Lab: Deployment in Watson Studio Local

November 19, 2018

Author: Elena Lowery elowery@us.ibm.com

Table of contents

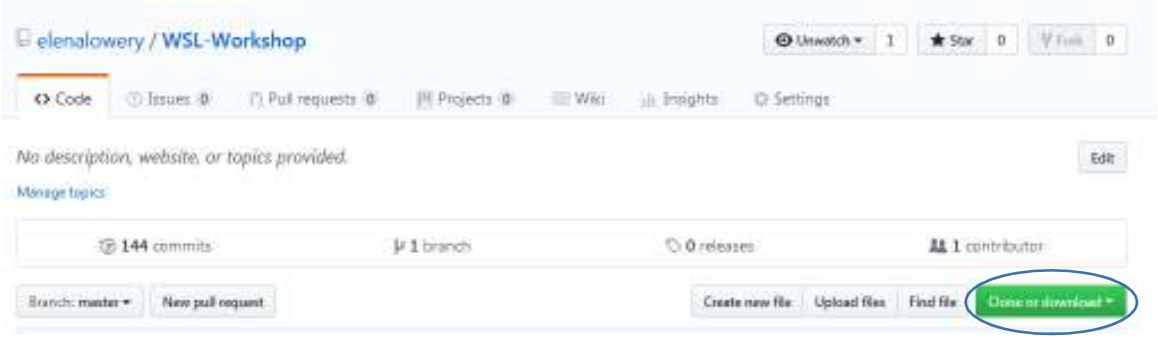
Contents

Overview

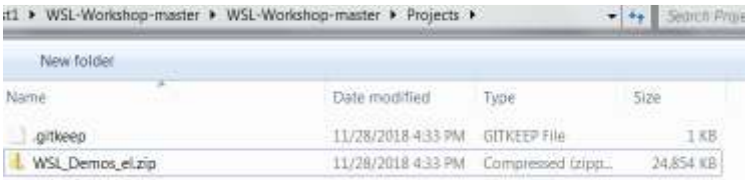
In this lab you will learn how to deploy analytical assets in **Watson Studio Local** (WSL).

Required software, access, and files

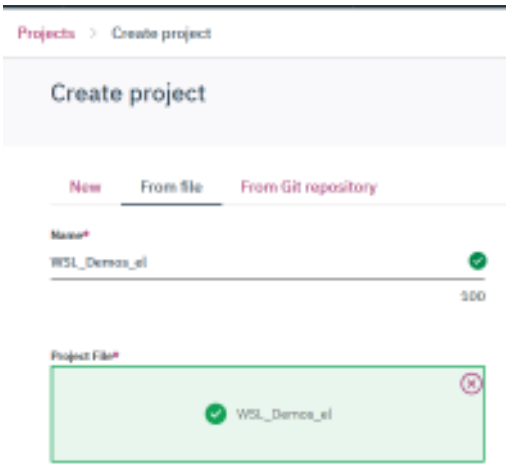
- To complete this lab, you will need access to a **Watson Studio Local** cluster.
- You you will also need to complete the following steps to import the sample project:
    - Download and unzip this GitHub repository: https://github.com/elenalowery/WSL-Workshop



    - In the **Projects** directory of the unzipped file, rename *WSL_Demos.zip* to a unique name, for example, add your initials.



    - Log in to **WSL** and create a project **From File**, using the *WSL_Demos.zip* file that you just renamed.



Deployment in Watson Studio Local

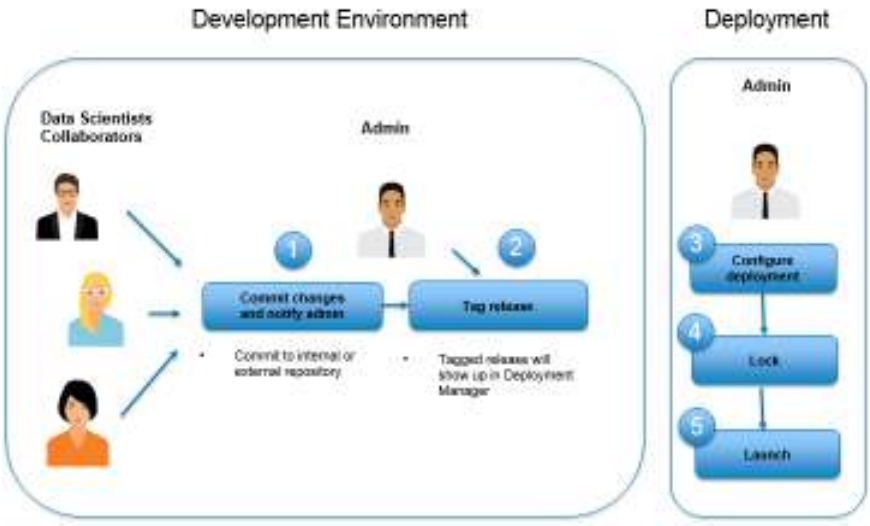
Deployment in WSL is accessed through **Watson Machine Learning (WML)**, which is available in the WSL UI for users with the *deployment* role.

Deployment provides the following capabilities:

- Deployment of models and scripts for real time scoring
- Deployment of notebooks and scripts for batch execution
- Deployment of Shiny applications
- Scheduling of model evaluation.

Some of the deployment tasks can be done in the development environment, but from the licensing perspective, these tasks should only be used for testing.

WML provides important separation of the development and deployment tasks. Here's how the deployment workflow is implemented in WSL.



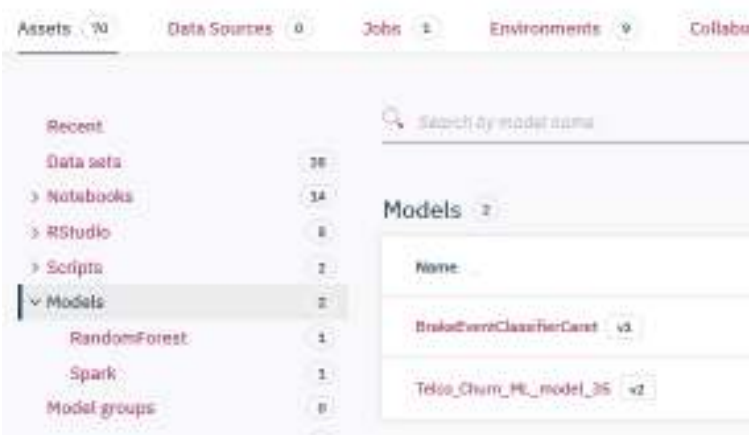
1. Data scientists collaborate on a project, and when they're done with development and testing, they notify the admin that the project is ready to be deployed in production.
2. The admin tags the project release and uses **WML** UI to configure deployments.
3. When the project is launched, the REST APIs for deployed assets become available and all scheduled jobs will run as configured.

Part 1: Test Assets that will be deployed

Note: testing is done in development environment.

Test Real Time Scoring of Models

1. The models that we use in this lab have been created in notebooks:
   - *TelcoChurn_ML_Model_35* is a Spark ML model created in *TecloChurn_Spark_ML_35* notebook
   - *BreakEventClassifier* is an R model created in *DriverClassification* Jupyter notebook



If you wish, you can open the notebooks and review them. The models are saved to WSL repository the following sections of notebooks.

Spark ML model:

Step 9: Save Model in ML repository

```
from dsx_ml.ml import save

model_name = "Telco_Churn_ML_model_35"
save(name = model_name,
     model = model,
     algorithm_type = 'Classification',
     test_data = test)
```

R model:

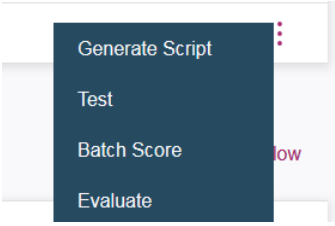
```
library(modelAccess)
library(jsonlite)

saveModel(model = brakeEventModel, name = "BrakeEventClassifierCaret")
```

Both notebooks show a programmatic way to test models, but in this section we will use testing functionality in the UI.

2. In the *WSL_Demos* project navigate to **Models**. Click on the vertical ellipses next to the *Telco_Churn_ML_model_35*.

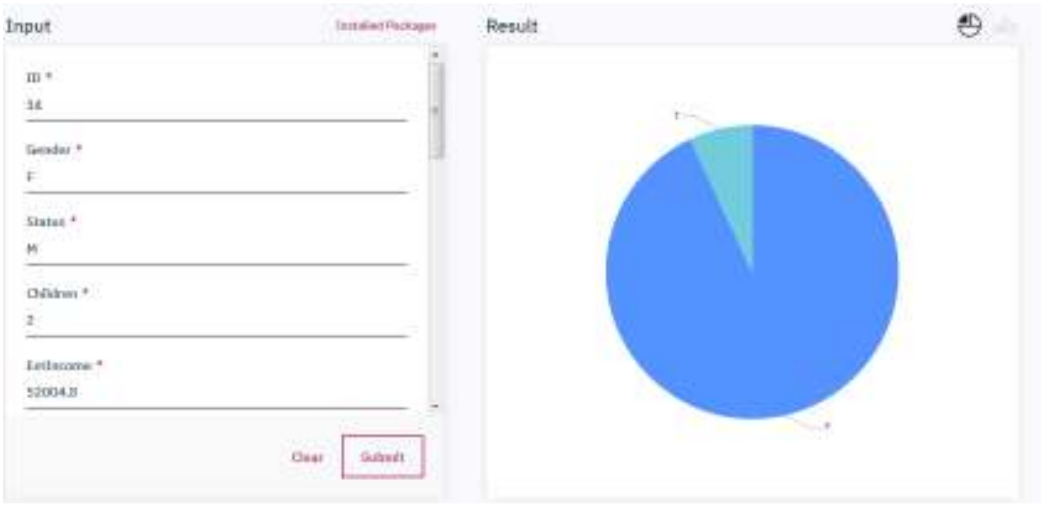
   Notice that you have **Generate Script** option for the model. If you generate a script for online model deployment, you can add addiitonal operations prior to invoking scoring. Generating a script is an *optional* task for online scoring. If you don't generate a script, a default script will be used during deployment.



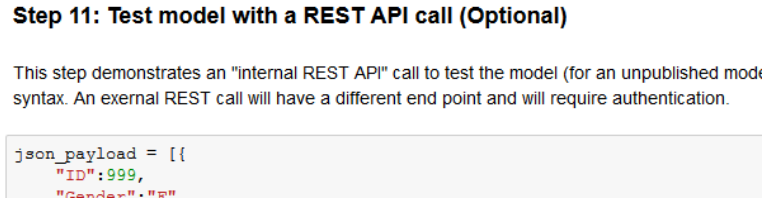
   If you would like to take a look at the default script, click the **Generate Script** button. Take a note of the script name – later you can use it for depoyment.

3. Click on the vertical ellipses next to the model and select **Real-time score**.

   The data for testing is prefilled because we save the training data with the model. You can test with the default data or change some values. For example, change **Status** to *S*.



   It's also possible to test with the *internal* REST API, as shown in the *TelcoChurn_SparkML_35* notebook. The external REST API will be availabe when we deploy the model in the **WML** (later in this lab).

