# Machine Learning and Data Munging in H2O Driverless AI with datatable

Parul Pandey, Data Science Evangelist, H2O.ai

# H2O Product Suite

| H2O | Spark + H2O SPARKLING WATER | H2O4GPU |
|---|---|---|
| In-memory, distributed machine learning algorithms with H2O Flow GUI | H2O AI open source engine integration with Spark | Lightning fast machine learning on GPUs |

- 100% open source – Apache V2 licensed
- Built for data scientists – interface using R, Python or H2O Flow (interactive notebook interface)
- Enterprise support subscriptions

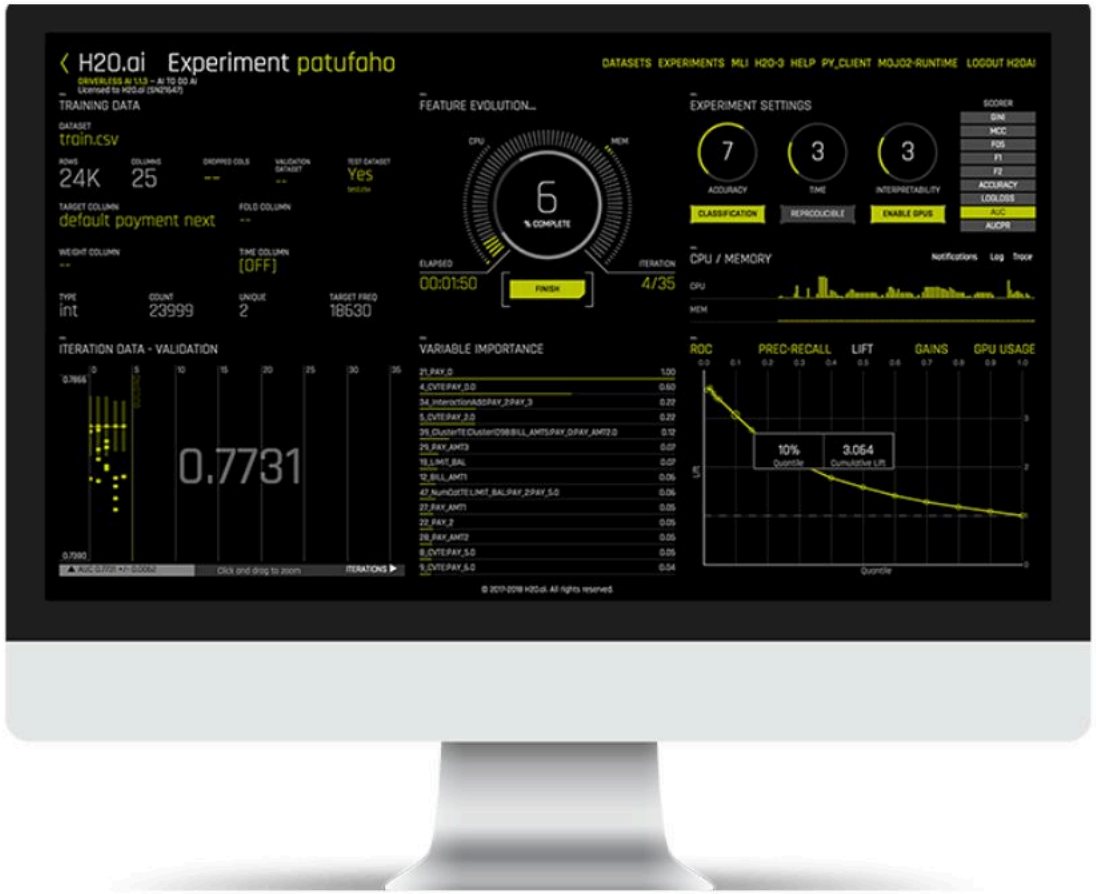
**DRIVERLESSAI**

Automatic feature engineering, machine learning and interpretability

- Fully automated machine learning from ingest to deployment
- User licenses on a per seat basis (annual subscription)

# H2O Driverless AI

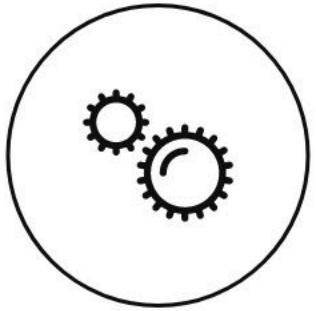
## Automatic Machine Learning Platform



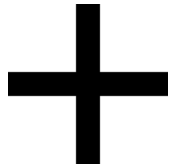

- Automatic Feature Engineering
- Custom Recipies
- Model Deployment and Operations
- Machine Learning Interpretability
- Automatic Visualization
- Natural Language Processing
- Time Series
- Flexibility of Data Ingestion and Compute Technologies

