Data Manipulation

H2OFrame

class `h2o.H2OFrame` ***(python_obj=None, destination_frame=None, header=0, separator=', ', column_names=None, column_types=None, na_strings=None, skipped_columns=None)*** [source]

Primary data store for H2O.

H2OFrame is similar to pandas' `DataFrame`, or R's `data.frame`. One of the critical distinction is that the data is generally not held in memory, instead it is located on a (possibly remote) H2O cluster, and thus `H2OFrame` represents a mere handle to that data.

Create a new H2OFrame object, possibly from some other object.

Parameters

- **python_obj** –

  object that will be converted to an `H2OFrame`. This could have multiple types:

  - None: create an empty H2OFrame
  - A list/tuple of strings or numbers: create a single-column H2OFrame containing the contents of this list.
  - A dictionary of `{name: list}` pairs: create an H2OFrame with multiple columns, each column having the provided `name` and contents from `list`. If the source dictionary is not an OrderedDict, then the columns in the H2OFrame may appear shuffled.
  - A list of lists of strings/numbers: construct an H2OFrame from a rectangular table of values, with inner lists treated as rows of the table. I.e. `H2OFrame([[1, 'a'], [2, 'b'], [3, 'c']])` will create a frame with 3 rows and 2 columns, one numeric and one string.
  - A Pandas dataframe, or a Numpy ndarray: create a matching H2OFrame.
  - A Scipy sparse matrix: create a matching sparse H2OFrame.

- **header** (*int*) – if `python_obj` is a list of lists, this parameter can be used to indicate whether the first row of the data represents headers. The value of -1 means the first row is data, +1 means the first row is the headers, 0 (default) allows H2O to guess whether the first row contains data or headers.
- **column_names** (*List[str]*) – explicit list of column names for the new H2OFrame. This will override any column names derived from the data. If the python_obj does not contain explicit column names, and this parameter is not given, then the

columns will be named “C1”, “C2”, “C3”, etc.

- **column_types** – explicit column types for the new H2OFrame. This could be either a list of types for each column, or a dictionary of {column name: column type} pairs. In the latter case you may override types for only few columns, and let H2O choose the types of the rest.
- **na_strings** – List of strings in the input data that should be interpreted as missing values. This could be given on a per-column basis, either as a list-of-lists, or as a dictionary {column name: list of nas}.
- **destination_frame** (*str*) – (internal) name of the target DKV key in the H2O backend.
- **separator** (*str*) – (deprecated)

Example

```
>>> python_obj = [1, 2, 2.5, -100.9, 0]
>>> frame = h2o.H2OFrame(python_obj)
>>> frame
```

abs () [source]

Calculate the absolute value of the current frame.

Returns

new H2OFrame equal to elementwise absolute value of the current frame.

Examples

```
>>> python_obj = [1, 2, 2.5, -100.9, 0]
>>> frame = h2o.H2OFrame(python_obj)
>>> abs(frame)
```

acos () [source]

Create a new H2OFrame equal to elementwise arc cosine of the current frame.

Returns

New H2OFrame equal to elementwise arc cosine of the current frame.

Examples

```
>>> python_obj = [1, 2, 2.5, -100.9, 0]
>>> frame = h2o.H2OFrame(python_obj)
>>> frame.acos()
```

acosh () [source]

Create a new H2OFrame equal to elementwise inverse hyperbolic cosine of the current frame

Returns

New H2OFrame equal to elementwise inverse hyperbolic cosine of the current frame.

Examples

```
>>> python_obj = [1, 2, 2.5, -100.9, 0]
>>> frame = h2o.H2OFrame(python_obj)
>>> frame.acosh()
```

all () [source]

Determine whether every element in the frame is either True, non-zero, or NA.

Returns

(bool) True if every element in the frame is either True, non-zero or NA.

Examples

```
>>> python_obj = [1,2,2.5,-100.9,0]
>>> frame = h2o.H2OFrame(python_obj)
>>> frame.all()
```

any () [source]

Determine whether any element in the frame is either True, non-zero, or NA.

Returns

(bool) True if any element in the frame is either True, non-zero or NA.

Examples

```
>>> python_obj = [1,2,2.5,-100.9,0]
>>> frame = h2o.H2OFrame(python_obj)
>>> frame.any()
```

any_na_rm () [source]

Determine whether any value in the frame is non-zero.

Returns

(bool) True if any value in the frame is non-zero (disregarding all NAs).

Example

```
>>> python_obj = [1,2,2.5,-100.9,0]
>>> frame = h2o.H2OFrame(python_obj)
>>> frame.any_na_rm()
```

`anyfactor` () [source]

Determine if there are any categorical columns in the frame.

Returns

(bool) True if there are any categorical columns in the frame.

Examples

```
>>> python_obj = [1,2,2.5,-100.9,0]
>>> frame = h2o.H2OFrame(python_obj)
>>> frame.anyfactor()
```

`apply` (*fun=None*, *axis=0*) [source]

Apply a lambda expression to an H2OFrame.

Parameters

- **fun** – a lambda expression to be applied per row or per column.
- **axis** – 0 = apply to each column; 1 = apply to each row

Returns

a new H2OFrame with the results of applying `fun` to the current frame.

Examples

```
>>> python_lists = [[1,2,3,4], [1,2,3,4]]
>>> h2oframe = h2o.H2OFrame(python_obj=python_lists,
...                         na_strings=['NA'])
>>> colMean = h2oframe.apply(lambda x: x.mean(), axis=0)
>>> rowMean = h2oframe.apply(lambda x: x.mean(), axis=1)
>>> colMean
>>> rowMean
```

`as_data_frame` (*use_pandas=True*, *header=True*) [source]

Obtain the dataset as a python-local object.

Parameters

- **use_pandas** (*bool*) – If True (default) then return the H2OFrame as a pandas DataFrame (requires that the `pandas` library was installed). If False, then return the contents of the H2OFrame as plain nested list, in a row-wise order.
- **header** (*bool*) – If True (default), then column names will be appended as the first row in list

Returns

A python object (a list of lists of strings, each list is a row, if use_pandas=False, otherwise a pandas DataFrame) containing this H2OFrame instance's data.

Examples

```
>>> airlines= h2o.import_file("https://s3.amazonaws.com/h2o-public-test-
data/smalldata/airlines/allyears2k_headers.zip")
>>> airlines["Year"]= airlines["Year"].asfactor()
>>> airlines["Month"]= airlines["Month"].asfactor()
>>> airlines["DayOfWeek"] = airlines["DayOfWeek"].asfactor()
>>> airlines["Cancelled"] = airlines["Cancelled"].asfactor()
>>> airlines['FlightNum'] = airlines['FlightNum'].asfactor()
>>> df = airlines.as_data_frame()
>>> df
```

`as_date` (*format*) [source]

Convert the frame (containing strings / categoricals) into the `date` format.

Parameters

format (*str*) – the format string (e.g. "%Y-%m-%d")

Returns

new H2OFrame with "int" column types

Examples

```
>>> hdf = h2o.import_file("https://s3.amazonaws.com/h2o-public-test-
data/smalldata/jira/v-11-eurodate.csv")
>>> hdf["ds5"].as_date("%d.%m.%y %H:%M")
```

`ascharacter` () [source]

Convert all columns in the frame into strings.

Returns

new H2OFrame with columns of “string” type.

Examples

```
>>> h2o = h2o.import_file("https://s3.amazonaws.com/h2o-public-test-
data/smalldata/junit/cars_20mpg.csv")
>>> h2o['cylinders'] = h2o['cylinders'].asfactor()
>>> h2o['cylinders'].ascharacter()
```

asfactor () [source]

Convert columns in the current frame to categoricals.

Returns

new H2OFrame with columns of the “enum” type.

Examples

```
>>> h2o = h2o.import_file("https://s3.amazonaws.com/h2o-public-test-
data/smalldata/junit/cars_20mpg.csv")
>>> h2o['cylinders'] = h2o['cylinders'].asfactor()
>>> h2o['cylinders']
```

asin () [source]

Create a new H2OFrame equal to elementwise arc sine of the current frame.

Returns

New H2OFrame equal to elementwise arc sine of the current frame.

Examples

```
>>> python_obj = [1, 2, 2.5, -100.9, 0]
>>> frame = h2o.H2OFrame(python_obj)
>>> frame.asin()
```

asinh () [source]

Create a new H2OFrame equal to elementwise inverse hyperbolic sine of the current frame.

Returns

New H2OFrame equal to elementwise inverse hyperbolic sine of the current frame.

Examples

```
>>> python_obj = [1, 2, 2.5, -100.9, 0]
>>> frame = h2o.H2OFrame(python_obj)
>>> frame.asinh()
```

asnumeric () [source]

Create a new frame with all columns converted to numeric.

Returns

New frame with all columns converted to numeric.

Examples

```
>>> cars = h2o.import_file("https://s3.amazonaws.com/h2o-public-test-
data/smalldata/junit/cars_20mpg.csv")
>>> cars.asnumeric()
```

atan () [source]

Create a new H2OFrame equal to elementwise arc tangent of the current frame.

Returns

New H2OFrame equal to elementwise arc tangent of the current frame.

Examples

```
>>> python_obj = [1,2,2.5,-100.9,0]
>>> frame = h2o.H2OFrame(python_obj)
>>> frame.atan()
```

atanh () [source]

Create a new H2OFrame equal to elementwise inverse hyperbolic tangent of the current frame.

Returns

New H2OFrame equal to elementwise inverse hyperbolic tangent of the current frame.

Examples

```
>>> python_obj = [1, 2, 2.5, -100.9, 0]
>>> frame = h2o.H2OFrame(python_obj)
>>> frame.atanh()
```

bottomN (*column=0*, *nPercent=10*) [source]

Given a column name or one column index, a percent N, this function will return the bottom N% of the values of the column of a frame. The column must be a numerical column.

Parameters

- **column** – a string for column name or an integer index
- **nPercent** – a bottom percentage of the column values to return

Returns

a H2OFrame containing two columns. The first column contains the original row indices where the bottom values are extracted from. The second column contains the bottom nPercent values.

Examples

```
>>> import numpy as np
>>> from random import randint
>>> dataFrame = h2o.import_file("https://s3.amazonaws.com/h2o-public-test-data/bigdata/laptop/jira/TopBottomNRep4.csv.zip")
>>> bottomAnswer = h2o.import_file("https://s3.amazonaws.com/h2o-public-test-data/smalldata/jira/Bottom20Per.csv.zip")
>>> nPercentages = [1,2,3,4]
>>> frameNames = dataFrame.names
>>> tolerance=1e-12
>>> nsample=100
>>> nP = nPercentages[randint(0, len(nPercentages)-1)]
>>> colIndex = randint(0, len(frameNames)-2)
>>> dataFrame.bottomN(frameNames[colIndex], nP)
```

categories () [source]

Make a list of levels for an enum (categorical) column. This function can only be applied to single-column categorical frame.

Returns

The list of levels for an enum column.

Examples

```
>>> iris = h2o.import_file("http://h2o-public-test-
data.s3.amazonaws.com/smalldata/iris/iris_wheader.csv")
>>> category_list =  iris['class'].categories()
>>> print(category_list)
```

`cbind` (*data*) [source]

Append data to this frame column-wise.

Parameters

data (*H2OFrame*) – append columns of frame `data` to the current frame. You can also cbind a number, in which case it will get converted into a constant column.

Returns

new H2OFrame with all frames in `data` appended column-wise.

Examples

```
>>> iris = h2o.import_file("http://h2o-public-test-
data.s3.amazonaws.com/smalldata/iris/iris_wheader.csv")
>>> iris = iris.cbind(iris[4] == "Iris-setosa")
>>> iris[5] = iris[5].asfactor()
>>> iris.set_name(5,"C6")
>>> iris = iris.cbind(iris[4] == "Iris-virginica")
>>> iris[6] = iris[6].asfactor()
>>> iris.set_name(6, name="C7")
>>> print(iris)
```

`ceil` () [source]

Apply the ceiling function to the current frame.

`ceil(x)` is the smallest integer greater or equal to `x`.

Returns

new H2OFrame of ceiling values of the original frame.

Examples

```
>>> from random import randrange
>>> import math
>>> import numpy as np
>>> row_num = randrange(1,10)
>>> col_num = randrange(1,10)
>>> length_out_r = math.ceil(0.78*row_num)
>>> length_out_c = math.ceil(col_num*0.4)
>>> python_lists = np.random.randint(-5,5, (row_num, col_num))
>>> h2oframe = h2o.H2OFrame(python_obj=python_lists)
```

property `col_names`

Displays the column names. Same as `self.names`.

Returns

Column names.

Examples

```
>>> python_obj = [1,2,2.5,-100.9,0]
>>> frame = h2o.H2OFrame(python_obj)
>>> frame.col_names
```

property `columns`

Displays the column names. Same as `self.names`.

Returns

Column names.

Examples

```
>>> python_obj = [1,2,2.5,-100.9,0]
>>> frame = h2o.H2OFrame(python_obj)
>>> frame.columns
```

`columns_by_type` (***coltype='numeric'***) [source]

Extract columns of the specified type from the frame.

Parameters

coltype (*str*) –

A character string indicating which column type to filter by. This must be one of the following:

- `"numeric"` - Numeric, but not categorical or time
- `"categorical"` - Integer, with a categorical/factor String mapping
- `"string"` - String column
- `"time"` - Long msec since the Unix Epoch - with a variety of display/parse options
- `"uuid"` - UUID

- `"bad"` - No none-NA rows (triple negative! all NAs or zero rows)

Returns

list of indices of columns that have the requested type

Examples

```
>>> frame = h2o.create_frame(rows=10,
...                          integer_fraction=1,
...                          binary_ones_fraction=0,
...                          missing_fraction=0)
>>> num = frame.columns_by_type(coltype="numeric")
>>> str = frame.columns_by_type(coltype="string")
>>> num
>>> string
```

`concat` (***frames***, ***axis=1***) [source]

Append multiple H2OFrames to this frame, column-wise or row-wise.

Parameters

- **frames** (*List[H2OFrame]*) – list of frames that should be appended to the current frame.
- **axis** (*int*) – if 1 then append column-wise (default), if 0 then append row-wise.

Returns

an H2OFrame of the combined datasets.

Examples

```
>>> df1 = h2o.create_frame(integer_fraction=1,binary_fraction=0,
...                        categorical_fraction=0,seed=1)
>>> df2 = h2o.create_frame(integer_fraction=1,binary_fraction=0,
...                        categorical_fraction=0,seed=2)
>>> df3 = h2o.create_frame(integer_fraction=1,binary_fraction=0,
...                        categorical_fraction=0,seed=3)
>>> df123 = df1.concat([df2,df3])
```

`convert_H2OFrame_2_DMatrix` (***predictors***, ***yresp***, ***h2oXGBoostModel***) [source]

This method requires that you import the following toolboxes: xgboost, pandas, numpy and scipy.sparse.

This method will convert an H2OFrame to a DMatrix that can be used by native XGBoost. The H2OFrame contains numerical and enum columns alone. Note that H2O one-hot-encoding introduces a missing(NA) column. There can be NAs in any columns.

