

data
iku

Dataiku Training

Hands-on Day 2



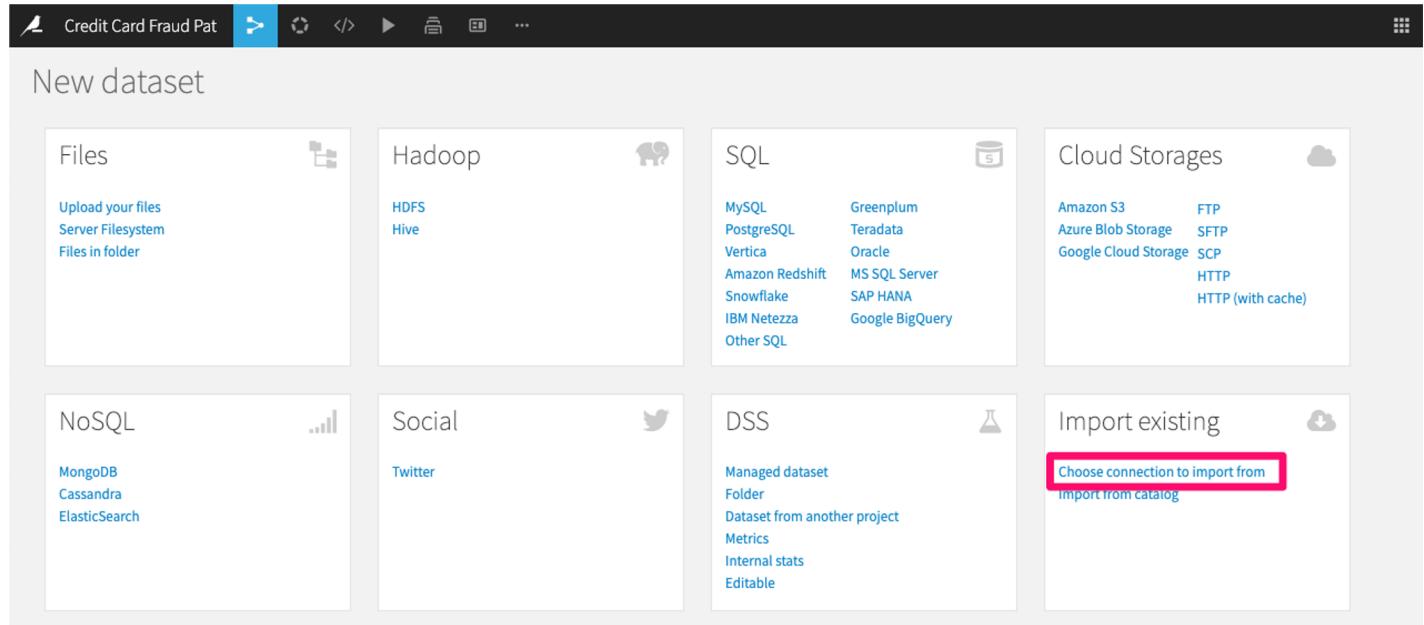
Agenda

- *Dataiku introduction*
- *Demo - basic project*
- Session 1 - hands-on Dataiku project
- **Session 2 - hands-on Dataiku project**
 - Advanced data preparation
 - Machine learning
 - Feature Engineering
- Session 3 - hands-on Dataiku project

Import data from PostgreSQL

+ IMPORT YOUR FIRST DATASET

Select “Choose connection to import from”



The screenshot shows the Dataiku DSS interface with a red box highlighting the "Choose connection to import from" button in the "Import existing" section.

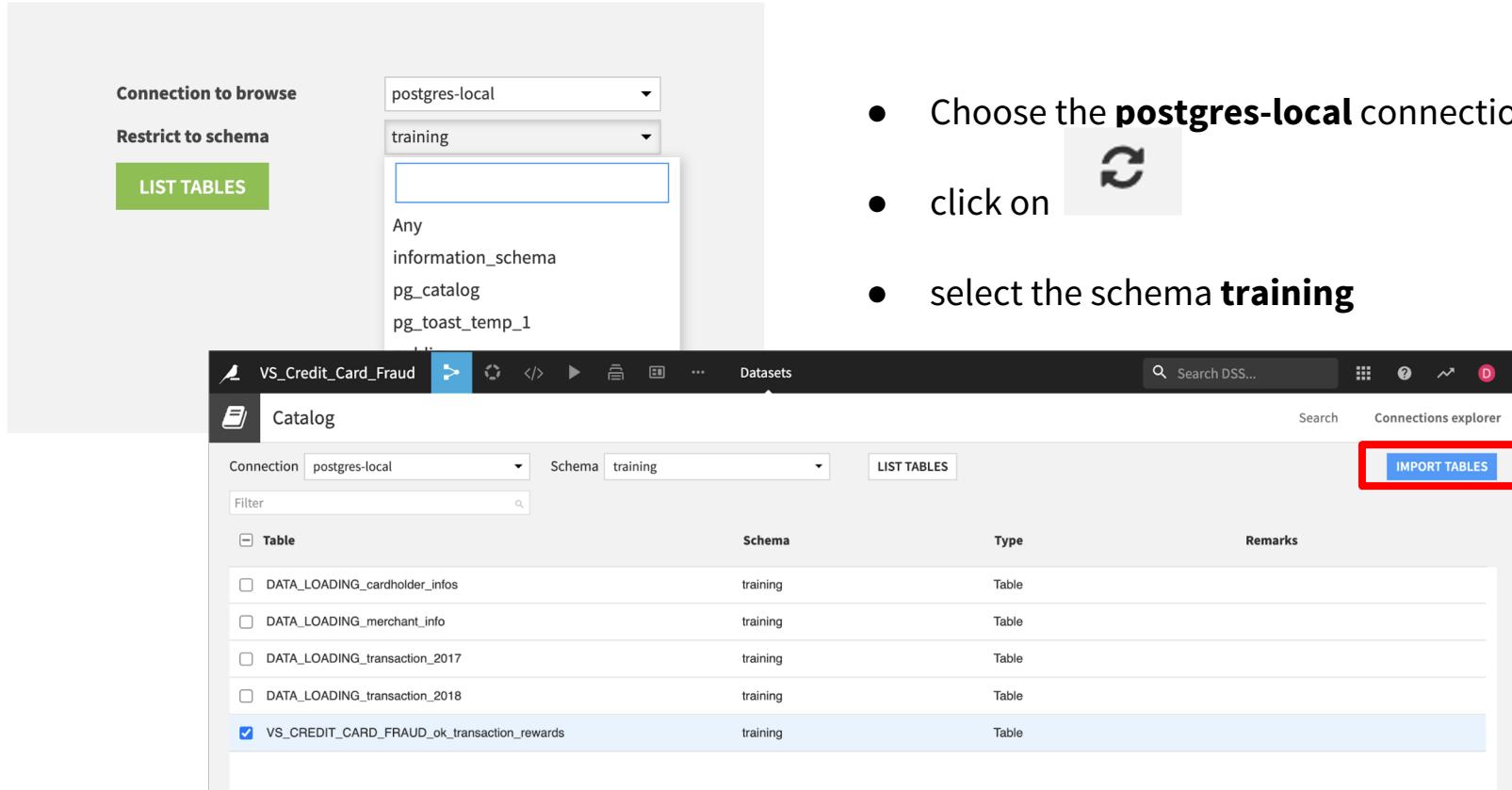
New dataset

Choose connection to import from

Import existing

- Files**
 - Upload your files
 - Server Filesystem
 - Files in folder
- Hadoop**
 - HDFS
 - Hive
- SQL**
 - MySQL
 - PostgreSQL
 - Vertica
 - Amazon Redshift
 - Snowflake
 - IBM Netezza
 - Other SQL
 - Greenplum
 - Teradata
 - Oracle
 - MS SQL Server
 - SAP HANA
 - Google BigQuery
- Cloud Storages**
 - Amazon S3
 - Azure Blob Storage
 - Google Cloud Storage
 - FTP
 - SFTP
 - SCP
 - HTTP
 - HTTP (with cache)
- NoSQL**
 - MongoDB
 - Cassandra
 - ElasticSearch
- Social**
 - Twitter
- DSS**
 - Managed dataset
 - Folder
 - Dataset from another project
 - Metrics
 - Internal stats
 - Editable
- Choose connection to import from**
 - Import from catalog

Import data from PostgreSQL



The screenshot shows the DataStax Studio (DSS) interface for importing data from a PostgreSQL database.

Top Left: Connection to browse dropdown set to **postgres-local**.
Restrict to schema dropdown set to **training**.
LIGHT GREEN BUTTON: LIST TABLES

Bottom Left: Catalog view showing tables from the **training** schema of the **postgres-local** connection. The table **VS_CREDIT_CARD_FRAUD_ok_transaction_rewards** has a checked checkbox and is highlighted with a light blue background.

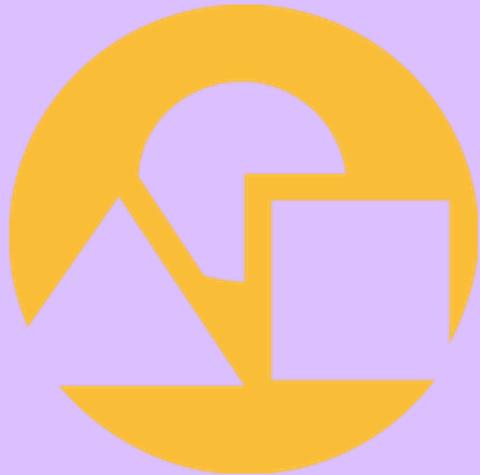
Top Right: A list of instructions:

- Choose the **postgres-local** connection
- click on 
- select the schema **training**

Bottom Right: A red box highlights the **IMPORT TABLES** button in the Catalog view header.

Table	Schema	Type	Remarks
DATA_LOADING_cardholder_infos	training	Table	
DATA_LOADING_merchant_info	training	Table	
DATA_LOADING_transaction_2017	training	Table	
DATA_LOADING_transaction_2018	training	Table	
VS_CREDIT_CARD_FRAUD_ok_transaction_rewards	training	Table	

Group by



Use this recipe to calculate summary information from groups of rows in a dataset

Group the **transaction_reward_value** dataset

Group by:

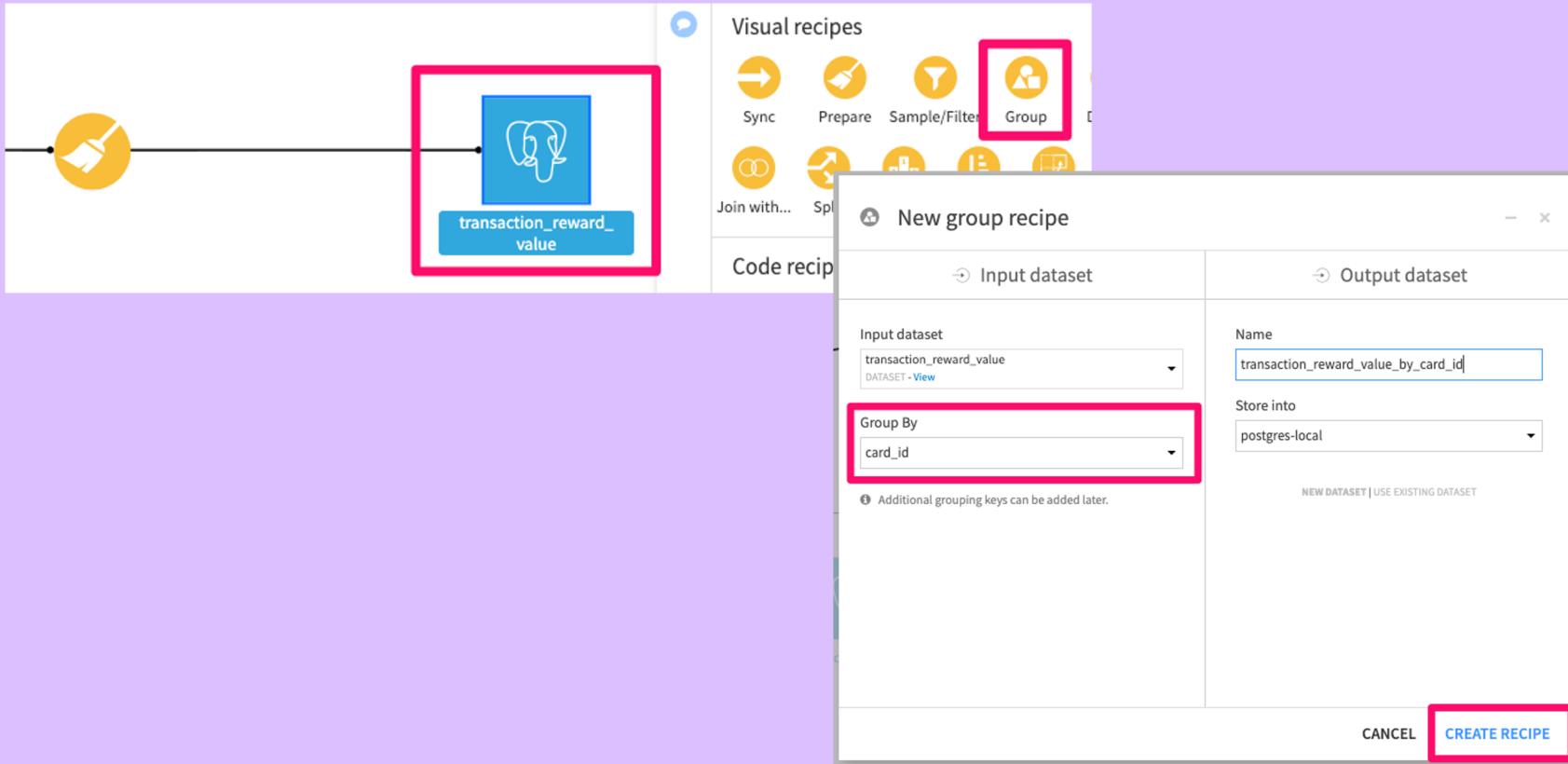
- **card_id**

Then compute these new columns:

- **Sum purchase_amount**
- **Sum reward_cash_back**
- **Sum reward_dining**
- **Sum reward_travel**
- **First card_fico_score**

Our output dataset will have one row per cardholder, and will calculate the theoretical benefits from each reward program across their purchase history

Group the ok_transactions_rewards dataset



The screenshot shows a dataiku workflow interface. On the left, a step icon featuring a broom and a blue elephant icon is connected to a step labeled "transaction_reward_value". This step is highlighted with a red box. To the right, a "Visual recipes" panel is open, showing various icons for Sync, Prepare, Sample/Filter, and Group. The "Group" icon is highlighted with a red box. A larger modal window titled "New group recipe" is displayed over the main interface. It contains fields for "Input dataset" (set to "transaction_reward_value"), "Name" (set to "transaction_reward_value_by_card_id"), "Group By" (set to "card_id"), and "Store into" (set to "postgres-local"). A note at the bottom says "Additional grouping keys can be added later." At the bottom right of the modal, there are "CANCEL" and "CREATE RECIPE" buttons, with the "CREATE RECIPE" button also highlighted with a red box.

New group recipe

transaction_reward_value

transaction_reward_value_by_card_id

card_id

postgres-local

CANCEL CREATE RECIPE

Choose the aggregations

compute_ok_transaction_rewards_by_card_id

Group Keys

Create a group for each unique combination of these variables

card_id string

Select key to add or [create a new computed column](#)

Compute count for each group

Per field aggregations Select [Aggregations](#) to be computed for each field

Field	Type	boolean	Distinct	Min	Max	Avg	Sum	Concat	Std. dev.	Count	First	Last
category_1	boolean	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>					
merchant_category_id	bigint	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>					
purchase_amount	double	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>								
reward_cashback	double	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>					
reward_travel	double	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>					
reward_dining	double	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>								
purchase_date	date	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>					

0 / 27 Hide unused variables

Group

Pre-filter

Computed columns

Custom aggregations

Post-filter

...and run the
recipe

Group the `ok_transactions_rewards` dataset

Group by:

- `region`
- `merchant_subsector_description`

Then compute these new columns:

- Sum `purchase_amount`
- Avg `purchase_amount`

Our output dataset will have one row per state and merchant category, and will calculate the average and total purchase amounts for each

Group the ok_transactions_rewards dataset

The screenshot shows a data processing interface with the following details:

- Top Bar:** Includes icons for file operations (New, Open, Save, etc.), a search bar ("Search DSS..."), and a "D" icon.
- Left Sidebar:** Lists available datasets: "VS_Credit_Card_Fraud" (selected), "compute_ok_transaction_rewards_by_region" (highlighted in orange), and "Recipes".
- Current Recipe Details:**
 - Title:** compute_ok_transaction_rewards_by_region
 - Summary:** A brief description of the recipe.
 - Settings:** The active tab, showing settings for grouping and aggregation.
 - Input / Output:** Configuration for inputs and outputs.
 - Advanced:** Advanced options.
 - History:** History of runs.
 - ACTIONS:** Buttons for "SAVE" and "RUN".
- Group Keys Section:** Configures group keys for unique combinations of variables.
 - Selected keys: "region" (string) and "merchant_subsector_description" (string).
 - Options: "Select key to add" dropdown, "ADD" button, and link to "create a new computed column".
 - Checkboxes: "Compute count for each group" (checked).
- Per field aggregations Section:** Configures aggregations for each field.
 - Header: "Select Aggregations" (button).
 - Table:

Field	Type	Distinct	Min	Max	Avg	Sum	Concat	Std. dev.	Count	First	Last
category_1	boolean	Distinct	Min	Max	Avg	Sum	Concat	Std. dev.	Count	First	Last
merchant_category_id	bigint	Distinct	Min	Max	Avg	Sum	Concat	Std. dev.	Count	First	Last
purchase_amount	double	Distinct	Min	Max	Avg	Sum	Concat	Std. dev.	Count	First	Last
reward_cashback	double	Distinct	Min	Max	Avg	Sum	Concat	Std. dev.	Count	First	Last
reward_travel	double	Distinct	Min	Max	Avg	Sum	Concat	Std. dev.	Count	First	Last
reward_dining	double	Distinct	Min	Max	Avg	Sum	Concat	Std. dev.	Count	First	Last
- Bottom Navigation:** Buttons for "RUN" and "In-database (SQL)".

