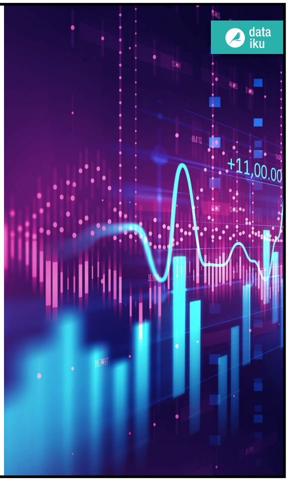
data
iku

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 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)

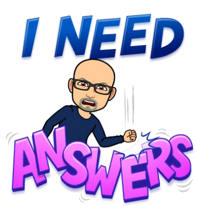
Understanding our Problem

The "**failure**" Dataset

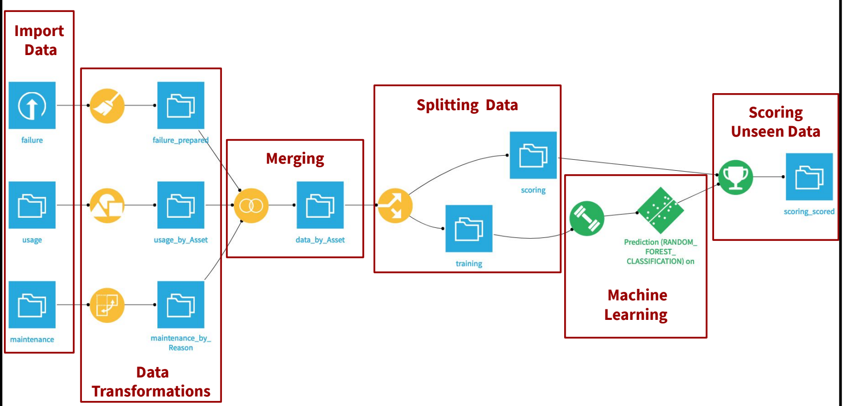
Asset	failure_bin
string	bigint
Text	Integer
A003146	0
A046288	0
A082667	
A115725	1
A148200	

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

- 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?



Workflow Overview

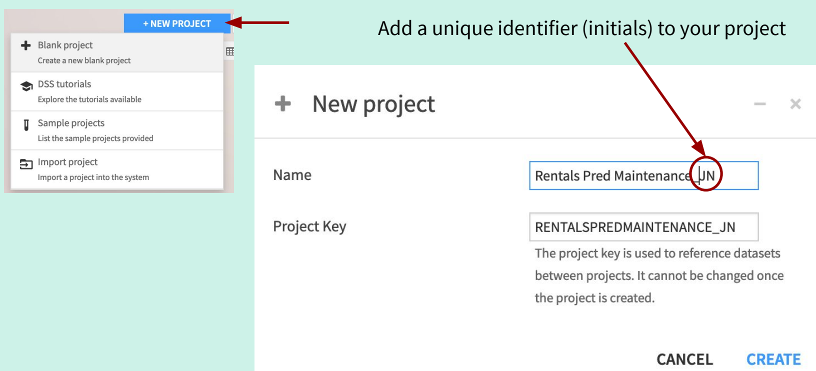


Connecting to your instances

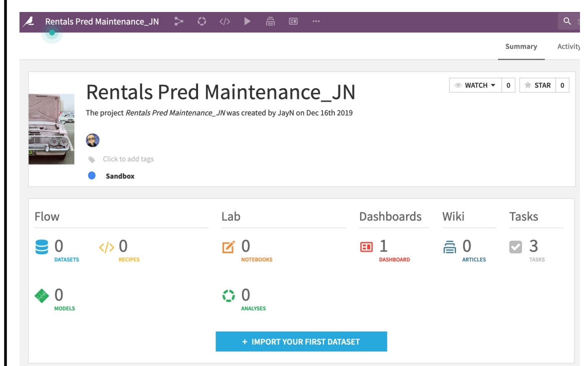
https://dssX-design-bern.training.dataiku.com/

