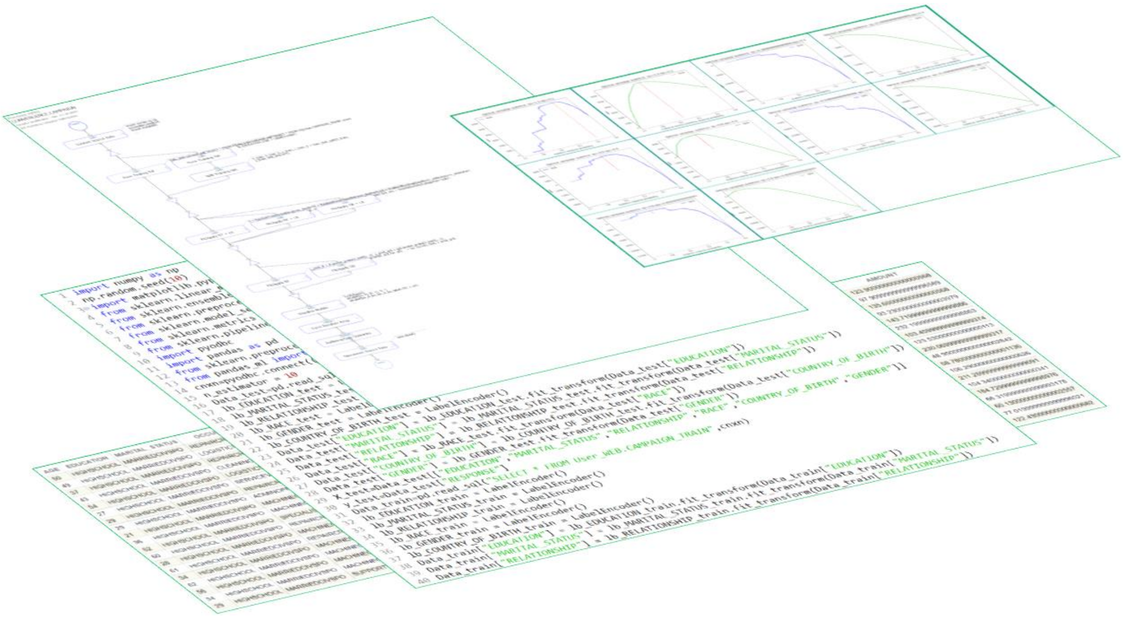
TRAINING

Machine Learning (ML) Toolkit

ML Toolkit Fundamentals



InterSystems
Health | Business | Government

Document details

Title:	Machine Learning (ML) Toolkit
Customer:	None
Project:	None
Module(s):	None
Training name:	ML Toolkit Fundamentals
Description:	Information required for installation and basic use of ML Toolkit. The functionality of ML Toolkit is a prototype to be adjusted to the needs of the user. The user installs and applies ML Toolkit at their own discretion and risk.
Document ID:	ML_Toolkit_Fundamentals
Version:	01_2
Created by:	Sergey Lukyanchikov, Eduard Lebedyuk

Document controls

Document Modifications			
Version	Date	Description of Change	Modified By
01	2019-02-19	Initial version	Sergey Lukyanchikov Eduard Lebedyuk

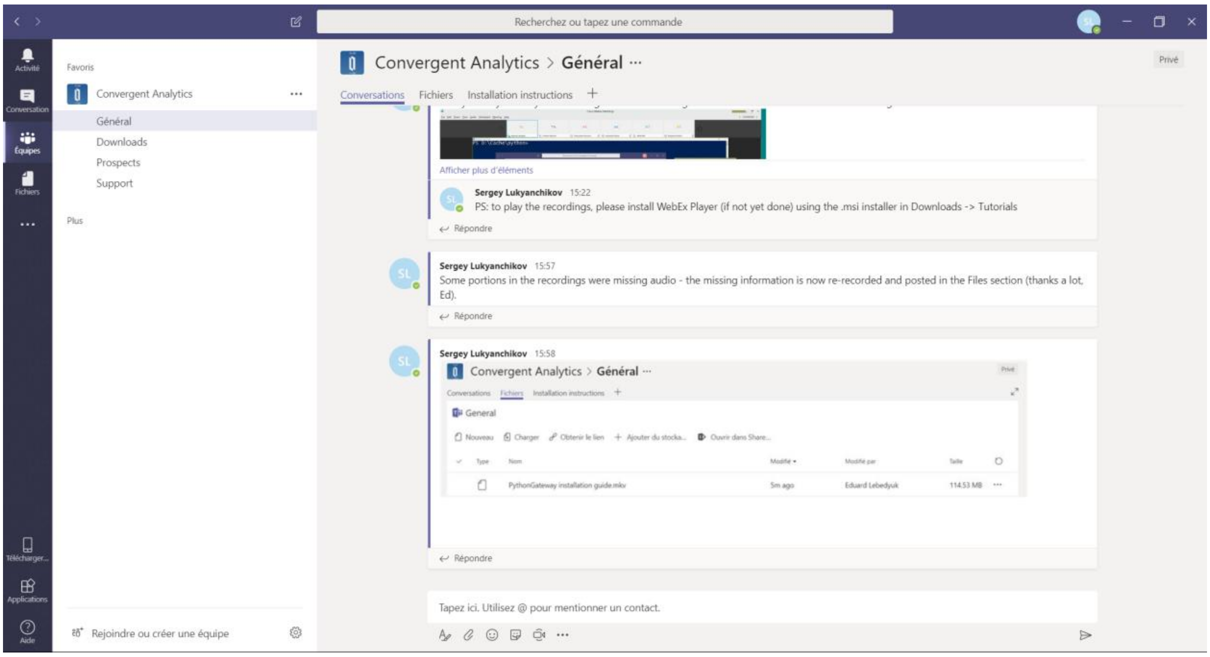
Document Authorization		
InterSystems	**Name**	Sergey Lukyanchikov, Eduard Lebedyuk
	Role	Sales Engineer
	Signature	< Insert electronic signature >
	Date	2019-02-19
None	**Name**	< Insert customer contact >
	Role	< e.g. Project Manager >
	Signature	< Insert electronic signature >
	Date	< Select date >

Contents

1. General

1.1. Internal Users: MS Teams Group

The starting point for accessing all the internal ML Toolkit resources is the MS Teams group Convergent Analytics:



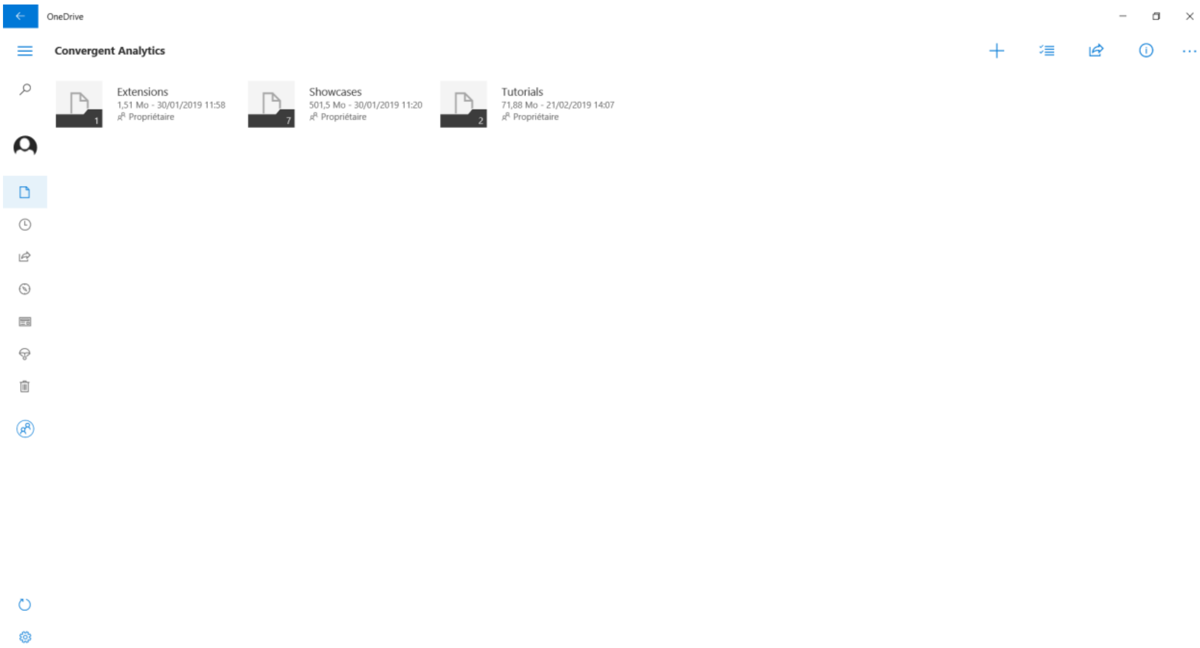
To apply for membership, please use [this link].

Once you are in the group, you have access to the following communication channels:

- **General** – receive updates about ML Toolkit feature development and availability, download slide decks and screenshots, learn about webinars and events, dialog with the group
- **Downloads** – a shortcut to the copy of the internal OneDrive folder with MS Toolkit extensions, showcases and tutorials
- **Prospects** – post updates on your opportunities that involve ML Toolkit, read updates from our colleagues
- **Support** – post questions and problems, receive answers and support

1.2. Internal Users: OneDrive Folder

Having been granted membership in the MS Teams group, you are also granted access to the OneDrive folder Convergent Analytics:



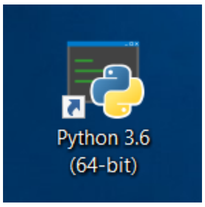
Inside the folder, the following subfolders are maintained:

- **Extensions** – contains functional extensions to IRIS functionality:
  - Python – a set of integration components required to call Python out from IRIS
- **Showcases** – contains analytical content for running via the functional extensions:
  - 001 Sentiment Analysis
  - 002 Engine Condition Classification
  - 003 Reimbursement Request Check
  - 004 Retail Cannibalization Analysis
  - 005 Marketing Campaign Optimization
  - 006 Rail Time Series Discovery
  - 007 Housing Debts Prediction
- **Tutorials** – contains educational materials focused on ML Toolkit

