

# "h2o"

May 22, 2015

## R topics documented:

h2o-package . . . . .	4
aaa . . . . .	5
apply,H2OFrame-method . . . . .	6
as.character,H2OFrame-method . . . . .	7
as.data.frame.H2OFrame . . . . .	7
as.environment,H2OFrame-method . . . . .	8
as.factor,H2OFrame-method . . . . .	8
as.h2o . . . . .	9
as.matrix.h2o . . . . .	9
as.numeric,H2OFrame-method . . . . .	10
ASTNode-class . . . . .	11
colnames<-,H2OFrame,H2OFrame-method . . . . .	11
cut.H2OFrame . . . . .	12
h2o.aic . . . . .	13
h2o.anomaly . . . . .	14
h2o.anyFactor . . . . .	14
h2o.assign . . . . .	15
h2o.auc . . . . .	15
h2o.betweeness . . . . .	16
h2o.cbind . . . . .	17
h2o.centers . . . . .	17
h2o.centersSTD . . . . .	18
h2o.clearLog . . . . .	18
h2o.clusterInfo . . . . .	19
h2o.clusterIsUp . . . . .	19
h2o.clusterStatus . . . . .	20
h2o.cluster_sizes . . . . .	20
h2o.coef . . . . .	21
h2o.coef_norm . . . . .	21
h2o.confusionMatrix . . . . .	21
h2o.createFrame . . . . .	22
h2o.day . . . . .	24
h2o.dayOfWeek . . . . .	25

h2o.ddply . . . . .	25
h2o.deepfeatures . . . . .	26
h2o.deeplearning . . . . .	27
h2o.dim . . . . .	31
h2o.downloadAllLogs . . . . .	31
h2o.downloadCSV . . . . .	32
h2o.download_pojo . . . . .	33
h2o.exportFile . . . . .	33
h2o.exportHDFS . . . . .	34
h2o.filterNACols . . . . .	35
h2o.gbm . . . . .	35
h2o.getConnection . . . . .	37
h2o.getFrame . . . . .	37
h2o.getGLMModel . . . . .	38
h2o.getModel . . . . .	38
h2o.getTimezone . . . . .	39
h2o.giniCoef . . . . .	39
h2o.glm . . . . .	40
h2o.glm . . . . .	43
h2o.group_by . . . . .	44
h2o.gsub . . . . .	45
h2o.head . . . . .	45
h2o.hist . . . . .	46
h2o.hit_ratio_table . . . . .	47
h2o.hour . . . . .	47
h2o.ifelse . . . . .	48
h2o.importFile . . . . .	49
h2o.impute . . . . .	50
h2o.init . . . . .	51
h2o.insertMissingValues . . . . .	54
h2o.interaction . . . . .	54
h2o.killMinus3 . . . . .	56
h2o.kmeans . . . . .	56
h2o.length . . . . .	57
h2o.levels . . . . .	58
h2o.listTimezones . . . . .	59
h2o.loadModel . . . . .	59
h2o.logAndEcho . . . . .	60
h2o.logloss . . . . .	61
h2o.ls . . . . .	61
h2o.makeGLMModel . . . . .	62
h2o.match . . . . .	62
h2o.mean . . . . .	63
h2o.merge . . . . .	64
h2o.metric . . . . .	64
h2o.month . . . . .	66
h2o.mse . . . . .	67
h2o.naiveBayes . . . . .	68

h2o.networkTest . . . . .	69
h2o.nlevels . . . . .	69
h2o.nrow . . . . .	70
h2o.null_deviance . . . . .	70
h2o.null_dof . . . . .	71
h2o.num_iterations . . . . .	71
h2o.openLog . . . . .	72
h2o.parseRaw . . . . .	72
h2o.parseSetup . . . . .	73
h2o.performance . . . . .	74
h2o.prcomp . . . . .	74
h2o.r2 . . . . .	75
h2o.randomForest . . . . .	76
h2o.rbind . . . . .	77
h2o.removeAll . . . . .	78
h2o.removeVecs . . . . .	79
h2o.rep_len . . . . .	79
h2o.residual_deviance . . . . .	80
h2o.residual_dof . . . . .	80
h2o.rm . . . . .	81
h2o.runif . . . . .	81
h2o.saveModel . . . . .	82
h2o.scale . . . . .	83
h2o.scoreHistory . . . . .	84
h2o.sd . . . . .	84
h2o.setLevel . . . . .	85
h2o.setLevels . . . . .	85
h2o.setTimezone . . . . .	86
h2o.shim . . . . .	86
h2o.shutdown . . . . .	87
h2o.splitFrame . . . . .	88
h2o.startGLMJob . . . . .	88
h2o.startLogging . . . . .	90
h2o.stopLogging . . . . .	91
h2o.strsplit . . . . .	91
h2o.sub . . . . .	92
h2o.summary . . . . .	92
h2o.svd . . . . .	93
h2o.table . . . . .	94
h2o.tolower . . . . .	95
h2o.totss . . . . .	95
h2o.tot_withinss . . . . .	96
h2o.toupper . . . . .	96
h2o.trim . . . . .	96
h2o.var . . . . .	97
h2o.varimp . . . . .	97
h2o.week . . . . .	98
h2o.withinss . . . . .	98

h2o.year . . . . .	99
H2OClusteringModel-class . . . . .	99
H2OConnection-class . . . . .	100
H2OFrame-class . . . . .	101
H2OFrame-Extract . . . . .	101
H2OModel-class . . . . .	102
H2OModelFuture-class . . . . .	103
H2OModelMetrics-class . . . . .	104
H2OObject-class . . . . .	104
H2ORawData-class . . . . .	105
H2OS4groupGeneric . . . . .	106
H2OW2V-class . . . . .	107
is.factor,H2OFrame-method . . . . .	107
median,H2OFrame-method . . . . .	108
ModelAccessors . . . . .	108
Node-class . . . . .	109
predict.H2OModel . . . . .	110
print.H2OTable . . . . .	111
quantile . . . . .	111
sapply,H2OFrame-method . . . . .	112
str.H2OFrame . . . . .	113
summary,H2OModel-method . . . . .	113
transform.H2OFrame . . . . .	114

**Index** **115**

---

h2o-package	<i>H2O R Interface</i>
-------------	------------------------

---

**Description**

This is a package for running H2O via its REST API from within R. To communicate with a H2O instance, the version of the R package must match the version of H2O. When connecting to a new H2O cluster, it is necessary to re-run the initializer.

**Details**

```

Package: h2o
Type: Package
Version: 3.0.0.10
Branch: rel-shannon
Date: Fri May 22 20:06:15 PDT 2015
License: Apache License (== 2.0)
Depends: R (>= 2.13.0), RCurl, rjson, statmod, tools, methods, utils

```

This package allows the user to run basic H2O commands using R commands. In order to use it, you must first have H2O running. To run H2O on your local machine, call `h2o.init` without any arguments, and H2O will be automatically launched at `localhost:54321`, where the IP is "127.0.0.1" and the port is 54321. If H2O is running on a cluster, you must provide the IP and port of the remote machine as arguments to the `h2o.init()` call.

H2O supports a number of standard statistical models, such as GLM, K-means, and Random Forest classification. For example, to run GLM, call `h2o.glm` with the H2O parsed data and parameters (response variable, error distribution, etc...) as arguments. (The operation will be done on the server associated with the data object where H2O is running, not within the R environment).

Note that no actual data is stored in the R workspace; and no actual work is carried out by R. R only saves the named objects, which uniquely identify the data set, model, etc on the server. When the user makes a request, R queries the server via the REST API, which returns a JSON file with the relevant information that R then displays in the console.

If you are using an older version of H2O, use the following porting guide to update your scripts:  
[Porting Scripts](#)

### Author(s)

Anqi Fu, Tom Kraljevic and Petr Maj, with contributions from the Oxdata team

Maintainer: Tom Kraljevic <tomk@oxdata.com>

### References

- [Oxdata Homepage](#)
- [H2O Documentation](#)
- [H2O on Github](#)

---

aaa

*Starting H2O For examples*

---

### Description

Starting H2O For examples

### Examples

```
h2o.init()
```

---

apply,H2OFrame-method *Apply on H2O Datasets*

---

## Description

Method for apply on [H2OFrame](#) objects.

## Usage

```
## S4 method for signature H2OFrame
apply(X, MARGIN, FUN, ...)
```

## Arguments

X	an <a href="#">H2OFrame</a> object on which apply will operate.
MARGIN	the vector on which the function will be applied over, either 1 for rows or 2 for columns.
FUN	the function to be applied.
...	optional arguments to FUN.

## Value

Produces a new [H2OFrame](#) of the output of the applied function. The output is stored in H2O so that it can be used in subsequent H2O processes.

## See Also

[apply](#) for the base generic

## Examples

```
localH2O = h2o.init()
irisPath = system.file("extdata", "iris.csv", package="h2o")
iris.hex = h2o.importFile(localH2O, path = irisPath, destination_frame = "iris.hex")
summary(apply(iris.hex, 1, sum))
```

---

`as.character,H2OFrame-method`*Convert H2O Data to Characters*

---

**Description**

Converts an H2O column into character columns.

**Usage**

```
## S4 method for signature H2OFrame
as.character(x)
```

**Arguments**

`x` a column from an [H2OFrame](#) data set. `localH2O <- h2o.init()` `iris.hex <- as.h2o(iris)`  
`iris.hex[,5] <- as.character(iris.hex[,5])`

---

`as.data.frame.H2OFrame`*Converts a Parsed H2O data into a Data Frame*

---

**Description**

Downloads the H2O data and then scans it in to an R data frame.

**Usage**

```
## S3 method for class H2OFrame
as.data.frame(x, ...)
```

**Arguments**

`x` An [H2OFrame](#) object.  
`...` Further arguments to be passed down from other methods.

**Examples**

```
localH2O <- h2o.init()
prosPath <- system.file("extdata", "prostate.csv", package="h2o")
prostate.hex <- h2o.uploadFile(localH2O, path = prosPath)
as.data.frame(prostate.hex)
```

---

as.environment,H2OFrame-method

*Convert H2O Data to an R Environment*

---

### Description

Converts an [H2OFrame](#) to an environment.

### Usage

```
## S4 method for signature H2OFrame
as.environment(x)
```

### Arguments

x                    an [H2OFrame](#) class object.

### Value

Returns an R environment object based on the [H2OFrame](#). localH2O <- h2o.init() prosPath <- system.file("extdata", "prostate.csv", package="h2o") prostate.hex <- h2o.uploadFile(localH2O, path = prosPath) names(as.environment) aa <- as.environment(prostate.hex) ls(aa)

---

as.factor,H2OFrame-method

*Convert H2O Data to Factors*

---

### Description

Convert a column into a factor column.

### Usage

```
## S4 method for signature H2OFrame
as.factor(x)
```

### Arguments

x                    a column from an [H2OFrame](#) data set.

### See Also

[is.factor](#).



**Examples**

```

localH2O <- h2o.init()
prosPath <- system.file("extdata", "prostate.csv", package="h2o")
prostate.hex <- h2o.uploadFile(localH2O, path = prosPath)
prostate.hex[,2] <- as.factor(prostate.hex[,2])
summary(prostate.hex)

```

---

as.h2o *R data.frame -> H2OFrame*

---

**Description**

Import a local R data frame to the H2O cloud.

**Usage**

```
as.h2o(object, conn = h2o.getConnection(), destination_frame = "")
```

**Arguments**

object            An R data frame.

conn              An [H2OConnection](#) object containing the IP address and port number of the H2O server.

destination\_frame    A string with the desired name for the H2O Frame.

---

as.matrix.h2o *Converts H2O Data to an R Matrix*

---

**Description**

Convert an [H2OFrame](#) object to a matrix, which allows subsequent data frame operations within the R environment.

**Usage**

```
## S3 method for class H2OFrame
as.matrix(x, ...)
```

**Arguments**

x                  An [H2OFrame](#) object

...                Additional arguments to be passed to or from

**Value**

Returns a matrix in the R environment.

**Note**

This call establishes the data set in the R environment and subsequent operations on the matrix take place within R, not H2O. When data are large, users may experience significant slowdown.

**See Also**

[as.matrix](#) for the base R implementation.

**Examples**

```
library(h2o)
localH2O <- h2o.init()
prosPath <- system.file("extdata", "prostate.csv", package="h2o")
prostate.hex <- h2o.uploadFile(localH2O, path = prosPath)
prostate.matrix <- as.matrix(prostate.hex)
summary(prostate.matrix)
head(prostate.matrix)
```

---

as.numeric,H2OFrame-method

*Convert H2O Data to Numeric*

---

**Description**

Converts an H2O column into a numeric value column.

**Usage**

```
## S4 method for signature H2OFrame
as.numeric(x)
```

**Arguments**

x a column from an [H2OFrame](#) data set. localH2O <- h2o.init() prosPath <- system.file("extdata", "prostate.csv", package="h2o") prostate.hex <- h2o.uploadFile(localH2O, path = prosPath) prostate.hex[,2] <- as.factor(prostate.hex[,2]) prostate.hex[,2] <- as.numeric(prostate.hex[,2])

---

ASTNode-class	<i>The ASTNode class.</i>
---------------	---------------------------

---

**Description**

This class represents a node in the abstract syntax tree. An ASTNode has a root. The root has children that either point to another ASTNode, or to a leaf node, which may be of type ASTNumeric or ASTFrame.

**Usage**

```
## S4 method for signature ASTNode
show(object)
```

**Arguments**

object            An ASTNode class object.

**Slots**

root    Object of type Node  
children    Object of type list

---

colnames<- ,H2OFrame,H2OFrame-method	<i>Returns Column Names for a Parsed H2O Data Object.</i>
--------------------------------------	---

---

**Description**

Returns column names for an [H2OFrame](#) object.

**Usage**

```
## S4 replacement method for signature H2OFrame,H2OFrame
colnames(x) <- value
```

```
## S4 replacement method for signature H2OFrame,character
colnames(x) <- value
```

```
## S4 method for signature H2OFrame
names(x)
```

```
## S4 replacement method for signature H2OFrame
names(x) <- value
```

```
## S4 method for signature H2OFrame
colnames(x)
```

```
## S4 method for signature H2OFrame
names(x)
```

### Arguments

x                    An [H2OFrame](#) object.

value                a character string to rename columns.

### See Also

[colnames](#) for the base R method.

### Examples

```
library(h2o)
localH2O <- h2o.init()
irisPath <- system.file("extdata", "iris.csv", package="h2o")
iris.hex <- h2o.uploadFile(localH2O, path = irisPath)
summary(iris.hex)
colnames(iris.hex)
```

---

cut.H2OFrame	<i>Cut H2O Numeric Data to Factor</i>
--------------	---------------------------------------

---

### Description

Divides the range of the H2O data into intervals and codes the values according to which interval they fall in. The leftmost interval corresponds to the level one, the next is level two, etc.

### Usage

```
## S3 method for class H2OFrame
cut(x, breaks, labels = NULL, include.lowest = FALSE,
    right = TRUE, dig.lab = 3, ...)
```

### Arguments

x                    An [H2OFrame](#) object with numeric columns.

breaks              A numeric vector of two or more unique cut points.

labels               Labels for the levels of the resulting category. By default, labels are constructed using "(a,b]" interval notation.

include.lowest      Logical, indicating if an 'x[i]' equal to the lowest (or highest, for right = FALSE) 'breaks' value should be included

right	/codeLogical, indicating if the intervals should be closed on the right (opened on the left) or vice versa.
dig.lab	Integer which is used when labels are not given, determines the number of digits used in formatting the beak numbers.
...	Further arguments passed to or from other methods.

**Value**

Returns an [H2OFrame](#) object containing the factored data with intervals as levels.

**Examples**

```
library(h2o)
localH2O <- h2o.init()
irisPath <- system.file("extdata", "iris_wheader.csv", package="h2o")
iris.hex <- h2o.uploadFile(localH2O, path = irisPath, destination_frame = "iris.hex")
summary(iris.hex)

# Cut sepal length column into intervals determined by min/max/quantiles
sepal_len.cut = cut(iris.hex$sepal_len, c(4.2, 4.8, 5.8, 6, 8))
head(sepal_len.cut)
summary(sepal_len.cut)
```

---

h2o.aic	<i>Retrieve the AIC.</i>
---------	--------------------------

---

**Description**

Retrieve the AIC.

**Usage**

```
h2o.aic(object, valid = FALSE, ...)
```

**Arguments**

object	An <a href="#">H2OModel</a> or <a href="#">H2OModelMetrics</a> .
valid	Retrieve the validation AIC
...	extra arguments to be passed if 'object' is of type <a href="#">H2OModel</a> (e.g. train=TRUE)

---

h2o.anomaly

*Anomaly Detection via H2O Deep Learning Model*


---

**Description**

Detect anomalies in a H2O dataset using a H2O deep learning model with auto-encoding.

**Usage**

```
h2o.anomaly(object, data)
```

**Arguments**

object	An <a href="#">H2OAutoEncoderModel</a> object that represents the model to be used for anomaly detection.
data	An <a href="#">H2OFrame</a> object.

**Value**

Returns an [H2OFrame](#) object containing the reconstruction MSE.

**See Also**

[h2o.deeplearning](#) for making an [H2OAutoEncoderModel](#).

**Examples**

```
library(h2o)
localH2O = h2o.init()
prosPath = system.file("extdata", "prostate.csv", package = "h2o")
prostate.hex = h2o.importFile(localH2O, path = prosPath)
prostate.dl = h2o.deeplearning(x = 3:9, training_frame = prostate.hex, autoencoder = TRUE,
                             hidden = c(10, 10), epochs = 5)
prostate.anon = h2o.anomaly(prostate.dl, prostate.hex)
head(prostate.anon)
```

---

h2o.anyFactor

*Check H2OFrame columns for factors*


---

**Description**

Determines if any column of an [H2OFrame](#) object contains categorical data.

**Usage**

```
h2o.anyFactor(x)
```

**Arguments**

x                    An [H2OFrame](#) object.

**Value**

Returns a logical value indicating whether any of the columns in x are factors.

**Examples**

```
library(h2o)
localH2O <- h2o.init()
irisPath <- system.file("extdata", "iris_wheader.csv", package="h2o")
iris.hex <- h2o.importFile(localH2O, path = irisPath)
h2o.anyFactor(iris.hex)
```

---

h2o.assign	<i>Rename an H2O object.</i>
------------	------------------------------

---

**Description**

Makes a copy of the data frame and gives it the desired the key.

**Usage**

```
h2o.assign(data, key, deepCopy = FALSE)
```

**Arguments**

data                An [H2OFrame](#) object  
key                  The hex key to be associated with the H2O parsed data object  
deepCopy            Should it do a deepCopy of the frame. Default is FALSE.

---

h2o.auc	<i>Retrieve the AUC</i>
---------	-------------------------

---

**Description**

Retrieves the AUC value from an [H2OBinomialMetrics](#).

**Usage**

```
h2o.auc(object, valid = FALSE, ...)
```

**Arguments**

object	An <a href="#">H2OBinomialMetrics</a> object.
valid	Retrieve the validation AUC
...	extra arguments to be passed if 'object' is of type <a href="#">H2OModel</a> (e.g. train=TRUE)

**See Also**

[h2o.giniCoef](#) for the Gini coefficient, [h2o.mse](#) for MSE, and [h2o.metric](#) for the various threshold metrics. See [h2o.performance](#) for creating [H2OModelMetrics](#) objects.

**Examples**

```
library(h2o)
h2o.init()

prosPath <- system.file("extdata", "prostate.csv", package="h2o")
hex <- h2o.uploadFile(prosPath)

hex[,2] <- as.factor(hex[,2])
model <- h2o.gbm(x = 3:9, y = 2, training_frame = hex, distribution = "bernoulli")
perf <- h2o.performance(model, hex)
h2o.auc(perf)
```

---

h2o.betweenss	<i>Get the between cluster sum of squares.</i>
---------------	--

---

**Description**

Get the between cluster sum of squares.

**Usage**

```
h2o.betweenss(object, valid = FALSE, ...)
```

**Arguments**

object	An <a href="#">H2OClusteringModel</a> object.
valid	Retrieve the validation metric.
...	further arguments to be passed on (currently unimplemented)



---

h2o.cbind	<i>Combine H2O Datasets by Columns</i>
-----------	--

---

**Description**

Takes a sequence of H2O data sets and combines them by column

**Usage**

```
h2o.cbind(...)
```

**Arguments**

... A sequence of [H2OFrame](#) arguments. All datasets must exist on the same H2O instance (IP and port) and contain the same number of rows.

