If the data does not come to R, R must go to the data

Olga Kalinina

Helmholtz Institute for Pharmaceutical Research Saarland, Saarland University

FOSDEM PGDay 2019
Who am I?
Who am I?

- Bioinformatics = computational biology
Who am I?

- Bioinformatics = computational biology
  - Analysis of data to gain new biological insights
Who am I?

- Bioinformatics = computational biology
  - Analysis of data to gain new biological insights
  - Molecular biology
Who am I?

- Bioinformatics = computational biology
  - Analysis of data to gain new biological insights
  - Molecular biology
- Head of research group for drug bioinformatics at Helmholtz Institute for Pharmaceutical Research Saarland
Who am I?

- Bioinformatics = computational biology
  - Analysis of data to gain new biological insights
  - Molecular biology
- Head of research group for drug bioinformatics at Helmholtz Institute for Pharmaceutical Research Saarland
  - Find new bioactive compounds
Data in (life) sciences
Data in (life) sciences
Data in (life) sciences
Data in (life) sciences
Data in (life) sciences
Data in (life) sciences
Where does the data come from?
Where does the data come from?

• Experiment
Where does the data come from?

- Experiment
  - Genome sequencing
Where does the data come from?

• Experiment
  • Genome sequencing
Where does the data come from?

- Experiment
  - Genome sequencing
    - $\Rightarrow \sim 4 \times 10^{12} \text{ bp}$
Where does the data come from?

• Experiment
  • Genome sequencing
    • => ~4×10^{12} bp
  • Other types of experiment
Where does the data come from?

- Experiment
  - Genome sequencing
    - $=> \sim 4 \times 10^{12}$ bp
  - Other types of experiment
    - Determination of protein 3D structure
Where does the data come from?

- Experiment
  - Genome sequencing
    - => ~$4 \times 10^{12}$ bp
  - Other types of experiment
    - Determination of protein 3D structure
    - Gene expression
Where does the data come from?

• Experiment
  • Genome sequencing
    • => $\sim 4 \times 10^{12}$ bp
  • Other types of experiment
    • Determination of protein 3D structure
    • Gene expression
    • Computational predictions
How BIG is the data?
How BIG is the data?

• All DNA sequences: $\sim 4 \times 10^{12}$ bp = $\sim 9$ GB + metadata
How BIG is the data?

- All DNA sequences: \(\sim 4 \times 10^{12} \text{ bp} = \sim 9 \text{ GB} + \text{metadata}\)

- In this talk:
How BIG is the data?

- All DNA sequences: $\sim 4 \times 10^{12}$ bp = $\sim 9$ GB + metadata

- In this talk:
  - Clinically relevant mutations: 13 MB = 84,426 rows
How BIG is the data?

- All DNA sequences: $\sim 4 \times 10^{12}$ bp = $\sim 9$ GB + metadata

- In this talk:
  - Clinically relevant mutations: 13 MB = 84,426 rows
  - All human proteins + annotations: 1.9 GB = 23,095,049 rows
How BIG is the data?

- All DNA sequences: $\sim4 \times 10^{12}$ bp = $\sim9$ GB + metadata

- In this talk:
  - Clinically relevant mutations: 13 MB = 84,426 rows
  - All human proteins + annotations: 1.9 GB = 23,095,049 rows
  - (Cross-references from human proteins to other data sources: 147 MB = 6,026,631 rows)
Typical data analysis pipeline
Typical data analysis pipeline

Experiment (up to TBs of data)
Typical data analysis pipeline

Experiment (up to TBs of data)
Typical data analysis pipeline

Experiment (up to TBs of data)

Initial data processing, cross-referencing
Typical data analysis pipeline

Experiment (up to TBs of data)

Initial data processing, cross-referencing
Typical data analysis pipeline

1. Experiment (up to TBs of data)
2. Initial data processing, cross-referencing
3. Store in a DB
Typical data analysis pipeline

- Experiment (up to TBs of data)
- Initial data processing, cross-referencing
- Store in a DB
Typical data analysis pipeline

1. Experiment (up to TBs of data)
2. Initial data processing, cross-referencing
3. Store in a DB
4. Select relevant data
Typical data analysis pipeline

Experiment (up to TBs of data)