Follow the steps below to compare H2OXGBoost and native XGBoost:

1. Train the H2OXGBoost model with H2OFrame trainFile and generate a prediction: h2oModelD = H2OXGBoostEstimator(**h2oParamsD) # parameters specified as a dict() h2oModelD.train(x=myX, y=y, training_frame=trainFile) # train with H2OFrame trainFile h2oPredict = h2oPredictD = h2oModelD.predict(trainFile)

2. Derive the DMatrix from H2OFrame: nativeDMatrix = trainFile.convert_H2OFrame_2_DMatrix(myX, y, h2oModelD)

3. Derive the parameters for native XGBoost: nativeParams = h2oModelD.convert_H2OXGBoostParams_2_XGBoostParams()

4. Train your native XGBoost model and generate a prediction: nativeModel = xgb.train(params=nativeParams[0], dtrain=nativeDMatrix, num_boost_round=nativeParams[1]) nativePredict = nativeModel.predict(data=nativeDMatrix, ntree_limit=nativeParams[1].

5. Compare the predictions h2oPredict from H2OXGBoost, nativePredict from native XGBoost.

> **Parameters**
>
> - **h2oFrame** – H2OFrame to be converted to DMatrix for native XGBoost
> - **predictors** – List of predictor columns, can be column names or indices
> - **yresp** – response column, can be column index or name
> - **h2oXGBoostModel** – H2OXGboost model that are built with the same H2OFrame as input earlier

> **Returns**
>
> DMatrix that can be an input to a native XGBoost model

> **Examples**

```
>>> import xgboost as xgb
>>> from h2o.estimators.xgboost import *
>>> data = h2o.import_file("https://s3.amazonaws.com/h2o-public-test-
data/smalldata/jira/adult_data_modified.csv")
>>> data[14] = data[14].asfactor()
>>> myX = list(range(0, 13))
>>> y='income'
>>> h2oParamsD = {"ntrees":30, "max_depth":4, "seed":2,
...               "learn_rate":0.7,"col_sample_rate_per_tree" : 0.9,
...               "min_rows" : 5, "score_tree_interval": 30+1,
...               "tree_method": "exact", "backend":"cpu"}
>>> h2oModelD = H2OXGBoostEstimator(**h2oParamsD)
>>> h2oModelD.train(x=myX, y=y, training_frame=data)
>>> h2oPredictD = h2oModelD.predict(data)
>>> nativeXGBoostParam = h2oModelD.convert_H2OXGBoostParams_2_XGBoostParams()
>>> nativeXGBoostInput = data.convert_H2OFrame_2_DMatrix(myX,
...                                                      y,
...                                                      h2oModelD)
>>> nativeModel = xgb.train(params=nativeXGBoostParam[0],
...                         dtrain=nativeXGBoostInput,
...                         num_boost_round=nativeXGBoostParam[1])
>>> nativePred = nativeModel.predict(data=nativeXGBoostInput,
...                                  ntree_limit=nativeXGBoostParam[1])
```

`cor` (*y=None, na_rm=False, use=None, method='Pearson'*) [source]

Compute the correlation matrix of one or two H2OFrames.

Parameters

- **y** (*H2OFrame*) – If this parameter is provided, then compute correlation between the columns of `y` and the columns of the current frame. If this parameter is not given, then just compute the correlation matrix for the columns of the current frame.
- **use** (*str*) –

  A string indicating how to handle missing values. This could be one of the following:

  - `"everything"` : outputs NaNs whenever one of its contributing observations is missing
  - `"all.obs"` : presence of missing observations will throw an error
  - `"complete.obs"` : discards missing values along with all observations in their rows so that only complete observations are used
- **na_rm** (*bool*) – an alternative to `use` : when this is True then default value for `use` is `"everything"` ; and if False then default `use` is `"complete.obs"` . This parameter has no effect if `use` is given explicitly.
- **method** (*str*) – Which method to use - value must be in [“Pearson”, “Spearman”]. Defaults to “Pearson”.

Returns

An H2OFrame of the correlation matrix of the columns of this frame (if `y` is not given), or with the columns of `y` (if `y` is given). However when this frame and `y` are both single rows or single columns, then the correlation is returned as a scalar.

Examples

```
>>> import numpy as np
>>> iris = h2o.import_file("http://h2o-public-test-
data.s3.amazonaws.com/smalldata/iris/iris_wheader.csv")
>>> irisnp = np.genfromtxt(("http://h2o-public-test-
data.s3.amazonaws.com/smalldata/iris/iris_wheader.csv"), delimiter=',', skip_header=1,
usecols=(0,1,2,3))
>>> cor_np = h2o.H2OFrame(np.corrcoef(irisnp,rowvar=0))
>>> cor_h2o = iris[0:4].cor()
>>> cor_diff = abs(cor_h2o - cor_np)
>>> print(cor_diff)
```

cos () [source]

Create a new H2OFrame equal to elementwise cosine of the current frame.

Returns

New H2OFrame equal to elementwise cosine of the current frame.

Examples

```
>>> python_obj = [1, 2, 2.5, -100.9, 0]
>>> frame = h2o.H2OFrame(python_obj)
>>> frame.cos()
```

cosh () [source]

Create a new H2OFrame with values equal to the hyperbolic cosines of the values in the current frame.

Returns

New H2OFrame with values equal to the hyperbolic cosines of the values in the current frame.

Examples

```
>>> python_obj = [1,2,2.5,-100.9,0]
>>> frame = h2o.H2OFrame(python_obj)
>>> frame.cosh()
```

cospi () [source]

Create a new H2OFrame equal to elementwise cosine of the current frame multiplied by Pi.

Returns

New H2OFrame equal to elementwise cosine of the current frame multiplied by Pi.

Examples

```
>>> python_obj = [1, 2, 2.5, -100.9, 0]
>>> frame = h2o.H2OFrame(python_obj)
>>> frame.cospi()
```

countmatches (*pattern*) [source]

For each string in the frame, count the occurrences of the provided pattern. If countmatches is applied to a frame, all columns of the frame must be type string, otherwise, the returned frame will contain errors.

The pattern here is a plain string, not a regular expression. We will search for the occurrences of the pattern as a substring in element of the frame. This function is applicable to frames containing only string or categorical columns.

Parameters

pattern (*str*) – The pattern to count matches on in each string. This can also be a list of strings, in which case all of them will be searched for.

Returns

numeric H2OFrame with the same shape as the original, containing counts of matches of the pattern for each cell in the original frame.

Examples

```
>>> iris = h2o.import_file("http://h2o-public-test-
data.s3.amazonaws.com/smalldata/iris/iris_wheader.csv")
>>> result = iris["class"].countmatches("o")
>>> result2 = iris["class"].countmatches("s")
>>> result
>>> result2
```

cummax (*axis=0*) [source]

Compute cumulative maximum over rows / columns of the frame.

Parameters

axis (*int*) – 0 for column-wise, 1 for row-wise

Returns

new H2OFrame with running maximums of the original frame.

Examples

```
>>> foo = h2o.H2OFrame([[x,y] for x,
...                     y in zip(list(range(10)),
...                     list(range(9,-1,-1)))])
>>> cummax1 = foo[0].cummax()
>>> cummax1
>>> cummax2 = foo[1].cummax()
>>> cummax2
```

`cummin` (*axis=0*) [source]

Compute cumulative minimum over rows / columns of the frame.

Parameters

axis (*int*) – 0 for column-wise, 1 for row-wise

Returns

new H2OFrame with running minimums of the original frame.

Examples

```
>>> foo = h2o.H2OFrame([[x,y] for x,
...                     y in zip(list(range(10)),
...                     list(range(9,-1,-1)))])
>>> cummin1 = foo[0].cummin()
>>> cummin1
>>> cummin2 = foo[1].cummin()
>>> cummin2
```

`cumprod` (*axis=0*) [source]

Compute cumulative product over rows / columns of the frame.

Parameters

axis (*int*) – 0 for column-wise, 1 for row-wise

Returns

new H2OFrame with cumulative products of the original frame.

Examples

```
>>> foo = h2o.H2OFrame([[x,y] for x,
...                     y in zip(list(range(10)),
...                     list(range(9,-1,-1)))])
>>> cumprod1 = foo[1:10,0].cumprod()
>>> cumprod1
>>> cumprod2 = foo[0:9,1].cumprod()
>>> cumprod2
```

`cumsum` (*axis=0*) [source]

Compute cumulative sum over rows / columns of the frame.

Parameters

axis (*int*) – 0 for column-wise, 1 for row-wise

Returns

new H2OFrame with cumulative sums of the original frame.

Examples

```
>>> foo = h2o.H2OFrame([[x,y] for x,
...                     y in zip(list(range(10)),
...                     list(range(9,-1,-1)))])
>>> cumsum1 = foo[0].cumsum()
>>> cumsum1
>>> cumsum2 = foo[1].cumsum()
```

`cut` (*breaks*, *labels=None*, *include_lowest=False*, *right=True*, *dig_lab=3*) [source]

Cut a numeric vector into categorical “buckets”.

This method is only applicable to a single-column numeric frame.

Parameters

- **breaks** (*List[float]*) – The cut points in the numeric vector.
- **labels** (*List[str]*) – Labels for categorical levels produced. Defaults to set notation of intervals defined by the breaks.
- **include_lowest** (*bool*) – By default, cuts are defined as intervals `(lo, hi]`. If this parameter is True, then the interval becomes `[lo, hi]`.
- **right** (*bool*) – Include the high value: `(lo, hi]`. If False, get `(lo, hi)`.
- **dig_lab** (*int*) – Number of digits following the decimal point to consider.

Returns

Single-column H2OFrame of categorical data.

Examples

```
>>> import numpy as np
>>> python_lists = np.random.uniform(-2,2,(100,1))
>>> h2oframe = h2o.H2OFrame(python_obj=python_lists)
>>> breaks = [-2,1,0,1,2]
>>> newframe = h2oframe.cut(breaks,
...                         labels=None,
...                         include_lowest=False,
...                         right=True,
...                         dig_lab=3)
>>> newframe
```

day () [source]

Extract the “day” part from a date column.

Returns

a single-column H2OFrame containing the “day” part from the source frame.

Examples

```
>>> df = h2o.create_frame(rows=10,
...                       cols=3,
...                       factors=10,
...                       categorical_fraction=1.0/3,
...                       time_fraction=1.0/3,
...                       real_fraction=1.0/3,
...                       real_range=100,
...                       missing_fraction=0.0,
...                       seed=123)
>>> df["C1"].day()
```

dayOfWeek () [source]

Extract the “day-of-week” part from a date column.

Returns

a single-column H2OFrame containing the “day-of-week” part from the source frame.

Examples

```
>>> df = h2o.create_frame(rows=10,
...                       cols=3,
...                       factors=10,
...                       categorical_fraction=1.0/3,
...                       time_fraction=1.0/3,
...                       real_fraction=1.0/3,
...                       real_range=100,
...                       missing_fraction=0.0,
...                       seed=123)
>>> df["C1"].dayOfWeek()
```

`describe` (*chunk_summary=False*) [source]

Generate an in-depth description of this H2OFrame.

This will print to the console the dimensions of the frame; names/types/summary statistics for each column; and finally first ten rows of the frame.

Parameters

chunk_summary (*bool*) – Retrieve the chunk summary along with the distribution summary

Returns

The dimensions of the frame; names/types/summary statistics for each column; first ten rows of the frame.

Examples

```
>>> python_lists = [[1,2,3],[4,5,6],["a","b","c"],[1,0,1]]
>>> col_names=["num1","num2","str1","enum1"]
>>> dest_frame="newFrame"
>>> heads=-1
>>> sep=','
>>> col_types=['numeric','numeric','string','enum']
>>> na_str=['NA']
>>> h2oframe = h2o.H2OFrame(python_obj=python_lists,
...                         destination_frame=dest_frame,
...                         header=heads,
...                         separator=sep,
...                         column_names=col_names,
...                         column_types=col_types,
...                         na_strings=na_str)
>>> h2oframe.describe(chunk_summary=True)
```

`detach` () [source]

Detach the Python object from the backend, usually by clearing its key

Returns

Removed H2OFrame

Examples

```
>>> from random import randrange
>>> import numpy as np
>>> row_num = randrange(2,10)
>>> col_num = randrange(2,10)
>>> python_lists = np.random.randint(-5,5, (row_num, col_num))
>>> h2oframe = h2o.H2OFrame(python_obj=python_lists)
>>> h2oframe.detach()
>>> h2oframe
```

difflag1 () [source]

Conduct a diff-1 transform on a numeric frame column.

Returns

an H2OFrame where each element is equal to the corresponding element in the source frame minus the previous-row element in the same frame.

Examples

```
>>> import pandas as pd
>>> import numpy as np
>>> df = pd.DataFrame(np.random.randint(0,100,size=(1000000, 1)),
...                   columns=list('A'))
>>> df_diff = df.diff()
>>> df_diff_h2o = h2o.H2OFrame(df_diff)
>>> fr = h2o.H2OFrame(df)
>>> fr_diff = fr.difflag1()
>>> fr_diff
```

digamma () [source]

Create a new H2OFrame equal to elementwise digamma function of the current frame.

Returns

New H2OFrame equal to elementwise digamma function of the current frame.

Examples

```
>>> python_obj = [1, 2, 2.5, -100.9, 0]
>>> frame = h2o.H2OFrame(python_obj)
>>> frame.digamma()
```

property dim

Gives the dimensions of the frame. Same as `list(self.shape)`.

Returns

Frame dimensions.

Examples

```
>>> iris = h2o.import_file("http://h2o-public-test-
data.s3.amazonaws.com/smalldata/iris/iris_wheader.csv")
>>> iris.dim
```

`distance` (***y***, ***measure=None***) [source]

Compute a pairwise distance measure between all rows of two numeric H2OFrames.

Parameters

- **y** (*H2OFrame*) – Frame containing queries (small)
- **use** (*str*) –

  A string indicating what distance measure to use. Must be one of:

  - `"l1"` : Absolute distance (L1-norm, >=0)
  - `"l2"` : Euclidean distance (L2-norm, >=0)
  - `"cosine"` : Cosine similarity (-1…1)
  - `"cosine_sq"` : Squared Cosine similarity (0…1)

Returns

An H2OFrame of the matrix containing pairwise distance / similarity between the rows of this frame (N x p) and `y` (M x p), with dimensions (N x M).

Examples

```
>>> iris_h2o = h2o.import_file("http://h2o-public-test-
data.s3.amazonaws.com/smalldata/iris/iris_wheader.csv")
>>> references = iris_h2o[10:150,0:4]
>>> queries = iris_h2o[0:10,0:4]
>>> A = references.distance(queries, "l1")
>>> B = references.distance(queries, "l2")
>>> C = references.distance(queries, "cosine")
>>> D = references.distance(queries, "cosine_sq")
>>> E = queries.distance(references, "l1")
>>> (E.transpose() == A).all()
```

`drop` (***index***, ***axis=1***) [source]

Drop a single column or row or a set of columns or rows from a H2OFrame.

Dropping a column or row is not in-place. Indices of rows and columns are zero-based.

Parameters

- **index** – A list of column indices, column names, or row indices to drop; or a string to drop a single column by name; or an int to drop a single column by index.
- **axis** (*int*) – If 1 (default), then drop columns; if 0 then drop rows.

Returns

a new H2OFrame with the respective dropped columns or rows. The original H2OFrame remains unchanged.

Examples

```
>>> pros = h2o.import_file("http://s3.amazonaws.com/h2o-public-test-
data/smalldata/prostate/prostate.csv.zip")
>>> nc = pros.ncol
>>> nr = pros.nrow
>>> dropped_col_int = pros.drop(0)
>>> dropped_col_int
```

`drop_duplicates` ***(columns, keep='first')*** [source]

Drops duplicated rows across specified columns. :param columns: Columns to compare during the duplicate detection process. :param keep: Which rows to keep. Two possible values: [“first”, “last”]. The “first” value (default) keeps

the first row and deletes the rest. The “last” value keeps the last row.

