MATLAB Programming Techniques

Terasoft

Application Engineer

Jeffrey Liu

Outline

- Programing Flow Control
  - Decision branching
  - For-loop
  - While-loop
- Structuring Code
  - Private function
  - Local function
  - Nested function
  - Precedence rule
- MATLAB data types
  - Tables
  - Categorical array
  - Timetables

Decision Branching

- Conditional branching can be achieved using the if-elseif-else construction

```
if (logical test 1)
    statements 1
elseif (logical test 2)
    statements 2
else
    statements 3
end
```

- If there is a finite set of discrete possible values for a variable, you can use the switch-case construction.

```
switch expression
    case value 1         % executes if
        statements 1     % expression == value 1
    case value 2
        statements 2
    otherwise            % "else"
        statements 3
end
```

For-Loops

- The loop index is regular MATLAB variable and can therefore be used in statement within the loop

```
for index = first:increment:last
    statements
end
```

- The loop index persists in memory (with the value of last) after the loop terminates.

```
v = [1,2,3,5,8,13];
for k = v
    statements
end
```

While-Loops

- The code contained in statements will be evaluated as long as the logical condition evaluates true

```
while condition
    statements
end
```

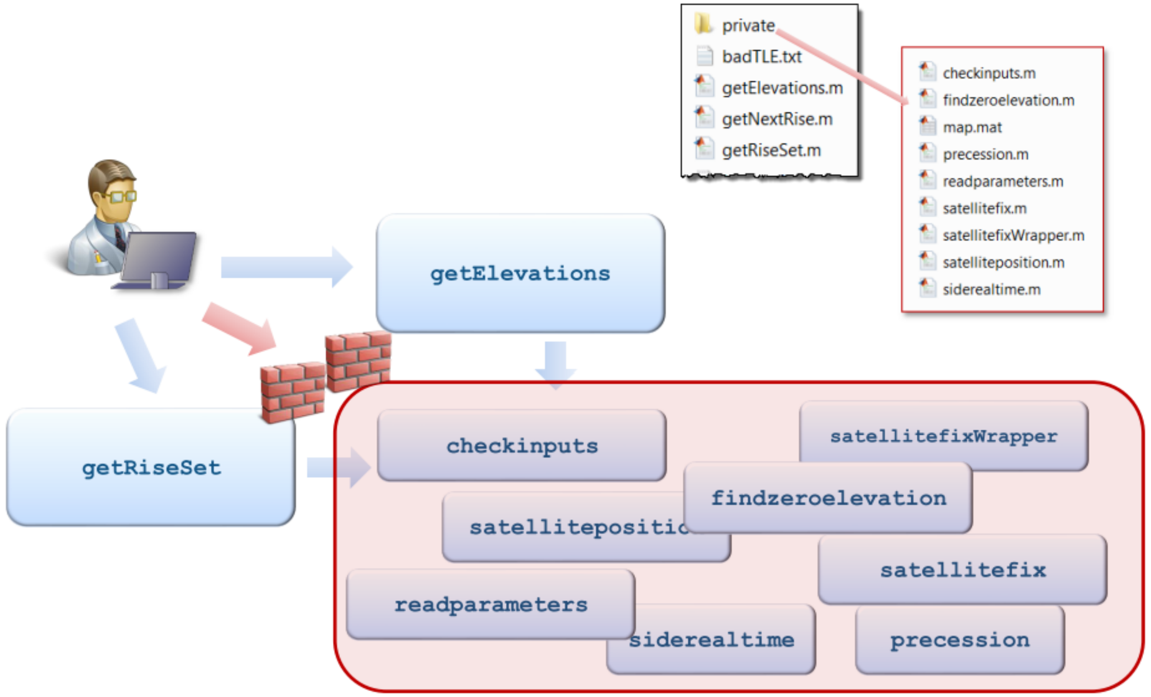
- A common problem when writing code with while-loop is the creation of infinite loops.
  - If it happens, execution can be interrupted by pressing **Ctrl+C**

```
x = 3;
while (x>2)
    x = 2*x;
end
```

Private Functions

- To make a private function
  1. Create a subfolder named private
  2. Move the function file into this subfolder

- Private function can be called:
  – Only by the function in the same folder or parent folder
  – From the command line only if the private folder is current folder

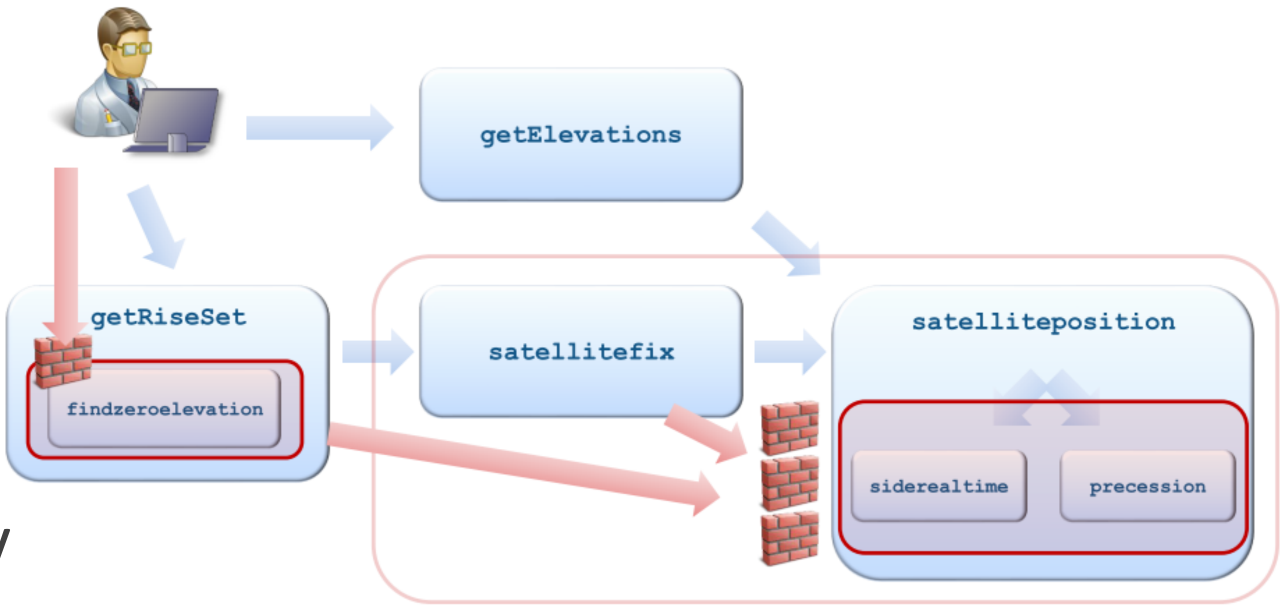


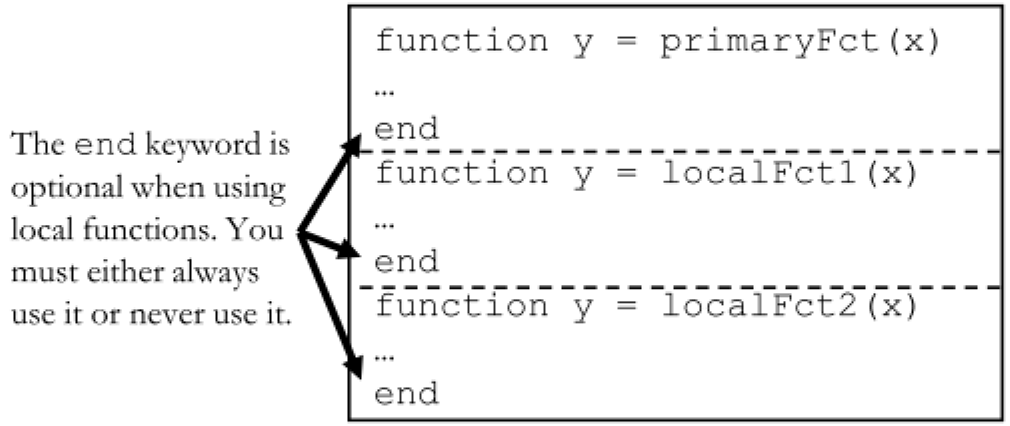
Local Functions (subfunctions)

- To make a local function
  - Enter the keyword function in the function file (also the end)

- Local function can be called only by its primary function

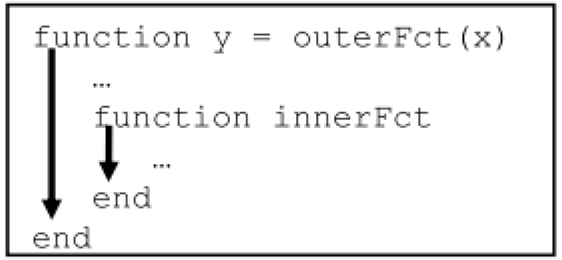
- Local function maintain their own separate workspace




The end keyword is optional when using local functions. You must either always use it or never use it.

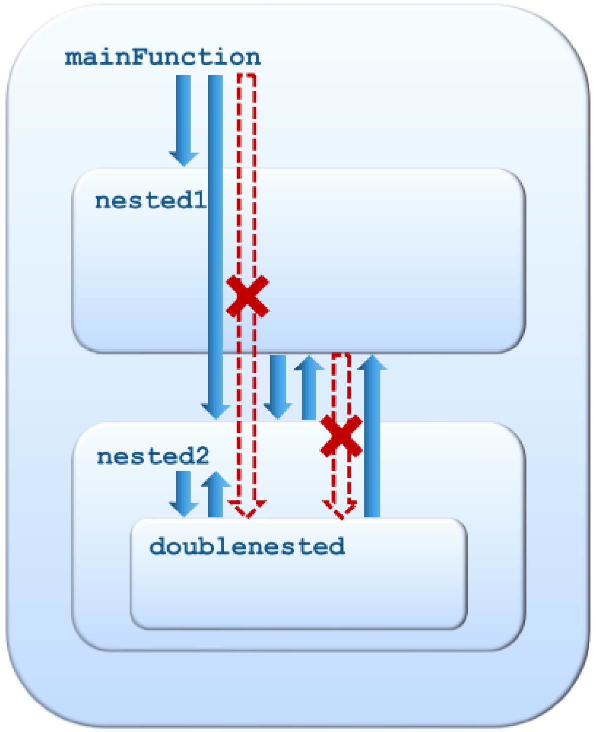
```
function y = primaryFct(x)
...
end
function y = localFct1(x)
...
end
function y = localFct2(x)
...
end
```

Nested functions

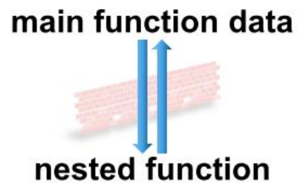


```
function y = outerFct(x)
    …
    function innerFct
        …
    end
end
```

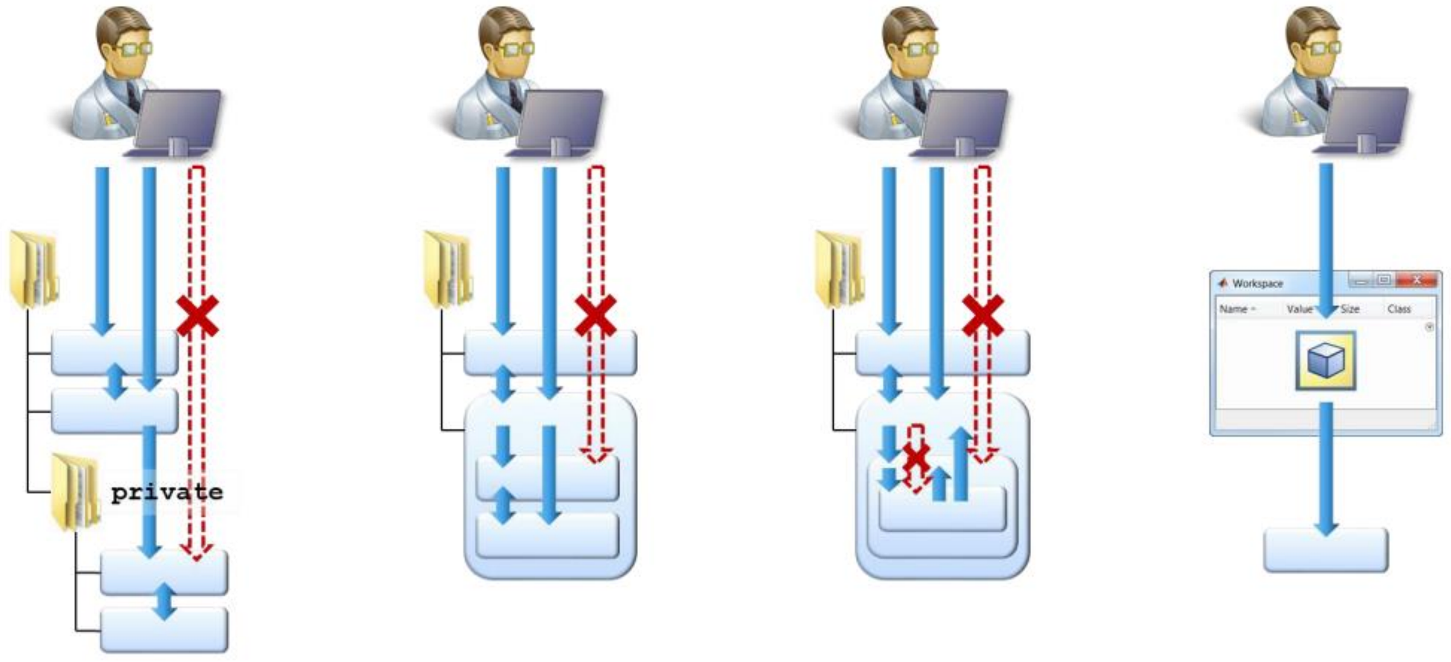
- To make a nested function:
  - The extends of the function must marked explicitly by using keyword function and end
- A nested function can be called from:
  - The level immediately above
  - A function at the same level within the same parent
  - A nested function at any lower level



