

[Hands - On] Introduction to R, Python, and Flow



Amy Wang
amy@h2o.ai

H2O Prerequisites

To Launch H2O and Flow the only prerequisite is:

- 64 bit Java 6+

Consequences of not having 64 bit version:

- You will not be able to launch a cluster larger than 1 GB

Please install it from USB drive from the appropriate Windows or Mac Directory!

Flow Users

- Web-based UI that comes prepackaged in H2O
- To launch Flow you simply have to launch H2O either through the command line, R, or python
- To launch via the command line:
 - Unpack the h2o-3.8.3.3.zip file in Windows/ or OSX/
 - In h2o-3.8.3.3 directory execute from a terminal or command prompt: `java -Xmx4g -jar h2o.jar`
 - Access flow at <http://localhost:54321>

H₂O Thanks for downloading! X H₂O Flow X

localhost:54321/flow/index.html

H₂O FLOW

Untitled Flow

Flow ▾ Cell ▾ Data ▾ Model ▾ Score ▾ Admin ▾ Help ▾

CS assist 214ms

assist

?

Assistance

Routine	Description
importFiles	Import file(s) into H ₂ O
getFrames	Get a list of frames in H ₂ O
splitFrame	Split a frame into two or more frames
getModels	Get a list of models in H ₂ O
getGrids	Get a list of grid search results in H ₂ O
getPredictions	Get a list of predictions in H ₂ O
getJobs	Get a list of jobs running in H ₂ O
buildModel	Build a model
importModel	Import a saved model
predict	Make a prediction

OUTLINE FLOWS CLIPS HELP

?

Help

Using Flow for the first time?

Quickstart Videos

Or, view example Flows to explore and learn H₂O.

STAR H₂O ON GITHUB!

Star 1,188

GENERAL

- Flow Web UI ...

Ready Connections: 0 H₂O

R Users

- Prerequisite: R installed (for Windows Users please installed in a folder with no spaces e.g. Program Files)
- Add R to your PATH
- To quickly Install Dependent Packages:
 - R CMD INSTALL Windows/RPackages/*
- Install H2O package
 - R CMD INSTALL h2o-3.8.3.3/R/*

If you're having difficulties installing from the terminal or command prompt, you can just as easily install from a R console or R Studio windows:

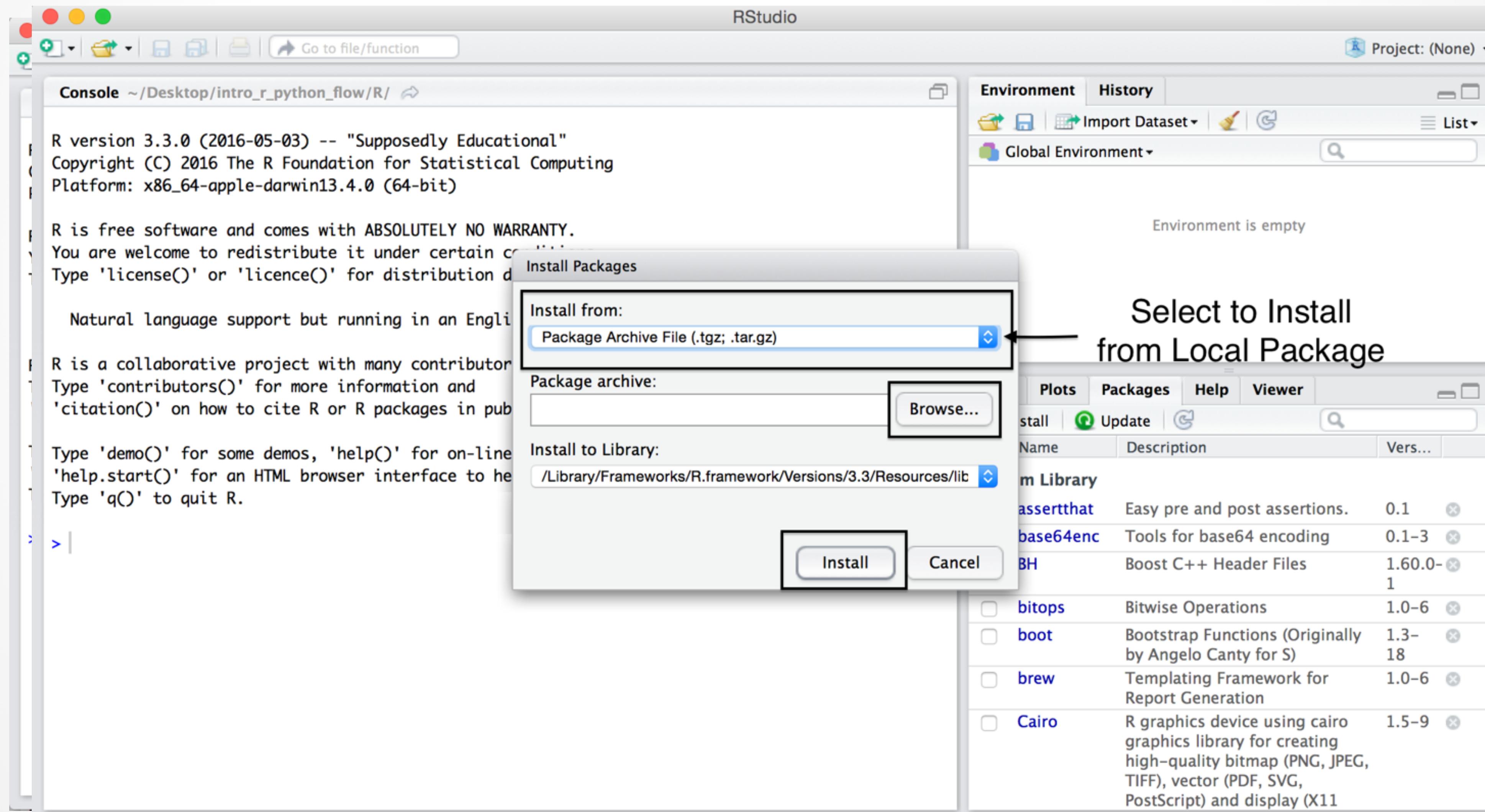
If you're having difficulties installing from the terminal or command prompt, you can just as easily install from a R console or R Studio windows:

The screenshot shows the RStudio interface. On the left is the R Console window displaying the standard R startup message. On the right are two panes: the Environment pane, which shows "Environment is empty", and the Packages pane. The Packages pane has tabs for Files, Plots, Packages, Help, and Viewer, with the Packages tab selected. A sub-menu is open under the Packages tab, showing options for "Install", "Update", and "Import Dataset". An arrow points from the text "To Install R packages" to the "Install" option in the sub-menu.

To Install R packages

Name	Description	Vers...
assertthat	Easy pre and post assertions.	0.1
base64enc	Tools for base64 encoding	0.1-3
BH	Boost C++ Header Files	1.60.0-1
bitops	Bitwise Operations	1.0-6
boot	Bootstrap Functions (Originally by Angelo Canty for S)	1.3-18
brew	Templating Framework for Report Generation	1.0-6
Cairo	R graphics device using cairo graphics library for creating high-quality bitmap (PNG, JPEG, TIFF), vector (PDF, SVG, PostScript) and display (X11)	1.5-9

If you're having difficulties installing from the terminal or command prompt, you can just as easily install from a R console or R Studio windows:



Python Users

- Prerequisite: Python 2.7 or 3.3 installed
- To Install Dependent modules:
 - pip install Windows/PythonModules/requests-2.8.1.tar.gz
 - pip install Windows/PythonModules/tabulate-0.7.5.tar.gz
- Install H2O wheel
 - pip install h2o-3.8.3.3-py2.py3-none-any.whl

Getting Started with H2O

Objective

- Learn how R, Flow, and Python sends commands to compute in H2O
- FAQ on writing R, Flow, and Python expressions
- Hands on introduction into data science
- Understanding model outputs
- Note the limitations of the basic workflow to improve upon later

