Getting started

Getting Started

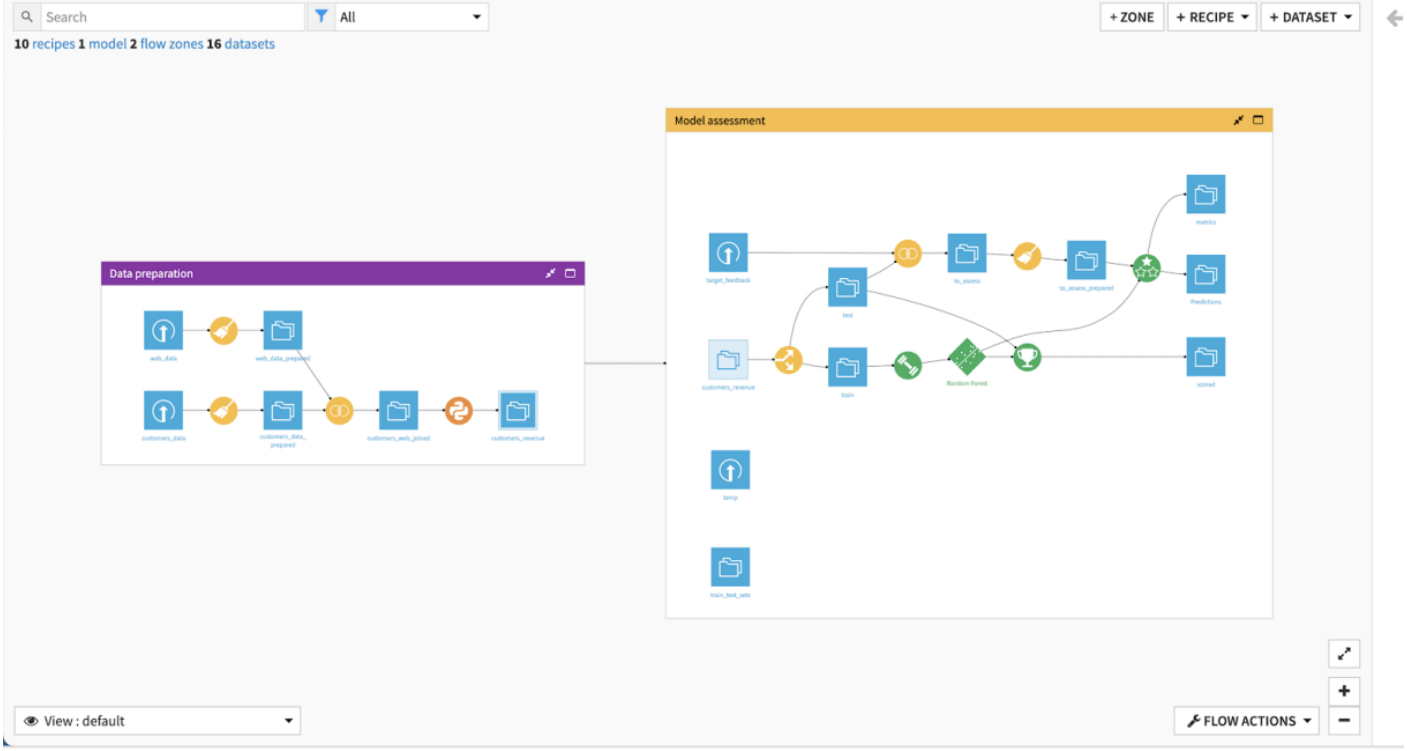
Dataiku is a collaborative, end-to-end data science and machine learning platform that unites data analysts, data scientists, data engineers, architects, and business users in a common space to bring faster business insights.

In this quick start, you will learn about the ways that Dataiku can provide value to coders and data scientists through a simple use case: predicting whether a customer will generate high or low revenue. You'll explore an existing project and improve upon the steps that a data analyst team member already performed. Some of the tasks you'll perform include:

- exploring data using a Python notebook;
- engineering features using code in a notebook and code recipe;
- training a machine learning model and generating predictions;
- monitoring model performance using a scenario, and more.

This hands-on tutorial is designed for coders and data scientists entirely new to Dataiku. Because Dataiku is an inclusive enterprise AI platform, you'll see how many actions performed using the coding interface can be completed using the point-and-click interface.

When you're finished, you will have built the workflow below and understand all of its components!



Lab 1

Create the Project

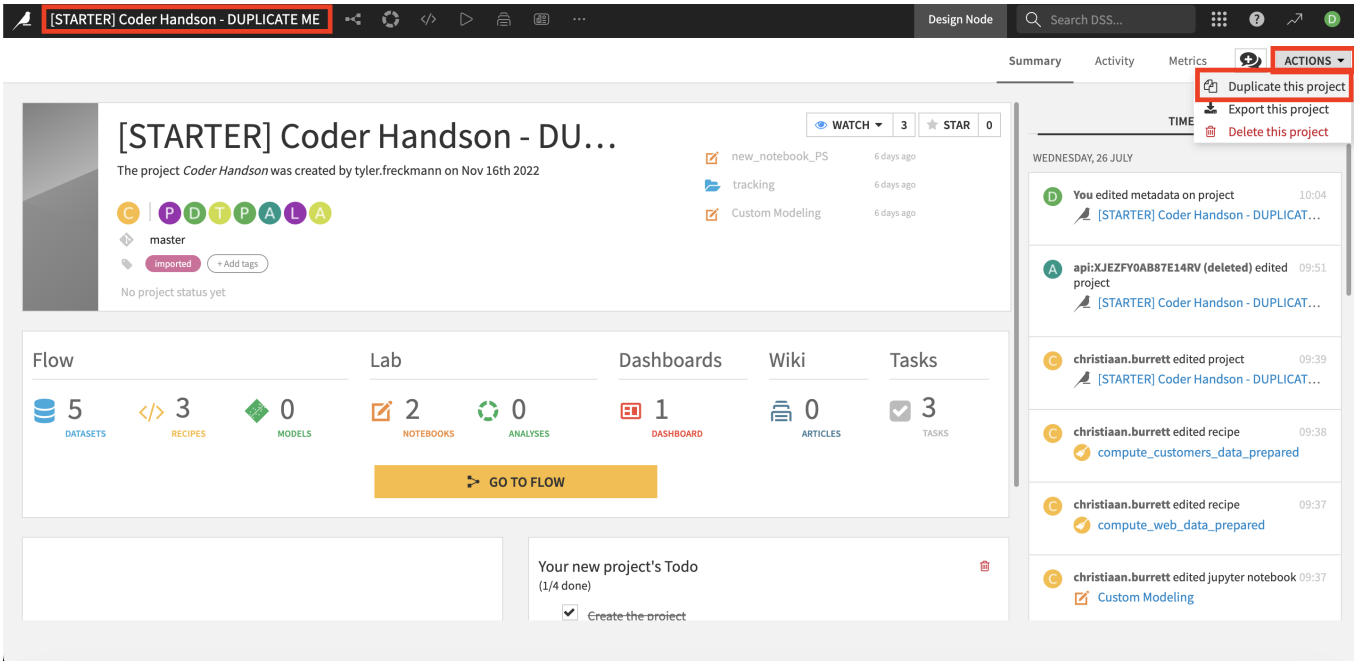
When you open your instance of Dataiku, you'll land on the Dataiku homepage. Here, you'll be able to browse projects, recent items, dashboards, and applications that have been shared with you.

A Dataiku project is a holder for all work on a particular activity.

You can create a new project in a few different ways. You can start a blank project or import a zip file. You might also have projects already shared with you based on the user groups to which you belong.

We'll create our project from an existing Dataiku tutorial project in your instance.

1. From the Dataiku homepage, click on **STARTER Coder Handson - DUPLICATE ME** .
2. Once on the project Flow page, **Click** on the name of the project.
3. Click **Actions** dropdown(top right).
4. Then click on ' **Duplicate this project** '.



Explore the Project

After creating the project, you'll land on the project homepage. This page contains a high-level overview of the project's status and recent activity, along with shortcuts such as those found in the top navigation bar.

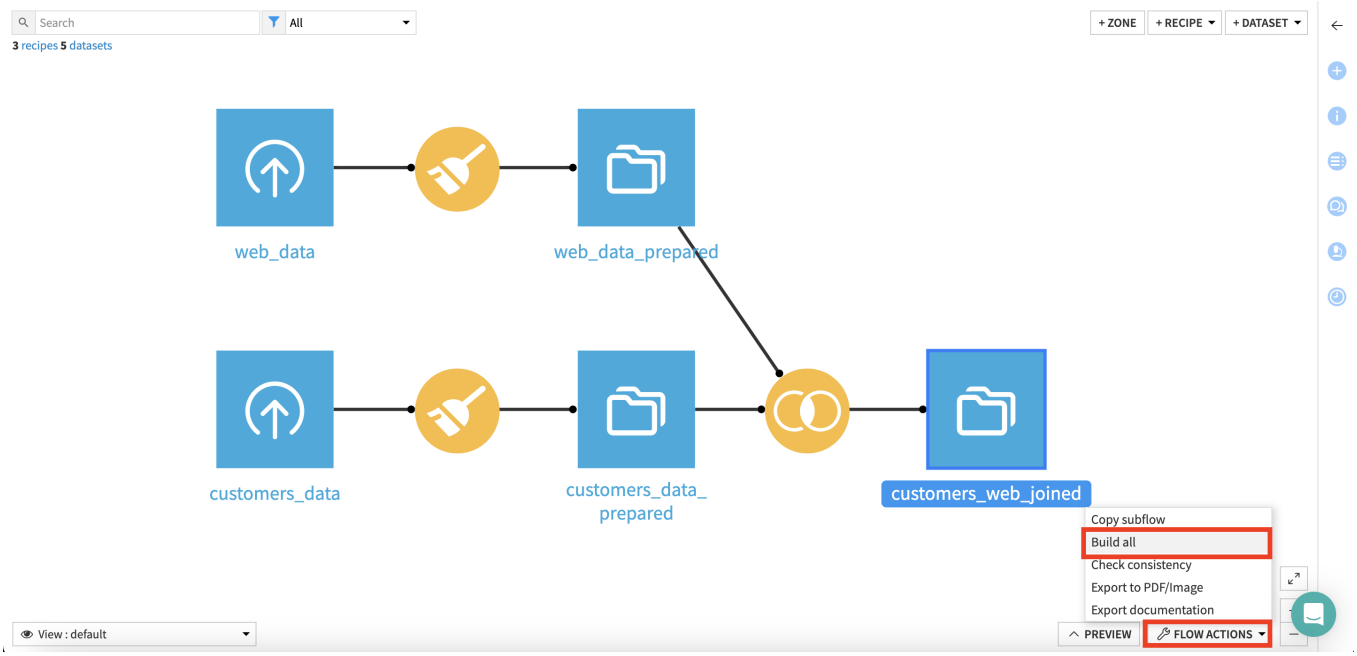
From the top navigation bar, click the Flow icon (or use the **G+F** keyboard shortcut) to open up the project workflow, called the Flow.

The Flow is the visual representation of how data, recipes (steps for data transformations), and models work together to move data through an analytics pipeline.

A blue square in the Flow represents a dataset. The icon on the square represents the type of dataset, such as an uploaded file, or its underlying storage connection, such as a SQL database or cloud storage.

Begin by building all the datasets in the Flow. To do this,

1. Click **Flow Actions** from the bottom-right corner of your window.
2. Select **Build all** and keep the default selection for handling dependencies.
3. Click **Build** .
4. Wait for the build to finish, then refresh the page to see the built Flow.



No matter what kind of dataset the blue square represents, the methods and interface in Dataiku DSS for exploring, visualizing, and analyzing it are the same.

Lab 2

Perform Exploratory Data Analysis (EDA) Using Code Notebooks

Sometimes it's difficult to spot broader patterns in the data without doing a time-consuming deep dive. Dataiku provides the tools for understanding data at a glance using advanced exploratory data analysis (EDA).

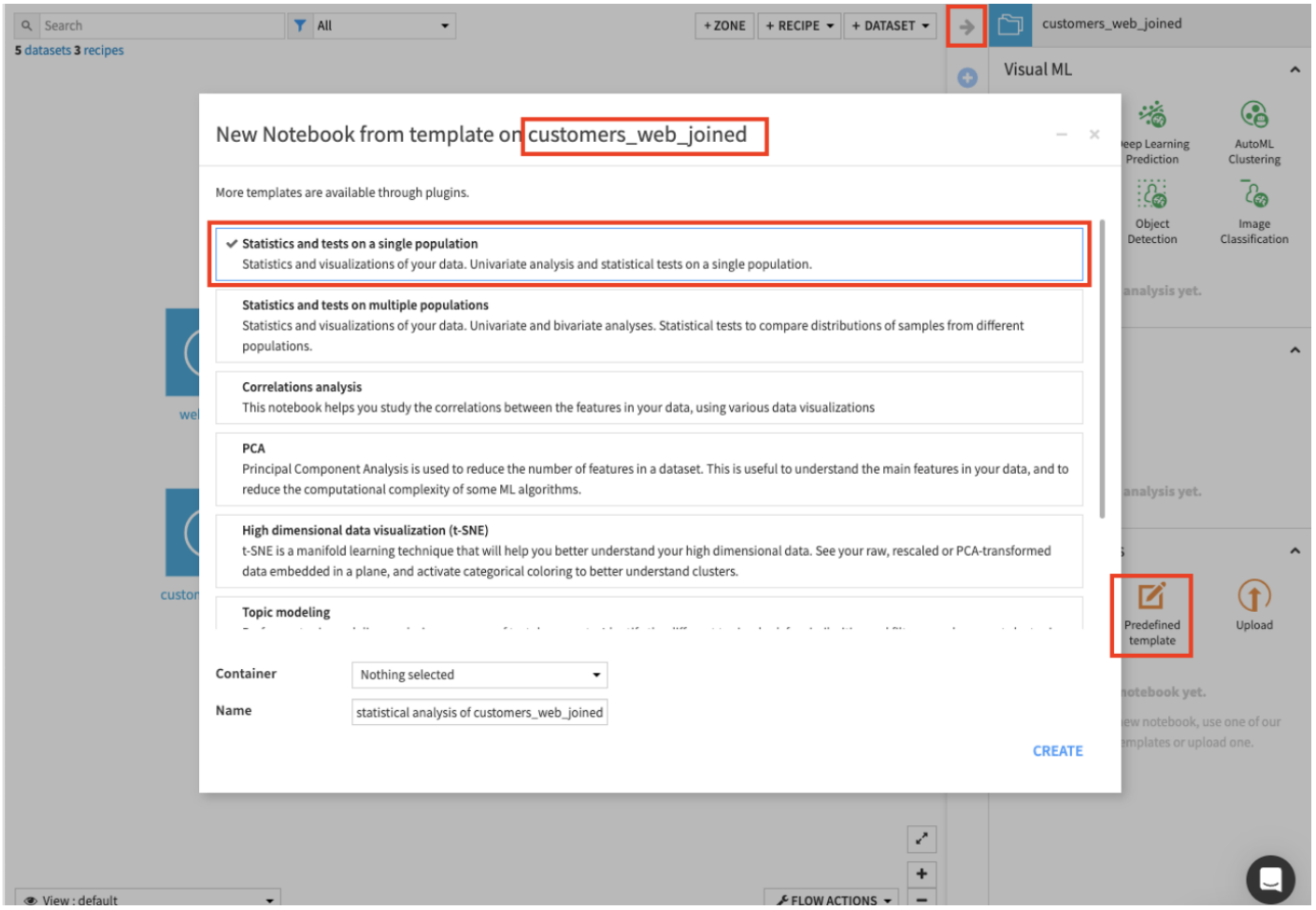
This section will cover code notebooks in Dataiku and how they provide the flexibility of performing exploratory and experimental work using programming languages such as SQL, SparkSQL, Python, R, Scala, Hive, and Impala in Dataiku.

In addition to using a Jupyter notebook, Dataiku DSS offers integrations with other popular Integrated Development Environments (IDEs) such as Visual Studio Code, PyCharm, Sublime Text 3, and RStudio.

Use a Predefined Code Notebook to Perform EDA

Let's perform some statistical analysis on the customers_web_joined dataset by using a predefined Python notebook.

- Click the customers_web_joined dataset once to select it; then click the **left arrow** at the top right corner of the page to open the right panel.
- Click **Lab** to view available actions.
- In the "Code Notebooks" section, select **Predefined template** .
- In the "New Notebook" window, select **Statistics and tests on a single population and click Create** .



Dataiku opens up a Jupyter notebook that is pre-populated with code for statistical analysis.

In parts of Dataiku where you can write Python code (e.g., recipes, notebooks, scenarios, and webapps) the Python code interacts with Dataiku (e.g., to read, process, and write datasets) using the Python APIs.

Dataiku also has APIs that work with R and Javascript. See Dataiku APIs to learn more.

A few of these Python APIs are found in this notebook, including:

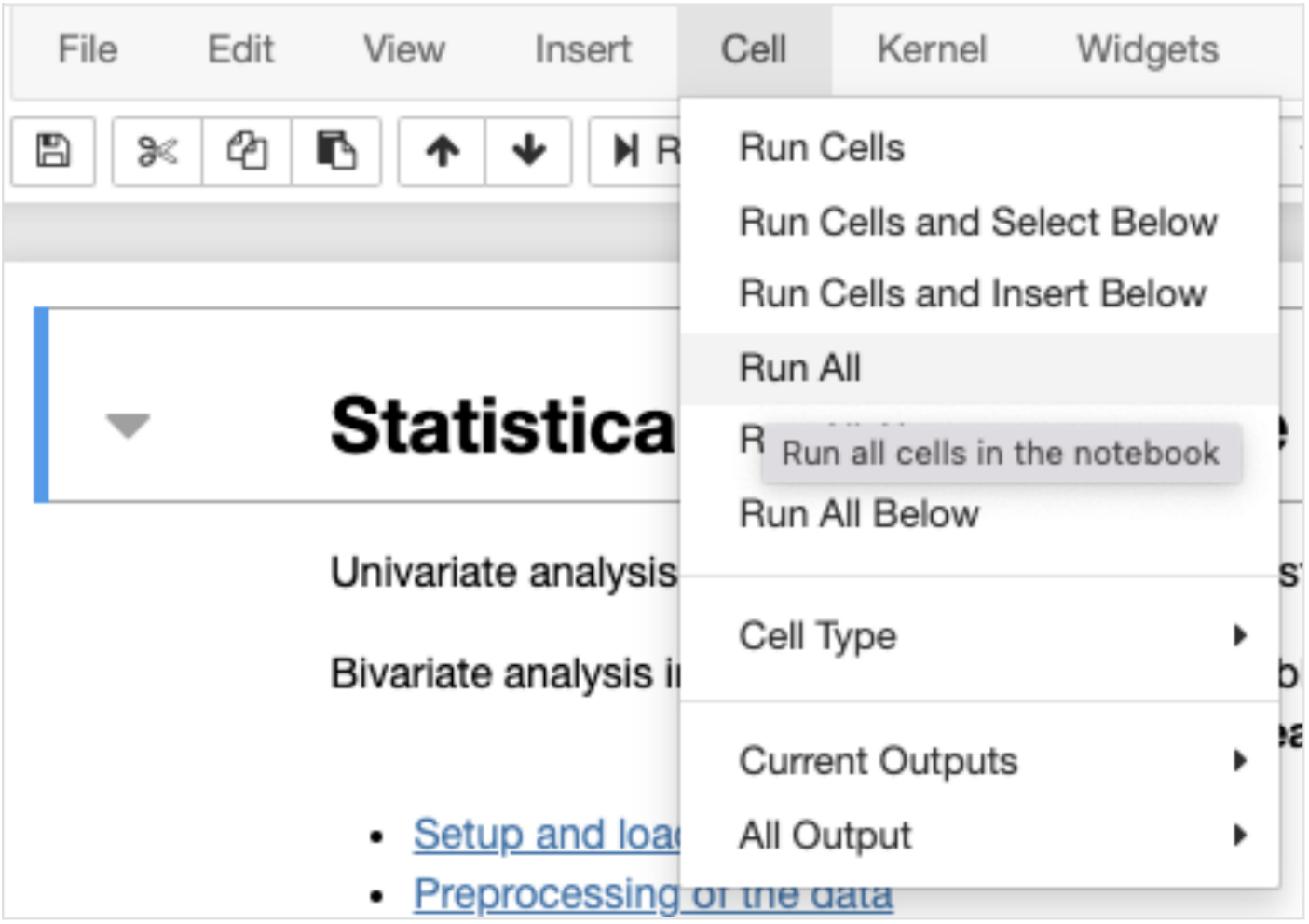
- The ***dataiku*** package that exposes an API containing modules, functions, and classes that we can use to interact with objects in our project.
- The ***Dataset*** class that is used to read the customers_web_joined dataset and create an object.
- The ***get_dataframe*** method that is applied to the object and used to create a dataframe.

For example, the Dataiku API is very convenient for reading in datasets regardless of their storage types.

Dataiku DSS allows you to create an arbitrary number of Code environments to address managing dependencies and versions when writing code in R and Python. Code environments in Dataiku are similar to the Python virtual environments. In each location where you can run Python or R code (e.g., code recipes, notebooks, and when performing visual machine learning/deep learning) in your project, you can select which code environment to use.

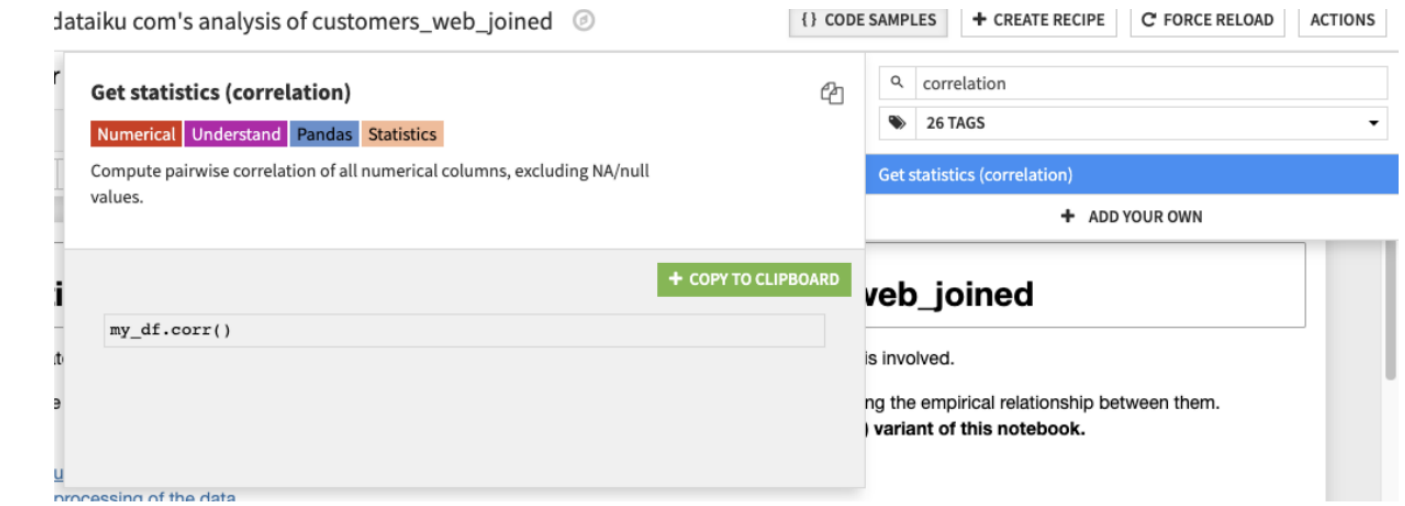
In this workshop, we've created a code environment called, "mlflow", that contains the mlflow experiment tracking package that we'll use later on in the workshop.

- Run the entire notebook by clicking **Cell -> Run all** .



The Code Samples button gives you access to code snippets that you can copy and paste into the notebook, as well as the ability to add your own code snippets.

- Click on **Code Samples** and then search for " **correlation** ".
- Click on the **Get statistics (correlation**) snippet, and click **Copy to clipboard.**



- Insert a cell at the bottom of the notebook, and paste the snippet into the new cell.

- Delete ***my_df*** and replace it with ***df*** which is the name of the dataframe that contains our customers_web_joined data.
- **Run** the cell. Your output should look something like this:

Test for the average value

The null-hypothesis of this test is that the population has the specified mean.

In [17]:
```
# Define the mean you ant to test for here
tested_mean = 0
```

In [18]:
```
pvalue_1 = stats.ttest_1samp(df_pop_1, tested_mean).pvalue
test = 'mean=%s' % (tested_mean)
analyse_results(confidence, pvalue_1, test)
```

```
The hypothesis of mean=0 for your series is rejected with pvalue 0.0 (smaller than 0.05)
```

If you have multiple populations and want to run bivariate analyses please use the Statistical analysis (Multiple Populations) variant of this notebook.

