



**data  
iku**

©2018 dataiku, Inc. | [dataiku.com](http://dataiku.com) | [contact@dataiku.com](mailto:contact@dataiku.com) | [@dataiku](https://www.twitter.com/@dataiku)

## Hands on! Exercise One

---



## Predictive Maintenance - Car Rentals

### Business Case



- Breakdowns are costly!
  - Repairs
  - Unavailability
  - Customer dissatisfaction.
- Our Goal:  
Replace those cars that are mostly likely to breakdown **before** the problem occurs



## Predictive Maintenance - Car Rentals

### Business Case



To predict vehicle failures, we will build an end-to-end predictive model yielding insights on:

- Common factors behind failures
- Which cars will be most likely to fail



## Supporting Data - Three Datasets

- usage:  
number of miles the cars have been driven, collected at various points
- maintenance:  
service records, date, which parts were serviced, the reason for service, and the quantity of parts replaced during maintenance
- failure:  
whether a vehicle had a recorded failure (not all cases are labelled)



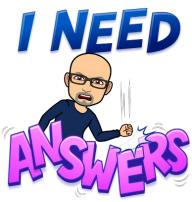
## Understanding our Problem

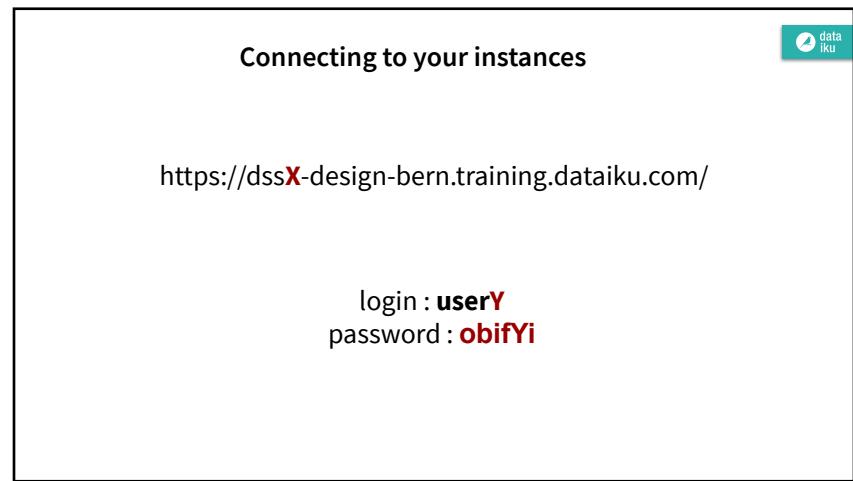
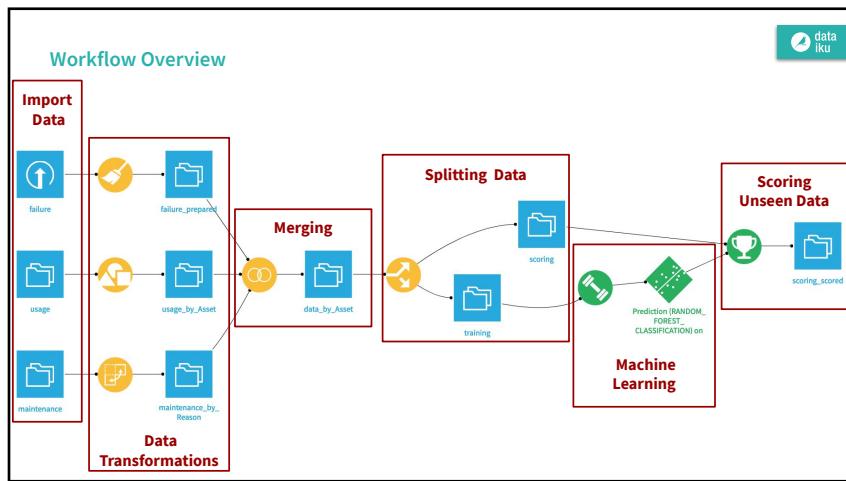
### The “failure” Dataset

Asset	failure_bin
string	bigint
Text	Integer
A003146	0
A046288	0
A082667	
A115725	1
A148200	

- What values are in the column “failure\_bin”?
- What do they represent?
- What type of ML problem is this?
- Specifically, what type of class of ML problem is this?

Asset ID, available in each file, uniquely identifies each car





**ToDo: Create a New Project**

+ NEW PROJECT

Add a unique identifier (initials) to your project

**+ New project**

Name: Rentals Pred Maintenance **JN**

Project Key: RENTALSPREDMAINTENANCE\_JN

The project key is used to reference datasets between projects. It cannot be changed once the project is created.