Returns

A new H2OFrame with rows deduplicated

property `dtype`

Returns the numpy.dtype of the first column of this data frame. Works only for single-column data frames. Used mainly for using H2OFrames in conjunction with scikit-learn APIs.

Returns

Numpy dtype of the first column

`entropy` **()** [source]

For each string compute its Shannon entropy, if the string is empty the entropy is 0.

Returns

an H2OFrame of Shannon entropies.

Examples

```
>>> frame = h2o.H2OFrame.from_python(["redrum"])
>>> frame.entropy()
```

`exp` () [source]

Create a new H2OFrame equal to elementwise exponent (i.e. `e^x`) of the current frame.

Returns

New H2OFrame equals to elementwise exponent (i.e. `e^x`) of the current frame.

Examples

```
>>> python_obj = [1, 2, 2.5, -100.9, 0]
>>> frame = h2o.H2OFrame(python_obj)
>>> frame.exp()
```

`expm1` () [source]

Create a new H2OFrame equal to elementwise exponent minus 1 (i.e. `e^x - 1`) of the current frame.

Returns

New H2OFrame equal to elementwise exponent minus 1 (i.e. `e^x - 1`) of the current frame.

Examples

```
>>> python_obj = [1, 2, 2.5, -100.9, 0]
>>> frame = h2o.H2OFrame(python_obj)
>>> frame.expm1()
```

`fillna` (*method='forward'*, *axis=0*, *maxlen=1*) [source]

Return a new Frame that fills NA along a given axis and along a given direction with a maximum fill length.

Parameters

- **method** – `"forward"` or `"backward"`

- **axis** – 0 for columnar-wise or 1 for row-wise fill
- **maxlen** – Max number of consecutive NA's to fill

Returns

A new Frame that fills NA along a given axis and along a given direction with a maximum fill length.

Examples

```
>>> python_obj = [1,2,2.5,-100.9,0,',',',',',',']
>>> frame = h2o.H2OFrame(python_obj)
>>> frame
>>> frame.fillna(method='forward', axis=0, maxlen=3)
```

`filter_na_cols` (*frac=0.2*) [source]

Filter columns with proportion of NAs greater or equals than `frac`.

Parameters

frac (*float*) – Maximum fraction of NAs in the column to keep.

Returns

A list of indices of columns that have fewer NAs than `frac`. If all columns are filtered, None is returned.

Examples

```
>>> prostate = h2o.import_file("http://s3.amazonaws.com/h2o-public-test-
data/smalldata/prostate/prostate.csv.zip")
>>> include_cols1 = prostate.filter_na_cols()
>>> include_cols1
>>> include_cols2 = prostate.filter_na_cols(0.001)
>>> include_cols2
```

`flatten` () [source]

Convert a 1x1 frame into a scalar.

Returns

content of this 1x1 frame as a scalar (`int`, `float`, or `str`).

Raises

H2OValueError – if current frame has shape other than 1x1

Examples

```
>>> python_obj = [1, 2, 2.5, -100.9, 0]
>>> frame1 = h2o.H2OFrame(python_obj)
>>> frame1.flatten()
# Should receive "H2OValueError: Not a 1x1 Frame"
>>> frame2 = h2o.H2OFrame.from_python(["redrum"])
>>> frame2.flatten()
```

`floor` () [source]

Apply the floor function to the current frame. `floor(x)` is the largest integer smaller or equal to `x`.

Returns

new H2OFrame of floor values of the original frame.

Examples

```
>>> python_obj = [1, 2, 2.5, -100.9, 0]
>>> frame = h2o.H2OFrame(python_obj)
>>> frame
>>> frame.floor()
```

property `frame_id`

Internal id of the frame (str).

Returns

Internal id of the frame (str).

Examples

```
>>> iris = h2o.import_file("http://h2o-public-test-
data.s3.amazonaws.com/smalldata/iris/iris_wheader.csv")
>>> print(iris.frame_id)
```

static `from_python` (*python_obj*, *destination_frame=None*, *header=0*, *separator=', '*, *column_names=None*, *column_types=None*, *na_strings=None*) [source]

[DEPRECATED] Use constructor `H2OFrame()` instead.

`gamma` () [source]

Create a new H2OFrame equal to elementwise gamma function of the current frame.

Returns

new H2OFrame equals to elementwise gamma function of the current frame.

Examples

```
>>> python_obj = [1, 2, 2.5, -100.9, 0]
>>> frame = h2o.H2OFrame(python_obj)
>>> frame.gamma()
```

static `get_frame` **(*frame_id*, *rows=10*, *rows_offset=0*, *cols=-1*, *full_cols=-1*, *cols_offset=0*, *light=False*)** [source]

Retrieve an existing H2OFrame from the H2O cluster using the frame's id.

Parameters

- **frame_id** (*str*) – id of the frame to retrieve
- **rows** (*int*) – number of rows to fetch for preview (10 by default)
- **rows_offset** (*int*) – offset to fetch rows from (0 by default)
- **cols** (*int*) – number of columns to fetch (all by default)
- **full_cols** – number of columns to fetch together with backed data
- **cols_offset** (*int*) – offset to fetch rows from (0 by default)
- **light** (*bool*) – whether to use light frame endpoint or not

Returns

an existing H2OFrame with the id provided; or None if such frame doesn't exist.

Examples

```
>>> iris = h2o.import_file("http://h2o-public-test-
data.s3.amazonaws.com/smalldata/iris/iris_wheader.csv")
>>> h2o.get_frame(iris.frame_id)
```

`get_frame_data` **()** [source]

Get frame data as a string in csv format.

This will create a multiline string, where each line will contain a separate row of frame's data, with individual values separated by commas.

Returns

Frame data as a string in csv format.

Examples

```
>>> iris = h2o.import_file("http://h2o-public-test-
data.s3.amazonaws.com/smalldata/iris/iris_wheader.csv")
>>> iris.get_frame_data()
```

getrow () [source]

Convert a 1xn frame into an n-element list.

Returns

content of this 1xn frame as a Python list.

Raises

H2OValueError – if current frame has more than one row.

Examples

```
>>> import scipy.sparse as sp
>>> A = sp.csr_matrix([[1, 2, 0, 5.5], [0, 0, 3, 6.7], [4, 0, 5, 0]])
>>> fr = h2o.H2OFrame(A)
>>>  assert fr.shape == (3, 4)
>>> assert fr.as_data_frame(False) ==
...     [['C1', 'C2', 'C3', 'C4'], ['1', '2', '0', '5.5'],
...     ['0', '0', '3', '6.7'], ['4', '0', '5', '0.0']]
>>> A = sp.lil_matrix((1000, 1000))
>>> A.setdiag(10)
>>> for i in range(999):
...     A[i, i + 1] = -3
...     A[i + 1, i] = -2
>>> fr = h2o.H2OFrame(A)
>>> assert fr.shape == (1000, 1000)
>>> means = fr.mean().getrow()
>>> assert means == [0.008] + [0.005] * 998 + [0.007]
>>> means
```

grep (*pattern*, *ignore_case=False*, *invert=False*, *output_logical=False*) [source]

Searches for matches to argument *pattern* within each element of a string column.

Default behavior is to return indices of the elements matching the pattern. Parameter *output_logical* can be used to return a logical vector indicating if the element matches the pattern (1) or not (0).

Parameters

- **pattern** (*str*) – A character string containing a regular expression.
- **ignore_case** (*bool*) – If True, then case is ignored during matching.
- **invert** (*bool*) – If True, then identify elements that do not match the pattern.

- **output_logical** (*bool*) – If True, then return logical vector of indicators instead of list of matching positions

Returns

H2OFrame holding the matching positions or a logical list if *output_logical* is enabled.

Examples

```
>>> iris = h2o.import_file("http://h2o-public-test-
data.s3.amazonaws.com/smalldata/iris/iris_wheader.csv")
>>> pattern = "Iris-setosa"
>>> iris["class"].grep(pattern, output_logical=True)
```

`group_by` **(*by*)** [source]

Return a new `GroupBy` object using this frame and the desired grouping columns.

The returned groups are sorted by the natural group-by column sort.

Parameters

by – The columns to group on (either a single column name, or a list of column names, or a list of column indices).

Returns

New `GroupBy` object, sorted by the natural group-by column sort.

Examples

```
>>> iris = h2o.import_file("http://h2o-public-test-
data.s3.amazonaws.com/smalldata/iris/iris_wheader.csv")
>>> na_handling = ["rm","ignore","all"]
>>> for na in na_handling:
...     grouped = iris.group_by("class")
...     grouped
...             .count(na=na)
...             .min(na=na)
...             .max(na=na)
...             .mean(na=na)
...             .var(na=na)
...             .sd(na=na)
...             .ss(na=na)
...             .sum(na=na)
...     print(grouped.get_frame())
...     print(grouped.get_frame())
```

`gsub` **(*pattern*, *replacement*, *ignore_case=False*)** [source]

Globally substitute occurrences of pattern in a string with replacement.

Parameters

- **pattern** (*str*) – A regular expression.
- **replacement** (*str*) – A replacement string.
- **ignore_case** (*bool*) – If True then pattern will match case-insensitively.

Returns

an H2OFrame with all occurrences of `pattern` in all values replaced with `replacement`.

Examples

```
>>> iris = h2o.import_file(("http://h2o-public-test-
data.s3.amazonaws.com/smalldata/iris/iris.csv"),
...                         col_types=["numeric","numeric",
...                                    "numeric","numeric",
...                                    "string"])
>>> iris["C5"].gsub("s","z",ignore_case=False)
```

`head` (*rows=10*, *cols=200*) [source]

Return the first `rows` and `cols` of the frame as a new H2OFrame.

Parameters

- **rows** (*int*) – maximum number of rows to return
- **cols** (*int*) – maximum number of columns to return

Returns

a new H2OFrame cut from the top left corner of the current frame, and having dimensions at most `rows` x `cols`.

Examples

```
>>> import numpy as np
>>> from h2o.frame import H2OFrame
>>> iris = h2o.import_file("http://h2o-public-test-
data.s3.amazonaws.com/smalldata/iris/iris_wheader.csv")
>>> df = H2OFrame.from_python(np.random.randn(100, 4).tolist(),
...                           column_names=list("ABCD"),
...                           column_types=["enum"] * 4)
>>> df.head()
```

`hist` (*breaks='sturges'*, *plot=True*, ***kwargs*) [source]

Compute a histogram over a numeric column.

Parameters

- **breaks** – Can be one of `"sturges"`, `"rice"`, `"sqrt"`, `"doane"`, `"fd"`, `"scott"`; or a single number for the number of breaks; or a list containing the split points, e.g: `[-50, 213.2123, 9324834]`. If breaks is "fd", the MAD is used over the IQR in computing bin width.
- **plot** (*bool*) – If True (default), then a plot will be generated using `matplotlib`.

Returns

If `plot` is False, return H2OFrame with these columns: breaks, counts, mids_true, mids, and density; otherwise this method draws a plot and returns nothing.

Examples

```
>>> iris = h2o.import_file("http://h2o-public-test-
data.s3.amazonaws.com/smalldata/iris/iris_wheader.csv")
>>> iris.describe()
>>> iris[0].hist(breaks=5,plot=False)
```

`hour` **()** [source]

Extract the "hour-of-day" part from a date column.

Returns

a single-column H2OFrame containing the "hour-of-day" part from the source frame.

Examples

```
>>> df = h2o.create_frame(rows=10,
...                       cols=3,
...                       factors=10,
...                       categorical_fraction=1.0/3,
...                       time_fraction=1.0/3,
...                       real_fraction=1.0/3,
...                       real_range=100,
...                       missing_fraction=0.0,
...                       seed=123)
>>> df["C1"].hour()
```

`idxmax` ***(skipna=True, axis=0)*** [source]

Get the index of the max value in a column or row

Parameters

- **skipna** (*bool*) – If True (default), then NAs are ignored during the search. Otherwise presence of NAs renders the entire result NA.

- **axis** (*int*) – Direction of finding the max index. If 0 (default), then the max index is searched columnwise, and the result is a frame with 1 row and number of columns as in the original frame. If 1, then the max index is searched rowwise and the result is a frame with 1 column, and number of rows equal to the number of rows in the original frame.

Returns

either a list of max index values per-column or an H2OFrame containing max index values per-row from the original frame.

Examples

```
>>> f1 = h2o.create_frame(rows = 10000,
...                       cols = 100,
...                       categorical_fraction = 0,
...                       missing_fraction = 0,
...                       seed=1234)
>>> f1.idxmax()
```

`idxmin` (***skipna=True, axis=0***) [source]

Get the index of the min value in a column or row

Parameters

- **skipna** (*bool*) – If True (default), then NAs are ignored during the search. Otherwise presence of NAs renders the entire result NA.
- **axis** (*int*) – Direction of finding the min index. If 0 (default), then the min index is searched columnwise, and the result is a frame with 1 row and number of columns as in the original frame. If 1, then the min index is searched rowwise and the result is a frame with 1 column, and number of rows equal to the number of rows in the original frame.

Returns

either a list of min index values per-column or an H2OFrame containing min index values per-row from the original frame.

Examples

```
>>> f1 = h2o.create_frame(rows = 10000,
...                       cols = 100,
...                       categorical_fraction = 0,
...                       missing_fraction = 0,
...                       seed=1234)
>>> f1.idxmin()
```

`ifelse` (***yes***, ***no***) [source]

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

Based on the booleans in the test vector, the output has the values of the yes and no vectors interleaved (or merged together). All Frames must have the same row count. Single column frames are broadened to match wider Frames. Scalars are allowed, and are also broadened to match wider frames.

Parameters

- **yes** – Frame to use if `test` is true; may be a scalar or single column
- **no** – Frame to use if `test` is false; may be a scalar or single column

Returns

an H2OFrame of the merged yes/no frames/scalars according to the test input frame.

Examples

```
>>> import numpy as np
>>> from h2o.frame import H2OFrame
>>> python_lists = np.random.uniform(-1,1, (5,5))
>>> h2oframe = h2o.H2OFrame(python_obj=python_lists)
>>> newFrame = (h2oframe>0).ifelse(1, -1)
>>> newFrame
```

`impute` (***column=-1***, ***method='mean'***, ***combine_method='interpolate'***, ***by=None***, ***group_by_frame=None***, ***values=None***) [source]

Impute missing values into the frame, modifying it in-place.

Parameters

- **column** (*int*) – Index of the column to impute, or -1 to impute the entire frame.
- **method** (*str*) – The method of imputation: `"mean"`, `"median"`, or `"mode"`.
- **combine_method** (*str*) – When the method is `"median"`, this setting dictates how to combine quantiles for even samples. One of `"interpolate"`, `"average"`, `"low"`, `"high"`.
- **by** – The list of columns to group on.
- **group_by_frame** (*H2OFrame*) – Impute the values with this pre-computed grouped frame.
- **values** (*List*) – The list of impute values, one per column. None indicates to skip the column.

Returns

A list of values used in the imputation or the group-by result used in imputation.

Examples

```
>>> prostate = h2o.import_file("http://s3.amazonaws.com/h2o-public-test-
data/smalldata/prostate/prostate.csv.zip")
>>> prostate.dim
>>> prostate.impute("DPROS", method="mean")
```

`insert_missing_values` (*fraction=0.1*, *seed=None*) [source]

Insert missing values into the current frame, modifying it in-place.

Randomly replaces a user-specified fraction of entries in a H2O dataset with missing values.

Parameters

- **fraction** (*float*) – A number between 0 and 1 indicating the fraction of entries to replace with missing.
- **seed** (*int*) – The seed for the random number generator used to determine which values to make missing.