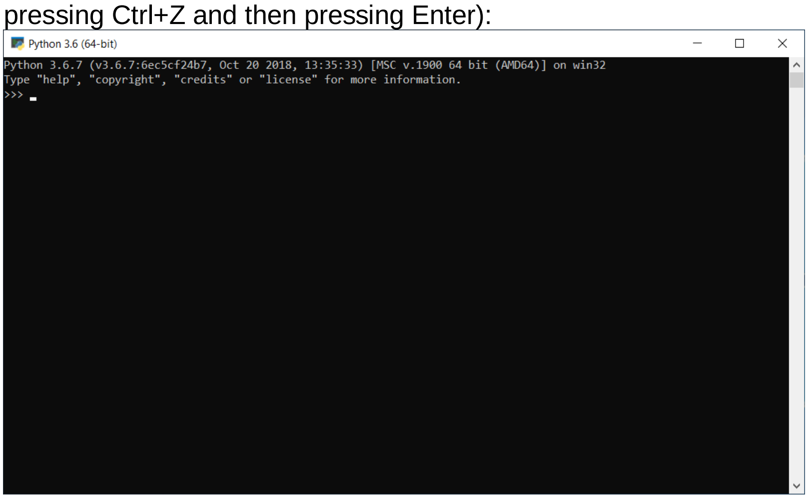
2. Python Tools

2.1. Installation

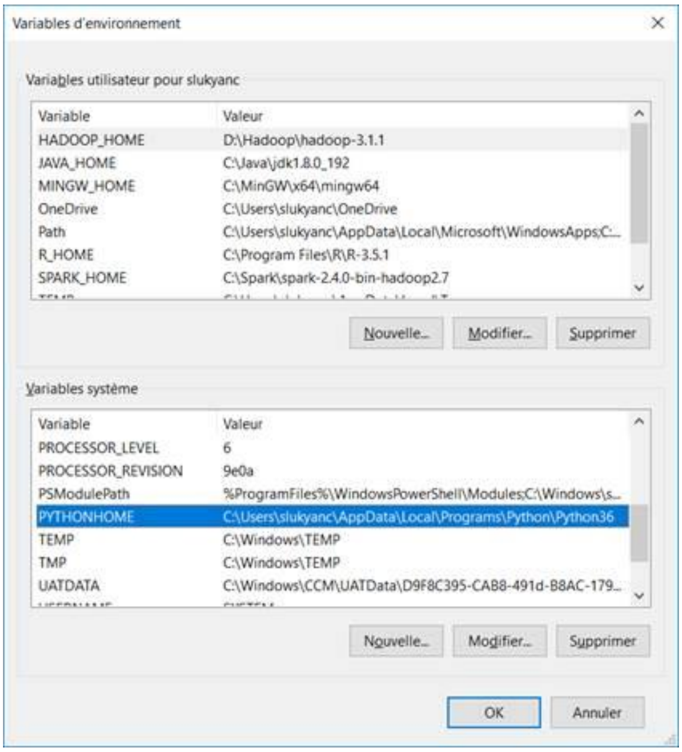
1. Install IRIS (**IMPORTANT:** make sure that the IRIS instance service runs under your Windows account and not under a system service account. This is because Python by default installs into your Windows user workspace. You may run IRIS under system service account, but you need to make sure that IRIS has access to Python executable and plugin folders).
2. Install Python and its modules, prepare the environment
   a. Install Python 3.6.7
      i. Download Python 3.6.7 64-bit, from the [download page](). Select the installer that matches your OS and bitness (for example: Windows 64-bit)
      ii. Install Python 3.6.7 into a default directory (`C:\Users\<USER>\AppData\Local\Programs\Python\Python36` on Windows)
      iii. After the installation, an icon like that appears on your desktop (a Windows-based example):



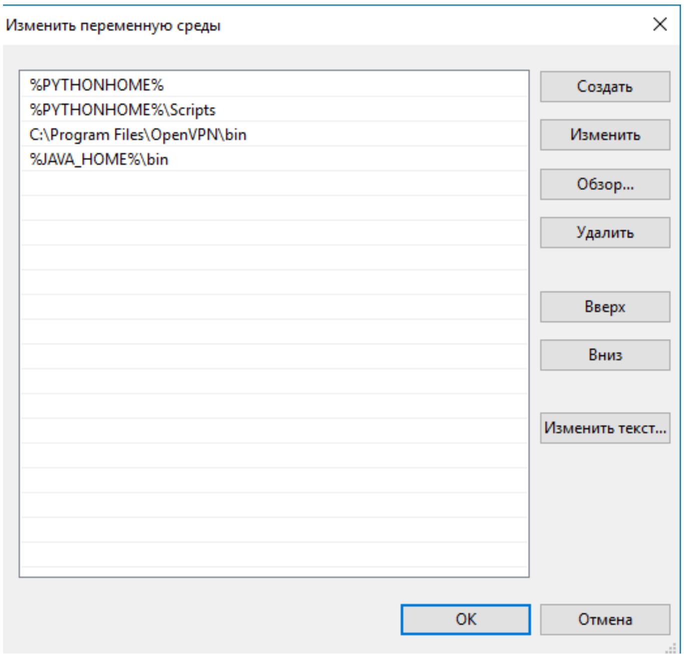
      iv. If you double-click it, the following window opens (you can leave it by pressing Ctrl+Z and then pressing Enter):



      v. Check that your `PYTHONHOME` system environment variable points to the folder where your Python was installed (for example: `C:\Users\<USER>\AppData\Local\Programs\Python\Python36`):



Also, check that your `PATH` system environment variable includes the path to Python:

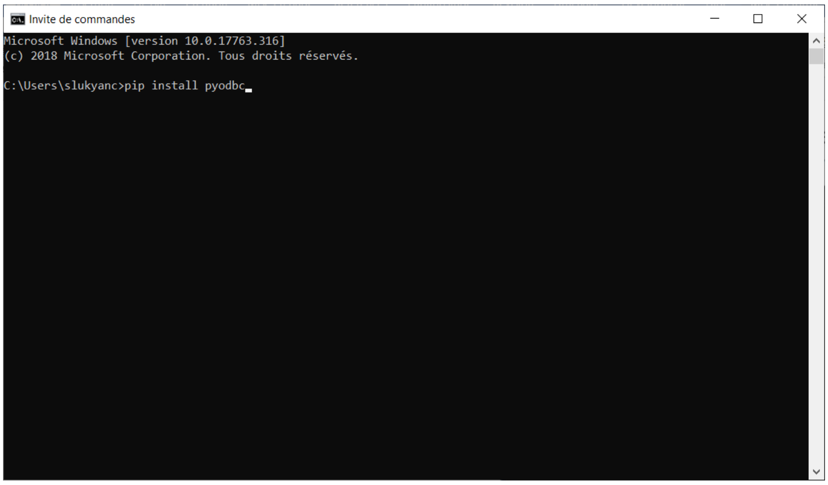


If you are on Linux or Mac, check that your `PATH` system environment variable includes `/usr/lib` and `/usr/lib/x86_64-linux-gnu`. Use `/etc/environment` file to set the environment variables.

vi. Restart your computer for the environment variable changes to take effect.

b. Install Python modules (presuming Python 3.6.7)

i. Start a command prompt window (e.g., PowerShell or CMD in Windows) and install one by one the following Python modules using `pip install <module name>` command (you need to be connected to the Internet while doing this):



- `pyodbc`
- `matplotlib`
- `numpy`
- `pandas`
- `seaborn`
- `sklearn`
- `statsmodels`
- `itertools`
- `tensorflow`
- `logging`
- `multiprocessing`
- `genism`

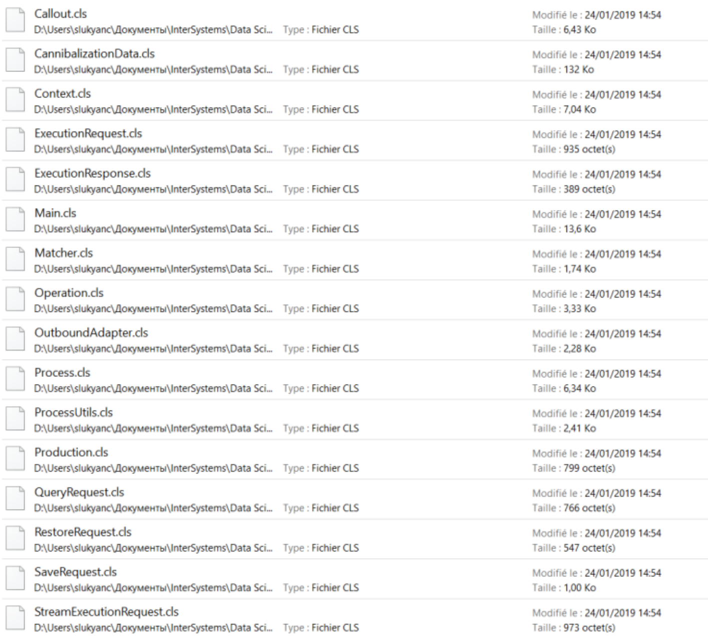
ii. The warning about `pip` version not being up to date can be ignored (we can update it any time after the modules installation).

iii. During the installation various warnings or even error messages can be thrown – we will ignore them for the time being.

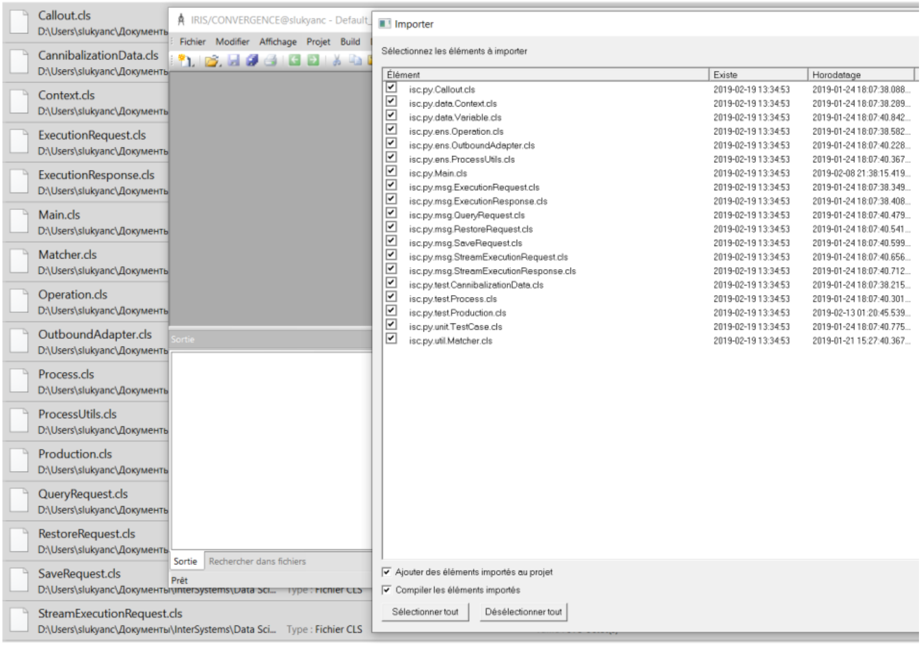
c. Configure an ODBC connection to your IRIS instance.

