



Introduction to H2O
And
Sparkling Water

SETTING UP QWIKLABS

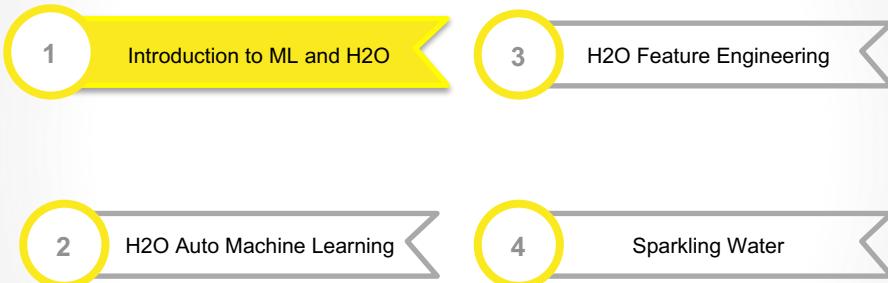
Qwiklabs Instructions

1. Go to: <https://h2oai.qwiklab.com/>
2. Click "Join" in upper right hand corner to create an account
 - o If you have already done a Qwiklabs you can click "Sign In"
3. Go to the "Catalog" view
4. Click on the lab: "H2O and Sparkling Water Workshop"
5. Click Start Lab
6. Enter License

"Confidential and property of H2O.ai. All rights reserved"



H2O Training



Introduction:

INTRODUCTION TO MACHINE LEARNING

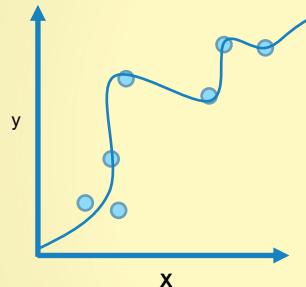
What is Machine Learning?

- Set of statistical and optimization tools to model data
 - Supervised Learning
 - Unsupervised Learning
- Focus on “production analytics”
 - Reducing need for
 - Sampling
 - Periodic revision of models
 - Subjective priors
 - Hand coding models in a production language

Supervised Learning

Regression:

How much will a customer spend?

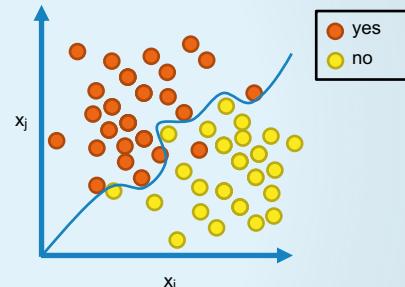


H2O algos:

- Penalized Linear Models
- Random Forest
- Gradient Boosting
- Neural Networks
- Stacked Ensembles

Classification:

Will a customer make a purchase? Yes or No



H2O algos:

- Penalized Linear Models
- Naïve Bayes
- Random Forest
- Gradient Boosting
- Neural Networks
- Stacked Ensembles

H₂O.ai

Stacked Ensembles

$$\begin{bmatrix} y \end{bmatrix} \approx \begin{bmatrix} \hat{y}_l \end{bmatrix} = f_l \left(\begin{bmatrix} x_1 \dots x_p \end{bmatrix} \right) \quad \begin{bmatrix} y \end{bmatrix} \approx g \left(\begin{bmatrix} \hat{y}_1 \dots \hat{y}_L \end{bmatrix} \right)$$

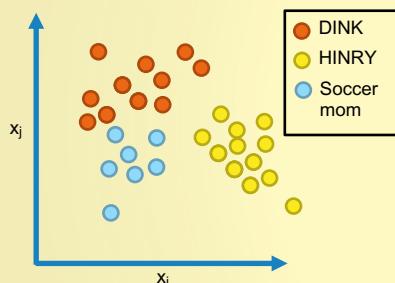
- Engineer original feature set
- Train L models using strong learners (GLM, RF, GBM, ...)
- Control overfitting using k-fold CV
- Stack these predicted values to form new feature set
- Train metalearner on stacked predictions

H₂O.ai

Unsupervised Learning

Clustering:

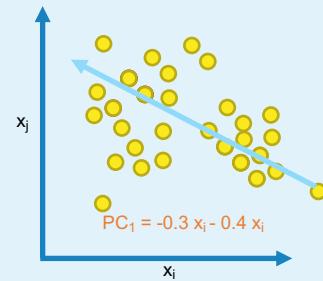
Grouping rows – e.g. creating groups of similar customers



H₂O algos:
k – means

Feature extraction:

Grouping columns – Create a small number of new representative dimensions

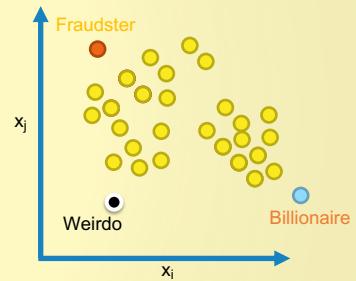


H₂O algos:

Principal components
Generalized low rank models
Autoencoders
Word2Vec

Anomaly detection:

Detecting outlying rows - Finding high-value, fraudulent, or weird customers



H₂O algos:

Principal components
Generalized low rank models
Autoencoders

H₂O.ai

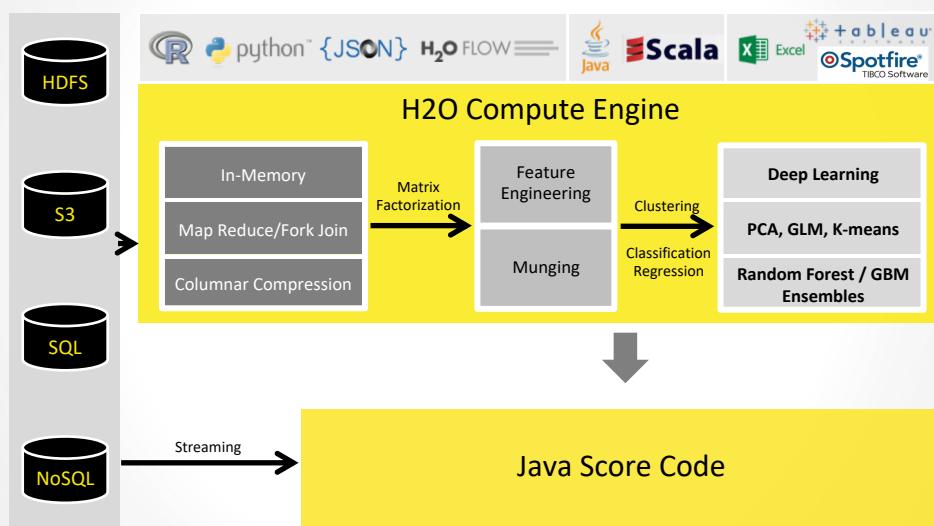
Introduction:

INTRODUCTION TO H₂O

H2O Machine Learning Methods

Supervised Learning		Unsupervised Learning	
Statistical Analysis	<ul style="list-style-type: none"> Penalized Linear Models: Super-fast, super-scalable, and interpretable CoxPH: Cox Proportional Hazards Survival Analysis Naïve Bayes: Straightforward linear classifier 	Clustering	<ul style="list-style-type: none"> K-means: Partitions observations into similar groups; automatically detects number of groups
Decision Tree Ensembles	<ul style="list-style-type: none"> Distributed Random Forest: Easy-to-use tree-bagging ensembles Gradient Boosting Machine: Highly tunable tree-boosting ensembles eXtreme Gradient Boosting: Popular XGBoost algorithm in H2O 	Dimensionality Reduction	<ul style="list-style-type: none"> Principal Component Analysis: Transforms correlated variables to independent components Generalized Low Rank Models: Extends the idea of PCA to handle arbitrary data consisting of numerical, Boolean, categorical, and missing data
Stacking	<ul style="list-style-type: none"> Stacked Ensemble: Combine multiple types of models for better predictions Automatic Machine Learning: Automated exploration of supervised learning approaches 	Aggregator	<ul style="list-style-type: none"> Aggregator: Efficient, advanced sampling that creates smaller data sets from larger data sets
Neural Networks		H2O.ai	
Multilayer Perceptron	<ul style="list-style-type: none"> Deep neural networks: Multi-layer feed-forward neural networks for standard data mining tasks Convolutional neural networks: Sophisticated architectures for pattern recognition in images, sound, and text 	Anomaly Detection	<ul style="list-style-type: none"> Autoencoders: Find outliers using a nonlinear dimensionality reduction technique Word2vec: Generate context-sensitive numerical representations of a large text corpus
Deep Learning		Term Embeddings	

Data and Client Agnostic



R, Python and Flow

The screenshot displays two parallel environments: a Jupyter notebook on the left and the H2O Flow interface on the right.