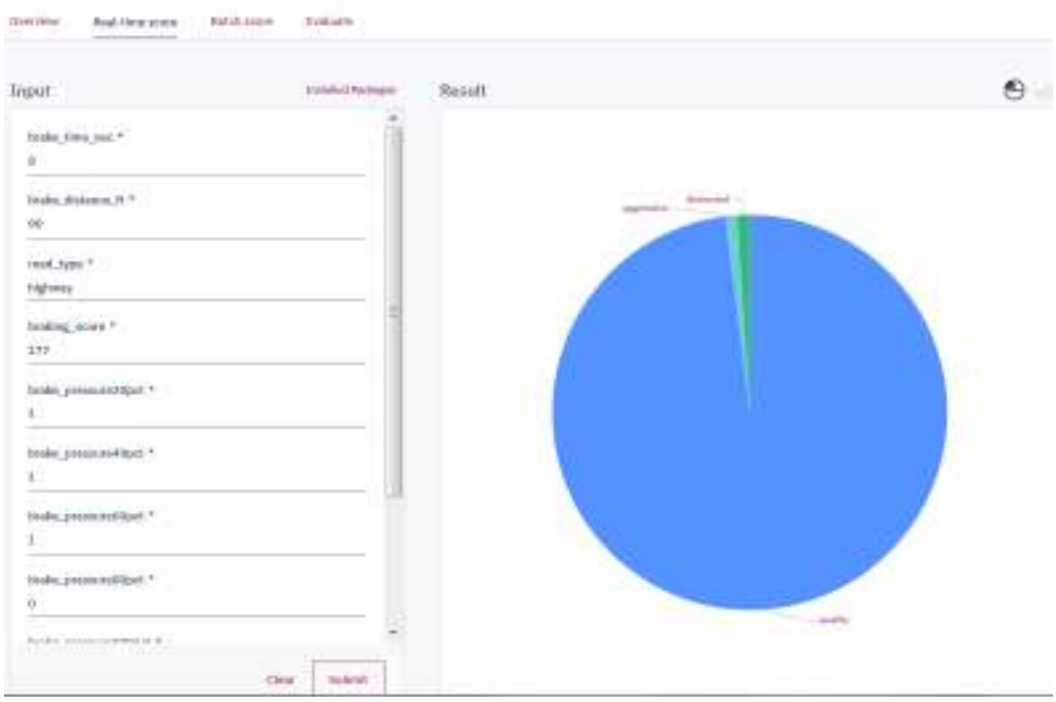


4. In the *WSL_Demos* project navigate to **Models**. Click on the vertical ellipses next to the *BreakEvenClassifierCarret* and select **Real-time Score**.

   Sample data is not stored with the R model - we have to enter it manually. You can use the following data for testing (enter 1 row at a time):

type	brake_time_sec	brake_distance_ft	road_type	braking_score	brake_pressure20pct	brake_pressure40pct	brake_pressure60pct	brake_pressure80pct	brake_pressure100pct	abs_event	travel_speed
quality	7.87	90.04	highway	177	1	1	1	0	0	0	60
quality	5.14	59.37	main road	141	0	0	0	0	0	0	46
quality	4.45	27.09	residential	196	1	1	1	0	0	0	29

   Scoring results



You have finished testing real time scoring in the development environment.

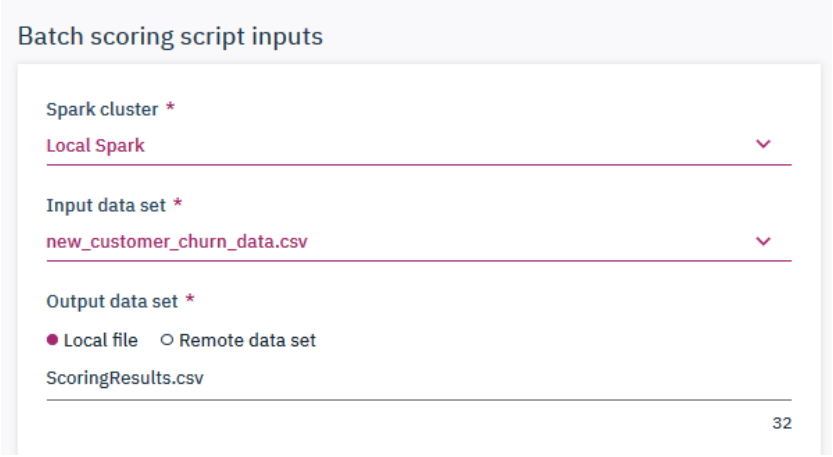
Create Batch Scripts and Test Batch Jobs

In order to deploy a batch job in **WML**, we first need to create a script in the development environment. A data scientist can also test a batch job.

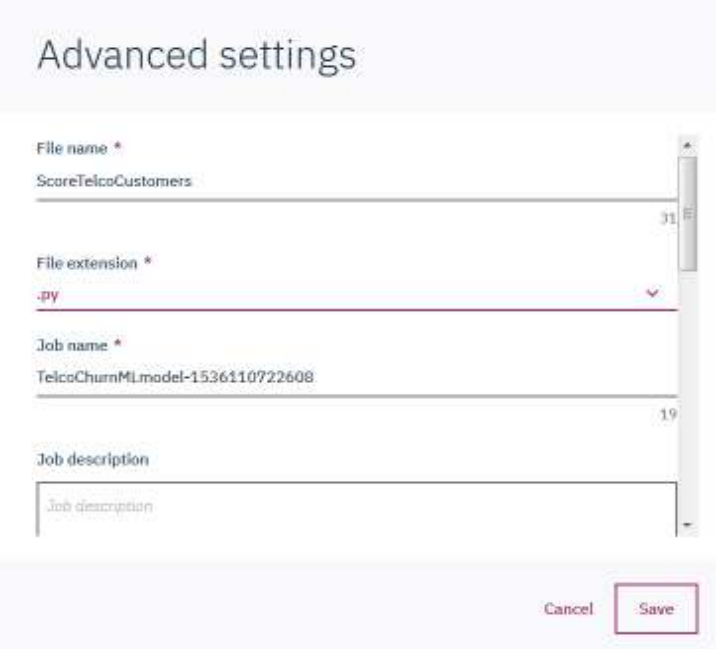
1. Click on the ellipses next to the *TelcoChurn_ML_Model_35* model and select **Batch Score**.

2. Fill out the required fields:

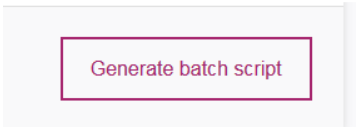
   - **Input data set** (Local): *new_customer_churn_data.csv*
   - **Output data set**: *ScoringResults.csv*. Make sure to provide .csv extension – otherwise you won't be able to preview and download the output.



3. Click on **Advanced Settings** and change the file name to *ScoreTelcoCustomers*. Click **Save**.



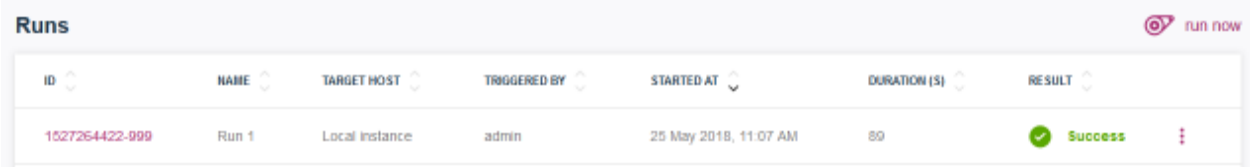
4. Click **Generate Batch Script**.



Note: the batch script can be edited. Examples of changes to the batch script are
- *Invoking another script (for example, ETL) prior to model scoring*
- *Changing input and output data sources.*

5. Click **Run now** and wait till the status changes to *Success*. If you would like to look at the log files, you can click on the job id.

   Note: if the status is Pending for more than a minute, refresh the page.



6. If you wish, navigate back to **Data Sets** to view the batch job results.

7. If you want to test batch scoring of an R model, use the *NewBreakEvents.csv* as the input file for batch scoring.

You have finished testing batch scoring of a model in the development environment.

Create Evaluation Script and Test Evaluation

In order to deploy an evaluation job in **WML**, we first need to create a script in the development environment. A data scientist can also test evaluation.

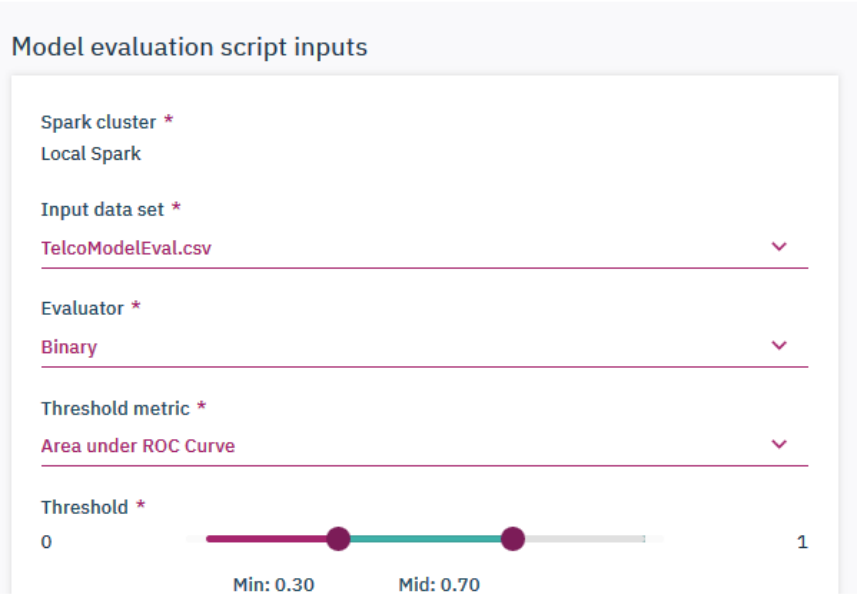
Note: In our example the "Evaluation data set" is subset of data used for modeling. We chose this approach for convenience and demonstration. In a production environment the "Evaluation data set" is the new set of historical data that's used to verify that the model is still accurate. This data set can be automatically uploaded to the data source that's used for evaluation either by a script in WSL or an external process.

1. Click on the ellipses next to the *TelcoChurn_ML_Model_35* model and select **Evaluate**.

2. Select the data source for evaluation (*TelcoModelEval.csv* file which we generated in a notebook).

When we ran evaluation in the notebook we used *BinaryClassificationEvalutor* and *Area Under Roc Curve* as the metric. We suggest that you use the same values when creating the evaluation script.

Keep the default *Threshold* values.

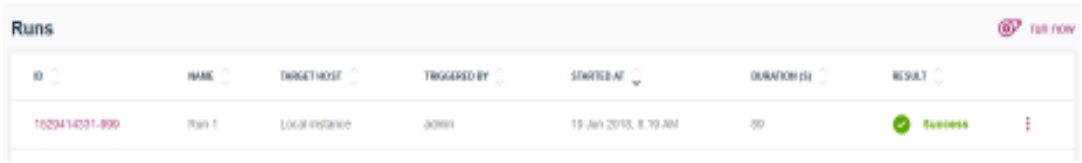


3. Click on **Advanced Settings** and change the name of the script. For example, you can name it *TelcoChurnEvalScript*. Click **Save**.

4. Click **Generate Evaluation Script**.

5. Click **Run now**.

6. Scroll down to review the results and wait till the run has finished.



7. Navigate to **Project** view and click on the model - you will see model evaluation results.

8. If you wish, you can configure 2 more evaluation jobs with different test datasets – *TelcoModelEval2.csv* and *TelcoModelEval3.csv*

   After you run these evaluation jobs, you can look up evaluation results in the Model details view. Notice that accuracy is different when using different evaluation datasets.

