

# Incorta Machine Learning – Getting Started Guide

June 2020

## Contents

[Why ML in an Analytics Platform?](#)

[How Incorta Supports Data Science](#)

[Data Enrichment in Incorta](#)

[About the Dataset](#)

[Enriching Data with Spark and Notebooks](#)

[A Simple Machine Learning Example in Incorta](#)

[Machine Learning with Incorta-ML](#)

[Reference](#)

[Comments? Questions?](#)

---

## Why ML in an Analytics Platform?

Before diving into Python code and notebooks, let's discuss why Incorta included machine learning capabilities inside of our analytics platform.

As an industry, we continue to see survey after survey revealing high expectations for ML:

**“96% of companies expect to see an explosion of machine learning projects in production by 2020”<sup>1</sup>**

But there are so many obstacles to deploying a project, once an initial model has been developed.

**“87% of data science projects never make it into production”<sup>2</sup>**

What are some of these obstacles?

- Data Acquisition -- When developing a model, being able to access production data, to ensure that the models are valid and will still work when deployed.
- Data Fidelity -- Having access to detailed data that hasn't been tampered with.
- Speed -- Being able to explore, investigate and extract any needed training or testing datasets without having to wait.
- Security -- Granular controls on sensitive data.
- Infrastructure -- The complexity of provisioning and maintaining servers and software.
- Scheduling -- The ability to train, re-train and predict based on continuously updated data from enterprise application systems.
- Presentation -- Having an easy way to host and present the results to data consumers.

Thankfully, all of these blockers to model deployment are things that are easily handled by the Incorta platform.

---

<sup>1</sup> [Univa Machine Learning Survey](http://www.univa.com/resources/univa-machine-learning-survey.php), <http://www.univa.com/resources/univa-machine-learning-survey.php>

<sup>2</sup> [Venture Beat](https://venturebeat.com/2019/07/19/why-do-87-of-data-science-projects-never-make-it-into-production/),

<https://venturebeat.com/2019/07/19/why-do-87-of-data-science-projects-never-make-it-into-production/>

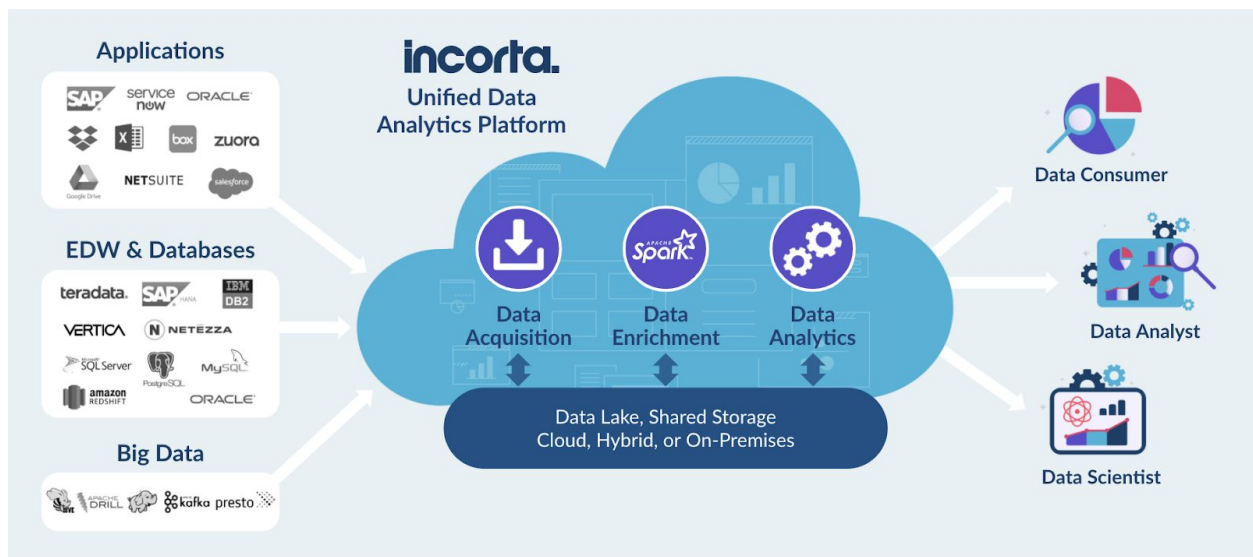
# How Incorta Supports Data Science

Incorta is a unified analytics platform, meaning that it incorporates and integrates several different functions together: from easily connecting to and blending data from a variety of sources including applications, databases, streams, and files -- to delivering that data to any kind of user, whether a business level data consumer, a data analyst or even a data scientist.

Incorta is used by some of the largest companies in the Fortune 500, and can be hosted in the cloud, or run on-premises. It can work against shared storage, networked storage, cloud storage, or a hybrid storage model.

However, deploying machine learning models is complex because the model is only one component of a predictive system. In addition to data management, there is process management, monitoring, resource management and analysis of the output.

Incorta incorporates the most essential parts of data pipeline tools, data science and data enrichment tools, and data analytics tools into a single platform.



The main difference between Incorta and other other analytics platforms is that it doesn't require that data be reshaped and aggregated to fit an analytical, or dimensional model. Incorta is able to analyze full-fidelity, raw transactional data. And this means that we not only serve the needs of data consumers and analysts but also data scientists because we maintain all the raw, transactional data available for machine learning models.

By merging data management, analytics and data science into a single platform, Incorta means more data science with less data wrangling.

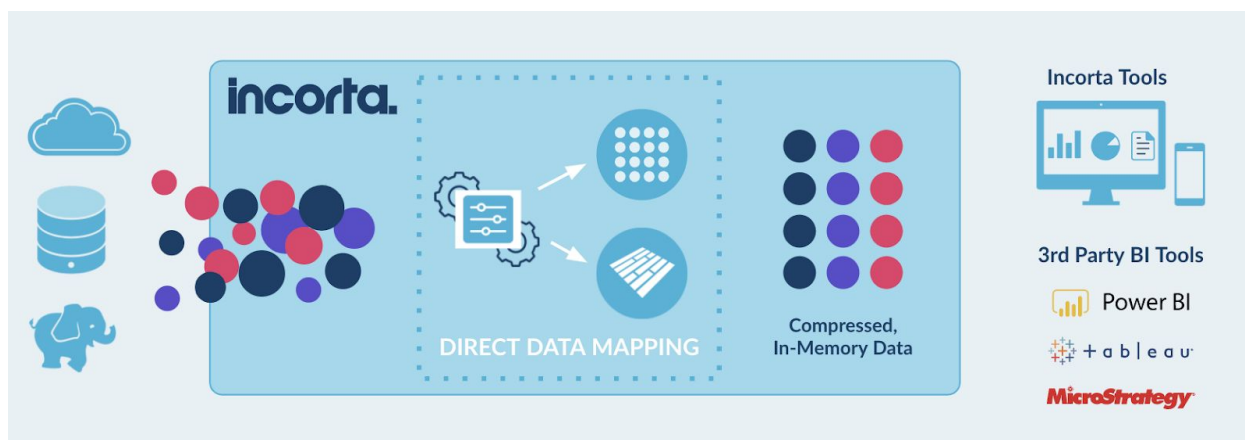
## Direct Data Mapping

The secret to Incorta's ability to maintain full-fidelity data for machine learning while delivering incredible query speed is **Direct Data Mapping (DDM)**.

DDM is Incorta's proprietary technology that pre-processes or "maps" incoming data such that it does not have to be transformed, reshaped, or aggregated to achieve blazing fast query performance even on data that is complex and requires many joins.

As Incorta's DDM processes data, it results in a data set that is actually smaller than the source data, as opposed to most dimensional modeling schemes which create a data set that is larger than the source data. This means efficient cluster data transfers, and fast data loads into memory.

And, while most other solutions just compress data on-disk and uncompress it as it is read into memory, Incorta maintains a columnar and compressed data organization both on-disk, and in-memory. This makes it possible to analyze 10x to 20x more data than would normally fit into available memory.



However, Incorta is not memory-bound. Data is dynamically loaded from disc as needed, unless a data table has been specifically configured to always remain in memory.

Incorta DDM replaces cost- and rule-based optimizers, and does not require any query tuning or query statistics maintenance for good analytics performance. This means easier operations and more consistent performance.

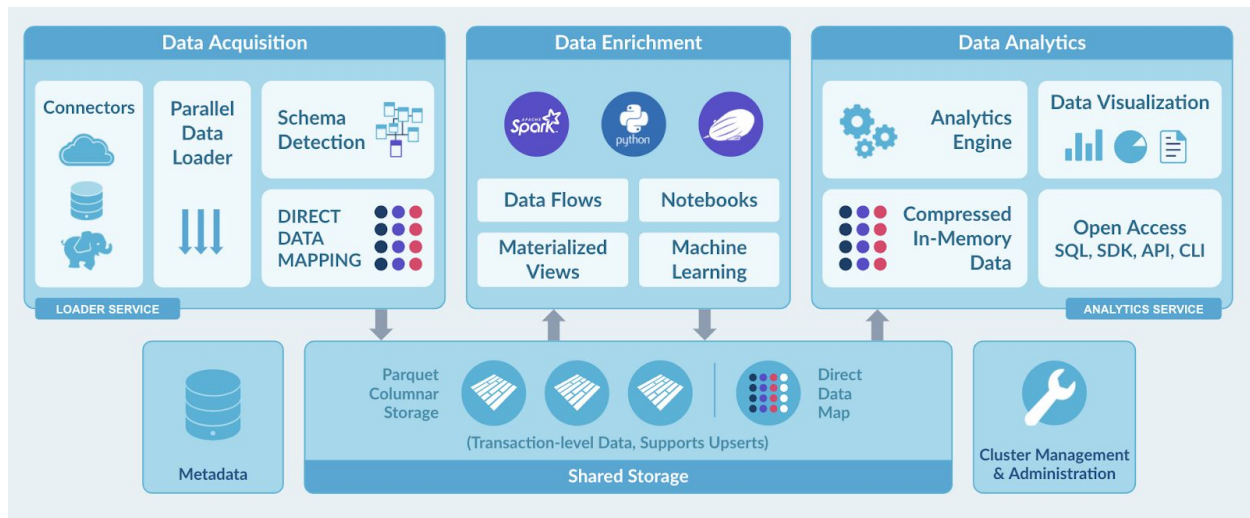
## The Incorta Architecture

The Incorta architecture is a complete end-to-end solution including data acquisition, data management, data enrichment, data science and data analytics.

Data acquisition in Incorta results in a highly optimized, standardized, and compacted data lake alongside a data catalog, and the Direct Data Mapping files.

Incorta also includes native, built-in support for **Spark** which makes it easy to support powerful data enrichments without needing another integration tool

Data is stored in an open-standard file format, **Parquet**, which is durable, open, compressed and reasonably performant for analytics queries. This makes it possible to run different workloads against the same data. So, data enrichment and other data science activities can be running against the data while the data is also loaded into our in-memory analytics engine to feed data exploration and data visualization.



## An Ideal Environment for Data Science

This is great for data scientists, because the platform handles many of the things that keep data scientists from doing more productive work, such as:

- Connecting to every database, cloud application, data stream and file format
- Managing the data and securing access to it
- Providing end-users easy access to raw data and analytical results
- Running the infrastructure

By developing and running machine learning on Incorta organizations get:

- A platform that allows Data scientists to focus on the work, on creating models
- A Single Source of Truth: Data engineers, data analysts, data scientists and business users all access and work against the same data
- Faster experimentation and more iterations.