Jupyter Notebook (Left):

- In [1]:** R code for reading a dataset and performing a t-test on variable importance.
- In [2]:** Python code for training a Random Forest model and printing variable importance.
- In [3]:** R code for training a Deep Learning model and printing AUC metrics.
- In [4]:** Python code for creating a new DataFrame of crime observations.
- Out[?]:** Output showing AUC Train and AUC Test for GBM, DL, and RF models.

H2O Flow (Right):

- Setup Parse:** Configuration for parsing a CSV file from S3.
- PARSE CONFIGURATION:** Set to CSV, separator is '|', column headers are 'Auto'.
- EDIT COLUMN NAMES AND TYPES:** Shows a table of columns 1 through 8 with types: Numeric.

H2O.ai

R Interface Overview

Action	R	H2O
Reading data	<code>read_csv(data_path)</code>	<code>h2o.importFile(data_path)</code>
Summarizing data	<code>summary(data_frame)</code>	<code>h2o.summary(h2o_frame)</code>
Summary statistics	<code>mean(data_frame[, "x"])</code>	<code>h2o.mean(h2o_frame)</code>
Combining rows	<code>rbind(data_frame1, data_frame2)</code>	<code>h2o.rbind(h2o_frame1, h2o_frame2)</code>
Combining columns	<code>cbind(data_frame1, data_frame2)</code>	<code>h2o.cbind(h2o_frame1, h2o_frame2)</code>
Data selection	<code>data_frame[,]</code>	<code>h2o_frame[,]</code>
Transforming columns	<code>log(data_frame[, "x"])</code> <code>sqrt(data_frame[, "x"])</code>	<code>log(h2o_frame[, "x"])</code> <code>sqrt(h2o_frame[, "x"])</code>
Building Random Forest	<code>model = randomForest(y ~ x, data_frame)</code>	<code>model = h2o.randomForest(x, y, train_frame)</code>
Model Prediction	<code>predict(model, data_frame)</code>	<code>h2o.predict(model, h2o_frame)</code>
Model Metrics	<code>performance(model)</code> <code>auc(model)</code>	<code>metrics = model.model_performance(frame)</code> <code>h2o.auc(model)</code>

H2O.ai

Python Interface Overview

Action	Pandas or scikit-learn	H2O
Reading data	pandas.read_csv(data_path)	h2o.import_file(data_path)
Summarizing data	pandas_frame.describe()	h2o_frame.describe()
Summary statistics	pandas_frame.mean()	h2o_frame.mean()
Combining rows	pandas.concat(list[frame1,frame2])	h2o_frame.rbind(h2o_frame2)
Combining columns	pandas.concat(list[frame1,frame2],axis = 1)	h2o_frame.cbind(h2o_frame2)
Data selection	pandas_frame[:, :]	h2o_frame[:, :]
Transforming columns	np.log(pandas_frame[x]) np.sqrt(pandas_frame[x])	h2o_frame[x].log() h2o_frame[x].sqrt()
Building Random Forest	model = RandomForestClassifier(n_estimators = 100) model = model.fit(x_frame, y_frame)	model = H2ORandomForestClassifier(n_trees = 100) model = model.train(x, y, train_frame)
Model Prediction	model.predict	model.predict
Model Metrics	metrics.auc	metrics = model.model_performance(frame) metrics.auc()

H2O.ai

Reading Data into H2O with Python

STEP 1



Python user

→ h2o_df = h2o.import_file("../data/allyears2k.csv")