   In this screenshot the top row is when evaluation is done with *TelcoModelEval3.csv* and the second from the top is with *TelcoModelEval2.csv*

9. If you would like to test evaluation of an R model, use *Break_Events_Eval.csv* as the evaluation script.

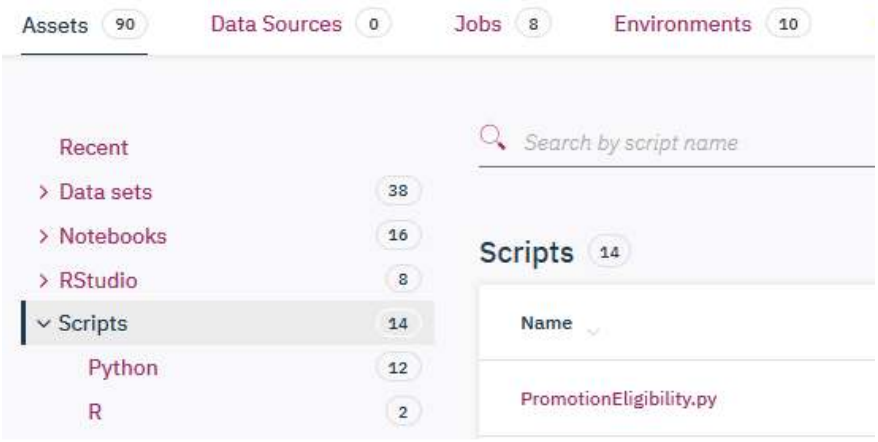
You have finished testing evaluation in the development environment.

Optional: Test Script as a Web Service

WSL supports deployment of *Python* and *R scripts* as Web services. The scripts must be written in a specific format (details provided in documentation). A single function of the script is invoked with a Web services call.

1. Navigate to the *WSL_Demos* project and switch to the **Assets** tab.

2. Scroll to **Scripts,** then click on **PromotionEligibilty.py**.



This script is an example of a script that can be deployed as a Web service. It has one function, *determineEligibility*, which applies business rules to determine a promotion code for a customer.

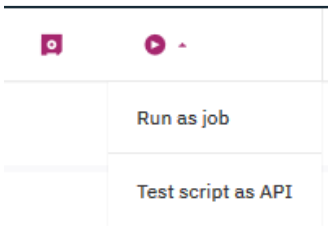
A script can include multiple functions, all of which can be called individually. The script requirements are available in product documentation: https://content-dsxlocal.mybluemix.net/docs/content/SSAS34_current/local-dev/dsxl-scripts-as-web-services.html

```
def determineEligibility(args):

  if args["customerStatus"] == "N" and args["purchaseAmount"] > 50:
      if args["customerSegment"] == "Family":
          promotion = "PW3033"
      else:
          promotion = "PW3034"
  elif args["customerStatus"] == "E" and args["purchaseAmount"] > 100:
      if args["customerSegment"] == "Electronics":
          promotion = "PW3045"
      else:
          promotion = "PW3059"
      promotion = "PW3035"
  else:
      promotion="PW7809"
  return promotion
```

Note: testing can also be done after configuring deployment, and it's a faster process. If you are short on time, you can skip the rest of this section and test in WML.

3. We can test the script with a *curl* command. First, click on the run icon (in the top right corner and select **Test script as API**.

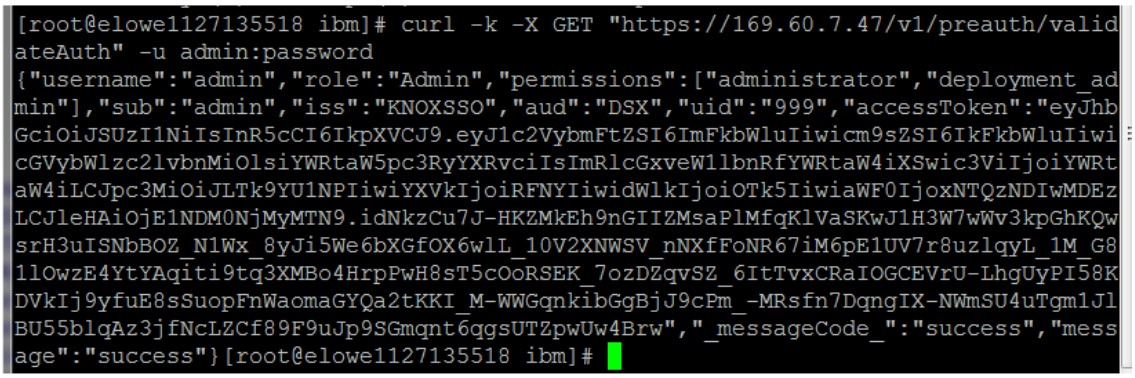


4. Copy both curl commands to *Notepad*. The first command will generate a user token, and the second one is the Web service call.

5. Run the first command from your preferred shell environment (*Command Prompt* in Windows) to get the token. Copy the token to the *Notepad*.

   In this example the **userid** and **password** are *admin/password*. Replace the values with your userid/password.

   Sample output



6. Next, we need to update the curl command to call the Web service.

   Make sure to update the following
   - **Function**: *determineEligibility*
   - **Token**: *the generated token*
   - **Args**: : *{"customerStatus": "N", "purchaseAmount": 55, "customerSegment": "Family"}*

curl -k -X POST "https://169.60.7.47/dsx-py3-script/ibmdsxuser-999/1541102766004/determineEligibility" -H "Content-Type: application/json" -H "Authorization: Bearer eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6ImFkbWluIiwicm9sZSI6IkFkbWluIiwicGVybWlzc2lvbnMiOlsiYWRtaW5pc3RyYXRvciIsImRlcGxveW1lbnRfYWRtaW4iXSwic3ViIjoiYWRtaW4iLCJpc3MiOiJLTk9YU1NPIiwiYXVkIjoiRFNYIiwidWlkIjoiOTk5IiwiaWF0IjoxNTQzMzQ2NTMzLCJleHAiOjE1NDMzODk3MzN9.vkLE42OoR_Rsu-hgFROLODRCBhy0jol0Vszvual00guGBEp1c_umfZAL6V_IwbSop5ltsm6643M_2KeIg0kb7xRnwyYVii3WH7EIA6NQ40CsGYETAhRoOxq8_2o-fsJrd87Yn5Z_oRhEFo_GC9KhdoCsaAlCjHj5JTmLtoCCWHO73rjAaFnbeGONOq5kldsiDTz9WMkJvqrc-aIrmNoh37C0YVabZeZr0G-eUbQVCWVSxGJ08U6ev-pqKSMStwoY8G_kfuuVo1F_Kc5i6R390aAN7ze3FRA5UFTJQpZ5U7iCc6xvr31k5uYtem-Vz5d_dvkDLSqcIdUn9t_UbGJApg" -d '{ "relativeScriptPath": "scripts/PromotionEligibility.py","args": {"customerStatus": "N", "purchaseAmount": 55, "customerSegment": "Family"} }'

Sample output

You have finished testing script as a Web service.

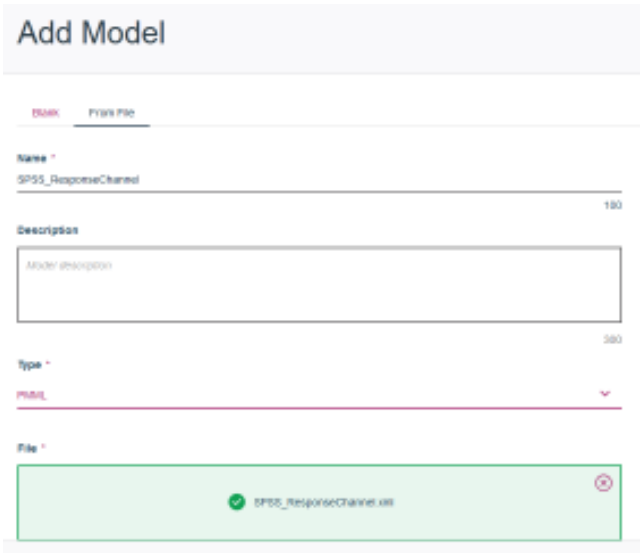
Optional: Import and Test PMML

1. Navigate to the *WSL_Demos* project and switch to the **Assets** tab.

2. Scroll to **Models** and click **add model**. Select **From File**. Provide *model name* and select model type *PMML*.

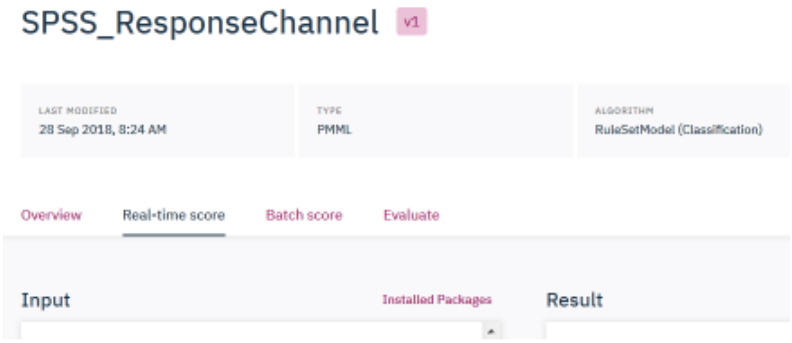
3. Click **Browse**. Navigate to the *PMML/PMML files* folder of the unzipped GitHub repository and select *SPSS_ResponseChannel.xml*. Click **Create**.

Note: this PMML file was generated from a model created in SPSS Modeler. It predicts the recommended marketing channel (direct mail, mobile, e-mail) for a customer.



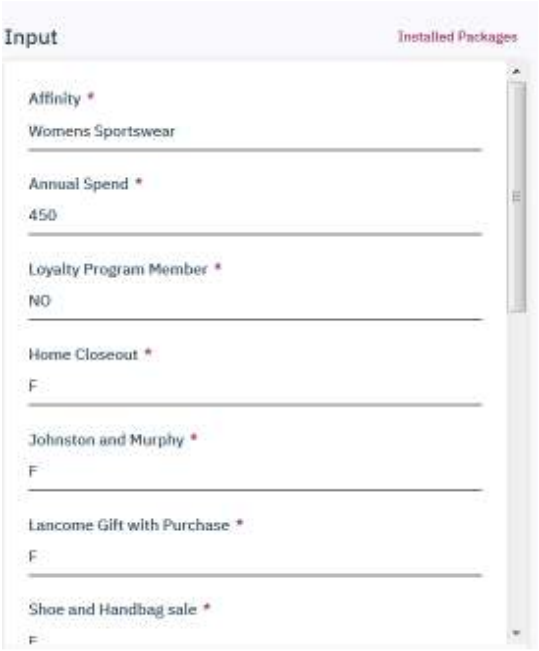
4. Once the model is imported, we can test real time and batch scoring.