H2O.ai

# Driverless AI = Feature Engineering+ Model Fitting



Feature Engineering



Model Fitting

# DataMunging in Driverless AI

- Pandas
  - Feature Rich
  - Too slow for big data
  - memory hungry
  - limited support fro missing values

- pygdf: GPU Dataframe
  - very fast
  - data size limited by GPU memory
  - consumes GPU resources

H$_2$O.ai

# Data Munging in Driverless AI

# What is datatable anyways?

- R data.table is one of the top 10 most popular R packages
- Python datatable was started in 2017 in an  attempt to mimic the internal design  and API of R data.table
- It is a toolkit for performing big data operations on a single-node machine, at the maximum speed possible
- Fully Open sourced

# datatable capabilities

- CSV/binary reading/writing
- Multicolumn sorting/grouping
- Row filtering
- Column stats calculation, including by-group
- Frame joining/appending

# Load and view data

[1]:
```
from datatable import *
DT = fread("~/datasets/airlines_all.05p.csv")
DT
```

[1]:

| | Year | Month | DayofMonth | DayOfWeek | DepTime | CRSDepTime | ArrTime | CRSArrTime | UniqueCarrier | FlightNum | ... | NASDelay | Secu |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1988 | 1 | 9 | 6 | 1348 | 1331 | 1458 | 1435 | PI | 942 | ... | NA | |
| 1 | 1988 | 1 | 29 | 5 | 1339 | 1331 | 1442 | 1435 | PI | 942 | ... | NA | |
| 2 | 1988 | 1 | 23 | 6 | 950 | 950 | 1041 | 1050 | PI | 943 | ... | NA | |
| 3 | 1988 | 1 | 18 | 1 | 1124 | 1110 | 1213 | 1145 | PI | 943 | ... | NA | |
| 4 | 1988 | 1 | 10 | 7 | 1503 | 1500 | 1602 | 1550 | PI | 944 | ... | NA | |
| 5 | 1988 | 1 | 30 | 6 | 1500 | 1500 | 1558 | 1550 | PI | 944 | ... | NA | |
| 6 | 1988 | 1 | 20 | 3 | 1750 | 1705 | 1900 | 1810 | PI | 944 | ... | NA | |
| 7 | 1988 | 1 | 10 | 7 | 1616 | 1610 | 1632 | 1630 | PI | 944 | ... | NA | |
| 8 | 1988 | 1 | 30 | 6 | 1610 | 1610 | 1627 | 1630 | PI | 944 | ... | NA | |
| 9 | 1988 | 1 | 22 | 5 | 2026 | 2031 | 2135 | 2142 | PI | 945 | ... | NA | |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋱ | ⋮ | |
| 5,834,758 | 2008 | 12 | 13 | 6 | 1105 | 1115 | 1347 | 1431 | DL | 1505 | ... | NA | |
| 5,834,759 | 2008 | 12 | 13 | 6 | 1758 | 1800 | 1921 | 1921 | DL | 1534 | ... | NA | |
| 5,834,760 | 2008 | 12 | 13 | 6 | 1633 | 1635 | 1931 | 1926 | DL | 1559 | ... | NA | |
| 5,834,761 | 2008 | 12 | 13 | 6 | 1134 | 1134 | 1833 | 1855 | DL | 1594 | ... | NA | |
| 5,834,762 | 2008 | 12 | 13 | 6 | 1552 | 1520 | 1735 | 1718 | DL | 1620 | ... | 0 | |

