TRAINING

Machine Learning (ML) Toolkit

ML Toolkit Fundamentals

InterSystems
Creative data technology

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 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: 05_2
Created by: Sergey Lukyanchikov, Eduard Lebedyuk, Shiao-Bin Soong

Document controls

Document Modifications			
Version	Date	Description of Change	Modified By
01	2019-02-19	Initial version with Python Tools	Sergey Lukyanchikov Eduard Lebedyuk
02	2019-04-11	Adding R Tools	Sergey Lukyanchikov Eduard Lebedyuk Shiao-Bin Soong
03	2019-12-18	Adding Jupyter Tools	Sergey Lukyanchikov Eduard Lebedyuk
04	2020-06-03	Adding Julia Tools	Sergey Lukyanchikov Eduard Lebedyuk
05	2020-12-08	Restructuring according to DevOps	Sergey Lukyanchikov

Document Authorization		
InterSystems	**Name**	Sergey Lukyanchikov, Eduard Lebedyuk, Shiao-Bin Soong
	Role	Sales Engineer, Sales Engineer, Developer
	Signature	< Insert electronic signature >
	Date	2024-02-26
None	**Name**	< Insert customer contact >
	Role	< e.g. Project Manager >
	Signature	< Insert electronic signature >
	Date	< Select date >

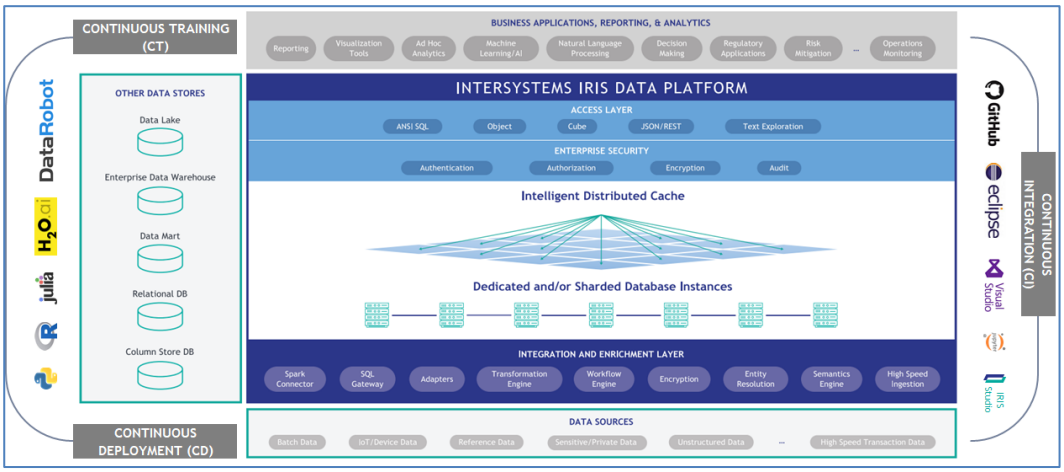
Contents

1. General

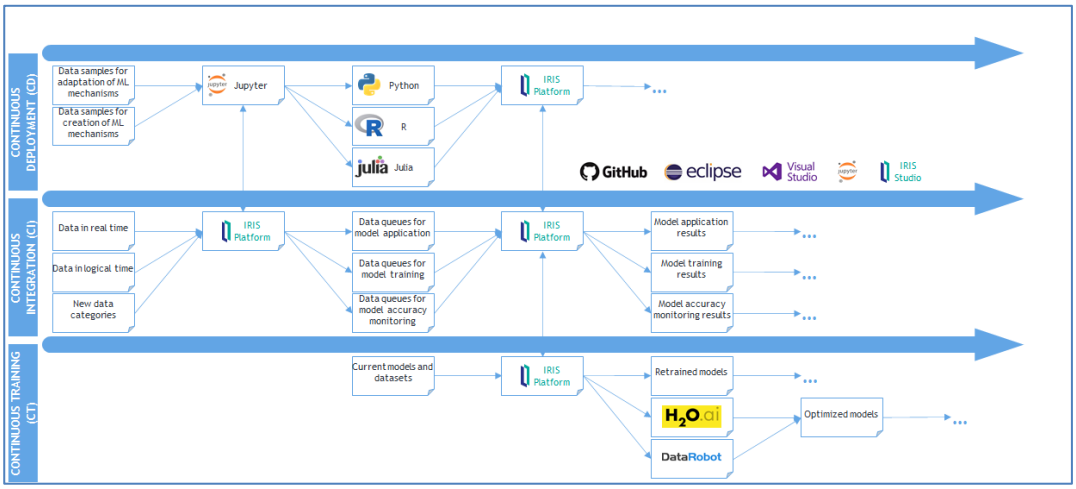
1.1. InterSystems IRIS – the All-Purpose Universal Platform for Real-Time AI/ML

InterSystems IRIS possesses key in-platform capabilities for supporting real-time AI/ML solutions. These capabilities can be grouped in three major categories (see also the diagram below):

- **Continuous Deployment/Delivery (CD)** of new or modified existing AI/ML mechanisms in a production solution functioning in real time based on InterSystems IRIS platform
- **Continuous Integration (CI)** of inbound and outgoing process data flows, AI/ML model application/training/accuracy control queues, data/code/orchestration around real-time interactions with mathematical modeling environments – in a single production solution in InterSystems IRIS platform
- **Continuous Training (CT)** of AI/ML mechanisms performed in mathematical modeling environments using data, code, and orchestration (“decision making”) passed from InterSystems IRIS platform



The below diagram combines the “basis” (CD/CI/CT) with the information flows among the working elements of the platform. Visualization begins with CD macromechanism and continues through CI/CT macromechanisms.



1.1.1. Continuous Development (CD)

The platform users (the AI/ML solution developers) adapt the already existing and/or create new AI/ML mechanisms using a specialized AI/ML code editor: Jupyter (the full title: Jupyter Notebook; for brevity, the documents created in this editor are also often called by the same title). In Jupyter, a developer can write, debug and test (using visual representations, as well) a concrete AI/ML mechanism before its transmission ("deployment") to InterSystems IRIS. It is clear that the new mechanism developed in such a manner will enjoy only a basic debugging capability (in particular, because Jupyter does not handle real-time data flows) – but we are fine with that since the main objective of developing code in Jupyter is verification, in principle, of the functioning of a separate AI/ML mechanism. In a similar fashion, an AI/ML mechanism already deployed in the platform (see the other macromechanisms) may require a "rollback" to its "pre-platform" version (reading data from files, accessing data via xDBC instead of local tables or globals – multi-dimensional data arrays in InterSystems IRIS – etc.) before debugging.

An important distinctive aspect of CD implementation in InterSystems IRIS: there is a bidirectional integration between the platform and Jupyter that allows deploying in the platform (with a further in-platform processing) Python, R and Julia content (all the three being programming languages of their respective open-source mathematical modeling environments). That said, AI/ML content developers obtain a capability to "continuously deploy" their content in the platform while working in their usual Jupyter editor with usual function libraries available through Python, R, Julia, delivering basic debugging (in case of necessity) outside the platform.

1.1.2. Continuous Integration (CI)

Continuing with CI macromechanism in InterSystems IRIS. The previous diagram presents the macroprocess for a "real-time robotizer" (a bundle of data structures, business processes and fragments of code in mathematical environment languages, as well as in ObjectScript – the native development language of InterSystems IRIS – orchestrated by them). The objective of the macroprocess is: to support data processing queues required for the functioning of AI/ML mechanisms (based on the data flows transmitted into the platform in real time), to make decisions on sequencing and "assortment" of AI/ML mechanisms (a.k.a. "mathematical algorithms", "models", etc. – can be called in a number of different ways depending on implementation specifics and terminology preferences), to keep up to date the analytical structures for intelligence around AI/ML outputs (cubes, tables, multidimensional data arrays, etc. – resulting into reports, dashboards, etc.).

An important distinctive aspect of CI implementation in InterSystems IRIS: there is a bidirectional integration among the platform and mathematical modeling environments that allows executing in-platform content written in Python, R or Julia in the respective environments and receiving back execution results. That integration works both in a "terminal mode" (i.e., the AI/ML content is formulated as ObjectScript code performing callouts to mathematical environments), and in a "business process mode" (i.e., the AI/ML content is formulated as a business process using the visual composer, or, sometimes, using Jupyter, or, sometimes, using an IDE – IRIS Studio, Eclipse, Visual Studio Code). The availability of business processes for editing in Jupyter is specified in the diagram using a link between IRIS within CI layer and Jupyter within CD layer. A more detailed overview of integration with mathematical modeling environments is provided further.

1.1.3. Continuous Training (CT)

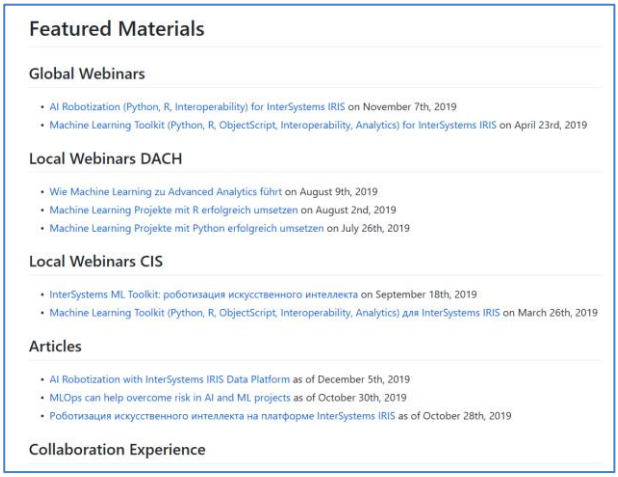
And finally, the crucial macromechanism: CT. The essence of CT is the ability of the platform to operate the "artifacts" of machine learning and artificial intelligence directly in the sessions of mathematical modeling environments: models, distribution tables,

vectors/matrices, neural network layers, etc. This “interoperability”, in the majority of the cases, is manifested through creation of the mentioned artifacts in the environments (for example, in the case of models, “creation” consists of model specification and subsequent estimation of its parameters – the so-called “training” of a model), their application (for models: computation with their help of the “modeled” values of target variables – forecasts, category assignments, event probabilities, etc.), and improvement of the already created plus applied artifacts (for example, through re-definition of the input variables of a model based on its performance – in order to improve forecast accuracy, as one possible option).

An important distinctive aspect of CT implementation in InterSystems IRIS: using the above-mentioned integration with mathematical modeling environments, the platform can extract their artifacts from sessions in the mathematical environments orchestrated by it, and (the most important) convert them into in-platform data objects. For example, a distribution table just created in a Python session can be (without pausing the Python session) transferred into the platform as, say, a global (a multidimensional data array in InterSystems IRIS) – and further re-used for computations in a different AI/ML mechanism (implemented using the language of a different environment – like R) – or as a virtual table. Another example: in parallel with “routine” functioning of a model (in a Python session), its input dataset is processed using “auto ML” – an automated search for optimized input variables and model parameters. Together with “routine” training, the production model receives in real time “optimization suggestions” as to basing its specification on an adjusted set of input variables, on adjusted model parameter values (no longer as an outcome of training in Python, but as the outcome of training of an “alternative” version of it using, for example, H2O framework), allowing the overall AI/ML solution to handle in an autonomous way unforeseen drift in the input data and in the modeled objects/processes.

1.2. Resources for External Users: Convergent Analytics Community

Convergent Analytics is a public GitHub repository maintained by InterSystems and external practitioners of the so-called “convergent analytics” concept: a platform-centric approach to developing analytical tools and solutions aiming to maximize the advantages of combining multiple analytic toolsets (AI/ML, BI, SQL, Quantum, IoT, MapReduce, NLP, DL/CV, etc.) in a single analytic process or a system of agent processes.



Convergent Analytics provides the following publicly available resources:

- **Featured Materials** – links to webinar recordings, articles, testimonials, tutorials
- **Required Downloads** – links to component downloads for a setup of an InterSystems IRIS AI/ML landscape

- **Root Resources** – manuals that assist setting up and usage of an InterSystems IRIS AI/ML landscape
- **Showcases** – data, table definitions and code to re-produce a pre-configured analytical scenario in an InterSystems IRIS AI/ML landscape

1.3. Resources for External Users: Open Exchange Pages

Open Exchange is a public catalog maintained by InterSystems for a free download of functional extensions for its standard products. Those extensions are a developer community initiative with respective licenses and disclaimers specified. The following toolsets of ML Toolkit are available via Open Exchange as of the publish date of this document:

- **Python Gateway**

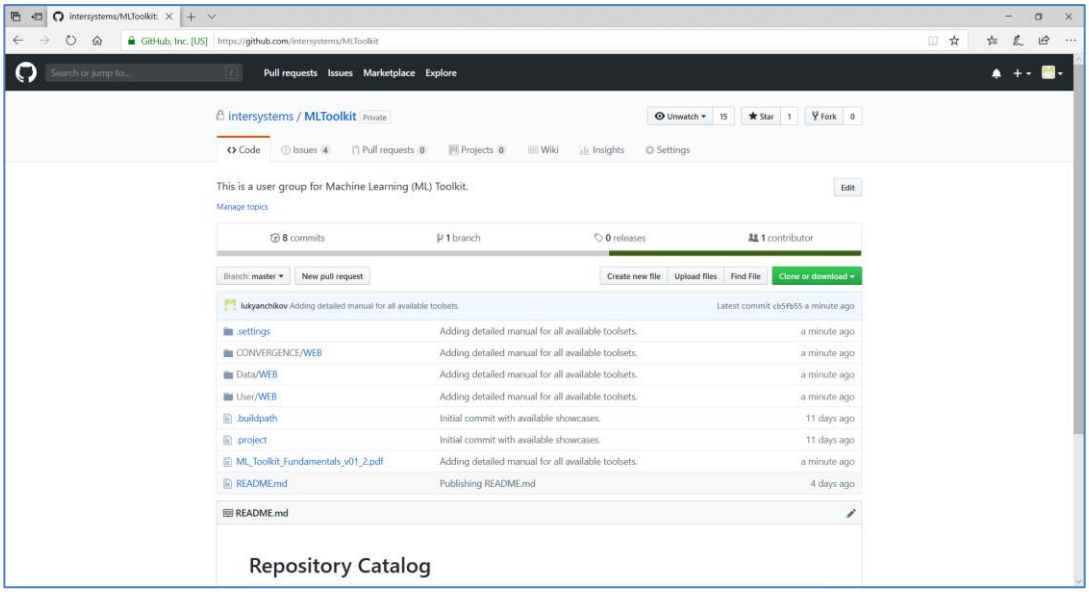
- **R Gateway**

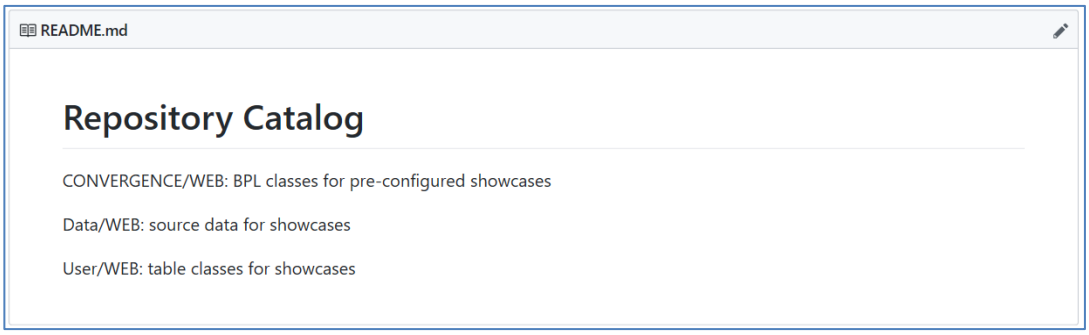
- **Julia Gateway**

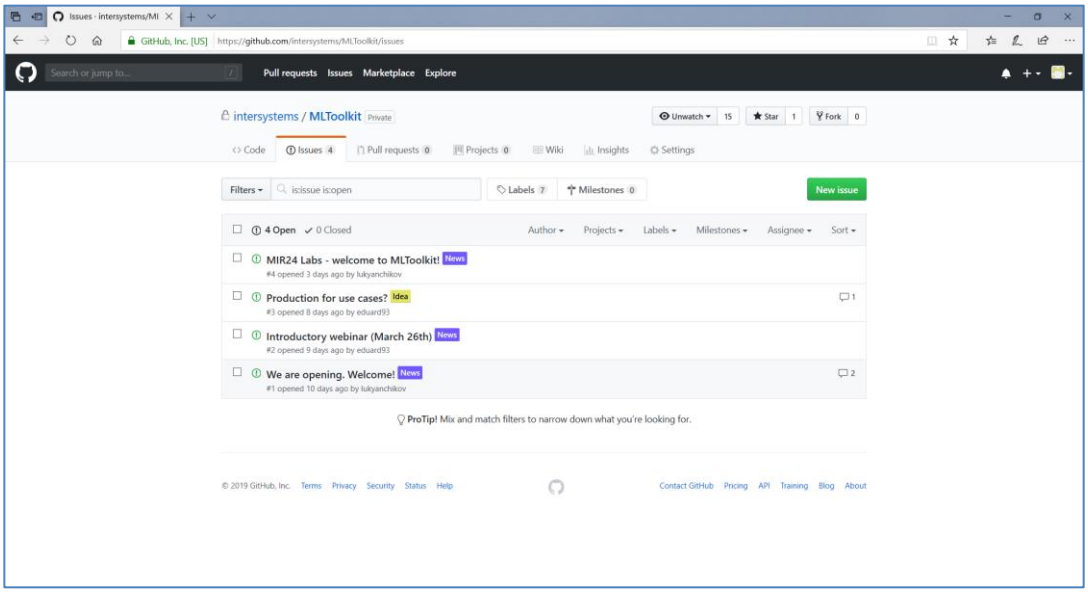
1.4. Resources for External Users: ML Toolkit User Group

ML Toolkit user group is a private GitHub repository set up as part of InterSystems corporate GitHub organization. It is addressed to the external users that are installing, learning or are already using ML Toolkit components as specified in this document. To join ML Toolkit user group, please send a short e-mail at the following address: MLToolkit@intersystems.com and indicate in your e-mail the following details (needed for the group members to get to know and identify you during discussions):

- GitHub account
- Full Name (your first name followed by your last name in Latin script)
- Organization (you are working for, or you study at, or your home office)
- Position (your actual position in your organization, or “Student”, or “Independent”)
- Country (you are based in)

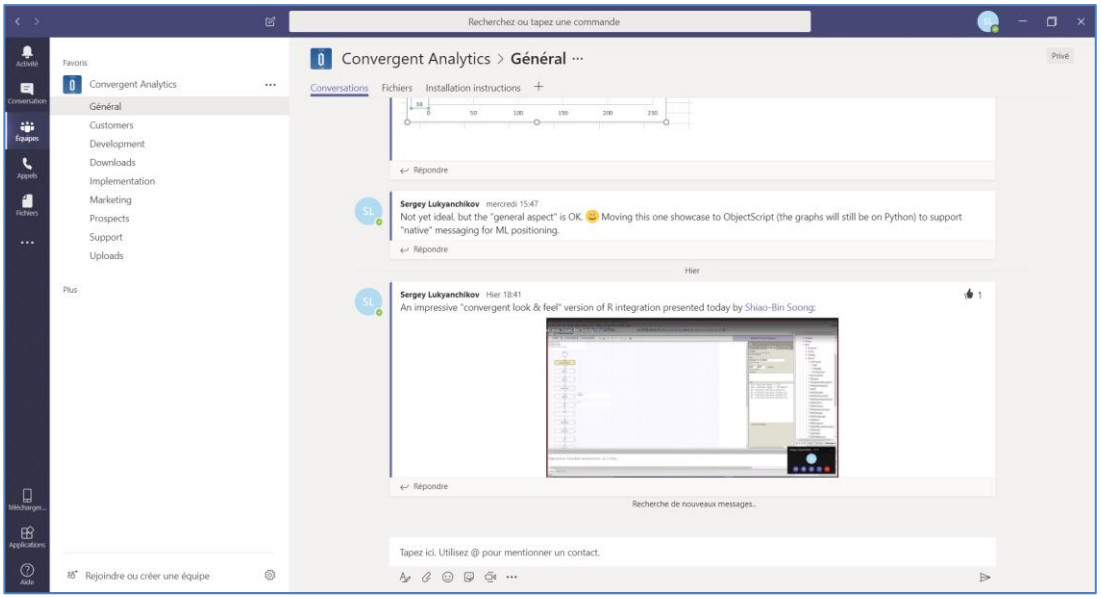






1.5. Resources for Internal Users: MS Teams Group

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
- **Customers** – read and contribute updates about ML Toolkit experience by our valued Customers
- **Development** – stay in sync with vision and thoughts from our code contributors
- **Implementation** – send questions and problems regarding the methodology side of ML Toolkit, leverage help from our colleagues
- **Marketing** – learn about ML Toolkit events, webinars and campaigns driven by our Marketing
- **Downloads** – a shortcut to the copy of the internal OneDrive folder with MS Toolkit extensions, showcases and tutorials
- **Prospects** – share updates on your opportunities that involve ML Toolkit, read updates from our colleagues
- **Support** – send questions and problems regarding the technical side of ML Toolkit, receive answers and support
- **Thought** – contribute links to articles, interviews, posts and other thought leadership materials
- **Uploads** – a channel to upload any occasional material relevant to your posts in the other channels

1.6. Resources for 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
  - R – a set of integration components required to call R out from IRIS
  - ObjectScript – various utility-level extensions that complement the main extensions for working with Python and R
- **Marketing** – contains marketing materials (recordings, presentations, etc.)
  - AI-ML Highlights – positioning presentations
  - Conferences – slides and recordings from conferences
  - Webinars – slides and recordings from webinars
- **Showcases** – contains pre-configured examples and explanatory slides
- **Tutorials** – contains educational materials focused on ML Toolkit

2. Continuous Development (CD)

2.1. Jupyter Notebook

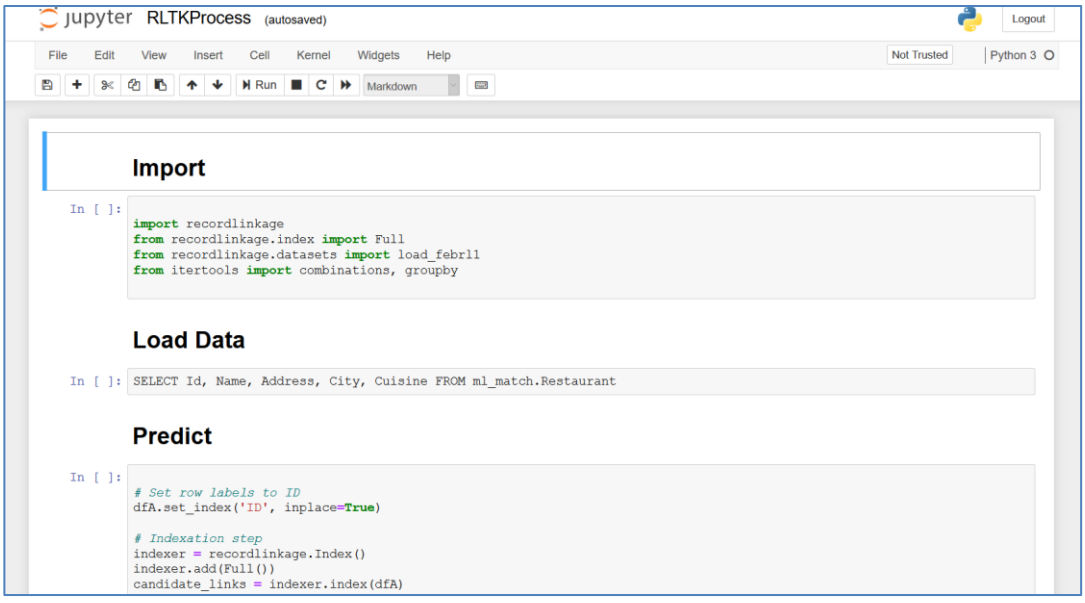
2.1.1. General

Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. Our extension allows you to browse and edit InterSystems IRIS BPL processes as Jupyter notebooks. Note that currently default Python 3 executor is used. Our extension assumes that annotations contain Python code and uses activity names as preceding read-only headings.

2.1.2. Screenshots

Process Explorer

Process Editor



2.1.3. Installation

- You will need InterSystems IRIS 2019.2 or higher
- Install PythonGateway v0.8 or higher (only `isc.py.util.Jupyter` and `isc.py.ens.ProcessUtils` are required)

Automatic installation:

- Run `do ##class(isc.py.util.Jupyter).Install()` and follow the prompt

Manual installation:

- Install IRISNative for Python 3.6.7 (`cp36` should be in the filename, the wheel is in `\dev\python\` folder inside InterSystems IRIS installation path): `pip install <IRIS>\dev\python\irisnative-*.whl`
- Install Jupyter: `pip install jupyter`
- Check `jupyter_notebook_config.py`. It assumes the following defaults for IRIS connection:
  - host: `localhost`
  - port: `51773`
  - namespace: `USER`
  - user: `_SYSTEM`
  - password: `SYS`
- If you need other connection parameters values, modify `jupyter_notebook_config.py`. For example, to connect to InterSystems IRIS instance on port `51776` you will need to add this line to the bottom of `jupyter_notebook_config.py`: `c.MLContentsManager.port = 51776`

Running:

- Open OS bash in the folder with `jupyter_notebook_config.py` and `MLContentsManager.py` and start Jupyter with: `jupyter notebook`

2.1.4. Notes

- Ignore “Checkpoint failed” warning on save

3. Continuous Integration (CI): Python Tools

3.1. Python Gateway

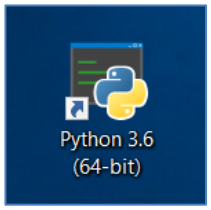
Python Gateway for InterSystems data platform. Execute Python code and more from InterSystems IRIS. This toolset brings you the power of Python right into your InterSystems IRIS environment:

- Execute arbitrary Python code
- Seamlessly transfer data from InterSystems IRIS into Python or from Python to InterSystems IRIS
- Build intelligent Interoperability business processes with Python Interoperability Adapter
- Save, examine, modify and restore Python context from InterSystems IRIS

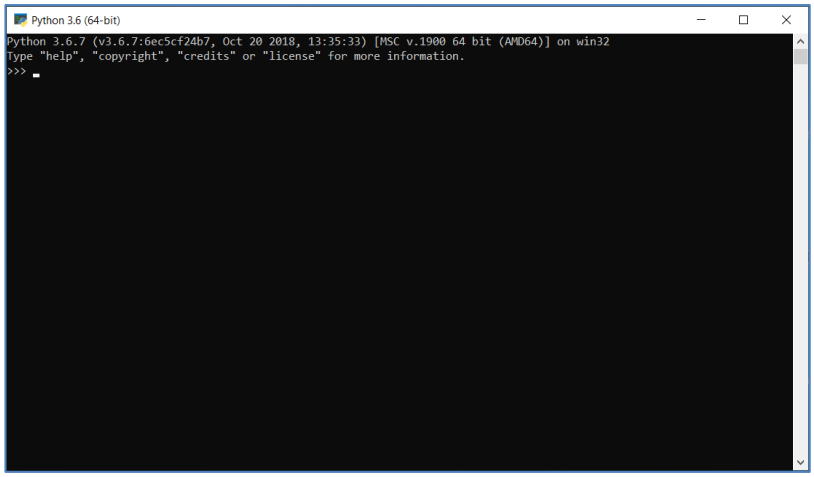
IMPORTANT: this project is a community-supported bridge between InterSystems IRIS and Python. Starting with 2020.3, InterSystems IRIS provides out-of-the-box Python Gateway implementation, officially supported by InterSystems. Documentation is available here: https://docs.intersystems.com/irislatest/csp/docbook/Doc.View.cls?KEY=BPYGATE

3.2. Installation

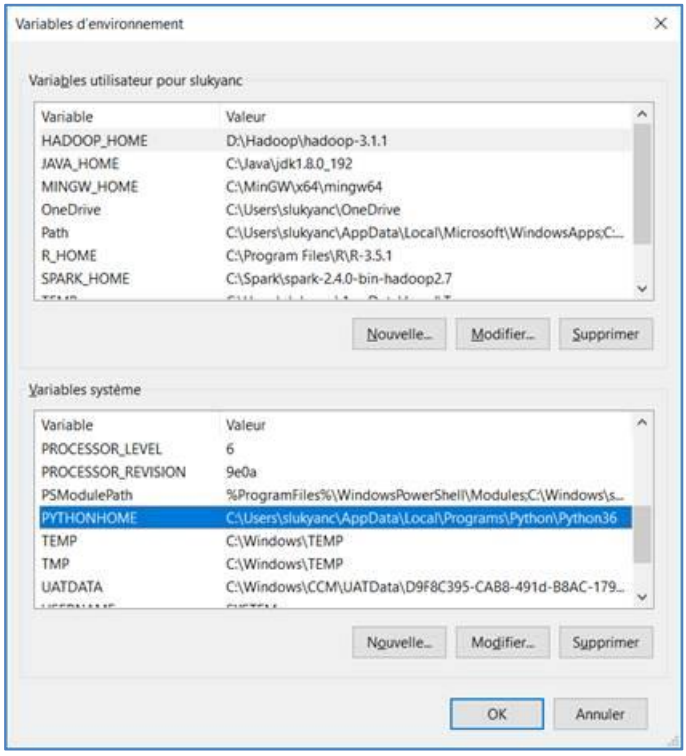
1. Install IRIS (**IMPORTANT:** make sure that the IRIS instance service runs under your Windows account and not under a system service account. This is because Python by default installs into your Windows user workspace. You may run IRIS under system service account, but you need to make sure that IRIS 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):



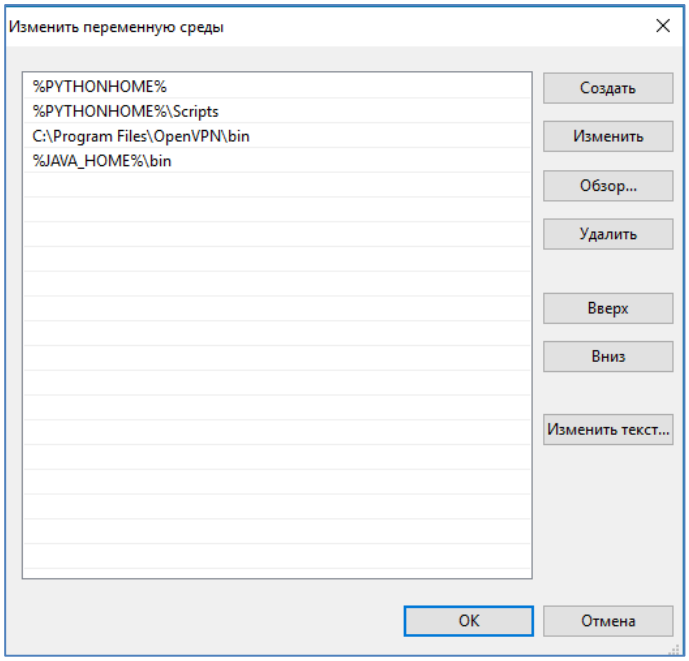
        iv. If you double-click it, the following window opens (you can leave it by pressing Ctrl+Z and then pressing Enter):



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, check that your `PATH` system environment variable includes `/usr/lib` and `/usr/lib/x86_64-linux-gnu`, preferably at the beginning. Use `/etc/environment` file to set the environment variables. In case of errors check Troubleshooting section `undefined symbol: _Py_TrueStruct` and specify `PythonLib` property.
If you are on Mac, only python 3.6.7 from www.python.org is currently supported. Check `PATH` variable.

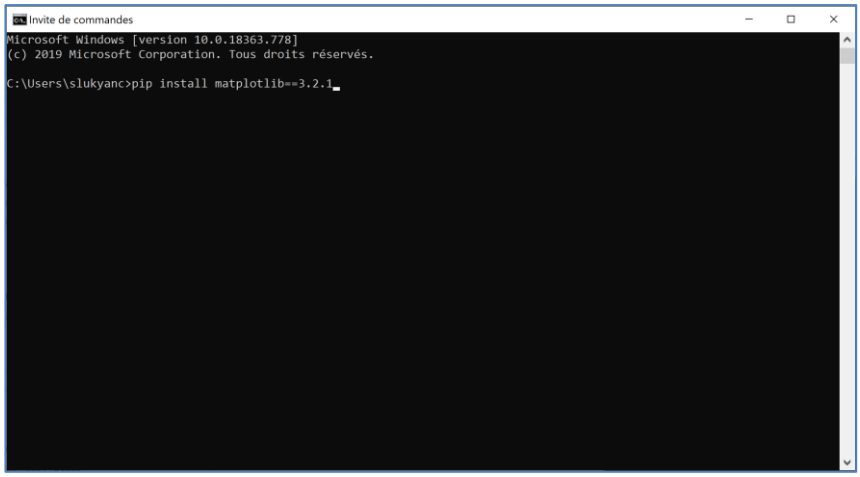
vi. Restart your computer for the environment variable changes to take effect.

vii. In the InterSystems IRIS Terminal, run:

1. `write $SYSTEM.Util.GetEnviron("PYTHONHOME")` and verify it prints out the directory of Python installation
2. `write $SYSTEM.Util.GetEnviron("PATH")` and verify it prints out the directory of Python installation and Scripts folder inside Python installation

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 module versions using `pip install <module name>==<version>` command (you need to be connected to the Internet while doing this):



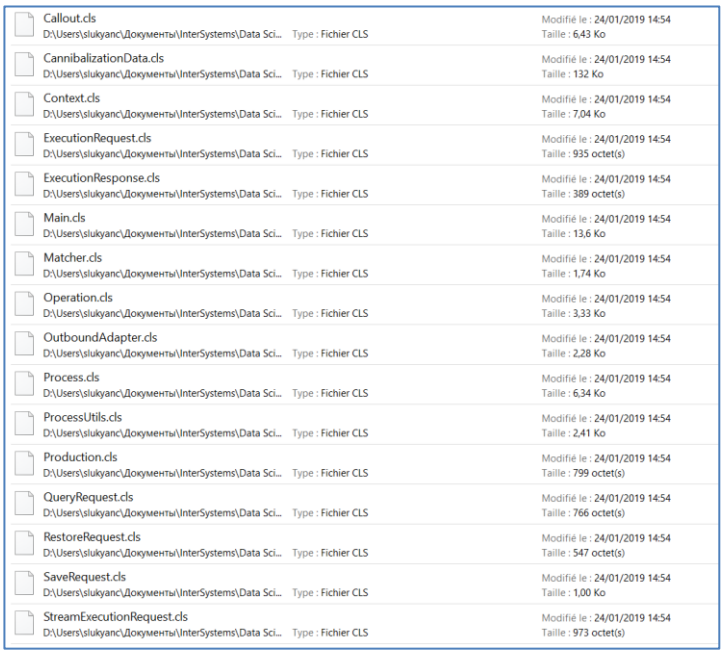
- `dill 0.3.1.1` (required for context harvesting)
- `matplotlib 3.2.1`
- `numpy 1.18.4`
- `pandas 0.24.2`
- `seaborn 0.10.1`
- `sklearn 0.20.0`
- `statsmodels 0.11.1`
- `tensorflow 1.5.0`
- `gensim 3.8.3`
- `pandas_ml 0.6.1`
- `scipy 1.4.1`

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

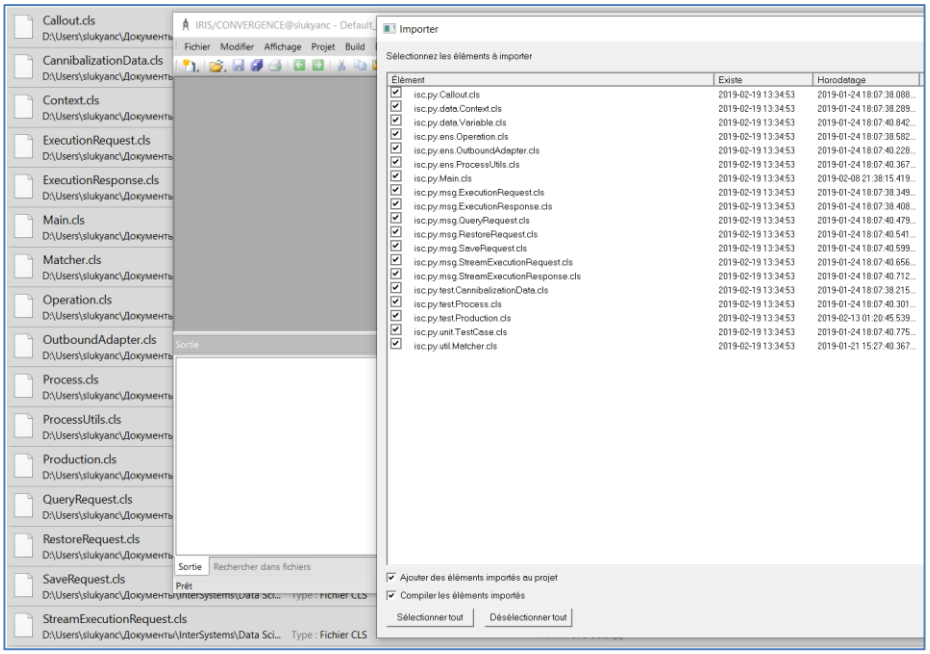
iii. During the installation various warnings or even error messages can be thrown – we will ignore them for the time being.