login : **userY**
password : **obifYi**

ToDo: Create a New Project



Project Homepage

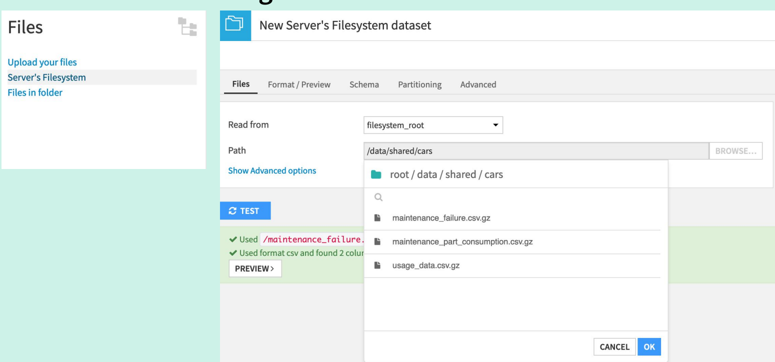


- This is a **Project**
- Landing page
- Overview of assets & objects

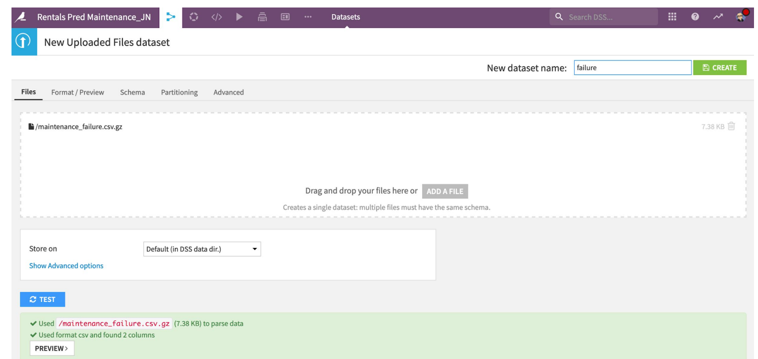
- G+F hotkeys to the **Flow**

Import the "maintenance" Dataset

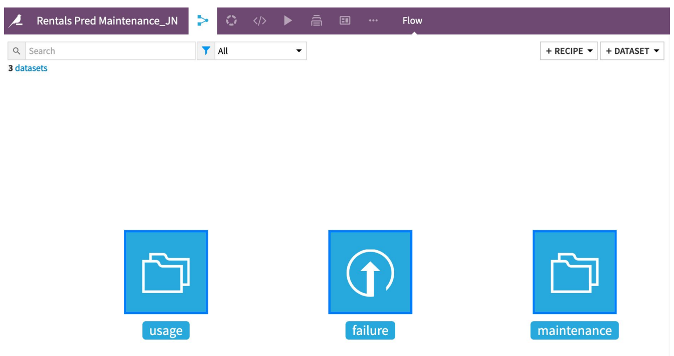
Show them how to get to the data



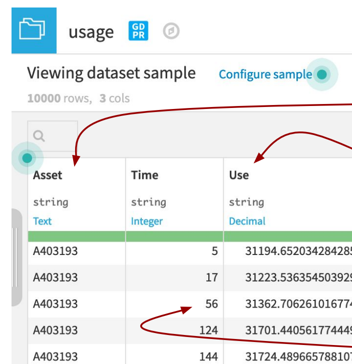
Import the "maintenance" Dataset



Three Datasets



The “usage” dataset



usage

Viewing dataset sample Configure sample

10000 rows, 3 cols

Asset	Time	Use
string	string	string
Text	Integer	Decimal
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 the total number of miles a car
has driven at the specified “*Time*”

Unclear reference to “*Time*”