getRiseSet
findzeroelevation



Comparison of function types

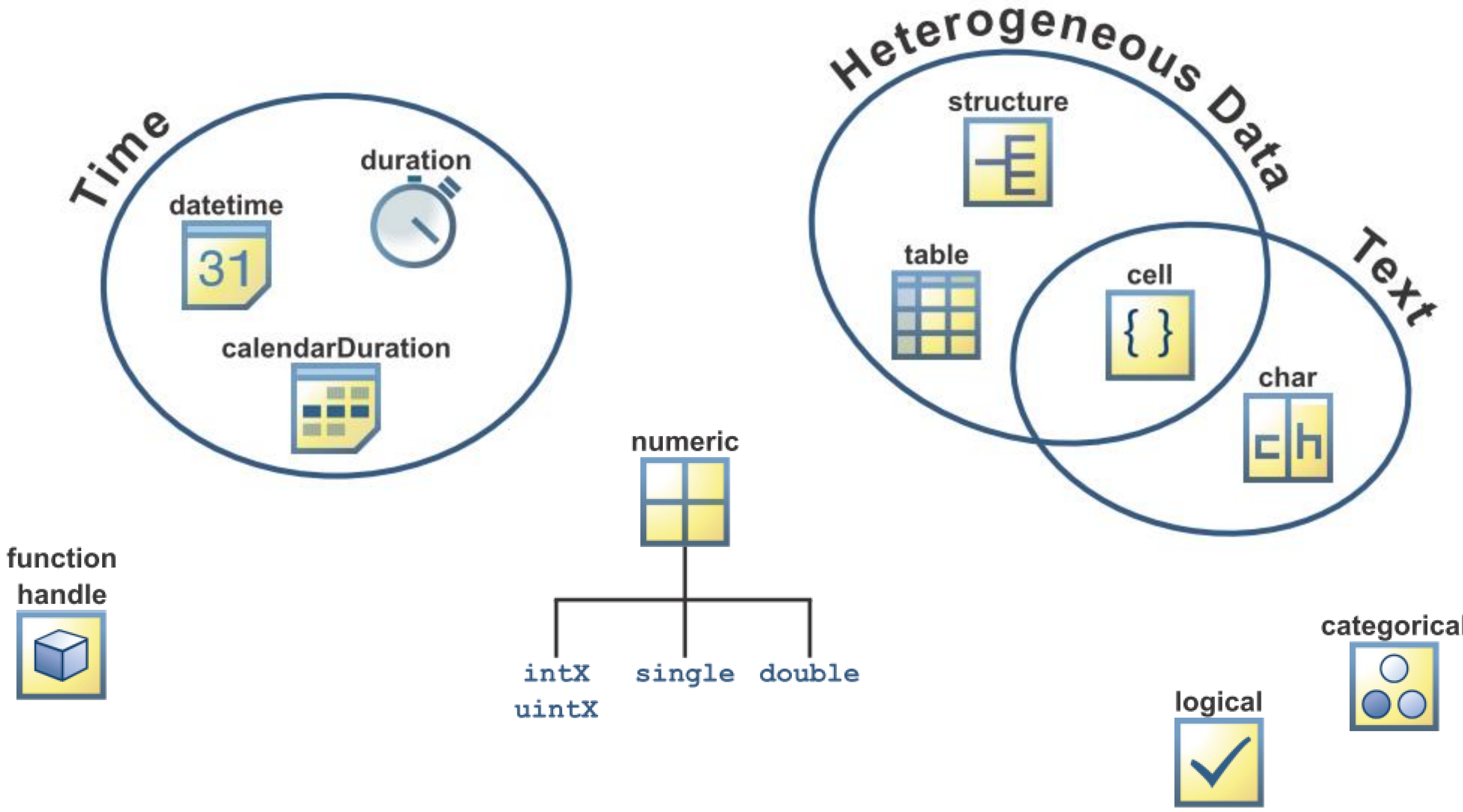


	Private	Local	Nested	Anonymous
File	Separate	Shared	Shared	None
Workspace	Separate	Separate	Shared	Variable
Access	Files in `private` and functions in parent folder	Within file	Within file • parent • sibling • any lower level	Via function handle variable
Typical use	Project specific functionality	Hide utilities	Share application data	Change of function interface

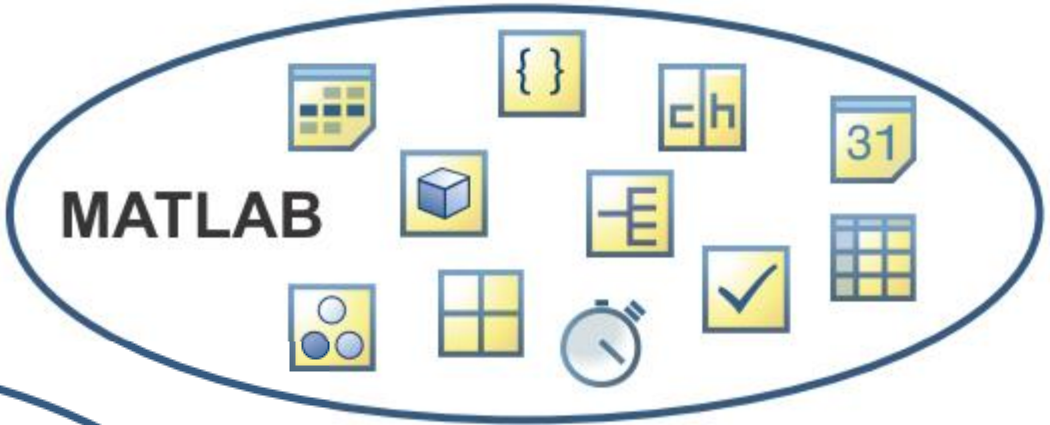
Precedence Rule

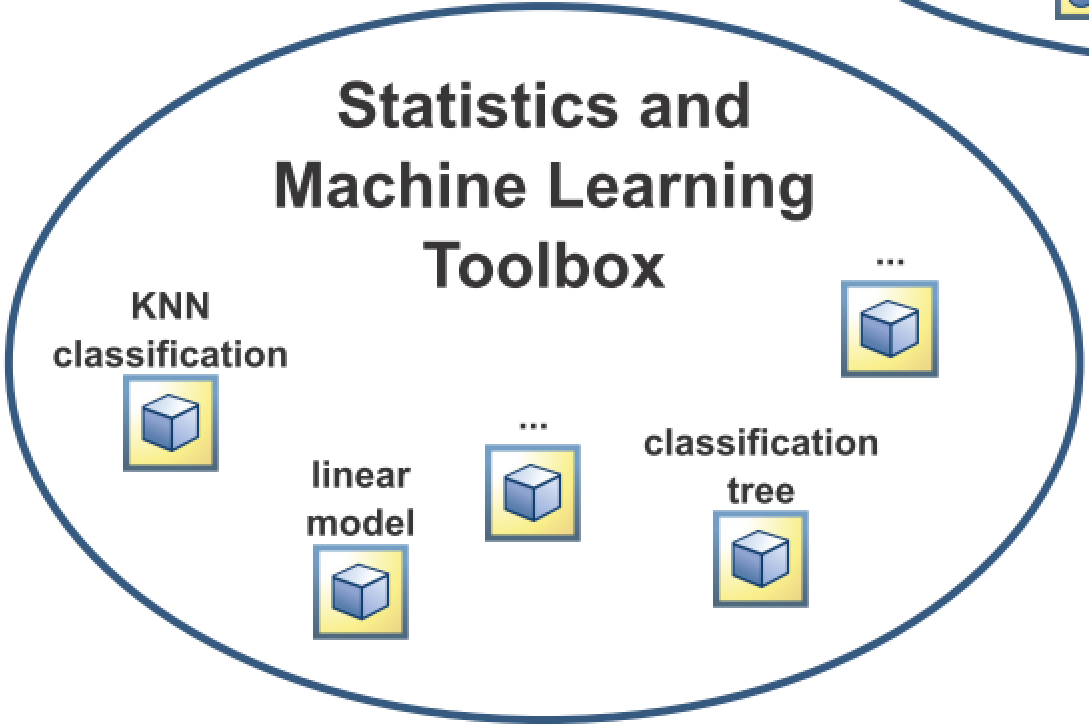
- It is important to understand how MATLAB decides what to do if it encounters conflicting identifiers, although you should aim at avoiding any such conflict.
- MATLAB results the following list of precedence:
  1. Variable in the local workspace
  2. Function from an imported package
  3. Nested function
  4. Local function
  5. Private function
  6. Class method
  7. Function in current folder
  8. Function on MATLAB path