Reading Data into H2O with R

STEP 1



```
h2o_df = h2o.importFile("../data/allyears2k.csv")
```

R User

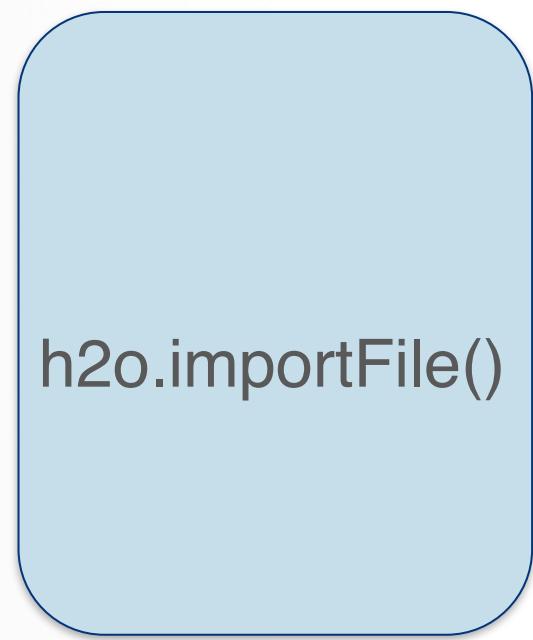
Reading Data into H2O with R

STEP 2

Reading Data into H2O with R

STEP 2

R

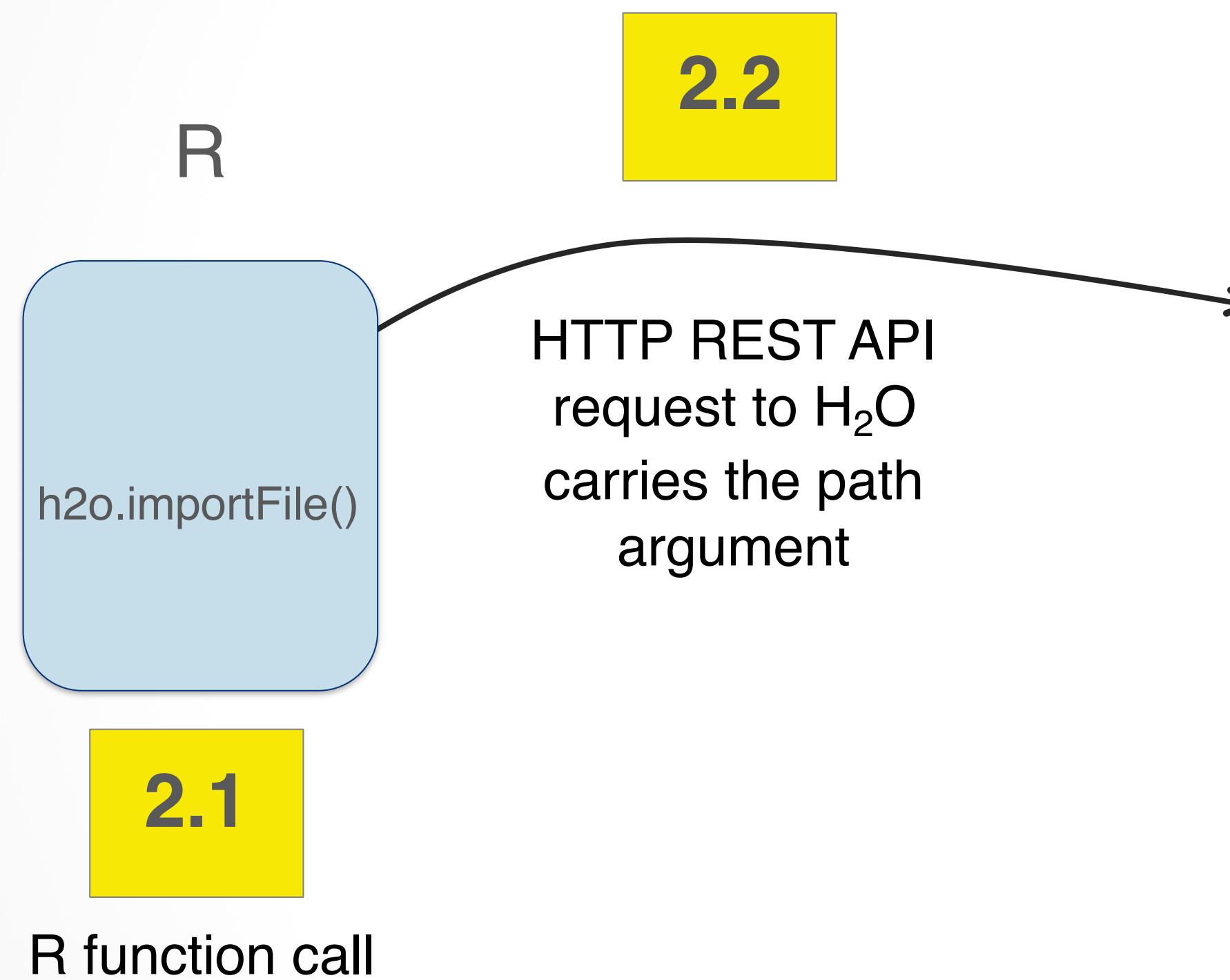


2.1

R function call

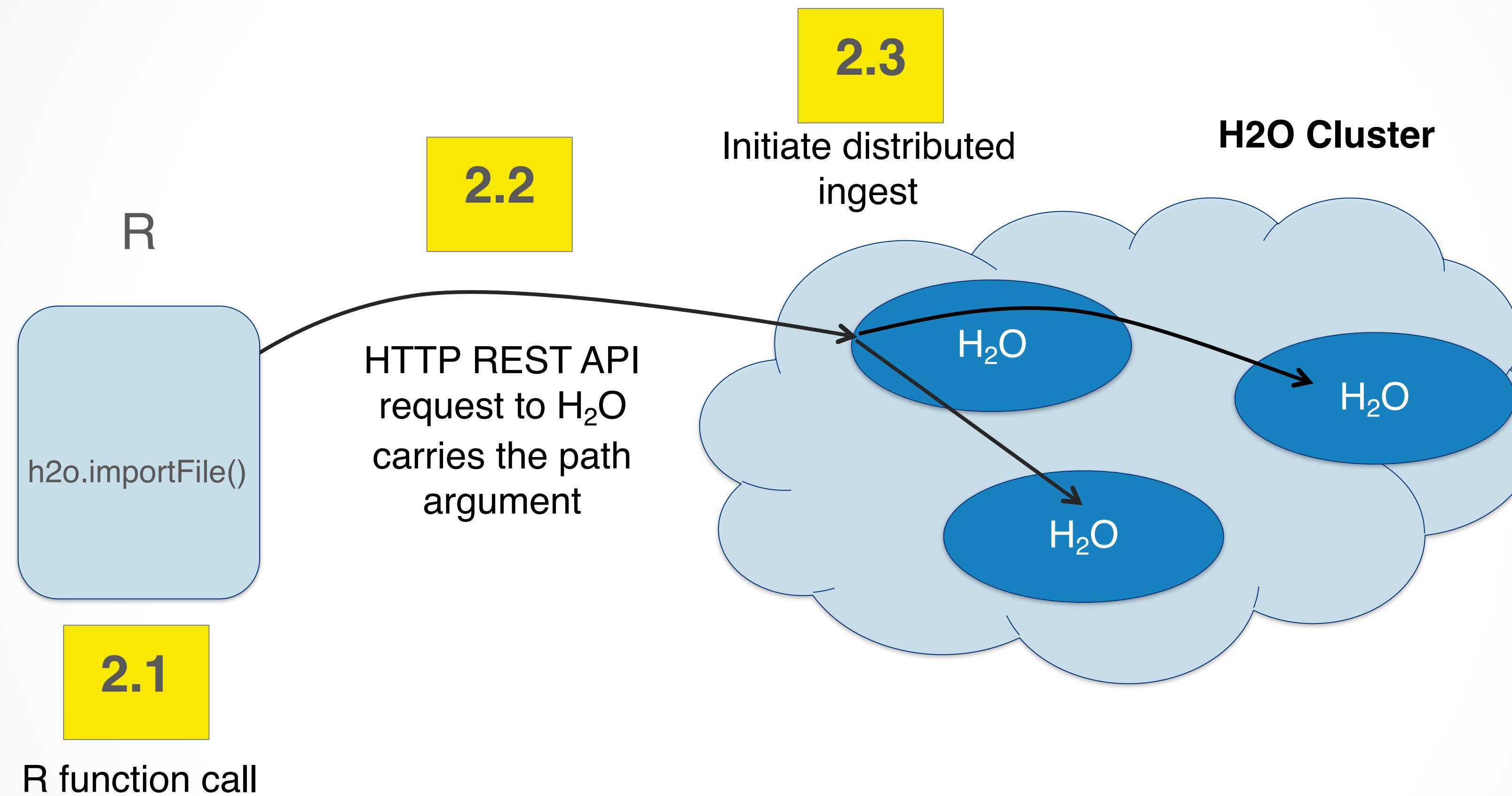
Reading Data into H₂O with R

STEP 2



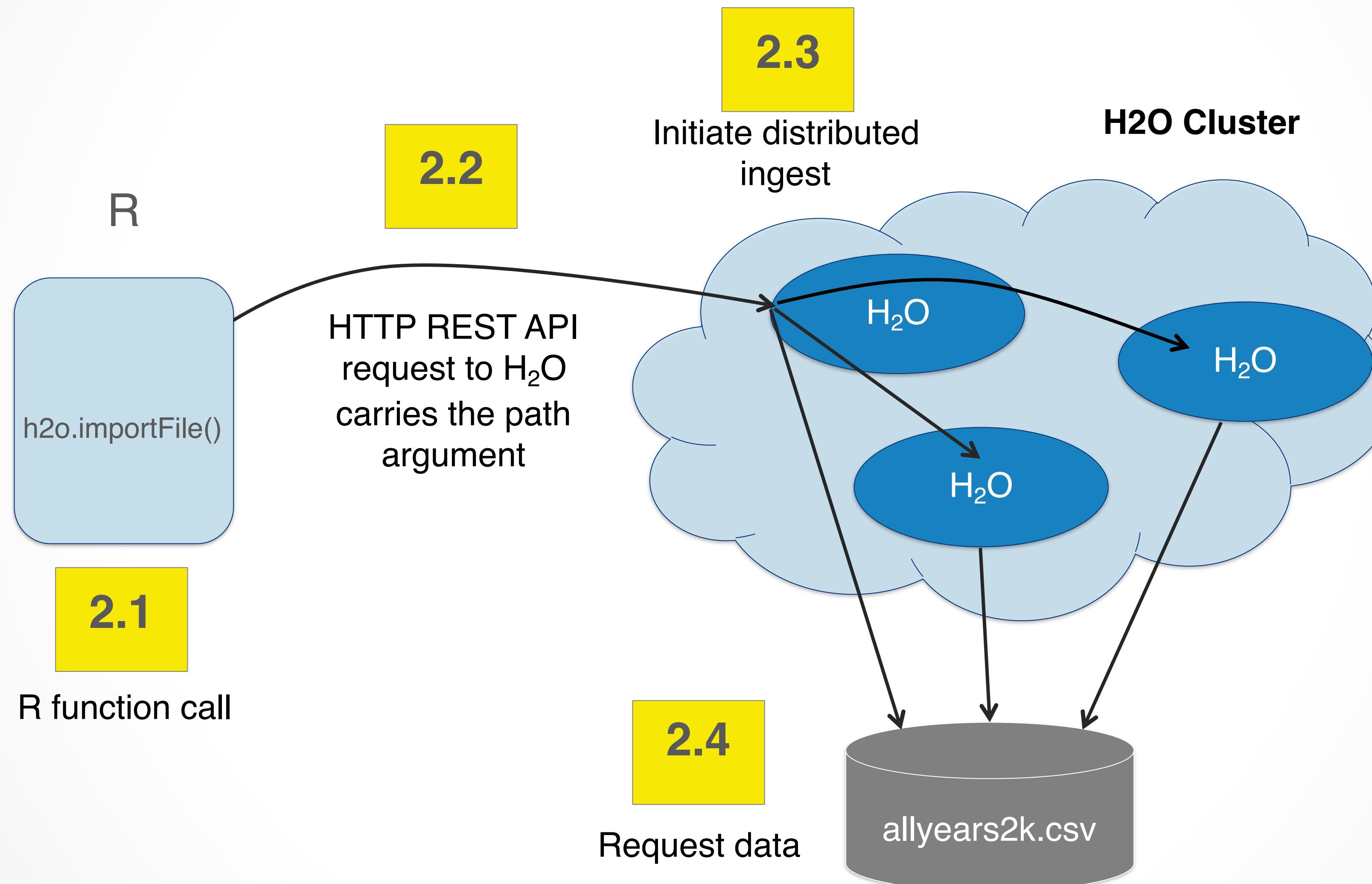
Reading Data into H₂O with R

STEP 2



Reading Data into H₂O with R

STEP 2



Reading Data into H2O with R

STEP 3

Reading Data into H2O with R

STEP 3

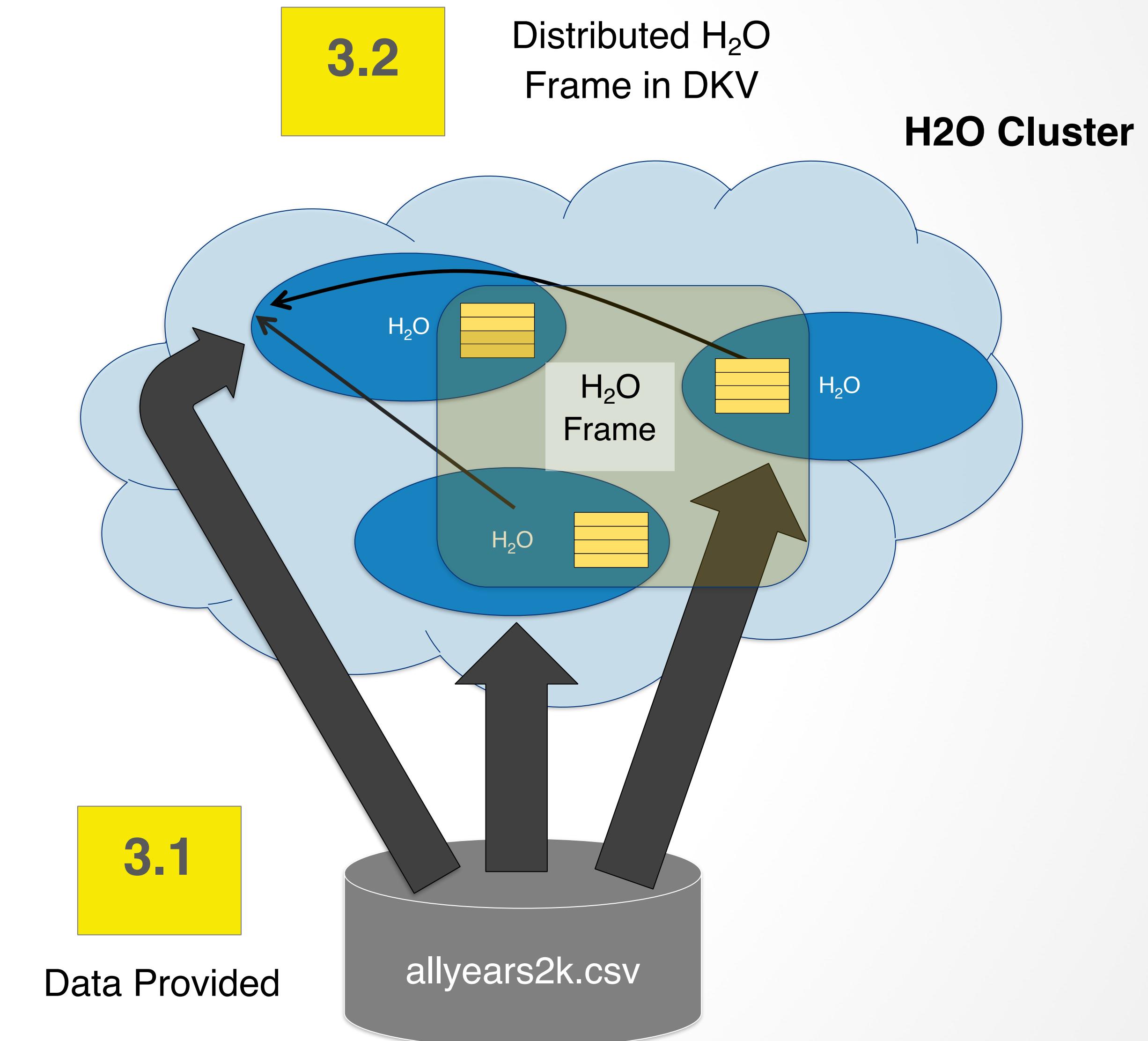
3.1

Data Provided



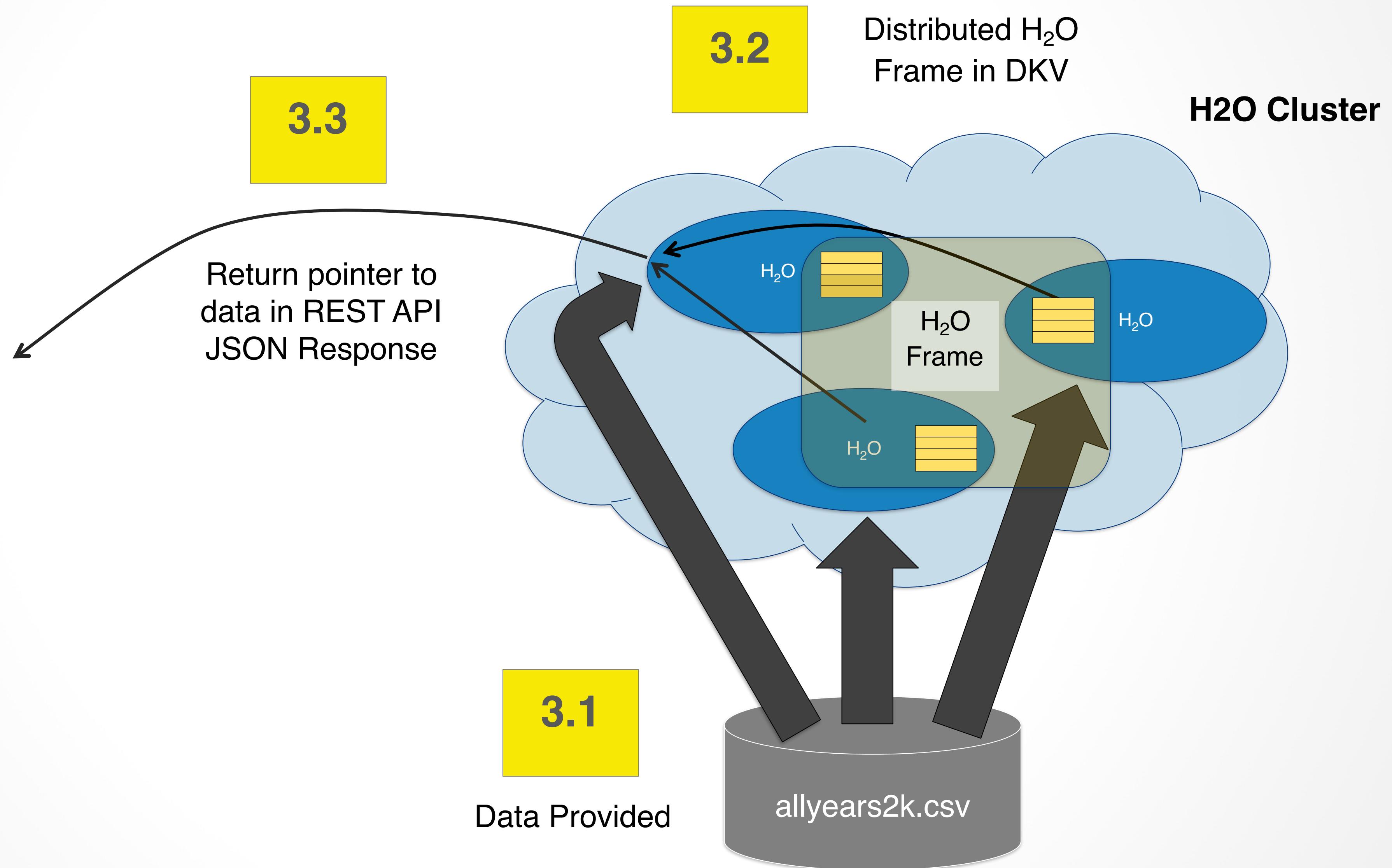
Reading Data into H2O with R