3. Import ObjectScript classes using IRIS Studio

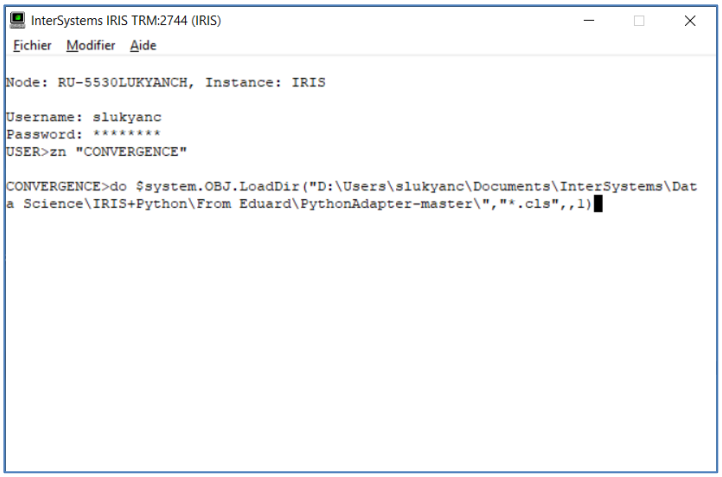
   a. in the folder where you keep ML Toolkit installation set, run a file search using `*.cls` mask:

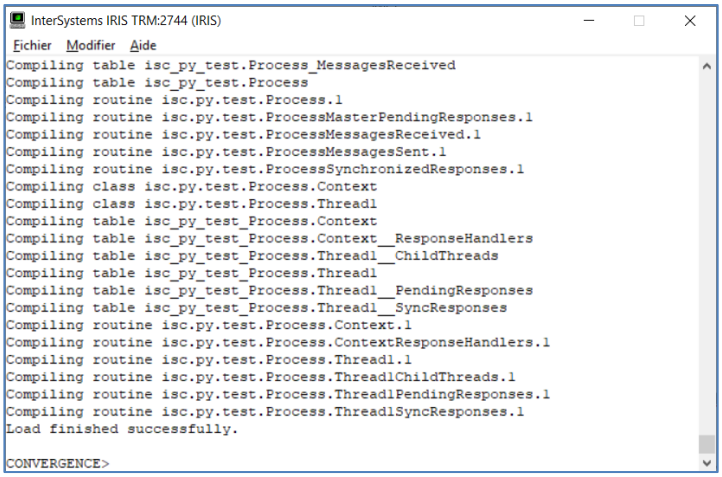


   b. select the files found by the file search above, drag and drop them into your IRIS Studio:

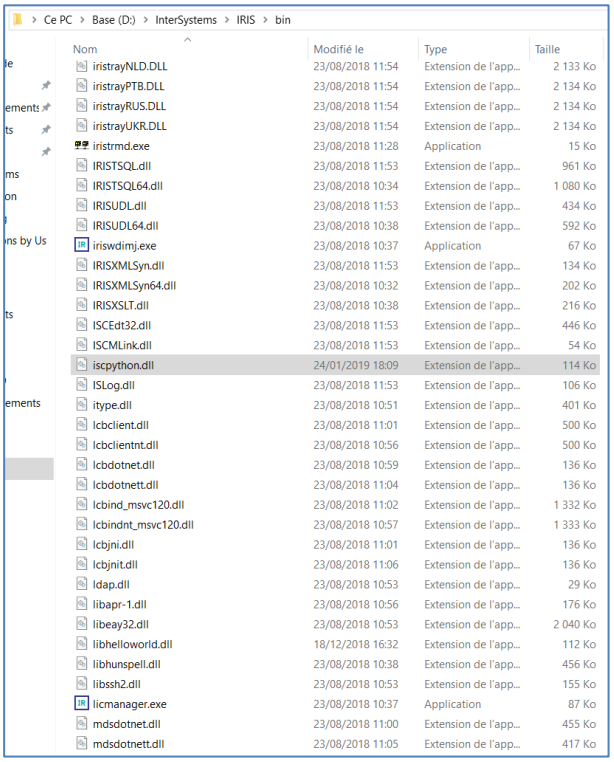


4. Import ObjectScript classes using IRIS Terminal (an alternative to IRIS Studio)
    a. in IRIS Terminal execute the following command (adjust the path to point to the folder where you placed the ML Toolkit class files): `do $system.OBJ.LoadDir("C:\path\to\toolkit","*.cls",,1)` in production (Ensemble-enabled) namespace. In case you want to Ensemble-enable namespace, call: `write ##class(%EnsembleMgr).EnableNamespace($Namespace, 1)`





5. Copy the library file 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 (the library file should be placed into a path returned by `write ##class(isc.py.Callout).GetLib()`



3.3. Docker

1. To build a Docker image:
   - Copy `iscpython.so` into repository root (if it's not there already)
   - Execute in repository root `docker build --force-rm --tag intersystemsdc/irispy:latest`. By default, the image is built upon `intersystems/iris:2019.4.0.383.0` image, however you can change that by providing `IMAGE` variable. To build from InterSystems IRIS Community Edition execute: `docker build --build-arg`

```
IMAGE=store/intersystems/iris-community:2019.4.0.383.0 --
force-rm --tag intersystemsdc/irispy:latest.
```

2. To run the Docker image execute (key is not needed for Community based images):

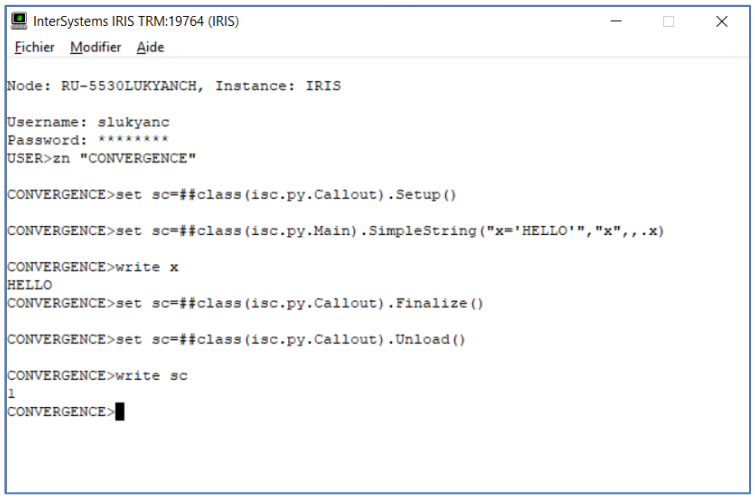
```
docker run -d \
  -p 52773:52773 \
  -v /<HOST-DIR-WITH-iris.key>/:/mount \
  --name irispy \
  intersystemsdc/irispy:latest \
  --key /mount/iris.key \
```

3. Test process `isc.py.test.Process` saves image artifact into `temp` directory. You might want to change that path to a mounted directory. To do that, edit annotation for `Correlation Matrix: Graph` call, specifying valid filepath for `f.savefig` function.
4. For terminal access execute: `docker exec -it irispy sh`.
5. Access SMP with SuperUser/SYS or Admin/SYS user/password.
6. To stop the container execute: `docker stop irispy && docker rm --force irispy`.

3.4. Use

3.4.1. General Use

1. Execute (once per system start) the following call: `set sc=##class(isc.py.Callout).Setup()` (add to `ZSTART`: docs, sample routine available in `rtn` folder)
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()`



3.4.2. Terminal API

1. Overall, `isc.py.Main` is the general interface to Python. It offers the following methods (all return `%Status`):

   Code execution (allow execution of arbitrary Python code):

   a. `ImportModule(module,.imported,.alias)` – import a `module` with an `alias`

   b. `SimpleString(code,returnVariable,serialization,.result)` – for cases where the code and the variable are both strings

   c. `ExecuteCode(code,variable)` – execute a `code` (a string or a stream), optionally set the result to a `variable`

   d. `ExecuteFunction(function, positionalArguments, keywordArguments, variable, serialization, .result)` - execute Python function or method, write result into Pyhton `variable`, return chosen serialization in `result`

   e. `ExecuteFunctionArgs(function, variable, serialization, .result, args...)` - execute Python function or method, write result into Pyhton `variable`, return chosen serialization in `result`. Builds `positionalArguments` and `keywordArguments` and passes them to `ExecuteFunction`. It is recommended to use `ExecuteFunction`. More information can be found in Gateway docs.

   Data transfer (transfer data into and from Python):

   Python -> InterSystems IRIS:

   f. `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`

   g. `GetVariableJson(variable,.stream,useString)` – get JSON serialization of a `variable`

   h. `GetVariablePickle(variable,.stream,useString,useDill)` – get Pickle (or Dill) serialization of a variable

   InterSystems IRIS -> Python:

   i. `ExecuteQuery(query,variable,type,namespace)` – create a resultset (of `pandas` dataframe or list `type`) from SQL `query` and set it to a `variable`. `isc.py` package must be available in `namespace`

   j. `ExecuteGlobal(global, variable, type, start, end, mask, labels, namespace)` - transfer global data (from start to end) to Python variable of type: list of tuples or `pandas` dataframe. For mask and labels arguments specification check class docs and Data Transfer docs

   k. `ExecuteClass(class, variable, type, start, end, properties, namespace)` - transfer class data to Python list of tuples or `pandas` dataframe. `properties` - comma-separated list of properties to form dataframe from. `*` and `?` wildcards are supported. Defaults to `*` (all properties). `%%CLASSNAME` property is ignored. Only stored properties can be used

l. `ExecuteTable(table, variable, type, start, end, properties, namespace)` - transfer table data to Python list of tuples or `pandas` dataframe

`ExecuteQuery` is universal (any valid SQL query would be transferred into Python). `ExecuteGlobal` and its wrappers `ExecuteClass` and `ExecuteTable`, however, operate with several limitations. But they are much faster (3-5 times faster than ODBC driver and 20 times faster than `ExecuteQuery`). More information in Data Transfer docs.

Auxiliary (support methods):

m. `GetVariableInfo(variable,serialization,.defined,.type,.length)` – get information on a `variable`: is it `defined`, `type` and `serialization` length

n. `GetVariableDefined(variable, .defined)` - is variable defined

o. `GetVariableType(variable, .type)` - get variable FQCN

p. `GetStatus()` – returns the last occurred exception in Python and clears it

q. `GetModuleInfo(module,.imported,.alias)` – get the `module alias` and its currently `imported` status

r. `GetFunctionInfo(function, .defined, .type, .docs, .signature, .arguments)` - get function information

s. Possible serialization functions:

   i. `##class(isc.py.Callout).SerializationStr` – a serialization by `str()` function if set to 1 (the default value is 0)

   ii. `##class(isc.py.Callout).SerializationRepr` – a serialization by `repr()` function if set to 1 (the default value is 1)

3.4.3. Shell

To open Python shell: `do ##class(isc.py.util.Shell).Shell()`. To exit press enter.

3.4.4. 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. Currently modules, functions and variables are saved.

3.4.5. 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. Transfer data into Python from an global/class/table with `isc.py.msg.GlobalRequest/isc.py.msg.ClassRequest/isc.py.msg.TableRequest` and get the response via `Ens.Response`
5. Save a Python context with `isc.py.msg.SaveRequest` and get the response via `Ens.StringResponse` with the context ID
6. Restore a Python context with `isc.py.msg.RestoreRequest`

Check request/response classes documentation for more details.

Settings:

- `Initializer` - select a class implementing `isc.py.init.Abstract`. It can be used to load functions, modules, classes and so on. It would be executed at process start
- `PythonLib` - (Linux only) if you see loading errors set it to `libpython3.6m.so` or even to a full path to the shared library

Note A: `isc.py.util.BPEmulator` class is added to allow easy testing of Python Interoperability business processes. It can execute business process (python parts) in a current job.

Note B: for debugging business operations, read this article.

Variable substitution

All business processes inheriting from `isc.py.ens.ProcessUtils` can use `GetAnnotation(name)` method to get the value of activity annotation by activity name. Activity annotation can contain variables which would be calculated on ObjectScript side before being passed to Python. This is the syntax for variable substitution:

- `${class:method:arg1:...:argN}` - execute method
- `#{expr}` - execute ObjectScript code

Check test `isc.py.test.Process` business process for example in `Correlation Matrix: Graph` activity:

```
f.savefig(r'#{process.WorkDirectory}SHOWCASE${%PopulateUtils:Integer:1:100}.png')
```

In this example:

- `#{process.WorkDirectory}` returns `WorkDirectory` property of process object which is an instance of `isc.py.test.Process` class and current business process

- `${%PopulateUtils:Integer:1:100}` calls `Integer` method of `%PopulateUtils` class passing arguments 1 and 100, returning random integer in range 1...100

3.4.6. Test Business Process

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 production, for 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)

3.4.7. Unit Tests

To run unit tests, execute:

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

3.4.8. ZPY Command

Install `ZLANG` routine from `rtn` folder (under your ML Toolkit installation set folder) to add `zpy` command:

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

```
zpy "x"
>0.4157151243124494
```

Argumentless `zpy` command opens Python shell.

3.4.9. 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: `zzz*`. Please report any leakage of `zzz*` variables. System code should always clear them.
3. Functions. Do not redefine these functions: `zzz*()`
4. Context persistence. Only pickled/dill variables can be restored correctly. Module imports are supported.

3.5. Development

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

3.5.1. Commits

Commits should follow the pattern: `module: description issue`. List of modules:

- Callout - C and ObjectScript callout interface in `isc.py.Callout`.
- API - terminal API, mainly `isc.py.Main`.
- Gateway - proxy classes generation.
- Proxyless Gateway - `isc.py.gw.DynamicObject` class.
- Interoperability - support utilities for Interoperability Business Processes.
- Tests - unit tests and test production.
- Docker - containers.
- Docs - documentation.

3.5.2. Building

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` or `mingw-w64.bat`)
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`

3.5.3. 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
    d. If you use PyEnv you need to install Python 3.6.7. with shared library support:

    ```
    env PYTHON_CONFIGURE_OPTS="--enable-shared" pyenv install 3.6.7
    pyenv global 3.6.7
    ```

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.Operation` 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
4. PyODBC on Linux and Mac
   a. Install unixodbc: `apt-get install unixodbc-dev`
   b. Install PyODBC: `pip install pyodbc`. To install on Linux, read the notes
   c. Set the connection string: `cnxn=pyodbc.connect(('Driver=/<IRIS directory>/bin/libirisodbcu35.so;Server=localhost;Port=51773;database=USER;UID=_SYSTEM;PWD=SYS'),autocommit=True)`

   Call `set sc = ##class(isc.py.util.Installer).ConfigureTestProcess(user, pass, host, port, namespace)` to configure test process automatically.

3.6. Gateway Functionality

3.6.1. Execute Function

Executes function by name. This API consists of two methods:

- `ExecuteFunction`
- `ExecuteFunctionArgs`

The difference between them is caller signature. `ExecuteFunction` accepts `%List`, `%Collection.AbstractArray` and JSON object separated into positional and keyword arguments. `ExecuteFunctionArgs` accepts `args...` and parses them into positional and keyword arguments. After that `ExecuteFunctionArgs` calls `ExecuteFunction`.

It is caller responsibility to escape argument values. Use `isc.py.util.Converter` class to escape:

- string
- boolean
- date
- time
- timestamp

ExecuteFunction:

`ExecuteFunction` method from `isc.py.Main` class. Signature:

- `function` - name of the function to invoke. Can be nested, i.e. `random.randint`
- `variable` - name of the python variable to write the result to.
- `positionalArguments` - positional arguments for Python function. Can be one of:
  - `$lb(val1, val2, ..., valN)`
  - `%Collection.AbstractIterator` object
  - JSON array
- `keywordArguments` - keyword arguments for Python function. Can be one of:
  - `$lb($lb(name1, val1), $lb(name2, val2), ..., $lb(nameN, valN))`
  - `%Collection.AbstractArray` object
  - flat JSON object
- `serialization` - how to serialize the result
- `result` - write the result into this variable

All arguments besides `function` are optional.

Here is an example of how it works:

```
set sc = ##class(isc.py.Main).ImportModule("random", ,.random)

set posList = $lb(1, 100)
set posCollection = ##class(%ListOfDataTypes).%New()
do posCollection.Insert(1)
do posCollection.Insert(100)
set posDynamic = [1, 100]
```

```
for positionalArguments = posList,posCollection,posDynamic {
    set sc = ##class(isc.py.Main).ExecuteFunction(random _
".randint", positionalArguments,,,.result)
    write result,!
}

set kwList = $lb($lb("a", 1), $lb("b", 100))
set kwCollection = ##class(%ArrayOfDataTypes).%New()
do kwCollection.SetAt(1, "a")
do kwCollection.SetAt(100, "b")
set kwDynamic = { "a": 1, "b": 100}

for kwArguments = kwList,kwCollection,kwDynamic {
    set sc = ##class(isc.py.Main).ExecuteFunction(random _
".randint", ,kwArguments,,.result)
    write result,!
}

set posList = $lb(1)
set kwDynamic = {"b": 100}
set sc = ##class(isc.py.Main).ExecuteFunction(random _ ".randint",
posList, kwDynamic,,.result)
write result,!

set posList =
##class(isc.py.util.Converter).EscapeStringList($lb("Positional:
{0} {1}! Keyword: {name}, {name2}", "Hello", "World"))
set kwDynamic =
{"name":(##class(isc.py.util.Converter).EscapeString("Alice")),

"name2":(##class(isc.py.util.Converter).EscapeString("Bob"))}
set sc = ##class(isc.py.Main).ExecuteFunction("str.format",
posList, kwDynamic,,.result)
write result,!
```

ExecuteFunctionArgs:

`ExecuteFunctionArgs` method from `isc.py.Main` class. Signature:

`function` - name of the function to invoke. Can be nested, i.e. `random.randint`

`variable` - name of the python variable to write the result to.

`serialization` - how to serialize the result

`result` - write the result into this variable

`args...` - function arguments.

`ExecuteFunctionArgs` attempts to determine correct positional and keyword arguments from the function signature (if available). It is recommended to call

`ExecuteFunction` directly if `ExecuteFunctionArgs` is unable to construct a correct argument spec (and opens an issue). Example:

```
set sc = ##class(isc.py.Main).ImportModule("random", ,.random)
set sc = ##class(isc.py.Main).ExecuteFunctionArgs(random _
".randint", , ,.result, 1, 100)
write result,!

set string =
##class(isc.py.util.Converter).EscapeString("Positional: {0}, {1},
{2}, {3}")
set arg1 = ##class(isc.py.util.Converter).EscapeString("Hello")
set arg2 = ##class(isc.py.util.Converter).EscapeString("World")
set arg3 = ##class(isc.py.util.Converter).EscapeString("Alice")
set arg4 = ##class(isc.py.util.Converter).EscapeString("Bob")
set sc =
##class(isc.py.Main).ExecuteFunctionArgs("str.format",,,.result,
string, arg1, arg2, arg3, arg4)
write result,!

set string =
##class(isc.py.util.Converter).EscapeString("Positional: {0} {1}!
Keyword: {name}, {name2}")
set arg1 = ##class(isc.py.util.Converter).EscapeString("Hello")
set arg2 = ##class(isc.py.util.Converter).EscapeString("World")
set kwargs = "**" _ {"name":"Alice","name2":"Bob"}.%ToJSON()
set sc = ##class(isc.py.Main).ExecuteFunctionArgs("str.format",,,
.result, string, arg1, arg2, kwargs)
write result,!
```

3.6.2. Proxyless Gateway

Proxyless gateway allows user to bind Python variables to InterSystems IRIS variables. This allows user to:

- Get/Set object properties
- Call object methods
- Serialize variable to: Str, Repr, Pickle, Dill, JSON, Dynamic Object.

Example.

1. Load Python class Person: `do ##class(isc.py.init.Test).Initialize(,1)`

Note: here is `Person` class definition for reference:

```
class Person(object):
    def __init__(self, name, age, city):
        self.name = name
        self.age = age
        self.city = city
    def getAge(self):
        return self.age
```

```
    def getAgePlus(self, add):
        return self.age + add
```

2. Create Proxy variable:

```
set obj = ##class(isc.py.gw.DynamicObject).%New("Person", "p1",
"'Ed'", "25", "'Test'"
```

In this call we create Python variable `p1` of `Person` class and pass three methods to constructor `'Ed'`, `25` and `'Test'`.

3. Now we can interact with the object, let us get and set some properties:

```
write obj.name
set obj.name="Bob"
write obj.name
write obj.age
```

4. We can set some new properties too (unlike `ExecuteFunction` values are escaped automatically if `%EscapeOnSet` property is 1, which is default. You can also set properties to other dynamic objects. In that case the unescaped Python variable name would be used):

```
set obj.pet = "Dog"
write obj.pet
```

5. And we can call object methods:

```
write obj.getAge()
write obj.getAgePlus(10)
```

6. Finally we can convert an object:

```
set sc = obj.%ToJSON(.json)
set sc = obj.%ToDynObj(.dynObj)
set sc = obj.%ToPickle(.pickle)
set sc = obj.%ToStream(.stream)
```

To create a proxy object from an existing proxy object just skip the type argument:

```
kill obj
set p1 = ##class(isc.py.gw.DynamicObject).%New(, "p1")
```

Module objects can be proxied this way too:

```
set module = "random"
set sc = ##class(isc.py.Main).ImportModule(module)
set random = ##class(isc.py.gw.DynamicObject).%New(,module)
write random.randint(1,100)
```

Now for a more complex example. In case of primitives (int, bool, str, float) a proxy object returns a serialized value. Otherwise (if a method call or a variable get returns a complex type) it returns another proxy object pointing to that result.

```
set sc = ##class(isc.py.Main).ImportModule("numpy",,"np")
set np = ##class(isc.py.gw.DynamicObject).%New(,"np")
set arr = ##class(isc.py.gw.DynamicObject).%New("np.array",
"arr","[[1.5,2],[4,5]]")
set exp = np.exp(arr)
write $replace(exp.%GetString(),$c(10), $c(13,10))
```

And here is an example of setting a property to a proxy object:

```
do ##class(isc.py.init.Test).Initialize(,1)
set obj = ##class(isc.py.gw.DynamicObject).%New("Person", "p1", "'Ed'", "25", "'Test'")
set obj2 = ##class(isc.py.gw.DynamicObject).%New("Person", "p2", "'Bob'", "22", "'Test2'")
write obj.%GetJSON()

set obj.relative = obj2
set obj3 = obj.relative
write obj3.%GetJSON()
```

You can use `%EscapeOnSet` and `%EscapeOnCall` properties and `%IsPrimitive` method to affect default serialization behavior.

3.7. Data Transfer

Transfer data into and from Python. All the methods are defined in `isc.py.Main`. All methods return `%Status`.

3.7.1. Python -> InterSystems IRIS

- `GetVariable(variable, serialization, .stream, useString)` - get serialization of variable in stream. If `useString` is 1 and variable serialization can fit into string, then string is returned instead of the stream
- `GetVariableJson(variable, .stream, useString)` - get JSON serialization of variable
- `GetVariablePickle(variable, .stream, useString, useDill)` - get Pickle (or Dill) serialization of variable

3.7.2. InterSystems IRIS -> Python

Load data from InterSystems IRIS to Python. All these methods support data transfer from any local namespace. `isc.py` package must be available in your working namespace.

ExecuteQuery

`ExecuteQuery(query, variable, type, namespace)` - transfer results from any valid SQL query into Python. It is the slowest method of data transfer. Use it if `ExecuteGlobal` and its wrappers are unavailable.

Arguments:

- `query` - sql query
- `variable` - target variable on a Python side
- `type` - list or Pandas dataframe

ExecuteGlobal

`ExecuteGlobal(global, variable, type, start, end, mask, labels, namespace)` - transfer global data to Python.

Arguments:

- `global` - global name without ^
- `variable` - target variable on a Python side
- `type` - list or Pandas dataframe
- `start` - initial global key. Must be integer.

- `end` - final global key. Must be integer.
- `mask` - string, mask for global values. Mask may be shorter than the number of global value fields (in this case fields at the end would be skipped). How to format mask:
  - `+` use field as is
  - `-` skip field
  - `b` - boolean (0 - False, anything else - True)
  - `d` - date (from `$horolog`, on Windows only from 1970, on Linux from 1900 see notes for details)
  - `t` - time (`$horolog`, seconds since midnight)
  - `m` - (moment) timestamp string in YEAR-MONTH-DAY HOUR:MINUTE:SECOND format.
- `labels` - `%List` of column names, first element is key column name. Therefore: list length must be mask symbol length + 1

ExecuteClass

Wrapper for `ExecuteGlobal`. Effectively it parses compiled class definition, constructs `ExecuteGlobal` arguments and calls it.

`ExecuteClass(class, variable, type, start, end, properties, namespace)` - transfer class data to Python list of tuples or pandas dataframe. `properties` - comma-separated list of properties to form dataframe from. `*` and `?` wildcards are supported. Defaults to `*` (all properties). `%%CLASSNAME` property is ignored. Only stored properties can be used.

Arguments:

- `class` - class name
- `variable` - target variable on a Python side
- `type` - list or Pandas dataframe
- `start` - initial object id. Must be integer.
- `end` - final object id. Must be integer.
- `properties` - comma-separated list of properties to form dataframe from. `*` and `?` wildcards are supported. Defaults to `*` (all properties). `%%CLASSNAME` property is ignored. Only stored properties can be used.

All properties transferred as is except properties of `%Date`, `%Time`, `%Boolean` and `%TimeStamp` types. They are converted to respective Python datatypes.

ExecuteTable

Wrapper for `ExecuteClass`. Translates table name to class name and calls `ExecuteClass`. Signature:

`ExecuteTable(table, variable, type, start, end, properties, namespace)` - transfer table data to Python list of tuples or pandas dataframe.

Arguments:

- `table` - table name.

Other arguments are passed as is to `ExecuteClass`.

Notes

- `ExecuteGlobal`, `ExecuteClass` and `ExecuteTable` generally offer the same speed (as the time to parse class definition is negligible)

- `ExecuteGlobal` is 3-5 times faster than ODBC driver and up to 20 times faster than `ExecuteQuery` on measurable workloads (>0.01 second)
- `ExecuteGlobal`, `ExecuteClass` and `ExecuteTable` only work on the globals with this structure: `^global(key) = $lb(prop1, prop2, ..., propN)` where `key` must be an integer
- For `ExecuteGlobal`, `ExecuteClass` and `ExecuteTable` supported `%Date` range equals `mktime` range (Windows: 1970-01-01, Linux: 1900-01-01, Mac). Use `%TimeStamp` to transfer dates outside of this range
- For `ExecuteGlobal`, `ExecuteClass` and `ExecuteTable` all arguments besides source (global, class, table) and variable are optional

Examples

Let's say we have `isc.py.test.Person` class. Here's how we can use all methods of data transfer:

```
// All the ways to transfer data
set global = "isc.py.test.PersonD"
set class = "isc.py.test.Person"
set table = "isc_py_test.Person"
set query = "SELECT * FROM isc_py_test.Person"

// Common arguments
set variable = "df"
set type = "dataframe"
set start = 1
set end = $g(^isc.py.test.PersonD, start)

// Approach 0: ExecuteGlobal without arguments
set sc = ##class(isc.py.Main).ExecuteGlobal(global, variable _ 0,
type)

// Approach 1: ExecuteGlobal with arguments
// For global transfer labels are not calculated automatically
// globalKey - is global subscript
set labels = $lb("globalKey", "Name", "DOB", "TS", "RandomTime",
"AgeYears", "AgeDecimal", "AgeDouble", "Bool")

// mask is 1 element shorter than labels because "globalKey" is
global subscript label
// Here we want to skip %%CLASSNAME field
set mask = "-+dmt+++b"

set sc = ##class(isc.py.Main).ExecuteGlobal(global, variable _ 1,
type, start, end, mask, labels)

// Approach 2: ExecuteClass
```

```
set sc = ##class(isc.py.Main).ExecuteClass(class, variable _ 2,
type, start, end)

// Approach 3: ExecuteTable
set sc = ##class(isc.py.Main).ExecuteTable(table, variable _ 3,
type, start, end)

// Approach 4: ExecuteQuery
set sc = ##class(isc.py.Main).ExecuteQuery(query, variable _ 4,
type)
```

You can call this method: `do ##class(isc.py.test.Person).Test()` to check how these data transfer methods work.

4. Continuous Integration (CI): R Tools

4.1. R Gateway

R Gateway for InterSystems data platform. Execute R code and more from InterSystems IRIS. This toolset brings you the power of R right into your InterSystems IRIS environment:

- Execute arbitrary R code
- Seamlessly transfer data from InterSystems IRIS into R or from R to InterSystems IRIS
- Build intelligent Interoperability business processes with R Interoperability Adapter

4.2. Installation

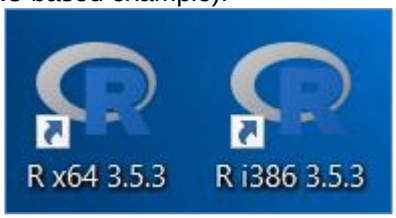
1. Install IRIS.
2. Install R and its modules, prepare the environment

a. Install R

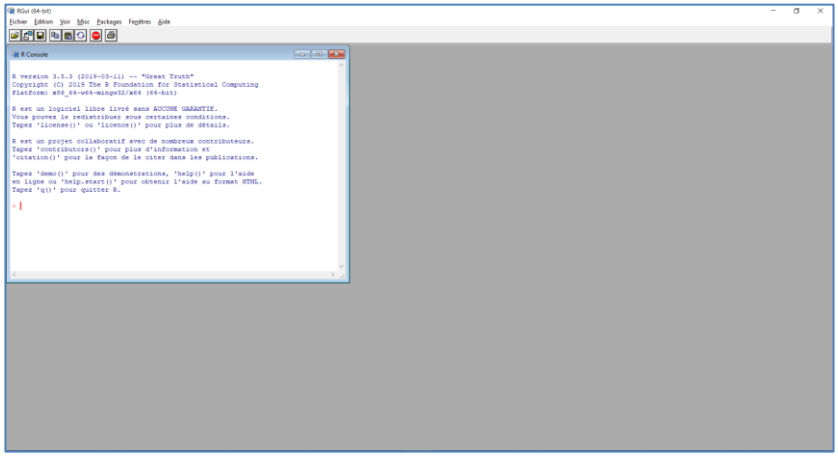
i. Download R, from the download page. Select the installer that matches your OS (for example: Windows). By default, Windows installer will install both 32 and 64-bit executables/dlls – we recommend accepting the default installation.

ii. Install R into a default directory (`C:\Program Files\R\R-#.#.#` on Windows)

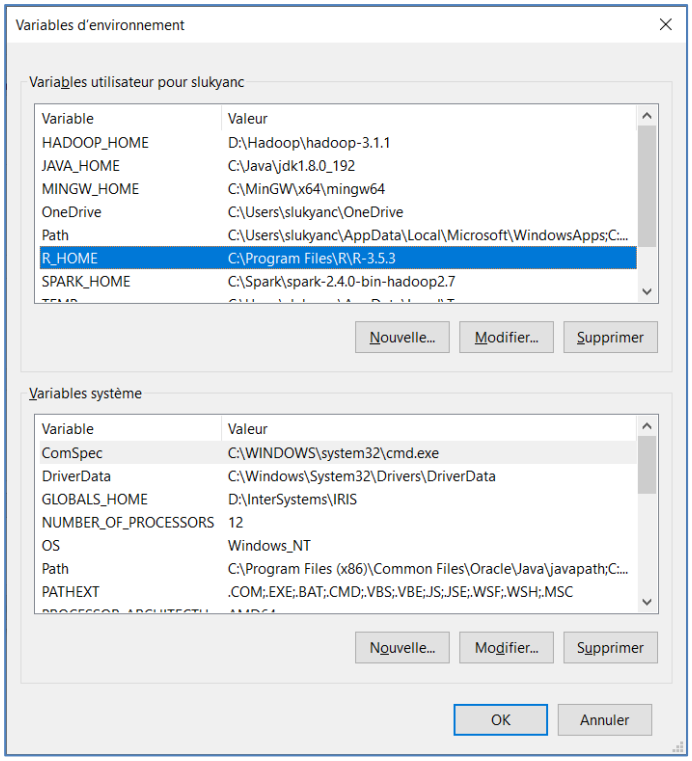
iii. After the installation, two icons like the ones below appear on your desktop (a Windows-based example):



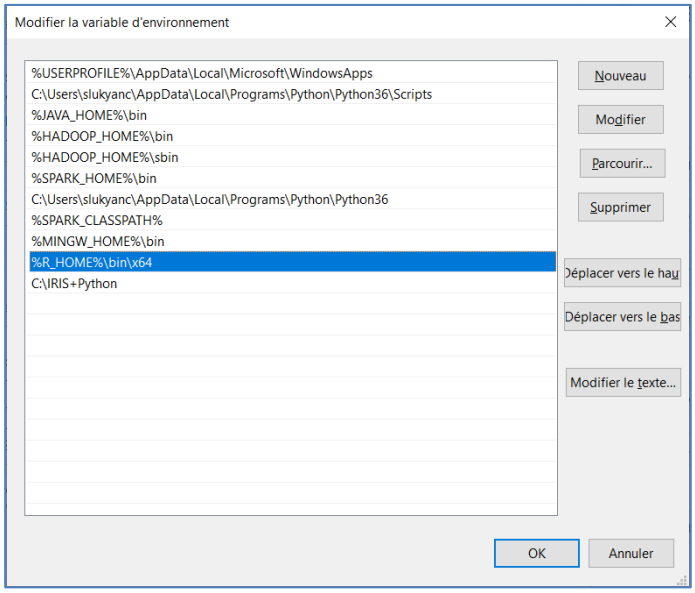
iv. If you double-click the one that corresponds to 64 bits, the following window opens (you can leave it by typing `q()` and then pressing Enter):



v. Check that your `R_HOME` system environment variable points to the folder where your R was installed (for example: `C:\Program Files\R\R-#.#.#`):



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

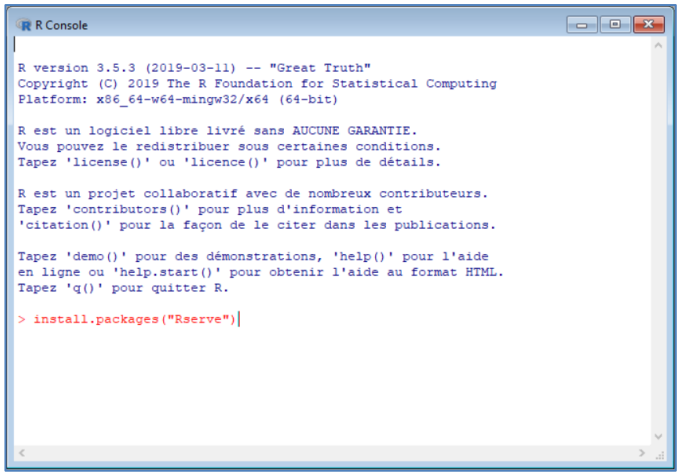


If you are on Linux or Mac, check that your `PATH` system environment variable includes `/usr/lib` and `/usr/lib/x86_64-linux-gnu`. Use `/etc/environment` file to set the environment variables.

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

b. Install R packages

i. Start R as administrator (right-click on the R icon in Windows and select running it as administrator) and install one by one the following R packages using `install.packages("<module name>")` command (you need to be connected to the Internet while doing this):



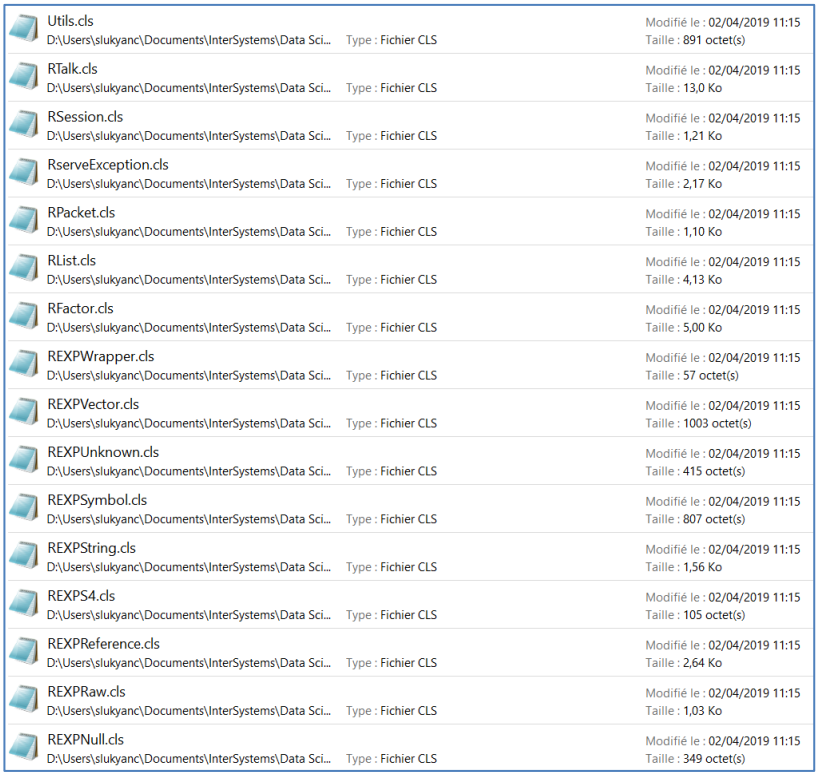
- `Rserve`
- `igraph`

ii. Before proceeding to a module download, R will prompt you to a download server selection list.

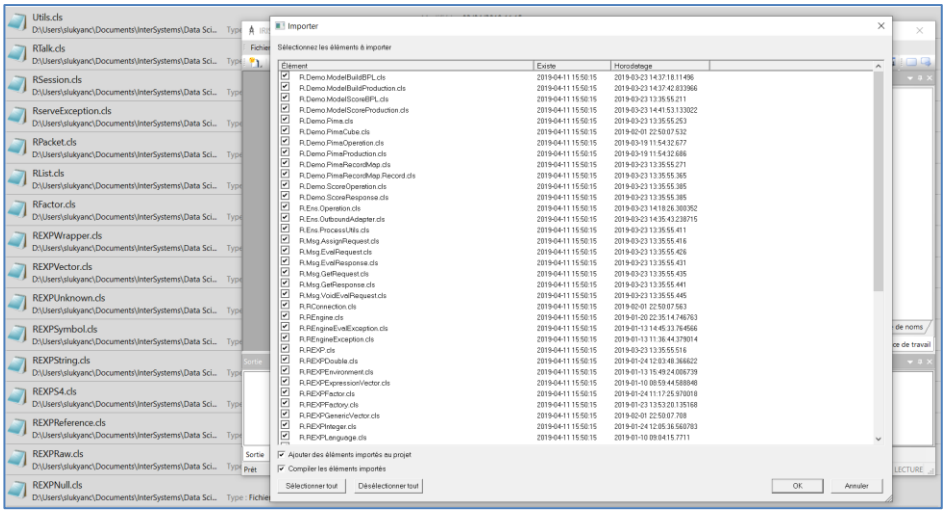
iii. After the download server has been selected, the module download starts.