PIVOT



Use this recipe to reshape and aggregate your dataset

Group the **transaction_reward_value** dataset

pivot by:

- `transaction_date_year`

Identify rows by:

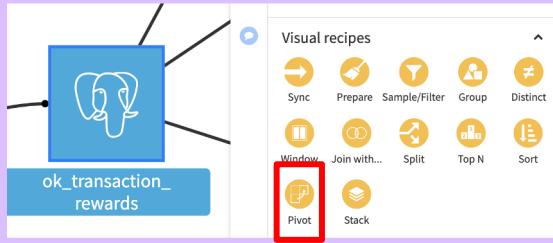
- `merchant_subsector_description`

Then compute these new columns:

- `Sum purchase_amount`

Our output dataset will have one row per merchant type, and will calculate the total revenue per year for each category

Pivot the ok_transactions_rewards dataset



New pivot recipe

Input dataset

Input dataset

ok_transaction_rewards
DATASET - View

Pivot By

purchase_date_year

Output dataset

Name

ok_transaction_rewards_by_year

Store into

postgres-local

NEW DATASET | USE EXISTING DATASET

CANCEL CREATE RECIPE

The screenshot shows a 'New pivot recipe' dialog box. It has two main sections: 'Input dataset' and 'Output dataset'. In the 'Input dataset' section, 'ok_transaction_rewards' is selected. In the 'Pivot By' dropdown, 'purchase_date_year' is chosen and highlighted with a red box. In the 'Output dataset' section, the 'Name' is set to 'ok_transaction_rewards_by_year' and 'Store into' is set to 'postgres-local'. At the bottom, there are 'NEW DATASET | USE EXISTING DATASET' buttons and 'CANCEL' and 'CREATE RECIPE' buttons.

Pivot the ok_transactions_rewards dataset

compute_ok_transaction_rewards_by_year

Summary Settings Input / Output Advanced History [SAVE](#) [ACTIONS](#)

Examples

Simple count

Pivot table

Pivot values

Frequency table

Various statistics

Settings

Column: **year**

Row: —

Content: **count of records**

Pre-filter

Computed columns

Pivot

Other columns

Output

Input

Count of records

id	country	year	qty
1	US	2016	7
2	US	2017	12
3	US	2017	23
4	FR	2017	8

Create columns with

purchase_date_year Add new column

Pivoted values most frequent 20 values

Row identifiers explicit list

merchant_subsector_description

Add new column

Populate content with

Count of records

sum(purchase_amount)

Add new CUSTOM AGGREGATES...

[RUN](#) [⚙️](#)

In-database (SQL) ⚙️

Pivot the ok_transactions_rewards dataset

Go back to the pivot recipe and change the settings, what is the output ?

compute_ok_transaction_rewards_by_year

Summary Settings Input / Output Advanced History SAVED! ACTIONS

Pre-filter → Computed columns → Pivot → Other columns → Output

Examples

Simple count

Settings: Column: year, Row: —, Content: count of records

Pivot table, Pivot values, Frequency table, Various statistics

Input:

	id	country	year	qty
1	US	2016	7	
2	US	2017	12	
3	US	2017	23	
4	FR	2017	8	

Count of records:

	2016	2017
1	1	3

Create columns with

region → Add new column

Pivoted values: most frequent, 20 values

Row identifiers

explicit list

merchant_subsector_description, purchase_date_year

Populate content with

Count of records, sum(purchase_amount)

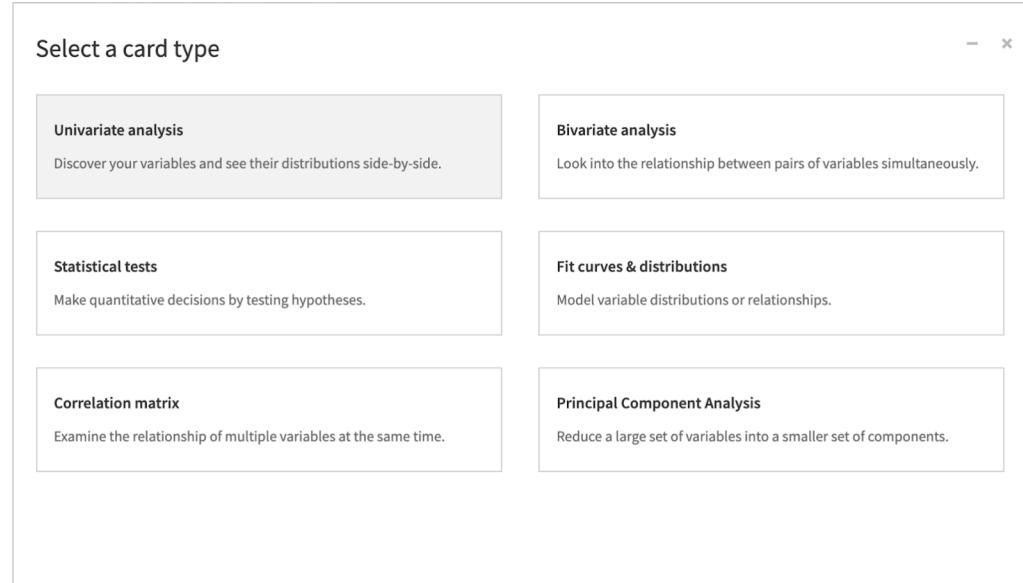
CUSTOM AGGREGATES...

RUN In-database (SQL)

Visual Statistics

Data Audit, EDA and Statistical Insights

Since DSS7.0, we provide a full UI for statistical tests:



Select a card type

Univariate analysis
Discover your variables and see their distributions side-by-side.

Bivariate analysis
Look into the relationship between pairs of variables simultaneously.

Statistical tests
Make quantitative decisions by testing hypotheses.

Fit curves & distributions
Model variable distributions or relationships.

Correlation matrix
Examine the relationship of multiple variables at the same time.

Principal Component Analysis
Reduce a large set of variables into a smaller set of components.

Statistics are located in the “**Statistics**” tab when double clicking on a Dataset if the Flow

Full online courses on Visual Statistics

<https://academy.dataiku.com/interactive-visual-statistics-open>

Create Univariate Analysis

Start from dataset `ok_transactions_rewards`

Click **+ New Card** and click Univariate Analysis

← Univariate analysis

1. Select variables

22 available variables

Search	
# authorized_flag	
A card_id	
# city_id	
A category_1	
# merchant_category_id	
A purchase_date	
# client_seniority	
# purchase_date_year	
...	

Use Shift + Click to select a range of variables.

2. Add variables

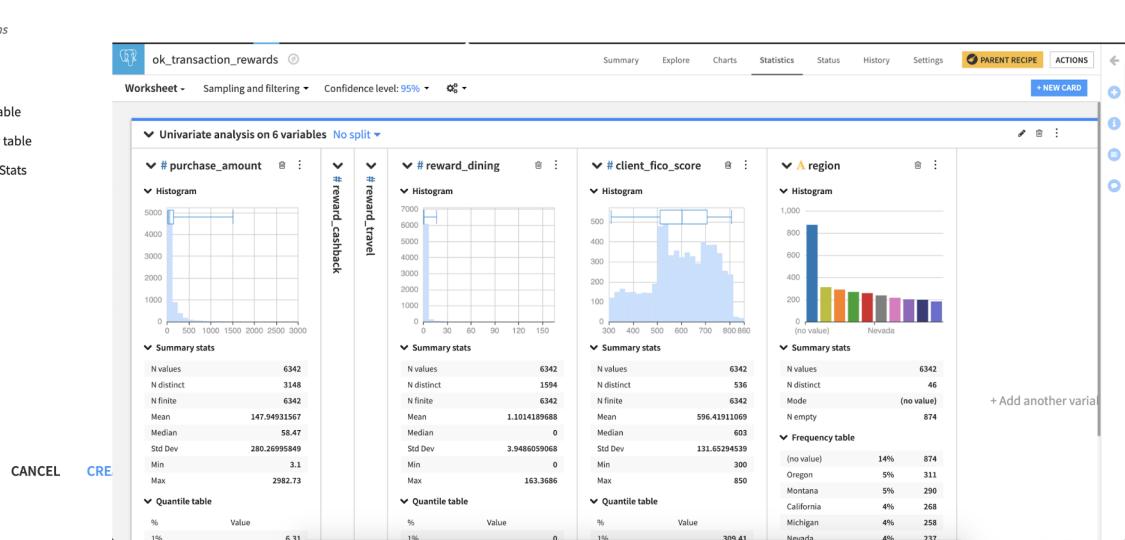
Variables to describe

purchase_amount
reward_cashback
reward_travel
reward_dining
client_fico_score
A region

3. Select options

Options

- Histogram
- Quantile table
- Frequency table
- Summary Stats
- Box Plot



Create Univariate Analysis

Sampling and Publishing

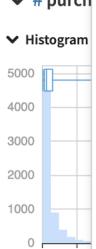
Be careful of sampling methods

Worksheet ▾ Sampling and filtering ▾ Confidence level: 95% ▾

Univariate

purch

Histogram



Sampling method

Random (approx. ratio)

No sampling (whole data)
Takes the whole data

First records
Takes the first N rows of the dataset. Very fast (only reads N rows) but may result in a very biased view of the dataset.

ORDERING

Enable filtering

Random (approx. ratio)
Randomly selects approximately X% of the rows. Requires a full pass reading the data.

ADVANCED

Auto Refresh Sample

SAVE AND REFRESH SAMPLE

N values 6342 N distinct 3148

N values 6342 N distinct 1594

Summary stats

Sampling and filtering ▾

Sampling method

Random (approx. ratio)

No sampling (whole data)
Takes the whole data

First records
Takes the first N rows of the dataset. Very fast (only reads N rows) but may result in a very biased view of the dataset.

ORDERING

Enable filtering

Random (approx. ratio)
Randomly selects approximately X% of the rows. Requires a full pass reading the data.

ADVANCED

Auto Refresh Sample

SAVE AND REFRESH SAMPLE

N values 6342 N distinct 3148

N values 6342 N distinct 1594

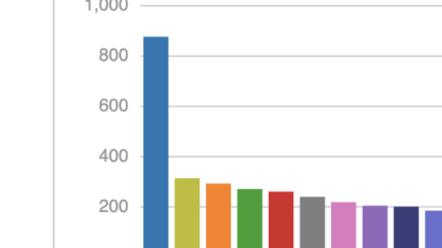
Summary stats

Publish cards to dashboard

+ NEW CARD

A region

Histogram



Duplicate as new card

View JSON...

Publish...

Create 2D Fit Curve to create variable correlation

Start from dataset `ok_transactions_rewards`

Fit curves & distributions

Fit Distribution

Estimate the parameters of the most common distributions (normal, exponential, ...)

2D Fit Distribution

Visualize the density of bivariate distributions (KDE and bivariate gaussian).

What can you analyse from this graph ?

Fit Curve

Model the relationship between two variables.

Fit 2D distribution

X Variable

`purchase_amount`

2D KDE

Joint Normal

Y Variable

`client_fico_score`

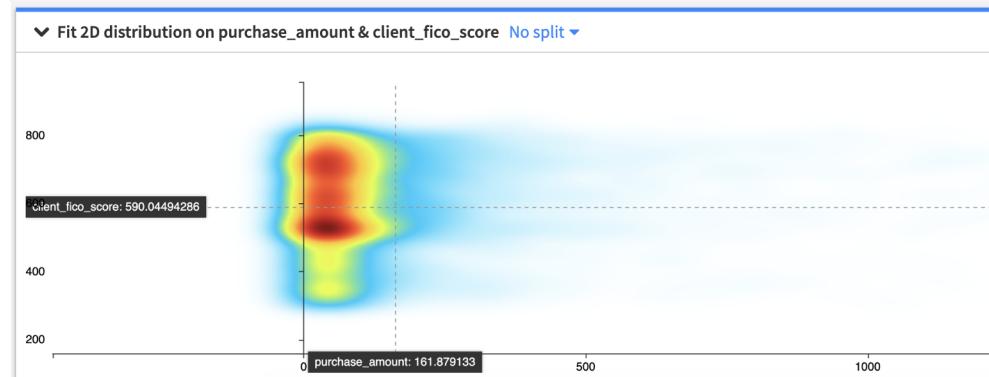
X relative bandwidth (%)

15

Y relative bandwidth (%)

15

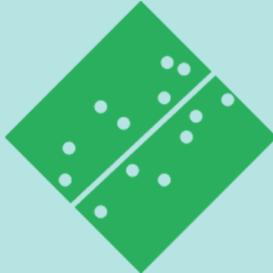
CANCEL APPLY



Challenge 1

Green Track

Predict fraudulent transactions using machine learning

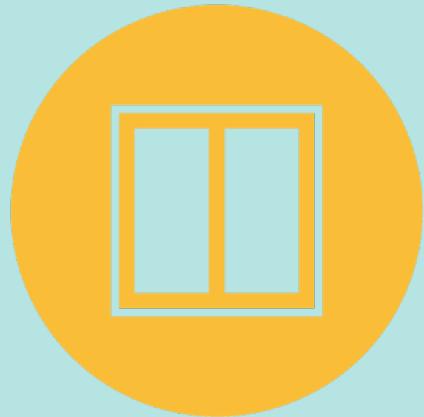


Challenge 1 Steps

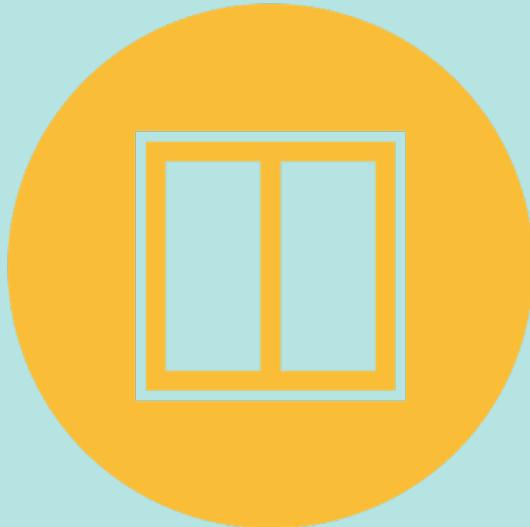


- Create lagged variables based on historical data
- Train a machine learning model in DSS using known 2017 data
- Use the model to make predictions for unknown 2018 data

Window



Window



Use this recipe to calculate summary information from groups of rows in a dataset, **while keeping the structure of the dataset intact**

Window function

- When we build an ML model to predict which transactions are fraudulent, we want to use **historical data** to supplement the information we already know about the transaction in question
- We can use a **window function** to generate these historical features, like lags and moving averages

Window function examples

- Rank
 - is it the first, second, third, ... transaction of the customer ?
- Lag
 - What did the customer buy in his/her previous transaction ?
- Rolling average
 - Average amount of transactions the 3 previous days
- Rolling average by key
 - Rolling average number of sales **per customer** across his / her 3 previous orders

Window recipe

Rank

➤ Rank

- is it the first, second, third, ... order of the customer ?

Group by		Order by
Customer	Quantity	Date
Caroline	5	10/1/2017
Alice	1	10/1/2017
Bob	4	10/2/2017
Alice	2	10/5/2017
Caroline	4	10/7/2017
Caroline	2	10/13/2017
Bob	1	10/13/2017
Alice	3	10/14/2017



Customer	Quantity	Date	Customer Order #
Caroline	5	10/1/2017	1
Caroline	4	10/7/2017	2
Caroline	2	10/13/2017	3
Alice	1	10/1/2017	1
Alice	2	10/5/2017	2
Alice	3	10/14/2017	3
Bob	4	10/2/2017	1
Bob	1	10/13/2017	2

Caroline's first order

Alice's first order

Bob's first order

Window recipe

Lag

➤ Lag

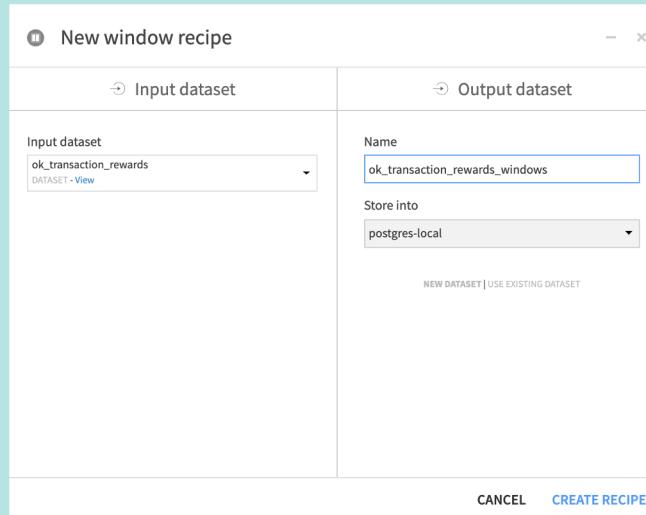
- What did the customer buy in his/her previous order ?

