RAPIDS

Cheat Sheet
www.RAPIDS.ai

TIDY DATA A foundation for wrangling in pandas

In a tidy data set:

Each **variable** is saved in its own **column**.

Each **observation** is saved in its own **row**.

Tidy data complements pandas' **vectorized operations**. pandas will automatically preserve observations as you manipulate variables. No other format works as intuitively.

SYNTAX Creating DataFrames

	a	b	c
1	4	7	10
2	5	8	11
3	6	9	12

```
gdf = cudf.DataFrame([
                ("a", [4 ,5, 6]),
                ("b", [7, 8, 9]),
                ("c", [10, 11, 12])
                    ])
```

Specify values for each column.

```
gdf = cudf.DataFrame.from_records(
                [[4, 7, 10],
                [5, 8, 11],
                [6, 9, 12]],
                index=[1, 2, 3],
                columns=['a', 'b', 'c'])
```

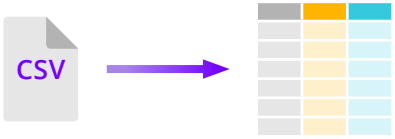
Specify values for each row.

METHOD CHAINING

Most pandas methods return a DataFrame so another pandas method can be applied to the result. This improves readability of code.

```
gdf = cudf.from_pandas(df)
            .query('val >= 200')
            .nlargest('val',3)
```

INGESTING AND RESHAPING DATA Change the layout of a data set




```
gdf = cuDF.read_csv(filename, delimiter=',',
                    names=col_names, dtype =col_types)
```

Planned for Future Release

```
df.pivot(columns='var', values='val')
```

spread rows into columns

gdf.sort_values('mpg')
Order rows by values of a column (low to high).

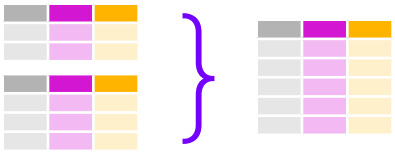
gdf.sort_values('mpg',ascending=False)
Order rows by values of a column (high to low).

Planned for Future Release

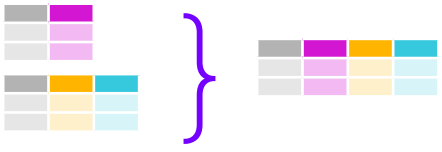
gdf.sort_index()
Sort the index of a DataFrame.

gdf.set_index()
Return a new DataFrame with a new index.

gdf.drop_column('Length')
Drop column from DataFrame.



cudf.concat([gdf1,gdf2])
Append rows of DataFrames.



gdf.add_column('name', gdf1['name'])
Append columns of DataFrames.

SUBSET OBSERVATIONS

gdf.query('Length > 7')
Extract rows that meet logical criteria.

```
df.drop_duplicates()
```

Remove duplicate rows (only considers columns)

Planned for Future Release

```
df.head(n)
df.tail(n)
```

Select first/last n rows

```
df.sample(frac=0.5)
```

Randomly select fraction of rows

```
df.sample(n=10)
```

Randomly select n rows

```
df.iloc[10:20]
```

Select rows by position

Planned for Future Release

gdf.nlargest(n, 'value')
Select and order top n entries.

gdf.nsmallest(n, 'value')
Select and order bottom n entries.

LOGIC IN PYTHON (AND PANDAS)

<	Less than	!=	Not equal to
>	Greater than	df.column.isin(values)	Group membership
==	Equals	pd.isnull(obj)	Is NaN
<=	Less than or equals	pd.notnull(obj)	Is not NaN
>=	Greater than or equals	&,|,~,^,df.any(),df.all()	Logical and, or, not, xor, any, all

SUBSET VARIABLES (COLUMNS)

gdf[['width','length','species']]
Select multiple columns with specific names.

gdf['width'] or gdf.width
Select single column with specific name.

```
df.filter(regex='regex')
```

Select columns whose name matches regular expression regex

REGEX (REGULAR EXPRESSIONS) EXAMPLES

'.'	Matches strings containing a period '.'
'Length$'	Matches strings ending with word 'Length'
'^Sepal'	Matches strings beginning with the word 'Sepal'
'^x[1-5]$'	Matches strings beginning with 'x' and ending with 1,2,3,4,5
'^(?!Species$).*'	Matches strings except the string 'Species'

Planned for Future Release

gdf.loc[2:5,['x2','x4']]
Get rows from index 2 to index 5 from 'a' and 'b' columns.

```
df.iloc[:,[1,2,5]]
```

Select columns in positions 1, 2 and 5 (first column is 0)

```
df.loc[df['a'] > 10, ['a','c']]
```

Select rows meeting logical condition, and only the specific columns

Planned for Future Release

SUMMARIZE DATA

gdf['w'].value_counts()
Count number of rows with each unique value of variable.

len(gdf)
# of rows in DataFrame.

gdf['w'].unique_count()
# of distinct values in a column.