CANCEL CREATE

**Project Homepage**

Rentals Pred Maintenance\_JN

The project *Rentals Pred Maintenance\_JN* was created by JayN on Dec 16th 2019

Click to add tags  
Sandbox

**Flow**

0 DATASETS	0 RECIPES	0 NOTEBOOKS	1 DASHBOARD	0 ARTICLES	3 TASKS
0 DATASETS	0 RECIPES	0 NOTEBOOKS	1 DASHBOARD	0 ARTICLES	3 TASKS

+ IMPORT YOUR FIRST DATASET

- This is a **Project**
- Landing page
- Overview of assets & objects
- G+F hotkeys to the **Flow**

**Import the “maintenance” Dataset**  
**Show them how to get to the data**

Files

Upload your files  
Server's Filesystem  
Files in folder

New Server's Filesystem dataset

Files Format / Preview Schema Partitioning Advanced

Read from filesystem\_root

Path /data/shared/cars BROWSE...

Show Advanced options

root / data / shared / cars

TEST

Used /maintenance\_failure, Used format csv and found 2 columns

PREVIEW >

CANCEL OK

**Import the “maintenance” Dataset**

Rentals Pred Maintenance\_JN > Datasets Search DSS

New Uploaded Files dataset

New dataset name: failure CREATE

Files Format / Preview Schema Partitioning Advanced

/maintenance\_failure.csv.gz T.38 KB

Drag and drop your files here or ADD A FILE

Creates a single dataset: multiple files must have the same schema.

Store on Default (in DSS data dir)

Show Advanced options

TEST

Used ./maintenance\_failure.csv.gz (7.38 KB) to parse data  
Used format csv and found 2 columns

PREVIEW >

### Three Datasets

The screenshot shows a data management interface with a purple header bar containing icons for search, filter, and navigation, followed by the word "Flow". Below the header is a toolbar with a magnifying glass icon, a dropdown menu, and buttons for "+ RECIPE" and "+ DATASET". A top right corner features the "data iku" logo. The main area displays three dataset cards:

- usage**: Represented by a blue folder icon.
- failure**: Represented by a blue folder icon with a circular arrow and upward arrow icon.
- maintenance**: Represented by a blue folder icon.

### The “usage” dataset

The screenshot shows a dataset sample view for the "usage" dataset. At the top, it says "Viewing dataset sample" and "Configure sample". It indicates there are "10000 rows, 3 cols". The table has three columns: "Asset", "Time", and "Use". The "Asset" column contains string values like "A403193". The "Time" column contains integer values like "5", "17", "56", "124", and "144". The "Use" column contains decimal values like "31194.652034284285", "31223.536354503929", "31362.706261016774", "31701.440561774449", and "31724.489665788107". Red arrows point from the text descriptions to specific parts of the table.

Asset	Time	Use
A403193	5	31194.652034284285
A403193	17	31223.536354503929
A403193	56	31362.706261016774
A403193	124	31701.440561774449
A403193	144	31724.489665788107

The identifier ID for the asset (car)  
Multiple observations per “Asset”

The total number of miles a car  
has driven at the specified “Time”

Unclear reference to “Time”

## Working the “usage” Dataset

- For most individual cars, we have many *Use* readings (rows) at many different times.
- However, we want the data to reflect the individual car so that we can model outcomes at the vehicle-level.
- How would you normally collapse data with to a level of singular (vehicle) granularity?



## Group By

### Collapse on Asset

- Use a visual recipe to collapse rows based on an aggregation - GroupBy
- But first convert the stored data types to formats that are conducive to aggregation type transformations

usage

Viewing dataset sample Configure sample

10000 rows, 3 cols

Asset	Time	Use
string	string	string
Text	integer	decimal
A403193	5	31194.652034284285

Asset Time Use

Asset	Time	Use
string	bigint	double
Text	integer	Decimal
A403193	5	31194.652034284285
A403193	17	31223.53635450393
A403193	56	31362.706261016774



Step 1:  
Navigate to “Settings”

Step 2:  
Select “Schema”

Step 3:  
Change  
“Time” to bigint and  
“Use” to double

## Group By

### To Do: Collapse on Asset

Schema corrected! ... Let's proceed with the GroupBy:

1. From the **usage** dataset, initiate a **Group By** recipe from the Actions menu.
2. Choose to **Group By Asset** in the dropdown menu.
3. Keep the default output dataset name **usage\_by\_Asset**.
4. In the **Group** step, we want the count for each group (selected by default). Add to this the **Min** and **Max** for both **Time** and **Use**.
5. Run the recipe, updating the schema to six output columns.