Group by		Order by
Customer	Quantity	Date
Caroline	5	10/1/2017
Alice	1	10/1/2017
Bob	4	10/2/2017
Alice	2	10/5/2017
Caroline	4	10/7/2017
Caroline	2	10/13/2017
Bob	1	10/13/2017
Alice	3	10/14/2017



Customer	Quantity	Date	Customer Order #
Caroline	5	10/1/2017	--
Caroline	4	10/7/2017	5
Caroline	2	10/13/2017	4
Alice	1	10/1/2017	--
Alice	2	10/5/2017	1
Alice	3	10/14/2017	2
Bob	4	10/2/2017	--
Bob	1	10/13/2017	4

Define a window on ok_transaction_rewards

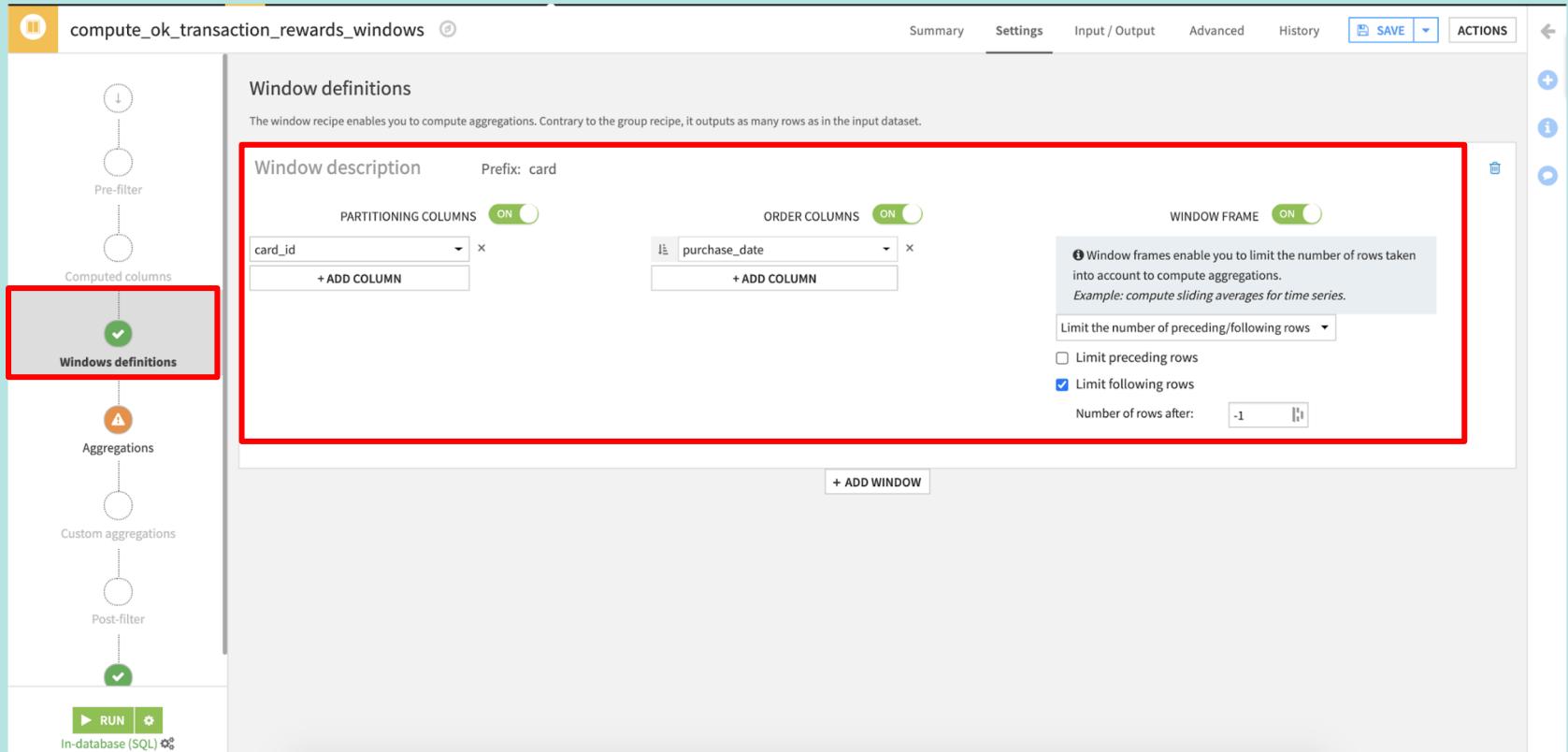


Window setup

- **partition (group) by 'card_id'**
- **order rows by 'purchase_date_parsed' (ascending)**
- **limit the following rows by -1 (to exclude the current row from your window calculations)**

Define your window

add prefix to your window



The screenshot shows the Dataiku DSS interface for a recipe named "compute_ok_transaction_rewards_windows". The interface includes a sidebar with nodes like Pre-filter, Computed columns, Windows definitions (highlighted with a red box), Aggregations, Custom aggregations, and Post-filter. The main area displays "Window definitions" with a description: "The window recipe enables you to compute aggregations. Contrary to the group recipe, it outputs as many rows as in the input dataset." It shows "PARTITIONING COLUMNS" set to "ON" with "card_id" selected, and "ORDER COLUMNS" set to "ON" with "purchase_date" selected. A "WINDOW FRAME" is also set to "ON". A tooltip explains that window frames limit the number of rows taken into account for aggregations, with an example of computing sliding averages for time series. Below the frame settings, there are checkboxes for "Limit preceding rows" (unchecked) and "Limit following rows" (checked), with a field for "Number of rows after: -1". A "RUN" button at the bottom left indicates the recipe is ready to execute.

Define aggregations

- AVG, SUM, STD DEV of purchase amount
- LAG 1 of purchase_amount
- AVG of purchase_day_of_week

and whatever else you want to try!

Define aggregations

compute_ok_transaction_rewards_windows 

Summary Settings Input / Output Advanced History  ACTIONS

Pre-filter

Computed columns

Windows definitions

Aggregations

Custom aggregations

Post-filter

 RUN  In-database (SQL) 

Compute rank for each row

You can compute the "ranks" of each row, according to the ordering defined in the [windows definitions](#), with the following methods.
A rank is computed for each defined window.

Compute: Rank Dense rank Row number Cumulative distribution Quantile

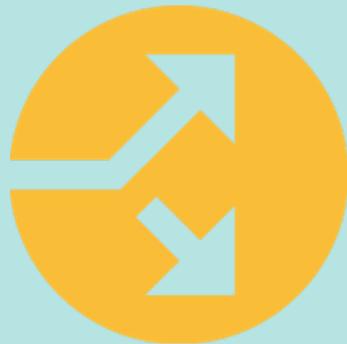
Compute aggregations for each field

Each aggregation will be computed separately within each window.
Ex: if you defined 2 windows, selecting "min" for a column will compute two different aggregations.

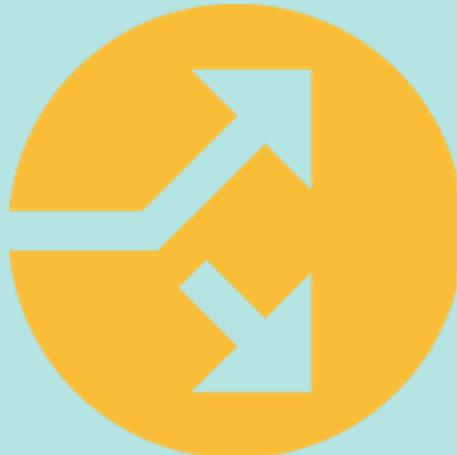
0 / 28 Filter column... Hide unused variables

Column	Type	Operations	Min	Max	Avg	Sum	Concat	Std. dev.	Count	First	Last	Lag	LagDiff	Lead	LeadDiff	
merchant_category_id	bigint	Retrieve	Min	Max	Avg	Sum	Concat	Std. dev.	Count	First	Last	Lag	LagDiff	Lead	LeadDiff	
 purchase_amount	double	Retrieve	Min	Max	Avg	Sum	Concat	Std. dev.	Count	First	Last	Lag	LagDiff	1	Lead	LeadDiff
reward_cashback	double	Retrieve	Min	Max	Avg	Sum	Concat	Std. dev.	Count	First	Last	Lag	LagDiff	Lead	LeadDiff	
reward_travel	double	Retrieve	Min	Max	Avg	Sum	Concat	Std. dev.	Count	First	Last	Lag	LagDiff	Lead	LeadDiff	
reward_dining	double	Retrieve	Min	Max	Avg	Sum	Concat	Std. dev.	Count	First	Last	Lag	LagDiff	Lead	LeadDiff	
purchase_date	date	Retrieve	Min	Max	Avg	Sum	Concat	Std. dev.	Count	First	Last	Lag	LagDiff	Lead	LeadDiff	
client_seniority	bigint	Retrieve	Min	Max	Avg	Sum	Concat	Std. dev.	Count	First	Last	Lag	LagDiff	Lead	LeadDiff	
purchase_date_year	bigint	Retrieve	Min	Max	Avg	Sum	Concat	Std. dev.	Count	First	Last	Lag	LagDiff	Lead	LeadDiff	
purchase_date_month	bigint	Retrieve	Min	Max	Avg	Sum	Concat	Std. dev.	Count	First	Last	Lag	LagDiff	Lead	LeadDiff	
purchase_date_day	bigint	Retrieve	Min	Max	Avg	Sum	Concat	Std. dev.	Count	First	Last	Lag	LagDiff	Lead	LeadDiff	
 purchase_day_of_week	bigint	Retrieve	Min	Max	Avg	Sum	Concat	Std. dev.	Count	First	Last	Lag	LagDiff	Lead	LeadDiff	

Split



Split

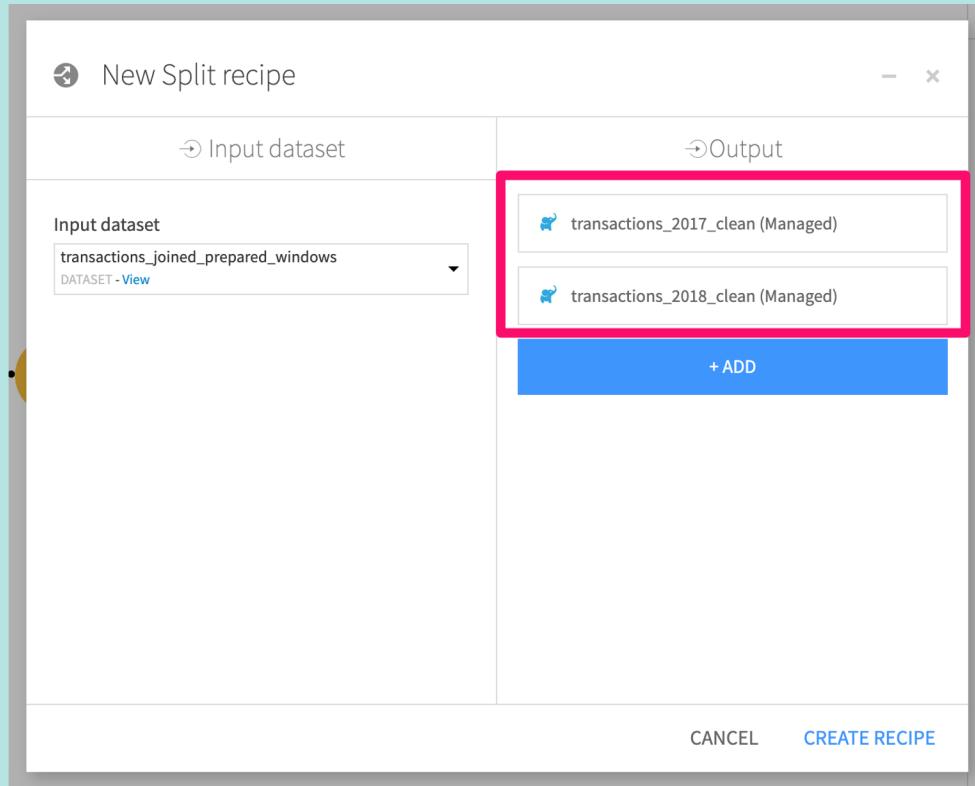


- Split a dataset
 - Randomly
 - Based on some condition

Split the joined dataset into 2017 and 2018 sets

1. Create two new datasets, “transactions_2017_clean” and “transactions_2018_clean”
2. Define a filter
 - where column “authorized_flag” is defined → “transactions_2017_clean”
 - all other rows → “transactions_2018_clean”

Create the two output datasets



Split into 2017 and 2018

split_transactions_joined_prepared_windows

Summary Settings Input / Output Advanced History   

Define a filter for each output dataset 

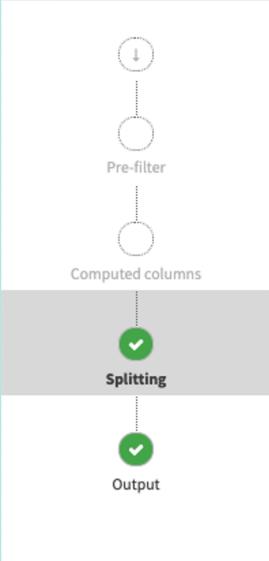
Filters

Keep only rows that satisfy  



All other values 



```
graph TD; A(( )) --> B(( )); B --> C(( )); C --> D(( )); D --> E(( )); E --> F(( )); F --> G(( ));
```

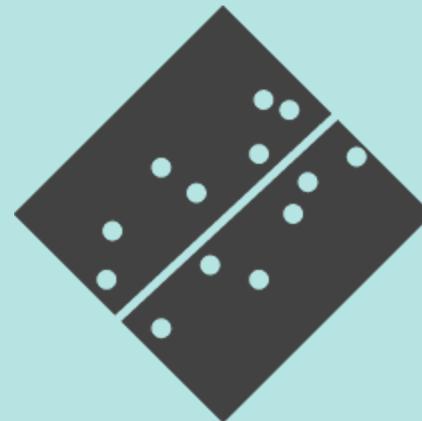
Pre-filter

Computed columns

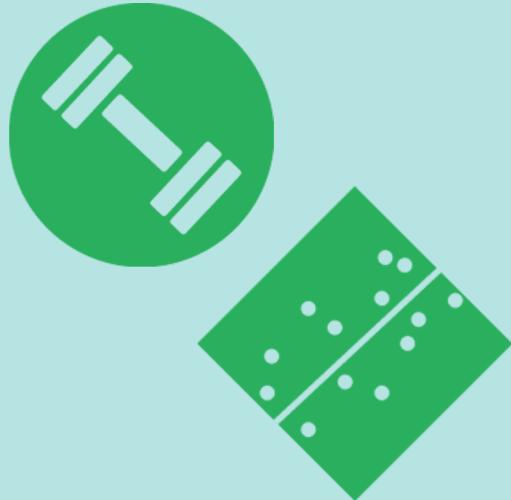
Splitting

Output

Machine Learning



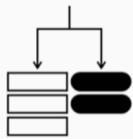
Machine Learning



- Create machine learning models using the visual interface
 - Leverage open source packages like scikit-learn, xgboost, MLLib, Keras, Tensorflow, and H2O
- Perform batch-scoring on new data using a trained model
- Deploy models as REST API endpoints

Types of Machine Learning

Introduction

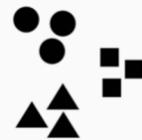


Prediction

Goal: Create a model that can *predict* a target variable

Examples:

- Predict the sales price of an apartment
- Forecast the winner of an election
- Diagnose a disease



Clustering

Goal: Separate data into *clusters* based on similarity

Examples:

- Find groups of similar apartments
- Segment voters into demographic groups
- Group diseases based on symptoms

Types of Machine Learning

Introduction

Choose a type of machine learning
for the following

Predicting mortgage defaults

Forecasting lifetime spending of customer

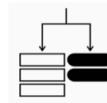
Grouping songs into genres

Predicting amount of snowfall

Segmenting website visitors

Recommending movies to Netflix users

Detecting unusual financial transactions



Prediction



Clustering

✓ (classification)
)

✓ (regression)
)

✓ (regression)
)

✓

✓

✓

✓

✓ (regression)
}
✓ (classification)
)

Types of Data

A Categorical

Data can be one of several categories

Examples:

- Gender
- Nationality
- Hair color

Numerical

Data is a number

Examples:

- Age
- Weight
- Salary

I Text

Data is free-form text

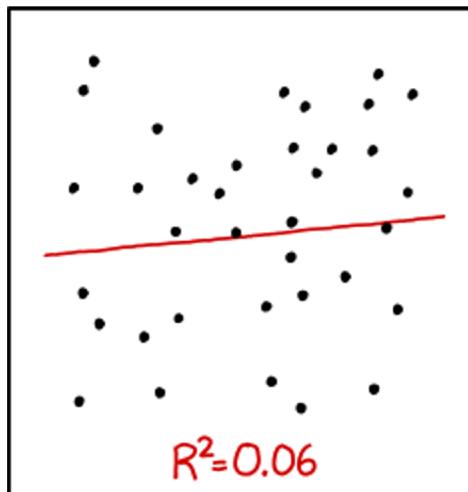
Examples:

- Tweets
- Documents
- Business name

Can you think of others ?