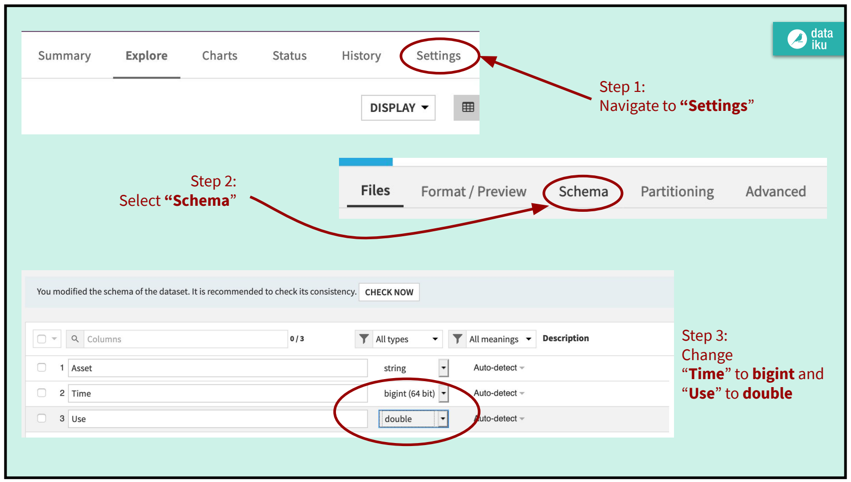
Working the “usage” Dataset

- 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

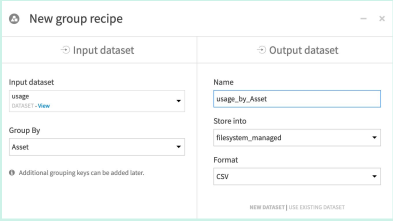


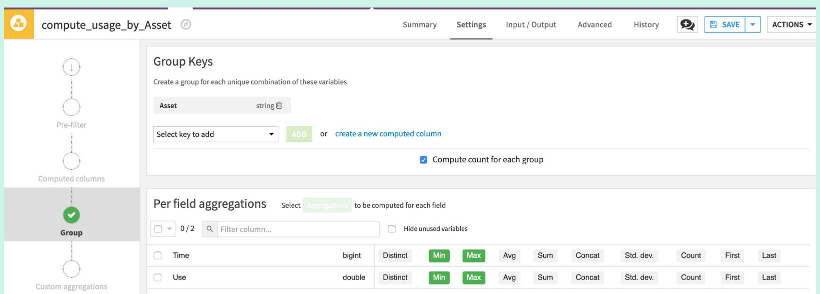
Group By

To Do: Collapse on Asset

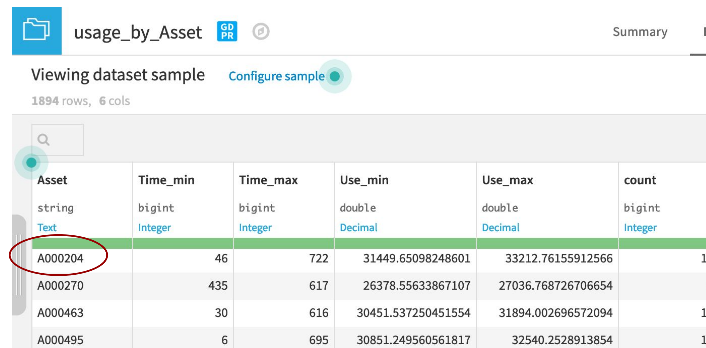
Schema corrected! … Let's proceed with the GroupBy:

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.



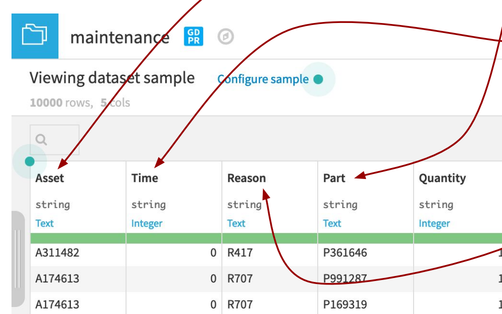


We have aggregated at the level of a unique rental car



Your Flow (G+F) Should Look Similar

The “maintenance” Dataset



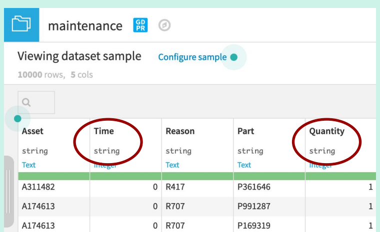
The ***maintenance*** dataset documents activity that has occurred with respect to:

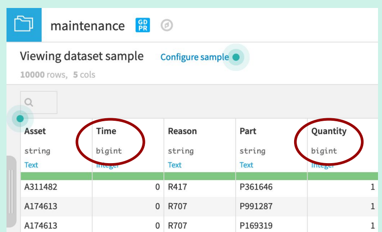
- a given *Asset*
- organized by *Part* (what was repaired) and
- *Time* (when it was repaired).
- A *Reason* variable codifies the nature of the problem.

ToDo: Adjust the schema on the "**maintenance**" Dataset

As we did before, set the schema to have:

- "**Time**" as bigint
- "**Quantity**" as a bigint





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**

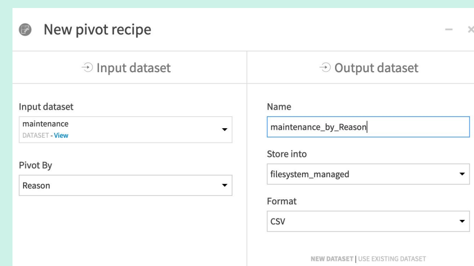
Asset	Time	Reason	Part	Quantity
string	bigint	string	string	bigint
Text	Integer	Text	Text	Integer
A311482	0	R417	P361646	1
A174613	0	R707	P991287	1
A174613	0	R707	P169319	1
A060723	0	R193	P184448	0
A174613	0	R707	P097048	1
A174613	0	R707	P169319	1
A174613	0	R707	P610186	1
A174613	0	R707	P168211	1

