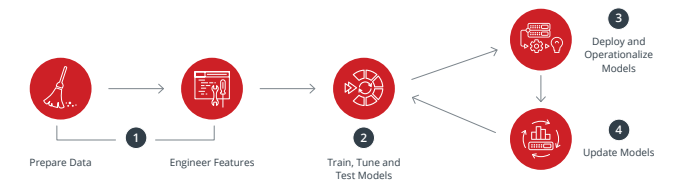
Pentaho and Machine Learning Orchestration

With Pentaho’s machine learning orchestration, the process of building and deploying advanced analytics models maximizes efficiency. Most enterprises struggle to put models to work because data professionals often operate in silos and the workflows - from data preparation to updating models - create bottlenecks.

Pentaho’s platform enables collaboration and removes bottlenecks in four key areas:



1. Prepare Data and Engineer New Features

Pentaho helps data scientists and engineers easily prepare and blend traditional sources like ERP, EAM and big data sources like sensors and social media. Pentaho also accelerates the notoriously difficult and costly task of feature engineering by automating data onboarding, data transformation and data validation in an easy-to-use drag and drop environment.

2. Train, Tune, and Test Models

Data scientists often apply trial and error to strike the right balance of complexity, performance and accuracy in their models. With integrations for languages like R and Python, and for machine learning packages like Spark MLlib and Weka, Pentaho allows data scientists to seamlessly train, tune, build and test models faster.

3. Deploy and Operationalize Models

A completely trained, tuned and tested machine learning model still needs to be deployed. Pentaho allows data professionals to easily embed models developed by the data scientist directly in a data workflow. They can leverage existing data and feature engineering efforts, significantly reducing time-to-deployment. With embeddable APIs, organizations can also include the full power of Pentaho within existing applications.

4. Update Models Regularly

With Pentaho, data engineers and scientists can re-train existing models with new data sets or make feature updates using custom execution steps for R, Python, Spark MLlib and Weka. Pre-built workflows can automatically update models and archive existing ones.

Demonstration: Hello World – Python Script using Spyder

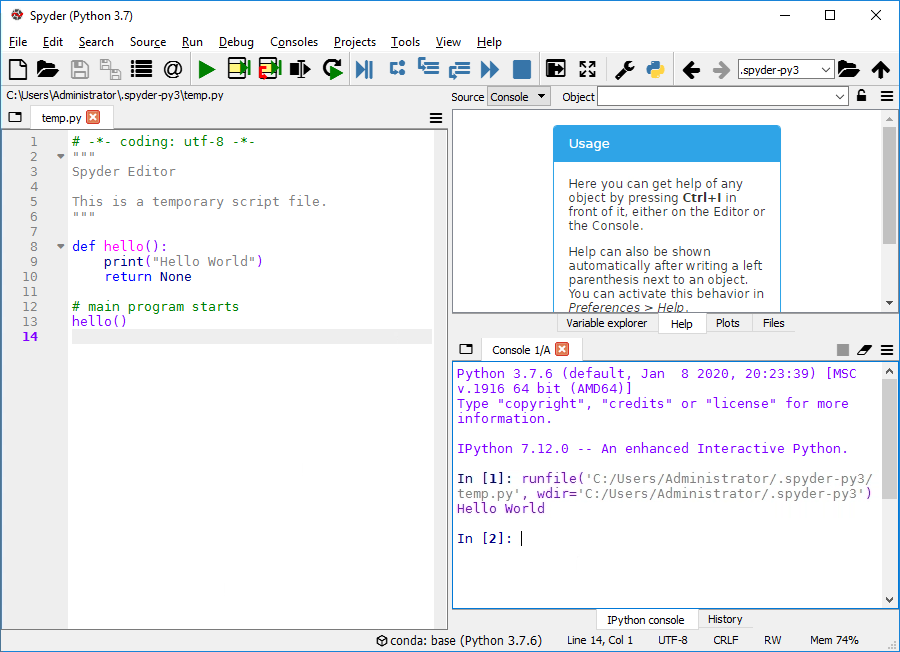
|  |  |  |  |
| --- | --- | --- | --- |
| Introduction | Spyder is an open source cross-platform integrated development environment (IDE) for scientific programming in the Python language. Spyder integrates NumPy, SciPy, Matplotlib and IPython, as well as other open source software packages. |  |  |

|  |  |
| --- | --- |
| Objectives | In this Lab, you will gain an:   * Overview of Spyder. * Execute a simple Python Script. |