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:

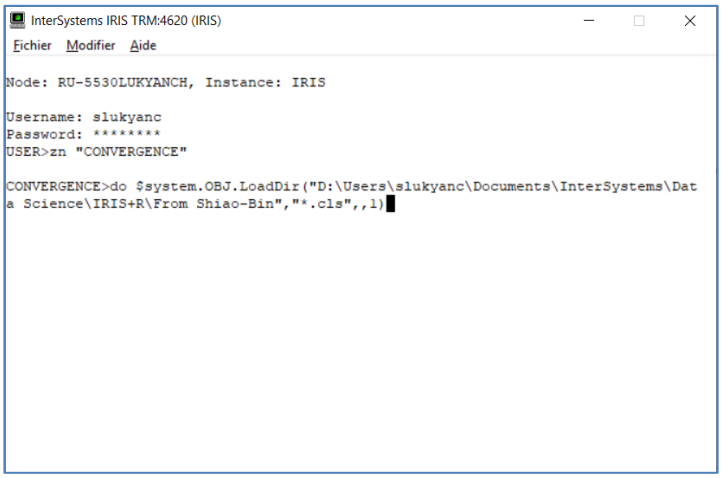


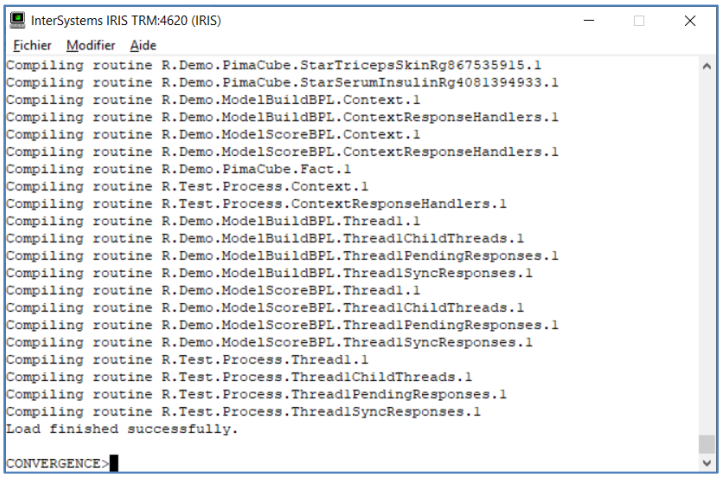
b. select the files found by the file search above, drag and drop them into your IRIS Studio:



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)`



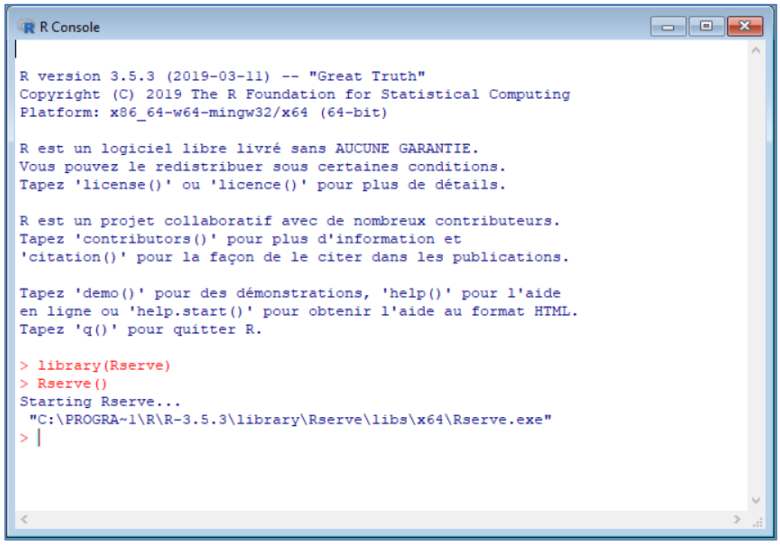


4.3. Use

4.3.1. General Use

1. Before starting the use of R Gateway, start the Rserve component by typing in R: `library(Rserve)`

```
Rserve()
```



2. Class `R.Ens.Operation` is the general interface to R. It offers the following methods (all return `%Status`):

a. `Assign(request,.response)` – assigns values to a named variable in R

b. `Eval(request,.response)` – execute an R code and evaluate in IRIS-mappaple terms code's named variables

c. `VoidEval(request,.response)` – execute an R code without evaluating in IRIS-mappaple terms code's named variables

d. `Get(request,.response)` – retrieves values from a named variable in R

e. `AssignSQL(request, .response)` – execute an SQL statement locally and assign the `resultset` as dataframe to a named variable in R

4.3.2. IRIS Interoperability Adapter

IRIS Interoperability adapter `R.Ens.Operation` enables interaction with an R process from Interoperability productions. The following requests are currently supported:

1. Assign values to a named variable in R via `R.Msg.AssignRequest`
2. Execute an R code and evaluate in IRIS-mappaple terms code's named variables via `R.Msg.EvalRequest` and get the response via `R.Msg.EvalResponse`
3. Execute an R code without evaluating in IRIS-mappaple terms code's named variables with `R.Msg.VoidEvalRequest`
4. Retrieve values from a named variable in R with `R.Msg.GetRequest` and get the response via `R.Msg.GetResponse`
5. Execute an SQL statement and assign its resultset as dataframe to a named variable in R via `R.Msg.AssignSQLRequest`

The above requests contain a serial object `R.Msg.RData`. This class maps all common R data types to IRIS Interoperability requests so that R data elements can show up in message traces. It also provides some convenient methods to work with data stored in globals. The details are documented in class `R.Msg.RData`.

Check respective request/response classes documentation for more details.

4.3.3. Notes

- If you want to use ODBC connection, on Windows install package `RODBC` or packages `DBI` and `odbc`
- If you want to use JDBC connection, install `RJDBC`

4.3.4. Sample Business Process and Production

There are two sample productions in the demo package `R.Demo`: `ModelBuildProduction` is used to train, test and save a model which predicts the likelihood of a patient being diagnosed with diabetes. `ModelScoreProduction` shows how to make predictions for new patients using the previously saved model.

To build a predictive model:

1. Run the following command to populate the sample dataset:
   ```
   do ##class(R.Demo.Pima).Import()
   ```
2. Configure and start production `ModelBuildProduction`.
3. Send an empty `Ens.Request` message to `ModelBuildBPL` process.

To make new predictions:

1. Configure and start production `ModelScoreProduction`.
2. Copy sample file `pima-new-dataset.csv` to the file path of `Enslib.RecordMap.Service.FileService`.
3. If the likelihood of a patient being diagnosed with diabetes is more than 50%, an alert will be sent.

Note that on Windows machines, each instance of `Rserve` can only support one R Gateway connection. Be sure to stop the current running production before start a new one.

4.3.5. Unit Tests

All test code is under `R.Test` package. `R.Test.API` contains the tests for the underlying API. `R.Test.Production` and `R.Test.Process` cover the usage of `R.Ens.Operation` and `R.Msg` messages.

4.3.6. Limitations

1. It is known that each instance of `Rserve` on Windows only supports one R Gateway connection. To support multiple connections, you can start multiple instance of `Rserve`, each with a different port.
2. `Rserve` is thread-safe across connections, but `*eval*` methods are not thread-safe within one connection. This means that multiple threads should not use the same connection unless they guarantee that no `*eval*` calls are run in parallel.

4.3.7. Rserve Configuration

The default `Rserve` server port is `6311`. It can be started with a different port through a configuration file. On Linux or Mac computer, the configuration file is at `/etc/Rserve.conf`. On Windows machine, it can be specified by `-RS-conf` option during start up. For example, `Rserve(args="--RS-conf C:\\InterSystems\\Rserve.conf")`.

All supported configuration directives are listed below:

- `log.io` (boolean) if enabled the content of I/O messages is logged in debug mode (has no effect in normal mode)
- `daemon` (boolean) if enabled `Rserve` will daemonize
- `close.all.stdio` (boolean) if enabled, `stdout/stderr` are closed and re-directed to `/dev/null`
- `msg.id` (boolean) if enabled `Rserve` uses message ID in the headers
- `remote` (boolean) if enabled servers listen on all external interfaces so they can be used over the network, otherwise just loopback is bound
- `tag.argv` (boolean) if enabled the program name is changed after forking to distinguish the server instance form working sessions
- `forward.stdio` (boolean) if enabled stdout/stderr is forwarded using OOB send
- `ulog` string, defining destination for logging. Can be either a part to a local socket or either of `tcp://<host>[:port]` or `udp://<host>:[port]` to send logging to an external server (if port is not specified `514` is used). The protocol is compatible with syslogd
- `keep.alive` (boolean) if enabled `SO_KEEPALIVE` is enabled on the socket
- `switch.qap.tls` (boolean) if enabled `TLS` switch is allowed for QAP connections
- `qap.oc` (boolean, alternate name `rserve.oc`) if enabled Object-Capability mode of the QAP protocol is used
- `console.oob` (boolean) if enabled R console I/O generates OOB send messages (OOB must be enabled for this to have effect)
- `console.input` (boolean) if enabled R `ReadConsole` API invokes OOB message to query the client for input (has no effect unless `console.oob` is also enabled)
- `websockets.qap.oc` (boolean) if enabled the WebSockets QAP server uses Object-Capability mode
- `random.uid` (boolean) if enabled random `uid` is used when switching `uid` after connect
- `random.gid` (boolean) if enabled random gid is used when switching gid after connect
- `random.uid.range` (string of the form xxx-yyy) range for randomly generated `uid`
- `auto.uid` (boolean) if enabled `uid` is determined from the password file on authentication
- `auto.gid` (boolean) if enabled `gid` is determined from the password file on authentication
- `default.uid` (integer) `uid` to use as fallback if it cannot be determined during authentication
- `default.gid` (integer) `gid` to use as fallback if it cannot be determined during authentication
- `oob.idle.interval` (integer) interval in seconds after which an "idle" OOB send packet is sent to the client (mostly to prevent proxies from dropping the connection)
- `qap.port` (integer, alternative name port) port to use by the QAP server
- `ipv6` (boolean) if enabled servers listen on IPv6
- `http.upgrade.websockets` (boolean) if enabled HTTP server allows upgrade to the WebSockets protocol on the same connection

- `http.raw.body` (boolean) if enabled HTTP callback send raw (unparsed) body
- `websockets.port` (integer) port to use for the WebSockets server
- `http.port` (integer) port to use for the HTTP server
- `tls.key` (path) path to the file containing the RSA key to be used for TLS
- `tls.ca` (path) path to the file (in PEM format) containing Certificate Authority certificates the be registered with TLS
- `tls.cert` (path) path to the file (in PEM format) containing the certificate to use for TLS servers
- `pid.file` (path) path to the file that will hold the PID of the `Rserve` server process once started
- `rsa.key` (path) path to the RSA key to use for RSA authentication
- `qap.tls.port` (integer, alternative name `tls.port`) port of the secure server running QAP wrapped in TLS
- `http.tls.port` (integer, alternative name `https.port`) port of the secure server running HTTP wrapped in TLS
- `websockets.tls.port` (integer) port of the secure server running WebSockets wrapped in TLS
- `qap` (boolean, alternative name `rserve`) if enabled QAP server is started
- `websockets.qap` (boolean) if enabled WebSockets mode with QAP protocol is started
- `websockets.text` (boolean) if enabled WebSockets mode with text protocol is started (deprecated and discouraged)
- `websockets` (boolean) if enabled all WebSocket modes are enabled
- `maxinbuf` (integer) limit for the size of incoming packets in kB
- `source` (path) path to a file to use via `source()` prior to starting the servers
- `eval` (string) the string will be parsed and evaluated in the global environment prior to starting the servers
- `maxsendbuf` (integer) limit for the size of the output buffer in kB
- `su` (string, one of `now`, `server` or client) determines the time at which a user switch is performed
- `http.user` (string) username to switch to when running the HTTP server
- `https.user` (string) username to switch to when running the HTTP/TLS server
- `websockets.user` (string) username to switch to when running the WebSockets server
- `uid` (integer/oct) user ID to switch to
- `gid` (integer/oct) group ID to switch to
- `chroot` (path) path to use as a chroot jail
- `umask` (integer/oct) umask to use
- `allow` (string) IP address to add to the white-list (only IPv4 is supported at this point)
- `control` (boolean) if enabled control commands (`server.eval`, `server.source`, `server.shutdown`) are allowed

- `shutdown` (boolean) if enabled `CMD_shutdown` is allowed (it does not affect shutdown via `SIGINT`)
- `workdir` (path) path to the working directory used as root for per-connection directories
- `workdir.clean` (boolean) if enabled the working directory for a closed connection is removed upon exit even if it is dirty
- `workdir.mode` (integer/oct) mode (UNIX permissions) for the per-connection working directories
- `workdir.parent.mode` (integer/oct) more for the root of all per-connection working directories
- `encoding` (string) string encoding to use in the protocol. It must be an encoding recognized by R `iconv` facilities
- `socket` (path) path to the local socket used for QAP
- `sockmod` (integer/oct) mode for the local socket
- `pwdfile` (path) path to the password file. The file is expected to contain one user/password pair per line, separated by a whitespace
- `auth.function` (string) name of an R function to use for authentication instead of the built-in `Rserve` facilities
- `auth` (boolean or require - only first character is checked) if enabled authentication is required before any other command can be used
- `interactive` (boolean) if enabled R is run in interactive mode, otherwise not
- `plaintext` (boolean) if enabled plain-text authentication is allowed
- `oob` (boolean) if enabled out-of-band (OOB) messages (send and msg) can be used by the R code run in the session
- `fileio` (boolean) if enabled file I/O QAP commands are allowed
- `r.control` (boolean, alternative name r-control) if enabled the R session can use self-command to issue control commands (as opposed to control commands initiated by the client)
- `cachepwd` (boolean or indefinitely) controls caching of passwords. If enabled the passwords file is read on connection (before `su` is executed), if disabled it is read at the time of authentication and if indefinitely then it only read at the time of server start

4.3.8. Development

Development of ObjectScript code can be done with IRIS Studio, or Eclipse. R scripts development and test are done with R Studio and R Terminal.

4.3.9. Troubleshooting

1. Takes a long time to make an R Gateway connection or it simply hangs while connecting: a connection may be still active on the specified port. Just kill the `Rserve` process associated with the port.

4.3.10. Gateway Functionality

The public interfaces of R Gateway are all under class `R.RConnection`. `R.Ens.OutboundAdapter` and `R.Ens.Operation` encapsulate the API details in the context of Ensemble production.

- To create an R Gateway connection:

```
set c = ##class(R.RConnection).%New(host, port)
```

- To assign a double value to a R variable:

```
set x = ##class(R.REXPDouble).%New(3.0)
do c.assign("x", x)
```

- To evaluate R script:

```
do c.eval("y<-sqrt(x)")
```

- To retrieve the value of an R variable:

```
set y = c.get("y")
```

The API do not return a status code. It is advised to wrap all ObjectScript code in a try/catch block. Please refer to `R.RConnection` class documentation for details. More examples can be found in `R.Test.API`.

`R.REXP` is the super class of all supported R data types. It contains many useful utility methods:

- `getJSON()` converts the R data type to an IRIS `%DynamicObject`
- `toString()` is the shortened string representation of the R data type
- `toDebugString()` is the string representation of the underlying `%DynamicObject`
- `asDoubleMatrix()` converts the internal matrix data to an IRIS multidimensional array
- `createDoubleMatrix()` takes a multidimensional array and converts it to an R matrix object
- `createDataFrame()` creates an internal R data frame object
- `createDataFrameFromSQL()` executes the SQL statement and converts the result set to an R data frame object

There are more generic utility methods in R.Utils class.

To simulate an R terminal from an IRIS terminal: `do ##class(R.Utils).RTerminal()`.

5. Continuous Integration (CI): Julia Tools

5.1. Julia Gateway

Interface to Julia programming language for InterSystems IRIS. Execute Julia code and more from InterSystems IRIS. This projects brings you the power of Julia right into your InterSystems IRIS environment:

- Execute arbitrary Julia code
- Seamlessly transfer data from InterSystems IRIS into Julia
- Build intelligent Interoperability business processes with Julia Interoperability Adapter

5.2. Installation

1. Install Julia 1.4.0 64 bit (other Julia versions are untested). Follow the OS-specific instructions for installing Julia (Windows, Linux, Mac).
2. Download latest Julia Gateway release and unpack it.
3. From the InterSystems IRIS terminal, load Julia Gateway code. To do that, execute: `do $system.OBJ.ImportDir("/path/to/unpacked/juliagateway","*.cls","c",,1))` in your (Ensemble-enabled) namespace. In case you want to production-enable namespace, call: `write ##class(%EnsembleMgr).EnableNamespace($Namespace, 1).`
4. Place callout DLL/SO/DYLIB in the `bin` folder of your InterSystems IRIS installation. The library file should be placed into a path returned by `write ##class(isc.julia.Callout).GetLib()`

5.2.1. Windows

1. Check that your JULIA_HOME environment variable points to Julia 1.4.0.
2. Check that your SYSTEM PATH environment variable has:

   `%JULIA_HOME%\bin` variable (or directory it points to)
3. Restart InterSystems IRIS.
4. In the InterSystems IRIS Terminal, run:
   - `write $SYSTEM.Util.GetEnviron("JULIA_HOME")` and verify it prints out the directory of Julia installation
   - `write $SYSTEM.Util.GetEnviron("PATH")` and verify it prints out the bin directory of Julia installation

5.2.2. Linux and Mac

1. Set `LibPath` configuration parameter to the value of `$JULIA_HOME/lib` (if you installed to `/tmp/julia`, set `LibPath=/tmp/julia/lib`).
2. Restart InterSystems IRIS.

5.2.3. Post-Installation (Windows, Mac, Linux)

After the installation you will need these packages. In OS bash, run:

```
import Pkg;
Pkg.add(["JSON", "CSV", "DataFrames", "ROCAnalysis", "Plots"])
```

```
using JSON, CSV, DataFrames, ROCAnalysis, Plots
```

5.2.4. Docker

1. To build a docker image:

- Copy `iscjulia.so` into repository root (if it is not there already)
- Execute in the repository root `docker build --force-rm --tag intersystemsdc/irisjulia:latest`. By default, the image is built upon `store/intersystems/iris-community:2019.4.0.383.0` image, however you can change that by providing `IMAGE` variable. To build from InterSystems IRIS, execute: `docker build --build-arg IMAGE=store/intersystems/iris:2019.4.0.383.0 --force-rm --tag intersystemsdc/irisjulia:latest`

2. To run the docker image execute (key is not needed for community images):

```
docker run -d \
 -p 52773:52773 \
 -v /<HOST-DIR-WITH-iris.key>/:/mount \
 --name iris \
 intersystemsdc/irisjulia:latest \
 --key /mount/iris.key \
```

3. For terminal access execute: `docker exec -it iris iris session iris`.
4. Access SMP with SuperUser/SYS or Admin/SYS user/password.
5. To stop container, execute: `docker stop iris && docker rm --force iris`

5.3. Use

1. Call: `set sc = ##class(isc.julia.Callout).Setup()` once per systems start (add to `ZSTART`: docs, sample routine available in `rtn` folder).
2. Initialize Julia once per process start: `set sc = ##class(isc.julia.Callout).Initialize()`
3. Call the main method (can be called many times, context persists): `write ##class(isc.julia.Main).SimpleString(code, .result)`
4. Call: `set sc = ##class(isc.julia.Callout).Finalize()` to free Julia context.
5. Call: `set sc = ##class(isc.julia.Callout).Unload()` to free the callout library.

```
set sc = ##class(isc.julia.Callout).Setup()
set sc = ##class(isc.julia.Callout).Initialize()
set sc = ##class(isc.julia.Main).SimpleString("sqrt(4)", .result)
write result
set sc = ##class(isc.julia.Callout).Finalize()
set sc = ##class(isc.julia.Callout).Unload()
```

5.4. Terminal API

Generally the main interface to Julia is `isc.julia.Main`. It offers these methods (all return `%Status`), which can be separated into three categories:

- Code execution
- Data transfer
- Auxiliary

5.4.1. Code Execution

These methods allow execution of arbitrary Julia code:

- `ImportModule(module)` - import module
- `SimpleString(code, .result)` - execute `code` for cases where both `code` and `result` are less than `$$$MaxStringLength` in length
- `ExecuteCode(code, variable, .result)` - execute `code` (it may be a stream or string), optionally set code into `variable`

5.4.2. Data Transfer

Transfer data into and from Julia.

`GetVariable(variable, serialization, .stream, useString)` - get `serialization` of `variable` in `stream`. If `useString` is 1 and variable serialization can fit into string then string is returned instead of the stream.

InterSystems IRIS -> Julia

`ExecuteQuery(query, variable, type, namespace)` - create `DataFrame` from sql `query` and set it into `variable`. `isc.julia` package must be available in `namespace` (the available `type` is `DataFrame`).

5.4.3. Auxiliary

Support methods:

- `GetVariableInfo(variable, serialization, .defined, .type, .length)` - get info about variable: is it defined, type and serialized length
- `GetVariableDefined(variable, .defined)` - is variable defined
- `GetVariableType(variable, .type)` - get variable fully qualified class name

Possible serializations:

- `string` - serialization by `string()` function
- `json` - serialization by JSON module

5.5. Shell

To open Julia shell: `do ##class(isc.julia.util.Shell).Shell()`. To exit, press Enter. In `rtn` folder `zj` command is also available. Import it into `%SYS` namespace.

5.6. Interoperability Adapter

Interoperability adapter `isc.julia.ens.Operation` offers ability to interact with Julia process from Interoperability productions. Currently three requests are supported:

- Execute Julia code via `isc.julia.msg.ExecutionRequest`. Returns `isc.julia.msg.ExecutionResponse` with requested variable values
- Execute Julia code via `isc.julia.msg.StreamExecutionRequest`. Returns `isc.julia.msg.StreamExecutionResponse` with requested variable values. Same as above but accepts and returns streams instead of strings

- Create dataframe from an SQL query with `isc.julia.msg.QueryRequest`. Returns `Ens.Response`

Check request/response classes documentation for details.

Settings:

- `Initializer` - select a class implementing `isc.julia.init.Abstract`. It can be used to load functions, modules, classes and so on. It would be executed at process start.

5.7. Test Business Process

1. Execute in OS bash:
   `julia <repo-dir>\install.jl`
2. In InterSystems IRIS terminal execute: `write ##class(isc.julia.test.AMES).Import()` to load the dataset
3. Start `isc.julia.test.Production` production
4. Send empty `Ens.Request` message to `isc.julia.test.Process`

5.8. Variable Substitution

All business processes inheriting from `isc.julia.ens.ProcessUtils` can use `GetAnnotation(name)` method to get value of activity annotation by activity name. Activity annotation can contain variables which would be calculated on ObjectScript side before being passed to Julia. This is the syntax for variable substitution:

- `${class:method:arg1:...:argN}` - execute method
- `#{expr}` - execute ObjectScript code

Example:

```
save(r'#{process.WorkDirectory}SHOWCASE${%PopulateUtils:Integer:1:100}.png')
```

In this example:

- `#{process.WorkDirectory}` returns `WorkDirectory` property of `process` object which is an instance of the current business process
- `${%PopulateUtils:Integer:1:100}` calls `Integer` method of `%PopulateUtils` class passing arguments `1` and `100`, returning random integer in range `1...100`

5.9. Unit Tests

To run tests, execute:

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

5.10. ZLANGC00

Install `ZLANG` routine from `rtn` folder to add `zj` command:

```
zj "sqrt(2)"
zj
```

Argumentless `zj` command opens Julia shell.

5.11. Limitations

There are several limitations associated with the use of Julia Gateway.

1. `Pkg` is not supported on Windows (if used via Julia Gateway).
2. Variables: do not use these variables: `zzz*` variables. Please report any leakage of these variables. System code should always clear them.
3. Functions: do not redefine `zzz*()` functions.

5.12. Development

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

5.13. Commits

Commits should follow the pattern: `module: description issue`. List of modules:

- Callout - C and ObjectScript callout interface in `isc.julia.Callout`
- API - terminal API, mainly `isc.julia.Main`
- Interoperability - support utilities for Interoperability business processes
- Tests - unit tests and test production
- Docker - containers
- Docs - documentation

5.14. Building

5.14.1. Windows

1. Install MinGW-w64: you'll need `make` and `gcc`
2. Rename `mingw32-make.exe` to `make.exe` in `mingw64\bin` directory
3. Set `GLOBALS_HOME` environment variable to the root of InterSystems IRIS installation
4. Set `JULIA_HOME` environment variable to the root of Julia installation
5. Open MinGW bash (`mingw64env.cmd` or `mingw-w64.bat`)
6. In `<Repository>\c\` execute `make`

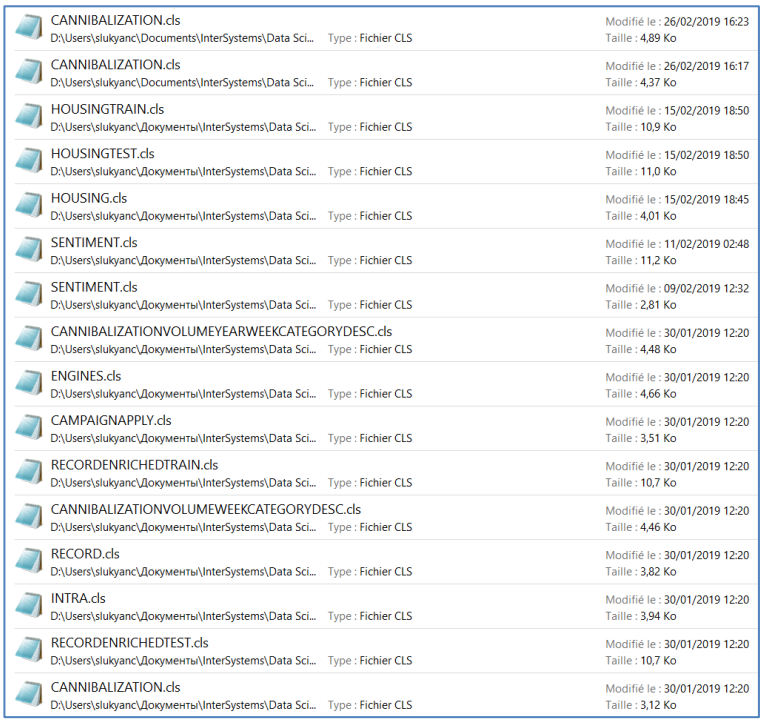
5.14.2. Linux/Mac

1. Install Julia.
2. Install: `apt install build-essential` (for Mac install `gcc` compiler and `make`)
3. Set `GLOBALS_HOME` environment variable to the root of InterSystems IRIS installation
4. Set `JULIA_HOME` environment variable to the root of Julia installation
5. In `<Repository>/c/` execute `make`

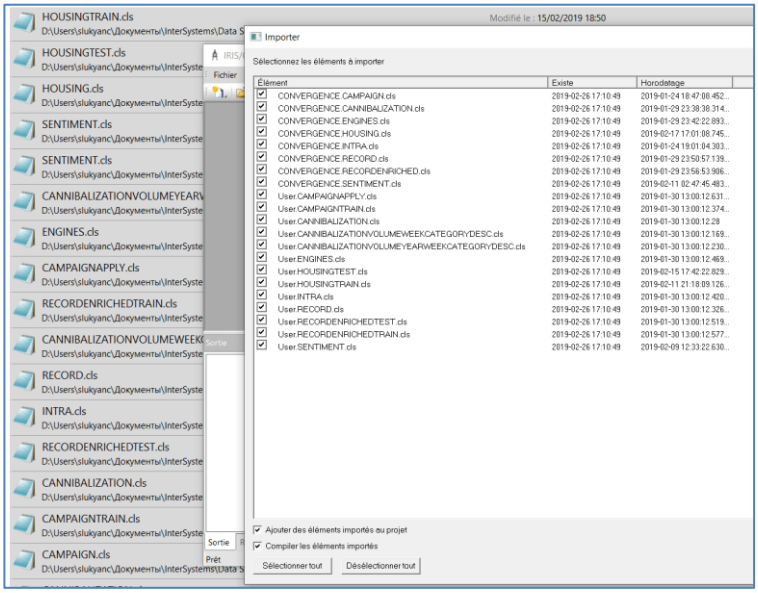
6. Continuous Integration (CI): IRIS Interoperability Tools

6.1. Installation

1. Import ObjectScript classes using IRIS Studio (**IMPORTANT:** run installation for all the tool dependencies that are assumed by the content you are installing in this section)
    a. in the folder where you keep ML Toolkit showcases, run a file search using *.cls mask:

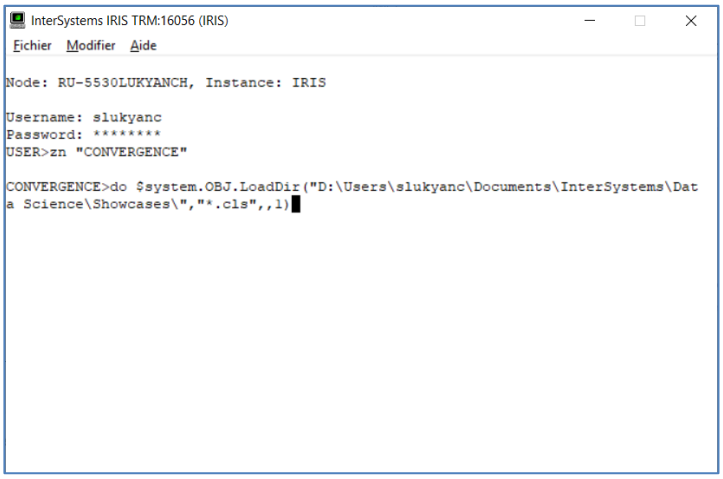


    b. select the files found by the file search above, drag and drop them into your IRIS Studio:



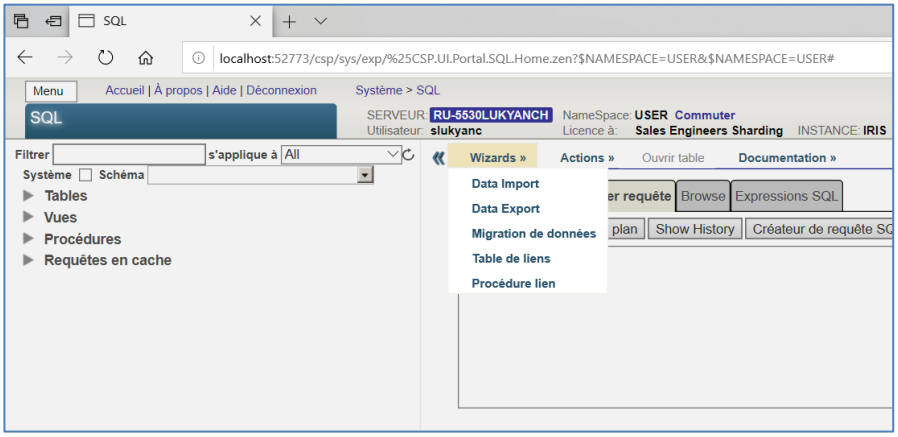
2. Import ObjectScript classes using IRIS Terminal (an alternative to IRIS Studio)

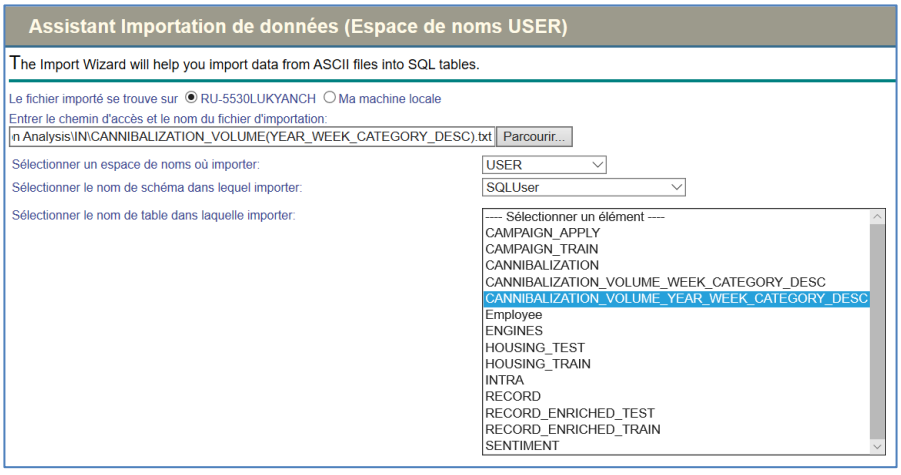
a. in IRIS Terminal execute the following command (adjust the path to point to the folder where you placed the ML Toolkit showcase class files): `do $system.OBJ.LoadDir("C:\path\to\showcases","*.cls",,1)`

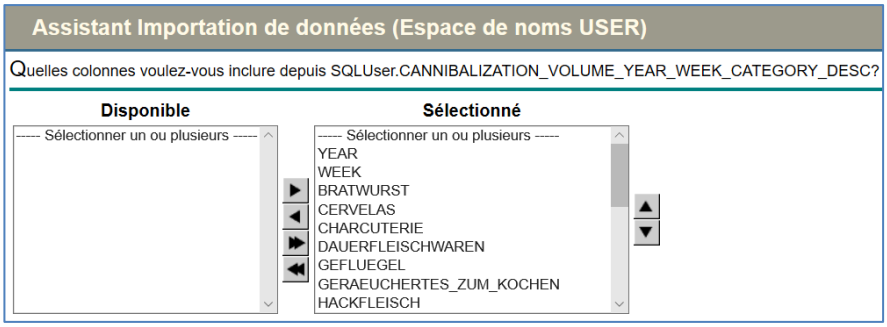


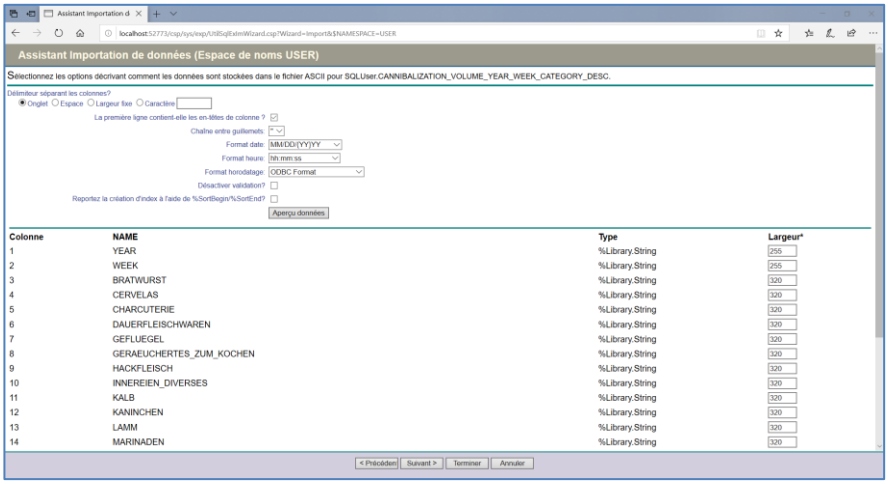
3. Import showcase data

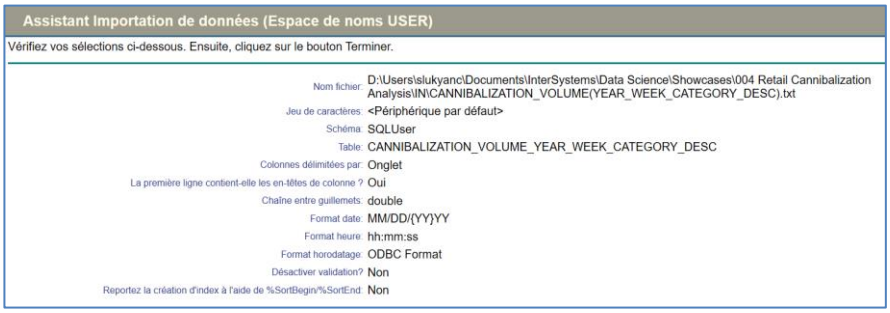
b. from the folder where you keep ML Toolkit showcase input data, import the showcase dataset into its respective IRIS table using Data Import wizard:











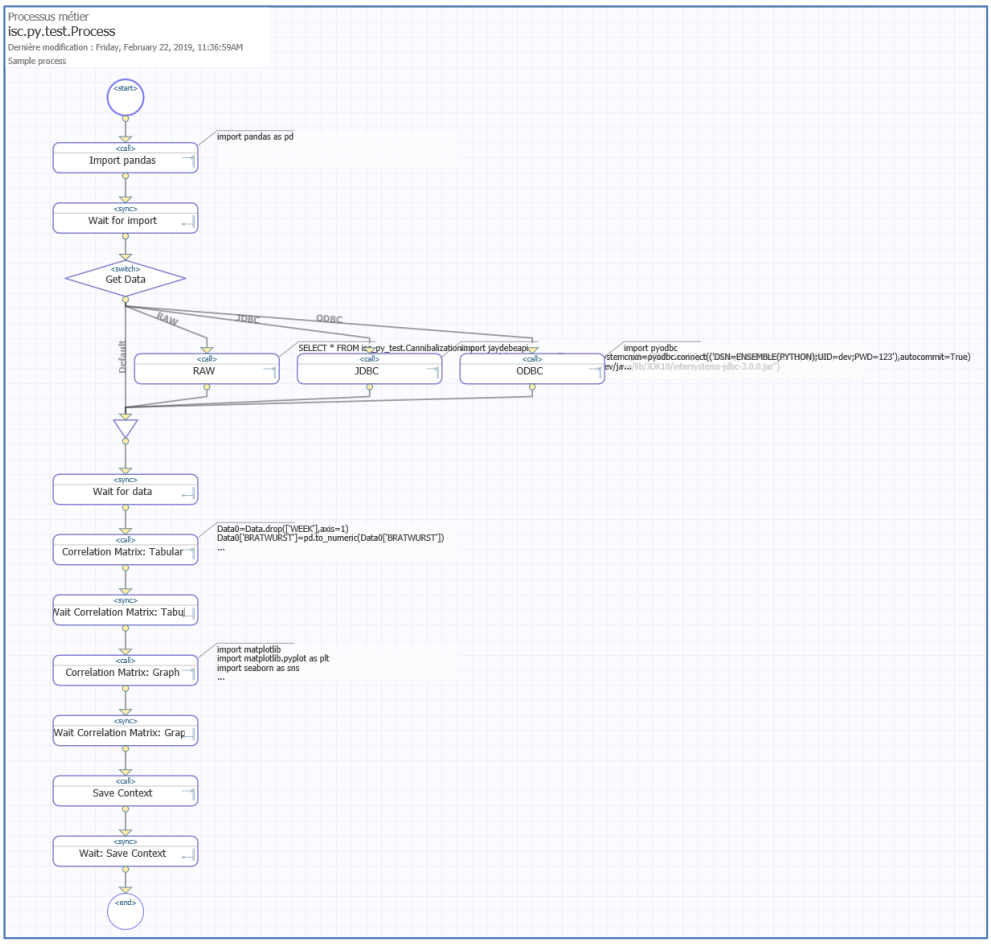
Finish the wizard and the showcase dataset will be imported.

6.2. Use

6.2.1. General Use

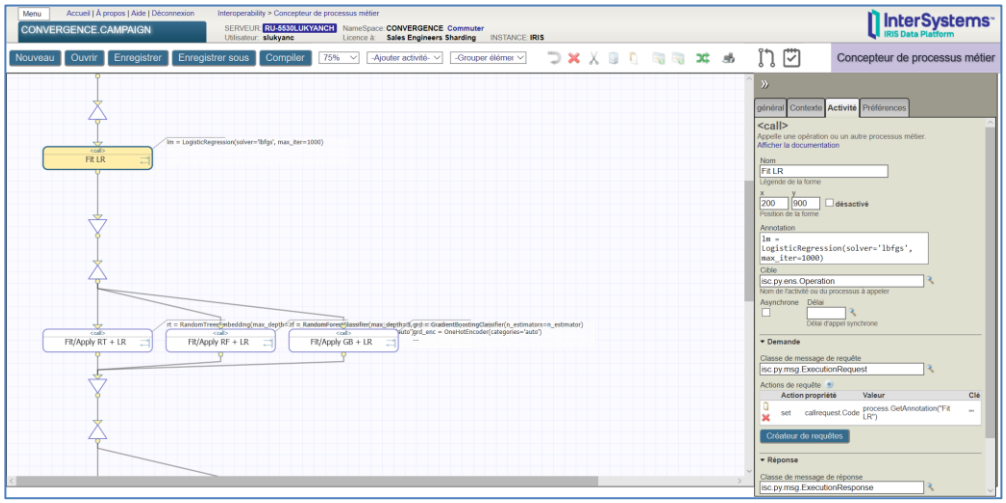
The visual business process designer in IRIS Interoperability is one of the tools for implementing adaptive analytical processes (find more at the following link). All toolsets (e.g., Python Tools, R Tools, etc.) from ML Toolkit can be leveraged in an adaptive analytical process, one process combining 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 external 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.

