TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

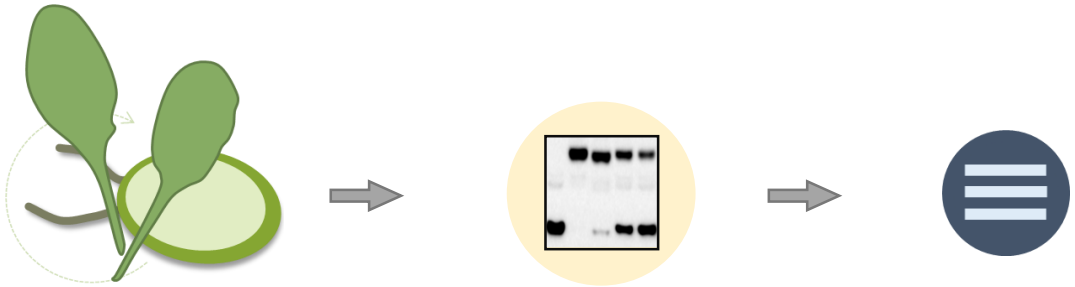
TRR 175

Introduction to data exploration using Deedle

CSB COMPUTATIONAL SYSTEMS BIOLOGY

David Zimmer
Computational Systems Biology
Kaiserslautern University of Technology

Big Data in Biology

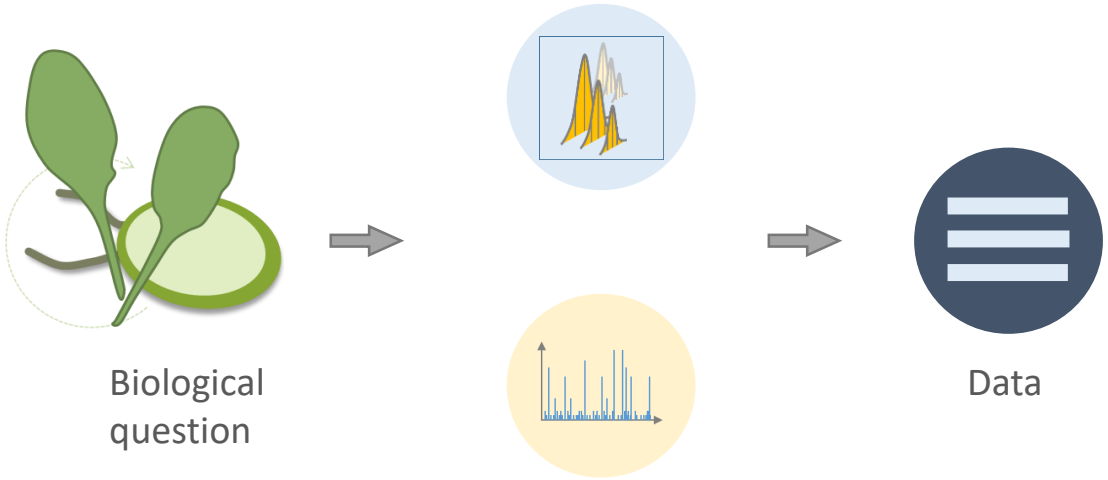


Biological question

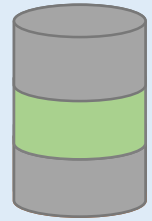
molecular biological technologies

Data

Big Data in Biology



Data can be stored in various formats

DSVs: delimiter separated values	Markup language	Data bases
Peptides.csv	Peptides.xml	 Peptides.SQLite
Peptides.tab		
Peptides.tab	model.xml	Proteins.SQL

Delimiter separated formats

Columns

Rows

Records	Name	Wt1	Wt2	Mut1	Mut2	Sig
Record	RBCS	10.1	11.1	5.5	6.1	true
Record	RBCL	2.9	3.0	RBCL	3.6	false
Record	D1	5.2	5.1	D1	5.5	false
…	…	…	…	…	…	…
…	…	…	…	…	…	…
…	…	…	…	…	…	…
…	…	…	…	…	…	…

Delimiter separated formats store:

- Series of individual Records (e.g. Genes), each stored in a individual row.
- Each Record can have multiple fields, the field name is commonly given by the first line of the file (the header)
- Each Column stores the value of one individual field with a given type.