Initial data processing, cross-referencing

Store in a DB

Select relevant data
Typical data analysis pipeline

- Experiment (up to TBs of data)
- Initial data processing, cross-referencing
- Store in a DB
- Select relevant data
- Write to disc (text files, MBs to GBs)
Typical data analysis pipeline

- Experiment (up to TBs of data)
- Initial data processing, cross-referencing
- Store in a DB
- Select relevant data
- Write to disc (text files, MBs to GBs)
Typical data analysis pipeline

1. Experiment (up to TBs of data)
2. Initial data processing, cross-referencing
3. Store in a DB
4. Select relevant data
5. Write to disc (text files, MBs to GBs)
6. Analyze with dedicated statistical software (Python, SAS, R), typically in RAM
R programming language
R programming language

- Free software environment for statistical computing and graphics
R programming language

- Free software environment for statistical computing and graphics
- Introduced in 1993
R programming language

- Free software environment for statistical computing and graphics
- Introduced in 1993
- Multi-paradigm, including array: many generalized functions for multi-dimensional data (vectors, matrices, …)
R programming language

- Free software environment for statistical computing and graphics
- Introduced in 1993
- Multi-paradigm, including array: many generalized functions for multi-dimensional data (vectors, matrices, …)
- R project: https://www.r-project.org/
R programming language

• Free software environment for statistical computing and graphics

• Introduced in 1993

• Multi-paradigm, including array: many generalized functions for multi-dimensional data (vectors, matrices, …)

• R project: https://www.r-project.org/

• CRAN — 13,626 packages for various types of analysis: https://cran.r-project.org/
• R is still widely used, especially in academia
R is still widely used, especially in academia

Source: https://www.burtchworks.com/2017/06/19/2017-sas-r-python-flash-survey-results/
• R is still widely used, especially in academia

Source: https://www.burtchworks.com/2017/06/19/2017-sas-r-python-flash-survey-results/
R is still widely used, especially in academia.

Source: https://www.burtchworks.com/2017/06/19/2017-sas-r-python-flash-survey-results/
R

- R is still widely used, especially in academia
- R is very well suited to do statistical / machine learning

Source: https://www.burtchworks.com/2017/06/19/2017-sas-r-python-flash-survey-results/
R is still widely used, especially in academia.

R is very well suited to do statistical / machine learning.

Due to details of implementation, calculations in R are very efficient.

Source: https://www.burtchworks.com/2017/06/19/2017-sas-r-python-flash-survey-results/
PL/R
PL/R

- Procedural language that allows to write PostgreSQL functions and aggregate functions in R
PL/R

- Procedural language that allows to write PostgreSQL functions and aggregate functions in R
- Developed by Joe Conway since 2003
PL/R

- Procedural language that allows to write PostgreSQL functions and aggregate functions in R
- Developed by Joe Conway since 2003
- Implements full R functionality
This talk

• No technical details of implementation or management

• User perspective
Is it possible to do full cycle of data analysis using only PL/R?
Biology for dummies
Biology for dummies
Biology for dummies
Biology for dummies
Biology for dummies

The Cell

- membrane
- fluid
- nucleus

organelles
Biology for dummies

The Cell

Biological molecules: DNA, RNA, proteins
Proteins
Proteins

• Biological machines, responsible for (almost) all processes within the cell
Proteins

• Biological machines, responsible for (almost) all processes within the cell

• Encoded in genome as a sequence of characters
Proteins

• Biological machines, responsible for (almost) all processes within the cell

• Encoded in genome as a sequence of characters

• => synthesized as a chain of similar, yet not identical (chemically) units
Proteins

• Biological machines, responsible for (almost) all processes within the cell

• Encoded in genome as a sequence of characters

• => synthesized as a chain of similar, yet not identical (chemically) units

• Folded into 3D structures that makes them functional
Proteins

• Biological machines, responsible for (almost) all processes within the cell

• Encoded in genome as a sequence of characters

• => synthesized as a chain of similar, yet not identical (chemically) units

• Folded into 3D structures that makes them functional
Mutations
Mutations

- Happen in DNA
Mutations

- Happen in DNA
- Sources:
  - Spontaneous mistakes of DNA polymerase
  - Endogenous DNA damage
  - Exogenous DNA damage
Mutations