STEP 3



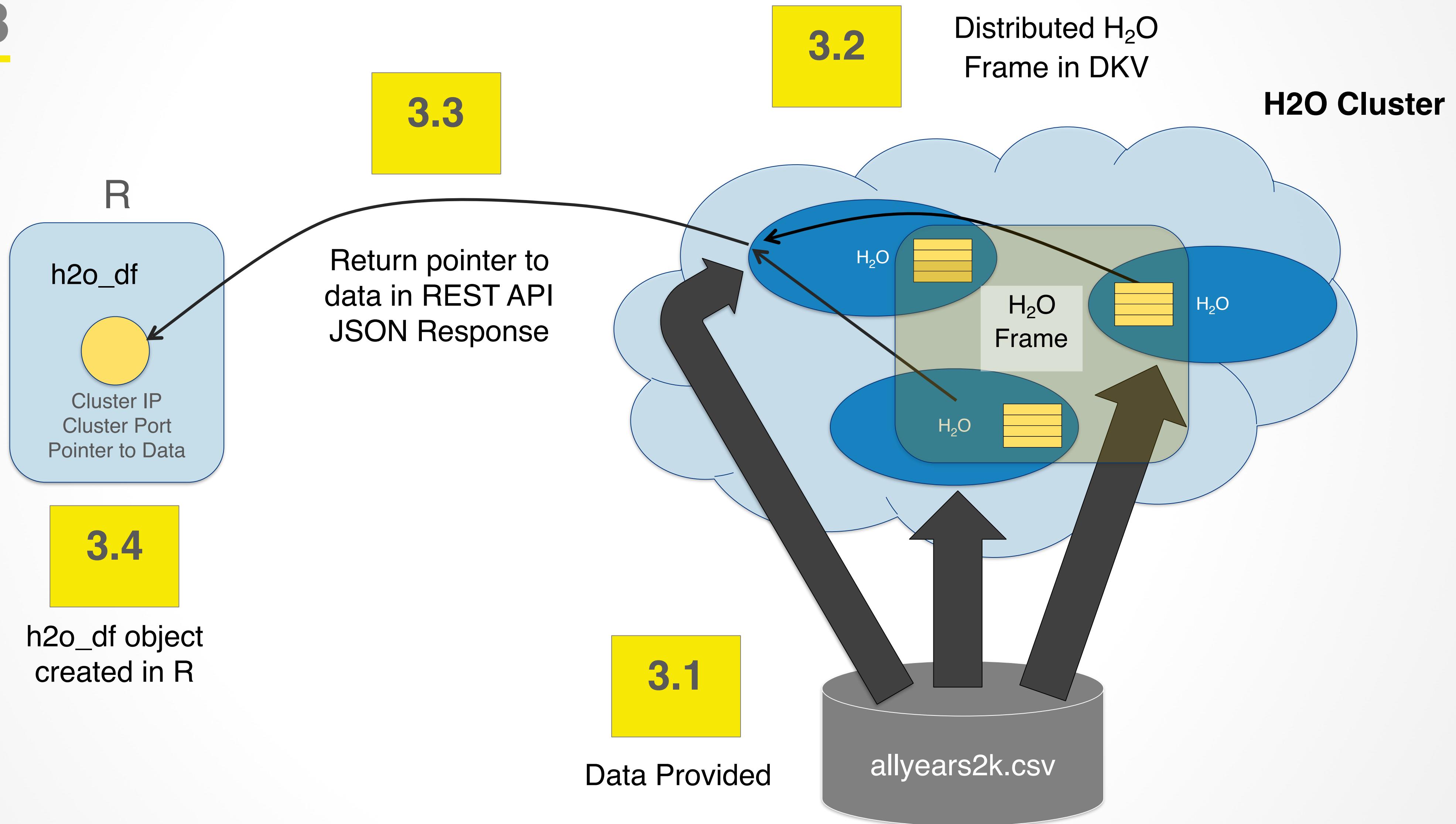
Reading Data into H2O with R

STEP 3



Reading Data into H2O with R

STEP 3



R Parity

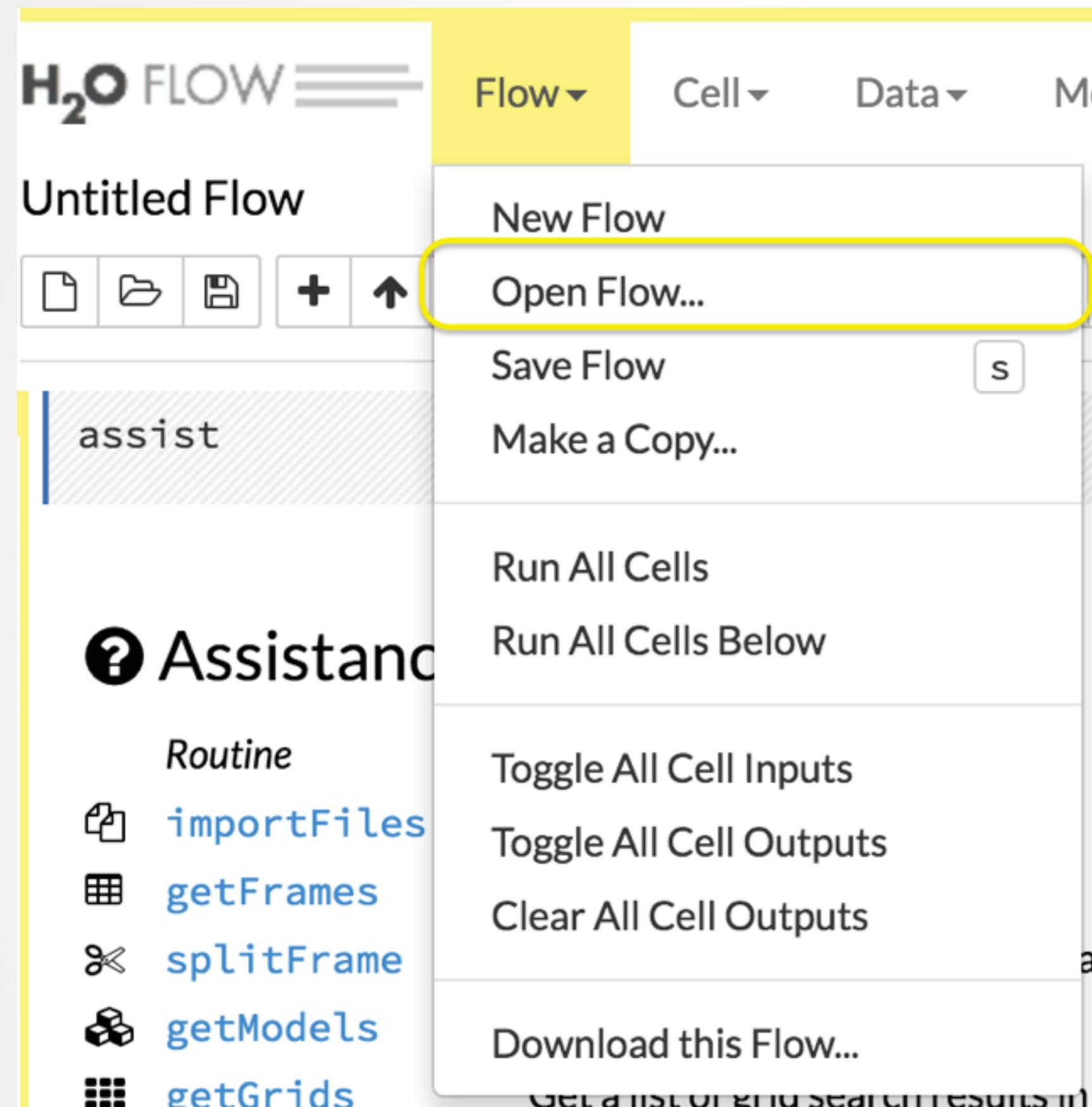
	Standard R	R on H2O
<i>Reading in data</i>	read.csv, read.table, etc	h2o.importFile
<i>Summarizing data</i>	summary	summary, h2o.summary
<i>Combining rows or columns</i>	cbind, rbind	h2o.cbind, h2o.rbind
<i>Unary or Binary Operations</i>	+, -, *, /, ^, %%, %/%, etc	+, -, *, /, ^, %%, %/%, etc
<i>Building GLM model</i>	glm, glmnet	h2o.glm
<i>Predict with Model</i>	predict	h2o.predict
<i>Obtaining Metrics</i>	auc, mse, logLoss, etc	h2o.auc, h2o.mse, h2o.logloss
<i>Unsupported Functions</i>	smooth	UNSUPPORTED