And this means fast performance whether you are using Incorta's internal tools, or are using various 3rd party BI tools like Power BI or Tableau, or you are doing data science inside of a Spark notebook.

# Data Enrichment in Incorta

The architecture of Incorta’s platform including Direct Data Mapping is well suited to all kinds of data enrichment including machine learning development and deployment: Apache Spark integration, the Incorta ML library, and integrated notebooks.

## Incorta Spark Integration

Apache Spark is bundled with the Incorta platform both for data enrichment, and for data science and machine learning.

Spark comes bundled and preconfigured and optimized -- and works directly against our data store.

We have a job scheduling and orchestration system that makes it easy to put Spark jobs into production.

And if you already have a Spark cluster, that’s no problem, you can continue to use it.



Incorta Spark Integration:

- Shipped and bundled with Incorta
- Predefined configuration optimizations
- No dependency on Hive metadata store, which is very difficult to maintain
- Full access to Incorta metadata (schemas over underlying Parquet files)
- Scheduled jobs
- Can also use existing spark clusters in the customer environment
- Provides nearly unlimited data enrichment capabilities

## Incorta ML

Incorta ML is a Python library that provides helpful utilities and a uniform wrapper over a number of familiar libraries such as Spark ML and others.

With Incorta ML, we have made it easy to create and save DataFrames to Incorta, and allow data scientists to focus on model building instead of writing boilerplate.

Incorta ML provides access to a variety of algorithms, and includes an Auto-ML function.

```
%pyspark
a_model_name = "Iris_Auto"
build_model(training_data, model_name=a_model_name,
            algorithm_name="auto", label_column_name=label_column_name,
            mode="classification", params=None)

Best algorithm name LogisticRegression
```

```
%pyspark
df_eval = evaluate(testing_data, model_name=a_model_name)
z.show(df_eval)
```

metric_name	value
f1	0.9722222222222223
weightedPrecision	0.9743589743589745
weightedRecall	0.9722222222222221
accuracy	0.9722222222222222

Auto-ML will automatically prepare a dataset and then select the best model for prediction based on the data.

It can be used both to streamline work for data scientists and also users who are new to machine learning.

A very useful application for this is when you have to generate many different models to account for different factors like geography or product line or seasonality, and it is impractical to create these models by hand. This is common in a number of industries such as retail and insurance.

The `build_model()` function will save the model to disk so that it can be used in the data pipeline for prediction and scheduled to run on a regular basis.

## Notebooks

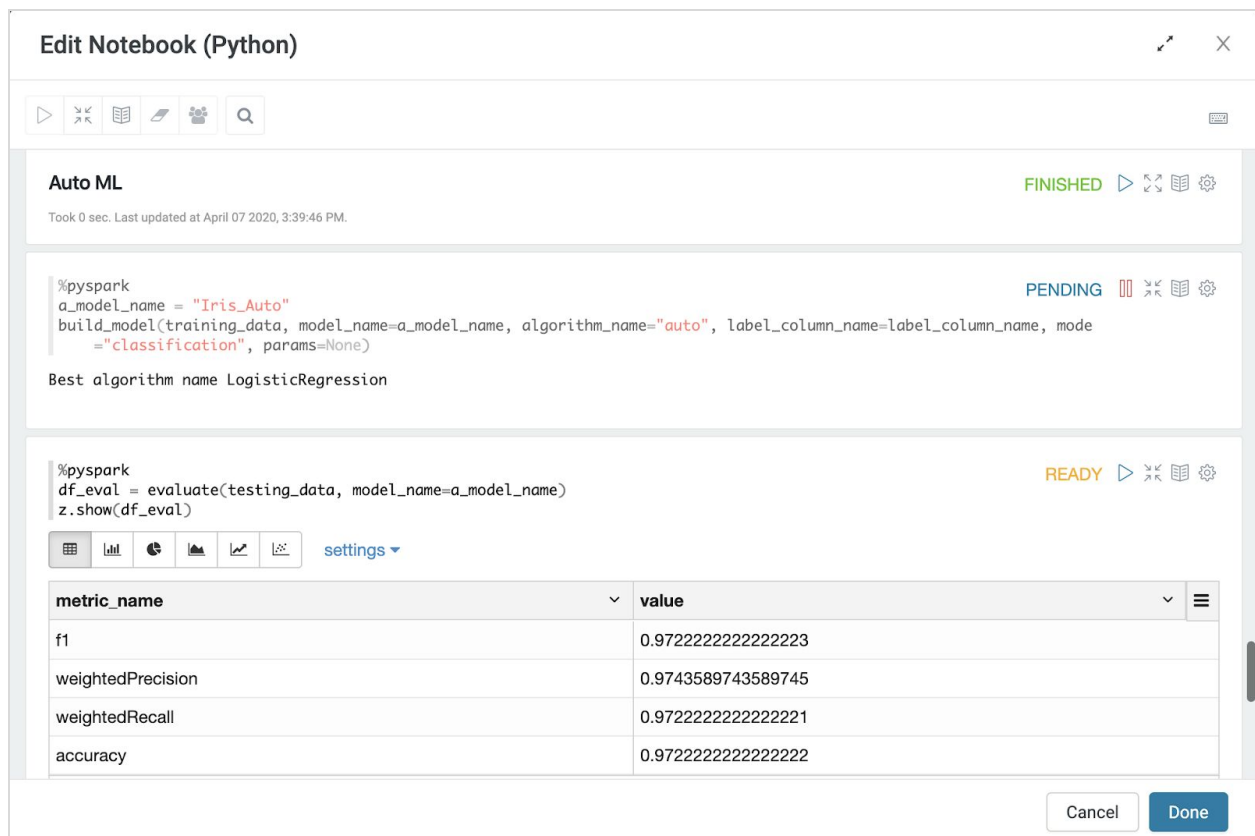
A standard notebook interface is embedded in the Incorta UI. Notebooks can be coded in SQL, Python, Scala, and R. Reading data from Incorta is as simple as:

```
read("Incorta_Schema.incorta_table")
```

And writing data back to Incorta is as simple as:

```
save(dataframe)
```

You can use the Incorta ML library to get started, but you don't have to stop there. Additional Python libraries can be added to both on-premises and cloud Incorta instances,



**Edit Notebook (Python)**

**Auto ML** FINISHED

Took 0 sec. Last updated at April 07 2020, 3:39:46 PM.

```
%pyspark
a_model_name = "Iris_Auto"
build_model(training_data, model_name=a_model_name, algorithm_name="auto", label_column_name=label_column_name, mode
="classification", params=None)
```

Best algorithm name LogisticRegression

```
%pyspark
df_eval = evaluate(testing_data, model_name=a_model_name)
z.show(df_eval)
```

metric_name	value
f1	0.9722222222222223
weightedPrecision	0.9743589743589745
weightedRecall	0.9722222222222221
accuracy	0.9722222222222222

Cancel Done

## About the Dataset

To get you starting using machine learning in Incorta, we will use a well-known [Kaggle](#) dataset and machine learning example so that you can see how easy it is to create and run models, without the complexity of an actual data source.

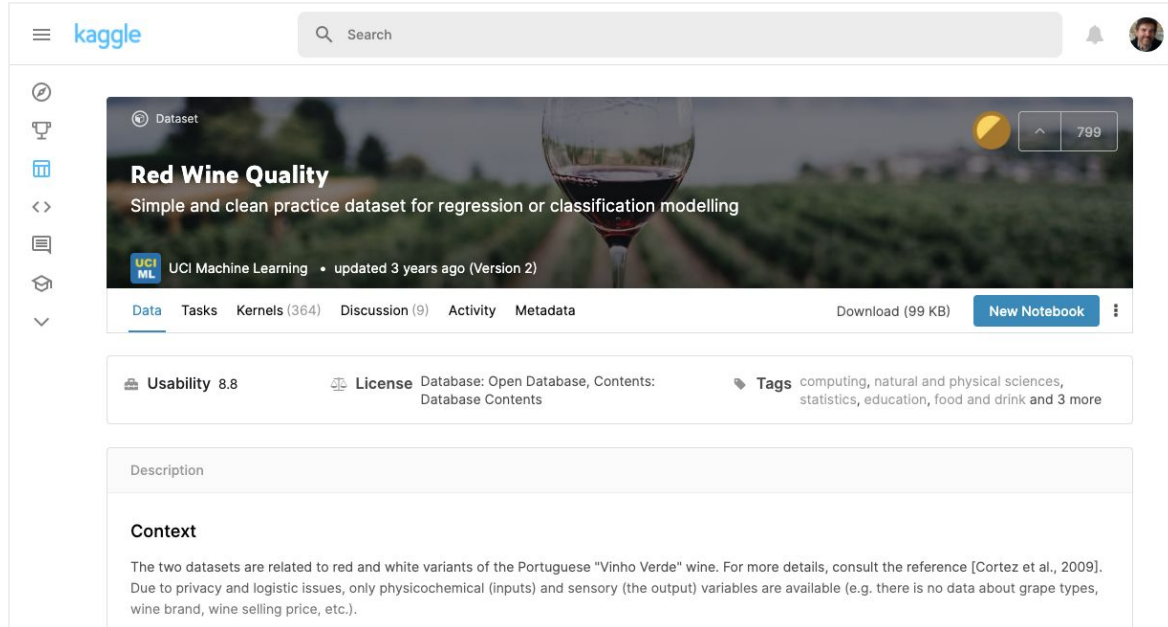
Of course, complexity is Incorta's strength, and in actual production systems, Incorta is able to traverse and join dozens of tables together and quickly feed machine learning model training and inference jobs in an efficient and simple way.

The example is called [Red Wine Quality](#). The task is to use machine learning to determine which physicochemical properties make a wine 'good'. Once we have a good model, we can use this model to predict the quality of new wine batches based on the measurements.

This example is not that far off from a real proof-of-concept done with a leading California winemaker a few years ago -- it was a large operation and they wanted to apply automation to all the measurements to help predict yield and forecast pricing.

In an actual Incorta business instance, you would likely be continuously reading data from production systems, and periodically running predictions on that data, however, this static data set gives us the opportunity to compare our results in Incorta with known results from the data science community.

- ❏ Review the [Red Wine Quality](#) dataset page on Kaggle and download the dataset.



The screenshot shows the Kaggle interface for the 'Red Wine Quality' dataset. At the top, there is a search bar and a user profile icon. The dataset title 'Red Wine Quality' is prominently displayed, along with a subtitle 'Simple and clean practice dataset for regression or classification modelling'. Below the title, it indicates the dataset is from 'UCI Machine Learning' and was updated 3 years ago (Version 2). There are navigation tabs for 'Data', 'Tasks', 'Kernels (364)', 'Discussion (9)', 'Activity', and 'Metadata'. A 'Download (99 KB)' button and a 'New Notebook' button are also visible. The 'Usability' is 8.8, and the license is 'Database: Open Database, Contents: Database Contents'. Tags include 'computing, natural and physical sciences, statistics, education, food and drink and 3 more'. The 'Description' section is partially visible, and the 'Context' section provides additional details about the dataset's origin and limitations.



## Enriching Data with Spark and Notebooks

Before diving into the specifics of machine learning in Incorta, we will see how Incorta uses Spark to enrich data in general.

To continue, you will need an Incorta instance set up and I strongly recommend that you create a trial [Incorta Cloud](#) instance since the default configuration is known to work with this example.

- ❑ Create an account on [Incorta Cloud](#)

Note: If you are running your own instance of Incorta on-premises or on a laptop, be sure that Spark is configured and running. For more information, see: [About Spark](#) in the Incorta documentation.