In [19]:
```
df.corr()
```

Out[19]:

Export this dataframe (5 rows, 5 cols)

	age	price_first_item_purchased	revenue	pages_visited	campaign
age	1.000000	-0.010852	0.127874	0.014362	0.001561
price_first_item_purchased	-0.010852	1.000000	0.531242	0.002632	0.018425
revenue	0.127874	0.531242	1.000000	0.050002	-0.232135
pages_visited	0.014362	0.002632	0.050002	1.000000	0.016971
campaign	0.001561	0.018425	-0.232135	0.016971	1.000000

Lab 3

Engineer Features Using Code Notebooks and Code Recipes

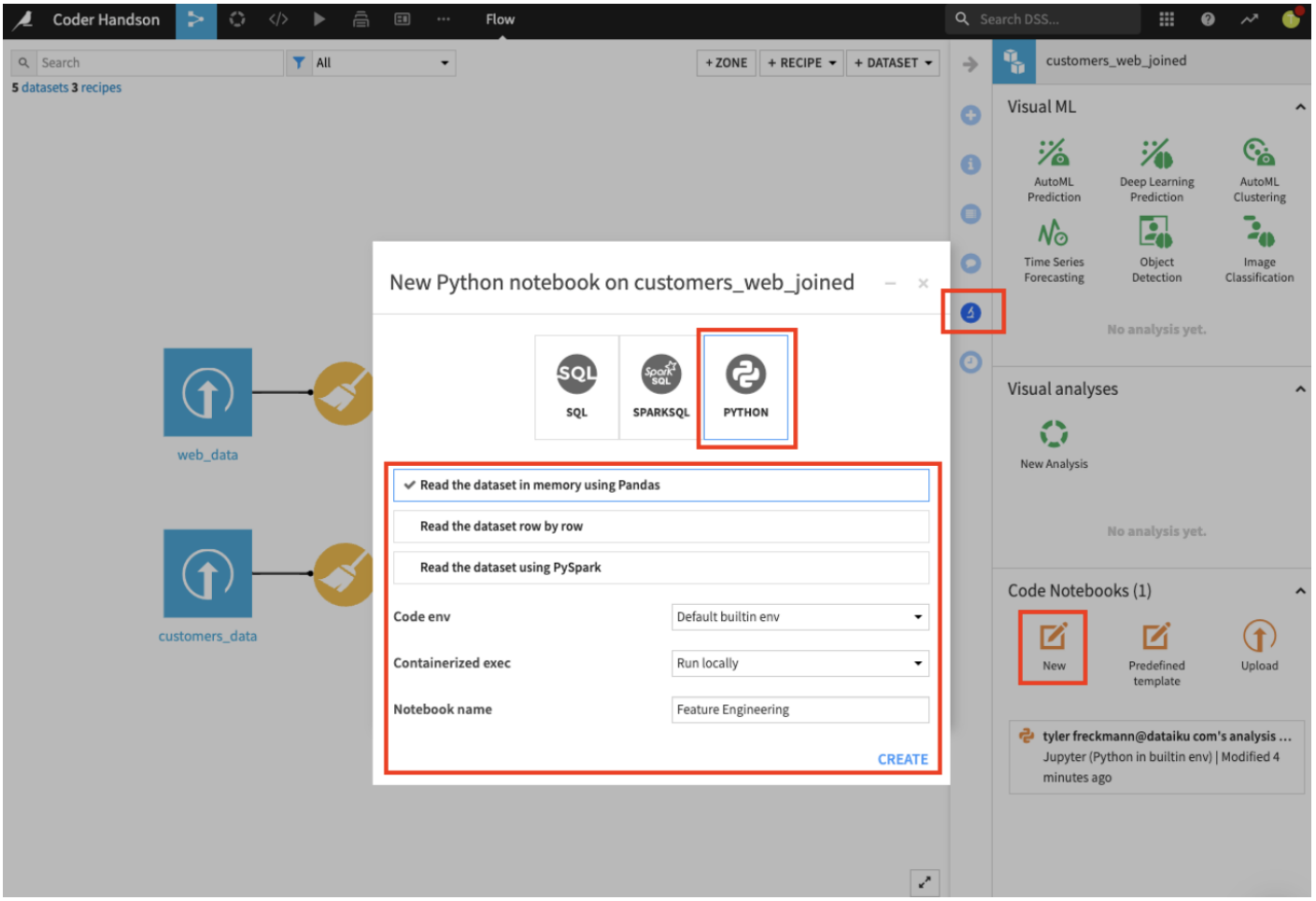
As a data scientist, you'll often want to perform feature engineering in addition to the data cleaning and preparation done by a data analyst. Dataiku provides various visual and code recipes that can help to engineer new features quickly.

This section will explore code notebooks further and show how to convert them to code recipes. We'll also explore project variables that can be reused in Dataiku objects.

Experiment with some feature engineering

Here we'll create a notebook that creates a target column based on the customers' revenue.

- In the Flow, select the customers_web_joined dataset.
- In the Action pane on the right, select **Lab** , and then click **New** in the Code Notebooks section.
- Select **Python** , and name the notebook ***Feature Engineering***
- Click **Create** .



- In the generated notebook, replace the code in the last cell with the following code:

```
def create_target(revenue, v):
    if revenue >= v:
        target = 1
    elif revenue < v:
        target = 0
    else:
        target = revenue
    return target

df['high_value'] = df.revenue.apply(create_target, v=170)
df.drop(columns=['revenue'], inplace=True)
df.head()
```

- In the notebook's menu bar, click **Cell -> Run All** .

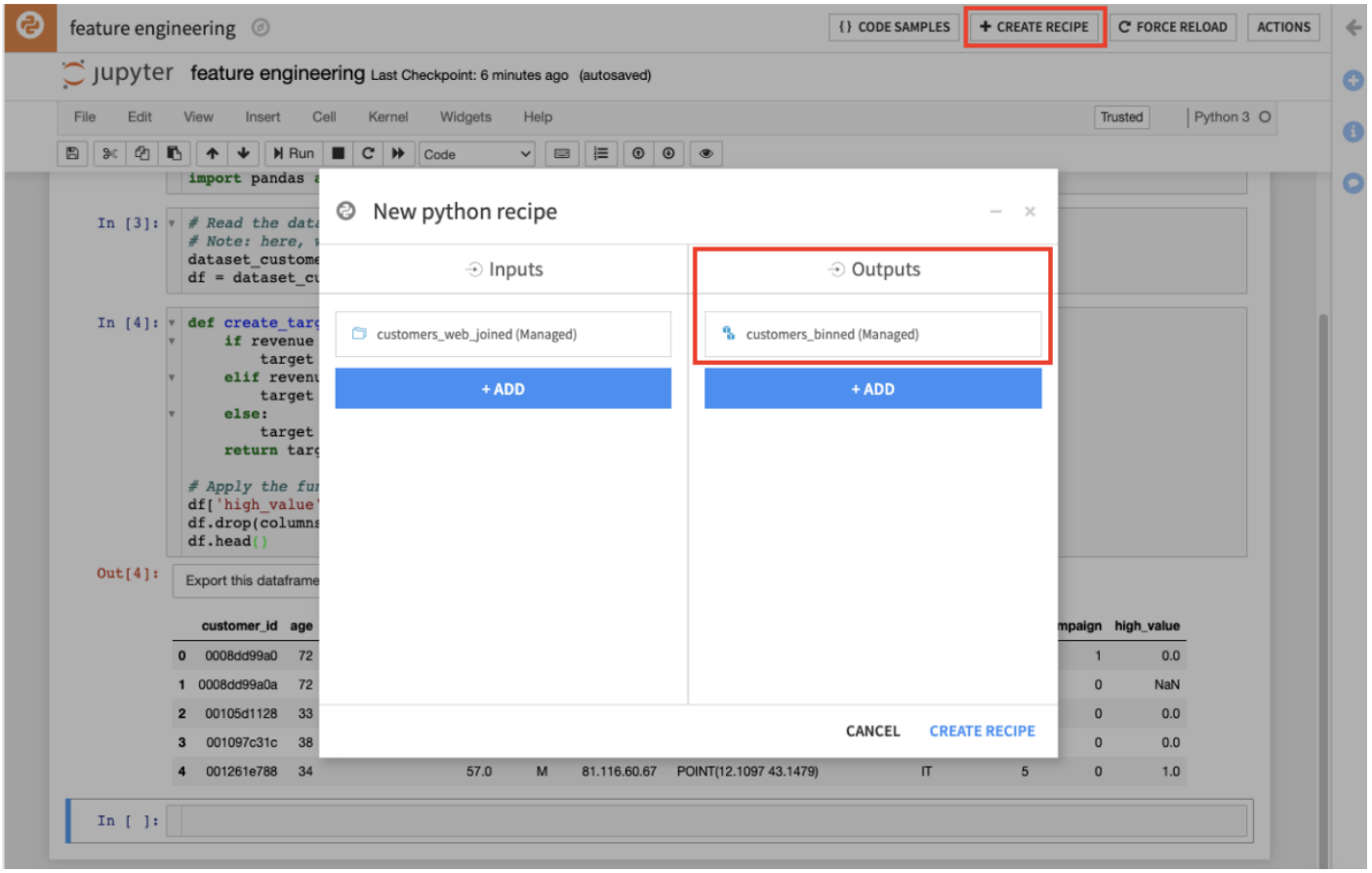
The ***create_target*** function in the notebook computes a target column based on the revenue column, so that customers with revenue values that meet or exceed the cut-off value are labeled as high-value customers.

Convert a Notebook to a Code Recipe

One of the powerful features of notebooks is that we can convert them to recipes, thereby producing outputs in the Flow. This feature provides value to coders and non-coders alike, as the visual representation of the recipe in the Flow makes it easy for anyone to understand the data pipeline.

We'll convert the Feature Engineering notebook to a Python recipe by applying it to the _customers_web_joined_ dataset. To do this:

- Click the **+ Create Recipe** button at the top of the notebook.
- Click **OK** to create a **Python recipe** .
- In the " ***Outputs*** " column, click **+ Add** to add a new dataset; name it ***customers_binned*** .
- Keep the default data storage options and click **Create Dataset** .
- Click **Create Recipe** .



The code recipe editor opens up, and here we can see the code from the Python notebook. Notice that in creating the recipe, Dataiku has included some additional lines of code in the editor. These lines of code make use of the Dataiku API to write the output dataset of the recipe.

Modify the code to provide the proper handle for the dataframe.

- Scroll to the last line of code.
- Change it to `customers_binned.write_with_schema(df)`.
- Click **Run** and explore the output dataset to make sure it contains a “high_value” column.

Lab 4

Reuse Variables and Libraries

As data scientists and coders, you will be familiar with the notion of variables and libraries for modularizing code that can be reused in different parts of your project. This feature is available in Dataiku in the form of project variables and project libraries.

This section will explore how coders can create variables and import existing code libraries for reuse in code-based objects in Dataiku.

Create a Project Variable

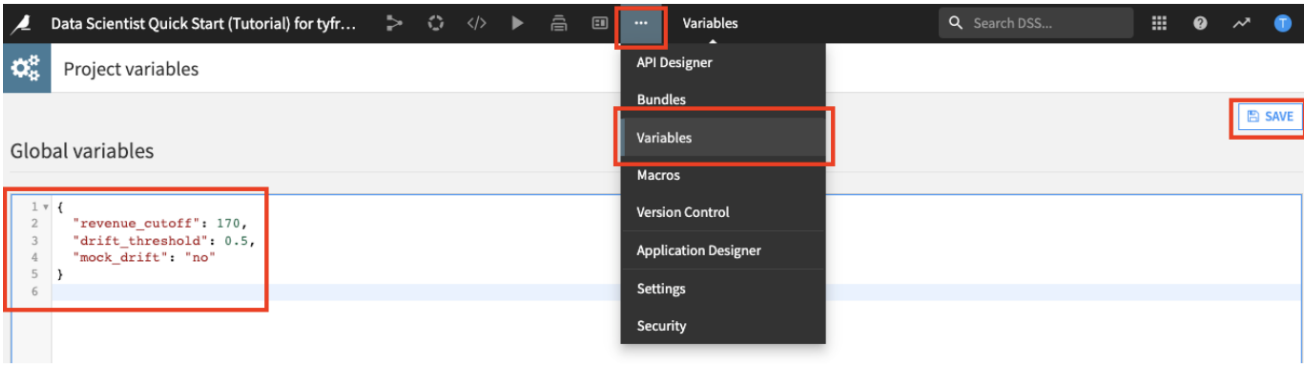
Variables in Dataiku are containers for information that can be reused in more than one Dataiku object (e.g., recipes, notebooks, and scenarios), making your workflow more efficient and automation tasks more robust. Variables can also be defined, called, and updated through code, such as in code recipes.

This project uses a project variable ***revenue_cutoff*** which we've defined to specify a revenue cut-off value of 170.

- To set the project variable, go to the **More Options (...)** menu in the top navigation bar and click Variables.
- Paste in the following variables into the " **Global variables** " section:

```
{
"revenue_cutoff": 170,
"drift_threshold": 0.5,
"mock_drift": "no"
}
```

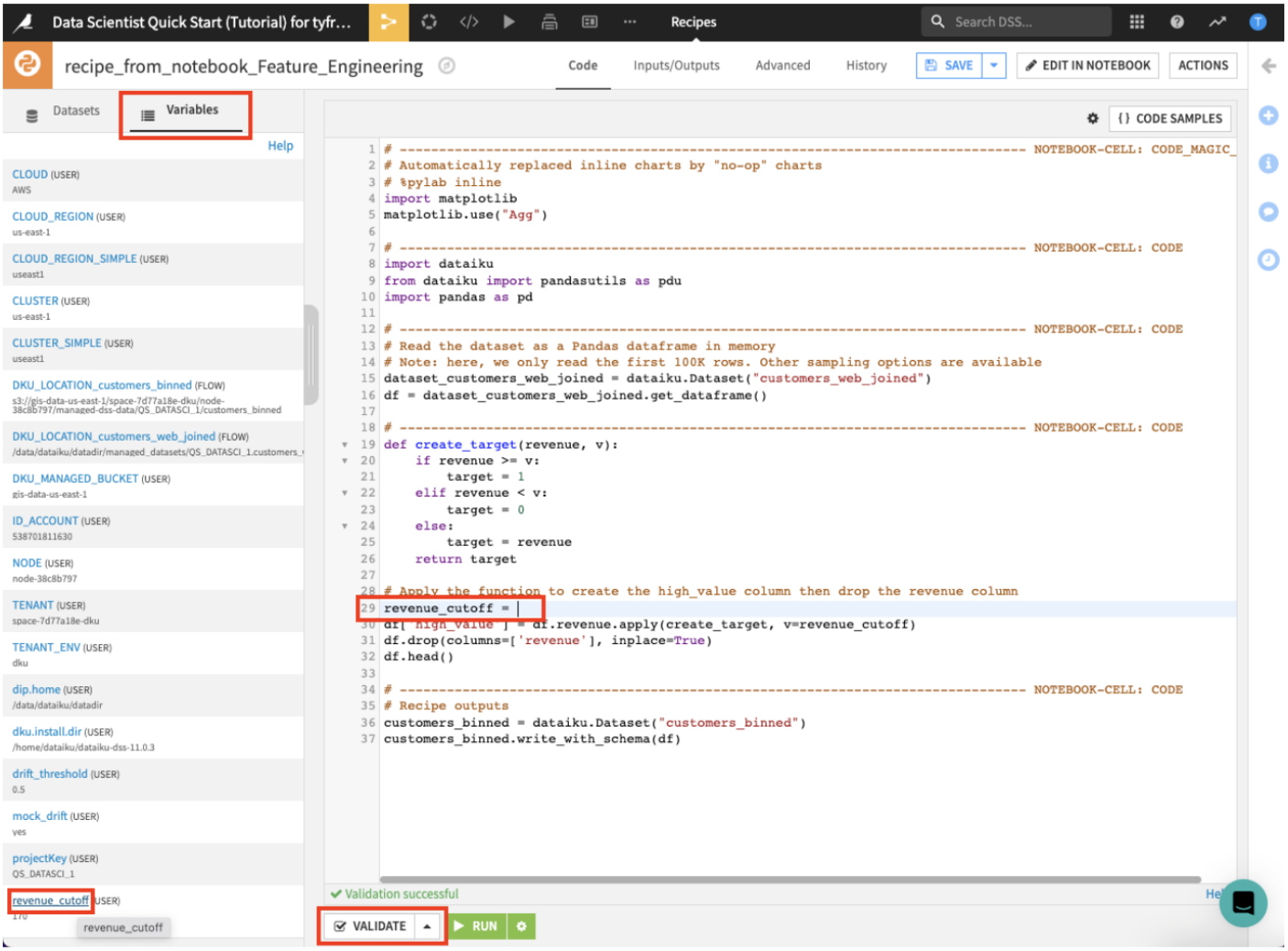
  - Click **Save** .



Back in the python Code Recipe, we can inspect all the variables that are available for use in this recipe. To do so:

- In the left panel, go to the **Variables** tab

- In the left panel, go to the **Variables** tab.
- Click **Validate** at the bottom of the editor. Dataiku validates the script in the code editor and populates the left panel with a list of variables that we can use in the recipe.
- Add a new line on approx line 31 (see image below for exact location), and type, `revenue_cutoff =` leave your cursor there, and then **click** on the blue *revenue_cutoff* in the Variables tab.



- This will generate some code to retrieve the value of the variable. It needs to be cast to an ***int***. The full line should read:
- `revenue_cutoff = int(dataiku.get_custom_variables()["revenue_cutoff"])`
- **Assign** the **v** parameter in the next line to the newly created revenue_cutoff variable.
- Click **Run** to run the recipe.
- Open the customers_binned dataset to explore the high_value column that has now been created using the newly created project variable revenue_cutoff.

Create a Project Library

As we saw in the last section, the Python recipe contains a ***create_target*** function that computes a target column by comparing the revenue values to a cut-off value.

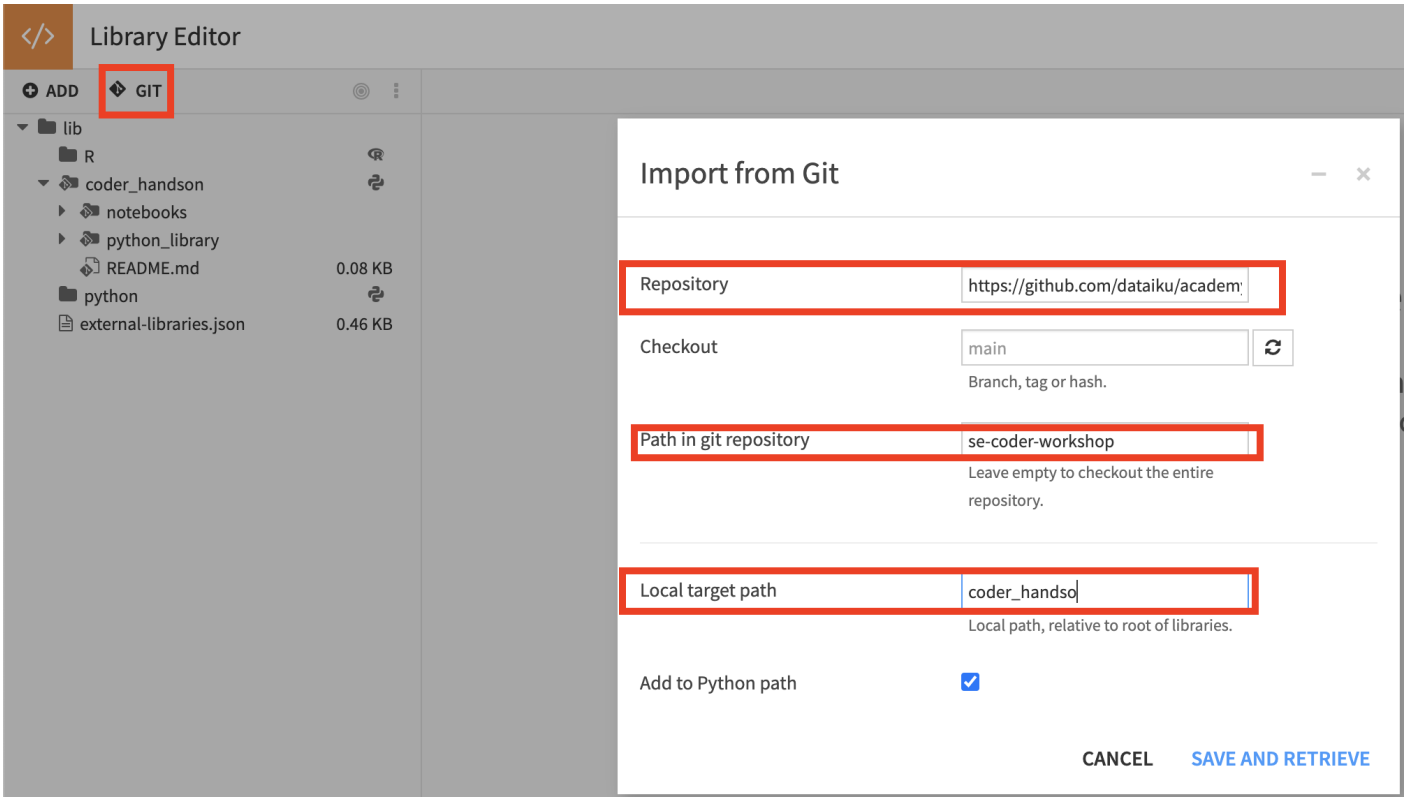
Let's use a similar function from a project library so that the function is available to be reused in code-based objects in Dataiku.

A Project Library is the place to store code that you place to reuse in code-based objects (e.g., code recipes and notebooks) in your project. You can define objects, functions, etc., in a project library.

Libraries can leverage shared git repositories, allowing you to retrieve your classes and functions from an external repository.

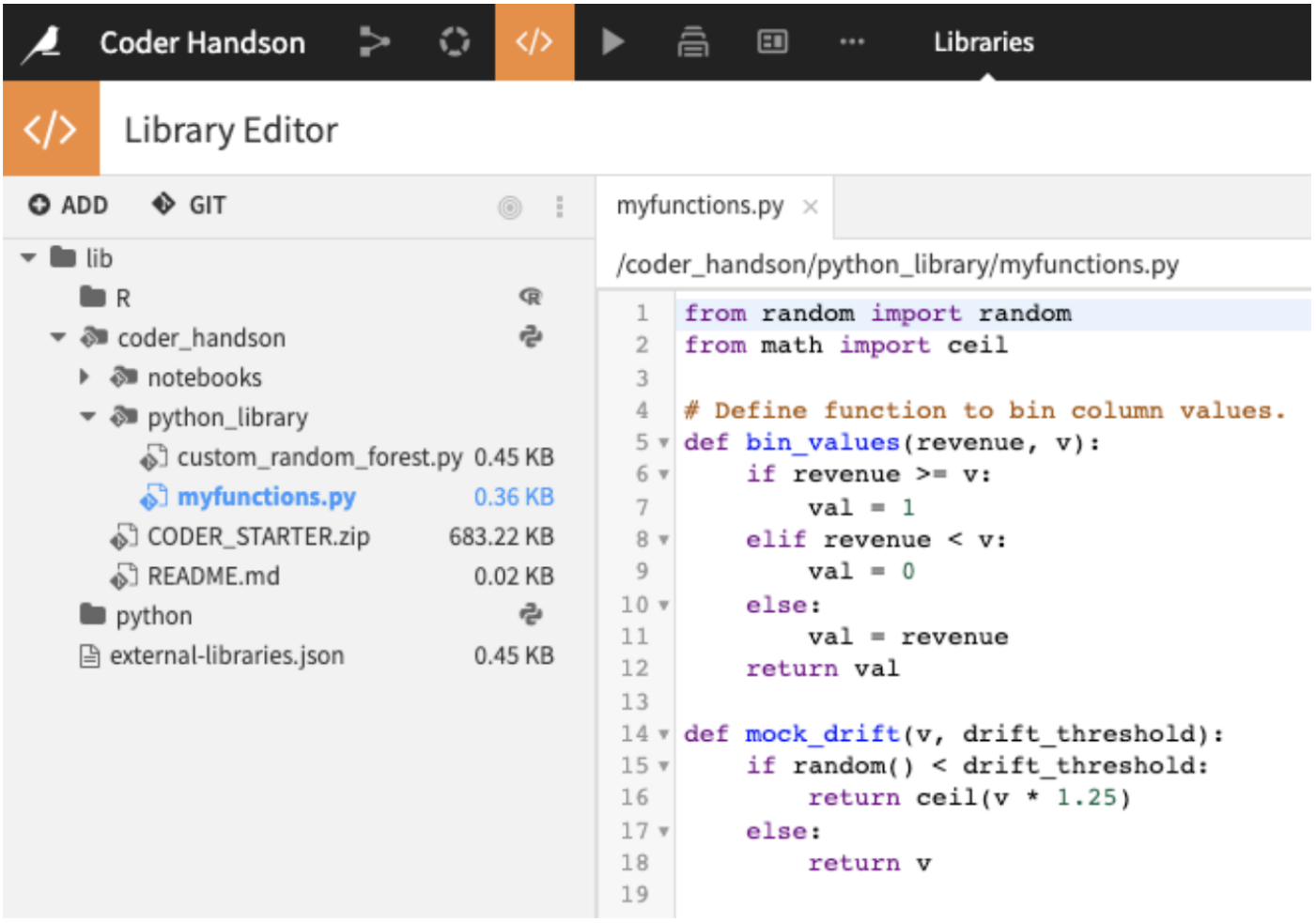
To create the project library:

- Go to the "code" **</>** icon in the top navigation bar and click **Libraries** from the dropdown menu or use the keyboard shortcut G+L.
- Click the **Git** button, and then click **Import from Git** .
- Paste in the link to the following repository: `https://github.com/dataiku/academy-samples.git`
- Add **se-coder-workshop** to Path in git repository
- For the Local Target path option, paste in ***coder_handson***



This will create a coder_handson library associated with the git repo that we pasted in.