**compute\_usage\_by\_Asset**

**Group Keys**  
Create a group for each unique combination of these variables  
Asset string @  
Select key to add  or [create a new computed column](#)  
 Compute count for each group

**Per field aggregations** Select  to be computed for each field  
0 / 2   Hide unused variables

Time	bigint	Distinct	Min	Max	Avg	Sum	Concat	Std. dev.	Count	First	Last
Use	double	Distinct	Min	Max	Avg	Sum	Concat	Std. dev.	Count	First	Last

Pre-filter  
Computed columns  
Custom aggregations

We have aggregated at the level of a unique rental car

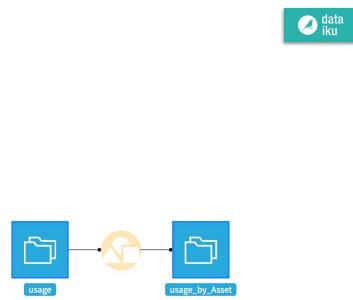
**usage\_by\_Asset**

**Viewing dataset sample** [Configure sample](#)

1894 rows, 6 cols

Asset	Time_min	Time_max	Use_min	Use_max	count
A000204	46	722	31449.65098248601	33212.76155912566	15
A000270	435	617	26378.55633867107	27036.768726706654	4
A000463	30	616	30451.537250451554	31894.002696572094	18
A000495	6	695	30851.249560561817	32540.2528913854	16

## Your Flow (G+F) Should Look Similar



## The “maintenance” Dataset

A screenshot of a data visualization interface showing a dataset sample. The table has columns: Asset (Text, string), Time (string, Integer), Reason (Text, string), Part (Text, string), and Quantity (string, Integer). The data rows are:

Asset	Time	Reason	Part	Quantity
A311482	0	R417	P361646	1
A174613	0	R707	P991287	1
A174613	0	R707	P169319	1

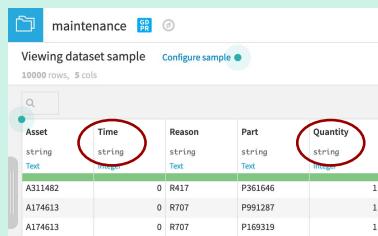
The **maintenance** dataset documents activity that has occurred with respect to:

- a given *Asset*
- organized by *Part* (what was repaired)
- *Time* (when it was repaired).
- A *Reason* variable codifies the nature of the problem.

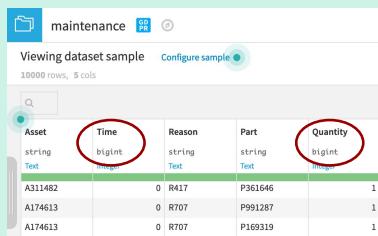
### **ToDo: Adjust the schema on the “maintenance” Dataset**

As we did before, set the schema to have:

- “Time” as bigint
- “Quantity” as a bigint



Asset	Time	Reason	Part	Quantity
A311482	0 R417	P361646		1
A174613	0 R707	P991287		1
A174613	0 R707	P169319		1



Asset	Time	Reason	Part	Quantity
A311482	0 R417	P361646		1
A174613	0 R707	P991287		1
A174613	0 R707	P169319		1

### **Preparing the “maintenance” Dataset**

#### **Introducing the Pivot recipe**

As before, we want to organize the *maintenance* dataset to the level of unique vehicles.

Previously, we used the **Group** recipe. Here, we'll use the **Pivot** recipe.

- The current dataset has many observations for each vehicle
- We need the output dataset to be “**pivoted**” at the level of each vehicle that is:  
**Transformed from Narrow to Wide.**

## Preparing the “maintenance” Dataset

Introducing the **Pivot** recipe .. where we are going:

Narrow → Wide

Asset	Time	Reason	Part	Quantity
string Text	bigint integer	string Text	string Text	bigint integer
A311482	0	R417	P361646	1
A174613	0	R707	P991287	1
A174613	0	R707	P169319	1
A060723	0	R193	P184448	0
A174613	0	R707	P097048	1
A174613	0	R707	P169319	1
A174613	0	R707	P610186	1
A174613	0	R707	P168211	1

Asset	R193_Quantity_sum	R364_Quantity_sum	R446_Quantity_sum	R565_Quantity_sum	R707_Quantity_sum
string Text	bigint integer	bigint integer	bigint integer	bigint integer	bigint integer
A000204	88	3	3	106	194
A000270	96	1	5	145	229
A000463	123	13	39	394	108
A001201	7				87
A001115	5	1	14	148	148
A002431		2	2		96
A002551	16	0	5	300	342

## Pivoting “maintenance”

Use the **Pivot** recipe to restructure the “maintenance” dataset at the level of each vehicle.