**Value**

An [H2OFrame](#) object containing the combined ... arguments column-wise.

**See Also**

[cbind](#) for the base R method.

**Examples**

```
library(h2o)
localH2O <- h2o.init()
prosPath <- system.file("extdata", "prostate.csv", package="h2o")
prostate.hex <- h2o.uploadFile(localH2O, path = prosPath)
prostate.cbind <- h2o.cbind(prostate.hex, prostate.hex)
head(prostate.cbind)
```

---

h2o.centers	<i>Retrieve the Model Centers</i>
-------------	-----------------------------------

---

**Description**

Retrieve the Model Centers

**Usage**

```
h2o.centers(object, ...)
```

**Arguments**

object An [H2OClusteringModel](#) object.  
... further arguments to be passed on (currently unimplemented)

---

h2o.centersSTD	<i>Retrieve the Model Centers STD</i>
----------------	---------------------------------------

---

**Description**

Retrieve the Model Centers STD

**Usage**

```
h2o.centersSTD(object, ...)
```

**Arguments**

object	An <a href="#">H2OClusteringModel</a> object.
...	further arguments to be passed on (currently unimplemented)

---

h2o.clearLog	<i>Delete All H2O R Logs</i>
--------------	------------------------------

---

**Description**

Clear all H2O R command and error response logs from the local disk. Used primarily for debugging purposes.

**Usage**

```
h2o.clearLog()
```

**See Also**

[h2o.startLogging](#), [h2o.stopLogging](#), [h2o.openLog](#)

**Examples**

```
library(h2o)
localH2O = h2o.init()
h2o.startLogging()
ausPath = system.file("extdata", "australia.csv", package="h2o")
australia.hex = h2o.importFile(localH2O, path = ausPath)
h2o.stopLogging()
h2o.clearLog()
```

---

h2o.clusterInfo	<i>Print H2O cluster info</i>
-----------------	-------------------------------

---

**Description**

Print H2O cluster info

**Usage**

```
h2o.clusterInfo(conn = h2o.getConnection())
```

**Arguments**

conn	H2O connection object
------	-----------------------

---

h2o.clusterIsUp	<i>Determine if an H2O cluster is up or not</i>
-----------------	---

---

**Description**

Determine if an H2O cluster is up or not

**Usage**

```
h2o.clusterIsUp(conn = h2o.getConnection())
```

**Arguments**

conn	H2O connection object
------	-----------------------

**Value**

TRUE if the cluster is up; FALSE otherwise

---

h2o.clusterStatus      *Return the status of the cluster*

---

**Description**

Retrieve information on the status of the cluster running H2O.

**Usage**

```
h2o.clusterStatus(conn = h2o.getConnection())
```

**Arguments**

conn                    the [H2OConnection](#) object containing the IP address and port of the server running H2O.

**See Also**

[H2OConnection](#), [h2o.init](#)

**Examples**

```
localH2O <- h2o.init()
h2o.clusterStatus(localH2O)
```

---

h2o.cluster\_sizes      *Retrieve the cluster sizes*

---

**Description**

Retrieve the cluster sizes

**Usage**

```
h2o.cluster_sizes(object, valid = FALSE, ...)
```

**Arguments**

object                  An [H2OClusteringModel](#) object.  
valid                    Retrieve the validation metric.  
...                      further arguments to be passed on (currently unimplemented)

---

h2o.coef	<i>Retrieve the model coefficients</i>
----------	--

---

**Description**

Retrieve the model coefficients

**Usage**

```
h2o.coef(object)
```

**Arguments**

object            an [H2OModel](#) object.

---

h2o.coef_norm	<i>Retrieve the normalized coefficients</i>
---------------	---

---

**Description**

Retrieve the normalized coefficients

**Usage**

```
h2o.coef_norm(object)
```

**Arguments**

object            an [H2OModel](#) object.

---

h2o.confusionMatrix	<i>Access H2O Confusion Matrices</i>
---------------------	--------------------------------------

---

**Description**

Retrieve either a single or many confusion matrices from H2O objects.

**Usage**

```
h2o.confusionMatrix(object, ...)

## S4 method for signature H2OModel
h2o.confusionMatrix(object, newdata, valid = FALSE, ...)

## S4 method for signature H2OModelMetrics
h2o.confusionMatrix(object, thresholds = NULL,
  metrics = NULL)
```

**Arguments**

object	Either an <a href="#">H2OModel</a> object or an <a href="#">H2OModelMetrics</a> object.
...	Extra arguments for extracting train or valid confusion matrices.
newdata	An <a href="#">H2OFrame</a> object that can be scored on. Requires a valid response column.
valid	Retrieve the validation metric.
thresholds	(Optional) A value or a list of valid values between 0.0 and 1.0. This value is only used in the case of <a href="#">H2OBinomialMetrics</a> objects.
metrics	(Optional) A metric or a list of valid metrics ("min_per_class_accuracy", "absolute_MCC", "tnr", "fnr", "fpr", "tpr", "precision", "accuracy", "f0point5", "f2", "f1"). This value is only used in the case of <a href="#">H2OBinomialMetrics</a> objects.

**Details**

The [H2OModelMetrics](#) version of this function will only take [H2OBinomialMetrics](#) or [H2OMultinomialMetrics](#) objects. If no threshold is specified, all possible thresholds are selected.

**Value**

Calling this function on [H2OModel](#) objects returns a confusion matrix corresponding to the [predict](#) function. If used on an [H2OBinomialMetrics](#) object, returns a list of matrices corresponding to the number of thresholds specified.

**See Also**

[predict](#) for generating prediction frames, [h2o.performance](#) for creating [H2OModelMetrics](#).

**Examples**

```
library(h2o)
h2o.init()
prosPath <- system.file("extdata", "prostate.csv", package="h2o")
hex <- h2o.uploadFile(prosPath)
hex[,2] <- as.factor(hex[,2])
model <- h2o.gbm(x = 3:9, y = 2, training_frame = hex, distribution = "bernoulli")
h2o.confusionMatrix(model, hex)
# Generating a ModelMetrics object
perf <- h2o.performance(model, hex)
h2o.confusionMatrix(perf)
```

---

h2o.createFrame

*Data Frame Creation in H2O*


---

**Description**

Creates a data frame in H2O with real-valued, categorical, integer, and binary columns specified by the user.

**Usage**

```
h2o.createFrame(conn = h2o.getConnection(), key = "", rows = 10000,
  cols = 10, randomize = TRUE, value = 0, real_range = 100,
  categorical_fraction = 0.2, factors = 100, integer_fraction = 0.2,
  integer_range = 100, binary_fraction = 0.1, binary_ones_fraction = 0.02,
  missing_fraction = 0.01, response_factors = 2, has_response = FALSE,
  seed)
```

**Arguments**

conn	A <a href="#">H2OConnection</a> object.
key	A string indicating the destination key. If empty, this will be auto-generated by H2O.
rows	The number of rows of data to generate.
cols	The number of columns of data to generate. Excludes the response column if <code>has_response = TRUE</code> .
randomize	A logical value indicating whether data values should be randomly generated. This must be <code>TRUE</code> if either <code>categorical_fraction</code> or <code>integer_fraction</code> is non-zero.
value	If <code>randomize = FALSE</code> , then all real-valued entries will be set to this value.
real_range	The range of randomly generated real values.
categorical_fraction	The fraction of total columns that are categorical.
factors	The number of (unique) factor levels in each categorical column.
integer_fraction	The fraction of total columns that are integer-valued.
integer_range	The range of randomly generated integer values.
binary_fraction	The fraction of total columns that are binary-valued.
binary_ones_fraction	The fraction of values in a binary column that are set to 1.
missing_fraction	The fraction of total entries in the data frame that are set to NA.
response_factors	If <code>has_response = TRUE</code> , then this is the number of factor levels in the response column.
has_response	A logical value indicating whether an additional response column should be prepended to the final H2O data frame. If set to <code>TRUE</code> , the total number of columns will be <code>cols+1</code> .
seed	A seed used to generate random values when <code>randomize = TRUE</code> .

**Value**

Returns a [H2OFrame](#) object.

## Examples

```
library(h2o)
localH2O <- h2o.init()
hex <- h2o.createFrame(localH2O, rows = 1000, cols = 100, categorical_fraction = 0.1,
                      factors = 5, integer_fraction = 0.5, integer_range = 1,
                      has_response = TRUE)

head(hex)
summary(hex)

hex2 <- h2o.createFrame(localH2O, rows = 100, cols = 10, randomize = FALSE, value = 5,
                      categorical_fraction = 0, integer_fraction = 0)

summary(hex2)
```

---

h2o.day

*Convert Milliseconds to Day of Month in H2O Datasets*

---

## Description

Converts the entries of a [H2OFrame](#) object from milliseconds to days of the month (on a 1 to 31 scale).

## Usage

```
h2o.day(x)

day(x)

## S3 method for class H2OFrame
day(x)
```

## Arguments

x                    An [H2OFrame](#) object.

## Value

A [H2OFrame](#) object containing the entries of x converted to days of the month.

## See Also

[h2o.month](#)



---

h2o.dayOfWeek	<i>Convert Milliseconds to Day of Week in H2O Datasets</i>
---------------	--

---

**Description**

Converts the entries of a [H2OFrame](#) object from milliseconds to days of the week (on a 0 to 6 scale).

**Usage**

```
h2o.dayOfWeek(x)
```

```
dayOfWeek(x)
```

```
## S3 method for class H2OFrame
dayOfWeek(x)
```

**Arguments**

x                    An [H2OFrame](#) object.

**Value**

A [H2OFrame](#) object containing the entries of x converted to days of the week.

**See Also**

[h2o.day](#), [h2o.month](#)

---

h2o.ddply	<i>Split H2O Dataset, Apply Function, and Return Results</i>
-----------	--

---

**Description**

For each subset of an H2O data set, apply a user-specified function, then combine the results.

**Usage**

```
h2o.ddply(.data, .variables, .fun = NULL, ..., .progress = "none")
```

**Arguments**

.data	An <a href="#">H2OFrame</a> object to be processed.
.variables	Variables to split .data by, either the indices or names of a set of columns.
.fun	Function to apply to each subset grouping.
.progress	Name of the progress bar to use. #TODO: (Currently unimplemented)
...	Additional arguments passed on to .fun. #TODO: (Currently unimplemented)

**Value**

Returns a [H2OFrame](#) object containing the results from the split/apply operation, arranged

**See Also**

[ddply](#) for the plyr library implementation.

**Examples**

```
library(h2o)
localH2O <- h2o.init()

# Import iris dataset to H2O
irisPath <- system.file("extdata", "iris_wheader.csv", package = "h2o")
iris.hex <- h2o.uploadFile(localH2O, path = irisPath, destination_frame = "iris.hex")
# Add function taking mean of sepal_len column
fun = function(df) { sum(df[,1], na.rm = T)/nrow(df) }
# Apply function to groups by class of flower
# uses h2os ddply, since iris.hex is an H2OFrame object
res = h2o.ddply(iris.hex, "class", fun)
head(res)
```

---

h2o.deepfeatures

*Feature Generation via H2O Deep Learning Model*


---

**Description**

Extract the non-linear feature from an H2O data set using an H2O deep learning model.

**Usage**

```
h2o.deepfeatures(object, data, layer = 1)
```

**Arguments**

object	An <a href="#">H2OModel</a> object that represents the deep learning model to be used for feature extraction.
data	An <a href="#">H2OFrame</a> object.
layer	Index of the hidden layer to extract.

**Value**

Returns an [H2OFrame](#) object with as many features as the number of units in the hidden layer of the specified index.

**See Also**

`link{h2o.deepLearning}` for making deep learning models.

**Examples**

```

library(h2o)
localH2O = h2o.init()
prosPath = system.file("extdata", "prostate.csv", package = "h2o")
prostate.hex = h2o.importFile(localH2O, path = prosPath)
prostate.dl = h2o.deeplearning(x = 3:9, y = 2, training_frame = prostate.hex,
                              hidden = c(100, 200), epochs = 5)
prostate.deepfeatures_layer1 = h2o.deepfeatures(prostate.dl, prostate.hex, layer = 1)
prostate.deepfeatures_layer2 = h2o.deepfeatures(prostate.dl, prostate.hex, layer = 2)
head(prostate.deepfeatures_layer1)
head(prostate.deepfeatures_layer2)

```

h2o.deeplearning

*Build a Deep Learning Neural Network***Description**

Performs Deep Learning neural networks on an [H2OFrame](#)

**Usage**

```

h2o.deeplearning(x, y, training_frame, model_id = "",
  overwrite_with_best_model, n_folds = 0, validation_frame, checkpoint,
  autoencoder = FALSE, use_all_factor_levels = TRUE,
  activation = c("Rectifier", "Tanh", "TanhWithDropout",
    "RectifierWithDropout", "Maxout", "MaxoutWithDropout"), hidden = c(200,
    200), epochs = 10, train_samples_per_iteration = -2, seed,
  adaptive_rate = TRUE, rho = 0.99, epsilon = 1e-08, rate = 0.005,
  rate_annealing = 1e-06, rate_decay = 1, momentum_start = 0,
  momentum_ramp = 1e+06, momentum_stable = 0,
  nesterov_accelerated_gradient = TRUE, input_dropout_ratio = 0,
  hidden_dropout_ratios, l1 = 0, l2 = 0, max_w2 = Inf,
  initial_weight_distribution = c("UniformAdaptive", "Uniform", "Normal"),
  initial_weight_scale = 1, loss = c("Automatic", "CrossEntropy",
    "MeanSquare", "Absolute", "Huber"), score_interval = 5,
  score_training_samples, score_validation_samples, score_duty_cycle,
  classification_stop, regression_stop, quiet_mode, max_confusion_matrix_size,
  max_hit_ratio_k, balance_classes = FALSE, class_sampling_factors,
  max_after_balance_size, score_validation_sampling, diagnostics,
  variable_importances, fast_mode, ignore_const_cols, force_load_balance,
  replicate_training_data, single_node_mode, shuffle_training_data, sparse,
  col_major, average_activation, sparsity_beta, max_categorical_features,
  reproducible = FALSE, export_weights_and_biases = FALSE, ...)

```

**Arguments**

x                    A vector containing the character names of the predictors in the model.

y	The name of the response variable in the model.
training_frame	An <a href="#">H2OFrame</a> object containing the variables in the model.
model_id	(Optional) The unique id assigned to the resulting model. If none is given, an id will automatically be generated.
overwrite_with_best_model	Logical. If TRUE, overwrite the final model with the best model found during training. Defaults to TRUE.
n_folds	(Optional) Number of folds for cross-validation. If n_folds >= 2, then validation must remain empty.
validation_frame	(Optional) An <a href="#">H2OFrame</a> object indicating the validation dataset used to construct the confusion matrix. If left blank, this defaults to the training data when n_folds = 0
checkpoint	"Model checkpoint (either key or H2ODeepLearningModel) to resume training with."
autoencoder	Enable auto-encoder for model building.
use_all_factor_levels	Logical. Use all factor levels of categorical variance. Otherwise the first factor level is omitted (without loss of accuracy). Useful for variable importances and auto-enabled for autoencoder.
activation	A string indicating the activation function to use. Must be either "Tanh", "TanhWithDropout", "Rectifier", "RectifierWithDropout", "Maxout", or "MaxoutWithDropout"
hidden	Hidden layer sizes (e.g. c(100,100))
epochs	How many times the dataset should be iterated (streamed), can be fractional
train_samples_per_iteration	Number of training samples (globally) per MapReduce iteration. Special values are: <b>0</b> one epoch; <b>-1</b> all available data (e.g., replicated training data); or <b>-2</b> auto-tuning (default)
seed	Seed for random numbers (affects sampling) - Note: only reproducible when running single threaded
adaptive_rate	Logical. Adaptive learning rate (ADAELTA)
rho	Adaptive learning rate time decay factor (similarity to prior updates)
epsilon	Adaptive learning rate parameter, similar to learn rate annealing during initial training phase. Typical values are between 1.0e-10 and 1.0e-4
rate	Learning rate (higher => less stable, lower => slower convergence)
rate_annealing	Learning rate annealing: $(rate)/(1 + rate_{annealing} * samples)$
rate_decay	Learning rate decay factor between layers (N-th layer: $rate * \alpha^{(N - 1)}$ )
momentum_start	Initial momentum at the beginning of training (try 0.5)
momentum_ramp	Number of training samples for which momentum increases
momentum_stable	Final momentum after the ramp is over (try 0.99)

nesterov_accelerated_gradient	Logical. Use Nesterov accelerated gradient (recommended)
input_dropout_ratio	A fraction of the features for each training row to be omitted from training in order to improve generalization (dimension sampling).
hidden_dropout_ratios	Input layer dropout ration (can improve generalization) specify one value per hidden layer, defaults to 0.5
l1	L1 regularization (can add stability and improve generalization, cause many weights to become 0)
l2	L2 regularization (can add stability and improve generalization, causes many weights to be small)
max_w2	Constraint for squared sum of incoming weights per unit (e.g. Rectifier)
initial_weight_distribution	Can be "Uniform", "UniformAdaptive", or "Normal"
initial_weight_scale	Unifrom: -value ... value, Normal: stddev
loss	Loss function: Automatic, CrossEntropy (for classification only), MeanSquare, Absolute (experimental) or Huber (experimental)
score_interval	Shortest time interval (in secs) between model scoring
score_training_samples	Number of training set samples for scoring (0 for all)
score_validation_samples	Number of validation set samples for scoring (0 for all)
score_duty_cycle	Maximum duty cycle fraction for scoring (lower: more training, higher: more scoring)
classification_stop	Stopping criterion for classification error fraction on training data (-1 to disable)
regression_stop	Stopping criterion for regression error (MSE) on training data (-1 to disable)
quiet_mode	Enable quiet mode for less output to standard output
max_confusion_matrix_size	Max. size (number of classes) for confusion matrices to be shown
max_hit_ratio_k	Max number (top K) of predictions to use for hit ration computation(for multi-class only, 0 to disable)
balance_classes	Balance training data class counts via over/under-sampling (for imbalanced data)
class_sampling_factors	Desired over/under-sampling ratios per class (in lexicographic order). If not specified, sampling factors will be automatically computed to obtain class balance during training. Requires balance_classes.

<code>max_after_balance_size</code>	Maximum relative size of the training data after balancing class counts (can be less than 1.0)
<code>score_validation_sampling</code>	Method used to sample validation dataset for scoring
<code>diagnostics</code>	Enable diagnostics for hidden layers
<code>variable_importances</code>	Compute variable importances for input features (Gedeon method) - can be slow for large networks)
<code>fast_mode</code>	Enable fast mode (minor approximations in back-propagation)
<code>ignore_const_cols</code>	Ignore constant columns (no information can be gained anyway)
<code>force_load_balance</code>	Force extra load balancing to increase training speed for small datasets (to keep all cores busy)
<code>replicate_training_data</code>	Replicate the entire training dataset onto every node for faster training
<code>single_node_mode</code>	Run on a single node for fine-tuning of model parameters
<code>shuffle_training_data</code>	Enable shuffling of training data (recommended if training data is replicated and <code>train_samples_per_iteration</code> is close to <code>numRows * numNodes</code> )
<code>sparse</code>	Sparse data handling (Experimental)
<code>col_major</code>	Use a column major weight matrix for input layer. Can speed up forward propagation, but might slow down backpropagation (Experimental)
<code>average_activation</code>	Average activation for sparse auto-encoder (Experimental)
<code>sparsity_beta</code>	Sparsity regularization (Experimental)
<code>max_categorical_features</code>	Max. number of categorical features, enforced via hashing (Experimental)
<code>reproducible</code>	Force reproducibility on small data (will be slow - only uses 1 thread)
<code>export_weights_and_biases</code>	Whether to export Neural Network weights and biases to H2O Frames"
<code>...</code>	extra parameters to pass onto functions (not implemented)

**See Also**

[predict.H2OModel](#) for prediction.

**Examples**

```
library(h2o)
localH2O <- h2o.init()

irisPath <- system.file("extdata", "iris.csv", package = "h2o")
iris.hex <- h2o.uploadFile(localH2O, path = irisPath)
iris.dl <- h2o.deeplearning(x = 1:4, y = 5, training_frame = iris.hex)
```

---

h2o.dim	Returns the Dimensions of a Parsed H2O Data Object.
---------	---

---

### Description

Returns the number of rows and columns for an [H2OFrame](#) object.