Hands on Introduction to Running H2O

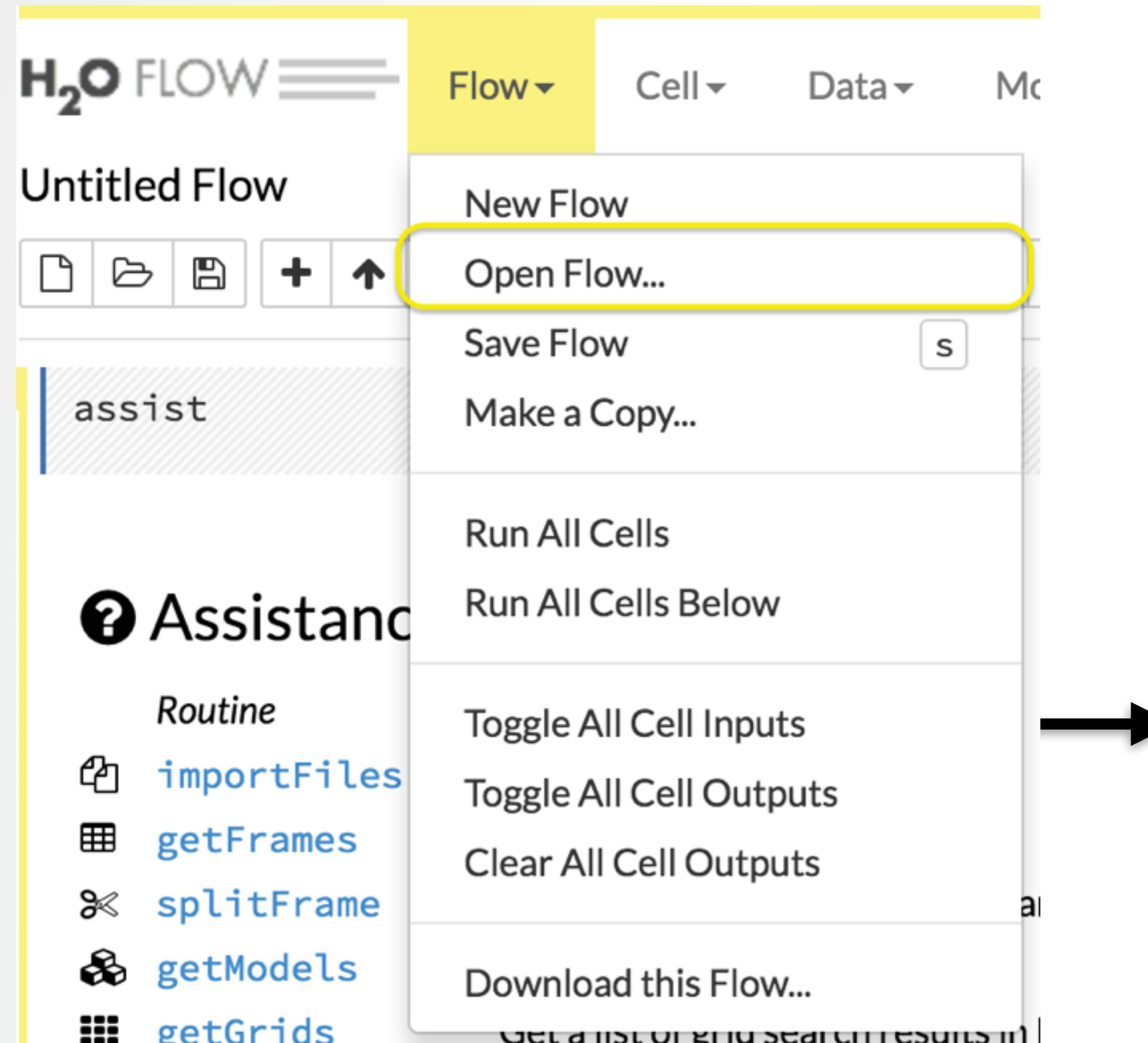
- Import & parse a small row airlines dataset
- Run a logistic regression
- Build a gradient boosting machine model
- Review the model outputs
- Join the flight data with weather data
- Rebuild logistic regression and GBM model
- Review new model outputs