Spyder

Spyder is the IDE for Python language. To access the environment:

1. On your Desktop, double-click on the Spyder icon. The following user interface is loaded:



1. In the Editor pane, enter the following script:

def hello():

print("Hello World")

return None

# main program starts

hello()

1. To execute the program, select Run -> Run (or press F5), and confirm the Run settings if required (if pop-up window appeared).
2. You should see output like (the paths may differ):

In [1]: runfile('C:/Users/Administrator/.spyder-py3/temp.py', wdir='C:/Users/ Administrator/.spyder-py3')

Hello World

In [2]:

If so, then you have just run your first Python program -- well done.

So what happens when you Execute the script?

* Python reads the file line by line, ignoring comments
* when it comes across the def keyword, it knows that a function is DEFined in this and the next (one or more) lines. In the hello\_world.py file, Python thus creates a function object with name hello. All indented lines following def hello(): belong to the function body.

Note that the function object is just created at this point in the file, but the function is not yet called (i.e. not executed).

* when Python comes across commands (other than def ... and a few other keywords) that are written in the left-most column, it will execute these immediately. In the hello\_world.py file this is only the line reading hello() which will actually call (i.e. execute) the function with name hello.

If you remove the line hello() from the program and run the whole file again (by pressing F5, or selecting run -> run), nothing will be printed. Why?

Demonstration: Hello World – R Script using R Studio

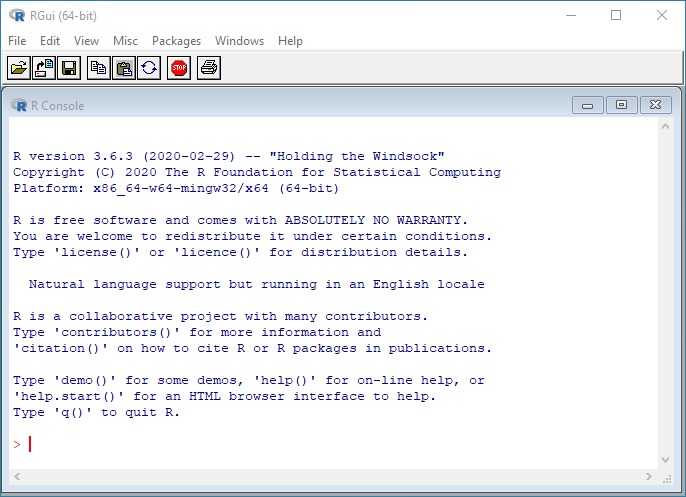
|  |  |  |  |
| --- | --- | --- | --- |
| Introduction | RStudio is an integrated development environment (IDE) for R. It includes a console, syntax-highlighting editor that supports direct code execution, as well as tools for plotting, history, debugging and workspace management. |  |  |

|  |  |
| --- | --- |
| Objectives | In this workshop, you will:   * Overview of R Studio. * Execute a simple R Script. |

RStudio

RStudio makes R easier to use. It includes a code editor, debugging & visualization tools. To access the environment:

1. On your Desktop, double-click on the RStudio icon. The following user interface is loaded:



1. In the R Console, at the prompt enter the following:

# We can use the print() function

> print("Hello World!")

[1] "Hello World!"

