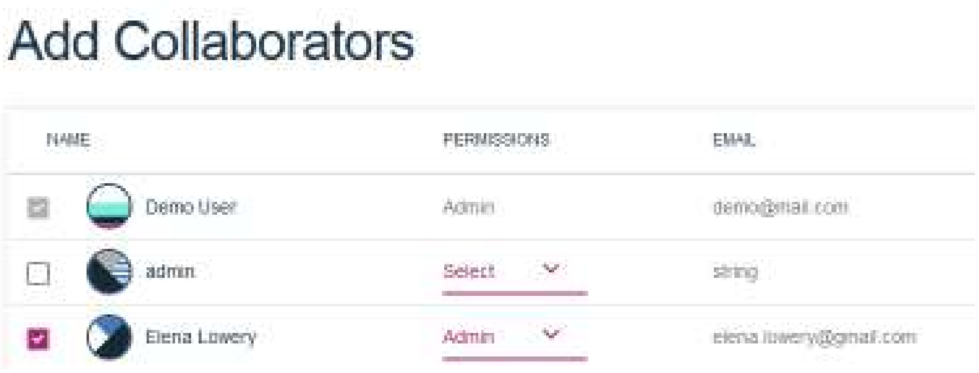
12. Now all navigation options for the project are enabled.



13. Click on **Collaborator**. Here you can add collaborators to the project and give them a specific role. Collaborators are registered users in DSX.

   - A **Viewer** can only view assets in the project.
   - An **Editor** can add, modify, and remove assets.
   - An **Admin** has the same permissions as **Editor** and can also delete the project.

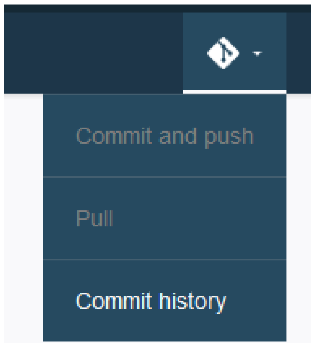
   If you wish, add a collaborator.



Add Collaborators

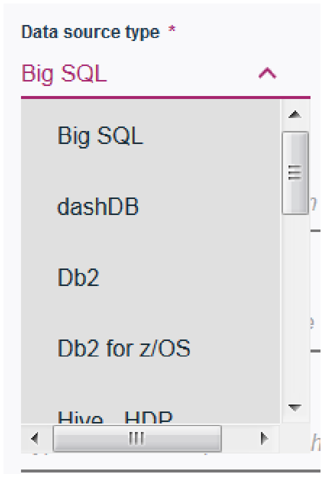
	NAME	PERMISSIONS	EMAIL
✓	Demo User	Admin	demo@mail.com
☐	admin	Select	string
✓	Elena Lowery	Admin	elena.lowery@gmail.com

14. When a collaborator works on a project, they work on their own copy. To commit the changes, a user will select the **Git actions** button in the menu bar. The actions will be disabled until changes to the project are made.

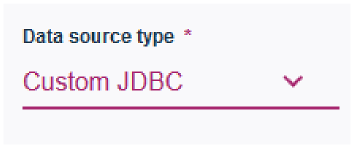


15. In the **Project** view, click on **Data Sources**, then **add data source**. Click on the **Data source type** dropdown.

DSX includes jdbc drivers for every database data source shown in the dropdown. DSX also provides built-in connectivity for the listed Hadoop data sources.



If the data source that you need to connect to is not listed, then **Custom JDBC** option can be used.



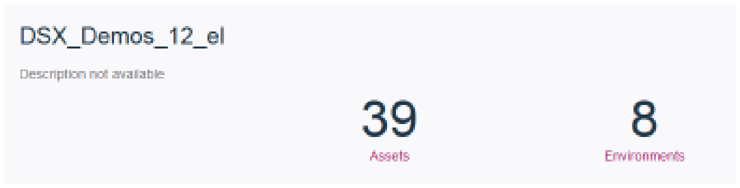
While it's possible to connect to data directly from IDE using Python, R or Scala code, defining a data source has several benefits:

- **Better organization**: we can see which data sources are used in a project.
- **Security**: credentials that are used to access a data source are encrypted. Credentials are not shared between collaborators.
- DSX provides **code generation** for datasets that are associated with data sources.

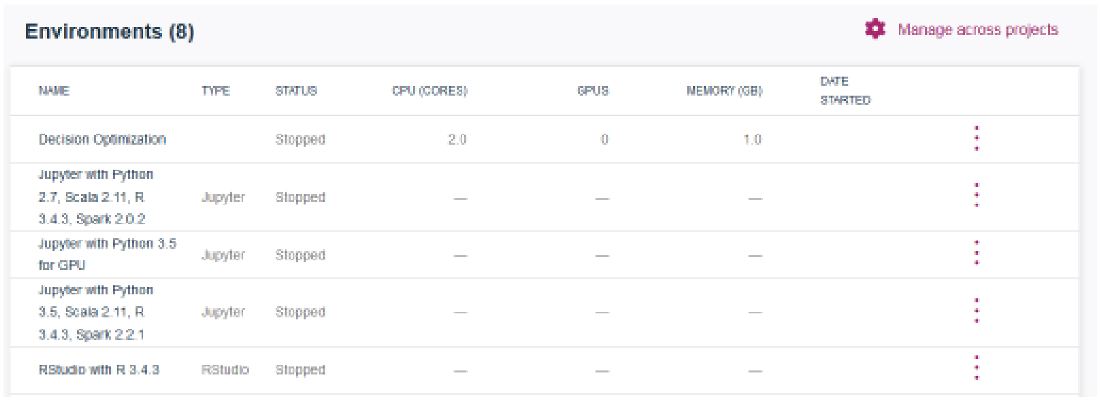
If you would like to create a data source and review code generation, see **Appendix A**.

IBM compatibility reports list data sources that are officially supported by DSX (see **Reference**). Since the majority of IDEs in DSX are open source IDEs, there is no limitation for which data source can be used. The *supported* data sources are the ones that have been tested by IBM and will be supported by IBM tech support.

16. In the **Project** view, click on **Environments**.



Environments are the different runtimes that we use when working in DSX. Each project has the same environments.



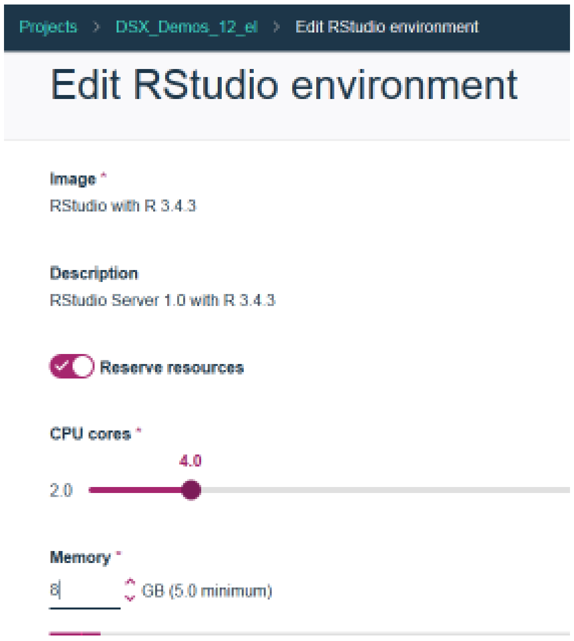
Environments (8)

Manage across projects

NAME	TYPE	STATUS	CPU (CORES)	GPUS	MEMORY (GB)	DATE STARTED
Decision Optimization		Stopped	2.0	0	1.0	
Jupyter with Python 2.7, Scala 2.11, R 3.4.3, Spark 2.0.2	Jupyter	Stopped	—	—	—	
Jupyter with Python 3.5 for GPU	Jupyter	Stopped	—	—	—	
Jupyter with Python 3.5, Scala 2.11, R 3.4.3, Spark 2.2.1	Jupyter	Stopped	—	—	—	
RStudio with R 3.4.3	RStudio	Stopped	—	—	—	

Click on the ellipses next to *RStudio* environment and select **Edit Settings**.