- With *maintenance* chosen as the input dataset, choose to **Pivot By Reason**.

New pivot recipe

Input dataset	Output dataset
maintenance DATASET - View	Name maintenance_by_Reason
Pivot By	Store into filesystem_managed
Reason	Format CSV

NEW DATASET | USE EXISTING DATASET

**compute\_maintenance\_by\_Reason**

Pre-filter

Computed columns

Pivot

Other columns

Output

Simple count

Pivot table

Pivot values

Frequency table

Various statistics

Reason

Column: year

Pivoted values: most frequent

Count of records

Count of records

Asset

Add new column

CUSTOM AGGREGATES...

RUN DSS OG

### Pivoting “maintenance”

- At the Pivot step, select **Asset** as the row identifier.

**Row identifiers**

**Asset**

Add new column

explicit list

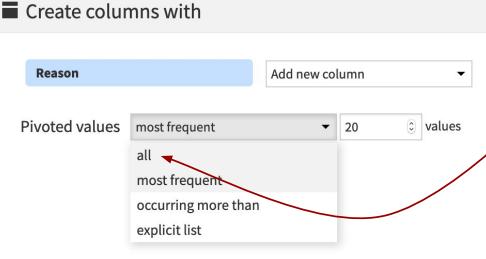
### Pivoting “maintenance”

- Reason should already be selected under **Create columns with**.
- Although it should make no difference in this case, change **Pivoted values** to **all** so that all values of **Reason** are pivoted into columns.

**Create columns with**

Reason Add new column ▾

Pivoted values most frequent ▾ 20 values  
all  
most frequent  
occurring more than explicit list



### Pivoting “maintenance”

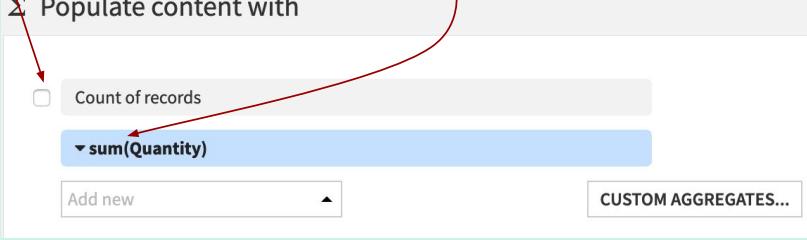
- Populate content with the **sum of Quantity**.
- Deselect Count of records.

**Σ Populate content with**

Count of records

▼ **sum(Quantity)**

Add new ▾ CUSTOM AGGREGATES...



**compute\_maintenance\_by\_Reason**

Pre-filter

Pivot

Output

Simple count

Settings

Column: year

Pivoted values all

Count of records

Asset

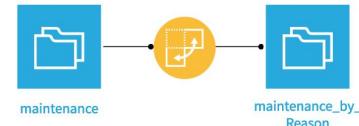
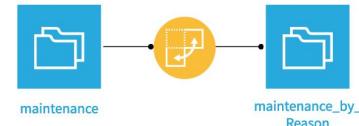
Count of records

sum(Quality)

CUSTOM AGGREGATES...

RUN DSS

## Your Flow (G+F) Should Look Similar



## An Short Introduction to Prepare Recipe Modify “failure” Dataset

New data preparation recipe

Input dataset	Output dataset
failure <small>DATASET · View</small>	Name failure_prepared
	Store into filesystem_managed
	Format CSV

NEW DATASET | USE EXISTING DATASET

## An Short Introduction to Prepare Recipe Modify “failure” Dataset

+ ADD A NEW STEP

An Short Introduction to Prepare Recipe

### Modify “failure” Dataset

Rename column 'failure\_bin' to 'failed'