### Usage

```
## S4 method for signature H2OFrame
dim(x)
```

### Arguments

x                    An [H2OFrame](#) object.

### See Also

[dim](#) for the base R method.

### Examples

```
localH2O <- h2o.init()
irisPath <- system.file("extdata", "iris.csv", package="h2o")
iris.hex <- h2o.uploadFile(localH2O, path = irisPath)
dim(iris.hex)
```

---

h2o.downloadAllLogs	Download H2O Log Files to Disk
---------------------	--------------------------------

---

### Description

`h2o.downloadAllLogs` downloads all H2O log files to local disk. Generally used for debugging purposes.

### Usage

```
h2o.downloadAllLogs(conn = h2o.getConnection(), dirname = ".",
  filename = NULL)
```

### Arguments

conn                    An [H2OConnection](#) object pointing to a running H2O cluster.

dirname                (Optional) A character string indicating the directory that the log file should be saved in.

filename               (Optional) A character string indicating the name that the log file should be saved to.

**See Also**

[H2OConnection](#)

---

h2o.downloadCSV	<i>Download H2O Data to Disk</i>
-----------------	----------------------------------

---

**Description**

Download an H2O data set to a CSV file on the local disk

**Usage**

```
h2o.downloadCSV(data, filename)
```

**Arguments**

data	an <a href="#">H2OFrame</a> object to be downloaded.
filename	A string indicating the name that the CSV file should be should be saved to.

**Warning**

Files located on the H2O server may be very large! Make sure you have enough hard drive psace to accomoadet the entire file.

**Examples**

```
library(h2o)
localH2O <- h2o.init()
irisPath <- system.file("extdata", "iris_wheader.csv", package = "h2o")
iris.hex <- h2o.uploadFile(localH2O, path = irisPath)

myFile <- paste(getwd(), "my_iris_file.csv", sep = .Platform$file.sep)
h2o.downloadCSV(iris.hex, myFile)
file.info(myFile)
file.remove(myFile)
```



---

h2o.download_pojo	<i>Download the Scoring POJO of An H2O Model</i>
-------------------	--

---

**Description**

Download the Scoring POJO of An H2O Model

**Usage**

```
h2o.download_pojo(model, path = "", conn = h2o.getConnection())
```

**Arguments**

model	An H2OModel
path	The path to the directory to store the POJO (no trailing slash). If "", then print to console. The file name will be a compilable java file name.
conn	An H2OClient object.

**Value**

If path is "", then pretty print the POJO to the console. Otherwise save it to the specified directory.

**Examples**

```
library(h2o)
h <- h2o.init(nthreads=-1)
fr <- as.h2o(iris)
my_model <- h2o.gbm(x=1:4, y=5, training_frame=fr)

h2o.download_pojo(my_model) # print the model to screen
# h2o.download_pojo(my_model, getwd()) # save to the current working directory, NOT RUN
```

---

h2o.exportFile	<i>Export an H2O Data Frame to a File</i>
----------------	---

---

**Description**

Exports an [H2OFrame](#) (which can be either VA or FV) to a file. This file may be on the H2O instace's local filesystem, or to HDFS (preface the path with hdfs://) or to S3N (preface the path with s3n://).

**Usage**

```
h2o.exportFile(data, path, force = FALSE)
```

**Arguments**

data	An <a href="#">H2OFrame</a> data frame.
path	The path to write the file to. Must include the directory and filename. May be prefaced with hdfs:// or s3n://. Each row of data appears as line of the file.
force	logical, indicates how to deal with files that already exist.

**Details**

In the case of existing files `force = TRUE` will overwrite the file. Otherwise, the operation will fail.

**Examples**

```
## Not run:
library(h2o)
localH2O <- h2o.init()
irisPath <- system.file("extdata", "iris.csv", package = "h2o")
iris.hex <- h2o.uploadFile(localH2O, path = irisPath)

# These arent real paths
# h2o.exportFile(iris.hex, path = "/path/on/h2o/server/filesystem/iris.csv")
# h2o.exportFile(iris.hex, path = "hdfs://path/in/hdfs/iris.csv")
# h2o.exportFile(iris.hex, path = "s3n://path/in/s3/iris.csv")

## End(Not run)
```

---

h2o.exportHDFS

*Export a Model to HDFS*


---

**Description**

Exports an [H2OModel](#) to HDFS.

**Usage**

```
h2o.exportHDFS(object, path, force = FALSE)
```

**Arguments**

object	an <a href="#">H2OModel</a> class object.
path	The path to write the model to. Must include the directory and filename.
force	logical, indicates how to deal with files that already exist.

---

h2o.filterNACols	<i>Filter NA Coluns</i>
------------------	-------------------------

---

**Description**

Filter NA Coluns

**Usage**

```
h2o.filterNACols(data, frac = 0.2)
```

**Arguments**

data	A dataset to filter on.
frac	The threshold of NAs to allow per column (columns $\geq$ this threshold are filtered)

---

h2o.gbm	<i>Gradient Boosted Machines</i>
---------	----------------------------------

---

**Description**

Builds gradient boosted classification trees, and gradient boosted regression trees on a parsed data set.

**Usage**

```
h2o.gbm(x, y, training_frame, model_id, distribution = c("AUTO", "gaussian",
  "bernoulli", "multinomial"), ntrees = 50, max_depth = 5, min_rows = 10,
  learn_rate = 0.1, nbins = 20, validation_frame = NULL,
  balance_classes = FALSE, max_after_balance_size = 1, seed, nfolds,
  score_each_iteration, ...)
```

**Arguments**

x	A vector containing the names or indices of the predictor variables to use in building the GBM model.
y	The name or index of the response variable. If the data does not contain a header, this is the column index number starting at 0, and increasing from left to right. (The response must be either an integer or a categorical variable).
training_frame	An <a href="#">H2OFrame</a> object containing the variables in the model.
model_id	(Optional) The unique id assigned to the resulting model. If none is given, an id will automatically be generated.

distribution	A character string. The loss function to be implemented. Must be "AUTO", "bernoulli", "multinomial", or "gaussian"
ntrees	A nonnegative integer that determines the number of trees to grow.
max_depth	Maximum depth to grow the tree.
min_rows	Minimum number of rows to assign to terminal nodes.
learn_rate	An integer from 0.0 to 1.0
nbins	Number of bins to use in building histogram.
validation_frame	An <a href="#">H2OFrame</a> object indicating the validation dataset used to construct the confusion matrix. If left blank, this defaults to the training data when <code>nfolds = 0</code>
balance_classes	logical, indicates whether or not to balance training data class counts via over/under-sampling (for imbalanced data)
max_after_balance_size	Maximum relative size of the training data after balancing class counts (can be less than 1.0)
seed	Seed for random numbers (affects sampling) - Note: only reproducible when running single threaded
nfolds	(Optional) Number of folds for cross-validation. If <code>nfolds &gt;= 2</code> , then validation must remain empty. <b>**Currently not supported**</b>
score_each_iteration	Attempts to score each tree.
...	extra arguments to pass on (currently no implemented)

### Details

The default distribution function will guess the model type based on the response column type. Properly the response column must be an numeric for "gaussian" or an enum for "bernoulli" or "multinomial".

### See Also

[predict.H2OModel](#) for prediction.

### Examples

```
library(h2o)
localH2O = h2o.init()

# Run regression GBM on australia.hex data
ausPath <- system.file("extdata", "australia.csv", package="h2o")
australia.hex <- h2o.uploadFile(localH2O, path = ausPath)
independent <- c("premax", "salmax", "minairtemp", "maxairtemp", "maxsst",
               "maxsoilmoist", "Max_czcs")
dependent <- "runoffnew"
h2o.gbm(y = dependent, x = independent, training_frame = australia.hex,
        ntrees = 3, max_depth = 3, min_rows = 2)
```

---

h2o.getConnection	<i>Retrieve an H2O Connection</i>
-------------------	-----------------------------------

---

**Description**

Attempt to recover an h2o connection.

**Usage**

```
h2o.getConnection()
```

**Value**

Returns an [H2OConnection](#) object.

---

h2o.getFrame	<i>Get an R Reference to an H2O Dataset</i>
--------------	---

---

**Description**

Get the reference to a frame with the given frame\_id in the H2O instance.

**Usage**

```
h2o.getFrame(frame_id, conn = h2o.getConnection(), linkToGC = FALSE)
```

**Arguments**

frame_id	A string indicating the unique frame of the dataset to retrieve.
conn	<a href="#">H2OConnection</a> object containing the IP address and port of the server running H2O.
linkToGC	a logical value indicating whether to remove the underlying frame from the H2O cluster when the R proxy object is garbage collected.

---

h2o.getGLMModel	<i>Resolve a GLM H2O Futures Model</i>
-----------------	--

---

**Description**

Turns an [H2OModelFuture](#) into a model of the correct type.

**Usage**

```
h2o.getGLMModel(keys, conn)
```

**Arguments**

keys	an <a href="#">H2OModelFuture</a> or correct job key.
conn	a corresponding <a href="#">H2OConnection</a> class object.

**Value**

Returns the correct [H2OModel](#) for the created model.

---

h2o.getModel	<i>Get an R reference to an H2O model</i>
--------------	---

---

**Description**

Returns a reference to an existing model in the H2O instance.

**Usage**

```
h2o.getModel(model_id, conn = h2o.getConnection(), linkToGC = FALSE)
```

**Arguments**

model_id	A string indicating the unique model_id of the model to retrieve.
conn	<a href="#">H2OConnection</a> object containing the IP address and port of the server running H2O.
linkToGC	a logical value indicating whether to remove the underlying model from the H2O cluster when the R proxy object is garbage collected.

**Value**

Returns an object that is a subclass of [H2OModel](#).

**Examples**

```
library(h2o)
localH2O <- h2o.init()

iris.hex <- as.h2o(iris, localH2O, "iris.hex")
model_id <- h2o.gbm(x = 1:4, y = 5, training_frame = iris.hex)@model_id
model.retrieved <- h2o.getModel(model_id, localH2O)
```

---

h2o.getTimezone	<i>Get the Time Zone on the H2O Cloud</i>
-----------------	---

---

**Description**

Get the Time Zone on the H2O Cloud

**Usage**

```
h2o.getTimezone(conn = h2o.getConnection())
```

**Arguments**

conn	An H2OConnection object.
------	--------------------------

---

h2o.giniCoef	<i>Retrieve the GINI Coefficient</i>
--------------	--------------------------------------

---

**Description**

Retrieves the GINI coefficient from an [H2OBinomialMetrics](#).

**Usage**

```
h2o.giniCoef(object, valid = FALSE, ...)
```

**Arguments**

object	an <a href="#">H2OBinomialMetrics</a> object.
valid	TRUE to extract the metric from validation set metrics; otherwise, training is assumed
...	extra arguments to be passed if 'object' is of type <a href="#">H2OModel</a> (e.g. train=TRUE)

**See Also**

[h2o.auc](#) for AUC, [h2o.giniCoef](#) for the GINI coefficient, and [h2o.metric](#) for the various. See [h2o.performance](#) for creating H2OModelMetrics objects. threshold metrics.

**Examples**

```

library(h2o)
h2o.init()

prosPath <- system.file("extdata", "prostate.csv", package="h2o")
hex <- h2o.uploadFile(prosPath)

hex[,2] <- as.factor(hex[,2])
model <- h2o.gbm(x = 3:9, y = 2, training_frame = hex, distribution = "bernoulli")
perf <- h2o.performance(model, hex)
h2o.giniCoef(perf)

```

h2o.glm

*H2O Generalized Linear Models***Description**

Fit a generalized linear model, specified by a response variable, a set of predictors, and a description of the error distribution.

**Usage**

```

h2o.glm(x, y, training_frame, model_id, validation_frame, max_iterations = 50,
  beta_epsilon = 0, solver = c("IRLSM", "L_BFGS"), standardize = TRUE,
  family = c("gaussian", "binomial", "poisson", "gamma", "tweedie"),
  link = c("family_default", "identity", "logit", "log", "inverse",
  "tweedie"), tweedie_variance_power = NaN, tweedie_link_power = NaN,
  alpha = 0.5, prior = 0, lambda = 1e-05, lambda_search = FALSE,
  nlambda = -1, lambda_min_ratio = -1, nfold, beta_constraints = NULL,
  ...)

```

**Arguments**

x	A vector containing the names or indices of the predictor variables to use in building the GLM model.
y	A character string or index that represent the response variable in the model.
training_frame	An <a href="#">H2OFrame</a> object containing the variables in the model.
model_id	(Optional) The unique id assigned to the resulting model. If none is given, an id will automatically be generated.
validation_frame	An <a href="#">H2OFrame</a> object containing the variables in the model.
max_iterations	A non-negative integer specifying the maximum number of iterations.
beta_epsilon	A non-negative number specifying the magnitude of the maximum difference between the coefficient estimates from successive iterations. Defines the convergence criterion for h2o.glm.



solver	A character string specifying the solver used: IRLSM (supports more features), L_BFGS (scales better for datasets with many columns)
standardize	A logical value indicating whether the numeric predictors should be standardized to have a mean of 0 and a variance of 1 prior to training the models.
family	A character string specifying the distribution of the model: gaussian, binomial, poisson, gamma, tweedie.
link	A character string specifying the link function. The default is the canonical link for the family. The supported links for each of the family specifications are: "gaussian": "identity", "log", "inverse" "binomial": "logit", "log" "poisson": "log", "identity" "gamma": "inverse", "log", "identity" "tweedie": "tweedie"
tweedie_variance_power	A numeric specifying the power for the variance function when family = "tweedie".
tweedie_link_power	A numeric specifying the power for the link function when family = "tweedie".
alpha	A numeric in [0, 1] specifying the elastic-net mixing parameter. The elastic-net penalty is defined to be: $P(\alpha, \beta) = (1 - \alpha)/2 \ \beta\ _2^2 + \alpha \ \beta\ _1 = \sum_j [(1 - \alpha)/2\beta_j^2 + \alpha \beta_j ]$ , making alpha = 1 the lasso penalty and alpha = 0 the ridge penalty.
prior	(Optional) A numeric specifying the prior probability of class 1 in the response when family = "binomial". The default prior is the observational frequency of class 1.
lambda	A non-negative shrinkage parameter for the elastic-net, which multiplies $P(\alpha, \beta)$ in the objective function. When lambda = 0, no elastic-net penalty is applied and ordinary generalized linear models are fit.
lambda_search	A logical value indicating whether to conduct a search over the space of lambda values starting from the lambda max, given lambda is interpreted as lambda min.
nlambda	The number of lambda values to use when lambda_search = TRUE.
lambda_min_ratio	Smallest value for lambda as a fraction of lambda.max. By default if the number of observations is greater than the the number of variables then lambda_min_ratio = 0.0001; if the number of observations is less than the number of variables then lambda_min_ratio = 0.01.
nfolds	(Currently Unimplemented)
beta_constraints	A data.frame or H2OParsedData object with the columns ["names", "lower_bounds", "upper_bounds", "beta_given"], where each row corresponds to a predictor in the GLM. "names" contains the predictor names, "lower"/"upper_bounds", are the lower and upper bounds of beta, and "beta_given" is some supplied starting values for the
...	(Currently Unimplemented) coefficients.

**Value**

A subclass of `H2OModel` is returned. The specific subclass depends on the machine learning task at hand (if it's binomial classification, then an `H2OBinomialModel` is returned, if it's regression then a `H2ORegressionModel` is returned). The default print-out of the models is shown, but further GLM-specific information can be queried out of the object. To access these various items, please refer to the `sealso` section below.

Upon completion of the GLM, the resulting object has coefficients, normalized coefficients, residual/null deviance, aic, and a host of model metrics including MSE, AUC (for logistic regression), degrees of freedom, and confusion matrices. Please refer to the more in-depth GLM documentation available here: <http://h2o-release.s3.amazonaws.com/h2o-dev/rel-shannon/2/docs-website/h2o-docs/index.html#Data+Science+Algorithms-GLM>,

**See Also**

`predict.H2OModel` for prediction, `h2o.mse`, `h2o.auc`, `h2o.confusionMatrix`, `h2o.performance`, `h2o.giniCoef`, `h2o.logloss`, `h2o.varimp`, `h2o.scoreHistory`

**Examples**

```
localH2O = h2o.init()

# Run GLM of CAPSULE ~ AGE + RACE + PSA + DCAPS
prostatePath = system.file("extdata", "prostate.csv", package = "h2o")
prostate.hex = h2o.importFile(localH2O, path = prostatePath, destination_frame = "prostate.hex")
h2o.glm(y = "CAPSULE", x = c("AGE", "RACE", "PSA", "DCAPS"), training_frame = prostate.hex,
        family = "binomial", nfolds = 0, alpha = 0.5, lambda_search = FALSE)

# Run GLM of VOL ~ CAPSULE + AGE + RACE + PSA + GLEASON
myX = setdiff(colnames(prostate.hex), c("ID", "DPROS", "DCAPS", "VOL"))
h2o.glm(y = "VOL", x = myX, training_frame = prostate.hex, family = "gaussian",
        nfolds = 0, alpha = 0.1, lambda_search = FALSE)

## Not run:
# GLM variable importance
# Also see:
# https://github.com/h2oai/h2o/blob/master/R/tests/testdir_demos/runit_demo_VI_all_algos.R
data.hex = h2o.importFile(
  localH2O,
  path = "https://raw.githubusercontent.com/h2oai/h2o/master/smалldata/bank-additional-full.csv",
  destination_frame = "data.hex")
myX = 1:20
myY="y"
my.glm = h2o.glm(x=myX, y=myY, training_frame=data.hex, family="binomial", standardize=TRUE,
                 lambda_search=TRUE)

## End(Not run)
```

h2o.glm

*Generalized Low Rank Model***Description**

Generalized low rank decomposition of a H2O dataset.

**Usage**

```
h2o.glm(training_frame, x, k, destination_key, loading_key,
  transform = c("NONE", "DEMEAN", "DESCALE", "STANDARDIZE", "NORMALIZE"),
  loss = c("L2", "L1", "Huber", "Poisson", "Hinge", "Logistic"),
  regularization_x = c("L2", "L1"), regularization_y = c("L2", "L1"),
  gamma_x = 0, gamma_y = 0, max_iterations = 1000, init_step_size = 1,
  min_step_size = 0.001, init = c("PlusPlus", "SVD", "User"),
  recover_pca = FALSE, seed)
```

**Arguments**

training_frame	An <a href="#">H2OFrame</a> object containing the variables in the model.
x	(Optional) A vector containing the data columns on which k-means operates.
k	The rank of the resulting decomposition. This must be between 1 and the number of columns in the training frame, inclusive.
destination_key	(Optional) The unique hex key assigned to the resulting model. Automatically generated if none is provided.
loading_key	(Optional) The unique hex key assigned to the loading matrix X in the XY decomposition. Automatically generated if none is provided.
transform	A character string that indicates how the training data should be transformed before running PCA. Possible values are "NONE": for no transformation, "DEMEAN": for subtracting the mean of each column, "DESCALE": for dividing by the standard deviation of each column, "STANDARDIZE": for demeaning and dcaling, and "NORMALIZE": for demeaning and dividing each column by its range (max - min).
loss	A character string indicating the loss function. Possible values are "L2", "L1", "Huber", "Poisson", "Hinge" and "Logistic".
regularization_x	A character string indicating the regularization function for the X matrix. Possible values are "L2" and "L1".
regularization_y	A character string indicating the regularization function for the Y matrix. Possible values are "L2" and "L1".
gamma_x	The weight on the X matrix regularization term. For no X regularization, set this value to zero.

gamma_y	The weight on the Y matrix regularization term. For no Y regularization, set this value to zero.
max_iterations	The maximum number of iterations to run the optimization loop. Each iteration consists of an update of the X matrix, followed by an update of the Y matrix.
init_step_size	Initial step size. Divided by number of columns in the training frame when calculating the proximal gradient update. The algorithm begins at init_step_size and decreases the step size at each iteration until a termination condition is reached.
min_step_size	Minimum step size upon which the algorithm is terminated.
init	A character string indicating how to select the initial Y matrix. Possible values are "PlusPlus": for initialization using the clusters from k-means++ initialization, or "SVD": for initialization using the first k right singular vectors. Additionally, the user may specify the initial Y as a matrix, data.frame, H2OFrame, or list of vectors.
recover_pca	A logical value indicating whether the principal components should be recovered during post-processing of the generalized low rank decomposition.
seed	(Optional) Random seed used to initialize the X and Y matrices.