DSX supports "reservations" of hardware resources. A "reservation" means that the specified CPU and memory will be allocated for the specific runtime in the specific project, for example *RStudio* in the *DSX_Demos* project. The reservation will allocate the resources whether or not they are being used.



Projects > DSX_Demos_12_el > Edit RStudio environment

Edit RStudio environment

Image *
RStudio with R 3.4.3

Description
RStudio Server 1.0 with R 3.4.3

Reserve resources

CPU cores *
2.0 — 4.0

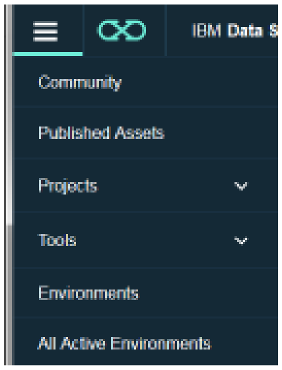
Memory *
8 GB (5.0 minimum)

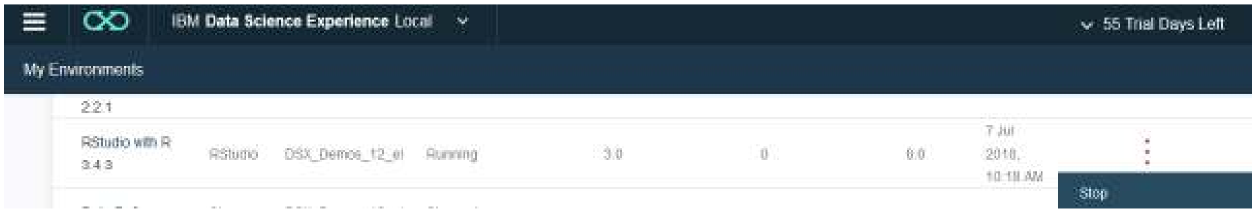
By default reservations are turned off. With this option, RStudio will use all available resources on the server, which could be more or less than the "reserved" option. The majority of server software works this way today (because it doesn't provide the option to "reserve" resources).

The option to reserve resources is a great "self-service" capability for data scientists because they can request more hardware resources without engaging IT. However, it should be used only when needed. If too many users reserve the resources, DSX will display "*Free resources*" message, and will not allow opening any new environments if the system has too many reservations.

DSX provides admin capabilities for managing environments and reservations, but the data scientist should still be aware of best practices.

To release the reservation, the user should stop the runtime from the **Environments** menu.



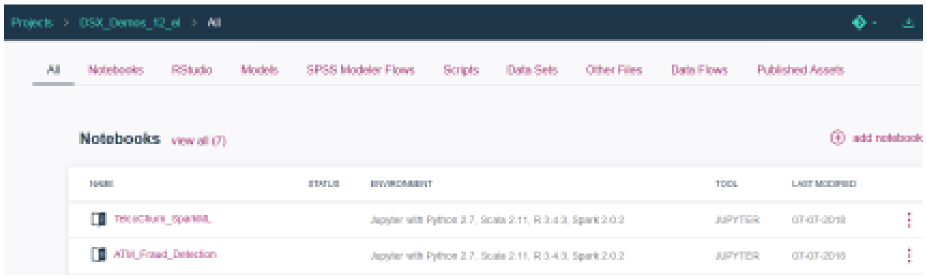


2.2.1							
RStudio with R 3.4.3	RStudio	DSX_Demos_12_el	Running	3.0	0	8.0	7 Jul 2018, 10:18 AM

The **All Active Environments** menu is the view that's available to *Admin* users in DSX, and it's just one of the ways to monitor utilization of a DSX cluster. More detailed management options, which are typically used by the system admin, are available in the **Admin Console**.

Note: If you are sharing the cluster in a workshop, please don't reserve resources or release the reservation if you enabled it.

17. In the **Project** view click **Assets**. This view contains all assets that can be stored in the DSX project.



NAME	STATUS	ENVIRONMENT	TOOL	LAST MODIFIED
TelcoChurn_SparkML		Jupyter with Python 2.7, Scala 2.11, R 3.4.3, Spark 2.0.2	JUPYTER	07-07-2018
ATM_Fraud_Detection		Jupyter with Python 2.7, Scala 2.11, R 3.4.3, Spark 2.0.2	JUPYTER	07-07-2018

In the next section we will review several open source IDEs that can be used in DSX:

- **Jupyter**
- **Zeppelin**
- **RStudio**

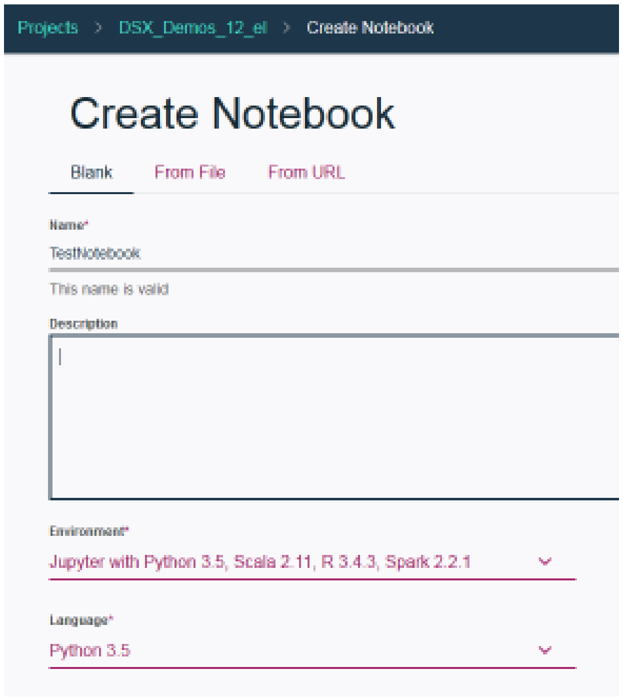
We don't provide detailed instructions for how to use each IDE because many reference resources exist for each one of these IDEs. However, we will point out additional capabilities that have been added for the IDEs in DSX.

Work in Jupyter Notebooks

DSX includes the *Anaconda* distribution of *Jupyter* notebooks. The versions are listed in **Compatibility Reports** for DSX (see **Reference** section).

In addition to prepackaged libraries, users can add libraries for user and global scope. If you would like to learn more about library management in DSX, see **Appendix B**.

1. Create a new *Jupyter 3.5* notebook by clicking **add notebook**.

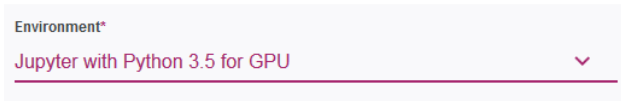


Name*
TestNotebook
This name is valid
Description

Environment*
Jupyter with Python 3.5, Scala 2.11, R 3.4.3, Spark 2.2.1

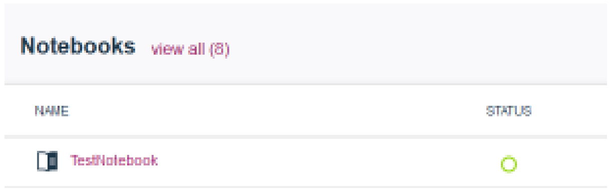
Language*
Python 3.5

When we click **add notebook**, we have several options for the environment and language. One of the options is *Jupyter with Python 3.5 with GPU*. If the DSX cluster has servers with GPUs, then Python 3.5 Jupyter notebooks will automatically run on GPUs when this environment is selected.



Environment*
Jupyter with Python 3.5 for GPU

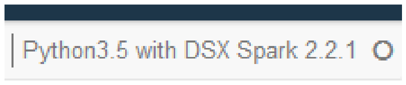
2. When you first open the notebook, it has to start the *Jupyter* server, and it typically take longer than subsequent startups. If you navigate back to the **Assets** view, you will notice that the notebook is now in the running status.