3. Import ObjectScript classes using IRIS Studio

a. in the folder where you keep ML Toolkit installation set, run a file search using `*.cls` mask:

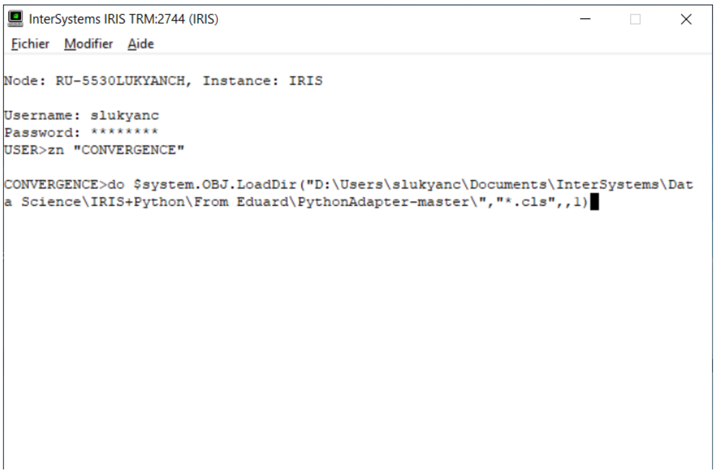


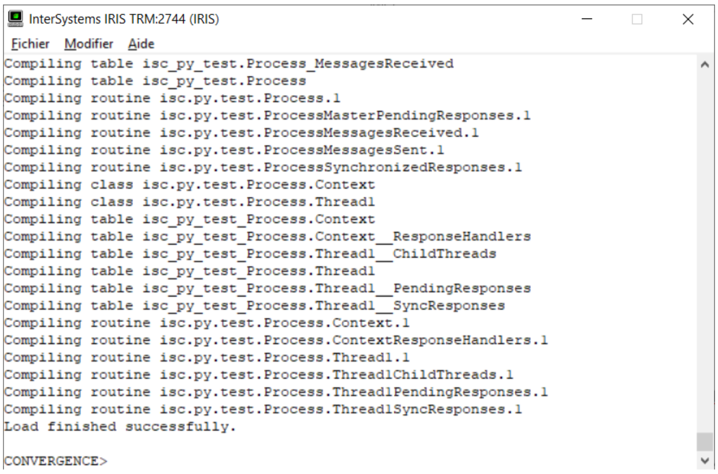
b. select the files found by the file search above, drag and drop them into your IRIS Studio:



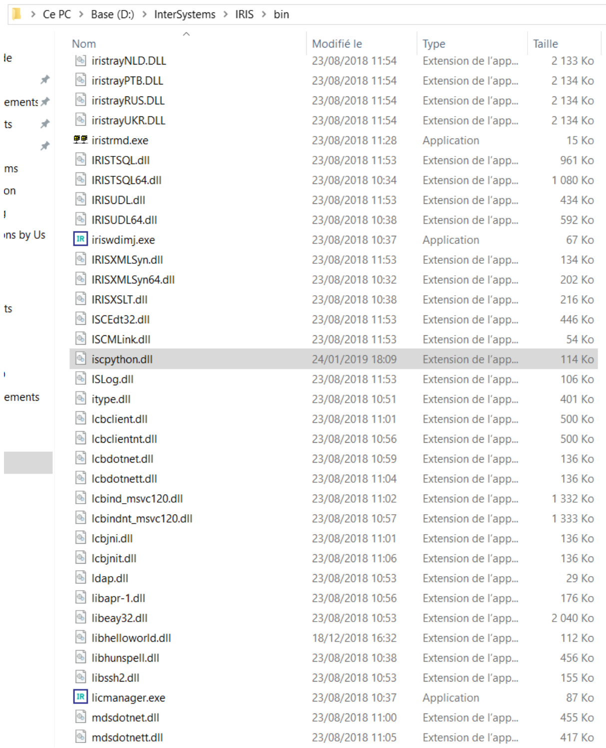
4. Import ObjectScript classes using IRIS Terminal (an alternative to IRIS Studio)

   a. in IRIS Terminal execute the following command (adjust the path to point to the folder where you placed the ML Toolkit class files): `do $system.OBJ.LoadDir("C:\path\to\toolkit","*.cls",,1)`





5. Copy the library file
    a. from the folder where you keep ML Toolkit installation set copy `iscpython.dll` (if you are on Windows), or `iscpython.so` (if you are on Linux), or `iscpython.dylib` (if you are on Mac) to the bin subfolder of your IRIS installation folder, and restart IRIS instance:

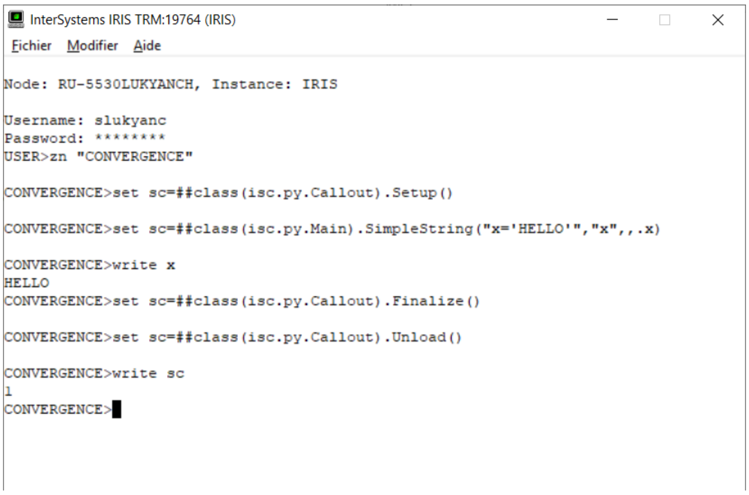


2.2. Use

2.2.1. General Use

1. Execute (once per system start) the following call: `set sc=##class(isc.py.Callout).Setup()`
2. Continue with calling the main method (can be called multiple times, the context persists): `set sc=##class(isc.py.Main).SimpleString("x='HELLO'","x",,.x)`
3. Check the call result: `write x`
4. To free Python context: `set sc=##class(isc.py.Callout).Finalize()`

5. To free the callout library: `set sc=##class(isc.py.Callout).Unload()`



6. Overall, `isc.py.Main` is the general interface to Python. It offers the following methods (all return `%Status`):
    a. `SimpleString(code,returnVariable,serialization,.result)` – for cases where the code and the variable are both strings
    b. `ExecuteCode(code,variable)` – execute a `code` (a string or a stream), optionally set the result to a `variable`
    c. `GetVariable(variable,serialization,.stream,useString)` – get a `serialization` of a `variable` in a `stream`. If `useString` is set to 1, and the `variable serialization` can be fit into a string then the string is returned instead of the `stream`
    d. `GetVariableInfo(variable,serialization,.defined,.type,.length)` – get information on a `variable`: is it `defined`, `type` and `serialization` length
    e. `GetStatus()` – returns the last occurred exception in Python and clears it
    f. `GetVariableJson(variable,.stream,useString)` – get JSON serialization of a `variable`
    g. `GetVariablePickle(variable,.stream,useString)` – get Pickle serialization of a variable
    h. `ExecuteQuery(query,variable,type)` – create a resultset (of `pandas` dataframe or list `type`) from SQL `query` and set it to a `variable`
    i. `ImportModule(module,.imported,.alias)` – import a `module` with an `alias`
    j. `GetModuleInfo(module,.imported,.alias)` – get the `module alias` and its currently `imported` status
7. Possible serialization parameters:
    a. `##class(isc.py.Callout).SerializationStr` – a serialization by `str()` function if set to 1 (the default value is 0)
    b. `##class(isc.py.Callout).SerializationRepr` – a serialization by `repr()` function if set to 1 (the default value is 1)

2.2.2. Context Persistence

Python context can be persisted into IRIS and restored later. There are the following functions:

1. Save the context: `set sc=##class(isc.py.data.Context).SaveContext(.context,maxLength,mask,verbose)` where `maxLength` is the maximum length of the saved variable. If the variable serialization is longer than that, it will be ignored. Set to 0 to get them all. The other parameters: `mask` is a comma-separated list of the variables to save (special symbols * and ? are recognized), `verbose` specifies displaying the `context` after saving and `context` is the resulting Python `context`. Get the `context` ID with `context.%Id()`
2. Display the context: `do ##class(isc.py.data.Context).DisplayContext(id)` where `id` is the ID of a stored context. Leave empty to display the current context
3. Restore the context: `do ##class(isc.py.data.Context).RestoreContext(id,verbose,clear)` where `clear` kills the currently loaded context if set to 1

A context is saved into `isc.py.data` package and can be viewed/edited by SQL and object methods.

2.2.3. IRIS Interoperability Adapter

IRIS Interoperability adapter `isc.py.ens.Operation` enables interaction with a Python process from Interoperability productions. The following requests are currently supported:

1. Execute Python code via `isc.py.msg.ExecutionRequest` and get the response via `isc.py.msg.ExecutionResponse` with requested variable values as strings
2. Execute Python code via `isc.py.msg.StreamExecutionRequest` and get the response via `isc.py.msg.StreamExecutionResponse` with requested variable values as streams
3. Transfer data into Python from an SQL query with `isc.py.msg.QueryRequest` and get the response via `Ens.Response`
4. Save a Python context with `isc.py.msg.SaveRequest` and get the response via `Ens.StringResponse` with the context ID
5. Restore a Python context with `isc.py.msg.RestoreRequest`

Check request/response classes documentation for more details.

2.2.4. Sample Business Process and Production

To use the sample business process and production:

1. In OS shell execute: `pip install pandas matplotlib seaborn`
2. Execute this code in terminal to populate the data: `do ##class(isc.py.test.CannibalizationData).Import()`
3. In the sample business process `isc.py.test.Process`, edit the annotation for the `Correlation Matrix: Graph` call specifying a valid file path in `f.savefig` function
4. Save and compile the business process
5. Start `isc.py.test.Production` production
6. Send an empty `Ens.Request` message to `isc.py.test.Process`

Notes:

- If you want to use ODBC connectivity – on Windows install `pyodbc`: `pip install pyodbc`, on Linux install `apt-get install unixodbc unixodbc-dev python-pyodbc`
- If you want to use JDBC connectivity – install `JayDeBeAPI`: `pip install JayDeBeApi`, on Linux you may need to install system packages beforehand: `apt-get install python-apt`
- If you are getting errors similar to "undefined symbol: _Py_TrueStruct", in `isc.py.ens.Operation` operation set `PythonLib` setting to `libpython3.6m.so` or even to a full path to the shared library. Check troubleshooting section for more details.
- In the sample business process `isc.py.test.Process` edit the annotations for ODBC or JDBC connection calls specifying a correct connection string
- In the sample business process `isc.py.test.Process` set `ConnectionType` setting to a preferred connection type (defaults to `RAW`, change only if you need to test xDBC connectivity)

2.2.5. Unit Tests

To run unit tests, execute:

```
set repo=##class(%SourceControl.Git.Utils).TempFolder()
set ^UnitTestRoot=##class(%File).SubDirectoryName(##class(%File).SubDirectoryName(##class(%File).SubDirectoryName(repo,"isc"),"py"),"unit",1)
set sc=##class(%UnitTest.Manager).RunTest(,"/nodelete")
```

2.2.6. ZPY Command

Install [this ZLANG routine] to add `zpy` command:

```
zpy "import    random"
zpy "x=random.random()"
zpy "x"
>0.4157151243124494
```

2.2.7. Limitations

There are several limitations:

1. Module reinitialization. Some modules may only be loaded once per process lifetime (i.e. `numpy`). While Finalization clears the context of the process, repeated loading of such libraries terminates the process. Discussions: 1, 2.
2. Variables. Do not use these variables: `zzzcolumns, zzzdata, zzzdef, zzzalias, zzzerr, zzzvar, zzztype, zzzlen, zzzjson, zzzpickle, zzzcount, zzzitem, zzzmodules, zzzvars`. Please report any leakage of `zzz*` variables. System code should always clear them.
3. Functions. Do not redefine these functions: `zzzmodulesfunc(), zzzvarsfunc(), zzzgetalias(), zzztoserializable()`
4. Context persistence. Only pickled variables can be restored correctly. User functions are currently not supported. Module imports are supported.