---

h2o.group\_by

*Group and Apply by Column*


---

## Description

Performs a group by and apply similar to dply.

## Usage

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

## Arguments

data	an <a href="#">H2OFrame</a> object.
by	a list of column names
order.by	Takes a vector column names or indices specifying how to order the group by result.
gb.control	a list of how to handle NA values in the dataset as well as how to name output columns. See <a href="#">Details</a> : for more help.
...	any supported aggregate function.

**Details**

In the case of `na.methods` within `gb.control`, there are three possible settings. "all" will include NAs in computation of functions. "rm" will completely remove all NA fields. "ignore" will remove NAs from the numerator but keep the rows for computational purposes. If a list smaller than the number of columns groups is supplied, the list will be padded by "ignore".

Similar to `na.methods`, `col.names` will pad the list with the default column names if the length is less than the number of columns groups supplied.

**Value**

Returns a new [H2OFrame](#) object with columns equivalent to the number of groups created

---

h2o.gsub	<i>String Global Substitute</i>
----------	---------------------------------

---

**Description**

Mutates the input. Changes the all occurrences of pattern with replacement.

**Usage**

```
h2o.gsub(pattern, replacement, x, ignore.case = FALSE)
```

**Arguments**

pattern	The pattern to replace.
replacement	The replacement pattern.
x	The column on which to operate.
ignore.case	Case sensitive or not

---

h2o.head	<i>Return the Head or Tail of an H2O Dataset.</i>
----------	---

---

**Description**

Returns the first or last rows of an H2O parsed data object.

**Usage**

```
## S4 method for signature H2OFrame
head(x, n = 6L, ...)
```

```
## S4 method for signature H2OFrame
tail(x, n = 6L, ...)
```

**Arguments**

x	An <a href="#">H2OFrame</a> object.
n	(Optional) A single integer. If positive, number of rows in x to return. If negative, all but the n first/last number of rows in x.
...	Further arguments passed to or from other methods.

**Value**

A data frame containing the first or last n rows of an [H2OFrame](#) object.

**Examples**

```
library(h2o)
localH2O <- h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
ausPath <- system.file("extdata", "australia.csv", package="h2o")
australia.hex <- h2o.uploadFile(localH2O, path = ausPath)
head(australia.hex, 10)
tail(australia.hex, 10)
```

---

h2o.hist

*Compute A Histogram*


---

**Description**

Compute a histogram over a numeric column. If `breaks=="FD"`, the MAD is used over the IQR in computing bin width.

**Usage**

```
h2o.hist(x, breaks = "Sturges")
```

**Arguments**

x	A single numeric column from an <a href="#">H2OFrame</a> .
breaks	Can be one of the following: A string: "Sturges", "Rice", "sqrt", "Doane", "FD", "Scott" A single number for the number of breaks splitting the range of the vec into number of breaks bins of equal width A vector of numbers giving the split points, e.g., <code>c(-50,213.2123,9324834)</code>

---

h2o.hit\_ratio\_table     *Retrieve the Hit Ratios*

---

**Description**

Retrieve the Hit Ratios

**Usage**

```
h2o.hit_ratio_table(object, valid = FALSE, ...)
```

**Arguments**

object	An <a href="#">H2OModel</a> object.
valid	Retrieve the validation metric.
...	further arguments to be passed on (currently unimplemented)

---

h2o.hour                     *Convert Milliseconds to Hour of Day in H2O Datasets*

---

**Description**

Converts the entries of a [H2OFrame](#) object from milliseconds to hours of the day (on a 0 to 23 scale).

**Usage**

```
h2o.hour(x)
```

```
hour(x)
```

```
## S3 method for class H2OFrame  
hour(x)
```

**Arguments**

x	An <a href="#">H2OFrame</a> object.
---	-------------------------------------

**Value**

A [H2OFrame](#) object containing the entries of x converted to hours of the day.

**See Also**

[h2o.day](#)

---

`h2o.ifelse`*H2O Apply Conditional Statement*

---

## Description

Applies conditional statements to numeric vectors in H2O parsed data objects when the data are numeric.

## Usage

```
## S4 method for signature H2OFrame,ANY,ANY
ifelse(test, yes, no)

## S4 method for signature ANY,H2OFrame,H2OFrame
ifelse(test, yes, no)
```

## Arguments

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

## Details

Only numeric values can be tested, and only numeric results can be returned for either condition. Categorical data is not currently supported for this function and returned values cannot be categorical in nature.

## Value

Returns a vector of new values matching the conditions stated in the ifelse call.

## Examples

```
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
ausPath = system.file("extdata", "australia.csv", package="h2o")
australia.hex = h2o.importFile(localH2O, path = ausPath)
australia.hex[,9] <- ifelse(australia.hex[,3] < 279.9, 1, 0)
summary(australia.hex)
```



---

h2o.importFile	<i>Import Files into H2O</i>
----------------	------------------------------

---

### Description

Imports files into an H2O cloud. The default behavior is to pass-through to the parse phase automatically.

### Usage

```
h2o.importFolder(path, conn = h2o.getConnection(), pattern = "",
  destination_frame = "", parse = TRUE, header = NA, sep = "",
  col.names = NULL, na.strings = NULL)
```

```
h2o.importURL(path, conn = h2o.getConnection(), destination_frame = "",
  parse = TRUE, header = NA, sep = "", col.names = NULL,
  na.strings = NULL)
```

```
h2o.importHDFS(path, conn = h2o.getConnection(), pattern = "",
  destination_frame = "", parse = TRUE, header = NA, sep = "",
  col.names = NULL, na.strings = NULL)
```

```
h2o.uploadFile(path, conn = h2o.getConnection(), destination_frame = "",
  parse = TRUE, header = NA, sep = "", col.names = NULL,
  col.types = NULL, na.strings = NULL, progressBar = FALSE)
```

### Arguments

path	The complete URL or normalized file path of the file to be imported. Each row of data appears as one line of the file.
conn	an <a href="#">H2OConnection</a> class object.
pattern	(Optional) Character string containing a regular expression to match file(s) in the folder.
destination_frame	(Optional) The unique hex key assigned to the imported file. If none is given, a key will automatically be generated based on the URL path.
parse	(Optional) A logical value indicating whether the file should be parsed after import.
header	(Optional) A logical value indicating whether the first line of the file contains column headers. If left empty, the parser will try to automatically detect this.
sep	(Optional) The field separator character. Values on each line of the file are separated by this character. If sep = "", the parser will automatically detect the separator.
col.names	(Optional) A <a href="#">H2ORawData</a> or <a href="#">H2OFrame</a> (version = 2) object containing a single delimited line with the column names for the file.

na.strings	(Optional) H2O will interpret these strings as missing.
col.types	(Optional) A vector to specify whether columns should be forced to a certain type upon import parsing.
progressBar	(Optional) When FALSE, tell H2O parse call to block synchronously instead of polling. This can be faster for small datasets but loses the progress bar.

### Details

Other than `h2o.uploadFile`, if the given path is relative, then it will be relative to the start location of the H2O instance. Additionally, the file must be on the same machine as the H2O cloud. In the case of `h2o.uploadFile`, a relative path will resolve relative to the working directory of the current R session.

Import an entire directory of files. If the given path is relative, then it will be relative to the start location of the H2O instance. The default behavior is to pass-through to the parse phase automatically.

`h2o.importURL` and `h2o.importHDFS` are both deprecated functions. Instead, use `h2o.importFile`

### Examples

```
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
prosPath = system.file("extdata", "prostate.csv", package = "h2o")
prostate.hex = h2o.uploadFile(localH2O, path = prosPath, destination_frame = "prostate.hex")
class(prostate.hex)
summary(prostate.hex)
```

---

```
h2o.impute      # children <- list(c(paste0('nAggs'), unlist(lapply(aggs, function(l)
                    .args.to.ast(.args=l))))))
```

---

### Description

Basic Imputation of H2O Vectors

### Usage

```
h2o.impute(data, column, method = c("mean", "median", "mode"),
  combine_method = c("interpolate", "average", "lo", "hi"), by = NULL,
  inplace = TRUE)
```

### Arguments

data	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
inplace	Perform the imputation inplace or make a copy. Default is to perform the imputation in place.

### Details

Perform simple imputation on a single vector by filling missing values with aggregates computed on the "na.rm'd" vector. Additionally, it's possible to perform imputation based on groupings of columns from within data; these columns can be passed by index or name to the by parameter. If a factor column is supplied, then the method must be one "mode". Anything else results in a full stop.

The default method is selected based on the type of the column to impute. If the column is numeric then "mean" is selected; if it is categorical, then "mode" is selected. Otherwise column types (e.g. String, Time, UUID) are not supported.

### Value

a H2OFrame with imputed values

### Examples

```
h2o.init()
fr <- as.h2o(iris, destination_frame="iris")
fr[sample(nrow(fr),40),5] <- NA # randomly replace 50 values with NA
# impute with a group by
h2o.impute(fr, "Species", "mode", by=c("Sepal.Length", "Sepal.Width"))
```

---

h2o.init

*Initialize and Connect to H2O*

---

### Description

Attempts to start and/or connect to and H2O instance.

### Usage

```
h2o.init(ip = "127.0.0.1", port = 54321, startH2O = TRUE,
forceDL = FALSE, Xmx, beta = FALSE, assertion = TRUE, license = NULL,
nthreads = -2, max_mem_size = NULL, min_mem_size = NULL,
ice_root = tempdir(), strict_version_check = TRUE)
```

**Arguments**

<code>ip</code>	Object of class character representing the IP address of the server where H2O is running.
<code>port</code>	Object of class numeric representing the port number of the H2O server.
<code>startH2O</code>	(Optional) A logical value indicating whether to try to start H2O from R if no connection with H2O is detected. This is only possible if <code>ip = "localhost"</code> or <code>ip = "127.0.0.1"</code> . If an existing connection is detected, R does not start H2O.
<code>forcedL</code>	(Optional) A logical value indicating whether to force download of the H2O executable. Defaults to FALSE, so the executable will only be downloaded if it does not already exist in the h2o R library resources directory <code>h2o/java/h2o.jar</code> . This value is only used when R starts H2O.
<code>Xmx</code>	(Optional) (DEPRECATED) A character string specifying the maximum size, in bytes, of the memory allocation pool to H2O. This value must a multiple of 1024 greater than 2MB. Append the letter m or M to indicate megabytes, or g or G to indicate gigabytes. This value is only used when R starts H2O.
<code>beta</code>	(Optional) A logical value indicating whether H2O should launch in beta mode. This value is only used when R starts H2O.
<code>assertion</code>	(Optional) A logical value indicating whether H2O should be launched with assertions enabled. Used mainly for error checking and debugging purposes. This value is only used when R starts H2O.
<code>license</code>	(Optional) A character string value specifying the full path of the license file. This value is only used when R starts H2O.
<code>nthreads</code>	(Optional) Number of threads in the thread pool. This relates very closely to the number of CPUs used. -2 means use the CRAN default of 2 CPUs. -1 means use all CPUs on the host. A positive integer specifies the number of CPUs directly. This value is only used when R starts H2O.
<code>max_mem_size</code>	(Optional) A character string specifying the maximum size, in bytes, of the memory allocation pool to H2O. This value must a multiple of 1024 greater than 2MB. Append the letter m or M to indicate megabytes, or g or G to indicate gigabytes. This value is only used when R starts H2O.
<code>min_mem_size</code>	(Optional) A character string specifying the minimum size, in bytes, of the memory allocation pool to H2O. This value must a multiple of 1024 greater than 2MB. Append the letter m or M to indicate megabytes, or g or G to indicate gigabytes. This value is only used when R starts H2O.
<code>ice_root</code>	(Optional) A directory to handle object spillage. The default varies by OS.
<code>strict_version_check</code>	(Optional) Setting this to FALSE is unsupported and should only be done when advised by technical support.

**Details**

By default, this method first checks if an H2O instance is connectible. If it cannot connect and `start = TRUE` with `ip = "localhost"`, it will attempt to start an instance of H2O at `localhost:54321`. Otherwise it stops with an error.

When initializing H2O locally, this method searches for h2o.jar in the R library resources (system.file("java", "h2o.jar") and if the file does not exist, it will automatically attempt to download the correct version from Amazon S3. The user must have Internet access for this process to be successful.

Once connected, the method checks to see if the local H2O R package version matches the version of H2O running on the server. If there is a mismatch and the user indicates she wishes to upgrade, it will remove the local H2O R package and download/install the H2O R package from the server.

### Value

this method will load it and return a H2OConnection object containing the IP address and port number of the H2O server.

### Note

Users may wish to manually upgrade their package (rather than waiting until being prompted), which requires that they fully uninstall and reinstall the H2O package, and the H2O client package. You must unload packages running in the environment before upgrading. It's recommended that users restart R or R studio after upgrading

### See Also

[H2O R package documentation](#) for more details. [h2o.shutdown](#) for shutting down from R.

### Examples

```
## Not run:
# Try to connect to a local H2O instance that is already running.
# If not found, start a local H2O instance from R with the default settings.
localH2O = h2o.init()

# Try to connect to a local H2O instance.
# If not found, raise an error.
localH2O = h2o.init(startH2O = FALSE)

# Try to connect to a local H2O instance that is already running.
# If not found, start a local H2O instance from R with 5 gigabytes of memory.
localH2O = h2o.init(max_mem_size = "5g")

# Try to connect to a local H2O instance that is already running.
# If not found, start a local H2O instance from R that uses 5 gigabytes of memory.
localH2O = h2o.init(max_mem_size = "5g")

## End(Not run)
```

---

 h2o.insertMissingValues

*Inserting Missing Values to an H2O DataFrame*


---

### Description

\*This is primarily used for testing\*. Randomly replaces a user-specified fraction of entries in a H2O dataset with missing values.

### Usage

```
h2o.insertMissingValues(data, fraction = 0.1, seed = -1)
```

### Arguments

data	An <a href="#">H2OFrame</a> object representing the dataset.
fraction	A number between 0 and 1 indicating the fraction of entries to replace with missing.
seed	A random number used to select which entries to replace with missing values. Default of seed = -1 will automatically generate a seed in H2O.

### WARNING

This will modify the original dataset. Unless this is intended, this function should only be called on a subset of the original.

### Examples

```
library(h2o)
localH2O <- h2o.init()
irisPath <- system.file("extdata", "iris.csv", package = "h2o")
iris.hex <- h2o.importFile(localH2O, path = irisPath)
summary(iris.hex)
irismiss.hex <- h2o.insertMissingValues(iris.hex, fraction = 0.25)
head(irismiss.hex)
summary(irismiss.hex)
```

---

 h2o.interaction

*Categorical Interaction Feature Creation in H2O*


---

### Description

Creates a data frame in H2O with n-th order interaction features between categorical columns, as specified by the user.

**Usage**

```
h2o.interaction(data, destination_frame, factors, pairwise, max_factors,
               min_occurrence)
```

**Arguments**

data	An <a href="#">H2OFrame</a> object containing the categorical columns.
destination_frame	A string indicating the destination key. If empty, this will be auto-generated by H2O.
factors	Factor columns (either indices or column names).
pairwise	Whether to create pairwise interactions between factors (otherwise create one higher-order interaction). Only applicable if there are 3 or more factors.
max_factors	Max. number of factor levels in pair-wise interaction terms (if enforced, one extra catch-all factor will be made)
min_occurrence	Min. occurrence threshold for factor levels in pair-wise interaction terms

**Value**

Returns a [H2OFrame](#) object.

**Examples**

```
library(h2o)
localH2O <- h2o.init()

# Create some random data
myframe = h2o.createFrame(localH2O, framekey, rows = 20, cols = 5,
                          seed = -12301283, randomize = TRUE, value = 0,
                          categorical_fraction = 0.8, factors = 10, real_range = 1,
                          integer_fraction = 0.2, integer_range = 10,
                          binary_fraction = 0, binary_ones_fraction = 0.5,
                          missing_fraction = 0.2,
                          response_factors = 1)

# Turn integer column into a categorical
myframe[,5] <- as.factor(myframe[,5])
head(myframe, 20)

# Create pairwise interactions
pairwise <- h2o.interaction(myframe, destination_frame = pairwise,
                           factors = list(c(1,2),c("C2","C3","C4")),
                           pairwise=TRUE, max_factors = 10, min_occurrence = 1)

head(pairwise, 20)
h2o.levels(pairwise,2)

# Create 5-th order interaction
higherorder <- h2o.interaction(myframe, destination_frame = higherorder, factors = c(1,2,3,4,5),
                              pairwise=FALSE, max_factors = 10000, min_occurrence = 1)

head(higherorder, 20)
```

```

# Limit the number of factors of the "categoricalized" integer column
# to at most 3 factors, and only if they occur at least twice
head(myframe[,5], 20)
trim_integer_levels <- h2o.interaction(myframe, destination_frame = trim_integers, factors = "C5",
                                       pairwise = FALSE, max_factors = 3, min_occurrence = 2)

head(trim_integer_levels, 20)

# Put all together
myframe <- h2o.cbind(myframe, pairwise, higherorder, trim_integer_levels)
myframe
head(myframe,20)
summary(myframe)

```

---

h2o.killMinus3	<i>Dump the stack into the JVM's stdout.</i>
----------------	--

---

### Description

A poor man's profiler, but effective.

### Usage

```
h2o.killMinus3(conn = h2o.getConnection())
```

### Arguments

conn            an [H2OConnection](#) class object.

---

h2o.kmeans	<i>KMeans Model in H2O</i>
------------	----------------------------

---

### Description

Performs k-means clustering on an H2O dataset.

### Usage

```
h2o.kmeans(training_frame, x, k, model_id, max_iterations = 1000,
           standardize = TRUE, init = c("Furthest", "Random", "PlusPlus"), seed)
```



**Arguments**

training_frame	An <a href="#">H2OFrame</a> object containing the variables in the model.
x	(Optional) A vector containing the data columns on which k-means operates.
k	The number of clusters. Must be between 1 and 1e7 inclusive. k may be omitted if the user specifies the initial centers in the init parameter. If k is not omitted, in this case, then it should be equal to the number of user-specified centers.
model_id	(Optional) The unique id assigned to the resulting model. If none is given, an id will automatically be generated.
max_iterations	The maximum number of iterations allowed. Must be between 0
standardize	Logical, indicates whether the data should be standardized before running k-means.
init	A character string that selects the initial set of k cluster centers. Possible values are "Random": for random initialization, "PlusPlus": for k-means plus initialization, or "Furthest": for initialization at the furthest point from each successive center. Additionally, the user may specify a the initial centers as a matrix, data.frame, H2OFrame, or list of vectors. For matrices, data.frames, and H2OFrames, each row of the respective structure is an initial center. For lists of vectors, each vector is an initial center.
seed	(Optional) Random seed used to initialize the cluster centroids.

**Value**

Returns an object of class [H2OClusteringModel](#).

**See Also**

[h2o.cluster\\_sizes](#), [h2o.totss](#), [h2o.num\\_iterations](#), [h2o.betweenss](#), [h2o.tot\\_withinss](#), [h2o.withinss](#), [h2o.centersSTD](#), [h2o.centers](#)

**Examples**

```
library(h2o)
localH2O <- h2o.init()
prosPath <- system.file("extdata", "prostate.csv", package="h2o")
prostate.hex <- h2o.uploadFile(localH2O, path = prosPath)
h2o.kmeans(training_frame = prostate.hex, k = 10, x = c("AGE", "RACE", "VOL", "GLEASON"))
```

---

h2o.length

*Returns the Length of a Parsed H2O Data Object.*

---

**Description**

Returns the length of an [H2OFrame](#)

**Usage**

```
## S4 method for signature H2OFrame
length(x)
```

**Arguments**

x                    An [H2OFrame](#) object.

**See Also**

[length](#) for the base R method.

**Examples**

```
localH2O <- h2o.init()
irisPath <- system.file("extdata", "iris.csv", package = "h2o")
iris.hex <- h2o.uploadFile(localH2O, path = irisPath)
length(iris.hex)
```

---

h2o.levels

*Return the levels from the column requested column.*

---

**Description**

Return the levels from the column requested column.

**Arguments**

x                    An [H2OFrame](#) object.  
i                    The index of the column whose domain is to be returned.

**See Also**

[levels](#) for the base R method.