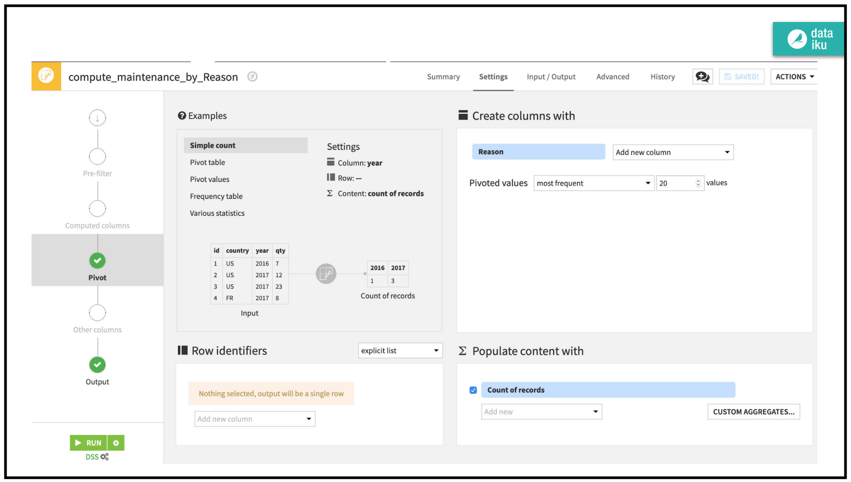
Asset	R193_Quantity_sum	R364_Quantity_sum	R446_Quantity_sum	R565_Quantity_sum	R707_Quantity_sum
string	bigint	bigint	bigint	bigint	bigint
Text	Integer	Integer	Integer	Integer	Integer
A000204	88	3	3	106	194
A000270					108
A000463	96	1	5		229
A000495	123	13	39	394	145
A001201	7				87
A001215	5	1	14	148	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.

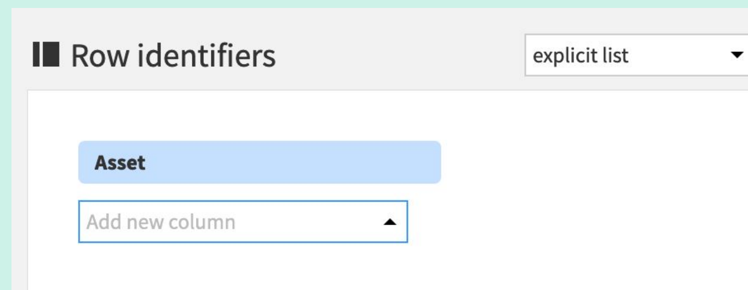
- With *maintenance* chosen as the input dataset, choose to **Pivot By** ***Reason***.





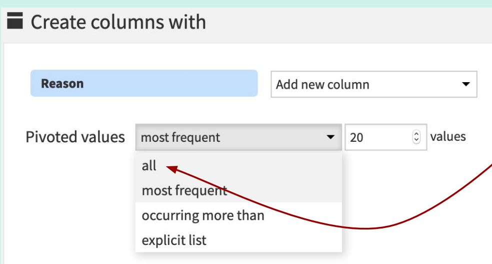
Pivoting "maintenance"

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



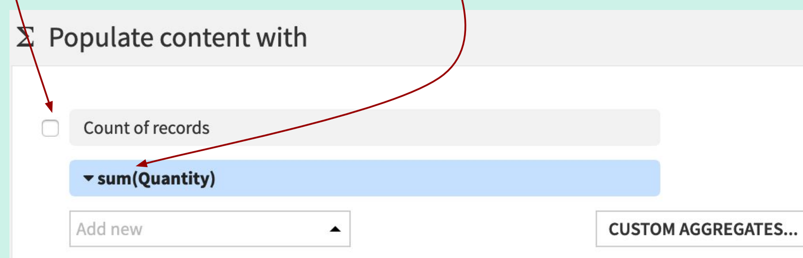
Pivoting “**maintenance**”

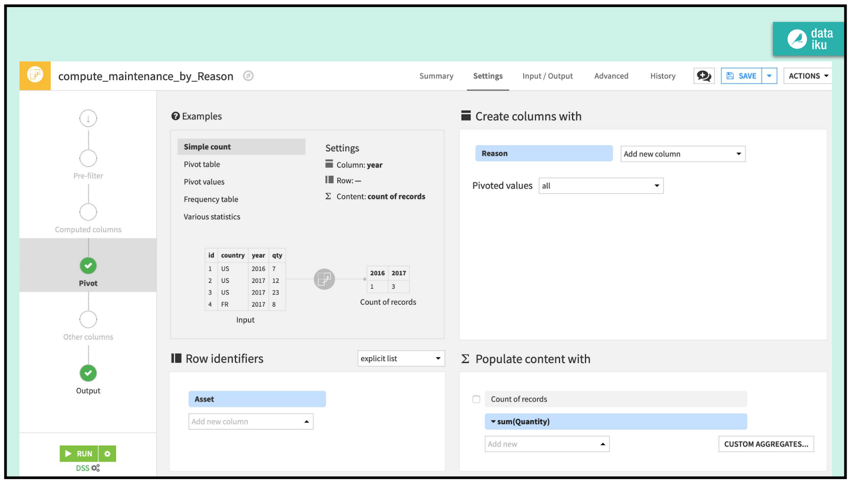
- ***Reason*** should already be selected under **Create columns with**.
- Although it should make no difference in this case, change **Pivoted values** to **all** so that all values of ***Reason*** are pivoted into columns.