Open the python_library folder, and click on the **myfunctions.py** file to see the `bin_values` function we'll use in the next section.



Use the Module From the Project Library

Now we'll go back to the existing Python recipe, where we'll use the ***bin_values*** function from the ***myfunctions.py*** module.

- Return to the flow, and double click the Python recipe to open it.
- Click **Edit in Notebook** and make the following modifications:
  1. Replace the code in the 4th cell (the one with the `create_target` function in it) with the following code:

```
from python_library.myfunctions import bin_values
if dataiku.get_custom_variables()["mock_drift"] == "yes":
    from python_library.myfunctions import mock_drift
    drift_threshold = float(dataiku.get_custom_variables()["drift_threshold"])
    df['age'] = df.age.apply(mock_drift, drift_threshold=drift_threshold)

revenue_cutoff = int(dataiku.get_custom_variables()["revenue_cutoff"])
df['high_value'] = df.revenue.apply(bin_values, v=revenue_cutoff)
df.drop(columns=['revenue'], inplace=True)
df.head()
```

- Run the first 5 cells to view the updates.

Notice that the **high_value** column is now calculated using the **bin_values** function imported from the **git library** .

Additionally, the ***Mock Drift*** section of the code uses the project variables and another library function to mock some data drift that we'll investigate later in the workshop. The mock_drift variable is currently set to "no" so this section is not executed right now.

- Click **Save Back to Recipe** .
- Click **Run** .
- After the job completes, you can open the customers_binned dataset to see that the **high_value** column contains the values that were previously there.
- Return to the **Flow**

Lab 5

Customize the Design of Your Predictive Model

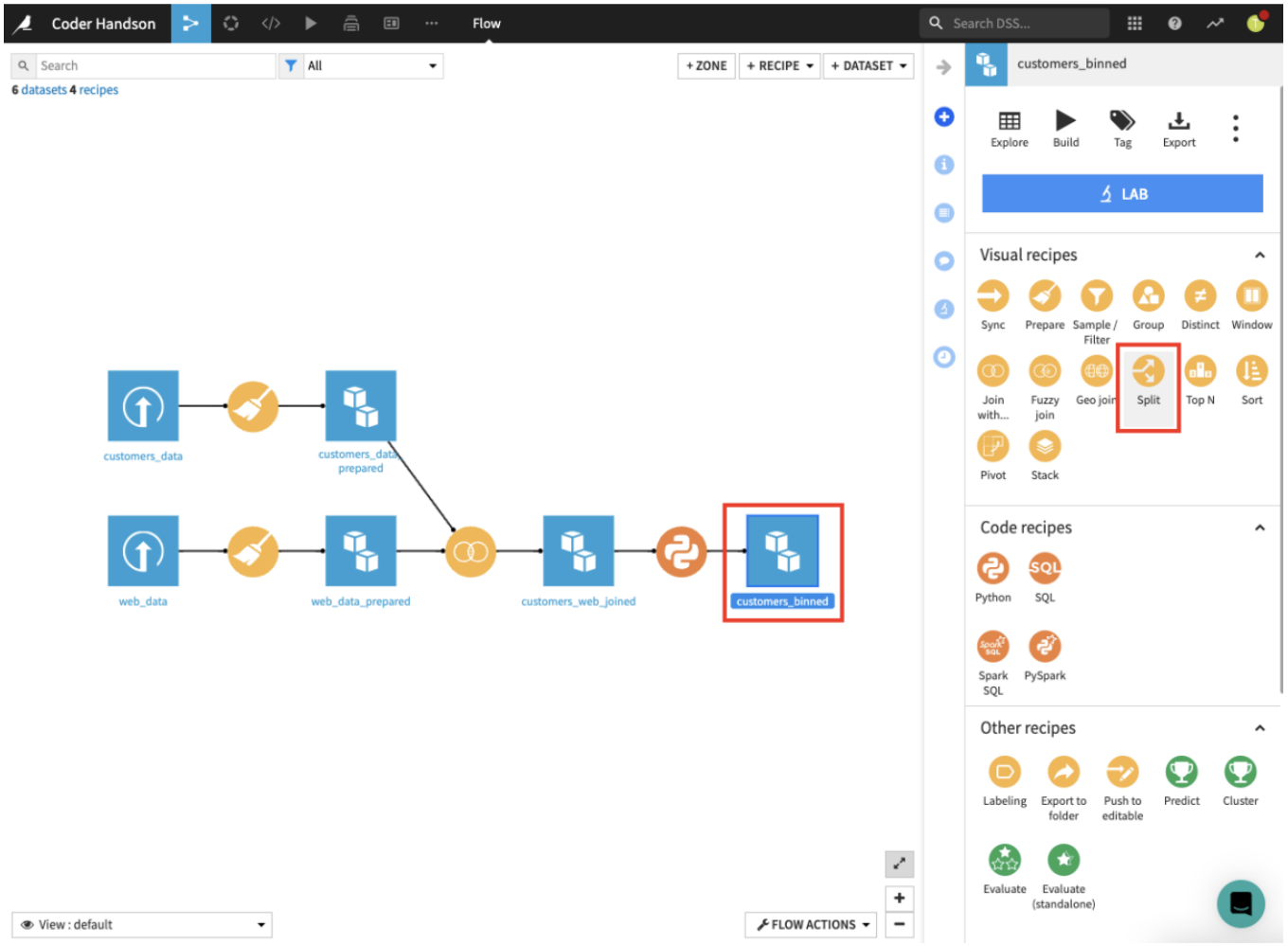
The kinds of predictive modeling to perform in data science projects vary based on many factors. As a result, it is important to be able to customize machine learning models as needed. Dataiku DSS provides this capability to data scientists and coders via coding and visual tools.

In this section, you'll see one way that Dataiku DSS makes it possible to deploy a model to the flow using code.

Split the Data into Labeled and Unlabeled Datasets

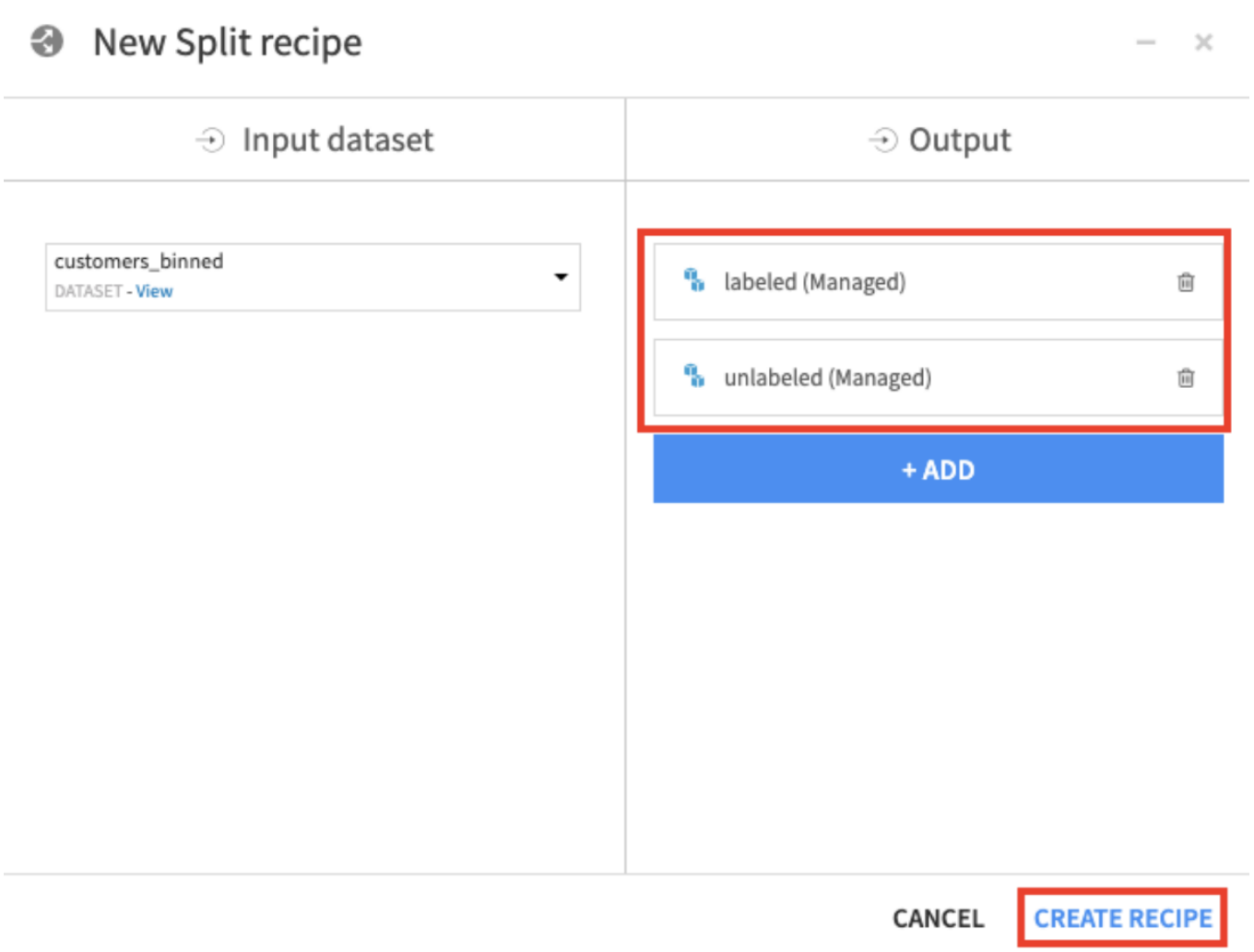
Before implementing the machine learning part, we first need to split the data in customers_binned into labeled and unlabeled datasets. For this, we'll apply the Split Recipe in Dataiku DSS to customers_binned.

- Select the **customers_binned** dataset.
- In the Actions pane, select the **Split** recipe.

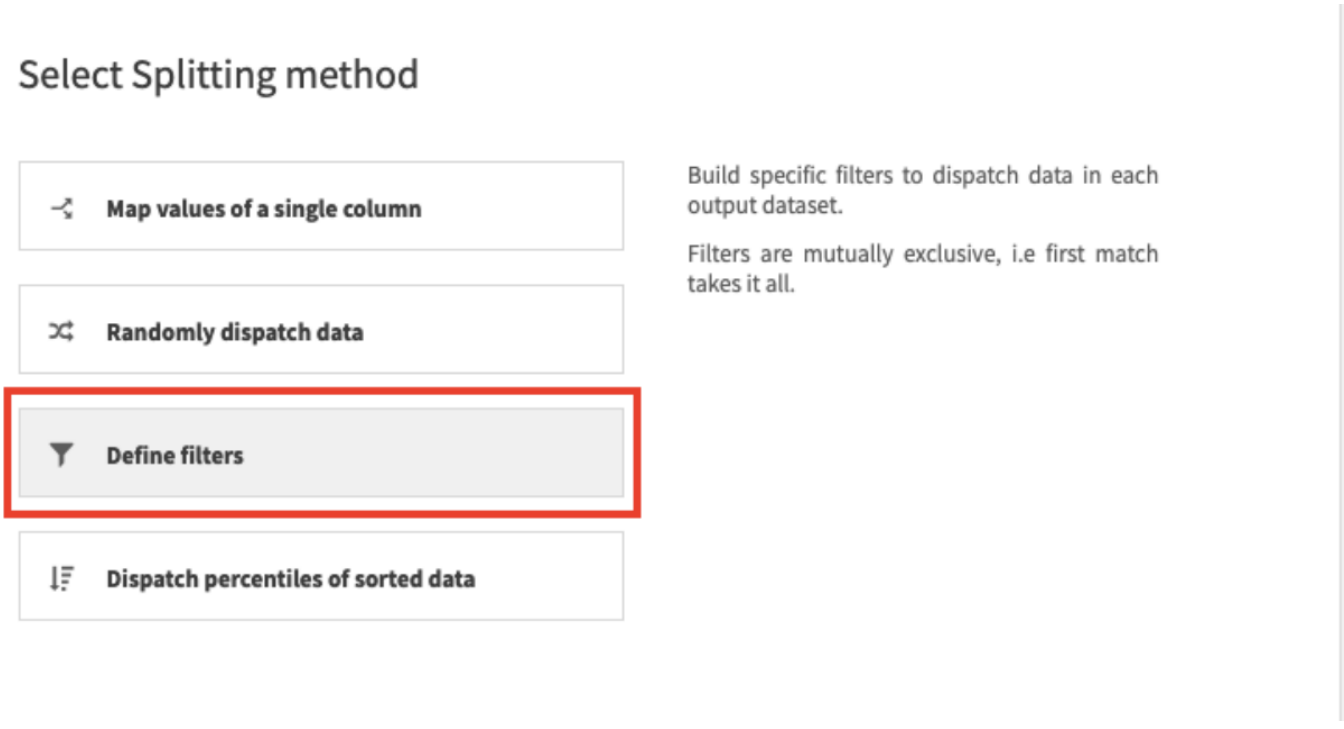


- Click **Add** to add a new output dataset.
- Name it " ***labeled*** ".
- Keep the default storage options.
- Click **Add** to add another new output dataset.
- Name it " ***unlabeled*** ".
- Keep the default storage options.

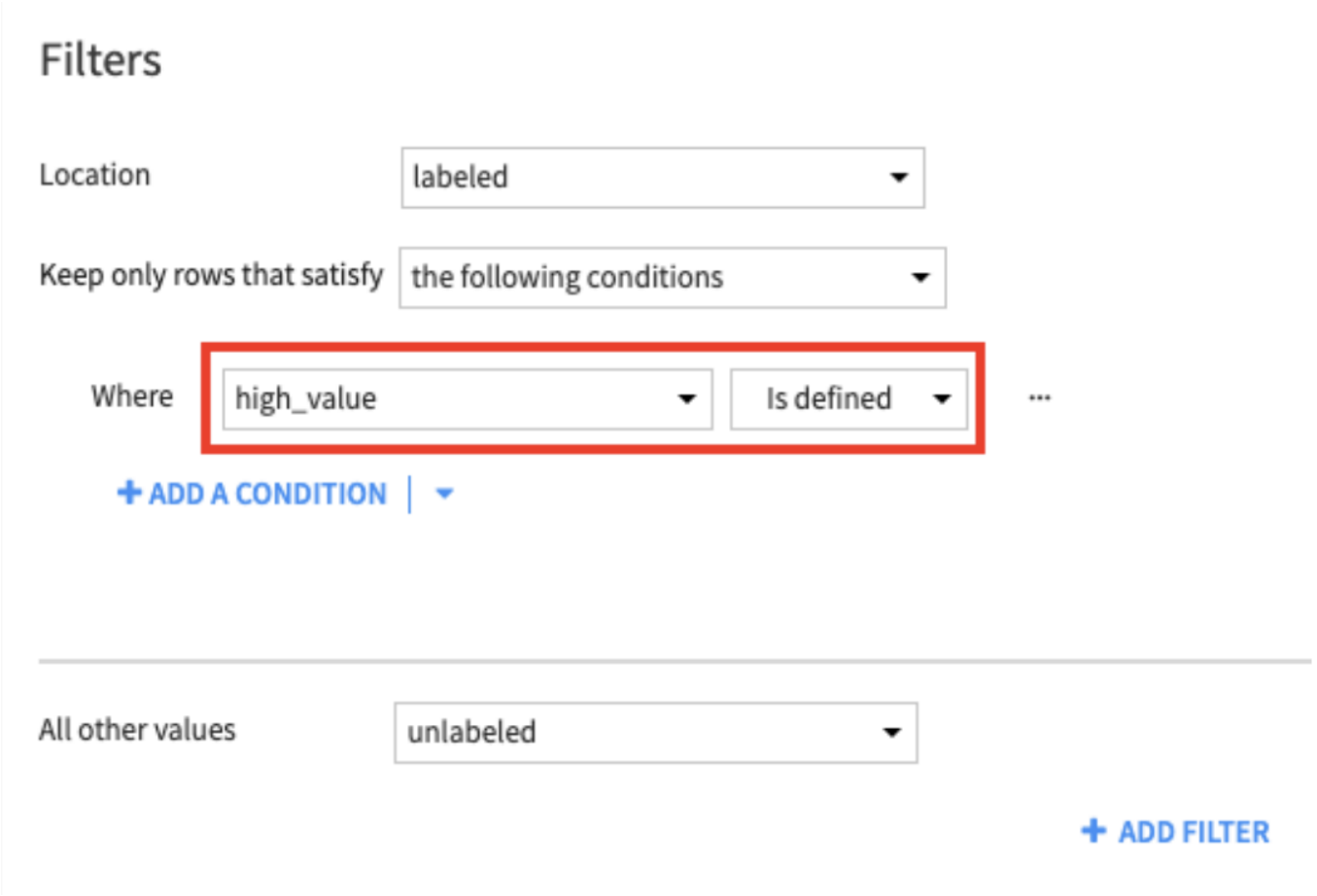
- Click **Create Recipe** .



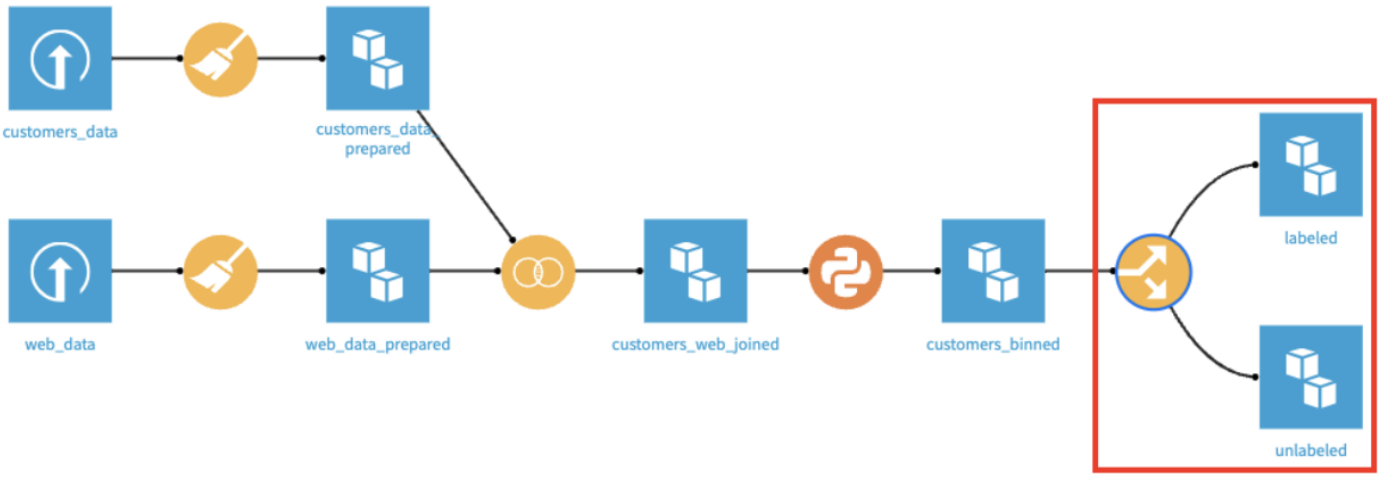
- In the “Select Splitting method” section, select **Define filters** .



- Set the “Where” clause of the filter to be “ **Where high_value is defined** ”



- Click **Run** and view the output datasets to make sure they have been split into a "labeled" dataset where all rows have high_value defined and an "unlabeled" dataset where all rows have missing values for high_value.
- The flow should look like this:

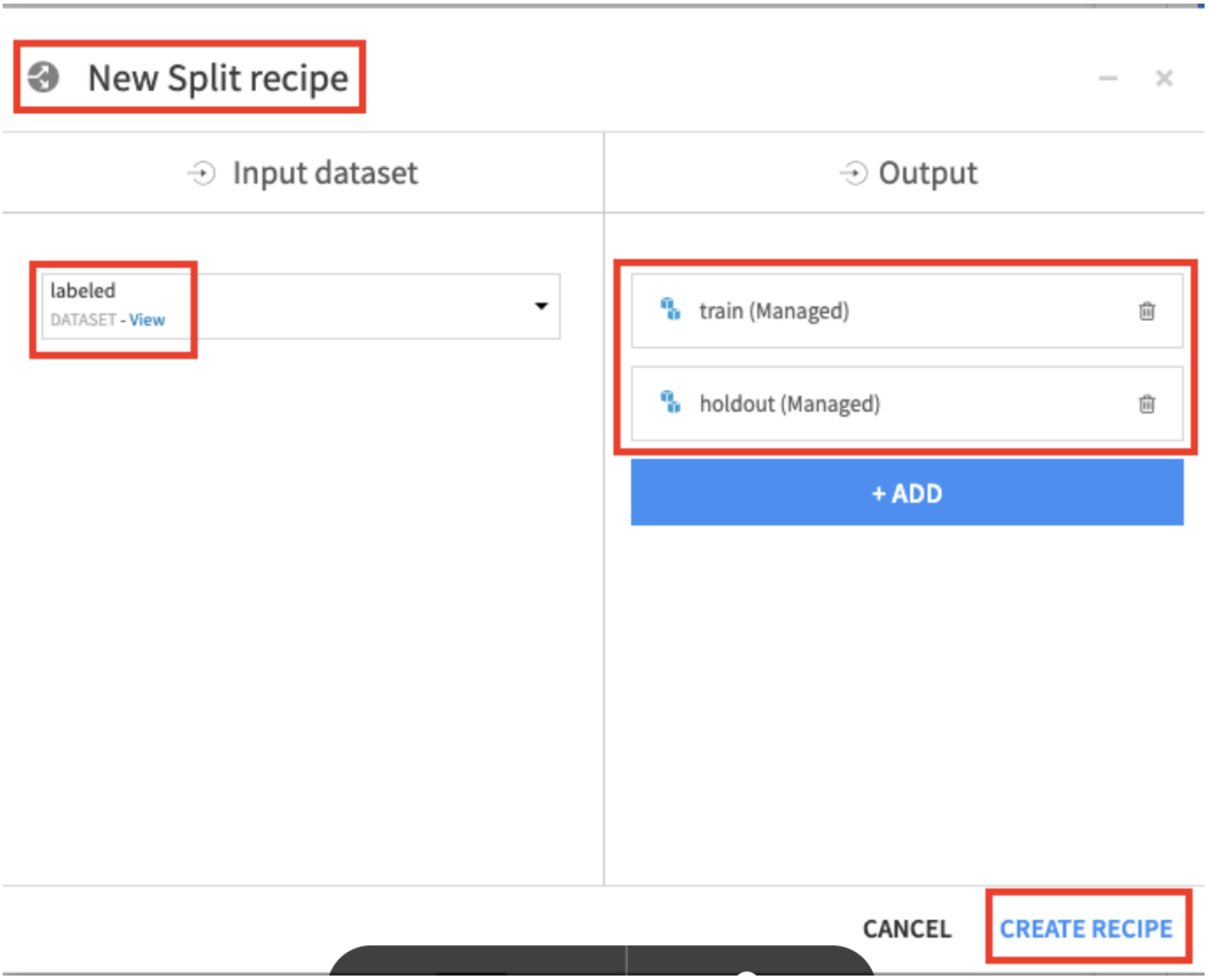


Split the Data into Train and Holdout Datasets

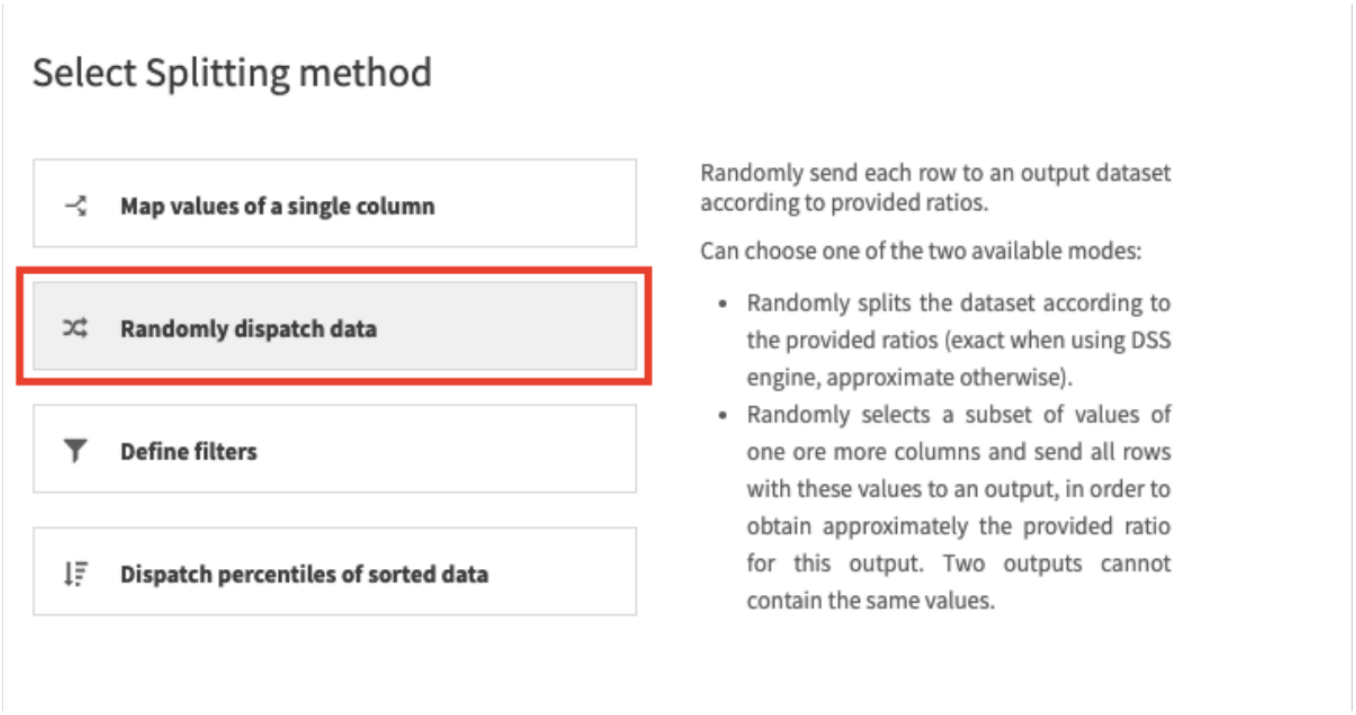
Now we're going to further split the labeled dataset into train and holdout datasets. We'll use the train dataset to train the models, and we'll use the holdout dataset to evaluate the models.