**Examples**

```
localH2O <- h2o.init()
iris.hex <- as.h2o(localH2O, iris)
h2o.levels(iris.hex, 5) # returns "setosa"    "versicolor" "virginica"
```

---

h2o.listTimezones	<i>List all of the Time Zones Acceptable by the H2O Cloud.</i>
-------------------	--

---

**Description**

List all of the Time Zones Acceptable by the H2O Cloud.

**Usage**

```
h2o.listTimezones(conn = h2o.getConnection())
```

**Arguments**

conn	An H2OConnection object.
------	--------------------------

---

h2o.loadModel	<i>Load H2O Model from HDFS or Local Disk</i>
---------------	---

---

**Description**

Load a saved H2O model from disk. Currently not implemented.

**Usage**

```
h2o.loadModel(path, conn = h2o.getConnection())
```

**Arguments**

path	The path of the H2O Model to be imported.
conn	an <a href="#">H2OConnection</a> object containing the IP address and port of the server running H2O.

**Value**

Returns a [H2OModel](#) object of the class corresponding to the type of model built.

**See Also**

[h2o.saveModel](#), [H2OModel](#)

## Examples

```
## Not run:
# library(h2o)
# localH2O = h2o.init()
# prosPath = system.file("extdata", "prostate.csv", package = "h2o")
# prostate.hex = h2o.importFile(localH2O, path = prosPath, destination_frame = "prostate.hex")
# prostate.glm = h2o.glm(y = "CAPSULE", x = c("AGE", "RACE", "PSA", "DCAPS"),
#   training_frame = prostate.hex, family = "binomial", alpha = 0.5)
# glmmodel.path = h2o.saveModel(object = prostate.glm, dir = "/Users/UserName/Desktop")
# glmmodel.load = h2o.loadModel(localH2O, glmmodel.path)

## End(Not run)
```

---

h2o.logAndEcho

*Log a message on the server-side logs*

---

## Description

This is helpful when running several pieces of work one after the other on a single H2O cluster and you want to make a notation in the H2O server side log where one piece of work ends and the next piece of work begins.

## Usage

```
h2o.logAndEcho(message, conn = h2o.getConnection())
```

## Arguments

message	A character string with the message to write to the log.
conn	An H2OConnection object pointing to a running H2O cluster.

## Details

h2o.logAndEcho sends a message to H2O for logging. Generally used for debugging purposes.

## See Also

[H2OConnection](#)

---

h2o.logloss	<i>Retrieve the Log Loss Value</i>
-------------	------------------------------------

---

**Description**

Retrieves the log loss output for a [H2OBinomialMetrics](#) or [H2OMultinomialMetrics](#) object

**Usage**

```
h2o.logloss(object, valid = FALSE, ...)
```

**Arguments**

object	a <a href="#">H2OModelMetrics</a> object of the correct type.
valid	Retrieve the validation metric.
...	Extra arguments to be passed if 'object' is of type <a href="#">H2OModel</a> (e.g. train=TRUE)

---

h2o.ls	<i>List Keys on an H2O Cluster</i>
--------	------------------------------------

---

**Description**

Accesses a list of object keys in the running instance of H2O.

**Usage**

```
h2o.ls(conn = h2o.getConnection())
```

**Arguments**

conn	An <a href="#">H2OConnection</a> object containing the IP address and port number of the H2O server.
------	--

**Value**

Returns a list of hex keys in the current H2O instance.

**Examples**

```
library(h2o)
localH2O <- h2o.init()
prosPath <- system.file("extdata", "prostate.csv", package="h2o")
prostate.hex <- h2o.uploadFile(localH2O, path = prosPath)
h2o.ls(localH2O)
```

---

h2o.makeGLMModel	<i>Remake an H2O GLM Model</i>
------------------	--------------------------------

---

**Description**

This function allows the usage of new beta constraints to create an GLM model, from an existing model.

**Usage**

```
h2o.makeGLMModel(model, beta)
```

**Arguments**

model	an <a href="#">H2OModel</a> corresponding from a h2o.glm call.
beta	a new set of beta_constraints

---

h2o.match	<i>Value Matching in H2O</i>
-----------	------------------------------

---

**Description**

match and %in% return values similar to the base R generic functions.

**Usage**

```
## S4 method for signature H2OFrame
match(x, table, nomatch = 0, incomparables = NULL)

## S4 method for signature H2OFrame
x %in% table
```

**Arguments**

x	a categorical vector from an <a href="#">H2OFrame</a> object with values to be matched.
table	an R object to match x against.
nomatch	the value to be returned in the case when no match is found.
incomparables	a vector of values that cannot be matched. Any value in x matching a value in this vector is assigned the nomatch value.

**See Also**

[match](#) for base R implementation.

**Examples**

```
h2o.init()
hex <- as.h2o(iris)
match(hex[,5], c("setosa", "versicola"))
```

---

h2o.mean	<i>Mean of a column</i>
----------	-------------------------

---

**Description**

Obtain the mean of a column of a parsed H2O data object.

**Usage**

```
## S4 method for signature H2OFrame
mean(x, trim = 0, na.rm = FALSE, ...)
```

**Arguments**

x	An <a href="#">H2OFrame</a> object.
trim	The fraction (0 to 0.5) of observations to trim from each end of x before the mean is computed.
na.rm	A logical value indicating whether NA or missing values should be stripped before the computation.
...	Further arguments to be passed from or to other methods.

**See Also**

[mean](#) for the base R implementation.

**Examples**

```
localH2O <- h2o.init()
prosPath <- system.file("extdata", "prostate.csv", package="h2o")
prostate.hex <- h2o.uploadFile(localH2O, path = prosPath)
mean(prostate.hex$AGE)
```

---

h2o.merge	<i>Merge Two H2O Data Frames</i>
-----------	----------------------------------

---

**Description**

Merges two [H2OFrame](#) objects by shared column names. Unlike the base R implementation, `h2o.merge` only supports merging through shared column names.

**Usage**

```
h2o.merge(x, y, all.x = FALSE, all.y = FALSE)
```

**Arguments**

<code>x,y</code>	<a href="#">H2OFrame</a> objects
<code>all.x</code>	a logical value indicating whether or not shared values are preserved or ignored in x.
<code>all.y</code>	a logical value indicating whether or not shared values are preserved or ignored in y.

**Details**

In order for `h2o.merge` to work in multinode clusters, one of the datasets must be small enough to exist in every node. Currently, this function only supports `all.x = TRUE`. All other permutations will fail.

**Examples**

```
h2o.init()
left <- data.frame(fruit = c(apple, orange, banana, lemon, strawberry, blueberry),
  color = c(red, orange, yellow, yellow, red, blue))
right <- data.frame(fruit = c(apple, orange, banana, lemon, strawberry, watermelon),
  citrus = c(FALSE, TRUE, FALSE, TRUE, FALSE, FALSE))
l.hex <- as.h2o(left)
r.hex <- as.h2o(right)
left.hex <- h2o.merge(l.hex, r.hex, all.x = TRUE)
```

---

h2o.metric	<i>H2O Model Metric Accessor Functions</i>
------------	--

---

**Description**

A series of functions that retrieve model metric details.



**Usage**

```
h2o.metric(object, thresholds, metric)

h2o.F0point5(object, thresholds)

h2o.F1(object, thresholds)

h2o.F2(object, thresholds)

h2o.accuracy(object, thresholds)

h2o.error(object, thresholds)

h2o.maxPerClassError(object, thresholds)

h2o.mcc(object, thresholds)

h2o.precision(object, thresholds)

h2o.tpr(object, thresholds)

h2o.fpr(object, thresholds)

h2o.fnr(object, thresholds)

h2o.tnr(object, thresholds)

h2o.recall(object, thresholds)

h2o.sensitivity(object, thresholds)

h2o.fallout(object, thresholds)

h2o.missrate(object, thresholds)

h2o.specificity(object, thresholds)
```

**Arguments**

object	An <a href="#">H2OModelMetrics</a> object of the correct type.
thresholds	A value or a list of values between 0.0 and 1.0.
metric	A specified paramter to retrieve.

**Details**

Many of these functions have an optional thresholds parameter. Currently only increments of 0.1 are allowed. If not specified, the functions will return all possible values. Otherwise, the function will return the value for the indicated threshold.

Currently, these functions are only supported by [H2OBinomialMetrics](#) objects.

### Value

Returns either a single value, or a list of values.

### See Also

[h2o.auc](#) for AUC, [h2o.giniCoef](#) for the GINI coefficient, and [h2o.mse](#) for MSE. See [h2o.performance](#) for creating [H2OModelMetrics](#) objects.

### Examples

```
library(h2o)
h2o.init()

prosPath <- system.file("extdata", "prostate.csv", package="h2o")
hex <- h2o.uploadFile(prosPath)

hex[,2] <- as.factor(hex[,2])
model <- h2o.gbm(x = 3:9, y = 2, training_frame = hex, distribution = "bernoulli")
perf <- h2o.performance(model, hex)
h2o.F1(perf)
```

---

h2o.month

*Convert Milliseconds to Months in H2O Datasets*

---

### Description

Converts the entries of a [H2OFrame](#) object from milliseconds to months (on a 0 to 11 scale).

### Usage

```
h2o.month(x)

month(x)

## S3 method for class H2OFrame
month(x)
```

### Arguments

x                    An [H2OFrame](#) object.

### Value

A [H2OFrame](#) object containing the entries of x converted to months of the year.

**See Also**[h2o.year](#)

---

h2o.mse	<i>Retrieves Mean Squared Error Value</i>
---------	---

---

**Description**

Retrieves the mean squared error value from an [H2OModelMetrics](#) object.

**Usage**

```
h2o.mse(object, valid = FALSE, ...)
```

**Arguments**

object	An <a href="#">H2OModelMetrics</a> object of the correct type.
valid	Retrieve the validation metric.
...	Extra arguments to be passed if 'object' is of type <a href="#">H2OModel</a> (e.g. train=TRUE)

**Details**

This function only supports [H2OBinomialMetrics](#), [H2OMultinomialMetrics](#), and [H2ORegressionMetrics](#) objects.

**See Also**

[h2o.auc](#) for AUC, [h2o.mse](#) for MSE, and [h2o.metric](#) for the various threshold metrics. See [h2o.performance](#) for creating [H2OModelMetrics](#) objects.

**Examples**

```
library(h2o)
h2o.init()

prosPath <- system.file("extdata", "prostate.csv", package="h2o")
hex <- h2o.uploadFile(prosPath)

hex[,2] <- as.factor(hex[,2])
model <- h2o.gbm(x = 3:9, y = 2, training_frame = hex, distribution = "bernoulli")
perf <- h2o.performance(model, hex)
h2o.mse(perf)
```

---

h2o.naiveBayes

*Naive Bayes Model in H2O*


---

### Description

Compute naive Bayes probabilities on an H2O dataset.

### Usage

```
h2o.naiveBayes(x, y, training_frame, model_id, laplace = 0,
              threshold = 0.001, eps = 0, compute_metrics = TRUE)
```

### Arguments

x	A vector containing the names or indices of the predictor variables to use in building the model.
y	The name or index of the response variable. If the data does not contain a header, this is the column index number starting at 0, and increasing from left to right. The response must be a categorical variable with at least two levels.
training_frame	An <a href="#">H2OFrame</a> object containing the variables in the model.
model_id	(Optional) The unique id assigned to the resulting model. If none is given, an id will automatically be generated.
laplace	A positive number controlling Laplace smoothing. The default zero disables smoothing.
threshold	The minimum standard deviation to use for observations without enough data. Must be at least 1e-10.
eps	A threshold cutoff to deal with numeric instability, must be positive.
compute_metrics	A logical value indicating whether model metrics should be computed. Set to FALSE to reduce the runtime of the algorithm.

### Details

The naive Bayes classifier assumes independence between predictor variables conditional on the response, and a Gaussian distribution of numeric predictors with mean and standard deviation computed from the training dataset. When building a naive Bayes classifier, every row in the training dataset that contains at least one NA will be skipped completely. If the test dataset has missing values, then those predictors are omitted in the probability calculation during prediction.

The naive Bayes classifier assumes independence between predictor variables conditional on the response, and a Gaussian distribution of numeric predictors with mean and standard deviation computed from the training dataset. When building a naive Bayes classifier, every row in the training dataset that contains at least one NA will be skipped completely. If the test dataset has missing values, then those predictors are omitted in the probability calculation during prediction.

**Value**

Returns an object of class [H2OBinomialModel](#) if the response has two categorical levels, and [H2OMultinomialModel](#) otherwise.

**Examples**

```
## Not run:
localH2O <- h2o.init()
votesPath <- system.file("extdata", "housevotes.csv", package="h2o")
votes.hex <- h2o.uploadFile(localH2O, path = votesPath, header = TRUE)
h2o.naiveBayes(x = 2:17, y = 1, training_frame = votes.hex, laplace = 3)

## End(Not run)
```

---

h2o.networkTest	<i>View Network Traffic Speed</i>
-----------------	-----------------------------------

---

**Description**

View speed with various file sizes.

**Usage**

```
h2o.networkTest(conn = h2o.getConnection())
```

**Arguments**

conn                    an [H2OConnection](#) object.

**Value**

Returns a table listing the network speed for 1B, 10KB, and 10MB.

---

h2o.nlevels	<i>Return the number of levels in the column.</i>
-------------	---

---

**Description**

If a frame or non-categorical column is passed, returns 0.

**Usage**

```
h2o.nlevels(object)
```

**Arguments**

object                    An [H2OFrame](#) object.

---

h2o.nrow	<i>The Number of Rows/Columns of an H2O Dataset</i>
----------	---

---

**Description**

Returns a count of the number of rows or columns in an [H2OFrame](#) object.

**Usage**

```
## S4 method for signature H2OFrame
nrow(x)

## S4 method for signature H2OFrame
ncol(x)
```

**Arguments**

x                    An [H2OFrame](#) object.

**See Also**

[dim](#) for all the dimensions. [nrow](#) for the default R method.

**Examples**

```
library(h2o)
localH2O <- h2o.init()
irisPath <- system.file("extdata", "iris.csv", package="h2o")
iris.hex <- h2o.uploadFile(localH2O, path = irisPath)
nrow(iris.hex)
ncol(iris.hex)
```

---

h2o.null_deviance	<i>Retrieve the null deviance</i>
-------------------	-----------------------------------

---

**Description**

Retrieve the null deviance

**Usage**

```
h2o.null_deviance(object, valid = FALSE, ...)
```

**Arguments**

object              An [H2OModel](#) or [H2OModelMetrics](#)  
 valid                Retrieve the validation metric.  
 ...                  further arguments to be passed to/from this method.

---

h2o.null_dof	<i>Retrieve the null degrees of freedom</i>
--------------	---

---

**Description**

Retrieve the null degrees of freedom

**Usage**

```
h2o.null_dof(object, valid = FALSE, ...)
```

**Arguments**

object	An <a href="#">H2OModel</a> or <a href="#">H2OModelMetrics</a>
valid	Retrieve the validation metric.
...	further arguments to be passed to/from this method.

---

h2o.num_iterations	<i>Retrieve the number of iterations.</i>
--------------------	---

---

**Description**

Retrieve the number of iterations.

**Usage**

```
h2o.num_iterations(object)
```

**Arguments**

object	An <a href="#">H2OClusteringModel</a> object.
...	further arguments to be passed on (currently unimplemented)

---

h2o.openLog	<i>View H2O R Logs</i>
-------------	------------------------

---

**Description**

Open existing logs of H2O R POST commands and error responses on local disk. Used primarily for debugging purposes.

**Usage**

```
h2o.openLog(type)
```

**Arguments**

type	Currently unimplemented.
------	--------------------------

**See Also**

[h2o.startLogging](#), [h2o.stopLogging](#), [h2o.clearLog](#)

**Examples**

```
## Not run:
localH2O = h2o.init()

h2o.startLogging()
ausPath = system.file("extdata", "australia.csv", package="h2o")
australia.hex = h2o.importFile(localH2O, path = ausPath)
h2o.stopLogging()

# Not run to avoid windows being opened during R CMD check
# h2o.openLog("Command")
# h2o.openLog("Error")

## End(Not run)
```

---

h2o.parseRaw	<i>H2O Data Parsing</i>
--------------	-------------------------

---

**Description**

The second phase in the data ingestion step.

**Usage**

```
h2o.parseRaw(data, destination_frame = "", header = NA, sep = "",
  col.names = NULL, col.types = NULL, na.strings = NULL,
  blocking = FALSE)
```



**Arguments**

data	An <a href="#">H2ORawData</a> object to be parsed.
destination_frame	(Optional) The hex key assigned to the parsed file.
header	(Optional) A logical value indicating whether the first row is the column header. If missing, H2O will automatically try to detect the presence of a header.
sep	(Optional) The field separator character. Values on each line of the file are separated by this character. If sep = "", the parser will automatically detect the separator.
col.names	(Optional) A <a href="#">H2OFrame</a> object containing a single delimited line with the column names for the file.
col.types	(Optional) A vector specifying the types to attempt to force over columns.
na.strings	(Optional) H2O will interpret these strings as missing.
blocking	(Optional) Tell H2O parse call to block synchronously instead of polling. This can be faster for small datasets but loses the progress bar.

**Details**

Parse the Raw Data produced by the import phase.

---

h2o.parseSetup	<i>Get a parse setup back for the staged data.</i>
----------------	--

---

**Description**

Get a parse setup back for the staged data.

**Usage**

```
h2o.parseSetup(data, destination_frame = "", header = NA, sep = "",
  col.names = NULL, col.types = NULL, na.strings = NULL)
```

**Arguments**

data	An <a href="#">H2ORawData</a> object to be parsed.
destination_frame	(Optional) The hex key assigned to the parsed file.
header	(Optional) A logical value indicating whether the first row is the column header. If missing, H2O will automatically try to detect the presence of a header.
sep	(Optional) The field separator character. Values on each line of the file are separated by this character. If sep = "", the parser will automatically detect the separator.
col.names	(Optional) A <a href="#">H2OFrame</a> object containing a single delimited line with the column names for the file.
col.types	(Optional) A vector specifying the types to attempt to force over columns.
na.strings	(Optional) H2O will interpret these strings as missing.

---

h2o.performance

*Model Performance Metrics in H2O*


---

### Description

Given a trained h2o model, compute its performance on the given dataset

### Usage

```
h2o.performance(model, data = NULL, valid = FALSE, ...)
```

### Arguments

model	An <a href="#">H2OModel</a> object
data	An <a href="#">H2OFrame</a> . The model will make predictions on this dataset, and subsequently score them. The dataset should match the dataset that was used to train the model, in terms of column names, types, and dimensions. If data is passed in, then train and valid are ignored.
valid	A logical value indicating whether to return the validation metrics (constructed during training).
...	Extra args passed in for use by other functions.

### Value

Returns an object of the [H2OModelMetrics](#) subclass.

### Examples

```
library(h2o)
localH2O <- h2o.init()
prosPath <- system.file("extdata", "prostate.csv", package="h2o")
prostate.hex <- h2o.uploadFile(localH2O, path = prosPath)
prostate.hex$CAPSULE <- as.factor(prostate.hex$CAPSULE)
prostate.gbm <- h2o.gbm(3:9, "CAPSULE", prostate.hex)
h2o.performance(model = prostate.gbm, data=prostate.hex)
```

---

h2o.prcomp

*Principal Components Analysis*


---

### Description

Principal components analysis of a H2O dataset using the power method to calculate the singular value decomposition of the Gram matrix.

**Usage**

```
h2o.prcomp(training_frame, x, k, retx = TRUE, model_id,
           max_iterations = 1000, transform = "NONE", seed)
```

**Arguments**

training_frame	An <a href="#">H2OFrame</a> object containing the variables in the model.
x	(Optional) A vector containing the data columns on which SVD operates.
k	The number of principal components to be computed. This must be between 1 and <code>min(ncol(training_frame), nrow(training_frame))</code> inclusive.
retx	A logical value indicating whether the projected variables should be returned.
model_id	(Optional) The unique hex key assigned to the resulting model. Automatically generated if none is provided.
max_iterations	The maximum number of iterations to run each power iteration loop. Must be between 1 and <code>1e6</code> inclusive.
transform	A character string that indicates how the training data should be transformed before running PCA. Possible values are "NONE": for no transformation, "DEMEAN": for subtracting the mean of each column, "DESCALE": for dividing by the standard deviation of each column, "STANDARDIZE": for demeaning and descaling, and "NORMALIZE": for demeaning and dividing each column by its range (max - min).
seed	(Optional) Random seed used to initialize the right singular vectors at the beginning of each power method iteration.