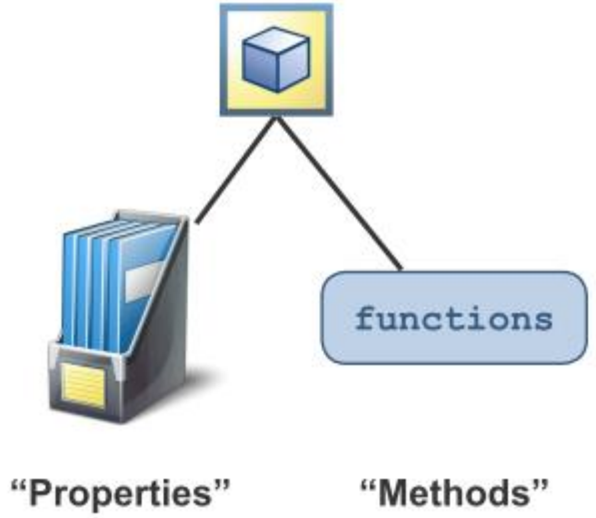
MATLAB data types



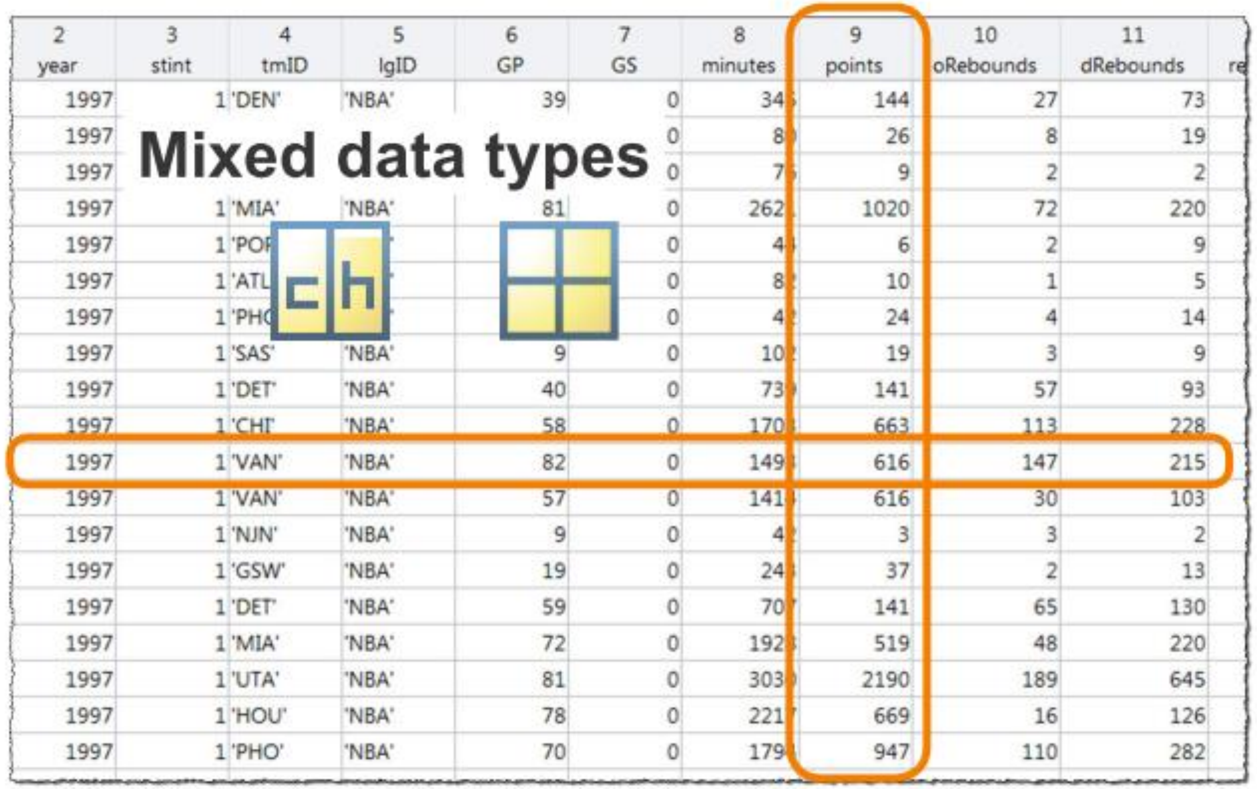
Specialized data types







Tables



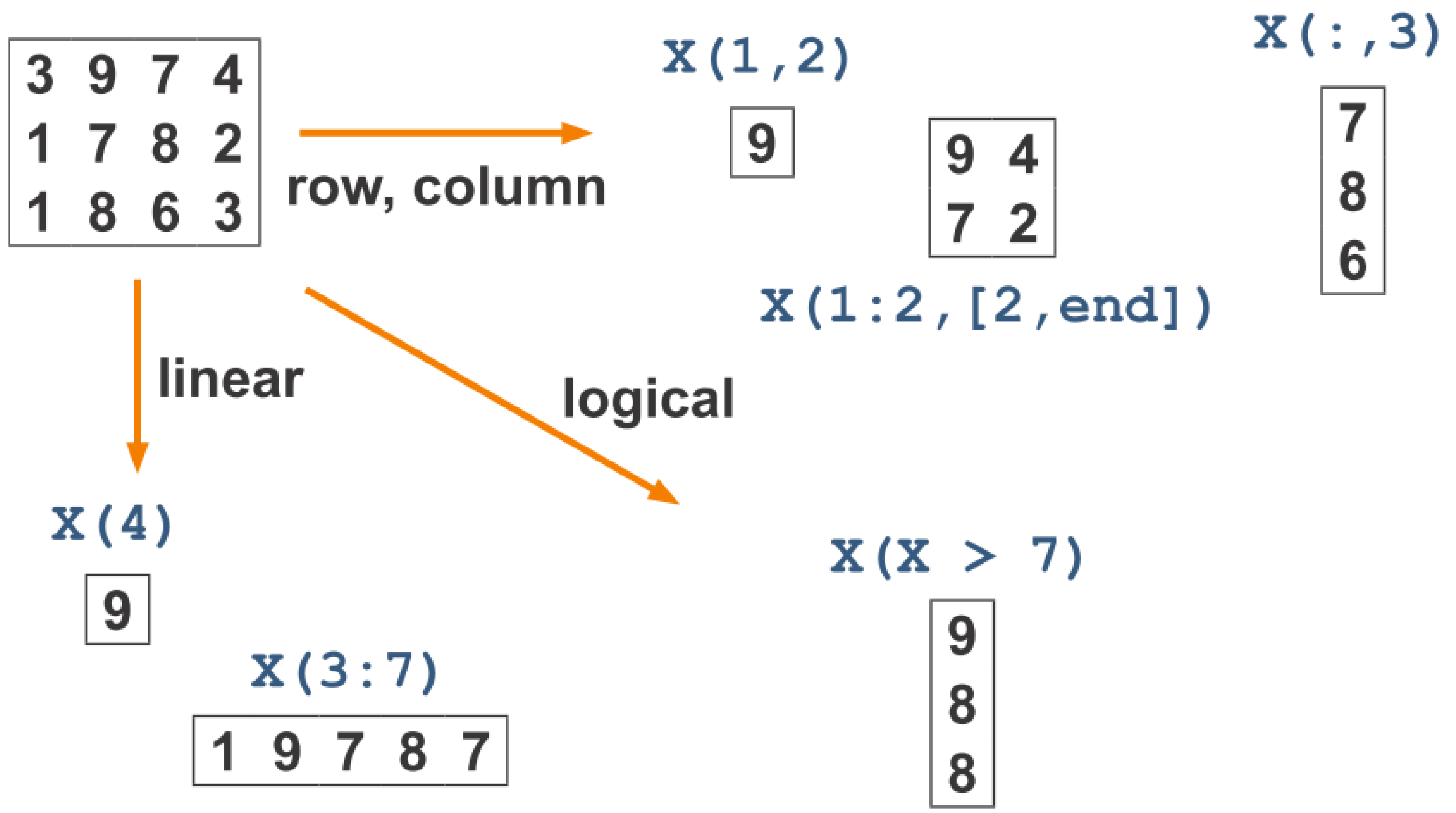
Mixed data types

2 year	3 stint	4 tmID	5 lgID	6 GP	7 GS	8 minutes	9 points	10 oRebounds	11 dRebounds	re
1997	1	'DEN'	'NBA'	39	0	34	144	27	73	
1997	[illegible]	[illegible]	[illegible]	[illegible]	0	8	26	8	19	
1997	[illegible]	[illegible]	[illegible]	[illegible]	0	7	9	2	2	
1997	1	'MIA'	'NBA'	81	0	262	1020	72	220	
1997	1	'PO	[illegible]	[illegible]	0	4	6	2	9	
1997	1	'ATL	[illegible]	[illegible]	0	8	10	1	5	
1997	1	'PH	[illegible]	[illegible]	0	4	24	4	14	
1997	1	'SAS'	'NBA'	9	0	10	19	3	9	
1997	1	'DET'	'NBA'	40	0	73	141	57	93	
1997	1	'CHI'	'NBA'	58	0	170	663	113	228	
1997	1	'VAN'	'NBA'	82	0	149	616	147	215	
1997	1	'VAN'	'NBA'	57	0	141	616	30	103	
1997	1	'NJN'	'NBA'	9	0	4	3	3	2	
1997	1	'GSW'	'NBA'	19	0	24	37	2	13	
1997	1	'DET'	'NBA'	59	0	70	141	65	130	
1997	1	'MIA'	'NBA'	72	0	192	519	48	220	
1997	1	'UTA'	'NBA'	81	0	303	2190	189	645	
1997	1	'HOU'	'NBA'	78	0	221	669	16	126	
1997	1	'PHO'	'NBA'	70	0	179	947	110	282	

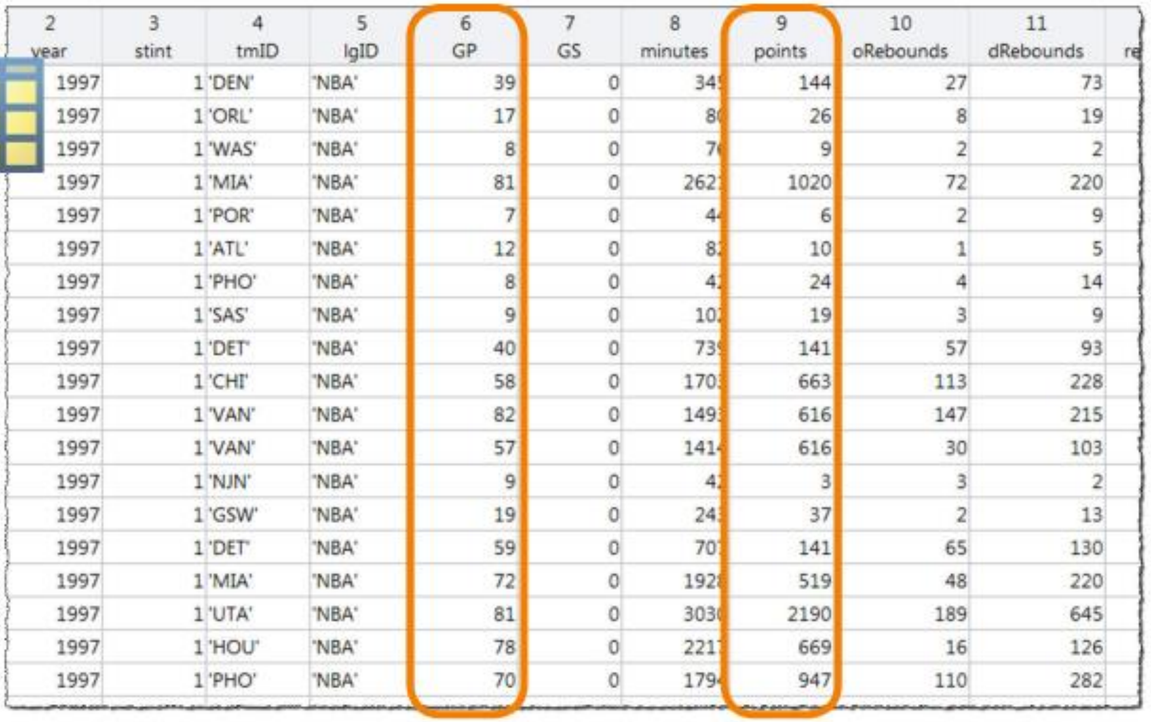
Each row is a set of observations.

Each column is a named variable.

Referencing Elements of data array



Indexing with Tables

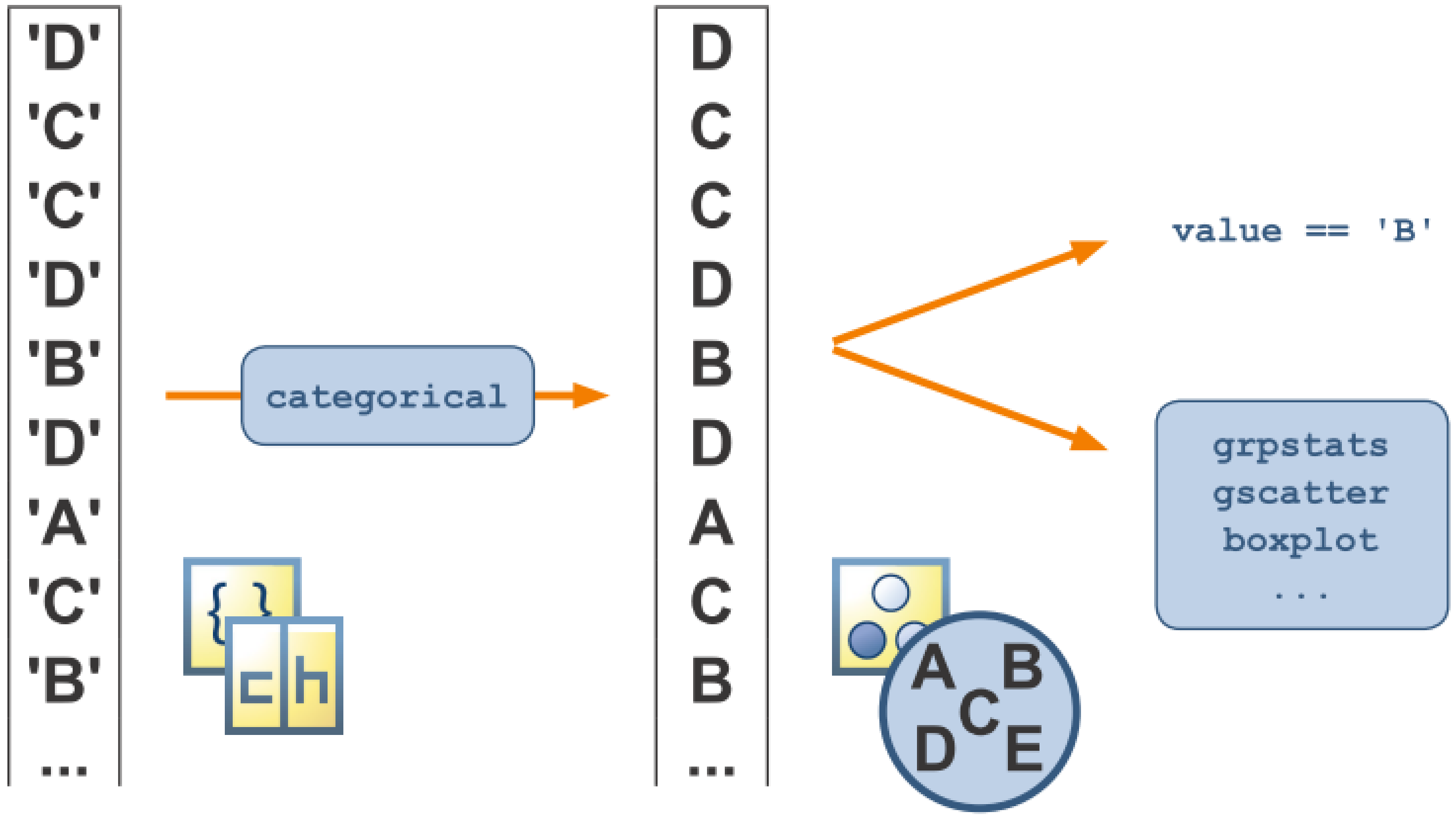


2 year	3 stint	4 tmID	5 lgID	6 GP	7 GS	8 minutes	9 points	10 oRebounds	11 dRebounds	re
1997	1	'DEN'	'NBA'	39	0	34	144	27	73	
1997	1	'ORL'	'NBA'	17	0	8	26	8	19	
1997	1	'WAS'	'NBA'	8	0	7	9	2	2	
1997	1	'MIA'	'NBA'	81	0	262	1020	72	220	
1997	1	'POR'	'NBA'	7	0	4	6	2	9	
1997	1	'ATL'	'NBA'	12	0	8	10	1	5	
1997	1	'PHO'	'NBA'	8	0	4	24	4	14	
1997	1	'SAS'	'NBA'	9	0	10	19	3	9	
1997	1	'DET'	'NBA'	40	0	73	141	57	93	
1997	1	'CHI'	'NBA'	58	0	170	663	113	228	
1997	1	'VAN'	'NBA'	82	0	149	616	147	215	
1997	1	'VAN'	'NBA'	57	0	141	616	30	103	
1997	1	'NJN'	'NBA'	9	0	4	3	3	2	
1997	1	'GSW'	'NBA'	19	0	24	37	2	13	
1997	1	'DET'	'NBA'	59	0	70	141	65	130	
1997	1	'MIA'	'NBA'	72	0	192	519	48	220	
1997	1	'UTA'	'NBA'	81	0	303	2190	189	645	
1997	1	'HOU'	'NBA'	78	0	221	669	16	126	
1997	1	'PHO'	'NBA'	70	0	179	947	110	282	