   Real Time Scoring test

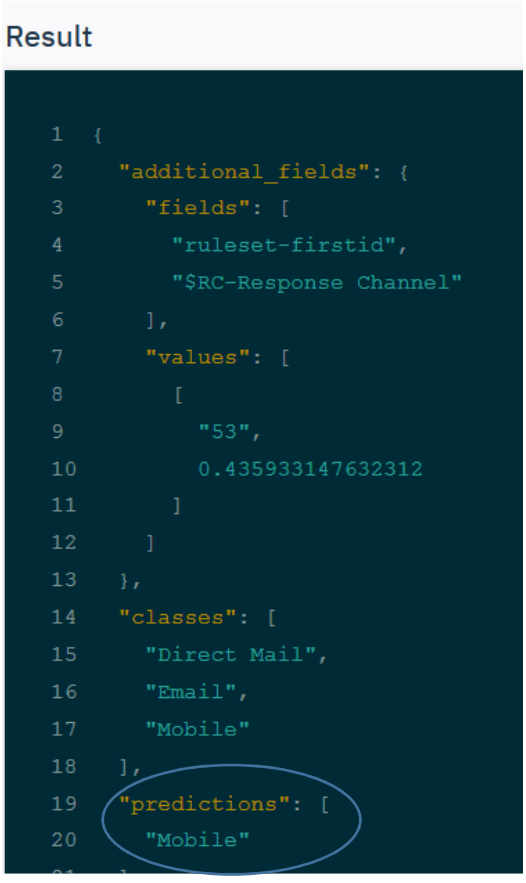


   You can use the following values for real-time score test.

   Affinity: Womens Sportwear
   Annual Spend: 450
   Loyalty Program Member:NO
   Home Closeout:F
   Johnston and Murphy:F
   Lancome Gift with Purchase:F
   Shoe and Handbag sale:F
   Save 20% Career Suits:T
   Anastasia Beverly Hills:F
   Discount of Womens Active Shoes:F
   Save 10% with store pickup:T
   Womens Sweater Sale:T
   Free shipping on orders over $49:F
   Gender: F
   Dress sale - 30 percent off:F

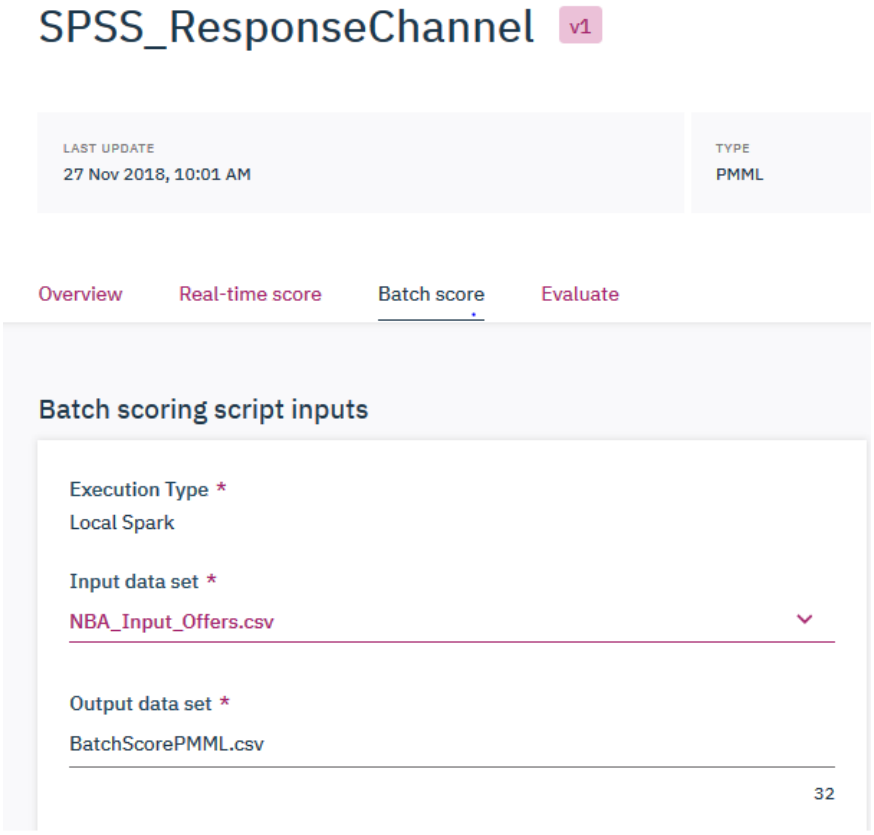


The output will look similar to the following screenshot. In this output we can see that the prediction is to use the *mobile* channel for marketing.



Batch Scoring Test

When configuring batch scoring, use *NBA_Input_Offers.csv* as the input file.



PMML models can be deployed in production just like other models in WSL. We will walk through deployment in the next section.

You have finished testing the PMML model.

Optional: Test Model Groups

Model groups provide the capability to compare the performance of several models. You can also deploy a model group as a single Web service, which we will do in the second part of the lab.

You can learn more about Model Groups in WS documentation: https://content-dsxlocal.mybluemix.net/docs/content/SSAS34_current/local/modelgroups.html

1. Navigate to the *WSL_Demos* project and switch to the **Assets** tab.
2. Scroll to **Models** and click **add model**. Select **From File**. Provide *model name* and select model type *PMML*.

3. Click **Browse**. Navigate to the *PMML/PMML files* folder of the unzipped GitHub repository and select *MortgageDefault_C5.xml*. Click **Create**.

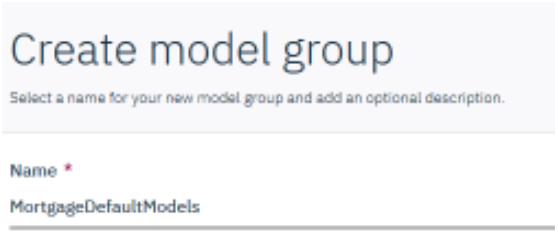
4. Repeat the same steps to import the second PMML model - *MortgageDefault_CRT.xml*

Note: these PMML files were generated from models created in SPSS Modeler. They predict the risk of mortgage default.

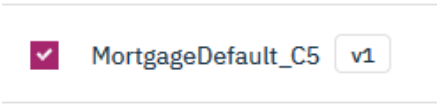
5. Navigate to the *WSL_Demos* project and switch to the **Assets** tab.

6. Select **Model Groups**, then click **Add Model Group**.

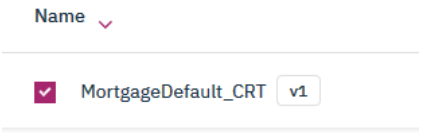
7. Provide group name, for example, *MortgageDefaultModels*, and click **Next**.



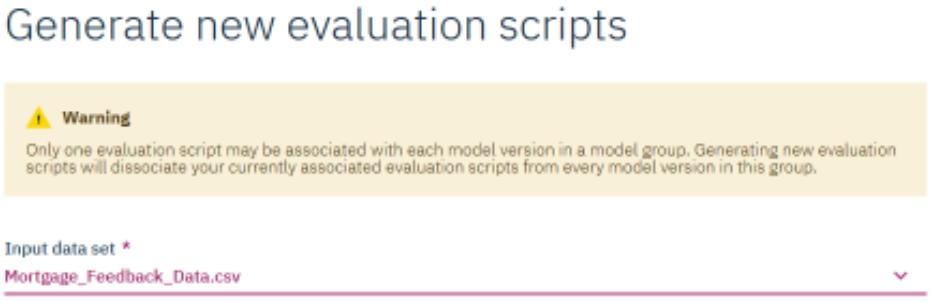
8. Select the *MortgageDefault_C5* model as the leader and click **Next**.



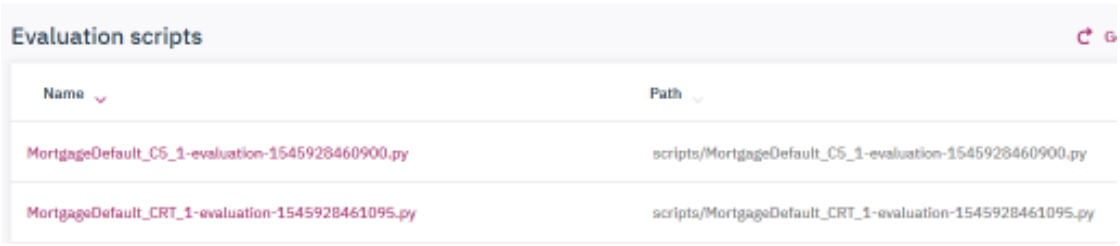
9. Select the *MortgageDefault_CRT* model. Click Next, then **Create**.



10. Click **Generate Evaluation Script**. Select *Mortgage_Feedback_Data*.csv as the *Input data set*.

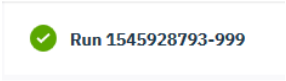


11. The generated scripts are shown in **Model group** details.



To run the evaluation jobs, click on the script, then in the script editor click the **Run** icon.

The run is finished when the run icon has changed to a checkbox.



Note: this is a warning, not an error.

```
14 ERROR StatusLogger No log4j2 configuration
```

12. After the jobs are done, navigate to the Model Group details, and review the evaluation results.

In our example the *weighted precision* is different for the two models. If you would like to see it on the graph, make sure to select the weighted precision in the dropdown.

To see several evaluation, run the same batch job several times. If you would like to see different accuracy – you will need to modify the data used for model evaluation .

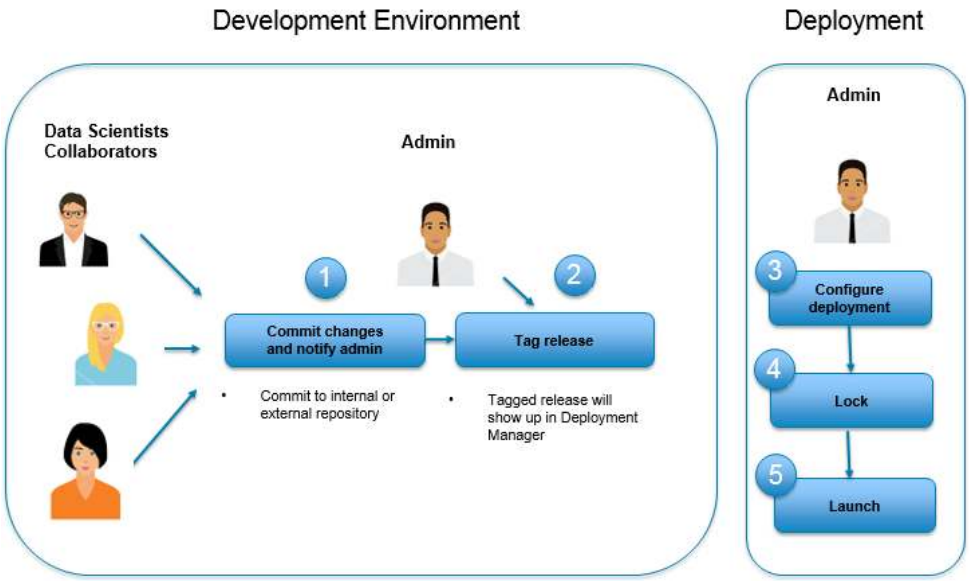
You have finished testing the model groups.

Part 2: Deploy project to production

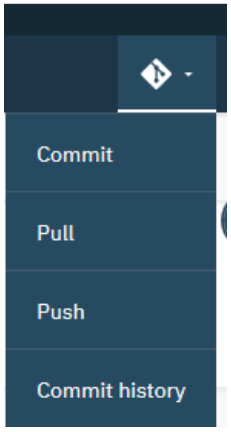
Deploy models and notebooks

Deployment in WSL can be done by any user who has admin role for both WSL and the project. Many companies have a designated person who's responsible for deployment of analytics.