### Loading data, creating the schema and exploring the data

Once you have successfully signed in to your Incorta Cloud tenant, you can load the sample data.

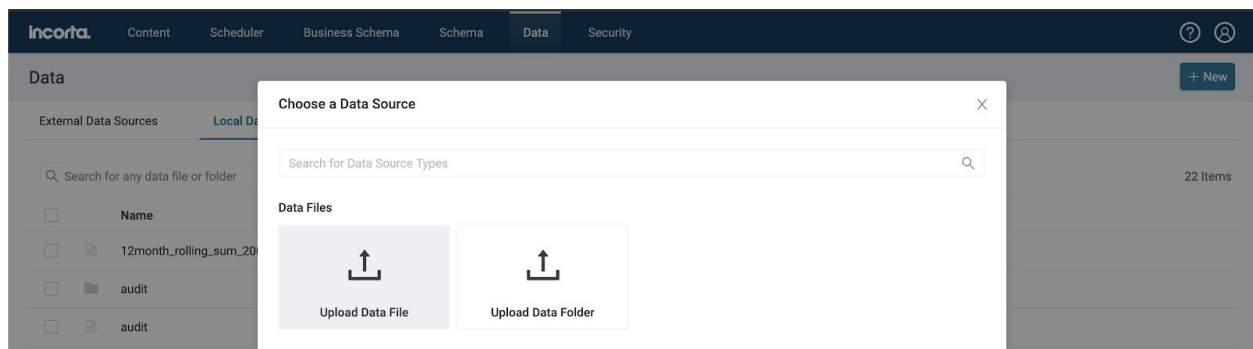
#### Loading the data file

We are going to simply upload the CSV file you downloaded from Kaggle to the Incorta local staging area. Then we will create a schema for the data, load the data into memory, and explore it.

- ❑ Navigate to the **Data** tab, then **Local Data Files**.
- ❑ Click the '+ New' button, and then **Add Data**.

It might be worth pausing here for just a minute to view the extensive list of data sources from which you can import data into Incorta. Everything from databases to applications to data lakes.

- ❑ Click **Upload Data File** from the **Choose a Data Source** dialog.



- ❑ Drag and drop the CSV data file you downloaded from Kaggle to upload it to Incorta.

The list of Local Data Files will update automatically to show the newly uploaded file. Note that it might be on Page 2 of the listing.

#### Creating the Schema

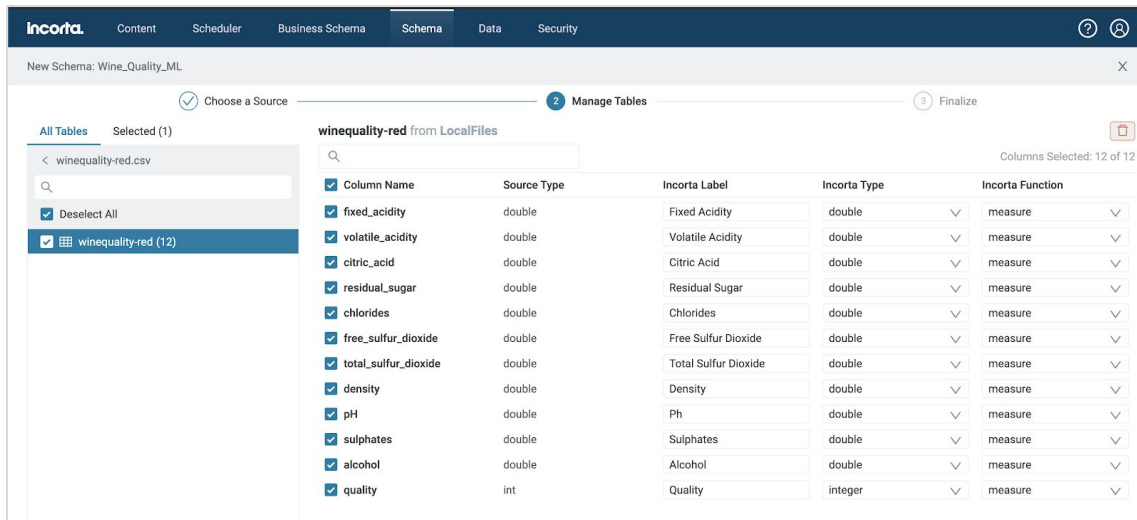
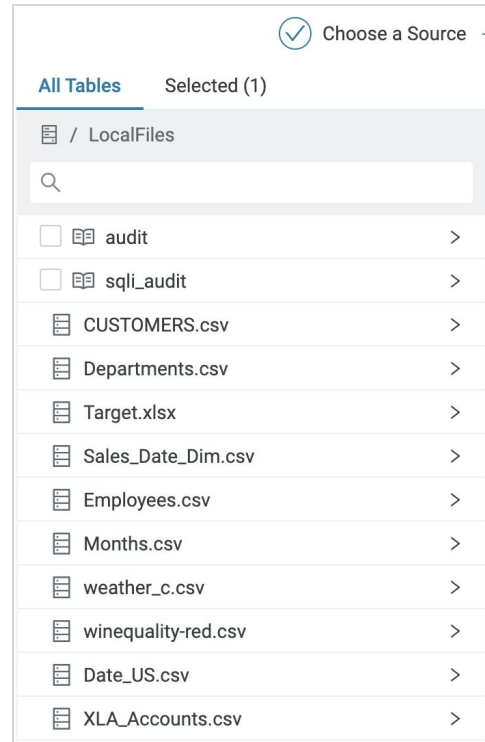
By uploading the data file, we have, in effect, created a data source. Before we can utilize that data, we need to describe it to Incorta, and load it into the analytics engine.

- ❑ Navigate to the **Schema** tab. Schemas are collections of tables.
- ❑ Click the **+ New** button and select **Schema Wizard**.
- ❑ For the **Name**, you can enter `Wine_Quality_ML_Model`
- ❑ For the Datasource, choose **LocalFiles**
- ❑ For the Description you can enter: `Data and notebook for training the wine quality ML model`
- ❑ Click **Next**

On the next page, you will see a list of data sources of type 'LocalFiles'. This might seem odd, as it is mostly a list of CSV files, however, in a production system this would probably be folders of flat files, similar to the `audit` and `sqli_audit` folders.

- Click on `winequality-red.csv` collection.
- Check the box next to `winequality-red` table.

Note that the Schema Wizard has nicely populated the schema for you. No changes are required, so we can accept all the values as-is.



- ❑ Click **Next**, and on the next page, click **Create Schema**.

Now we have a data source, and a schema.

The screenshot shows the Incorta interface with the 'Schema' tab selected. The main header includes 'Content', 'Scheduler', 'Business Schema', 'Schema', 'Data', and 'Security'. A search bar is on the left, and 'Load', 'Explore Data', 'Diagram', and '+ New' buttons are on the right. The main content area displays 'Wine\_Quality\_ML\_Model' with a 'Last Load Status' of 'Please load data'. Below this, a table 'winequality\_red' is shown with columns for 'Table', 'Data Source', 'Performance', 'Columns', 'Joins', 'Rows', and 'Data Size'. The table has 12 columns, 0 joins, 0 rows, and 0.00 KB of data. A 'Joins (0)' section below the table indicates that no joins have been created yet, with a 'Create new join' button.

Next we need to load the data.

## Loading the Data

The Schema has been created, but before we can look at the data, it must be loaded into the analytics engine. Remember, so far it is just a source of data. The load that we are going to perform manually is what is typically scheduled periodically in a production environment.

- ❑ From the Schema tab, choose **Load -> Load Now -> Full**. When the dialog appears, click on **Load**.

The screenshot shows the 'Load' dropdown menu in the Incorta interface. The menu options are 'Full', 'Incremental', and 'Staging'. The 'Load now' option is highlighted, and a red arrow points to the 'Full' option.

The loading will begin, and complete rather quickly. You can check the load status page by clicking on the date and time directly under 'Last Load Status':

The screenshot shows the 'Last Load Status' page with the date and time: May 10, 2020, 08:51:44.

The load status page shows the status of the load job(s), and the count of rows extracted from the data source, then number of rows rejected, and the number of rows loaded.

The screenshot shows the Incorta interface with the following data:

Node Name	Start Time	End Time	Status	Duration
localNode.LoaderService	01:58:30 PM 05/13/2020	01:58:30 PM 05/13/2020	✓	0ms

Table Name	Status	Load Type	Extract Start	Extra...	Extracted Rows	Rejected Rows	Load Start	Load ...	Loaded Rows	Message
winequality_red	Loading Finished	Full	01:58:30 PM 05/13/2020	0ms	1,599	0	01:58:30 PM 05/13/2020	0ms	1,599	✓

You can click on the **Wine\_Quality\_ML\_Model** breadcrumb, to return to the schema.



Now let's have a quick look at the data before diving into machine learning.

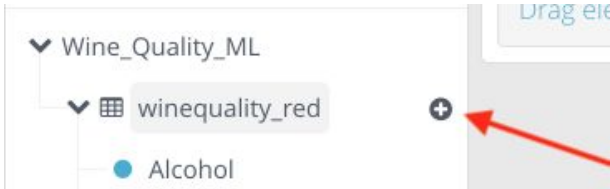
### Exploring the Data

Incorta includes an easy-to-use environment for exploring data, and creating visualizations.

- From the **Schema** page for **Wine\_Quality\_ML\_Model**, click **Explore Data**

This opens the [Analyzer](#). The easiest way to look at all the columns in the table is to click the plus sign to the right of the table name.

- ☐ Click the plus sign to the right of the `winequality_red` table name.



Now you can have a look at the data:

The screenshot shows the Incorta interface with the 'Schema' tab selected. On the left, a navigation pane shows a tree structure under 'Wine\_Quality\_ML\_Model' with 'winequality\_red' expanded. The main area displays a table with 12 columns: Alcohol, Chlorides, Citric Acid, Density, Fixed Acidity, Free Sulfur Dioxide, Ph, Quality, Residual Sugar, Sulphates, Total Sulfur Dioxide, and Volatile Acidity. The table contains 20 rows of data. Above the table, there is a 'Click to Edit Insight Title' and 'Click to Edit Insight Description' section. At the top of the main area, there is a search bar and a 'New Formula' button. The top bar includes the Incorta logo, navigation tabs (Content, Scheduler, Business Schema, Schema, Data, Security), and utility icons (SQL, filters, etc.).

A complete explanation of the Analyzer is beyond the scope of this guide, but feel free to click around. In particular, you might want to see if you can find any correlations between the Quality column and any of the other columns.

## Creating a Simple Data Enrichment

Incorta uses Spark to create Materialized Views of your data. You can create Materialized Views to enrich or harmonize your data, prepare the data for data science activities, or run machine learning. You write code in SQL, Python, Scala or R using a notebook interface.

### Exploring the Notebook interface

Notebooks are a well-known tool for data scientists to explore data and build models. In Incorta, notebooks additionally provide us with a way to create complex data enrichment logic.

The notebook interface is a version of the open-source Apache Zeppelin project.

The output of a notebook goes into a materialized view, a table of derived data. So the first step is to create a materialized view.