1819

Renamings

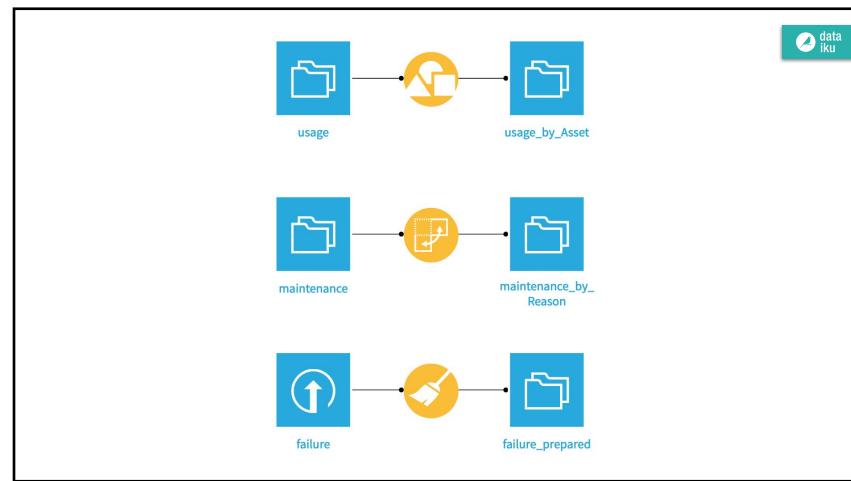
failure\_bin → failed

ADD ANOTHER

Raw text edit

X<sup>2</sup>

This screenshot shows a step in the Dataiku Prepare interface titled "Modify ‘failure’ Dataset". It displays a "Renamings" section where a column named "failure\_bin" is being renamed to "failed". The interface includes a counter "1819", a toolbar with various icons, and a "Raw text edit" option.



## Merging Data

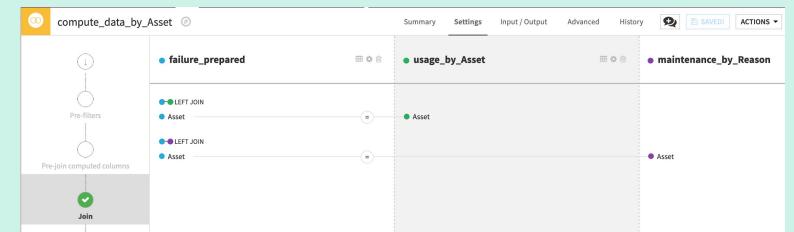
### Introducing the “Join” recipe

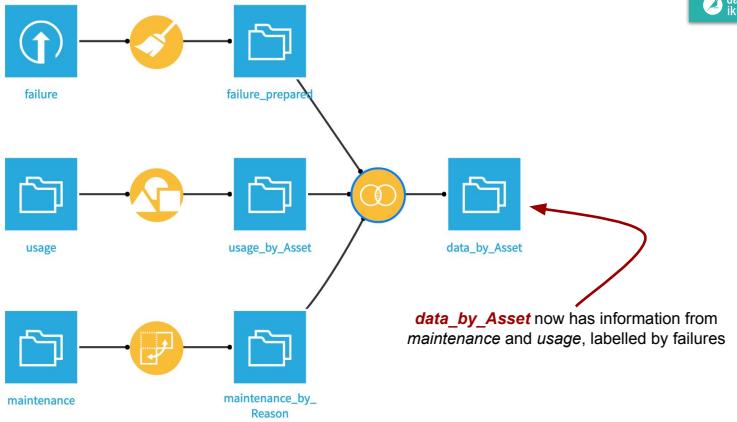
- Now have three datasets at the same level of granularity: the *Asset*, i.e. an individual rental car.
- Joining them together will give us the most possible information for a model.
- The Asset ID can serve as the common component for the joins.



### ToDo: Join all on Asset

- From the *failure\_prepared* dataset, initiate a Join recipe.
- Add *usage\_by\_Asset* as the second input dataset.
- Name the output *data\_by\_Asset*. Click **Create Recipe**.
- Add *maintenance\_by\_Reason* as the third input dataset.
- Both joins should be **Left Joins**. *Asset* should be the joining key in all cases.
- Run the recipe and update the schema to 21 columns.





### Splits

## Working toward a Training And to-Score Datasets

To train models, we'll use the **Split** recipe to create two separate datasets from the merged dataset, *data\_by\_Asset*.

