Transforming Data

Reindert-Jan Ekker

@rjekker http://nl.linkedin.com/in/rjekker

Overview

Calculations

- Basic math operators
- Function application

Grouping and aggregation

Structural transformation

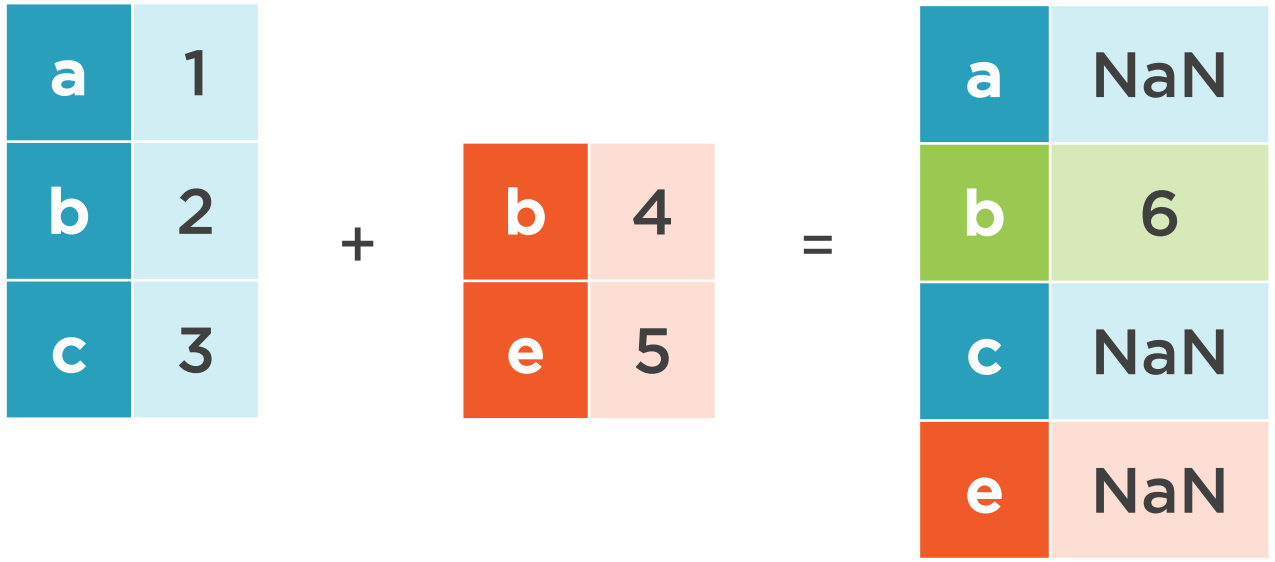
- Rows to columns
- Columns to rows

Combining datasets

Demo

Basic math operators

Function application

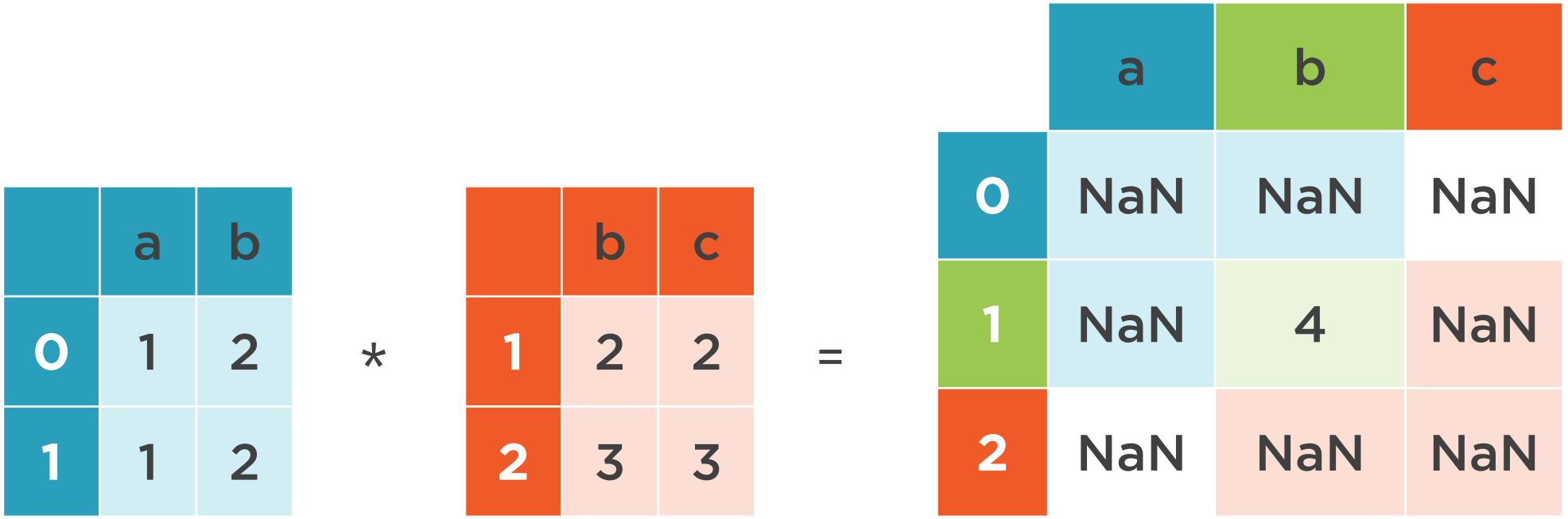


Operating on Two Series

Returns a new Series object

With all indices from both inputs

Results only filled where indices overlap; NaN everywhere else

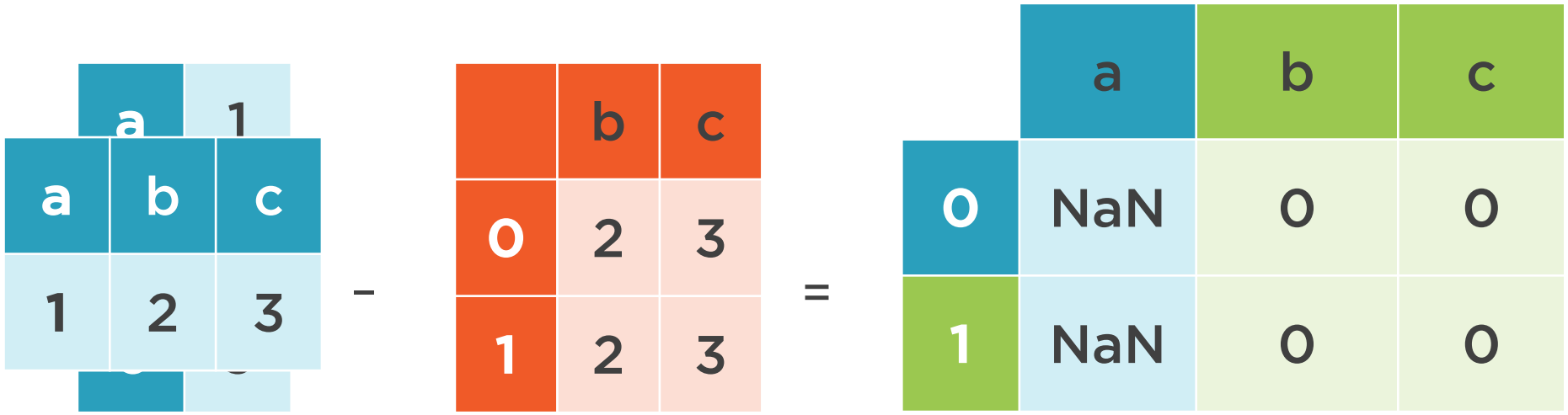


Operating on Two DataFrames

Returns a new DataFrame object

With all column and row labels from both inputs

Results only filled where indices overlap; NaN everywhere else



Operating on Series and DataFrame

Labels are matched on *columns*

Returns a DataFrame

Results only filled where indices overlap; NaN everywhere else

Binary Operator Functions

These support the axis argument

df.add(x): df + x df.radd(x): x + df	df.sub(x): df - x df.rsub(x): x - df	df.mul(x): df * x df.rmul(x): x * df

df.div(x): df / x
df.rdiv(x): x / df
	df.floordiv(x): df // x
	df.rfloordiv(x): x // df
		df.pow(x), df.rpow(x)
		df.mod(x), df.rmod(x)

```
# Compute the sine of all cells in df
np.sin(df)

# Compute e^x for every x in df
np.exp(df)
```

Numpy ufuncs

Functions that work on entire DataFrames/Series

Many mathematical operations, see https://goo.gl/ESR8nX

```
# DataFrame.applymap() applies a function to each cell
df.applymap(my_func)

# Series.apply() does the same for values in a Series
s.apply(my_func)
```

Applying Functions to Cells

Pass the function to `df.applymap()` – no parentheses!