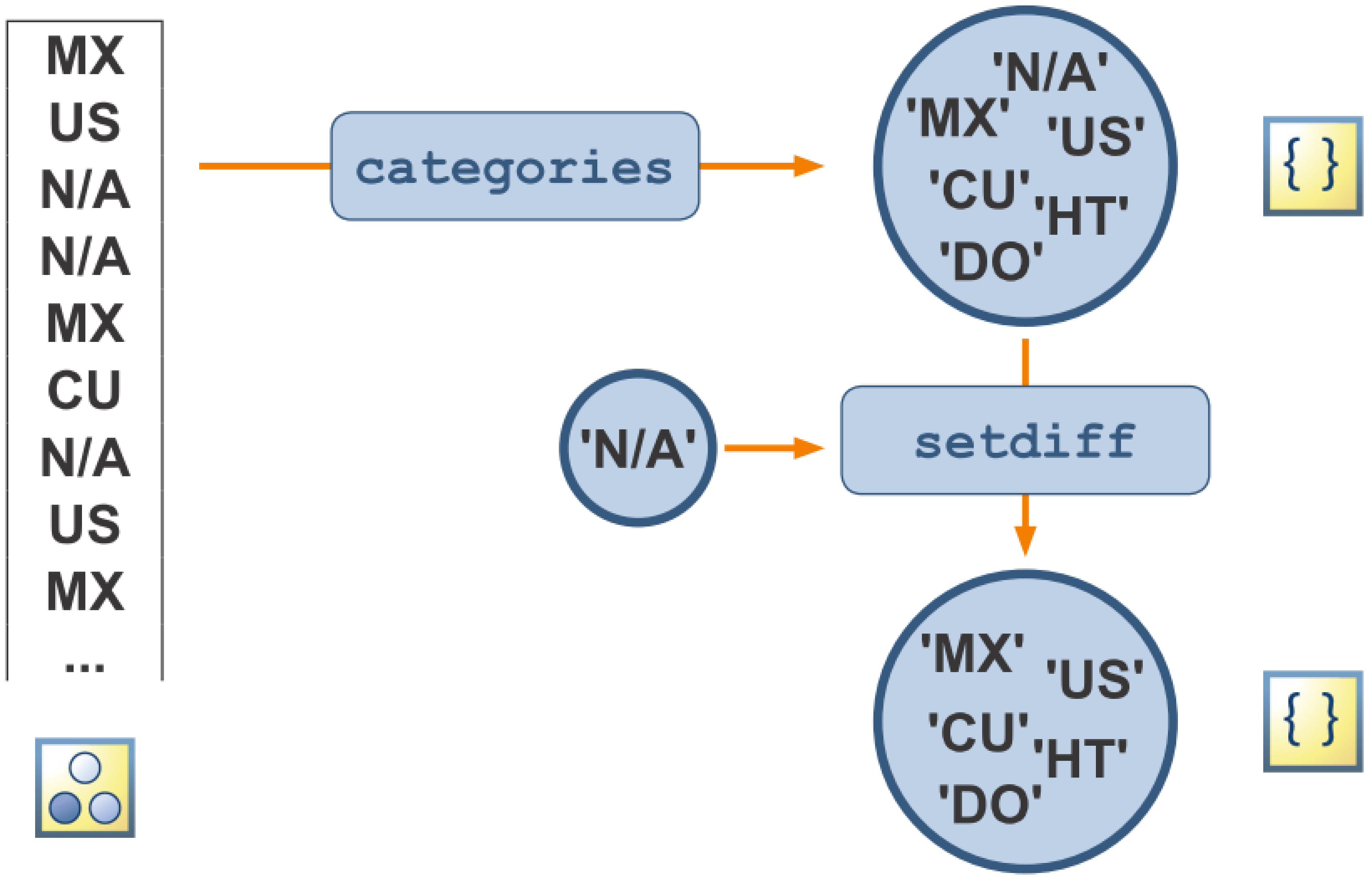
```
data(:,{'GP','points'})
```

```
data{:,{'GP','points'}}
```

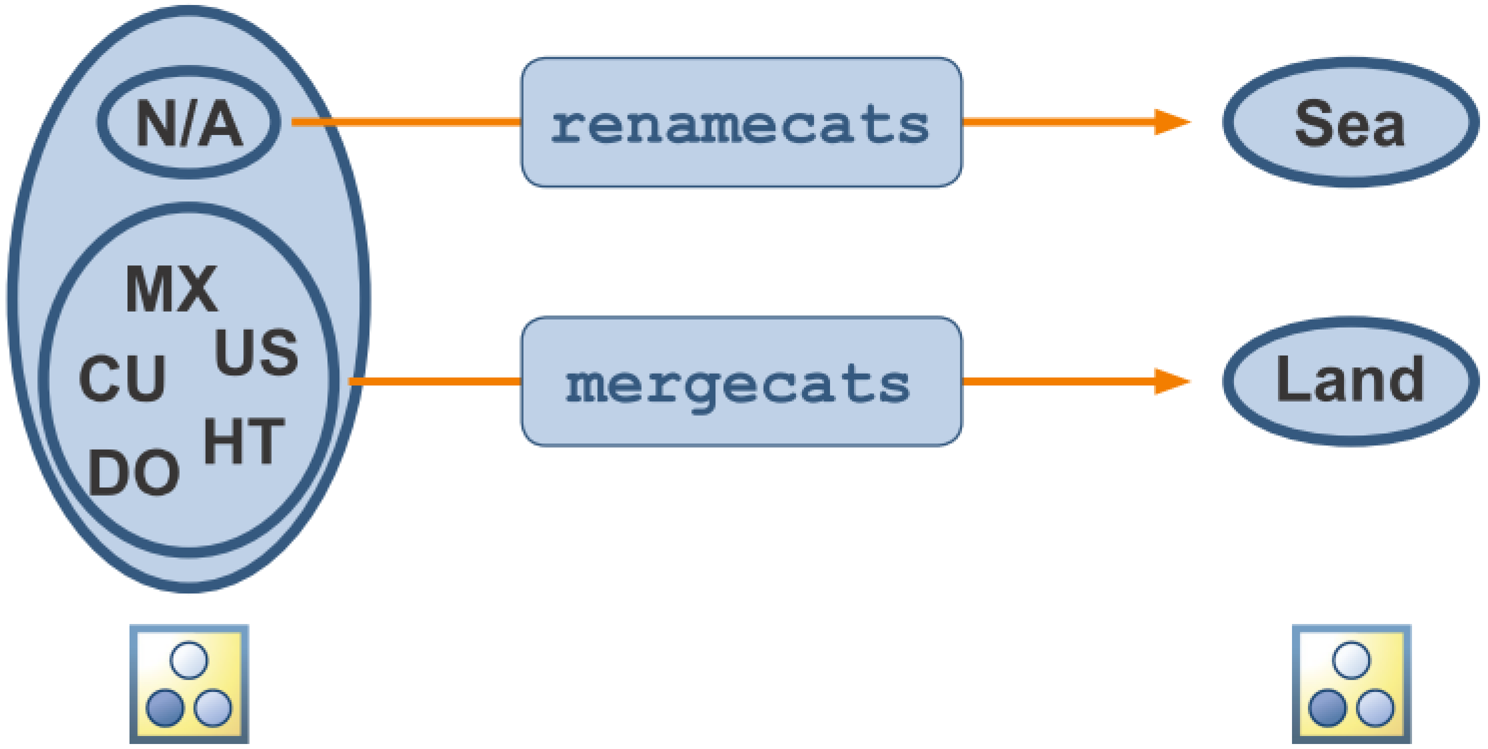
Categorical data



Categories and Set Operations



Modifying Categories



Grouped Operations

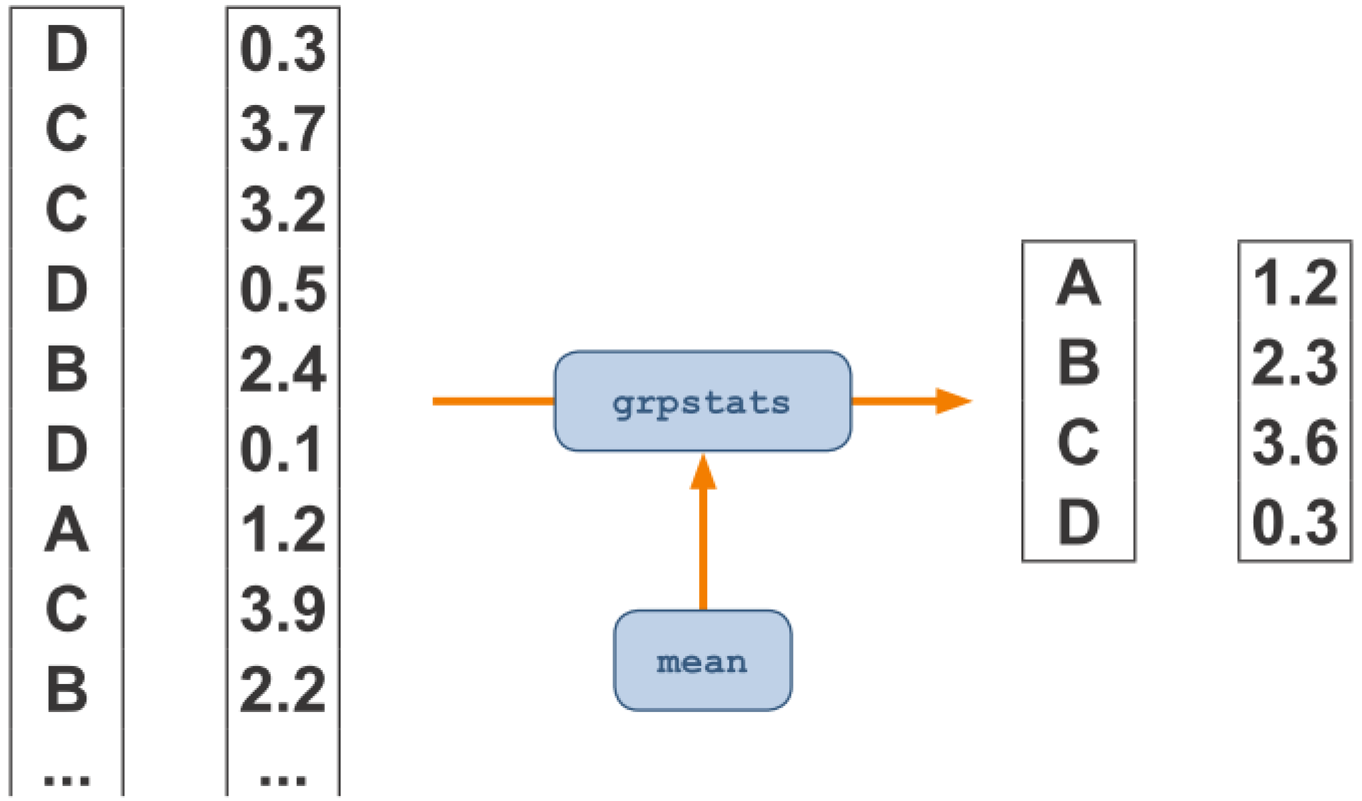
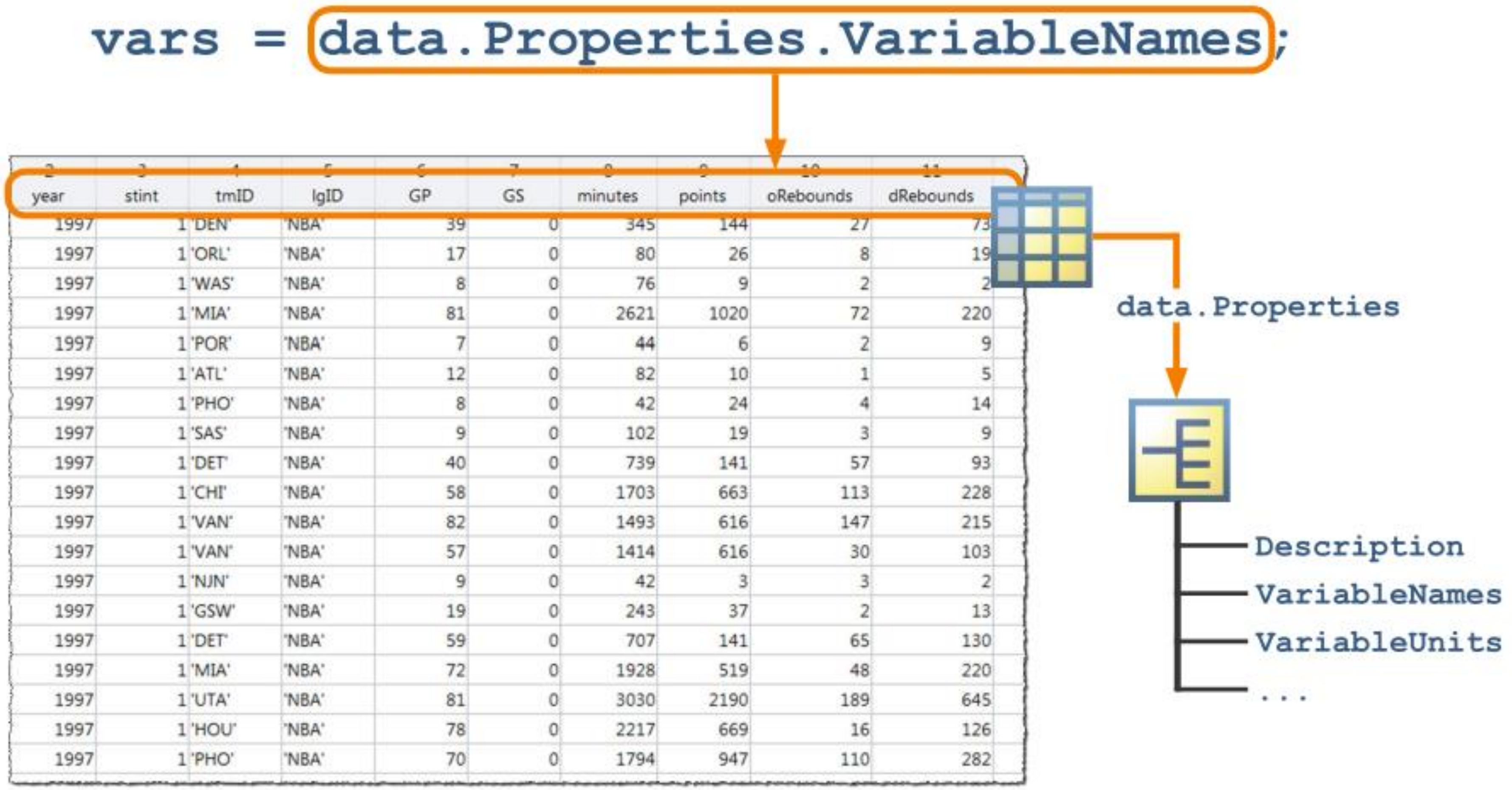