Returns

the original H2OFrame with missing values inserted.

Examples

```
>>> data = [[1, 2, 3, 1, 'a', 1, 9],
...         [1, 6, 4, 2, 'a', 1, 9],
...         [2, 3, 8, 6, 'b', 1, 9],
...         [3, 4, 3, 2, 'b', 3, 8],
...         [4, 5, 9, 5, 'c', 2, 8],
...         [5, 7, 10,7, 'b', 8, 8]]
>>> h2o_data = h2o.H2OFrame(data)
>>> h2o_data.insert_missing_values(fraction = 0.0)
```

`interaction` (*factors*, *pairwise*, *max_factors*, *min_occurrence*, *destination_frame=None*) [source]

Categorical Interaction Feature Creation in H2O.

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

Parameters

- **factors** – list of factor columns (either indices or column names).

- **pairwise** (*bool*) – Whether to create pairwise interactions between factors (otherwise create one higher-order interaction). Only applicable if there are 3 or more factors.
- **max_factors** (*int*) – Max. number of factor levels in pair-wise interaction terms (if enforced, one extra catch-all factor will be made).
- **min_occurrence** (*int*) – Min. occurrence threshold for factor levels in pair-wise interaction terms.
- **destination_frame** (*str*) – (internal) string indicating the key for the frame created.

Returns

an H2OFrame

Examples

```
>>> iris = h2o.import_file("http://h2o-public-test-
data.s3.amazonaws.com/smalldata/iris/iris_wheader.csv")
>>> iris = iris.cbind(iris[4] == "Iris-setosa")
>>> iris[5] = iris[5].asfactor()
>>> iris.set_name(5,"C6")
>>> iris = iris.cbind(iris[4] == "Iris-virginica")
>>> iris[6] = iris[6].asfactor()
>>> iris.set_name(6, name="C7")
>>> two_way_interactions = h2o.interaction(iris,
...                                        factors=[4,5,6],
...                                        pairwise=True,
...                                        max_factors=10000,
...                                        min_occurrence=1)
>>> two_way_interactions
```

`isax` (***num_words*, *max_cardinality*, *optimize_card=False*, ***kwargs***) [source]

Compute the iSAX index for DataFrame which is assumed to be numeric time series data.

References:

- http://www.cs.ucr.edu/~eamonn/SAX.pdf
- http://www.cs.ucr.edu/~eamonn/iSAX_2.0.pdf

Parameters

- **num_words** (*int*) – Number of iSAX words for the timeseries, i.e. granularity along the time series
- **max_cardinality** (*int*) – Maximum cardinality of the iSAX word. Each word can have less than the max

- **optimized_card** (*bool*) – An optimization flag that will find the max cardinality regardless of what is passed in for `max_cardinality`.

Returns

An H2OFrame with the name of time series, string representation of iSAX word, followed by binary representation.

Examples

```
>>> df = h2o.create_frame(rows=1,
...                       cols=256,
...                       real_fraction=1.0,
...                       missing_fraction=0.0,
...                       seed=123)
>>> df2 = df.cumsum(axis=1)
>>> res = df2.isax(num_words=10,max_cardinality=10)
>>> res
```

`ischaracter` () [source]

[DEPRECATED] Use `frame.isstring()`.

`isfactor` () [source]

Test which columns in the current frame are categorical.

Returns

a list of True/False indicating for each column in the frame whether it is categorical.

Examples

```
>>> aa = {'h1': [1, 8, 4, 3, 6],
...       'h2': ["fish", "cat", "fish", "dog", "bird"],
...       'h3': [0, 1, 0, 0, 1]}
>>> df_hex = h2o.H2OFrame(aa)
>>> df_hex['h1'].isfactor()
>>> df_hex['h1'] = df_hex['h1'].asfactor()
>>> df_hex['h1'].isfactor()
```

`isin` (*item*) [source]

Test whether elements of an H2OFrame are contained in the `item`.

Parameters

items – An item or a list of items to compare the H2OFrame against.

Returns

An H2OFrame of 0s and 1s showing whether each element in the original H2OFrame is contained in item.

Examples

```
>>> fr = h2o.create_frame(rows=100, cols=1, categorical_fraction=1, factors=3)
>>> f2 = ~fr["C1"].isin(["c0.l0", "c0.l2"])
>>> f2
```

isna () [source]

For each element in an H2OFrame, determine if it is NA or not.

Returns

an H2OFrame of 1s and 0s, where 1s mean the values were NAs.

Examples

```
>>> from collections import OrderedDict
>>> frame = h2o.H2OFrame.from_python(OrderedDict([
...                                  ("A", [1, 0, 3, 4, 8, 4, 7]),
...                                  ("B", [2, nan, -1, nan, nan, 9, 0]),
...                                  ("C", ["one", "", "two", "", "seventeen", "1",
""]),
...                                  ("D", ["oneteen", "", "twoteen", "", "sixteen",
"twenteen", ""])
...                                  ]), na_strings=[""],
...                                  column_types={"C": "enum", "D": "string"})
>>> frame.isna()
```

isnumeric () [source]

Test which columns in the frame are numeric.

Returns

a list of True/False indicating for each column in the frame whether it is numeric.

Examples

```
>>> iris = h2o.import_file("http://h2o-public-test-
data.s3.amazonaws.com/smalldata/iris/iris_wheader.csv")
>>> iris.summary()
# Look at the column headers: [0:3] are numeric; [4] is not
>>> iris[0].isnumeric()
# Return as True
>>> iris[4].isnumeric()
# Return as False
```

`isstring` () [source]

Test which columns in the frame are string.

Returns

a list of True/False indicating for each column in the frame whether it is string.

Examples

```
>>> import numpy as np
>>> from random import randrange
>>> row_num = randrange(1,10)
>>> col_num = randrange(1,10)
>>> python_lists = np.random.randint(-5,5, (row_num, col_num))
>>> h2oframe = h2o.H2OFrame(python_obj=python_lists)
>>> h2oframe.isstring()
>>> newFrame = h2oframe.asfactor().ascharacter()
>>> newFrame.isstring()
```

property `key`

Displays the unique key representing the object on the backend.

Returns

the unique key representing the object on the backend

Examples

```
>>> frame = h2o.import_file("http://h2o-public-test-
data.s3.amazonaws.com/smalldata/iris/iris.csv")
>>> frame.key
```

`kfold_column` (*n_folds=3*, *seed=-1*) [source]

Build a fold assignments column for cross-validation.

This method will produce a column having the same data layout as the source frame.

Parameters

- **n_folds** (*int*) – An integer specifying the number of validation sets to split the training data into.
- **seed** (*int*) – Seed for random numbers as fold IDs are randomly assigned.

Returns

A single column H2OFrame with the fold assignments.

Examples

```
>>> from random import randrange
>>> import numpy as np
>>> python_lists = np.random.randint(-5,5, (1000, 2))
>>> k = randrange(2,10)
>>> h2oframe = h2o.H2OFrame(python_obj=python_lists)
>>> clist = h2oframe.kfold_column(n_folds=k, seed=12345)
>>> clist
```

kurtosis (*na_rm=False*) [source]

Compute the kurtosis of each column in the frame.

We calculate the common kurtosis, such that kurtosis(normal distribution) is 3.

Parameters

na_rm (*bool*) – If True, then ignore NAs during the computation.

Returns

A list containing the kurtosis for each column (NaN for non-numeric columns).

Examples

```
>>> import numpy as np
>>> from random import randrange
>>> python_lists = np.random.normal(0,1, (10000, 1))
>>> h2oframe = h2o.H2OFrame(python_obj=python_lists)
>>> h2oframe.kurtosis(na_rm=True)
```

levels () [source]

Get the factor levels.

Returns

A list of lists, one list per column, of levels.

Examples

```
>>> import numpy as np
>>> from random import randrange
>>> python_lists = np.random.randint(-2,2, (10000,2))
>>> h2oframe = h2o.H2OFrame(python_obj=python_lists,
...                         column_types=['enum', 'enum'])
>>> h2oframe.levels()
```

lgamma () [source]

Create a new H2OFrame equal to elementwise logarirth of the gamma function of the current frame.

Returns

New H2OFrame equal to elementwise logarithm of the gamma function of the current frame.

Examples

```
>>> python_obj = [1, 2, 2.5, -100.9, 0]
>>> frame = h2o.H2OFrame(python_obj)
>>> frame.lgamma()
```

`log` () [source]

Create a new H2OFrame equal to elementwise natural logarithm of the current frame.

Returns

New H2OFrame equal to elementwise natural logarithm of the current frame.

Examples

```
>>> python_obj = [1, 2, 2.5, -100.9, 0]
>>> frame = h2o.H2OFrame(python_obj)
>>> frame.log()
```

`log10` () [source]

Create new H2OFrame equal to elementwise decimal logarithm of the current frame.

Returns

New H2OFrame equal to elementwise decimal logarithm of the current frame.

Examples

```
>>> python_obj = [1, 2, 2.5, -100.9, 0]
>>> frame = h2o.H2OFrame(python_obj)
>>> frame.log10()
```

`log1p` () [source]

Create a new H2Oframe equal to elementwise `ln(1 + x)` for each `x` in the current frame.

Returns

New H2OFrame equals to elementwise `ln(1 + x)` for each `x` in the current frame.

Examples

```
>>> python_obj = [1, 2, 2.5, -100.9, 0]
>>> frame = h2o.H2OFrame(python_obj)
>>> frame.log1p()
```

`log2` () [source]

Create a new H2OFrame equal to elementwise binary logarithm of the current frame.

Returns

New H2OFrame equal to elementwise binary logarithm of the current frame.

Examples

```
>>> python_obj = [1, 2, 2.5, -100.9, 0]
>>> frame = h2o.H2OFrame(python_obj)
>>> frame.log2()
```

`logical_negation` () [source]

Create a new H2OFrame equal to elementwise Logical NOT applied to the current frame.

Returns

New H2OFrame equal to elementwise Logical NOT applied to the current frame.

Examples

```
>>> python_obj = [1, 2, 2.5, -100.9, 0]
>>> frame = h2o.H2OFrame(python_obj)
>>> frame.logical_negation()
```

`lstrip` (*set=' '*) [source]

Return a copy of the column with leading characters removed.

The set argument is a string specifying the set of characters to be removed. If omitted, the set argument defaults to removing whitespace.

Parameters

set (*character*) – The set of characters to lstrip from strings in column.

Returns

a new H2OFrame with the same shape as the original frame and having all its values trimmed from the left (equivalent of Python's `str.lstrip()`).

Examples

```
>>> iris = h2o.import_file("http://h2o-public-test-
data.s3.amazonaws.com/smalldata/iris/iris.csv")
>>> iris["C5"].lstrip("Iris-")
```

`match` (*table*, *nomatch=0*) [source]

Make a vector of the positions of (first) matches of its first argument in its second.

Only applicable to single-column categorical/string frames.

Parameters

- **table** (*List*) – the list of items to match against
- **nomatch** (*int*) – value that should be returned when there is no match.

Returns

a new H2OFrame containing for each cell from the source frame the index where the pattern `table` first occurs within that cell.

Examples

```
>>> iris = h2o.import_file("http://h2o-public-test-
data.s3.amazonaws.com/smalldata/iris/iris.csv")
>>> matchFrame = iris["C5"].match(['Iris-versicolor'])
>>> matchFrame
>>> matchFrame = iris["C5"].match(['Iris-setosa'])
>>> matchFrame
```

`max` () [source]

Show the maximum value of all frame entries.

Returns

The maximum value of all frame entries.

Examples

```
>>> iris = h2o.import_file("http://h2o-public-test-
data.s3.amazonaws.com/smalldata/iris/iris.csv")
>>> iris.max()
>>> iris["C1"].max()
>>> iris["C2"].max()
>>> iris["C3"].max()
>>> iris["C4"].max()
```

`mean` (***skipna=True*, *axis=0*, ***kwargs***) [source]

Compute the frame's means by-column (or by-row).

Parameters

- **skipna** (*bool*) – If True (default), then NAs are ignored during the computation. Otherwise presence of NAs renders the entire result NA.
- **axis** (*int*) – Direction of mean computation. If 0 (default), then mean is computed columnwise, and the result is a frame with 1 row and number of columns as in the original frame. If 1, then mean is computed rowwise and the result is a frame with 1 column (called "mean"), and number of rows equal to the number of rows in the original frame.

Returns

either a list of mean values per-column (old semantic); or an H2OFrame containing mean values per-column/per-row from the original frame (new semantic). The new semantic is triggered by either providing the `return_frame=True` parameter, or having the `general.allow_breaking_changed` config option turned on.

Examples

```
>>> iris = h2o.import_file("http://h2o-public-test-
data.s3.amazonaws.com/smalldata/iris/iris.csv")
>>> iris.mean()
```

`median` (***na_rm=False***) [source]

Compute the median of each column in the frame.

Parameters

na_rm (*bool*) – If True, then ignore NAs during the computation.

Returns

A list containing the median for each column (NaN for non-numeric columns).

Examples

```
>>> iris = h2o.import_file("http://h2o-public-test-
data.s3.amazonaws.com/smalldata/iris/iris.csv")
>>> iris.median()
```

`melt` (*id_vars*, *value_vars=None*, *var_name='variable'*, *value_name='value'*, *skipna=False*) [source]

Converts an H2OFrame to key-value representation while optionally skipping NA values. Inverse operation to pivot.

Parameters

- **id_vars** – Columns used as identifiers.
- **value_vars** – What columns will be converted to key-value pairs (default: complement to id_vars).
- **var_name** – Name of the key-column (default: “variable”).
- **value_name** – Name of the value-column (default: “value”).
- **skipna** – If enabled, do not include NAs in the result.

Returns

Returns an unpivoted H2OFrame.

Examples

```
>>> import pandas as pd
>>> from h2o.frame import H2OFrame
>>> df = pd.DataFrame({'A': {0: 'a', 1: 'b', 2: 'c'},
...                    'B': {0: 1, 2: 5},
...                    'C': {0: 2, 1: 4, 2: 6}})
>>> df
>>> frozen_h2o = H2OFrame(df)
>>> frozen_h2o
>>> melted = frozen_h2o.melt(id_vars=["A"], value_vars=["B"])
>>> melted
```

`merge` (*other*, *all_x=False*, *all_y=False*, *by_x=None*, *by_y=None*, *method='auto'*) [source]

Merge two datasets based on common column names. We do not support all_x=True and all_y=True. Only one can be True or none is True. The default merge method is auto and it will default to the radix method. The radix method will return the correct merge result regardless of duplicated rows in the right frame. In addition, the radix method can perform merge even if you have string columns in your frames. If there are duplicated rows in your rite frame, they will not be included if you use the hash method. The hash method cannot perform merge if you have string columns in your left frame. Hence, we consider the radix method superior to the hash method and is the default method to use.

Parameters

- **other** (*H2OFrame*) – The frame to merge to the current one. By default, must have at least one column in common with this frame, and all columns in common are used as the merge key. If you want to use only a subset of the columns in common, rename the other columns so the columns are unique in the merged result.
- **all_x** (*bool*) – If True, include all rows from the left/self frame
- **all_y** (*bool*) – If True, include all rows from the right/other frame
- **by_x** – list of columns in the current frame to use as a merge key.
- **by_y** – list of columns in the `other` frame to use as a merge key. Should have the same number of columns as in the `by_x` list.
- **method** – string representing the merge method, one of auto(default), radix or hash.

Returns

New H2OFrame with the result of merging the current frame with the `other` frame.