Demonstration: Hello World – Python & R Scripts

|  |  |  |  |
| --- | --- | --- | --- |
| Introduction | The workshop illustrates how to execute Python and R scripts with a Pentaho Job. |  |  |

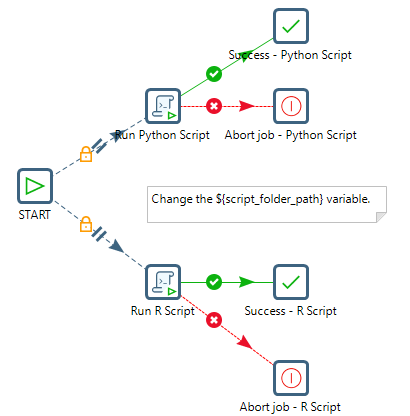
|  |  |
| --- | --- |
| Objectives | In this workshop, you will:   * Configure R Executor step. * Build and Train a Forest Tree Model. * Deploy and Test the model. |

Job – Shell Scripts

The Job illustrates the required configuration settings for the ‘Execute Shell Script’ step.

1. In Spoon open the following transformation

C:\DataOps\MLO\demonstration\kjb\_shell\jb\_run\_python\_R\_scripts.kjb



1. Double-click on the Run Python Script step and note the configuration settings.
2. Repeat with the Run R Script step.

Lab: Detect Credit Card Fraud

|  |  |  |
| --- | --- | --- |
| Introduction | Imagine that a direct retailer wants to reduce losses due to orders involving fraudulent use of credit cards. They accept orders via phone and their web site, and ship goods directly to the customer. Basic customer details, such as customer name, date of birth, billing address and preferred shipping address, are stored in a relational database.  Orders, as they come in, are stored in a database. There is also a report of historical instances of fraud contained in a CSV spreadsheet. |  |

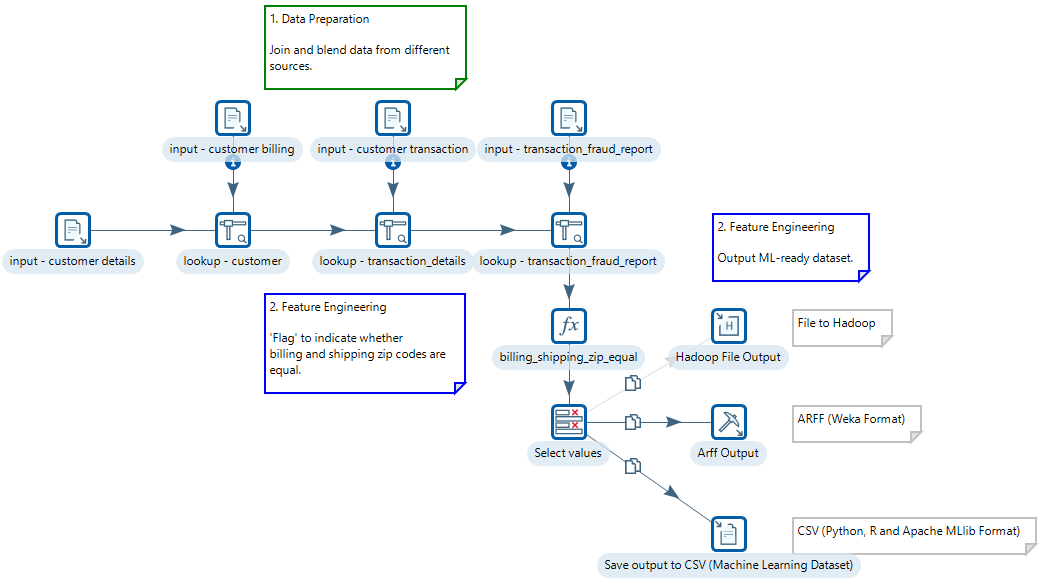
|  |  |
| --- | --- |
| Objectives | In this workshop, you will:   * Configure R Executor step. * Build and Train a Forest Tree Model. * Deploy and Test the model. |