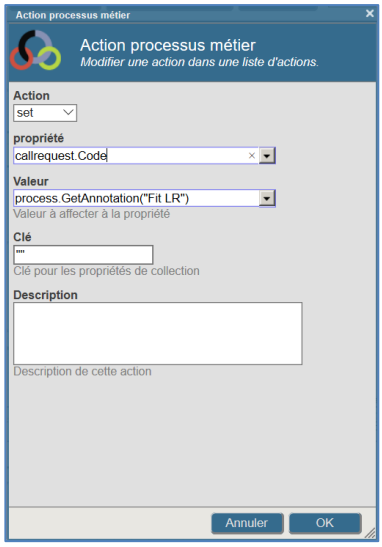
6.2.2. Python Gateway



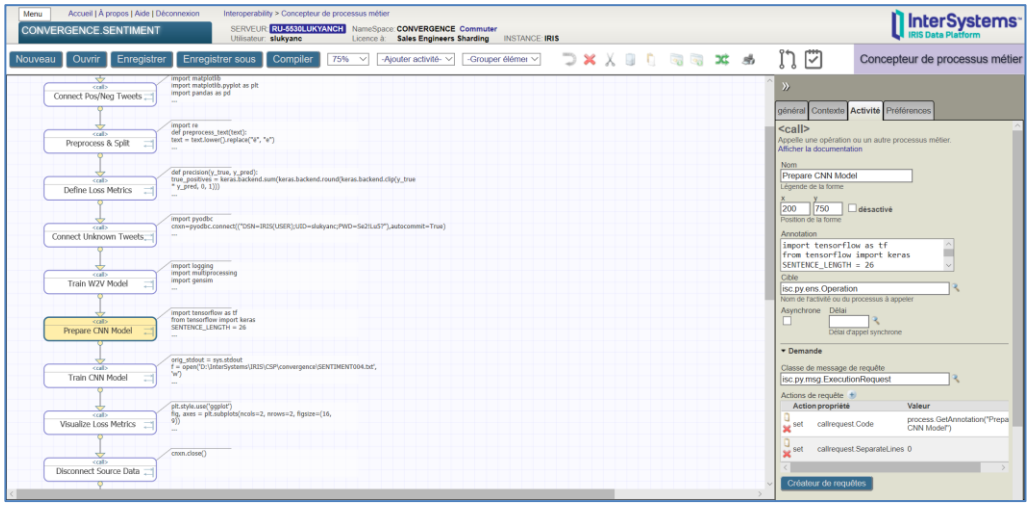
`isc.py.ens.Operation` is the ML Toolkit operation that is added to your IRIS Interoperability production. The following ML Toolkit requests are added:

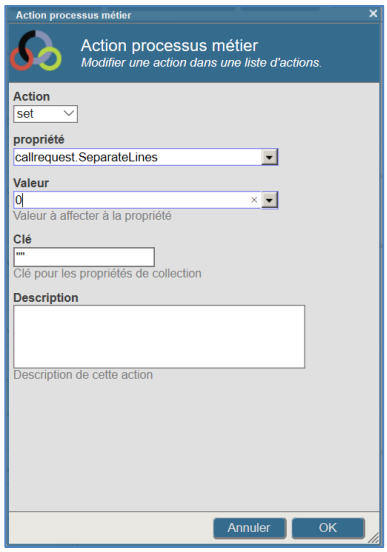
1. Execute Python code via `isc.py.msg.ExecutionRequest` and get the response via `isc.py.msg.ExecutionResponse` with requested variable values as strings:



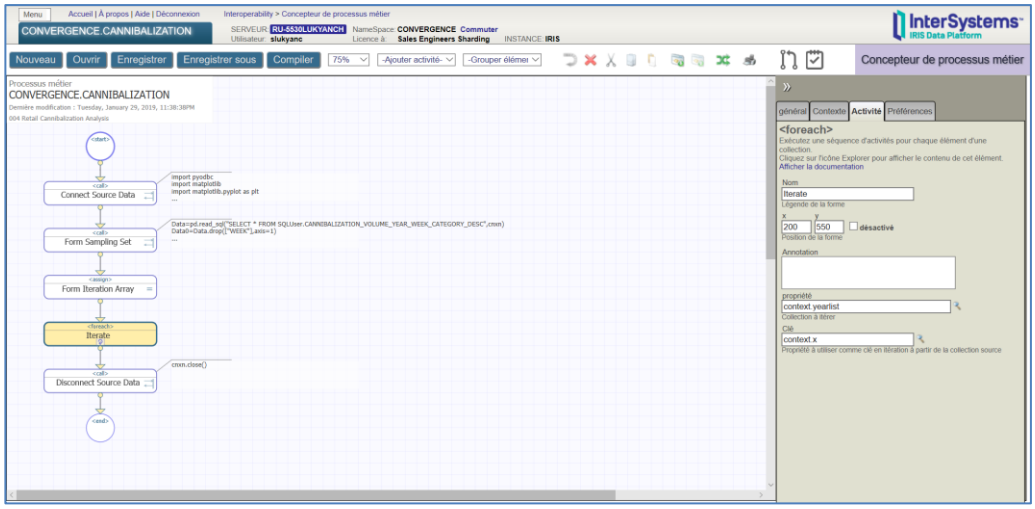


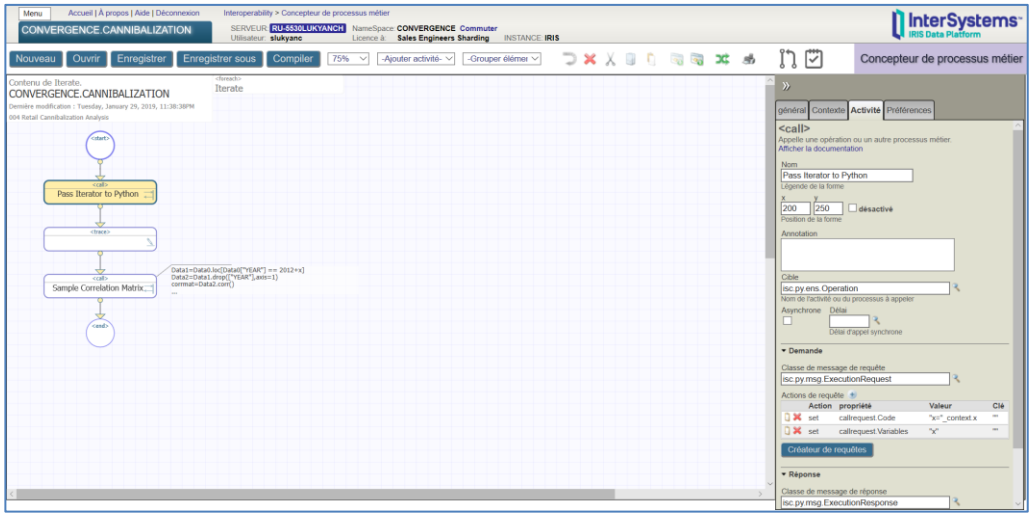
Another use of the same request (adding action to prevent line-by-line interpreting):

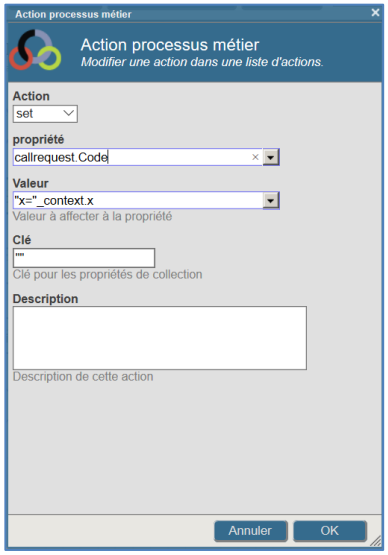


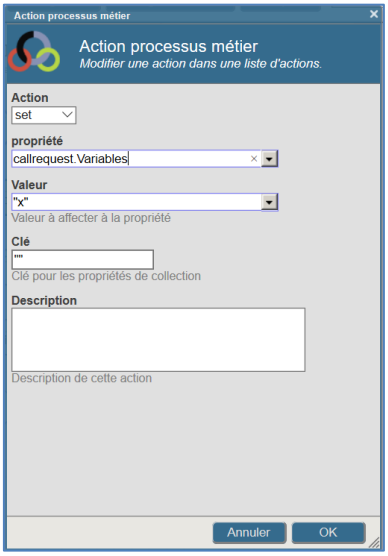


One other use of this request (adding action to pass variables between IRIS and Python contexts):



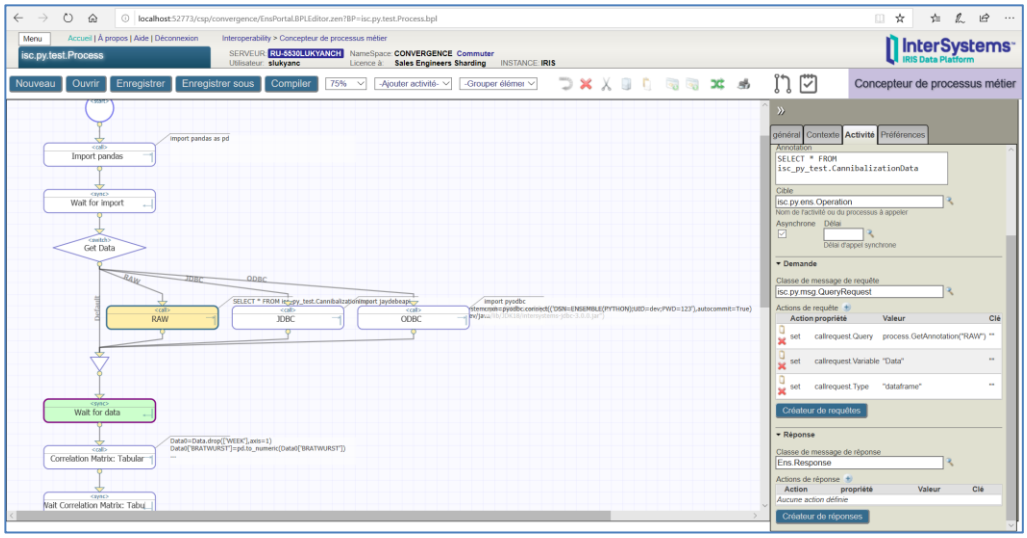


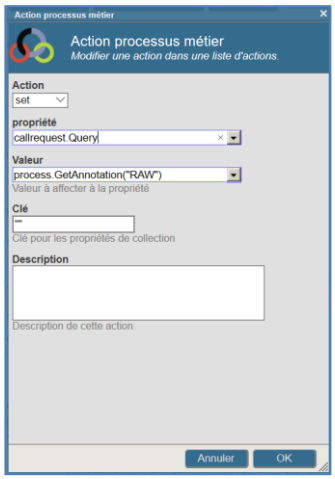


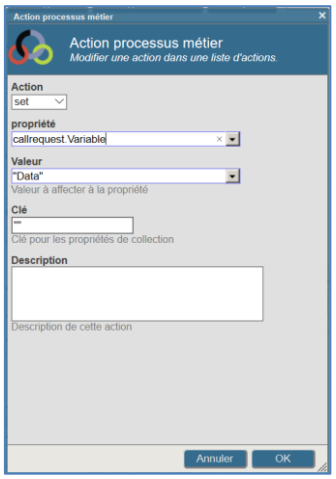


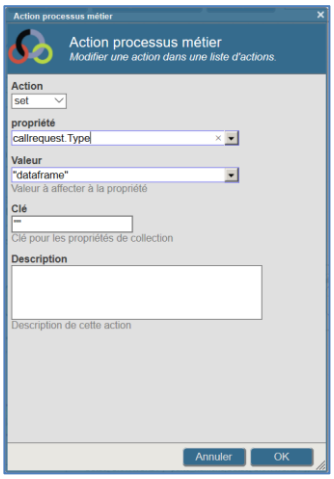
2. Execute Python code via `isc.py.msg.StreamExecutionRequest` and get the response via `isc.py.msg.StreamExecutionResponse` with requested variable values as streams (everything is identical to `isc.py.msg.ExecutionRequest`/ `isc.py.msg.ExecutionResponse` except that all code and variable values are streams)

3. Transfer data into Python from an SQL query with `isc.py.msg.QueryRequest` and get the response via `Ens.Response`

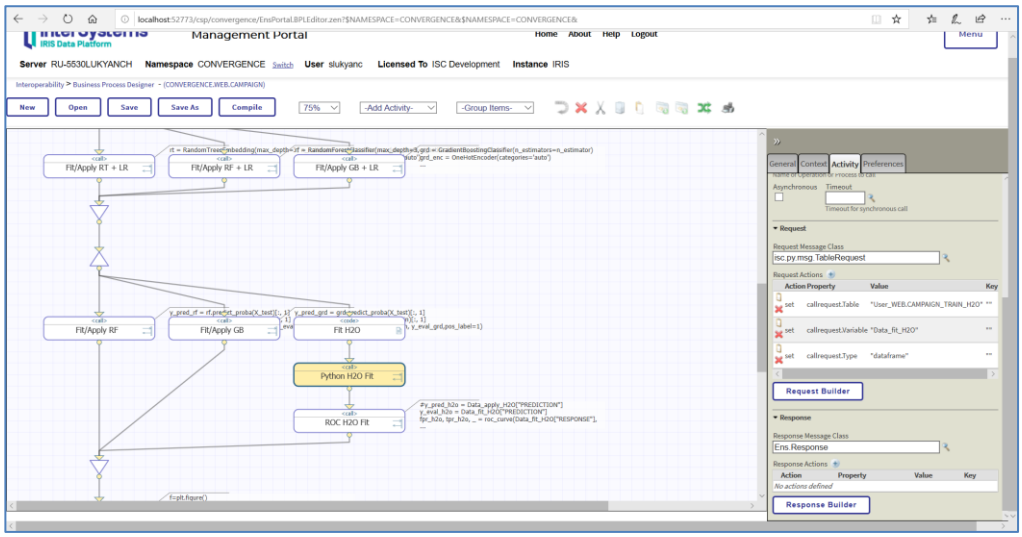


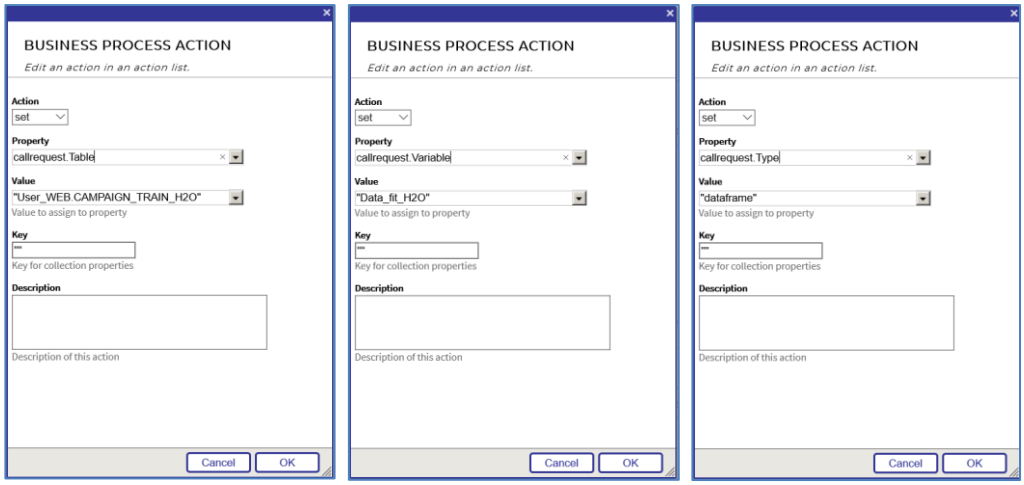




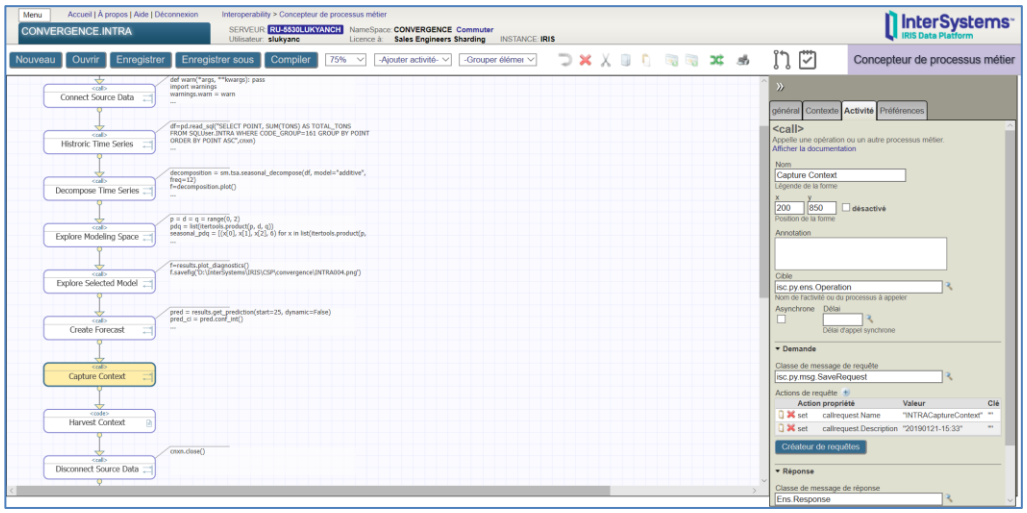


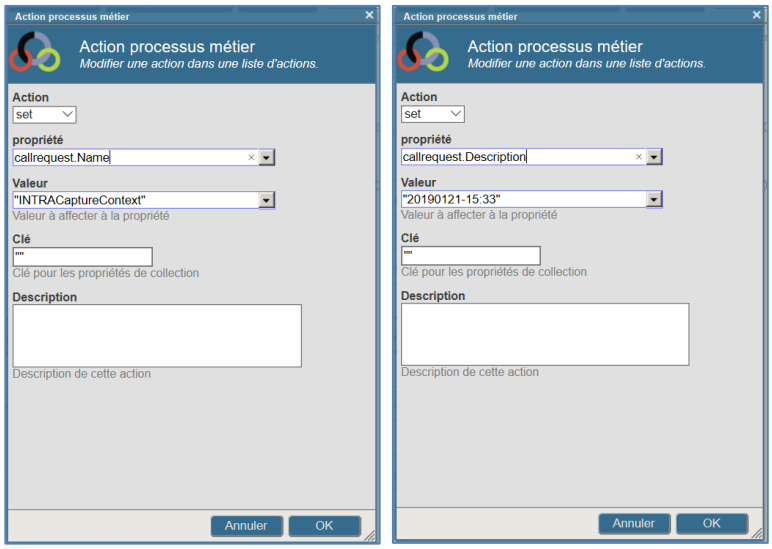
4. Transfer data into Python from an IRIS table with `isc.py.msg.QueryTable` and get the response via `Ens.Response`



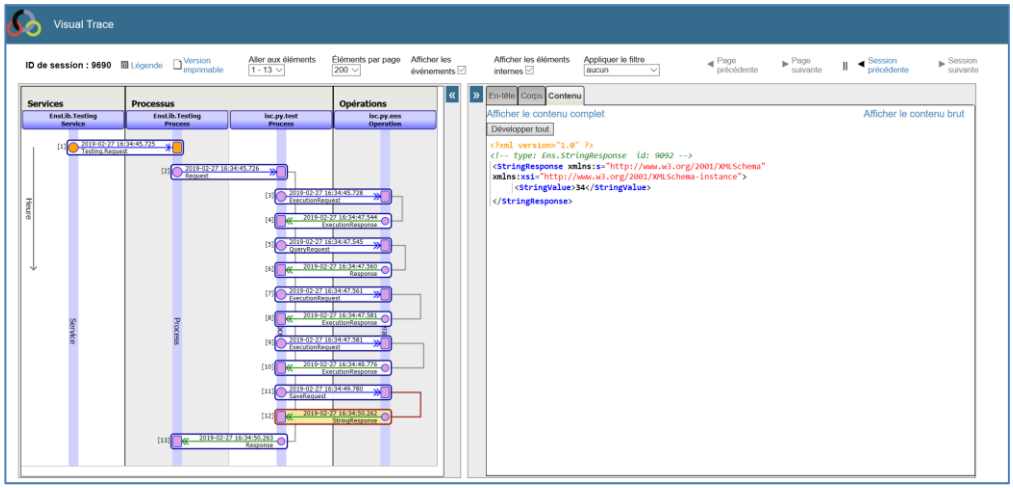


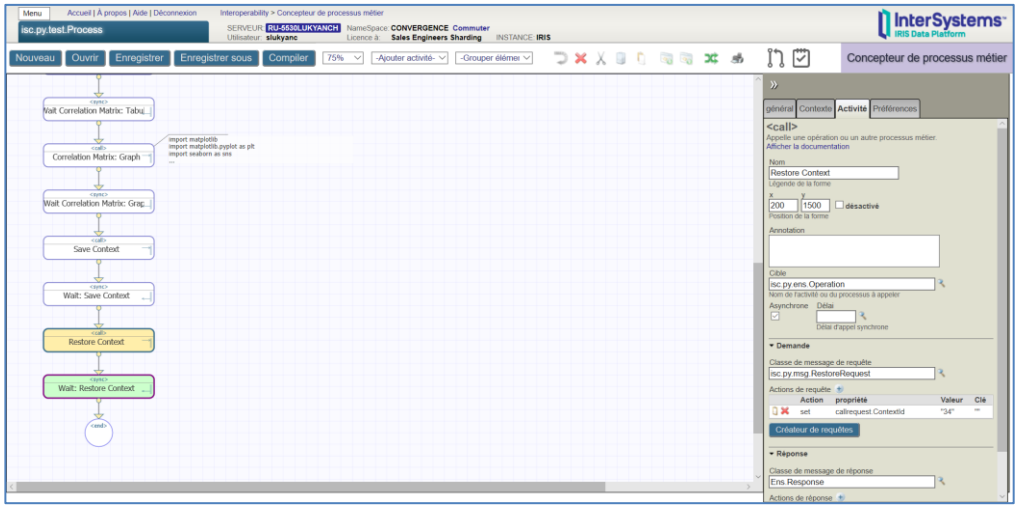
5. Save a Python context with `isc.py. msg.SaveRequest` and get the response via `Ens.StringResponse` with the context ID:

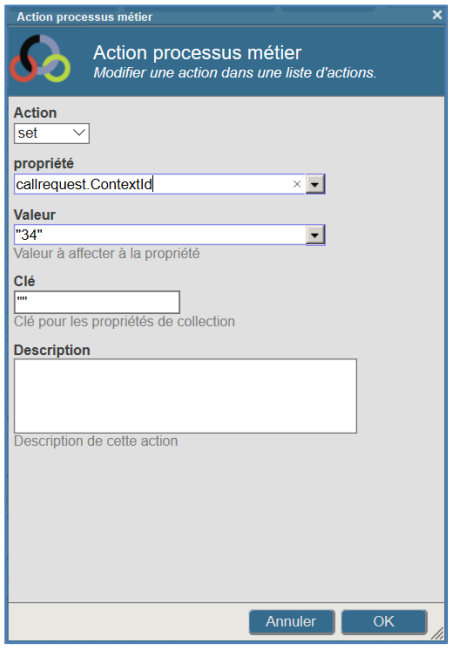


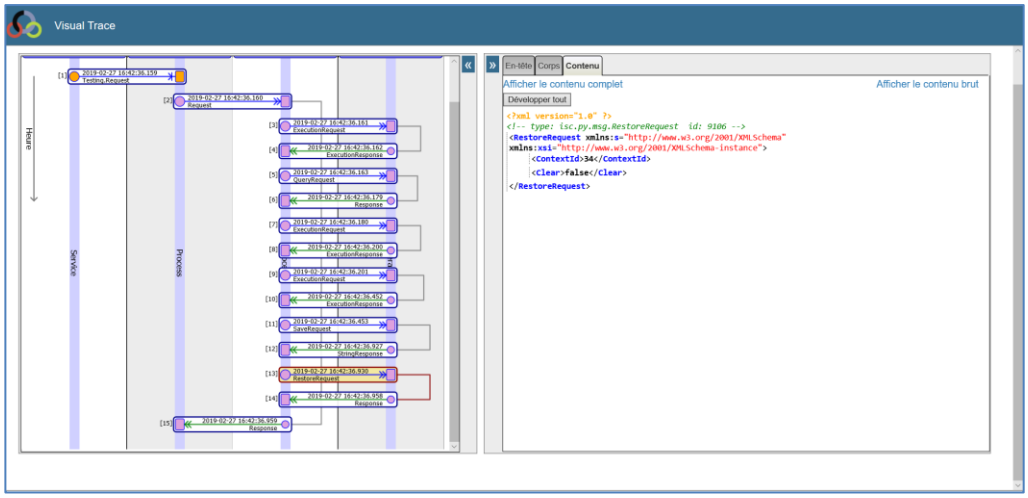


6. Restore a Python context with `isc.py.msg.RestoreRequest`

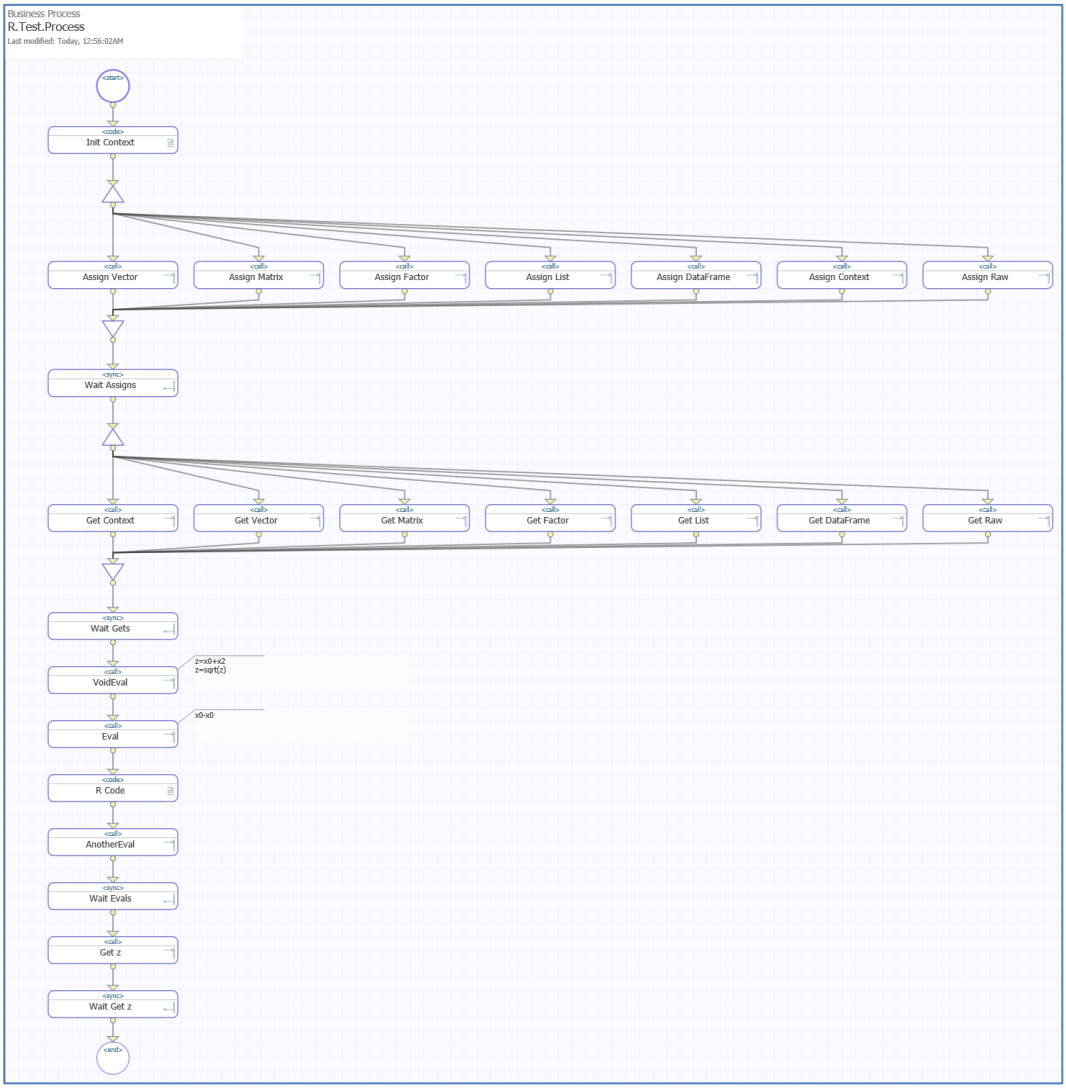

Visual Trace
ID de session : 9690
Services
Processus
Opérations
Afficher le contenu complet
Afficher le contenu brut
Développer tout


Menu
isc.py.test.Process
Nouveau
Ouvrir
Enregistrer
Enregistrer sous
Compiler
Concepteur de processus métier
Wait Correlation Matrix: Tabu
Correlation Matrix: Graph
Wait Correlation Matrix: Grap
Save Context
Wait: Save Context
Restore Context
Wait: Restore Context
Activité
Restore Context
isc.py.ens.Operation
isc.py.msg.RestoreRequest
Ens.Response


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

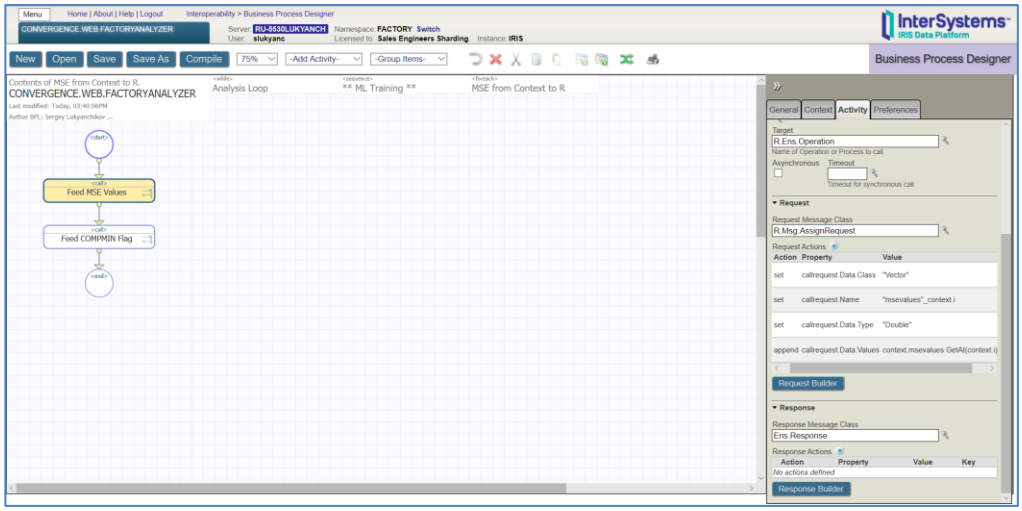


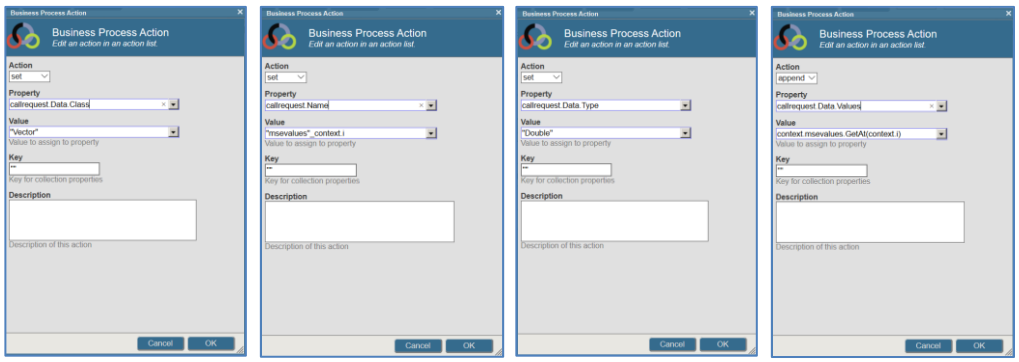
6.2.3. R Gateway



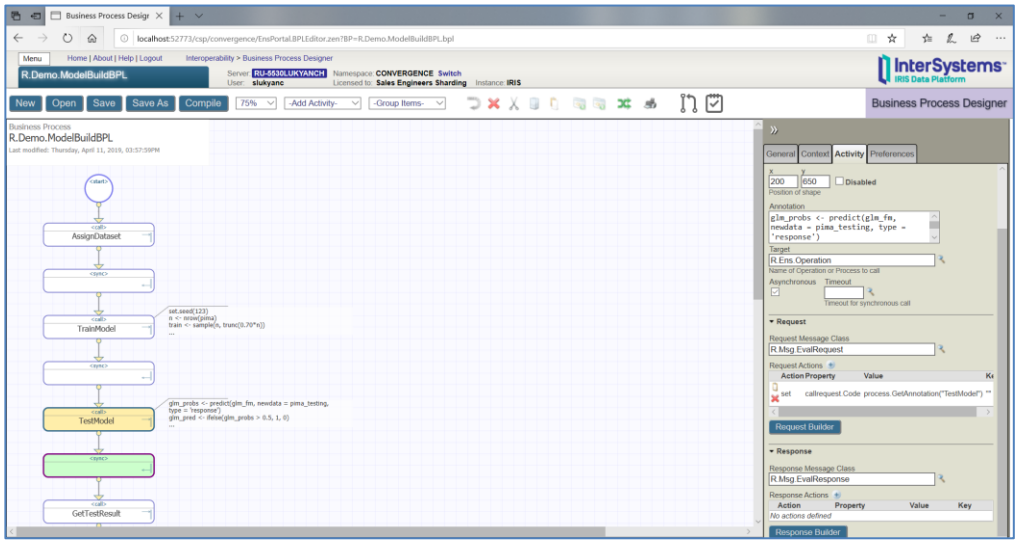
`R.Ens.Operation` is the ML Toolkit operation that is added to your IRIS Interoperability production. The following ML Toolkit requests are added:

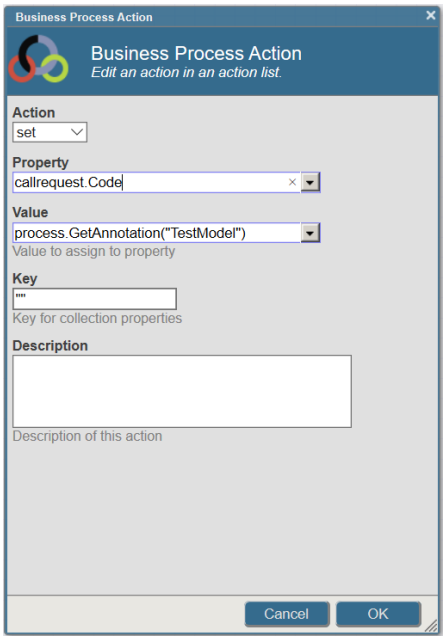
1. Assign values to a named variable in R via `R.Msg.AssignRequest`:



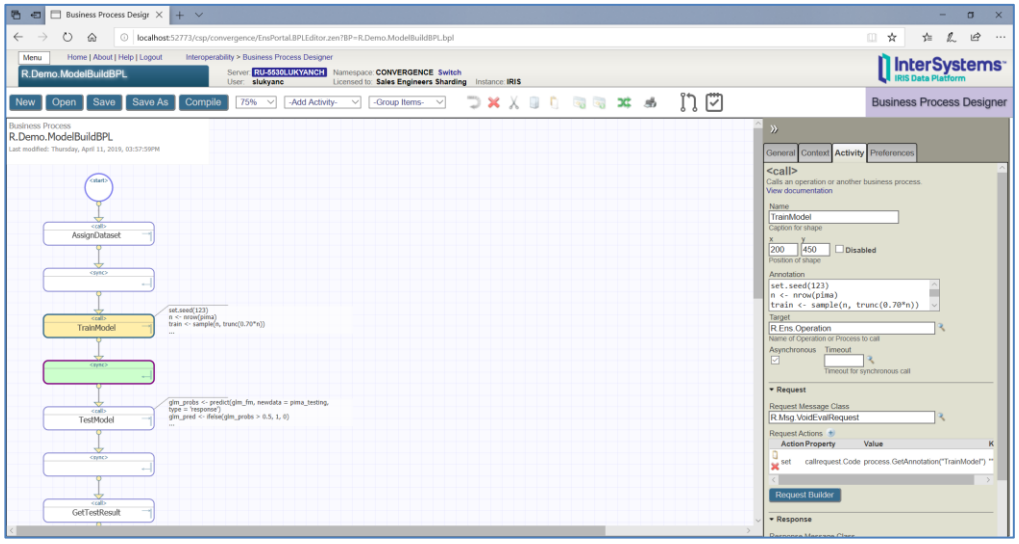


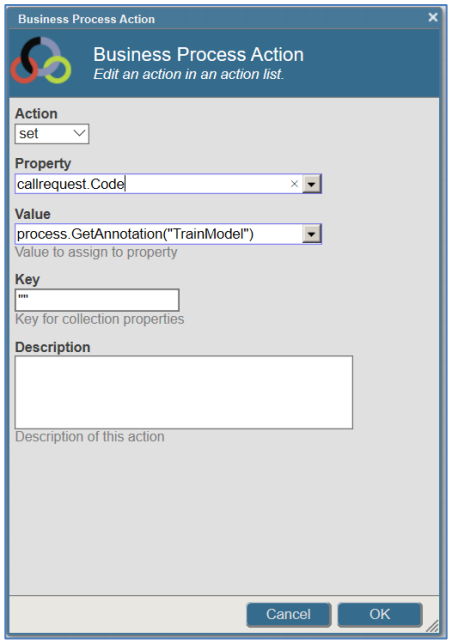
2. Execute an R code and evaluate in IRIS-mappaple terms code's named variables via `R.Msg.EvalRequest` and get the response via `R.Msg.EvalResponse`:



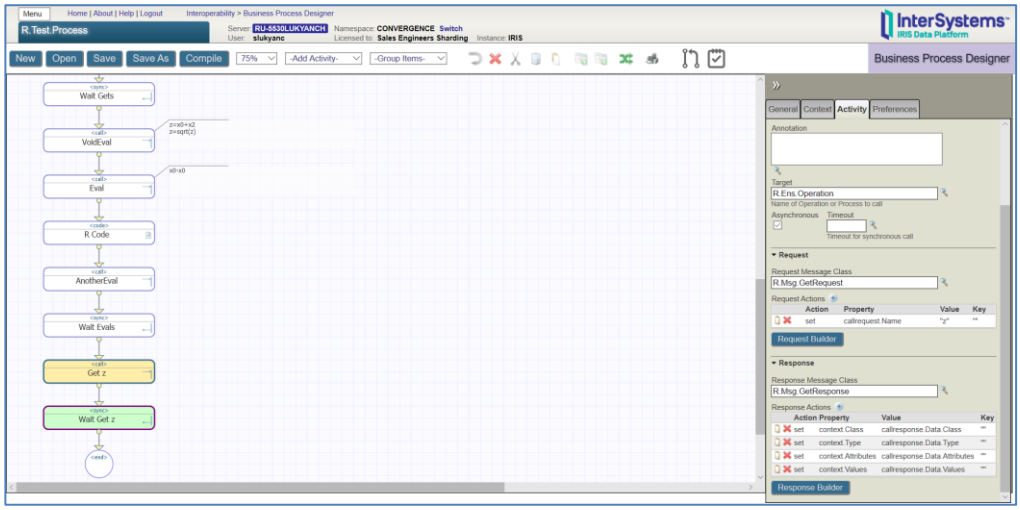


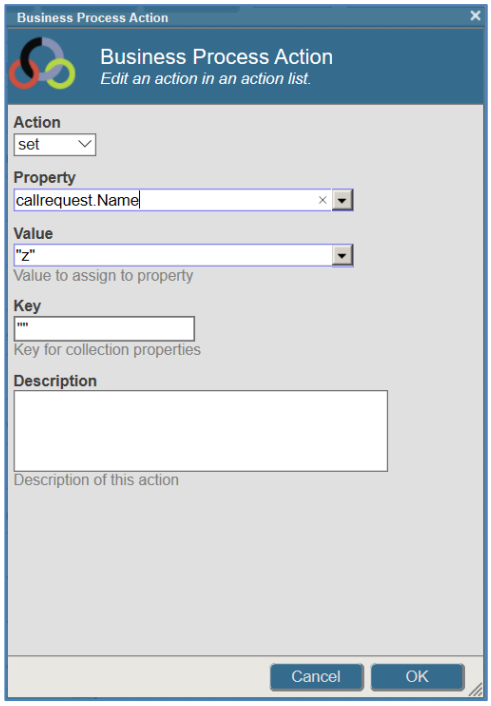
3. Execute an R code without evaluating in IRIS-mappaple terms code's named variables with `R.Msg.VoidEvalRequest`:

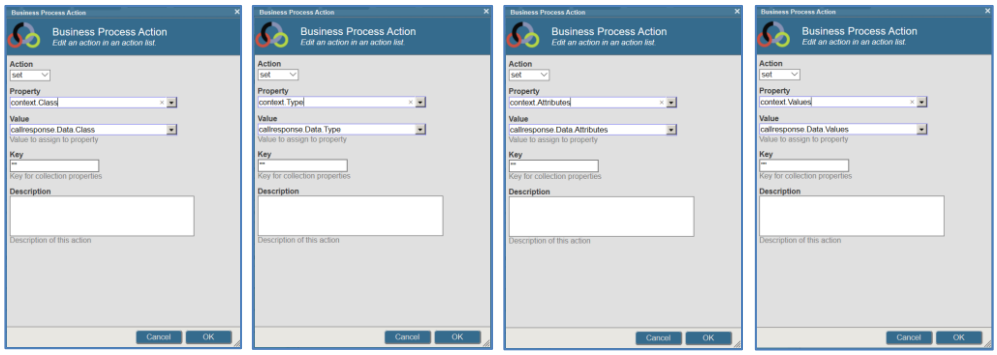




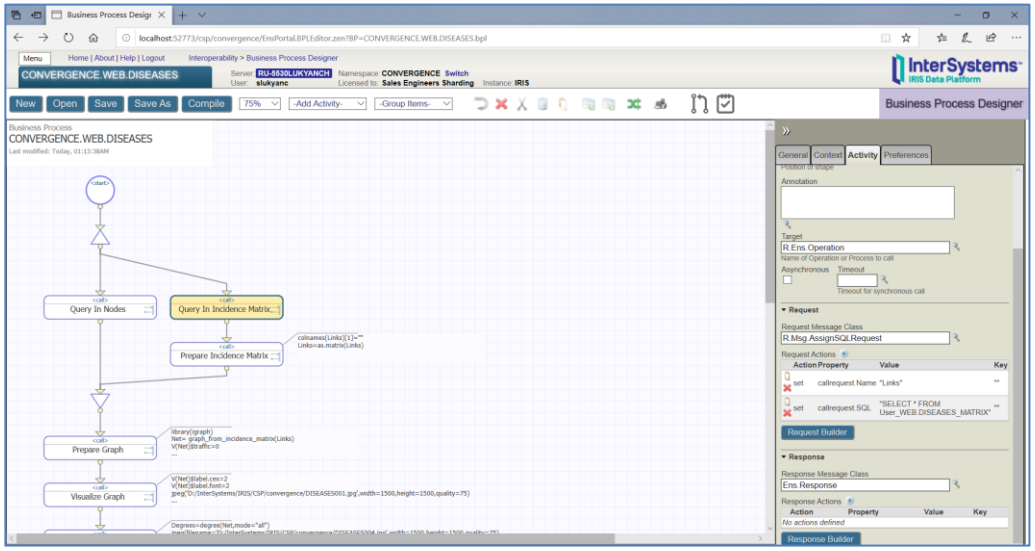
4. Retrieve values from a named variable in R with `R.Msg.GetRequest` and get the response via `R.Msg.GetResponse`:

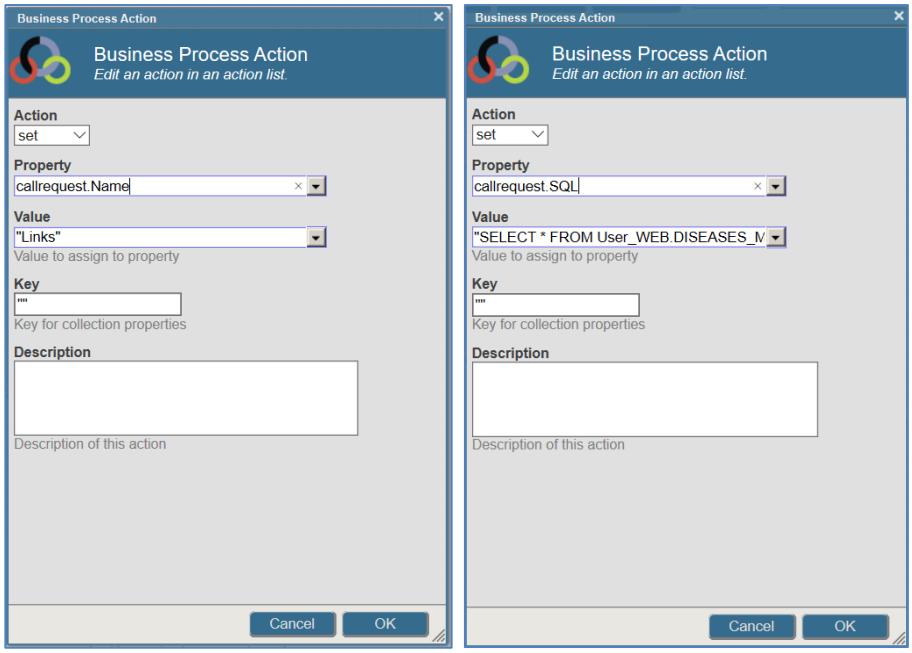




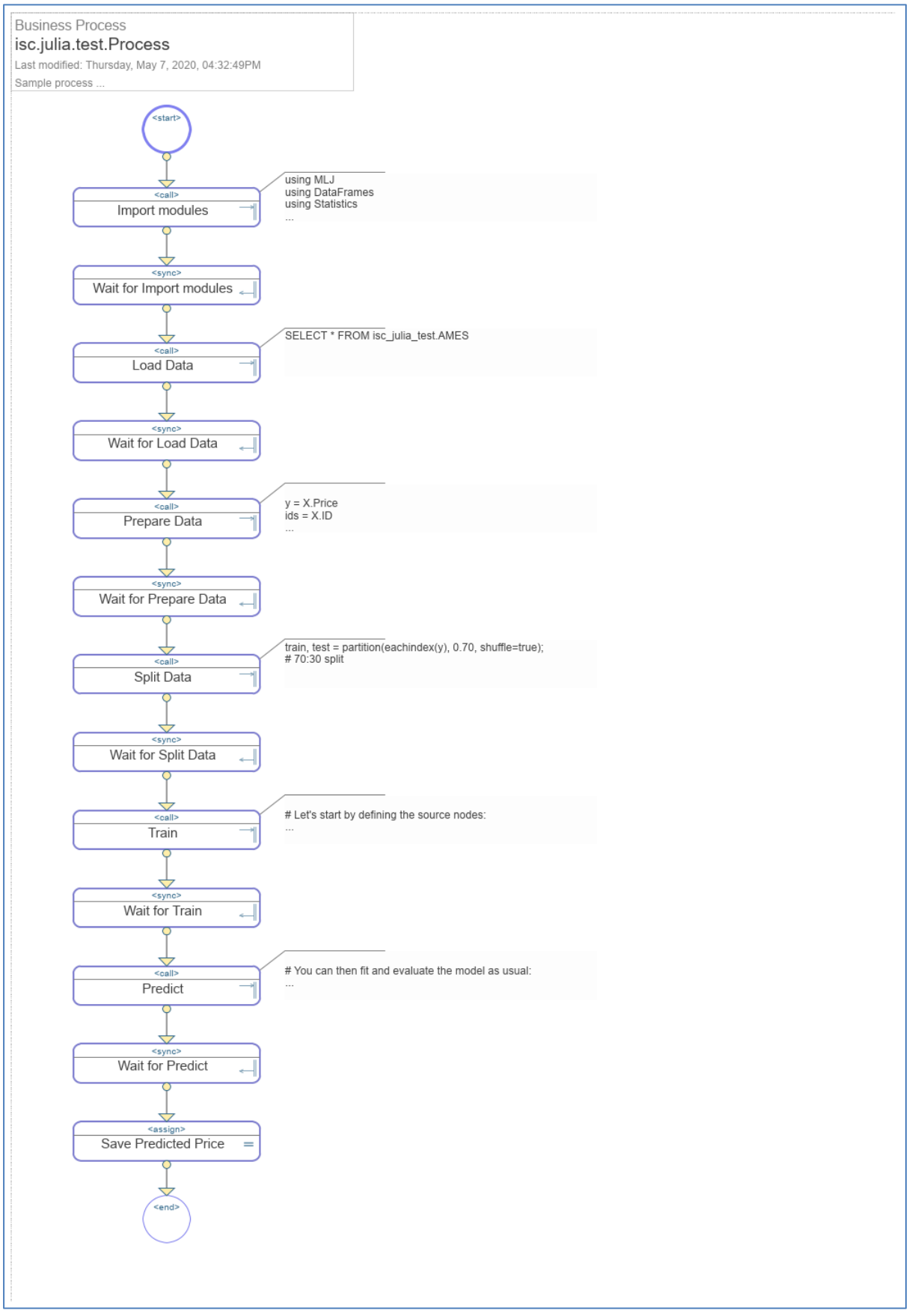


5. Execute an SQL statement and assign its resultset as dataframe to a named variable in R via `R.Msg.AssignSQLRequest`:



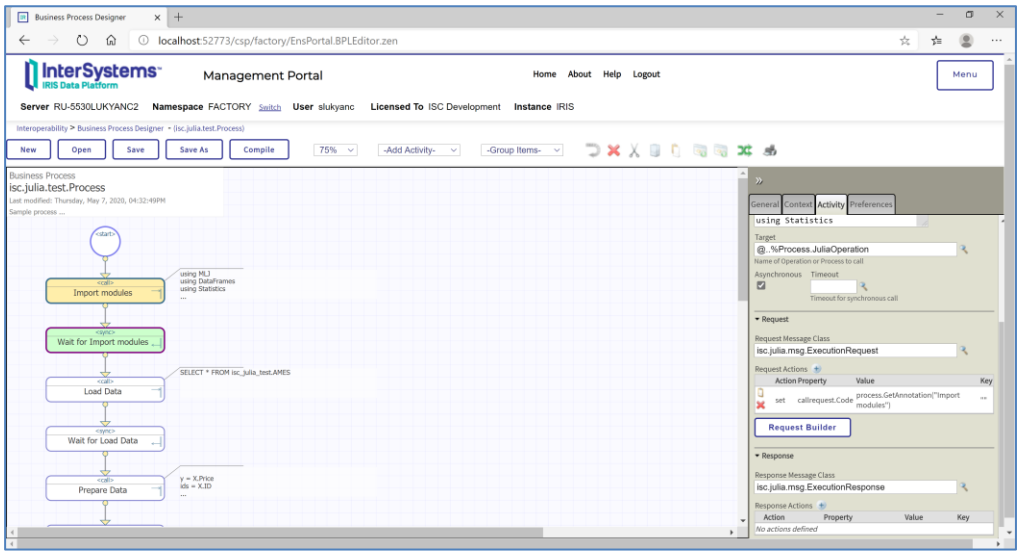


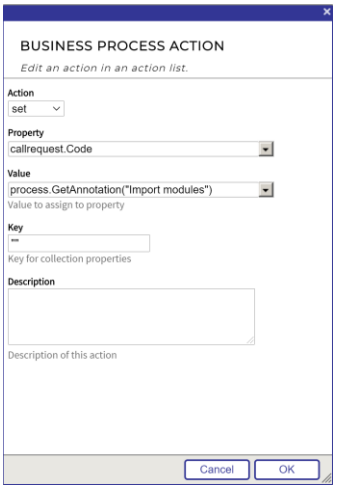
6.2.4. Julia Gateway



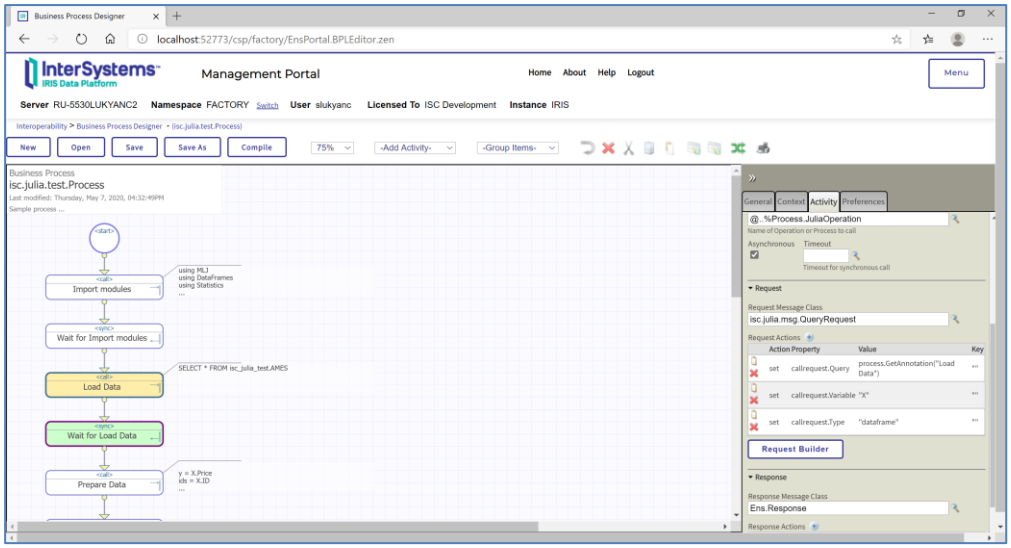
Interoperability adapter `isc.julia.ens.Operation` offers ability to interact with Julia process from Interoperability productions. Currently three requests are supported:

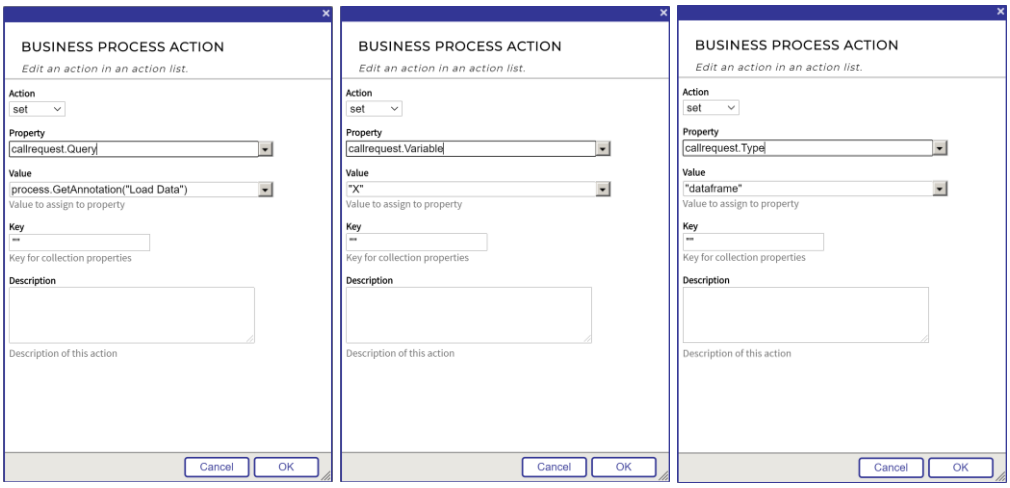
1. Execute Julia code via `isc.julia.msg.ExecutionRequest`. Returns `isc.julia.msg.ExecutionResponse` with requested variable values:





2. Execute Julia code via `isc.julia.msg.StreamExecutionRequest`. Returns `isc.julia.msg.StreamExecutionResponse` with requested variable values. Same as above but accepts and returns streams instead of strings
3. Create dataframe from an SQL query with `isc.julia.msg.QueryRequest`. Returns `Ens.Response`


Business Process Designer
localhost:52773/csp/factory/EnsPortal.BPLEditor.zen
InterSystems
IRIS Data Platform
Management Portal
Home
About
Help
Logout
Menu
Server RU-5530LUKYANC2
Namespace FACTORY
Switch
User slukyanc
Licensed To ISC Development
Instance IRIS
Interoperability > Business Process Designer - (isc.julia.test.Process)
New
Open
Save
Save As
Compile
75%
-Add Activity-
-Group Items-
Business Process
isc.julia.test.Process
Import modules
Wait for Import modules
Load Data
Wait for Load Data
Prepare Data
SELECT * FROM isc_julia_test.AMES
General
Context
Activity
Preferences
@..%Process.JuliaOperation
Name of Operation or Process to call
Asynchronous
Timeout
Timeout for synchronous call
Request
Request Message Class
isc.julia.msg.QueryRequest
Request Actions
Action
Property
Value
Key
set
callrequest.Query
process.GetAnnotation("Load Data")
set
callrequest.Variable
"X"
set
callrequest.Type
"dataframe"
Request Builder
Response
Response Message Class
Ens.Response
Response Actions

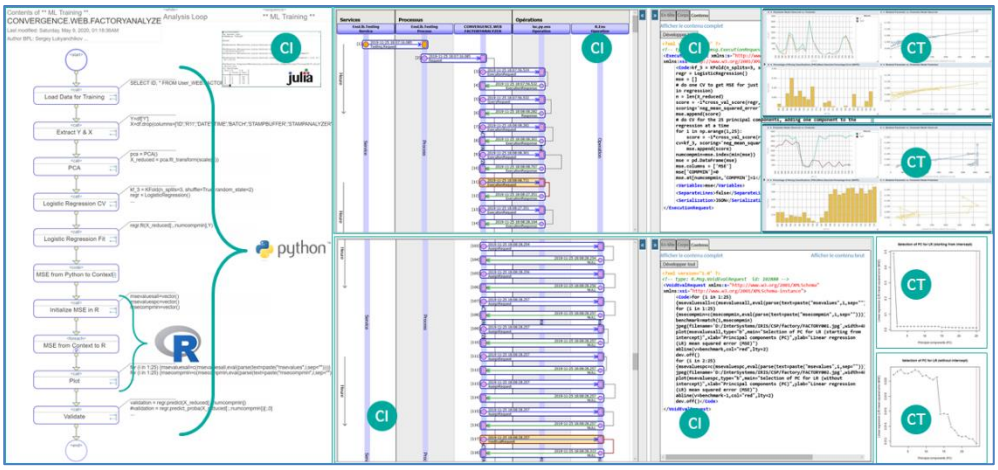

BUSINESS PROCESS ACTION
Edit an action in an action list.
Action
set
Property
callrequest.Query
Value
process.GetAnnotation("Load Data")
Value to assign to property
Key
Key for collection properties
Description
Description of this action
Cancel
OK
BUSINESS PROCESS ACTION
Edit an action in an action list.
Action
set
Property
callrequest.Variable
Value
"X"
Value to assign to property
Key
Key for collection properties
Description
Description of this action
Cancel
OK
BUSINESS PROCESS ACTION
Edit an action in an action list.
Action
set
Property
callrequest.Type
Value
"dataframe"
Value to assign to property
Key
Key for collection properties
Description
Description of this action
Cancel
OK

7. Continuous Training (CT)

7.1. Robotized Real-Time Decision Making Using IRIS Interoperability

7.1.1. Interoperability with Mathematical Modeling Toolsets

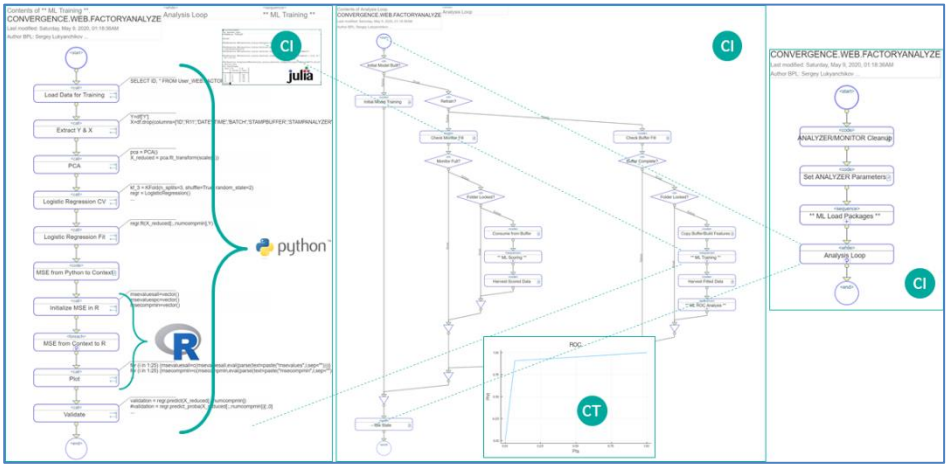
In the below diagram, in the left part of the image we see the fragment of a business process that implements execution of Python and R scripts. In the central part – we see the visual logs following execution of those scripts, in Python and in R accordingly. Next after them – examples of the content in both languages, passed for execution in respective environments. In the right part – visualizations based on the script outputs. The visualizations in the upper right corner are developed using IRIS Analytics (the data is transferred from Python to InterSystems IRIS platform and is put on a dashboard using platform functionality), in the lower right corner – obtained directly in R session and transferred from there to graphical files. An important remark: the discussed business process fragment is responsible in this prototype for model training (equipment condition classification) based on the data received in real time from the equipment imitator process, that is triggered by the classification accuracy monitor process that monitors performance of the classification model as it is being applied. Implementing an AI/ML solution as a set of interacting business processes (“agents”) will be discussed further in the text.



7.1.2. Adaptability of In-Platform Processes – Graphical Format

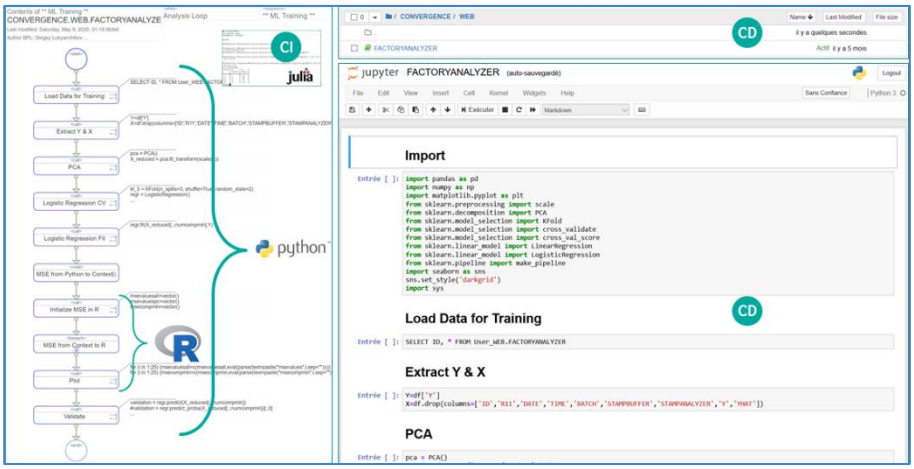
In-platform processes (a.k.a. “business processes”, “analytical processes”, “pipelines”, etc.– depending on the context) can be edited, first of all, using the visual business process composer in the platform, in such a way that both the process diagram and its corresponding AI/ML mechanism (code) are created at the same time. By saying “an AI/ML mechanism is created”, we mean hybridity from the very start (at a process level): the content written in the languages of mathematical modeling environments neighbors the content written in SQL (including IntegratedML extensions), in InterSystems ObjectScript, as well as other supported languages. Moreover, the in-platform paradigm opens a very wide spectrum of capability for “drawing” processes as sets of embedded fragments (as shown in the below diagram), helping with efficient structuring of sometimes rather complex content, avoiding “dropouts” from visual composition (to “non-visual” methods/classes/procedures, etc.). I.e., in case of necessity (likely in most projects), the entire AI/ML solution can be implemented in a visual self-documenting format. We draw your attention to the central part of the below diagram that illustrates a “higher-up embedding layer” and shows that apart from model training as such (implemented using Python and R), there is analysis of the so-called ROC curve of the trained model allowing

to assess visually (and computationally) its training quality – this analysis is implemented using Julia language (executes in its respective Julia environment).



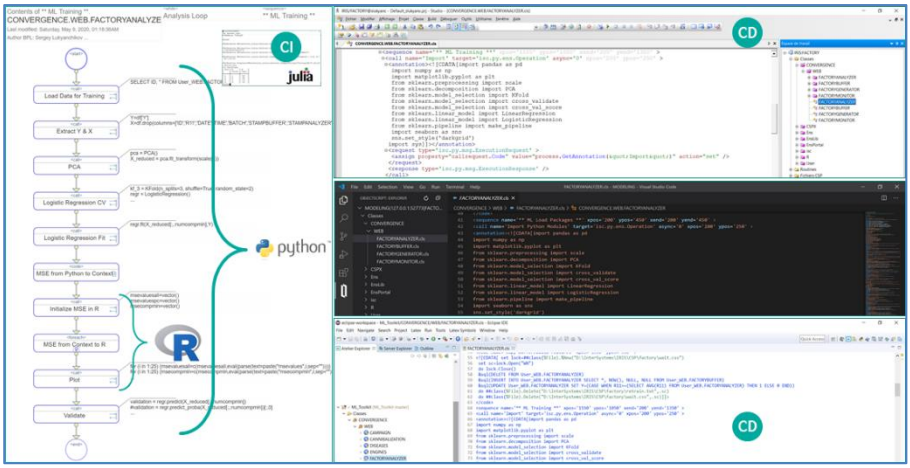
7.1.3. Adaptability of In-Platform Processes – Notebook Format

As mentioned before, the initial development and (in other cases) adjustment of the already implemented in-platform AI/ML mechanisms will be performed outside the platform in Jupyter editor. In the below diagram we can find an example of editing an existing in-platform process (the same process as in the diagram above) – this is how its model training fragment looks in Jupyter. The content in Python language is available for editing, debugging, viewing inline graphics in Jupyter. Changes (if required) can be immediately replicated to the in-platform process, including its production version. Similarly, newly developed content can be replicated to the platform (a new in-platform process is created automatically).



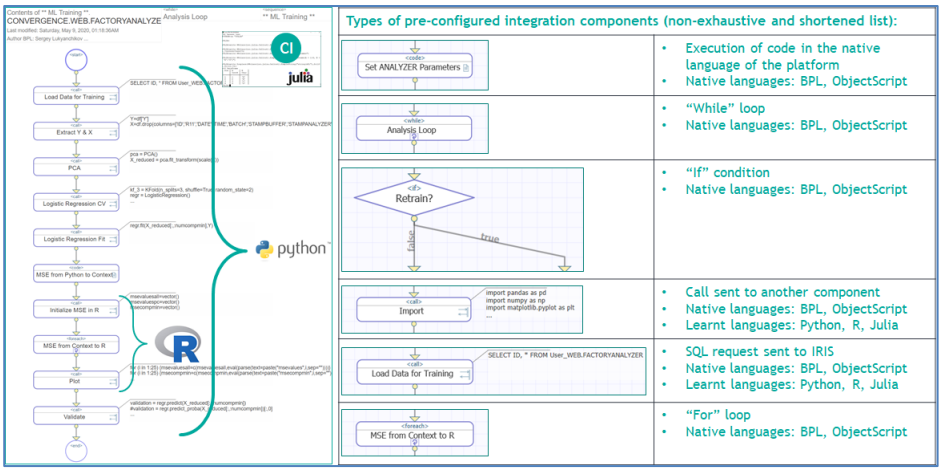
7.1.4. Adaptability of In-Platform Processes – IDE Format

Editing of an in-platform process can be performed not only in a visual or a notebook format – but in a “complete” IDE (Integrated Development Environment) format as well. The IDEs being IRIS Studio (the native IRIS development studio), Visual Studio Code (an InterSystems IRIS extension for VSCode) and Eclipse (Atelier plugin). In certain cases, simultaneous usage by a development team of all the three IDEs is possible. In the diagram below we see an example of editing all the same process in IRIS Studio, in Visual Studio Code and in Eclipse. Absolutely any portion of the content is available for editing: Python/R/Julia/SQL, ObjectScript and the business process elements.



7.1.5. Adaptiveness of In-Platform Processes

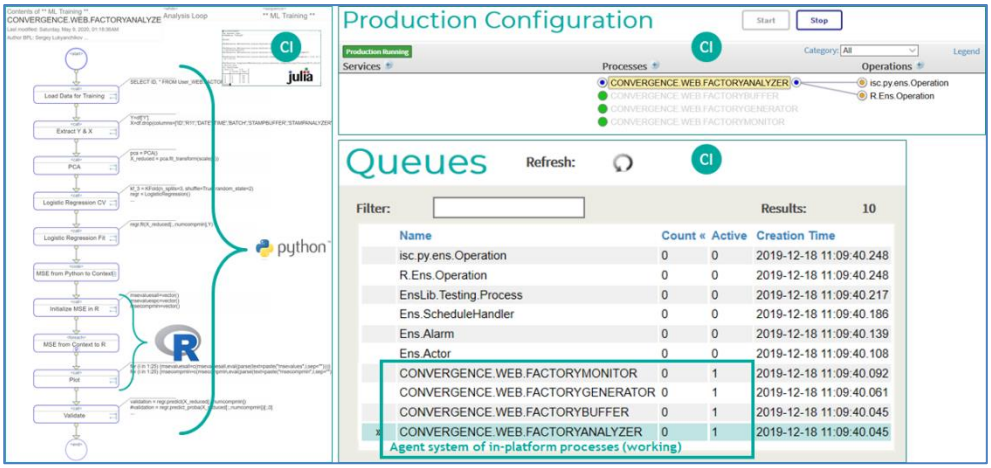
The means of composition and execution of business processes in InterSystems IRIS using Business Process Language (BPL), are worth a special mentioning. BPL allows using “pre-configured integration components” (activities) in business processes. Pre-configured business process components (activities and links among them) are extremely powerful accelerators for AI/ML solution assembly. And not only for assembly: due to activities and their links, an “autonomous management layer” is introduced above disparate AI/ML mechanisms, capable of making real-time decisions depending on the situation.



Find more information on pre-configured integration components in InterSystems IRIS.

7.1.6. Agency of In-Platform Processes

The concept of agent systems (a.k.a. “multiagent systems”) has strong acceptance in robotization, and InterSystems IRIS platform provides organic support for it through its “production/process” construct. Besides unlimited capabilities for “arming” each process with the functionality required for the overall solution, “agency” as the property of an in-platform processes family, enables creation of efficient solutions for very unstable modeled phenomena (behavior of social/biological systems, partially observed manufacturing processes, etc.).



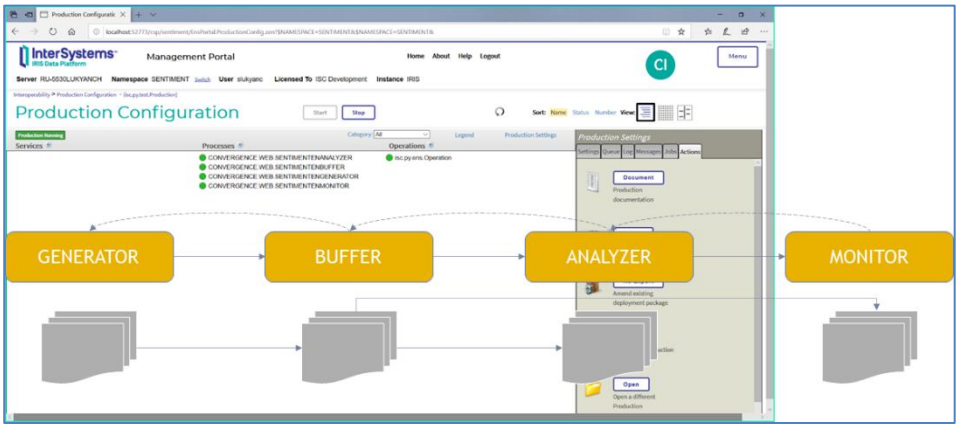
Find more information on production functionality in InterSystems IRIS.

7.2. Applied Use Domains

We proceed by presenting applied use domains containing solutions for entire classes of real-time scenarios (a fairly detailed discovery of some of the in-platform AI/ML best practices based on InterSystems IRIS is provided in one of our previous webinars).

7.2.1. AI/ML Robotization (Agent Systems)

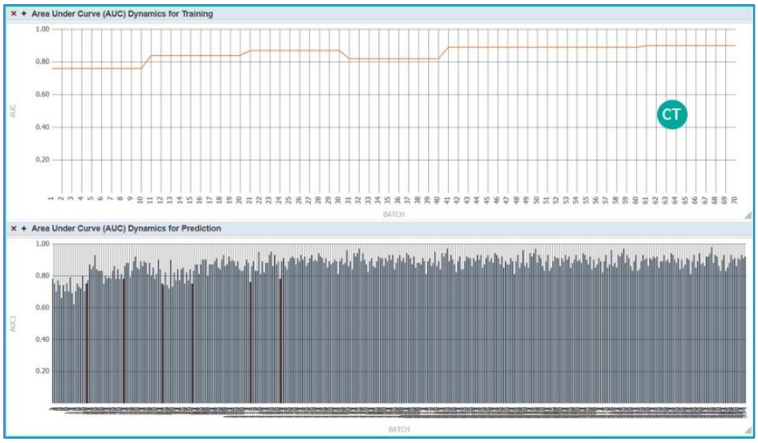
In “hot pursuit” of the diagram from the section above, we provide below a more illustrative diagram of an agent system. In that diagram, the same all prototype is shown with its four agent processes plus the interactions among them: GENERATOR – simulates data generation by equipment sensors, BUFFER – manages data processing queues, ANALYZER – executes machine learning, properly speaking, MONITOR – monitors machine learning quality and signals the necessity for model retrain.



Implement Robotized ML showcase to discover this domain.

7.2.2. AI/ML Robotization (Autonomy and Self-Learning)

The diagram below illustrates the functioning of a different robotized prototype (text sentiment analysis) over a period. In the upper part – the model training quality metric evolution (quality increasing), in the lower part – dynamics of the model application quality metric and retrains (red stripes). As we can see, the solution has shown an effective and autonomous self-training while continuing to function at the required level of quality (the quality metric values stay above 80%).

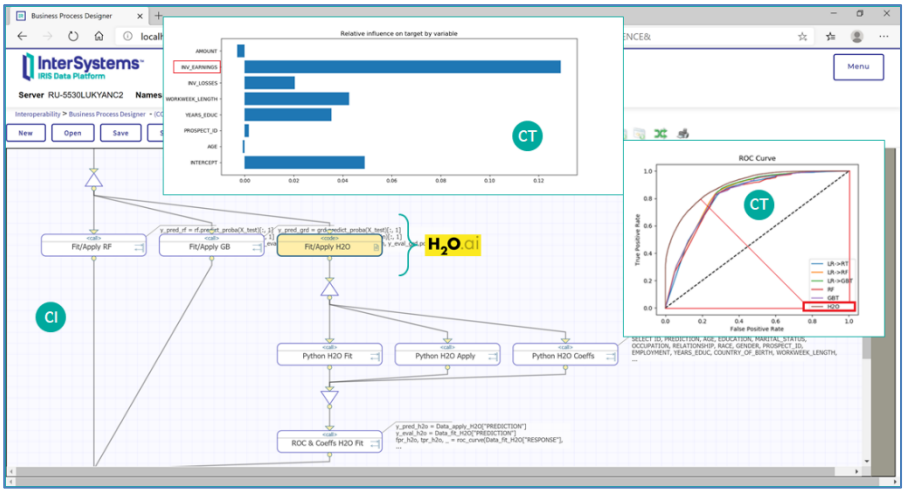


Implement Robotized Sentiment Analysis showcase to discover this domain.

7.2.3. Automated AI/ML Model Selection and Parameter Determination

We were already mentioning “auto ML” in the introduction to this document, and in the below diagram we are now providing more details about this functionality using one other prototype as an example. In the diagram of a business process fragment, we see an activity that launches modeling in H2O framework, as well as the outcomes of that modeling (a clear supremacy of the obtained model in terms of ROC curves, compared to the other “hand-made” models, plus automated detection of the “most influential variables” among the ones available in the original dataset). An important aspect here is the saving of time and expert resources that is gained due to “auto ML”: our in-platform process delivers in half a minute what may take an expert from one week to one month (determining and proofing of an optimal model).

Find more information on IntegratedML functionality in InterSystems IRIS.

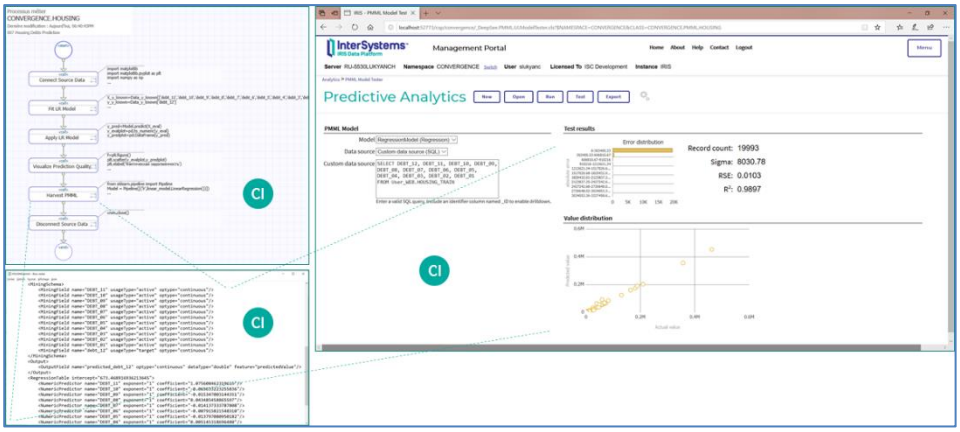


Implement Marketing Campaign Optimization showcase to discover this domain.

7.2.4. Consumption of PMML Specifications

The diagram below shows that the platform can receive from an external source a so-called PMML specification of a model that was trained in an instrument that is not being orchestrated by the platform – and then keep applying that model in real time from the moment of its PMML specification import. It is important to keep in mind that not every given AI/ML artifact can be resolved into a PMML specification, although the majority of the most widely used AI/ML artifacts allow doing this.

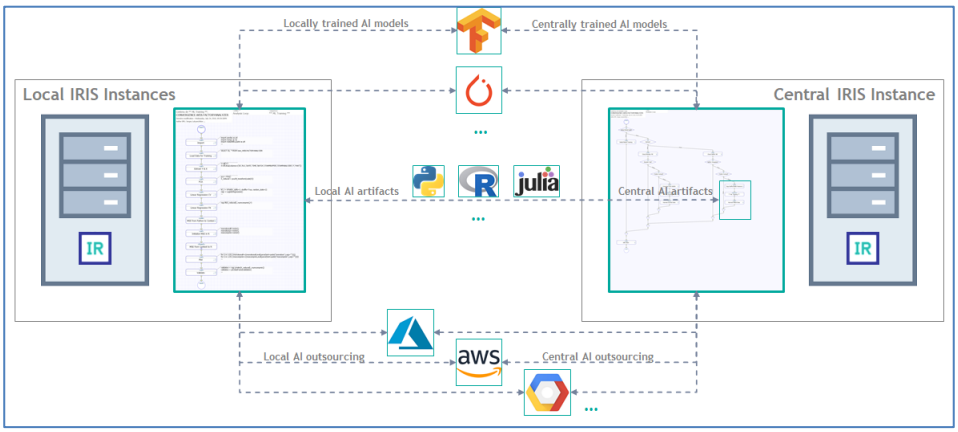
Find more information on PMML support in InterSystems IRIS.