NAME	STATUS
TestNotebook	○

3. Open the notebook again – the startup time should be faster.

4. The notebook automatically connects to a Spark kernel. If connection terminates, DSX will automatically reconnect to Spark.



However, the notebook will not run in Spark unless Spark APIs are used. This is the same behavior in all IDEs that use Python/R Spark (including IDEs outside of DSX). If Python or R API is used, then code will run in Jupyter runtime. Earlier we reviewed the "environments configuration", which determined hardware resources used by the runtime.

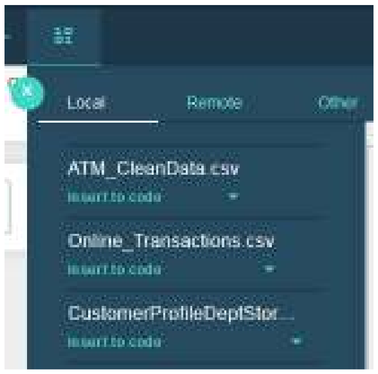
If Spark APIs are used (PySpark, SparkR, Scala, etc.), then the process will run in Spark, which means that it can be distributed across the cluster.

DSX configures the default number of Spark executors per process, but it can be changed programmatically. See **Reference** section for more information.

5. In your test notebook click on the **Data** icon.

Local datasets are flat files that have been imported into a DSX project. They are stored securely on a file system in the cluster. Please note that any collaborator on the project will be able to view local datasets.

Files can be loaded manually or via an API (for example, as a batch load process).

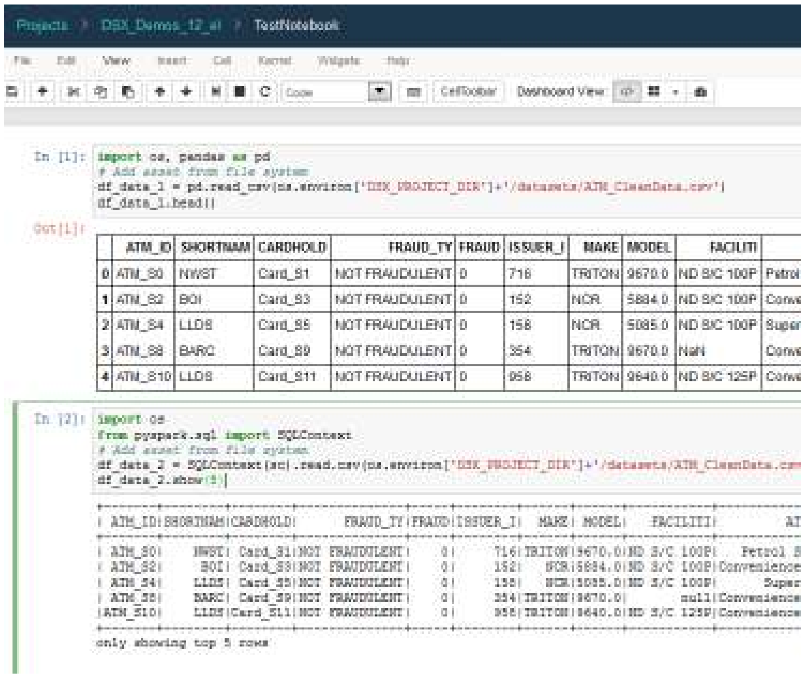


Remote datasets are database tables and Hadoop data sources (HDFS/Hive). Since credentials are not shared between collaborators, while the dataset will be visible in the UI, the user won't be able to access the data unless they specify their credentials.

When we connect to a data source (local or remote), data is loaded into the Notebook environment when the notebook runs. DSX supports remote execution of Spark, Python, and R code. If remote execution is used, then data is not brought over to DSX, which can potentially improve performance. We cover this in more detail in **Remote Execution** lab.

6. Try generating code by clicking on the **Insert to Code** option (both *Pandas* and *Spark* data frames) for any of the data sets, then run the code.

   Hint: insert code for each data frame type into individual cells to view the output.


Projects / DSX_Demos_12_el / TestNotebook
File Edit View Insert Cell Kernel Widgets Help
Code
CellToolbar
Dashboard View

```
In [1]: import os, pandas as pd
        # Add asset from file system
        df_data_1 = pd.read_csv(os.environ['DSX_PROJECT_DIR']+'/datasets/ATM_CleanData.csv')
        df_data_1.head()
Out[1]:
```

	ATM_ID	SHORTNAM	CARDHOLD	FRAUD_TY	FRAUD	ISSUER_I	MAKE	MODEL	FACILITI	
0	ATM_S0	NWST	Card_S1	NOT FRAUDULENT	0	716	TRITON	9670.0	ND S/C 100P	Petrol
1	ATM_S2	BOI	Card_S3	NOT FRAUDULENT	0	152	NCR	5884.0	ND S/C 100P	Conve
2	ATM_S4	LLDS	Card_S5	NOT FRAUDULENT	0	158	NCR	5085.0	ND S/C 100P	Super
3	ATM_S8	BARC	Card_S9	NOT FRAUDULENT	0	354	TRITON	9670.0	NaN	Conve
4	ATM_S10	LLDS	Card_S11	NOT FRAUDULENT	0	958	TRITON	9640.0	ND S/C 125P	Conve

```
In [2]: import os
        from pyspark.sql import SQLContext
        # Add asset from file system
        df_data_2 = SQLContext(sc).read.csv(os.environ['DSX_PROJECT_DIR']+'/datasets/ATM_CleanData.csv
        df_data_2.show(5)

        +------+--------+--------+--------------+-----+--------+------+------+-----------+------------
        | ATM_ID|SHORTNAM|CARDHOLD|      FRAUD_TY|FRAUD|ISSUER_I|  MAKE| MODEL|   FACILITI|          AT
        +------+--------+--------+--------------+-----+--------+------+------+-----------+------------
        | ATM_S0|    NWST| Card_S1|NOT FRAUDULENT|    0|     716|TRITON|9670.0|ND S/C 100P|   Petrol S
        | ATM_S2|     BOI| Card_S3|NOT FRAUDULENT|    0|     152|   NCR|5884.0|ND S/C 100P|Convenience
        | ATM_S4|    LLDS| Card_S5|NOT FRAUDULENT|    0|     158|   NCR|5085.0|ND S/C 100P|       Super
        | ATM_S8|    BARC| Card_S9|NOT FRAUDULENT|    0|     354|TRITON|9670.0|       null|Convenience
        |ATM_S10|    LLDS|Card_S11|NOT FRAUDULENT|    0|     958|TRITON|9640.0|ND S/C 125P|Convenience
        +------+--------+--------+--------------+-----+--------+------+------+-----------+------------
        only showing top 5 rows
```

7. Now let's take a look at a *Jupyter* notebook that predicts customer churn for a telco company. Navigate back to the **Assets** view and open the *TelcoChurn_SparkML* notebook.

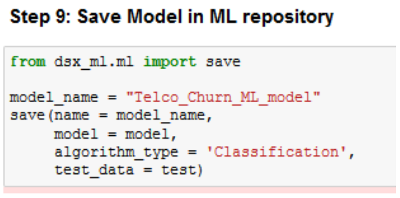
   TelcoChurn_SparkML

8. Review and run the notebook.

   The majority of the code in the notebook is standard open source code that's used for various steps in the predictive analytics process.

   The first API that's provided by IBM is in the cell that saves the model to DSX repository.

   After the model is saved, it's displayed in the **Models** section of the **Assets**.