- Select the **labeled** dataset and create a **Split** recipe.

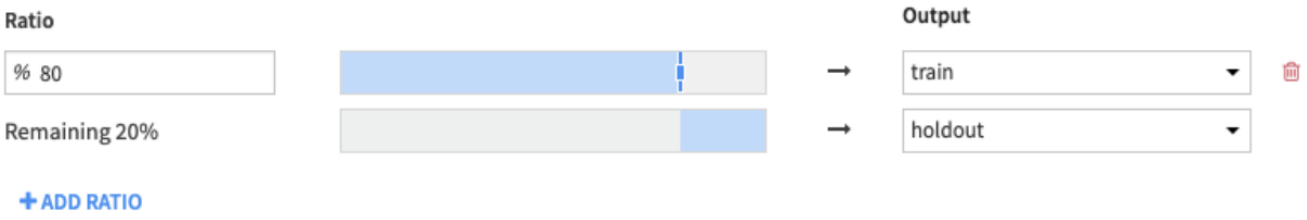
- Click **Add** to create a new output dataset named ***train*** . Keep the default storage options.
- Click **Add** to create a new output dataset named ***holdout*** . Keep the default storage options.
- Click **Create recipe** .



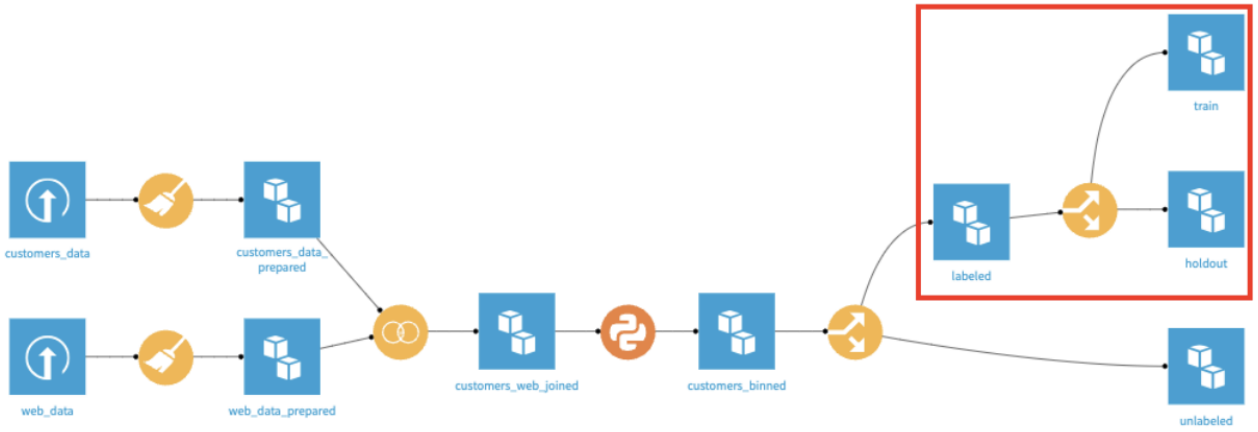
- In the “Select Splitting method” section, select **Randomly dispatch data** .



- Adjust the Ratio to send **80%** to **train** and Remaining **20%** to **holdout** .
- Click **Run** .



The flow should look like this:



Customize the Design of Your Predictive Model Using a Notebook

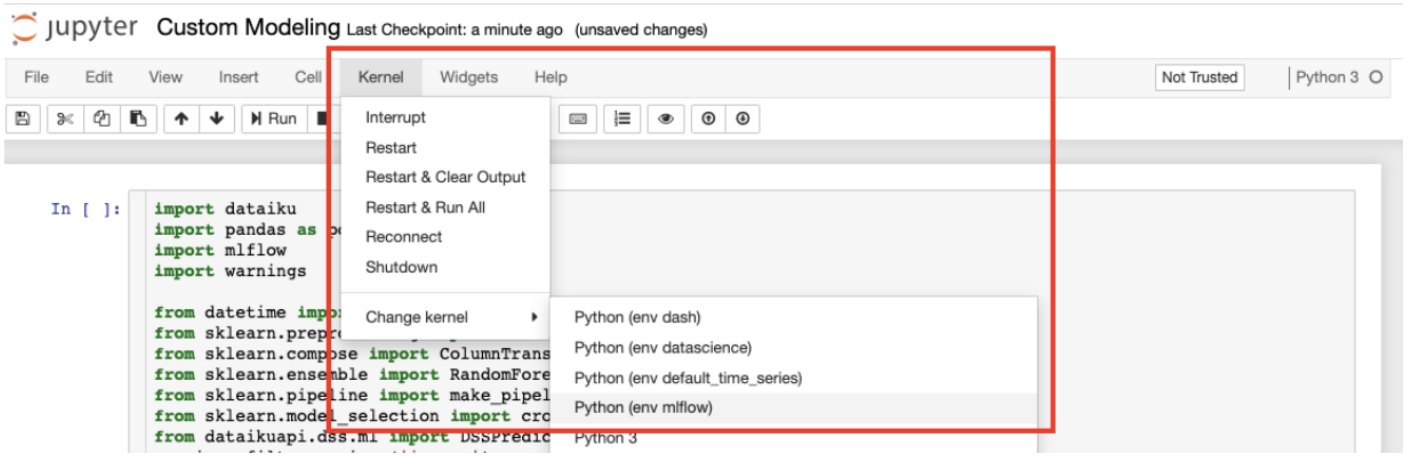
As we saw in previous sections, we can create Python notebooks and write code to implement our processing logic. Now, we'll see how Dataiku DSS allows the flexibility of fully implementing machine learning steps. Here we could import a Notebook from a git repository, upload one or write our own. For this we will import from another git repository

`https://github.com/dataiku/academy-samples.git`

- Click the Code icon (**</>**) in the top navigation bar.
- Click on **+New notebook** then paste the url above
- Click on **List notebooks**
- Click on **Import the Custom Modeling notebook**
- Click on the **Custom Modeling** notebook to open it.

This notebook will use the custom code environment we created for you with the *mlflow* experiment tracking package.

- From the notebook menu bar, select **Kernel -> Change kernel -> Python (env mlflow)**



Let's kick off the model training session before exploring the code.

- Click **Cell -> Run all**
  The comments and markdown in the notebook describe what's happening. This notebook leverages Dataiku's Experiment Tracking feature. Check to see whether the training session has completed by scrolling to the bottom of the notebook and checking for a printed line that says, " *DONE! Your artifacts are available at dss-managed-folder://…* "

Once the run has finished training, let's try a different model.

- Go to the code block called **Defining the parameters** of our run
- Replace the ***hparams*** and ***clf*** with the following code:

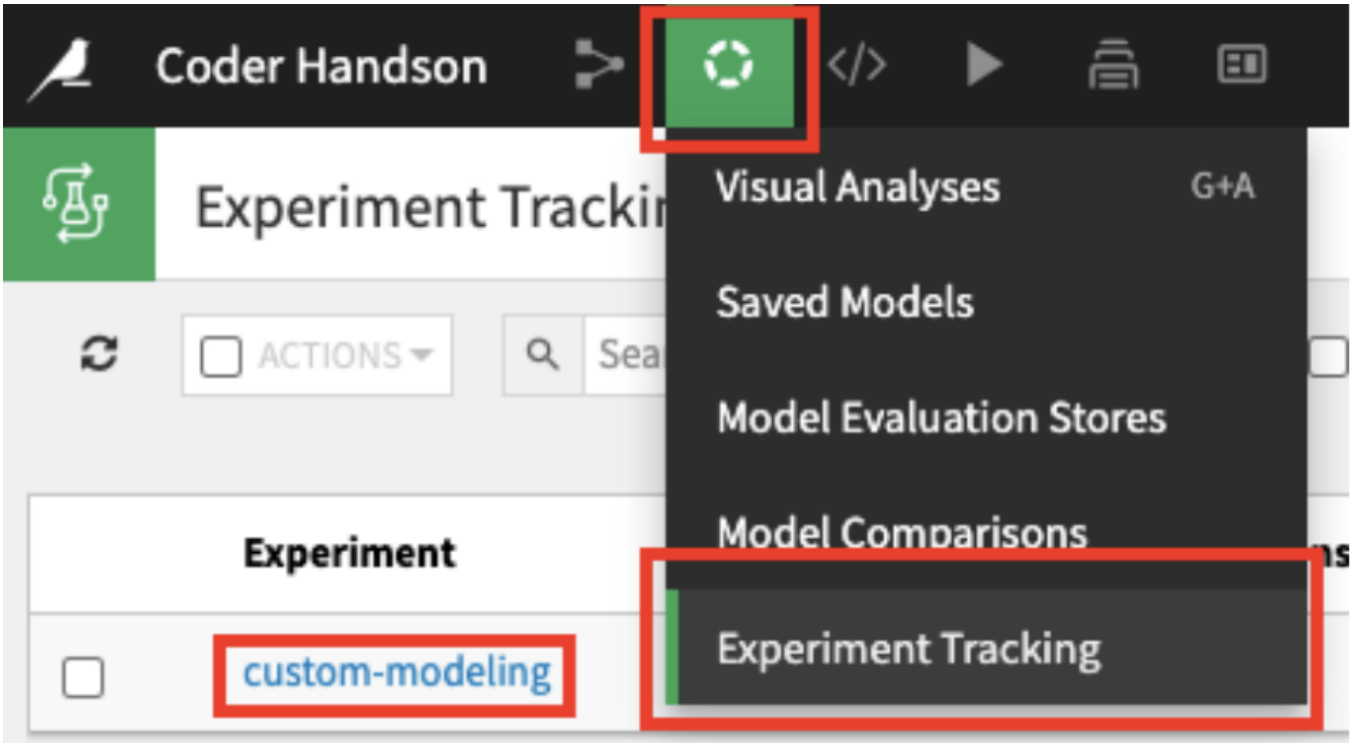
```
hparams = {"n_estimators": 50,
       "criterion": "gini",
       "max_depth": 5,
       "min_samples_split": 3,
       "random_state": 42}
clf = RandomForestClassifier(**hparams)
```

Defining the parameters of our run

In [ ]:

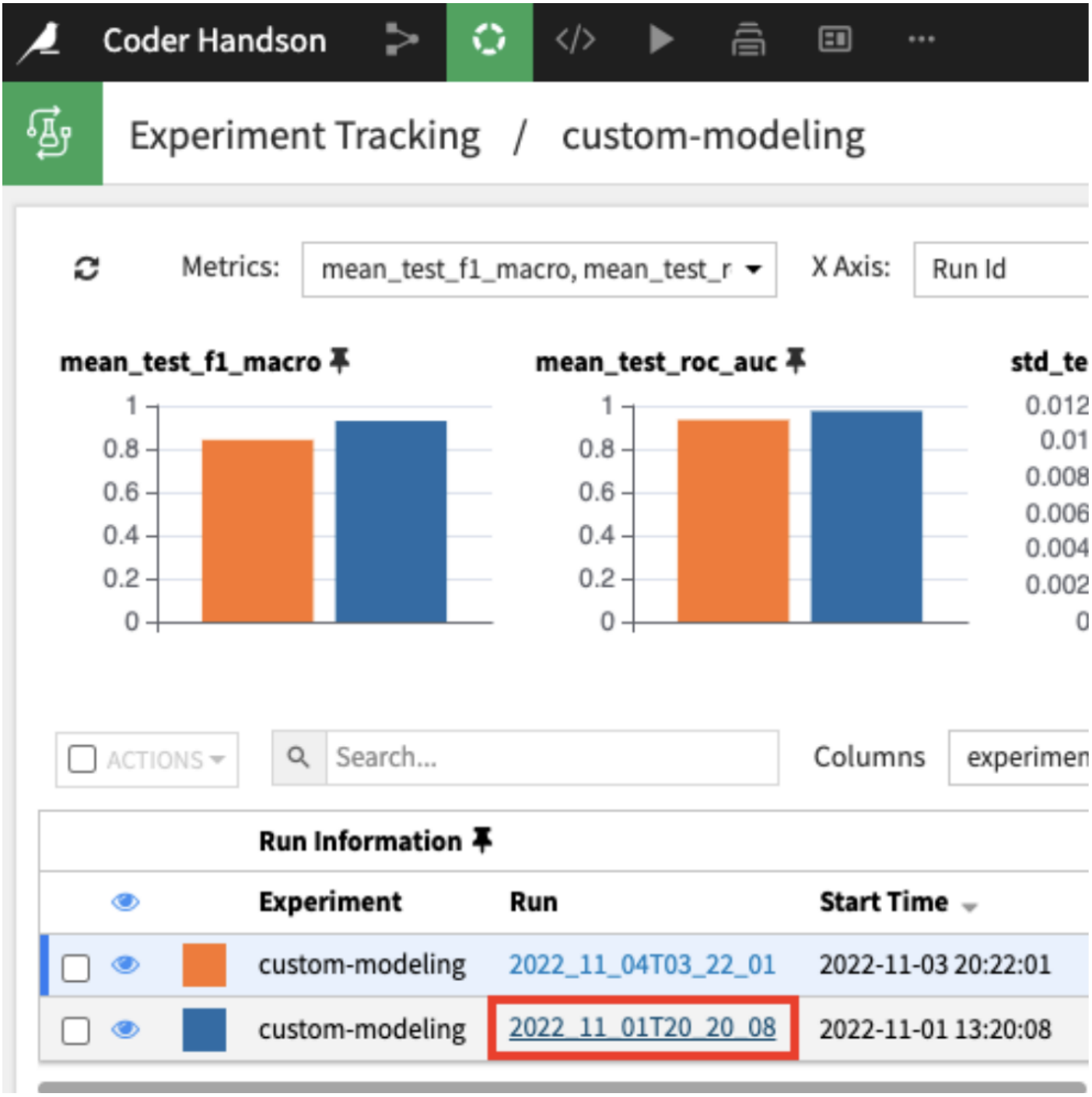
```
# Create run name
run_params = {}
run_metrics = {}

# Define run parameters
# -- Which columns to retain ?
categorical_cols = ["gender", "ip_country_code"]
run_params["categorical_cols"] = categorical_cols
numerical_cols = ["age", "price_first_item_purchased", "pages_visited", "campaign"]
run_params["numerical_cols"] = numerical_cols

# --Which algorithm to use? Which hyperparameters for this algo to try?
# --- Example: Random Forest
hparams = {"n_estimators": 50,
           "criterion": "gini",
           "max_depth": 5,
           "min_samples_split": 3,
           "random_state": 42}
clf = RandomForestClassifier(**hparams)
model_algo = type(clf).__name__
run_params["model_algo"] = model_algo
for hp in hparams.keys():
    run_params[hp] = hparams[hp]

# --Which cross-validation settings to use?
n_cv_folds = 5
cv = StratifiedKFold(n_splits=n_cv_folds)
run_params["n_cv_folds"] = n_cv_folds
metrics = ["f1_macro", "roc_auc"]

# --Let's print all of that to get a recap:
print(f"Parameters to log:\n {run_params}")
print(100*'-')
print(f"Metrics to log:\n {metrics}")
```

- Click **Cell -> Run all** to kick off another run of this experiment Now let's check out the logged runs of our experiment in the *Experiment Tracking interface* .
- Click the green "analysis" symbol, in the top DSS menu bar, and then select **Experiment Tracking**
- You should see an experiment named ***custom-modeling*** . Click on it to open view the runs of the experiment.



Here, you can see the two runs of the experiment side-by-side.

- Click on one of the runs to view the details.

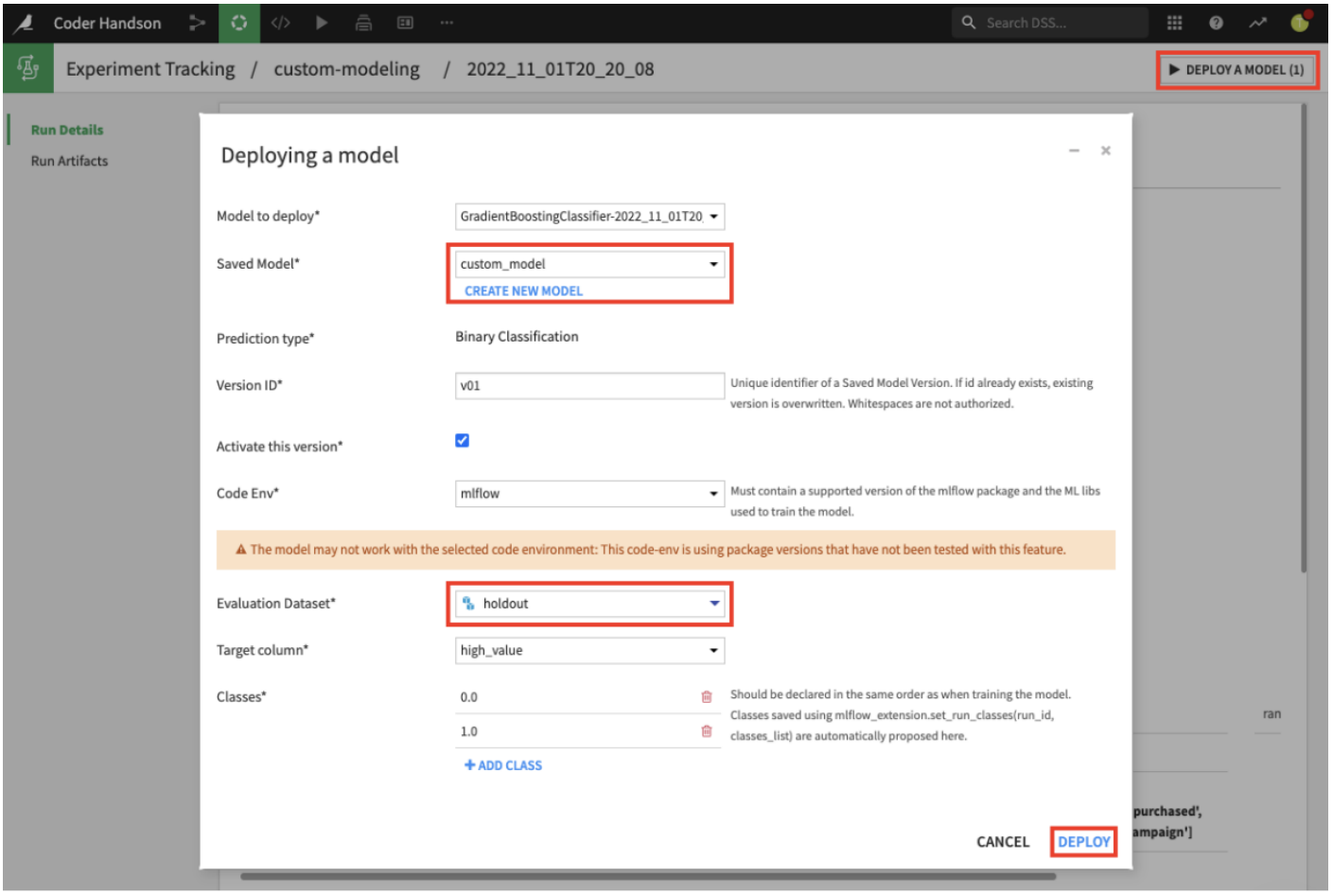


On the Run Details page, you'll see all the metadata about the run and the logged metrics, hyperparameters, and model(s). If you click on the model, you'll be taken to the *Run Artifacts* page where you can view all the files associated with the model. Namely, this includes files describing the model's required code environment and the .pkl model itself.

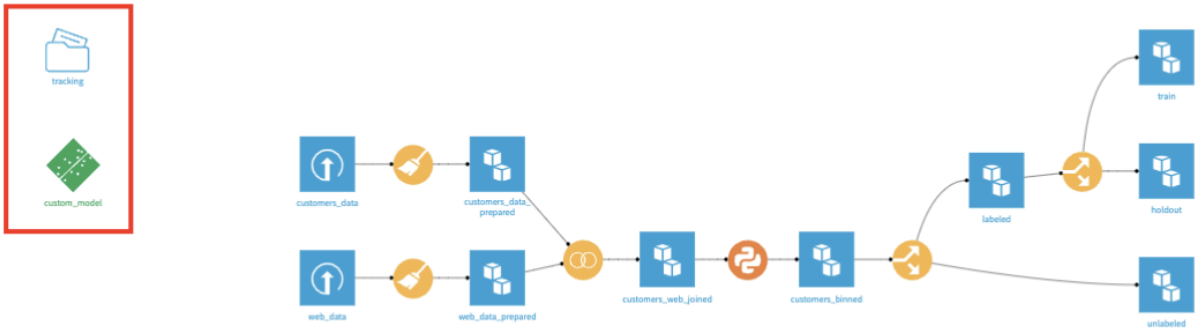
Deploy Your Custom Code Model to the Flow

Next, we'll deploy the model to the Flow to apply it to the unlabeled data.

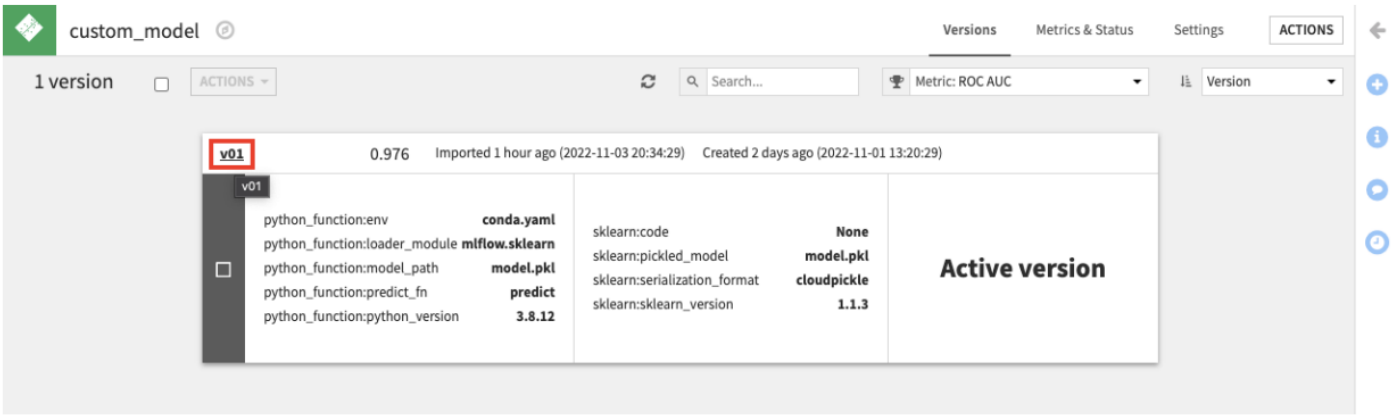
- Click **Deploy The Model** in the top right of the run page
- In the Deploying a model popup, click **Create New Model** , and name it ***custom_model***
- For **Evaluation Dataset** , select ***holdout***
- Click **Deploy**



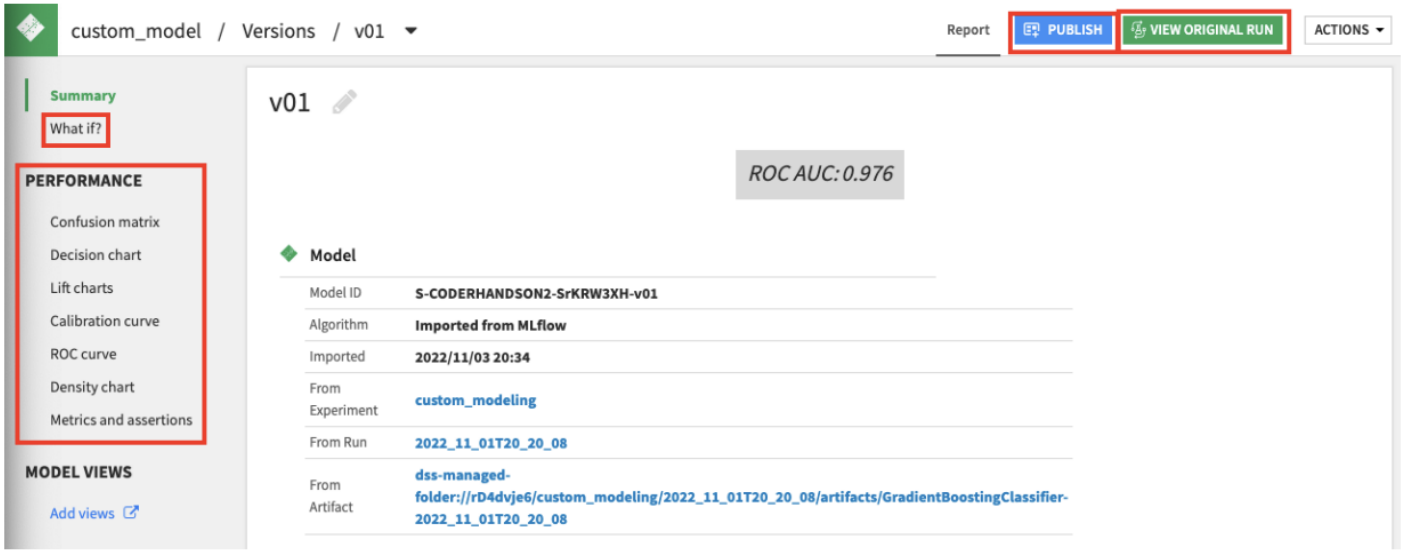
The code in the notebook created a DSS-managed folder named tracking to store the experiments and run information. When you clicked Deploy, a newly created model object, named custom_model, was saved to the flow. It should look something like this:



- Double-click on the newly created **custom_model** object to open it. You should see one version of the model with some of its metadata. Notice the “0.976” which represents that ROC AUC metric for the model.
- Click on the “ **v01** ” to open this version and inspect it further.