H2O.ai

Reading Data into H2O with Python

STEP 2

Python

`h2o.import_file()`

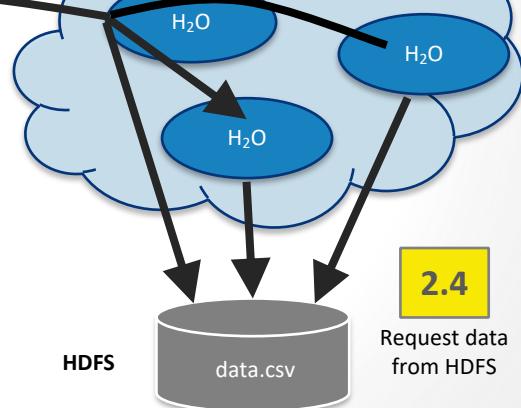
2.1

Python function call

2.2
HTTP REST API request to H2O has HDFS path

2.3

Initiate distributed ingest



2.4

Request data from HDFS

H2O.ai

Reading Data into H2O with Python

STEP 3

Python

`h2o_df`

Cluster IP
Cluster Port
Pointer to Data

3.4

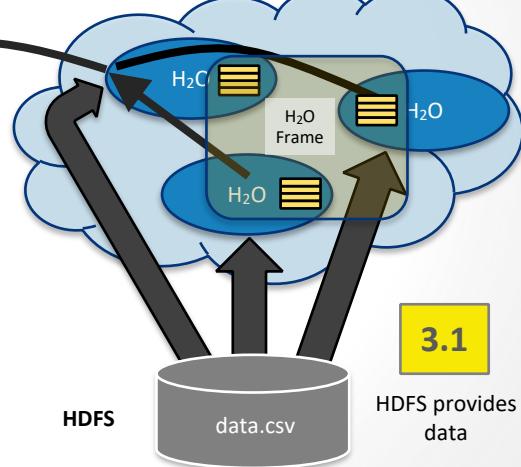
`h2o_df` object created in Python

3.3

Return pointer to
data in REST API
JSON Response

3.2

Distributed H2O
Frame in DKV

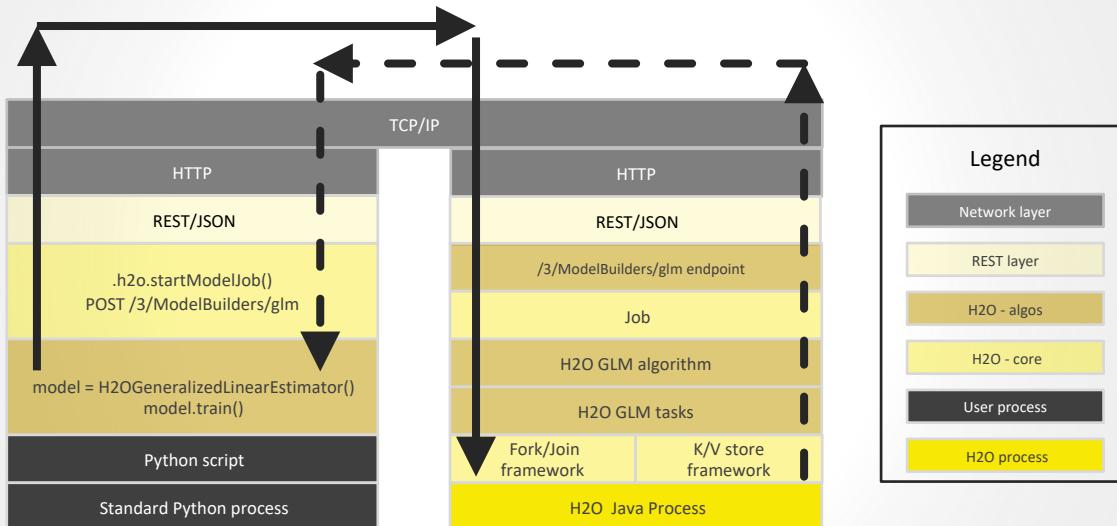


3.1

HDFS provides
data

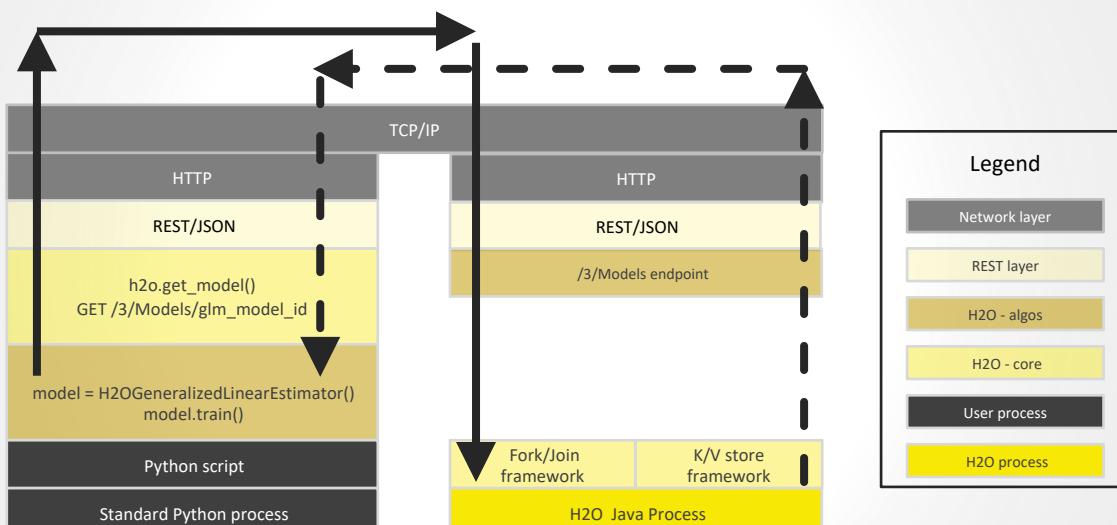
H2O.ai

Training GLM from Python



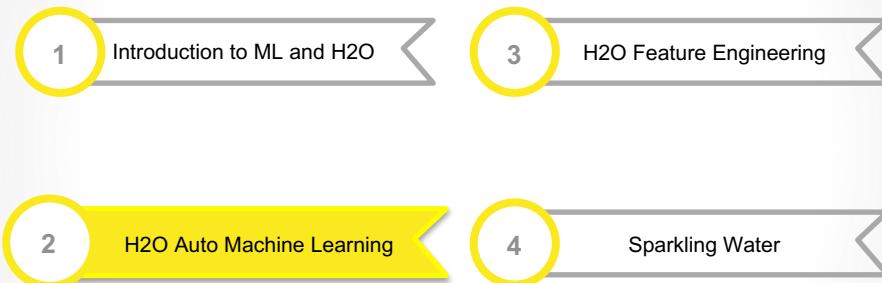
H2O.ai

Retrieving GLM Result from Python



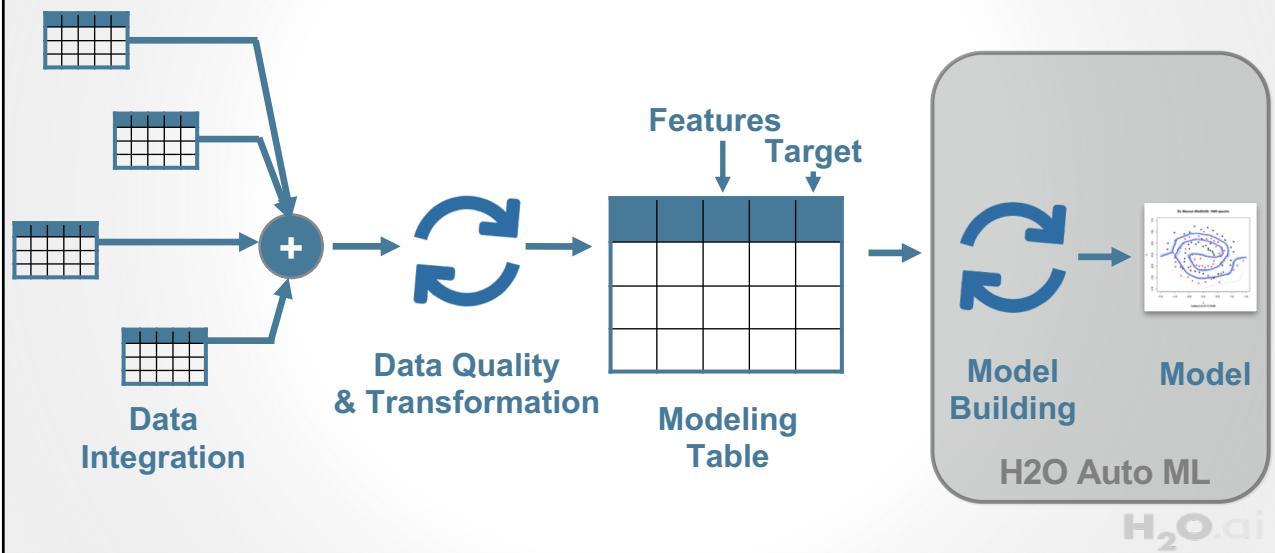
H2O.ai

H2O Training

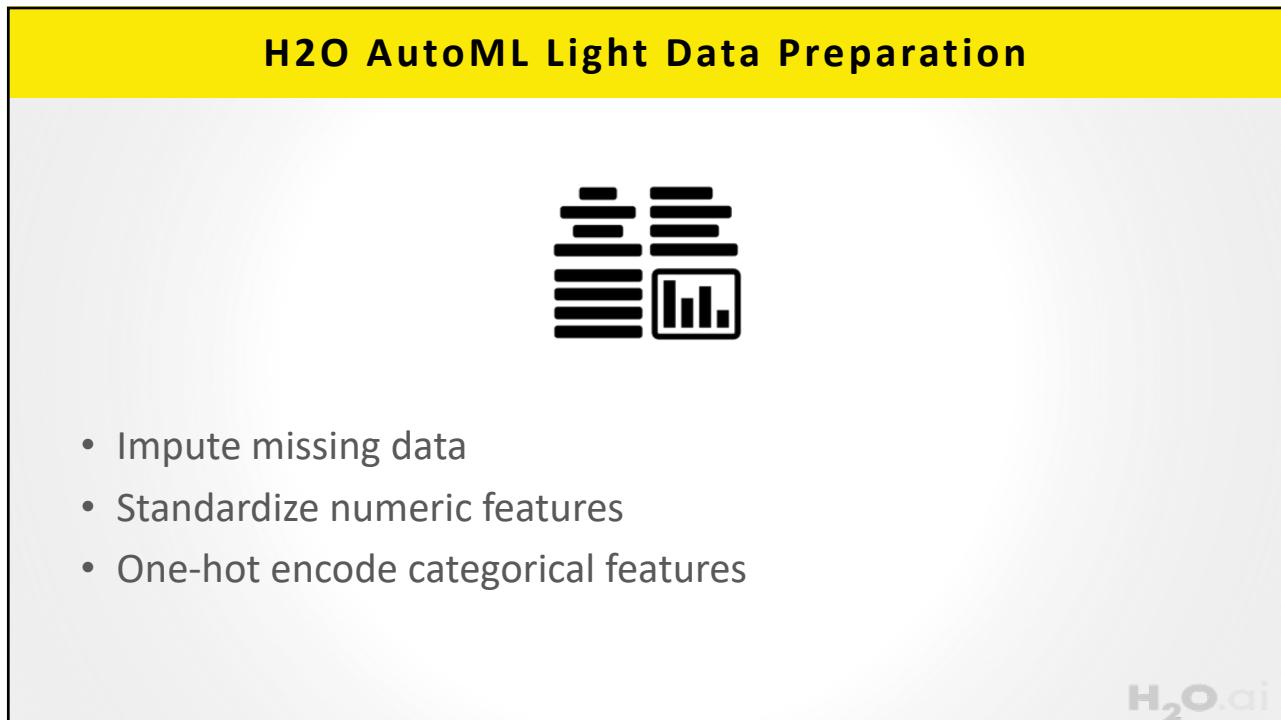
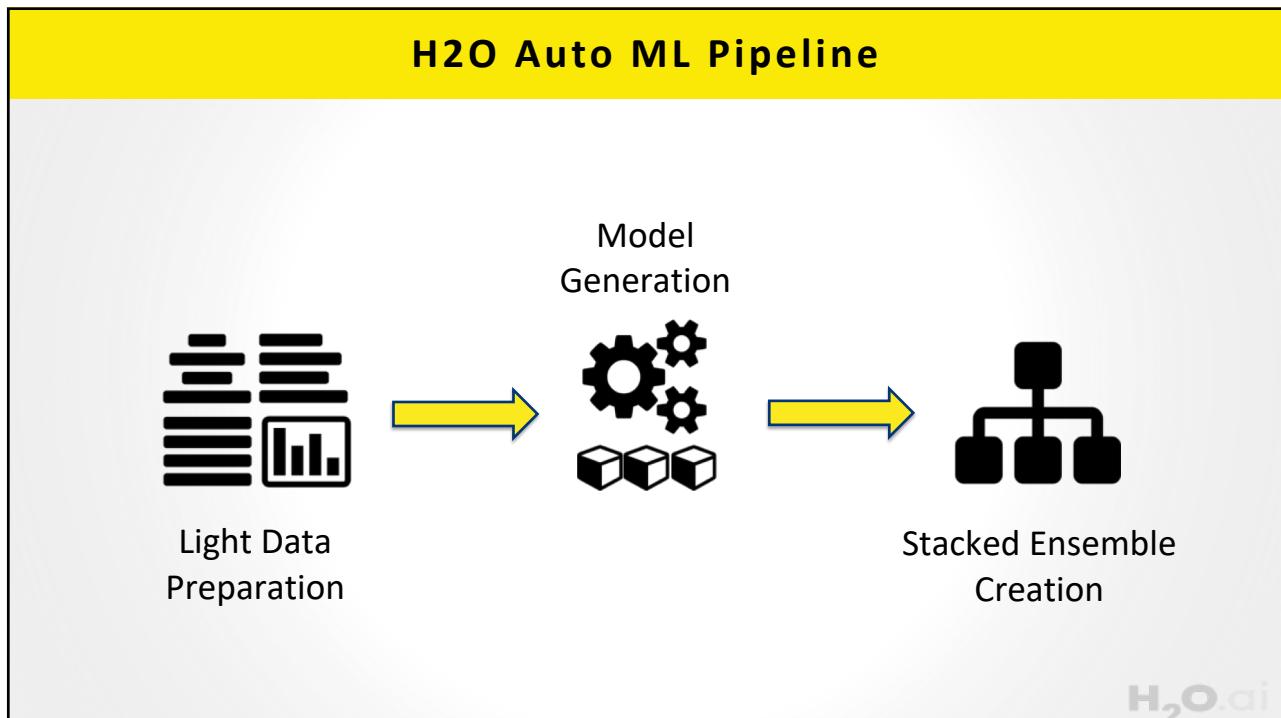


H2O.ai

H2O Auto ML in Typical Enterprise Workflow



H2O.ai



H2O AutoML Model Generation



- Explore multiple algorithms
 - GLMs, Random Forests, GBMs, Deep Neural Networks
- Perform random grid search over hyper-parameter space
- Use early stopping rules for models and grids

H₂O.ai

H2O AutoML Stacked Ensemble Creation



- Stacked ensembles of all models
- Stacked ensembles of best model for each algorithm

H₂O.ai

H2OAutoML R Syntax

```
h2o.automl(x, y,
            training_frame,
            validation_frame = NULL,
            leaderboard_frame = NULL,
            nfolds = 5,
            fold_column = NULL,
            weights_column = NULL,
            max_runtime_secs = 3600,
            max_models = NULL,
            stopping_metric = c("AUTO", "deviance", "logloss", "MSE",
                               "RMSE", "MAE", "RMSLE", "AUC", "lift_top_group",
                               "misclassification", "mean_per_class_error"),
            stopping_tolerance = NULL,
            stopping_rounds = 3,
            seed = NULL,
            project_name = NULL,
            exclude_algos = NULL)
```



H₂O.ai

H2OAutoML Python Syntax



```
# Set up automatic machine learning experiment
aml = H2OAutoML(nfolds = 5, max_runtime_secs = 3600, max_models = None,
                 stopping_metric = 'AUTO', stopping_tolerance = None,
                 stopping_rounds = 3, seed = None, project_name = None,
                 exclude_algos = None)
```

```
# Train models
aml.train(x = None, y = None, training_frame = None, fold_column = None,
          weights_column = None, validation_frame = None,
          leaderboard_frame = None)
```

H₂O.ai

Case Study: Lending Club Dataset

https://github.com/h2oai/h2o-tutorials/blob/master/training/h2o_3_hands_on/automl/lending_club_automl.ipynb

- Loan data from 2007 up until 2015 including rejected applications and accepted applications.