Choosing an algorithm

What is the best way to learn from your data?



I DON'T TRUST LINEAR REGRESSIONS WHEN IT'S HARDER
TO GUESS THE DIRECTION OF THE CORRELATION FROM THE
SCATTER PLOT THAN TO FIND NEW CONSTELLATIONS ON IT.

Building a model

Algorithm, Loss, and Optimization

Algorithm: what do you think is the best structure?

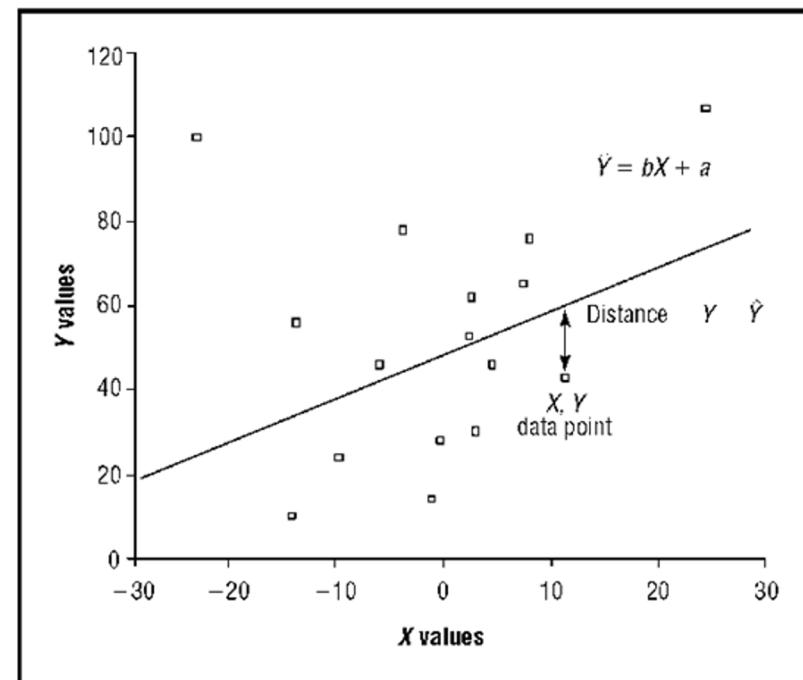
$$y_i = \beta_1 x_{i1} + \cdots + \beta_p x_{ip} + \varepsilon_i = \mathbf{x}_i^T \boldsymbol{\beta} + \varepsilon_i,$$

Loss function: how do you compare the estimated values?

$$S(b) = \sum_{i=1}^n (y_i - x_i^T b)^2$$

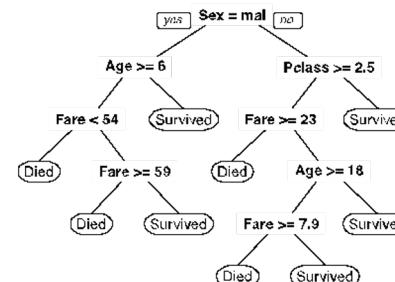
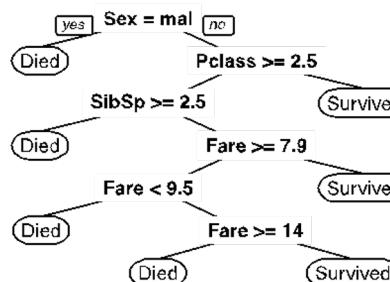
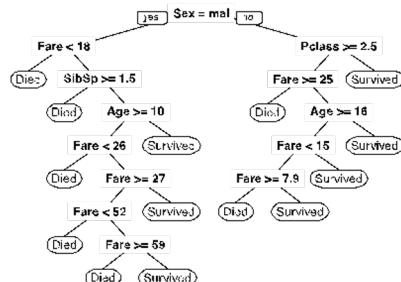
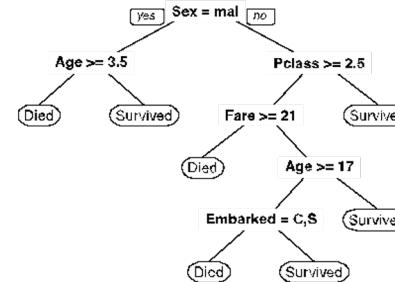
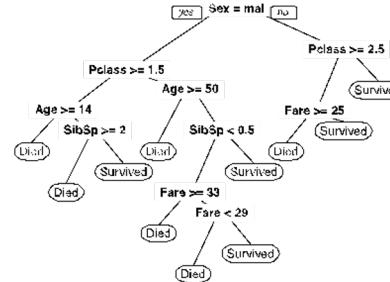
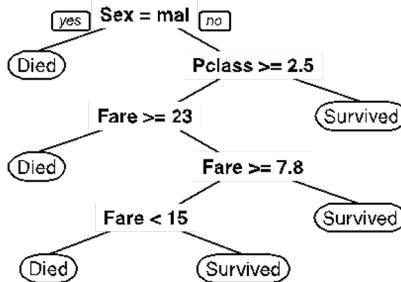
Optimization: how do you solve for your loss function?

Gradient descent....



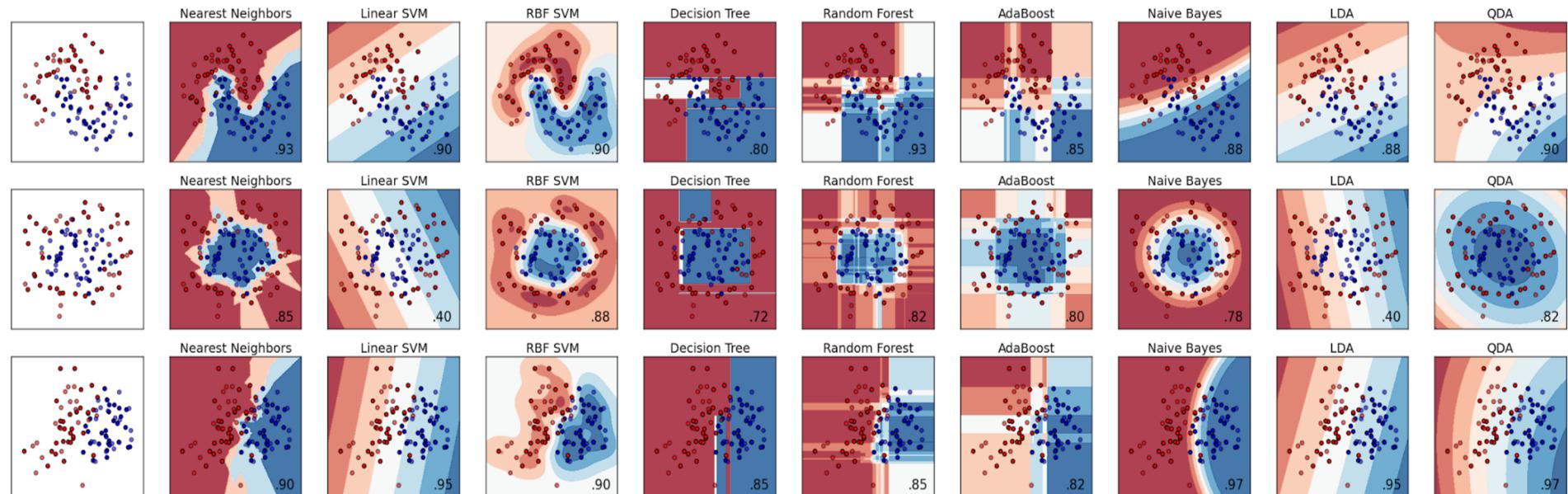
A Random “Forest”

Averaging the vote



Why different algorithms?

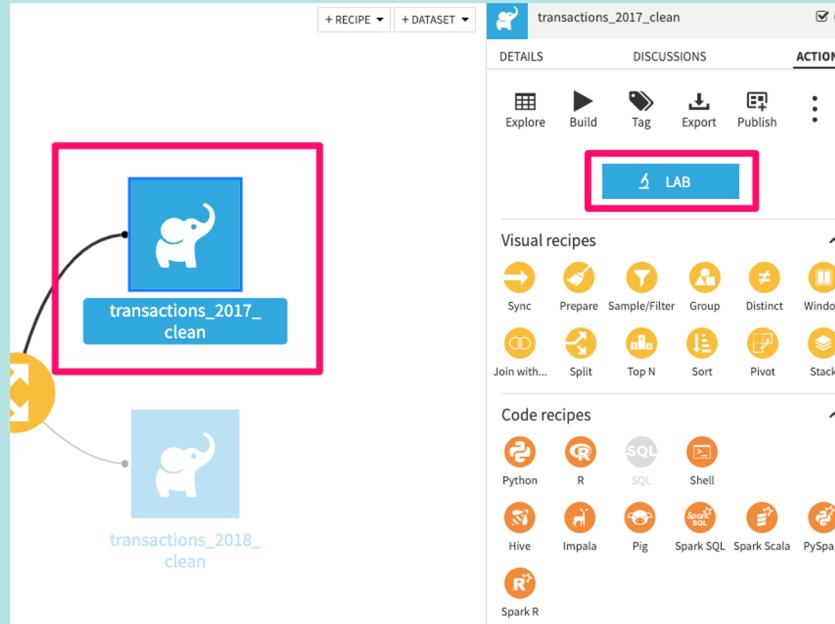
Each algorithm captures a specific pattern



Train a model to predict “authorized_flag”

1. Go to the Lab for the “transactions_2017_clean” dataset
2. Create a new visual model
3. In the Design tab, turn off all columns except:
 - a. lagged features
 - b. time-related features
4. Try different algorithms and hyperparameters
5. Train your models - find the best one

Click on the “transactions_2017_clean” dataset and to go to the Lab



The screenshot shows a data visualization interface. On the left, there are two datasets represented by blue elephant icons: "transactions_2017_clean" and "transactions_2018_clean". A red box highlights the "transactions_2017_clean" icon. On the right, the detailed view for "transactions_2017_clean" is displayed. The top bar includes "+ RECIPE" and "+ DATASET" buttons. The main area has tabs for "DETAILS", "DISCUSSIONS", and "ACTIONS". Under "ACTIONS", there are buttons for "Explore", "Build", "Tag", "Export", "Publish", and a more options menu. A prominent blue button labeled "LAB" is highlighted with a red box. Below this, sections for "Visual recipes" and "Code recipes" are shown, each with several icons and their corresponding names.

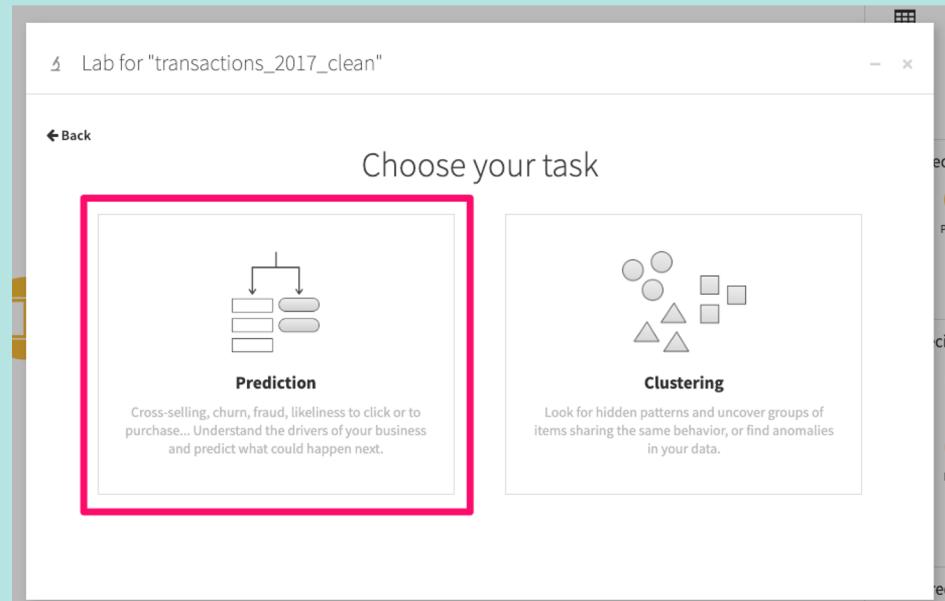
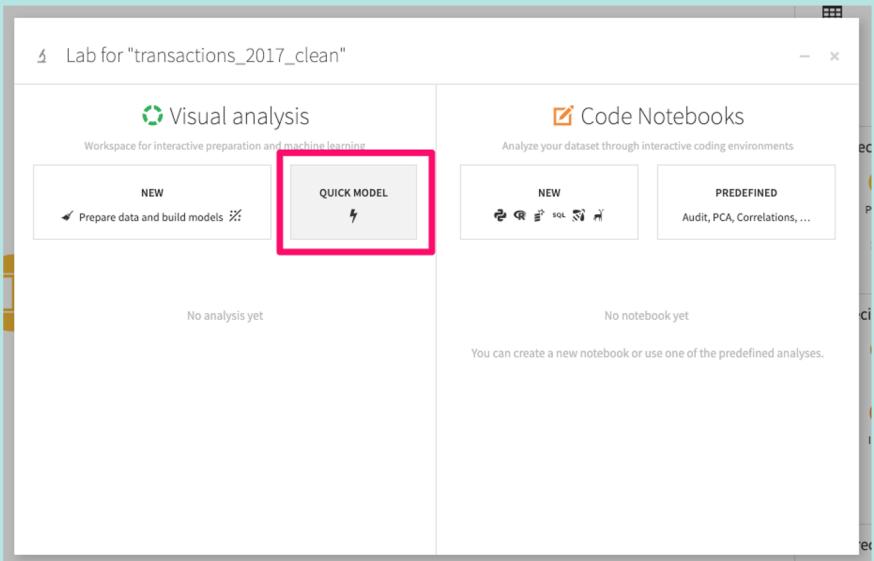
- Visual recipes:

 - Sync
 - Prepare
 - Sample/Filter
 - Group
 - Distinct
 - Window
 - Join with...
 - Split
 - Top N
 - Sort
 - Pivot
 - Stack

- Code recipes:

 - Python
 - R
 - SQL
 - Shell
 - Hive
 - Impala
 - Pig
 - Spark SQL
 - Spark Scala
 - PySpark
 - Spark R

Build a quick prediction model



Choose your own path - I will try automated ML and quick prototypes

Lab for "transactions_2017_clean"

Choose your prediction style

Select your target variable: authorized_flag

Automated Machine Learning
Let Dataiku create your models.

Expert Mode
Have full control over the creation of your models.

Lab for "transactions_2017_clean"

Automated Machine Learning

Quick Prototypes
Get some models, generic and quick.

Engine In-memory ▾

Interpretable Models for Business Analysts
Produce decision tree and simple linear models.

Engine In-memory ▾

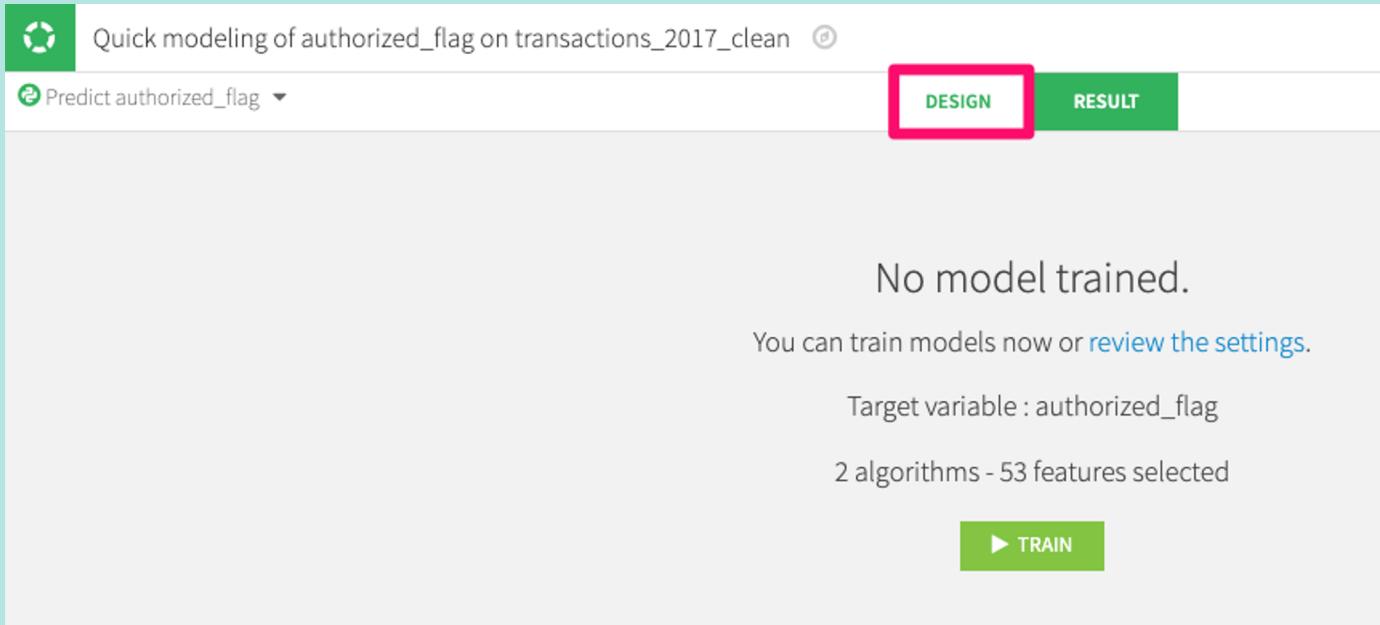
High Performance Models
Be patient and get even more accurate models.