1. Data Preparation

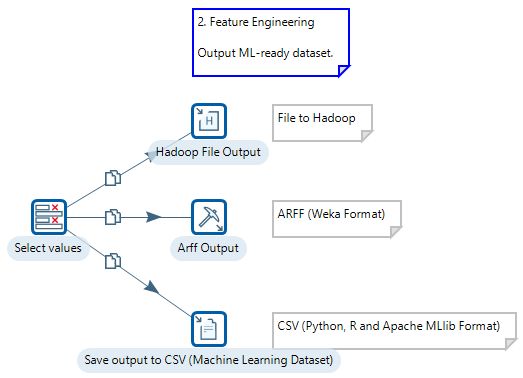
With the goal of preparing a dataset for ML, we can use PDI to combine these disparate data sources and engineer some features for learning from it. The following figure shows a transformation demonstrating an example of just that and includes some steps for deriving new fields. To begin with customer data is joined from several data sources, and then blended with transactional data and historical fraud occurrences contained in a CSV file.

2. Feature Engineering

The Feature Engineering is set to: billing zip code = shipping zip code



Following this, there are steps for deriving additional fields that might be useful for predictive modeling. These include computing the customer's age, extracting the hour of the day the order was placed, and setting a flag to indicate whether the shipping and billing addresses have the same zip code.



This process culminates with output of flattened (a Data Scientist’s preferred data shape) data in both CSV and ARFF (Attribute Relational File Format) data, the latter being the native file format used by PDM (Pentaho Data Mining, AKA WEKA). We end up with 100,000 examples (rows).

From this list, for the purposes of predictive modelling, we can drop the customer name, ID fields, email addresses, phone numbers and physical addresses. These fields are unlikely to be useful for learning purposes and, in fact, can be detrimental due to the large number of distinct values they contain.

3. Train, Tune, Test Machine Learning Models to Identify the Most Accurate Model

So, what does the data scientist do at this point?

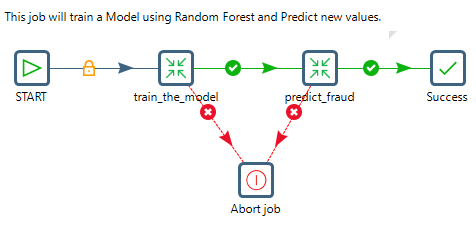
Typically, they will want to get a feel for the data by examining simple summary statistics and visualizations, followed by applying quick techniques for assessing the relationship between individual attributes (fields) and the target of interest which, in this example, is the “reported\_as\_fraud\_historic" field.

Following that, if there are attributes that look promising, quick tests with common supervised classification algorithms will be next on the list. This comprises the initial stages of experimental data mining - i.e. the process of determining which predictive techniques are going to give the best result for a given problem.

The model that will be used: **Random Forest**

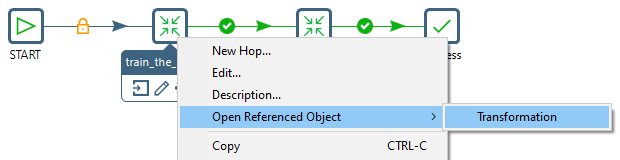
1. In Spoon open the following main Job:

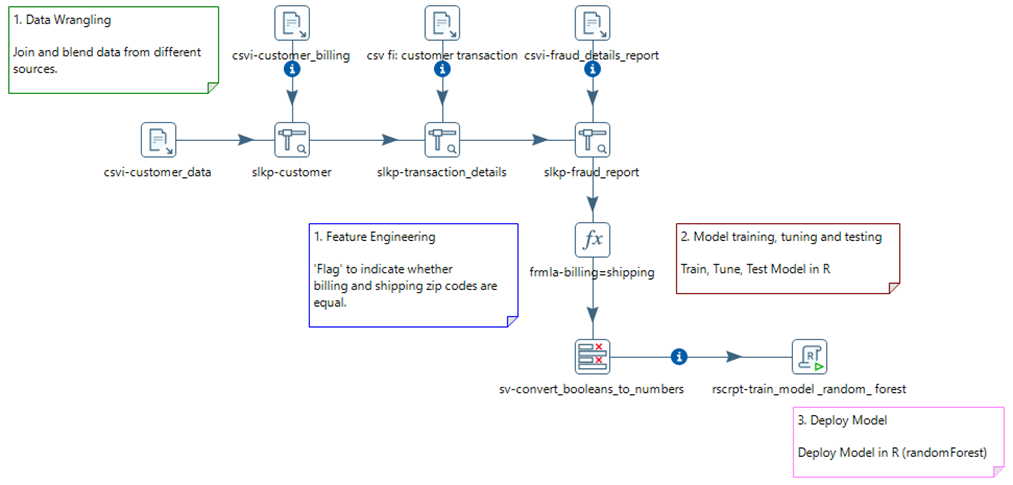
C:\DataOps\MLO\labs\fraud\jb\_fraud\_main\_job.kjb