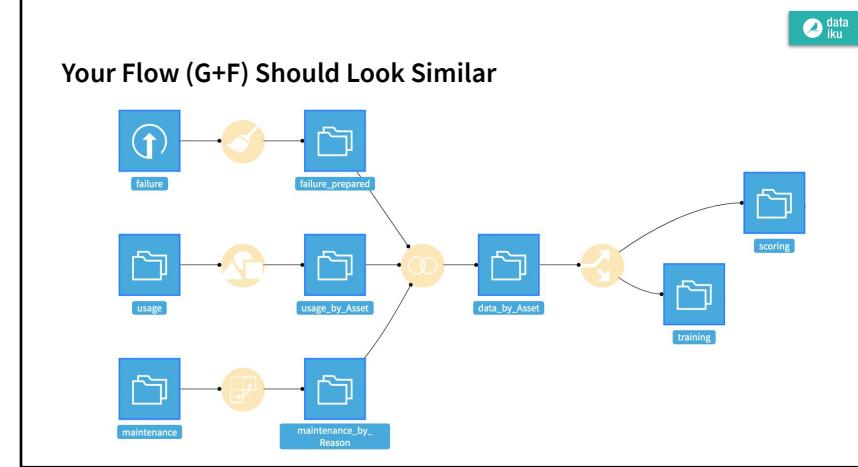
- a *training* dataset will contain labels for whether or not there was a failure event on an asset (car). We'll use it to train a predictive model.
- a *scoring* dataset will contain no data on failures, i.e. unlabelled. We will use it to predict whether or not these assets have a high probability of failure.

## ToDo - Split to Create training and scoring Datasets

- From the **data\_by\_Asset** recipe, initiate a **Split** recipe.
- Add two output datasets, named **training** and **scoring**, selecting **Create Dataset** each time. Then **Create Recipe**.
- At the **Splitting** step, choose to **Map values of a single column**. Then choose **failure\_bin** as the column on which to split.
- Assign values of **0** and **1** to the **training** set, and all "Other values" to the **scoring** set.
- Run the recipe.**

The screenshot shows the 'New Split recipe' dialog and the configuration for the 'split\_data\_by\_Asset' step. In the dialog, an 'Input dataset' is selected and two outputs are defined: 'training (Managed)' and 'scoring (Managed)'. The configuration window shows the 'split\_data\_by\_Asset' step with the 'failure\_bin' column mapped to two output datasets: 'failure\_prepared' (containing 'failure' and 'usage') and 'data\_by\_Asset' (containing 'usage\_by\_Asset' and 'maintenance').

## Your Flow (G+F) Should Look Similar



## Feature Generation

- Before making our first model on the *training* dataset, let's create a few more features that may be useful in predicting failure outcomes.
- Because we are still designing this workflow, we'll create a sandbox environment that won't create an output dataset, yet.
- By going into the **Lab**, we can test out such transformations as well as try out some modeling strategies, plus much more. Nothing is added back to the Flow until we are done testing and ready to deploy!

## Working in Lab Mode

1. With the **training** dataset selected, find the **Lab** in the Actions menu or in the right-click menu.
2. Under the **Visual Analysis** side, select **New** and accept the default name **Analyze training**.

The screenshot shows the dataiku platform interface. On the left, there is a sidebar with a teal header containing the dataiku logo. Below it, the title "Lab for 'training'" is displayed. The main area is titled "Visual analysis" with the subtitle "Workspace for interactive preparation and machine learning". A text input field says "Name your analysis" followed by "Analyze training" and a "CREATE" button. To the right, there is a section titled "Code Notebooks" with the subtitle "Analyze your dataset through interactive coding environments". It shows two options: "NEW" (with icons for Python, R, SQL, and Scala) and "PREDEFINED" (Audit, PCA, Correlations, ...). Below these sections, a message says "No notebook yet" and "You can create a new notebook or use one of the predefined analyses." The overall background of the interface is light green.

## Working in Lab Mode

1. In the screen which looks similar to a **Prepare** recipe, **create two new variables** with the **formula processor**

- **distance** from the expression:

Use\_max - Use\_min

- **time\_in\_service** from the expression:

Time\_max - Time\_min

The screenshot shows the Dataiku Analyze training interface. A single step script is running, with 1624 rows and 21 columns. The step details show a 'Create column distance with formula Use\_max - Use\_min' step. The output column is named 'distance' and is set to type 'Decimal'. The expression is 'Use\_max - Use\_min'. Below the table, there is a preview of the first few rows of the 'distance' column.

## Working in Lab Mode

2. Use the **Fill empty cells with fixed value** processor to replace empty values with **0** in columns starting with the letter **R**

You can use the regular expression `^R.*_Quantity_sum$` to apply across multiple columns

3. To make the model results more interpretable, use the **Rename columns** processor according to the table below:

Old column name	New column name
count	times_measured
Time_min	age_initial
Time_max	age_last_known
Use_min	distance_initial
Use_max	distance_last_known

The screenshot shows the Dataiku Analyze training interface with a multi-step script. The steps include:

- Create column distance with formula Use\_max - Use\_min
- Create column time\_in\_service with formula Time\_max - Time\_min
- Fill empty cells of columns matching '^R.\*' with '0'
- Rename 5 columns

A green button at the bottom right says '+ ADD A NEW STEP'.