- Of the 500k accepted applicants about 160k loans have either been completely paid off or defaulted.
- There are about 4 million applicants in the rejected loans dataset.
- **Use Case 1:** Predict the likelihood of a user defaulting based on the information supplied when applying for a loan.
- **Use Case 2:** Determine the interest rate Lending Club would have offered the user based on the information supplied when applying for a loan.
- Full Data: <https://www.kaggle.com/wendykan/lending-club-loan-data>
- H2O Subset: <https://s3.amazonaws.com/h2o-public-test-data/bigdata/laptop/lending-club/loan.csv>

H₂O.ai

Case Study: Lending Club Dataset

	Column Name	Description	Unit	Role
1	loan_amnt	Requested loan amount	US dollars	Predictor
2	term	Longest term length	Months	Predictor
3	int_term	Recommended interest rate	Rate	Response
4	emp_length	Employment length	Years	Predictor
5	home_ownership	Housing status	Categorical	Predictor
6	annual_inc	Annual income	US dollars	Predictor
7	purpose	Purpose for the loan	Categorical	Predictor
8	addr_state	State of residence	Categorical	Predictor
9	dti	Debt to income ratio	Percent	Predictor
10	delinq_2yrs	Number of delinquencies in the past 2 years	Count	Predictor
11	revol_util	Revolving credit line utilized	Percent	Predictor
12	total_acc	Number of active accounts	Count	Predictor
13	bad_loan	Bad loan indicator	Boolean	Response
14	longest_credit_length	Age of oldest active account	Years	Predictor
15	verification_status	Income verification status	Boolean	Predictor

Case Study: Auto ML of Lending Club Dataset

```

1 # Load package and connect to cluster
2 library(h2o)
3 h2o.init(max_mem_size = "6g")

5 # Import data and manage data types
6 train_path <- "https://raw.githubusercontent.com/h2oai/app-consumer-loan/master/data/loan.csv"
7 train <- h2o.importFile(train_path, destination_frame = "loan_train")
8 train[["bad loan"]] = h2o.asfactor(train[["bad loan"]])

10 # Set target and predictor variables
11 y <- "bad_loan"
12 x <- h2o.colnames(train)
13 x <- setdiff(x, c(y, "int rate"))

15 # Use Auto ML to train models
16 aml <- h2o.automl(x = x, y = y, training_frame = train, max_runtime_secs = 300)

```



H₂O.ai

Case Study: Auto ML of Lending Club Dataset

```

1 # Load package and connect to cluster
2 import h2o
3 h2o.init(max_mem_size = "6g")

5 # Import data and manage data types
6 train_path = "https://raw.githubusercontent.com/h2oai/app-consumer-loan/master/data/loan.csv"
7 train = h2o.import_file(train_path, destination_frame = "loan_train")
8 train[["bad loan"]] = train[["bad loan"]].asfactor()

10 # Set target and predictor variables
11 y = "bad_loan"
12 x = train.col_names
13 x.remove(y)
14 x.remove("int rate")

16 # Use Auto ML to train models
17 from h2o.automl import H2OAutoML
18 aml = H2OAutoML(max_runtime_secs = 300)
19 aml.train(x = x, y = y, training frame = train)