- Navigate to **Schema -> Wine\_Quality\_ML\_Model**
- Click the **+** **New** button and choose **Materialized View**
- For the **Language** drop down, choose **Spark Python**
- Under **Script:** click the **Edit in Notebook** button to launch the notebook interface

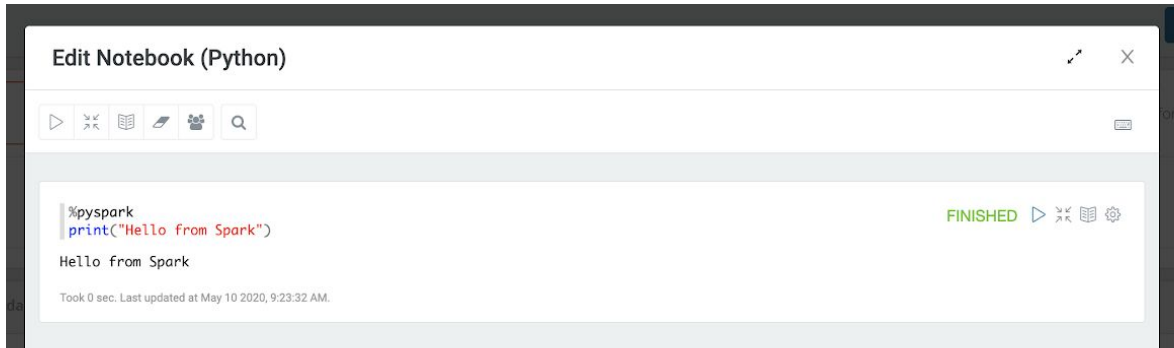
This is a standard notebook, except that a Spark context to the Incorta analytics engine is provided for you. All code is executed by Spark on the server.

- ❑ In the first notebook cell, add a sample line of Python code, and click the run button (the triangle) to execute the cell.

```
%pyspark
print("Hello from Spark")
```

The first time you execute a cell in the Notebook, it might take a couple of seconds for the Spark backend to wake up.

After a few seconds, the result should appear.



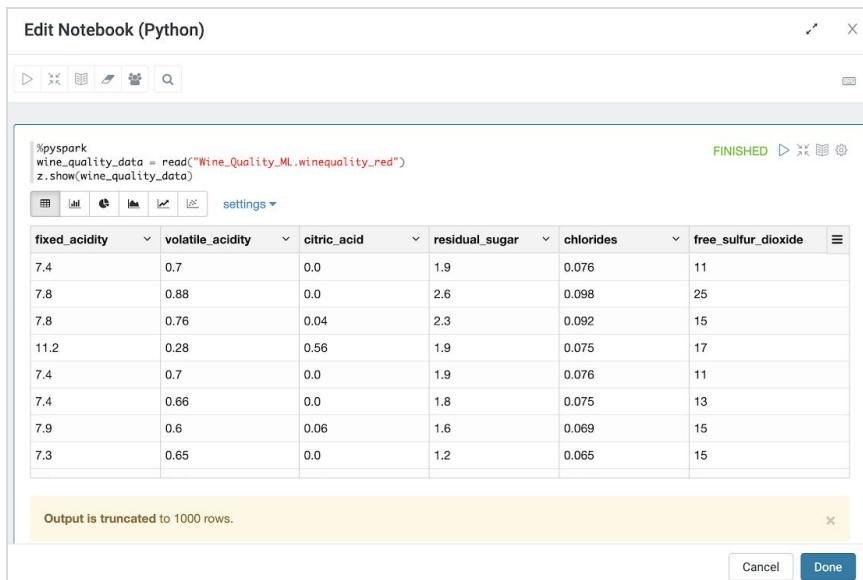
Let's do something slightly more exciting, and read in some data from Incorta.

- ❑ Replace the first notebook cell with the following code and run the cell.

```
%pyspark
wine_quality_data = read("Wine_Quality_ML_Model.winequality_red")
z.show(wine_quality_data)
```

Note that both read() and z.show() are built-in Incorta and Zeppelin notebook functions, respectively.

You should be able to see the data from the Incorta table.



We can also use the built-in charts to examine the data. This example uses standard Spark Dataframe methods to count up the wine batches by the quality rating.

- ❑ In the second notebook cell enter the following code

```
%pyspark
z.show(wine_quality_data.groupby('quality').count())
```

- ❑ Click on the pie chart icon, and then click the **settings** dropdown
- ❑ Drag **count** to the **values** box. Click **settings**.

The screenshot shows a notebook cell with the following code: `%pyspark` and `z.show(wine_quality_data.groupby('quality').count())`. The status is **FINISHED**. The visualization toolbar includes a pie chart icon. The **Available Fields** section lists `quality` and `count`. The **keys** box contains `quality`. The **values** box contains `count SUM`. A legend at the bottom shows color-coded dots for quality values 3 through 8. **Cancel** and **Done** buttons are at the bottom right.

The visualizations are made available when you invoke the `z.show()` function.

Now we can see a pie chart showing the distribution of the `quality` values:

The screenshot shows a notebook cell with the following code: `%pyspark` and `z.show(wine_quality_data.groupby('quality').count())`. The status is **READY**. The visualization toolbar includes a pie chart icon. The legend at the bottom shows color-coded dots for quality values 3 through 8. The pie chart shows a distribution where quality 5 is the largest segment, followed by quality 6, quality 7, quality 4, and quality 3.

## Adding a Derived Column

To make the data more meaningful to users looking at dashboards, we would like to add a classification to each of the rows based on the `quality` column.

We will classify the wine quality measurement into three categories.

- If the quality is 4 or less, then we will classify the wine as “Bad”. Perhaps we can make cooking wine from it.
- If the quality is 5 or 6 (the bulk of our sample) then we will classify the wine as “Good”. This will be the wine that we sell at a normal price.
- If the quality is 7 or better, then the wine will be classified as “Excellent”, and we can charge a premium for these batches!

Let’s add some code to create this new column.

- ❑ In the next notebook cell, add the following Python code, and click the run button.

```
%pyspark
from pyspark.sql.functions import *

wine_quality_data = wine_quality_data.withColumn("class", \
    when(col('quality') >= 7, "Excellent") \
    .otherwise(when(col('quality') >= 5, "Good").otherwise("Bad")))

wine_quality_data.select('quality', 'class').groupBy('quality', 'class').count().sort('quality').show()
```

That code is a little convoluted, but it gets the job done.

The last line of code is just to print a table of all the values we generated, and it shows off several of the PySpark DataFrame SQL functions.

```
%pyspark
from pyspark.sql.functions import *

wine_quality_data = wine_quality_data.withColumn("class", \
    when(col('quality') >= 7, "Excellent") \
    .otherwise(when(col('quality') >= 5, "Good").otherwise("Bad")))

wine_quality_data.select('quality', 'class').groupBy('quality', 'class').count().sort('quality').show()
```

```
+-----+-----+-----+
|quality|  class|count|
+-----+-----+-----+
|      3|    Bad|   10|
|      4|    Bad|   53|
|      5|   Good|  681|
|      6|   Good|  638|
|      7|Excellent| 199|
|      8|Excellent|  18|
+-----+-----+-----+
```



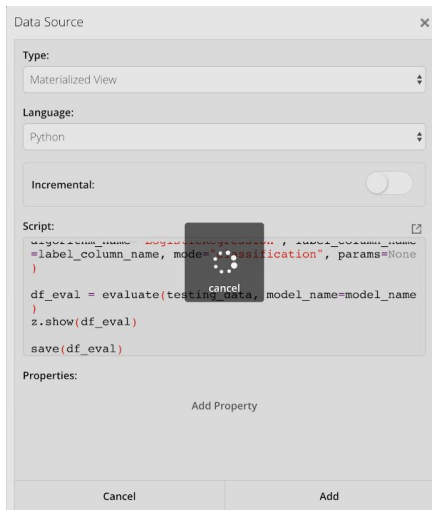
Now that we have added the column to the DataFrame, all we need to do is save it back to the Incorta Materialized View. This can be very simply done with the `save()` function, which saves a dataset to the Incorta table:

- ❑ Add the following cell at the bottom of the notebook

```
%pyspark
save(wine_quality_data)
```

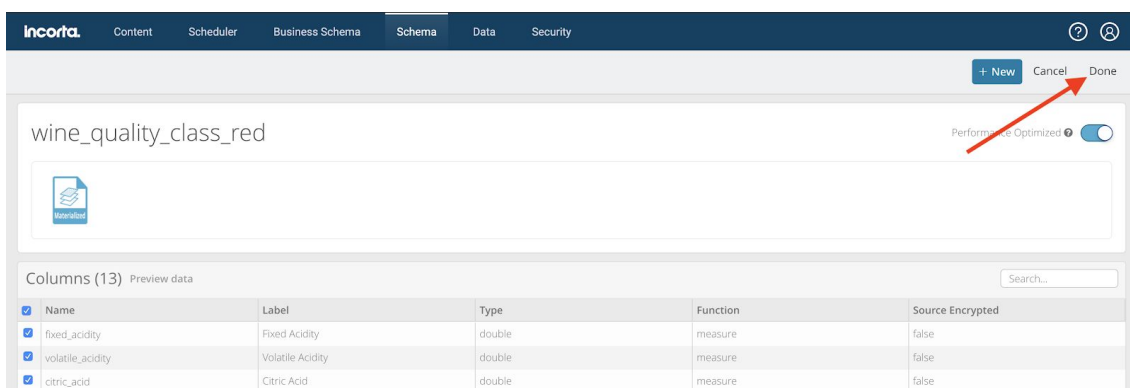
- ❑ Click **Done** at the bottom of the notebook. This returns you to the Data Source dialog, but with the Script filled in.
- ❑ Click **Add** in the Data Source dialog box.

After you click add, the script will be saved. This will take a minute or two to complete.



Give the new materialized view a name.

- ❑ Type `wine_quality_class_red` in the **Table Name** field highlighted in red.
- ❑ Click **Done**. (Important!)

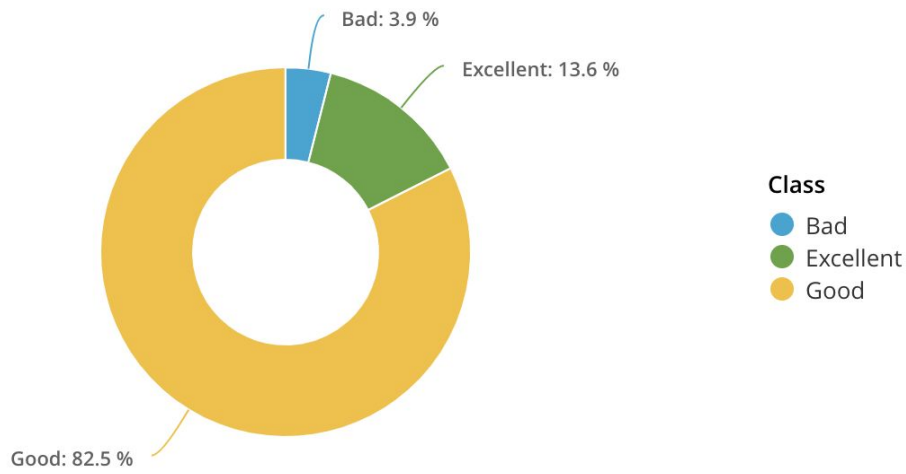


- ❑ From the **Schema -> Wine\_Quality\_ML\_Model** page, choose **Load -> Load Now -> Full**. When the dialog appears, click on **Load**.

The materialized view is populated using the notebook code.

❏ Click Explore Data to see the data enrichment results.

See if you can create a chart showing the number of 'Excellent', 'Good', and 'Bad' wine batches.



# A Simple Machine Learning Example in Incorta

## Training a Model using Spark ML

Data enrichment logic in Incorta can be developed in SQL, Python, Scala and R. You can utilize many different libraries and tools, and are not limited to the Incorta-bundled libraries.