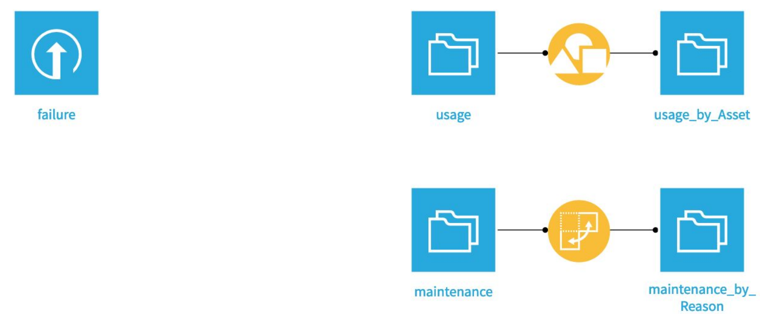
Pivoting “**maintenance**”

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



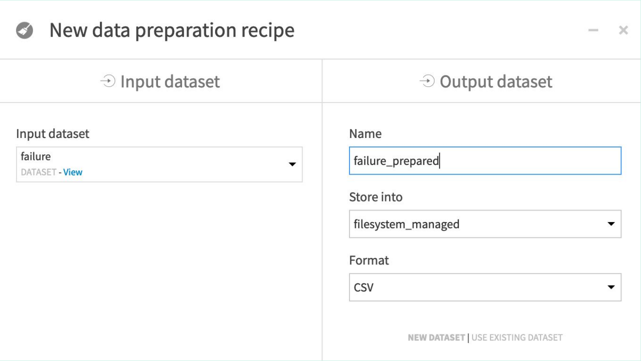


Your Flow (G+F) Should Look Similar



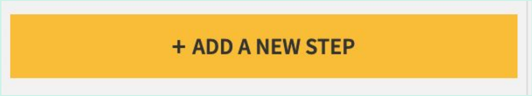
An Short Introduction to **Prepare** Recipe

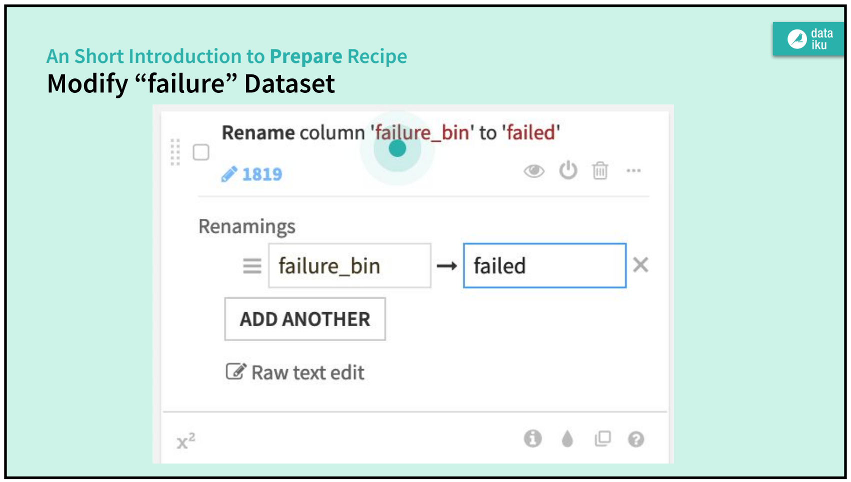
Modify “failure” Dataset

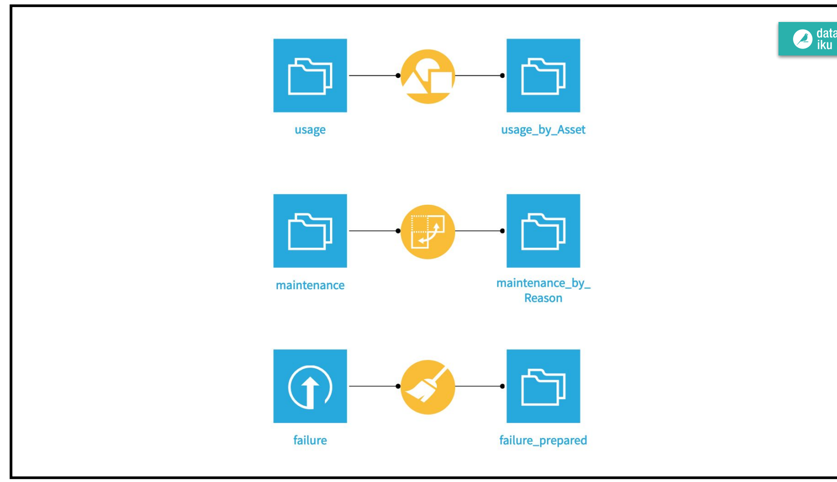


An Short Introduction to **Prepare** Recipe

Modify “failure” Dataset




data iku
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


data iku
usage
usage_by_Asset
maintenance
maintenance_by_Reason
failure
failure_prepared