```



H₂O.ai

Operationalizing Machine Learning:

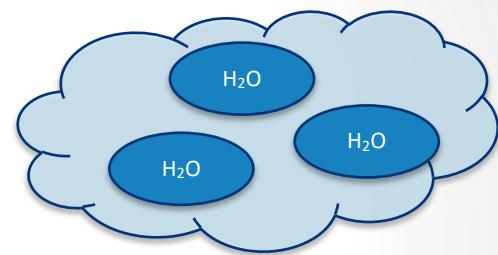
PRODUCTIONALIZING MODELS

Productionalizing Models



Immediate Action

- H2O Real-Time (RT) Scoring
 - POJO/MOJO + JRE
 - *Feature engineering not included*



Later Action

- H2O Off-Line (Batch) Scoring
 - H2O Cluster
 - *Feature engineering supported*

R Syntax H2O Scoring

```
# Save Real-Time Scoring Code
h2o.download_pojo(model, path = NULL, get_jar = TRUE, jar_name = "")

h2o.download_mojo(model, path = getwd(), get_genmodel_jar = FALSE,
                  genmodel_name = "")
```



```
# Save / Load Batch Scoring Model
h2o.saveModel(object, path = "", force = FALSE)
model <- h2o.loadModel(path)

# Batch Scoring
h2o.predict(model, newdata, ...)
```



Python Syntax H2O Scoring



```
# Save Real-Time Scoring Code
model.download_pojo(path = u'', get_genmodel_jar = False, genmodel_name = u'')

model.download_mojo(path = u'', get_genmodel_jar = False, genmodel_name = u'')
```

```
# Save / Load Batch Scoring Model
h2o.save_model(model, path = u'', force = False)
model = h2o.load_model(path)

# Batch Scoring
model.predict(test_data, custom_metric = None, custom_metric_func = None)
```



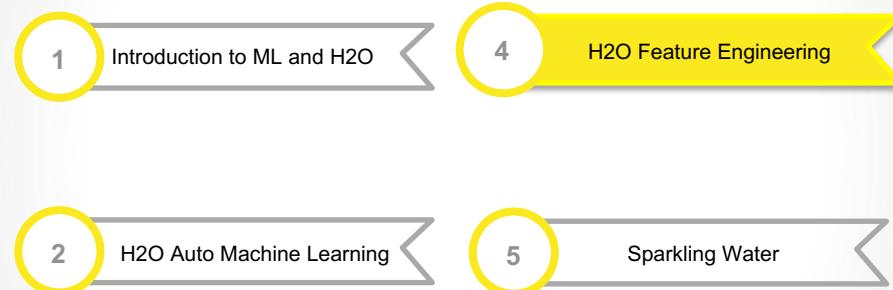
Java Syntax H2O Real-Time Scoring

```
import java.io.*;
import hex.genmodel.easy.RowData;
import hex.genmodel.easy.EasyPredictModelWrapper;
import hex.genmodel.easy.prediction.*;

public class main {
    private static String modelClassName = "model_pojo";
    public static void main(String[] args) throws Exception {
        hex.genmodel.GenModel rawModel;
        rawModel = (hex.genmodel.GenModel)
            Class.forName(modelClassName).newInstance();
        EasyPredictModelWrapper model = new EasyPredictModelWrapper(rawModel);
        RowData row = new RowData();
        BinomialModelPrediction p = model.predictBinomial(row);
    }
}
```

H₂O.ai

H2O Training



H₂O.ai

H2O Data Munging

- Data Transfer
- H2OFrame Attributes
- Data Column Types
- Row & Columns Selection
- Categorical Data Operations
- Filters & Logical Operations
- Missing Data Handling
- Summary & Aggregation
- Numeric Data Summaries
- Numeric Data Transformations
- Date/Time Manipulations
- String Munging
- Joins Between Two H2OFrames
- Histogram of Numeric Columns

H2O.ai

Data Transfer between Python and H2O

`h2o.H2OFrame`

- Turns python pandas frame into an H2OFrame
- Saves dataframe to CSV and then performs `h2o.upload()` to H2O cluster

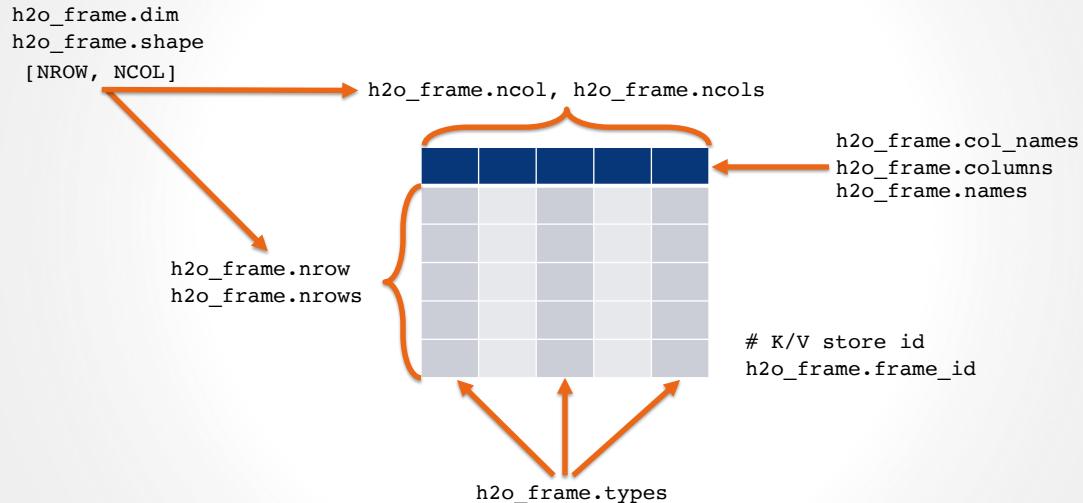
`h2o.as_list` OR `h2o_frame.as_data_frame`

- Turns H2OFrame to lists or list or pandas frame
- Downloads the H2OFrame locally and will parse with pandas if `use_pandas` is set to True

Note: Be mindful of `h2o.as_list`, it is difficult to fit a large H2OFrame into a Python session

H2O.ai

H2OFrame Attributes



H2O.ai

H2O Column Types

Enum

```
data[x].asfactor()
```

Used to convert given variable to an enumerated type variable

(booleans are coded as enum)

Numeric

```
data[x].asnumeric()
```

Used to convert given variable to an numeric variable

Dates

```
data[x].year()
```

Convert the entries of an H2OFrame object from milliseconds to years, indexed starting from 1900.

```
data[x].month()
```

Converts the entries of an H2OFrame object from milliseconds to months (on a 1 to 12 scale).