Engine In-memory ▾

Name your analysis: Quick modeling of fraudulent on transaction

CREATE

Click the Design tab to change ML settings



Quick modeling of authorized_flag on transactions_2017_clean ⚙️

Predict authorized_flag ▾

DESIGN **RESULT**

No model trained.

You can train models now or [review the settings](#).

Target variable : authorized_flag

2 algorithms - 53 features selected

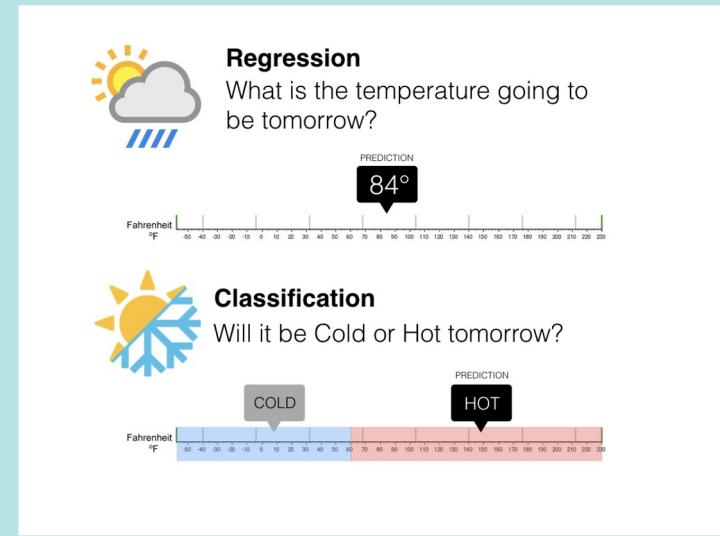
▶ TRAIN

Machine learning concepts

- **Regression** - predict a continuous numeric target variable
- **Classification** - predict a discrete “class” of your target variable

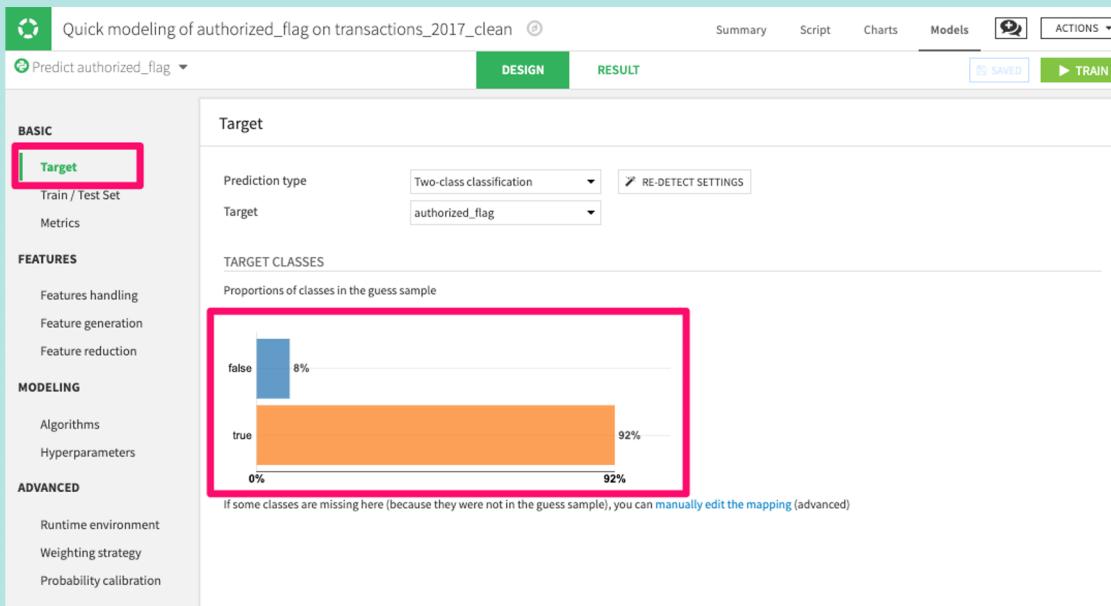
What type of problem is ours?

Classification



Classification - Imbalanced data

Notice that our target column ‘authorized_flag’ is imbalanced - this is problematic



Two things we can try:

1. Rebalance our train/test sets to undersample rows from the more popular class
2. Weight error metrics according to the inverse frequency of each class

Classification - Imbalanced data

Quick modeling of authorized_flag on transactions_2017_clean

DESIGN **RESULT**

BASIC

Target

Train / Test Set (highlighted with a red box)

Metrics

FEATURES

- Features handling
- Feature generation
- Feature reduction

MODELING

- Algorithms
- Hyperparameters

ADVANCED

- Runtime environment
- Weighting strategy
- Probability calibration

Train / test set

Policy: Split the dataset

SAMPLING

If your dataset does not fit in your RAM, you may want to subsample the set on which splitting will be performed.

Sampling method: First records

- No sampling (whole data) (highlighted with a red box)

Takes the whole data
- First records

Takes the first N rows of the dataset. Very fast (only reads N rows) but may result in a very biased view of the dataset.
- Random (approx. ratio)

Randomly selects approximately X% of the rows. Requires a full pass reading the data.
- Random (approx. nb. records)

Randomly selects approximately N rows. Requires 2 full passes.
- Column values subset (approx. nb. records)

Randomly selects a subset of values and chooses all rows with these values, in order to obtain approximately N rows. This is useful for selecting a subset of customers, for example. Requires 2 full passes.
- Class rebalance (approx. nb. records) (highlighted with a red box)

Randomly selects approximately N rows, trying to rebalance equally all modalities of a column. Does not oversample, only undersample (so some rare modalities may remain under-represented). Rebalancing is not exact. Requires 2 full passes.
- Class rebalance (approx. ratio)

Randomly selects approximately X% of the rows, trying to rebalance equally all modalities of a column. Does not oversample, only undersample (so some rare modalities may remain under-represented). Rebalancing is not exact. Requires 2 full passes.

Choose N records from each target class for the train set (highlighted with a pink box)

Choose 100,000 records balanced from the 'authorized_flag' column

SAMPLING

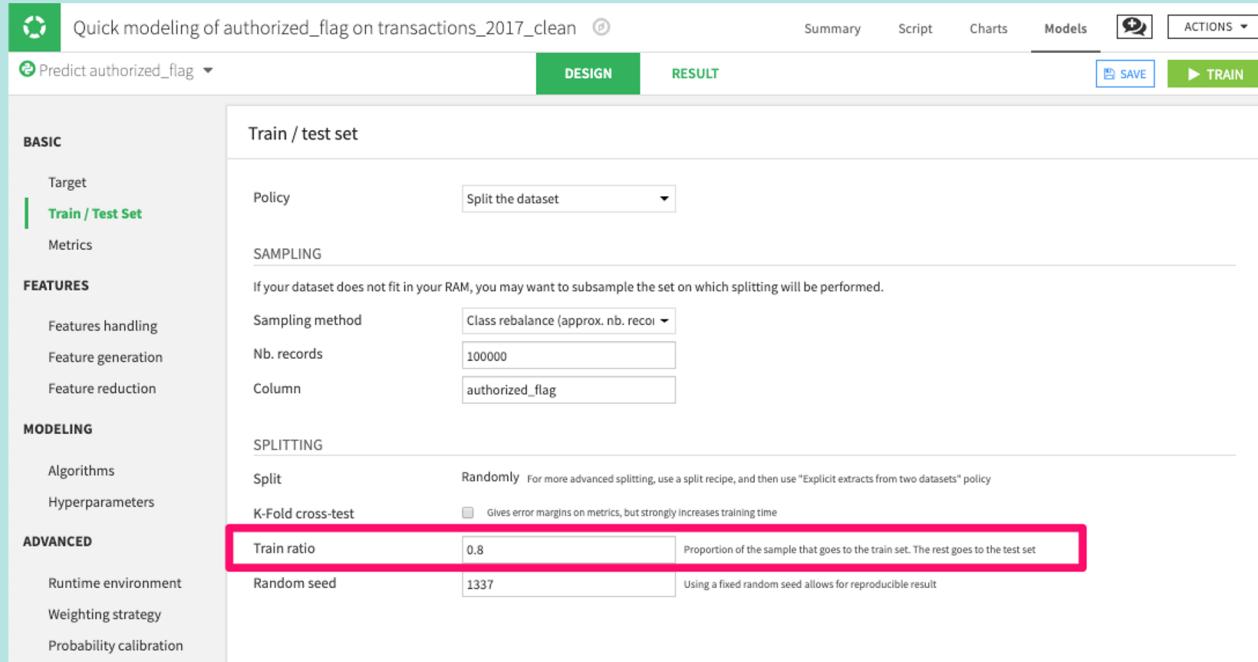
If your dataset does not fit in your RAM, you may want to subsample the set on which splitting will be performed.

Sampling method: Class rebalance (approx. nb. records)

Nb. records: 100000

Column: authorized_flag (highlighted with a red box)

Classification - Imbalanced data



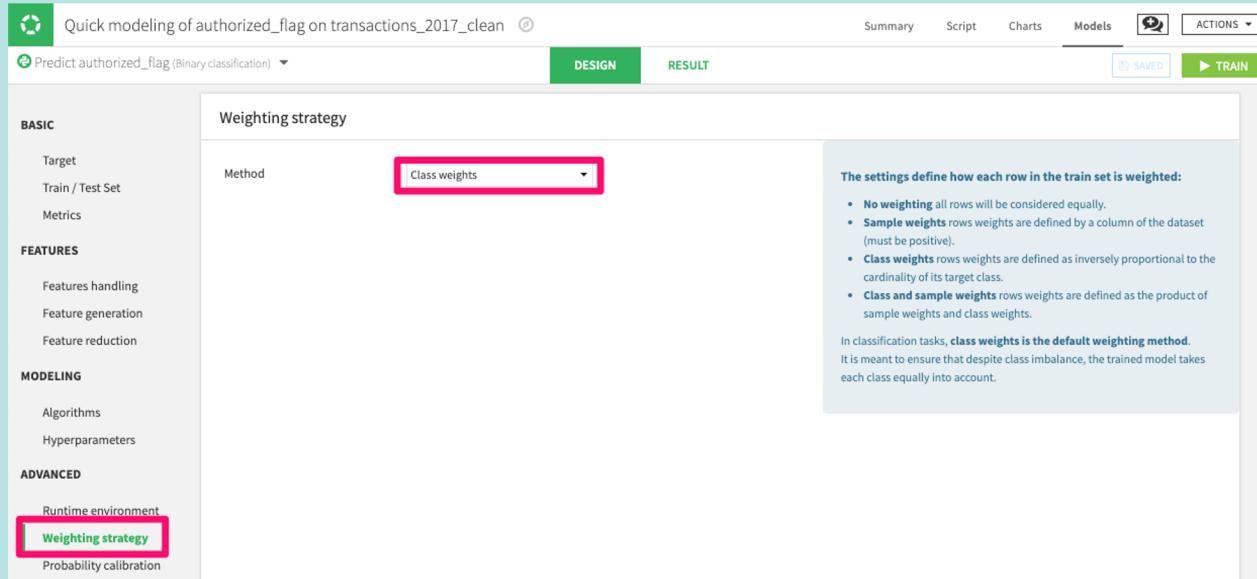
The screenshot shows the dataiku interface with the following details:

- Project:** Quick modeling of authorized_flag on transactions_2017_clean
- Tab:** DESIGN (selected)
- Actions:** Summary, Script, Charts, Models, Help, ACTIONS ▾, SAVE, TRAIN
- BASIC** section:
 - Target: Predict authorized_flag
 - Train / Test Set (selected)
 - Metrics
- FEATURES** section:
 - Features handling
 - Feature generation
 - Feature reduction
- MODELING** section:
 - Algorithms
 - Hyperparameters
- ADVANCED** section:
 - Runtime environment
 - Weighting strategy
 - Probability calibration
- Train / test set** configuration:
 - Policy:** Split the dataset
 - SAMPLING**: If your dataset does not fit in your RAM, you may want to subsample the set on which splitting will be performed.
 - Sampling method: Class rebalance (approx. nb. recor ▾)
 - Nb. records: 100000
 - Column: authorized_flag
 - SPLITTING**:
 - Split:** Randomly. For more advanced splitting, use a split recipe, and then use "Explicit extracts from two datasets" policy
 - K-Fold cross-test:** Gives error margins on metrics, but strongly increases training time
 - Train ratio:** 0.8 (highlighted with a red box)
 - Proportion of the sample that goes to the train set. The rest goes to the test set
 - Random seed:** 1337
 - Using a fixed random seed allows for reproducible result

It's common to split your known data into:

- **80% - data to train the model**
- **20% - hold-out data to test your model performance**

Classification - Imbalanced data



Quick modeling of authorized_flag on transactions_2017_clean

Predict authorized_flag (Binary classification)

DESIGN RESULT ACTIONS

SUMMARY SCRIPT CHARTS MODELS

BASIC

- Target
- Train / Test Set
- Metrics

FEATURES

- Features handling
- Feature generation
- Feature reduction

MODELING

- Algorithms
- Hyperparameters

ADVANCED

- Runtime environment
- Weighting strategy**
- Probability calibration

Weighting strategy

Method: Class weights

The settings define how each row in the train set is weighted:

- No weighting all rows will be considered equally.
- Sample weights rows weights are defined by a column of the dataset (must be positive).
- Class weights rows weights are defined as inversely proportional to the cardinality of its target class.
- Class and sample weights rows weights are defined as the product of sample weights and class weights.

In classification tasks, class weights is the default weighting method. It is meant to ensure that despite class imbalance, the trained model takes each class equally into account.

Look out for blue boxes for tips

You can also weight error calculations based on the target column class (or another column)

- Class weights is generally a good place to start

Feature selection

Choose the columns in our dataset to consider when modeling credit card fraud

Consider:

- **Only include columns which you'll have prior the moment you need to make a prediction (no data leakage)**
- **Intuition and subject matter expertise are important**

You can turn features “On” in the “Features handling” tab

Quick modeling of authorized_flag on transactions_2017_clean

Predict authorized_flag (Binary classification)

DESIGN **RESULT**

BASIC

- Target
- Train / Test Set
- Metrics

FEATURES

- Features handling** (highlighted with a red box)
- Feature generation
- Feature reduction

MODELING

- Algorithms
- Hyperparameters

ADVANCED

- Runtime environment
- Weighting strategy
- Probability calibration

Features Handling

Handling of "installments"

Role	Reject	Input	Variable type	A Categorical
<input checked="" type="checkbox"/> A authorized_flag	<input type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/> # Numerical	
<input checked="" type="checkbox"/> A card_id	<input type="radio"/> Reject	<input type="radio"/>	<input type="radio"/> I Text	
<input checked="" type="checkbox"/> # city_id	<input type="radio"/> Reject	<input type="radio"/>	<input type="radio"/> [] Vector	
<input checked="" type="checkbox"/> A category_1	<input type="radio"/> Reject	<input type="radio"/>		
<input checked="" type="checkbox"/> # installments	<input type="radio"/> Avg-std rescaling	<input checked="" type="radio"/> ON		
<input checked="" type="checkbox"/> A category_3	<input type="radio"/> Reject	<input type="radio"/>		
<input checked="" type="checkbox"/> # merchant_category_id	<input type="radio"/> Reject	<input type="radio"/>		
<input checked="" type="checkbox"/> # month_lag	<input type="radio"/> Reject	<input type="radio"/>		
<input checked="" type="checkbox"/> # purchase_amount	<input type="radio"/> Avg-std rescaling	<input checked="" type="radio"/> ON		
<input checked="" type="checkbox"/> A purchase_date	<input type="radio"/> Reject	<input type="radio"/>		

Numerical handling: Keep as a regular numerical fe.

Rescaling: Standard rescaling

Make derived feats.: Generate sqrt(x), x^2, ... features

Missing values: Impute ...