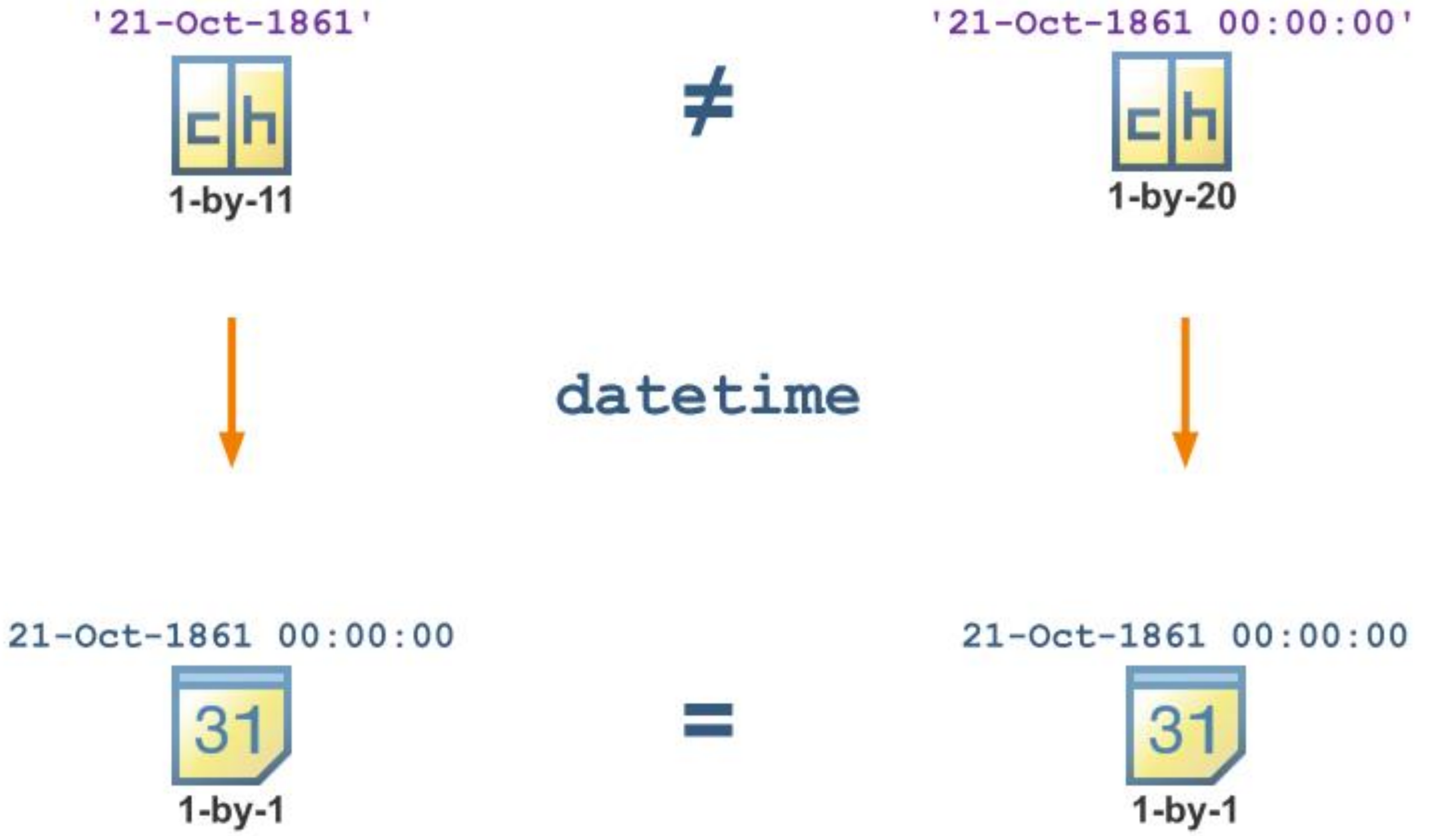
Table Properties



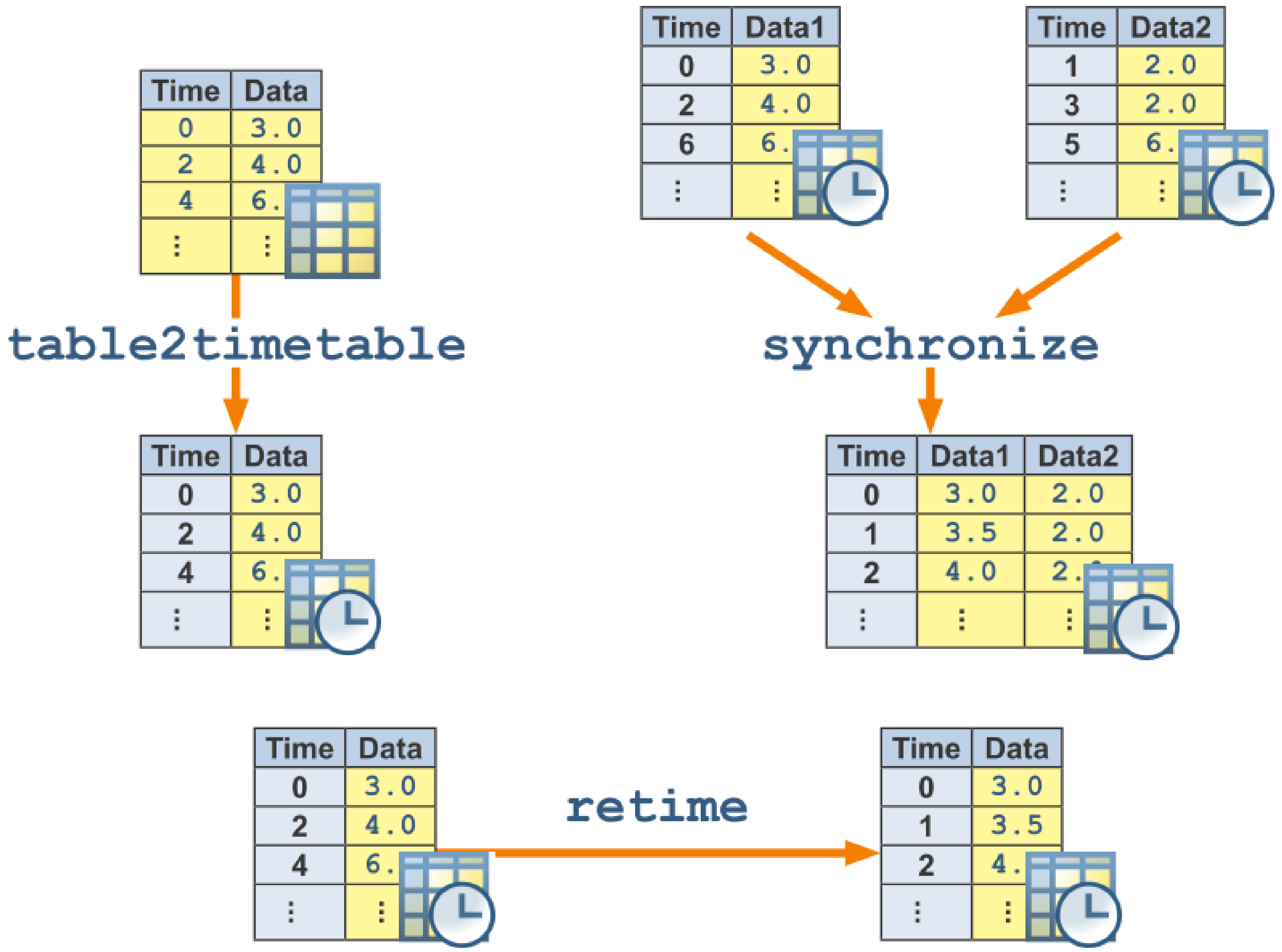
```
vars = data.Properties.VariableNames;
```

2	3	4	5	6	7	8	9	10	11
year	stint	tmID	lgID	GP	GS	minutes	points	oRebounds	dRebounds
1997	1	'DEN'	'NBA'	39	0	345	144	27	73
1997	1	'ORL'	'NBA'	17	0	80	26	8	19
1997	1	'WAS'	'NBA'	8	0	76	9	2	2
1997	1	'MIA'	'NBA'	81	0	2621	1020	72	220
1997	1	'POR'	'NBA'	7	0	44	6	2	9
1997	1	'ATL'	'NBA'	12	0	82	10	1	5
1997	1	'PHO'	'NBA'	8	0	42	24	4	14
1997	1	'SAS'	'NBA'	9	0	102	19	3	9
1997	1	'DET'	'NBA'	40	0	739	141	57	93
1997	1	'CHI'	'NBA'	58	0	1703	663	113	228
1997	1	'VAN'	'NBA'	82	0	1493	616	147	215
1997	1	'VAN'	'NBA'	57	0	1414	616	30	103
1997	1	'NJN'	'NBA'	9	0	42	3	3	2
1997	1	'GSW'	'NBA'	19	0	243	37	2	13
1997	1	'DET'	'NBA'	59	0	707	141	65	130
1997	1	'MIA'	'NBA'	72	0	1928	519	48	220
1997	1	'UTA'	'NBA'	81	0	3030	2190	189	645
1997	1	'HOU'	'NBA'	78	0	2217	669	16	126
1997	1	'PHO'	'NBA'	70	0	1794	947	110	282