Load up an example Flow from Disk

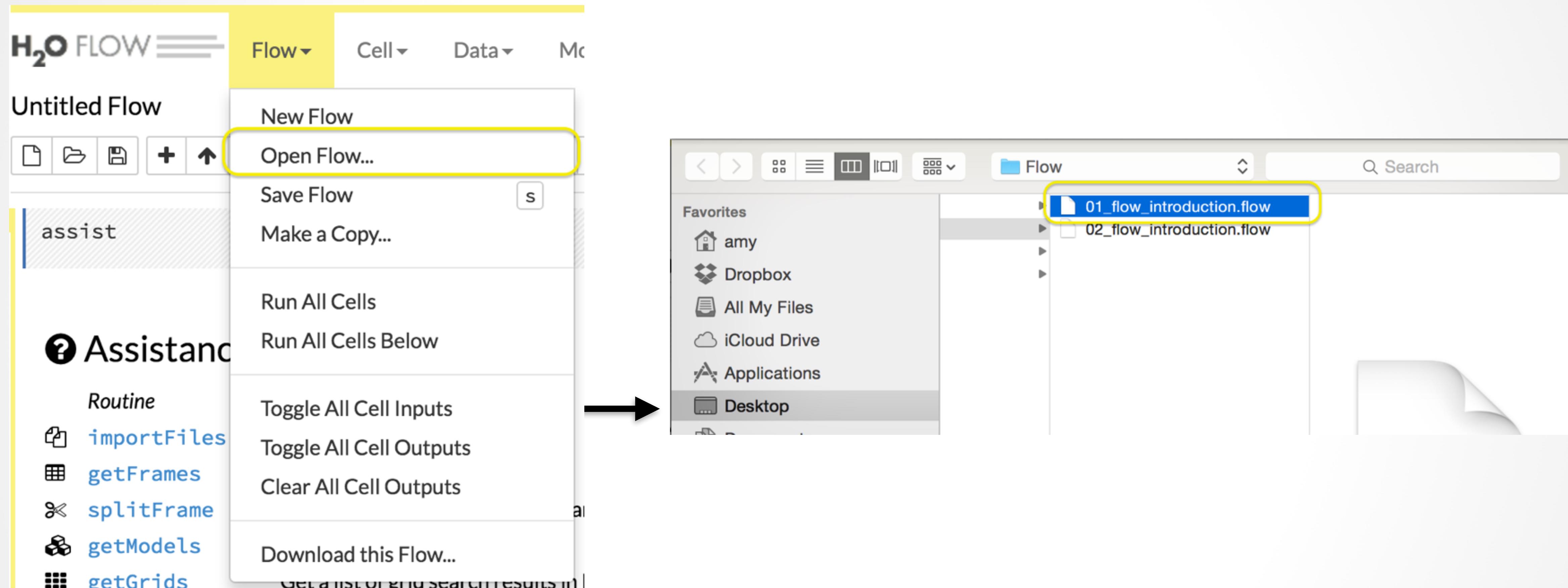
Load up an example Flow from Disk



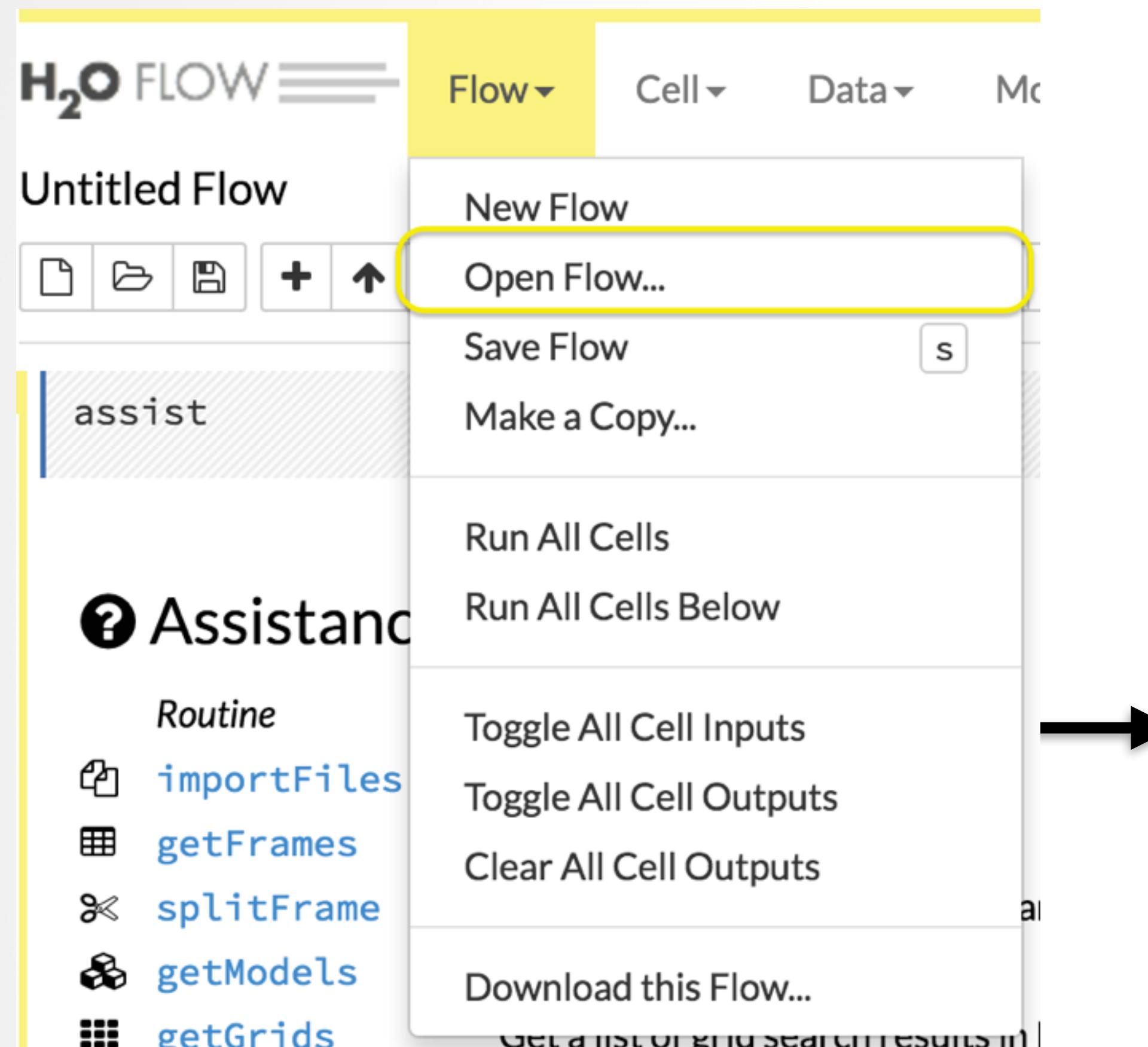
Load up an example Flow from Disk



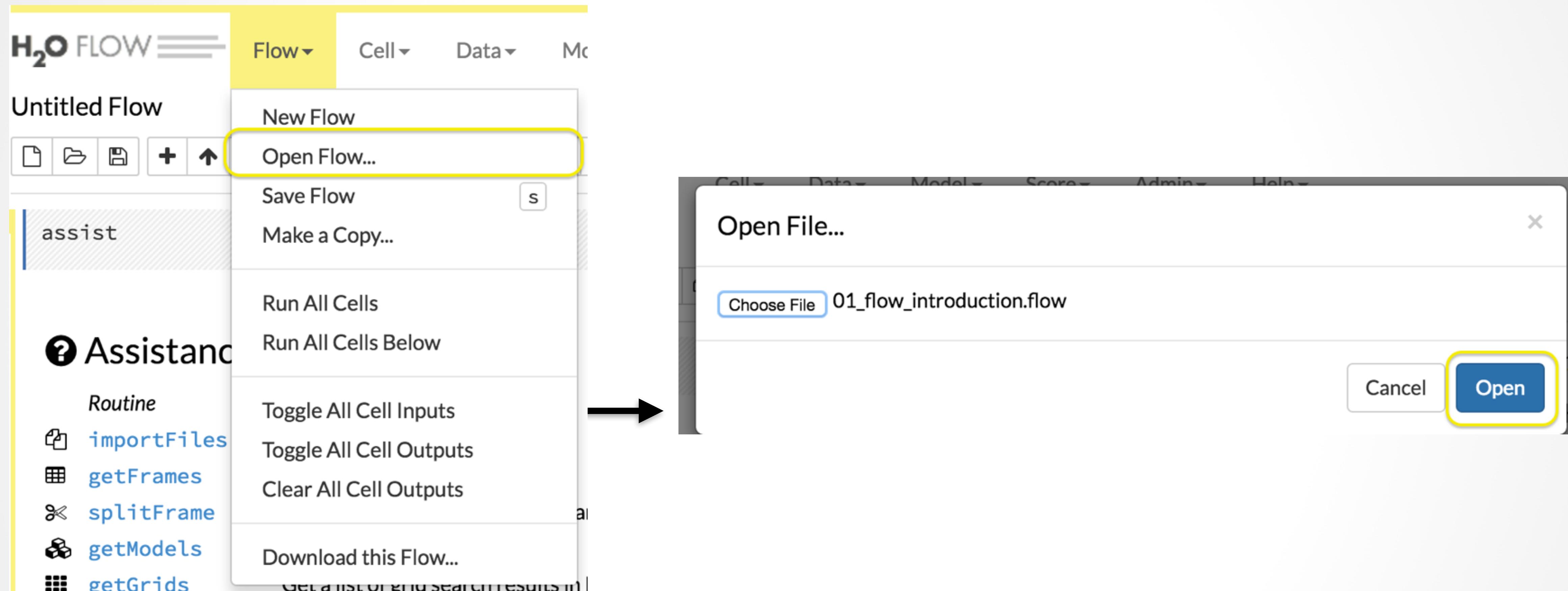
Load up an example Flow from Disk



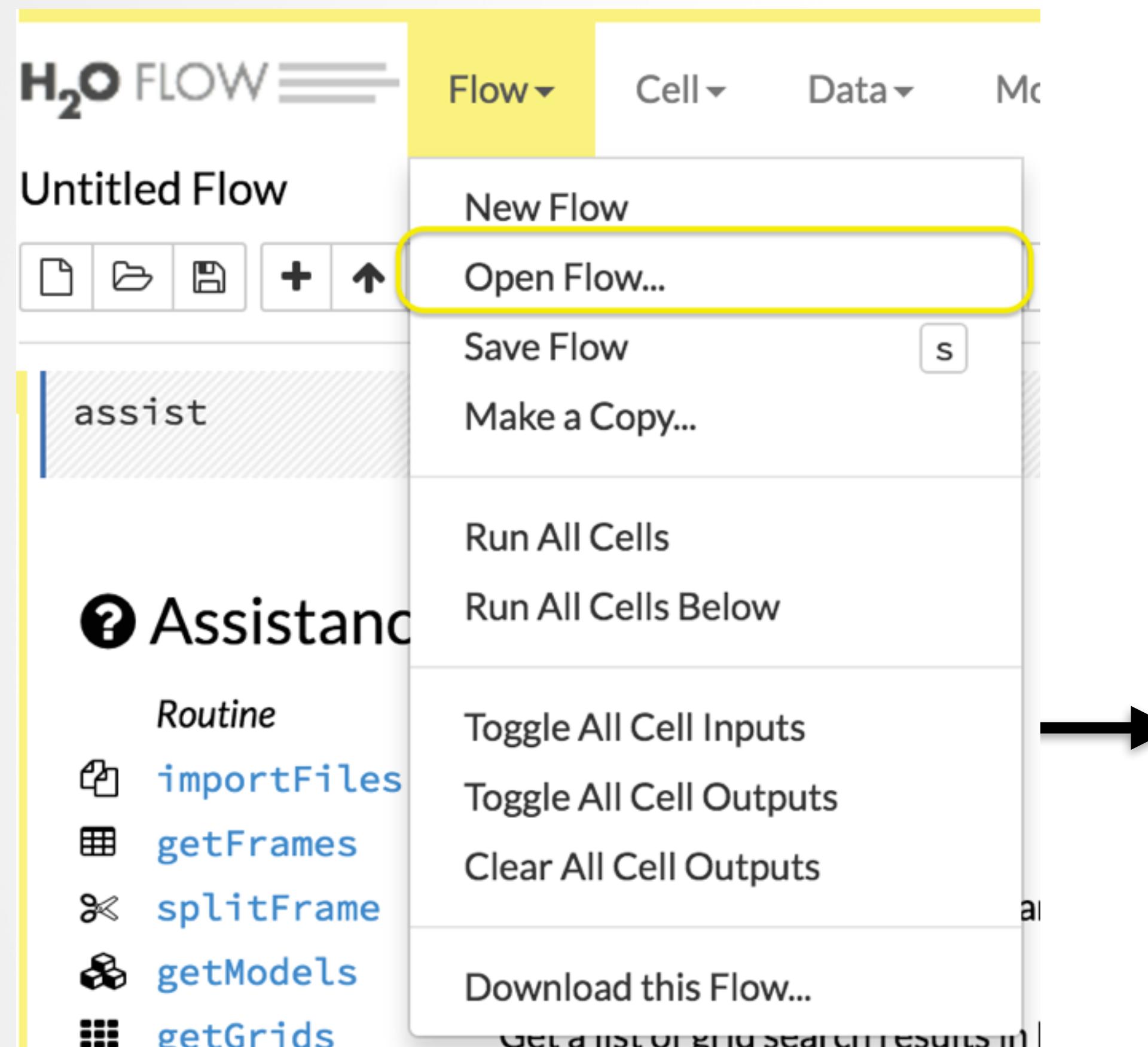
Load up an example Flow from Disk



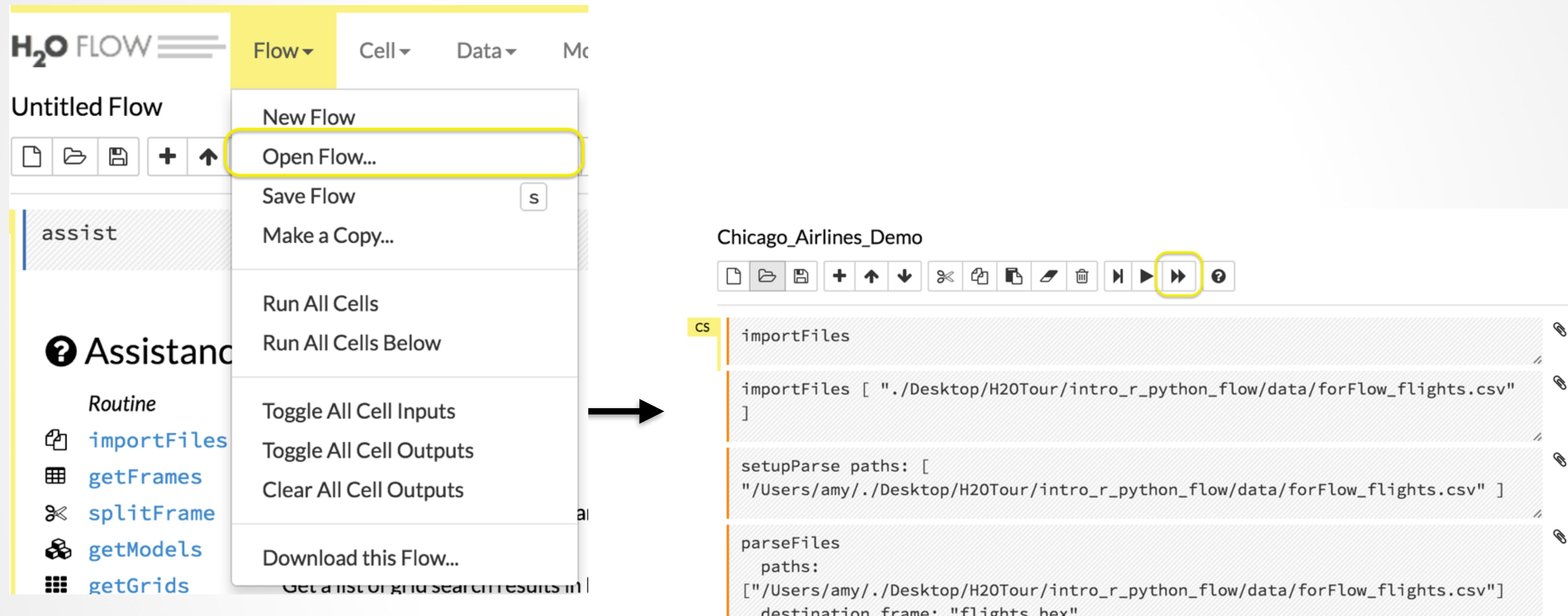
Load up an example Flow from Disk



Load up an example Flow from Disk



Load up an example Flow from Disk



Starting up H2O and Preloaded Workbook

Flow Users

From terminal, change the directory to where h2o.jar file is sitting and run:

```
> java -jar h2o.jar
```

Then access the Flow UI at:

<https://localhost:54321>

R Users

Open the intro-to-r.md.R file and run from R (Native R or Rstudio):

```
library(h2o)
```

```
h2o.init(nthreads = -1)
```