Let’s look at the transformation that Trains for the model.

1. Right mouse click on the train\_the\_model Transformation and select: Open Referenced Object > Transformation



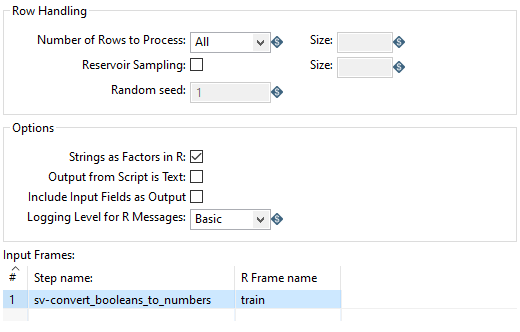


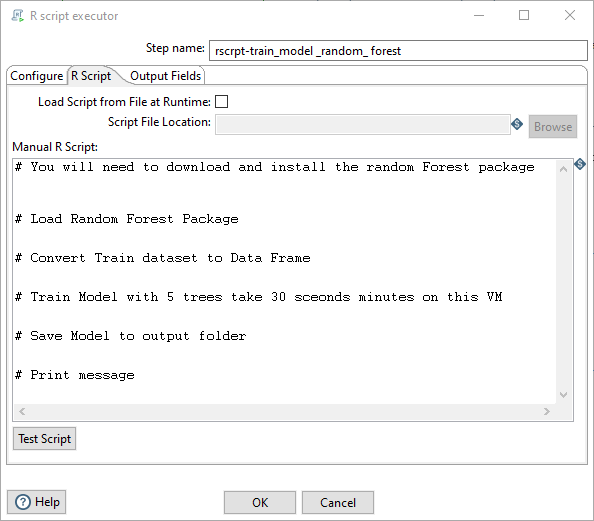
To configure the model for ‘training’:

1. Double-click on the ‘Train Model using Random Forest’ step to bring up the configuration settings.
2. Under the Configure tab, ensure the Input Frames points to the Step name:

sv-convert\_booleans\_to\_numbers

And that the R Frame name is: train



1. Select the R script tab. Copy and Paste the code snippets based on the Comments. 
2. The required script is located:

C:\DataOps\MLO\labs\fraud\scripts\train\_model.txt

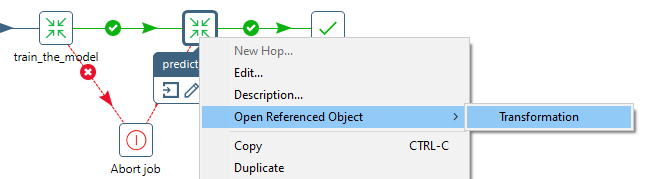
4. Deploy Predictive Models in Pentaho

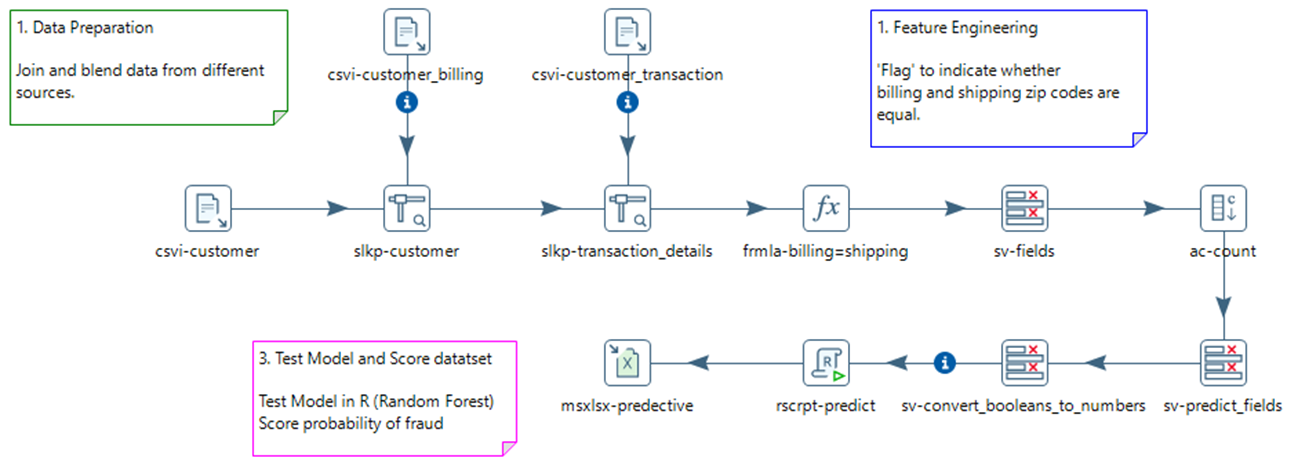
PDI also supports the data scientist who prefers to work directly in R or Python when developing predictive models and engineering features. Scripting steps for R and Python allow existing code to be executed on PDI data that has been converted into data frames. With respect to machine learning, care needs be taken when dealing with separate training and test sets in R and Python, especially with respect to categorical variables. Factor levels in R need to be consistent between datasets (same values and order); the same is true for Scikit-learn and, furthermore, because only numeric inputs are allowed, all categorical variables need to be converted to binary indicators via the one-hot-encoding (or similar). WEKA's wrappers around MLR and Scikit-learn take care of these details automatically, and ensure consistency between training and test sets.

Let’s look at the transformation that Deploys / Predicts for the model.

1. Right mouse click on the tr\_predict\_fraud Transformation and select:

Open Referenced Object > Transformation

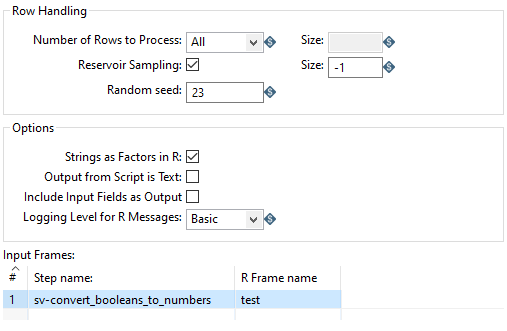


****

The csvi-fraud\_deatils\_report step is no longer required as the raw data is now used to predict fraud occurrence.

To configure the model for ‘predicting’:

1. Double-click on the ‘Predict’ step to bring up the configuration settings.
2. Under the Configure tab, ensure the Input Frames points to the Step name:





Reservoir Sampling

If selected, randomly samples rows from an incoming data stream. Use this option to sample a fixed number of rows from an incoming data stream when the total number of incoming rows is not known in advance. Using this option limits the number of processed rows by the R script to the number you enter in the Size field. Note that if Reservoir Sampling is active and the Number of Rows to Process is set to Row By Row, only the first data frame is used.

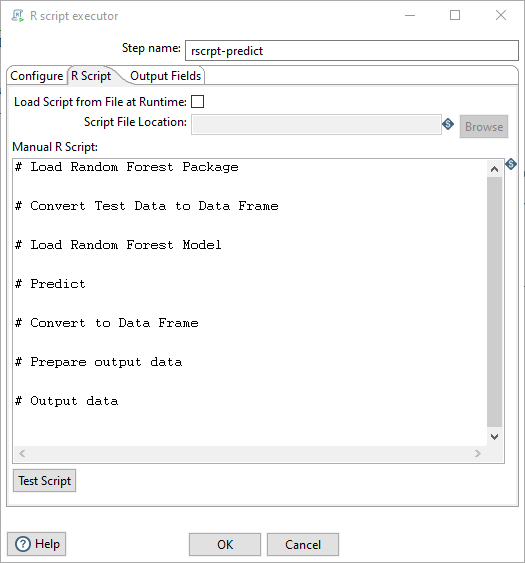
Size

The number of rows to sample from an incoming stream. Setting a value of -1 will sample 100,000 rows. This field becomes active if Reservoir Sampling is selected.