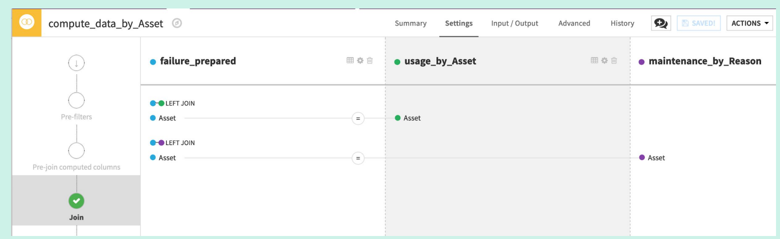
Merging Data

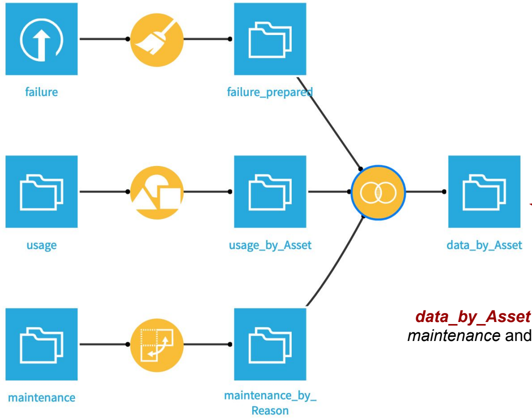
Introducing the “Join” recipe

- 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

1. From the ***failure_prepared*** dataset, initiate a **Join** recipe.
2. Add ***usage_by_Asset*** as the second input dataset.
3. Name the output ***data_by_Asset***. Click **Create Recipe**.
4. Add ***maintenance_by_Reason*** as the third input dataset.
5. Both joins should be **Left Joins**. *Asset* should be the joining key in all cases.
6. **Run** the recipe and update the schema to 21 columns.





data_by_Asset now has information from *maintenance* and *usage*, labelled by failures

Splits

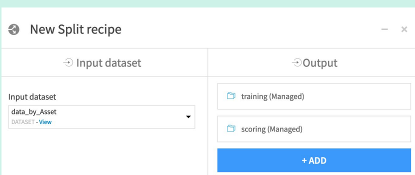
Working toward a Training And to-Score Datasets

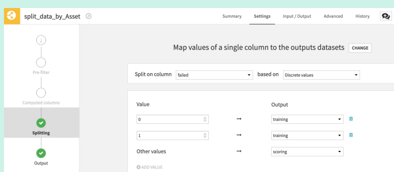
To train models, we'll use the **Split** recipe to create two separate datasets from the merged dataset, *data_by_Asset*:

- 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.

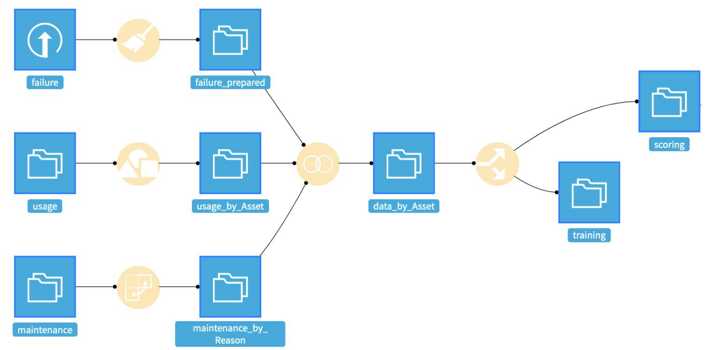
ToDo - Split to Create training and scoring Datasets

1. From the ***data_by_Asset*** recipe, initiate a **Split** recipe.
2. Add two output datasets, named ***training*** and ***scoring***, selecting **Create Dataset** each time. Then **Create Recipe**.
3. At the **Splitting** step, choose to **Map values of a single column**. Then choose ***failure_bin*** as the column on which to split.
4. Assign values of **0** and **1** to the ***training*** set, and all “Other values” to the ***scoring*** set.
5. **Run** the recipe.





Your Flow (G+F) Should Look Similar

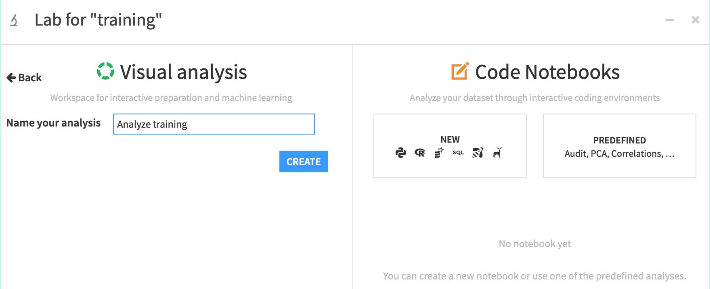


Feature Generation

- 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**.