Returns a new DataFrame with results

Equivalent function on a Series is called `apply()`

```
df.apply(f)  # apply f to every column of df

df.apply(sum)  # calculate sum of every column

df.apply(sum, axis=1)  # calculate sum of every row
```

Applying Functions to Rows/Columns

DataFrame.apply() and Series.apply() do different things!

DataFrame.apply() applies a function to entire rows/columns

Demo

Grouping and aggregation

```
g = df.groupby("class")
```

```
g.mean()
```

	class	name	grade
0	a	Ann	6
1	b	Bob	7
2	b	Joe	6
3	a	Eve	8
4	c	Kim	7

a	Ann	6
a	Eve	8

b	Bob	7
b	Joe	6

c	Kim	7

class	grade
a	*7*
b	*6.5*
c	*7*

Groupby

```
# Select one or more columns on the groupby object
# Before doing any actual calculations
athletes.groupby('sport')['gold'].sum()

# Group on multiple columns
athletes.groupby(['sport', 'nationality'])['gold'].sum()
```

Multi Level Index

Created by groupby

When used with multiple columns for grouping

```
athletes.groupby(['sport','sex']).min()
```

sport	sex	value
boxing	female	5
	male	6
cycling	female	8
	male	7
fencing	female	4
	male	4

```
count(), sum()
mean(), median()
std(), var()
min(), max()
first(), last()
```

Functions Available for Groupby Objects

Functions shown above are *optimized* for GroupBy objects

But you can call *all* methods on the underlying object (DF/Series)

Or use `GroupBy.apply()` to call your own function

Demo

Structural transformation

- Columns to rows
- Rows to columns
- stack(), unstack()
- pivot(), melt()

```
df.pivot("index", "columns", "values")
```

Pivot: Transforming One Column into Many

pivot() takes 3 arguments: index, columns, and values

Each of these takes a column name from the original DataFrame

Returns a DataFrame; rows and columns taken from *index* and *columns*, values taken from *values*

```
# Convert value column into 2 new columns: price, stock
df.pivot("prod_id", "item", "value")
```

	prod_id	item	value
0	1	price	11
1	2	price	9
2	2	stock	2
3	1	stock	50
4	3	price	7

	price	stock
1	11	50
2	9	2
3	7	NaN

```
df.melt(id_vars="prod_id")
```

Melt: Transforming Many Columns into One

Returns a DataFrame; Use id_vars to list columns that contain group identifiers

Column labels will go into "variable" column

All other values not set as id_vars will end up in "value" column

```
# prepare the DataFrame for melt()
df.reset_index().rename(columns={'index': 'prod_id'})
```

	price	stock
1	11	50
2	9	2
3	7	NaN

	prod_id	price	stock
0	1	11	50
1	2	9	2
2	3	7	NaN

```
df.melt(id_vars="prod_id")
```

	prod_id	price	stock
0	1	11	50
1	2	9	2
2	3	7	NaN

	prod_id	variable	value
0	1	price	11
1	2	price	9
2	3	price	7
3	1	stock	50
4	2	stock	2
5	3	stock	Nan

```
# stack() moves all data into 1 column
# with a multi level index
df.stack()
```

	price	stock
1	11	50
2	9	2
3	7	NaN

		value
1	price	11
	stock	50
2	price	9
	stock	2
3	price	7

```
# unstack() creates columns for the innermost index level
# and moves data from the rows into these columns
df.unstack()
```

		value
1	price	11
	stock	50
2	price	9
	stock	2
3	price	7

	price	stock
1	11	50
2	9	2
3	7	NaN

Demo

Combining datasets

Summary

Calculations

- Basic math operators
- Function application

Grouping and aggregation

Structural transformation

- Rows to columns
- Columns to rows

Combining datasets