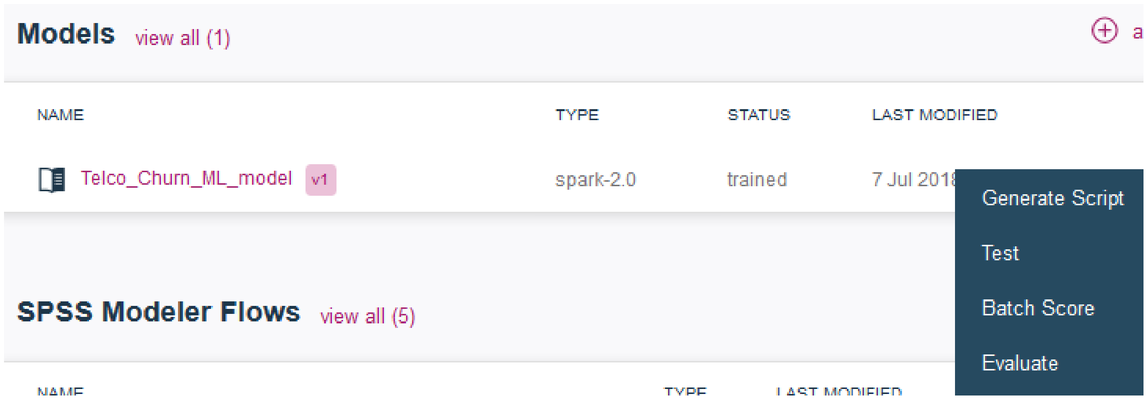


Step 9: Save Model in ML repository

```
from dsx_ml.ml import save

model_name = "Telco_Churn_ML_model"
save(name = model_name,
     model = model,
     algorithm_type = 'Classification',
     test_data = test)
```

Saving model in the DSX repository has several benefits:

- ***Governance***: Models are stored in the same project as the notebook that created it.
- ***Versioning***: DSX provides automatic model versioning. If you run the notebook more than once, the model version in the Models view is incremented.
- ***Simplified testing and deployment***: after the model is displayed in Models view, we can use DSX-built functions to deploy it for online and batch scoring. We cover deployment in more detail in the **DSX Deployment** hands-on lab.



NAME	TYPE	STATUS	LAST MODIFIED
Telco_Churn_ML_model v1	spark-2.0	trained	7 Jul 2018

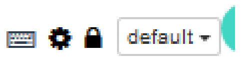
Please let the instructor know if you have any questions about *Jupyter* in DSX.

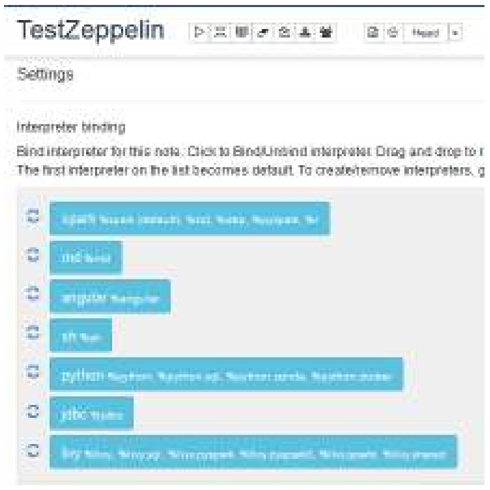
Work in Zeppelin

All DSX-specific information that we reviewed for *Jupyter* also applies to *Zeppelin*.

1. If you wish, create a new *Zeppelin* notebook and try displaying installed libraries and data source code generation as we have done in *Jupyter*.

   Note: before running any cells in a Zeppelin, change the settings to move the ***Livy Interpreter*** *to the bottom. Livy is used for remote Spark execution.*





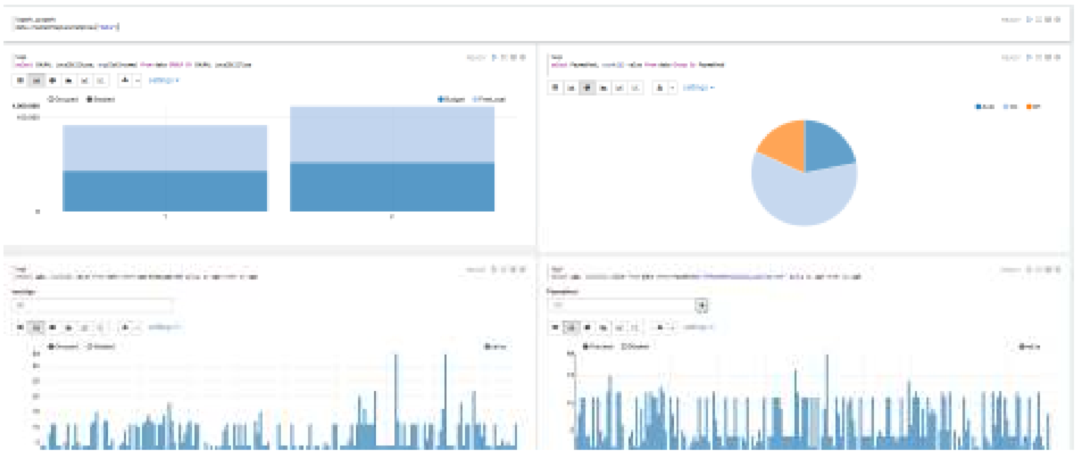
Settings

Interpreter binding

Bind interpreter for this note. Click to Bind/Unbind interpreter. Drag and drop to r
The first interpreter on the list becomes default. To create/remove interpreters, g

2. Next, open the *TelcoChurnZeppelin* notebook. Review and run it. The notebook implements the same use case as the Jupyter notebook we reviewed in the previous section.

   One of the differences is the interactive SQL visualizations, which are only available in *Zeppelin*.



Please let the instructor know if you have any questions about *Zeppelin* in DSX.

Work in RStudio

Projects that are created in *RStudio* are automatically saved in the DSX project, which means that RStudio projects are saved in the DSX or external Git repository (depends on project configuration). By default, DSX repository is enabled for HA and backup.

If an R project contains a Shiny application, it will be automatically detected by DSX, and an option to publish a Shiny application (accessible via URL) will be presented.

Similar to notebook IDEs, the API to save R models in the DSX repository is available in RStudio. As discussed earlier, the saved model can take advantage of built-in governance, versioning, and deployment capabilities.

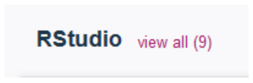
While "insert to code" is not supported for datasets in RStudio, we can generate the code in a Jupyter/R notebook and paste it to RStudio .

You can find more information in DSX documentation: https://content-dsxlocal.mybluemix.net/docs/content/local-dev/ml-r-models.html

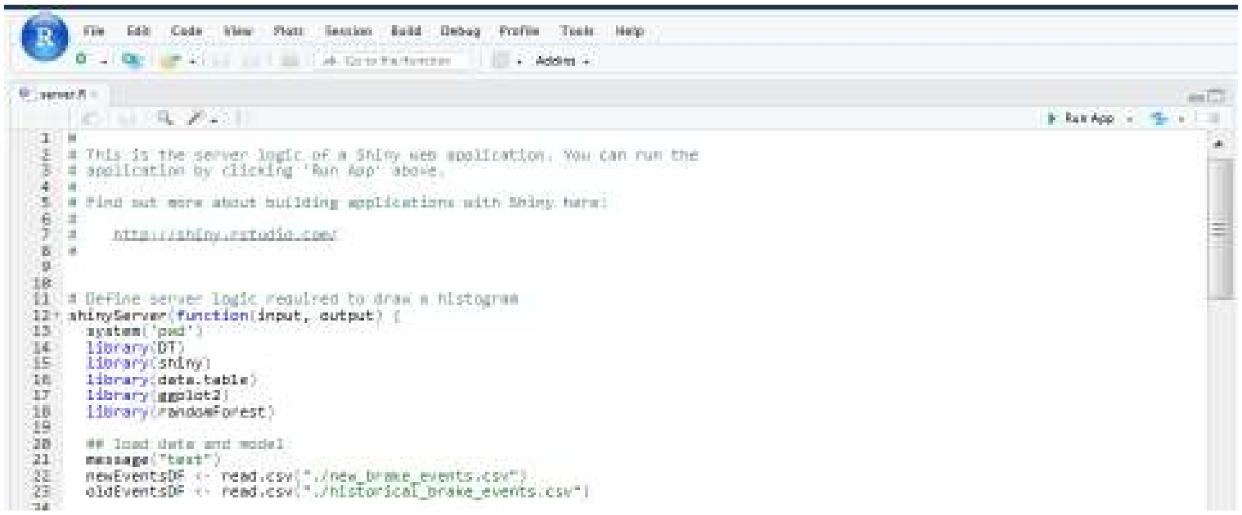
1. From the **Assets** view click open the *DriverClassification* notebook and review it.