Implement Housing Debts Prediction showcase to discover this domain.

7.3. Distributed AI/ML Computations

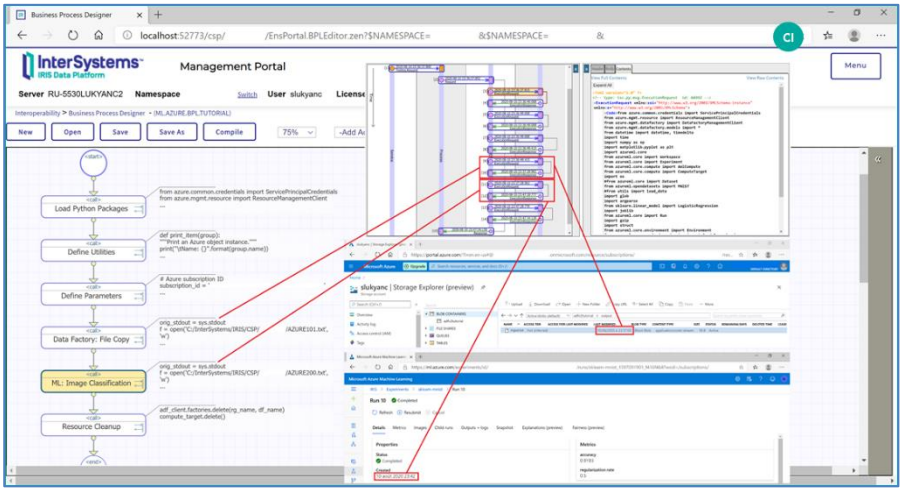
Distributed AI/ML with InterSystems IRIS is a synergy of federated data, all-purpose toolsets and specialized AI/ML on-premise as well as cloud-based frameworks:

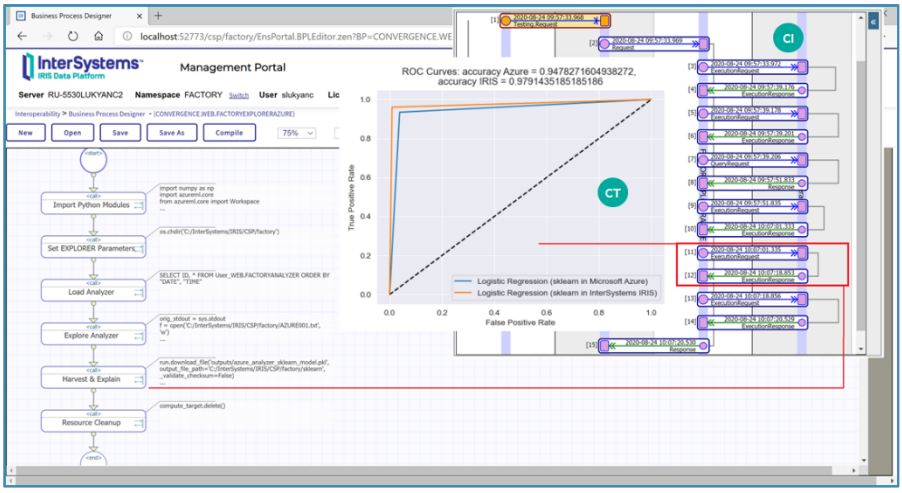


Find more information on Cloud Manager functionality in InterSystems IRIS.

7.3.1. Orchestration of Azure, AWS, GCP

In the case of a need to have a “second opinion” on the detection accuracy obtained through local computations in InterSystems IRIS, we can create an advisor process to redo the model training/application on a control dataset using cloud providers (for example: Microsoft Azure, Amazon Web Services, Google Cloud Platform, etc.):



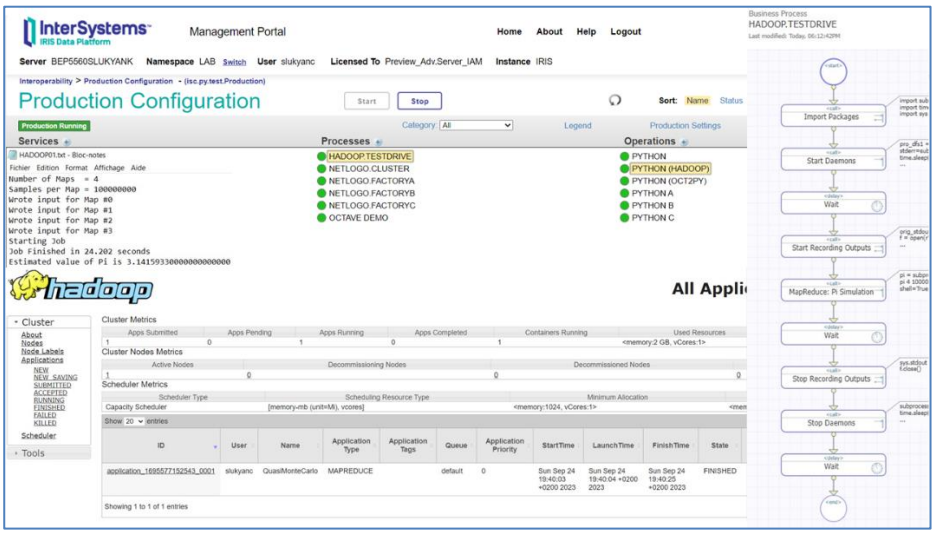


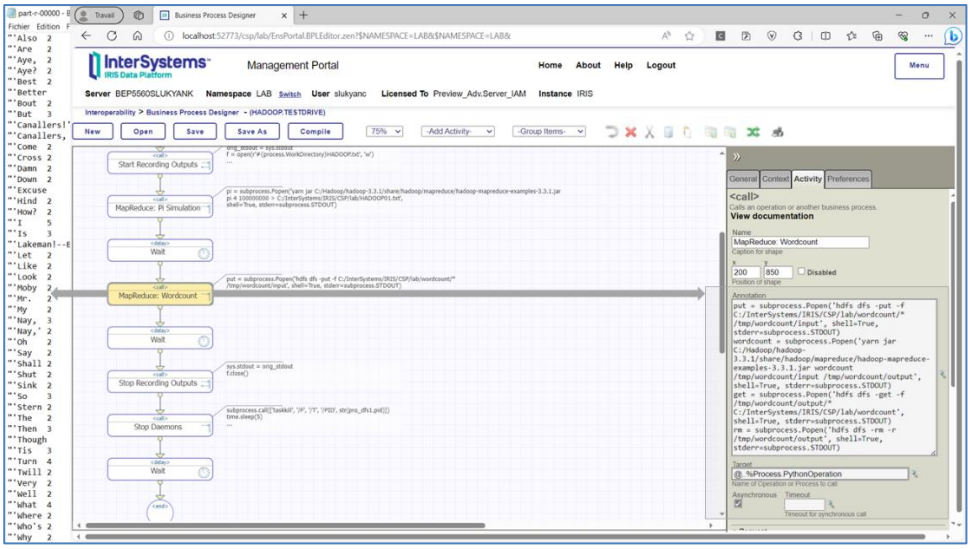

```
PYTHON>w $zv

IRIS for UNIX (Ubuntu Server LTS for x86-64 Containers) 2019.1 (Build 509U) Wed Mar 13 2019 08:35:04 EDT

PYTHON>set sc = ##class(isc.py.Main).ExecuteCode(build_sm_model)

2019-08-01 14:19:24 Starting - Starting the training job...

2019-08-01 14:19:26 Starting - Launching requested ML instances......

2019-08-01 14:20:57 Starting - Preparing the instances for training......

2019-08-01 14:21:55 Downloading - Downloading input data

2019-08-01 14:21:55 Training - Downloading the training image..

2019-08-01 14:22:09,977 sagemaker-containers INFO   Imported framework sagemaker_sklearn_container.training

2019-08-01 14:22:09,980 sagemaker-containers INFO   No GPUs detected (normal if no gpus installed)

2019-08-01 14:22:09,999 sagemaker_sklearn_container.training INFO   Invoking user training script.

...

2019-08-01 14:22:12,913 sagemaker-containers INFO   Reporting training SUCCESS

2019-08-01 14:22:20 Uploading - Uploading generated training model

2019-08-01 14:22:20 Completed - Training job completed

Billable seconds: 36

PYTHON>
```

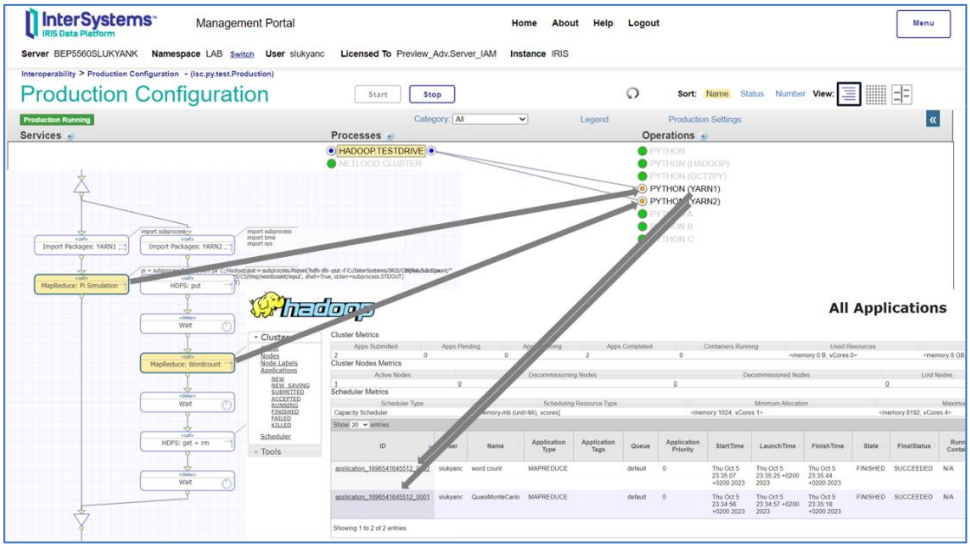
Implement Robotized ML showcase to discover this domain.

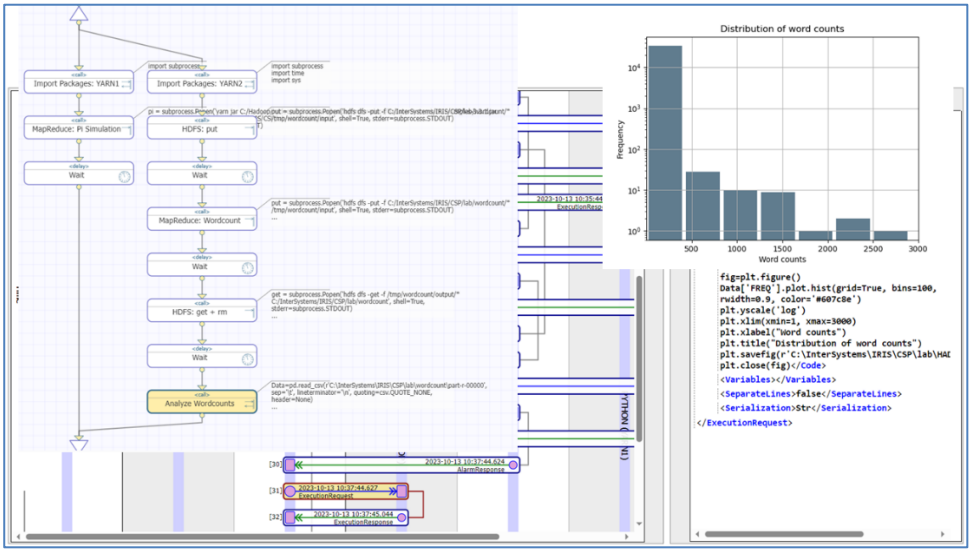
7.3.2. Orchestration of Hadoop

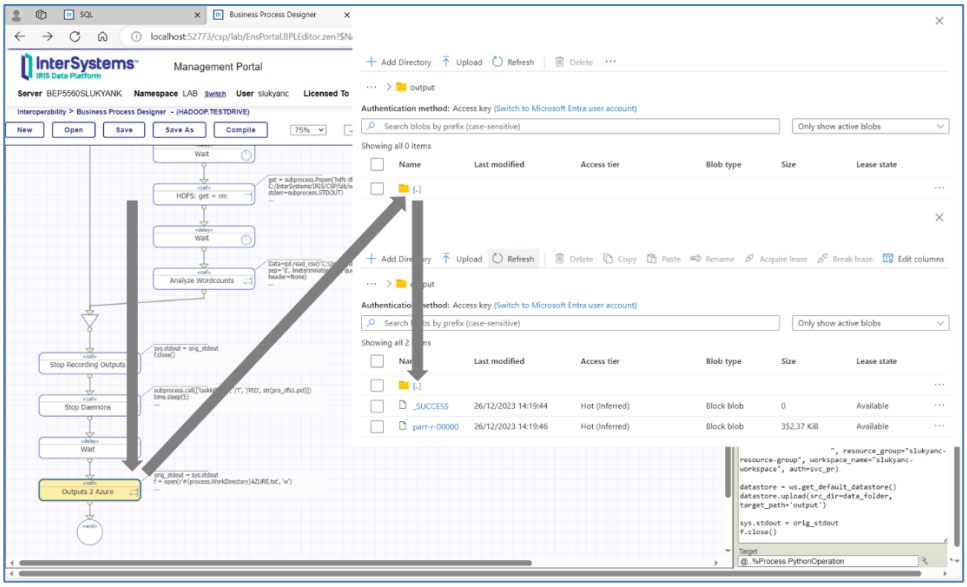
Distributed computations in their most direct sense are traditionally associated with Hadoop stack. ML Toolkit allows adding Hadoop to our “composition palette” and orchestrating lakes in parallel with clouds:




Management Portal
Start Recording Outputs
MapReduce: Pi Simulation
Wait
MapReduce: Wordcount
Stop Recording Outputs
Stop Daemons

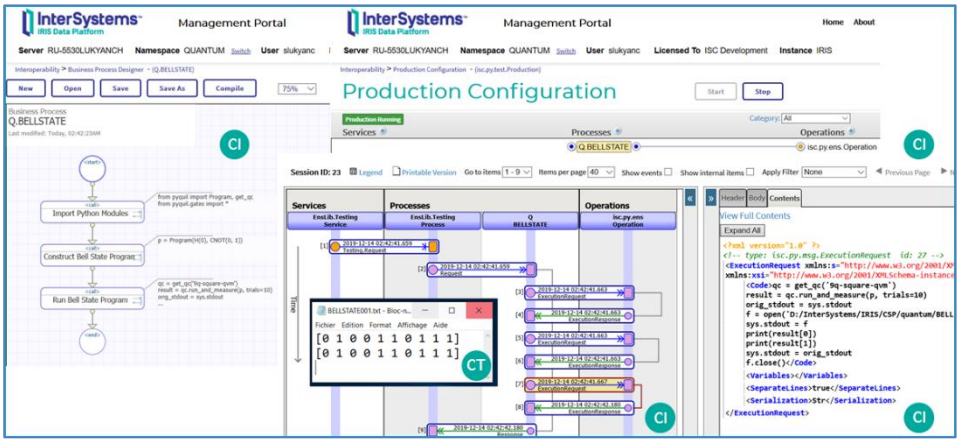

Management Portal
Production Configuration
Services
Processes
Operations
All Applications


Import Packages: YARN1
Import Packages: YARN2
MapReduce: Pi Simulation
HDFS: put
Wait
MapReduce: Wordcount
HDFS: get + rm
Analyze Wordcounts
Distribution of word counts
Frequency
Word counts



7.3.3. Computations Based on Quantum Computers

The "Bell State" example automated using ML Toolkit with Rigetti QVM running in the background:



Implement Bell State Automation showcase to discover this domain.

8. Convergent Analytics Showcases

8.1. Featured Showcases: Robotized ML

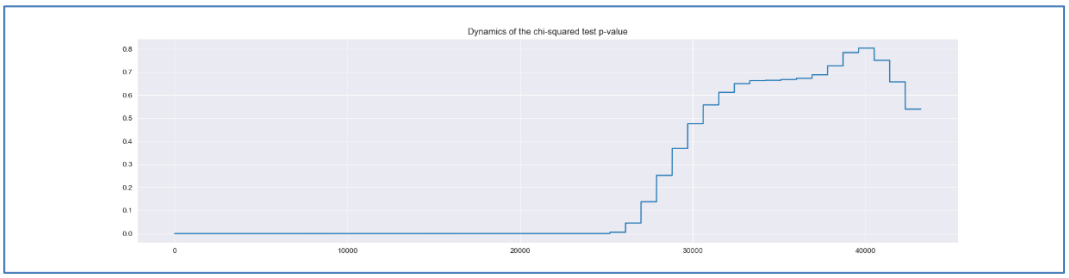
Toolsets: Python Gateway, R Gateway, Julia Gateway

Algorithms: PCA, Linear Regression, Chi-Square Test

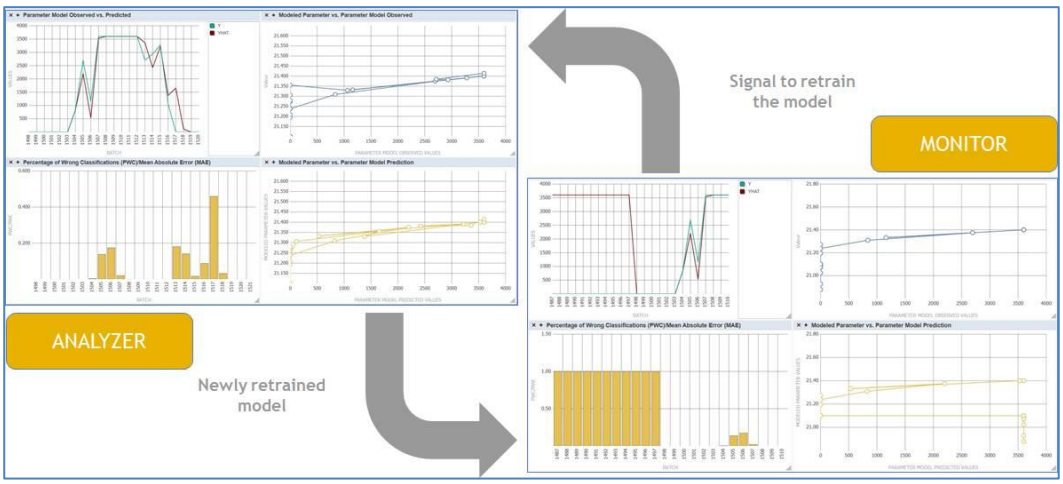
Download: read this section

8.1.1. Background

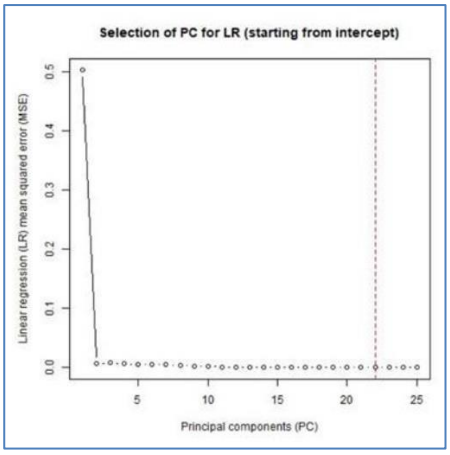
Robotized ML: we imitate data generated by a piece of equipment by forming standard-size batches of records from a file containing a large volume of previously recorded meter readings. Those batches are being read into a buffer (if the buffer is not full), we apply the chi-square test to assess how similar they are to the batch of records representing a detected defect:

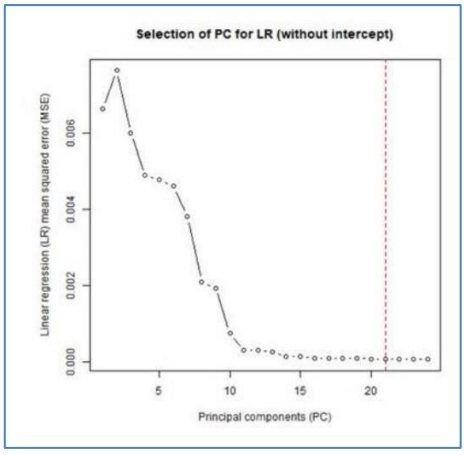


and an ML model trained on the “test-labeled” records from the buffer is used to calculate predictions (denoted YHAT) of the target “alarm or norm” variable (denoted Y). Prediction accuracy is being monitored and if it drops below a certain threshold, the model is retrained:

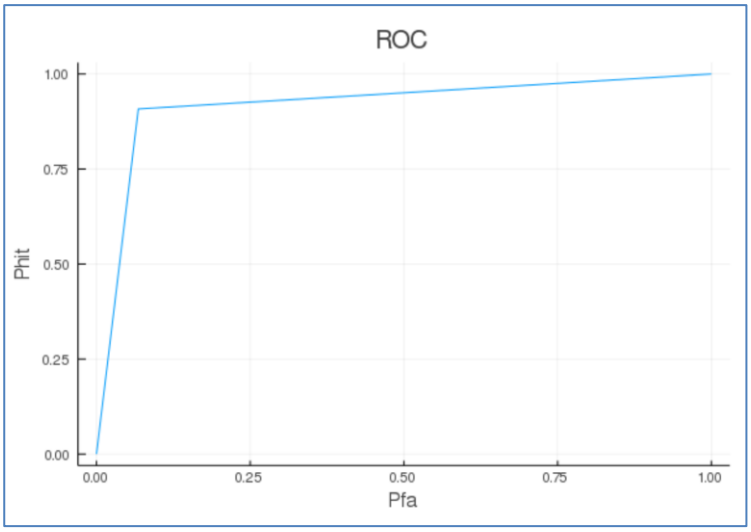


The model retraining applies a light optimization technique: from a PCA projection of the original data batch, only the principal components that secure a minimum MSE are taken into the linear regression model:





On each (re-)training, a model is validated, and the respective ROC curve is built:



8.1.2. Implementation

The showcase comprises four IRIS Interoperability processes:

- GENERATOR: produces a file with the next batch of meter reading records if the previous file was imported by BUFFER
- BUFFER: imports the file produced by GENERATOR into a buffering queue if the buffer has capacity to receive another batch of records
- ANALYZER: applies the statistical test and the ML model to records from the buffering queue (in which case it moves the processed records from the buffering queue to the monitoring queue), or trains/retrains the ML model (in which case it copies the records from the buffering queue and uses those records as a training sample after applying the statistical test)
- MONITOR: maintains the monitoring queue (by receiving new processed records from ANALYZER and deleting the oldest processed records from the monitoring queue), controls accuracy of the ML model and triggers its retraining if it drops below a given threshold

The showcase comprises forur IRIS Analytics dashboards that refresh automatically:

- Buffer Monitor: visualizes dynamics of the data in the buffering queue
- Analyzer Monitor: visualizes the data last used for model retraining and the model validation accuracy

- Monitor Monitor: visualizes dynamics of the data and the accuracy of the predictions in the monitoring queue
- Label Monitor: visualizes the dynamics of the model retraining and the model validation accuracy based on the batch of records representing a detected defect

The ML model used in this showcase is built to predict whether a record from the buffer would be scored above 0.5 by the statistical test (1) or below 0.5 (0).

8.1.3. Walkthrough

- **ACTION:** before starting the showcase, if you would like to limit the disk space consumption on your computer, execute the following commands in IRIS Terminal:
  - `zn "%SYS"`
  - `do ^JOURNAL`

  At the journaling management utility menu startup, choose the following options:
  - "2 – Stop Journaling"
  - Answer "Y" if asked whether you would like to stop journaling now

  If the showcase has been already started, left working and then stopped – all without a prior journaling stopped, in addition to the above options please add:
  - "6 – Purge Journal Files"
  - Choose sub-option "1 – Purge any journal NOT required for transaction rollback or crash recovery"

  NOTE: if the computer you are using for the showcase is not under your administration, you are requested to consult the administrator before proceeding with the above actions.
- **ACTION:** before starting the showcase, copy the file `provide.csv` to a folder chosen as the showcase working folder (**NOTE:** the folder you choose as working must be accessible for IRIS, therefore it is optimal to select the working folder under your IRIS installation path).
- **ACTION:** before starting the showcase, change the `C:\InterSystems\IRIS\CSP\factory` path (can contain either `\` or `/` characters) to your working folder in all the showcase classes.
- **ACTION:** before starting the showcase, import the records from the file `labels_only.csv` to the table `FACTORYLABELS`.
- For a more efficient control of the data queues and the model accuracy, use the following SQL queries (we recommend opening a separate IRIS Management Portal session for each of the below queries to simplify usage):
  - SELECT

    BATCH, MAX("DATE") AS "DATE", MAX("TIME") AS "TIME", COUNT(*) AS "COUNT"

    FROM User_WEB.FACTORYBUFFER

    GROUP BY BATCH
  - SELECT

    BATCH, MAX("DATE") AS "DATE", MAX("TIME") AS "TIME", SUM(Y) AS LABELED, SUM(YHAT) AS PREDICTED, COUNT(*) AS "COUNT"

    FROM User_WEB.FACTORYANALYZER

    GROUP BY BATCH
  - SELECT

```
BATCH, MAX("DATE") AS "DATE", MAX("TIME") AS "TIME", SUM(Y) AS
LABELED, SUM(YHAT) AS PREDICTED, COUNT(*) AS "COUNT"
FROM User_WEB.FACTORYMONITOR
GROUP BY BATCH
```

- To allow starting the showcase, create in the working folder the `go.txt` empty text file. To stop the showcase (all its four BPL processes) at any moment, delete the `go.txt` file from the working folder.
- **NOTE:** the order, in which you start the BPL processes is not important (they will all wait for enough action to be taken by each of them to start functioning). However, we do recommend the order below during demos to make your story easier to understand for your audience.
- Start GENERATOR process (all processes in this showcase are started by test-sending them a generic `Ens.Request` message). During demos, do not wait till a visual trace is generated – all processes in this showcase are supposed to be working indefinitely long.
- Start BUFFER process and wait till the buffering queue is filled (use the first from the SQL queries above).
- Start ANALYZER process and wait till its data table get filled (use the second from the SQL queries above), then wait till the first training of the ML model is finished (for that, use queue monitoring in IRIS Interoperability to capture the moment that `isc.py.ens.Operation`, `R.Ens.Operation` and `isc.julia.ens.Operation` finish their work and check the graphical files generated by the ML model in the working folder).
- Start MONITOR process and check that the monitoring queue gets filled with at least 12 record batches (use the third from the SQL queries above).
- Start Buffer Monitor, Analyzer Monitor and Monitor Monitor dashboards.

8.2. Featured Showcases: Robotized Sentiment Analysis (English)

Toolsets: Python Gateway

Algorithms: Word2Vec, RNN

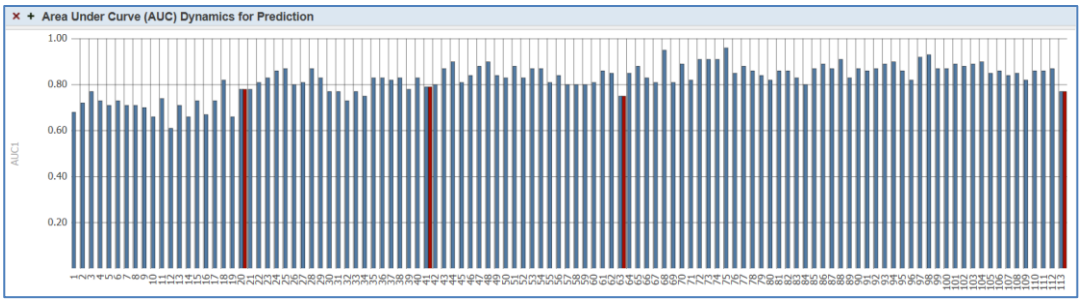
Download: read this section

Idea: https://stackabuse.com/python-for-nlp-movie-sentiment-analysis-using-deep-learning-in-keras

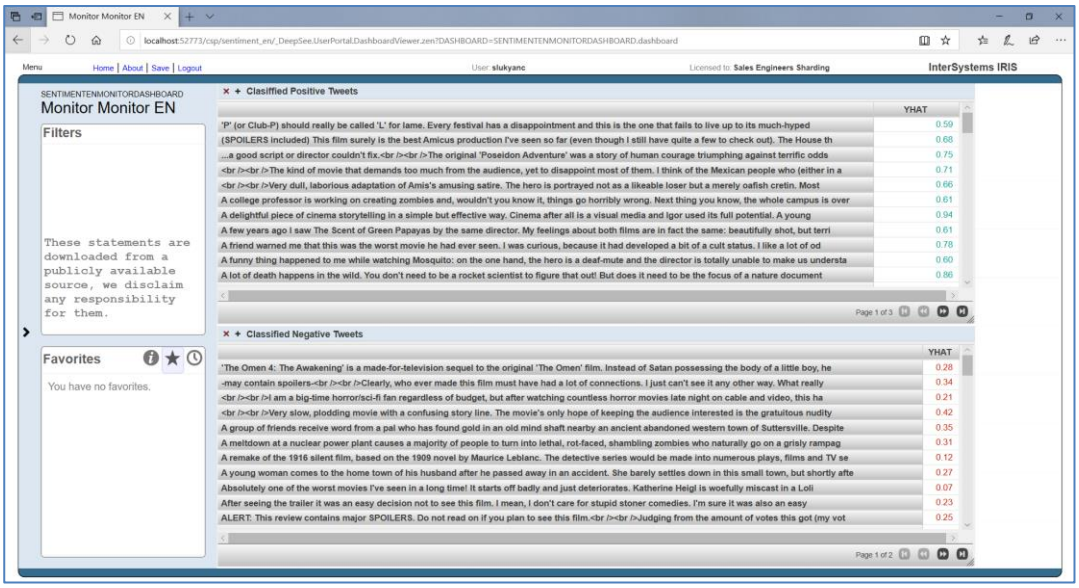
Data: https://www.kaggle.com/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews

8.2.1. Background

Robotized Sentiment Analysis: we imitate data produced by a social network subscription by forming standard-size batches of records from files containing labeled (positive and negative) and unlabeled posts in Twitter (tweets). Those batches are being read into respective buffers (if the buffers are not full), and an ML model trained on the records in the labeled buffers is used to calculate the probability of an unlabeled tweet to be positive. Prediction accuracy is being evaluated on the labeled records used for training; if it does not reach a certain accuracy threshold, the model is retrained:



The resulting sentiment classification is visualized using a dashboard:



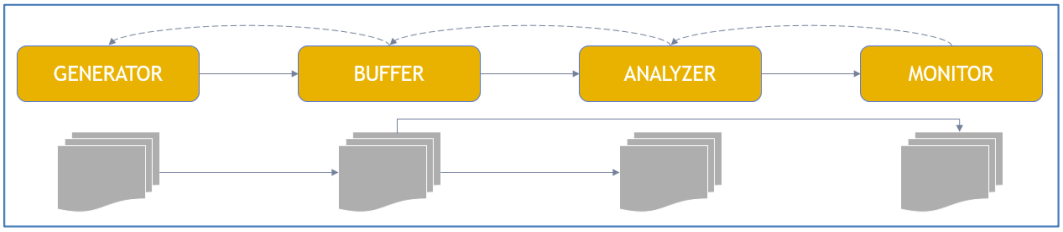
8.2.2. Implementation

The showcase comprises four IRIS Interoperability processes:

- GENERATOR: as previously generated files get consumed by the other processes, generates new files with input data (labeled – positive and negative tweets – as well as unlabeled tweets)

- BUFFER: as already buffered records are consumed by the other processes, reads new records from the files created by GENERATOR and deletes the files after having read records from them
- ANALYZER: consumes records from the unlabeled buffer and applies to them the trained RNN (recurrent neural network), transfers the “applied” records with respective “probability to be positive” values added to them, to the monitoring buffer; consumes records from labeled (positive and negative) buffers and trains the neural network based on them
- MONITOR: consumes records processed and transferred to its buffer by ANALYZER, evaluates the classification error metrics demonstrated by the neural network after the last training, and triggers new training by ANALYZER

The agent-based system of processes used in this showcase can be illustrated as follows:



The ML mechanisms used in this showcase are a file with “frozen” text vectorization results and a recurrent neural network to model sentiment; the training dataset is grown slowly to trigger multiple model training runs.

8.2.3. Walkthrough

- **ACTION:** before starting the showcase, if you would like to limit the disk space consumption on your computer, execute the following commands in IRIS Terminal:
  - `zn "%SYS"`
  - `do ^JOURNAL`

  At the journaling management utility menu startup, choose the following options:
  - “2 – Stop Journaling”
  - Answer “Y” if asked whether you would like to stop journaling now

  If the showcase has been already started, left working and then stopped – all without a prior journaling stopped, in addition to the above options please add:
  - “6 – Purge Journal Files”
  - Choose sub-option “1 – Purge any journal NOT required for transaction rollback or crash recovery”

  NOTE: if the computer you are using for the showcase is not under your administration, you are requested to consult the administrator before proceeding with the above actions.
- **ACTION:** before starting the showcase, copy the files `provide*.csv` and `glove.6B.100d.txt` to a folder chosen as the showcase working folder (**NOTE:** the folder you choose as working must be accessible for IRIS, therefore it is optimal to select the working folder under your IRIS installation path).
- For a more efficient control of the data queues and the model accuracy, use the following SQL queries (we recommend opening a separate IRIS Management Portal session for each of the below queries to simplify usage):
  - SELECT

BATCH, COUNT(*) AS RECORDS, ABS(ROUND(MAX(STAMP)-MIN(STAMP),2)) AS THRUPUT

FROM User_WEB.SENTIMENTENBUFFERUNK (or else, *POS and *NEG)

GROUP BY BATCH

ORDER BY BATCH ASC

- SELECT

  BATCH, COUNT(*) AS RECORDS

  FROM User_WEB.SENTIMENTENANALYZERPOS (also, *NEG)

  GROUP BY BATCH

  ORDER BY BATCH ASC

- SELECT

  BATCH, COUNT(*) AS RECORDS

  FROM User_WEB.SENTIMENTENMONITOR

  GROUP BY BATCH

  ORDER BY BATCH ASC

- SELECT *

  FROM User_WEB.SENTIMENTENANALYZERAUC

- To allow starting the showcase, create in the working folder the `go.txt` empty text file. To stop the showcase (all its four BPL processes) at any moment, delete the `go.txt` file from the working folder.
- **NOTE:** the order, in which you start the BPL processes is not important (they will all wait for enough action to be taken by each of them to start functioning). However, we do recommend the order below during demos to make your story easier to understand for your audience.
- Start GENERATOR process (all processes in this showcase are started by test-sending them a generic `Ens.Request` message). During demos, do not wait till a visual trace is generated – all processes in this showcase are supposed to be working indefinitely long.
- Start BUFFER process and wait till the buffering queues are filled (use the first from the SQL query templates above).
- Start ANALYZER process and wait till its data tables get filled (use the second and the fourth from the SQL query templates above), then wait till the first training of the ML models is finished (for that, use queue monitoring in IRIS Interoperability to capture the moment that `isc.py.ens.Operation` finishes its work and check the text/graphical files generated by the ML models in the working folder).
- Start MONITOR process and check that its data tables get filled (use the third and the fourth from the SQL query templates above).

Start Buffer Monitor, Analyzer Monitor and Monitor Monitor dashboards.

8.3. Featured Showcases: Robotized Sentiment Analysis (Russian)

Toolsets: Python Gateway

Algorithms: Word2Vec, CNN

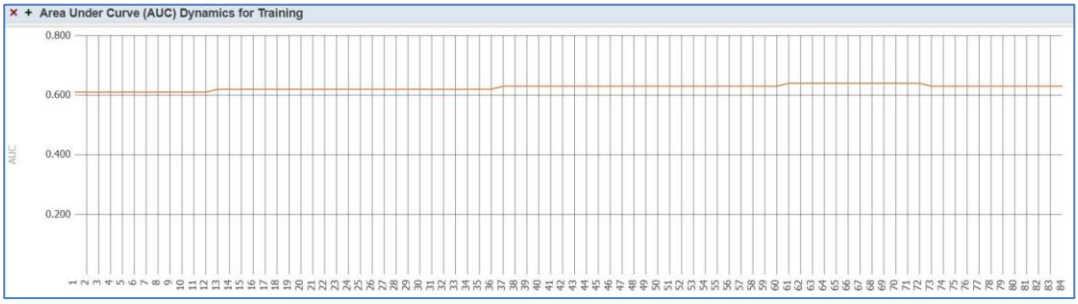
Download: read this section

Idea: https://github.com/sismetanin/sentiment-analysis-of-tweets-in-russian

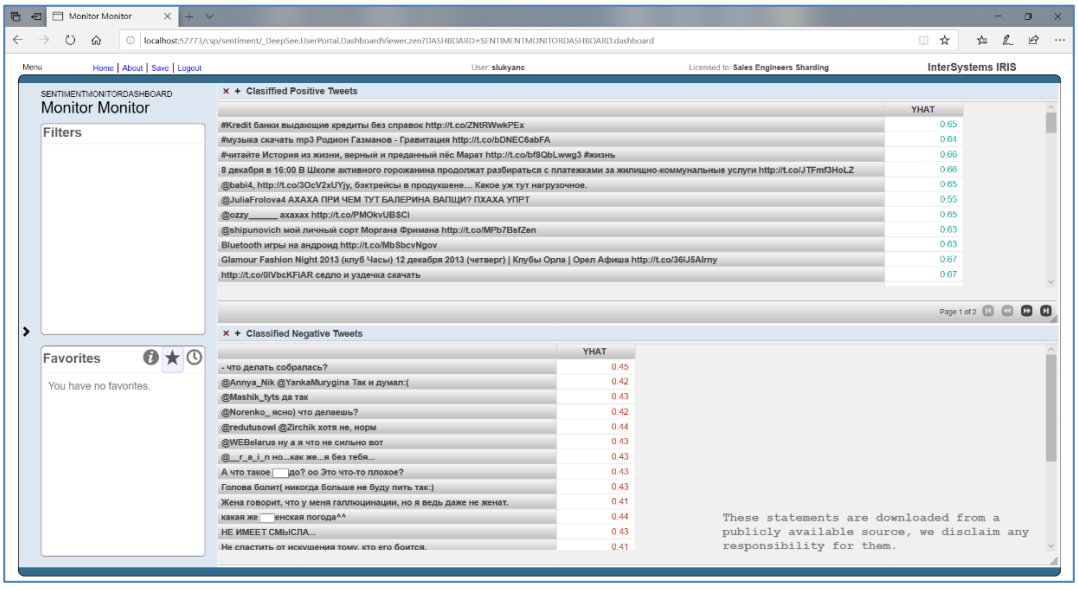
Data: http://study.mokoron.com/

8.3.1. Background

Robotized Sentiment Analysis: we imitate data produced by a social network subscription by forming standard-size batches of records from files containing labeled (positive and negative) and unlabeled posts in Twitter (tweets). Those batches are being read into respective buffers (if the buffers are not full), and an ML model trained on the records in the labeled buffers is used to calculate the probability of an unlabeled tweet to be positive. Prediction accuracy is being evaluated on the labeled records used for training; if it does not reach a certain accuracy threshold, the model is retrained:



The resulting sentiment classification is visualized using a dashboard:



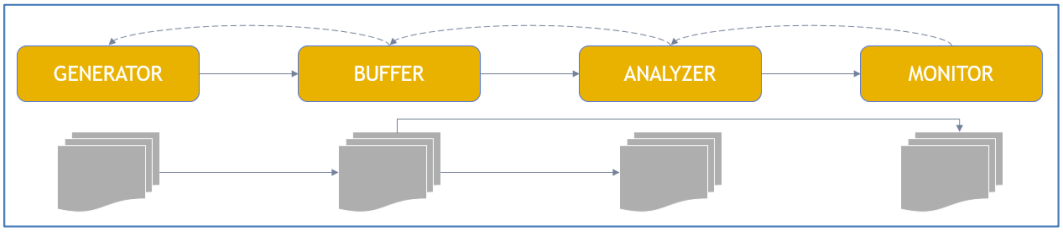
8.3.2. Implementation

The showcase comprises four IRIS Interoperability processes:

- GENERATOR: as previously generated files get consumed by the other processes, generates new files with input data (labeled – positive and negative tweets – as well as unlabeled tweets)

- BUFFER: as already buffered records are consumed by the other processes, reads new records from the files created by GENERATOR and deletes the files after having read records from them
- ANALYZER: consumes records from the unlabeled buffer and applies to them the trained CNN (convolutional neural network), transfers the “applied” records with respective “probability to be positive” values added to them, to the monitoring buffer; consumes records from labeled (positive and negative) buffers and trains the neural network based on them
- MONITOR: consumes records processed and transferred to its buffer by ANALYZER, evaluates the classification error metrics demonstrated by the neural network after the last training, and triggers new training by ANALYZER

The agent-based system of processes used in this showcase can be illustrated as follows:



The ML mechanisms used in this showcase are a text vectorizer and a convolutional neural network to model sentiment; the training dataset is grown slowly to trigger multiple model training runs.