Python Users

Open either intro-to-python.ipynb or intro-to-python.py with python:

```
import h2o
```

```
import ...
```

```
h2o.init()
```

Import Airlines Data into H2O

Flow Users

```
importFiles [ "./Desktop/H2OTour/intro_r_python_flow/data/forFlow_flights.csv" ]  
  
setupParse paths: [ "./Desktop/H2OTour/intro_r_python_flow/data/  
forFlow_flights.csv" ]  
  
...
```

R Users

```
flights_hex = h2o.importFile(path = flights_path, destination_frame =  
"flights_hex")  
  
weather_hex = h2o.importFile(path = weather_path, destination_frame  
= "weather_hex")
```

Python Users

```
flights_hex = h2o.import_file(path = flights_path, destination_frame =  
"flights_hex")  
  
weather_hex = h2o.import_file(path = weather_path,  
destination_frame = "weather_hex")
```

The Airlines Data

- Goal: To predict departure delays using historical airlines data with Destination to Chicago O'Hare Airport or ORD
- Enumerator columns: Origin, and Unique Carrier have a cardinality of 132 and 10 respectively
- Numeric Columns: DayOfMonth, Year, DayOfWeek, Month, and Distance
- Binary Response Column: IsArrDelayed

Build a Logistic Regression Model

Flow Users

```
buildModel 'glm', {"model_id":"glm_model","training_frame":"allyears2k.hex","ignored_columns":  
["DayofMonth","DepTime","CRSDepTime","ArrTime","CRSArrTime","TailNum","ActualElapsedTime","CRSElapsedTime"  
,"AirTime","ArrDelay","DepDelay","TaxiIn","TaxiOut","Cancelled","CancellationCode","Diverted","CarrierDela  
y","WeatherDelay","NASDelay","SecurityDelay","LateAircraftDelay","IsArrDelayed"],"ignore_const_cols":true,  
"response_column":"IsDepDelayed","family":"binomial","solver": "IRLSM","alpha": [0.5],"lambda":  
[0.00001],"lambda_search":false,"standardize":true,"non_negative":false,"score_each_iteration":false,"max_  
iterations": -1,"link": "family_default","intercept": true,"objective_epsilon": 0.00001,"beta_epsilon": 0.0001,  
"gradient_epsilon": 0.0001,"prior": -1,"max_active_predictors": -1}
```



4.1s

R Users

```
y <- "IsDepDelayed"  
x <- c("Dest", "Origin", "DayofMonth", "Year", "UniqueCarrier", "DayOfWeek", "Month",  
"Distance")  
arr_delay_glm = h2o.glm(x = myX,y = myY, training_frame = train, validation_frame = valid,  
family = "binomial", alpha = 0.5,lambda_search = T)
```

Python Users

```
myY = "IsDepDelayed"  
myX = ["Dest", "Origin", "DayofMonth", "Year", "UniqueCarrier", "DayOfWeek", "Month", "Distance"]  
glm_model = H2OGeneralizedLinearEstimator(family = "binomial", standardize = True, alpha = 0.5,  
lambda_search = True )  
glm_model.train( x = myX, y = myY, training_frame = train, validation_frame = valid)
```

GLM Model Output

Model

Model ID: glm-b59c111a-cf92-48d1-b920-79047c3d659f

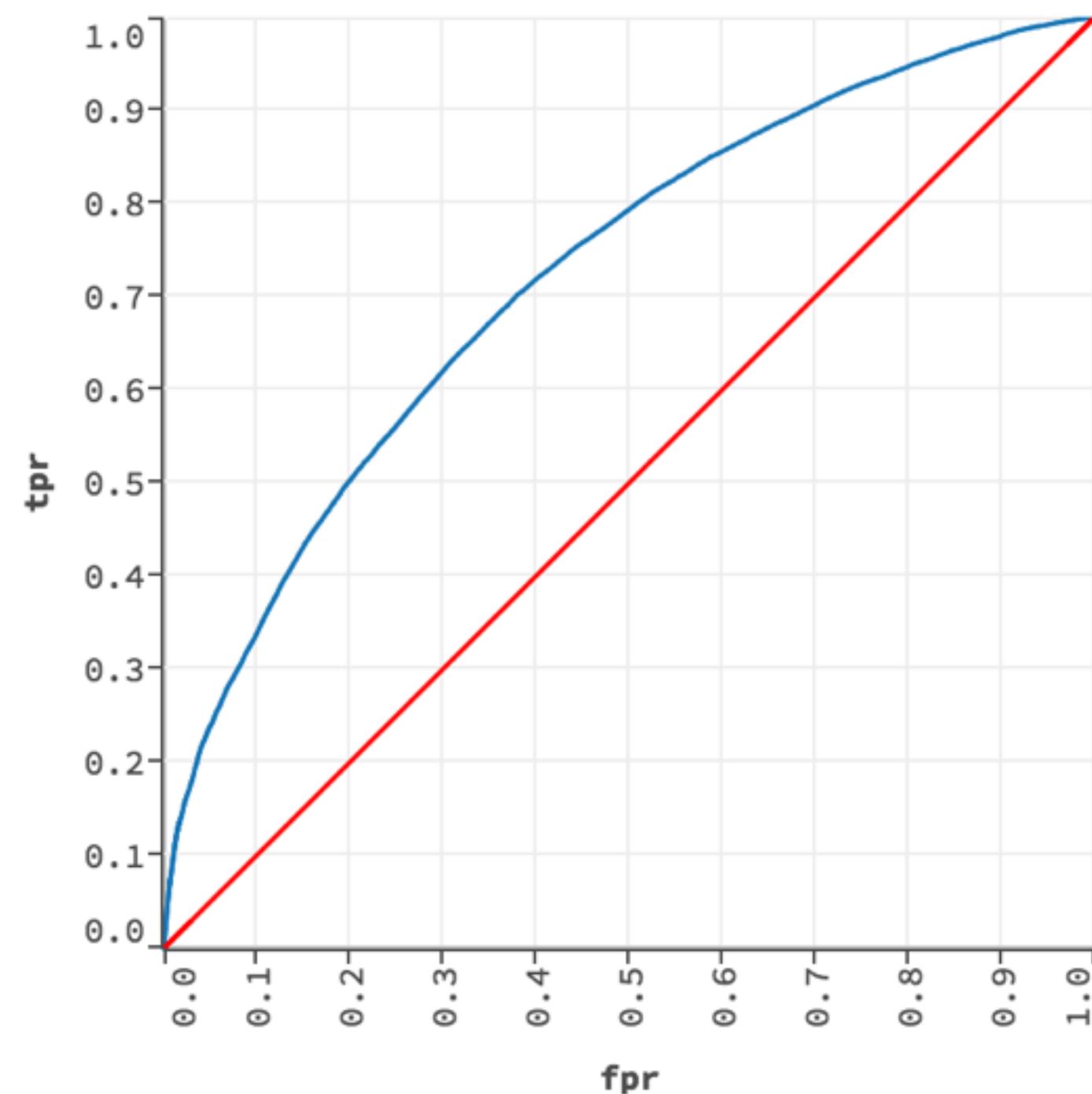
Algorithm: Generalized Linear Modeling

Actions: [Refresh](#) [Predict...](#) [Download POJO](#) [Export](#) [Inspect](#) [Delete](#)

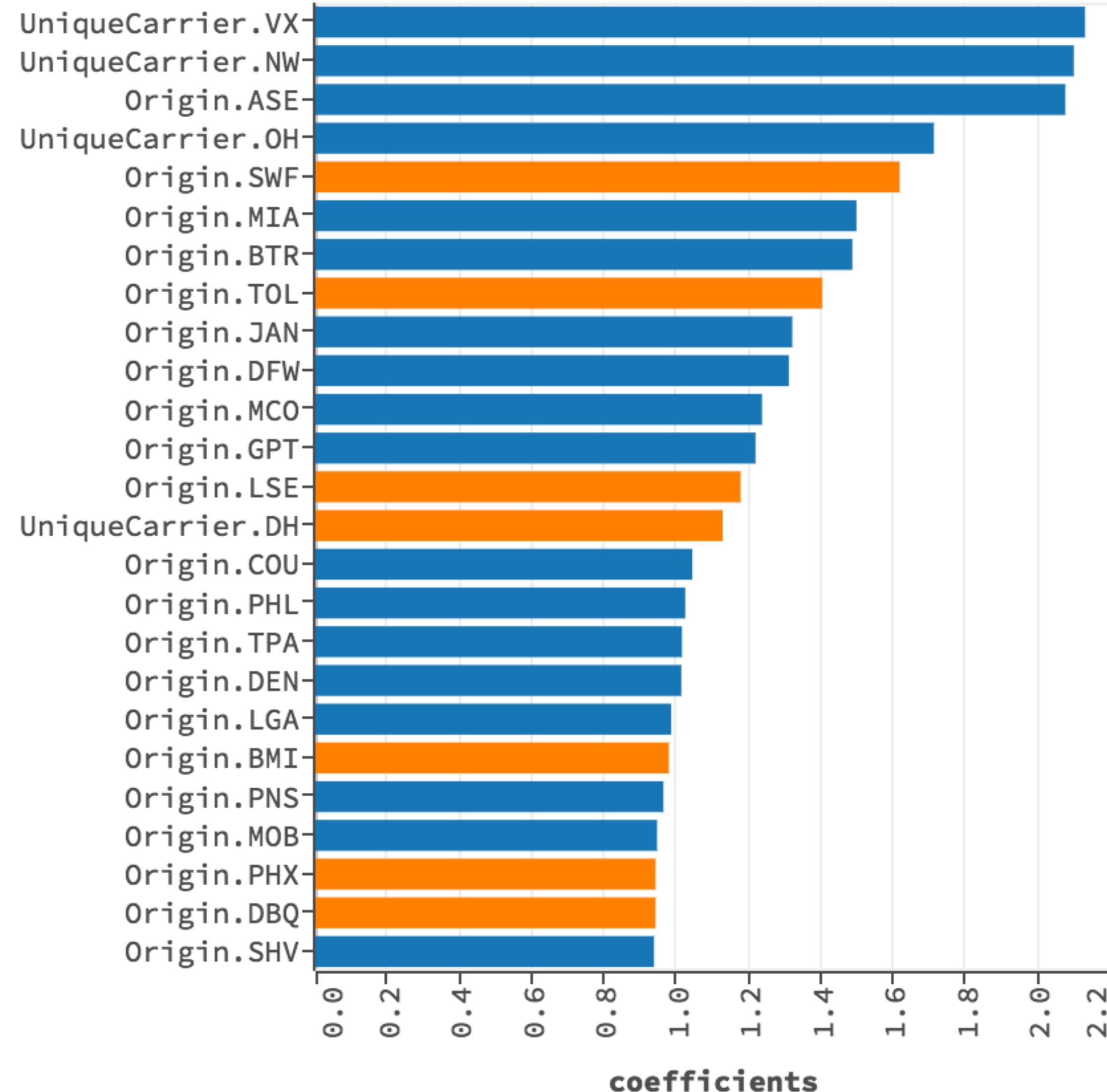
MODEL PARAMETERS

SCORING HISTORY

ROC CURVE - TRAINING METRICS , AUC = 0.722933



STANDARDIZED COEFFICIENT MAGNITUDES



Build a GBM Model

Flow Users

```
buildModel 'gbm', {"model_id":"gbm-ec9ed500-382b-4766-8af4-dc040ab0f6a5","training_frame":"train","validation_frame":"valid","nfolds":0,"response_column":"IsArrDelayed","ignored_columns":["DepTime","CRSDepTime","ArrTime","CRSArrTime","FlightNum","TailNum","ActualElapsedTime","CRSElapsedTime","AirTime","ArrDelay","DepDelay","Dest","TaxiIn","TaxiOut","Cancelled","CancellationCode","Diverted","CarrierDelay","WeatherDelay","NASDelay","SecurityDelay","LateAircraftDelay","IsDepDelayed", "IsWeatherDelayedNumeric","IsArrDelayedNumeric"],"ignore_const_cols":true,"ntrees":50,"max_depth":5,"min_rows":10,"nbins":20,"seed":-1,"learn_rate":0.1,"distribution":"AUTO","sample_rate":1,"col_sample_rate":1,"score_each_iteration":false,"score_tree_interval":0,"balance_classes":false,"nbins_top_level":1024,"nbins_cats":1024,"r2_stopping":0.999999,"stopping_rounds":0,"stopping_metric":"AUTO","stopping_tolerance":0.001,"max_runtime_secs":0,"learn_rate_annealing":1,"checkpoint":"","col_sample_rate_per_tree":1,"min_split_improvement":0.00001,"histogram_type":"AUTO","build_tree_one_node":false,"sample_rate_per_class":[],"col_sample_rate_change_per_level":1,"max_abs_leafnode_pred":1.7976931348623157e+308}
```