Delimiter separated formats

Columns

Rows

Records	Name	Wt1	Wt2	Mut1	Mut2	Sig
Record	RBCS	10.1	11.1	5.5	6.1	true
Record	RBCL	2.9	3.0	RBCL	3.6	false
Record	D1	5.2	5.1	D1	5.5	false
…	…	…	…	…	…	…
…	…	…	…	…	…	…
…	…	…	…	…	…	…
…	…	…	…	…	…	…

Advantage of Delimiter separated formats:

- Each Column stores the value of one individual field with a given type

Disadvantage of Delimiter separated formats:

- **Each Column stores the value of one individual field with a given type.**

- One Record can contain fields of multiple types

The solution: abstract rows as types

Columns

Rows

Records	Name	Wt1	Wt2	Mut1	Mut2	Sig
Record	RBCS	10.1	11.1	5.5	6.1	true
Record	RBCL	2.9	3.0	RBCL	3.6	false
Record	D1	5.2	5.1	D1	5.5	false
…	…	…	…	…	…	…
…	…	…	…	…	…	…
…	…	…	…	…	…	…
…	…	…	…	…	…	…

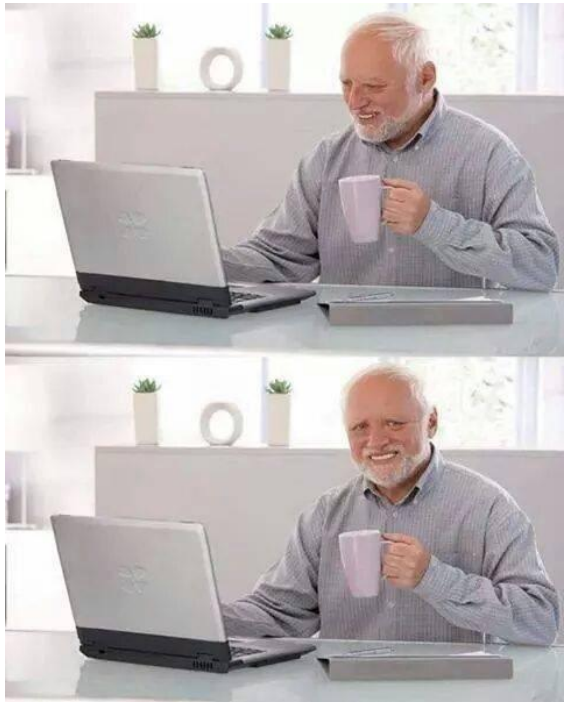
```
type QuantifiedProtein = {
    Name            : string
    WT1             : float
    WT2             : float
    Mut1            : float
    Mut2            : float
    isSignificant   : bool
    }
```

- Obvious downside: Data has to be inspected beforehand and the abstraction to a data structure has to be done manually

The problem: underestimating demands

Presenting Results

Freedom



```
type QuantifiedProtein = {
    Name                        : string
    WT1                         : float
    WT2                         : float
    Mut1                        : float
    Mut2                        : float
    isSignificant               : bool
    }
```