2.3. Development

Development of ObjectScript code is done via `cache-tort-git` in UDL mode. Development of C code is done in Eclipse.

2.3.1. Build

Windows:

1. Install MinGW-w64 – you will need `make` and `gcc`
2. In `mingw64\bin` directory, rename `mingw32-make.exe` to `make.exe`
3. Set `GLOBALS_HOME` environment variable to the root of IRIS installation
4. Set `PYTHONHOME` environment variable to the root Python installation (usually `C:\Users\<User>\AppData\Local\Programs\Python\Python3<X>`)
5. Open MinGW shell (`mingw64env.cmd`)
6. In `<Repository>\c\` execute `make`

Linux:

It is recommended to use Linux OS that uses Python 3.X by default, i.e. Ubuntu 18.04.1 LTS. Skip steps 1 and probably 2 if your OS has Python 3.6X as default (to check Python version: `python3 –version` or `python –version` or `python3.6 –version`)

1. Add Python 3.6 repo: add-apt-repository `ppa:jonathonf/python-3.6` and `apt-get update`
2. Install: `apt install python3.6 python3.6-dev libpython3.6-dev build-essential`
3. Set `GLOBALS_HOME` environment variable to the root of IRIS installation
4. Set environment variable `PYTHONVER` to the Python version you want to build, i.e. `export PYTHONVER=3.6`
5. In `<Repository>\c\` execute `make`

Mac OS X:

1. Install Python 3.6 and `gcc` compiler
2. Set `GLOBALS_HOME` environment variable to the root of IRIS installation
3. Set environment variable `PYTHONVER` to the Python version you want to build, i.e. `export PYTHONVER=3.6`
4. In `<Repository>\c\` execute:

   ```
   gcc -Wall -Wextra -fpic -O3 -fno-strict-aliasing -Wno-unused-parameter -I/Library/Frameworks/Python.framework/Versions/${PYTHONVER}/Headers -I${GLOBALS_HOME}/dev/iris-callin/include -c -o iscpython.o iscpython.c
   ```

   ```
   gcc -dynamiclib -L/Library/Frameworks/Python.framework/Versions/${PYTHONVER}/lib -L/usr/lib -lpython${PYTHONVER}m -lpthread -ldl -lutil -lm -Xlinker iscpython.o -o iscpython.dylib
   ```

   If you have a Mac please update `makefile` so we can build Mac version via `make`

2.3.2. Troubleshooting

1. `<DYNAMIC LIBRARY LOAD>` exception
   a. Check that the OS has the correct Python installed:

```
import sys
sys.version
```

The result should contain Python 3.6.7 and 64-bit. If it does not, install Python 3.6.7 64-bit

   b. Check OS-specific installation steps. Make sure that the path relevant for IRIS (usually, the system `PATH`) contains Python installation directory
   c. Make sure that IRIS can access Python installation

2. Module not found error

   Sometimes you may get “module not found” error. This is how you can fix it. Each step constitutes a complete solution requiring IRIS restart and check on whether the problem has been solved

   a. Check that OS and IRIS use the same Python. Open both Pythons, execute the below script in each and verify that the versions are the same:

   ```
   import sys
   ver=sys.version
   ver
   ```

   If they are not the same, search for the Python executable that is used by IRIS

   b. Check that the module is, in fact, installed. Open OS shell, execute `python` (or `python3` or `python36` in Linux) and in the open shell execute `import <module>`. If it fails with an error, in OS shell run `pip install <module>`. Note that a module name for `import` and a module name for `pip` can be different

   c. If you are sure that the module is installed, compare the paths used by Python (nothing to do with system `PATH`). Get the path with:

   ```
   import sys
   path=sys.path
   path
   ```

   The returned paths should be the same. If they are not the same, read how `PYTHONPATH` (Python) is formed here and adjust your OS environment to form it correctly, i.e. set `PYTHONPATH` (system environment variable) to `C:\Users\<USER>\AppData\Roaming\Python\Python36\site-packages` or to other directories where your modules reside (plus other missing directories)

   d. Compare Python paths again, and if they are still not the same or the problem persists, add the missing paths explicitly to `isc.py.ens.OutboundAdaptor` init code (for Interoperability) and on process start (for `Callout` wrapper):

   ```
   do ##class(isc.py.Main).SimpleString("import sys")
   do ##class(isc.py.Main).SimpleString("sys.path.append('C:\\Users\\<USER>\\AppData\\Roaming\\Python\\Python36\\site-packages')")
   ```

3. `undefined symbol: _Py_TrueStruct` or similar errors

   a. Check `ldconfig` and adjust it to point to the directory with Python shared library
   b. If it fails:

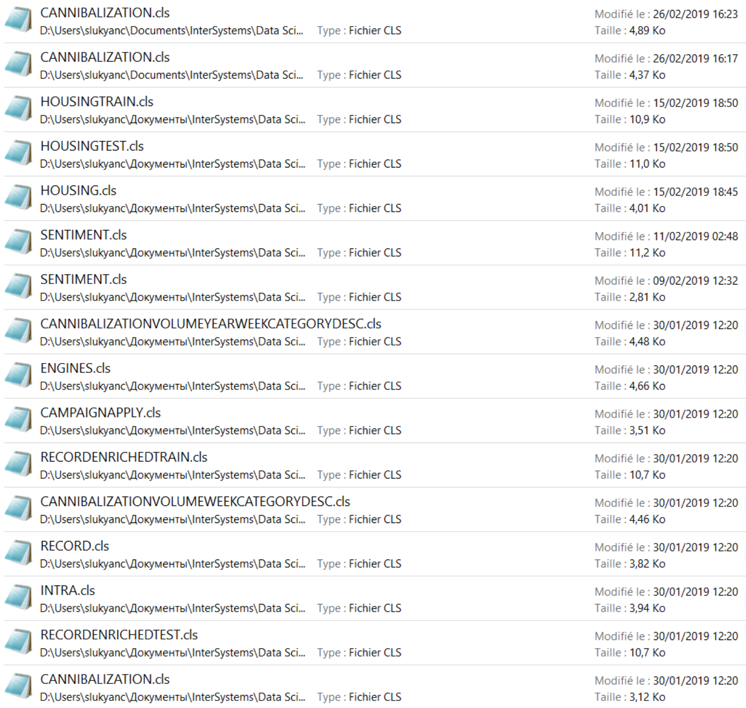
i. for Interoperability: in `isc.py.ens.Operation` operation set `PythonLib` setting at `libpython3.6m.so` or even at a full path to the shared library

ii. for `Callout` wrapper: on process start call `do ##class(isc.py.Callout).Initialize("libpython3.6m.so")`, alternatively pass a full path to the shared library

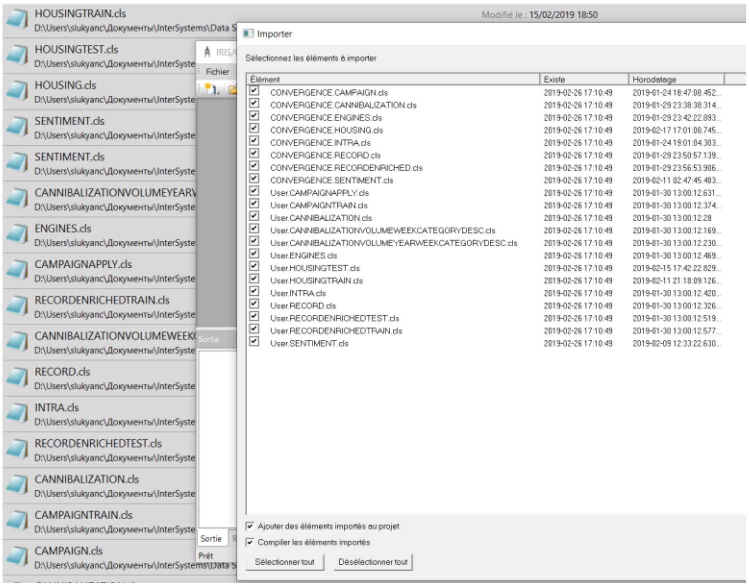
3. IRIS Interoperability Tools

3.1. Installation

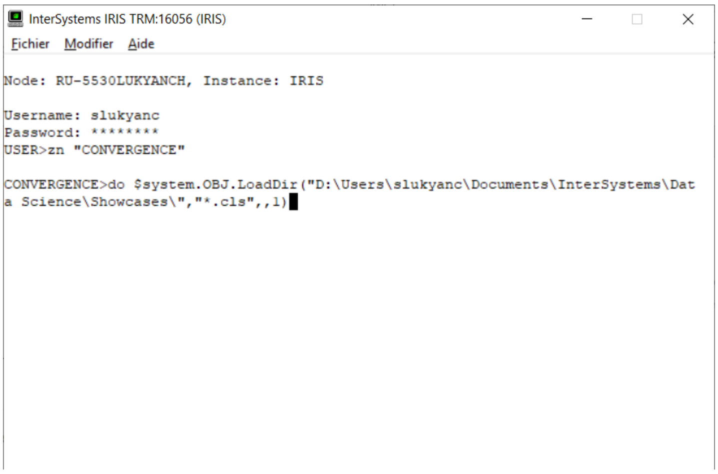
1. Import ObjectScript classes using IRIS Studio
    a. in the folder where you keep ML Toolkit showcases, run a file search using *.cls mask:



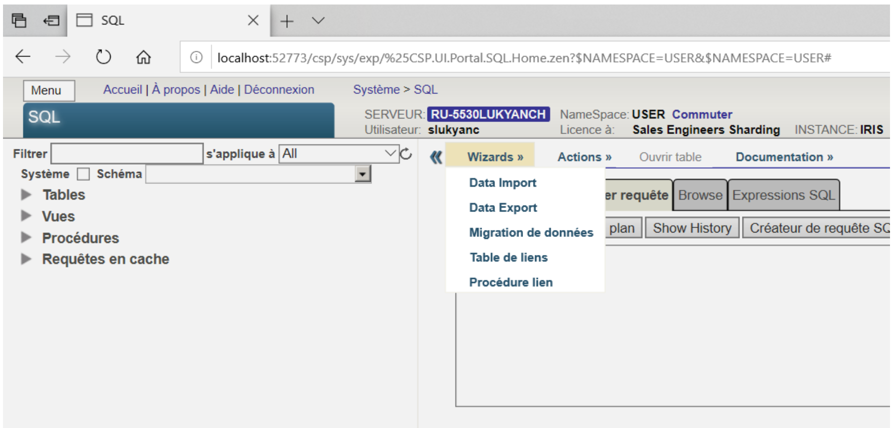
    b. select the files found by the file search above, drag and drop them into your IRIS Studio:

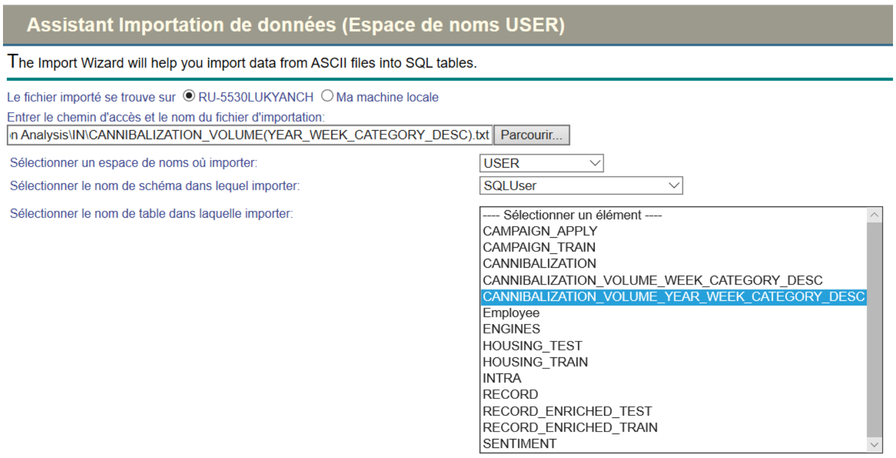


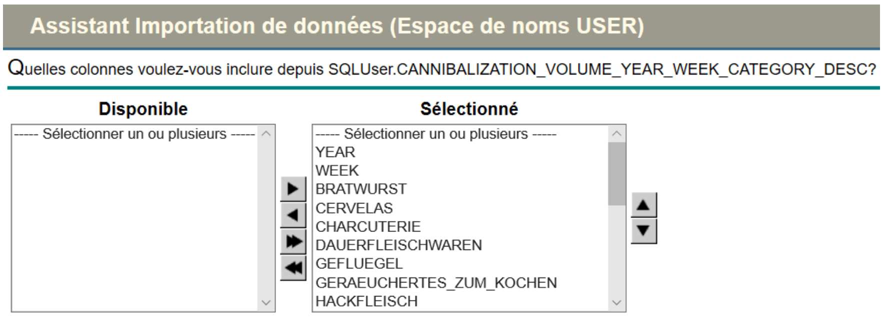
2. Import ObjectScript classes using IRIS Terminal (an alternative to IRIS Studio)
    a. in IRIS Terminal execute the following command (adjust the path to point to the folder where you placed the ML Toolkit showcase class files): `do $system.OBJ.LoadDir("C:\path\to\showcases","*.cls",,1)`

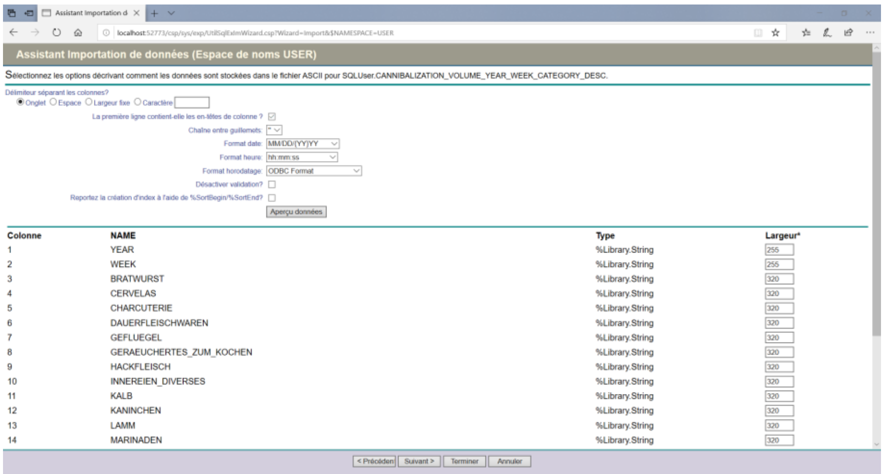


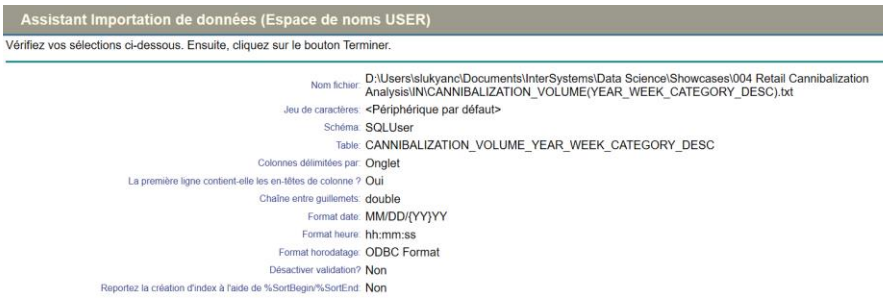
3. Import showcase data
    b. from the folder where you keep ML Toolkit showcase input data, import the showcase dataset into its respective IRIS table using Data Import wizard:









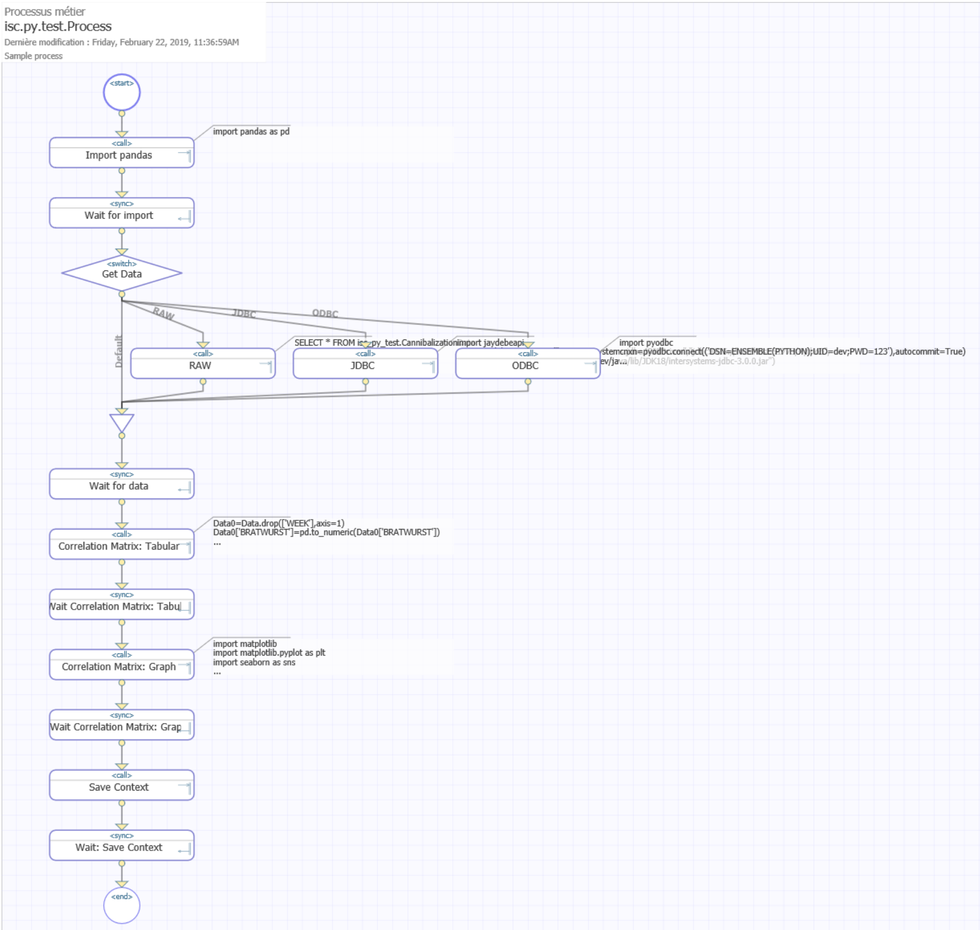


Finish the wizard and the showcase dataset will be imported.

3.2. Use

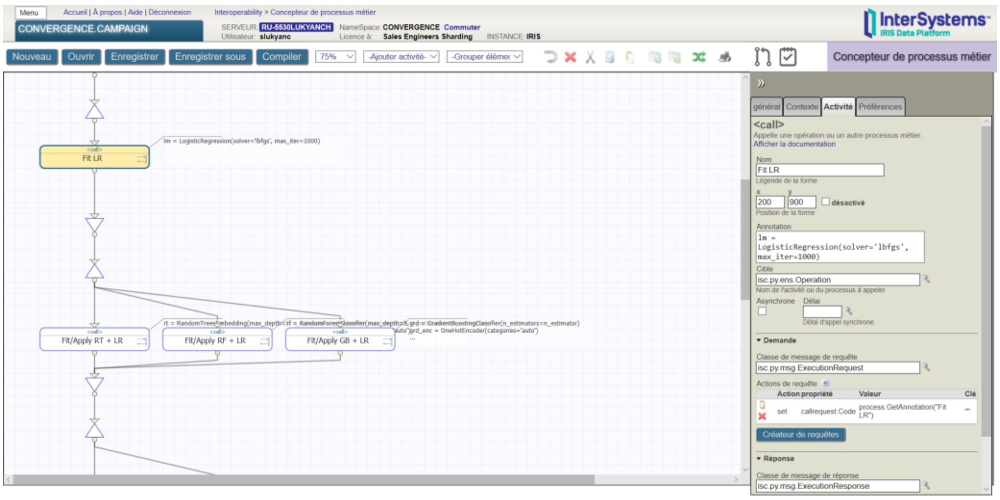
3.2.1. General Use

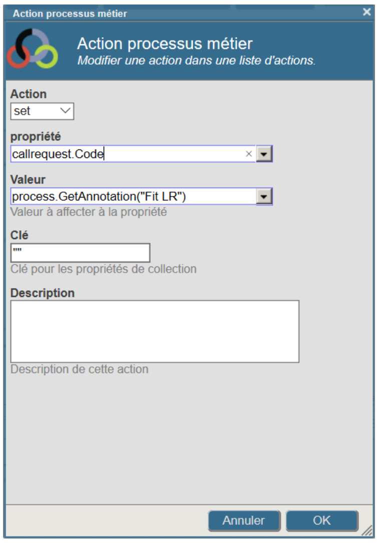
The visual business process designer in IRIS Interoperability is one of the tools for implementing adaptive analytical models (find more at the following link). All toolsets (e.g., Python Tools, etc.) from ML Toolkit can be leveraged in an adaptive model, one adaptive model combining tools coming from various toolsets and generating analytical objects (models, matrices, vectors, datasets, graphs, etc.) in the respective tool's context. As such, extracting analytical objects from the other tools' contexts to IRIS Interoperability context and saving them in IRIS as global variables ("globals"), objects and tables, creates a vast space for analytical interoperability and integration.