   This notebook builds a model to classify driver type based on break events. The notebook saves the model, which is used in the Shiny application for interactive scoring.

2. Click on **view all** next to RStudio. Then click on *demoBreakEvents* Shiny app to open it.

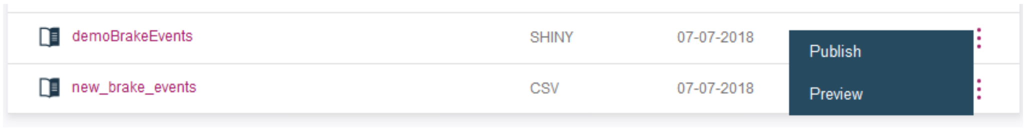


3. If you wish, review code – *server.R*. Click **Run App** and select *Try again* if you see the popup warning.



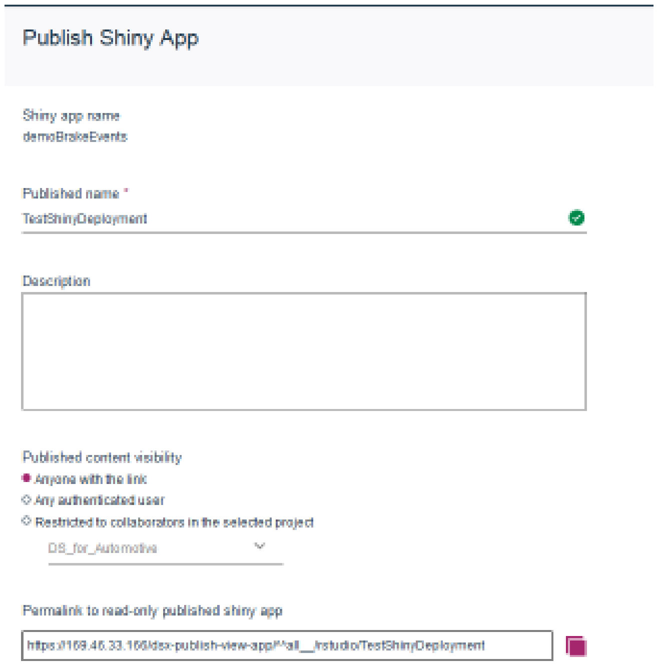
```
#
# This is the server logic of a Shiny web application. You can run the
# application by clicking 'Run App' above.
#
# Find out more about building applications with Shiny here:
#
#    http://shiny.rstudio.com/
#


# Define server logic required to draw a histogram
shinyServer(function(input, output) {
  system('pwd')
  library(DT)
  library(shiny)
  library(data.table)
  library(ggplot2)
  library(randomForest)

  ## load data and model
  message("test")
  newEventsDF <- read.csv("./new_brake_events.csv")
  oldEventsDF <- read.csv("./historical_brake_events.csv")
```

4. Shiny app is displayed. If you wish, you can explore visualizations and online scoring.
5. Navigate back to the Project view. Notice that the *demoBreakEvents* has the option to publish the Shiny app.



demoBrakeEvents	SHINY	07-07-2018
new_brake_events	CSV	07-07-2018

6. Click **Publish** and provide *Published name*. You can choose any *Published visibility*.



Publish Shiny App

Shiny app name
demoBrakeEvents

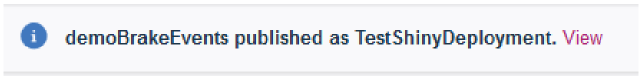
Published name *
TestShinyDeployment

Description

Published content visibility
[✓ Anyone with the link]
Any authenticated user
Restricted to collaborators in the selected project
DS_for_Automotive

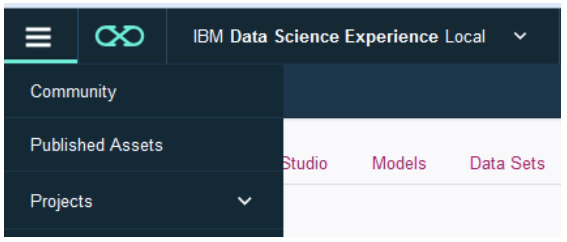
Permalink to read-only published shiny app
https://169.46.33.166/dsx-publish-view-app/^^all__/rstudio/TestShinyDeployment

7. After the application is published, DSX displays message with a shortcut to view it.



demoBrakeEvents published as TestShinyDeployment. View

Later you can look up the URL in the **Published Assets** view. You can also unpublish the app from the same view.



In this hands-on lab we published the Shiny app in the development environment. Typically this environment will be used for testing only. Shiny apps can also be published as a part of deployment. This process is explained in more detail in **DSX Deployment** hands-on lab.

Please let the instructor know if you have any questions about *RStudio* in DSX.

Appendix A: Configure Database Connections

In this section you will learn how to create a data source, datasets, and test connectivity in DSX.

Part 1: Set up a database

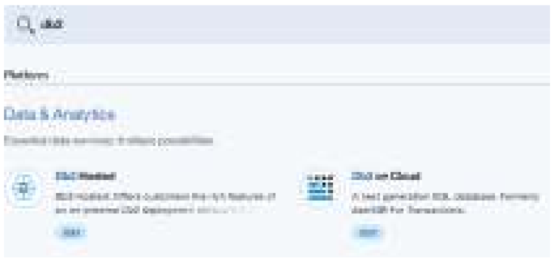
In this section we will set up a database in IBM Cloud so that we can test database access from DSX. If you already have an external database that you would like to use, you can skip this section.

At this time the names *dashDB* and *DB2 on Cloud* are used interchangeably.

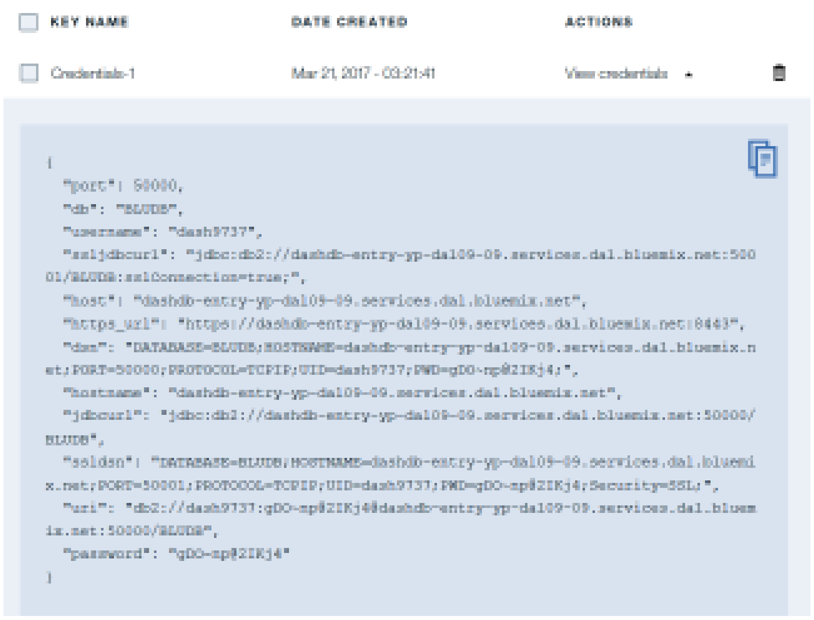
Note: If you are not able to create the DB2 on Cloud service, please ask the lab instructor for a pre-configured instance.

1. Create a DB2 on Cloud service in IBM Cloud

    - Login to **Bluemix**: bluemix.net
    - Search for "db2 on cloud" and create the service


db2
Platform
Data & Analytics
Db2 Hosted
Db2 on Cloud
A next generation SQL database. Formerly dashDB for Transactions.