R Users

```
arr_delay_gbm = h2o.gbm(x = myX,  
y = myY,  
training_frame = train,  
validation_frame = valid,  
distribution = "bernoulli",  
ntrees = 50)
```

Python Users

```
arr_delay_gbm = H2OGradientBoostingEstimator(distribution = "bernoulli", ntrees =50)  
arr_delay_gbm.train(x=myX,y=myY,training_frame =train,validation_frame=valid)
```

GBM Model Output

Model

Model ID: gbm-ec9ed500-382b-4766-8af4-dc040ab0f6a5
Algorithm: Gradient Boosting Method

Actions: [Refresh](#) [Predict...](#) [Download POJO](#) [Export](#) [Inspect](#) [Delete](#)

► MODEL PARAMETERS

► SCORING HISTORY - LOGLOSS

A line graph showing the relationship between the number of trees and the logloss. The x-axis is labeled "number_of_trees" and ranges from 0 to 50. The y-axis is labeled "training_logloss, validation_logloss" and ranges from 0.088 to 0.106. Two lines are plotted: a blue line for training logloss and an orange line for validation logloss. Both lines show a downward trend as the number of trees increases. The training logloss starts at approximately 0.105 and ends at 0.089. The validation logloss starts at approximately 0.104 and ends at 0.089.

number_of_trees	training_logloss	validation_logloss
0	0.105	0.104
5	0.101	0.099
10	0.098	0.096
15	0.096	0.094
20	0.094	0.092
25	0.092	0.090
30	0.090	0.089
40	0.089	0.088
50	0.089	0.088

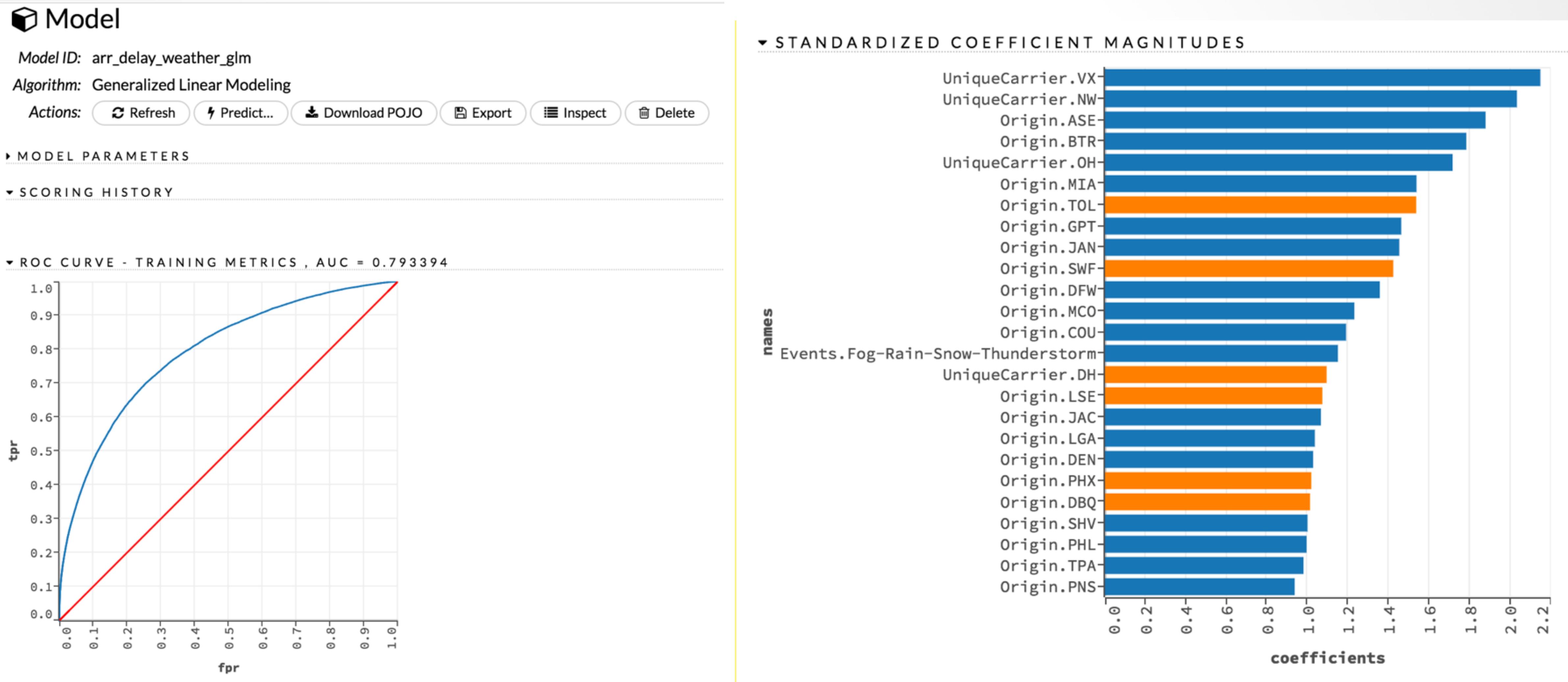
► ROC CURVE - TRAINING METRICS , AUC = 0.804336

An ROC curve plot showing the relationship between the True Positive Rate (tpr) and the False Positive Rate (fpr). The x-axis is labeled "fpr" and ranges from 0.0 to 1.0. The y-axis is labeled "tpr" and ranges from 0.0 to 1.0. A blue curve represents the model's performance, starting at (0,0) and ending at (1,1). A red diagonal line represents a random classifier. The area under the curve (AUC) is 0.804336.

Join with Weather Data

- Join Chicago weather data with flights data
- Many numeric columns featuring weather information including min, max, mean for temperature, humidity, visibility, windspeed, and precipitation
- Categorical column ‘Event’ with 18 levels indicating presence of fog, rain, snow, or thunderstorm

GLM Model With Weather Output



GBM Model With Weather Output

Model

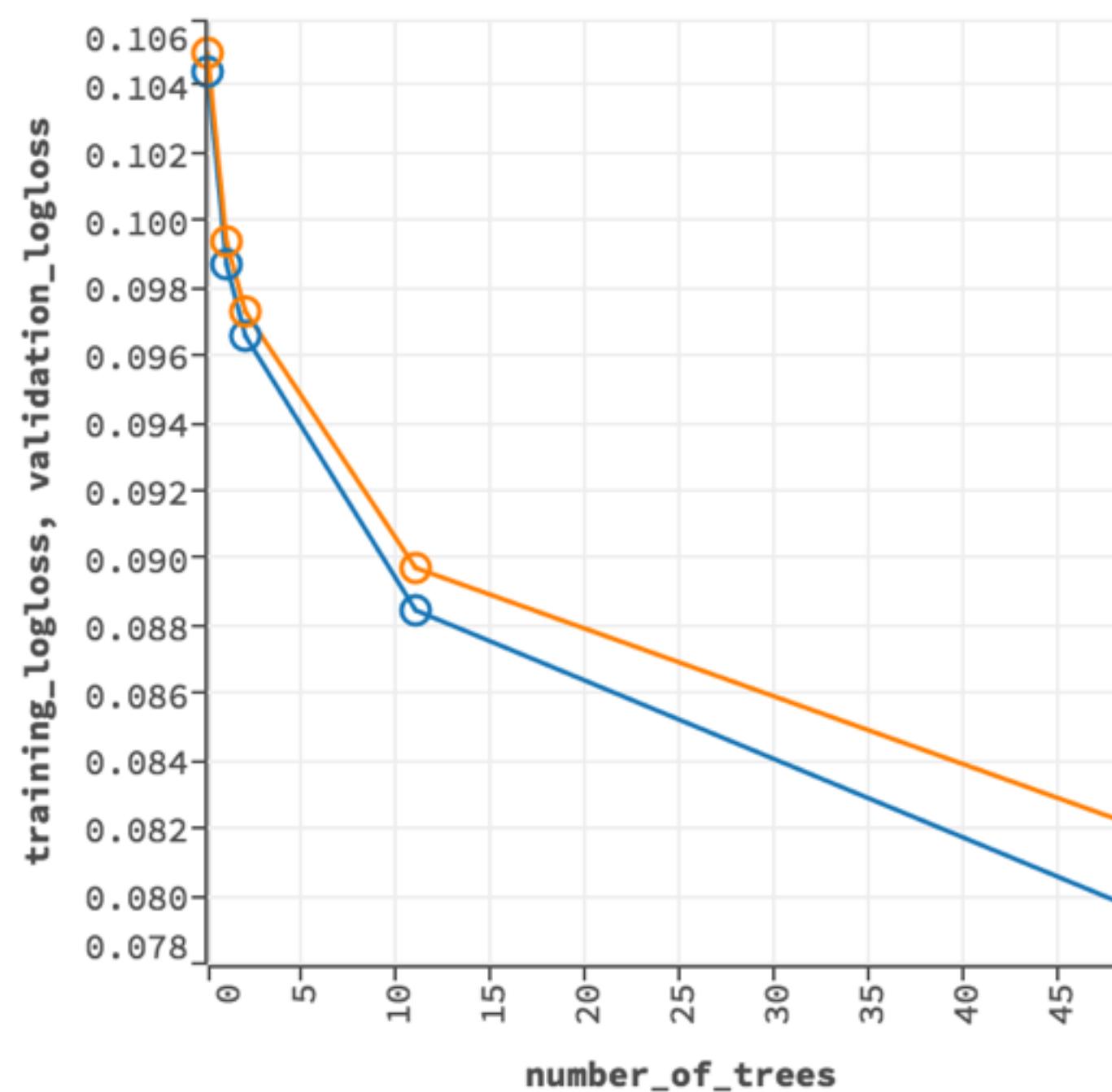
Model ID: arr_delay_weather_gbm

Algorithm: Gradient Boosting Method

Actions: [Refresh](#) [Predict...](#) [Download POJO](#) [Export](#) [Inspect](#) [Delete](#)