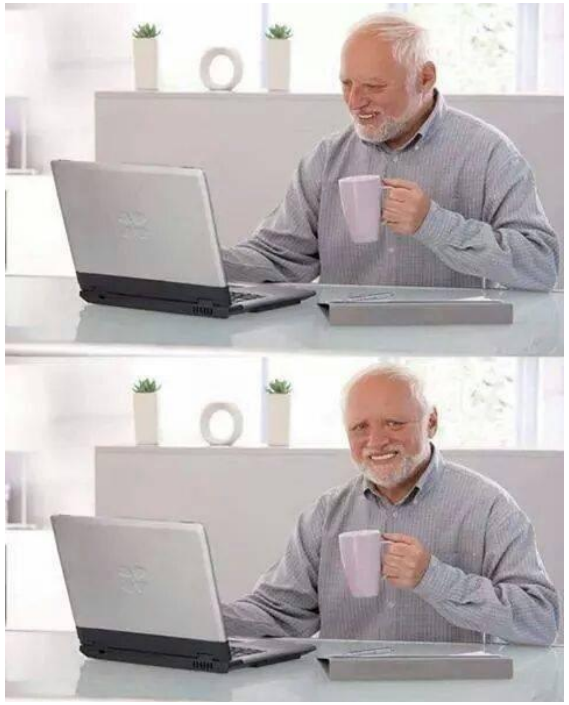
Incorporating Input / changing code

Computational
Systems Biology

The problem: underestimating demands

Presenting Results

Freedom



```
type QuantifiedProtein = {
    Name                        : string
    WT1                         : float
    WT2                         : float
    Mut1                        : float
    Mut2                        : float
    isSignificant               : bool
    Wt1VsMut1                   : float
    Wt1VsMut2                   : float
    }
```

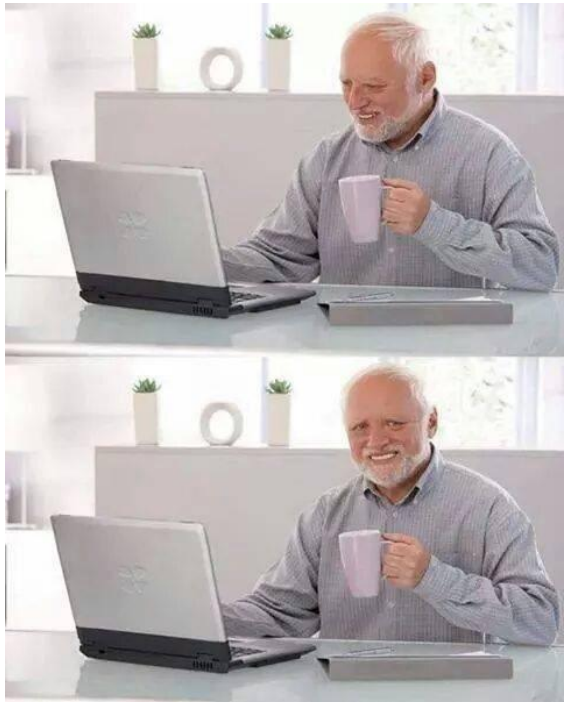
Incorporating Input / changing code

Computational
Systems Biology

The problem: underestimating demands

Presenting Results

Freedom



```
type QuantifiedProtein = {
    Name                        : string
    WT1                         : float
    WT2                         : float
    Mut1                        : float
    Mut2                        : float
    isSignificant               : bool
    Wt1VsMut1                   : float
    Wt1VsMut2                   : float
    Wt2VsMut1                   : float
    Wt2VsMut2                   : float
    }
```