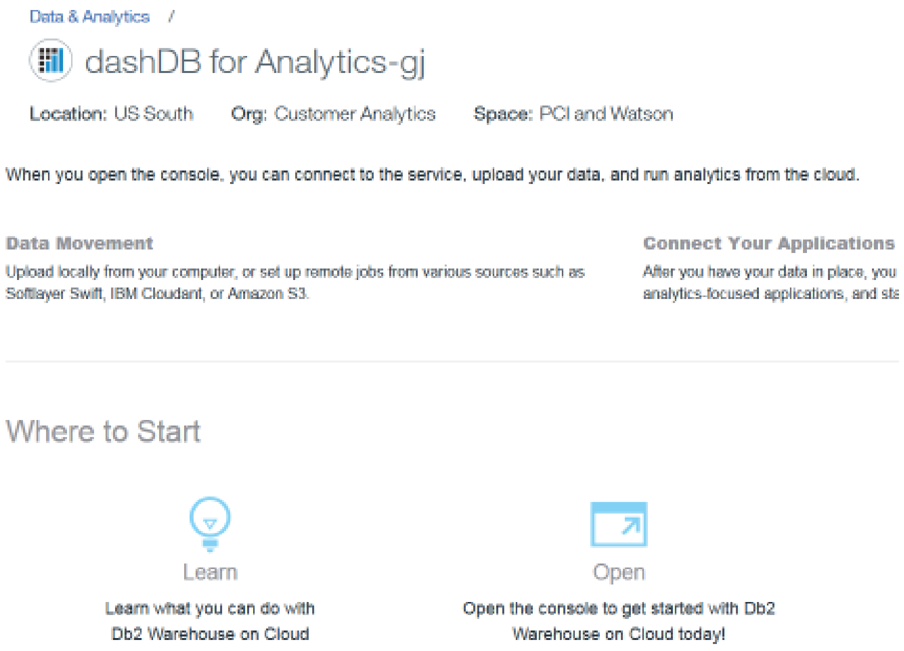
2. Lookup service credentials in Bluemix and save them in a notepad



KEY NAME	DATE CREATED	ACTIONS
Credentials-1	Mar 21, 2017 - 03:21:41	View credentials

```
{
  "port": 50000,
  "db": "BLUDB",
  "username": "dash9737",
  "ssljdbcurl": "jdbc:db2://dashdb-entry-yp-dal09-09.services.dal.bluemix.net:500
01/BLUDB:sslConnection=true;",
  "host": "dashdb-entry-yp-dal09-09.services.dal.bluemix.net",
  "https_url": "https://dashdb-entry-yp-dal09-09.services.dal.bluemix.net:8443",
  "dsn": "DATABASE=BLUDB;HOSTNAME=dashdb-entry-yp-dal09-09.services.dal.bluemix.n
et;PORT=50000;PROTOCOL=TCPIP;UID=dash9737;PWD=gDO-np@2IKj4;",
  "hostname": "dashdb-entry-yp-dal09-09.services.dal.bluemix.net",
  "jdbcurl": "jdbc:db2://dashdb-entry-yp-dal09-09.services.dal.bluemix.net:50000/
BLUDB",
  "ssldsn": "DATABASE=BLUDB;HOSTNAME=dashdb-entry-yp-dal09-09.services.dal.bluemi
x.net;PORT=50001;PROTOCOL=TCPIP;UID=dash9737;PWD=gDO-np@2IKj4;Security=SSL;",
  "uri": "db2://dash9737:gDO-np@2IKj4@dashdb-entry-yp-dal09-09.services.dal.bluem
ix.net:50000/BLUDB",
  "password": "gDO-np@2IKj4"
}
```

4. Click Open to open the dashDB console



Data & Analytics /

dashDB for Analytics-gj

Location: US South Org: Customer Analytics Space: PCI and Watson

When you open the console, you can connect to the service, upload your data, and run analytics from the cloud.

Data Movement

Upload locally from your computer, or set up remote jobs from various sources such as Softlayer Swift, IBM Cloudant, or Amazon S3.

Connect Your Applications

After you have your data in place, you analytics-focused applications, and sta

Where to Start

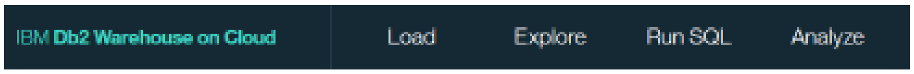
Learn

Learn what you can do with Db2 Warehouse on Cloud

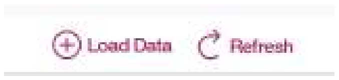
Open

Open the console to get started with Db2 Warehouse on Cloud today!

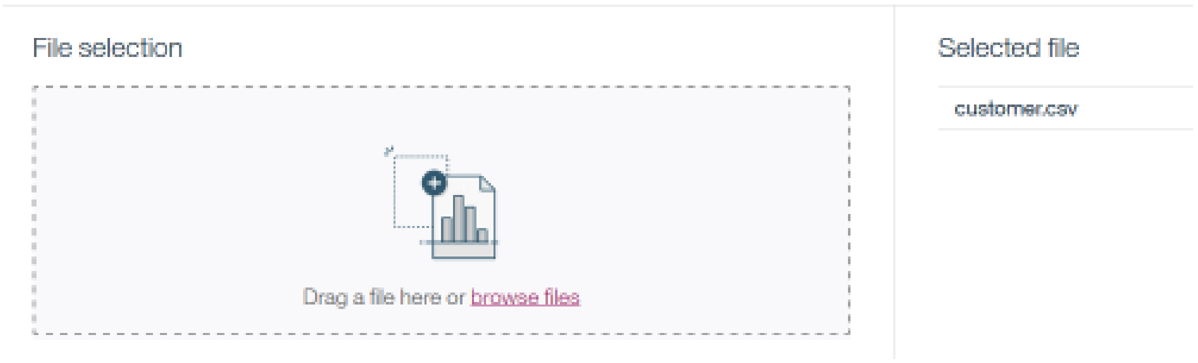
3. Click **Load**



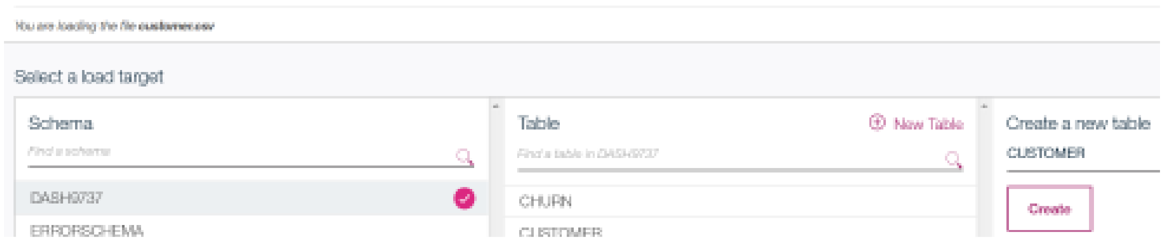
4. Click **Load Data**



5. Select **browse files** and navigate to the *data* folder of the unzipped GitHub repository. Select *customer.csv*. Click **Next**.



6. Select *Schema* (which will be different than the screenshot in your instance) and click **New Table**. Enter table name *CUSTOMER* and click **Create**. Click **Next**.