Representing Dates and Times



Timetables



Time	Data
0	3.0
2	4.0
4	6.
⋮	⋮

Time	Data
0	3.0
2	4.0
4	6.
⋮	⋮

Time	Data1
0	3.0
2	4.0
6	6.
⋮	⋮

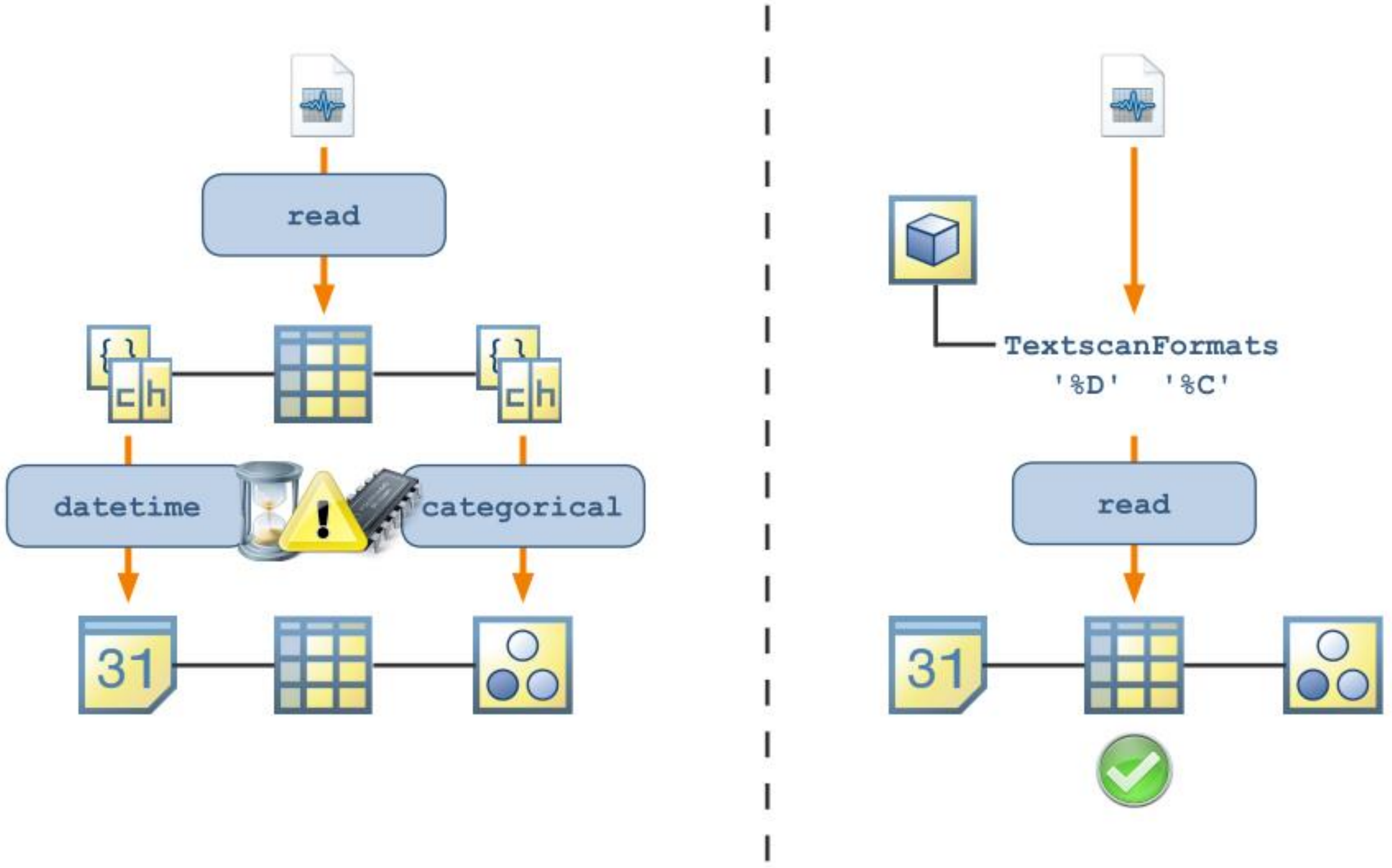
Time	Data2
1	2.0
3	2.0
5	6.
⋮	⋮

Time	Data1	Data2
0	3.0	2.0
1	3.5	2.0
2	4.0	2.
⋮	⋮	⋮

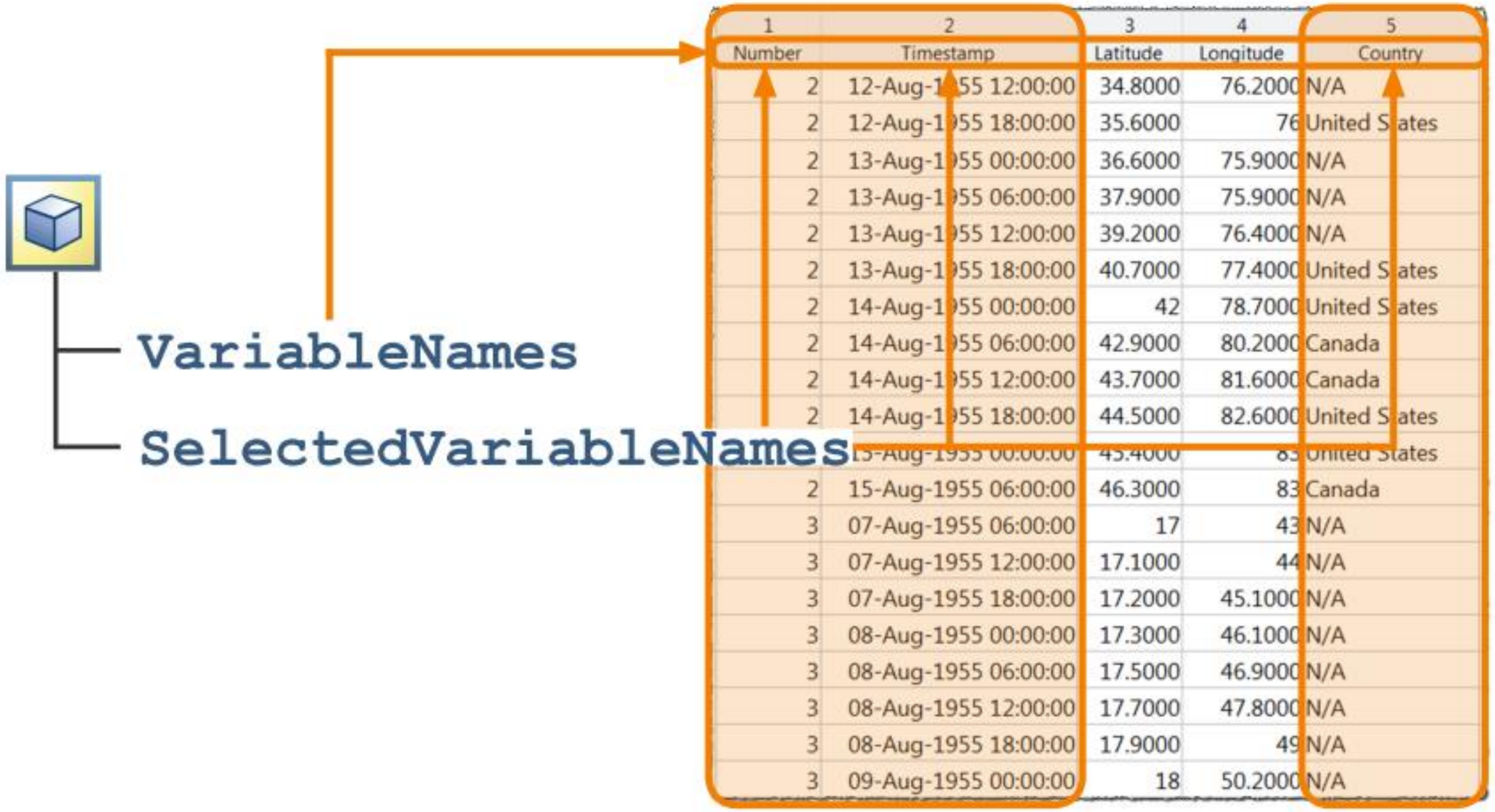
Time	Data
0	3.0
2	4.0
4	6.
⋮	⋮

Time	Data
0	3.0
1	3.5
2	4.
⋮	⋮

Importing Data Types Directly

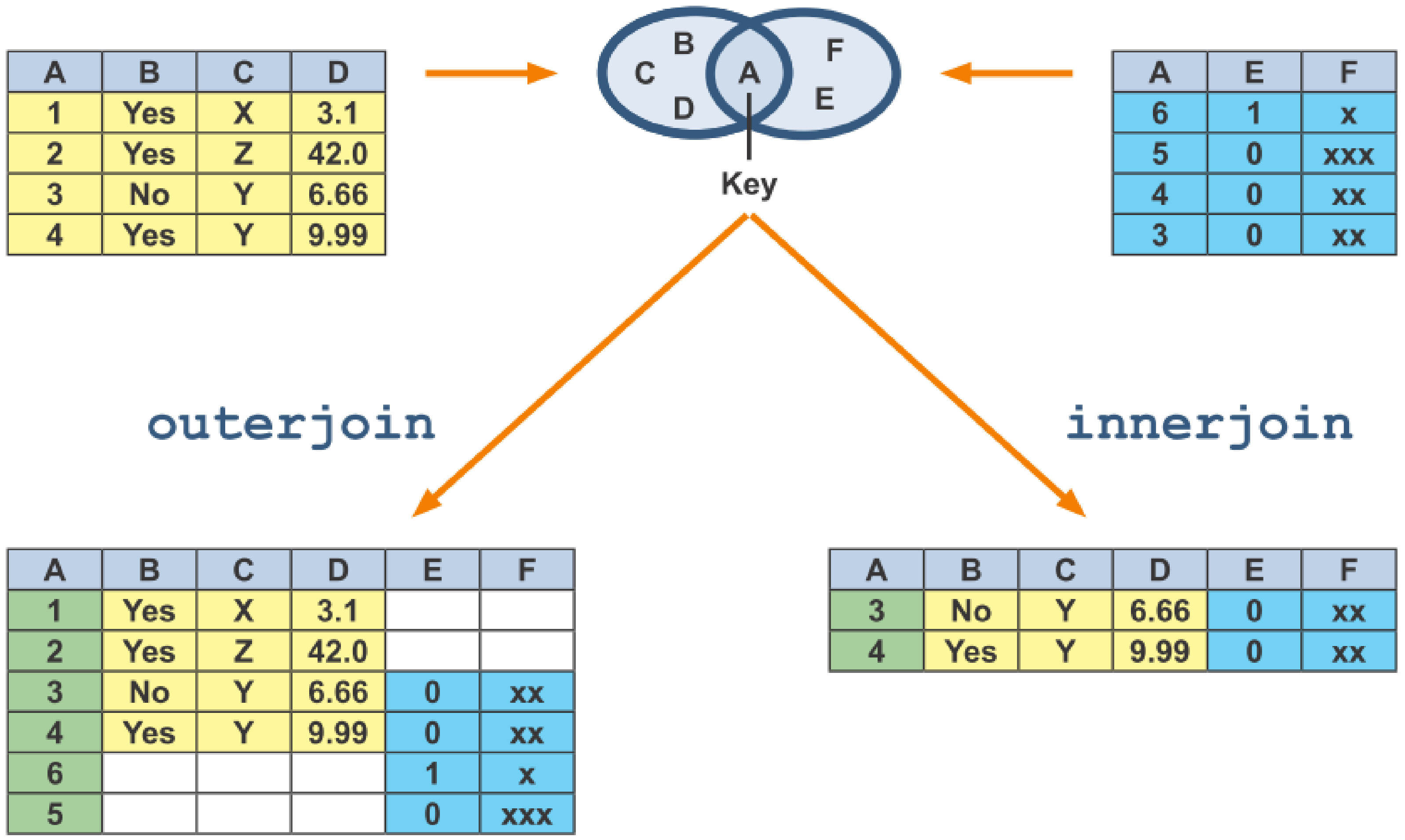


Skipping Columns of Data



1 Number	2 Timestamp	3 Latitude	4 Longitude	5 Country
2	12-Aug-1955 12:00:00	34.8000	76.2000	N/A
2	12-Aug-1955 18:00:00	35.6000	76	United States
2	13-Aug-1955 00:00:00	36.6000	75.9000	N/A
2	13-Aug-1955 06:00:00	37.9000	75.9000	N/A
2	13-Aug-1955 12:00:00	39.2000	76.4000	N/A
2	13-Aug-1955 18:00:00	40.7000	77.4000	United States
2	14-Aug-1955 00:00:00	42	78.7000	United States
2	14-Aug-1955 06:00:00	42.9000	80.2000	Canada
2	14-Aug-1955 12:00:00	43.7000	81.6000	Canada
2	14-Aug-1955 18:00:00	44.5000	82.6000	United States
	15-Aug-1955 00:00:00	45.4000	83	United States
2	15-Aug-1955 06:00:00	46.3000	83	Canada
3	07-Aug-1955 06:00:00	17	43	N/A
3	07-Aug-1955 12:00:00	17.1000	44	N/A
3	07-Aug-1955 18:00:00	17.2000	45.1000	N/A
3	08-Aug-1955 00:00:00	17.3000	46.1000	N/A
3	08-Aug-1955 06:00:00	17.5000	46.9000	N/A
3	08-Aug-1955 12:00:00	17.7000	47.8000	N/A
3	08-Aug-1955 18:00:00	17.9000	49	N/A
3	09-Aug-1955 00:00:00	18	50.2000	N/A