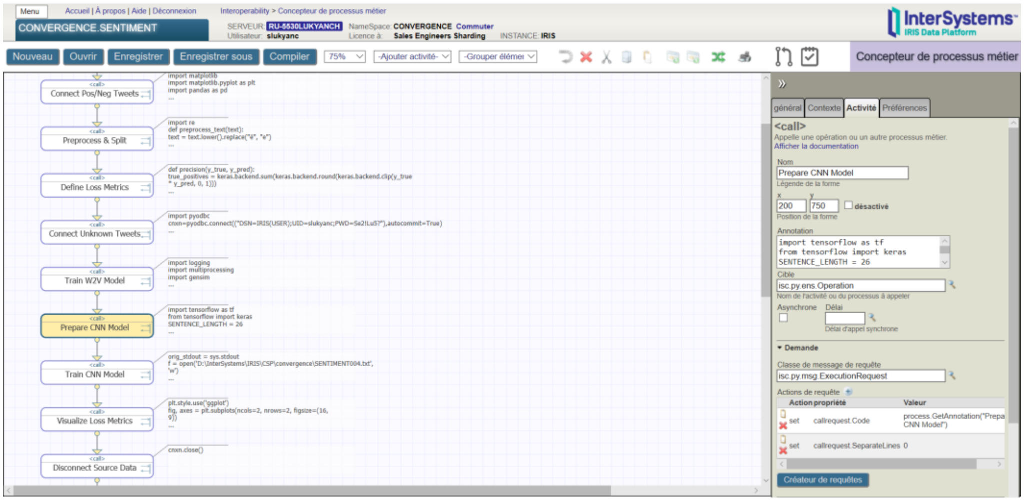
`isc.py.ens.Operation` is the ML Toolkit operation that is added to your IRIS Interoperability production. The following ML Toolkit requests are added:

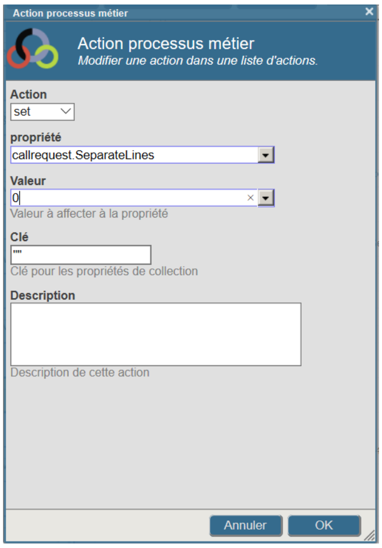
1. Execute Python code via `isc.py.msg.ExecutionRequest` and get the response via `isc.py.msg.ExecutionResponse` with requested variable values as strings:



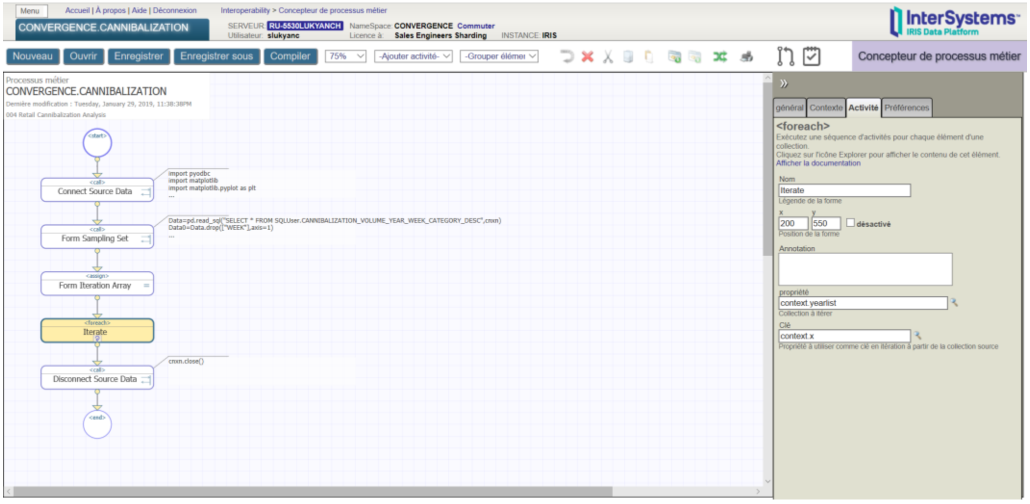


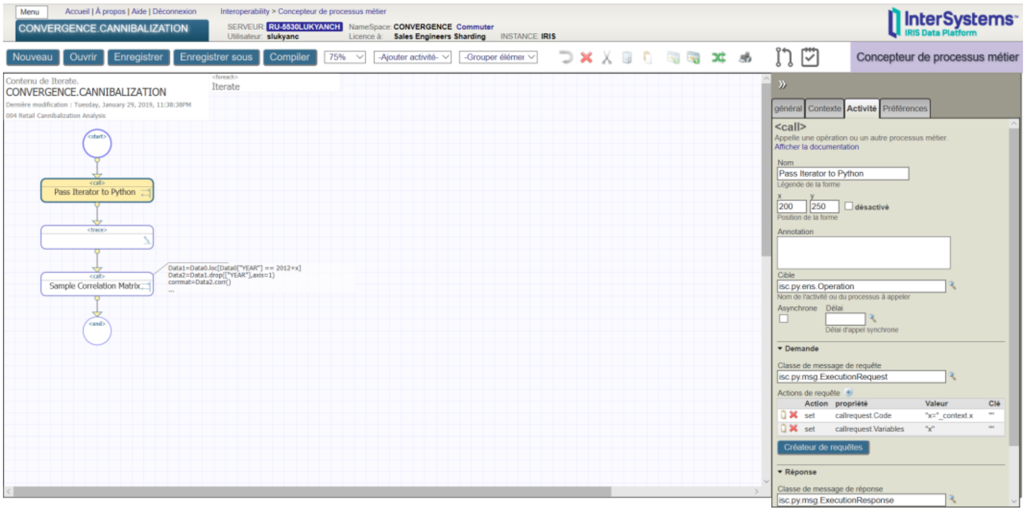
Another use of the same request (adding action to prevent line-by-line interpreting):

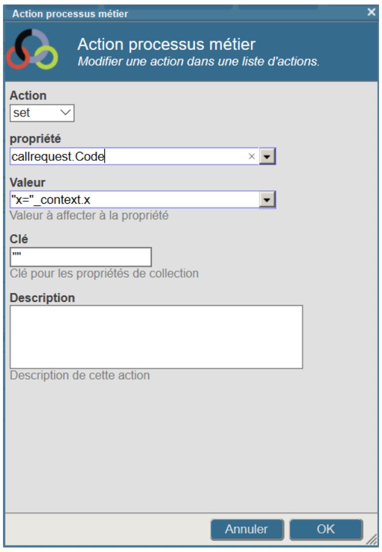


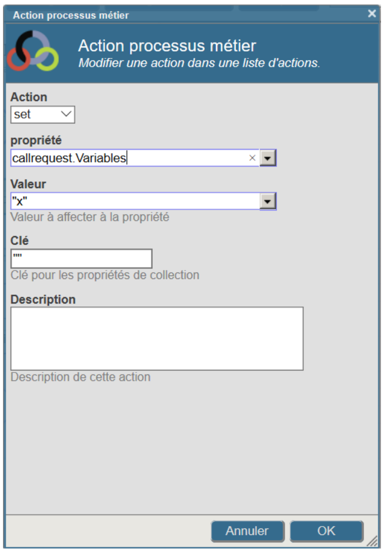


One other use of this request (adding action to pass variables between IRIS and Python contexts):