8.3.3. Walkthrough

- **ACTION:** before starting the showcase, if you would like to limit the disk space consumption on your computer, execute the following commands in IRIS Terminal:
  - `zn "%SYS"`
  - `do ^JOURNAL`

  At the journaling management utility menu startup, choose the following options:
  - “2 – Stop Journaling”
  - Answer “Y” if asked whether you would like to stop journaling now

  If the showcase has been already started, left working and then stopped – all without a prior journaling stopped, in addition to the above options please add:
  - “6 – Purge Journal Files”
  - Choose sub-option “1 – Purge any journal NOT required for transaction rollback or crash recovery”

  NOTE: if the computer you are using for the showcase is not under your administration, you are requested to consult the administrator before proceeding with the above actions.
- **ACTION:** before starting the showcase, copy the files `provide*.csv` to a folder chosen as the showcase working folder (**NOTE:** the folder you choose as working must be accessible for IRIS, therefore it is optimal to select the working folder under your IRIS installation path).
- For a more efficient control of the data queues and the model accuracy, use the following SQL queries (we recommend opening a separate IRIS Management Portal session for each of the below queries to simplify usage):
  - SELECT

BATCH, COUNT(*) AS RECORDS, ABS(ROUND(MAX(STAMP)-MIN(STAMP),2)) AS THRUPUT

FROM User_WEB.SENTIMENTRUBUFFERUNK (or else, *POS and *NEG)

GROUP BY BATCH

ORDER BY BATCH ASC

- SELECT

  BATCH, COUNT(*) AS RECORDS

  FROM User_WEB.SENTIMENTRUANALYZERPOS (also, *NEG)

  GROUP BY BATCH

  ORDER BY BATCH ASC

- SELECT

  BATCH, COUNT(*) AS RECORDS

  FROM User_WEB.SENTIMENTRUMONITOR

  GROUP BY BATCH

  ORDER BY BATCH ASC

- To allow starting the showcase, create in the working folder the `go.txt` empty text file. To stop the showcase (all its four BPL processes) at any moment, delete the `go.txt` file from the working folder.
- **NOTE:** the order, in which you start the BPL processes is not important (they will all wait for enough action to be taken by each of them to start functioning). However, we do recommend the order below during demos to make your story easier to understand for your audience.
- Start GENERATOR process (all processes in this showcase are started by test-sending them a generic `Ens.Request` message). During demos, do not wait till a visual trace is generated – all processes in this showcase are supposed to be working indefinitely long.
- Start BUFFER process and wait till the buffering queues are filled (use the first from the SQL query templates above).
- Start ANALYZER process and wait till its data tables get filled (use the second from the SQL query templates above), then wait till the first training of the ML models is finished (for that, use queue monitoring in IRIS Interoperability to capture the moment that `isc.py.ens.Operation` finishes its work and check the text/graphical files generated by the ML models in the working folder).
- Start MONITOR process and check that the monitoring queue gets filled with several record batches (use the third from the SQL query templates above).
- Start Buffer Monitor, Analyzer Monitor and Monitor Monitor dashboards.

8.4. 001 Sentiment Analysis

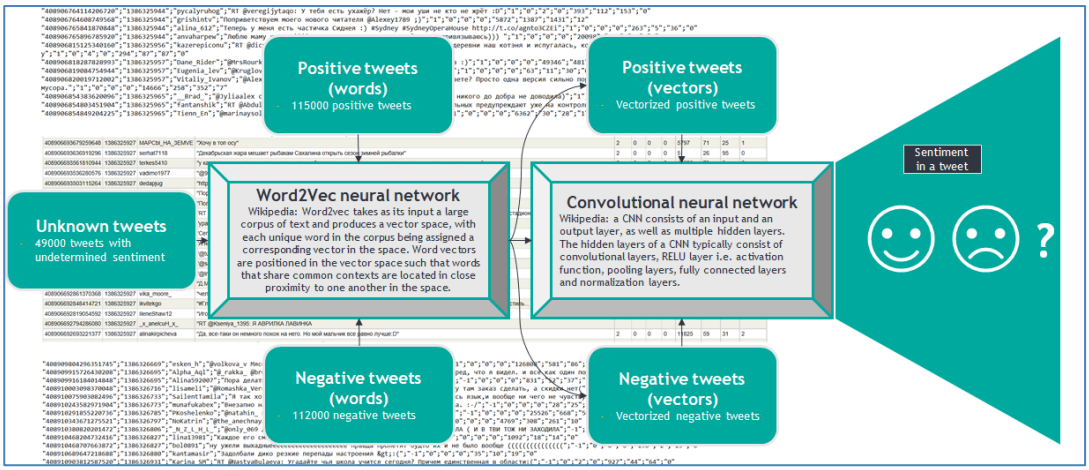
Toolsets: Python Gateway

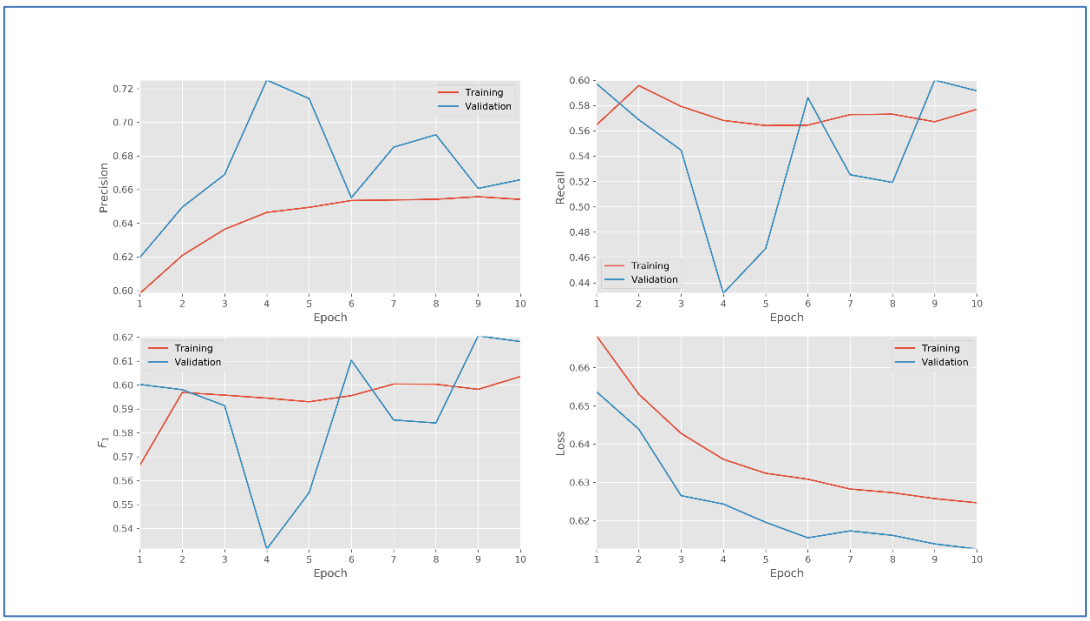
Algorithms: Word2Vec, CNN

Download: read this section

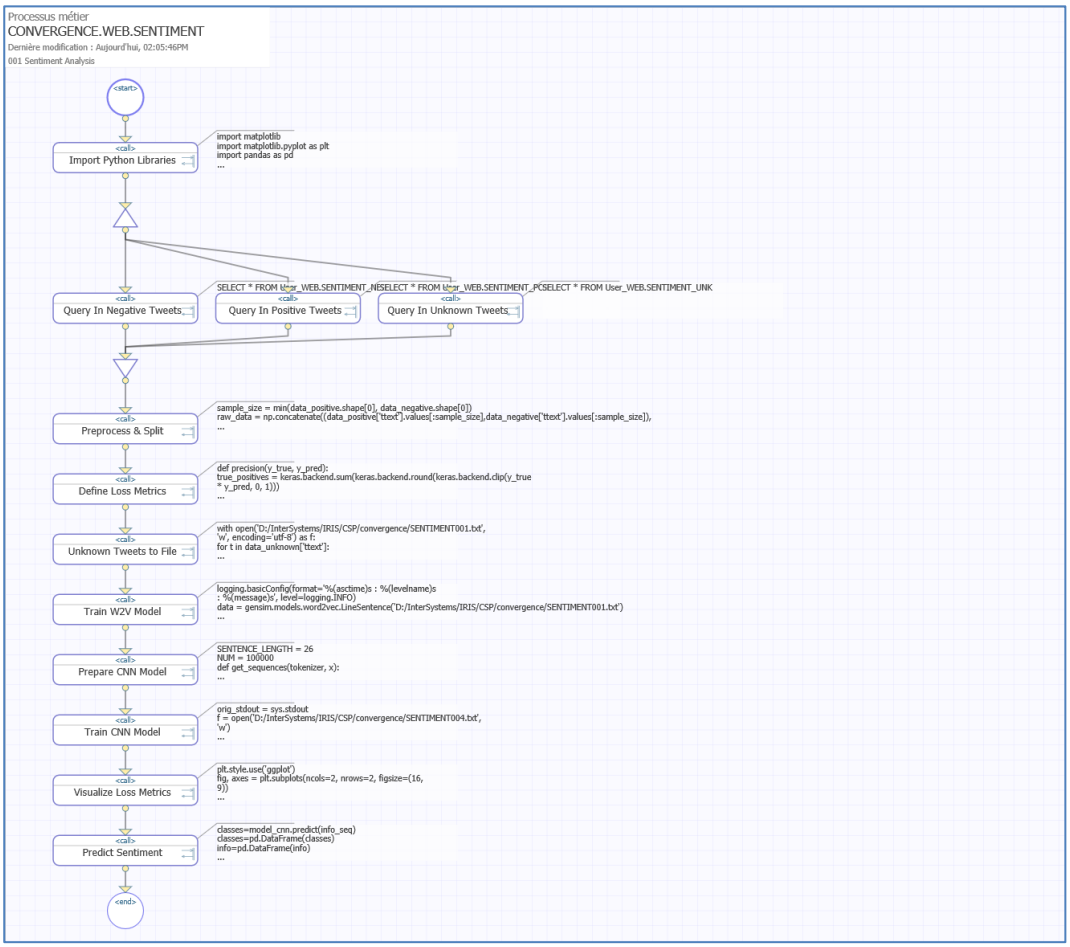
8.4.1. Background

Sentiment analysis: we transform words in the tweets into vectors, then train a model that relies on vector representation to establish proximity of a given tweet to positive or negative tweets (based on vectorized samples of both).





8.4.2. Implementation



8.4.3. Walkthrough

- Import Python Libraries: loads the required libraries to Python context.
- Query In Negative Tweets: loads a sample of negative tweets. **ACTION:** implement the data tables in the namespace(s) that fit your environment.
- Query In Positive Tweets: loads a sample of positive tweets.
- Query In Unknown Tweets: loads a sample of the tweets to be classified as either negative or positive.
- Preprocess & Split: adjusts texts in the negative and positive samples to ensure efficient processing via the vectorization and neural network components. **ACTION:** adjust the file paths to fit your environment.
- Define Loss Metrics: defines functions to calculate loss metrics that measure classification accuracy.
- Unknown Tweets to File: writes preprocessed texts from the unknown sample into a text file to be fed in the vectorization component.
- Train W2V Model: converts texts to numerical vectors to enable computations using the neural network.
- Prepare CNN Model: defines and initializes a convolutional neural network (CNN).
- Train CNN Model: trains and validates the CNN model.
- Visualize Loss Metrics: visualizes the progress that the loss metrics are making as epochs are being processed.

- Predict Sentiment: scores unknown tweets for their sentiment probability to be positive (scores are closer to 1) or negative (scores are closer to 0)

8.5. 002 Engine Condition Classification

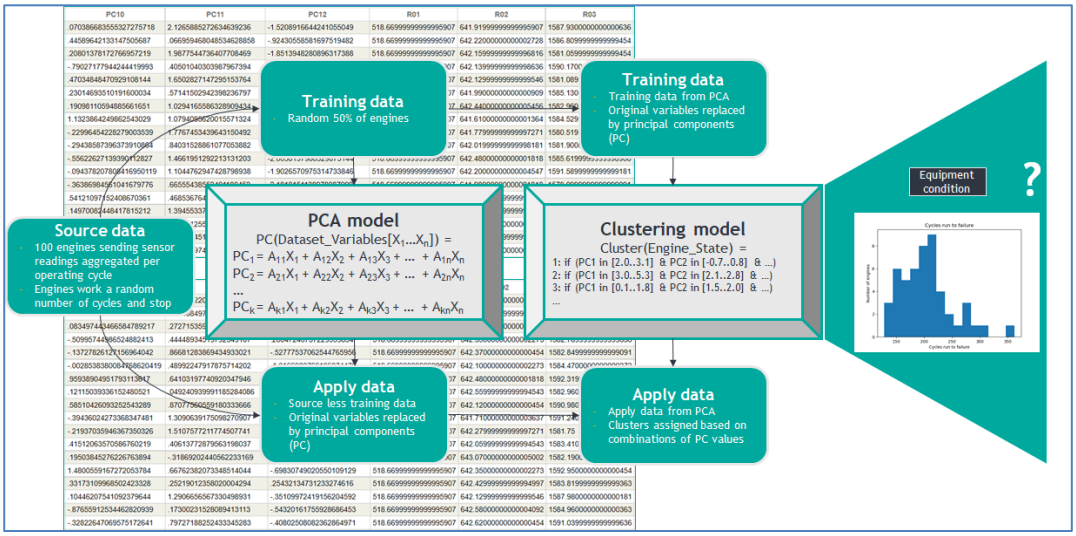
Toolsets: ObjectScript, Python Gateway

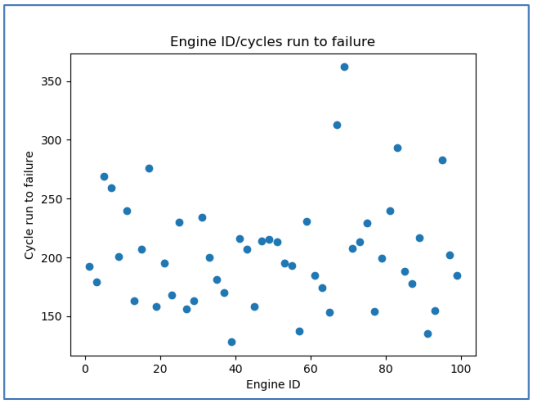
Algorithms: PCA, PAM, CLARA

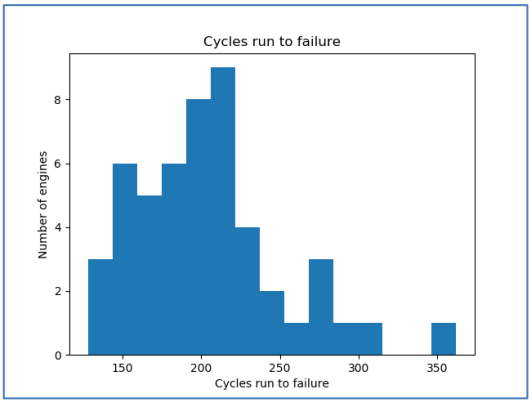
Download: read this section

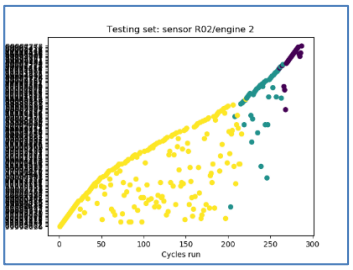
8.5.1. Background

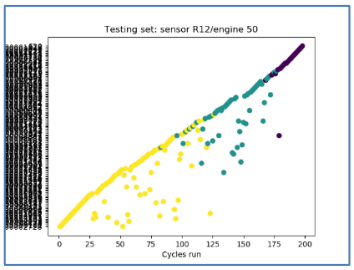
Object condition modeling: by applying dimensionality reduction and clustering methods to object state data, we dynamically determine presence of an object in “start”, “mid” or “end” phase of its lifecycle. Modeling results allow having a more efficient replacement process while avoiding unpredicted failures.

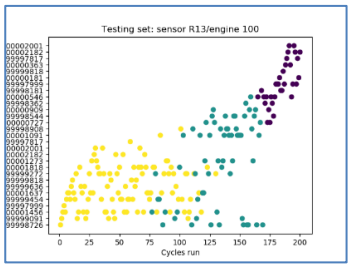




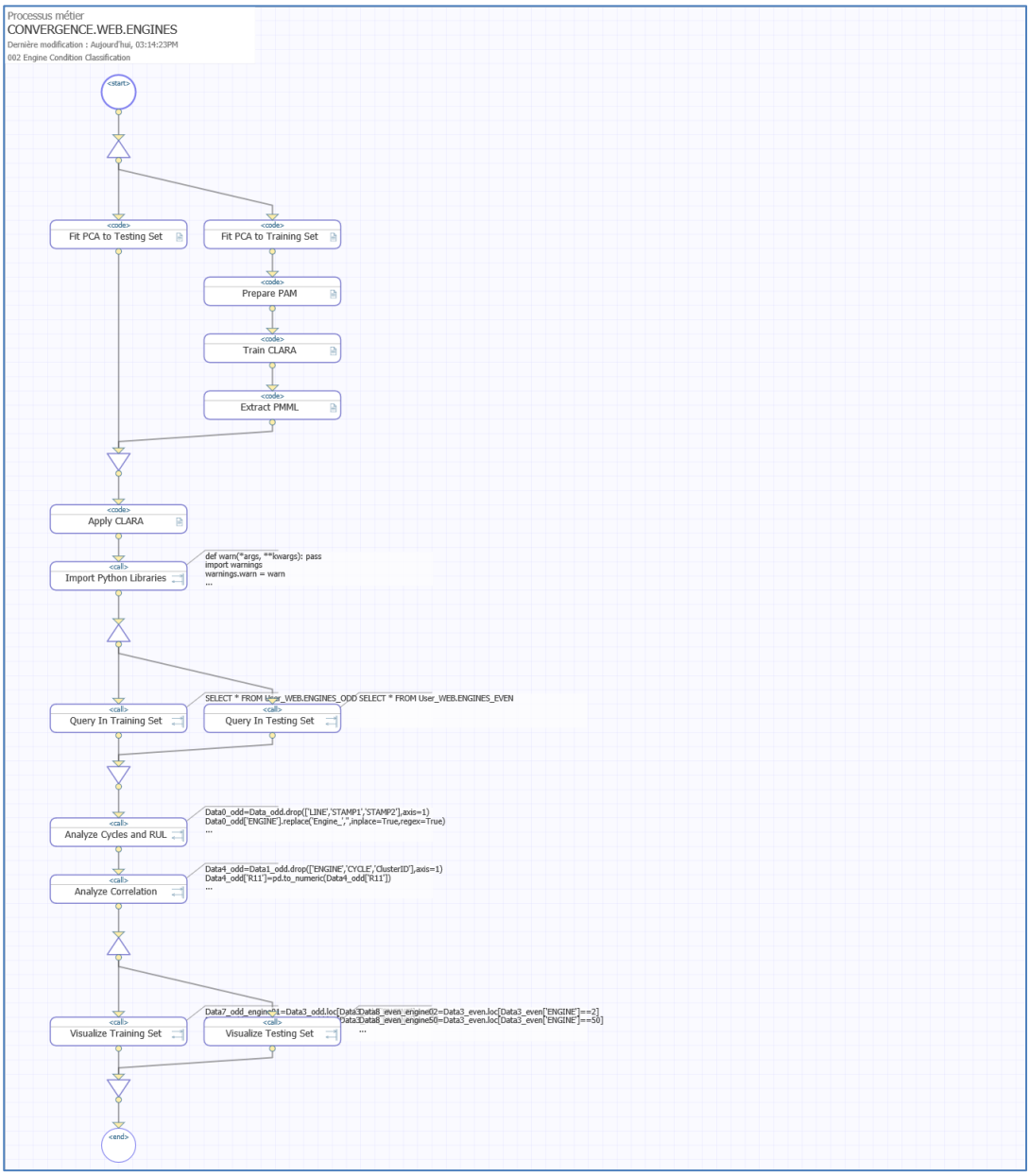








8.5.2. Implementation



8.5.3. Walkthrough

1. Fit PCA to Testing Set: calculates principal components representing the testing set of engines.
2. Fit PCA to Training Set: calculates principal components representing the training set of engines.
3. Prepare PAM: prepares the clustering model for being trained.
4. Train CLARA: trains the clustering model using a concrete algorithm.
5. Extract PMML: extracts clustering rules from the trained clustering model and converts them to PMML format.
6. Apply CLARA: applies the clustering model to the testing set to predict cluster numbers.
7. Import Python Libraries: loads the required libraries to Python context.

8. Query In Training Set: loads training data. **ACTION:** implement the data tables in the namespace(s) that fit your environment.
9. Query In Testing Set: loads testing data.
10. Analyze Cycles and RUL: calculate additional metrics needed for further visual analysis of survived cycles and remaining useful life (RUL). **ACTION:** adjust the file paths to fit your environment.
11. Analyze Correlation: calculate correlation matrices to study dependencies among variables.
12. Visualize Training Set: generate several graphs to support visual analysis and validation of modeling results using training data.
13. Visualize Testing Set: generate several graphs to support visual analysis and verification of modeling results using testing data.

8.6. 003 Reimbursement Request Check

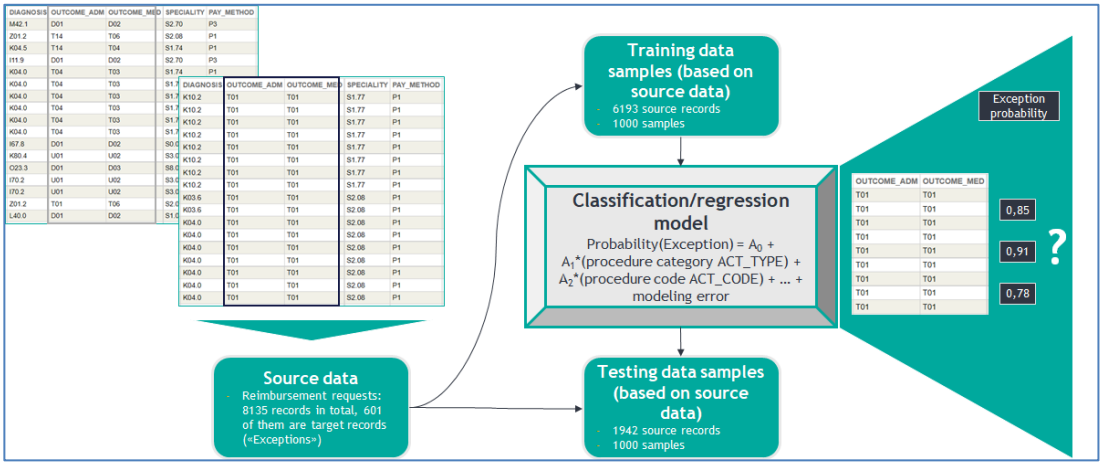
Toolsets: Python Gateway

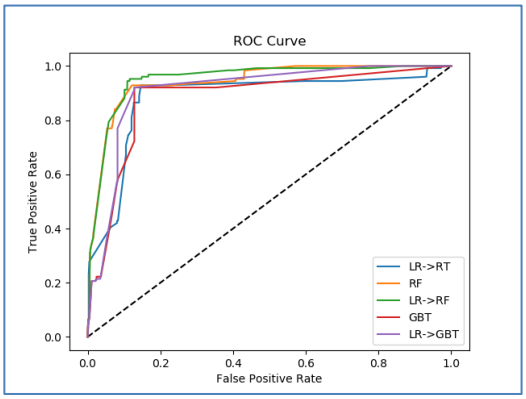
Algorithms: LR, RT, RF, GB, OneHotEncoder

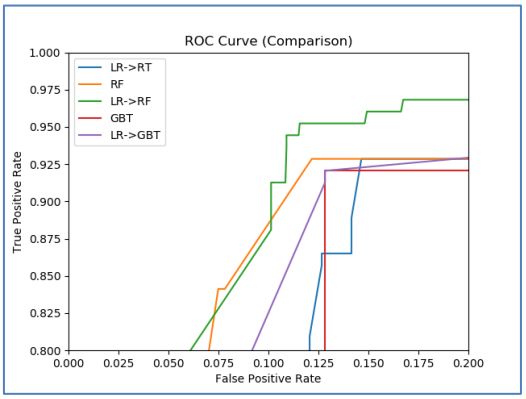
Download: read this section

8.6.1. Background

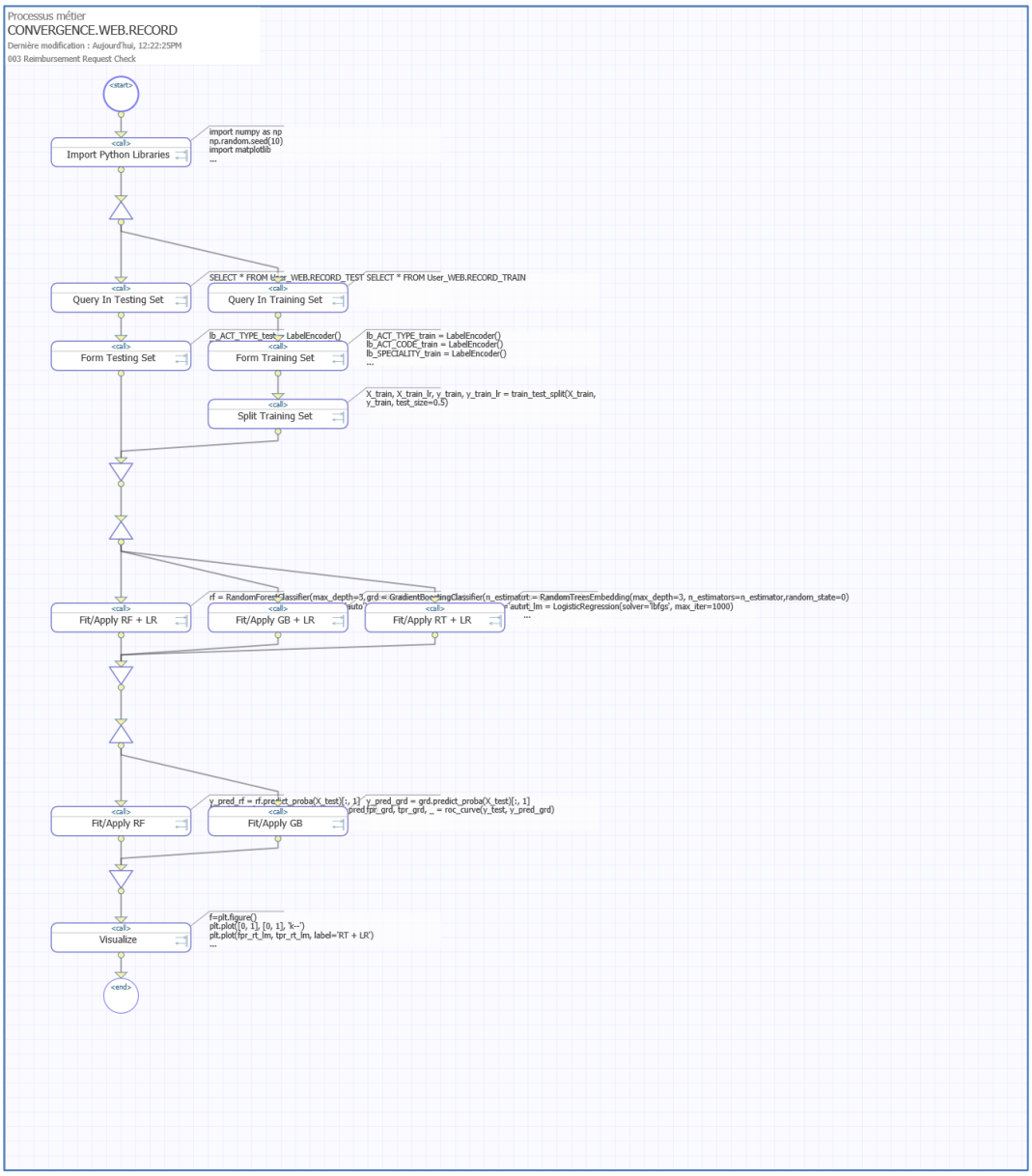
A classification model trained on samples of various types of deviations, detects them automatically in a flow of transactions.







8.6.2. Implementation



8.6.3. Walkthrough

1. Import Python Libraries: loads the required libraries to Python context.
2. Query In Testing Set: loads testing data. **ACTION:** implement the data tables in the namespace(s) that fit your environment.
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. Fit/Apply RT + LR: fits and applies a bundle of random tree and linear regression models (evaluation and testing modes)
8. Fit/Apply RF + LR: fits and applies a bundle of random forest and linear regression models (evaluation and testing modes)

9. Fit/Apply GB + LR: fits and applies a bundle of gradient boosting and linear regression models (evaluation and testing modes)
10. Fit/Apply RF: fits and applies a random forest model (evaluation and testing modes)
11. Fit/Apply GB: fits and applies a gradient boosting model (evaluation and testing modes)
12. Visualize ROC Curves: saves to graphical files the ROC charts (makes possible defining the winner model). **ACTION:** adjust the file paths to fit your environment.

8.7. 004 Retail Cannibalization Analysis

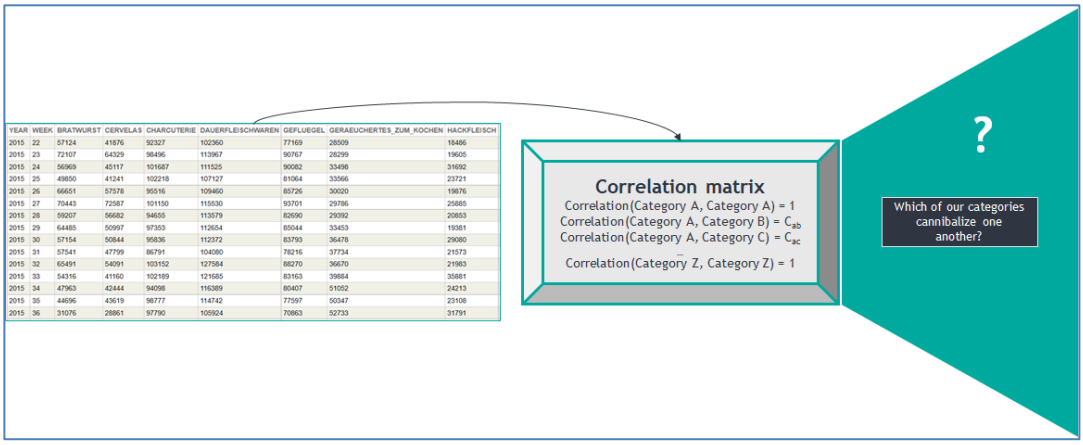
Toolsets: Python Gateway

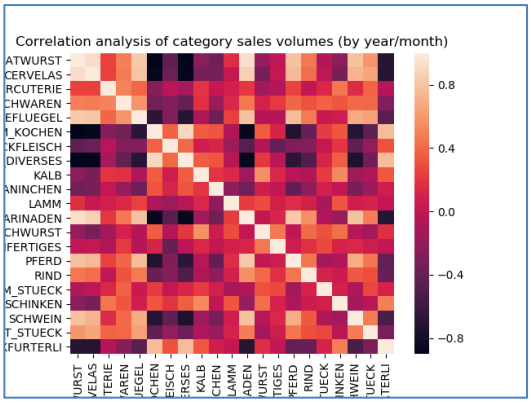
Algorithms: Statistical Functions

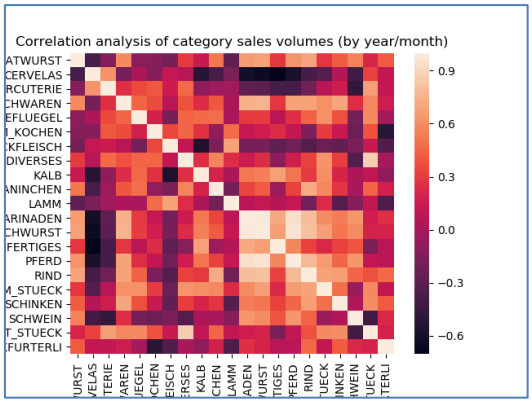
Download: read this section

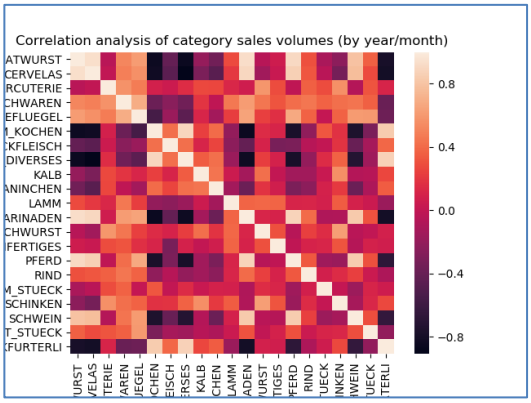
8.7.1. Background

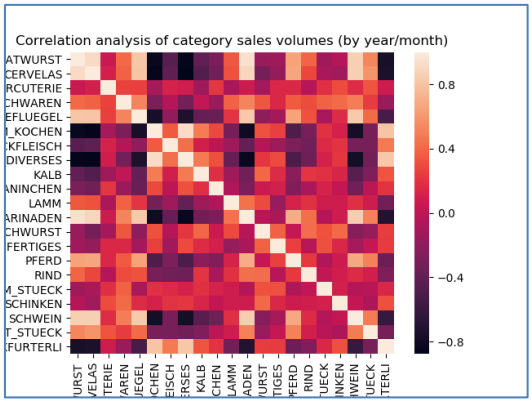
Correlation analysis is a universal method to discover potential mutual influence among a set of variables. One of the examples is product cannibalization in retail: certain product categories lose in sales volume if their “cannibalizing” counterparts show growing sales volume.