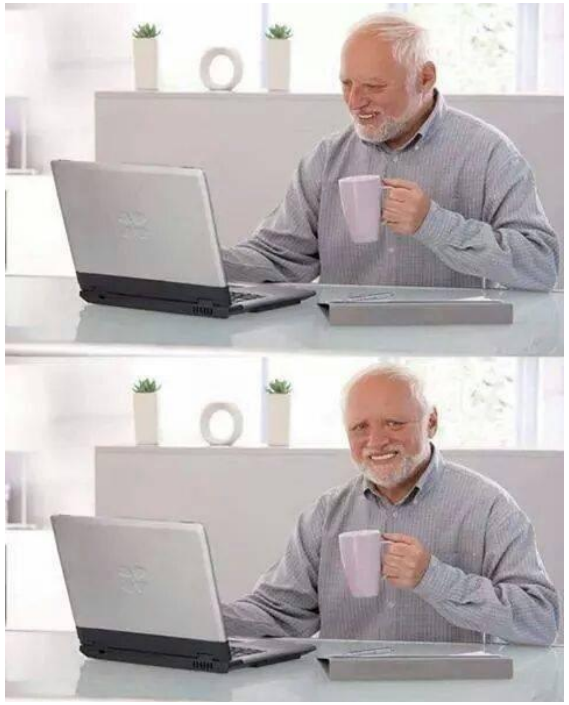
Incorporating Input / changing code

Computational Systems Biology

The problem: underestimating demands

Presenting Results

Freedom



```
type QuantifiedProtein = {
    Name                        : string
    WT1                         : float
    WT2                         : float
    Mut1                        : float
    Mut2                        : float
    isSignificant               : bool
    Wt1VsMut1                   : float
    Wt1VsMut2                   : float
    Wt2VsMut1                   : float
    Wt2VsMut2                   : float
    WtMeans                     : float
    MutMeans                    : float
    WtStdev                     : float
    MutStdev                    : float
    PearsonCorrWt1VsMut1        : float
    PearsonCorrWt1VsMut2        : float
    PearsonCorrWt2VsMut1        : float
```

Incorporating Input / changing code

Computational Systems Biology

The solution: Data Frame programming

Rows

Records
1
2
3
…
…
…
…

Name	Wt1	Wt2	Mut1	Mut2	Sig
RBCS	10.1	11.1	5.5	6.1	true
RBCL	2.9	3.0	RBCL	3.6	false
D1	5.2	5.1	D1	5.5	false
…	…	…	…	…	…
…	…	…	…	…	…
…	…	…	…	…	…
…	…	…	…	…	…

```
let rawData :Frame<int,string> =
    Frame.ReadCsv(rawDataPath,separators="\t",)
```

- Data Frames represent type structures that abstract tabular organized data in memory
- Data Frame objects can be seemlesly extended, reorderd, aggegated and also merged

The solution: Data Frame programming

Data Frame

Series 1 Series 2 Series 2 Series 3 Series 4 Series 4

Rows

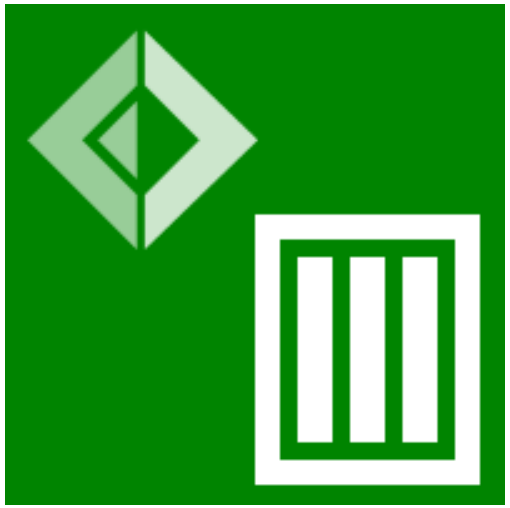
Records	Name	Wt1	Wt2	Mut1	Mut2	Sig
1	RBCS	10.1	11.1	5.5	6.1	true
2	RBCL	2.9	3.0	RBCL	3.6	false
3	D1	5.2	5.1	D1	5.5	false
…	…	…	…	…	…	…
…	…	…	…	…	…	…
…	…	…	…	…	…	…
…	…	…	…	…	…	…

```
let rawData :Frame<int,string> =
    Frame.ReadCsv(rawDataPath,separators="\t",)
```

- Most Data Frames API abstract a tables as a collection of Series of same length, but different types

- „Data Frame programming is fast becoming the "standard" way of doing multi-dimensional and statistical data processing in systems such as R, Python and now .NET. “

Deedle: .Net Data Frame library



- Deedle is an easy to use library for data and time series manipulation and for scientific programming in .Net.
- Deedle relies on two basic data structures, the Series and the Frame
- A Frame can be viewed as a list of Series, with each Series being a Column of the Frame. If possible: use the data frame in a column-wise way

Working with Deedle Data Series

```
let firstNames      = Series.ofValues ["Kevin";"Lukas";"Benedikt";"Michael"]
```