5,834,763 rows × 31 columns

Shows progress bar while parsing

Type and size of each column

Integer columns with NAs are parsed as integer

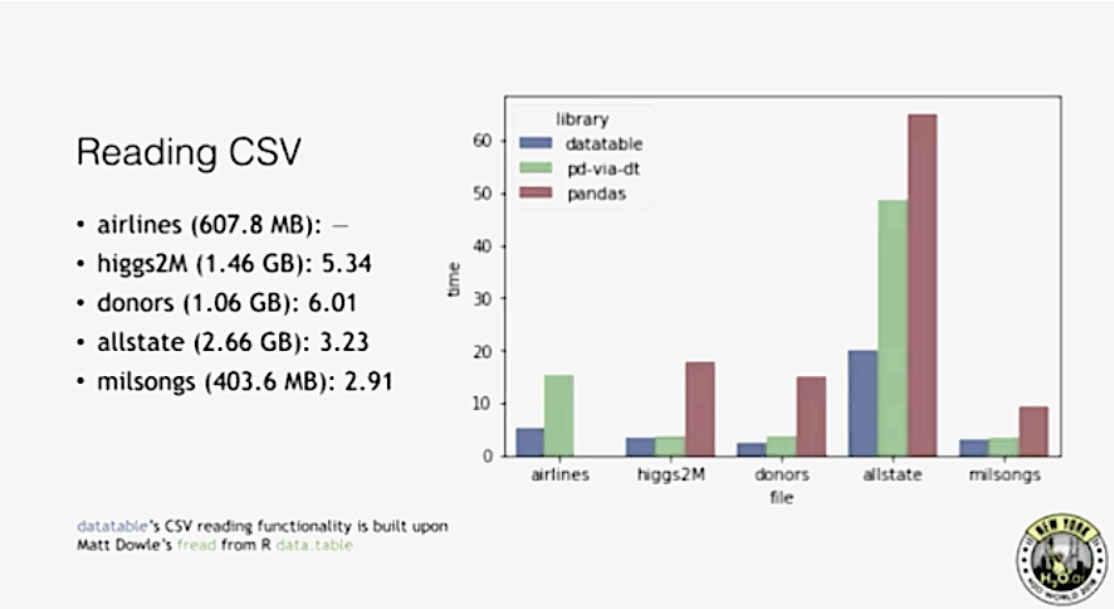
> 5x times faster than pandas.read_csv()

# fread: a doorway to Driverless AI

- A large portion of data is ingested into DAI through fread
- Automatically detects parse parameters
- Multi-threaded parsing
- Recovers from encoding errors
- Reads CSV and Excel files
- Reads files inside archives

# Performance



# Primary datatable syntax

```
DT[i, j, by(...),
      sort(...),
      join(...)]
```

```
SELECT j
FROM DT
JOIN join
WHERE|HAVING i
GROUP BY by
ORDER BY sort
```

# Examples

Find the average flight duration for each flight

```
[24]: AvgFlight = DT[:, mean(f.ArrTime - f.DepTime), by(f.FlightNum)]
```

Remove from DT all records where average flight duration is either negative or NA

```
[29]: del DT[(g.C0 < 0) | isna(g.C0), :, join(AvgFlight)]
```

For each carrier, select 3 longest flights

```
[34]: DT[:3, :, by(f.UniqueCarrier), sort(-f.AirTime)]
```

[25]:

| | FlightNum | C0 |
|---|---|---|
| 0 | 1 | 237.246 |
| 1 | 2 | 104.331 |
| 2 | 3 | 187.045 |
| 3 | 4 | 283.345 |
| 4 | 5 | 189.753 |
| 5 | 6 | 74.38 |
| 6 | 7 | 198.714 |
| 7 | 8 | 30.0504 |
| 8 | 9 | 135.452 |
| 9 | 10 | -66.6063 |
| ⋮ | ⋮ | ⋮ |
| 8065 | 9619 | 192 |
| 8066 | 9740 | 105 |
| 8067 | 9741 | NA |
| 8068 | 9761 | 310 |
| 8069 | 9912 | 108 |

8070 rows × 2 columns

# Machine Learning with datatable

**Microsoft Malware Prediction**

Can you predict if a machine will soon be hit with malware?

**$25,000**
Prize Money

Microsoft · 1,384 teams · a month to go (a month to go until merger deadline)

Overview Data Kernels Discussion Leaderboard Rules Team My Submissions **New Topic**

**olivier**
9th place

## H2O Light Fast Implementation of FTRL, 1 epoch 7sec

posted in Microsoft Malware Prediction a month ago

38

I'd like to share with you an amazing implementation of FTRL created by the guys at H2O.ai included in their port of datatable to python.

```
Start Fitting on    (8921483, 83)  @  2018-12-22 08:14:56.506164
Fitted complete on  (8921483, 83)  @  2018-12-22 08:15:03.645357
Current loss : 0.622195
```

Yep 7 seconds for 9 million rows, interested ?

https://www.kaggle.com/c/microsoft-malware-prediction/discussion/75478

# Python code example

```
import datatable as dt
from datatable.models import Ftrl

df = dt.fread('credit_card.csv')          # read data into a frame
X = df[:, :-1]                            # define training data
y = df[:, -1]                             # define target column
model = Ftrl(alpha=0.1, nbins=10**6)      # create an Ftrl object
model.fit(X, y)                           # train the model
fi = model.feature_importances            # get feature importances
p = model.predict(X)                      # make predictions
```

For detailed help please refer to https://datatable.readthedocs.io/en/latest/ftrl.html

# Resources

- https://datatable.readthedocs.io/en/latest/?badge=latest
- https://github.com/h2oai/datatable