Random Seed

The value to use for seeding the random number generator. Repeating a transformation with a different value for the seed will result in a different random sample being chosen.

1. Select the R script tab. Copy and Paste the code snippets based on the Comments.



1. The required script is located:

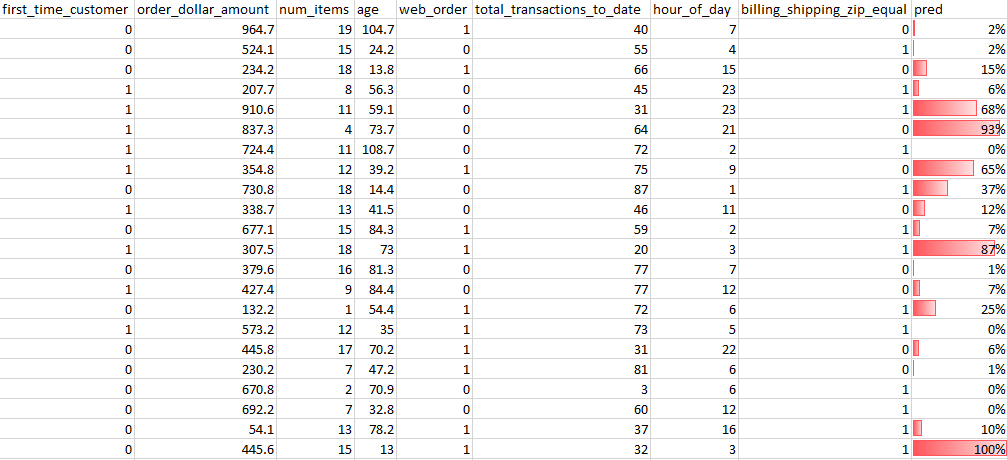
C:\DataOps\MLO\labs\fraud\scripts\predict\_model.txt

1. RUN the jb\_fraud\_main\_job.kjb

Results

The predictive results from the model are output to:

C:\DataOps\MLO\labs\fraud\output\credit\_card\_predict.xlsx



This predictive use-case walkthrough demonstrates the power and flexibility of Pentaho afforded to the data engineer and data scientist. From data preparation through to model deployment, Pentaho provides machine learning orchestration capabilities that streamline the entire workflow.

Appendix A – Train Model

# You will need to download and install the random Forest package

# install.packages("randomForest")

# Load Random Forest Package

library(randomForest)

# Check dataset is a Data Frame

train.df <- as.data.frame(train)

# Train Model with 5 trees take 30 sceonds minutes on this VM

rf <- randomForest(train.df$reported\_as\_fraud\_historic ~ ., train.df, ntree=5, importance=TRUE)

# Save Model to output folder: Note no spaces and double backslashes are required.

save(rf, file="C:\\DataOps\\MLO\\labs\\fraud\\solution\\train\_model\_output\\rf.rdata")

# Print message ok to indicate no probs..

ok <- "Finished"

ok.df <- as.data.frame(ok)

ok.df

Appendix B – Predict Model

# Load Random Forest Package

library(randomForest)

# Convert Test Data to Data Frame

test.df <- as.data.frame(test)

# Load Random Forest Model

load("C:\\DataOps\\MLO\\labs\\fraud\\solution\\train\_model\_output\\rf.rdata")

# Predict

pred <- predict(rf, newdata = test.df)

# Convert to Data Frame

pred.df <- as.data.frame(pred)

# Prepare output data

submission <- data.frame(cbind(test.df,pred.df))

# Output data

submission