1	Kevin
2	Lukas
3	Bene.
4	Micha.

```
Interactive Output: val firstNames : Series<int,string> =
                    0 -> Kevin
                    1 -> Lukas
                    2 -> Benedikt
                    3 -> Michael
```

Working with Deedle Data Series

```
let firstNames      = Series.ofValues ["Kevin";"Lukas";"Benedikt";"Michael"]
let coffeesPerWeek  = Series.ofValues [15;12;10;11]
```

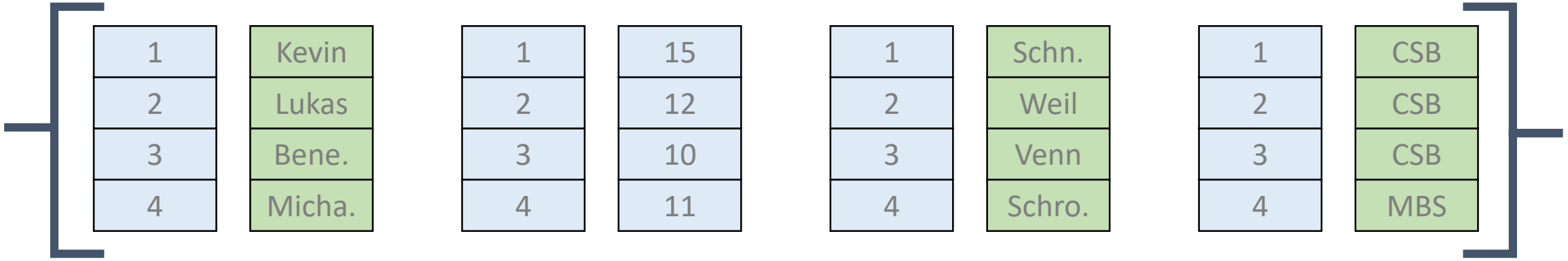
1	Kevin
2	Lukas
3	Bene.
4	Micha.

1	15
2	12
3	10
4	11

```
Interactive Output: val coffeesPerWeek : Series<int,int> =
                    0 -> 15
                    1 -> 12
                    2 -> 10
                    3 -> 11
```

Working with Deedle Data Series

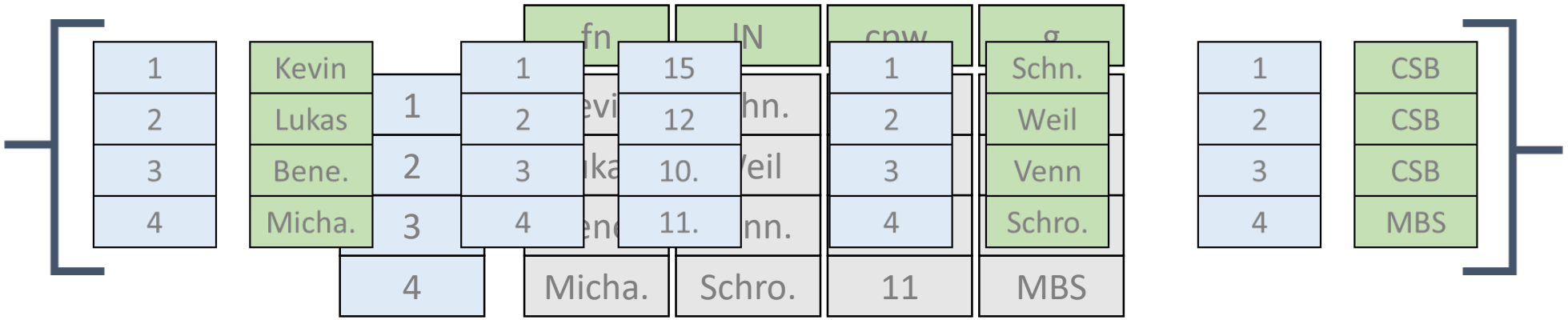
```
let firstNames      = Series.ofValues ["Kevin";"Lukas";"Benedikt";"Michael"]
let coffeesPerWeek  = Series.ofValues [15;12;10;11]
let lastNames       = Series.ofValues ["Schneider";"Weil";"Venn";"Schroda"]
let group           = Series.ofValues ["CSB";"CSB";"CSB";"MBS"]
```




```
Interactive Output: …
```