Working in Lab Mode

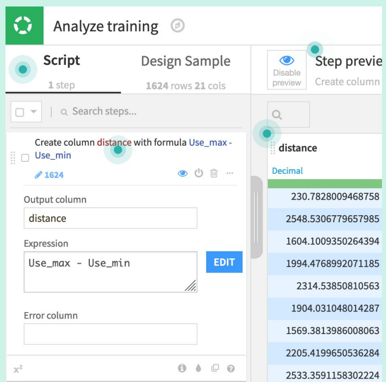
1. In the screen which looks similar to a **Prepare** recipe, **create two new variables** with the **formula processor**

- **distance** from the expression:

  Use_max - Use_min

- **time_in_service** from the expression:

  Time_max - Time_min



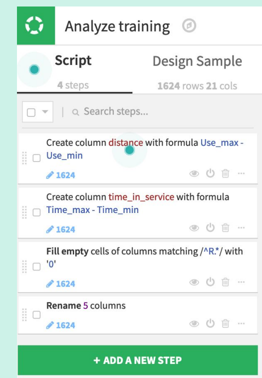
Working in Lab Mode

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



Working in the Lab

Note

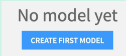
- It is not necessary to deploy a script created in the Lab to the Flow in order to make use of the new features in the modeling process.
- Any models created in a Visual Analysis have access to any features created in the same Visual Analysis.

Creating Models to Predict Car Breakdowns

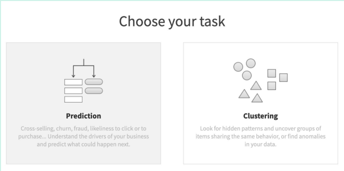
From the open Lab Script, navigate to the **Models** tab.

Create our First Model. Dataiku DSS lets us choose between two types of modeling tasks:

1. **Prediction** (or supervised learning): to predict a target variable (including labels), given a set of input features
2. **Clustering** (or unsupervised learning): to create groups of observations based on some shared patterns or characteristics



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.**

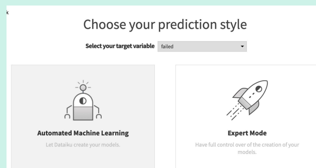
Automated Machine Learning helps with some important decisions like choosing the type of algorithms and parameters of those algorithms.

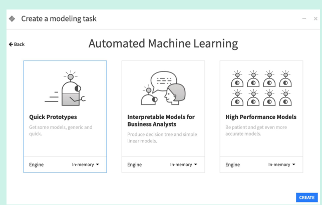
- Select **Automated Machine Learning** and then **Quick Prototypes**, the default suggestions.

Dataiku DSS then asks us to select the target variable.

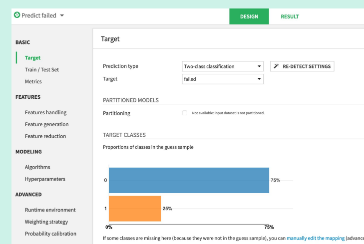
In this case, we want to calculate the probabilities for one of two outcomes: failure or non-failure, i.e. perform **two-class (binary) classification**.

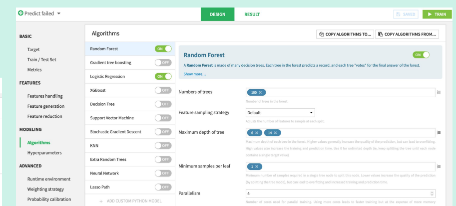
Accordingly, choose ***failed*** as the target variable.

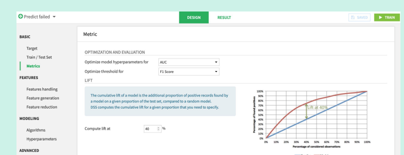




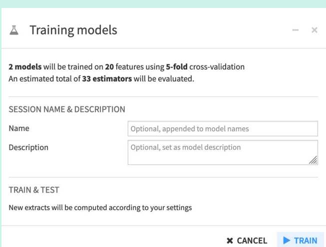
Review the Design Tab of your Models

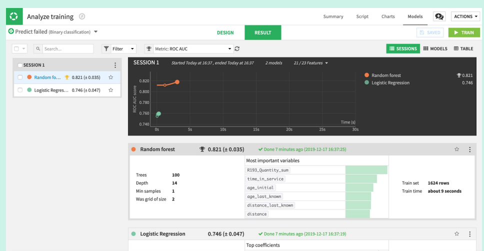






Train Some Models

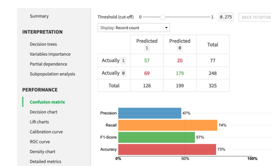




Understanding the Model

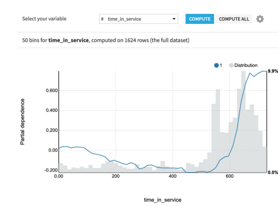
- Model metrics can be found under the **Results** tab. Dataiku

  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.