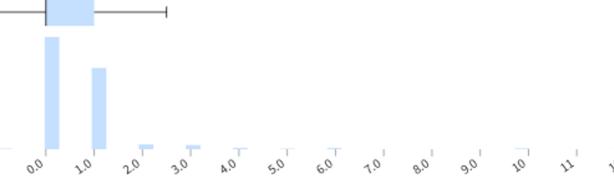
Impute with: Average of values

Distribution

Minimum: -1 Mean: 0.64550 Distinct values: 13 Empty cells: 0.0%

Maximum: 12 StdDev: 1.1765 Mode: 0 Invalid cells: 0.0%

Median: 0



Turn on the following features

(From the point of sale)

- installments
- purchase_amount
- purchase_month
- purchase_dow
- purchase_weekend
- purchase_hour
- state

(From our window function)

- cardholder_installments_avg
- cardholder_purchase_amount_avg
- cardholder_purchase_amount_stddev
- cardholder_purchase_amount_sum
- cardholder_purchase_weekend_avg
- merchant_installments_avg
- merchant_purchase_amount_avg
- merchant_purchase_amount_stddev
- merchant_purchase_amount_sum
- merchant_purchase_weekend_avg

(From the merchant database)

- merchant_avg_sales_lag3
- merchant_avg_purchases_lag3
- merchant_avg_sales_lag6
- merchant_avg_purchases_lag6
- merchant_avg_sales_lag12
- merchant_avg_purchases_lag12

(From our cardholder database)

- days_active
- card_fico_score

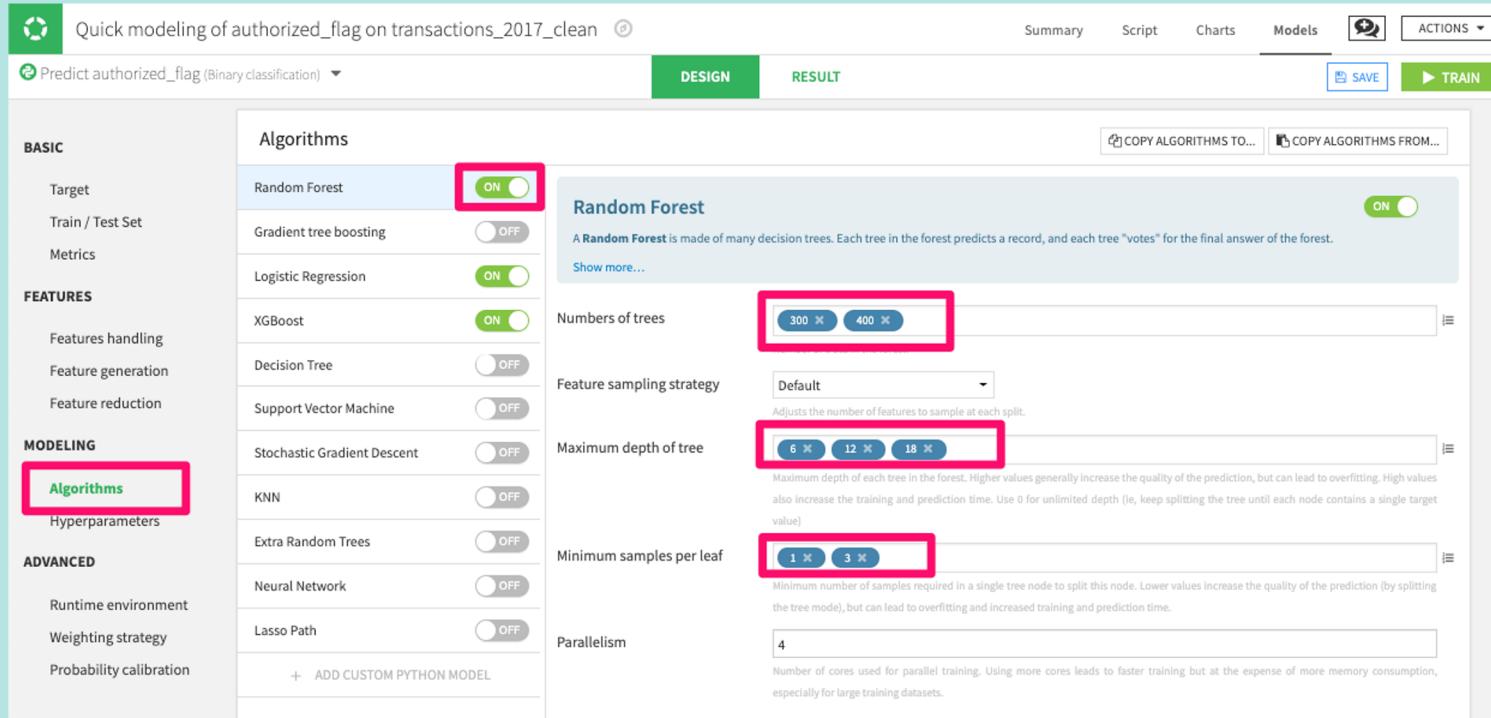
Algorithm selection and hyperparameter tuning

Try out different ML algorithms/hyperparameters - then see which works best

Consider:

- **Try both linear and non-linear models at first**
- **Try wider ranges of hyperparameters first, then hone in on the better performing values (e.g. start with random forest max depths of 6, 12, 18. If 12 performs best, then try 10, 12, 14....and so on)**

Add a Random Forest with these hyperparameters



The screenshot shows the Dataiku DSS interface with the following configuration for a Random Forest model:

- Algorithms:** Random Forest is selected and turned ON.
- Numbers of trees:** Set between 300 and 400.
- Maximum depth of tree:** Set between 6 and 18.
- Minimum samples per leaf:** Set between 1 and 3.
- Other Algorithms:** Gradient tree boosting, XGBoost, Logistic Regression, Decision Tree, Support Vector Machine, Stochastic Gradient Descent, KNN, Extra Random Trees, Neural Network, and Lasso Path are all turned OFF.

Add new hyperparameters
as blue bubbles

DSS will try all possible combinations
and show you the best one

What is a Random Forest?

Random Forest	<input checked="" type="button"/> ON
Gradient tree boosting	<input type="button"/> OFF
Logistic Regression	<input checked="" type="button"/> ON
XGBoost	<input checked="" type="button"/> ON
Decision Tree	<input type="button"/> OFF
Support Vector Machine	<input type="button"/> OFF
Stochastic Gradient Descent	<input type="button"/> OFF

Random Forest

A **Random Forest** is made of many decision trees. Each tree in the forest predicts a record, and each tree "votes" for the final answer of the forest. The forest chooses the class having the most votes.

A decision tree is a simple algorithm which builds a decision tree. Each node of the decision tree includes a condition on one of the input features.

When "growing" (ie, training) the forest:

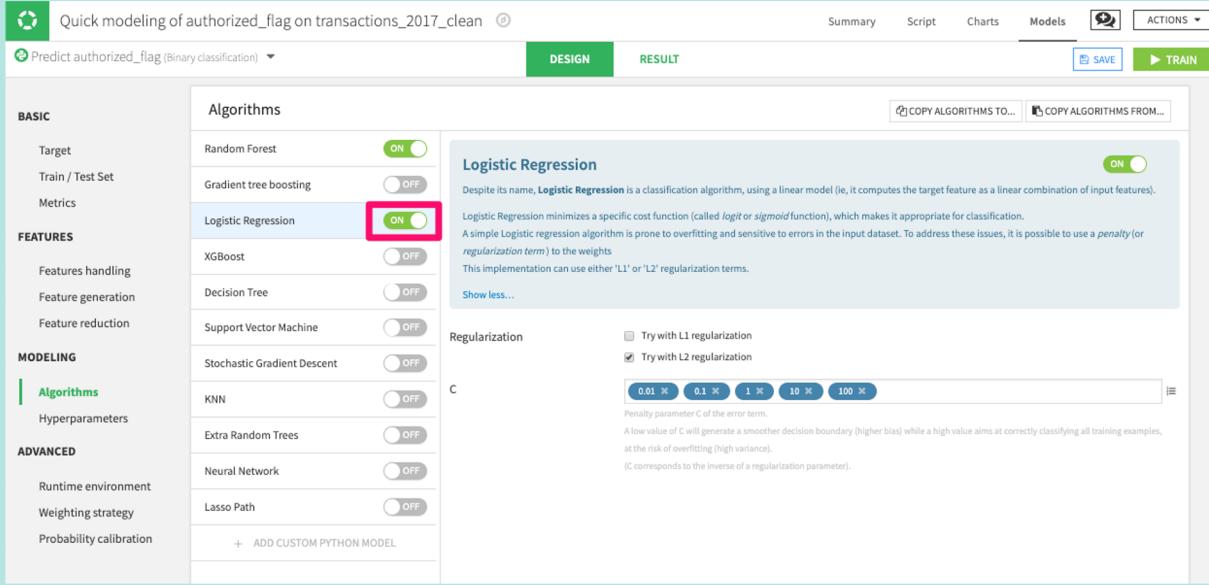
- for each tree, a random sample of the training set is used;
- for each decision point in the tree, a random subset of the input features is considered.

Random Forests generally provide good results, at the expense of "explainability" of the model.

[Show less...](#)

Check out the blue info boxes
for more on each algorithm

Add a Logistic Regression model



The screenshot shows the dataiku interface for creating a machine learning model. The top navigation bar includes 'Summary', 'Script', 'Charts', 'Models', and 'ACTIONS'. Below the title, it says 'Quick modeling of authorized_flag on transactions_2017_clean'. The main area has tabs 'DESIGN' and 'RESULT'. Under 'DESIGN', there's a sidebar with sections: 'BASIC', 'FEATURES', 'MODELING', and 'ADVANCED'. In the 'MODELING' section, 'Algorithms' is selected. The 'Algorithms' list includes: Random Forest (ON), Gradient tree boosting (OFF), Logistic Regression (ON, highlighted with a red box), XGBoost (OFF), Decision Tree (OFF), Support Vector Machine (OFF), Stochastic Gradient Descent (OFF), KNN (OFF), Extra Random Trees (OFF), Neural Network (OFF), and Lasso Path (OFF). There's also a '+ ADD CUSTOM PYTHON MODEL' button. To the right of the algorithm list is a detailed description of 'Logistic Regression'. It explains that despite its name, Logistic Regression is a classification algorithm using a linear model. It notes that while simple, it can overfit and be sensitive to errors. It mentions regularization terms (L1 and L2) and the parameter C. Buttons for 'TRY WITH L1 REGULARIZATION' and 'TRY WITH L2 REGULARIZATION' are shown, with 'TRY WITH L2 REGULARIZATION' checked. A slider for 'C' has values 0.01, 0.1, 1, 10, and 100.

This is a classic algorithm and easier to interpret than others

Algorithms available

DSS supports the following algorithms for visual, in memory ML:

- Prediction algorithms
 - Regression
 - Ordinary Least Squares
 - Ridge Regression
 - Lasso Regression
 - Classification
 - Logistic regression
 - Regression & Classification
 - Random Forests
 - Gradient Boosted Trees
 - XGBoost
 - Decision Tree
 - Support Vector Machine
 - Stochastic Gradient Descent
 - K Nearest Neighbors
 - Extra Random Trees
 - Artificial Neural Network
 - Lasso Path
 - Custom Models
- Clustering algorithms
 - K-means
 - Gaussian Mixture
 - Mini-batch K-means
 - Agglomerative Clustering
 - Spectral Clustering
 - DBSCAN
 - Interactive Clustering (Two-step clustering)
 - Isolation Forest (Anomaly Detection)
 - Custom Models

Train your models

▶ TRAIN

Questions

1. What are the most important variables used in your model?
1. What is your best ROC AUC?
1. Analyze your predictions for cardholders in each state - does your model make drastically higher or lower predictions for any states?

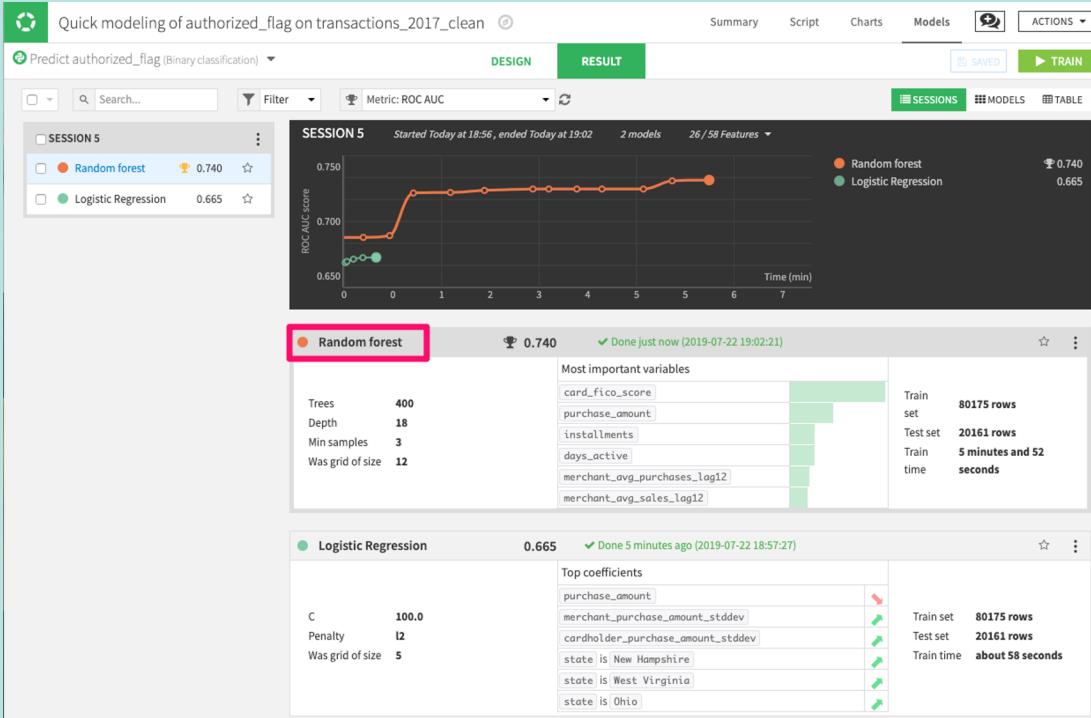
Analyze model performance

Metrics like Accuracy, ROC AUC, F1 score, and mean squared error are important.

Consider:

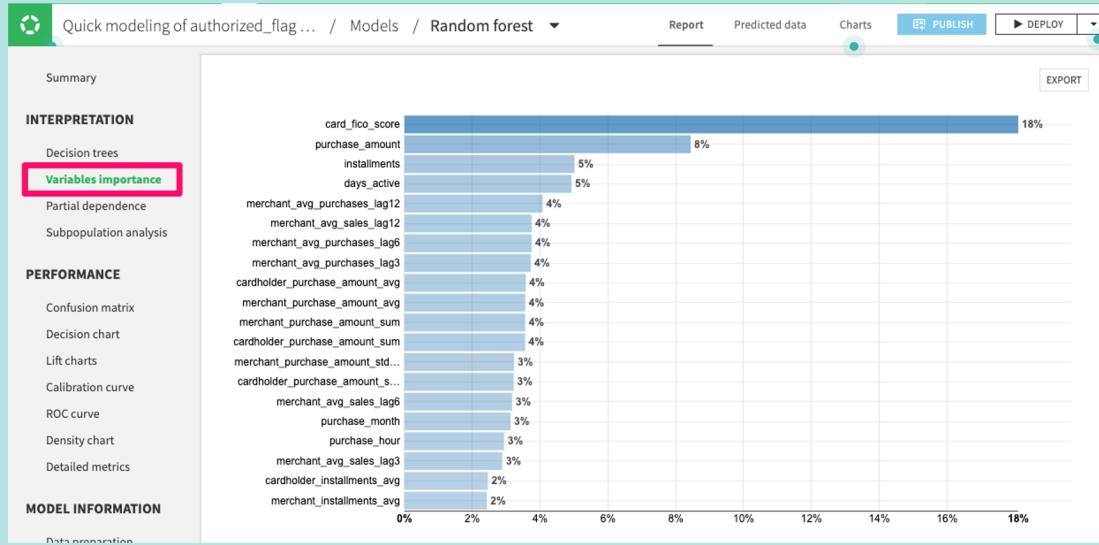
- Look at feature importances (how is each column considered by the model) - do they make sense?
- Look at the confusion matrix - do you care more about false negatives, false positives, true negatives, true positives?

Analyze model performance



Random forest has the highest AUC - let's check it out in detail

Variable importance



FICO score and purchase amount seem to be strong predictors of fraudulent transactions

Tip: some data scientists remove features below a certain importance threshold (e.g. 4%) in the next iteration

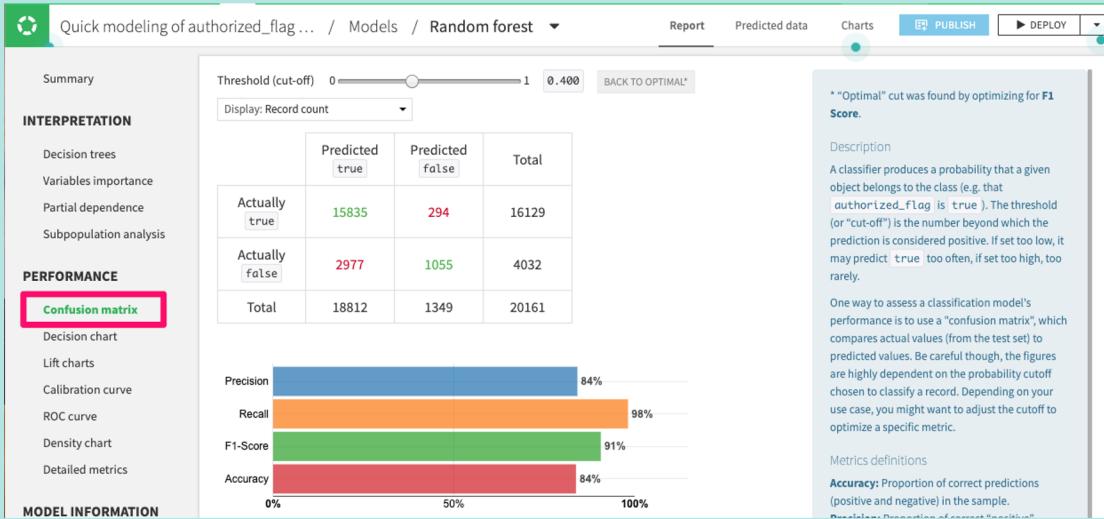
Variable importance

Takeaway:

- **Keep only important features (above 4% importance)**

- card_fico_score
- purchase_amount
- installments
- days_active
- merchant_avg_purchases_lag12
- merchant_avg_sales_lag12
- merchant_avg_purchases_lag6
- merchant_avg_purchases_lag3
- cardholder_purchase_amount_avg
- merchant_purchase_amount_avg
- merchant_purchase_amount_sum
- cardholder_purchase_amount_sum

Confusion matrix



This matrix shows the counts of true positives, true negatives, false positives, and false negatives from the holdout test set

Tip: consider your problem. Do you care more about catching all positives? Do you care about raw accuracy?