Merging data



A	B	C	D
1	Yes	X	3.1
2	Yes	Z	42.0
3	No	Y	6.66
4	Yes	Y	9.99

A	E	F
6	1	x
5	0	xxx
4	0	xx
3	0	xx

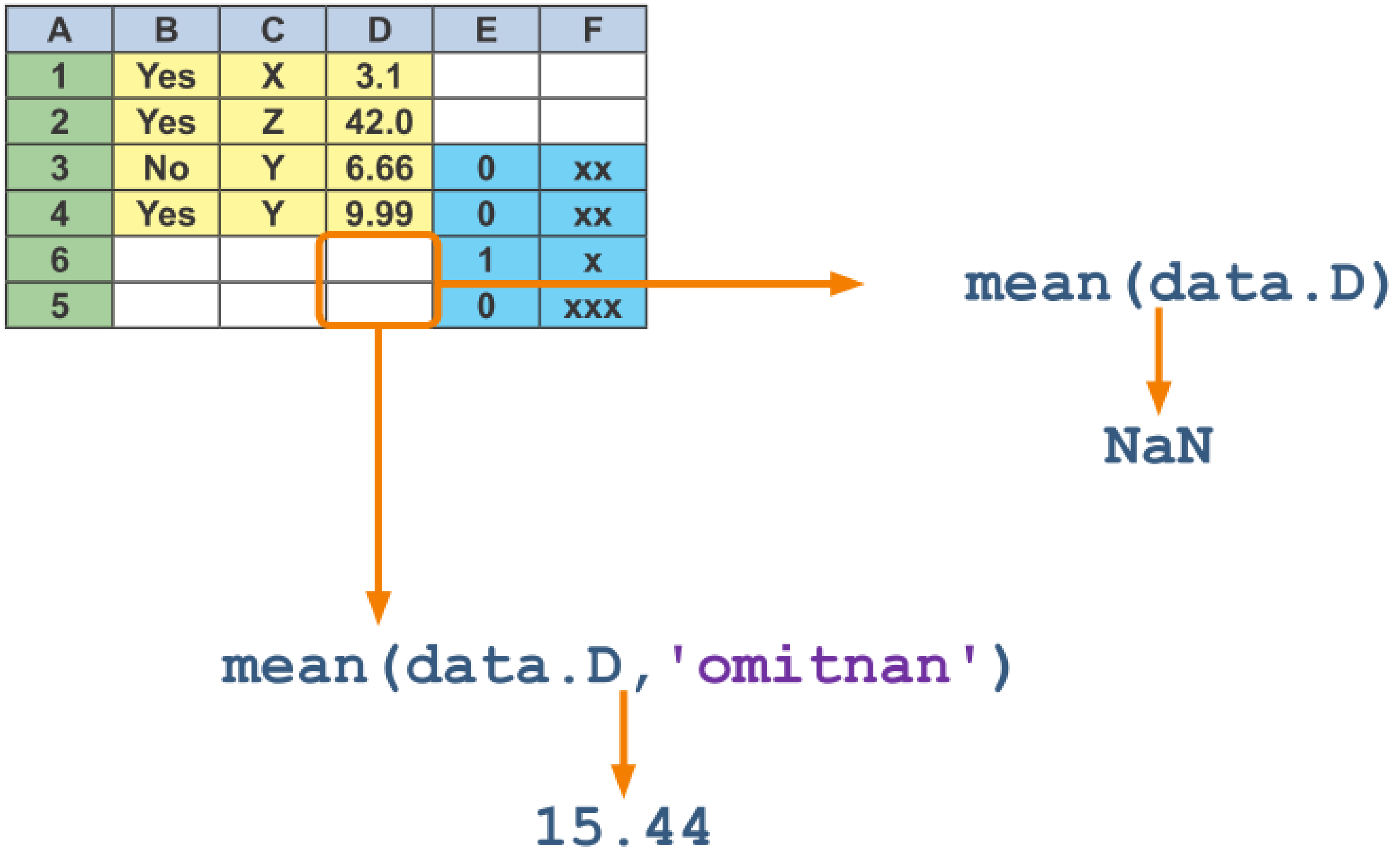
outerjoin

A	B	C	D	E	F
1	Yes	X	3.1		
2	Yes	Z	42.0		
3	No	Y	6.66	0	xx
4	Yes	Y	9.99	0	xx
6				1	x
5				0	xxx

innerjoin

A	B	C	D	E	F
3	No	Y	6.66	0	xx
4	Yes	Y	9.99	0	xx

Working with missing data

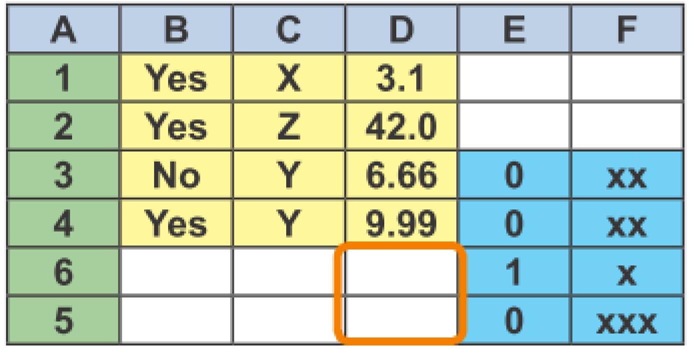


A	B	C	D	E	F
1	Yes	X	3.1		
2	Yes	Z	42.0		
3	No	Y	6.66	0	xx
4	Yes	Y	9.99	0	xx
6				1	x
5				0	xxx

```
mean(data.D)
NaN
```

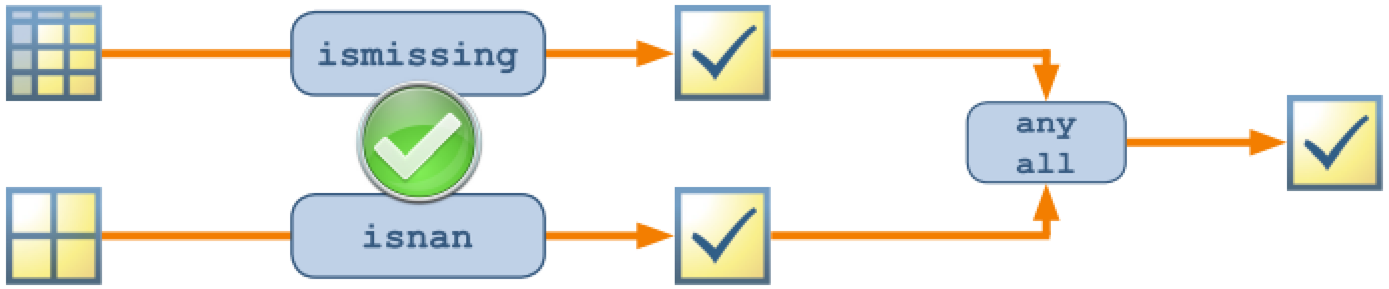
```
mean(data.D,'omitnan')
15.44
```

Locating Missing Data

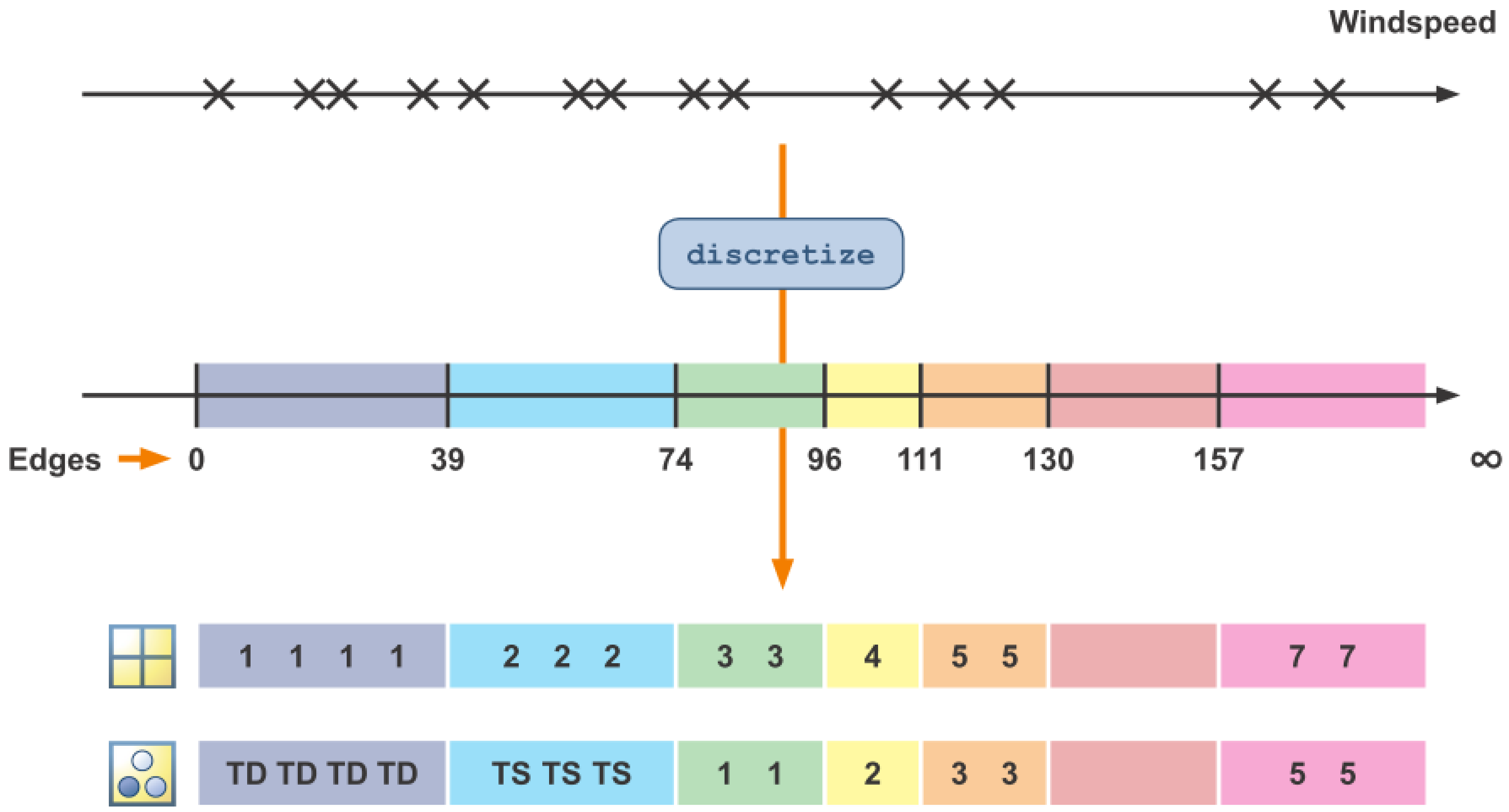


A	B	C	D	E	F
1	Yes	X	3.1		
2	Yes	Z	42.0		
3	No	Y	6.66	0	xx
4	Yes	Y	9.99	0	xx
6				1	x
5				0	xxx