String

```
data[x].ascharacter()
```

Used to convert given variable to an string type variable

H2O.ai

H2OFrame Example

```

1 import h2o
2 h2o.init()
3 data_path = "https://s3.amazonaws.com/h2o-public-test-data/smalldata/census_income/adult_data.csv"
4 census_data = h2o.import_file(data_path, destination_frame = "census_init")
Parse progress: |██████████| 100%
1 print(census_data.types)

{'age': 'int',
 'workclass': 'enum',
 'fnlwgt': 'int',
 'education': 'enum',
 'education-num': 'int',
 'marital-status': 'enum',
 'occupation': 'enum',
 'relationship': 'enum',
 'race': 'enum',
 'sex': 'enum',
 'capital-gain': 'int',
 'capital-loss': 'int',
 'hours-per-week': 'int',
 'native-country': 'enum',
 'income': 'enum'}

```

H₂O.ai

H2OFrame Example

```
1 census_data[0:5].summary() # census_data[0:5].describe()
```

	age	workclass	fnlwgt	education	education-num
type	int	enum	int	enum	int
mins	17.0		12285.0		1.0
mean	38.581646755321145		189778.36651208595		10.080679340315212
maxs	90.0		1484705.0		16.0
sigma	13.640432553581354		105549.97769702227		2.5727203320673966
zeros	0		0		0
missing	0	1836	0	0	0
0	39.0	State-gov	77516.0	Bachelors	13.0
1	50.0	Self-emp-not-inc	83311.0	Bachelors	13.0
2	38.0	Private	215646.0	HS-grad	9.0
3	53.0	Private	234721.0	11th	7.0

H₂O.ai

Row & Column Selection

Pandas-like convention for slicing and dicing data

(0-based indexes)

Extracting a single column

- `h2o_frame["x"] # column name x`
- `h2o_frame[2] # 3rd col`
- `h2o_frame[-2] # 2nd col from end`
- `h2o_frame[:, -1] # Last column`

Filtering rows

- `h2o_frame[0:5, :]`
- `h2o_frame[h2o_frame["x"] > 1, :]`

Extracting multiple columns

- `h2o_frame[["x", "y", "z"]]`
- `h2o_frame[[1, 5, 6]]`

Filtering rows for select columns

```
h2o_frame[0:50, [1,2,3]]  
  
med = h2o_frame["a"].median()  
h2o_frame[h2o_frame["a"] > med, "z"]
```

H₂O.ai

Filters & Logical Operations

• Logical Operators

- `h2o_frame[x].logical_negation()`
- `h2o_frame[x] & h2o_frame[y]`
- `h2o_frame[x] | h2o_frame[y]`

• Comparison Operators

- `h2o_frame[x] {==, !=, <, <=, >, >} value`
- `h2o_frame[x] {==, !=, <, <=, >, >} h2o_frame[y]`

• Logical Data Summaries

- `h2o_frame[x].all() # includes NAs`
- `h2o_frame[x].any() # includes NAs`
- `h2o_frame[x].any_na_rm() # disregards NAs`

H₂O.ai

Filters & Logical Operations

```
test.ifelse(yes, no)
```

Arguments

test	A logical description of the condition to be met (>, <, =, etc...)
yes	The value to return if the condition is TRUE.
no	The value to return if the condition is FALSE.

Equivalent to [y if t else n for t,y,n in zip(self,yes,no)]

Note: Only numeric values can be tested, and only numeric results can be returned.

H₂O.ai

Categorical Data Operations

- Metadata
 - h2o_frame[x].categories()
 - h2o_frame[x].levels()
 - h2o_frame[x].nlevels()
- Categorical Data Manipulation
 - h2o_frame.relevel(y)
 - h2o_frame.set_level(level)
 - h2o_frame.set_levels(levels)

H₂O.ai

Data Selection Example

```

1 tech_support_income = census_data[census_data["occupation"] == "Tech-support", "income"]
2 print(type(tech_support_income))
<class 'h2o.frame.H2OFrame'>
1 print(tech_support_income.table())
income  Count
<=50K    645
>50K    283

```

H₂O.ai

Missing Data Handling

- Missing Data Code
 - None
- Missing Data Filtering
 - h2o_frame.filter_na_cols(frac=0.2)
 - h2o_frame.na.omit()
- Missing Data Replacement
 - h2o_frame.impute(...)
 - H2OGeneralizedLowRankEstimator
- Missing Data Inclusion
 - h2o_frame.insert_missing_values(fraction=0.1, seed=None)
- Missing Data Summaries
 - h2o_frame.nacnt()

H₂O.ai

Missing Data Handling

```
h2o_frame.impute(column = 0, method = c("mean", "median",
"mode"), combine_method c("interpolate", "average", "lo", "hi"),
by = NULL, groupByFrame = NULL, values = NULL)
```

Arguments

h2o_frame	The dataset containing the column to impute.
column	The column to impute.
method	"mean" replaces NAs with the column mean; "median" replaces NAs with the column median; "mode" replaces with the most common factor (for factor columns only)
combine_method	If method is "median", then choose how to combine quantiles on even sample sizes. This parameter is ignored in all other cases
by	Group by columns
groupByFrame	Impute the column col with this pre-computed grouped frame.
values	A vector of impute values (one per column). NaN indicates to skip the column

H₂O.ai

Missing Data Handling

```
1 nacnts = dict(zip(census_data.col_names, census_data.nacnt()))
2 print(dict((k, int(v)) for (k, v) in nacnts.items() if v > 0))
{'workclass': 1836, 'occupation': 1843, 'native-country': 583}

1 codes = census_data["native-country"].asnumeric()
2 levels = census_data["native-country"].levels()[0]
3 levels.append("Unknown")
4
5 census_data["native-country-clean"] = h2o.H2OFrame.ifelse(codes != None, codes, len(levels))
6 census_data["native-country-clean"] = census_data["native-country-clean"].asfactor()
7 census_data["native-country-clean"] = census_data["native-country-clean"].set_levels(levels)
8
9 print((census_data["native-country-clean"] == "Unknown").table())

native-country-clean Count
0 31978
1 583
```

H₂O.ai

Summary & Aggregation

```
h2o_frame[x].table(dense = TRUE)
```

Arguments

h2o_frame	An H2OFrame object with at least one column
x	Column name
dense	A logical for dense representation, which lists only non-zero counts, 1 combination per row. Set to FALSE to expand counts across all combinations.

Value: Returns a tabulated H2OFrame Object

H₂O.ai

Summary & Aggregation

```
h2o.group_by(data, by, ..., gb.control =
list(na.methods = NULL, col.names = NULL))
```

Arguments

data	an H2OFrame object.
by	a list of column names
gb.control	a list of how to handle NA values in the dataset as well as how to name output columns
... .	Any supported aggregated function: mean, min, max, sum, sd, nrow

H₂O.ai

Numeric Data Summaries

- `h2o_frame[x].cor(y=None, na_rm=False, use=None)`
- `h2o_frame[x].kurtosis(na_rm=False)`
- `h2o_frame[x].max()`
- `h2o_frame[x].mean(skipna=False)`
- `h2o_frame[x].median(na_rm=False)`
- `h2o_frame[x].min()`
- `h2o_frame[x].prod(na_rm=False)`
- `h2o_frame[x].quantile(...)`
- `h2o_frame[x].sd(na_rm=False)`
- `h2o_frame[x].skewness(na_rm=False)`
- `h2o_frame[x].sum(skipna=True)`
- `h2o_frame[x].var(y=None, na_rm=False, use=None)`

H₂O.ai

Group By Aggregation

```
1 grouped_data = census_data[["occupation", "education-num"]].group_by(["occupation"])
2 stats = grouped_data.count(na = "ignore").median(na = "ignore").mean(na = "ignore").sd(na = "ignore")
3 stats.get_frame()
```

occupation	nrow	median_education-num	mean_education-num	sdev_education-num
	0	9	9.25339	2.60279
Adm-clerical	3770	10	10.1135	1.69805
Armed-Forces	9	9	10.1111	2.02759
Craft-repair	4099	9	9.11076	2.03865
Exec-managerial	4066	12	11.4491	2.14321
Farming-fishing	994	9	8.60865	2.75607
Handlers-cleaners	1370	9	8.51022	2.20338
Machine-op-inspct	2002	9	8.48751	2.28528
Other-service	3295	9	8.77967	2.29966
Priv-house-serv	149	9	7.36242	3.11104

H₂O.ai

Cross-Validated Mean Target Encoding

(Feature Engineering Sneak Peak)

```

1 def mean_target_encoding(data, x, y, fold_column):
2     grouped_data = data[[x, fold_column, y]].group_by([x, fold_column])
3     grouped_data.sum(na = "ignore").count(na = "ignore")
4     df = grouped_data.get_frame().as_data_frame()
5     df_list = []
6     nfold = int(data[fold_column].max()) + 1
7     for j in range(0, nfold):
8         te_x = "te_{}".format(x)
9         sum_y = "sum_{}".format(y)
10        oof = df.loc[df[fold_column] != j, [x, sum_y, "nrow"]]
11        stats = oof.groupby([x]).sum()
12        stats[x] = stats.index
13        stats[fold_column] = j
14        stats[te_x] = stats[sum_y] / stats["nrow"]
15        df_list.append(stats[[x, fold_column, te_x]])
16    return h2o.H2OFrame(pd.concat(df_list))