- Toggle the above threshold to change model sensitivity

Subpopulation analysis

Quick modeling of authorized_flag ... / Models / Random forest ▾

Report Predicted data Charts PUBLISH DEPLOY

INTERPRETATION

- Decision trees
- Variables importance
- Partial dependence
- Subpopulation analysis**

PERFORMANCE

- Confusion matrix
- Decision chart
- Lift charts
- Calibration curve
- ROC curve
- Density chart
- Detailed metrics

MODEL INFORMATION

- Data preparation
- Features
- Algorithm
- Grid-search optimization
- Training information

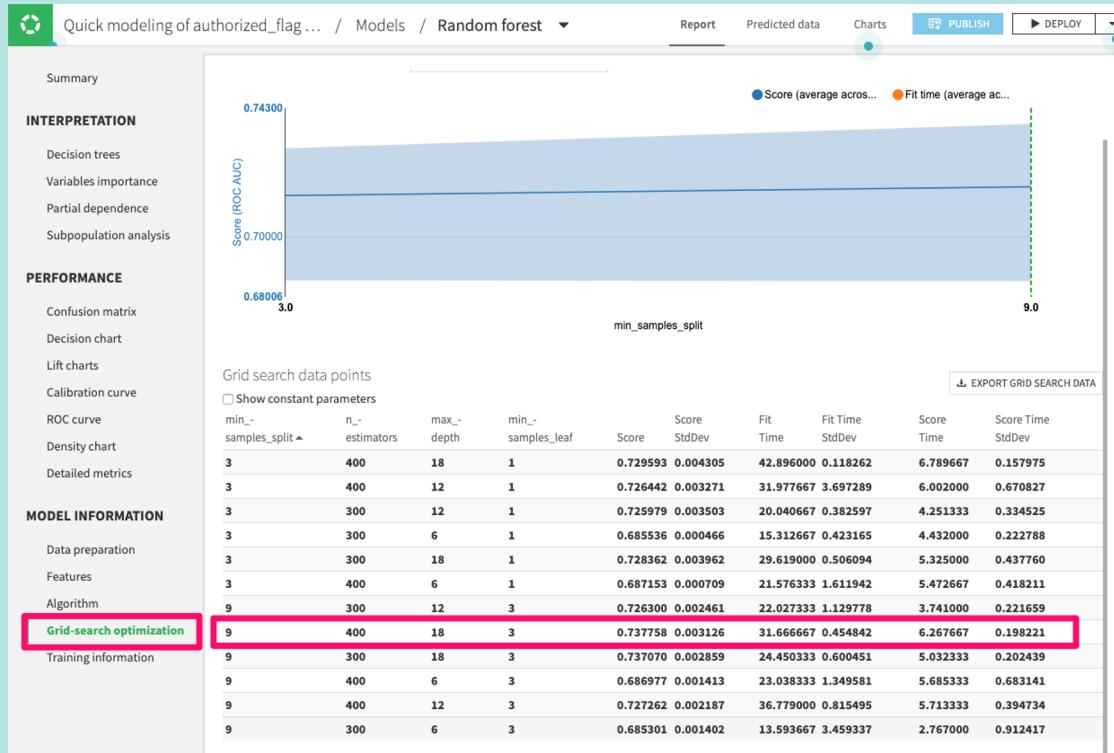
Select your variable A state COMPUTE

11 modalities for state (model input), computed on 20161 rows (test set). DISPLAYED METRICS ▾

Modality	Actually true	Predicted true	Metric: ROC AUC	Precision	Recall	Accuracy
100 %	80.0 %	93.3 %	0.740	0.8417	0.982	0.8378
15 % — Missing values	78.7 % -1.3%	91.5 % -1.8%	0.743	0.8337	0.970	0.8239
5 % — Montana	81.1 % +1.1%	96.2 % +2.9%	0.694	0.8314	0.986	0.8265
5 % — California	81.0 % +1%	95.6 % +2.3%	0.723	0.8445	0.997	0.8492
4 % — Oregon	78.7 % -1.3%	91.4 % -1.9%	0.826	0.8552	0.993	0.8622
4 % — Nevada	78.1 % -1.9%	94.1 % +0.8%	0.720	0.8243	0.993	0.8295
4 % — Michigan	81.8 % +1.8%	97.0 % +3.7%	0.723	0.8378	0.994	0.8375
4 % — Arizona	77.6 % -2.4%	91.2 % -2.1%	0.754	0.8243	0.969	0.8158
3 % — Wyoming	76.9 % -3.1%	92.4 % -0.9%	0.730	0.8201	0.985	0.8221
3 % — Colorado	81.0 % +1%	94.3 % +1%	0.703	0.8391	0.977	0.8293
3 % — North Dakota	81.8 % +1.8%	96.3 % +3%	0.731	0.8404	0.989	0.8375
51 % — Rest of 'state'	80.5 % +0.5%	93.2 % -0.1%	0.741	0.8482	0.981	0.8435

Make sure your model doesn't perform significantly different for different subpopulations

Hyperparameter tuning



Find the model configuration which had the highest score

For me, it was:

- **n_estimators = 400**
- **max_depth = 18**
- **min_samples_leaf = 3**

Hyperparameter tuning

Takeaway:

- Try a new grid search with hyperparameters closer to your previous best configuration

My best model:

- **n_estimators = 400**
- **max_depth = 18**
- **min_samples_leaf = 3**

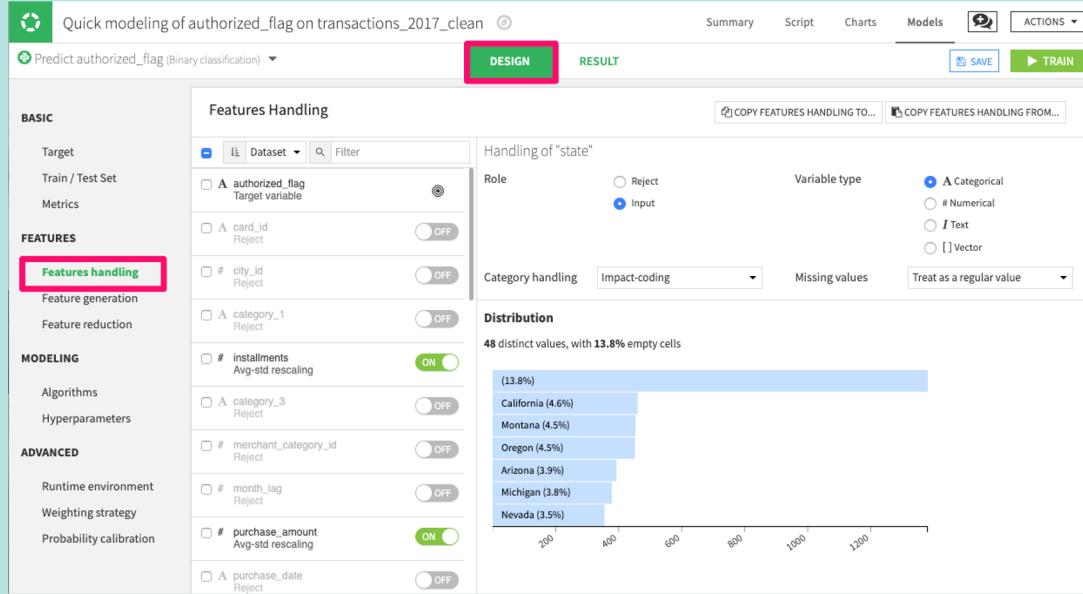
Next time try:

- **n_estimators = 400, 350, 450**
- **max_depth = 18, 16, 20**
- **min_samples_leaf = 3, 5**

Now improve your model

- 1. Remove unimportant features**
- 2. Add in ‘signature_provided’ feature**
- 3. Add back in ‘state’ - change the category handling from “dummy encoding” to “impact coding”**
- 4. Change your random forest grid search to new hyperparameters**

Remove unimportant features



The screenshot shows the dataiku interface with the 'DESIGN' tab selected. On the left, a sidebar lists sections: BASIC, FEATURES (with 'Features handling' highlighted), and MODELING/ADVANCED. The main area is titled 'Features Handling' and contains a 'Handling of "state"' section. It shows a list of features with their roles (e.g., 'authorized_flag' is 'Target variable', 'card_id' is 'Reject', etc.). A 'Role' dropdown is set to 'Input'. The 'Variable type' is set to 'A Categorical'. Below this is a 'Category handling' dropdown set to 'Impact-coding' and a 'Missing values' dropdown set to 'Treat as a regular value'. A distribution chart titled '(13.8%)' shows the percentage of distinct values for each state: California (4.6%), Montana (4.5%), Oregon (4.5%), Arizona (3.9%), Michigan (3.8%), and Nevada (3.5%).

Go back to the design tab -> features handling

Turn on 'signature_provided'

Quick modeling of authorized_flag on transactions_2017_clean

Predict authorized_flag (Binary classification)

DESIGN RESULT

BASIC

Target

Train / Test Set

Metrics

FEATURES

Features handling

Feature generation

Feature reduction

Features Handling

Dataset: Reject

merchant_id Reject

signature_provided Avg-std rescaling

card_first_active_month Reject

card_first_active_month_parsed Reject

Handling of "signature_provided"

Role: Input

Variable type

Numerical handling: Keep as a regular numerical fe.

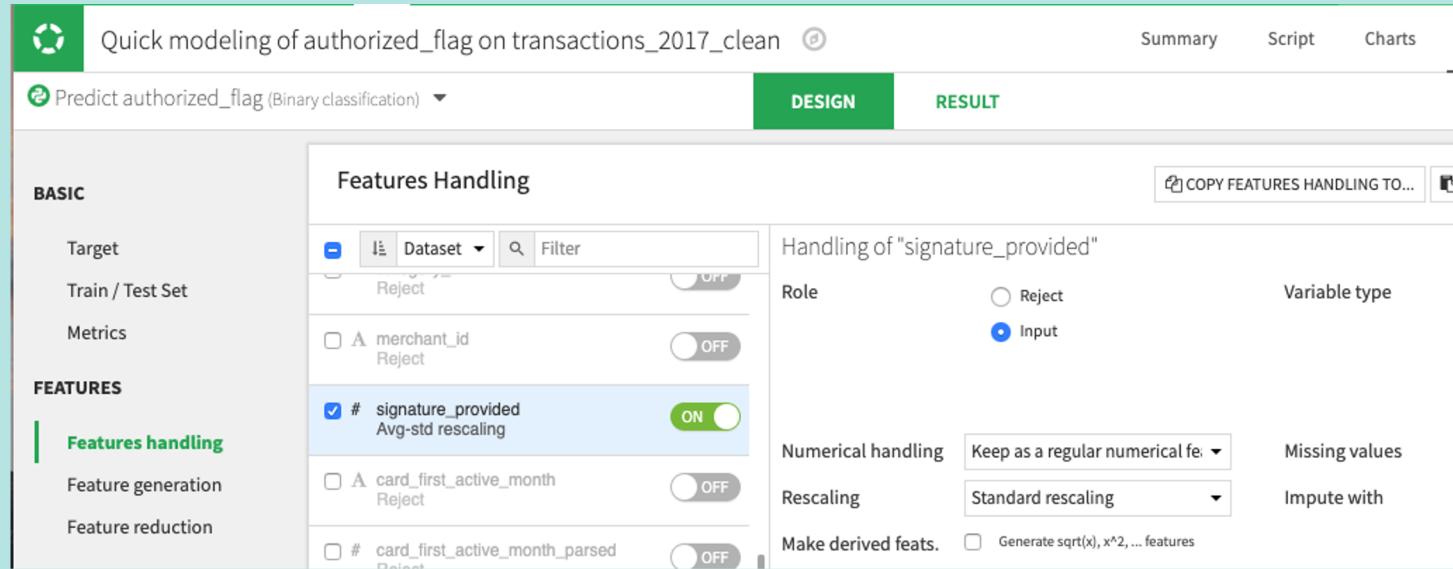
Rescaling: Standard rescaling

Missing values

Impute with

Make derived feats.: Generate sqrt(x), x^2, ... features

COPY FEATURES HANDLING TO...



Perhaps this is a new helpful variable

Keep ‘state’

The screenshot shows a user interface for a machine learning model titled "Quick modeling of authorized_flag on transactions_2017_clean". The interface has tabs for DESIGN and RESULT, with DESIGN selected. In the DESIGN tab, under the FEATURES section, there is a "Features handling" panel. The "state" feature is highlighted with a red box. Its role is set to "Input" and its variable type is "A Categorical". The category handling dropdown is set to "Impact-coding", also highlighted with a red box. Below this, a distribution chart shows the percentage of distinct values for each state: California (4.6%), Montana (4.5%), Oregon (4.5%), Arizona (3.9%), Michigan (3.8%), and Nevada (3.5%).

Change the handling to “Impact coding”

- This will reduce the sparsity of your dataset

Change the hyperparameter grid for Random Forest

The screenshot shows a user interface for a machine learning model. On the left, a sidebar lists categories: **BASIC** (Target, Train / Test Set, Metrics), **FEATURES** (Features handling, Feature generation, Feature reduction), **MODELING** (Algorithms, Hyperparameters), and **ADVANCED** (Runtime environment, Weighting strategy, Probability calibration). The **Algorithms** section is currently active, indicated by a green bar.

In the main area, under the **Algorithms** heading, "Random Forest" is selected (indicated by an **ON** toggle switch) and highlighted with a red box. Other options like Gradient tree boosting, Logistic Regression, XGBoost, Decision Tree, Support Vector Machine, Stochastic Gradient Descent, KNN, Extra Random Trees, Neural Network, and Lasso Path are listed with their respective toggle switches.

The **Random Forest** configuration panel contains the following settings:

- Numbers of trees:** A row of three buttons showing values 400, 350, and 450, each with a red box around it.
- Feature sampling strategy:** Set to "Default".
- Maximum depth of tree:** A row of three buttons showing values 18, 16, and 20, each with a red box around it.
- Minimum samples per leaf:** A row of two buttons showing values 3 and 5, each with a red box around it.
- Parallelism:** Set to 4.

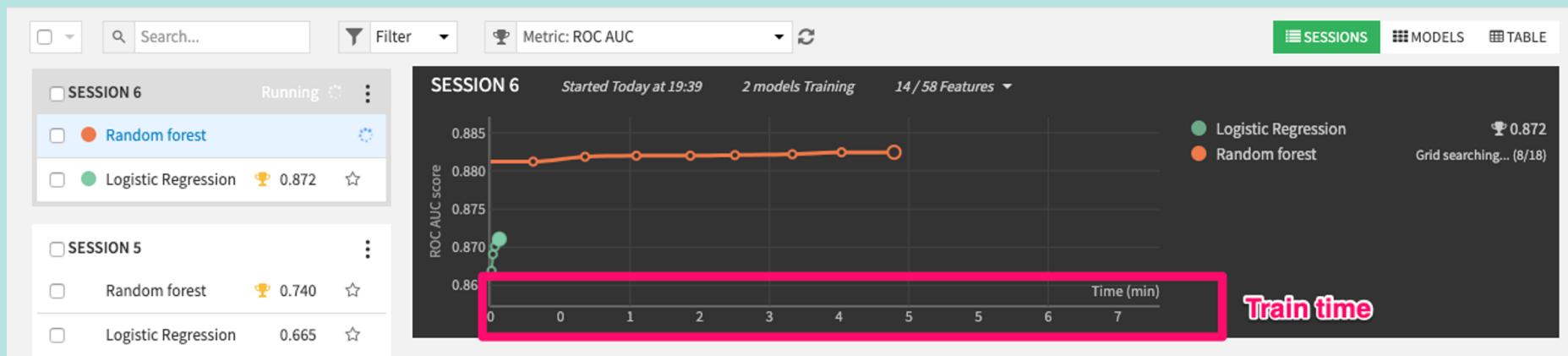
At the top of the interface, there are tabs for **DESIGN** and **RESULT**, and buttons for **SAVE** and **TRAIN**. Above the tabs, there are links for **Summary**, **Script**, and **Charts**. To the right of the tabs, there is a **ACTIONS** dropdown menu.