- Happen in DNA
- Sources:
  - Spontaneous mistakes of DNA polymerase
  - Endogenous DNA damage
  - Exogenous DNA damage
- Repair mechanisms => 1 mutation in $10^{10}$ nucleotides per cell division
Mutations

• Happen in DNA

• Sources:
  • Spontaneous mistakes of DNA polymerase
  • Endogenous DNA damage
  • Exogenous DNA damage

• Repair mechanisms => 1 mutation in $10^{10}$ nucleotides per cell division

• Cf. human genome size: $3 \times 10^9$ bp
The Central Dogma: flow of information in the living cells

Crick’s 1958 version of the “central dogma” of biology from Crick (1970, Figure 3, p. 562).
The Central Dogma: flow of information in the living cells

Crick’s 1958 version of the “central dogma” of biology from Crick (1970, Figure 3, p. 562).

The Central Dogma: flow of information in the living cells

Crick’s 1958 version of the “central dogma” of biology from Crick (1970, Figure 3, p. 562).
The Central Dogma: flow of information in the living cells

Crack’s 1958 version of the “central dogma” of biology from Crick (1970, Figure 3, p. 562).

The Central Dogma: flow of information in the living cells

Crick’s 1958 version of the “central dogma” of biology from Crick (1970, Figure 3, p. 562).

Protein thermodynamic stability
Protein thermodynamic stability

• Simple case: protein can unfold and refold rapidly, reversibly, via a two-state mechanism
Protein thermodynamic stability

- Simple case: protein can unfold and refold rapidly, reversibly, via a two-state mechanism

- \( \Delta G = G_{\text{unfolded}} - G_{\text{folded}} \)
Protein thermodynamic stability

• Simple case: protein can unfold and refold rapidly, reversibly, via a two-state mechanism

• $\Delta G = G_{\text{unfolded}} - G_{\text{folded}}$

• Upon mutations, $\Delta G$ can change:
  $\Delta \Delta G = \Delta G^\text{mut} - \Delta G^\text{WT}$
Protein thermodynamic stability

• Simple case: protein can unfold and refold rapidly, reversibly, via a two-state mechanism

\[ \Delta G = G_{\text{unfolded}} - G_{\text{folded}} \]

• Upon mutations, \( \Delta G \) can change:
\[ \Delta \Delta G = \Delta G_{\text{mut}} - \Delta G_{\text{WT}} \]

https://commons.wikimedia.org/w/index.php?curid=28353539
Some data (real-life)

- \( \Delta \Delta G \) estimates upon mutations

<table>
<thead>
<tr>
<th>chr</th>
<th>Gene</th>
<th>ClinicalSignificance</th>
<th>uniprot_ac</th>
<th>uniprot_pos</th>
<th>aa1</th>
<th>aa2</th>
<th>FX_ddG</th>
</tr>
</thead>
<tbody>
<tr>
<td>chr1</td>
<td>ISG15</td>
<td>Benign</td>
<td>P05161</td>
<td>83</td>
<td>S</td>
<td>N</td>
<td>-0.517133</td>
</tr>
<tr>
<td>chr2</td>
<td>DNMT3A</td>
<td>Pathogenic</td>
<td>Q9Y6K1</td>
<td>583</td>
<td>C</td>
<td>Y</td>
<td>33.0787</td>
</tr>
<tr>
<td>chr1</td>
<td>AGRN</td>
<td>Benign</td>
<td>O00468-6</td>
<td>15</td>
<td>P</td>
<td>R</td>
<td>?</td>
</tr>
</tbody>
</table>

... 

- 84,426 rows (13 MB)
Reading the data (R)

```r
> x <- read.table("clinvar.main.pph.ddg.uniprot.tsv", sep="\t", header=T)
> x[ x == "?" ] <- NA
> nrow(x)
84426

• => data frame
```
kalinina=# CREATE TABLE clinvar (chr text, tol bigint, ref text, alt text, GeneSymbol text, ClinicalSignificance text, ReviewStatus text, PhenotypeList text, uniprot_ac text, uniprot_pos int, aa1 char(1), aa2 char(1), prediction text, PDB_id text, PDB_pos text, PDB_ch char(1), ident float, FX_ddG float, IM_ddG float, M_ddG float, M_conf float);
CREATE TABLE

kalinina=# COPY clinvar FROM 'clinvar.main.pph.ddg.uniprot.tsv' WITH (NULL '?', DELIMITER E'\t');
COPY 84426
Calculate median (R)