As a reminder, we will be following the following steps to deploy analytic assets to production.



1. Data scientists have to commit changes to the project. You can commit assets by clicking on the **Git actions** icon in the top right corner. Select **Commit,** then **Push**.

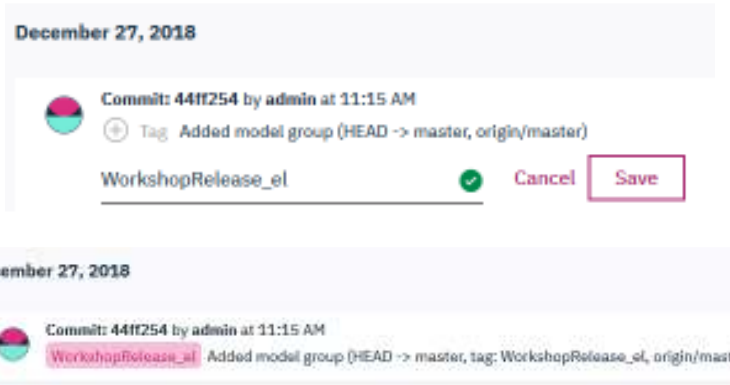


2. Provide the *Commit message* about the changes. At this time don't specify the tag.

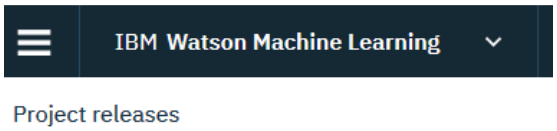
3. Now we are assuming that a different person, a *deployment admin*, is taking over deployment. Click on the **Git** icon and select **Commit History**. Notice that we can add a *tag* to the project.

   A *tag* is used to identify a specific version of the project. There may be many versions of the assets in the project, but only specific versions should be used in production.

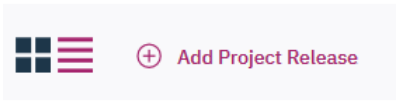
4. Provide a tag, for example, *WorkshopRelease_<your initials)*, and click **Save**.



5. Navigate to the **WML** view by selecting it from the main menu.

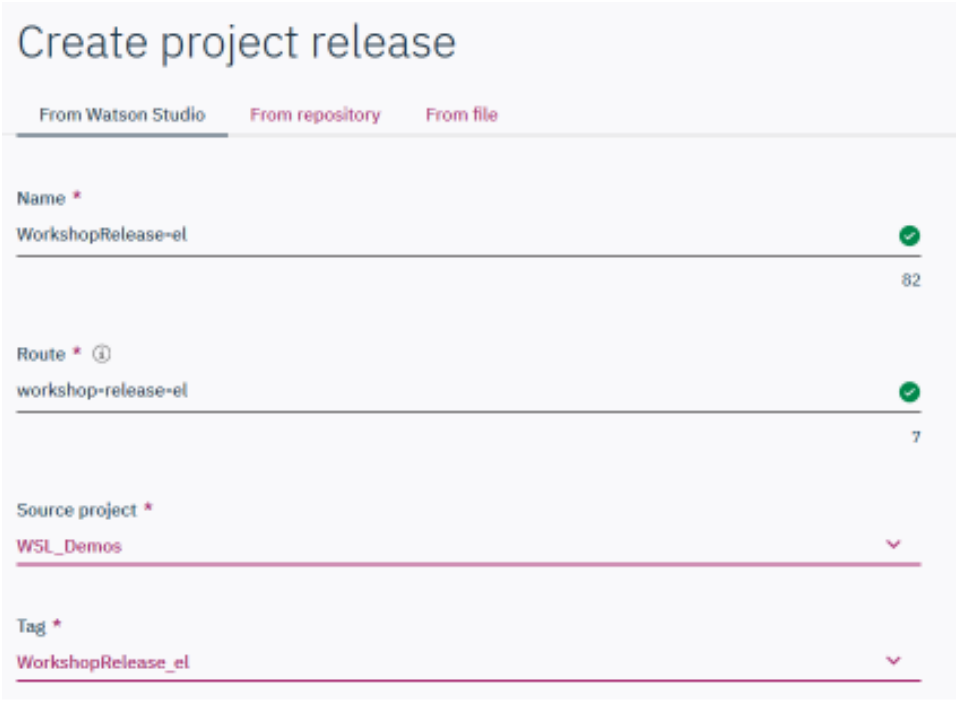


6. Click **Add Project Release**.



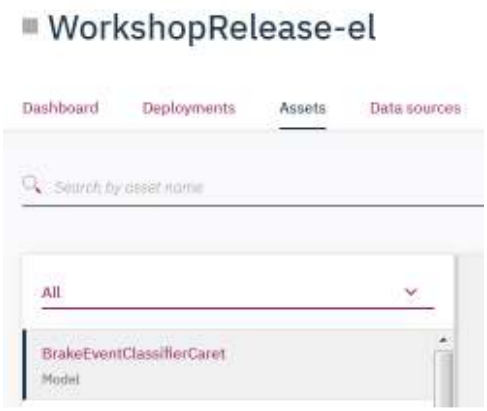
7. Enter the required fields

- **Name**: *WorkshopRelease1*
- **Route**: *release1* (this string will be used in all URLs that are generated by deployment, that's why it can't have spaces, upper case characters, and special characters). *Important: if you're sharing a cluster, make the route unique, for example, add initials to release1.*
- **Source project:** the project that you want to deploy
- **Tag**: tag that you specified in the earlier steps.

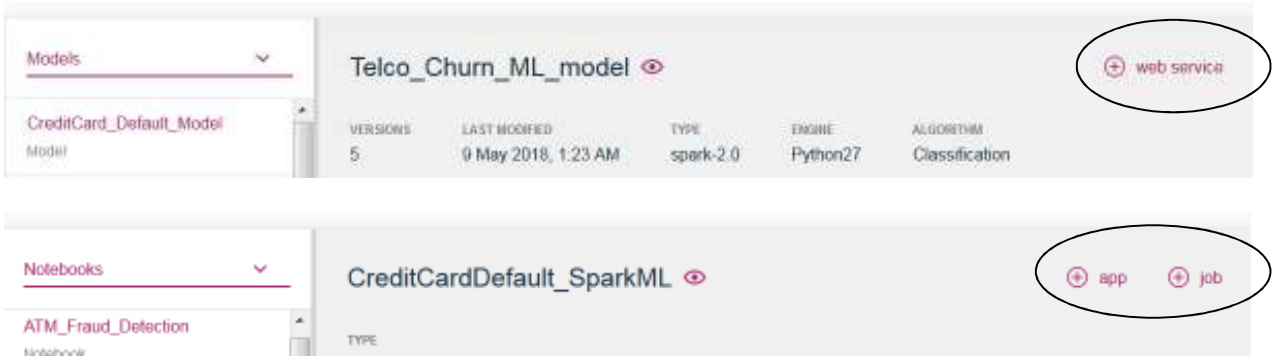


Click **Create**. This will take a few minutes because WSL is making a copy of all assets in the project.

8. The default view shows all assets that are a part of the project. Notice that you can filter them by type if you select the drop down.



9. Select different asset types (models, notebooks, scripts, etc.), and notice how deployment options (icons in the right corner) change.



WSL automatically determines applicable deployment type for each asset.

- **Models** can be deployed as Web services for **real time scoring** (invoked with a REST API). To deploy a model for batch scoring, the "script" deployment option is used. The script is generated on the development side, as we have done in *Part 1* of the lab.

- **Scripts** can be deployed for **real time** or **batch execution**. WSL automatically determines if the script can be used for online scoring (a script generated by WSL or a script that follows the specific format for online scoring). WSL supports deployment of any R or Python script. Documentation explains the required script format for online scoring: https://content-WSLlocal.mybluemix.net/docs/content/local-dev/WSLl-scripts-as-web-services.html

- **Model evaluation** is a type of **batch script deployment**.

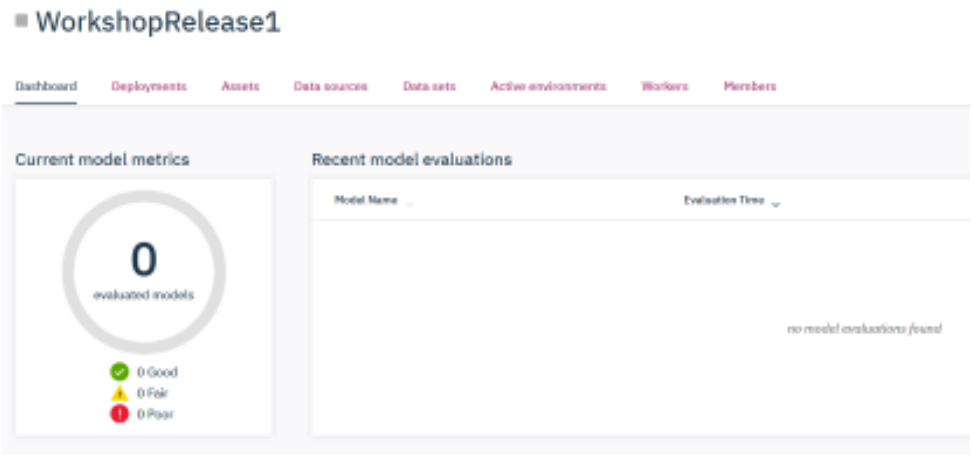
- **Notebooks** can be deployed for **batch execution** or as **an application**. Batch execution of notebooks can be used to perform many functions: scoring, model refresh, data preparation, etc. Publishing a notebook as "an application" makes it available as an HTML page that can be accessed with specified level of security.

- **Shiny applications** can be deployed as **an application**. Publishing Shiny as "an application" makes it available as an HTML page that can be accessed with specified level of security.

- **SPSS flows** can be deployed for batch execution.

10. Before we configure deployment, let's review properties of the *release*.

- On the **Dashboard** tab you will see results of evaluations. Since we haven't run any evaluation jobs, at this time we have no evaluation results.



- The **Deployments** tab is empty at this time because we haven't configured any deployments yet.

- The **Assets** and **Data Sources** tabs show the same assets and data sources as the development side.

- **Active environments** view is also empty because we haven't configured and launched the project yet.

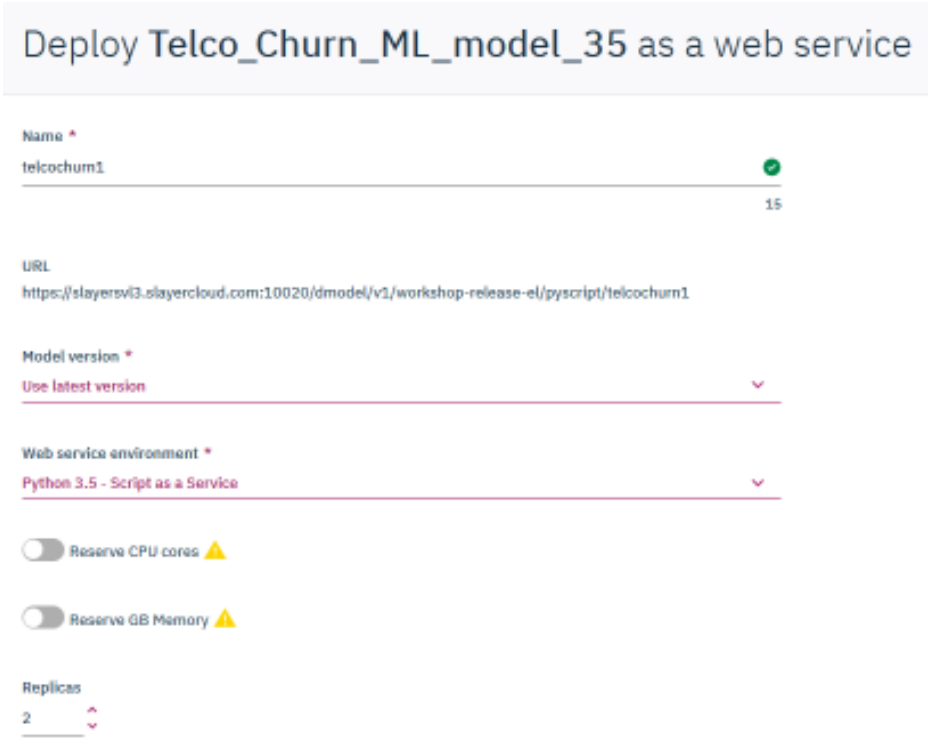
- **Workers** is another name for "environments". Here we can modify hardware configuration for each environment similar to the way we modify it in the development environment.