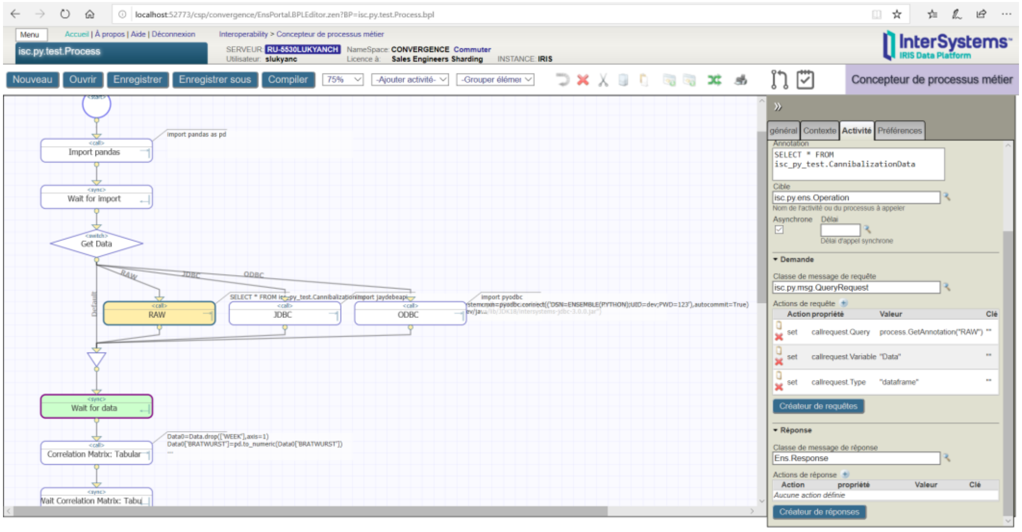


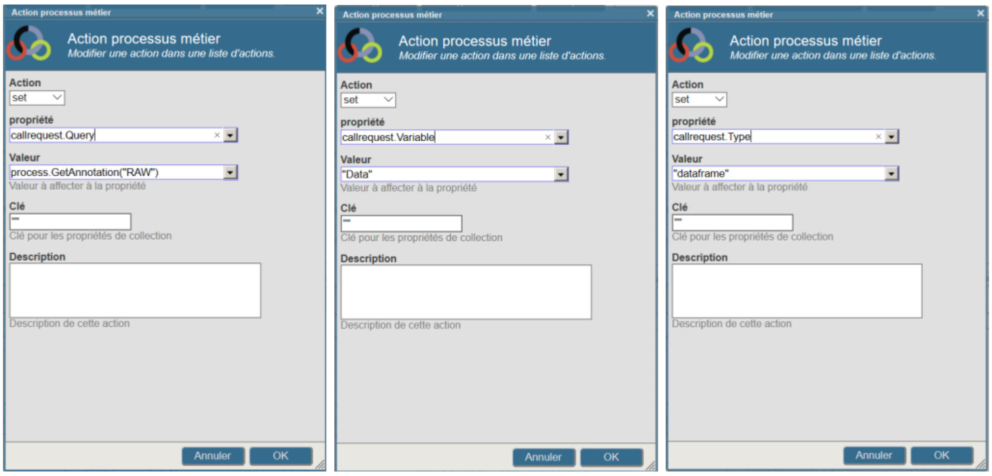




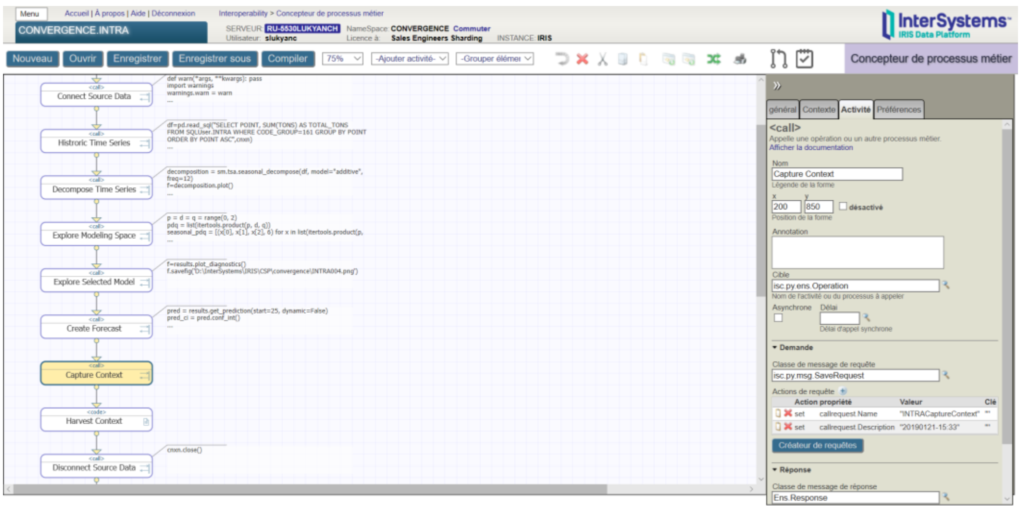


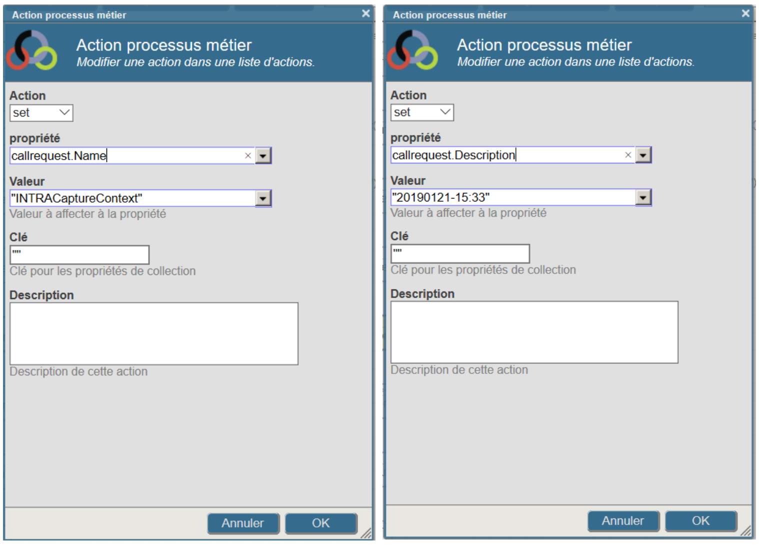
2. Execute Python code via `isc.py.msg.StreamExecutionRequest` and get the response via `isc.py.msg.StreamExecutionResponse` with requested variable values as streams (everything is identical to `isc.py.msg.ExecutionRequest`/ `isc.py.msg.ExecutionResponse` except that all code and variable values are streams)
3. Transfer data into Python from an SQL query with `isc.py.msg.QueryRequest` and get the response via `Ens.Response`



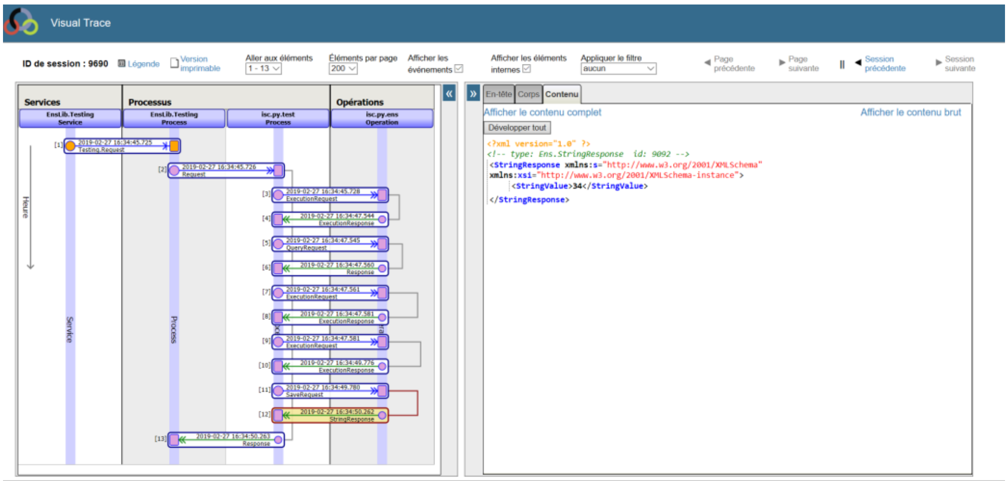


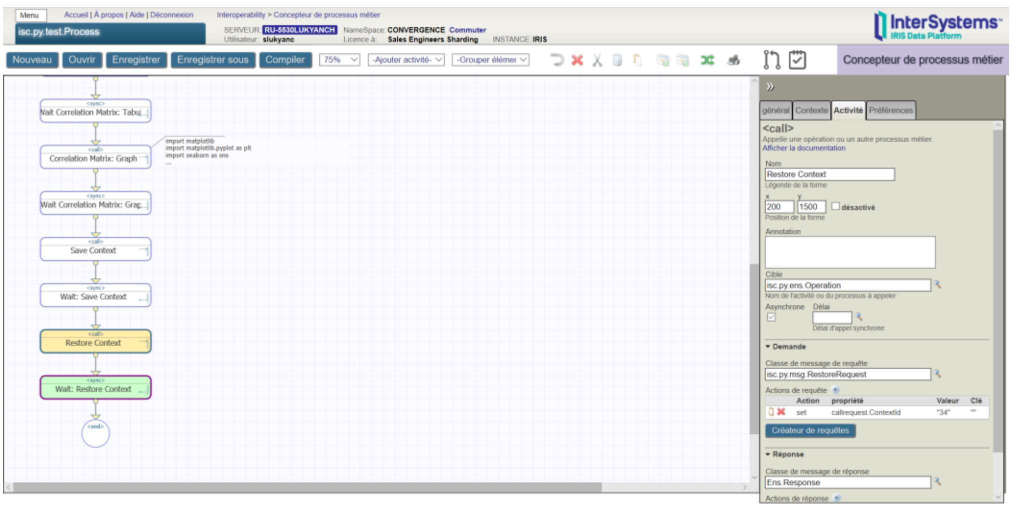
4. Save a Python context with `isc.py. msg.SaveRequest` and get the response via `Ens.StringResponse` with the context ID:

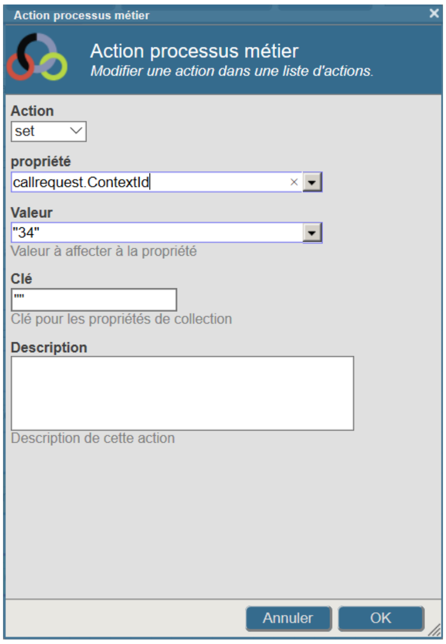


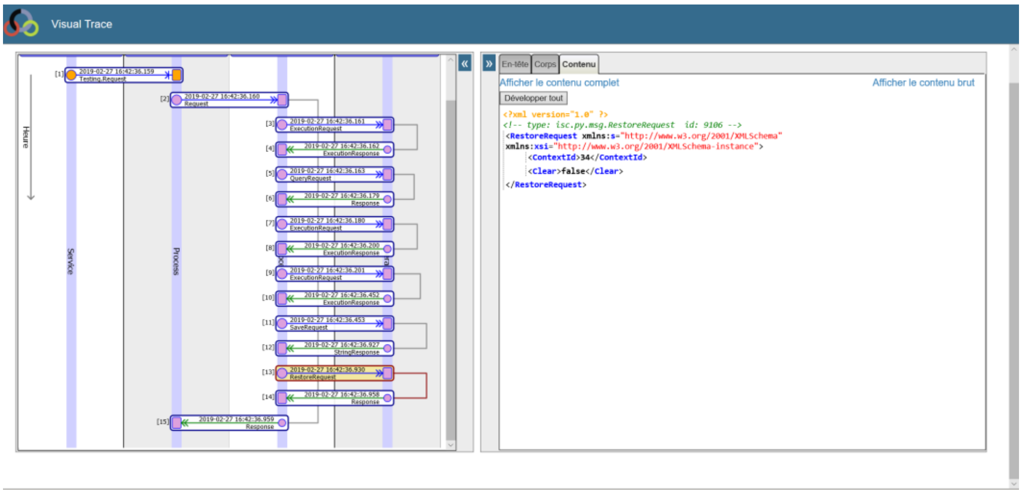


5. Restore a Python context with `isc.py.msg.RestoreRequest`






Action processus métier
Action processus métier
Modifier une action dans une liste d'actions.
Action
set
propriété
callrequest.ContextId
Valeur
"34"
Valeur à affecter à la propriété
Clé
""
Clé pour les propriétés de collection
Description
Description de cette action
Annuler
OK