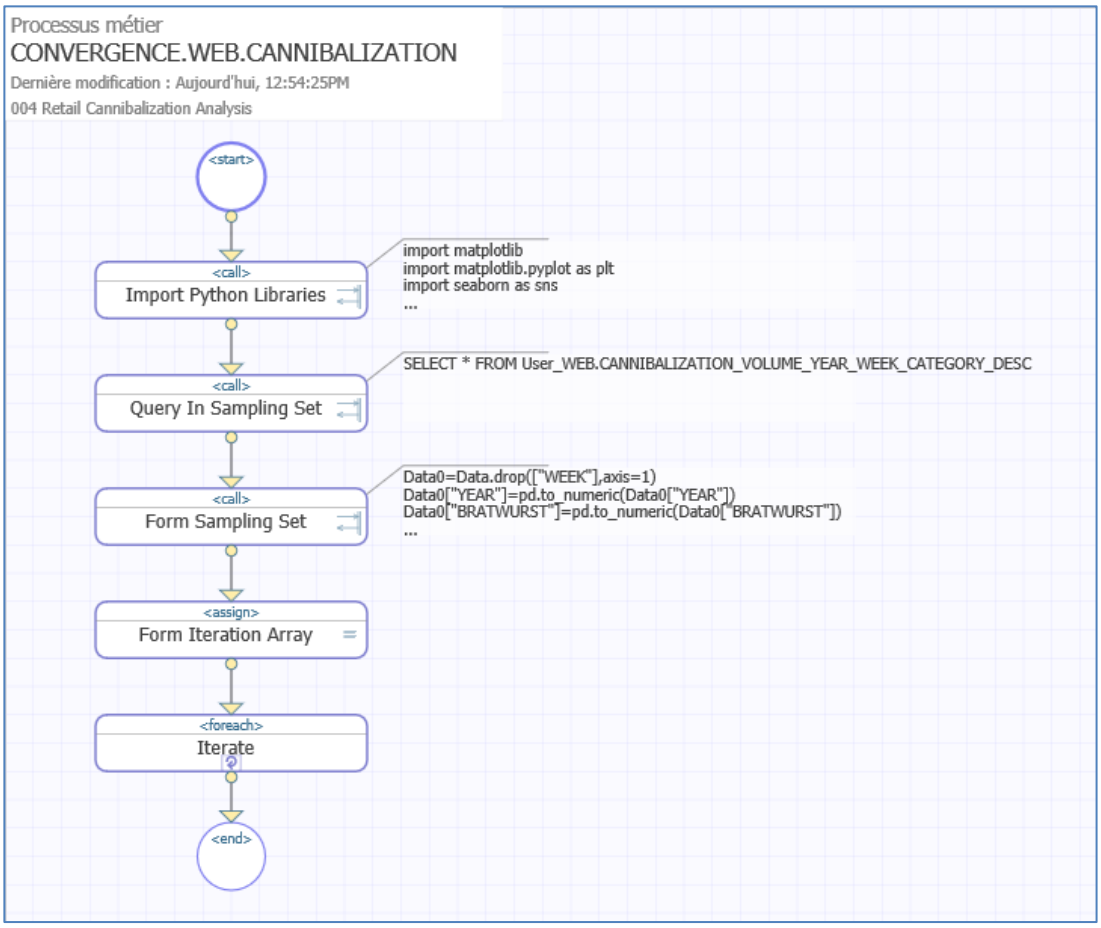








8.7.2. Implementation



8.7.3. Walkthrough

1. Import Python Libraries: loads the required libraries to Python context.
2. Query In Sampling Set: loads sampling data. **ACTION:** implement the data tables in the namespace(s) that fit your environment.
3. Form Sampling Set: prepares sampling data for a correlation analysis
4. Form Iteration Array: forms an array to iterate over while doing the correlation analysis (next step)
5. Iterate: iterates over the sampling data and builds sample correlation matrices per each iteration

8.8. 005 Marketing Campaign Optimization

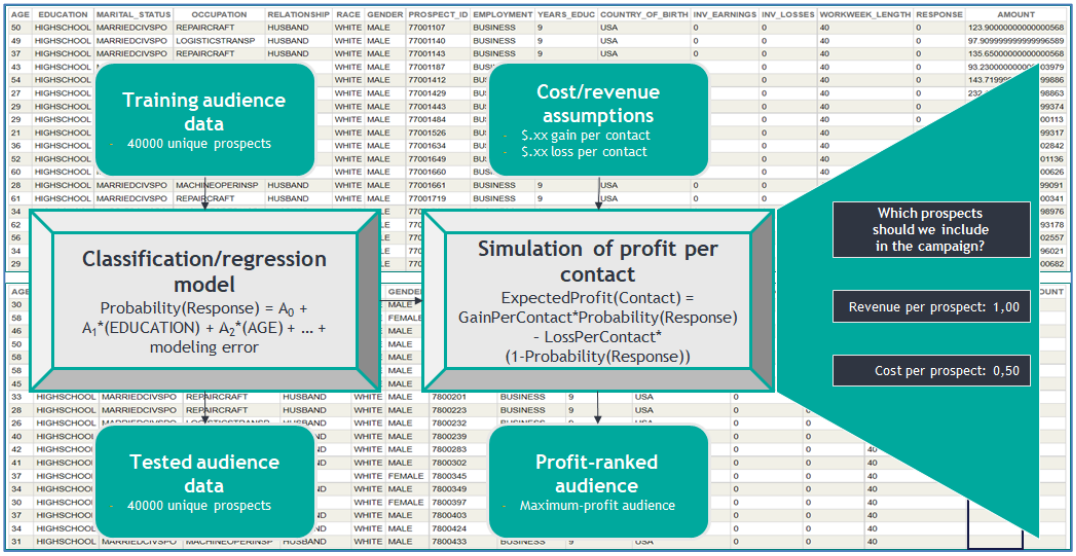
Toolsets: Python Gateway, ObjectScript, IntegratedML, H2O

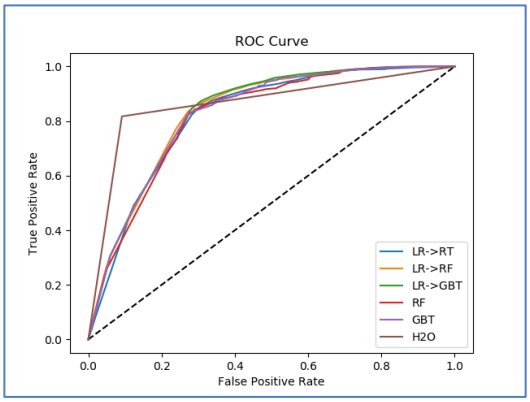
Algorithms: LR, RT, RF, GB, OneHotEncoder, AutoML

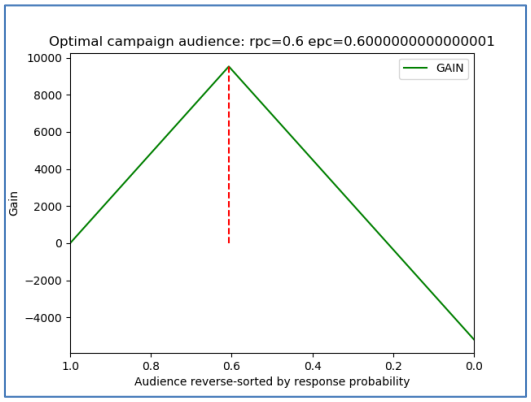
Download: read this section

8.8.1. Background

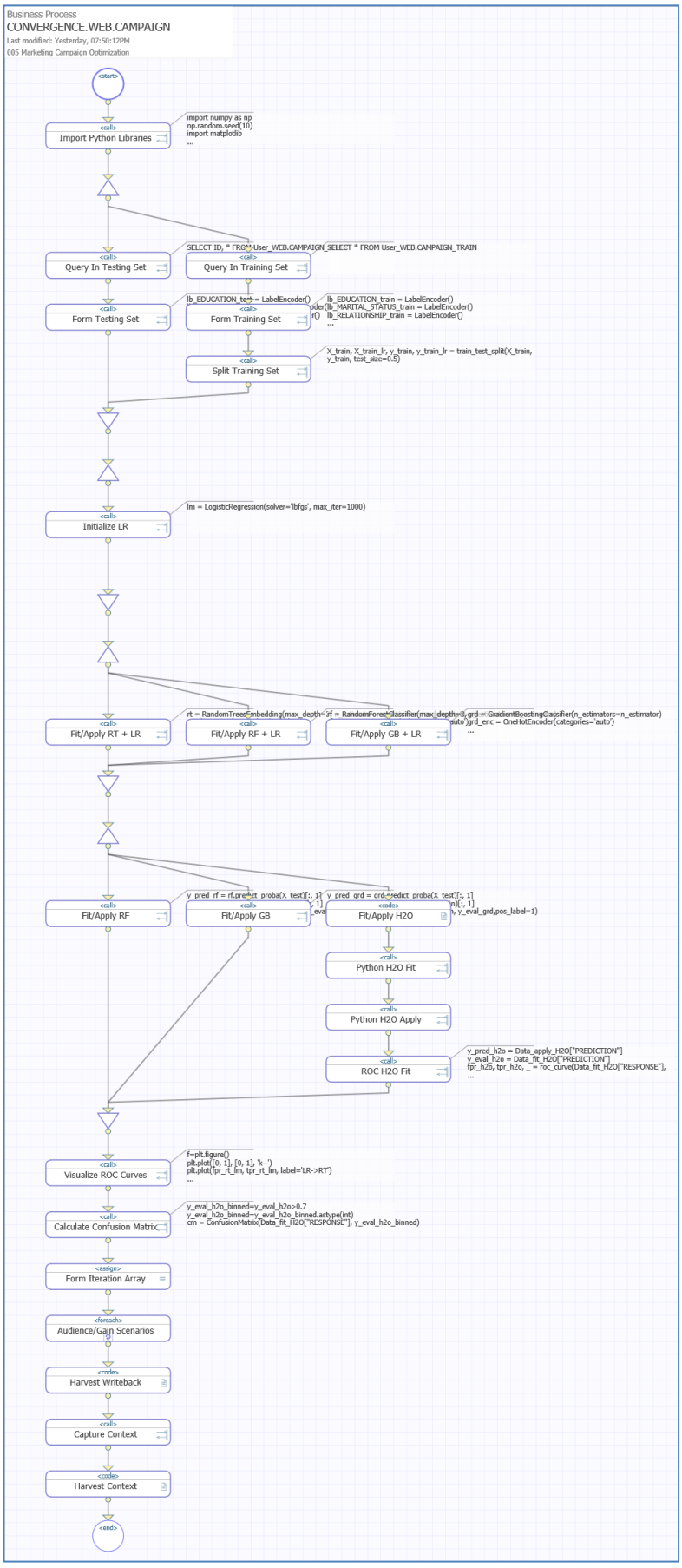
We implement a stack of classification models to have a robust estimate of response probability. We then use response probabilities to calculate the expected gain per prospect. We finally reverse-sort the audience on response probability and estimate the maximum summary gain for the campaign, as well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







8.8.2. Implementation



8.8.3. Walkthrough

1. Import Python Libraries: loads the required libraries to Python context.
2. Query In Testing Set: loads testing data. **ACTION:** implement the data tables in the namespace(s) that fit your environment.
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 validation modes)
9. Fit/Apply RF + LR: fits and applies a bundle of random forest and linear regression models (evaluation and validation modes)
10. Fit/Apply GB + LR: fits and applies a bundle of gradient boosting and linear regression models (evaluation and validation modes)
11. Fit/Apply RF: fits and applies a random forest model (evaluation and validation modes)
12. Fit/Apply GB: fits and applies a gradient boosting model (evaluation and validation modes)
13. Fit/Apply H2O, Python H2O Fit, Python H2O Apply, ROC H2O Fit: fits and applies an automated ML classification model (evaluation and validation modes)
14. Visualize ROC Curves: saves to graphical files the ROC charts (makes possible defining the winner model). **ACTION:** adjust the file paths to fit your environment.
15. Calculate Confusion Matrix: calculates a confusion matrix for the winner model
16. Form Iteration Array: forms an array to iterate over while doing the audience/gain sensitivity analysis (next step)
17. Audience/Gain Scenarios: iterates over a set of experimental scenarios to discover the range of possible optimal audience and maximum gain
18. Harvest Writeback: transfers the results of one of the experimental scenarios from Python to IRIS
19. Capture Context: saves all the objects currently existing in Python context to globals in IRIS
20. Harvest Context: extracts from the context 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.

8.9. 006 Rail Time Series Discovery

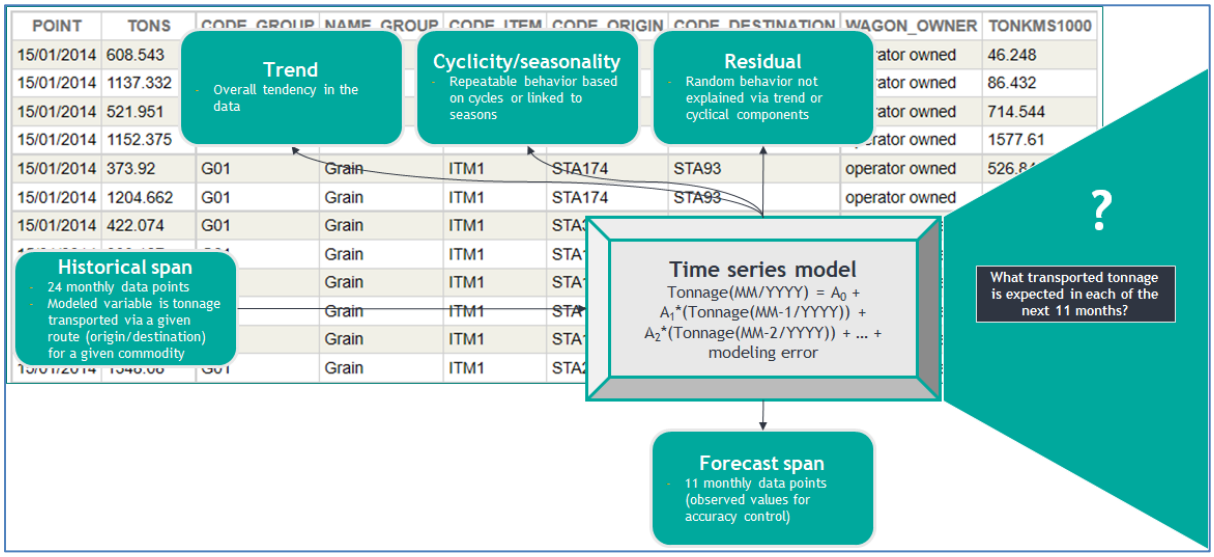
Toolsets: Python Gateway

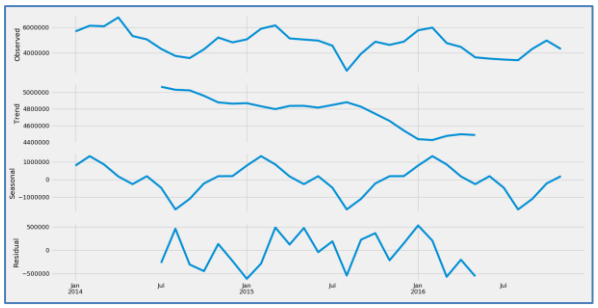
Algorithms: SARIMAX

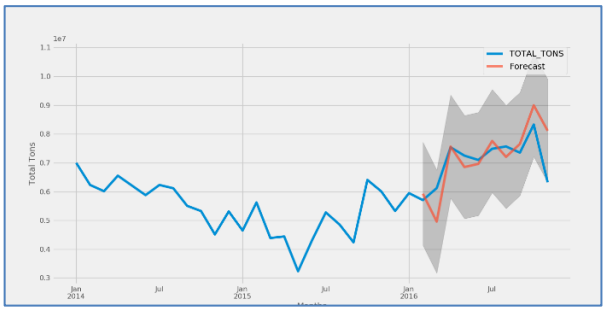
Download: read this section

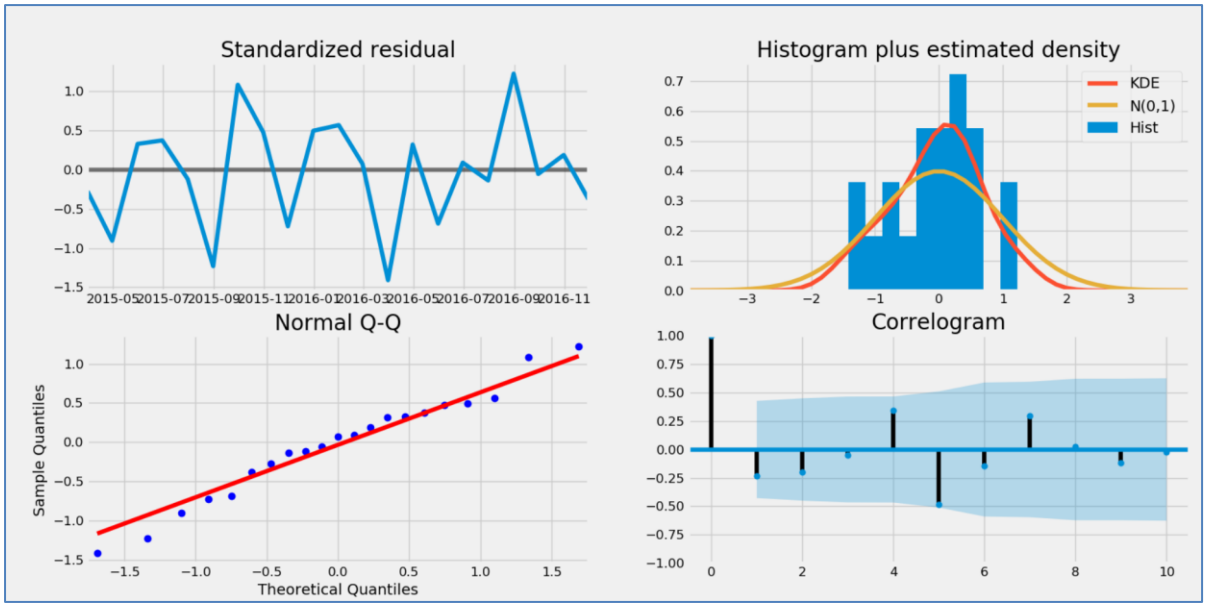
8.9.1. Background

Time series analysis: regression-based methods of time series analysis allow a decomposition of the "signal" (historical values available at the moment of modeling) into several components: trend, cyclical and "noise" parts. Each component is modeled individually, modeling results added up form a forecast.

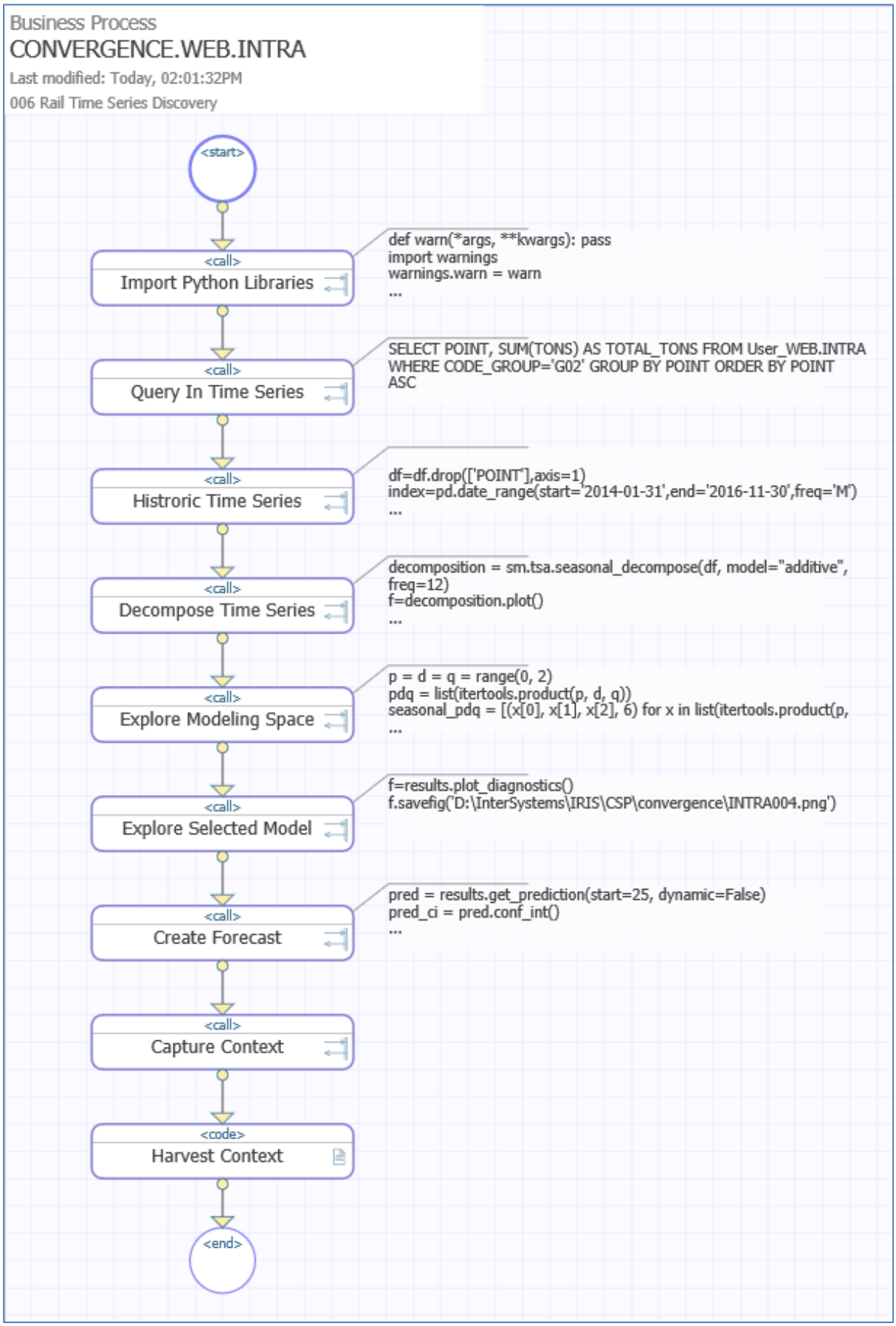








8.9.2. Implementation



8.9.3. Walkthrough

1. Import Python Libraries: loads the required libraries to Python context.
2. Query In Time Series: loads time series data. **ACTION:** implement the data tables in the namespace(s) that fit your environment.
3. Historic Time Series: transform for the analysis and visualize historic time series data. **ACTION:** adjust the file paths to fit your environment.
4. Decompose Time Series: “split” the signal represented by time series data into trend, cyclical and noise components. **ACTION:** adjust the file paths to fit your environment.
5. Explore Modeling Space: iterate through various combinations of time series model parameters to study their impact on modeling accuracy. **ACTION:** adjust the file paths to fit your environment.
6. Explore Selected Model: calculate and visualize the various model metrics helping to interpret modeling results. **ACTION:** adjust the file paths to fit your environment.
7. Create Forecast: calculate and visualize a forecast using the trained model. **ACTION:** adjust the file paths to fit your environment.
8. Capture Context: saves all the objects currently existing in Python context to globals in IRIS
9. Harvest Context: extracts from the context saved in the previous step the time series dataframe and saves it as a separate global in IRIS.

8.10. 007 Housing Debts Prediction

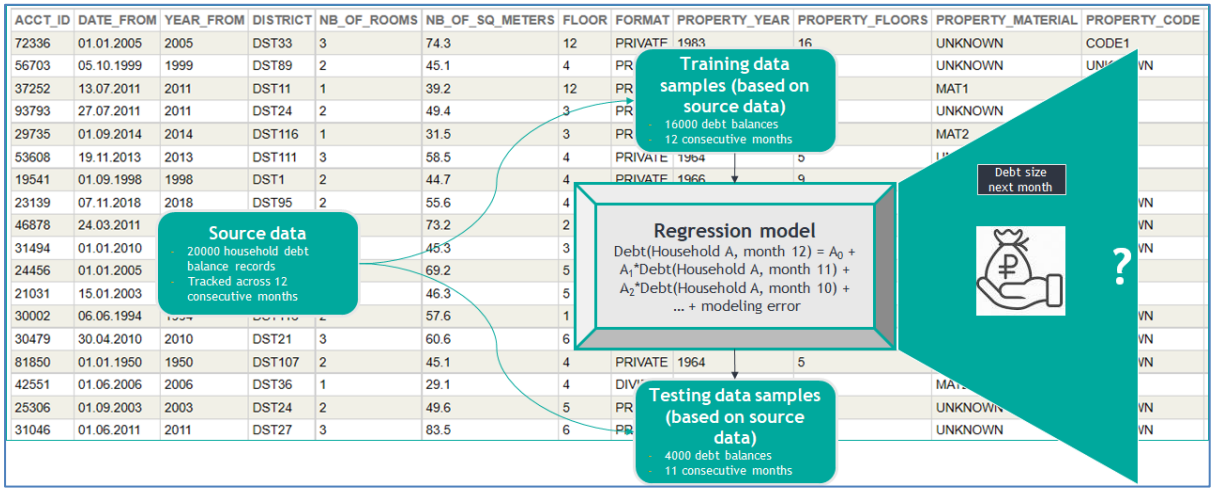
Toolsets: Python Gateway

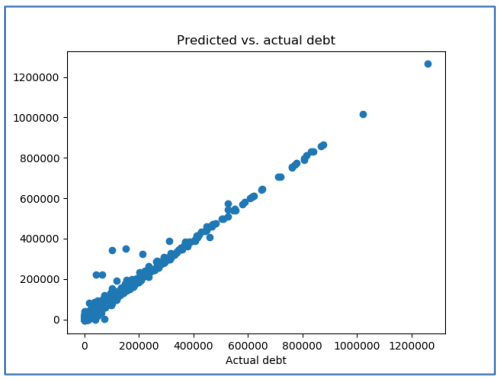
Algorithms: LR

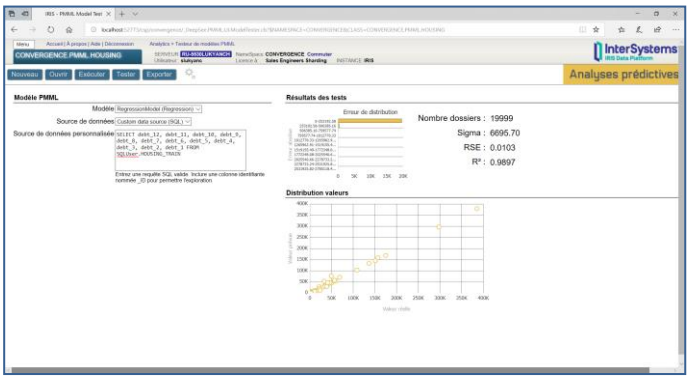
Download: read this section

8.10.1. Background

A regression model trained on a history of debt behaviors is applied to a new dataset to obtain a prediction consisted with the observed debt numbers.







8.10.2. Implementation

8.10.3. Walkthrough

1. Import Python Libraries: loads the required libraries to Python context.
2. Query In Training Set: loads training data. **ACTION:** implement the data tables in the namespace(s) that fit your environment.
3. Fit LR Model: fits a linear regression model.
4. Apply LR Model: applies a linear regression model.
5. Visualize Prediction Quality: saves to a graphical file the actual/predicted chart. **ACTION:** adjust the file paths to fit your environment.
6. Harvest PMML: extracts clustering rules from the trained regression model and converts them to PMML format.

8.11. 008 Diseases Network Analysis

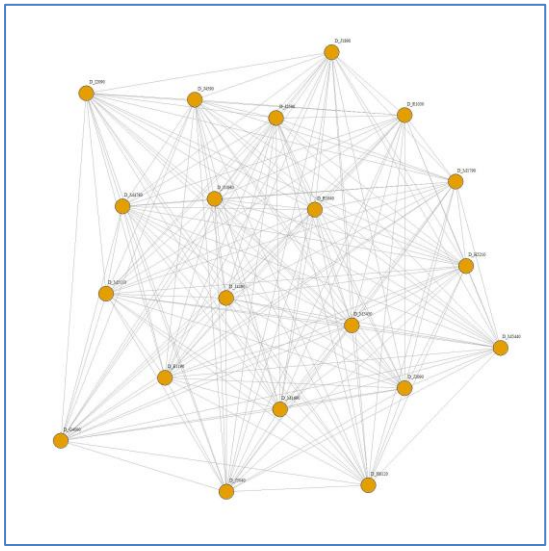
Toolsets: R Gateway

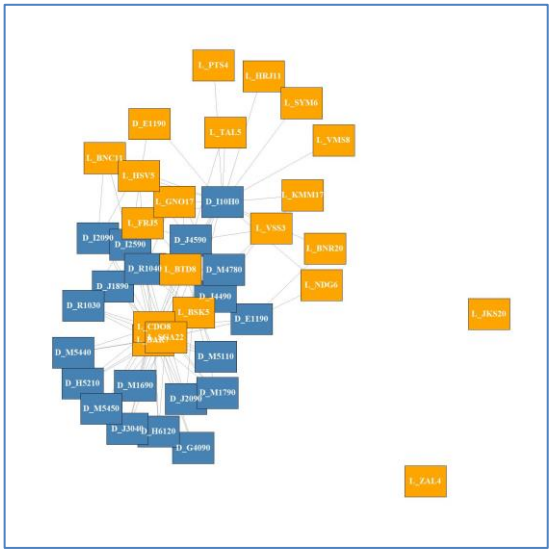
Algorithms: IGraph

Download: read this section

8.11.1. Background

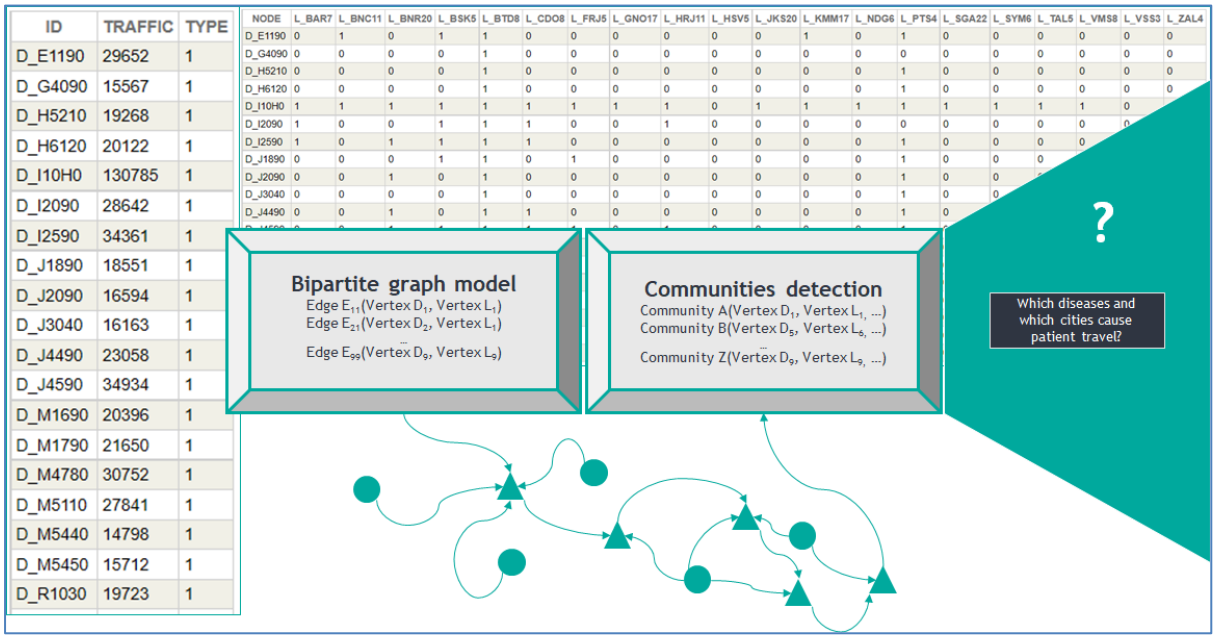
A graph model is instrumental to study data dependencies that are easier to detect via a network rather than a table representation.

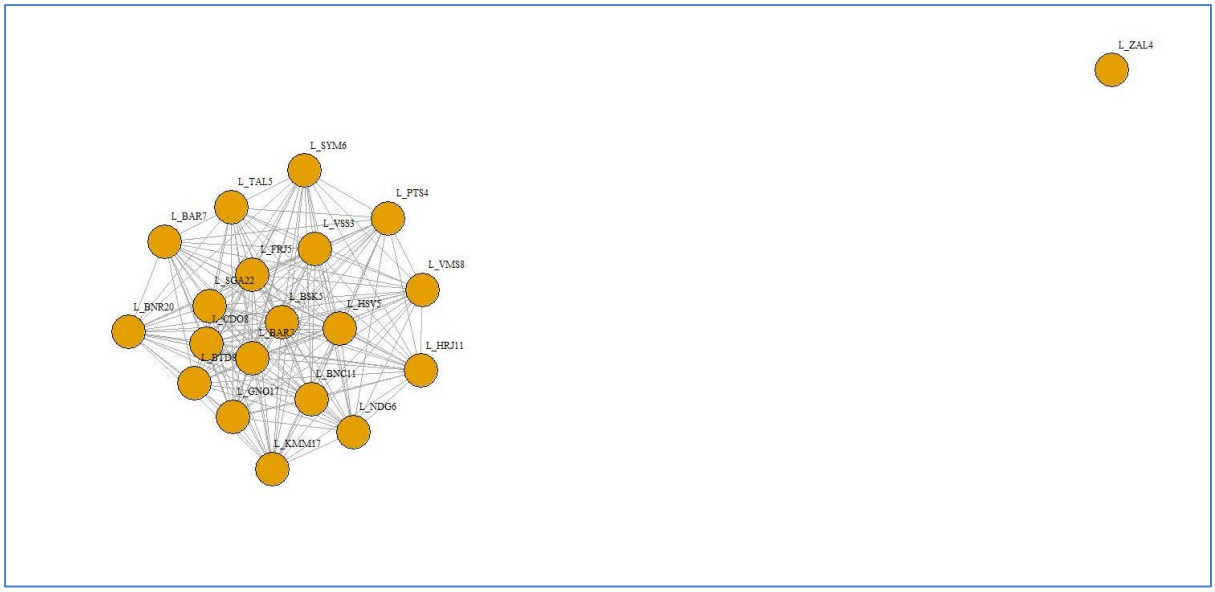


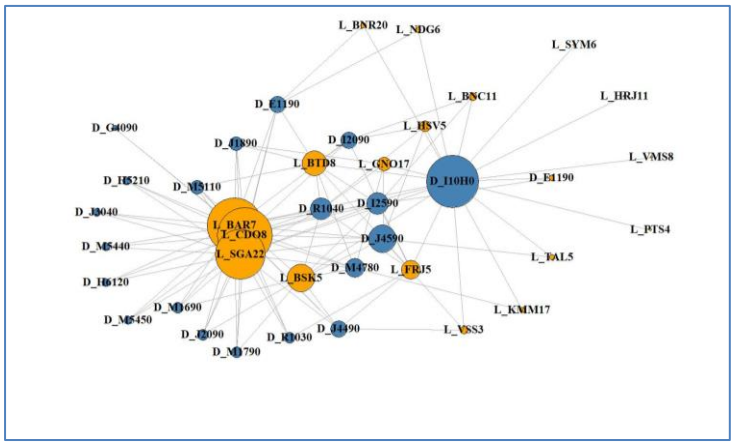


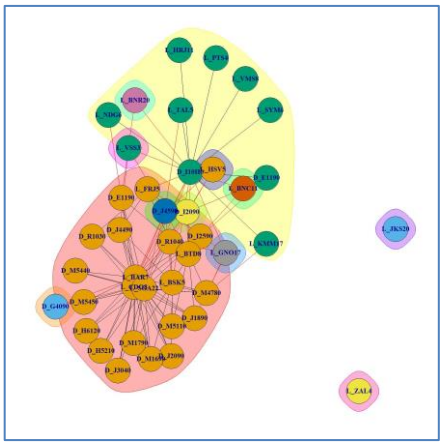
ID	TRAFFIC	TYPE
D_E1190	29652	1
D_G4090	15567	1
D_H5210	19268	1
D_H6120	20122	1
D_I10H0	130785	1
D_I2090	28642	1
D_I2590	34361	1
D_J1890	18551	1
D_J2090	16594	1
D_J3040	16163	1
D_J4490	23058	1
D_J4590	34934	1
D_M1690	20396	1
D_M1790	21650	1
D_M4780	30752	1
D_M5110	27841	1
D_M5440	14798	1
D_M5450	15712	1
D_R1030	19723	1

NODE	L_BAR7	L_BNC11	L_BNR20	L_BSK5	L_BTD8	L_CDO8	L_FRJ5	L_GNO17	L_HRJ11	L_HSV5	L_JKS20	L_KMM17	L_NDG6	L_PTS4	L_SGA22	L_SYM6	L_TAL5	L_VMS8	L_VSS3	L_ZAL4
D_E1190	0	1	0	1	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0
D_G4090	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
D_H5210	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
D_H6120	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
D_I10H0	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	0	
D_I2090	1	0	0	1	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	
D_I2590	1	0	1	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0		
D_J1890	0	0	0	1	1	0	1	0	0	0	0	0	0	1	0	0	0			
D_J2090	0	0	1	0	1	0	0	0	0	0	0	0	0	1	0	0				
D_J3040	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0				
D_J4490	0	0	1	0	1	1	0	0	0	0	0	0	0	1	0					

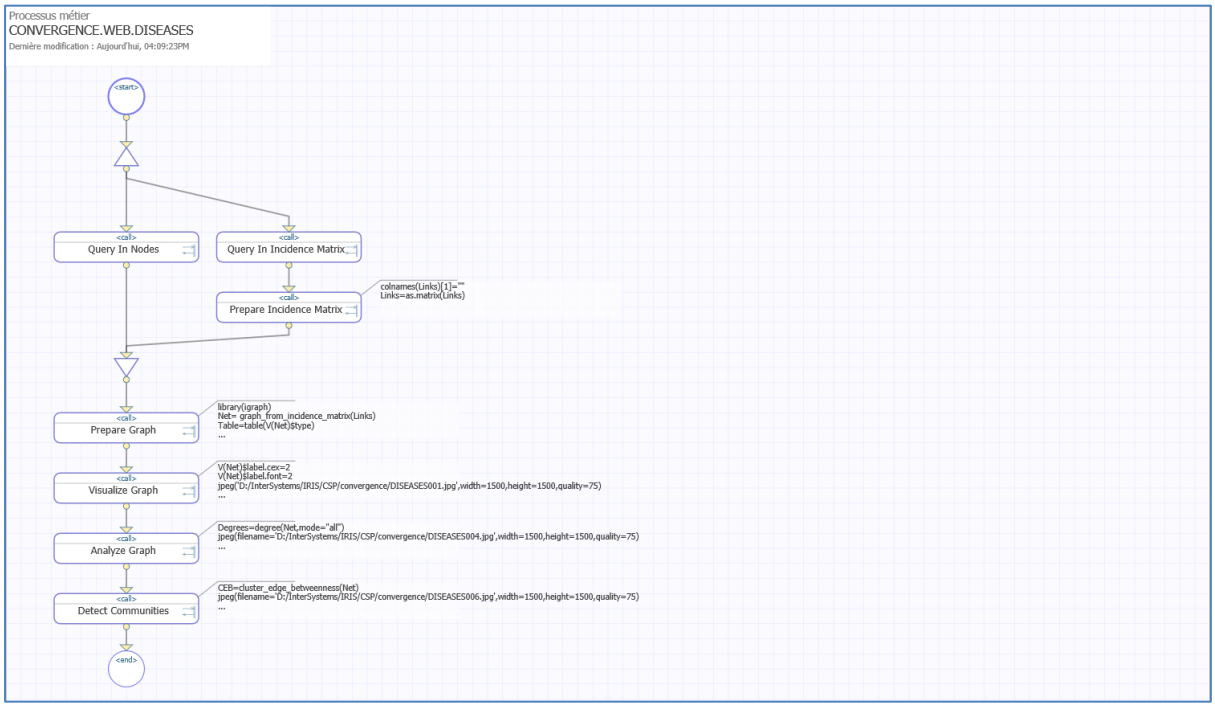








8.11.2. Implementation



8.11.3. Walkthrough

1. Query In Nodes: loads the graph nodes. **ACTION:** implement the data tables in the namespace(s) that fit your environment.
2. Query In Incidence Matrix: loads the graph incidence matrix.
3. Prepare Incidence Matrix: adjusts the incidence matrix dataframe for use in further graph analysis components.
4. Prepare Graph: defines and builds the graph.
5. Visualize Graph: saves to graphical files the graph projections. **ACTION:** adjust the file paths to fit your environment.
6. Analyze Graph: saves to graphical files the results of various graph analyses (bipartite visualization, links intensity, etc.).
7. Detect Communities: determines in the graph subsets of highly linked components, saves to a graphical file the communities visualization.

8.12. 009 Bell State Automation

Toolsets: Python Gateway

Algorithms: pyQuil

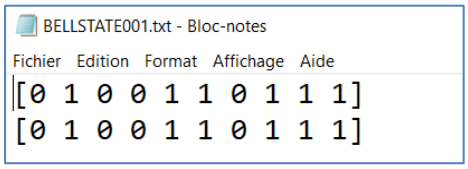
Download: read this section

Idea: http://docs.rigetti.com/en/latest/start.html#installation-and-getting-started

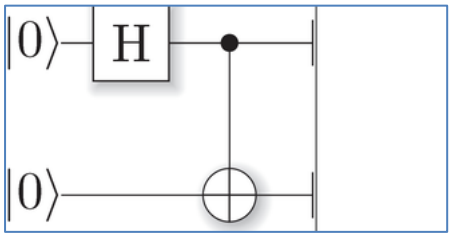
ACTION: install and start `qvm` and `quilc` servers as described under the link above

8.12.1. Background

A Bell state in quantum computations is a form of quantum entanglement under which the two entangled qubits show perfectly correlated measurements:

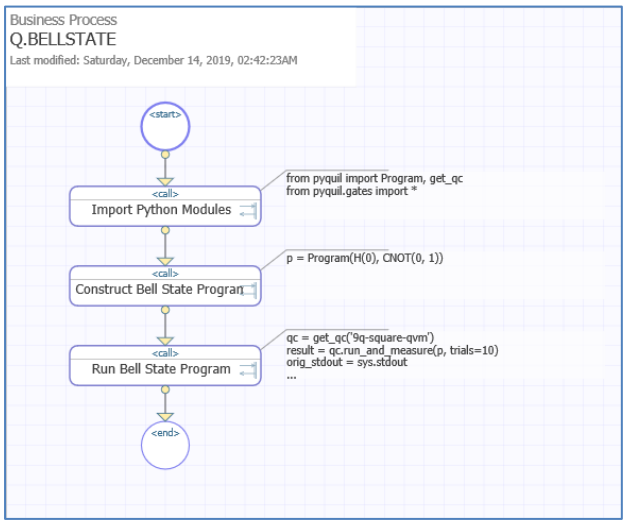


The most straightforward quantum circuit that implements a Bell state is like that:



In our showcase, we implement the above quantum circuit and run it 10 times.

8.12.2. Implementation



8.12.3. Walkthrough

1. Import Python Modules: loads the required modules to Python context.
2. Construct Bell State Program: constructs the quantum circuit object.
3. Run Bell State Program: runs calculations using the quantum circuit.

InterSystems Corporation
World Headquarters

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

InterSystems.com