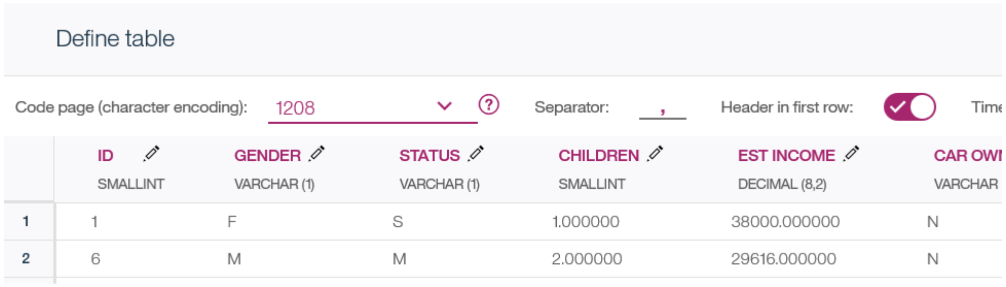


You are loading the file customer.csv

Select a load target

Schema	Table	Create a new table
Find a schema	Find a table in DASH9737	CUSTOMER
DASH9737 ✓	CHURN	Create
ERRORSCHEMA	CUSTOMER	

7. Leave the default values on the *Define Table* screen. Click **Next**. Then **Begin Load**.

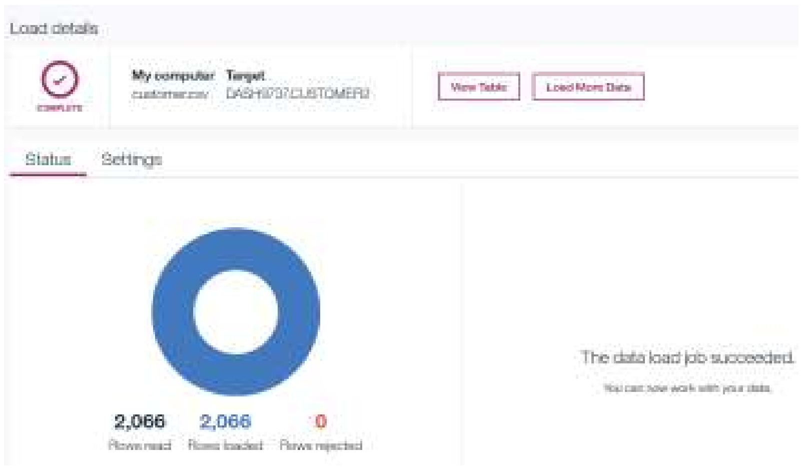


Define table

Code page (character encoding): 1208 Separator: , Header in first row: ✓ Time

	ID	GENDER	STATUS	CHILDREN	EST INCOME	CAR OWI
	SMALLINT	VARCHAR (1)	VARCHAR (1)	SMALLINT	DECIMAL (8,2)	VARCHAR
1	1	F	S	1.000000	38000.000000	N
2	6	M	M	2.000000	29616.000000	N

8. If you want to verify that data has been loaded successfully, click **View Table**.



Load details

COMPLETE

My computer: customer.csv
Target: DASH9737.CUSTOMER2

2,066 Rows read
2,066 Rows loaded
0 Rows rejected

The data load job succeeded.

You can now work with your data.

9. If you want to convert all sample notebooks to the database data sources, then repeat the data load steps for all files in the */data* directory.

Part 2: Configure database connection in DSX

In this section we will define a connection in the DSX UI and test it in a notebook.

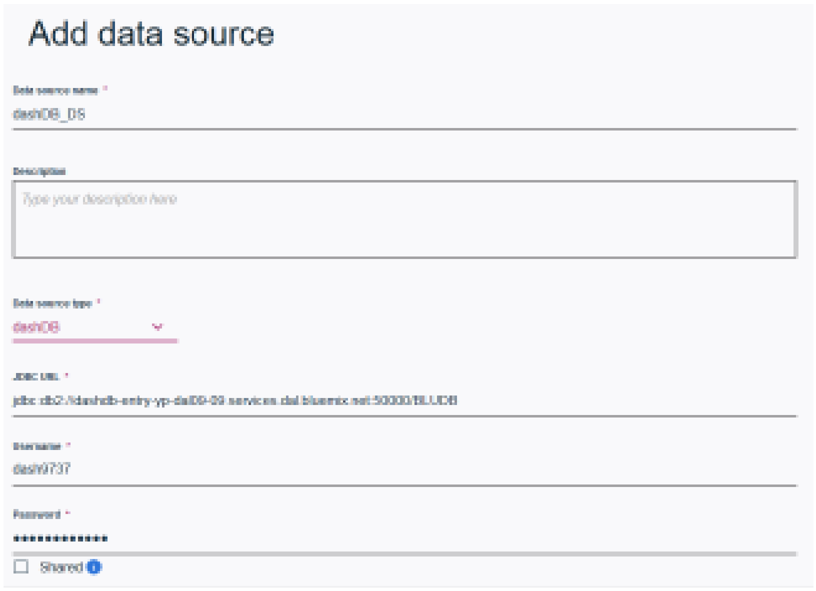
1. Open your DSX Local project.

2. Click on **Data Sources**, then **add data source**

3. Enter data source name (for example, dashDB_DS) and fill out the required fields (which you saved from *Service Credentials* view in **IBM Cloud**).

   Do not check the “Shared” checkbox. If you select it, then your credentials will be shared with collaborators on the project.

   Click **Create**.



Add data source

Data source name *
dashDB_DS

Description
Type your description here

Data source type *
dashDB

JDBC URL *
jdbc:db2://dashdb-entry-yp-dal09-09.services.dal.bluemix.net:50000/BLUDB

Username *
dash9737

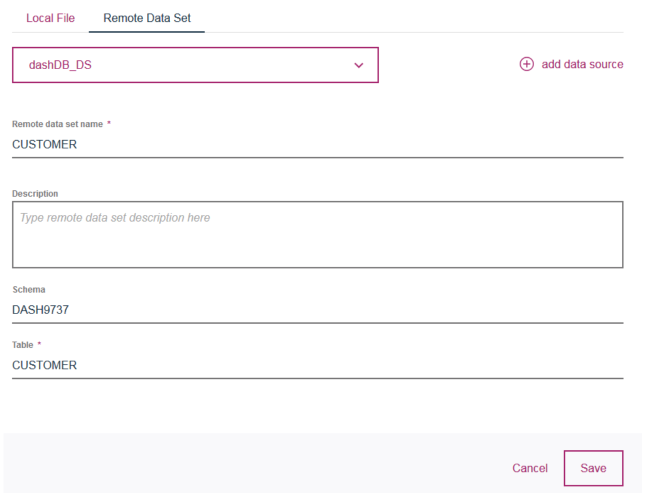
Password *
****

☐ Shared

4. Switch to the **Assets** view and scroll down to **Data Sets**. Click **add data set**. Select the created data source from the dropdown and enter the required fields.

   In our example we gave the same name to the data set as the table name – *CUSTOMER.* Your schema name will be different.

   Click **Save**.



Local File | Remote Data Set

dashDB_DS

add data source

Remote data set name *
CUSTOMER

Description
Type remote data set description here

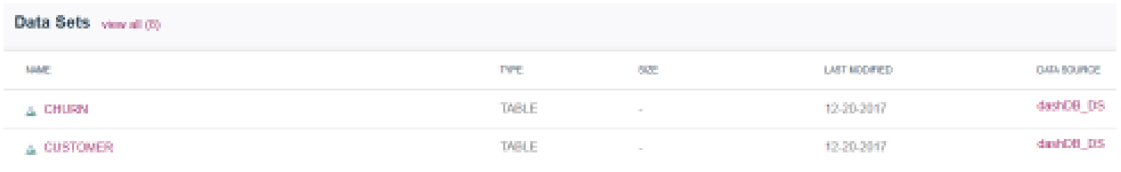
Schema
DASH9737

Table *
CUSTOMER

Cancel | Save

5. If you created tables from other CSV files, create the **Remote Data Source** for each of them.

   Notice that the remote data sets are shown as tables.