**Then train your models
again**

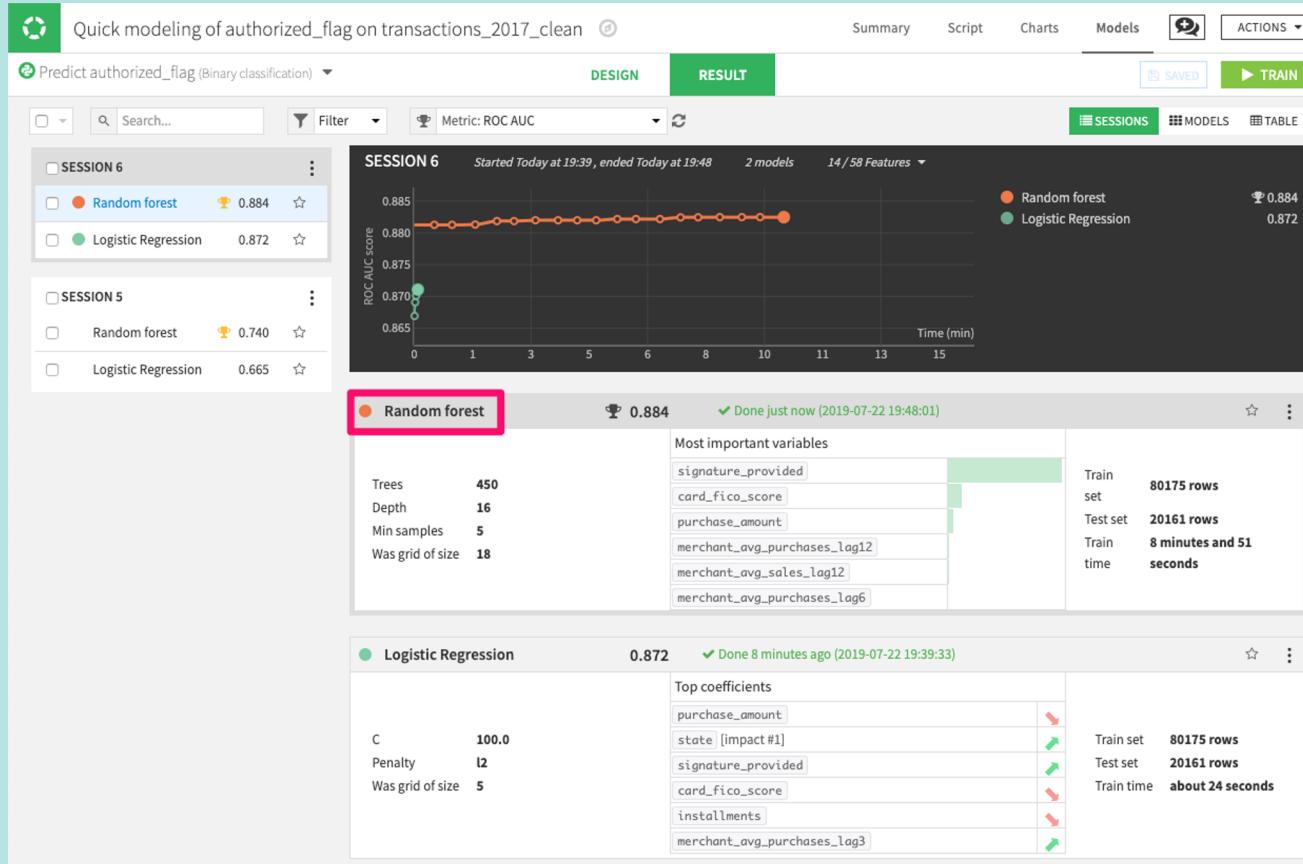
▶ TRAIN

Machine learning is an iterative process

Consider model training time



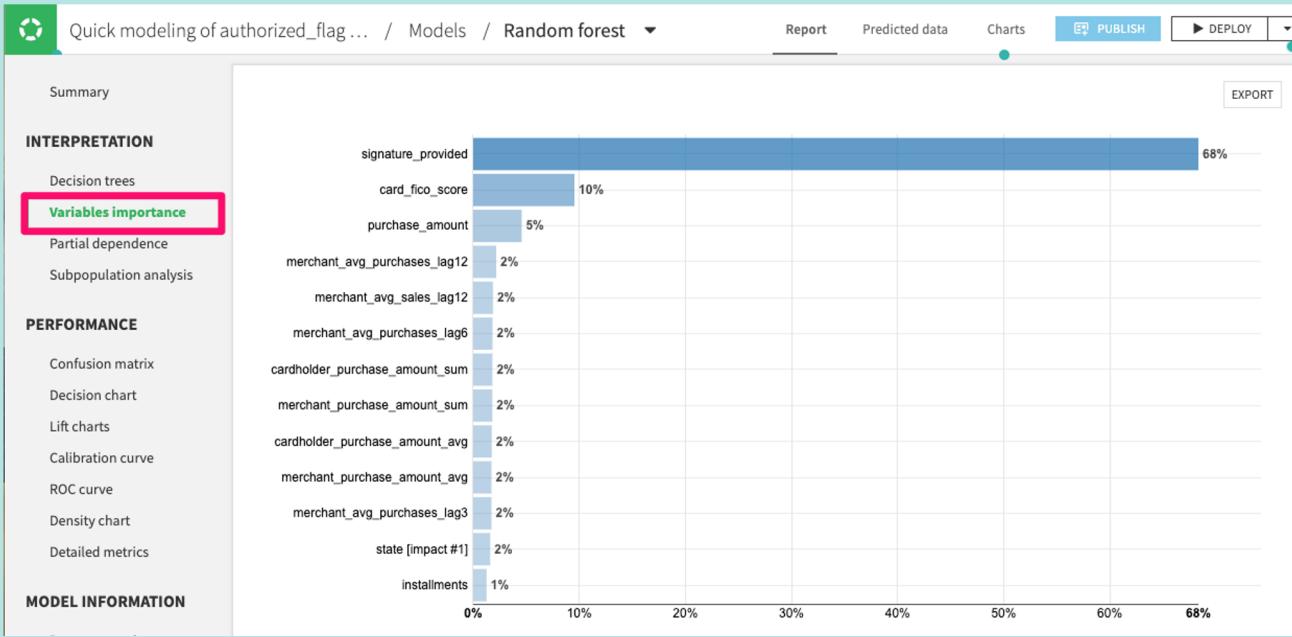
Analyze the Random Forest model again



We can see an improvement over the previous session

(larger AUC is better. AUC = 1 is a perfect model)

Variable importance



‘signature_provided’ is the most important feature in this model by far

Confusion matrix

Quick modeling of authorized_flag ... / Models / Random forest ▾

Report Predicted data

Summary

INTERPRETATION

- Decision trees
- Variables importance
- Partial dependence
- Subpopulation analysis

PERFORMANCE

Confusion matrix

Decision chart

Lift charts

Calibration curve

ROC curve

Density chart

Detailed metrics

MODEL INFORMATION

Threshold (cut-off) 0 — 1 0.400 BACK TO OPTIMAL*

Display: Record count

	Predicted true	Predicted false	Total
Actually true	16086	43	16129
Actually false	1331	2701	4032
Total	17417	2744	20161

Precision: 92%
Recall: 100%
F1-Score: 96%
Accuracy: 93%

This model caught 16,086 out of 16,129 fraudulent transactions in the hold out test set

A great model!

Go back to the Results tab and analyze the Logistic Regression

Quick modeling of authorized_flag on transactions_2017_clean

Predict authorized_flag (Binary classification)

DESIGN RESULT ACTIONS ▾

SESSION 6

- Random forest 0.884
- Logistic Regression 0.872

SESSION 5

- Random forest 0.740
- Logistic Regression 0.665

SESSION 6 Started Today at 19:39, ended Today at 19:48 2 models 14 / 58 Features ▾

ROC AUC score vs Time (min)

Random forest 0.884

Logistic Regression 0.872

Random forest 0.884 Done 6 minutes ago (2019-07-22 19:48:01)

Most important variables

Trees	450
Depth	16
Min samples	5
Was grid of size	18
signature_provided	
card_fico_score	
purchase_amount	
merchant_avg_purchases_lag12	
merchant_avg_sales_lag12	
merchant_avg_purchases_lag6	

Train set 80175 rows

Test set 20161 rows

Train time 8 minutes and 51 seconds

Logistic Regression 0.872 Done 15 minutes ago (2019-07-22 19:39:33)

Top coefficients

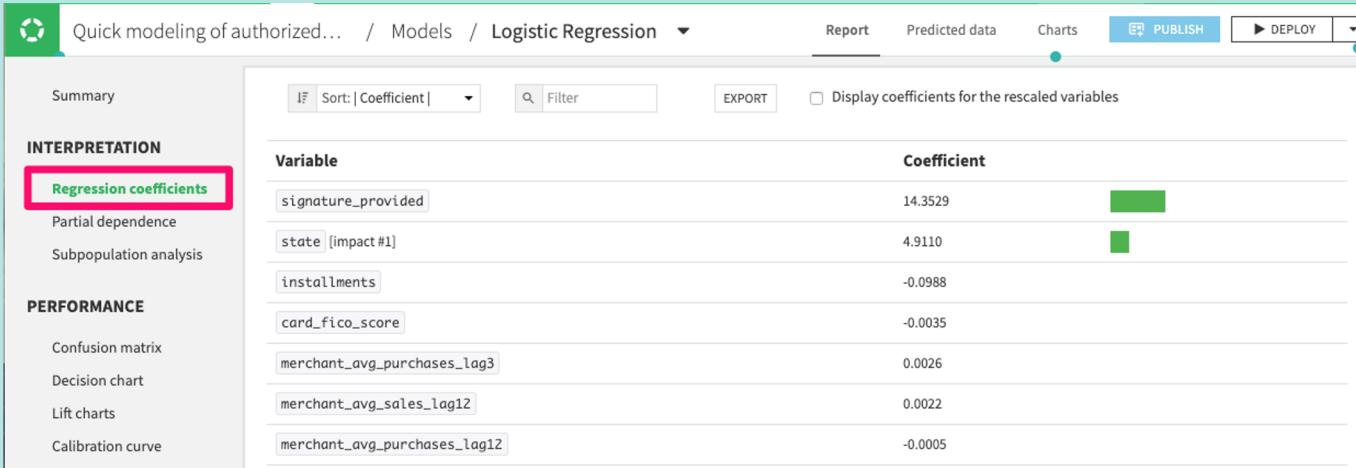
C	100.0
Penalty	l2
Was grid of size	5
purchase_amount	
state [impact #1]	
signature_provided	
card_fico_score	
installments	
merchant_avg_purchases_lag3	

Train set 80175 rows

Test set 20161 rows

Train time about 24 seconds

Regression coefficients



Variable	Coefficient
signature_provided	14.3529
state [impact #1]	4.9110
installments	-0.0988
card_fico_score	-0.0035
merchant_avg_purchases_lag3	0.0026
merchant_avg_sales_lag12	0.0022
merchant_avg_purchases_lag12	-0.0005

Logistic regression is a linear model, so we can see both the magnitude and directional impact of different variables on our target

- In this model, both ‘signature_provided’ and the impact-encoded ‘state’ variable have strong positive effects on fraud. All other features are hardly impactful

Choose a model and deploy it to your project flow

Consider:

- **Random Forest has a higher AUC (in my case)**
- **Logistic regression is easier to interpret (both the magnitude and direction of variable relationships)**

Deploy your model to the flow

Quick modeling of fraudulent on transactions_2017_... / Models / XGBoost ▾

Report Predicted data Charts PUBLISH ▶ DEPLOY EXPORT

Summary

INTERPRETATION

Variables importance

Partial Dependencies

PERFORMANCE

Confusion matrix

Decision chart

Lift charts

Calibration curve

ROC curve

Density chart

Detailed metrics

MODEL INFORMATION

Data preparation

Features

Algorithm

A horizontal bar chart titled "Variables importance" showing the percentage contribution of various features to the model's predictions. The x-axis represents the percentage from 0% to 6%. The y-axis lists features. The most important feature is "purchase_amount" at 6%, followed by "cardholder_purchase_amount_sum" and "purchase_date_parsed" both at 5%, and several other features contributing 4% or 3%.

Feature	Importance (%)
purchase_amount	6%
cardholder_purchase_amount_sum	5%
purchase_date_parsed	5%
cardholder_purchase_amount_s...	5%
days_active	5%
cardholder_purchase_amount_avg	5%
cardholder_fraudulent_avg	5%
merchant_purchase_amount_avg	4%
merchant_installments_avg	4%
merchant_fraudulent_avg	4%
card_fico_score	4%
merchant_latitude	4%
merchant_purchase_amount_sum	4%
merchant_purchase_amount_std...	4%
cardholder_installments_avg	3%
merchant_longitude	3%
purchase_hour	3%
merchant_avg_purchases_lag3	2%
merchant_avg_purchases_lag12	2%
merchant_fraudulent_sum	2%

Deploy your model to the flow

Deploy prediction model

No existing training recipe matches this prediction type, target & ML backend.
You can deploy this model as a new training recipe.

Train dataset: transactions_2017_clean
DATASET - View

Model name: Prediction (XGBOOST_CLASSIFICATI^{...})

ADVANCED CANCEL CREATE

Use your model to score the 2018 transactions

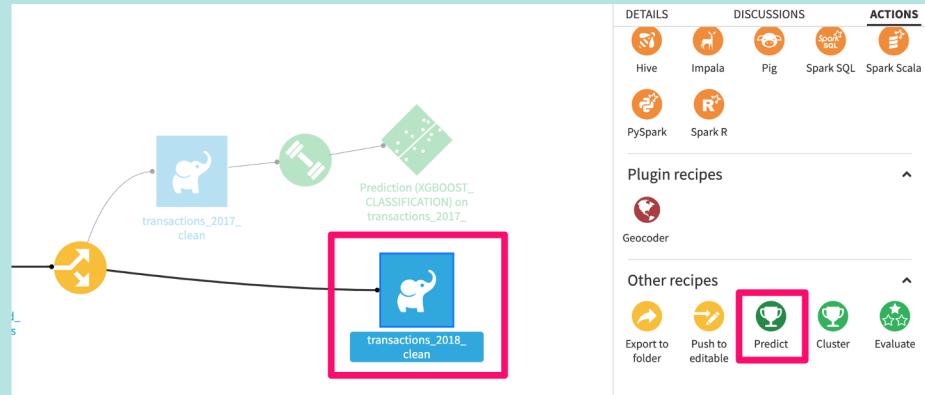
The diagram illustrates a data pipeline in Dataiku. It starts with a yellow icon representing a source dataset, which feeds into a blue square labeled "transactions_2017_clean". This dataset then flows through a green circle icon representing a transformation step, followed by a green diamond icon representing a machine learning model named "Prediction (XGBOOST_CLASSIFICATION) on transactions_2017_". The final output is a blue square labeled "transactions_2018_clean", which is highlighted with a red border.

DETAILS	DISCUSSIONS	ACTIONS
Hive	Impala	Pig
PySpark	Spark R	Spark SQL
Geocoder		Spark Scala

Plugin recipes

- Export to folder
- Push to editable
- Predict**
- Cluster
- Evaluate

Use your model to score the 2018 transactions



Score a dataset

Input dataset Output dataset

Input dataset: transactions_2018_clean
DATASET - View

Prediction Model:

- Prediction (XGBOOST_CLASSIFICATION) on transaction
- Prediction (XGBOOST_CLASSIFICATION) on transactions_2017_clean

Name: transactions_2018_clean_scored

Store into: hdfs_managed

CSV (hive compatible)

NEW DATASET | USE EXISTING DATASET

CANCEL CREATE RECIPE

The "Prediction Model" section is highlighted with a red rectangle, and the "hdfs_managed" dropdown in the "Store into" field is also highlighted with a red rectangle.

Your scored dataset will have new prediction columns

The screenshot shows a data science interface with a top navigation bar featuring tabs like 'Explore' (which is selected), 'Summary', 'Charts', 'Status', 'History', 'Settings', and 'Actions'. Below the navigation is a search bar and a message center. A yellow button labeled 'PARENT RECIPE' is visible. The main area is titled 'Viewing dataset sample' and shows '10000 rows, 61 cols'. A green bar highlights the first row of the table. A red box highlights the last four columns: 'proba_0', 'proba_1', 'prediction', and 'score'. The 'prediction' column is defined as a string type, while 'score' is an integer type.

nt_avg	merchant_purchase_amount_stddev	merchant_purchase_amount_sum	proba_0	proba_1	prediction	score
5863655373	166.33294844302412	175400.37999999968	0.9126831701816002	0.0873168298183998	string	0
1910112382	225.20084821397327	37052.85000000006	0.9522850256811692	0.047714974318830795	string	0
2723358451	21.155796487130235	20664.98000000001	0.9700175607269573	0.02998243927304264	string	0
8817204302	21.145690382282336	20680.12000000001	0.9393304734831462	0.06066952651685379	string	0
1104268162	592.21865209585	2.1289666399996206E7	0.9336796544986138	0.06632034550138617	string	0
1352154524	231.0625027316577	119836.60999999994	0.9499475316399966	0.05005246836000349	string	0
80.1521875	125.70928404481701	5764.87	0.9644601023844019	0.03553989761559805	string	0
9597248923	589.6357874440416	2.184791722999604E7	0.9511400640741698	0.04885993592583014	string	0
4432774904	588.1761513421206	2.34700930099957E7	0.9629459933421465	0.03705400665785353	string	0

Flow check