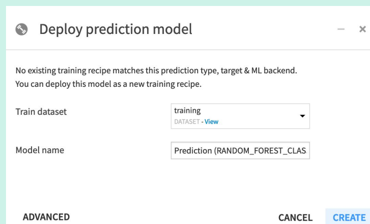
- By selecting a model, many additional insights and ready-made analysis are available.

  Examples: Confusion Matrix, ROC curves, Tree visualizations, Variable Importance …

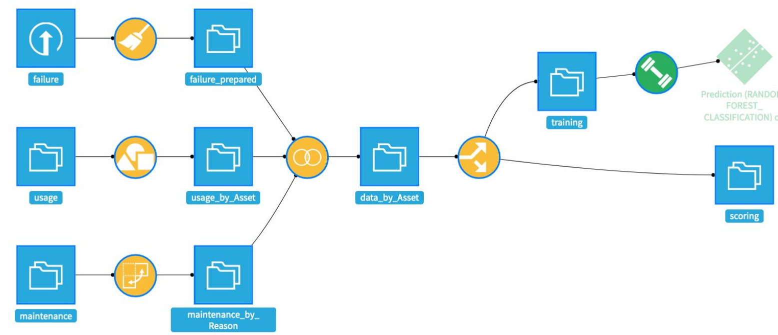


Deploy Top Performer to the Flow

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**.

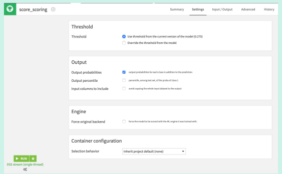


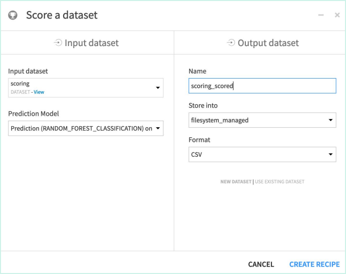
Your Flow (G+F) Should Look Similar



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.

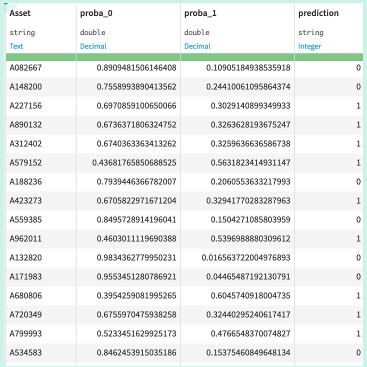




Scoring Unlabelled Data

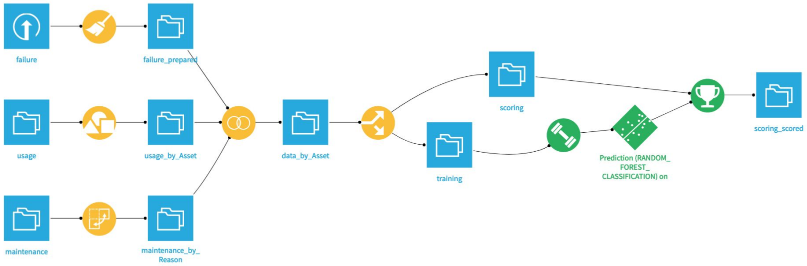
The resulting dataset now contains three new columns:

- **proba_1**: probability of failure
- **proba_0**: probability of non-failure (1 - proba_1)
- **prediction**: model prediction of failure or not (based on probability threshold)



Asset string Text	proba_0 double Decimal	proba_1 double Decimal	prediction string Integer
A082667	0.8909481506146408	0.10905184938535918	0
A148200	0.7558993890413562	0.24410061095864374	0
A227156	0.6970859100650066	0.3029140899349933	1
A890132	0.6736371806324752	0.3263628193675247	1
A312402	0.6740363363413262	0.3259636636586738	1
A579152	0.43681765850688525	0.5631823414931147	1
A188236	0.7939446366782007	0.2060553633217993	0
A423273	0.6705822971671204	0.32941770283287963	1
A559385	0.8495728914196041	0.1504271085803959	0
A962011	0.4603011119690388	0.5396988880309612	1
A132820	0.9834362779950231	0.016563722004976893	0
A171983	0.9553451280786921	0.04465487192130791	0
A680806	0.3954259081995265	0.6045740918004735	1
A720349	0.6755970475938258	0.32440295240617417	1
A799993	0.5233451629925173	0.4766548370074827	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 packaged 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!