To help you understand the range of solutions that are possible with Incorta, we will first create a machine learning model using the standard machine learning libraries that come with Spark, then we will try using the new Incorta ML library for comparison.

Start by creating a new Materialized View.

- ❑ Navigate to **Schema -> Wine\_Quality\_ML\_Model**
- ❑ Click the **+ New** button and choose **Materialized View**
- ❑ For the **Language** drop down, choose **Spark Python**
- ❑ Under **Script:** click the **Edit in Notebook** button to launch the notebook interface

First we will read in the data that we loaded earlier.

- ❑ In the first notebook cell, add the following Python code, and click the run button.

```
%pyspark
df = read("Wine_Quality_ML_Model.winequality_red")
df.take(1)
```

The second line of code just prints out one of the rows for inspection.

```
%pyspark
df = read("Wine_Quality_ML_Model.winequality_red")
df.take(1)

[Row(fixed_acidity=7.4, volatile_acidity=0.7, citric_acid=0.0, residual_sugar=1.9, chlorides=0.076, free_sulfur_dioxide=11.0, total_sulfur_dioxide=34.0, density=0.9978, pH=3.51, sulphates=0.56, alcohol=9.4, quality=5)]

Took 3 sec. Last updated at May 19 2020, 3:53:32 PM.
```

Next, the Spark ML LinearRegression algorithm expects all the features to be assembled into a single vector column.

At this point, many data scientists choose to explore the data, looking for potential correlations and possibly doing some feature engineering (derivations of features from other features to improve the model accuracy.) For the sake of simplicity, we are going to just use all the numeric features that are supplied with the dataset.

We will use the Pyspark ML `VectorAssembler` to assemble the features. First, we need to create a list of feature columns, leaving out the label column.

Note that we have to drop the `quality` column because we are trying to get the machine learning algorithm to do a classification, and since we derived the three classes from the `quality` column, it would indicate the class. In other words, we would not normally be given the wine quality in advance, so this would be cheating!

❑ In the next notebook cell, add the following Python code, and click the run button.

```
%pyspark
features = df.columns
features.remove('quality') # quality is the label or target
features
```

Next we will assemble the vector and add it to a new dataset.

Note that Spark DataFrames are immutable, so we create a new DataFrame every time we do a transformation. To make debugging easier, we are storing the new DataFrame into a new variable each time.

```
%pyspark
from pyspark.ml.feature import VectorAssembler

# Assemble the features into a vector column
assembler = VectorAssembler(inputCols=features, outputCol='features')
vdf = assembler.transform(df)
vdf.take(1)
```

Now we can see a new `DenseVector` has been added to the DataFrame.

```
%pyspark
from pyspark.ml.feature import VectorAssembler

# Assemble the features into a vector column
assembler = VectorAssembler(inputCols=features, outputCol='features')
vdf = assembler.transform(df)
vdf.take(1)
```

FINISHED ▶ ⌘ 📖 ⚙️

```
[Row(fixed_acidity=7.4, volatile_acidity=0.7, citric_acid=0.0, residual_sugar=1.9, chlorides=0.076, free_sulfur_dioxide=11.0, total_sulfur_dioxide=34.0, density=0.9978, pH=3.51, sulphates=0.56, alcohol=9.4, quality=5, features=DenseVector([7.4, 0.7, 0.0, 1.9, 0.076, 11.0, 34.0, 0.9978, 3.51, 0.56, 9.4]))]
```

The final bit of preparation before we train the model, is to split the data into a training data set (we will take 80% of the data for training) and a testing data set (the remaining 20%).

For those new to machine learning, the training data set is what is used to train the model on the data, and the testing data set (sometimes called a 'holdout') is used to test the accuracy of the trained model.

```
%pyspark
# Split the dataset into test and train
df_train, df_test = vdf.randomSplit([0.8, 0.2])
vdf.count(), df_train.count(), df_test.count()
```

Next we will train the model using the training data set.

```
%pyspark
from pyspark.ml.regression import LinearRegression

# Train the model
model = LinearRegression(labelCol='quality',
featuresCol='features').fit(df_train)

print("r2: %f" % model.summary.r2)
```

The model trains and then prints the r2 correlation metric:

```
%pyspark
from pyspark.ml.regression import LinearRegression

# Train the model
model = LinearRegression(labelCol='quality', featuresCol='features').fit(df_train)

print("r2: %f" % model.summary.r2)

r2: 0.367388
```

Took 9 sec. Last updated at June 05 2020, 10:33:55 AM.

Now we can test the model we have built against the test dataset.

```
# Run the model with the test data
predictions = model.transform(df_test)

# Evaluate the model using the test data
test_results = model.evaluate(df_test)
print("r2 on test data = %g" % test_results.r2)
print("Root Mean Squared Error (RMSE) on test data = %g" %
test_results.rootMeanSquaredError)
```

Since this is just an example, and we haven't really done any feature engineering, the result isn't impressive and won't win us any Kaggle competitions:

```
%pyspark
# Run the model with the test data
predictions = model.transform(df_test)

# Evaluate the model using the test data
test_results = model.evaluate(df_test)
print("r2 on test data = %g" % test_results.r2)
print("Root Mean Squared Error (RMSE) on test data = %g" % test_results.rootMeanSquaredError)

r2 on test data = 0.322027
Root Mean Squared Error (RMSE) on test data = 0.62859
```

The last thing we will do before moving on is to save our trained model to the filesystem, training results back to an Incorta table (the materialized view). That way, we can show results on a report or dashboard if we so choose.

First save the model to the filesystem:

```
%pyspark
# Save the trained model to the filesystem
model.write().overwrite().save("models/wine_quality_001")
```

This code simply creates a dataframe and puts the test results in it. The last line saves the dataframe to the materialized view and is a function supplied by Incorta.

```
%pyspark
# Save the training results
from pyspark.sql.types import *

cSchema = StructType([StructField("metric_name", StringType())\
                      ,StructField("value", DoubleType())])

metrics = [['r2', test_results.r2], ['Root Mean Squared Error',
test_results.rootMeanSquaredError]]

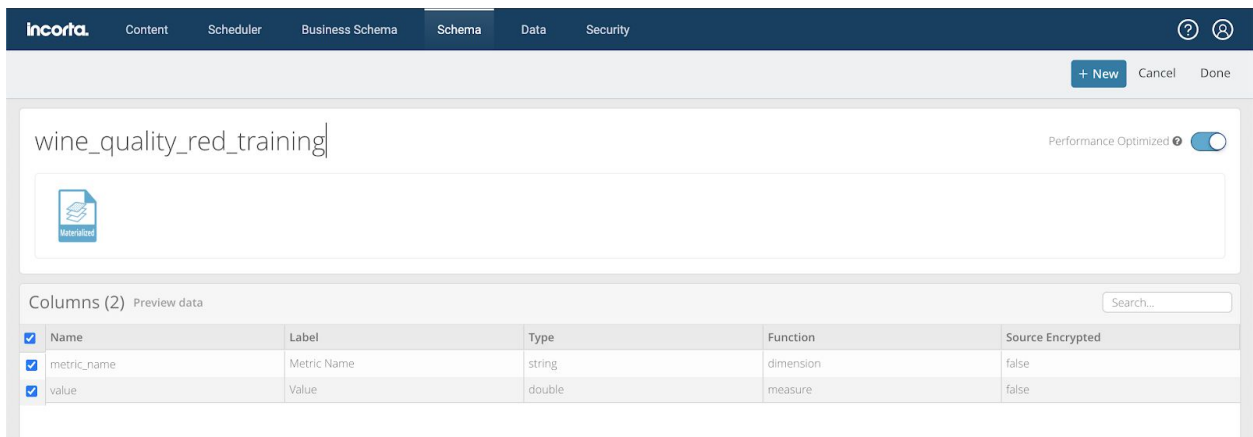
df = spark.createDataFrame(metrics, schema=cSchema)
save(df)
```

- Click **Done** at the bottom of the notebook. This returns you to the Data Source dialog, but with the Script filled in.
- Click **Add** in the Data Source dialog box.

After you click **Add**, the script will be saved. This will take a minute or two to complete.

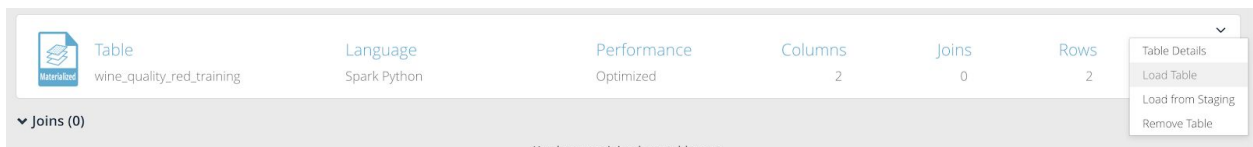
Give the new materialized view a name.

- Type `wine_quality_red_training` in the **Table Name** field highlighted in red.
- Click **Done**. (Important!)



Once again, we need to 'load' the data so we can see it -- in this case we are training the model and 'loading' the training results.

- ❑ From the **Schema -> Wine\_Quality\_ML\_Model** page, choose **Load -> Load Now -> Full**. When the dialog appears, click on **Load**.



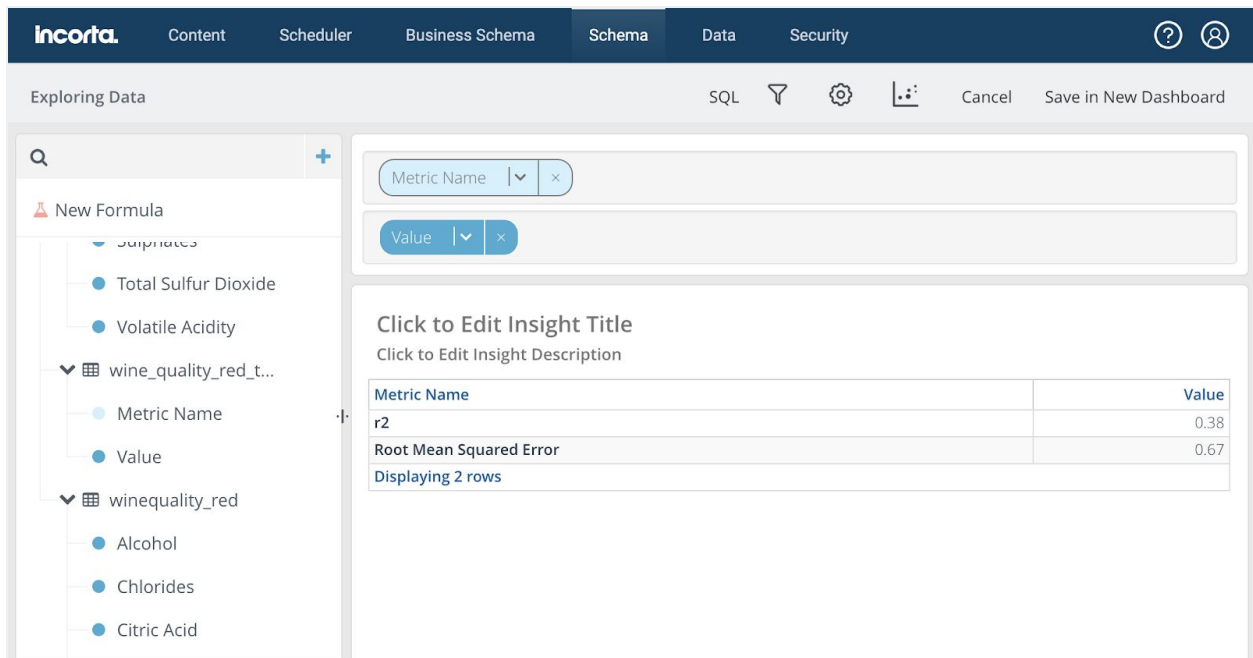
You can also choose **Load Table** for just the `wine_quality_red_training` table as shown above.