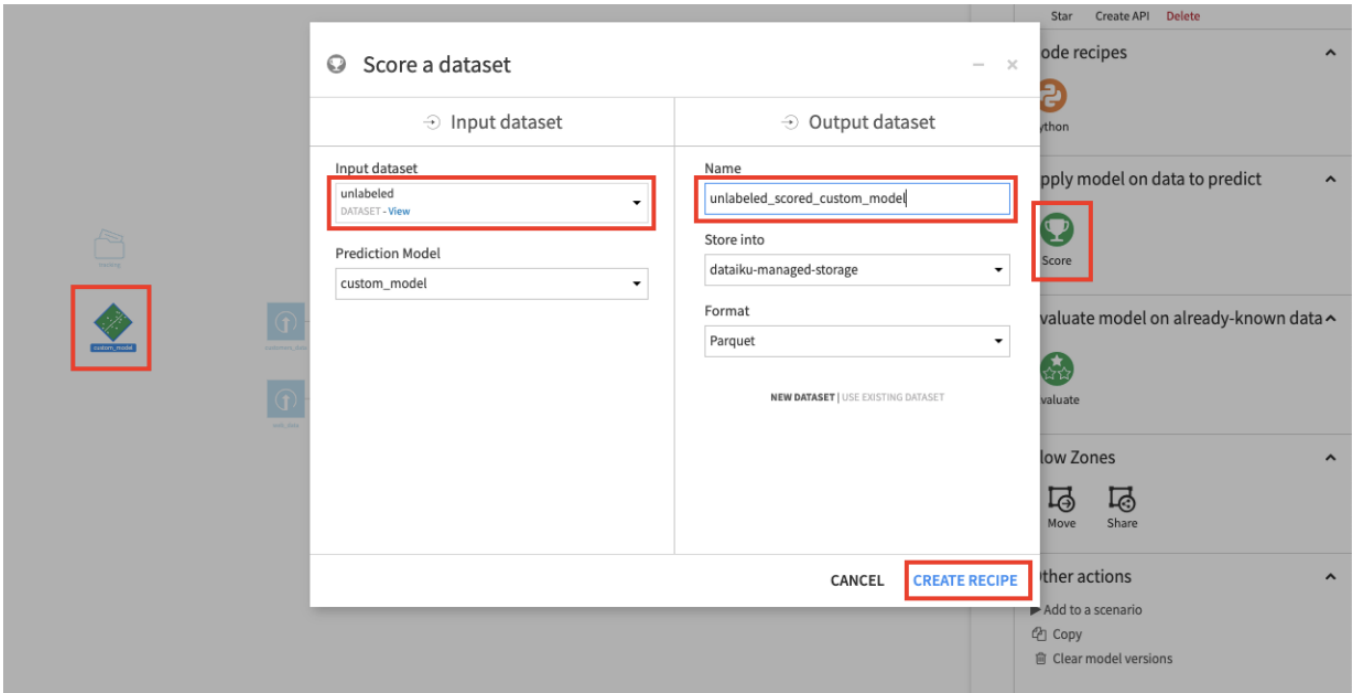
On the v01 details page, you'll see links back to the original experiment run that generated this model. You can also play with a ***What if?*** analysis that allows you (or business stakeholders) to investigate how different sample inputs would affect the prediction of the model. You can also view performance metrics graphs. You can also publish this page to a Dashboard to share with other stakeholders.



Score the custom code model

Let's apply this newly created model to the unlabeled dataset.

- In the Flow, click on the ***custom_model*** , and select the **Score** recipe from the Actions pane.
- Select ***unlabeled*** for the Input dataset
- Create a new Output dataset, named ***unlabeled_scored_custom_model*** , and keep the default storage options.
- Click **Create Recipe** .



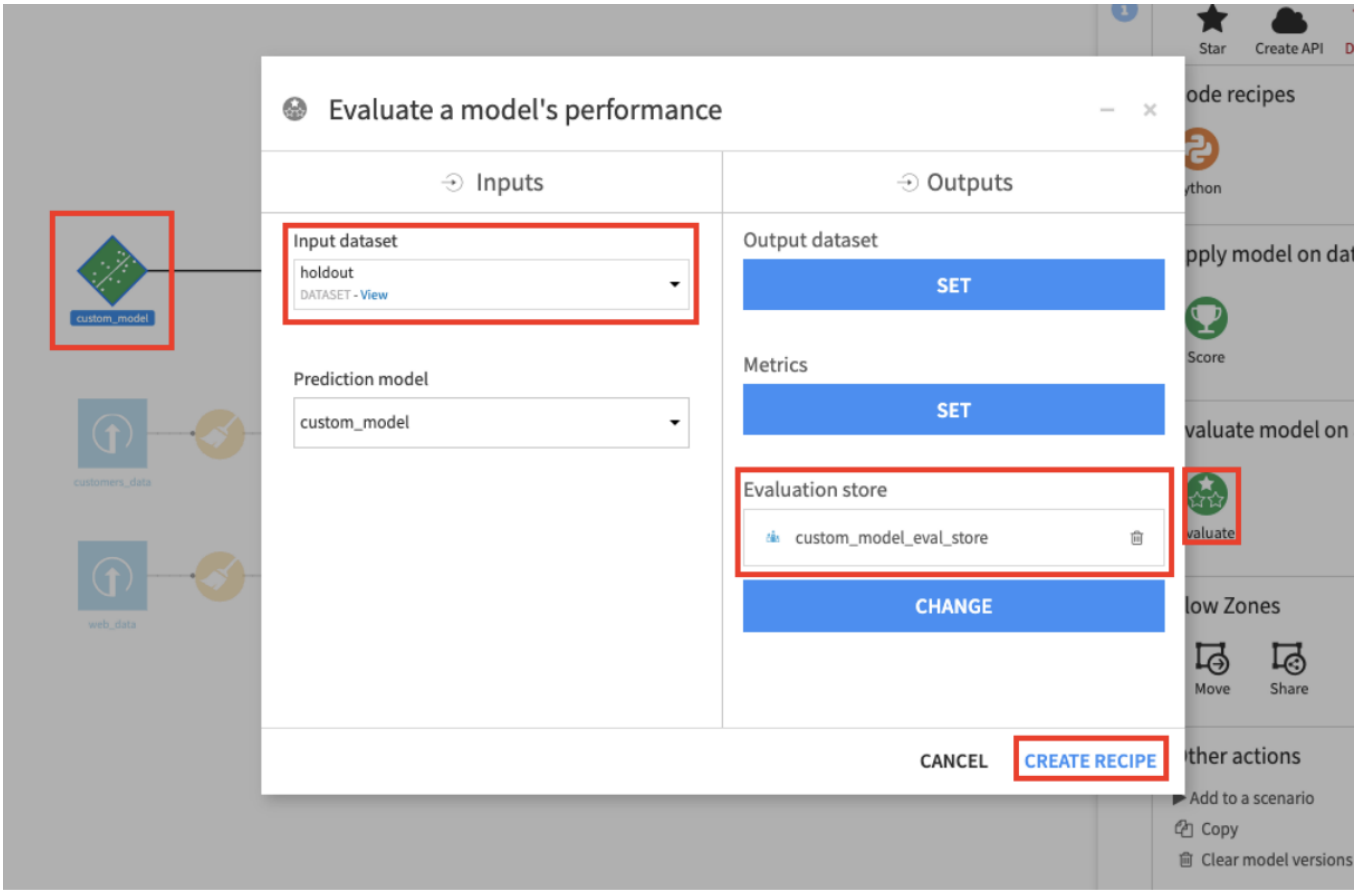
- Click the **Run** button in the bottom left on the Score recipe settings page. If you open on the *unlabeled_scored_custom_model* dataset, you should see the prediction, proba_0.0, and proba_1.0 columns that the model generated.

Create an Evaluate recipe

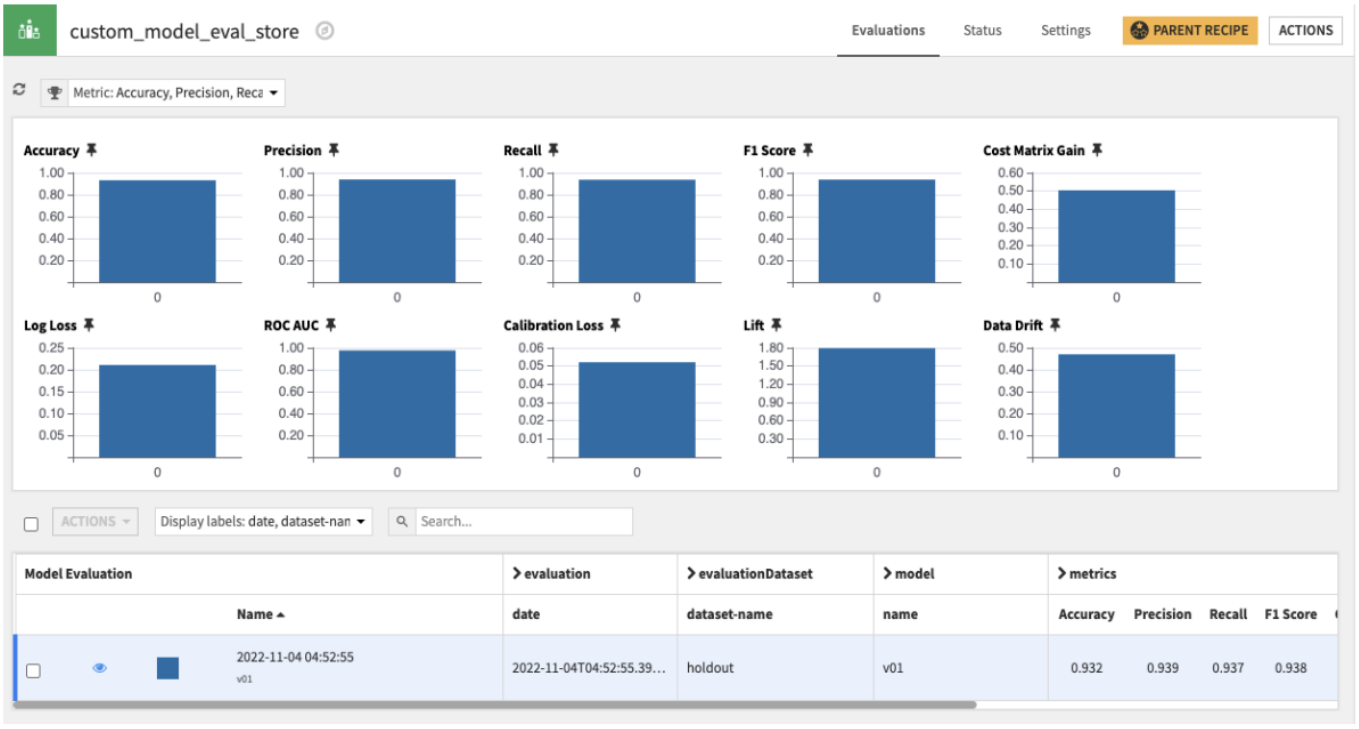
Once a model has been deployed to the Flow, you may want to track its performance on new data over time. This can be done with an Evaluate Recipe. The output of an Evaluation Recipe is an Evaluation Store which stores metrics of each time you evaluated your saved model against new data.

Let's set up an Evaluation recipe so that later in the workshop, we can automate the process of capturing our model's performance against new data.

- Click on the **custom_model** in the Flow
- In the Actions pain, select the **Evaluate** recipe
- Set the Input dataset to be ***holdout***
- In the Outputs section, create a **Evaluation store** and name it ***custom_model_eval_store***
- Click **Create Recipe**



- Click the green **Run** button on the bottom left of the Evaluation recipe Settings page If you open the newly created Evaluation Store, you'll see all the metrics of this evaluation. Right now, this will match the performance metrics of the model, since we indicated that the holdout dataset should be the "Evaluation Dataset" when deploying the model from the Experiment Tracking page. In the next section, we'll evaluate the model against newly labeled data to see how the model's performance changes with new data.



Lab 6

Automate Model Monitoring

In this section, we'll examine Dataiku's automation features by using a custom scenario. As a data scientist, you'll find scenarios in Dataiku useful for automating tasks that include: training and retraining models, updating active versions of models in the Flow, building datasets, and so on.

Let's create a scenario to automatically capture performance metrics of the model against new labeled data.

- From the Flow, go to the **Jobs** icon in the top navigation bar and click Scenarios.
- Click **+ New Scenario** to create a new scenario.
- Select **Custom Python script** .
- Name the scenario " ***Evaluate*** ".
- Click **Create** to create the new scenario.

A scenario has two components:

- **Triggers** that activate a scenario and cause it to run
- **Steps** , or actions, that a scenario takes when it runs

There are many predefined triggers and steps, making the process of automating Flow updates flexible and easy to do. For greater customization, you can create your own triggers and steps using Python or SQL.

Reporters are useful for setting up a reporting mechanism that notifies of scenario results (for example: by sending an email if the scenario fails).

Dataiku allows you to create step-based or custom Python scenarios. We're creating a Python-based scenario.

The scenario script is a sample, full-fledged Python program that executes scenario steps. This sample script uses the Dataiku API to access objects in the project. The sample script also uses the Scenarios API within the scenario in order to run steps such as building a dataset and training a model. Let's define our own script:

- Delete all the code in the sample script.
- Replace it with the following:

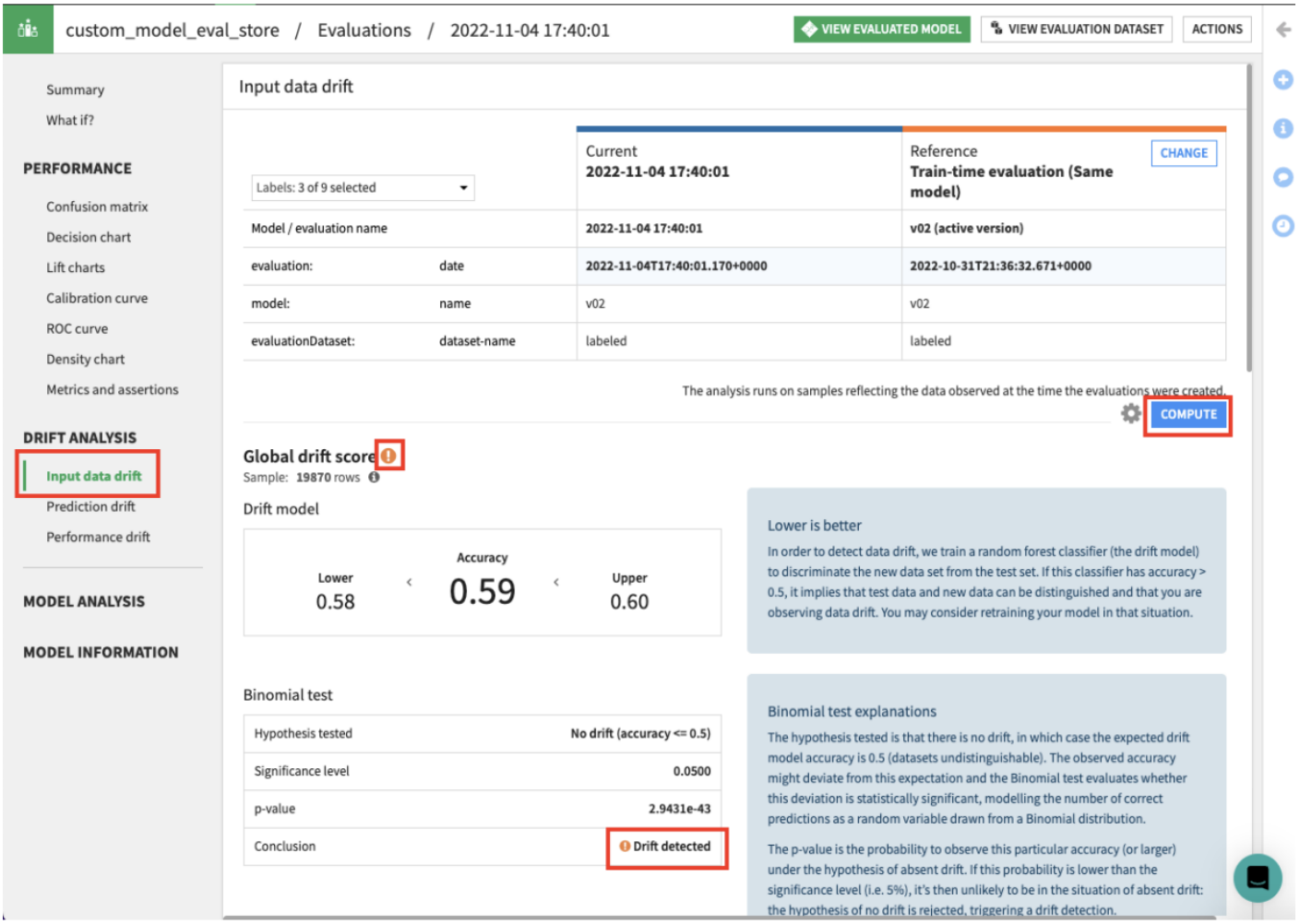
```
import dataiku
from dataiku.scenario import Scenario
```

```
scenario = Scenario()

scenario.build_dataset("customers_binned",build_mode='NON_RECURSIVE_FORCED_BUILD')

eval_store = dataiku.ModelEvaluationStore("custom_model_eval_store")
scenario.build_evaluation_store(eval_store.get_id())
```

- Click **Save** . When we run the scenario, it will perform the following tasks.
- Build the *customers_binned* .
- Build the *custom_model_eval_store* evaluation store. Because the customers_binned dataset has been changed in the previous step, Dataiku will re-build the labeled and holdout datasets with the new data to be evaluated.

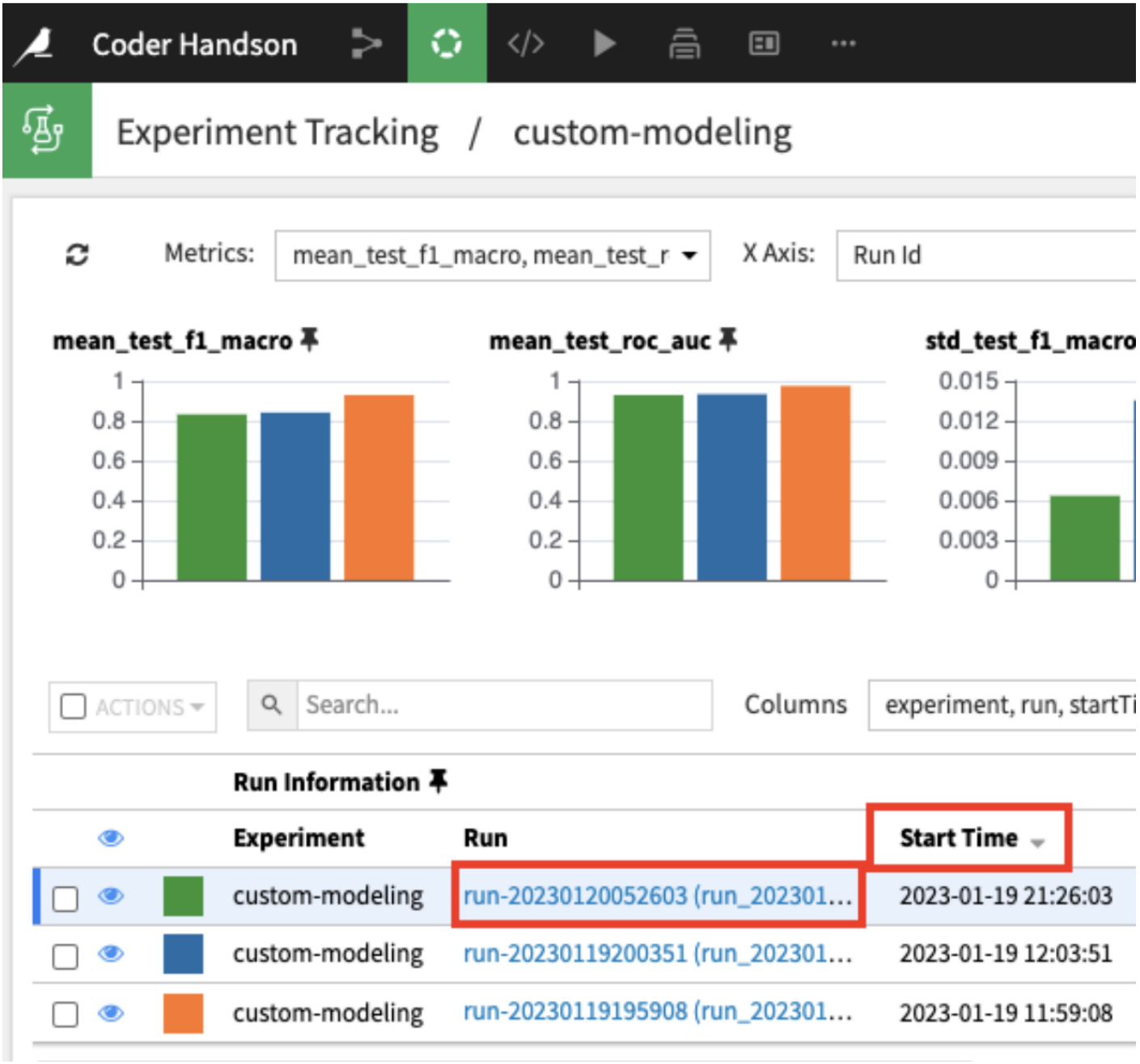
Before we run the scenario, we'll make a change to the project:

- Go to the project variables by going to the **More actions ...** menu in the top navigation bar.
- Change the value of the project variable " ***mock_drift*** " to **yes** .
- Make sure to click **Save** . When the customers_binned dataset is rebuilt with this new variable value, the `mock_drift` library function will mock some data drift on the input variables. The scenario is now ready to use. To test it,
- Return to the **Evaluate scenario** .
- Click **Run** and wait for the scenario to complete its jobs. You can switch to the **Last runs** tab of the scenario to follow its progress. Here, you can examine the details of the run, what jobs are triggered by each step in the scenario, and the outputs produced at each scenario step. You may have to click the Refresh button next to Run logs in order to refresh the build status.
- Return to Flow and double-click on the **custom_model_eval_store** to open it.
- You should see two evaluations. **Click** on the most recent one to open the report.
- In the report, click **Input data drift** and click **Compute** . You should see a report that indicates the data has drifted.

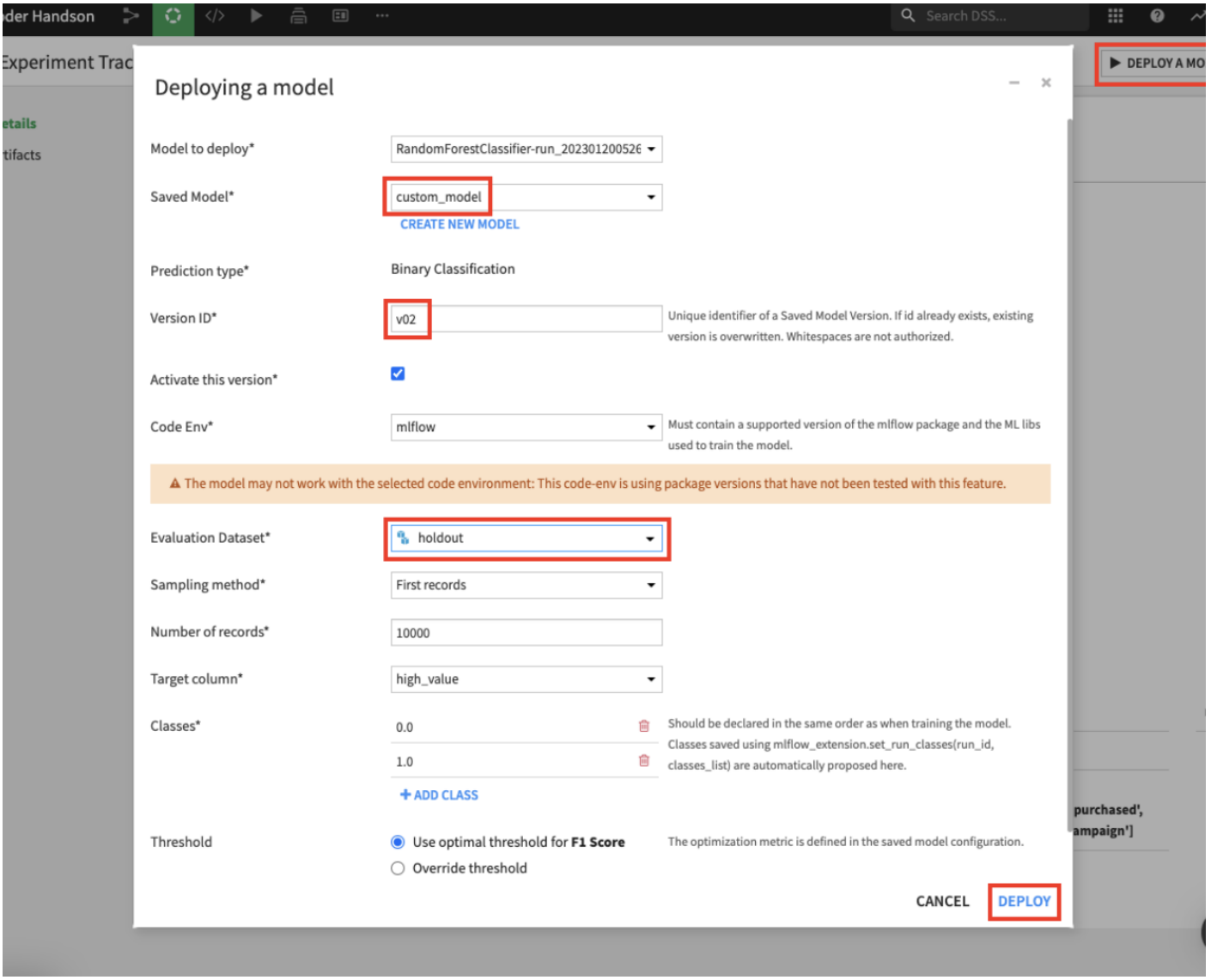


You can scroll down to investigate the drift of different input features. Namely, we mocked drift in the age feature.