**Value**

Returns an object of class [H2ODimReductionModel](#).

**Examples**

```
library(h2o)
localH2O <- h2o.init()
ausPath <- system.file("extdata", "australia.csv", package="h2o")
australia.hex <- h2o.uploadFile(localH2O, path = ausPath)
h2o.prcomp(training_frame = australia.hex, k = 8, transform = "STANDARDIZE")
```

---

h2o.r2

*Retrieve the R2 value*


---

**Description**

Retrieves the R2 value from an H2O model.

**Usage**

```
h2o.r2(object, valid = FALSE, ...)
```

**Arguments**

object	An <a href="#">H2OModel</a> object.
valid	Retrieve the validation set R2 if a validation set was passed in during model build time.
...	extra arguments to be passed if 'object' is of type <a href="#">H2OModel</a> (e.g. train=TRUE)

**Examples**

```
library(h2o)

h <- h2o.init()
fr <- as.h2o(iris)

m <- h2o.deeplearning(x=2:5,y=1,training_frame=fr)

h2o.r2(m)
```

---

h2o.randomForest	<i>Build a Big Data Random Forest Model</i>
------------------	---

---

**Description**

Builds a Random Forest Model on an [H2OFrame](#)

**Usage**

```
h2o.randomForest(x, y, training_frame, model_id, validation_frame,
  mtries = -1, sample_rate = 0.632, build_tree_one_node = FALSE,
  ntrees = 50, max_depth = 20, min_rows = 1, nbins = 20,
  balance_classes = FALSE, max_after_balance_size = 5, seed, ...)
```

**Arguments**

x	A vector containing the names or indices of the predictor variables to use in building the GBM model.
y	The name or index of the response variable. If the data does not contain a header, this is the column index number starting at 1, and increasing from left to right. (The response must be either an integer or a categorical variable).
training_frame	An <a href="#">H2OFrame</a> object containing the variables in the model.
model_id	(Optional) The unique id assigned to the resulting model. If none is given, an id will automatically be generated.
validation_frame	An <a href="#">H2OFrame</a> object containing the variables in the model.
mtries	Columns to randomly select at each level, or -1 for sqrt(#cols).
sample_rate	Sample rate, from 0 to 1.0.

build_tree_one_node	Run on one node only; no network overhead but fewer cpus used. Suitable for small datasets.
ntrees	A nonnegative integer that determines the number of trees to grow.
max_depth	Maximum depth to grow the tree.
min_rows	Minimum number of rows to assign to terminal nodes.
nbins	Number of bins to use in building histogram.
balance_classes	logical, indicates whether or not to balance training data class counts via over/under-sampling (for imbalanced data)
max_after_balance_size	Maximum relative size of the training data after balancing class counts (can be less than 1.0)
seed	Seed for random numbers (affects sampling) - Note: only reproducible when running single threaded
...	(Currently Unimplemented)

**Value**

Creates a [H2OModel](#) object of the right type.

**See Also**

[predict.H2OModel](#) for prediction.

---

h2o.rbind

*Combine H2O Datasets by Rows*


---

**Description**

Takes a sequence of H2O data sets and combines them by rows

**Usage**

```
h2o.rbind(...)
```

**Arguments**

... A sequence of [H2OFrame](#) arguments. All datasets must exist on the same H2O instance (IP and port) and contain the same number of rows.

**Value**

An [H2OFrame](#) object containing the combined ... arguments column-wise.

**See Also**

[rbind](#) for the base R method.

**Examples**

```
library(h2o)
localH2O <- h2o.init()
prosPath <- system.file("extdata", "prostate.csv", package="h2o")
prostate.hex <- h2o.uploadFile(localH2O, path = prosPath)
prostate.cbind <- h2o.rbind(prostate.hex, prostate.hex)
head(prostate.cbind)
```

---

h2o.removeAll

*Remove All Objects on the H2O Cluster*

---

**Description**

Removes the data from the h2o cluster, but does not remove the local references.

**Usage**

```
h2o.removeAll(conn = h2o.getConnection(), timeout_secs = 0)
```

**Arguments**

conn	An <a href="#">H2OConnection</a> object containing the IP address and port number of the H2O server.
timeout_secs	Timeout in seconds. Default is no timeout.

**See Also**

[h2o.rm](#)

**Examples**

```
library(h2o)
localH2O <- h2o.init()
prosPath <- system.file("extdata", "prostate.csv", package = "h2o")
prostate.hex <- h2o.uploadFile(localH2O, path = prosPath)
h2o.ls(localH2O)
h2o.removeAll(localH2O)
h2o.ls(localH2O)
```

---

h2o.removeVecs	<i>Delete Columns from a H2OFrame</i>
----------------	---------------------------------------

---

**Description**

Delete the specified columns from the H2OFrame. Returns a H2OFrame without the specified columns. This will trigger any lazy computation of the frame, and has side-effects.

**Usage**

```
h2o.removeVecs(data, cols)
```

**Arguments**

data	The H2OFrame.
cols	The columns to remove.

---

h2o.rep_len	<i>Replicate Elements of Vectors or Lists into H2O</i>
-------------	--

---

**Description**

h2o.rep performs just as rep does. It replicates the values in x in the H2O backend.

**Usage**

```
h2o.rep_len(x, length.out)
```

**Arguments**

x	a vector (of any mode including a list) or a factor
length.out	non negative integer. The desired length of the output vector.

**Value**

Creates a [H2OFrame](#) vector of the same type as x

---

h2o.residual\_deviance *Retrieve the residual deviance*

---

**Description**

Retrieve the residual deviance

**Usage**

```
h2o.residual_deviance(object, valid = FALSE, ...)
```

**Arguments**

object	An <a href="#">H2OModel</a> or <a href="#">H2OModelMetrics</a>
valid	Retrieve the validation metric.
...	further arguments to be passed to/from this method.

---

h2o.residual\_dof *Retrieve the residual degrees of freedom*

---

**Description**

Retrieve the residual degrees of freedom

**Usage**

```
h2o.residual_dof(object, valid = FALSE, ...)
```

**Arguments**

object	An <a href="#">H2OModel</a> or <a href="#">H2OModelMetrics</a>
valid	Retrieve the validation metric.
...	further arguments to be passed to/from this method.



---

`h2o.rm`*Delete Objects In H2O*

---

**Description**

Remove the h2o Big Data object(s) having the key name(s) from ids.

**Usage**

```
h2o.rm(ids, conn = h2o.getConnection())
```

**Arguments**

ids	The hex key associated with the object to be removed.
conn	An <a href="#">H2OConnection</a> object containing the IP address and port number of the H2O server.

**See Also**

[h2o.assign](#), [h2o.ls](#)

---

`h2o.runif`*Produe a Vector of Random Uniform Numbers*

---

**Description**

Creates a vector of random uniform numbers equal in length to the length of the specified H2O dataset.

**Usage**

```
h2o.runif(x, seed = -1)
```

**Arguments**

x	An <a href="#">H2OFrame</a> object.
seed	A random seed used to generate draws from the uniform distribution.

**Value**

A vector of random, uniformly distributed numbers. The elements are between 0 and 1.

## Examples

```
library(h2o)
localH2O = h2o.init()
prosPath = system.file("extdata", "prostate.csv", package="h2o")
prostate.hex = h2o.importFile(localH2O, path = prosPath, destination_frame = "prostate.hex")
s = h2o.runif(prostate.hex)
summary(s)

prostate.train = prostate.hex[s <= 0.8,]
prostate.train = h2o.assign(prostate.train, "prostate.train")
prostate.test = prostate.hex[s > 0.8,]
prostate.test = h2o.assign(prostate.test, "prostate.test")
nrow(prostate.train) + nrow(prostate.test)
```

---

h2o.saveModel

*Save an H2O Model Object to Disk*

---

## Description

Save an [H2OModel](#) to disk. Currently not implemented.

## Usage

```
h2o.saveModel(object, dir = "", name = "", filename = "", force = FALSE)
```

## Arguments

object	an <a href="#">H2OModel</a> object.
dir	string indicating the directory the model will be written to.
name	string name of the file.
filename	the full path to the file.
force	logical, indicates how to deal with files that already exist.

## Details

In the case of existing files `force = TRUE` will overwrite the file. Otherwise, the operation will fail.

## See Also

[h2o.loadModel](#) for loading a model to H2O from disk

**Examples**

```
## Not run:
# library(h2o)
# localH2O <- h2o.init()
# prostate.hex <- h2o.uploadFile(localH2O, path = paste("https://raw.github.com",
# "0xdata/h2o/master/smalldata/logreg/prostate.csv", sep = "/"),
# destination_frame = "prostate.hex")
# prostate.glm <- h2o.glm(y = "CAPSULE", x = c("AGE", "RACE", "PSA", "DCAPS"),
# training_frame = prostate.hex, family = "binomial", alpha = 0.5)
# h2o.saveModel(object = prostate.glm, dir = "/Users/UserName/Desktop", save_cv = TRUE,
# force = TRUE)

## End(Not run)
```

h2o.scale

*Scaling and Centering of an H2O Frame***Description**

Centers and/or scales the columns of an H2O dataset.

**Usage**

```
## S3 method for class H2OFrame
scale(x, center = TRUE, scale = TRUE)
```

**Arguments**

x	An <a href="#">H2OFrame</a> object.
center	either a logical value or numeric vector of length equal to the number of columns of x.
scale	either a logical value or numeric vector of length equal to the number of columns of x.

**Examples**

```
library(h2o)
localH2O <- h2o.init()
irisPath <- system.file("extdata", "iris_wheader.csv", package="h2o")
iris.hex <- h2o.uploadFile(localH2O, path = irisPath, destination_frame = "iris.hex")
summary(iris.hex)

# Scale and center all the numeric columns in iris data set
scale(iris.hex[, 1:4])
```

---

h2o.scoreHistory	<i>Retrieve Model Score History</i>
------------------	-------------------------------------

---

**Description**

Retrieve Model Score History

**Usage**

```
h2o.scoreHistory(object, ...)
```

**Arguments**

object	An <a href="#">H2OModel</a> object.
...	further arguments to be passed on (currently unimplemented)

---

h2o.sd	<i>Standard Deviation of a column of data.</i>
--------	--

---

**Description**

Obtain the standard deviation of a column of data.

**Usage**

```
## S4 method for signature H2OFrame
sd(x, na.rm = FALSE)
```

**Arguments**

x	An <a href="#">H2OFrame</a> object.
na.rm	logical. Should missing values be removed?

**See Also**

[h2o.var](#) for variance, and [sd](#) for the base R implementation.

**Examples**

```
localH2O <- h2o.init()
prosPath <- system.file("extdata", "prostate.csv", package="h2o")
prostate.hex <- h2o.uploadFile(localH2O, path = prosPath)
sd(prostate.hex$AGE)
```

---

h2o.setLevel	<i>Set a Factor Column to a Level</i>
--------------	---------------------------------------

---

**Description**

A method to set a factor column to one of the levels.

**Usage**

```
h2o.setLevel(x, level)
```

**Arguments**

x	a column from an <a href="#">H2OFrame</a> object.
level	The level at which the column will be set.

**Details**

Replace all other occurrences with 'level' in a factor column.

**Value**

An object of class [H2OFrame](#).

**Examples**

```
localH2O <- h2o.init()
hex <- as.h2o(localH2O, iris)
hex$Species <- h2o.setLevel(hex$Species, "versicolor")
```

---

h2o.setLevels	<i>Set Levels of H2O Factor Column</i>
---------------	--

---

**Description**

Works on a single categorical vector. New domains must be aligned with the old domains. This call has SIDE EFFECTS and mutates the column in place (does not make a copy).

**Usage**

```
h2o.setLevels(x, levels)
```

**Arguments**

x	A single categorical column.
levels	A character vector specifying the new levels. The number of new levels must match the number of old levels.

---

h2o.setTimezone	<i>Set the Time Zone on the H2O Cloud</i>
-----------------	---

---

**Description**

Set the Time Zone on the H2O Cloud

**Usage**

```
h2o.setTimezone(tz, conn = h2o.getConnection())
```

**Arguments**

tz	The desired timezone.
conn	An H2OConnection object.

---

h2o.shim	<i>Deprecated Script Shim</i>
----------	-------------------------------

---

**Description**

Due to the many improvements implemented in H2O-Dev and the differences in architecture between H2O and H2O-Dev, some parameters, options, and objects are no longer supported. To assist our legacy H2O users in upgrading their workflows for compatibility with H2O-Dev, we have developed the "Deprecated Script Shim" tool to detect deprecated parameters, options, and objects in H2O scripts being imported into H2O-Dev and suggest updated alternatives.

**Usage**

```
h2o.shim(enable = TRUE)
```

**Arguments**

enable	a logical value indicating whether the shim should be enabled or disabled.
--------	--

**See Also**

<https://github.com/h2oai/h2o-dev/blob/master/h2o-docs/src/product/upgrade/H2ODevPortingRScripts.md>, For more information on converting legacy H2O scripts so that they will run in H2O-Dev

---

h2o.shutdown	<i>Shut Down H2O Instance</i>
--------------	-------------------------------

---

**Description**

Shut down the specified instance. All data will be lost.

**Usage**

```
h2o.shutdown(conn = h2o.getConnection(), prompt = TRUE)
```

**Arguments**

conn	An <a href="#">H2OConnection</a> object containing the IP address and port of the server running H2O.
prompt	A logical value indicating whether to prompt the user before shutting down the H2O server.

**Details**

This method checks if H2O is running at the specified IP address and port, and if it is, shuts down that H2O instance.

**WARNING**

All data, models, and other values stored on the server will be lost! Only call this function if you and all other clients connected to the H2O server are finished and have saved your work.

**Note**

Users must call `h2o.shutdown` explicitly in order to shut down the local H2O instance started by R. If R is closed before H2O, then an attempt will be made to automatically shut down H2O. This only applies to local instances started with `h2o.init`, not remote H2O servers.

**See Also**

[h2o.init](#)

**Examples**

```
# Dont run automatically to prevent accidentally shutting down a cloud
## Not run:
library(h2o)
localH2O = h2o.init()
h2o.shutdown(localH2O)

## End(Not run)
```

---

h2o.splitFrame	<i>Split an H2O Data Set</i>
----------------	------------------------------

---

**Description**

Split an existing H2O data set according to user-specified ratios.

**Usage**

```
h2o.splitFrame(data, ratios = 0.75, destination_frames)
```

**Arguments**

data	An <a href="#">H2OFrame</a> object representing the dataset to split.
ratios	A numeric value or array indicating the ratio of total rows contained in each split. Must total up to less than 1.
destination_frames	An array of frame IDs equal to the number of ratios specified plus one.

**Examples**

```
library(h2o)
localH2O = h2o.init()
irisPath = system.file("extdata", "iris.csv", package = "h2o")
iris.hex = h2o.importFile(localH2O, path = irisPath)
iris.split = h2o.splitFrame(iris.hex, ratios = c(0.2, 0.5))
head(iris.split[[1]])
summary(iris.split[[1]])
```

---

h2o.startGLMJob	<i>Start an H2O Generalized Linear Model Job</i>
-----------------	--

---

**Description**

Creates a background H2O GLM job.

**Usage**

```
h2o.startGLMJob(x, y, training_frame, model_id, validation_frame,
  max_iterations = 50, beta_epsilon = 0, solver = c("IRLSM", "L_BFGS"),
  standardize = TRUE, family = c("gaussian", "binomial", "poisson", "gamma",
  "tweedie"), link = c("family_default", "identity", "logit", "log",
  "inverse", "tweedie"), tweedie_variance_power = NaN,
  tweedie_link_power = NaN, alpha = 0.5, prior = 0, lambda = 1e-05,
  lambda_search = FALSE, nlambdas = -1, lambda_min_ratio = 1,
  nfolds = 0, beta_constraints = NULL, ...)
```



**Arguments**

x	A vector containing the names or indices of the predictor variables to use in building the GLM model.
y	A character string or index that represent the response variable in the model.
training_frame	An <a href="#">H2OFrame</a> object containing the variables in the model.
model_id	(Optional) The unique id assigned to the resulting model. If none is given, an id will automatically be generated.
validation_frame	An <a href="#">H2OFrame</a> object containing the variables in the model.
max_iterations	A non-negative integer specifying the maximum number of iterations.
beta_epsilon	A non-negative number specifying the magnitude of the maximum difference between the coefficient estimates from successive iterations. Defines the convergence criterion for <code>h2o.glm</code> .
solver	A character string specifying the solver used: IRLSM (supports more features), L_BFGS (scales better for datasets with many columns)
standardize	A logical value indicating whether the numeric predictors should be standardized to have a mean of 0 and a variance of 1 prior to training the models.
family	A character string specifying the distribution of the model: gaussian, binomial, poisson, gamma, tweedie.
link	A character string specifying the link function. The default is the canonical link for the family. The supported links for each of the family specifications are: "gaussian": "identity", "log", "inverse" "binomial": "logit", "log" "poisson": "log", "identity" "gamma": "inverse", "log", "identity" "tweedie": "tweedie"
tweedie_variance_power	A numeric specifying the power for the variance function when family = "tweedie".
tweedie_link_power	A numeric specifying the power for the link function when family = "tweedie".
alpha	A numeric in [0, 1] specifying the elastic-net mixing parameter. The elastic-net penalty is defined to be: $P(\alpha, \beta) = (1 - \alpha)/2 \ \beta\ _2^2 + \alpha \ \beta\ _1 = \sum_j [(1 - \alpha)/2 \beta_j^2 + \alpha  \beta_j ]$ , making alpha = 1 the lasso penalty and alpha = 0 the ridge penalty.
prior	(Optional) A numeric specifying the prior probability of class 1 in the response when family = "binomial". The default prior is the observational frequency of class 1.
lambda	A non-negative shrinkage parameter for the elastic-net, which multiplies $P(\alpha, \beta)$ in the objective function. When lambda = 0, no elastic-net penalty is applied and ordinary generalized linear models are fit.

<code>lambda_search</code>	A logical value indicating whether to conduct a search over the space of lambda values starting from the lambda max, given lambda is interpreted as lambda min.
<code>nlambda</code>	The number of lambda values to use when <code>lambda_search = TRUE</code> .
<code>lambda_min_ratio</code>	Smallest value for lambda as a fraction of lambda.max. By default if the number of observations is greater than the the number of variables then <code>lambda_min_ratio = 0.0001</code> ; if the number of observations is less than the number of variables then <code>lambda_min_ratio = 0.01</code> .
<code>nfolds</code>	(Currently Unimplemented)
<code>beta_constraints</code>	A data.frame or H2OParsedData object with the columns ["names", "lower_bounds", "upper_bounds", "beta_given"], where each row corresponds to a predictor in the GLM. "names" contains the predictor names, "lower"/"upper_bounds", are the lower and upper bounds of beta, and "beta_given" is some supplied starting values for the
<code>...</code>	(Currently Unimplemented) coefficients.

**Value**

Returns a [H2OModelFuture](#) class object.

**See Also**

[h2o.getGLMModel](#) for resolving the H2OModelFuture object.

---

`h2o.startLogging`      *Start Writing H2O R Logs*

---

**Description**

Begin logging H2o R POST commands and error responses to local disk. Used primarily for debuggin purposes.

**Usage**

```
h2o.startLogging(file)
```

**Arguments**

`file`                    a character string name for the file, automatically generated

**See Also**

[h2o.stopLogging](#), [h2o.clearLog](#),                    [h2o.openLog](#)

**Examples**

```
library(h2o)
localH2O = h2o.init()
h2o.startLogging()
ausPath = system.file("extdata", "australia.csv", package="h2o")
australia.hex = h2o.importFile(localH2O, path = ausPath)
h2o.stopLogging()
```

---

h2o.stopLogging	<i>Stop Writing H2O R Logs</i>
-----------------	--------------------------------

---

**Description**

Halt logging of H2O R POST commands and error responses to local disk. Used primarily for debugging purposes.

**Usage**

```
h2o.stopLogging()
```

**See Also**

[h2o.startLogging](#), [h2o.clearLog](#), [h2o.openLog](#)

**Examples**

```
library(h2o)
localH2O = h2o.init()
h2o.startLogging()
ausPath = system.file("extdata", "australia.csv", package="h2o")
australia.hex = h2o.importFile(localH2O, path = ausPath)
h2o.stopLogging()
```

---

h2o.strsplit	<i>String Split</i>
--------------	---------------------

---

**Description**

String Split

**Usage**

```
h2o.strsplit(x, split)
```

**Arguments**

x	The column whose strings must be split.
split	The pattern to split on.