- **Members** are WSL users who can update the release.

11. We will start with configuring online deployment. On the **Assets** tab, select **Models** in the dropdown, then click on *Telco_Churn_ML_35* model.

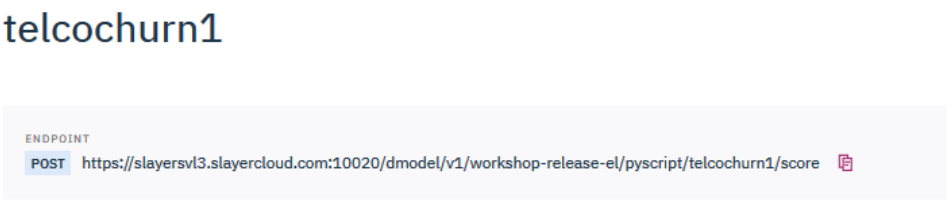
12. Click the **web service** button. Fill out the required fields.

- **Name**: *telcochurn1*. Notice that the name gets appended to the URL (REST endpoint), that's why it has to be lowercase with no special characters.
- **Model version**: use the latest version
- **Web service environment**: should be the same as the environment in which model was built (selected by default)
- **Reserve resources**: checking this option will provide dedicated CPUs and memory to online deployment
- **Replicas**: number of environments that host the service - select 2 for high availability.



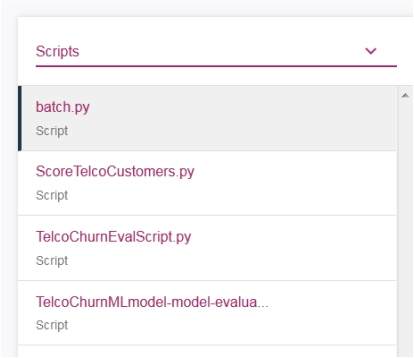
Click **Create**.

The **REST endpoint** is displayed in the model details. This endpoint won't be live until we launch the project, which we will do after we configure all other deployments.



13. Next, we will configure batch scoring and model evaluation.

    On the **Assets** tab select the **Scripts** from the dropdown. The two scripts that we created in the development environment are *ScoreTelcoCustomer.py* (used for batch scoring) and *TelcoChurnEvalScript.py* (used for model evaluation).



14. Select *ScoreTelcoCustomers.py* and click on **+job** button. Fill out the required fields.

    - **Name**: *telcobatch1*. The name gets added to the REST endoint. Batch jobs can be invoked with a REST API. Unlike online scoring, data is not passed in – the data sources that are defined in the script are used.

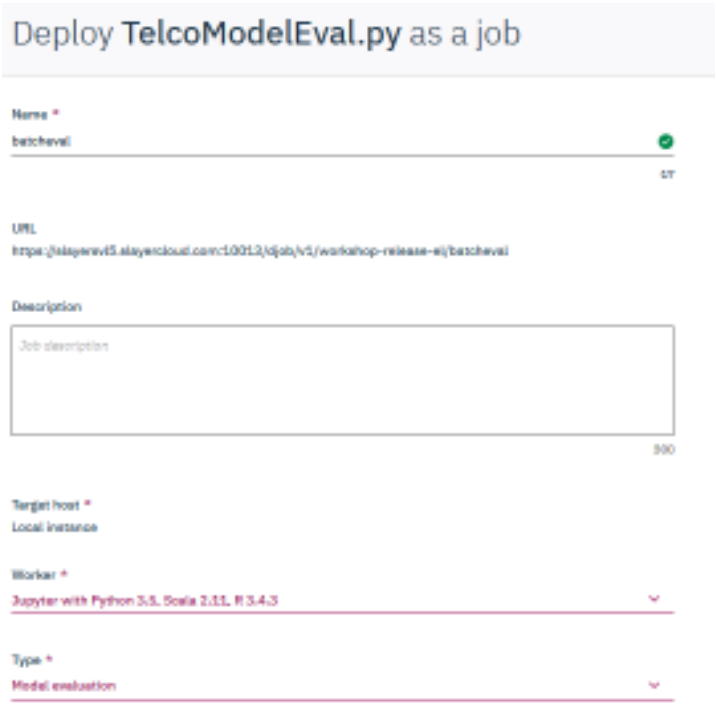
    - **Type**: batch scoring

    - **Worker**: *Jupyter with Python 3.5, Scala 2.11, R 3.4.3*. The worker should be the same runtime environment as was used for running the script in development.

      Scroll down and review the rest of the fields to review them (default values can be used). Click **Create**.

If you specified a schedule for invoking the job, it will take effect when the project is launched. You will also be able to invoke it "on demand" by selecting it in the **Deployments** view. We will complete this step later in the lab.

15. Repeat the same steps to create a model evaluation job using the evaluation script we created in *Part 1*. The only difference is the type – *Model Evaluation*.



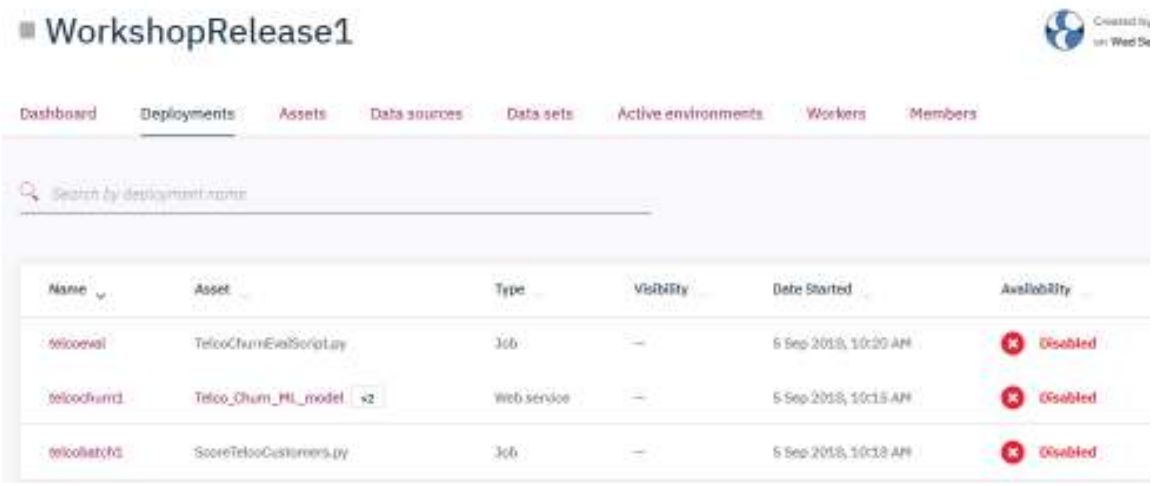
We don't have an option to modify data source (in our example .csv files) when we deploy assets for batch execution and evaluation. If we used a remote data source (for example, a database table or a file in HDFS), then we would be able to modify input/output by modifying the data source definition in the *project release*.

16. In this step we will deploy a notebook as an application.

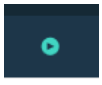
    In the **Assets** tab select **Notebooks**. Click on any notebook and click **+app**. Provide name and select the desired security setting.

The URL for the notebook will be "live" (accessible) after we launch the release.

If you click on the **Deployments** tab now, you'll see that every deployment is in *Disabled* status because we haven't launched the project.

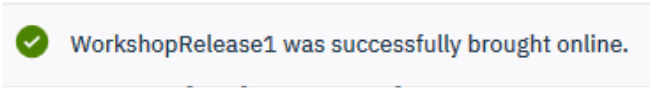


17. Now that we created a few deployments, we can lauch the release by clicking the **Launch** icon in the menu bar.



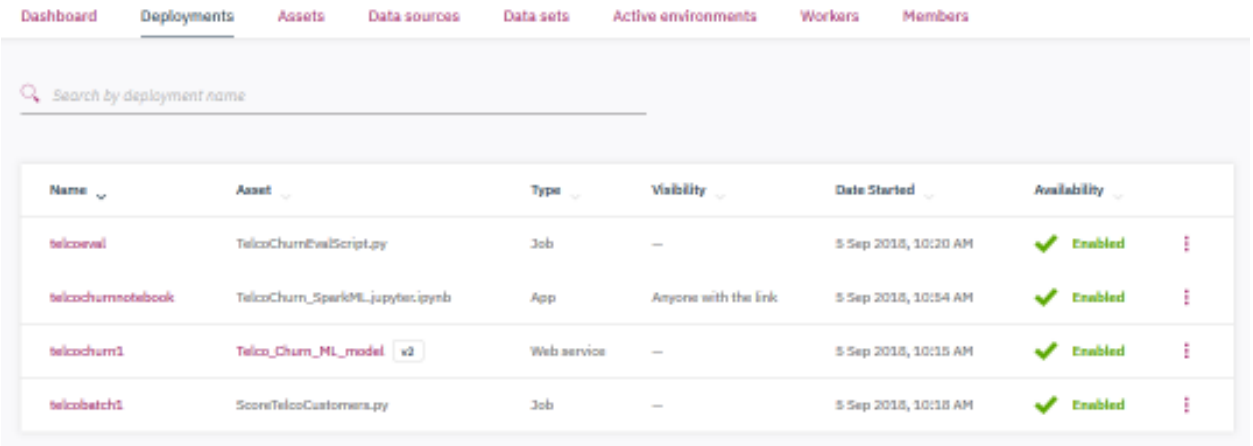
Launching the release will:

- Start all environments that will be used for deployment
- Enable the REST endpoints
- Enable URLs for "applications" (notebooks and Shiny)
- Enable schedules (if they are configured)
- Enable on-demand invocation of jobs.



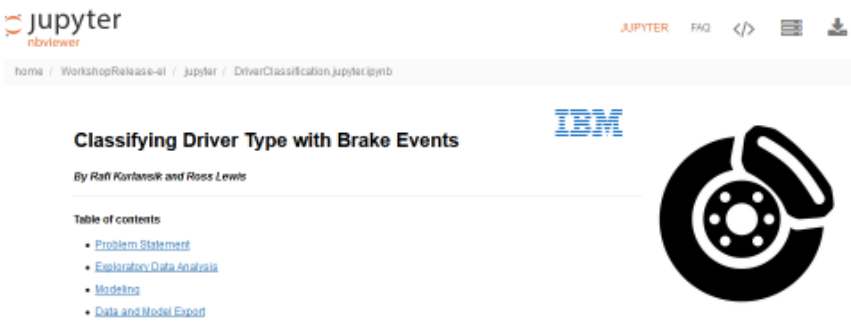
Click on **Deployments** tab. All deployments have been enabled.