Examples

```
>>> col = 10000* [0, 0, 1, 1, 2, 3, 0]
>>> fr = h2o.H2OFrame(list(zip(*[col])))
>>> fr.set_names(['rank'])
>>> mapping = h2o.H2OFrame(list(zip(*[[0,1,2,3],[6,7,8,9]])))
>>> mapping.set_names(['rank', 'outcome'])
>>> merged = fr.merge(mapping,
...                   all_x=True,
...                   all_y=False,
...                   by_x=None,
...                   by_y=None,
...                   method='auto')
>>> merged
```

`min` () [source]

Show the minimum value of all frame entries.

Returns

The minimum value of all frame entries.

Examples

```
>>> iris = h2o.import_file("http://h2o-public-test-
data.s3.amazonaws.com/smalldata/iris/iris.csv")
>>> iris.min()
>>> iris["C1"].min()
>>> iris["C2"].min()
>>> iris["C3"].min()
>>> iris["C4"].min()
```

`minute` () [source]

Extract the “minute” part from a date column.

Returns

a single-column H2OFrame containing the “minute” part from the source frame.

Examples

```
>>> df = h2o.create_frame(rows=10,
...                       cols=3,
...                       factors=10,
...                       categorical_fraction=1.0/3,
...                       time_fraction=1.0/3,
...                       real_fraction=1.0/3,
...                       real_range=100,
...                       missing_fraction=0.0,
...                       seed=123)
>>> df["C1"].minute()
```

static `mktime` (*year=1970*, *month=0*, *day=0*, *hour=0*, *minute=0*, *second=0*, *msec=0*) [source]

Deprecated, use `moment()` instead.

This function was left for backward-compatibility purposes only. It is not very stable, and counterintuitively uses 0-based months and days, so “January 4th, 2001” should be entered as `mktime(2001, 0, 3)`.

`modulo_kfold_column` (*n_folds=3*) [source]

Build a fold assignments column for cross-validation.

Rows are assigned a fold according to the current row number modulo `n_folds`.

Parameters

n_folds (*int*) – An integer specifying the number of validation sets to split the training data into.

Returns

A single-column H2OFrame with the fold assignments.

Examples

```
>>> from random import randrange
>>> import numpy as np
>>> python_lists = np.random.randint(-5,5, (1000, 2))
>>> k = randrange(2,10)
>>> h2oframe = h2o.H2OFrame(python_obj=python_lists)
>>> h2oframe.kfold_column(n_folds=k)
```

static `moment` **(*year=None*, *month=None*, *day=None*, *hour=None*, *minute=None*, *second=None*, *msec=None*, *date=None*, *time=None*)** [source]

Create a time column from individual components.

Each parameter should be either an integer, or a single-column H2OFrame containing the corresponding time parts for each row.

The "date" part of the timestamp can be specified using either the tuple `(year, month, day)`, or an explicit `date` parameter. The "time" part of the timestamp is optional, but can be specified either via the `time` parameter, or via the `(hour, minute, second, msec)` tuple.

Parameters

- **year** – the year part of the constructed date
- **month** – the month part of the constructed date
- **day** – the day-of-the-month part of the constructed date
- **hour** – the hours part of the constructed date
- **minute** – the minutes part of the constructed date
- **second** – the seconds part of the constructed date
- **msec** – the milliseconds part of the constructed date
- **date** (*date*) – construct the timestamp from the Python's native `datetime.date` (or `datetime.datetime`) object. If the object passed is of type `date`, then you can specify the time part using either the `time` argument, or `hour` … `msec` arguments (but not both). If the object passed is of type `datetime`, then no other arguments can be provided.
- **time** (*time*) – construct the timestamp from this Python's native `datetime.time` object. This argument cannot be used alone, it should be supplemented with either `date` argument, or `year` … `day` tuple.

Returns

H2OFrame with one column containing the date constructed from the provided arguments.

Examples

```
>>> df = h2o.create_frame(rows=10,
...                       cols=3,
...                       factors=10,
...                       categorical_fraction=1.0/3,
...                       time_fraction=1.0/3,
...                       real_fraction=1.0/3,
...                       real_range=100,
...                       missing_fraction=0.0,
...                       seed=123)
>>> df["C1"].moment(year=df["C1"].year(),
...                 month=df["C1"].month(),
...                 day=df["C1"].day(),
...                 hour=df["C1"].hour(),
...                 minute=df["C1"].minute(),
...                 second=df["C1"].second())
```

`month` () [source]

Extract the “month” part from a date column.

Returns

a single-column H2OFrame containing the “month” part from the source frame.

Examples

```
>>> df = h2o.create_frame(rows=10,
...                       cols=3,
...                       factors=10,
...                       categorical_fraction=1.0/3,
...                       time_fraction=1.0/3,
...                       real_fraction=1.0/3,
...                       real_range=100,
...                       missing_fraction=0.0,
...                       seed=123)
>>> df["C1"].month()
```

`mult` (*matrix*) [source]

Multiply this frame, viewed as a matrix, by another matrix.

Parameters

matrix – another frame that you want to multiply the current frame by; must be compatible with the current frame (i.e. its number of rows must be the same as number of columns in the current frame).

Returns

new H2OFrame, which is the result of multiplying the current frame by `matrix`.

Examples

```
>>> data = [[random.uniform(-10000,10000)] for c in range(100)]
>>> h2o_data = h2o.H2OFrame(data)
>>> h2o_mm = h2o_data.mult(h2o_data.transpose())
```

`na_omit` () [source]

Remove rows with NAs from the H2OFrame.

Returns

new H2OFrame with all rows from the original frame containing any NAs removed.

Examples

```
>>> iris = h2o.import_file("http://h2o-public-test-
data.s3.amazonaws.com/smalldata/iris/iris_wheader_NA_2.csv")
>>> iris
>>> newframe=iris.na_omit()
>>> newframe
```

`nacnt` () [source]

Count of NAs for each column in this H2OFrame.

Returns

A list of the na counts (one entry per column).

Examples

```
>>> iris = h2o.import_file("http://h2o-public-test-
data.s3.amazonaws.com/smalldata/iris/iris_wheader_NA_2.csv")
>>> iris.nacnt()
```

property `names`

The list of column names (List[str]).

Returns

The list of column names.

Examples

```
>>> iris = h2o.import_file("http://h2o-public-test-
data.s3.amazonaws.com/smalldata/iris/iris_wheader_NA_2.csv")
>>> iris.names
```

`nchar` () [source]

Count the length of each string in a single-column H2OFrame of string type.

Returns

A single-column H2OFrame containing the per-row character count.

Examples

```
>>> iris = h2o.import_file("http://h2o-public-test-
data.s3.amazonaws.com/smalldata/iris/iris_wheader_NA_2.csv")
>>> iris[4].nchar()
```

property `ncol`

Same as `self.ncols`.

Returns

Number of columns in the dataframe.

Examples

```
>>> iris = h2o.import_file("http://h2o-public-test-
data.s3.amazonaws.com/smalldata/iris/iris_wheader_NA_2.csv")
>>> iris.ncol
```

property `ncols`

Number of columns in the dataframe (int).

Returns

Number of columns in the dataframe.

Examples

```
>>> iris = h2o.import_file("http://h2o-public-test-
data.s3.amazonaws.com/smalldata/iris/iris_wheader_NA_2.csv")
>>> iris.ncols
```

`nlevels` () [source]

Get the number of factor levels for each categorical column.

Returns

A list of the number of levels per column.

Examples

```
>>> python_lists = np.random.randint(-2,2, (10000,2))
>>> h2oframe = h2o.H2OFrame(python_obj=python_lists,
...                         column_types=['enum', 'enum'])
>>> h2oframe.nlevels()
```

property `nrow`

Same as `self.nrows`.

Returns

Number of rows in the dataframe.

Examples

```
>>> iris = h2o.import_file("http://h2o-public-test-
data.s3.amazonaws.com/smalldata/iris/iris_wheader_NA_2.csv")
>>> iris.nrow
```

property `nrows`

Number of rows in the dataframe (int).

Returns

Number of rows in the dataframe.

```
>>> iris = h2o.import_file("http://h2o-public-test-
data.s3.amazonaws.com/smalldata/iris/iris_wheader_NA_2.csv")
>>> iris.nrows
```

`num_valid_substrings` (***path_to_words***) [source]

For each string, find the count of all possible substrings with 2 characters or more that are contained in the line-separated text file whose path is given.

Parameters

path_to_words (*str*) – Path to file that contains a line-separated list of strings considered valid.

Returns

An H2OFrame with the number of substrings that are contained in the given word list.

Examples

```
>>> path = "https://raw.githubusercontent.com/dwyl/english-words/master/words.txt"
# test empty strings
>>> string = h2o.H2OFrame.from_python([''],
...                                   column_types=['string'])
>>> enum = h2o.H2OFrame.from_python([''],
...                                 column_types=['enum'])
>>> string.num_valid_substrings(path)[0,0] == 0
>>> enum.num_valid_substrings(path)[0,0] == 0
```

`pivot` (***index***, ***column***, ***value***) [source]

Pivot the frame designated by the three columns: index, column, and value. Index and column should be of type enum, int, or time. For cases of multiple indexes for a column label, the aggregation method is to pick the first occurrence in the data frame.

Parameters

- **index** – Index is a column that will be the row label
- **column** – The labels for the columns in the pivoted Frame
- **value** – The column of values for the given index and column label

Returns

Returns a new H2OFrame with pivoted columns.

Examples

```
>>> df = h2o.create_frame(rows=1000000,
...                       cols=3,
...                       factors=10,
...                       categorical_fraction=1.0/3,
...                       time_fraction=1.0/3,
...                       real_fraction=1.0/3,
...                       real_range=100,
...                       missing_fraction=0.0,
...                       seed=1234)
>>> pdf = df.as_data_frame()
>>> ppdf = pdf.pivot(values="C3",index="C1",columns="C2")
>>> ppdf = ppdf.fillna(0.0)
>>> ppdfh2o = h2o.H2OFrame(ppdf)
>>> ppdfh2o
```

`pop` (***i***) [source]

Pop a column from the H2OFrame at index i.

Parameters

i – The index (int) or name (str) of the column to pop.

Returns

an H2OFrame containing the column dropped from the current frame; the current frame is modified in-place and loses the column.

Examples

```
>>> prostate = h2o.import_file("http://s3.amazonaws.com/h2o-public-test-
data/smalldata/prostate/prostate.csv.zip")
>>> nc = prostate.ncol
>>> prostate
>>> popped_col = prostate.pop(prostate.names[0])
>>> prostate
>>> popped_col
```

`prod` (*na_rm=False*) [source]

Compute the product of all values across all rows in a single column H2O frame. If you apply this command on a multi-column H2O frame, the answer may not be correct.

Parameters

na_rm (*bool*) – If True then NAs will be ignored during the computation.

Returns

product of all values in the frame (a float)

Examples

```
>>> import random
>>> import numpy as np
>>> data = [[random.uniform(1,10)] for c in range(10)]
>>> h2o_data = h2o.H2OFrame(data)
>>> np_data = np.array(data)
>>> h2o_data.prod(na_rm=True)
>>> np.prod(np_data)
```

`quantile` (*prob=None*, *combine_method='interpolate'*, *weights_column=None*) [source]

Compute quantiles.

Parameters

- **prob** (*List[float]*) – list of probabilities for which quantiles should be computed.

- **combine_method** (*str*) – for even samples this setting determines how to combine quantiles. This can be one of `"interpolate"`, `"average"`, `"low"`, `"high"`.
- **weights_column** – optional weights for each row. If not given, all rows are assumed to have equal importance. This parameter can be either the name of column containing the observation weights in this frame, or a single-column separate H2OFrame of observation weights.

Returns

a new H2OFrame containing the quantiles and probabilities.

Examples

```
>>> data = [[random.uniform(-10000,10000)] for c in range(1000)]
>>> h2o_data = h2o.H2OFrame(data)
>>> np_data = np.array(data)
>>> h2o_data.quantile(prob=None,
...                   combine_method='interpolate',
...                   weights_column=None)
```

`rank_within_group_by` (***group_by_cols***, ***sort_cols***, ***ascending=[]***, ***new_col_name='New_Rank_column'***, ***sort_cols_sorted=False***) [source]

This function will add a new column rank where the ranking is produced as follows:

1. Sorts the H2OFrame by columns sorted in by columns specified in group_by_cols and sort_cols in the directions specified by the ascending for the sort_cols. The sort directions for the group_by_cols are ascending only.

2. A new rank column is added to the frame which will contain a rank assignment performed next. The user can choose to assign a name to this new column. The default name is New_Rank_column.

3. For each groupby groups, a rank is assigned to the row starting from 1, 2, … to the end of that group.

4. If sort_cols_sorted is TRUE, a final sort on the frame will be performed frame according to the sort_cols and the sort directions in ascending. If sort_cols_sorted is FALSE (by default), the frame from step 3 will be returned as is with no extra sort. This may provide a small speedup if desired.

Parameters

- **group_by_cols** – The columns to group on (either a single column name/index, or a list of column names or column indices
- **sort_cols** – The columns to sort on (either a single column name/index, or a list of column names or column indices

- **ascending** – Optional Boolean array to denote sorting direction for each sorting column. True for ascending, False for descending. Default is ascending sort. Sort direction for enums will be ignored.
- **new_col_name** – Optional String to denote the new column names. Default to New_Rank_column.
- **sort_cols_sorted** – Optional Boolean to denote if the returned frame should be sorted according to sort_cols and sort directions specified in ascending. Default is False.

Returns

A new Frame with new rank (sorted by columns in sort_cols) column within the grouping specified by the group_by_cols.

Examples

```
>>> air = h2o.import_file("https://s3.amazonaws.com/h2o-airlines-
unpacked/allyears2k.csv")
# slice out all but the following five columns
>>> df = air[:, ["ArrDelay", "DepDelay", "Origin", "Dest", "Distance"]]
# group by "Distance" and sort by "Origin"
>>> ranked1 = df.rank_within_group_by(group_by_cols="Distance", sort_cols="Origin")
# group by "ArrDelay" and sort by "Origin"
>>> ranked2 = df.rank_within_group_by(group_by_cols="ArrDelay", sort_cols="Origin")
# group by "DepDelay" and sort by "Dest"
>>> ranked3 = df.rank_within_group_by(group_by_cols="DepDelay", sort_cols="Dest")
```

`rbind` (*data*) [source]

Append data to this frame row-wise.

Parameters

data – an H2OFrame or a list of H2OFrame's to be combined with current frame row-wise.

Returns

this H2OFrame with all frames in data appended row-wise.

Examples

```
>>> frame = h2o.import_file("https://s3.amazonaws.com/h2o-public-test-
data/smalldata/junit/cars.csv")
>>> nrows = frame.nrow
>>> nrows
>>> frame2 = frame.rbind(frame)
>>> nrows2 = frame2.nrow
>>> nrows2
```

refresh () [source]

Reload frame information from the backend H2O server.

Returns

Frame information from the backend H2O server.

Examples

```
>>> dataframe = {'A': [1,0,3,4],
...              'B': [5,6,-6, -1],
...              'C':[-4, -6, -7, 8]}
>>> frame = h2o.H2OFrame(dataframe)
>>> frame_asin = frame.asin()
>>> assert set(frame.names) ==
...           {"A", "B", "C"},
...            "Expected original colnames to remain the same after uniop operation"
>>> assert ["asin(%s)" % (name) for name in frame.names] ==
...         frame_asin.names,"Expected equal col names after",
...         " uniop operation"
>>> frame_asin.refresh()
```

relevel (*y*) [source]

Reorder levels of an H2O factor for one single column of a H2O frame

The levels of a factor are reordered such that the reference level is at level 0, all remaining levels are moved down as needed.