---

h2o.sub	<i>String Substitute</i>
---------	--------------------------

---

**Description**

Mutates the input. Changes the first occurrence of pattern with replacement.

**Usage**

```
h2o.sub(pattern, replacement, x, ignore.case = FALSE)
```

**Arguments**

pattern	The pattern to replace.
replacement	The replacement pattern.
x	The column on which to operate.
ignore.case	Case sensitive or not

---

h2o.summary	<i>Summarizes the columns of a H2O data frame.</i>
-------------	--

---

**Description**

A method for the [summary](#) generic. Summarizes the columns of an H2O data frame or subset of columns and rows using vector notation (e.g. dataset[row, col])

**Usage**

```
## S4 method for signature H2OFrame
summary(object, factors = 6L, ...)
```

**Arguments**

object	An <a href="#">H2OFrame</a> object.
factors	The number of factors to return in the summary. Default is the top 6.
...	Further arguments passed to or from other methods.

**Value**

A table displaying the minimum, 1st quartile, median, mean, 3rd quartile and maximum for each numeric column, and the levels and category counts of the levels in each categorical column.

**Examples**

```
library(h2o)
localH2O = h2o.init()
prosPath = system.file("extdata", "prostate.csv", package="h2o")
prostate.hex = h2o.importFile(localH2O, path = prosPath)
summary(prostate.hex)
summary(prostate.hex$GLEASON)
summary(prostate.hex[,4:6])
```

h2o.svd

*Singular Value Decomposition***Description**

Singular value decomposition of a H2O dataset using the power method.

**Usage**

```
h2o.svd(training_frame, x, nv, destination_key, max_iterations = 1000,
        transform = "NONE", seed)
```

**Arguments**

training_frame	An <a href="#">H2OFrame</a> object containing the variables in the model.
x	(Optional) A vector containing the data columns on which SVD operates.
nv	The number of right singular vectors to be computed. This must be between 1 and $\min(\text{ncol}(\text{training\_frame}), \text{nrow}(\text{training\_frame}))$ inclusive.
destination_key	(Optional) The unique hex key assigned to the resulting model. Automatically generated if none is provided.
max_iterations	The maximum number of iterations to run each power iteration loop. Must be between 1 and $1e6$ inclusive.
transform	A character string that indicates how the training data should be transformed before running PCA. Possible values are "NONE": for no transformation, "DEMEAN": for subtracting the mean of each column, "DESCALE": for dividing by the standard deviation of each column, "STANDARDIZE": for demeaning and descaling, and "NORMALIZE": for demeaning and dividing each column by its range (max - min).
seed	(Optional) Random seed used to initialize the right singular vectors at the beginning of each power method iteration.

**Value**

Returns an object of class [H2ODimReductionModel](#).

## Examples

```
library(h2o)
localH2O <- h2o.init()
ausPath <- system.file("extdata", "australia.csv", package="h2o")
australia.hex <- h2o.uploadFile(localH2O, path = ausPath)
h2o.svd(training_frame = australia.hex, nv = 8)
```

---

h2o.table

*Cross Tabulation and Table Creation in H2O*

---

## Description

Uses the cross-classifying factors to build a table of counts at each combination of factor levels.

## Usage

```
h2o.table(x, y = NULL)
```

## Arguments

x                    An [H2OFrame](#) object with at most two columns.  
y                    An [H2OFrame](#) similar to x, or NULL.

## Value

Returns a tabulated [H2OFrame](#) object.

## Examples

```
library(h2o)
localH2O <- h2o.init()
prosPath <- system.file("extdata", "prostate.csv", package="h2o")
prostate.hex <- h2o.uploadFile(localH2O, path = prosPath, destination_frame = "prostate.hex")
summary(prostate.hex)

# Counts of the ages of all patients
head(h2o.table(prostate.hex[,3]))
h2o.table(prostate.hex[,3])

# Two-way table of ages (rows) and race (cols) of all patients
head(h2o.table(prostate.hex[,c(3,4)]))
h2o.table(prostate.hex[,c(3,4)])
```

---

h2o.tolower	<i>To Lower</i>
-------------	-----------------

---

**Description**

Mutates the input!

**Usage**

```
h2o.tolower(x)
```

**Arguments**

x	An H2OFrame object whose strings should be lower'd
---	--

---

h2o.totss	<i>Get the total sum of squares.</i>
-----------	--------------------------------------

---

**Description**

Get the total sum of squares.

**Usage**

```
h2o.totss(object, valid = FALSE, ...)
```

**Arguments**

object	An <a href="#">H2OClusteringModel</a> object.
valid	Retrieve the validation metric.
...	further arguments to be passed on (currently unimplemented)

---

h2o.tot_withinss	<i>Get the total within cluster sum of squares.</i>
------------------	---

---

**Description**

Get the total within cluster sum of squares.

**Usage**

```
h2o.tot_withinss(object, valid = FALSE, ...)
```

**Arguments**

object	An <a href="#">H2OClusteringModel</a> object.
valid	Retrieve the validation metric.
...	further arguments to be passed on (currently unimplemented)

---

h2o.toupper	<i>To Upper</i>
-------------	-----------------

---

**Description**

Mutates the input!

**Usage**

```
h2o.toupper(x)
```

**Arguments**

x	An H2OFrame object whose strings should be upper'd
---	--

---

h2o.trim	<i>Trim Space</i>
----------	-------------------

---

**Description**

Trim Space

**Usage**

```
h2o.trim(x)
```

**Arguments**

x	The column whose strings should be trimmed.
---	---



---

h2o.var	<i>Variance of a column.</i>
---------	------------------------------

---

**Description**

Obtain the variance of a column of a parsed H2O data object.

**Usage**

```
## S4 method for signature H2OFrame
var(x, y = NULL, na.rm = FALSE, use)
```

**Arguments**

x	An <a href="#">H2OFrame</a> object.
y	NULL (default) or a column of an <a href="#">H2OFrame</a> object. The default is equivalent to y = x (but more efficient).
na.rm	logical. Should missing values be removed?
use	An optional character string to be used in the presence of missing values. This must be one of the following strings. "everything", "all.obs", or "complete.obs".

**See Also**

[var](#) for the base R implementation. [h2o.sd](#) for standard deviation.

**Examples**

```
localH2O <- h2o.init()
prosPath <- system.file("extdata", "prostate.csv", package="h2o")
prostate.hex <- h2o.uploadFile(localH2O, path = prosPath)
var(prostate.hex$AGE)
```

---

h2o.varimp	<i>Retrieve the variable importance.</i>
------------	--

---

**Description**

Retrieve the variable importance.

**Usage**

```
h2o.varimp(object, ...)
```

**Arguments**

object	An <a href="#">H2OModel</a> object.
...	further arguments to be passed on (currently unimplemented)

---

h2o.week	<i>Convert Milliseconds to Week of Week Year in H2O Datasets</i>
----------	--

---

**Description**

Converts the entries of a [H2OFrame](#) object from milliseconds to weeks of the week year (starting from 1).

**Usage**

```
h2o.week(x)
```

```
week(x)
```

```
## S3 method for class H2OFrame
```

```
week(x)
```

**Arguments**

x                    An [H2OFrame](#) object.

**Value**

A [H2OFrame](#) object containing the entries of x converted to weeks of the week year.

**See Also**

[h2o.month](#)

---

h2o.withinss	<i>Get the Within SS</i>
--------------	--------------------------

---

**Description**

Get the Within SS

**Usage**

```
h2o.withinss(object, ...)
```

**Arguments**

object                An [H2OClusteringModel](#) object.

...                    further arguments to be passed on (currently unimplemented)

---

h2o.year	<i>Convert Milliseconds to Years in H2O Datasets</i>
----------	--

---

**Description**

Convert the entries of a [H2OFrame](#) object from milliseconds to years, indexed starting from 1900.

**Usage**

```
h2o.year(x)
```

```
year(x)
```

```
## S3 method for class H2OFrame  
year(x)
```

**Arguments**

x                    An [H2OFrame](#) object.

**Details**

This method calls the function of the `MutableDateTime` class in Java.

**Value**

A [H2OFrame](#) object containig the entries of x converted to years starting from 1900, e.g. 69 corresponds to the year 1969.

**See Also**

[h2o.month](#)

---

H2OClusteringModel-class	<i>The H2OClusteringModel object.</i>
--------------------------	---------------------------------------

---

**Description**

This virtual class represents a clustering model built by H2O.

**Details**

This object has slots for the key, which is a character string that points to the model key existing in the H2O cloud, the data used to build the model (an object of class `H2OFrame`).

**Slots**

- conn** Object of class `H2OConnection`, which is the client object that was passed into the function call.
- model\_id** A character string specifying the key for the model fit in the H2O cloud's key-value store.
- finalizers** A list object containing environments with finalizers that remove keys from the H2O key-value store.
- algorithm** A character string specifying the algorithm that was used to fit the model.
- parameters** A list containing the parameter settings that were used to fit the model that differ from the defaults.
- allparameters** A list containing all parameters used to fit the model.
- model** A list containing the characteristics of the model returned by the algorithm.
- size** The number of points in each cluster.
  - totss** Total sum of squared error to grand mean.
  - withinss** A vector of within-cluster sum of squared error.
  - tot\_withinss** Total within-cluster sum of squared error.
  - betweenss** Between-cluster sum of squared error.
- finalizers** A list object containing environments with finalizers that remove keys from the H2O key-value store.

---

H2OConnection-class    *The H2OConnection class.*

---

**Description**

This class represents a connection to an H2O cloud.

**Usage**

```
## S4 method for signature H2OConnection
show(object)
```

**Arguments**

**object**            an `H2OConnection` object.

**Details**

Because H2O is not a master-slave architecture, there is no restriction on which H2O node is used to establish the connection between R (the client) and H2O (the server).

A new H2O connection is established via the `h2o.init()` function, which takes as parameters the 'ip' and 'port' of the machine running an instance to connect with. The default behavior is to connect with a local instance of H2O at port 54321, or to boot a new local instance if one is not found at port 54321.

**Slots**

- ip A character string specifying the IP address of the H2O cloud.
- port A numeric value specifying the port number of the H2O cloud.
- mutable An H2OConnectionMutableState object to hold the mutable state for the H2O connection.

---

H2OFrame-class	<i>The H2OFrame class</i>
----------------	---------------------------

---

**Description**

The H2OFrame class

**Usage**

```
## S4 method for signature H2OFrame
show(object)
```

**Arguments**

object An H2OConnection object.

**Slots**

- conn An H2OConnection object specifying the connection to an H2O cloud.
- frame\_id A character string specifying the identifier for the frame in the H2O cloud.
- finalizers A list object containing environments with finalizers that remove objects from the H2O cloud.
- mutable An H2OFrameMutableState object to hold the mutable state for the H2O frame.

---

H2OFrame-Extract	<i>Extract or Replace Parts of an H2OFrame Object</i>
------------------	---

---

**Description**

Operators to extract or replace parts of H2OFrame objects.

**Usage**

```
## S4 method for signature H2OFrame
x[i, j, ..., drop = TRUE]

## S4 method for signature H2OFrame
x$name

## S4 method for signature H2OFrame
x[[i, exact = TRUE]]

## S4 replacement method for signature H2OFrame
x[i, j, ...] <- value

## S4 replacement method for signature H2OFrame
x$name <- value

## S4 replacement method for signature H2OFrame
x[[i]] <- value
```

**Arguments**

x	object from which to extract element(s) or in which to replace element(s).
i, j, ...	indices specifying elements to extract or replace. Indices are numeric or character vectors or empty (missing) or will be matched to the names.
drop	a logical, whether or not to attempt to reduce dimensions to the lowest possible.
name	a literal character string or a name (possibly backtick quoted).
exact	controls possible partial matching of [[ when extracting a character
value	an array-like H2O object similar to x.

---

H2OModel-class

*The H2OModel object.*


---

**Description**

This virtual class represents a model built by H2O.

**Usage**

```
## S4 method for signature H2OModel
show(object)
```

**Arguments**

object	an H2OModel object.
--------	---------------------

**Details**

This object has slots for the key, which is a character string that points to the model key existing in the H2O cloud, the data used to build the model (an object of class H2OFrame).

**Slots**

conn Object of class H2OConnection, which is the client object that was passed into the function call.

model\_id A character string specifying the key for the model fit in the H2O cloud's key-value store.

finalizers A list object containing environments with finalizers that remove keys from the H2O key-value store.

algorithm A character string specifying the algorithm that were used to fit the model.

parameters A list containing the parameter settings that were used to fit the model that differ from the defaults.

allparameters A list containing all parameters used to fit the model.

model A list containing the characteristics of the model returned by the algorithm.

---

H2OModelFuture-class *H2O Future Model*

---

**Description**

A class to contain the information for background model jobs.

**Slots**

conn an [H2OConnection](#)

job\_key a character key representing the identification of the job process.

model\_id the final identifier for the model

**See Also**

[H2OModel](#) for the final model types.

---

H2OModelMetrics-class *The H2OModelMetrics Object.*

---

**Description**

A class for constructing performance measures of H2O models.

**Usage**

```
## S4 method for signature H2OModelMetrics
show(object)

## S4 method for signature H2OBinomialMetrics
show(object)

## S4 method for signature H2OMultinomialMetrics
show(object)

## S4 method for signature H2ORegressionMetrics
show(object)

## S4 method for signature H2OClusteringMetrics
show(object)

## S4 method for signature H2OAutoEncoderMetrics
show(object)
```

**Arguments**

object            An H2OModelMetrics object

---

H2OObject-class        *The H2OObject class*

---

**Description**

The H2OObject class

**Usage**

```
## S4 method for signature H2OObject
initialize(.Object, ...)
```



**Arguments**

`.Object` an H2Oobject  
`...` additional parameters to pass on to functions

**Slots**

`conn` An H2OConnection object specifying the connection to an H2O cloud.  
`id` A character string specifying the key in the H2O cloud's key-value store.  
`finalizers` A list object containing environments with finalizers that remove keys from the H2O key-value store.

---

H2ORawData-class      *The H2ORawData class.*

---

**Description**

This class represents data in a post-import format.

**Usage**

```
## S4 method for signature H2ORawData
show(object)
```

**Arguments**

`object` a H2ORawData object.

**Details**

Data ingestion is a two-step process in H2O. First, a given path to a data source is `_imported_` for validation by the user. The user may continue onto `_parsing_` all of the data into memory, or the user may choose to back out and make corrections. Imported data is in a staging area such that H2O is aware of the data, but the data is not yet in memory.

The H2ORawData is a representation of the imported, not yet parsed, data.

**Slots**

`conn` An H2OConnection object containing the IP address and port number of the H2O server.  
`frame_id` An object of class "character", which is the name of the key assigned to the imported data.  
`finalizers` A list object containing environments with finalizers that remove objects from the H2O cloud.

---

H2OS4groupGeneric      *S4 Group Generic Functions for H2O*

---

**Description**

Methods for group generic functions and H2O objects.

**Usage**

```
## S4 method for signature missing,H2OFrame  
Ops(e1, e2)  
  
## S4 method for signature H2OFrame,missing  
Ops(e1, e2)  
  
## S4 method for signature H2OFrame,H2OFrame  
Ops(e1, e2)  
  
## S4 method for signature numeric,H2OFrame  
Ops(e1, e2)  
  
## S4 method for signature H2OFrame,numeric  
Ops(e1, e2)  
  
## S4 method for signature H2OFrame,character  
Ops(e1, e2)  
  
## S4 method for signature character,H2OFrame  
Ops(e1, e2)  
  
## S4 method for signature H2OFrame  
Math(x)  
  
## S4 method for signature H2OFrame  
Math2(x, digits)  
  
## S4 method for signature H2OFrame  
Summary(x, ..., na.rm = FALSE)  
  
## S4 method for signature H2OFrame  
!x  
  
## S4 method for signature H2OFrame  
is.na(x)  
  
## S4 method for signature H2OFrame  
t(x)
```

```
## S4 method for signature H2OFrame
log(x, ...)

## S4 method for signature H2OFrame
trunc(x, ...)

## S4 method for signature H2OFrame,H2OFrame
x %*% y
```

### Arguments

<code>x,y,e1,e2</code>	objects.
<code>digits</code>	number of digits to be used in round or signif
<code>na.rm</code>	logical: should missing values be removed?
<code>...</code>	further arguments passed to or from methods

---

H2OW2V-class	<i>The H2OW2V object.</i>
--------------	---------------------------

---

### Description

This class represents a h2o-word2vec object.

---

<code>is.factor,H2OFrame-method</code>	<i>Is H2O Data Frame column a enum</i>
--	--

---

### Description

Is H2O Data Frame column a enum

### Usage

```
## S4 method for signature H2OFrame
is.factor(x)
```

### Arguments

<code>x</code>	an <a href="#">H2OFrame</a> object column.
----------------	--

### Value

Returns logical value.

---

median, H2OFrame-method

*H2O Median*

---

### Description

Compute the airthmetic mean of a [H2OFrame](#).

### Usage

```
## S4 method for signature H2OFrame
median(x, na.rm = TRUE)
```

### Arguments

x	An <a href="#">H2OFrame</a> object.
na.rm	a logical, indicating whether na's are omitted.

### Examples

```
localH2O <- h2o.init()
prosPath <- system.file("extdata", "prostate.csv", package="h2o")
prostate.hex <- h2o.uploadFile(localH2O, path = prosPath, destination_frame = "prostate.hex")
```

---

ModelAccessors

*Accessor Methods for H2OModel Object*

---

### Description

Function accessor methods for various H2O output fields.

### Usage

```
getParms(object)

## S4 method for signature H2OModel
getParms(object)

getCenters(object)

getCentersStd(object)

getWithinSS(object)

getTotWithinSS(object)
```

```

getBetweenSS(object)

getTotSS(object)

getIterations(object)

getClusterSizes(object)

## S4 method for signature H2OClusteringModel
getCenters(object)

## S4 method for signature H2OClusteringModel
getCentersStd(object)

## S4 method for signature H2OClusteringModel
getWithinSS(object)

## S4 method for signature H2OClusteringModel
getTotWithinSS(object)

## S4 method for signature H2OClusteringModel
getBetweenSS(object)

## S4 method for signature H2OClusteringModel
getTotSS(object)

## S4 method for signature H2OClusteringModel
getIterations(object)

## S4 method for signature H2OClusteringModel
getClusterSizes(object)

```

### Arguments

object            an [H2OModel](#) class object.

---

Node-class

*The Node class.*

---

### Description

An object of type Node inherits from an H2OFrame, but holds no H2O-aware data. Every node in the abstract syntax tree has as its ancestor this class.

This class represents an operator between one or more H2O objects. ASTApply nodes are always root nodes in a tree and are never leaf nodes. Operators are discussed more in depth in ops.R.

## Details

Every node in the abstract syntax tree will have a symbol table, which is a dictionary of types and names for all the relevant variables and functions defined in the current scope. A missing symbol is therefore discovered by looking up the tree to the nearest symbol table defining that symbol.

---

predict.H2OModel	<i>Predict on an H2O Model</i>
------------------	--------------------------------

---

## Description

Obtains predictions from various fitted H2O model objects.

## Usage

```
## S3 method for class H2OModel
predict(object, newdata, ...)

h2o.predict(object, newdata, ...)
```

## Arguments

object	a fitted <a href="#">H2OModel</a> object for which prediction is desired
newdata	A <a href="#">H2OFrame</a> object in which to look for variables with which to predict.
...	additional arguments to pass on.

## Details

This method dispatches on the type of H2O model to select the correct prediction/scoring algorithm.

## Value

Returns an [H2OFrame](#) object with probabilities and default predictions.

## See Also

[link{h2o.deepLearning}](#), [link{h2o.gbm}](#), [link{h2o.glm}](#), [link{h2o.randomForest}](#) for model generation in h2o.

---

print.H2OTable	<i>Print method for H2OTable objects</i>
----------------	--

---

**Description**

This will print a truncated view of the table if there are more than 20 rows.

**Usage**

```
## S3 method for class H2OTable
print(x, header = TRUE, ...)
```

**Arguments**

x	An H2OTable object
header	A logical value dictating whether or not the table name should be printed.
...	Further arguments passed to or from other methods.