```

H₂O.ai

Numeric Data Transformations

- h2o_frame[x].abs()
- h2o_frame[x].acos()
- h2o_frame[x].acosh()
- h2o_frame[x].asin()
- h2o_frame[x].asinh()
- h2o_frame[x].atan()
- h2o_frame[x].atanh()
- h2o_frame[x].ceil()
- h2o_frame[x].cos()
- h2o_frame[x].cosh()
- h2o_frame[x].cospi()
- h2o_frame[x].cut(breaks, ...)
- h2o_frame[x].digamma()
- h2o_frame[x].exp()
- h2o_frame[x].expm1()
- h2o_frame[x].floor()
- h2o_frame[x].gamma()
- h2o_frame[x].lgamma()
- h2o_frame[x].log()
- h2o_frame[x].log10()
- h2o_frame[x].log1p()
- h2o_frame[x].log2()
- h2o_frame[x].round(digits=0)
- h2o_frame[x].scale(center=True, scale=True)
- h2o_frame[x].sign()
- h2o_frame[x].signif(digits=6)
- h2o_frame[x].sin()
- h2o_frame[x].sinh()
- h2o_frame[x].sinpi()
- h2o_frame[x].sqrt()
- h2o_frame[x].tan()
- h2o_frame[x].tanh()
- h2o_frame[x].tanpi()
- h2o_frame[x].trigamma()
- h2o_frame[x].trunc()

H₂O.ai

Numeric Data Transformations

Transformation for skewed data with positive and negative values

```

1 import math
2
3 def pseudo_log10(x):
4     return math.asinh(x / 2) / math.log(10)
5
6 print("pseudo_log10(\u261{}) = \u261{:0.6f}".format(100000, pseudo_log10(100000)))
7 print("pseudo_log10(\u261{}) = \u261{:0.6f}".format(10000, pseudo_log10(10000)))
8 print("pseudo_log10(\u261{}) = \u261{:0.6f}".format(1000, pseudo_log10(1000)))
9 print("pseudo_log10(\u261{}) = \u261{:0.6f}".format(100, pseudo_log10(100)))
10 print("pseudo_log10(\u261{}) = \u261{:0.6f}".format(10, pseudo_log10(10)))
11 print("pseudo_log10(\u261{}) = \u261{:0.6f}".format(1, pseudo_log10(1)))
12 print("pseudo_log10({}) = {}".format(0, pseudo_log10(0)))

pseudo_log10(\u261100000) = \u2615.000000
pseudo_log10(\u26110000) = \u2614.000000
pseudo_log10(\u2611000) = \u2613.000000
pseudo_log10(\u261100) = \u2612.000043
pseudo_log10(\u26110) = \u2611.004279
pseudo_log10(\u2611) = \u2610.208988
pseudo_log10(0) = 0.0

```

H₂O.ai

Numeric Data Transformations

```

1 import numpy as np
2 breaks = np.linspace(10, 90, 9).tolist()
3 census_data["age_group"] = census_data["age"].cut(breaks)
4
5 census_data["log1p_capital-gain"] = census_data["capital-gain"].log1p()
6 census_data["log1p_capital-loss"] = census_data["capital-loss"].log1p()
7 print(census_data["age_group"].table())

```

age_group	Count
(10.0,20.0]	2410
(20.0,30.0]	8162
(30.0,40.0]	8546
(40.0,50.0]	6983
(50.0,60.0]	4128
(60.0,70.0]	1792
(70.0,80.0]	441
(80.0,90.0]	99

H₂O.ai

Date/Time Manipulations

- Date/Time Creation
 - `h2o_frame[x].as_date(format)`
- Date Extraction
 - `h2o_frame[x].day()`
 - `h2o_frame[x].dayOfWeek()`
 - `h2o_frame[x].month()`
 - `h2o_frame[x].week()`
 - `h2o_frame[x].year()`
- Time Extraction
 - `h2o_frame[x].hour()`
 - `h2o_frame[x].minute()`
 - `h2o_frame[x].second()`

H₂O.ai

String Munging

- String Matching
 - `h2o_frame[x].countmatches()`
 - `h2o_frame[x].grep()`
 - `h2o_frame[x].match()`
- Substitute pattern match
 - `h2o_frame[x].gsub() # Replace all occurrences`
 - `h2o_frame[x].sub() # Replace first occurrence`
- String Cleaning
 - `h2o_frame[x].substring()`
 - `h2o_frame[x].strsplit()`
 - `h2o_frame[x].trim()`
 - `h2o_frame[x].lstrip()`
 - `h2o_frame[x].rstrip()`

H₂O.ai

Joins Between Two H2OFrames

```
h2o_frame.merge(other, all_x=False, all_y=False, by_x=None,
                 by_y=None, method='auto')
```

Arguments

h2o_frame	left/self data set in the join.
other	right/other data set in the join.
all_x	If True, include all rows from the left/self frame.
all_y	If True, include all rows from the right/other frame.
by_x	list of columns in the left/self frame to use as a merge key.
by_y	list of columns in the right/other frame to use as a merge key.
method	string representing the merge method, one of auto(default), radix or hash.

H₂O.ai

Joins Between Two H2OFrames

```
1 census_data["cv_fold"] = census_data.kfold_column(n_folds = 5, seed = 123)
2 census_data["high_income"] = census_data["income"] == ">50K"
3 te_occupation = mean_target_encoding(census_data, x = "occupation", y = "high_income",
                                         fold_column = "cv_fold")
Parse progress: |██████████| 100%
1 census_data = census_data.merge(te_occupation, all_x = True)
```

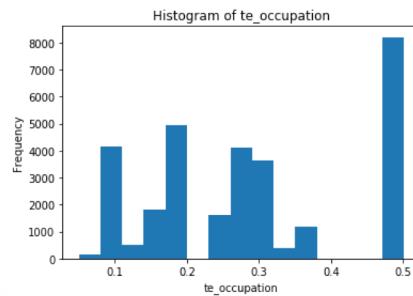
H₂O.ai

Histogram of Numeric Columns

```

1 import matplotlib.pyplot as plt
2 %matplotlib inline
3
4 import warnings
5 import matplotlib.cbook
6 warnings.filterwarnings("ignore", category = matplotlib.cbook.mplDeprecation)
7
8 census_data["te_occupation"].hist()

```



H₂O.ai

Case Study: Lending Club Dataset

- Loan data from 2007 up until 2015 including rejected applications and accepted applications.
- Of the 500k accepted applicants about 160k loans have either been completely paid off or defaulted.
- There are about 4 million applicants in the rejected loans dataset.
- **Use Case 1:** Predict the likelihood of a user defaulting based on the information supplied when applying for a loan.
- **Use Case 2:** Determine the interest rate Lending Club would have offered the user based on the information supplied when applying for a loan.
- Full Data: <https://www.kaggle.com/wendykan/lending-club-loan-data>
- H2O Subset: <https://s3.amazonaws.com/h2o-public-test-data/bigdata/laptop/lending-club/loan.csv>

H₂O.ai

Case Study: Lending Club Dataset

	Column Name	Description	Unit	Role
1	loan_amnt	Requested loan amount	US dollars	Predictor
2	term	Longest term length	Months	Predictor
3	int_term	Recommended interest rate	Rate	Response
4	emp_length	Employment length	Years	Predictor
5	home_ownership	Housing status	Categorical	Predictor
6	annual_inc	Annual income	US dollars	Predictor
7	purpose	Purpose for the loan	Categorical	Predictor
8	addr_state	State of residence	Categorical	Predictor
9	dti	Debt to income ratio	Percent	Predictor
10	delinq_2yrs	Number of delinquencies in the past 2 years	Count	Predictor
11	revol_util	Revolving credit line utilized	Percent	Predictor
12	total_acc	Number of active accounts	Count	Predictor
13	bad_loan	Bad loan indicator	Boolean	Response
14	longest_credit_length	Age of oldest active account	Years	Predictor
15	verification_status	Income verification status	Boolean	Predictor

Case Study: Lending Club Dataset

Predictor Column Name	Transformation
loan_amnt	Manage high and low amounts
term	Recode as 0/1 binary
emp_length	Create missing value indicator
home_ownership	Combine categories
annual_inc	Manage high and low annual incomes
purpose	Cross-Validated Mean Target Encoding
addr_state	Cross-Validated Mean Target Encoding
dti	Manage high and low debt-to-income ratios
delinq_2yrs	Manage high and low delinquency counts
revol_util	Manage high and low utilization
total_acc	Manage high and low number of accounts
longest_credit_length	Manage high and low credit lengths
verification_status	Recode as 0/1 binary

H2O Training



H₂O.ai

SPARKLING WATER

Why use Sparkling Water?