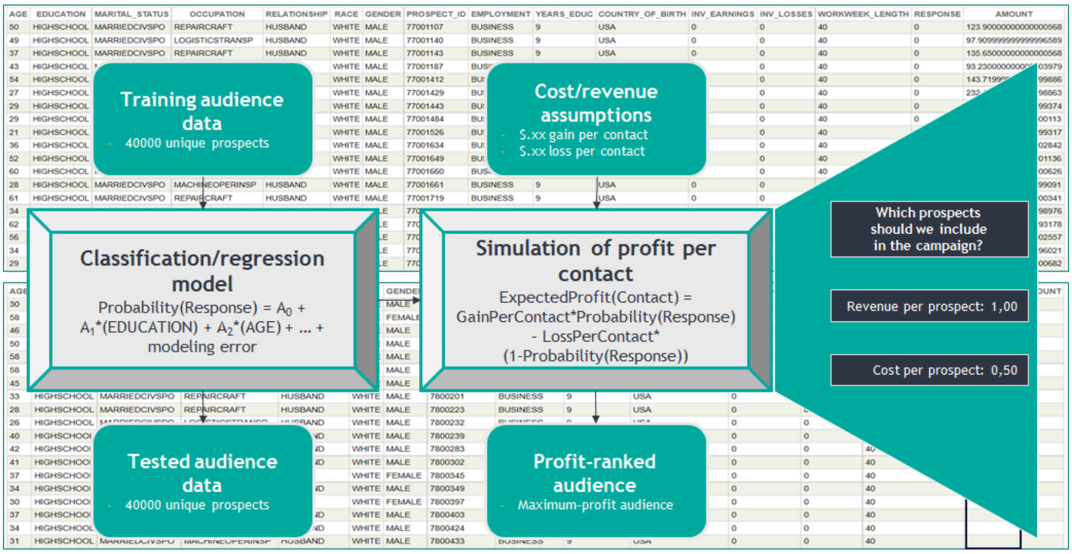

Visual Trace
En-tête
Corps
Contenu
Afficher le contenu complet
Afficher le contenu brut
Développer tout

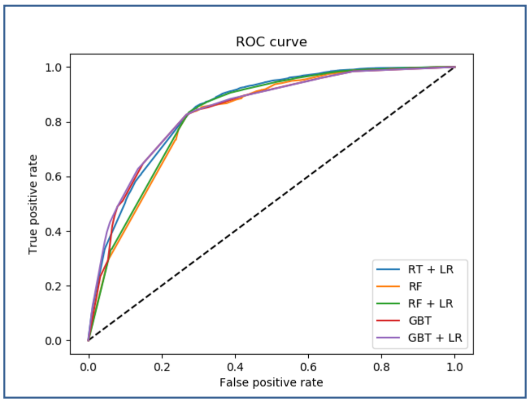
4. Convergent Analytics Showcases

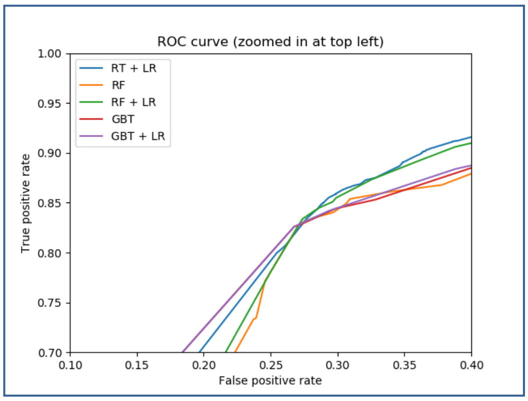
4.1. 005 Marketing Campaign Optimization

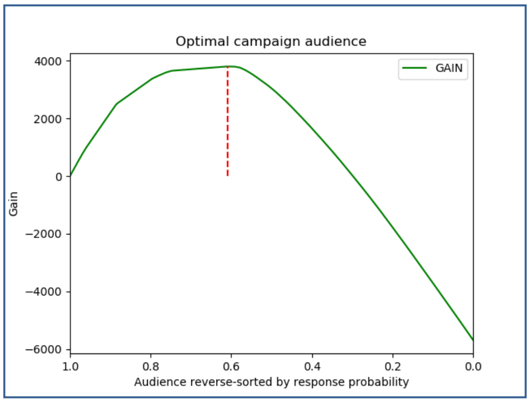
4.1.1. Background

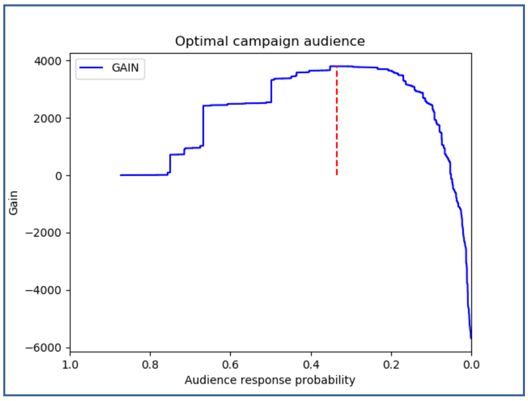
We implement a stack of classification models to have a robust estimate of response probability. We then use response probabilities to calculate the expected gain per prospect. We finally reverse-sort the audience on response probability and estimate the maximum summary gain for the campaign, as well as the part of the audience that generates it.



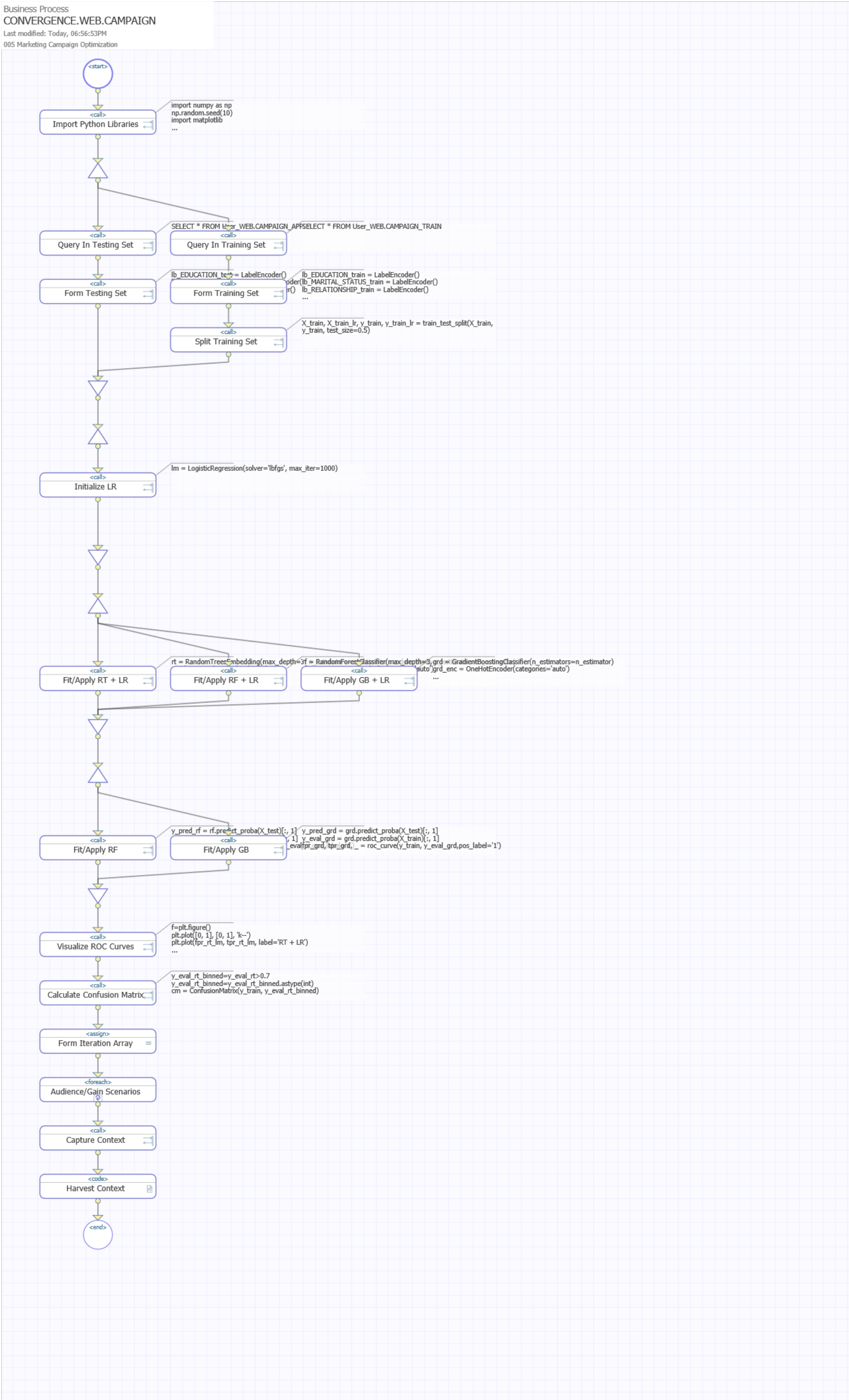








4.1.2. Implementation



4.1.3. Walkthrough

1. Import Python Libraries: loads the required libraries to Python context:
2. Query In Testing Set: loads testing data
3. Form Testing Set: encodes testing data from text labels to numbers
4. Query In Training Set: loads training data
5. Form Training Set: encodes training data from text labels to numbers
6. Split Training Set: splits training data into two subsets – one to train linear regression, another one to train the other models in the showcase
7. Initialize LR: initializes a linear regression model
8. Fit/Apply RT + LR: fits and applies a bundle of random tree and linear regression models (evaluation and modes)
9. Fit/Apply RF + LR: fits and applies a bundle of random forest and linear regression models (evaluation and modes)
10. Fit/Apply GB + LR: fits and applies a bundle of gradient boosting and linear regression models (evaluation and modes)
11. Fit/Apply RF: fits and applies a random forest model (evaluation and modes)
12. Fit/Apply GB: fits and applies a gradient boosting model (evaluation and modes)
13. Visualize ROC Curves: saves to graphical files the ROC charts (makes possible defining the winner model)
14. Calculate Confusion Matrix: calculates a confusion matrix for the winner model
15. Form Iteration Array: forms an array to iterate over while doing the audience/gain sensitivity analysis (next step)
16. Audience/Gain Scenarios: iterates over a set of experimental scenarios to discover the range of possible optimal audience and maximum gain
17. Capture Context: saves all the objects currently existing in Python context to globals in IRIS
18. Harvest Context: extracts from a global saved in the previous step the confusion matrix and saves it as a separate global in IRIS with each node corresponding to a cell value in the confusion matrix

InterSystems Corporation
World Headquarters

One Memorial Drive
Cambridge, MA 02142-1356
Tel: +1.617.621.0600

InterSystems.com