Note: the Web service deployment becomes available in 1-2 minutes. Refresh the page after 2 min if the status is not Enabled.

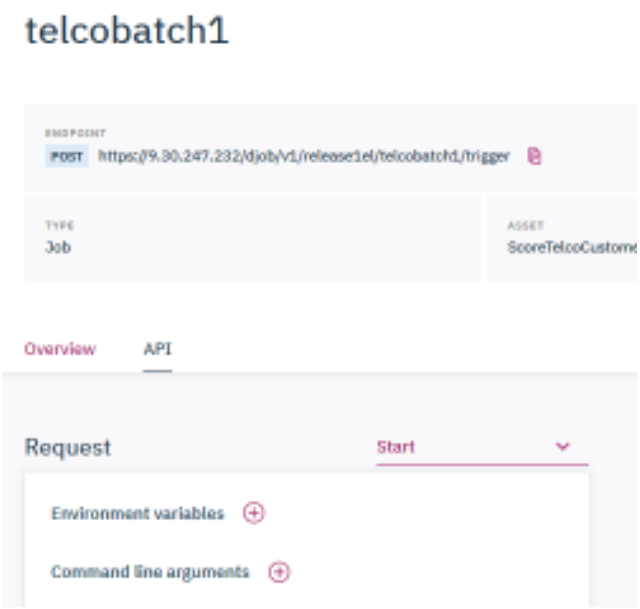


18. Now that all URLs and endpoints are active, we can test them.

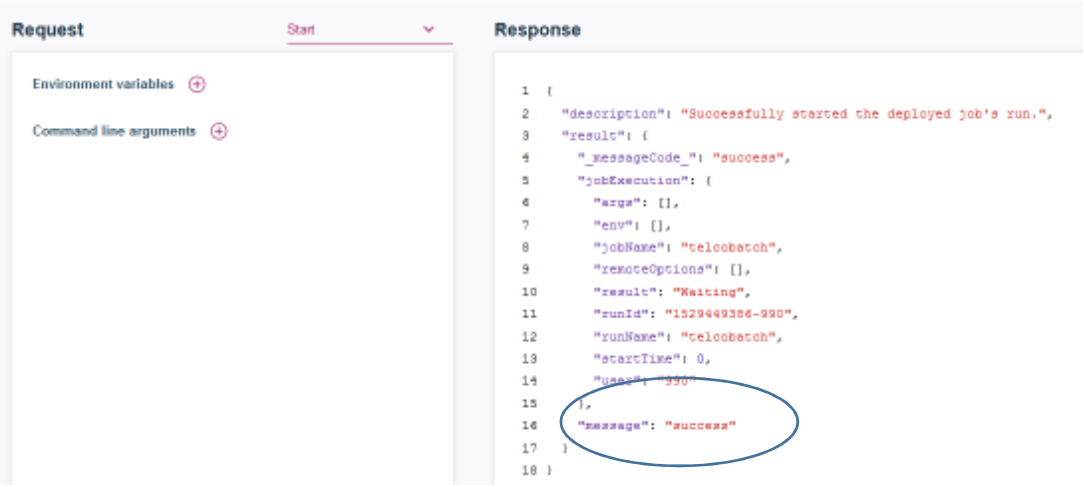
- To test the Notebook URL, click on the ellipses next to notebook deployment, select **Share Endpoint**, and copy and paste it to a new browser window.



- To test the batch job, click on the deployment (row in the table), which will bring you to deployment details. Click on the **API** tab.



- Click **Start** from the dropdown menu, then click **Submit**. This issues the REST request to invoke the batch job. Verify that it was successul.



If you click on the **Overview** tab of the deployment details, you will see the job runs that were invoked during testing. *Hint: you may need to refresh the page.*

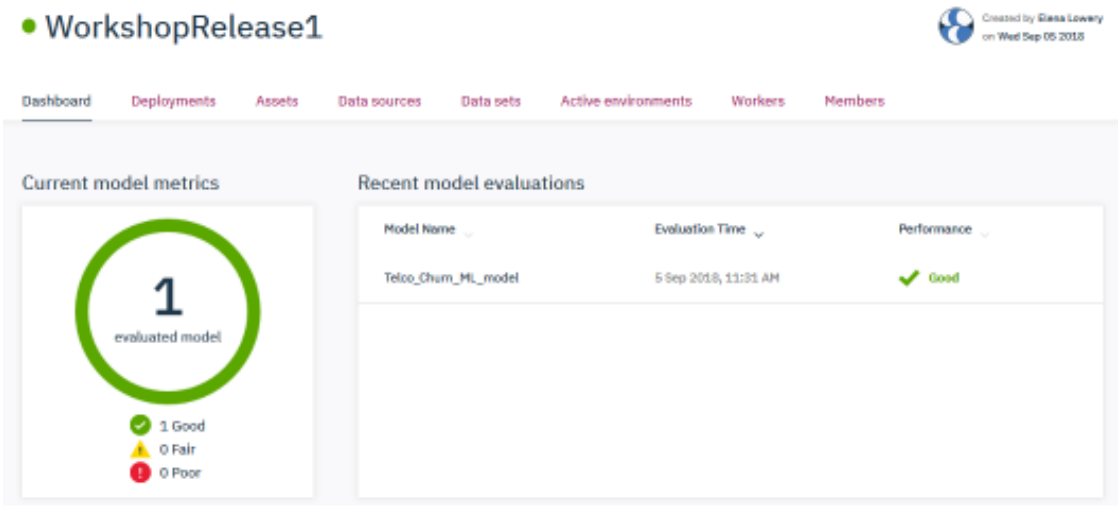
Optionally, you can click the **Generate Code** button, which shows the curl command for invoking the REST endpoint, and run it from a ssh session.

```
sh-4.2# curl -k -X POST \
>   https://169.55.181.211/djob/v1/release1/telcobatch/trigger \
>   -H 'Authorization: Bearer eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZ
SI6ImFkbWluIiwicGFja2FnZU5hbWUiOiJXb3Jrc2hvcFJlbGVhc2UxIiwicGFja2FnZVJvdXRlIjoic
mVsZWFzZTEiLCJpYXQiOjE1Mjk0MTc3Mzl9.T_jWzJ1tcw6MNbn52fZ-bCotCiWJ5MIyskSWznncJw3u
B8nCWePgJS4DzomEfkBULjiFBUNnIrEw93-HK-lKBchRxgSF5TRQVPQid-GJ6F3KFU7U0d6_nFN5qcZr
48qw2D1DQfyik0nkdtFYsIsfo0na-rD1iFIFylU3IzWW5x-8J8rflfX9Zj47KKW_vZs9xYWJEptfaqwz
eNkAU4vCbDkFIz8p6FkuD3qSrxIzWbsUXbJ4pOMRyP9jF55KMZO9eaUbYqWSE-FUPHBD4y1qinr2aa-c
DCbLln1x7CXtnrSboEtOxk-RCTbjs9praWCfy-qplkiKJ8Wre7Es_8zYaA' \
>   -H 'Cache-Control: no-cache' \
>   -H 'Content-Type: application/json' \
>   -d ' {"env":[],"args":[]} '
{"description":"Successfully started the deployed job's run.","result":{"_messag
eCode_":"success","jobExecution":{"args":[],"env":[],"jobName":"telcobatch","rem
oteOptions":[],"result":"Waiting","runId":"1529449672-990","runName":"telcobatch
","startTime":0,"user":"990"},"message":"success"}}sh-4.2#
```

- Next, we will test evaluation. Click on the **Dashboard** tab and review evaluation details (for example, timestamps for recent evaluations). This list will change after we run evaluation.

  Switch to **Deployments** tab and click on the evaluation batch job. Similar to batch job testing, switch to the **API** tab, select **Start** from the dropdown, then select **Submit**.

  Navigate back to the **Dashboard** tab. Now evaluation results are displayed.



- In this section we will test online scoring. Click on Web service deployment and switch to the API tab.

  Click **Submit**. Response is dispalyed.

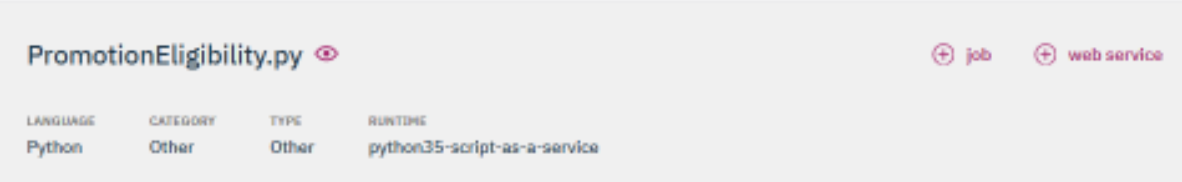
Optionally, you can click the **Generate Code** button, which shows the curl command for invoking the REST endpoint, and run it from a shh session.

If you navigate back to deployment view, you'll notice that invocation metrics have changed.

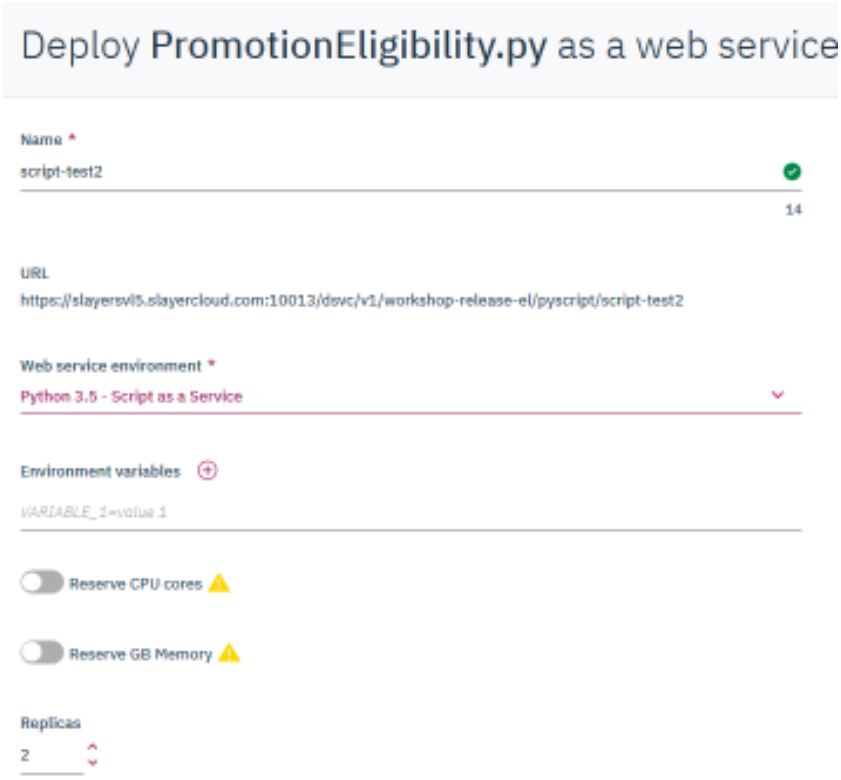
You have finished configuring and tesitng deployments.

