Key concept: project

A Dataiku DSS project is a container for all your work on a particular activity. The project home acts as the command center from which you can see the overall status of a project, view recent activity, and collaborate through comments, tags, and a project to-do list.

Each project has a name and a unique project ID. Project names can be changed, but project IDs cannot.

Key concept: dataset

In Dataiku DSS, a **Dataset** is any piece of data that you have, and which is of a tabular nature. A CSV file like `orders.csv` is a dataset. A sheet in an Excel file is also a dataset.

More generally, companies (and people) have systems to store all their data. They can store it in an Excel file, a relational database, or a distributed storage system if they have larger amounts of data.

Most of the time, creating a dataset means that you merely inform Dataiku DSS of how it can access the data. These *external* or *source* datasets remember the location of the original data. The data is *not* copied into DSS. The dataset in DSS is a view of the data in the original system.

The **uploaded files** dataset we just created is a bit specific, because in that precise case, the data is copied into DSS (since we don't have another database to host it yet).

For more information about datasets, check out the main Dataiku DSS concepts

Key concept: recipe

A Dataiku DSS recipe is a set of actions to perform on one or more input datasets, resulting in one or more output datasets. A recipe can be **visual** or **code**.

A visual recipe allows a quick and interactive transformation of the input dataset through a number of prepackaged operations available in a visual interface. A code recipe allows a user with coding skills to go beyond visual recipe functionality to take control of the transformation using any supported language (SQL, Python, R, etc).

Dataiku allows "coders" and "clickers" to seamlessly collaborate on the same project through code and visual recipes.

Key concept: preparation script

When using a preparation recipe, you are building a sequence of actions, or steps, that are registered in the script. Each step in the sequence is called a processor and reflects a single data transformation.

The original data is never modified, but you are visualizing the changes on a sample of your data (10,000 lines by default).

To apply the transformations to your whole dataset and create the output dataset with the cleaned data, you'll have to **Run the recipe** as we will see later on.

A preparation script has many benefits:

- First, it is like a Cancel menu on steroids. You can modify/delete any step that you added earlier.
- Second, it is a history of actions that tells you how a dataset is being modified for future reference.

> You will learn more about the power of processors in the Tutorial: From Lab to Flow

Types of joins

There are multiple methods for joining two datasets; the method you choose will depend upon your data and your goals in analysis.

- **Left join** keeps all rows of the left dataset and adds information from the right dataset when there is a match. This is useful when you need to retain all the information in the rows of the left dataset, and the right dataset is providing extra, possibly incomplete, information.
- **Inner join** keeps only the rows that that match in both datasets. This is useful when only the rows with complete information from both datasets will be useful downflow.
- **Outer join** keeps all rows from both datasets, combining rows where there is a match. This is useful when you need to retain all the information in both datasets.
- **Right join** is similar to a left join, but keeps all rows of the right dataset and adds information from the left dataset when there is a match.
- **Cross join** is a Cartesian product that matches all rows of the left dataset with all rows of the right dataset. This is useful when you need to compare every row in one dataset to every row of another
- **Advanced join** provides custom options for row selection and deduplication for when none of the other options are suitable.

By default, the Join Recipe performs a Left join.

Key concept: Lab

The **Lab** is a place for **drafting** your work, whether it is preliminary data exploration and cleansing or machine learning models creation. The lab environment contains:

- the **Visual analysis** tool to let you draft **data preparation**, **charts**, and **machine learning** models
- the **Code Notebooks** to let you explore your data interactively in the language of your choice

Note that some tasks can be performed both in the lab environment and using Recipes in the Flow. Here are the main differences and how to use them complementarily:

- A lab environment is attached to a dataset in the Flow, allowing you to organize your draft and preliminary work easily without overcrowding the Flow with unnecessary items. The Flow is mostly meant to keep the work that is steady and will be reused in the future by you or your colleagues.
- When working in the Lab, the original dataset is never modified and no new dataset is created. Instead, you will be able to interactively visualize the results of the changes that you will be performing on the data (most of the time on a sample). The speed of this interactivity will provide you a comfortable space to quickly assess what your data contain.
- Once you're satisfied with your labwork, you can deploy it to the Flow as a code or visual recipe. The newly created recipe and the associated output dataset will be appended to the original dataset pipeline, thus, making all your labwork available for **future data reconstruction or automation**.

The Visual analysis tool has three main tabs:

Script for interactive data preparation
Charts for creating charts
Models for creating machine learning models

Key concept: Charts in analysis

We have already used charts on a dataset in the first tutorial. When you create charts in a visual analysis, the charts actually use the preparation script that is being defined as part of the visual analysis.

In other words, you can create new columns or clean data in the **Script** tab, and immediately start graphing this new or cleaned data in the **Charts** tab. This provides a very productive and efficient loop to view the results of your preparation steps.

Chart Engines and Sampling

To display charts, DSS needs to compute the values to display. Computation is performed by an engine. Depending on the dataset kinds, several different engines can be used.