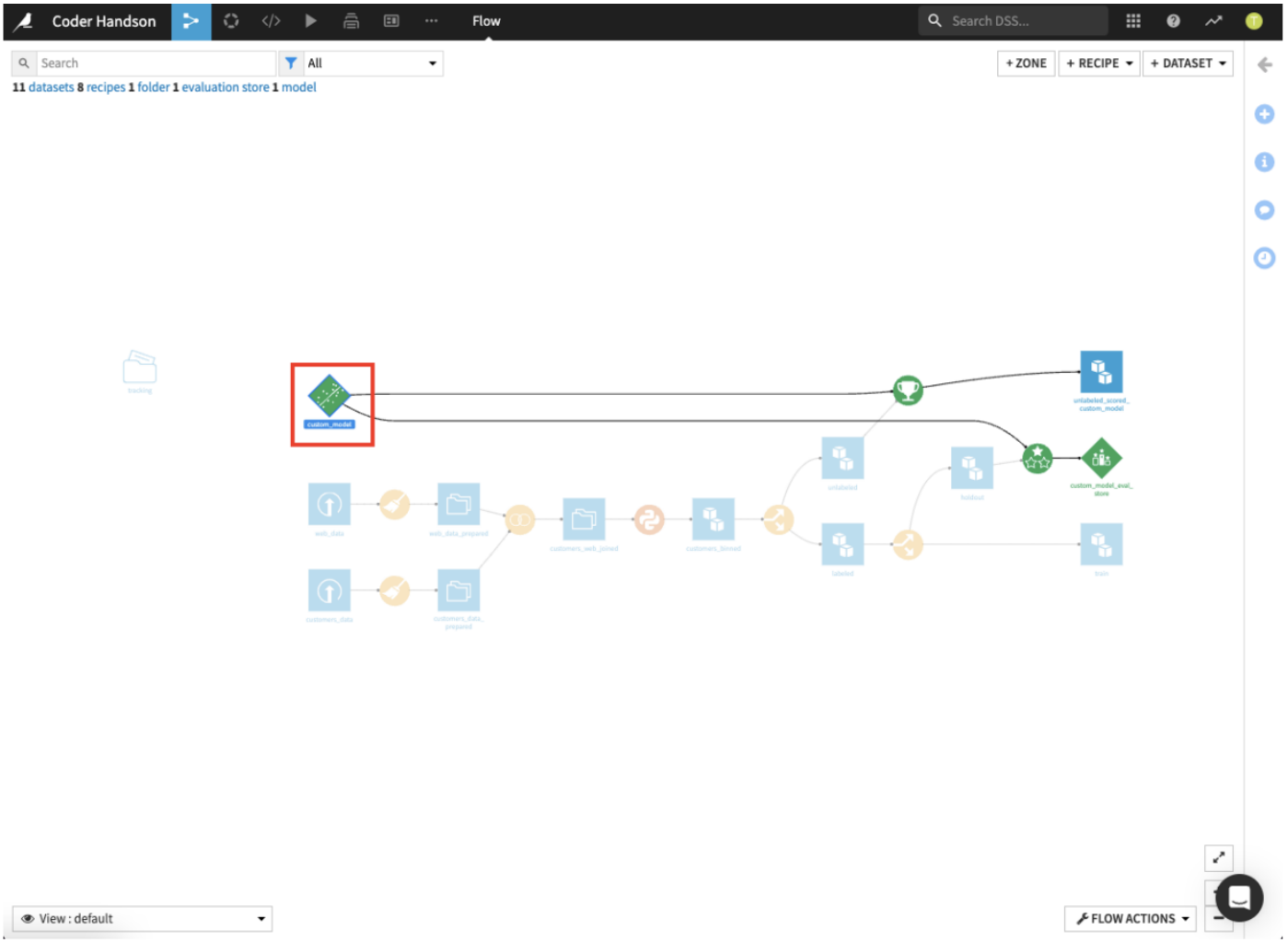
- Click the green **View Evaluated Model** on the top of the screen to take you back to the model that was evaluated.
- From there, you can click **View Original Analysis** to take you back to the experiment run that trained the model.
- Here, you might investigate other runs of the experiment to see whether they might be good candidates for a new version of the model. However, because our input data has drifted, let's retrain the model on the latest data.
- Go back to the **Custom Modeling** code notebook that trained the model.
- Click **Cell -> Run all** to re-run the notebook and retrain the model against the latest data.
- Click on the **most recent experiment** .



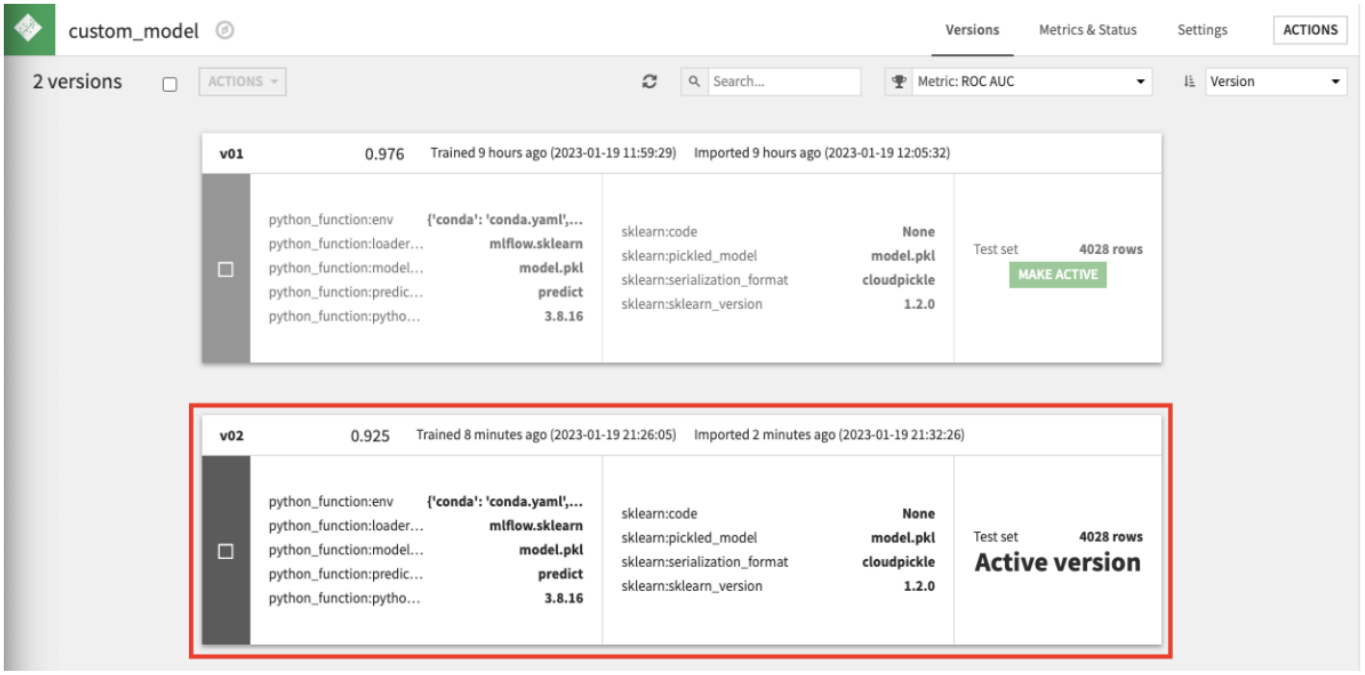
- Click **Deploy a Model (1) i** n the top right corner, and select the following options to deploy a new version of the model:



- Click on the **saved model in the Flow** .



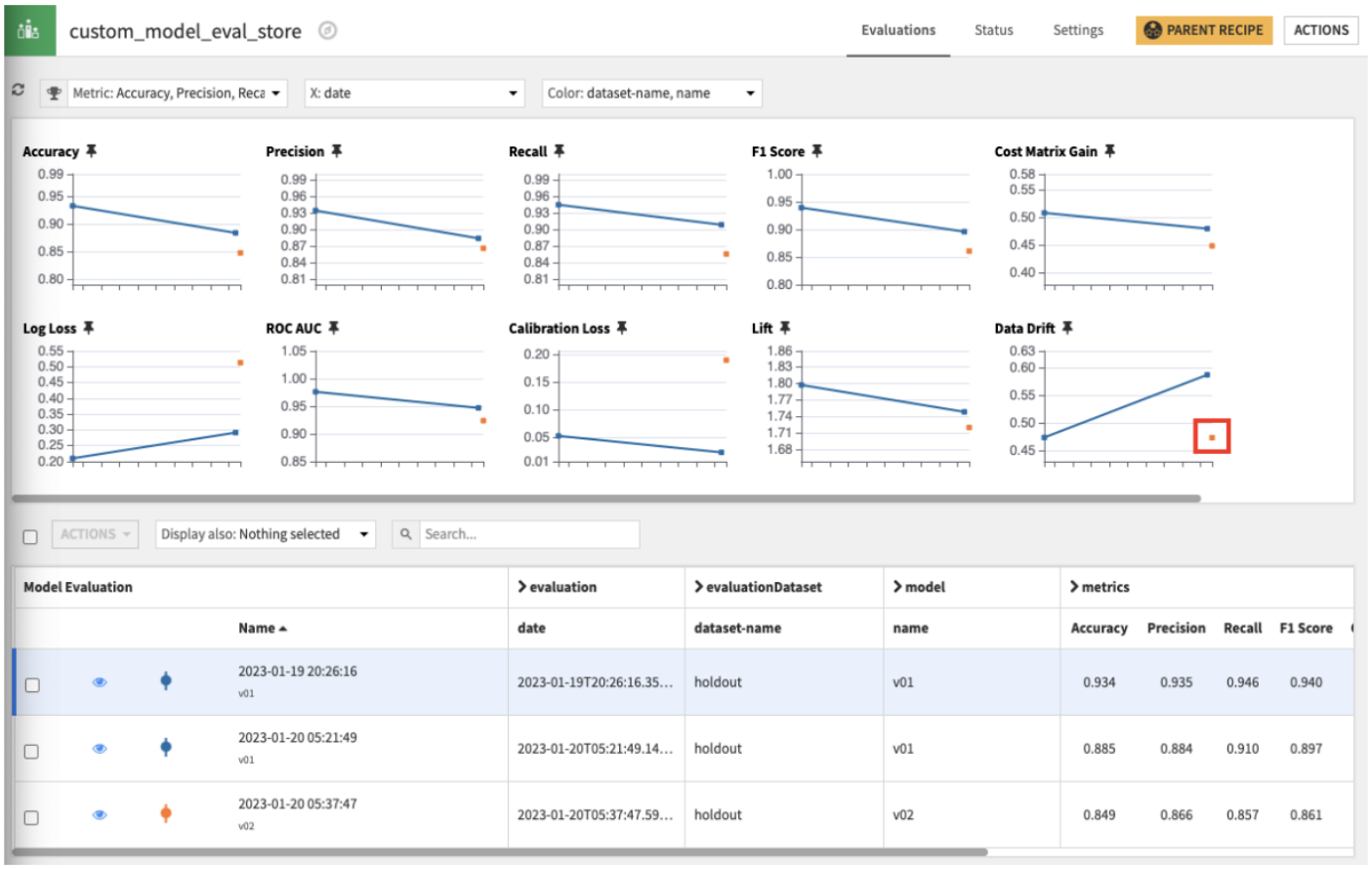
You should see that there is now a second version of the model, which is currently the Active Version. Dataiku automatically keeps track of all versions of a saved model.



- Re- **run** the **Evaluate** scenario by navigating back to the Scenarios page, clicking on the Evaluate scenario, and clicking **Run** .

- Once the scenario has finished, go back to the **Flow** , and double-click on the **eval_store** to view the most recent evaluation of the model.

Now you'll see the latest evaluation of the model was executed against the latest version, represented in orange. You'll also notice that the data drift score has gone back down because this version of the model was trained on the most recent data.



[Optional] Lab 7 - AutoML

Customize the Design of Your Predictive Model Using Visual ML

The process of building and iterating a machine learning model using code can quickly become tedious. The visual machine learning tool in Dataiku DSS simplifies the process of automating much of the feature handling and pipeline building, so you can focus on tweaking the experiment.

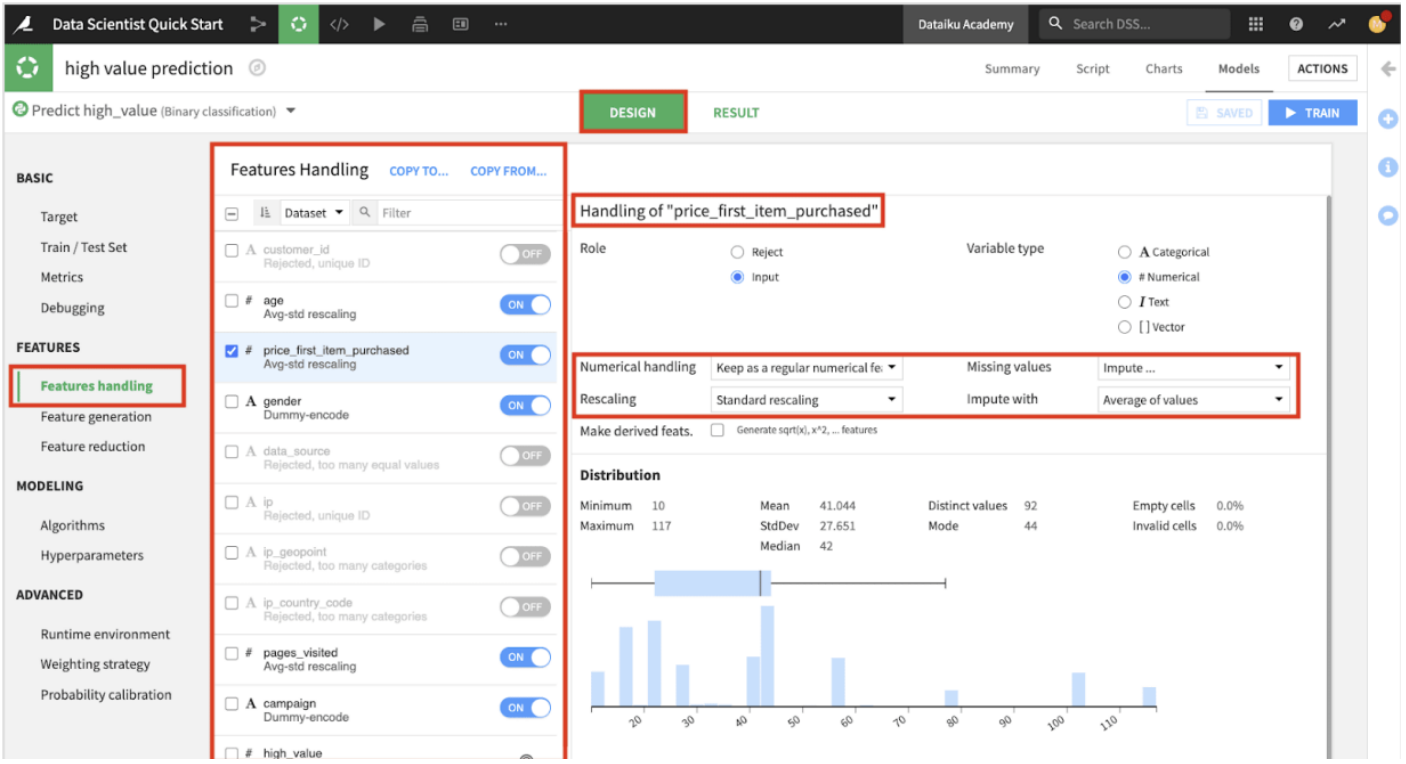
This section will explore how you can leverage this visual machine learning tool to perform custom machine learning. Specifically, we'll discover how to:

- Train several machine learning models in just a few steps;
- Customize preprocessing and model design using either code or the visual interface;
- Deploy models to the Flow to be scored or evaluated and used in scenarios.

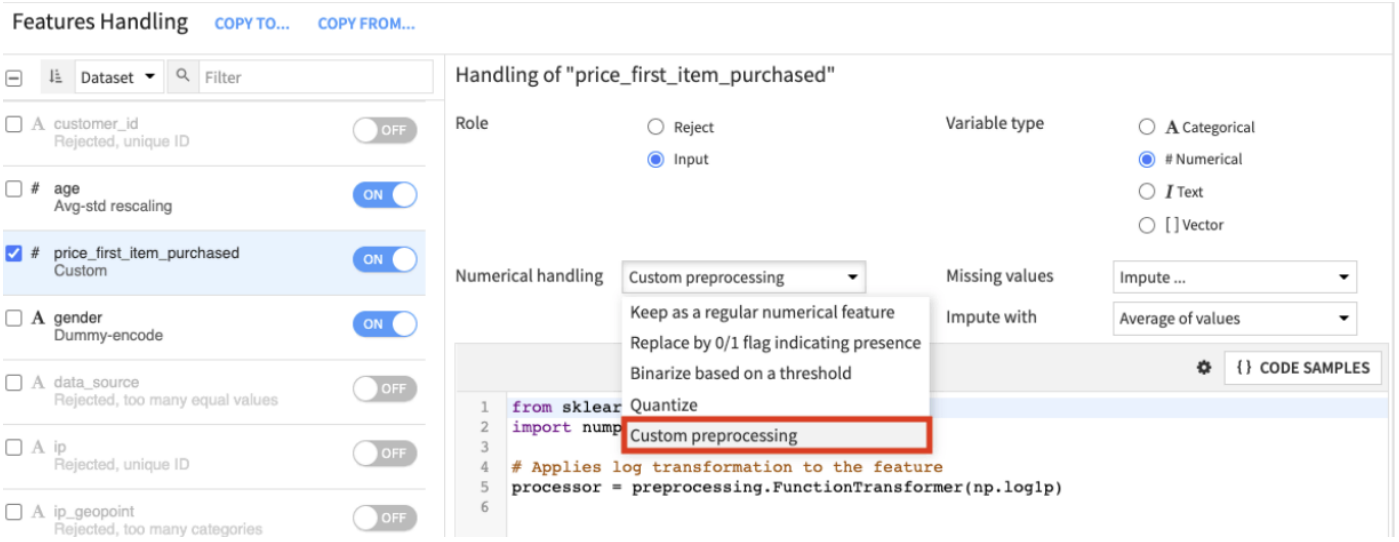
Train Machine Learning Models in the Visual ML Tool

- Select the ***train*** dataset in the Flow.
- Open the **right panel** and click **Lab** .
- Select **AutoML Prediction** from the "Visual analysis" options.
- In the window that pops up, select ***high_value*** as the feature on which to create the model.
- Keep the default option for now, and click **Create** .
- On the next page, click the **Design tab** to customize the design. Here, you can go through the panels on the left side of the page to view their details.
- Click the **Train / Test Set** panel on the left to configure the training and testing datasets used for the model training.
- In the **Policy** dropdown, select **Explicit extracts from two datasets** .
- Keep the **Train** set to be the default train dataset that we used to create this **AutoML experiment** .
- Set the **Test** set to be the holdout dataset.
- Click the **Features handling** panel to view the preprocessing. Notice that Dataiku has rejected a subset of these features that won't be useful for modeling. For the enabled features, Dataiku already implemented some preprocessing:
- For the numerical features: Imputing the missing values with the mean and performing standard rescaling

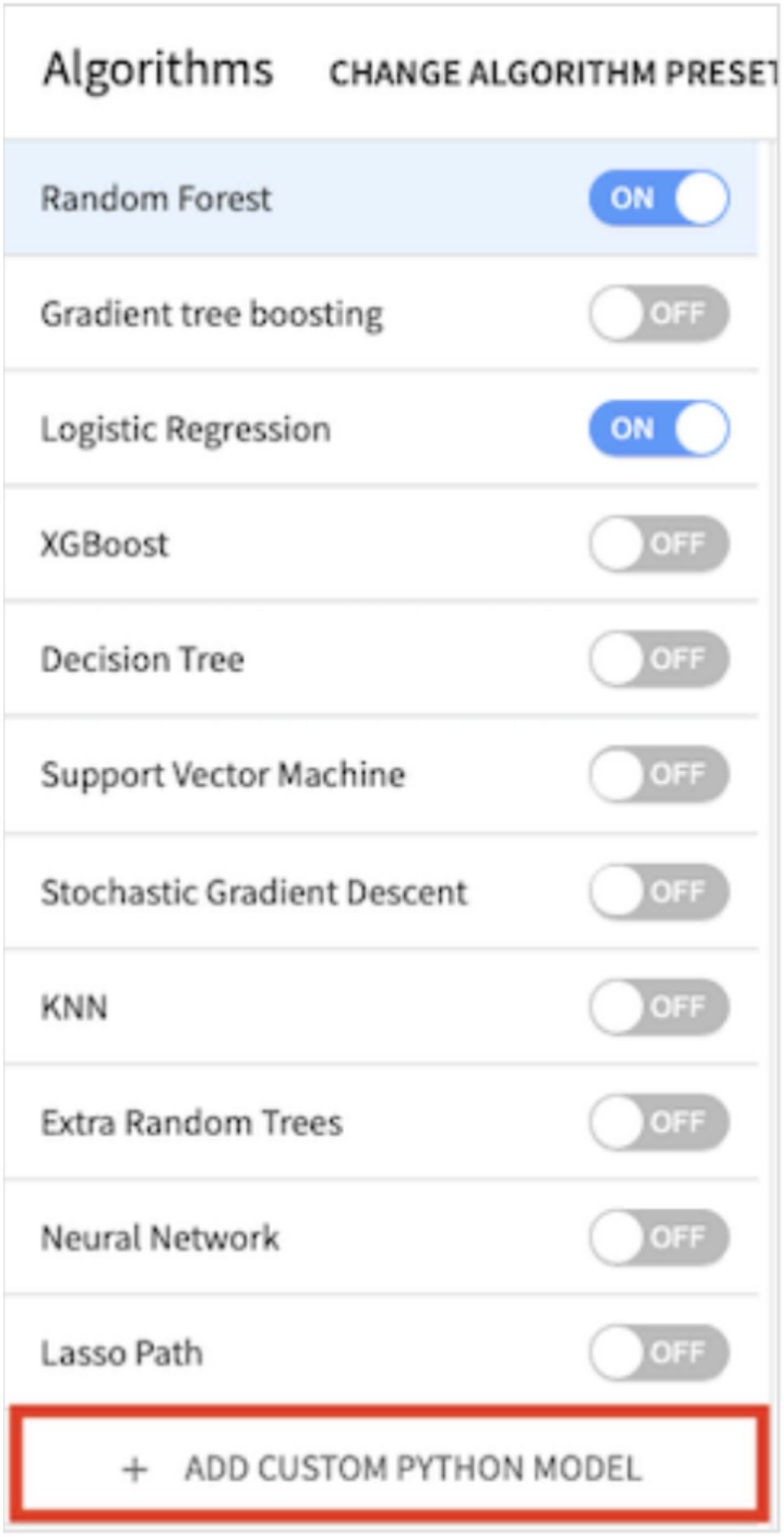
- For the categorical features: Dummy-encoding



If you prefer, you can customize your preprocessing by selecting **Custom preprocessing** as the type of "Numerical handling" (for a numerical feature) or "Category handling" (for a categorical feature). This will open up a code editor for you to write code for preprocessing the feature.



