H2O Training

1. Introduction to ML
2. Introduction to H2O
3. H2O Data Munging
4. Machine Learning Concepts
5. Sparkling Water
6. Productionalizing Models

H2O Data Munging in Python

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

Data Transfer between Python and H2O

`h2o.H2OFrame`

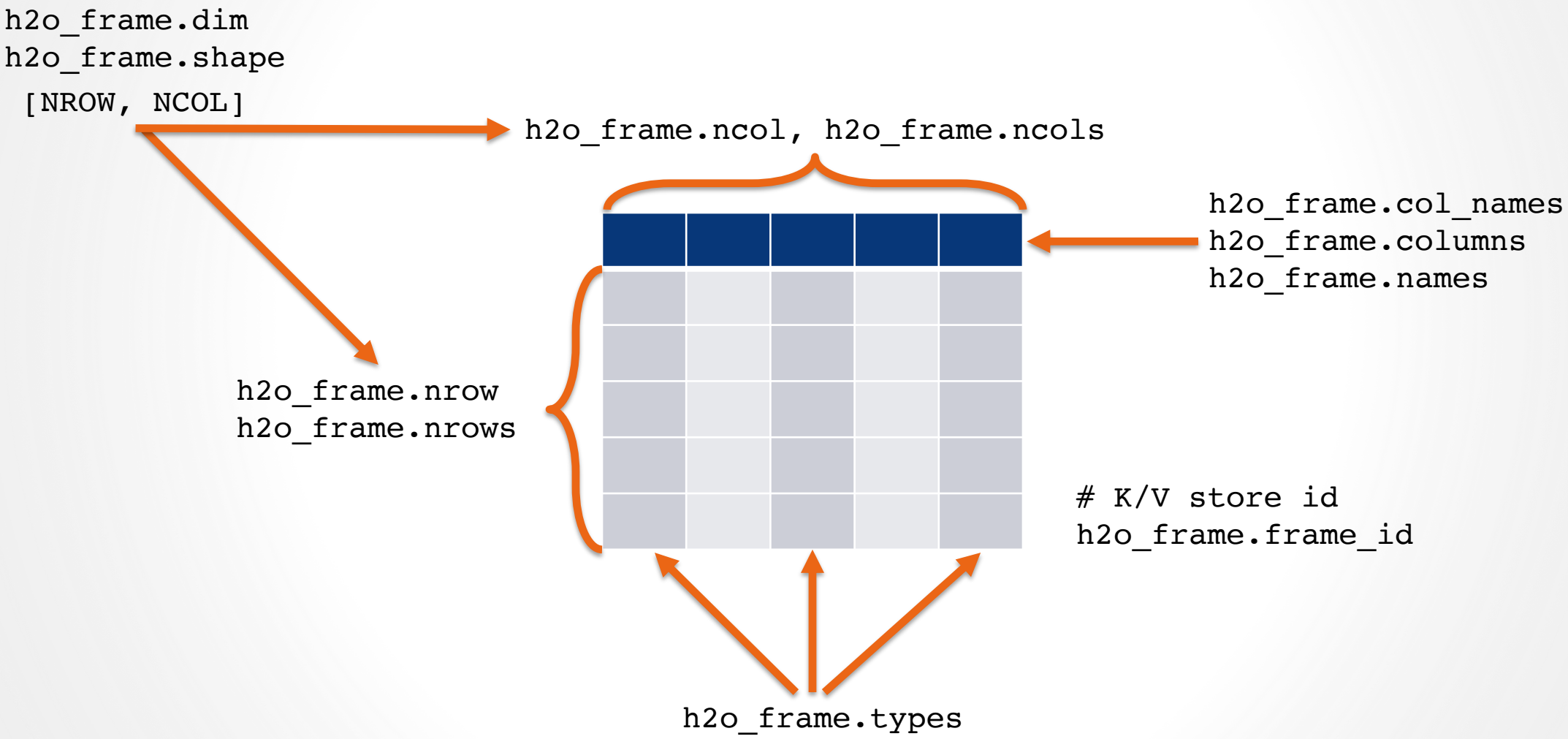
- Turns python pandas frame into an H2OFrame
- Saves dataframe to CSV and then performs `h2o.upload()` to H2O cluster

`h2o.as_list` OR `h2o_frame.as_data_frame`

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

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

H2OFrame Attributes



H2O Column Types

Enum

`data[x].asfactor()`

Used to convert given variable to an enumerated type variable

(booleans are coded as enum)

Numeric

`data[x].asnumeric()`

Used to convert given variable to an numeric variable

Dates

`data[x].year()`

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

`data[x].month()`

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

String

`data[x].ascharacter()`

Used to convert given variable to an string type variable

H2OFrame Example

```
1 import h2o
2 h2o.init()
3 data_path = "https://s3.amazonaws.com/h2o-public-test-data/smalldata/census_income/adult_data.csv"
4 census_data = h2o.import_file(data_path, destination_frame = "census_init")
```

```
Parse progress: |████████████████████████████████████████████████████████████████| 100%
```

```
1 print(census_data.types)
```

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

Row & Column Selection

Pandas-like convention for slicing and dicing data

(0-based indexes)

Extracting a single column

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

Filtering rows

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

Extracting multiple columns

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

Filtering rows for select columns

```
h2o_frame[0:50, [1,2,3]]

med = h2o_frame["a"].median()
h2o_frame[h2o_frame["a"] > med, "z"]
```

Filters & Logical Operations

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

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

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

Filters & Logical Operations

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

Arguments

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

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

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

Categorical Data Operations

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

Data Selection Example

```
1 tech_support_income = census_data[census_data["occupation"] == "Tech-support", "income"]
2 print(type(tech_support_income))
```

```
<class 'h2o.frame.H2OFrame'>
```

```
1 print(tech_support_income.table())
```

income	Count
<=50K	645
>50K	283

Missing Data Handling

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

Missing Data Handling

```
1 nacnts = dict(zip(census_data.col_names, census_data.nacnt()))
2 print(dict((k, int(v)) for (k, v) in nacnts.items() if v > 0))
```

```
{'workclass': 1836, 'occupation': 1843, 'native-country': 583}
```

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

native-country-clean	Count
0	31978
1	583

Summary & Aggregation

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

Arguments

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

Value: Returns a tabulated H2OFrame Object

Summary & Aggregation

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

Arguments

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

Numeric Data Summaries

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

Group By Aggregation

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

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

Numeric Data Transformations

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

Numeric Data Transformations

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

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

Date/Time Manipulations

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

String Munging

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

Joins Between Two H2OFrames

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

Arguments

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

Joins Between Two H2OFrames

```
census_data["cv_fold"] = census_data.kfold_column(n_folds = 5, seed = 123)
census_data["high_income"] = census_data["income"] == ">50K"
te_occupation = mean_target_encoding(census_data, x = "occupation", y = "high_income",
                                     fold_column = "cv_fold")
```

```
Parse progress: |████████████████████████████████████████████████████████████████| 100%
```

```
census_data = census_data.merge(te_occupation, all_x = True)
```

Histogram of Numeric Columns

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