>median(x$FX_ddG)
[1] NA
Calculate median (R)

```r
median(x$FX_ddG)
[1] NA

median(x$FX_ddG, na.rm=TRUE)
[1] 0.974858
```
Calculate median (R)

```r
> median(x$FX_ddG)
[1] NA

> median(x$FX_ddG, na.rm=TRUE)
[1] 0.974858

>(x[x$ClinicalSignificance=='Pathogenic',]$FX_ddG)
[1] 1.7756
```
Calculate median (R)

```r
> median(x$FX_ddG)
[1] NA

> median(x$FX_ddG, na.rm=TRUE)
[1] 0.974858

>(x[x$ClinicalSignificance==‘Pathogenic’,]$FX_ddG)
[1] 1.7756

> aggregate(FX_ddG ~ ClinicalSignificance, data = x, FUN = median)
  ClinicalSignificance  FX_ddG
1          Benign      0.62209
2 Pathogenic       1.77560
```
Calculate median (PL/R)

kalinina=# CREATE or REPLACE FUNCTION r_median(_float8) RETURNS float AS '
median(arg1)
' LANGUAGE 'plr';
CREATE FUNCTION

kalinina=# CREATE AGGREGATE median (sfunc = plr_array_accum,
basetype = float8, stype = _float8, finalfunc = r_median);
CREATE AGGREGATE

kalinina=# SELECT clinicalsignificance, median(fx_ddg) FROM clinvar GROUP BY clinicalsignificance ORDER BY clinicalsignificance;

clinicalsignificance | median
---------------------+----------
Benign               | 0.6220875
Pathogenic           | 1.7756
(2 rows)
### Summary statistics (R)

```r
> aggregate(FX_ddG ~ ClinicalSignificance, data = x, FUN = summary)

<table>
<thead>
<tr>
<th>ClinicalSignificance</th>
<th>FX_ddG.Min.</th>
<th>FX_ddG.1st Qu.</th>
<th>FX_ddG.Median</th>
<th>FX_ddG.Mean</th>
<th>FX_ddG.3rd Qu.</th>
<th>FX_ddG.Max.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Benign</td>
<td>-5.77969</td>
<td>-0.04082</td>
<td>0.62209</td>
<td>1.37172</td>
<td>1.91954</td>
<td>62.08970</td>
</tr>
<tr>
<td>Pathogenic</td>
<td>-18.09830</td>
<td>0.30438</td>
<td>1.77560</td>
<td>3.21887</td>
<td>4.21793</td>
<td>52.26050</td>
</tr>
</tbody>
</table>
```
Summary statistics (R)

```r
> aggregate(FX_ddG ~ ClinicalSignificance, data = x, FUN = summary)

          ClinicalSignificance FX_ddG.Min. FX_ddG.1st Qu. FX_ddG.Median FX_ddG.Mean FX_ddG.3rd Qu. FX_ddG.Max.
1         Benign         -5.77969       -0.04082          0.62209       1.37172       1.91954     62.08970
2 Pathogenic        -18.09830        0.30438          1.77560       3.21887       4.21793     52.26050

> aggregate(FX_ddG ~ ClinicalSignificance, data = x, FUN = summary)

          ClinicalSignificance FX_ddG.Min. FX_ddG.1st Qu. FX_ddG.Median
1         Benign         -5.77969       -0.04082          0.62209
2 Pathogenic        -18.09830        0.30438          1.77560

          FX_ddG.Mean FX_ddG.3rd Qu. FX_ddG.Max.
1         1.37172       1.91954         62.08970
2         3.21887       4.21793         52.26050
```
> aggregate(FX_ddG ~ ClinicalSignificance, data = x, FUN = summary)