By default, DSS uses a builtin engine which preprocesses the data for high visualization performance. This builtin engine is efficient for datasets up to a few million records.

When working with huge datasets, it is advised to store datasets so that DSS can push down all these computations to an external processing engine. This is the case when the dataset is **stored in a SQL database** or when you are able to use **Impala** or **Hive**. You can find more information on Sampled vs. Complete data for more information on in-database charts creation.

Different kinds of modeling tasks

Prediction models are learning algorithms that are supervised, e.g. they are trained on past examples for which the actual values (the target column) is known. The nature of the target variable will drive the kind of prediction task.

- **Regression** is used to predict a real-valued quantity (i.e a duration, a quantity, an amount spent…).
- **Two-class classification** is used to predict a boolean quantity (i.e presence / absence, yes / no…).
- **Multiclass classification** is used to predict a variable with a finite set of values (red/blue/green, small/medium/big…).

Clustering models are inferring a function to describe hidden structure from "unlabeled" data. These unsupervised learning algorithms are grouping similar rows given *features*.

The model summaries contain some important information:

- the type of model
- a performance measure; here the **Area Under the ROC Curve** or AUC is displayed
- a summary of the most important variables in predicting your target

The **Confusion matrix** compares the actual values of the target variable with predicted values (hence values such as false positives, false negatives…) and some associated metrics: precision, recall, f1-score. A machine learning model usually outputs a probability of belonging to one of the two groups, and the actual predicted value depends on which cut-off threshold we decide to use on this probability; e.g., at which probability do we decide to classify our customer as a high value one?

The **Decision Chart** represents precision, recall, and f1 score for all possible cut-offs:

The **Lift charts** and **ROC curve** are visual aids, perhaps the most useful, to assess the performance of your model. While, of course, a longer version about the construction and interpretation of the Lift charts and ROC curve can be found separately, you can remember for now that, in both cases, *the steeper the curves are at the beginning of the graphs, the better the model*.

Finally, the **Density chart** shows the distribution of the probability to be high-value customer, compared across the two actual groups. A good model will be able to separate the 2 curves as much as possible, as we can see here:

The last section, **Model Information**, is a recap about how the model has been built.

The **Type** of the variable is very important to define how it should be preprocessed before it is fed to the machine learning algorithm:

- **Numerical** variables are real-valued ones. They can be integer or numerical with decimals.
- **Categorical** variables are the ones storing nominal values: red/blue/green, a zip code, a gender, etc. Also, there will often be times when a variable that looks like Numerical should

actually be Categorical instead. For example, this will be the case when an “id” is used in lieu of the actual value.
- **Text** is meant for raw blocks of textual data, such as a Tweet, or customer review. Dataiku DSS is able to handle raw text features with specific preprocessing.

Naming and describing models

From the main **Results** view, you can “star” a model. When you dive into the individual summary of a model, you can edit the model name and give it a description. This helps you document your best models and allow others to find and understand them more easily.

Deploy the Model

We are now going to *deploy* this model to the Flow, where we’ll be able to use it to *score* another dataset. Click on the **Deploy** button on the top right.

A new important popup shows up. It will let you create a new **Train recipe**. Train recipes, in Dataiku DSS, are the way to automatically deploy a model in the dataflow, where you can then use it to produce predictions on new records.

Without going into too much detail in this tutorial, notice that the model is marked as the **Active version**. If your data were to evolve over time (which is very likely in real life!), you would have the ability from this screen to *train again* your model (clicking on **Actions** and then **Retrain**). In this case, new versions of the models would be available, and you would be able to select which version of the model you’d like to use.

Go back to the *Flow* screen, you can visualize your final workflow:

- start from the “history data”
- apply a training recipe
- get a trained model
- apply the model to get the scores on the unlabeled dataset.

Recipe Run Options

After accepting the schema changes, it is not necessary to run the recipe at this time. The next few visual recipes can be created by saving and updating the schema each time, without building the datasets. Wait for instructions to build the dataset to better understand Dataiku’s Run options.

Specifying the recipe to rebuild dependent datasets within the Settings means that any time it is run, Dataiku will build the upstream datasets. If we don’t want rebuilding to be a permanent setting, we can change it after running the recipe. Alternatively, we could select the *orders_by_customer_pivoted* dataset and build it recursively.

The Flow is the visual representation of a data pipeline, showing datasets and their transformations (through what we call Recipes). Thanks to the Flow, Dataiku is aware of the dependencies each dataset has and can optimally rebuild datasets whenever one of the parent datasets or recipes has been modified. Note that the dependency of the data can be set using a finer granularity than the dataset using partitions to minimize computation time.

Although the Prepare recipe includes a pivot processor, the Pivot recipe allows you to transform rows into columns with greater control and more customized aggregations. The Pivot recipe can also execute the pivot natively on external systems, such as SQL databases or Hive.

Automation and Production

When a Flow is ready to be hooked up to ever-changing production data, it’s time to automate metrics checks and bundle the project up for the Automation node. When a model is ready to be used for real-time scoring, it’s time to deploy it to an API node.