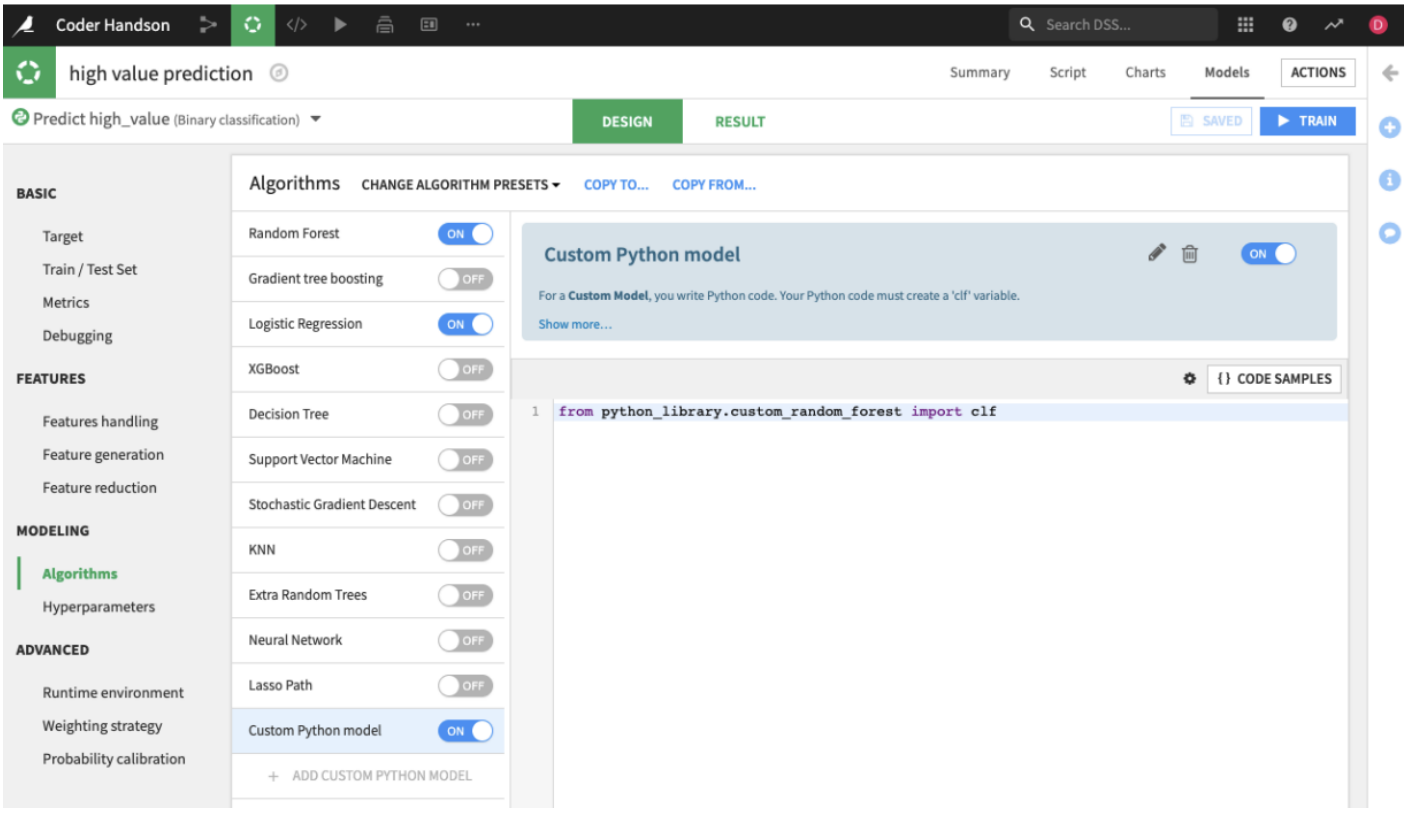
- Click the **Algorithms** panel to switch to modeling. Here, you can enable which algorithms you want to use and adjust their hyperparameter search space. We'll also create some custom models with code in this ML tool.
- Click + **Add Custom Python Model** from the bottom of the models list.



Dataiku DSS displays a code sample to get you started. The Code Samples button in the editor provides a list of models that can be imported from scikit-learn. You can also write your own model using Python code or import a custom ML algorithms that was defined in a project library. Note that

the code must follow some constraints. Here, the custom code must implement a classifier that has the same methods as the classifier in scikit-learn; that is, it must provide the methods `fit()`, `predict()`, and `predict_proba()` when they make sense. The Academy course on Custom Models in Visual ML covers how to add and optimize custom python models in greater detail.

- Delete the code in the editor and type: `from python_library.custom_random_forest import clf`
  This will import the clf custom random forest classifier from the git library that we imported earlier.

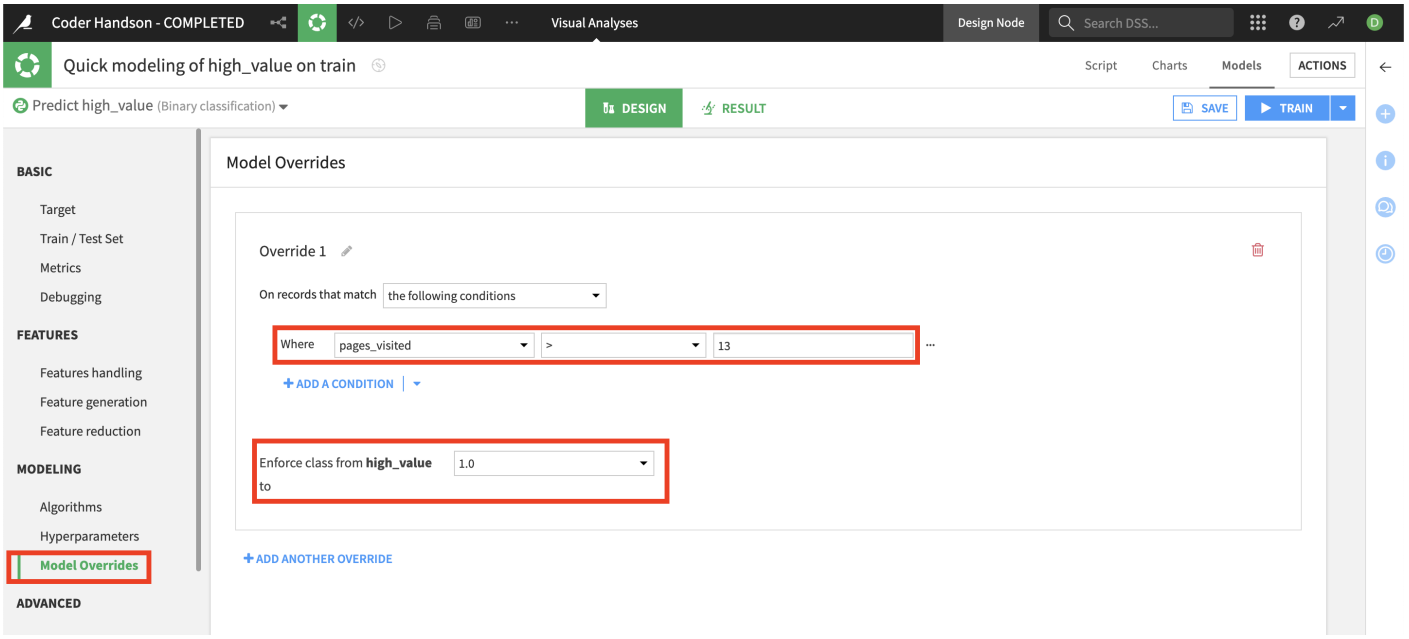


The selected code environment must include the packages that you are importing into the Visual ML tool. In our example, our custom library is using a scikit-learn model, which is already included in the DSS builtin code environment that is in use.

- **Save** the changes

Additionally, sometimes you might want to add a human layer of control over model predictions to ensure the final outcome complies with certain business rules or conditions. In our case we will want to make sure that any customer that visited our page more than 13 times is considered high_value.

- Click on **Model overrides** on the left hands side panel
- Click on **Add Override**
- In the Where tab select ***pages_visited***
- For the operator select **>** (greater than) then add **13** in the input field
- In the *Enforce class from high_value to* dropdown select **1.0**



- **Save** the changes then click **Train** .

The Result page for the sessions opens up. Here, you can monitor the optimization results of the models for which optimization results are available.

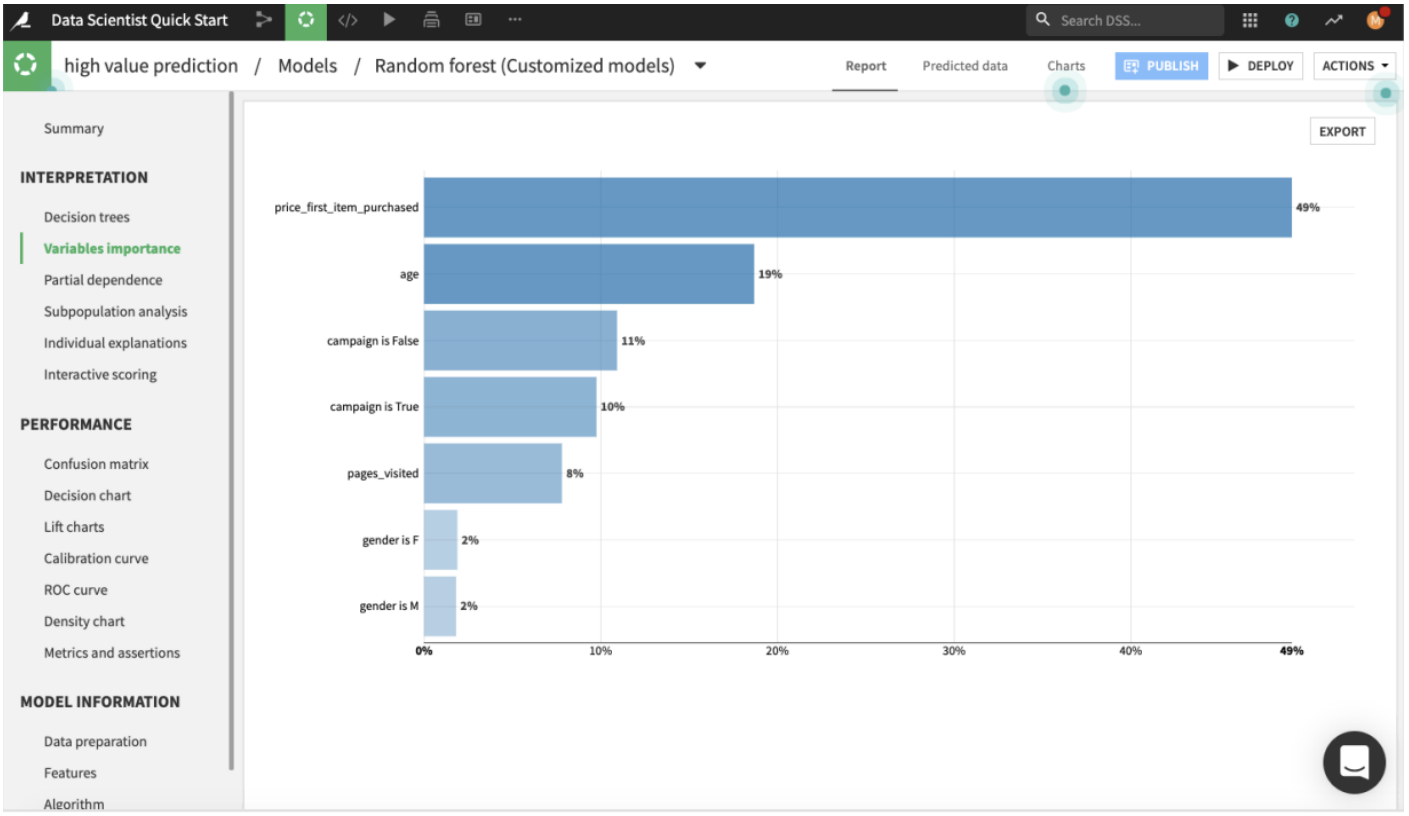
After training the models, the Result page shows the AUC metric for each trained model in this training session, thereby, allowing you to compare performance side by side.

Every time you train models, the training sessions are saved and listed on the Result page so that you can access the design details and results of the models in each session.

Deploy the ML Model to the Flow

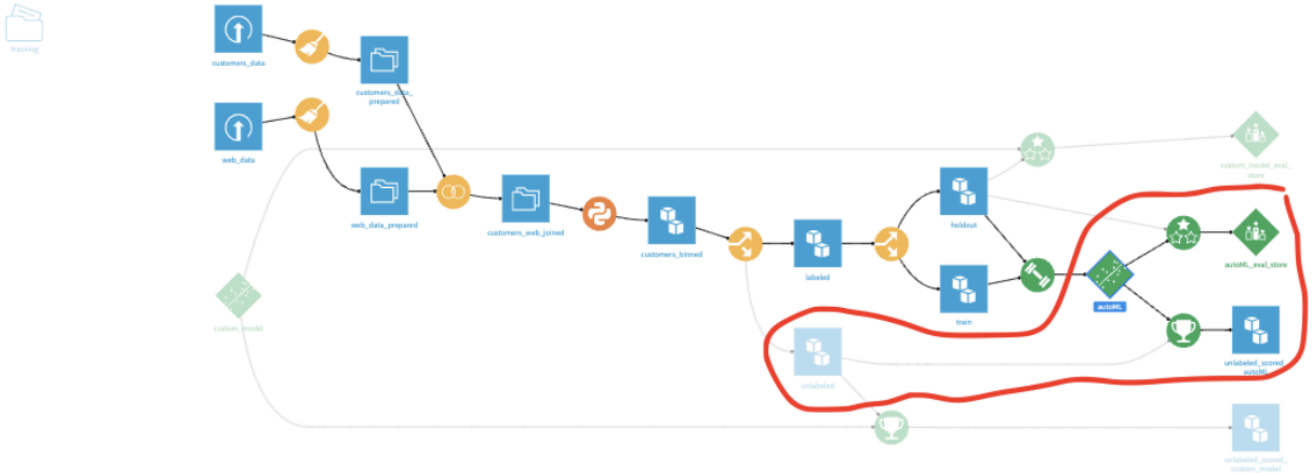
Next, we'll deploy a to the Flow to apply it to the test data and evaluate the test data predictions in light of the ground truth classes.

- Click **Random forest** from the list to go to the model's Report page.
- Explore the model report's content by clicking any of the items listed in the left panel of the report page, such as any of the model's interpretations, performance metrics, and model information. The following figure displays the "Variables importance" plot as well as feature effects if you scroll down, which shows the relative importance of the variables used in training the model. Model interpretation features like those found in the visual ML tool can help explain how predictions are made and ensure that fairness requirements around features (such as age and gender) have been met.



Let's say, overall, we are satisfied with the model. The next thing to do is deploy the model from the Lab to the Flow.

- Click **Deploy** from the top right corner of the page, and name the model ***autoML*** .
- Click **Create** . Back in the Flow, you can see that two new objects have been added to the Flow. The green diamond represents the Random Forest model. You can use this newly created saved model with Score and Evaluate recipes to build a model scoring and evaluation pipeline very similar to the one we built with the custom_model.



From there, you could define other execution scenarios to the models and evaluate the new model as well.

[Optional] Lab 8 - Metrics & checks

Adding metrics and checks to the dataset

It is crucial to control data quality and perform checks prior to running the models.

In Dataiku, we can implement various metrics on datasets using pre-build functionatilies or custom python/SQL -probes.

We will work with the customers_web_joined and implement the following metrics and checks:

Double-click to open the dataset and next select the **Status** tab.

Inside you will see default metrics: record couns and column counts. But we can add more metrics to it.

Click on **Edit** to open metrics and check editor.

From the predefined metrics we will select **Min** and **Max** on ***age*** column and **Count empty values** for the ***revenue*** column.

Next, we will create a custom python metric to compute:

Select **New Python probe** and name it: .

Next copy-paste this code:

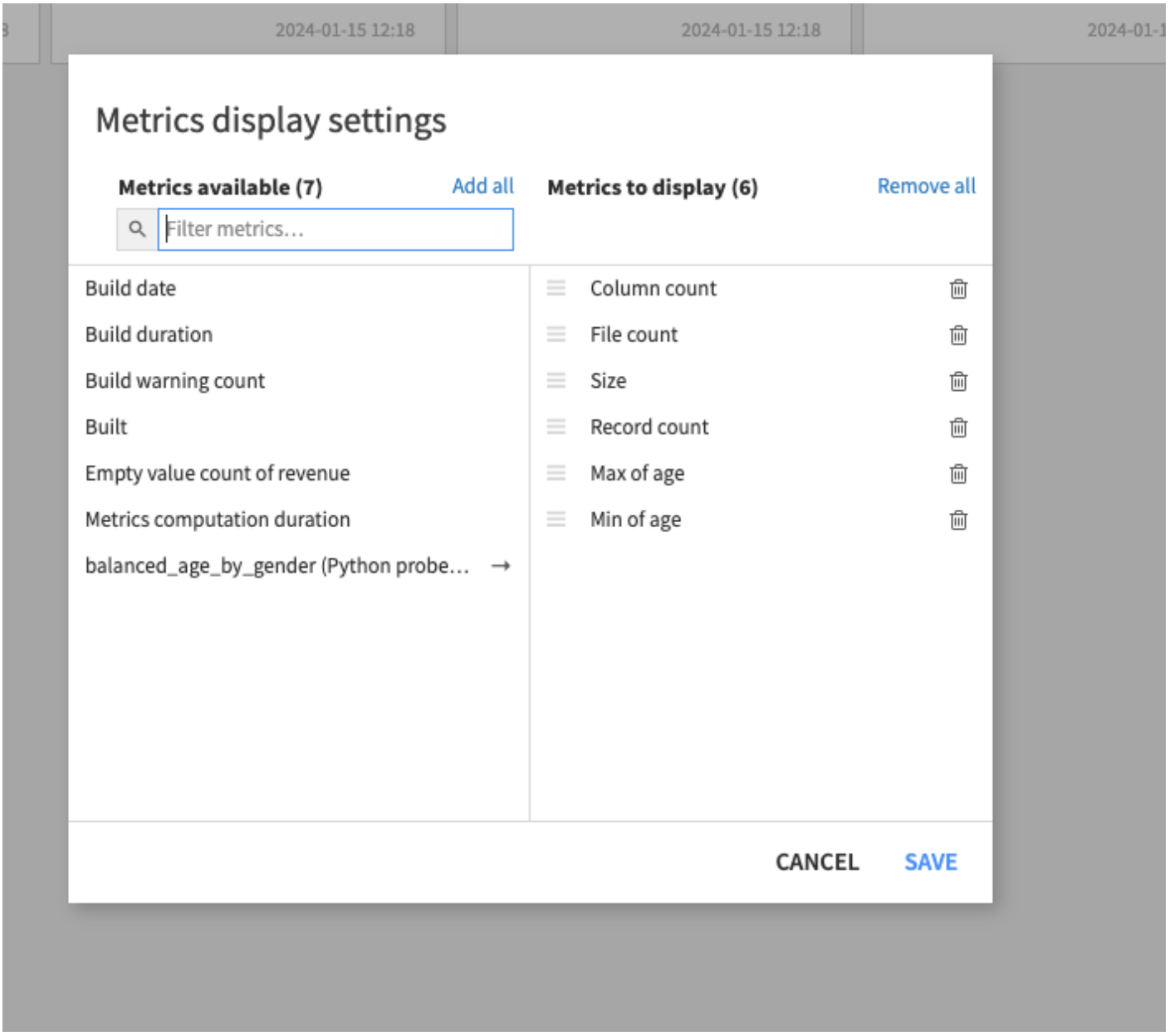
```
import numpy as np

def process(dataset, partition_id):

    # dataset is a dataiku.Dataset object
    df = dataset.get_dataframe()
    age_by_gender = df.groupby("gender").mean()["age"].reset_index()
    balanced_age = np.abs(age_by_gender["age"][0]-age_by_gender["age"][1])< 0.2
    return {'balanced_age_by_gender' : str(balanced_age)}
```

Next, test the metric by running **Click to run this now**

Do not forget to **Save** your new metrics once they have been created.

Return to the Metrics window, click on metrics list and select all new metrics from the list.

You should now be able to see it in the computed metrics.



!

Note: the count of empty values on the "Revenue" metric is different from the number of empty values that you can observe through the **Analyse**. This is because you only look at the Sample of a dataset through the analyse window while metrics are computed on the ***entire dataset***.

metrics_output.png

1. Let's create some **Checks** based on the new metric. Checks allow to control the Scenario flow: for example a scenario will stop if Check on a metric returns "False". Currently, there are no checks activated by default on this dataset.

Let's return to the **Edit** panel and select checks.

We will create two checks:

1. Check on the number of records in the dataset to ensure that it has at least 100 records.

2. Check on the range of min and max age values to ensure there are no outliers

To create a first check, select **Metric Value is in a numeric range**. Select ***record count*** as a metric and set a **Min** to 100. Click on **Check** to run the check. It should return "OK".

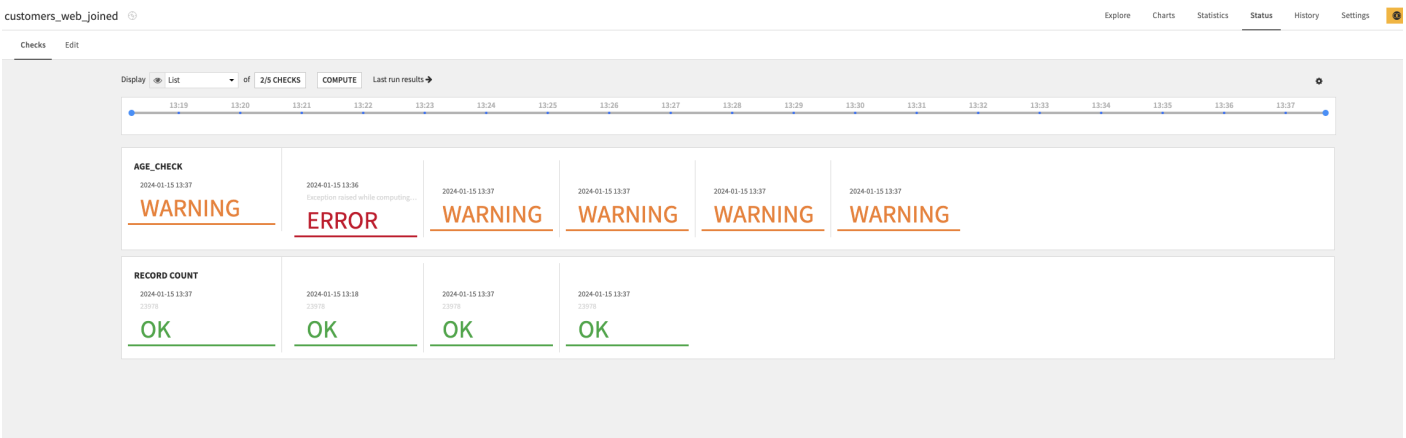
For the second check: open a **Custom python probe_** and paste the following code:

```
# Define here a function that returns the outcome of the check.
def process(last_values, dataset, partition_id):
    # last_values is a dict of the last values of the metrics,
    # with the values as a dataiku.metrics.MetricDataPoint.
    # dataset is a dataiku.Dataset object

    max_age = dataset.get_metric_history("Max of age")['lastValue']['value']
    min_age = dataset.get_metric_history("Min of age")['lastValue']['value']

    if (max_age < 100) and (min_age > 1):
        return 'OK'
    else:
        return 'WARNING'
```

Click on RUN to activate the check.

Next get back to the check list tab and select new checks to display.

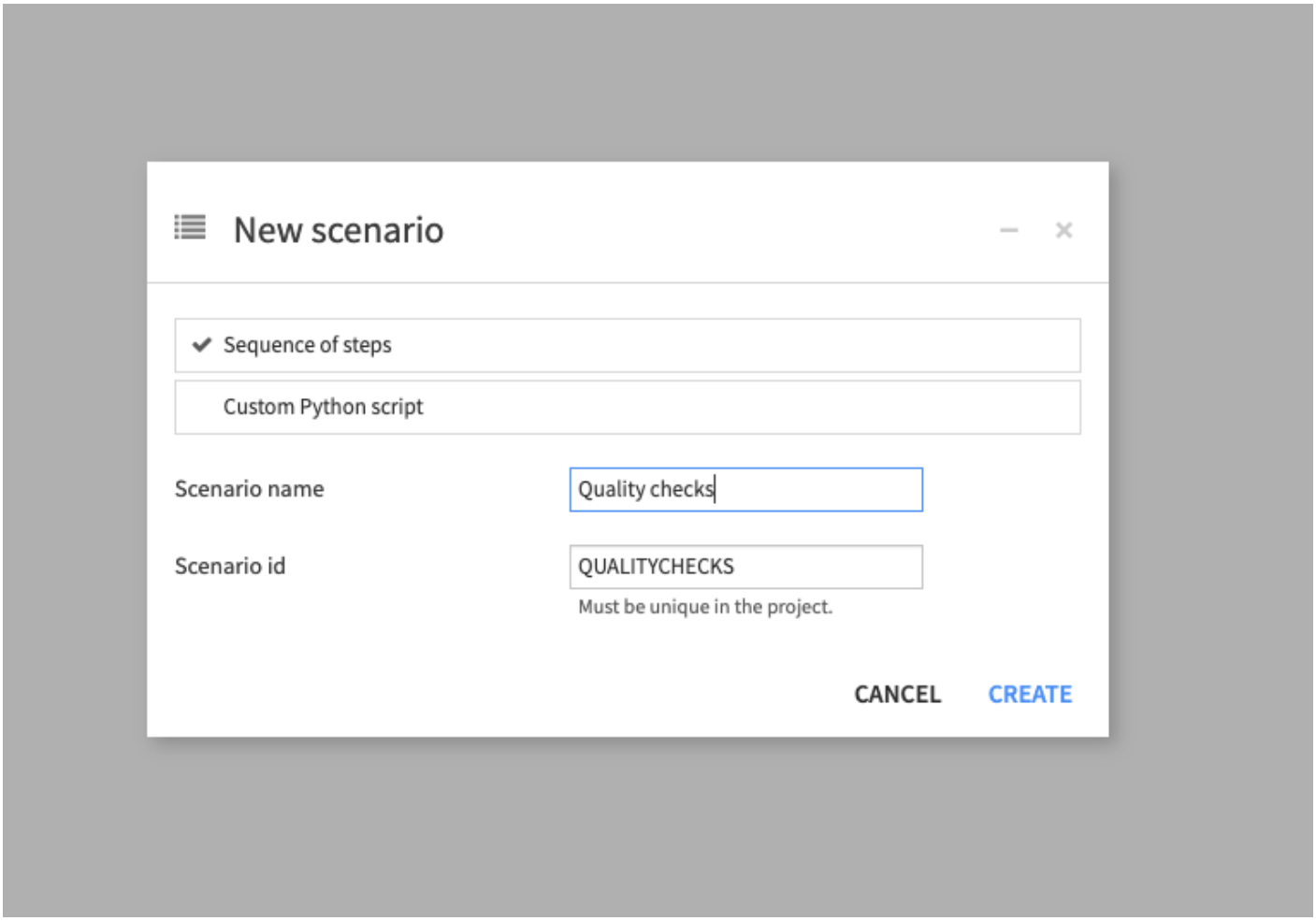
You should now be able to see the history of checks



1. **Adding metrics and checks to the scenario**
   So far we ran our metrics and checks manually by clickin on :COMPUTE. However, it is much more practical to automate these processes to ensure they run sequentially.