This process will take a minute or so to complete.

The materialized view is populated using the notebook code.

- ❑ Click **Explore Data** to see the model training results.
- ❑ In the list of tables on the left, you will see the `wine_quality_red_training` table, a table with only two columns. Add both columns to the visualization.

Now we can see the metrics associated with the model training, and we could put these on a dashboard if we wanted to.



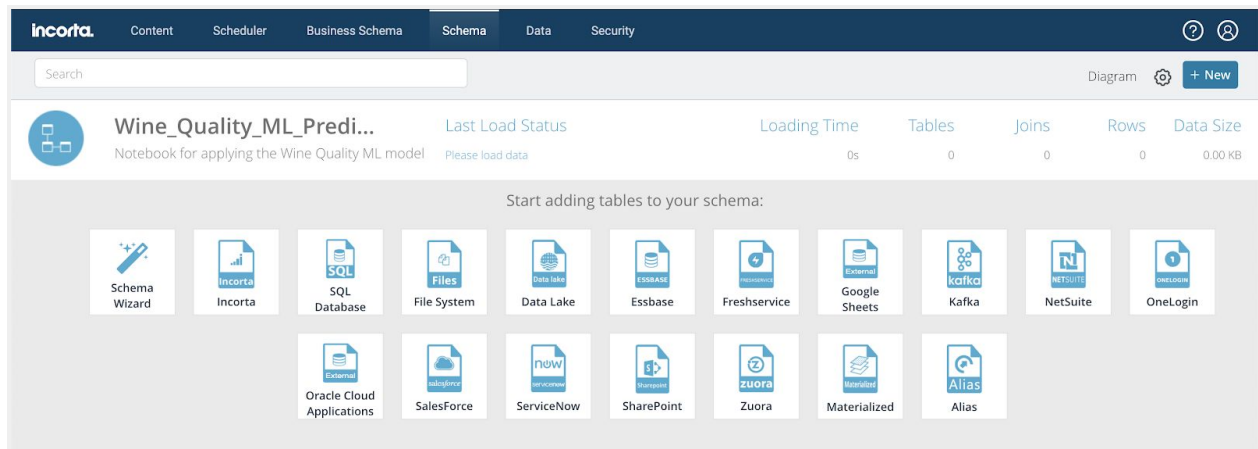
## Running Predictions with Spark ML

Now that we have trained a model and saved it, we can apply the model to our data over and over again.

Note that since this is a small and simple data set, we are going to simply show you how you would productionalize a model using the same data. In a real system, you would add predictions to new incoming data.

Create a new schema for running predictions.

- Select the **Schema** tab, then choose **+New** and **Create Schema**
- Set the Name of the schema to: **Wine\_Quality\_ML\_Prediction**
- Set the Description to: **Notebook for applying the Wine Quality ML model**





Next, create a Materialized View that will hold the wine quality data with the added predictions.

- ❑ Select **Materialized View**
- ❑ Set the **Language to Spark Python**
- ❑ Click **Edit in Notebook**

To keep things simple, we are going to just reuse the existing data. Again, in a real deployment, we would be reading new data from a remote system or data lake.

It is a best practice to import all the Python classes we need at the top of the notebook.

- ❑ In the first notebook cell, add the following Python code.

```
%pyspark
from pyspark.ml.regression import LinearRegressionModel
from pyspark.ml.feature import VectorAssembler
```

Next we read in the data that we want to score -- this is just the data that we loaded earlier for this example. To simulate new data has not yet been scored, we are removing the `quality` column. We will predict this column instead. Again, this is just to keep the example simple.

- ❑ In the first notebook cell, add the following Python code.

```
%pyspark
# Simulate reading in new (unscored) wine quality data
df = read("Wine_Quality_ML_Model.winequality_red").drop('quality')
df.take(1)
```

When we trained the model above, we saved it to disk. Here we load that same model from disk.

- ❑ Load the trained model from disk

```
%pyspark
model = LinearRegressionModel.load("models/wine_quality_001")
```

Next, when we call the `transform()` method to apply the model to the dataset, it expects to see all the features in exactly the same format as when the model was trained.

- ❑ Create the features column as a `DenseVector`

```
%pyspark
# Assemble the features into a vector column
assembler = VectorAssembler(inputCols=df.columns, outputCol='features')
vdf = assembler.transform(df)
vdf.take(1)
```

```
%pyspark
# Assemble the features into a vector column
assembler = VectorAssembler(inputCols=df.columns, outputCol='features')
vdf = assembler.transform(df)
vdf.take(1)
```

FINISHED ▶ ⌂ 📖 ⚙️

```
[Row(fixed_acidity=7.4, volatile_acidity=0.7, citric_acid=0.0, residual_sugar=1.9, chlorides=0.076, free_sulfur_dioxide=11.0, total_sulfur_dioxide=34.0, density=0.9978, pH=3.51, sulphates=0.56, alcohol=9.4, features=DenseVector([7.4, 0.7, 0.0, 1.9, 0.076, 11.0, 34.0, 0.9978, 3.51, 0.56, 9.4]))]
```

Finally, we will run the model, adding the predictions, the model output, to the dataset, and saving the dataset back to Incorta.

- Apply the model to the dataset and save the results to the materialized view.

```
%pyspark
predictions = model.transform(vdf)
save(predictions)
```

- Click **Done** at the bottom of the notebook. This returns you to the Data Source dialog, but with the Script filled in.
- Click **Add** in the Data Source dialog box.

After you click **Add**, the script will be saved. This will take a minute or two to complete.

Give the new materialized view a name.

- Type `wine_quality_red_prediction` in the **Table Name** field highlighted in red.
- Click **Done**. (Important!)

## Scheduling Predictions

Instead of running the model directly, by running the materialized view code (which is done by loading the schema) we will schedule the process so that it runs periodically.

The Incorta Scheduler can do this for us.

- From the **Scheduler** page, choose the subtab **Schema Loads**
- Choose **+New -> Add Schema Load**
- For **Schema Load Name**, use `Wine Quality Prediction`
- For Schema, choose `Wine_Quality_ML_Prediction`
- Set the **Every** (time period) to **1 Hour(s)** and set the **Starting at** time to a couple of minutes in the future.
- Click **Add**

### Add Schema Load ✕

Schema Load Name

Description

Schema

Load Type

Every  
1  Day(s)  No recurrence

Starting at  
4:27 PM  GMT-07:00

From  
2020-06-11  2020-06-12   Does not end

After a short time (or whatever time you set for the starting time) notebook code in the materialized view will run and the schema will be loaded.

- Select the **Schema -> Wine\_Quality\_ML\_Prediction** page
- Confirm the Last Load Status for the schema.

Table	Language	Performance	Columns	Joins	Rows	Data Size
wine_quality_red_prediction	python	Optimized	13	0	2K	71.67 KB

Now we have data with a machine learning prediction that we can examine.

- Click **Explore Data** to see the data enrichment results.
- In the list of tables on the left, you will see the `wine_quality_red_prediction` table. Click the plus sign to the right of the table name to add all the columns to the analyzer.

Now we can see all the original columns, plus the **Prediction** column that contains the machine learning results.

Incorta. Content Scheduler Business Schema Schema Data Security

Exploring Data SQL ↑↓ Filter Settings Cancel Save in New Dashboard

Drag elements here for grouping dimension

Alcohol Chlorides Citric Acid Density Fixed Acidity Free Sulfur Dioxide Ph Prediction Residual Sugar Sulphates Total Sulfur Dioxide Volatile Acidity

New Formula

Wine\_Quality\_ML\_Prediction

- wine\_quality\_red...
- Alcohol
- Chlorides
- Citric Acid
- Density
- Fixed Acidity
- Free Sulfur Dioxide
- Ph
- Prediction
- Residual Sugar
- Sulphates
- Total Sulfur Dioxide
- Volatile Acidity

Click to Edit Insight Title

Click to Edit Insight Description

Alcohol	Chlorides	Citric Acid	Density	Fixed Acidity	Free Sulfur Dioxide	Ph	Prediction	Residual Sugar	Sulphates	Total Sulfur Dioxide	Volatile Acidity
9.40	0.08	0.00	1.00	7.40	11.00	3.51	5.06	1.90	0.56	34.00	0.70
9.80	0.10	0.00	1.00	7.80	25.00	3.20	5.17	2.60	0.68	67.00	0.88
9.80	0.09	0.04	1.00	7.80	15.00	3.26	5.24	2.30	0.65	54.00	0.76
9.80	0.07	0.56	1.00	11.20	17.00	3.16	5.71	1.90	0.58	60.00	0.28
9.40	0.08	0.00	1.00	7.40	11.00	3.51	5.06	1.90	0.56	34.00	0.70
9.40	0.07	0.00	1.00	7.40	13.00	3.51	5.09	1.80	0.56	40.00	0.66
9.40	0.07	0.06	1.00	7.90	15.00	3.30	5.13	1.60	0.46	59.00	0.60
10.00	0.07	0.00	0.99	7.30	15.00	3.39	5.37	1.20	0.47	21.00	0.65
9.50	0.07	0.02	1.00	7.80	9.00	3.36	5.37	2.00	0.57	18.00	0.58
10.50	0.07	0.36	1.00	7.50	17.00	3.35	5.65	6.10	0.80	102.00	0.50
9.20	0.10	0.08	1.00	6.70	15.00	3.28	5.07	1.80	0.54	65.00	0.58
10.50	0.07	0.36	1.00	7.50	17.00	3.35	5.65	6.10	0.80	102.00	0.50
9.90	0.09	0.00	0.99	5.60	16.00	3.58	5.13	1.60	0.52	59.00	0.61
9.10	0.11	0.29	1.00	7.80	9.00	3.26	5.97	1.60	1.56	29.00	0.61
9.20	0.18	0.18	1.00	8.90	52.00	3.16	5.16	3.80	0.88	145.00	0.62
9.20	0.17	0.19	1.00	8.90	51.00	3.17	5.20	3.90	0.93	148.00	0.62

<< < 1 - 1,000 of 1,599 > >>

Collapse All

---

## Machine Learning with Incorta-ML

So far we have used Incorta Materialized Views and Notebooks to enrich data, and train a machine learning model, and then put that model into ‘production’ scoring new data automatically.

To create the machine learning model, we used the built-in Spark ML libraries, but could have used any open source or commercial machine learning library such as Scikit-learn, or H2O.ai.

In this last section, we will try the Incorta-ML library, an experimental Python wrapper over Spark ML that provides several benefits to both data scientists and also users who are new to machine learning.

- Supports both regression and classification along with time series predictions
- Automates feature encoding
- Automates feature selection – ignores non-significant features
- Automatically selects the best model based on the dataset provided
- Also supports manual algorithm selection
- Automatically evaluates models based on multiple metrics
- Saves chosen model for later prediction. Train once, predict many times.

The following example will differ from the prior examples in these ways:

- We will be using Incorta-ML to simplify the data preparation
- We will be invoking Incorta-ML’s auto-ML feature to automatically select an algorithm.
- This will be a classification problem instead of a regression problem, combining aspects of both the simple enrichment and simple ML examples.