Parameters

y (*str*) – The reference level

Returns

New reordered factor column

Examples

```
>>> import numpy as np
>>> python_lists = np.random.randint(-5,5, (100, 2))
>>> h2oframe = h2o.H2OFrame(python_obj=python_lists)
>>> newFrame = h2oframe.asfactor()
>>> allLevels = newFrame.levels()
>>> lastLevels = len(allLevels[0])-1
>>> newZeroLevel = allLevels[0][lastLevels]
>>> newFrame[0] = newFrame[0].relevel(newZeroLevel)
>>> newLevels = newFrame.levels()
>>> newLevels
```

rename (*columns=None*) [source]

Change names of columns in the frame.

Dict key is an index or name of the column whose name is to be set. Dict value is the new name of the column.

Parameters

columns – dict-like transformations to apply to the column names

Returns

Renamed columns

Examples

```
>>> iris = h2o.import_file("http://h2o-public-test-
data.s3.amazonaws.com/smalldata/iris/iris.csv")
>>> iris
>>> name = iris.rename(columns={'C2':'C1',
...                             'C1':'C2',
...                             'C3':'X3',
...                             'F0':'X0',
...                             'C3':'Y3'})
>>> name
```

`rep_len` (***length_out***) [source]

Create a new frame replicating the current frame.

If the source frame has a single column, then the new frame will be replicating rows and its dimensions will be `length_out x 1`. However if the source frame has more than 1 column, then then new frame will be replicating data in columnwise direction, and its dimensions will be `nrows x length_out`, where `nrows` is the number of rows in the source frame. Also note that if `length_out` is smaller than the corresponding dimension of the source frame, then the new frame will actually be a truncated version of the original.

Parameters

length_out (*int*) – Number of columns (rows) of the resulting H2OFrame

Returns

new H2OFrame with repeated data from the current frame.

Examples

```
>>> from random import randrange
>>> import numpy as np
>>> import math
>>> row_num = randrange(1,10)
>>> col_num = randrange(1,10)
>>> length_out_r = math.ceil(0.78*row_num)
>>> python_lists = np.random.randint(-5,5, (row_num, col_num))
>>> h2oframe = h2o.H2OFrame(python_obj=python_lists)
>>> h2oframe
>>> one_column = h2oframe[0].rep_len(length_out=(length_out_r+row_num))
>>> one_column
```

`round` (*digits=0*) [source]

Round doubles/floats to the given number of decimal places.

Parameters

digits (*int*) – The number of decimal places to retain. Rounding to a negative number of decimal places is not supported. For rounding we use the "round half to even" mode (IEC 60559 standard), so that `round(2.5) = 2` and `round(3.5) = 4`.

Returns

new H2OFrame with rounded values from the original frame.

Examples

```
>>> data = [[0.2348, 1.2380, 8.9032134],
...         [4.321321, 4.907432, 6.3]]
>>> h2o_data = h2o.H2OFrame(data)
>>> h2o_data.round(digits = 4)
>>> h2o_data.round(digits = 0)
```

`rstrip` (*set=' '*) [source]

Return a copy of the column with trailing characters removed.

The set argument is a string specifying the set of characters to be removed. If omitted, the set argument defaults to removing whitespace.

Parameters

set (*character*) – The set of characters to rstrip from strings in column

Returns

a new H2OFrame with the same shape as the original frame and having all its values trimmed from the right (equivalent of Python's `str.rstrip()`).

Examples

```
>>> iris = h2o.import_file("http://h2o-public-test-
data.s3.amazonaws.com/smalldata/iris/iris.csv")
>>> iris.levels()
>>> iris["C5"] = iris["C5"].rstrip("color")
>>> iris["C5"].levels()[0]
```

runif (*seed=None*) [source]

Generate a column of random numbers drawn from a uniform distribution [0,1) and having the same data layout as the source frame.

Parameters

seed (*int*) – seed for the random number generator.

Returns

Single-column H2OFrame filled with doubles sampled uniformly from [0,1).

Examples

```
>>> import numpy as np
>>> python_lists = np.random.uniform(0,1, 10000)
>>> h2oframe = h2o.H2OFrame(python_obj=python_lists)
>>> h2oframe.runif(seed=None)
```

save (*path*, *force=True*) [source]

Store frame data in H2O's native format.

This will store this frame's data to a file-system location in H2O's native binary format. Stored data can be loaded only with a cluster of the same size and same version the the one which wrote the data. The provided directory must be accessible from all nodes (HDFS, NFS).

Parameters

- **path** – a filesystem location where to write frame data
- **force** – overwrite already existing files (defaults to true)

Returns

Frame data as a string in csv format.

Examples

```
>>> iris = h2o.import_file("http://h2o-public-test-
data.s3.amazonaws.com/smalldata/iris/iris_wheader.csv")
>>> iris.save("hdfs://namenode/h2o_data")
```

`save_to_hive` (*jdbc_url*, *table_name*, *format='csv'*, *table_path=None*, *tmp_path=None*) [source]

Save contents of this data frame into a Hive table.

Parameters

- **jdbc_url** – Hive JDBC connection URL.
- **table_name** – Table name into which to store the data. The table must not exist as it will be created to match the structure of the the frame. The user must be allowed to create tables.
- **format** – Storage format of created Hive table, can be either `csv` (default) or `parquet`.
- **table_path** – If specified, the table will be created as an external table and this is where the data will be stored.
- **tmp_path** – Path where to store temporary data.

Examples

```
>>> airlines= h2o.import_file("https://s3.amazonaws.com/h2o-public-test-
data/smalldata/airlines/allyears2k_headers.zip")
>>> airlines["Year"] = airlines["Year"].asfactor()
>>> airlines.save_to_hive("jdbc:hive2://hive-server:10000/default", "airlines")
```

`scale` (*center=True*, *scale=True*) [source]

Center and/or scale the columns of the current frame.

Parameters

- **center** – If True, then demean the data. If False, no shifting is done. If `center` is a list of numbers then shift each column by the corresponding amount.
- **scale** – If True, then scale the data by each column's standard deviation. If False, no scaling is done. If `scale` is a list of numbers, then scale each column by the requested amount.

Returns

an H2OFrame with scaled values from the current frame.

Examples

```
>>> import numpy as np
>>> python_lists = np.random.uniform(1, 10, (10000,2))
>>> h2oframe = h2o.H2OFrame(python_obj=python_lists)
>>> h2oframe
>>> newframe = h2oframe.scale(center=True, scale=True)
>>> newframe
```

sd (*na_rm=False*) [source]

Compute the standard deviation for each column in the frame.

Parameters

na_rm (*bool*) – if True, then NAs will be removed from the computation.

Returns

A list containing the standard deviation for each column (NaN for non-numeric columns).

Examples

```
>>> import numpy as np
>>> python_lists = np.random.uniform(1, 10, (10000,2))
>>> h2oframe = h2o.H2OFrame(python_obj=python_lists)
>>> newframe = h2oframe.scale(center=True, scale=True)
>>> frameMean = newframe.mean()
>>> newframe.sd()
```

second () [source]

Extract the “second” part from a date column.

Returns

a single-column H2OFrame containing the “second” part from the source frame.

Examples

```
>>> df = h2o.create_frame(rows=10,
...                       cols=3,
...                       factors=10,
...                       categorical_fraction=1.0/3,
...                       time_fraction=1.0/3,
...                       real_fraction=1.0/3,
...                       real_range=100,
...                       missing_fraction=0.0,
...                       seed=123)
>>> df["C1"].second()
```

`set_level` (*level*) [source]

A method to set all column values to one of the levels.

Parameters

level (*str*) – The level at which the column will be set (a string)

Returns

H2OFrame with entries set to the desired level.

Examples

```
>>> import numpy as np
>>> import random
>>> python_lists = np.random.randint(-5,5, (10000, 2))
>>> h2oframe = h2o.H2OFrame(python_obj=python_lists)
>>> newFrame = h2oframe.asfactor()
>>> allLevels = newFrame.levels()
>>> lastLevel = allLevels[0][len(allLevels[0])-1]
>>> newFrame[0] = newFrame[0].set_level(level=lastLevel)
>>> firstLevel = allLevels[1][0]
>>> newFrame[1] = newFrame[1].set_level(level=firstLevel)
>>> newFrame
```

`set_levels` (*levels*) [source]

Replace the levels of a categorical column.

New levels must be aligned with the old domain. This call has copy-on-write semantics.

Parameters

levels (*List[str]*) – A list of strings specifying the new levels. The number of new levels must match the number of old levels.

Returns

A single-column H2OFrame with the desired levels.

Examples

```
>>> import numpy as np
>>> import random
>>> python_lists = np.random.randint(-5,5, (10000, 2))
>>> h2oframe = h2o.H2OFrame(python_obj=python_lists)
>>> newFrame = h2oframe.asfactor()
>>> allLevels = newFrame.levels()
>>> newLevel0 = random.sample(allLevels[0], len(allLevels[0]))
>>> newLevel1 = random.sample(allLevels[1], len(allLevels[1]))
>>> newFrame[0] = newFrame[0].set_levels(levels=newLevel0)
>>> newFrame[1] = newFrame[1].set_levels(levels=newLevel1)
>>> newFrame
```

`set_name` (***col=None***, ***name=None***) [source]

Set a new name for a column.

Parameters

- **col** – index or name of the column whose name is to be set; may be skipped for 1-column frames
- **name** – the new name of the column

Returns

The renamed column.

Examples

```
>>> import numpy as np
>>> import random
>>> row_num = random.randrange(1,10)
>>> col_num = random.randrange(1,10)
>>> python_lists = np.random.randint(-5,5, (row_num, col_num))
>>> h2oframe = h2o.H2OFrame(python_obj=python_lists)
>>> newNames = random.sample(h2oframe.names, col_num)
>>> h2oframe.set_names(names=newNames)
>>> newName = "Dolphine"
>>> h2oframe.set_name(col=0, name=newName)
>>> h2oframe
```

`set_names` (***names***) [source]

Change names of all columns in the frame.

Parameters

names (*List[str]*) – The list of new names for every column in the frame.

Returns

Frame with all new column names.

Examples

```
>>> import numpy as np
>>> import random
>>> row_num = random.randrange(1,10)
>>> col_num = random.randrange(1,10)
>>> python_lists = np.random.randint(-5,5, (row_num, col_num))
>>> h2oframe = h2o.H2OFrame(python_obj=python_lists)
>>> newNames = random.sample(h2oframe.names, col_num)
>>> h2oframe.set_names(names=newNames)
```

property `shape`

Number of rows and columns in the dataframe as a tuple `(nrows, ncols)`.

Returns

Number of rows and columns in the dataframe as a tuple

Examples

```
>>> iris = h2o.import_file("http://h2o-public-test-
data.s3.amazonaws.com/smalldata/iris/iris_wheader.csv")
>>> iris = iris[:, 0:4]
>>> iris.shape
```

`show` (***use_pandas=False***, ***rows=10***, ***cols=200***) [source]

Used by the H2OFrame.__repr__ method to print or display a snippet of the data frame.

If called from IPython, displays the results in HTML format. Otherwise, this prints a tabulated result.

Returns

snippet of the data frame.

Examples

```
>>> from random import randrange
>>> import numpy as np
>>> row_num = randrange(1,10)
>>> col_num = randrange(1,10)
>>> python_lists = np.random.randint(-5,5, (row_num,col_num))
>>> h20frame = h2o.H2OFrame(python_obj=python_lists)
>>> h2oframe.show(use_pandas=False)
>>> h2oframe.show(use_pandas=True)
```

`sign` () [source]

Return new H2OFrame equal to signs of the values in the frame: -1 , +1, or 0.

Returns

New H2OFrame equal to signs of the values in the frame: -1, +1, or 0.

Examples

```
>>> iris = h2o.import_file("http://h2o-public-test-
data.s3.amazonaws.com/smalldata/iris/iris_wheader.csv")
>>> iris.sign()
```

signif (*digits=6*) [source]

Round doubles/floats to the given number of significant digits.

Parameters

digits (*int*) – Number of significant digits to retain.

Returns

new H2OFrame with rounded values from the original frame.

Examples

```
>>> data = [[0.2348, 1.2380, 8.9032134],
...         [4.321321, 4.907432, 6.3]]
>>> h2o_data = h2o.H2OFrame(data)
>>> h2o_data
>>> h2o_data.signif(digits = 2)
```

sin () [source]

Create a new H2OFrame equal to elementwise sine of the current frame.

Returns

New H2OFrame equal to elementwise sine of the current frame.

Examples

```
>>> python_obj = [1, 2, 2.5, -100.9, 0]
>>> frame = h2o.H2OFrame(python_obj)
>>> frame.sin()
```

sinh () [source]

Create a new H2OFrame equal to elementwise hyperbolic sine of the current frame.

Returns

New H2OFrame equal to elementwise hyperbolic sine of the current frame.

Examples

```
>>> python_obj = [1,2,2.5,-100.9,0]
>>> frame = h2o.H2OFrame(python_obj)
>>> frame.sinh()
```

`sinpi` **()** [source]

Create a new H2OFrame equal to elementwise sine of the current frame multiplied by Pi.

Returns

New H2OFrame equal to elementwise sine of the current frame multiplied by Pi.

Examples

```
>>> python_obj = [1, 2, 2.5, -100.9, 0]
>>> frame = h2o.H2OFrame(python_obj)
>>> frame.sinpi()
```

`skewness` (*na_rm=False*) [source]

Compute the skewness of each column in the frame.

Parameters

na_rm (*bool*) – If True, then ignore NAs during the computation.

Returns

A list containing the skewness for each column (NaN for non-numeric columns).

Examples

```
>>> import numpy as np
>>> python_lists = np.random.uniform(-1,1, (10000,2))
>>> h2oframe = h2o.H2OFrame(python_obj=python_lists)
>>> h2oframe.skewness()
```

`sort` (*by*, *ascending=[]*) [source]

Return a new Frame that is sorted by column(s) in ascending order. A fully distributed and parallel sort. However, the original frame can contain String columns but sorting cannot be done on String columns. Default sorting direction is ascending.

Parameters

- **by** – The column to sort by (either a single column name, or a list of column names, or a list of column indices)
- **ascending** – Boolean array to denote sorting direction for each sorting column. True for ascending sort and False for descending sort.

Returns

a new sorted Frame

Examples

```
>>> df = h2o.create_frame(rows=10,
...                       cols=3,
...                       factors=10,
...                       categorical_fraction=1.0/3,
...                       time_fraction=1.0/3,
...                       real_fraction=1.0/3,
...                       real_range=100,
...                       missing_fraction=0.0,
...                       seed=123)
>>> df.sort("C1")
```

`split_frame` ***(ratios=None, destination_frames=None, seed=None)*** [source]

Split a frame into distinct subsets of size determined by the given ratios.

The number of subsets is always 1 more than the number of ratios given. Note that this does not give an exact split. H2O is designed to be efficient on big data using a probabilistic splitting method rather than an exact split. For example when specifying a split of 0.75/0.25, H2O will produce a test/train split with an expected value of 0.75/0.25 rather than exactly 0.75/0.25. On small datasets, the sizes of the resulting splits will deviate from the expected value more than on big data, where they will be very close to exact.