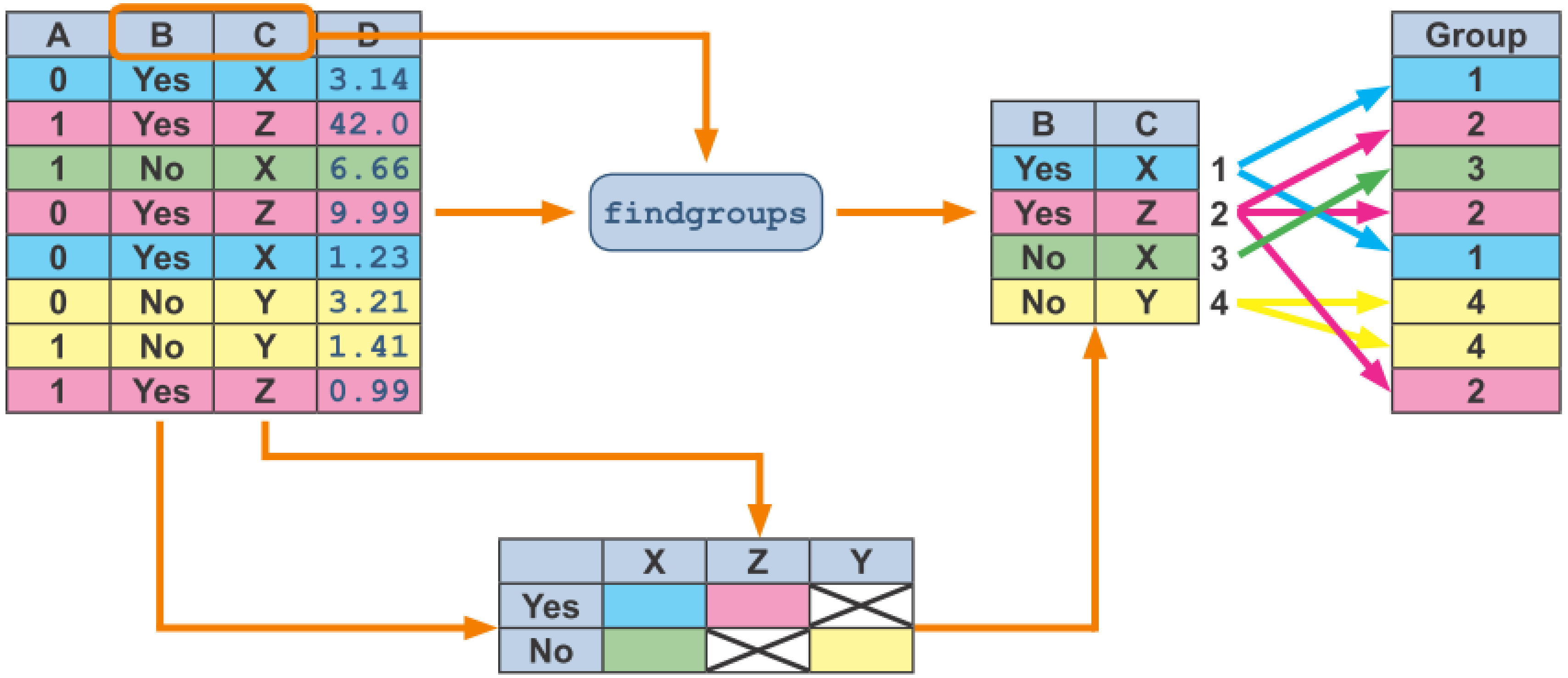
`data == NaN` ✗



Discretizing Continuous Data



Finding Unique Groups of Data



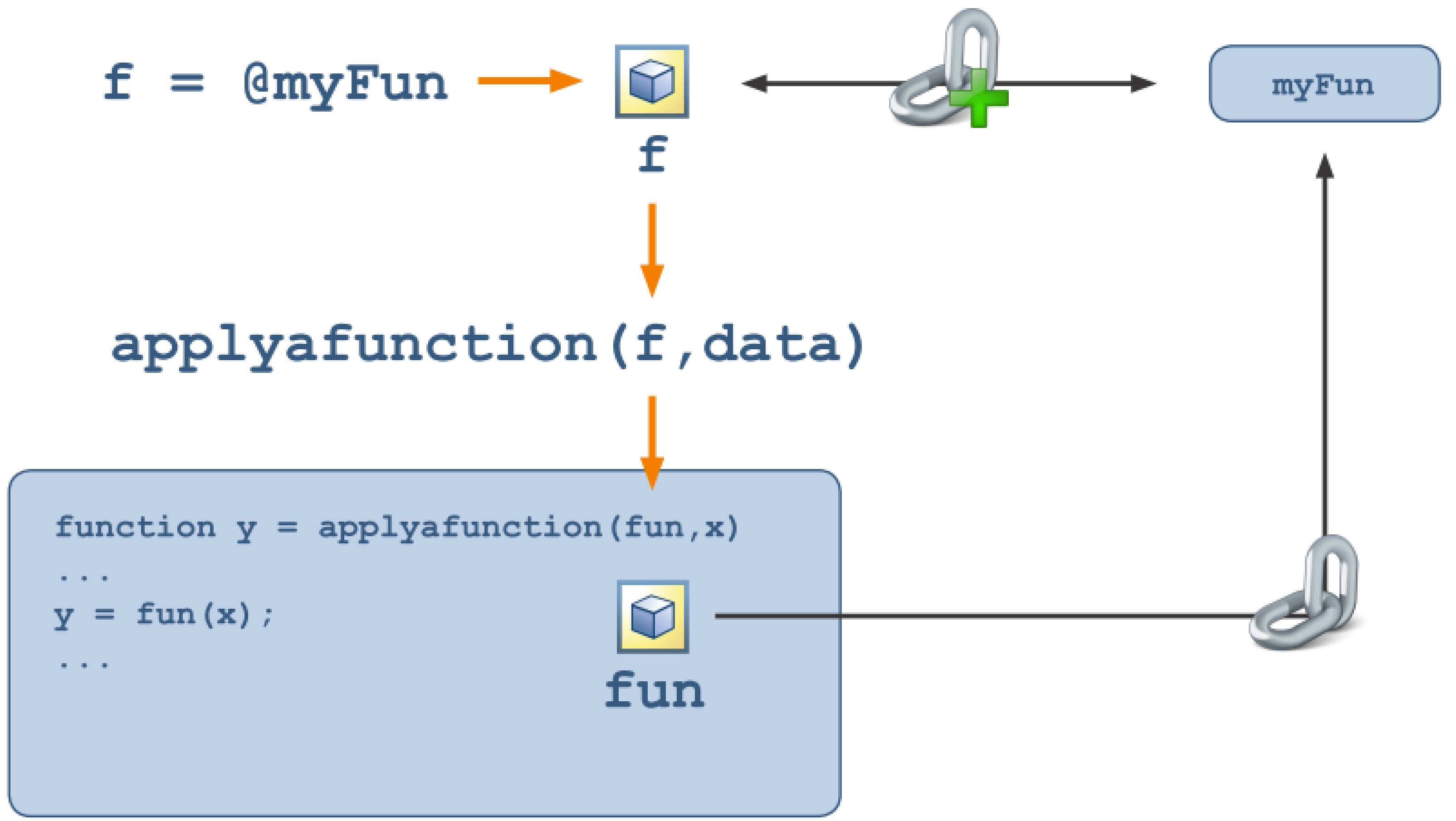
A	B	C	D
0	Yes	X	3.14
1	Yes	Z	42.0
1	No	X	6.66
0	Yes	Z	9.99
0	Yes	X	1.23
0	No	Y	3.21
1	No	Y	1.41
1	Yes	Z	0.99

	X	Z	Y
Yes	🟦	🟪	☒
No	🟩	☒	🟨

B	C	
Yes	X	1
Yes	Z	2
No	X	3
No	Y	4

Group
1
2
3
2
1
4
4
2

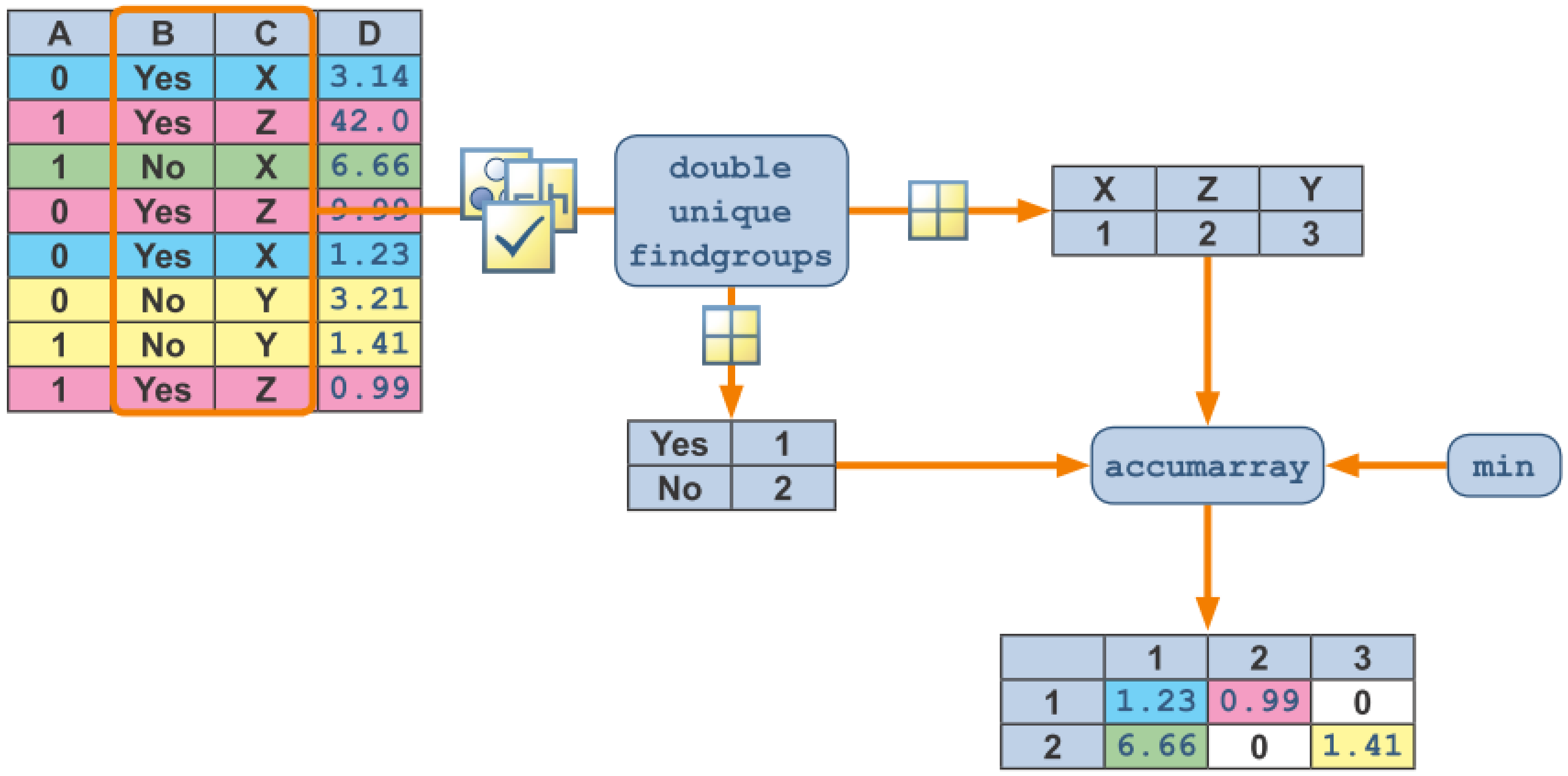
Function Handles



```
f = @myFun
```

```
function y = applyafunction(fun,x)
...
y = fun(x);
...
```

Aggregating Data Into a Prescribed Format



A	B	C	D
0	Yes	X	3.14
1	Yes	Z	42.0
1	No	X	6.66
0	Yes	Z	0.99
0	Yes	X	1.23
0	No	Y	3.21
1	No	Y	1.41
1	Yes	Z	0.99

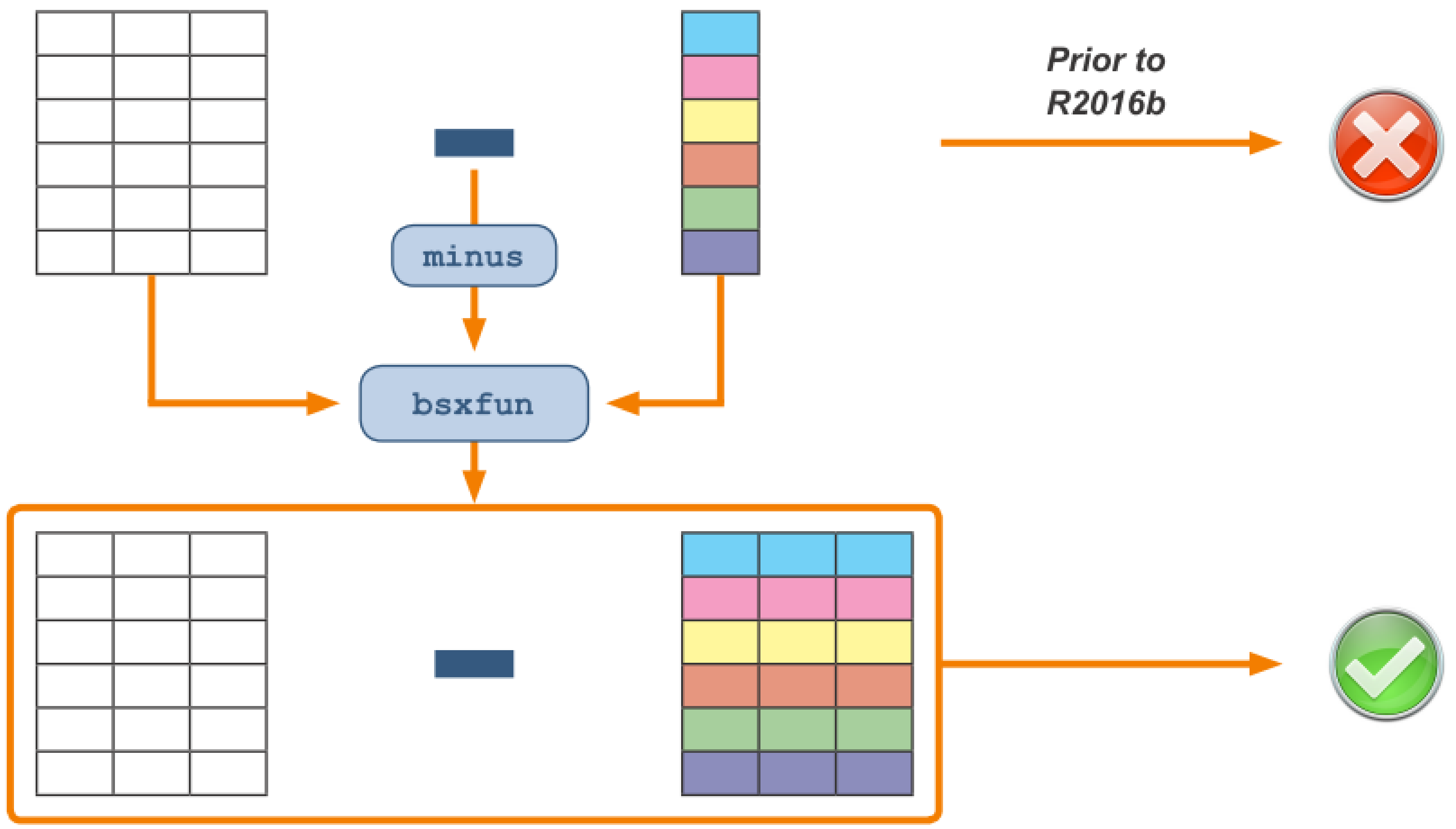
X	Z	Y
1	2	3

Yes	1
No	2

	1	2	3
1	1.23	0.99	0
2	6.66	0	1.41

Performing Array Operations on Unequal Dimensions:

bsxfun


minus
bsxfun
Prior to R2016b