Spark Cluster

Leverage Spark Cluster

- Launch H2O-3 Machine Learning Engine on top of your Spark Cluster

Spark Data Munging

Leverage Spark Data Munging Functionality

- Transparent integration of H2O with Spark ecosystem
- Transparent use of H2O data structures and algorithms with Spark API

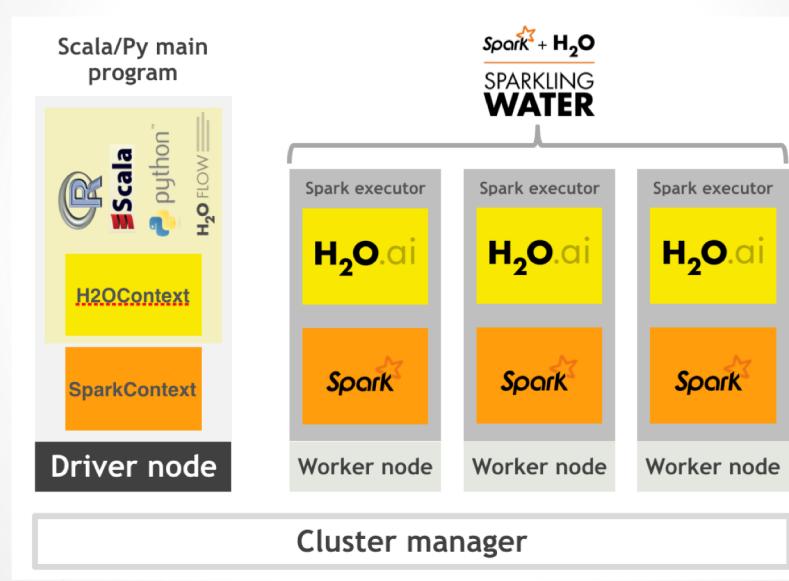
Spark Pipelines

Leverage Spark Pipelines

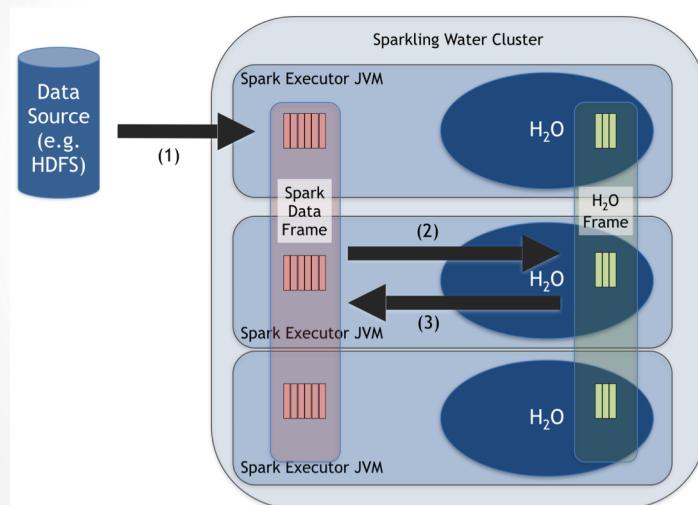
- Incorporate H2O-3 models into existing Spark pipelines
- Excels in existing Spark workflows requiring advanced Machine Learning algorithms

H₂O.ai

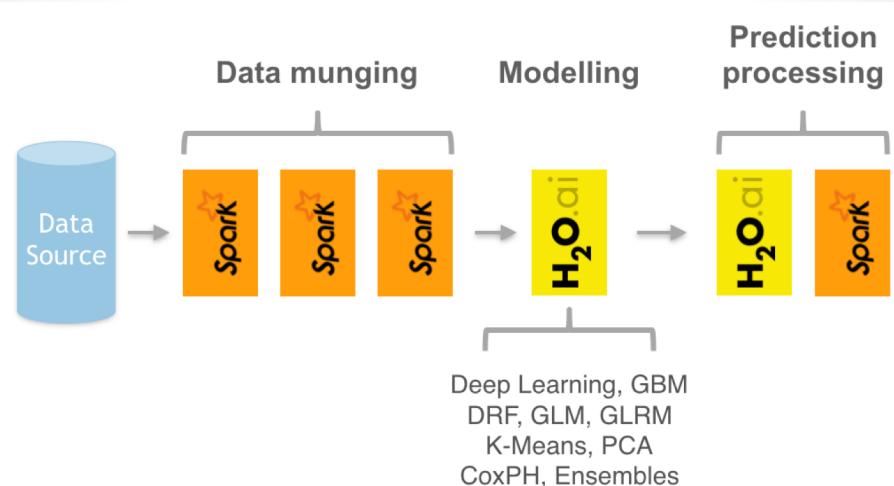
High Level Architecture



Sparkling Water Data Management

**H₂O.ai**

Use Case

**H₂O.ai**

Sparkling Water Data Conversion Functions

- Converting an H2OFrame into an RDD[T]

```
def asRDD[A <: Product : TypeTag : ClassTag](fr : H2OFrame) : RDD[A]
```

- Converting an H2OFrame into a DataFrame

```
def asDataFrame(fr : H2OFrame)(implicit sqlContext: SQLContext) : DataFrame
```

- Converting an RDD[T] into an H2OFrame

```
def asH2OFrame[A <: Product : TypeTag](rdd : RDD[A], frameName: Option[String]) : H2OFrame
```

- Converting a DataFrame into an H2OFrame

```
def asH2OFrame(rdd : DataFrame, frameName: Option[String]) : H2OFrame
```



Washington State Cannabis Sales

https://github.com/h2oai/h2o-tutorials/blob/master/training/sparkling_water_hands_on/time_series/WA_Cannabis_Sales_Daily.ipynb

- Washington State Cannabis Sales
 - Contains: daily sales metrics for different organizations in Washington state
 - Date Range: 2015-11-08 to 2017-03-04
 - Reference: <https://data.lcb.wa.gov/dataset/Dashboard-Usable-Sales-w-Weight-Daily/9wz2-qma2>
- Questions to answer:
 1. How many previous weeks influence cannabis sales?
 2. What days exhibit unusual sales?
 3. Do stores in counties that voted to legalize marijuana behave differently than those that didn't?

Washington State Cannabis Sales

Column Name	Description
SalesDate	Sales date
Organization	Organization name
County	County name
City	City name
SalesPrice	Transaction sales price
Freq	Number of times transaction occurred

H₂O.ai

Online Resources

- H2O Tutorials and Training Material
 - <https://github.com/h2oai/h2o-tutorials>
 - https://github.com/h2oai/h2o-tutorials/tree/master/training/lending_club_exercise
 - <https://github.com/h2oai/app-consumer-loan>
- Datasets
 - <https://s3.amazonaws.com/h2o-public-test-data/bigdata/laptop>
 - <https://s3.amazonaws.com/h2o-public-test-data/bigdata/laptop/lending-club/loan.csv>

H₂O.ai