Parameters

- **ratios** (*List[float]*) – The fractions of rows for each split.
- **destination_frames** (*List[str]*) – The names of the split frames.
- **seed** (*int*) – seed for the random number generator

Returns

A list of H2OFrames

Examples

```
>>> iris = h2o.import_file("http://h2o-public-test-
data.s3.amazonaws.com/smalldata/iris/iris_wheader.csv")
>>> iris
>>> train, valid = iris.split_frame(ratios=[.8])
>>> train
>>> valid
```

`sqrt` **()** [source]

Create a new H2OFrame equal to elementwise square root of the current frame.

Returns

New H2OFrame equal to elementwise square root of the current frame.

Examples

```
>>> python_obj = [1, 2, 2.5, -100.9, 0]
>>> frame = h2o.H2OFrame(python_obj)
>>> frame.sqrt()
```

`stratified_kfold_column` **(*n_folds=3*, *seed=-1*)** [source]

Build a fold assignment column with the constraint that each fold has the same class distribution as the fold column.

Parameters

- **n_folds** (*int*) – The number of folds to build.
- **seed** (*int*) – A seed for the random number generator.

Returns

A single column H2OFrame with the fold assignments.

Examples

```
>>> import numpy as np
>>> python_lists = np.random.randint(-3,3, (10000,2))
>>> h2oframe = h2o.H2OFrame(python_obj=python_lists).asfactor()
>>> h2oframe[1].stratified_kfold_column(n_folds=3, seed=-1)
```

`stratified_split` **(*test_frac=0.2*, *seed=-1*)** [source]

Construct a column that can be used to perform a random stratified split.

Parameters

- **test_frac** (*float*) – The fraction of rows that will belong to the “test”.
- **seed** (*int*) – The seed for the random number generator.

Returns

an H2OFrame having single categorical column with two levels: `"train"` and `"test"`.

Examples

```
>>> import numpy as np
>>> python_lists = np.random.randint(-3,3, (10000,2))
>>> h2oframe = h2o.H2OFrame(python_obj=python_lists).asfactor()
>>> h2oframe[1].stratified_split(test_frac=0.2, seed=-1)
```

`strdistance` **(*y*, *measure=None*, *compare_empty=True*)** [source]

Compute element-wise string distances between two H2OFrames. Both frames need to have the same shape and only contain string/factor columns.

Parameters

- **y** (*H2OFrame*) – A comparison frame.
- **measure** (*str*) –

  A string identifier indicating what string distance measure to use. Must be one of:

  - `"lv"`: Levenshtein distance
  - `"lcs"`: Longest common substring distance
  - `"qgram"`: q-gram distance
  - `"jaccard"`: Jaccard distance between q-gram profiles
  - `"jw"`: Jaro, or Jaro-Winker distance
  - `"soundex"`: Distance based on soundex encoding
- **compare_empty** – if set to FALSE, empty strings will be handled as NaNs

Returns

An H2OFrame of the matrix containing element-wise distance between the strings of this frame and `y`. The returned frame has the same shape as the input frames.

Examples

```
>>> x = h2o.H2OFrame.from_python(['Martha', 'Dwayne', 'Dixon'],
...                              column_types=['factor'])
>>> y = h2o.H2OFrame.from_python(['Marhta', 'Duane', 'Dicksonx'],
...                              column_types=['string'])
>>> x.strdistance(y, measure="jw")
```

`strsplit` (*pattern*) [source]

Split the strings in the target column on the given regular expression pattern.

Parameters

pattern (*str*) – The split pattern.

Returns

H2OFrame containing columns of the split strings.

Examples

```
>>> frame = h2o.import_file("http://h2o-public-test-
data.s3.amazonaws.com/smalldata/iris/iris.csv")
>>> frame["C5"].strsplit("-")
```

`structure` () [source]

Compactly display the internal structure of an H2OFrame.

Returns

Compact display of the internal structure of an H2OFrame.

Examples

```
>>> frame = h2o.import_file("http://h2o-public-test-
data.s3.amazonaws.com/smalldata/iris/iris.csv")
>>> frame.structure()
```

`sub` (*pattern*, *replacement*, *ignore_case=False*) [source]

Substitute the first occurrence of pattern in a string with replacement.

Parameters

- **pattern** (*str*) – A regular expression.
- **replacement** (*str*) – A replacement string.
- **ignore_case** (*bool*) – If True then pattern will match case-insensitively.

Returns

an H2OFrame with all values matching `pattern` replaced with `replacement`.

Examples

```
>>> frame = h2o.import_file("http://h2o-public-test-
data.s3.amazonaws.com/smalldata/iris/iris.csv")
>>> frame["C5"].sub('s', 'z', ignore_case=False)
```

`substring` (***start_index*, *end_index=None***) [source]

For each string, return a new string that is a substring of the original string.

If end_index is not specified, then the substring extends to the end of the original string. If the start_index is longer than the length of the string, or is greater than or equal to the end_index, an empty string is returned. Negative start_index is coerced to 0.

Parameters

- **start_index** (*int*) – The index of the original string at which to start the substring, inclusive.
- **end_index** (*int*) – The index of the original string at which to end the substring, exclusive.

Returns

An H2OFrame containing the specified substrings.

Examples

```
>>> frame = h2o.import_file("http://h2o-public-test-
data.s3.amazonaws.com/smalldata/iris/iris.csv")
>>> frame["C5"].substring(0,5)
```

`sum` (***skipna=True*, *axis=0*, ***kwargs***) [source]

Compute the frame's sum by-column (or by-row).

Parameters

- **skipna** (*bool*) – If True (default), then NAs are ignored during the computation. Otherwise presence of NAs renders the entire result NA.
- **axis** (*int*) – Direction of sum computation. If 0 (default), then sum is computed columnwise, and the result is a frame with 1 row and number of columns as in the original frame. If 1, then sum is computed rowwise and the result is a

frame with 1 column (called “sum”), and number of rows equal to the number of rows in the original frame. For row or column sums, the `return_frame` parameter must be True.

- **return_frame** (*bool*) – A boolean parameter that indicates whether to return an H2O frame or one single aggregated value. Default is False.

Returns

either an aggregated value with sum of values per-column (old semantic); or an H2OFrame containing sum of values per-column/per-row in the original frame (new semantic). The new semantic is triggered by either providing the `return_frame=True` parameter, or having the `general.allow_breaking_changed` config option turned on.

Examples

```
>>> from random import randrange
>>> import numpy as np
>>> row_num = randrange(1,10)
>>> col_num = randrange(1,10)
>>> python_lists = np.random.randint(-5,5,(row_num,col_num))
>>> h2oframe = h2o.H2OFrame(python_obj=python_lists)
>>> h2oframe.sum(skipna=False,axis=0)
```

`summary` (***return_data=False***) [source]

Display summary information about the frame.

Summary includes min/mean/max/sigma and other rollup data.

Parameters

return_data (*bool*) – Return a dictionary of the summary output

Returns

Summary of information about the frame

Examples

```
>>> frame = h2o.import_file("http://h2o-public-test-
data.s3.amazonaws.com/smalldata/iris/iris.csv")
>>> frame.summary()
```

`table` (***data2=None, dense=True***) [source]

Compute the counts of values appearing in a column, or co-occurence counts between two columns.

Parameters

- **data2** (*H2OFrame*) – An optional single column to aggregate counts by.
- **dense** (*bool*) – If True (default) then use dense representation, which lists only non-zero counts, 1 combination per row. Set to False to expand counts across all combinations.

Returns

H2OFrame of the counts at each combination of factor levels

Examples

```
>>> df = h2o.import_file("http://s3.amazonaws.com/h2o-public-test-
data/smalldata/prostate/prostate_cat.csv")
>>> df[['DPROS', 'RACE']].table(data2=None,dense=True)
```

`tail` (*rows=10*, *cols=200*) [source]

Return the last `rows` and `cols` of the frame as a new H2OFrame.

Parameters

- **rows** (*int*) – maximum number of rows to return
- **cols** (*int*) – maximum number of columns to return

Returns

a new H2OFrame cut from the bottom left corner of the current frame, and having dimensions at most `rows` x `cols`.

Examples

```
>>> from random import randrange
>>> import numpy as np
>>> row_num = randrange(2,10)
>>> col_num = randrange(2,10)
>>> python_lists = np.random.randint(-5,5, (row_num, col_num))
>>> h2oframe = h2o.H2OFrame(python_obj=python_lists)
>>> h2oframe
>>> new_row = randrange(1, row_num)
>>> new_col = randrange(1, col_num)
>>> h2oframe.tail(rows=new_row, cols=new_col)
```

`tan` () [source]

Create a new H2OFrame equal to elementwise tangent of the current frame.

Returns

New H2OFrame equal to elementwise tangent of the current frame.

Examples

```
>>> python_obj = [1, 2, 2.5, -100.9, 0]
>>> frame = h2o.H2OFrame(python_obj)
>>> frame.tan()
```

`tanh` **()** [source]

Create a new H2OFrame equal to elementwise hyperbolic tangent of the current frame.

Returns

New H2OFrame equal to elementwise hyperbolic tangent of the current frame.

Examples

```
>>> python_obj = [1, 2, 2.5, -100.9, 0]
>>> frame = h2o.H2OFrame(python_obj)
>>> frame.tanh()
```

`tanpi` **()** [source]

Create a new H2OFrame equal to elementwise tangent of the current frame multiplied by Pi.

Returns

New H2OFrame equal to elementwise tangent of the current frame multiplied by Pi.

Examples

```
>>> python_obj = [1, 2, 2.5, -100.9, 0]
>>> frame = h2o.H2OFrame(python_obj)
>>> frame.tanpi()
```

`tokenize` **(*split*)** [source]

Tokenize String

tokenize() is similar to strsplit(), the difference between them is that tokenize() will store the tokenized text into a single column making it easier for additional processing (filtering stop words, word2vec algo, …).

Parameters

split (*tokenize*) – The regular expression to tokenize on.

Returns

An H2OFrame with a single column representing the tokenized Strings. Original rows of the input DF are separated by NA.

Examples

```
>>> df1 = h2o.H2OFrame.from_python({'String':
...                                 [' this is a string ']})
>>> df1 = df1.ascharacter()
>>> df2 = h2o.H2OFrame.from_python({'String':
...                                 ['this is another string']})
>>> df2 = df2.ascharacter()
>>> df3 = h2o.H2OFrame.from_python({'String':
...                                 ['this is a longer string']})
>>> df3 = df3.ascharacter()
>>> df4 = h2o.H2OFrame.from_python({'String':
...                                 ['this is tall, this is taller']})
>>> df4 = df4.ascharacter()
>>> combined = df1.rbind([df2, df3, df4])
>>> combined
>>> tokenized = combined.tokenize(" ")
>>> tokenized.describe
```

`tolower` () [source]

Translate characters from upper to lower case for a particular column.

Returns

new H2OFrame with all strings in the current frame converted to the lowercase.

Examples

```
>>> frame = h2o.import_file("http://h2o-public-test-
data.s3.amazonaws.com/smalldata/iris/iris.csv")
>>> frame["C5"]
>>> frame["C5"].tolower()
```

`topN` (*column=0*, *nPercent=10*) [source]

Given a column name or one column index, a percent N, this function will return the top N% of the values of the column of a frame. The column must be a numerical column.

Parameters

- **column** – a string for column name or an integer index

- **nPercent** – a top percentage of the column values to return

Returns

a H2OFrame containing two columns. The first column contains the original row indices where the top values are extracted from. The second column contains the top nPercent values.

Examples

```
>>> import numpy as np
>>> from random import randint
>>> dataFrame = h2o.import_file("https://s3.amazonaws.com/h2o-public-test-data/bigdata/laptop/jira/TopBottomNRep4.csv.zip")
>>> topAnswer = h2o.import_file("https://s3.amazonaws.com/h2o-public-test-data/smalldata/jira/Top20Per.csv.zip")
>>> nPercentages = [1,2,3,4]
>>> frameNames = dataFrame.names
>>> tolerance=1e-12
>>> nsample=100
>>> nP = nPercentages[randint(0, len(nPercentages)-1)]
>>> colIndex = randint(0, len(frameNames)-2)
>>> dataFrame.topN(frameNames[colIndex], nP)
```

`topNBottomN` ***(column=0, nPercent=10, grabTopN=-1)*** [source]

Given a column name or one column index, a percent N, this function will return the top or bottom N% of the values of the column of a frame. The column must be a numerical column.

Parameters

- **column** – a string for column name or an integer index
- **nPercent** – a top or bottom percentage of the column values to return
- **grabTopN** – -1 to grab bottom N percent and 1 to grab top N percent

Returns

a H2OFrame containing two columns. The first column contains the original row indices where the top/bottom values are extracted from. The second column contains the values.

Examples

```
>>> import numpy as np
>>> from random import randint
>>> dataFrame = h2o.import_file("https://s3.amazonaws.com/h2o-public-test-
data/bigdata/laptop/jira/TopBottomNRep4.csv.zip")
>>> topAnswer = h2o.import_file("https://s3.amazonaws.com/h2o-public-test-
data/smalldata/jira/Top20Per.csv.zip")
>>> bottomAnswer = h2o.import_file("https://s3.amazonaws.com/h2o-public-test-
data/smalldata/jira/Bottom20Per.csv.zip")
>>> nPercentages = [1,2,3,4]
>>> frameNames = dataFrame.names
>>> tolerance=1e-12
>>> nsample=100
>>> nP = nPercentages[randint(0, len(nPercentages)-1)]
>>> colIndex = randint(0, len(frameNames)-2)
>>> dataFrame.topNBottomN(frameNames[colIndex], nP, grabTopN=1)
>>> dataFrame.topNBottomN(frameNames[colIndex], nP, grabTopN=-1)
```

toupper ()

[source]

Translate characters from lower to upper case for a particular column.

Returns

new H2OFrame with all strings in the current frame converted to the uppercase.

Examples

```
>>> frame = h2o.import_file("http://h2o-public-test-
data.s3.amazonaws.com/smalldata/iris/iris.csv")
>>> frame["C5"]
>>> frame["C5"].toupper()
```

transpose ()

[source]

Transpose rows and columns of this frame.

Returns

new H2OFrame where with rows/columns from the original frame transposed.

Examples

```
>>> from random import randrange
>>> import numpy as np
>>> row_num = randrange(1,10)
>>> col_num = randrange(1,10)
>>> python_lists = np.random.randint(-5,5, (row_num, col_num))
>>> h2oframe = h2o.H2OFrame(python_obj=python_lists)
>>> h2oframe.transpose()
```

`trigamma ()` [source]

Create a new H2OFrame equal to the elementwise trigamma function of the current frame.

Returns

new H2OFrame equal to elementwise trigamma function of the current frame.

Examples

```
>>> python_obj = [1, 2, 2.5, -100.9, 0]
>>> frame = h2o.H2OFrame(python_obj)
>>> frame.trigamma()
```

`trim ()` [source]

Trim white space on the left and right of strings in a single-column H2OFrame.

Returns

H2OFrame with trimmed strings.

Examples

```
>>> frame = h2o.import_file(("https://s3.amazonaws.com/h2o-public-test-
data/smalldata/junit/cars_trim.csv"),
...                         col_types=["string","numeric",
...                                    "numeric","numeric",
...                                    "numeric","numeric",
...                                    "numeric","numeric"])
>>> frame["name"].trim()
```

`trunc ()` [source]

Apply the numeric truncation function.

`trunc(x)` is the integer obtained from `x` by dropping its decimal tail. This is equal to `floor(x)` if `x` is positive, and `ceil(x)` if `x` is negative. Truncation is also called "rounding towards zero".

Returns

new H2OFrame of truncated values of the original frame.