Planned for Future Release

Pygdf provides a set of **summary functions** that operate on different kinds of pandas objects (DataFrame columns, Series, GroupBy) and produce single values for each of the groups. When applied to a DataFrame, the result is returned as a pandas Series for each column. Examples:

sum()
Sum values of each object.

count()
Count non-NA/null values of each object.

Planned for Future Release

applymap(function)
Apply function to each object.

min()
Minimum value in each object.

max()
Maximum value in each object.

mean()
Mean value of each object.

var()
Variance of each object.

std()
Standard deviation of each object.

GROUP DATA

gdf.groupby("col")
Return a GroupBy object, grouped by values in column named "col".

Planned for Future Release

All of the summary functions listed above can be applied to a group. Additional GroupBy functions:

Planned for Future Release

agg(function)
Aggregate group using function.

Planned for Future Release

HANDLING MISSING DATA

Planned for Future Release

gdf['length'].fillna(value)
Replace all NA/null data with value.

MAKE NEW COLUMNS

Planned for Future Release

gdf['Volume'] = gdf.Length*gdf.Height*gdf.Depth
Add single column.

Planned for Future Release

pandas provides a large set of **vector functions** that operate on all columns of a DataFrame or a single selected column (cuDF Series). These functions produce vectors of values for each of the columns, or a single Series for the individual Series. Examples:

max(axis=1)
Element-wise max.

clip(lower=-10,upper=10)
Trim values at input thresholds

min(axis=1)
Element-wise min.

abs()
Absolute value.

Define a kernal function:

```
>>> def kernel(in1, in2, in3, out1, out2, extra1, extra2):
        for i, (x, y, z) in enumerate(zip(in1, in2, in3)):
            out1[i] = extra2 * x - extra1 * y
            out2[i] = y - extra1 * z
```

Call the kernel with apply_rows:

```
>>> outdf = gdf.apply_rows(kernel,
                           incols=['in1', 'in2', 'in3'],
                           outcols=dict(out1=np.float64,
                                        out2=np.float64),
                           kwargs=dict(extra1=2.3, extra2=3.4))
```

WINDOWS

Planned for Future Release

ONE-HOT ENCODING

CuDF can convert pandas category data types into one-hot encoded or dummy variables easily.

```
pet_owner = [1, 2, 3, 4, 5]
pet_type = ['fish', 'dog', 'fish', 'bird', 'fish']
df = pd.DataFrame({'pet_owner': pet_owner, 'pet_type': pet_type})
df.pet_type = df.pet_type.astype('category')

my_gdf = cuDF.DataFrame.from_pandas(df)
my_gdf['pet_codes'] = my_gdf.pet_type.cat.codes

codes = my_gdf.pet_codes.unique()
enc_gdf = my_gdf.one_hot_encoding('pet_codes', 'pet_dummy', codes)
```

COMBINE DATA SETS

gdf1

x1	x2
A	1
B	2
C	3

+

gdf2

x1	x3
A	T
B	F
D	T

=

STANDARD JOINS

x1	x2	x3
A	1	T
B	2	F
C	3	NaN

gdf.merge(gdf2, how='left', on='x1')
Join matching rows from bdf to adf.

x1	x2	x3
A	1.0	T
B	2.0	F
D	NaN	T

gdf.merge(gdf1, gdf2, how='right', on='x1')
Join matching rows from gdf1 to gdf2.

x1	x2	x3
A	1	T
B	2	F

gdf.merge(gdf1, gdf2, how='inner', on='x1')
Join data. Retain only rows in both sets.

x1	x2	x3
A	1	T
B	2	F
C	3	NaN
D	NaN	T

gdf.merge(gdf1, gdf2, how='outer', on='x1')
Join data. Retain all values, all rows.

FILTERING JOINS

Planned for Future Release

gdf1

x1	x2
A	1
B	2
C	3

+

gdf2

x1	x2
B	2
C	3
D	4

=

SET-LIKE OPERATIONS

x1	x2
B	2
C	3

gdf.merge(gdf1, gdf2, how='inner')
Rows that appear in both ydf and zdf (Intersection).

x1	x2
A	1
B	2
C	3
D	4

gdf.merge(gdf1, gdf2, how='outer')
Rows that appear in either or both ydf and zdf (Union).

Planned for Future Release