Optional: Deploy script as a Web service

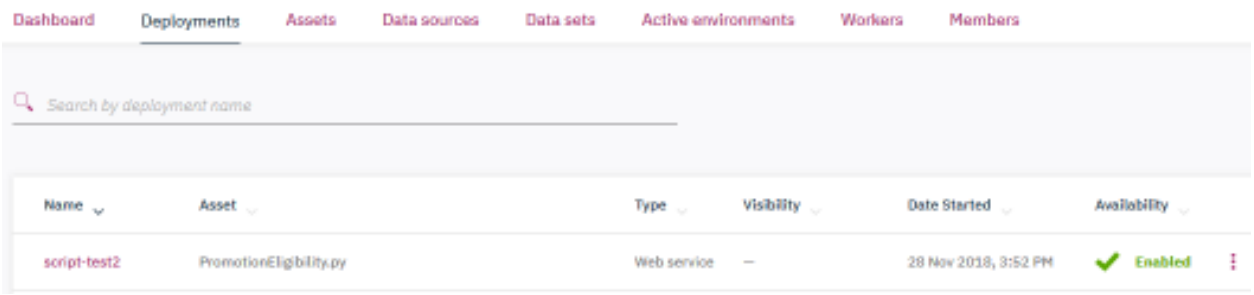
1. In the **Assets** view, find the *PromotionEligibility* script and click **web service**.



2. Provide the required information (make sure to select *Python 3.5 environment*) and click **Create**.



3. Enable the deployment on the **Deployments** tab and wait till status is changed to **Enabled**.



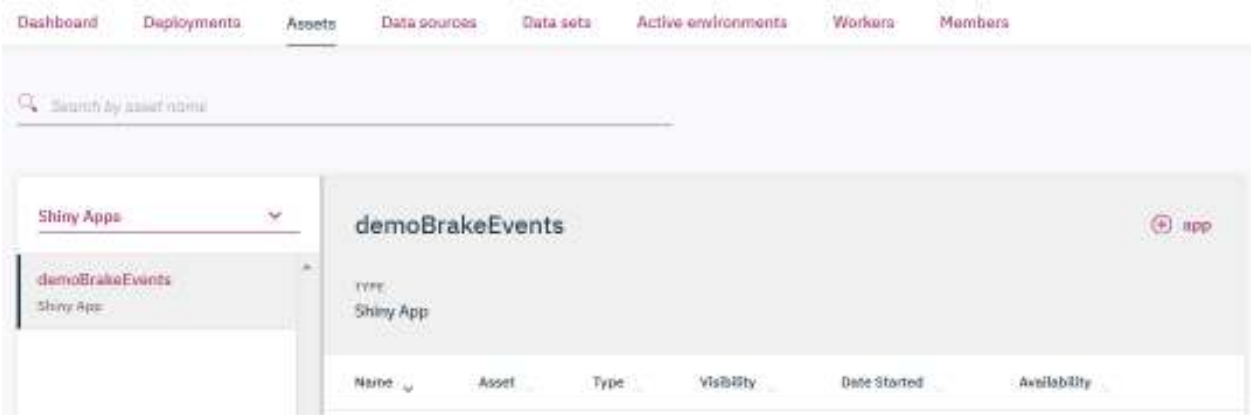
4. When testing on the API screen, use the following values:

   - **Function name**: *determineEligibility*
   - **Body:** *{"customerStatus": "N", "purchaseAmount": 55, "customerSegment": "Family"}*

Optional: Deploy a Shiny application

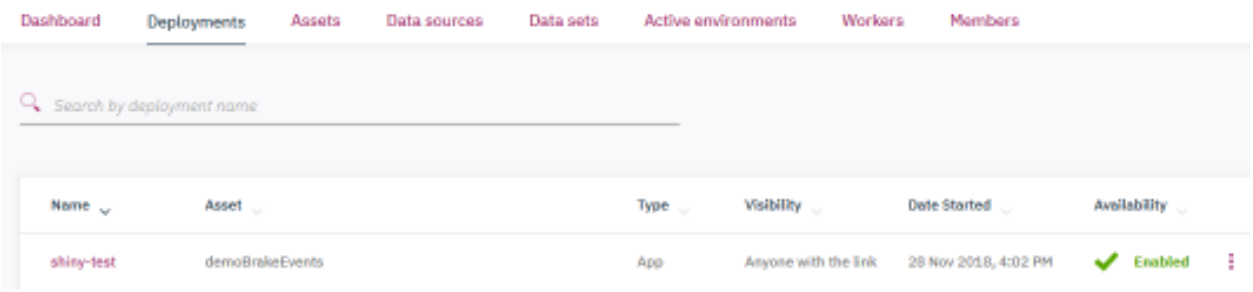
The Shiny application is included in the *WSL_Demos* project, and that's why it's available in the release.

1. Select **Shiny** from the **Assets** dropdown and click **app**.

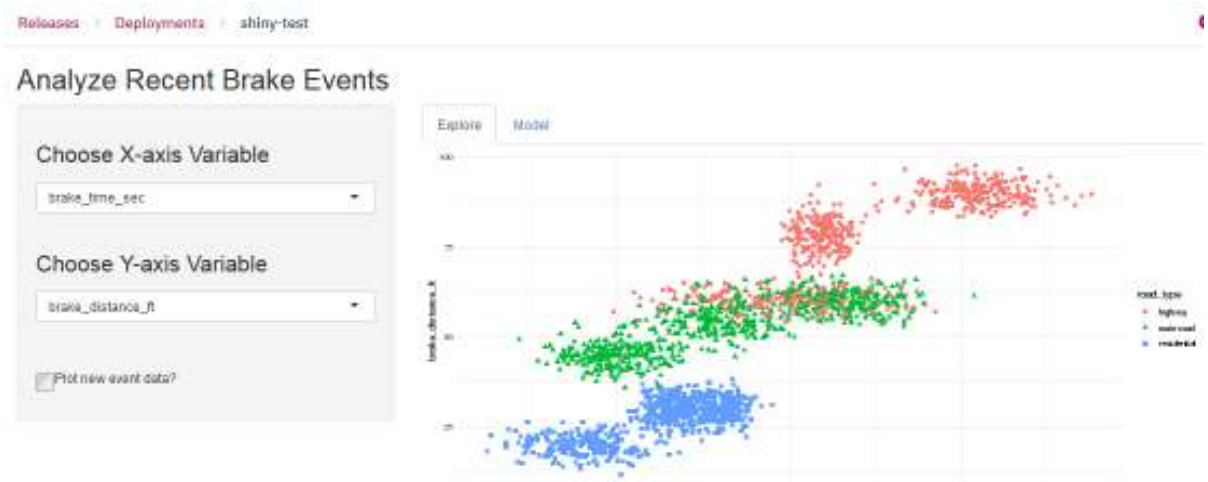


5. Provide the required information and click **Create**.

6. Enable the deployment on the **Deployments** tab and wait till status is changed to **Enabled**.

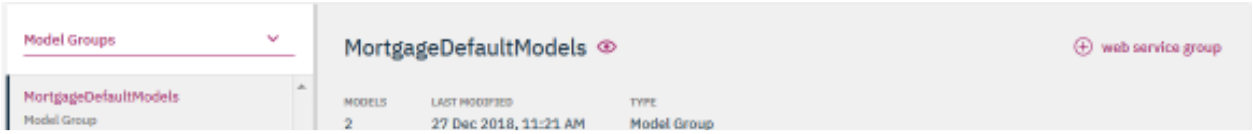


7. Click on deployment to test it. The application should look similar to this:



Optional: Create a Model Group deployment

1. Select **Model Groups** from the **Assets** dropdown and click **web service group**.



2. Provide deployment name and make sure to select the *All models* checkbox. Click **Create**.

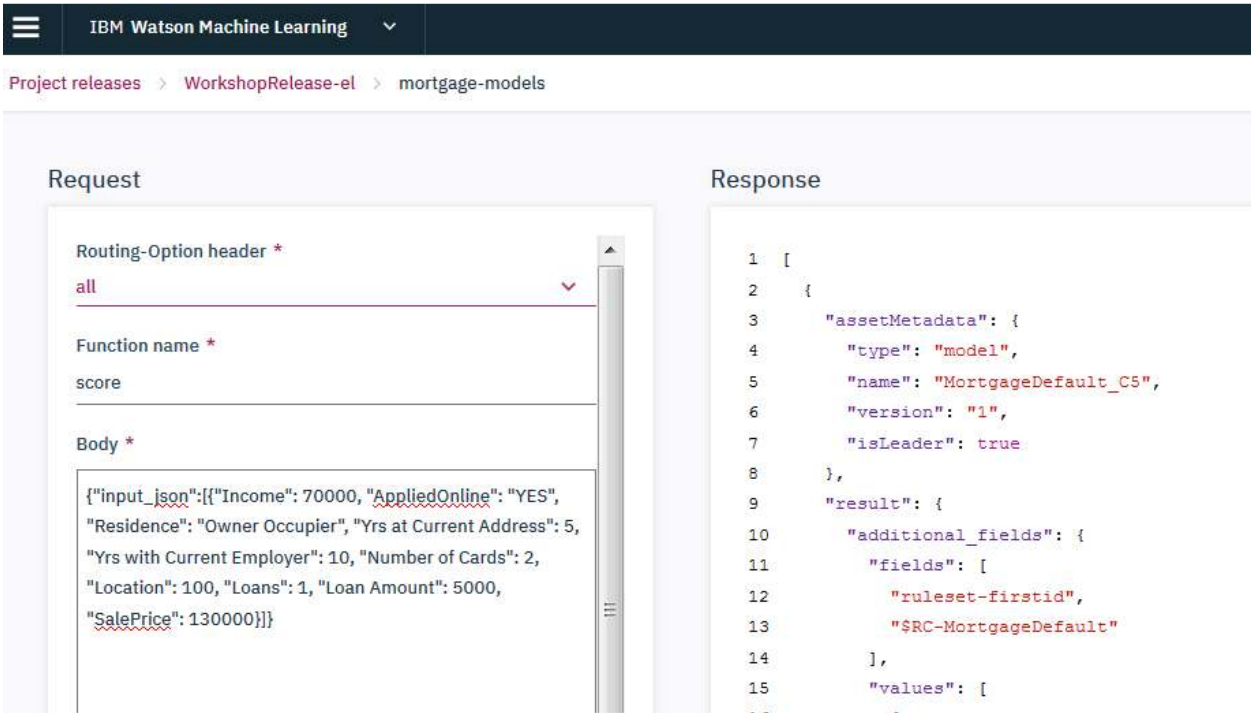
   See documentation (videos) for explanation of configuration options: https://content-dsxlocal.mybluemix.net/docs/content/SSAS34_current/local/modelgroups.html

3. Enable the created deployments. Make sure both models and model groups are in *enabled* status before testing

   Click on *mortgage-models* to bring up the details screen.

4. When testing, make sure to specify *score* as the funtion name. You can use this input in the *body* field.

{"input_json":[{"Income": 70000, "AppliedOnline": "YES", "Residence": "Owner Occupier", "Yrs at Current Address": 5, "Yrs with Current Employer": 10, "Number of Cards": 2, "Location": 100, "Loans": 1, "Loan Amount": 5000, "SalePrice": 130000}]}



Summary

You have finished the **Deployment in WSL** lab.

In this lab you completed the following steps:

- Tested assets in development environment (a data scientist task)
- Created deployment definitions (an admin task)
- Launched and tested deployment (an admin task).