As we did in the simple enrichment example, what we want to do is classify the wine measurements into three categories.

- If the quality is  $\leq 4$ , then we will classify the wine as “Bad”.
- If the quality is between 5 and 7 then we will classify the wine as “Good”.
- If the quality is  $\geq 7$  then the wine will be classified as “Excellent”.

The machine learning models require these classifiers to be numbers, so they will be encoded as 0 (bad), 1 (good) and 2 (excellent).

### Prepare the Data

Start by creating a new Materialized View.

If you didn’t follow the steps above for enriching data, you’ll need to go there now and connect to the data, create the schema and load the data.

- Navigate to **Schema -> Wine\_Quality\_ML\_Model**
- Click the **+ New** button and choose **Materialized View**
- For the **Language** drop down, choose **Spark Python**
- Under **Script:** click the **Edit in Notebook** button to launch the notebook interface

Now start to build your notebook.

- ❑ In the first cell, import the machine learning classes

```
%pyspark
from incorta_ml import *
from pyspark.ml.feature import *
from sklearn.metrics import confusion_matrix
```

To create the quality labels, we use the Bucketizer transformer.

- ❑ In the next cell, enter in the following code and run it.

```
%pyspark
# Read the data
wine_quality_data = read("Wine_Quality_ML_Model.winequality_red")

# Create a new DF with buckets
label_column_name = "bucketed_quality"
bucketizer = Bucketizer(splits=[float('-inf'), 5, 7, float('Inf')],
inputCol="quality", outputCol=label_column_name)
bucketed_wine_quality_data =
bucketizer.setHandleInvalid("keep").transform(wine_quality_data)
z.show(bucketed_wine_quality_data)
```

The bucket numbers are in the last column:

density	pH	sulphates	alcohol	quality	bucketed_quality
0.9981	3.11	0.97	10.1	6	1
0.9924	3.57	0.85	13	7	2
0.9948	3.51	0.43	11.4	4	0.0
0.99695	3.22	0.82	10.3	7	2
0.9975	3.04	1.03	9.3	5	1
0.99545	3.36	0.79	9.5	5	1
0.9961	3.34	0.55	9.2	5	1

Next we will split the data into training and test datasets.

- ❑ In the next cell, type in the following code and run it.

```
%pyspark
(training_data, testing_data) =
bucketed_wine_quality_data.drop('quality').randomSplit([0.7, (1-0.7)],
seed=42)
training_data.count(), testing_data.count()
```

As before, we have to drop the quality column because it is the target column (or label).

## Train the Model

Now it is finally time to build our machine learning model. We are going to let Incorta's Auto-ML choose the algorithm, which it will do by trying different algorithms and parameters, and examining the training metrics.

- ❑ In the next cell, type in the following code and run it.

```
model_name = "WineQuality_LogisticRegression"
build_model(training_data, model_name=model_name,
            algorithm_name="LogisticRegression", label_column_name=label_column_name,
            mode="classification", params=None)

model_name = "WineQuality_Auto"
build_model(training_data, model_name=model_name, algorithm_name="auto",
            label_column_name=label_column_name, mode="classification", params=None)
```

This will take a few seconds to run.

After a couple of minutes, the training complete and the chosen algorithm is reported:

```
%pyspark
model_name = "WineQuality_Auto"
build_model(training_data, model_name=model_name, algorithm_name="auto", label_column_name=label_column_name, mode
            ="classification", params=None)

('Best algorithm name', 'RandomForestClassifier')

Took 2 min 34 sec. Last updated at June 11 2020, 5:14:42 PM.
```

The output is a trained model that knows how to grade the wine 0, 1, or 2 based on all the measurements which is automatically saved to disk under the given `model_name`.

## Save the Training Metrics

Next, we evaluate the model training accuracy using the testing data set that we held out earlier.

- ❑ In the next cell, type in the following code and run it.

```
df_eval = evaluate(testing_data, model_name=model_name)
z.show(df_eval)
```

This will provide us with the model training results:

```
%pyspark
df_eval = evaluate(testing_data, model_name=model_name)
z.show(df_eval)
```

FINISHED ▶ ⌂ ⚙

settings ▾

metric_name	value
f1	0.8189645271979406
weightedPrecision	0.8390931871324028
weightedRecall	0.845679012345679
accuracy	0.845679012345679

The final step is to save these results for future reference to the materialized view table. This can be very simply done with the `save()` function, which saves a dataset to the Incorta table:

```
save(df_eval)
```

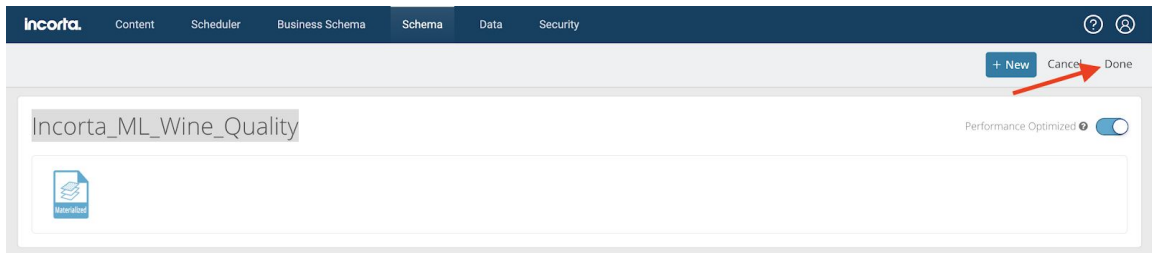
- ❑ Click **Done** at the bottom of the notebook. This returns you to the Data Source dialog, but with the Script filled in.
- ❑ Click **Add** in the Data Source dialog box

After you click add, the script will be saved. This will take a minute or two to complete.

Give the new materialized view a name.

- ❑ Type `ML_Train_Wine_Quality` in the **Table Name** field highlighted in red.
- ❑ Click **Done**. (Important!)





- ❑ From the **Schema -> Wine\_Quality\_ML\_Model** page, choose **Load -> Load Now -> Full**. When the dialog appears, click on **Load**.

The materialized view is populated with the accuracy metrics from the model training.

- ❑ Click **Explore Data** and then add the **ML\_Train\_Wine\_Quality** table to the analyzer to see the training results

## Running Predictions

Now that we have a trained model, we can deploy the model such that it can continuously run predictions on new incoming data.

Note that since we are not running a production system, we are just going to use the same data file that we used for model training, simulating new data. In a real system, we would be running predictions against actual new data.

## Creating the Production Schema

In order to run predictions on a separate schedule from training, we will create a separate schema where we will run the ML model 'in production'.

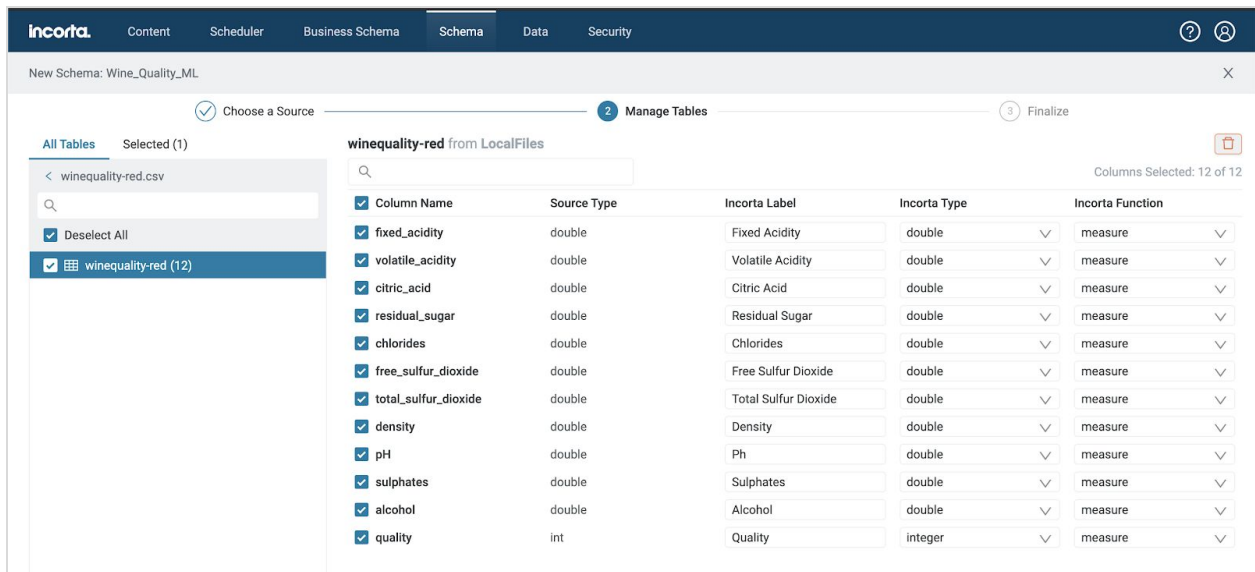
- ❑ Navigate to the **Schema** tab.
- ❑ Click the **'+ New'** button and select **Schema Wizard**.
- ❑ For the name, you can enter **Wine\_Quality\_ML\_Predict**
- ❑ For the Datasource, choose **LocalFiles**, click **Next**

As before, choose the CSV file that was uploaded earlier:

- Click on **winequality-red.csv** collection.
- Check the box next to **winequality-red** table.

Once again, the Schema Wizard has nicely populated the schema for you.

- ❑ Click **Next**, and on the next page, click **Create Schema**.



## Creating the Prediction Notebook

To access the notebook UI, start by creating a materialized view.

- Navigate to **Schema -> Wine\_Quality\_ML\_Predict**
- Click the '+ New' button and choose **Materialized View**
- For the **Language** drop-down, choose **Python**
- Under **Script**: click the **Edit in Notebook** button to launch the notebook interface

For each of the code blocks below, assume that you should place the code into a new notebook cell, and try it out by running the cell.

- Read in the wine data as before.

```
%pyspark
from incorta_ml import *
from pyspark.ml.feature import *

new_wine_data = read("Wine_Quality_ML_Predict.winequality_red")
z.show(new_wine_data)
```

Nothing new to see here, it is the same data as before. Remember, we are simulating new data.

- Now run the model on the "new" data.

```
%pyspark
model_name = "WineQuality_LogisticRegression"
wine_data_scored = predict(new_wine_data, model_name)
z.show(wine_data_scored)
```

This will take a few seconds to run. The `model_name` refers to the model that we trained and saved earlier.

In the very last column of results, you will see a new 'prediction' column.

```
%pyspark
model_name = "WineQuality_LogisticRegression"
wine_data_scored = predict(new_wine_data, model_name)
z.show(wine_data_scored)
```

FINISHED ▶ ✕ 📖 ⚙️

📊 📈 📉 📊 📈 📉 settings ▾

density	pH	sulphates	alcohol	quality	prediction
0.9978	3.51	0.56	9.4	5	1
0.9968	3.2	0.68	9.8	5	1
0.997	3.26	0.65	9.8	5	1
0.998	3.16	0.58	9.8	6	1
0.9978	3.51	0.56	9.4	5	1
0.9978	3.51	0.56	9.4	5	1
0.9964	3.3	0.46	9.4	5	1
0.9946	3.39	0.47	10	7	2

Before we save this data to Incorta, let's convert the numeric prediction to something with more meaning.

```
%pyspark
from pyspark.sql.functions import col, when
output_df = wine_data_scored.withColumn("grade",
    when(col("prediction") == 0, 'Bad')
    .when(col("prediction") == 1, 'Good')
    .when(col("prediction") == 2, 'Excellent')
    .otherwise('Unknown') )
z.show(output_df)
```

The last column will now show a consumer-friendly 'Grade' column:

```
%pyspark
from pyspark.sql.functions import col, when
output_df = wine_data_scored.withColumn("grade",
    when(col("prediction") == 0, 'Bad')
    .when(col("prediction") == 1, 'Good')
    .when(col("prediction") == 2, 'Excellent')
    .otherwise('Unknown'))
z.show(output_df)
```

FINISHED ▶ ⌂ ⚙

settings ▾

	pH	sulphates	alcohol	quality	prediction	grade
	3.23	0.73	9.7	7	2	Excellent
00001	3.5	0.48	9.8	4	0.0	Bad
	3.33	0.83	10.5	5	1	Good
	3.33	0.83	10.5	5	1	Good
	3.26	0.51	9.3	4	0.0	Bad
	3.21	0.9	10.5	6	1	Good

The final step is to save these results to the materialized view table.