Constructing a Deedle Data Frame

```
let persons = Frame(["fN";"lN";"cpw";"g"],[firstNames;lastNames;coffeesPerWeek;group])
```



Interactive Output:

```
val persons : Frame<int,string> =
     fN       lN          cpw g
0 -> Kevin    Schneider 15  CSB
1 -> Lukas    Weil      12  CSB
2 -> Benedikt Venn      10  CSB
3 -> Michael  Schroda   11  MBS
```

Computational Systems Biology

Constructing a Deedle Data Frame

```
let persons = Frame(["fN";"lN";"cpw";"g"],[firstNames;lastNames;coffeesPerWeek;group])
```

	fn	lN	cpw.	g
1	Kevin	Schn.	15	CSB
2	Lukas	Weil	12	CSB
3	Bene.	Venn.	10	CSB
4	Micha.	Schro.	11	MBS

Interactive Output:

```
val persons : Frame<int,string> =
     fN       lN         cpw g
0 -> Kevin    Schneider 15  CSB
1 -> Lukas    Weil      12  CSB
2 -> Benedikt Venn      10  CSB
3 -> Michael  Schroda   11  MBS
```

Constructing a Deedle Data Frame

```
let persons = Frame(["fN";"lN";"cpw";"g"],[firstNames;lastNames;coffeesPerWeek;group])
let coffeePerWeek' :Series<int,string> = persons |> Frame.getCol ("cpw")
```

	fn	lN	cpw.	g
1	Kevin	Schn.	15	CSB
2	Lukas	Weil	12	CSB
3	Bene.	Venn.	10	CSB
4	Micha.	Schro.	11	MBS

1	15
2	12
3	10
4	11

Interactive Output:

```
val coffeePerWeek' : Series<int,string> =
0 -> 15
1 -> 12
2 -> 10
3 -> 11
```

Constructing a Deedle Data Frame

```
let persons = Frame(["fN";"lN";"cpw";"g"],[firstNames;lastNames;coffeesPerWeek;group])
let coffeePerWeek' :Series<int,int> = persons |> Frame.getCol ("cpw")
```

	fn	lN	cpw.	g
1	Kevin	Schn.	15	CSB
2	Lukas	Weil	12	CSB
3	Bene.	Venn.	10	CSB
4	Micha.	Schro.	11	MBS

1	15
2	12
3	10
4	11

Interactive Output:

```
val coffeePerWeek' : Series<int,int> =
0 -> 15
1 -> 12
2 -> 10
3 -> 11
```

Constructing a Deedle Data Frame

```
let persons = Frame(["fN";"lN";"cpw";"g"],[firstNames;lastNames;coffeesPerWeek;group])
let coffeePerWeek' :Series<int,int> = persons |> Frame.getCol ("cpw")
```

	fn	lN	cpw.	g
1	Kevin	Schn.	15	CSB
2	Lukas	Weil	12	CSB
3	Bene.	Venn.	10	CSB
4	Micha.	Schro.	11	MBS

1	15
2	12
3	10
4	11

Interactive Output:

```
val coffeePerWeek' : Series<int,int> =
0 -> 15
1 -> 12
2 -> 10
3 -> 11
```

Grouping Rows of a Deedle Data Frame

```
let persons = Frame(["fN";"lN";"cpw";"g"],[firstNames;lastNames;coffeesPerWeek;group])
let groupedByG :Frame<string*int,_> = persons |> Frame.groupRowsBy "g"
```

		fn	lN	cpw.	g
CSB	1	Kevin	Schn.	15	CSB
	2	Lukas	Weil	12	CSB
	3	Bene.	Venn.	10	CSB
MBS	4	Micha.	Schro.	11	MBS