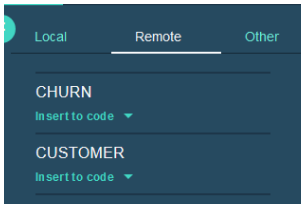
Data Sets view all (5)

NAME	TYPE	SIZE	LAST MODIFIED	DATA SOURCE
CHURN	TABLE	-	12-20-2017	dashDB_DS
CUSTOMER	TABLE	-	12-20-2017	dashDB_DS

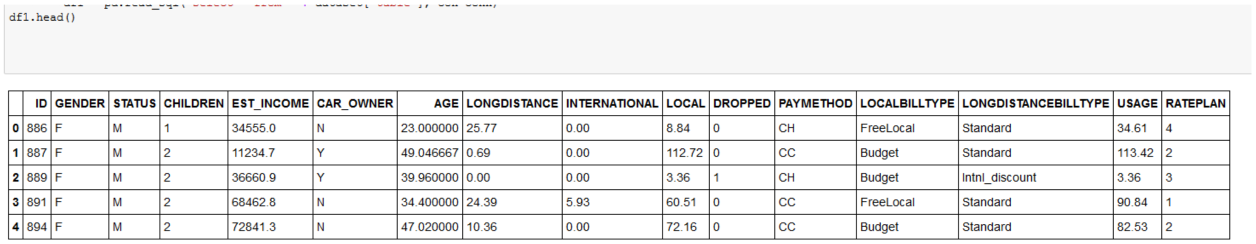
Part 3: Test Database Connection

1. To test the connection, create a new Jupyter/Python notebook.

2. Click on the data icon, then **Remote** tab and select the **Insert to code** option for one of the remote data sources. You can test both Spark and Pandas data frames.



3. Run the code and make sure data is displayed



```
df1.head()
```

	ID	GENDER	STATUS	CHILDREN	EST_INCOME	CAR_OWNER	AGE	LONGDISTANCE	INTERNATIONAL	LOCAL	DROPPED	PAYMETHOD	LOCALBILLTYPE	LONGDISTANCEBILLTYPE	USAGE	RATEPLAN
0	886	F	M	1	34555.0	N	23.000000	25.77	0.00	8.84	0	CH	FreeLocal	Standard	34.61	4
1	887	F	M	2	11234.7	Y	49.046667	0.69	0.00	112.72	0	CC	Budget	Standard	113.42	2
2	889	F	M	2	36660.9	Y	39.960000	0.00	0.00	3.36	1	CH	Budget	Intnl_discount	3.36	3
3	891	F	M	2	68462.8	N	34.400000	24.39	5.93	60.51	0	CC	FreeLocal	Standard	90.84	1
4	894	F	M	2	72841.3	N	47.020000	10.36	0.00	72.16	0	CC	Budget	Standard	82.53	2

4. If you wish, change the sample notebooks that use .csv to use a database data source.

   Make sure to insert the correct data frame type.
   - *TelcoChurn* notebook uses Spark data frames
   - *CreditCardDefault* notebook uses Spark data frames

   In addition to generating the code, you may need to change variable names. Please check with the lab instructor if you need help with understanding how to modify the code.

Appendix B: Load Libraries

In this section you will learn how to load libraries in DSX. DSX provides three options for loading libraries:

- User scope
- Global scope (for every user)
- Global scope with image management.

We will walk through "user scope" and "global scope" options.

Image management requires download of a docker image, which can be a time-consuming task. Image management is typically done by the admin when the customer has a requirement to use only specific libraries in an IDE (Jupyter, RStudio, Zeppelin). It's also useful when a customer wants to standardize on an environment for development, deployment, and remote execution (for Python only).

See official documentation for information about image management: https://content-dsxlocal.mybluemix.net/docs/content/local/images.html

In order to run install commands (*pip* in Python and *install.packages()* in R), the cluster must have Internet connectivity. Internet connectivity should be configured by the customer's Linux admin (i.e. it's not something that's done in DSX).

It is also possible to install packages on clusters that don't have Internet connectivity. Please see official documentation: https://content-dsxlocal.mybluemix.net/docs/content/local/admin-libraries.html

Part 1: Python

Python packages can be installed with user or global scope.

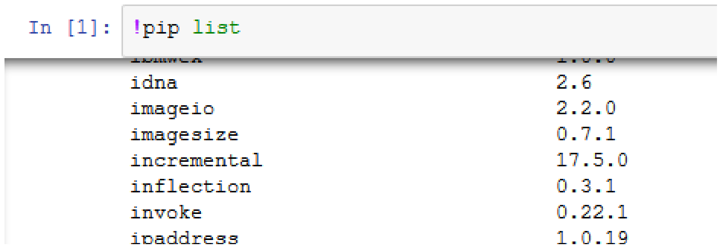
- User scope: **`!pip install <package> --user`**
    - Can be run by any user
    - This will install the package to */user-home/<uid>/.local/lib/python2.7/site-packages* and will be available to this user only.
- Global scope: **`!pip install --target /user-home/_global_/python-2.7 <package>`**
    - Should be run by *admin* user
    - The package will be available to all users.

Note: If installing a package for Python 3.5, replace 2.7 with 3.5 in the path.

1. Login to DSX as a non-admin user (i.e. userid other than *admin*).

2. Create a notebook.

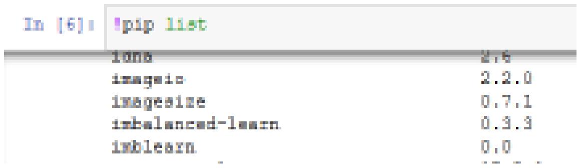
3. Type in command **`!pip list`**. Look at the output to make sure that package *imblearn* is not installed.



```
In [1]: !pip list
idna                                    2.6
imageio                                 2.2.0
imagesize                               0.7.1
incremental                             17.5.0
inflection                              0.3.1
invoke                                  0.22.1
ipaddress                               1.0.19
```

4. Install library *imblearn* for this user and run again to make sure it's installed.

   - **`!pip install imblearn –user`**
   - **`!pip list`**



```
In [6]: !pip list
idna                                    2.6
imageio                                 2.2.0
imagesize                               0.7.1
imbalanced-learn                        0.3.3
imblearn                                0.0
```

Note: you can first try running the command without the --user parameter. Notice that you'll get a permissions error.

5. Login to DSX as user *admin and* run **`!pip list`** to verify that *imblearn is not installed.*

6. Install *imblearn* into the global directory and verify that it's been installed.

   - **`!pip install --target /user-home/_global_/python-2.7 imblearn`**

7. If you would like to install a package into an offline cluster, follow instructions in documentation: https://content-dsxlocal.mybluemix.net/docs/content/local/admin-libraries.html#to-install-a-global-python-library-when-the-cluster-is-not-connected-to-the-internet

Part 2: R

Similar to Python, R packages can be installed with user or global scope.

See official documentation for detailed instructions

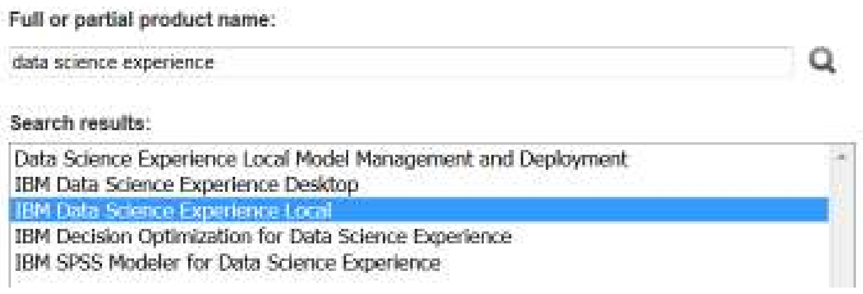
- User scope: https://content-dsxlocal.mybluemix.net/docs/content/analyze-data/importing-libraries.html
- Global scope: https://content-dsxlocal.mybluemix.net/docs/content/local/admin-libraries.html

Reference

DSX documentation: https://content-dsxlocal.mybluemix.net/docs/content/local/welcome.html

Spark Configuration in DSX: https://content-dsxlocal.mybluemix.net/docs/content/analyze-data/creating-notebooks.html#set-dsx-spark-resources

Compatibility Reports (data sources, versions of Anaconda, Spark, etc.: http://publib.boulder.ibm.com/infocenter/prodguid/v1r0/clarity/prereqsForProduct.html


Full or partial product name:

data science experience

Search results:

Data Science Experience Local Model Management and Deployment
IBM Data Science Experience Desktop
IBM Data Science Experience Local
IBM Decision Optimization for Data Science Experience
IBM SPSS Modeler for Data Science Experience