## Working in the Lab



### Note

- It is not necessary to deploy a script created in the Lab to the Flow in order to make use of the new features in the modeling process.
- Any models created in a Visual Analysis have access to any features created in the same Visual Analysis.

## Creating Models to Predict Car Breakdowns



From the open Lab Script, navigate to the **Models** tab.

Summary    Script    Charts    Models

No model yet

[CREATE FIRST MODEL](#)

### Choose your task



**Prediction**  
Choosing a model to predict the risk of purchase... understand the drivers of your business and predict what could happen next.



**Clustering**  
Used to identify homogeneous groups of items sharing the same behavior, or find anomalies in your data.

In this case, we are trying to determine whether or not a rental car will have problems. So, opt for a **Prediction** model.

## Creating Models to Predict Car Breakdowns

You can customize the model through either the option of **Automated Machine Learning** or **Expert Mode**.

**Automated Machine Learning** helps with some important decisions like choosing the type of algorithms and parameters of those algorithms.

- Select **Automated Machine Learning** and then **Quick Prototypes**, the default suggestions.

Dataiku DSS then asks us to select the target variable.

In this case, we want to calculate the probabilities for one of two outcomes: failure or non-failure, i.e. perform **two-class (binary) classification**.

Accordingly, choose **failed** as the target variable.

The screenshot shows the Dataiku DSS interface. In the first step, 'Choose your prediction style', the 'Automated Machine Learning' option is selected. In the second step, 'Create a modeling task', 'Quick Prototypes' is highlighted as the chosen method. This indicates the user has selected the 'Automated Machine Learning' path.

## Review the Design Tab of your Models

The screenshot shows the 'DESIGN' tab of a Dataiku DSS model. The 'Algorithms' section is set to 'Random Forest'. The 'Target' section shows 'Prediction type: Two-class classification' and 'Target: failed'. The 'PARTITION' section includes 'Partitioning' and 'K-fold'. The 'MODELLING' section includes 'Algorithms' (set to 'Hyperparameters'), 'ADVANCED' (set to 'None'), and a 'Probability calibration' chart. The chart shows two bars: one orange bar at 0% and one blue bar at 75%, with a note: 'If none classes are missing here (because they were 0 in the guess sample), you can manually edit the mapping (advanced)'.

## Train Some Models

**Training models**

2 models will be trained on 20 features using 5-fold cross-validation. An estimated total of 33 estimators will be evaluated.

**SESSION NAME & DESCRIPTION**

Name: Optional, appended to model names  
Description: Optional, set as model description

**TRAIN & TEST**

New extracts will be computed according to your settings

X CANCEL ► TRAIN

## Understanding the Model

- Model metrics can be found under the **Results** tab. Dataiku For example, we can compare how models performed against each other. By default, the AUC is graphed for each model.
- You can switch from the Sessions view to the **Table** view to see a side-by-side comparison of model performance across a number of metrics.
- By selecting a model, many additional insights and ready-made analysis are available. Examples: Confusion Matrix, ROC curves, Tree visualizations, Variable Importance ...

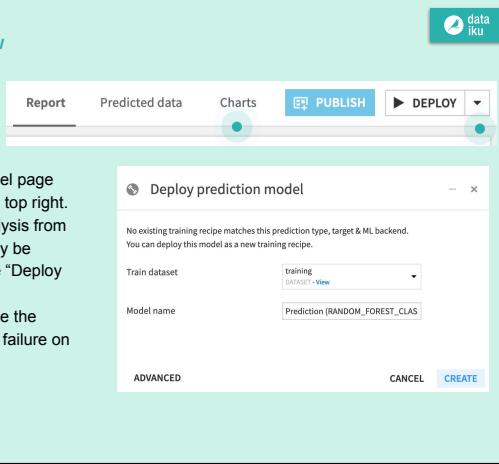
**INTERPRETATION**

	Predicted	Predicted	Total
Actuality: 1	57	20	77
Actuality: 0	69	179	248
Total	126	199	325