Interactive Output:

```
val groupedByG : Frame<(string * int),string> =
                 fN       lN         cpw g
CSB 0 -> Kevin    Schneider 15  CSB
    1 -> Lukas    Weil      12  CSB
    2 -> Benedikt Venn      10  CSB
MBS 3 -> Michael  Schroda   11  MBS
```

Computational Systems Biology

Slicing columns of a Deedle Data Frame

```
let groupedByG :Frame<string*int,_> = persons |> Frame.groupRowsBy "g"
let withOutG :Frame<string*int,_> = groupedByG |> Frame.sliceCols ["fN";"lN";"cpw"]
```

		fn	lN	cpw.
CSB	1	Kevin	Schn.	15
	2	Lukas	Weil	12
	3	Bene.	Venn.	10
MBS	4	Micha.	Schro.	11

Interactive Output:

```
val withOutG : Frame<(string * int),string> =
                fN        lN         cpw
CSB 0 -> Kevin    Schneider 15
    1 -> Lukas    Weil      12
    2 -> Benedikt Venn      10
MBS 3 -> Michael  Schroda   11
```

Computational Systems Biology

Aggregating By index

```
let groupedByG :Frame<string*int,_> = persons |> Frame.groupRowsBy "g"
let coffeePerWeek'' :Series<string*int,int>= groupedByG |> Frame.getCol ("cpw")
```

		fn	lN	cpw.	g
CSB	1	Kevin	Schn.	15	CSB
	2	Lukas	Weil	12	CSB
	3	Bene.	Venn.	10	CSB
MBS	4	Micha.	Schro.	11	MBS

CSB	1	15
	2	12
	3	10
MBS	4	11

```
Interactive Output:   val coffeePerWeek'' : Series<(string * int),int> =

                      CSB 0 -> 15
                          1 -> 12
                          2 -> 10
                      MBS 3 -> 11
```

Computational Systems Biology

Aggregating By index

```
let coffeePerWeek'' :Series<string*int,int>= groupedByG |> Frame.getCol ("cpw")
let coffeePerWeekPerGroup =
        Series.applyLevel Pair.get1Of2 (Series.values >> Seq.sum) coffeePerWeek''
```

		fn	lN	cpw.	g
CSB	1	Kevin	Schn.	15	CSB
	2	Lukas	Weil	12	CSB
	3	Bene.	Venn.	10	CSB
MBS	4	Micha.	Schro.	11	MBS

CSB	1	15
	2	12
	3	10
MBS	4	11

```
Interactive Output:   val it : Series<string,int> =

                      CSB -> 37
                      MBS -> 11
```

Aggregating By index

```
let coffeePerWeek'' :Series<string*int,int>= groupedByG |> Frame.getCol ("cpw")
let coffeePerWeekPerGroup =
        Series.applyLevel Pair.get1Of2 (Series.values >> Seq.sum) coffeePerWeek''
```

		fn	lN	cpw.	g
CSB	1	Kevin	Schn.	15	CSB
	2	Lukas	Weil	12	CSB
	3	Bene.	Venn.	10	CSB
MBS	4	Micha.	Schro.	11	MBS

CSB	37
MBS	11

Interactive Output:

```
val it : Series<string,int> =

CSB -> 37
MBS -> 11
```

Back to CSV – Data Frame from file

The training data set:

Idx		ProtId	Peptide	Wt_rep1	Wt_rep2	...
0	->	AT1G	MNGHOKWVTRCRUNOT	0.756559369801931	-0.436239204755912	...
1	->	AT2G	VGCLFRCRPNLHGWHPCCAL	0.346180296054525	1.12382655833822	...
3	->	AT4G	FOWUNPOSSCTLOIEREDPC	2.07291451373885	2.33281672094402	...
2	->	AT3G	FCWTENSRCFEEWINFAG	1.94744188867146	1.40618787884551	...
4	->	AT5G	GWNHGEHORLA	2.49191364989232	2.55526019285937	...
5	->	AT1G	EGEGFKFOSAQGSNSSOKD	0.463247366431074	-0.185142152934379	...
6	->	AT2G	FQVOKNGMGEAPTPK	1.35116967991367	1.19087642041569	...
7	->	AT3G	AMVQLLKGHHN	1.63563065144294	1.24054827708146	...
8	->	AT4G	INWIHOOLHNTQEQHOAA	2.2376101319133	2.03897108568878	...
9	->	AT5G	DEQRDLKWRHQHCA	2.32829708747299	2.39682265165475	...