Examples

```
>>> import math
>>> import numpy as np
>>> from random import randrange
>>> row_num = randrange(1,10)
>>> col_num = randrange(1,10)
>>> length_out_r = math.ceil(0.78*row_num)
>>> length_out_c = math.ceil(col_num*0.4)
>>> python_lists = np.random.randint(-5,5, (row_num, col_num))
>>> h2oframe = h2o.H2OFrame(python_obj=python_lists)
>>> h2oframe.trunc()
```

`type` (*col*) [source]

The type for the given column.

Parameters

col – either a name, or an index of the column to look up

Returns

type of the column, one of: `str`, `int`, `real`, `enum`, `time`, `bool`.

Raises

H2OValueError – if such column does not exist in the frame.

Examples

```
>>> iris = h2o.import_file("http://h2o-public-test-
data.s3.amazonaws.com/smalldata/iris/iris.csv")
>>> iris.type("C5")
```

property `types`

The dictionary of column name/type pairs.

Returns

Dictionary of column name/type pairs.

Examples

```
>>> iris = h2o.import_file("http://h2o-public-test-
data.s3.amazonaws.com/smalldata/iris/iris.csv")
>>> iris.types
```

`unique` (*include_nas=False*) [source]

Extract the unique values in the column.

Parameters

include_nas – If set to true, NAs are included. False (turned off) by default.

Returns

H2OFrame of just the unique values in the column.

Examples

```
>>> import numpy as np
>>> python_lists = np.random.randint(-5,5, (100,1))
>>> h2oframe = h2o.H2OFrame(python_obj=python_lists)
>>> h2oframe.unique()
```

`var` (***y=None, na_rm=False, use=None***) [source]

Compute the variance-covariance matrix of one or two H2OFrames.

Parameters

- **y** (*H2OFrame*) – If this parameter is given, then a covariance matrix between the columns of the target frame and the columns of `y` is computed. If this parameter is not provided then the covariance matrix of the target frame is returned. If target frame has just a single column, then return the scalar variance instead of the matrix. Single rows are treated as single columns.
- **use** (*str*) –

  A string indicating how to handle missing values. This could be one of the following:

  - `"everything"` : outputs NaNs whenever one of its contributing observations is missing
  - `"all.obs"` : presence of missing observations will throw an error
  - `"complete.obs"` : discards missing values along with all observations in their rows so that only complete observations are used
- **na_rm** (*bool*) – an alternative to `use` : when this is True then default value for `use` is `"everything"` ; and if False then default `use` is `"complete.obs"` . This parameter has no effect if `use` is given explicitly.

Returns

An H2OFrame of the covariance matrix of the columns of this frame (if `y` is not given), or with the columns of `y` (if `y` is given). However when this frame and `y` are both single rows or single columns, then the variance is returned as a scalar.

Examples

```
>>> iris = h2o.import_file("http://h2o-public-test-
data.s3.amazonaws.com/smalldata/iris/iris_wheader.csv")
>>> iris.var(y=iris, na_rm=True, use=None)
```

week ()

[source]

Extract the “week” part from a date column.

Returns

a single-column H2OFrame containing the “week” part from the source frame.

Examples

```
>>> df = h2o.create_frame(rows=10,
...                       cols=3,
...                       factors=10,
...                       categorical_fraction=1.0/3,
...                       time_fraction=1.0/3,
...                       real_fraction=1.0/3,
...                       real_range=100,
...                       missing_fraction=0.0,
...                       seed=123)
>>> df["C1"].week()
```

which ()

[source]

Compose the list of row indices for which the frame contains non-zero values.

Only applicable to integer single-column frames. Equivalent to comprehension `[index for index, value in enumerate(self) if value]`.

Returns

a new single-column H2OFrame containing indices of those rows in the original frame that contained non-zero values.

Examples

```
>>> import numpy as np
>>> python_lists = np.random.randint(1,5, (100,1))
>>> h2oframe = h2o.H2OFrame(python_obj=python_lists)
>>> h2oframe.which()
```

year ()

[source]

Extract the “year” part from a date column.

Returns

a single-column H2OFrame containing the “year” part from the source frame.

Examples

```
>>> df = h2o.create_frame(rows=10,
...                       cols=3,
...                       factors=10,
...                       categorical_fraction=1.0/3,
...                       time_fraction=1.0/3,
...                       real_fraction=1.0/3,
...                       real_range=100,
...                       missing_fraction=0.0,
...                       seed=123)
>>> df["C1"].year()
```

GroupBy

class `h2o.group_by.GroupBy` (*fr*, *by*) [source]

Bases: `object`

A class that represents the group by operation on an H2OFrame.

The returned groups are sorted by the natural group-by column sort.

Parameters

- **fr** (*H2OFrame*) – H2OFrame that you want the group by operation to be performed on.
- **by** – by can be a column name (str) or an index (int) of a single column, or a list for multiple columns denoting the set of columns to group by.

Sample usage:

```
>>> my_frame = h2o.import_file("http://h2o-public-test-
data.s3.amazonaws.com/smalldata/iris/iris_wheader.csv")
>>> grouped = my_frame.group_by(by=["sepal_len",
...                                 "sepal_wid"])
>>> grouped.sum(col="sepal_len",
...             na="all").mean(col="class", na="all").max()
>>> grouped.get_frame()
```

Any number of aggregations may be chained together in this manner. Note that once the aggregation operations are complete, calling the GroupBy object with a new set of aggregations will yield no effect. You must generate a new GroupBy object in order to apply a new aggregation on it. In addition, certain aggregations are only defined for numerical or categorical columns. An error will be thrown for calling aggregation on the wrong data types.

If no arguments are given to the aggregation (e.g. “max” in the above example), then it is assumed that the aggregation should apply to all columns but the group by columns. However, operations will not be performed on String columns. They will be skipped.

All GroupBy aggregations take parameter na, which controls treatment of NA values during the calculation. It can be one of:

- “all” (default) – any NAs are used in the calculation as-is; which usually results in the final result being NA too.
- “ignore” – NA entries are not included in calculations, but the total number of entries is taken as the total number of rows. For example, mean([1, 2, 3, nan], na=”ignore”) will produce 1.5. In addition, median([1, 2, 3, nan], na=”ignore”) will first sort the row as [nan, 1, 2, 3]. Next, the median is the mean of the two middle values in this case producing a median of 1.5.
- “rm” entries are skipped during the calculations, reducing the total effective count of entries. For example, mean([1, 2, 3, nan], na=”rm”) will produce 2. The median in this case will be 2 as the middle value.

Variance (var) and standard deviation (sd) are the sample (not population) statistics.

`count` **(*na='all'*)** [source]

Count the number of rows in each group of a GroupBy object.

Parameters

na (*str*) – one of ‘rm’, ‘ignore’ or ‘all’ (default).

Returns

the original GroupBy object (self), for ease of constructing chained operations.

Examples

```
>>> my_frame = h2o.import_file("http://h2o-public-test-
data.s3.amazonaws.com/smalldata/iris/iris_wheader.csv")
>>> grouped = my_frame.group_by(by=["sepal_len",
...                                 "sepal_wid"])
>>> grouped.count
>>> grouped.get_frame()
```

property `frame`

same as get_frame().

Examples

```
>>> my_frame = h2o.import_file("http://h2o-public-test-
data.s3.amazonaws.com/smalldata/iris/iris_wheader.csv")
>>> grouped = my_frame.group_by(by=["sepal_len", "sepal_wid"])
>>> grouped.frame
```

get_frame () [source]

Return the resulting H2OFrame containing the result(s) of aggregation(s) of the group by.

The number of rows denote the number of groups generated by the group by operation.

The number of columns depend on the number of aggregations performed, the number of columns specified in the col parameter. Generally, expect the number of columns to be

(len(col) of aggregation 0 + len(col) of aggregation 1 +…+ len(col) of aggregation n) x (number of groups of the GroupBy object) +1 (for group-by group names).

Note:

- the count aggregation only generates one column;
- if col is a str or int, len(col) = 1.

Returns

GroupBy H2OFrame

Examples

```
>>> my_frame = h2o.import_file("http://h2o-public-test-
data.s3.amazonaws.com/smalldata/iris/iris_wheader.csv")
>>> grouped = my_frame.group_by(by=["sepal_len", "sepal_wid"])
>>> grouped.get_frame()
```

max (*col=None*, *na='all'*) [source]

Calculate the maximum of each column specified in col for each group of a GroupBy object. If no col is given, compute the maximum among all numeric columns other than those being grouped on.

Parameters

- **col** – col can be None (default), a column name (str) or an index (int) of a single column, or a list for multiple columns
- **na** (*str*) – one of ‘rm’, ‘ignore’ or ‘all’ (default).

Returns

the original GroupBy object (self), for ease of constructing chained operations.

Examples

```
>>> my_frame = h2o.import_file("http://h2o-public-test-
data.s3.amazonaws.com/smalldata/iris/iris_wheader.csv")
>>> grouped = my_frame.group_by(by=["sepal_len", "sepal_wid"])
>>> grouped.sum(col="sepal_len",
...             na="all").mean(col="class", na="all").max()
>>> grouped.get_frame()
```

`mean` (*col=None, na='all'*) [source]

Calculate the mean of each column specified in col for each group of a GroupBy object. If no col is given, compute the mean among all numeric columns other than those being grouped on.

Parameters

- **col** – col can be None (default), a column name (str) or an index (int) of a single column, or a list for multiple columns
- **na** (*str*) – one of 'rm', 'ignore' or 'all' (default).

Returns

the original GroupBy object (self), for ease of constructing chained operations.

Examples

```
>>> my_frame = h2o.import_file("http://h2o-public-test-
data.s3.amazonaws.com/smalldata/iris/iris_wheader.csv")
>>> grouped = my_frame.group_by(by=["sepal_len", "sepal_wid"])
>>> grouped.sum(col="sepal_len",
...             na="all").mean(col="class", na="all").max()
>>> grouped.get_frame()
```

`median` (*col=None, na='all'*) [source]

Calculate the median of each column specified in col for each group of a GroupBy object. If no col is given, compute the median among all numeric columns other than those being grouped on.

Parameters

- **col** – col can be None (default), a column name (str) or an index (int) of a single column, or a list for multiple columns
- **na** (*str*) – one of 'rm', 'ignore' or 'all' (default).

Returns

the original GroupBy object (self), for ease of constructing chained operations.

Examples

```
>>> my_frame = h2o.import_file("http://h2o-public-test-
data.s3.amazonaws.com/smalldata/iris/iris_wheader.csv")
>>> grouped = my_frame.group_by(by=["sepal_len", "sepal_wid"])
>>> grouped.sum(col="sepal_len",
...             na="all").median(col="class", na="all").max()
>>> grouped.get_frame()
```

`min` (*col=None, na='all'*) [source]

Calculate the minimum of each column specified in col for each group of a GroupBy object. If no col is given, compute the minimum among all numeric columns other than those being grouped on.

Parameters

- **col** – col can be None (default), a column name (str) or an index (int) of a single column, or a list for multiple columns denoting the set of columns to group by.
- **na** (*str*) – one of 'rm', 'ignore' or 'all' (default).

Returns

the original GroupBy object (self), for ease of constructing chained operations.

Examples

```
>>> my_frame = h2o.import_file("http://h2o-public-test-
data.s3.amazonaws.com/smalldata/iris/iris_wheader.csv")
>>> grouped = my_frame.group_by(by=["sepal_len", "sepal_wid"])
>>> grouped.sum(col="sepal_len",
...             na="all").mean(col="class", na="all").min()
>>> grouped.get_frame()
```

`mode` (*col=None, na='all'*) [source]

Calculate the mode of each column specified in col for each group of a GroupBy object. If no col is given, compute the mode among all categorical columns other than those being grouped on.

Parameters

- **col** – col can be None (default), a column name (str) or an index (int) of a single column, or a list for multiple columns

- **na** (*str*) – one of ‘rm’, ‘ignore’ or ‘all’ (default).

Returns

the original GroupBy object (self), for ease of constructing chained operations.

Examples

```
>>> my_frame = h2o.import_file("http://h2o-public-test-
data.s3.amazonaws.com/smalldata/iris/iris_wheader.csv")
>>> grouped = my_frame.group_by(by=["sepal_len", "sepal_wid"])
>>> grouped.sum(col="sepal_len",
...             na="all").mode(col="class", na="all").max()
>>> grouped.get_frame()
```

`sd` (*col=None, na='all'*) [source]

Calculate the standard deviation of each column specified in col for each group of a GroupBy object. If no col is given, compute the standard deviation among all numeric columns other than those being grouped on.

Parameters

- **col** – col can be None (default), a column name (str) or an index (int) of a single column, or a list for multiple columns
- **na** (*str*) – one of ‘rm’, ‘ignore’ or ‘all’ (default).

Returns

the original GroupBy object (self), for ease of constructing chained operations.

Examples

```
>>> my_frame = h2o.import_file("http://h2o-public-test-
data.s3.amazonaws.com/smalldata/iris/iris_wheader.csv")
>>> grouped = my_frame.group_by(by=["sepal_len", "sepal_wid"])
>>> grouped.sd(col="sepal_len",
...             na="all").mean(col="class", na="all").max()
>>> grouped.get_frame()
```

`ss` (*col=None, na='all'*) [source]

Calculate the sum of squares of each column specified in col for each group of a GroupBy object. If no col is given, compute the sum of squares among all numeric columns other than those being grouped on.

Parameters

- **col** – col can be None (default), a column name (str) or an index (int) of a single column, or a list for multiple columns

- **na** (*str*) – one of ‘rm’, ‘ignore’ or ‘all’ (default).

Returns

the original GroupBy object (self), for ease of constructing chained operations.

Examples

```
>>> my_frame = h2o.import_file("http://h2o-public-test-
data.s3.amazonaws.com/smalldata/iris/iris_wheader.csv")
>>> grouped = my_frame.group_by(by=["sepal_len", "sepal_wid"])
>>> grouped.ss(col="sepal_len",
...             na="all").mean(col="class", na="all").max()
>>> grouped.get_frame()
```

sum (*col=None, na='all'*) [source]

Calculate the sum of each column specified in col for each group of a GroupBy object. If no col is given, compute the sum among all numeric columns other than those being grouped on.

Parameters

- **col** – col can be None (default), a column name (str) or an index (int) of a single column, or a list for multiple columns
- **na** (*str*) – one of ‘rm’, ‘ignore’ or ‘all’ (default).

Returns

the original GroupBy object (self), for ease of constructing chained operations.

Examples

```
>>> my_frame = h2o.import_file("http://h2o-public-test-
data.s3.amazonaws.com/smalldata/iris/iris_wheader.csv")
>>> grouped = my_frame.group_by(by=["sepal_len", "sepal_wid"])
>>> grouped.sum(col="sepal_len",
...             na="all").mean(col="class", na="all").max()
>>> grouped.get_frame()
```

var (*col=None, na='all'*) [source]

Calculate the variance of each column specified in col for each group of a GroupBy object. If no col is given, compute the variance among all numeric columns other than those being grouped on.

Parameters

- **col** – col can be None (default), a column name (str) or an index (int) of a single column, or a list for multiple columns

- **na** (*str*) – one of ‘rm’, ‘ignore’ or ‘all’ (default).

Returns

the original GroupBy object (self), for ease of constructing chained operations.

Examples

```
>>> my_frame = h2o.import_file("http://h2o-public-test-
data.s3.amazonaws.com/smalldata/iris/iris_wheader.csv")
>>> grouped = my_frame.group_by(by=["sepal_len", "sepal_wid"])
>>> grouped.var(col="sepal_len",
...             na="all").mean(col="class", na="all").max()
>>> grouped.get_frame()
```

TargetEncoder

Previous Next