**Value**

The original x object

---

quantile	<i>Quantiles of H2O Data Frame.</i>
----------	-------------------------------------

---

**Description**

Obtain and display quantiles for H2O parsed data.

**Usage**

```
## S3 method for class H2OFrame
quantile(x, probs = c(0.001, 0.01, 0.1, 0.25, 0.333, 0.5,
  0.667, 0.75, 0.9, 0.99, 0.999), combine_method = c("interpolate", "average",
  "avg", "low", "high"), ...)
```

**Arguments**

x	An <a href="#">H2OFrame</a> object with a single numeric column.
probs	Numeric vector of probabilities with values in [0,1].
combine_method	How to combine quantiles for even sample sizes. Default is to do linear interpolation. E.g., If method is "lo", then it will take the lo value of the quantile. Abbreviations for average, low, and high are acceptable (avg, lo, hi).
...	Further arguments passed to or from other methods.

**Details**

quantile.H2OFrame, a method for the [quantile](#) generic. Obtain and return quantiles for an [H2OFrame](#) object.

**Value**

A vector describing the percentiles at the given cutoffs for the [H2OFrame](#) object.

**Examples**

```
# Request quantiles for an H2O parsed data set:
library(h2o)
localH2O <- h2o.init()
prosPath <- system.file("extdata", "prostate.csv", package="h2o")
prostate.hex <- h2o.uploadFile(localH2O, path = prosPath)
# Request quantiles for a subset of columns in an H2O parsed data set
quantile(prostate.hex[,3])
for(i in 1:ncol(prostate.hex))
  quantile(prostate.hex[,i])
```

---

sapply,H2OFrame-method

*Apply Over a List in H2O*

---

**Description**

Functions equivalent to the default R `sapply`

**Usage**

```
## S4 method for signature H2OFrame
sapply(X, FUN, ..., simplify = TRUE, USE.NAMES = TRUE)
```

**Arguments**

X	an <a href="#">H2OFrame</a> object on which <code>sapply</code> will operate.
FUN	the function to be applied.
simplify, USE.NAMES	ignored parameters from base function
...	optional arguments to FUN.

**See Also**

`link[base]{sapply}` for the base implementation. `export`



---

str.H2OFrame	<i>Describe an H2OFrame object</i>
--------------	------------------------------------

---

**Description**

Describe an H2OFrame object

**Usage**

```
## S3 method for class H2OFrame  
str(object, cols = FALSE, ...)
```

**Arguments**

object	An H2OFrame object.
cols	Logical indicating whether or not to do the str for all columns.
...	Extra args

---

summary,H2OModel-method	<i>Print the Model Summary</i>
-------------------------	--------------------------------

---

**Description**

Print the Model Summary

**Usage**

```
## S4 method for signature H2OModel  
summary(object, ...)
```

**Arguments**

object	An <a href="#">H2OModel</a> object.
...	further arguments to be passed on (currently unimplemented)

---

transform.H2OFrame      *Transform Columns in an H2OFrame Object.*

---

### Description

Functions that facilitate column transformations of an [H2OFrame](#) object.

### Usage

```
## S3 method for class H2OFrame
transform(_data, ...)

## S3 method for class H2OFrame
within(data, expr, ...)
```

### Arguments

<code>_data, data</code>	An <a href="#">H2OFrame</a> object.
<code>...</code>	For transform method, column transformations in the form tag=value.
<code>expr</code>	For within method, column transformations specified as an expression.

### See Also

[transform](#), [within](#) for the base R methods.

### Examples

```
library(h2o)
localH2O <- h2o.init()
iris.hex <- as.h2o(iris, localH2O)
transformed1 <- transform(iris.hex,
                          Sepal.Ratio = Sepal.Length / Sepal.Width,
                          Petal.Ratio = Petal.Length / Petal.Width )

transformed1
transformed2 <- within(iris.hex,
                      {Sepal.Product <- Sepal.Length * Sepal.Width
                       Petal.Product <- Petal.Length * Petal.Width
                       Sepal.Petal.Ratio <- Sepal.Product / Petal.Product
                       Sepal.Length <- Sepal.Width <- NULL
                       Petal.Length <- Petal.Width <- NULL
                      })

transformed2
```

# Index

- !,H2OFrame-method (H2OS4groupGeneric),  
106
- \*Topic **package**
  - h2o-package, 4
- [,H2OFrame-method (H2OFrame-Extract),  
101
- [<-,H2OFrame-method (H2OFrame-Extract),  
101
- [[,H2OFrame-method (H2OFrame-Extract),  
101
- [[<-,H2OFrame-method  
(H2OFrame-Extract), 101
- \$,H2OFrame-method (H2OFrame-Extract),  
101
- \$<-,H2OFrame-method (H2OFrame-Extract),  
101
- %\*%,H2OFrame,H2OFrame-method  
(H2OS4groupGeneric), 106
- %in%,H2OFrame-method (h2o.match), 62
  
- aaa, 5
- apply, 6
- apply,H2OFrame-method, 6
- as.character,H2OFrame-method, 7
- as.data.frame.H2OFrame, 7
- as.environment,H2OFrame-method, 8
- as.factor,H2OFrame-method, 8
- as.h2o, 9
- as.matrix, 10
- as.matrix.h2o, 9
- as.matrix.H2OFrame (as.matrix.h2o), 9
- as.numeric,H2OFrame-method, 10
- ASTApply-class (Node-class), 109
- ASTBody-class (Node-class), 109
- ASTElse-class (Node-class), 109
- ASTEmpty-class (Node-class), 109
- ASTFor-class (Node-class), 109
- ASTFun-class (Node-class), 109
- ASTIf-class (Node-class), 109
- ASTNode (ASTNode-class), 11
  
- ASTNode-class, 11
- ASTReturn-class (Node-class), 109
- ASTSeries-class (Node-class), 109
- ASTSpan-class (Node-class), 109
  
- cbind, 17
- colnames, 12
- colnames,H2OFrame-method  
(colnames<- ,H2OFrame,H2OFrame-method),  
11
- colnames<- ,H2OFrame,character-method  
(colnames<- ,H2OFrame,H2OFrame-method),  
11
- colnames<- ,H2OFrame,H2OFrame-method,  
11
- cut.H2OFrame, 12
  
- day (h2o.day), 24
- dayOfWeek (h2o.dayOfWeek), 25
- ddply, 26
- dim, 31, 70
- dim,H2OFrame-method (h2o.dim), 31
  
- getBetweenSS (ModelAccessors), 108
- getBetweenSS,H2OClusteringModel-method  
(ModelAccessors), 108
- getCenters (ModelAccessors), 108
- getCenters,H2OClusteringModel-method  
(ModelAccessors), 108
- getCentersStd (ModelAccessors), 108
- getCentersStd,H2OClusteringModel-method  
(ModelAccessors), 108
- getClusterSizes (ModelAccessors), 108
- getClusterSizes,H2OClusteringModel-method  
(ModelAccessors), 108
- getIterations (ModelAccessors), 108
- getIterations,H2OClusteringModel-method  
(ModelAccessors), 108
- getParms (ModelAccessors), 108

- getParms, H2OModel-method (ModelAccessors), 108
- getTotSS (ModelAccessors), 108
- getTotSS, H2OClusteringModel-method (ModelAccessors), 108
- getTotWithinSS (ModelAccessors), 108
- getTotWithinSS, H2OClusteringModel-method (ModelAccessors), 108
- getWithinSS (ModelAccessors), 108
- getWithinSS, H2OClusteringModel-method (ModelAccessors), 108
  
- h2o (h2o-package), 4
- h2o-package, 4
- h2o.accuracy (h2o.metric), 64
- h2o.aic, 13
- h2o.anomaly, 14
- h2o.anyFactor, 14
- h2o.assign, 15, 81
- h2o.auc, 15, 39, 42, 66, 67
- h2o.betweenness, 16, 57
- h2o.cbind, 17
- h2o.centers, 17, 57
- h2o.centersSTD, 18, 57
- h2o.clearLog, 18, 72, 90, 91
- h2o.cluster\_sizes, 20, 57
- h2o.clusterInfo, 19
- h2o.clusterIsUp, 19
- h2o.clusterStatus, 20
- h2o.coef, 21
- h2o.coef\_norm, 21
- h2o.colnames (colnames<- , H2OFrame, H2OFrame-method), 11
- h2o.confusionMatrix, 21, 42
- h2o.confusionMatrix, H2OModel-method (h2o.confusionMatrix), 21
- h2o.confusionMatrix, H2OModelMetrics-method (h2o.confusionMatrix), 21
- h2o.createFrame, 22
- h2o.day, 24, 25, 47
- h2o.dayOfWeek, 25
- h2o.ddply, 25
- h2o.deepfeatures, 26
- h2o.deeplearning, 14, 27
- h2o.dim, 31
- h2o.download\_pojo, 33
- h2o.downloadAllLogs, 31
- h2o.downloadCSV, 32
- h2o.error (h2o.metric), 64
- h2o.exportFile, 33
- h2o.exportHDFS, 34
- h2o.F0point5 (h2o.metric), 64
- h2o.F1 (h2o.metric), 64
- h2o.F2 (h2o.metric), 64
- h2o.fallout (h2o.metric), 64
- h2o.filterNACols, 35
- h2o.fnr (h2o.metric), 64
- h2o.fpr (h2o.metric), 64
- h2o.gbm, 35
- h2o.getConnection, 37
- h2o.getFrame, 37
- h2o.getGLMModel, 38, 90
- h2o.getModel, 38
- h2o.getTimezone, 39
- h2o.giniCoef, 16, 39, 39, 42, 66
- h2o.glm, 5, 40
- h2o.glmr, 43
- h2o.group\_by, 44
- h2o.gsub, 45
- h2o.head, 45
- h2o.hist, 46
- h2o.hit\_ratio\_table, 47
- h2o.hour, 47
- h2o.ifelse, 48
- h2o.importFile, 49
- h2o.importFolder (h2o.importFile), 49
- h2o.importHDFS (h2o.importFile), 49
- h2o.importURL (h2o.importFile), 49
- h2o.impute, 50
- h2o.init, 20, 51, 87
- h2o.insertMissingValues, 54
- h2o.interaction, 54
- h2o.killMinus3, 56
- h2o.kmeans, 56
- h2o.length, 57
- h2o.levels, 58
- h2o.listTimezones, 59
- h2o.loadModel, 59, 82
- h2o.logAndEcho, 60
- h2o.logloss, 42, 61
- h2o.ls, 61, 81
- h2o.makeGLMModel, 62
- h2o.match, 62
- h2o.maxPerClassError (h2o.metric), 64
- h2o.mcc (h2o.metric), 64
- h2o.mean, 63

- h2o.merge, 64
- h2o.metric, 16, 39, 64, 67
- h2o.missrate (h2o.metric), 64
- h2o.month, 24, 25, 66, 98, 99
- h2o.mse, 16, 42, 66, 67, 67
- h2o.naiveBayes, 68
- h2o.networkTest, 69
- h2o.nlevels, 69
- h2o.nrow, 70
- h2o.null\_deviance, 70
- h2o.null\_dof, 71
- h2o.num\_iterations, 57, 71
- h2o.openLog, 18, 72, 90, 91
- h2o.parseRaw, 72
- h2o.parseSetup, 73
- h2o.performance, 16, 22, 39, 42, 66, 67, 74
- h2o.prcomp, 74
- h2o.precision (h2o.metric), 64
- h2o.predict (predict.H2OModel), 110
- h2o.r2, 75
- h2o.randomForest, 76
- h2o.rbind, 77
- h2o.recall (h2o.metric), 64
- h2o.removeAll, 78
- h2o.removeVecs, 79
- h2o.rep\_len, 79
- h2o.residual\_deviance, 80
- h2o.residual\_dof, 80
- h2o.rm, 78, 81
- h2o.runif, 81
- h2o.saveModel, 59, 82
- h2o.scale, 83
- h2o.scoreHistory, 42, 84
- h2o.sd, 84, 97
- h2o.sensitivity (h2o.metric), 64
- h2o.setLevel, 85
- h2o.setLevels, 85
- h2o.setTimezone, 86
- h2o.shim, 86
- h2o.shutdown, 53, 87
- h2o.specificity (h2o.metric), 64
- h2o.splitFrame, 88
- h2o.startGLMJob, 88
- h2o.startLogging, 18, 72, 90, 91
- h2o.stopLogging, 18, 72, 90, 91
- h2o.strsplit, 91
- h2o.sub, 92
- h2o.summary, 92
- h2o.svd, 93
- h2o.table, 94
- h2o.tnr (h2o.metric), 64
- h2o.tolower, 95
- h2o.tot\_withinss, 57, 96
- h2o.totss, 57, 95
- h2o.toupper, 96
- h2o.tpr (h2o.metric), 64
- h2o.trim, 96
- h2o.uploadFile (h2o.importFile), 49
- h2o.var, 84, 97
- h2o.varimp, 42, 97
- h2o.week, 98
- h2o.withinss, 57, 98
- h2o.year, 67, 99
- H2OAutoEncoderMetrics-class
  - (H2OModelMetrics-class), 104
- H2OAutoEncoderModel, 14
- H2OAutoEncoderModel-class
  - (H2OModel-class), 102
- H2OBinomialMetrics, 15, 16, 22, 39, 61, 66, 67
- H2OBinomialMetrics-class
  - (H2OModelMetrics-class), 104
- H2OBinomialModel, 42, 69
- H2OBinomialModel-class
  - (H2OModel-class), 102
- H2OClusteringMetrics-class
  - (H2OModelMetrics-class), 104
- H2OClusteringModel, 16–18, 20, 57, 71, 95, 96, 98
- H2OClusteringModel-class, 99
- H2OConnection, 9, 20, 23, 32, 37, 38, 49, 56, 59–61, 69, 78, 81, 87, 103
- H2OConnection (H2OConnection-class), 100
- H2OConnection-class, 100
- H2ODimReductionMetrics-class
  - (H2OModelMetrics-class), 104
- H2ODimReductionModel, 75, 93
- H2ODimReductionModel-class
  - (H2OModel-class), 102
- H2OFrame, 6–15, 17, 22–28, 31–36, 40, 43–47, 49, 54, 55, 57, 58, 62–64, 66, 68, 70, 73–77, 79, 81, 83–85, 88, 89, 92–94, 97–99, 107, 108, 110–112, 114
- H2OFrame (H2OFrame-class), 101
- H2OFrame-class, 101
- H2OFrame-Extract, 101

- H2OModel, [13](#), [16](#), [21](#), [22](#), [26](#), [34](#), [38](#), [39](#), [42](#), [47](#), [59](#), [61](#), [62](#), [67](#), [70](#), [71](#), [74](#), [76](#), [77](#), [80](#), [82](#), [84](#), [97](#), [103](#), [109](#), [110](#), [113](#)
- H2OModel (H2OModel-class), [102](#)
- H2OModel-class, [102](#)
- H2OModelFuture, [38](#), [90](#)
- H2OModelFuture-class, [103](#)
- H2OModelMetrics, [13](#), [22](#), [61](#), [65](#), [67](#), [70](#), [71](#), [74](#), [80](#)
- H2OModelMetrics (H2OModelMetrics-class), [104](#)
- H2OModelMetrics-class, [104](#)
- H2OMultinomialMetrics, [22](#), [61](#), [67](#)
- H2OMultinomialMetrics-class (H2OModelMetrics-class), [104](#)
- H2OMultinomialModel, [69](#)
- H2OMultinomialModel-class (H2OModel-class), [102](#)
- H2OObject (H2OObject-class), [104](#)
- H2OObject-class, [104](#)
- H2ORawData, [49](#), [73](#)
- H2ORawData (H2ORawData-class), [105](#)
- H2ORawData-class, [105](#)
- H2ORegressionMetrics, [67](#)
- H2ORegressionMetrics-class (H2OModelMetrics-class), [104](#)
- H2ORegressionModel, [42](#)
- H2ORegressionModel-class (H2OModel-class), [102](#)
- H2OS4groupGeneric, [106](#)
- H2OUnknownMetrics-class (H2OModelMetrics-class), [104](#)
- H2OUnknownModel-class (H2OModel-class), [102](#)
- H2OW2V (H2OW2V-class), [107](#)
- H2OW2V-class, [107](#)
- head, H2OFrame-method (h2o.head), [45](#)
- hour (h2o.hour), [47](#)
- ifelse, ANY, H2OFrame, H2OFrame-method (h2o.ifelse), [48](#)
- ifelse, H2OFrame, ANY, ANY-method (h2o.ifelse), [48](#)
- initialize, H2OObject-method (H2OObject-class), [104](#)
- is.factor, [8](#)
- is.factor, H2OFrame-method, [107](#)
- is.na, H2OFrame-method (H2OS4groupGeneric), [106](#)
- length, [58](#)
- length, H2OFrame-method (h2o.length), [57](#)
- levels, [58](#)
- log, H2OFrame-method (H2OS4groupGeneric), [106](#)
- match, [62](#)
- match, H2OFrame-method (h2o.match), [62](#)
- Math, H2OFrame-method (H2OS4groupGeneric), [106](#)
- Math2, H2OFrame-method (H2OS4groupGeneric), [106](#)
- mean, [63](#)
- mean, H2OFrame-method (h2o.mean), [63](#)
- median, H2OFrame-method, [108](#)
- ModelAccessors, [108](#)
- month (h2o.month), [66](#)
- names, H2OFrame-method (colnames<-, H2OFrame, H2OFrame-method), [11](#)
- names<-, H2OFrame-method (colnames<-, H2OFrame, H2OFrame-method), [11](#)
- ncol, H2OFrame-method (h2o.ncol), [70](#)
- Node (Node-class), [109](#)
- Node-class, [109](#)
- nrow, [70](#)
- nrow, H2OFrame-method (h2o.nrow), [70](#)
- Ops, character, H2OFrame-method (H2OS4groupGeneric), [106](#)
- Ops, H2OFrame, character-method (H2OS4groupGeneric), [106](#)
- Ops, H2OFrame, H2OFrame-method (H2OS4groupGeneric), [106](#)
- Ops, H2OFrame, missing-method (H2OS4groupGeneric), [106](#)
- Ops, H2OFrame, numeric-method (H2OS4groupGeneric), [106](#)
- Ops, missing, H2OFrame-method (H2OS4groupGeneric), [106](#)
- Ops, numeric, H2OFrame-method (H2OS4groupGeneric), [106](#)
- predict, [22](#)
- predict.H2OModel, [30](#), [36](#), [42](#), [77](#), [110](#)
- print.H2OTable, [111](#)
- quantile, [111](#), [112](#)

`rbind`, 78

`sapply`, `H2OFrame`-method, 112

`scale.H2OFrame` (`h2o.scale`), 83

`sd`, 84

`sd`, `H2OFrame`-method (`h2o.sd`), 84

`show`, `ASTNode`-method (`ASTNode`-class), 11

`show`, `H2OAutoEncoderMetrics`-method (`H2OModelMetrics`-class), 104

`show`, `H2OBinomialMetrics`-method (`H2OModelMetrics`-class), 104

`show`, `H2OClusteringMetrics`-method (`H2OModelMetrics`-class), 104

`show`, `H2OConnection`-method (`H2OConnection`-class), 100

`show`, `H2OFrame`-method (`H2OFrame`-class), 101

`show`, `H2OModel`-method (`H2OModel`-class), 102

`show`, `H2OModelMetrics`-method (`H2OModelMetrics`-class), 104

`show`, `H2OMultinomialMetrics`-method (`H2OModelMetrics`-class), 104

`show`, `H2ORawData`-method (`H2ORawData`-class), 105

`show`, `H2ORegressionMetrics`-method (`H2OModelMetrics`-class), 104

`str.H2OFrame`, 113

`summary`, 92

`Summary`, `H2OFrame`-method (`H2OS4groupGeneric`), 106

`summary`, `H2OFrame`-method (`h2o.summary`), 92

`summary`, `H2OModel`-method, 113

`t`, `H2OFrame`-method (`H2OS4groupGeneric`), 106

`tail`, `H2OFrame`-method (`h2o.head`), 45

`transform`, 114

`transform.H2OFrame`, 114

`trunc`, `H2OFrame`-method (`H2OS4groupGeneric`), 106

`var`, 97

`var`, `H2OFrame`-method (`h2o.var`), 97

`week` (`h2o.week`), 98

`within`, 114

`within.H2OFrame` (`transform.H2OFrame`), 114

`year` (`h2o.year`), 99