Dealing with CSV – Data Frame from/to file

```
let rawData :Frame<int,string> =
    Frame.ReadCsv(@"C:\Users\david\...\toyData.tab",separators="\t")
```

```
Interactive Output:
      Idx    ProtId Peptide                 Wt_rep1              Wt_rep2               ...
      0   -> AT1G   MNGHOKWVTRCRUNOT        0.756559369801931    -0.436239204755912    ...
      1   -> AT2G   VGCLFRCRPNLHGWHPCCAL    0.346180296054525    1.12382655833822      ...
      3   -> AT4G   FOWUNPOSSCTLOIEREDPC    2.07291451373885     2.33281672094402      ...
      2   -> AT3G   FCWTENSRCFEEWINFAG      1.94744188867146     1.40618787884551      ...
```

Dealing with CSV – Data Frame from/to file

```
let rawData :Frame<int,string> =
    Frame.ReadCsv(@"C:\Users\david\...\toyData.tab",separators="\t")

rawData.SaveCsv(@"C:\Users\david\...\toyData.tab",separator='\t',includeRowKeys=false)
```

```
Interactive Output:
      val it : unit = ()
```

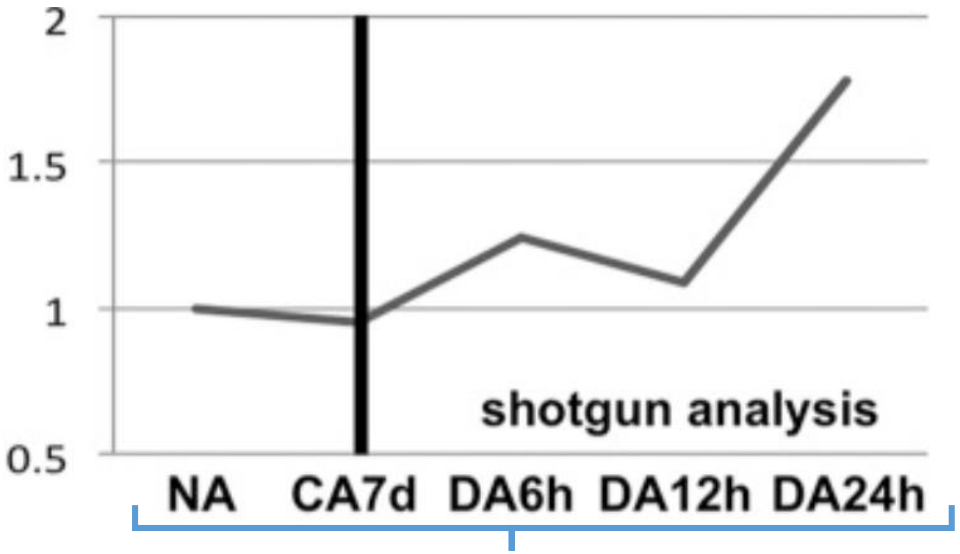
Your Enemy: The study

MOLECULAR & CELLULAR PROTEOMICS
Published by the American Society for Biochemistry and Molecular Biology

Mol Cell Proteomics. 2014 Dec; 13(12): 3602–3611.
Published online 2014 Oct 2. doi: 10.1074/mcp.M114.039081
PMCID: PMC4256508
PMID: 25277243

Analysis of Differential Expression Patterns of mRNA and Protein During Cold-acclimation and De-acclimation in *ArabidopsisS**

- Organism: Arabidopsis thaliana
- Aim : Study proteome dynamics during Cold-acclimation and De-acclimation
- Method : N14/N15 labled shotgun proteomics



5 timepoints
* 3 biological replicates
* 3 technical replicates
= 45 samples

Interactive programming

The end