- In the next cell, add the following code.

```
%pyspark
save(output_df)
```

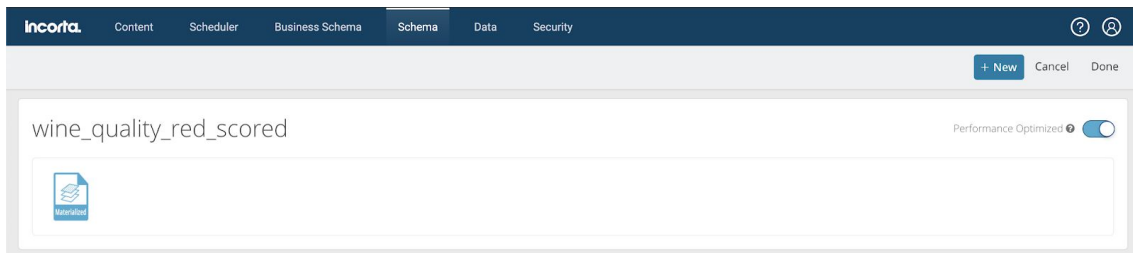
Save the notebook.

- Click **Done** at the bottom of the notebook. This returns you to the Data Source dialog, but with the Script filled in.
- Click **Add** in the Data Source dialog box

After you click add, the script will be saved. This will take a minute.

Give the new materialized view a name.

- Type `wine_quality_red_scored` in the **Table Name** field highlighted in red.
- (Important!) Click **Done** in the upper right corner of the page.



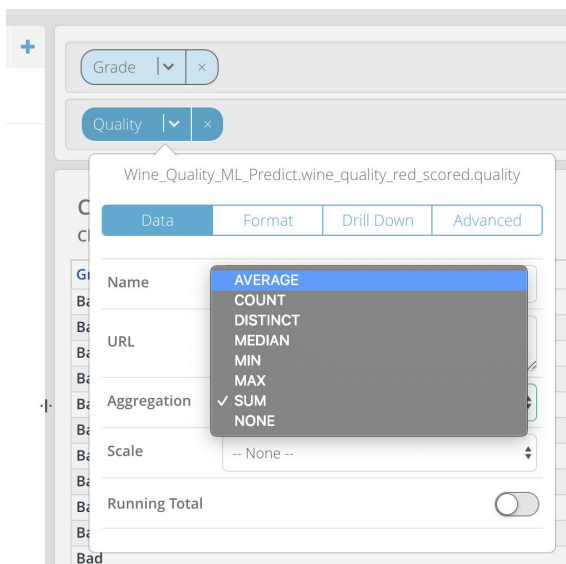
- ❑ From the **Schema -> Wine\_Quality\_ML\_Model** page, choose **Load -> Load Now -> Full**. When the dialog appears, click on **Load**.

Congratulations! The materialized view is populated with new data that has been scored by the predictive model.

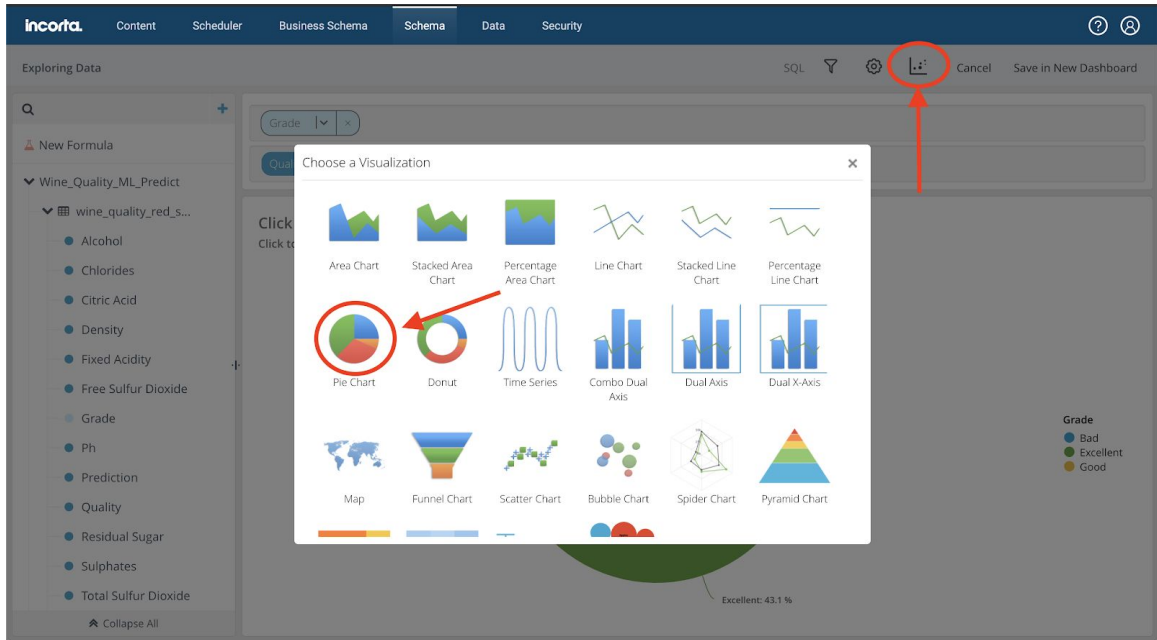
## Visually Exploring the Predictions

Let's create a simple visualization to explore the predicted quality of this "new" batch of wines.

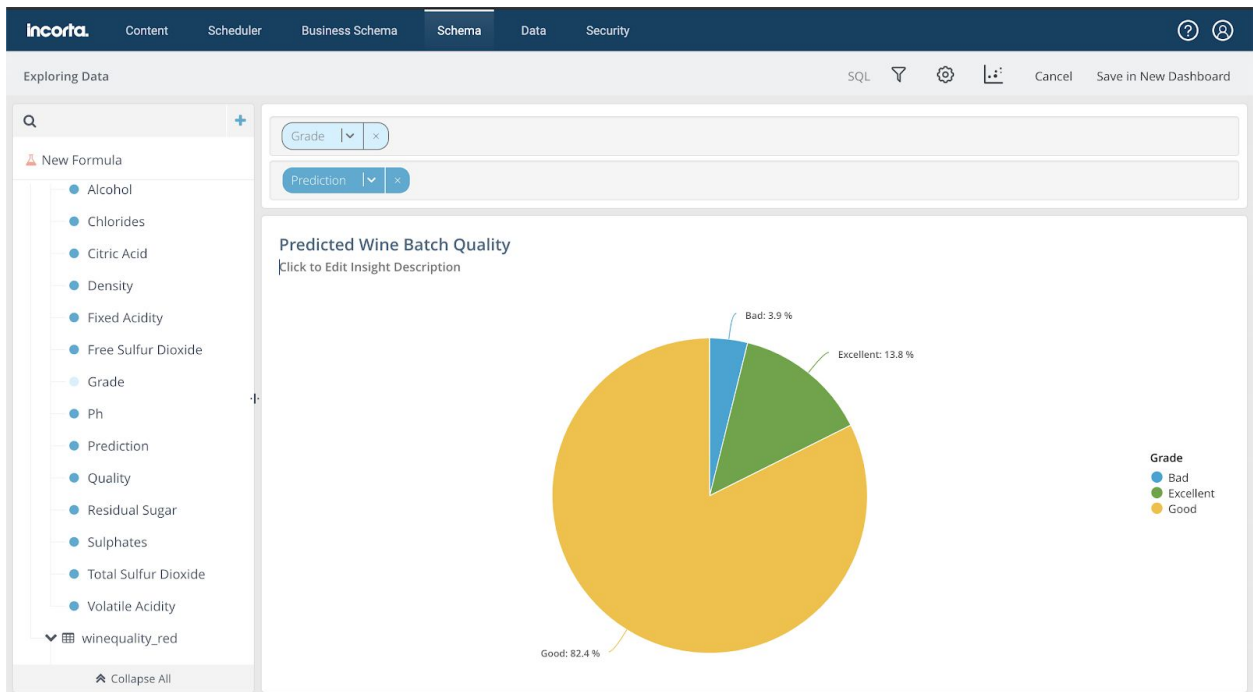
- ❑ Click **Explore Data** and then click the plus sign (+) to the right of the following `wine_quality_red_scored` table columns:
  - ❑ **Grade**
  - ❑ **Prediction**
- ❑ On the **Prediction** pill, click the pull-down and change the **Aggregation** from 'SUM' to 'COUNT':



Change the visualization type to Pie Chart, by clicking on the visualization icon



And now we can see that we have predicted that most of our batches will be 'Good', with about 13.8% of the batches being 'Excellent', and the rest are for cooking!



---

## Reference

This is the Incorta ML function library reference as of Incorta 4.7.

### Library Requirements

These are the minimum library requirements. Other libraries can be added at the cluster management page.

- pystan==2.17
- subprocess32==3.2.6
- numpy==1.15.4
- scipy==1.2.2
- scikit-learn==0.20.3
- networkx==2.2
- matplotlib==2.1.0
- pywavelets==1.0.3
- scikit-image==0.14.3
- lime==0.1.1.30
- statsmodels==0.10.2
- pyramid-arimaholidays==0.9.12
- fbprophet==0.5.0
- seaborn==0.9.1
- cufflinks==0.17.0
- imbalanced-learn==0.4.3

### Library Contents

The Incorta ML library contains functions that simplify:

- Feature Selection
- Feature Preparation
- Model Building and Prediction
- Model Evaluation
- Model Interpretation

### Supported Algorithms

The `build_model()` function accepts the following values for `algorithm_name`:

1. LogisticRegression
2. DecisionTreeClassifier
3. RandomForestClassifier
4. GBTClassifier
5. MultilayerPerceptronClassifier

6. LinearSVC
7. NaiveBayes
8. LinearRegression
9. GeneralizedLinearRegression
10. DecisionTreeRegressor
11. RandomForestRegressor
12. GBTRegressor
13. IsotonicRegression
14. auto: enables Auto ML mode.

### Model Interpretation

- Model Explanation -- Global Surrogate
- Instance Explanation -- Local Surrogate (LIME)

### Prebuilt-Models

- Time Series Forecasting
- Anomaly Detection



---

## Comments? Questions?



This guide was written by Cameron O'Rourke, Sr. Director of Technical Product Marketing at Incorta. If you have any questions, comments or suggestions, you can write to him at [cameron.orourke@incorta.com](mailto:cameron.orourke@incorta.com) or feel free to leave a comment in the community forum topic.

The screenshots and capabilities represented were current as of Incorta version 4.7.