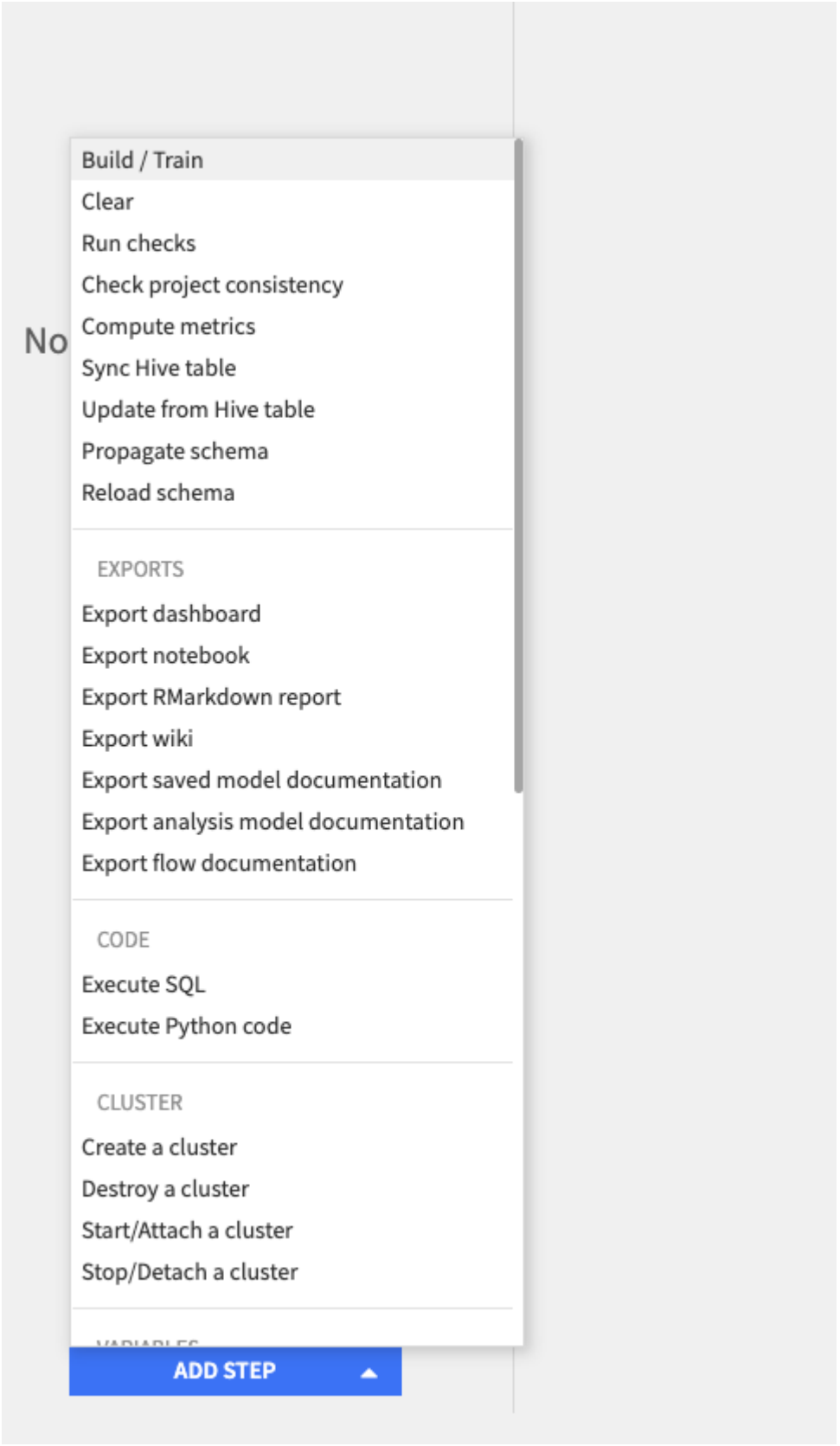
We will create a scenario and add metrics and checks computations as steps.

Go to the **Scenario** tab and click on **NEW SCENARIO**. Lets name it **QUALITY CHECK**

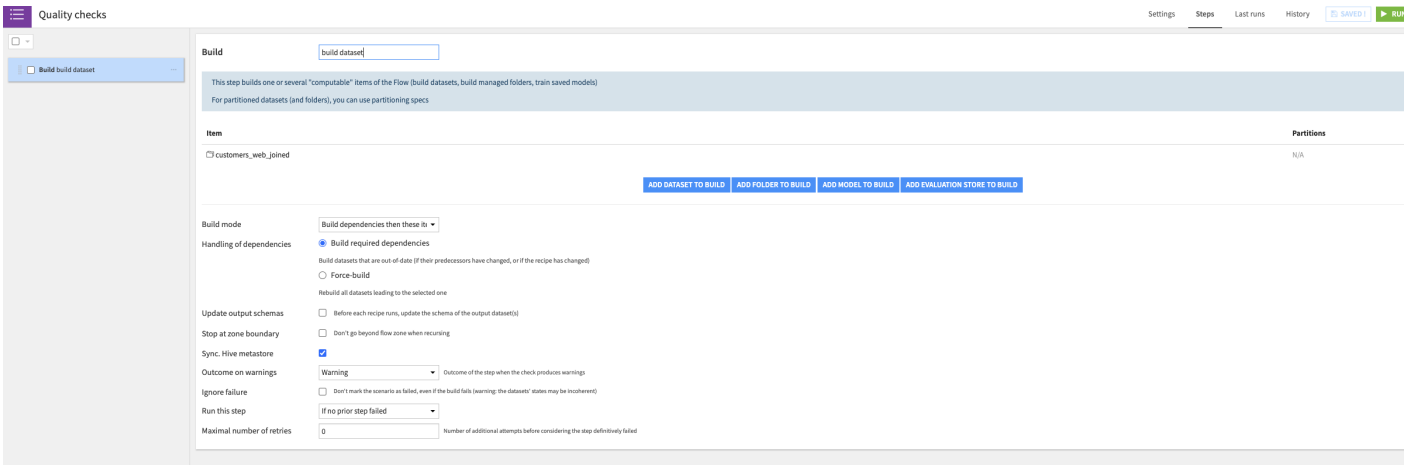


Inside the scenario, let's first build a customers_web_joined

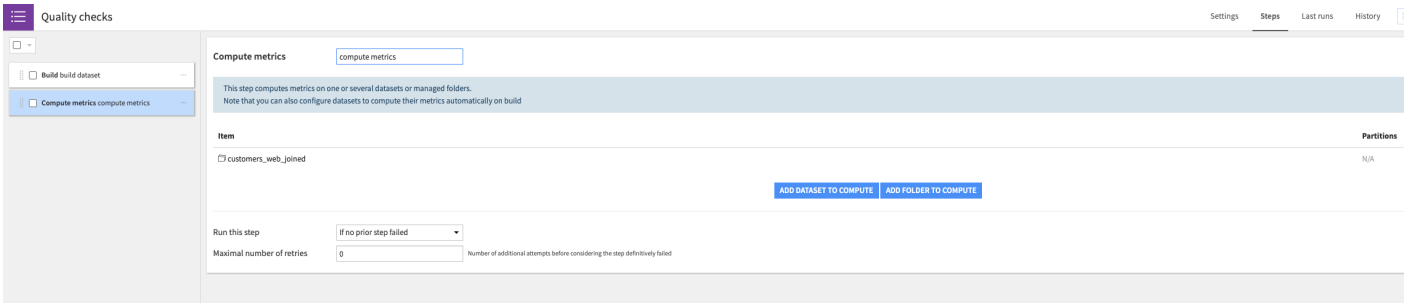
Select "ADD STEP" and choose "Build/Train". In the dropdown menu, select customers_web_joined as a dataset to build.



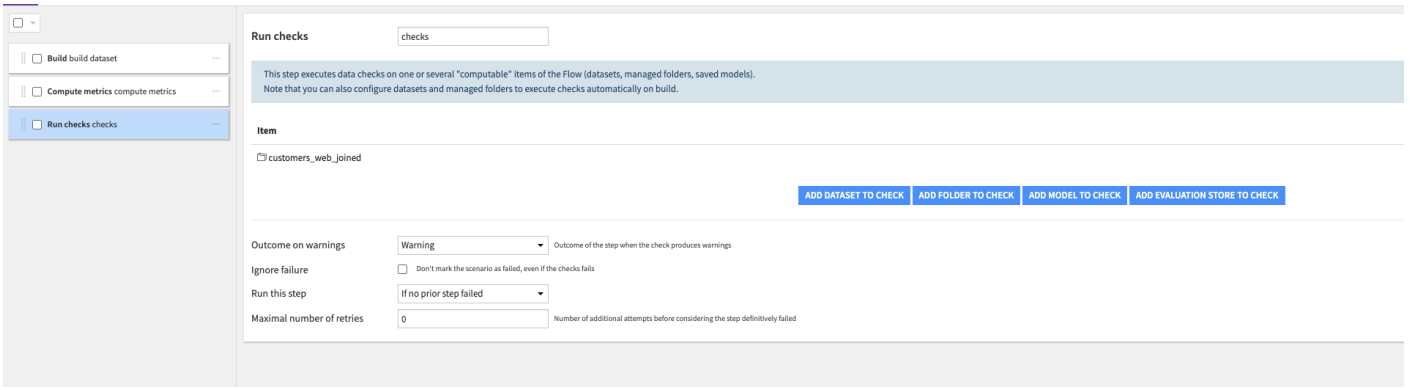
You should see a dataset listed:



Next, go back to "ADD STEP" and now choose " Compute metrics" as a step. Select the same customers_web_joined

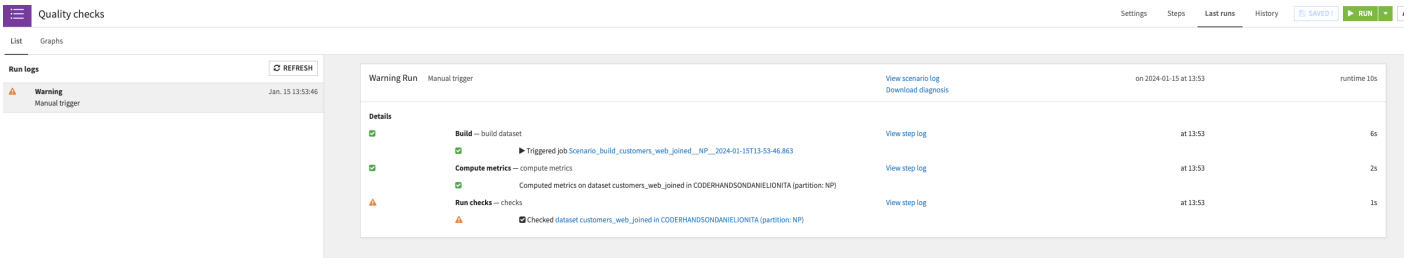


Finally, add a step with the checks: to do that, click on "ADD STEP" and select "Run checks". Again, select customers_web_joined as a dataset to compute the checks on.



Save your scenario and click on **RUN**

In the "Last run" tab you should see the outcome of your scenario returning a "WARNING" since one of our checks returns the warning



[Optional] Lab 9 - Unit tests

Unit tests for project libraries

Sometimes it is important to run unit tests on the code prior to pushing it to a remote repository.

You can implement unit tests in Dataiku

To run unit tests we will leverage `pytest==7.2.2` package.

It has to be installed in the code environment.

For this tutorial `pytest==7.2.2` package was added to the **mlflow** environment.

Example of unit tests implementation in Dataiku could be found here

While your Flow is now operational, you might want to ensure that your code meets expected behavior and errors can be caught earlier in the development process. To address those issues, we are going to write unit tests for the

functions used in the recipe_from_notebook_Feature_Engineering recipe.

In a nutshell, unit tests are assertions where you verify that atomic parts of your source code operate correctly. In our case, we'll focus on testing data transformations by submitting sample input values to the function for which we know the outcome, and comparing the result of the function with that outcome.

First, we will create a unit test for `bin_value` function

Next we will create a scenario which will first implement the unit tests and then build customers_binned using recipe_from_notebook_Feature_Engineering recipe.

We will start by creating a `unit_tests` folder under `python` folder in the **Project LIbraries***

Within this folder we will create two files:

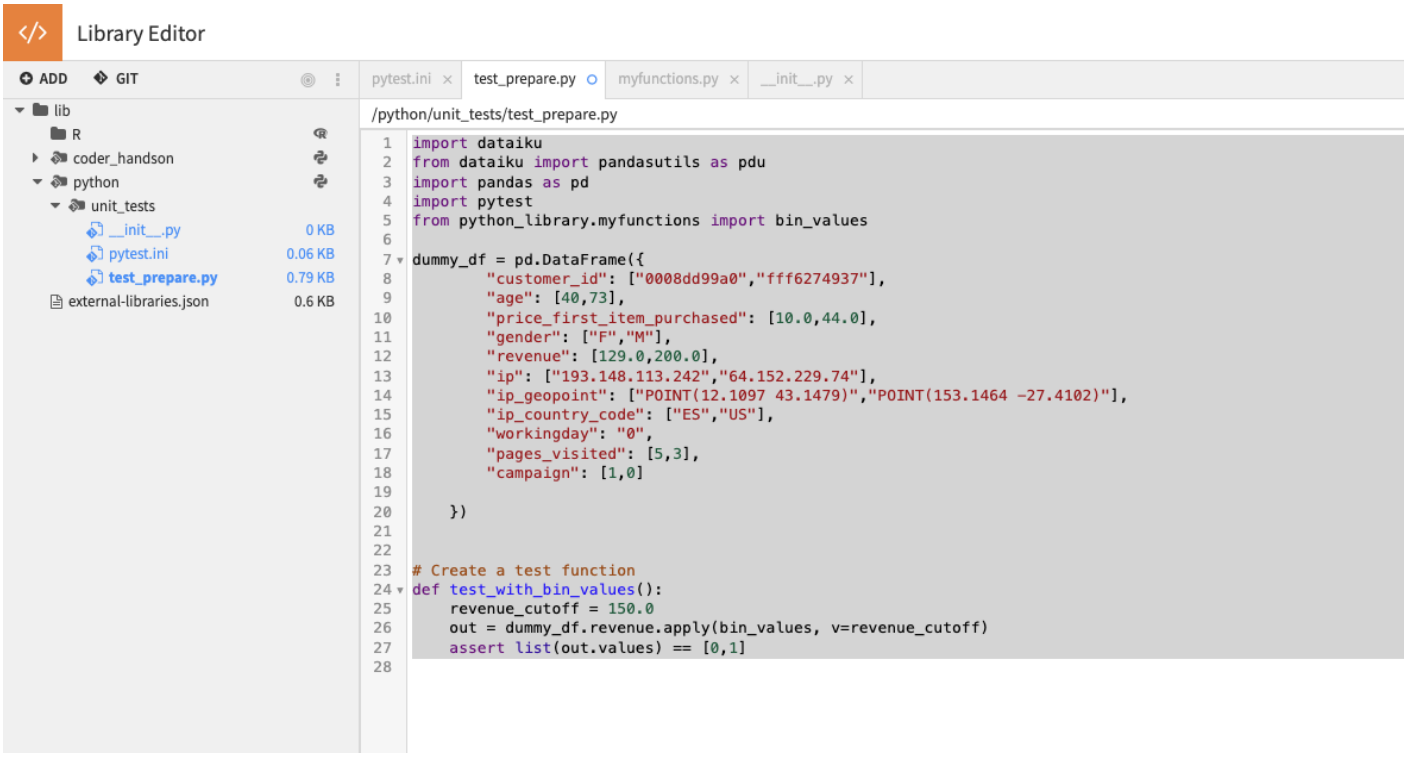
1. Empty `__init__.py` file
2. `pytest.ini` file to implement setting files to silence the warnigs:

```
[pytest]
filterwarnings =
ignore::DeprecationWarning
```

1. Next we will create `test_prepare.py` file where we will implement our unit test on the dummy dataset: We will check whether the output of the binned values is as expected.

```
import dataiku
from dataiku import pandasutils as pdu
import pandas as pd
import pytest
from python_library.myfunctions import bin_values

dummy_df = pd.DataFrame({
      "customer_id": ["0008dd99a0","fff6274937"],
      "age": [40,73],
      "price_first_item_purchased": [10.0,44.0],
      "gender": ["F","M"],
      "revenue": [129.0,200.0],
      "ip": ["193.148.113.242","64.152.229.74"],
      "ip_geopoint": ["POINT(12.1097 43.1479)","POINT(153.1464 -27.4102)"],
      "ip_country_code": ["ES","US"],
      "workingday": "0",
      "pages_visited": [5,3],
      "campaign": [1,0]

  })


def test_with_bin_values():
    revenue_cutoff = 150.0
    out = dummy_df.revenue.apply(bin_values, v=revenue_cutoff)
    assert list(out.values) == [0,1]
```

In Project Library, you code should be structured as follwes:

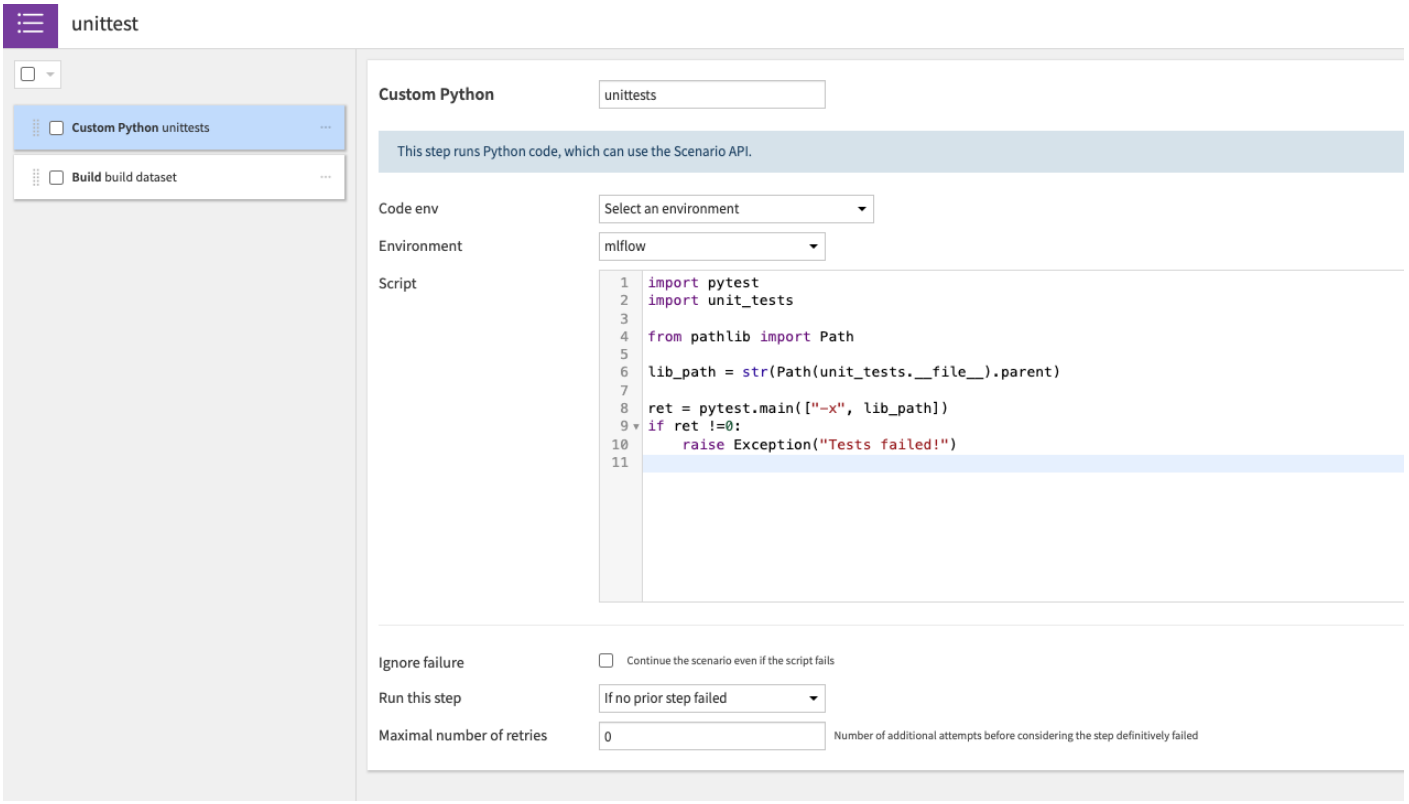


Next we will create a Scenario where the first step would be a custom python code where we will call our unit test first:

In the **Scenario** select new scenario and name it **UNITTEST**.

As a first step select **Execute Python Code**

As an environment select **mlflow** environment where we added **pytest** package.



Inside the step copy-paste the following code:

```
import pytest
import unit_tests

from pathlib import Path

lib_path = str(Path(unit_tests.__file__).parent)

ret = pytest.main(["-x", lib_path])
if ret !=0:
    raise Exception("Tests failed!")
```

As a second step select **ADD STEP** and choose ***Build/Train*** as a step. Select customers_binned

Save your scenario and hit **RUN**

What's Next?

Congratulations! In a short amount of time, you learned how Dataiku enables data scientists and coders through the use of:

- Python notebooks for EDA, creating code recipes, and building custom models;
- code libraries for code-reuse in code-based objects;
- experiment tracking and model evaluations;
- the visual ML tool for customizing and training ML models; and
- scenarios for workflow automation. You also learned about code environments and how to create reusable data products. To review your work, compare your project with the completed project in the Dataiku Gallery.
- Your project also does not have to stop here. Some ways to build upon this project are by:
- Documenting your workflow and results in a wiki;
- Creating webapps and Dataiku applications to make assets reusable;
- Sharing output datasets with other Dataiku DSS projects or using plugins to export them to tools like Tableau, Power BI, or Qlik. Finally, this quick start is only the starting point for the capabilities of Dataiku DSS. To learn more, please visit the Dataiku Academy, where you can find more courses, learning paths and can complete certificate examinations to test your knowledge for FREE.

Create Reusable Data Products

This section briefly covers some extended capabilities Dataiku DSS provides to coders and data scientists.

Dataiku's code integration allows you to develop a broad range of custom components and to create reusable data products such as Dataiku applications, webapps, and plugins.

Dataiku Applications

Dataiku Applications are a kind of DSS customization that allows projects to be reused by colleagues who simply want to apply the existing project's workflow to new data without understanding the project's details.

Therefore, a data scientist can convert the project into a Dataiku application so that anyone who wants to use the application only has to create their own instance of the Dataiku application. These applications can take one of two forms:

- Visual Applications that allow you to package a project with a GUI on top
- Applications-as-Recipes that allow you to package part of a Flow into a recipe usable in the Flows of other projects.

Webapps

You can extend the visualization capabilities of Dataiku DSS with interactive webapps. Dataiku DSS allows you to write three kinds of webapps:

- Standard webapps made at least of HTML, CSS, and Javascript code that you can write in the DSS webapp editor. DSS takes care of hosting your webapp and making it available to users with access to your data project.
- Shiny webapps that use the Shiny R library.
- Bokeh webapps that use the Bokeh Python library.
    - Dash webapps that use the Dash Python library. The Academy course on Webapps covers how to create interactive webapps in Dataiku DSS in detail.

Plugins

Plugins allow you to implement custom components that can be shared with others. Plugins can include components such as dataset connectors, notebook templates, recipes, processors, webapps, machine learning algorithms, and so on.

To develop a plugin, you program the backend using a language like Python or R. Then, you create the user interface by configuring parameters in .json files.

Plugins can be installed from the Dataiku Plugin store (in your Dataiku DSS instance) by uploading a Zip file or downloading from a Git repository.