MODEL PARAMETERS

SCORING HISTORY - LOGLOSS



Model

Model ID: arr_delay_weather_gbm

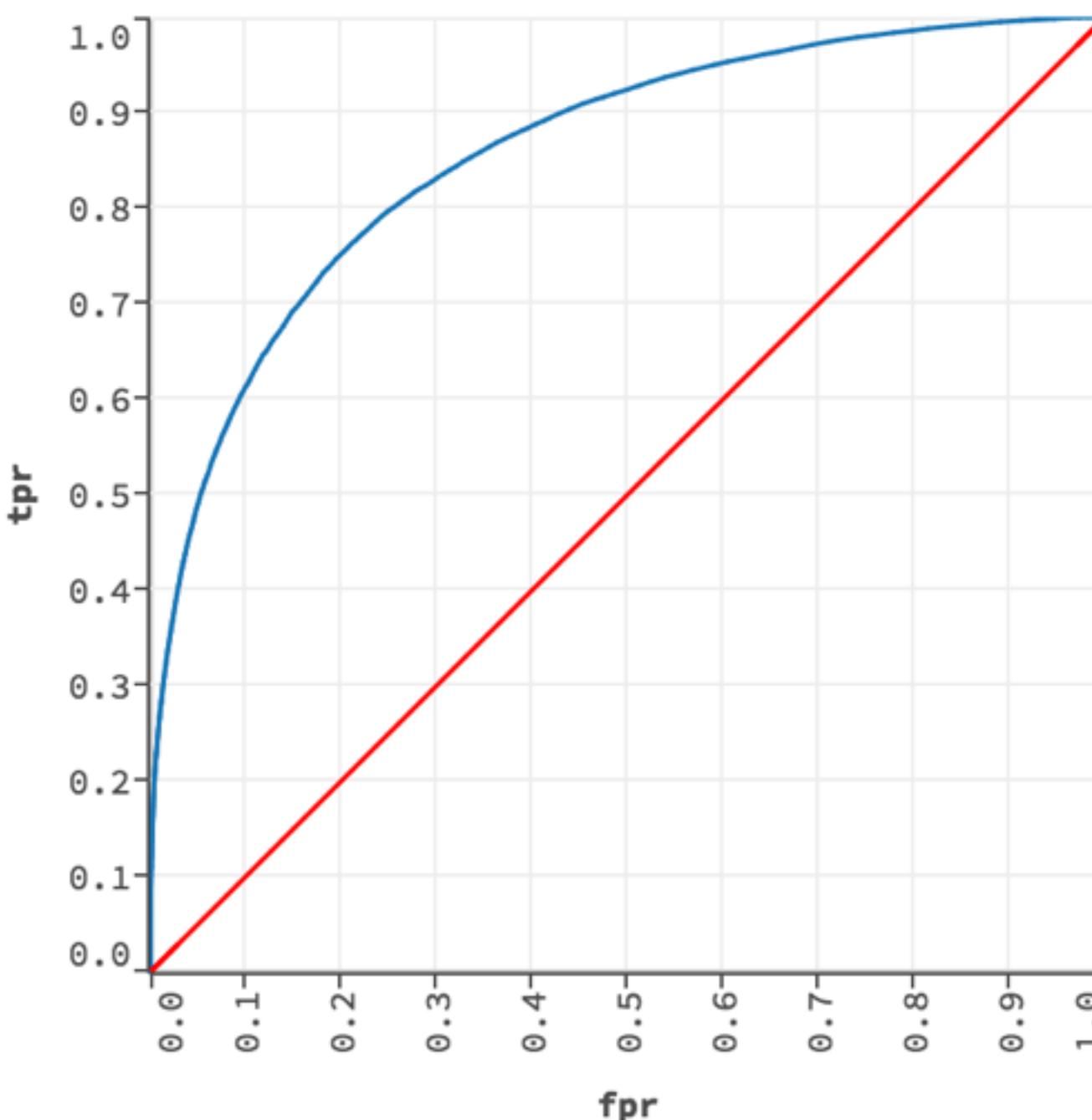
Algorithm: Gradient Boosting Method

Actions: [Refresh](#) [Predict...](#) [Download POJO](#) [Export](#) [Inspect](#) [Delete](#)

MODEL PARAMETERS

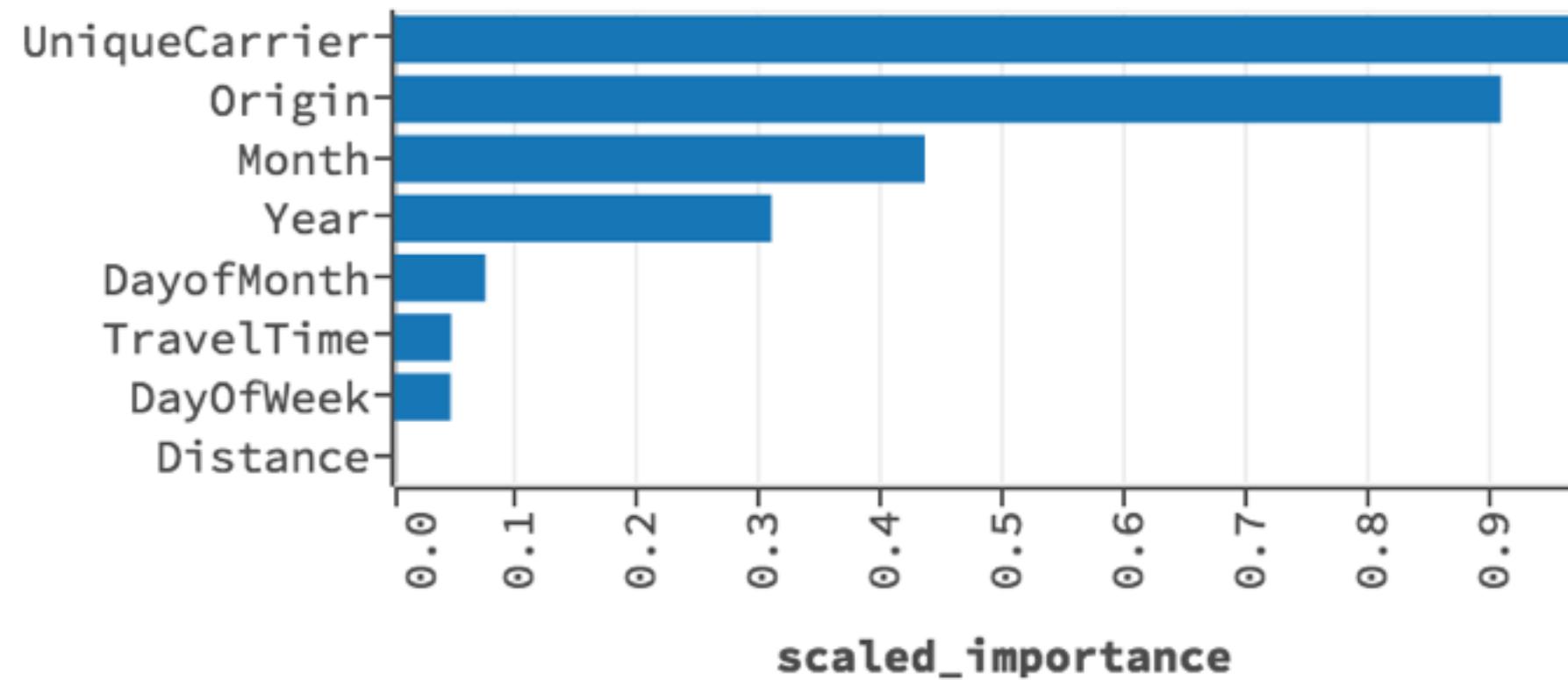
SCORING HISTORY - LOGLOSS

ROC CURVE - TRAINING METRICS , AUC = 0.857926

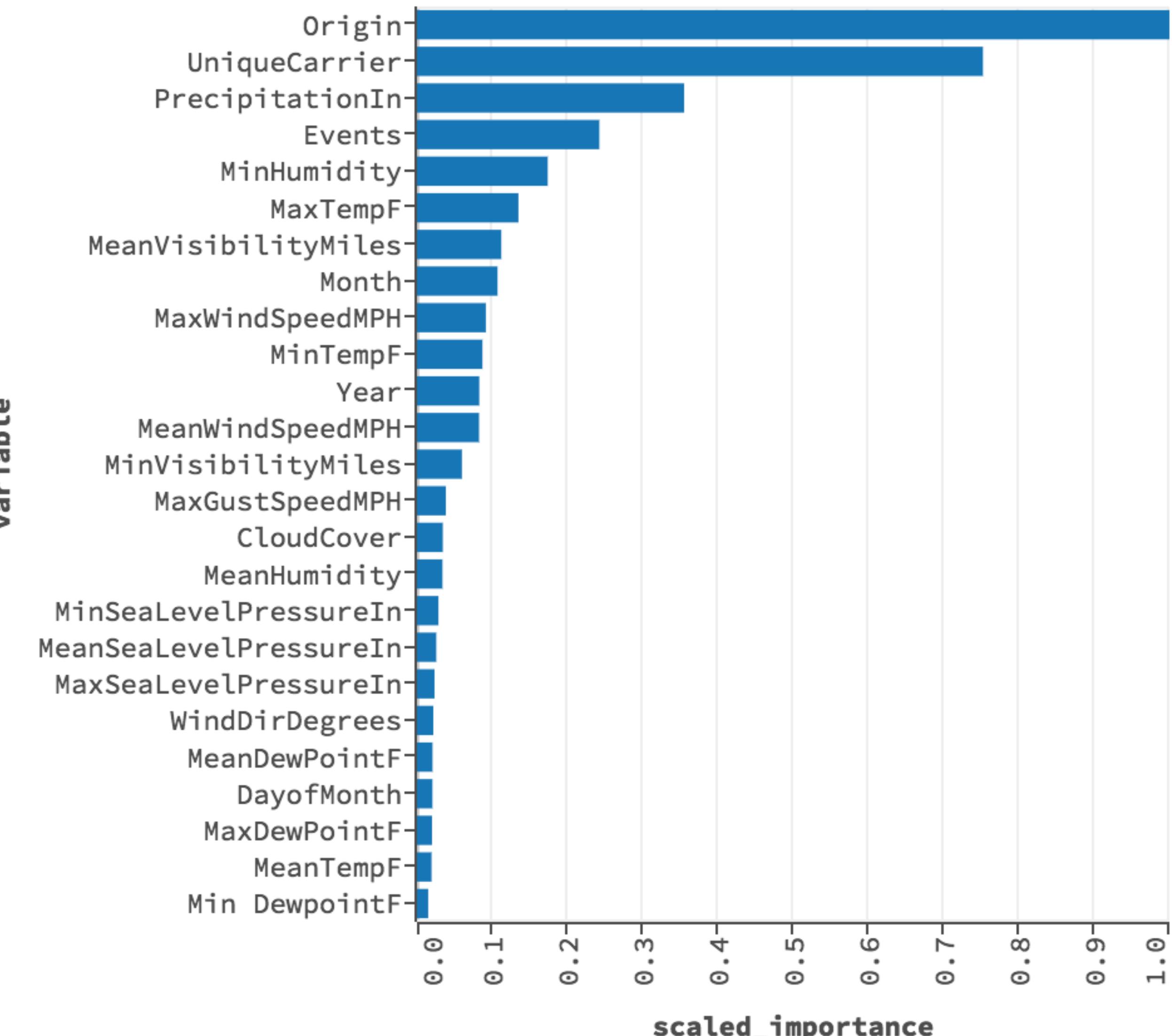


GBM Model Output - Variable Importance

▼ VARIABLE IMPORTANCES



▼ VARIABLE IMPORTANCES



Overview

- GBM models outperforms GLM with the same feature sets
- GLM models are more interpretable with beta values that relate each feature including expanded categorical levels to the response variable
- However you can combine what you know from a GBM model into a logistic regression model but creating interaction features using leaves from the GBM model
- There was a improvement when adding weather data for Chicago
- However the models can be further improved by grabbing weather data for all airports and doing a join with Origin in addition to weather of the destination

Overview

- Write R and Python expression to clean and munge the data in a parallelized and distributed fashion in H2O.
- Automate model builds by writing R and Python code and because all frames and models are generate in H2O, it is also accessible from the Flow UI.
- From the Web UI, the user can readily access the POJO which is the H2O independent Java representation of the models.