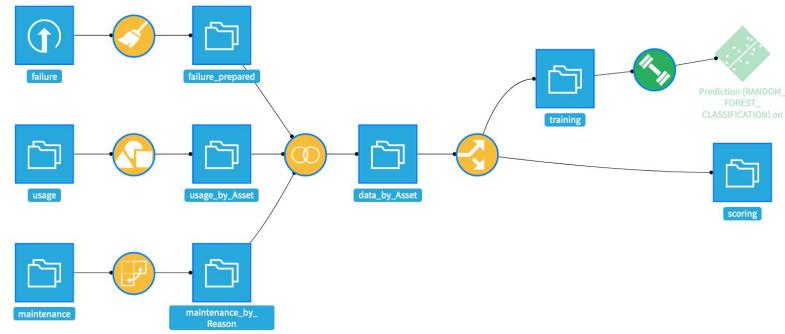
**PERFORMANCE**

### Deploy Top Performer to the Flow

1. From within the Lab, select the Model page and find the Deploy button near the top right.
2. Because we created this visual analysis from the *training* dataset, it should already be selected as the Train dataset in the "Deploy prediction model" window.
3. In the same window, you can change the default Model name to Prediction of failure on training data.
4. Click **Create**.

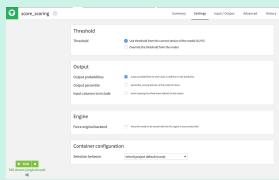


### Your Flow (G+F) Should Look Similar



## Scoring Unlabelled Data

1. In the Flow, select the model we just created. Initiate a **Score** recipe from the right sidebar.
2. Select **scoring** as the input dataset and the Prediction Model.
3. Name the output dataset **scoring\_scored**.
4. Create and run the recipe with the default settings.



**Score a dataset**

Input dataset: scoring  
Output dataset: scoring\_scored

Prediction Model: Prediction (RANDOM\_FOREST\_CLASSIFICATION) on:

Threshold: 0.5 (use model to calculate probability directly)

Output: Output probabilities (use model to calculate probability directly)

Engine: Python (using local Python environment)

Container configuration: Select container (empty)

**CANCEL** **CREATE RECIPE**

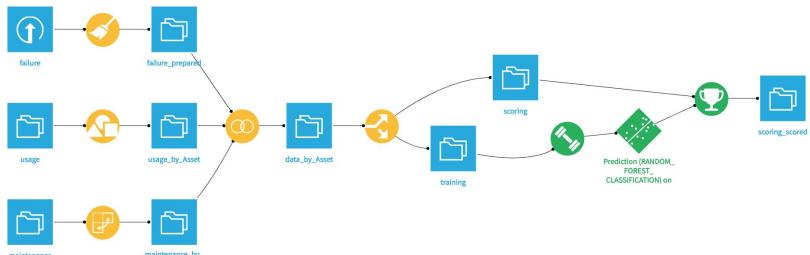
## Scoring Unlabelled Data

The resulting dataset now contains three new columns:

- **proba\_1**: probability of failure
- **proba\_0**: probability of non-failure ( $1 - \text{proba\_1}$ )
- **prediction**: model prediction of failure or not (based on probability threshold)

Asset	proba_0	proba_1	prediction
Type	double	double	string
A082667	0.8909481506146408	0.10905184938535918	0
A148200	0.7558993890413562	0.24410061095864374	0
A227156	0.8970859100650096	0.3029140899349933	1
A890132	0.6736371806324752	0.32636262193675247	1
A312402	0.67403636363413262	0.3259636636586738	1
A579152	0.43681765850688525	0.5631823414931147	1
A188236	0.793944636782007	0.2060555633217993	0
A423273	0.6705822971671204	0.32941770283287963	1
A559385	0.8495728914196041	0.1504271085803959	0
A962011	0.46030111119690388	0.539698880309612	1
A132820	0.9834362779950231	0.01656372204976893	0
A171983	0.9553451280786921	0.04465487192130791	0
A680806	0.3954259081995265	0.6045740918004735	1
A720349	0.6755970475938258	0.32440295240617417	1
A799993	0.5233451629925173	0.476654837074827	1
A534583	0.8462453915035186	0.15375460849648134	0

## Your Final Flow Should Look Like



## Remind Me ...



The goal here was to build an end-to-end data product to predict car failures from a workflow entirely in Dataiku DSS.

- We ingested, transformed, merged and split data.
- We engineered new features and
- Used Auto-ML to quickly prototype ML models
- Used the platform to interpret those models
- We scored unseen data!

This data product will help the company better identify car failures before they happen!

## Moving forward



Once we have a single working model built, we could try to go further to improve the accuracy of this predictive workflow, such as:

- Adding features to the model by combining information in datasets in more ways
- Trying different algorithms and hyper-parameter settings

To make the model more operational, we can package and deployed the models through a REST API, to be consumed in real time by external applications.

It is possible to do all of this using Dataiku DSS for an end-to-end deployment!