<table>
<thead>
<tr>
<th>ClinicalSignificance</th>
<th>FX_ddG.Min.</th>
<th>FX_ddG.1st Qu.</th>
<th>FX_ddG.Median</th>
<th>FX_ddG.Mean</th>
<th>FX_ddG.3rd Qu.</th>
<th>FX_ddG.Max.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Benign</td>
<td>-5.77969</td>
<td>-0.04082</td>
<td>0.62209</td>
<td>1.37172</td>
<td>1.91954</td>
<td>62.08970</td>
</tr>
<tr>
<td>Pathogenic</td>
<td>-18.09830</td>
<td>0.30438</td>
<td>1.77560</td>
<td>3.21887</td>
<td>4.21793</td>
<td>52.26050</td>
</tr>
</tbody>
</table>

> aggregate(FX_ddG ~ ClinicalSignificance, data = x, FUN = summary)

<table>
<thead>
<tr>
<th>ClinicalSignificance</th>
<th>FX_ddG.Min.</th>
<th>FX_ddG.1st Qu.</th>
<th>FX_ddG.Median</th>
<th>FX_ddG.Mean</th>
<th>FX_ddG.3rd Qu.</th>
<th>FX_ddG.Max.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Benign</td>
<td>-5.77969</td>
<td>-0.04082</td>
<td>0.62209</td>
<td>1.37172</td>
<td>1.91954</td>
<td>62.08970</td>
</tr>
<tr>
<td>Pathogenic</td>
<td>-18.09830</td>
<td>0.30438</td>
<td>1.77560</td>
<td>3.21887</td>
<td>4.21793</td>
<td>52.26050</td>
</tr>
</tbody>
</table>

You need additional code if you need to preserve a specific order of categories
Summary statistics
(PL/R)

kalinina=# CREATE or REPLACE FUNCTION r_summary(_float8) RETURNS _float8 AS 'summary(arg1)
' LANGUAGE 'plr';
CREATE FUNCTION

kalinina=# CREATE AGGREGATE summary (sfunc = plr_array_accum,
basetype = float8,
stype = _float8,
finalfunc = r_median);
CREATE AGGREGATE

kalinina=# SELECT clinicalsignificance, SELECT summary(fx_ddg) FROM clinvar GROUP BY clinicalsignificance ORDER BY clinicalsignificance;

clinicalsignificance | summary
---------------------+--------------------------------------------------------------------
Benign               | {-5.77969,-0.040819875,0.6220875,1.37171750416516,1.9195375,62.0897} 
Pathogenic           | {-18.0983,0.3043845,1.7756,3.21886833468419,4.217925,52.2605} 
(2 rows)
Boxplot (R)

```r
> boxplot(x[ x$ClinicalSignificance == 'Pathogenic', ]$FX_ddG)
```
Boxplot (R)

> boxplot(x[ x$ClinicalSignificance == 'Pathogenic', ]$FX_ddG)
Boxplot (R)

```r
> boxplot(x[ x$ClinicalSignificance == 'Pathogenic', ]$FX_ddG)
```

- Syntax for subsetting:
  ```r
  x[ x$<someFactor> == '<someValue>', ]
  ```
Boxplot (R)

```r
> boxplot(x[ x$ClinicalSignificance == 'Pathogenic', ]$FX_ddG)
```

- Syntax for subsetting:
  ```r
  x[ x$<someFactor> == '<someValue>', ]
  ```

- Output directly to active graphic device
Boxplot (R)

```r
> boxplot(x[ x$ClinicalSignificance == 'Pathogenic', ]$FX_ddG)
```

- Syntax for subsetting:
  ```r
  x[ x$<someFactor> == '<someValue>', ]
  ```

- Output directly to active graphic device
Boxplot (PL/R)

CREATE or REPLACE function r_boxplot2(_float8) RETURNS void AS '
pdf("~/Work/ddG/test.pdf") boxplot(arg1)
dev.off()
' language 'plr';
CREATE FUNCTION

kalinina=# CREATE AGGREGATE boxplot2pdf (sfunc = plr_array_accum,
basetype = float8,
stype = _float8,
finalfunc = r_boxplot2 )
CREATE AGGREGATE

kalinina=# SELECT boxplot2pdf(fx_ddg)
FROM clinvar WHERE clinicalsignificance = 'Pathogenic';
boxplot2pdf

(1 row)
Boxplot (PL/R)

CREATE or REPLACE function
r_boxplot2(_float8) RETURNS void AS 'pdf("~/Work/ddG/test.pdf")

kalinina=# CREATE AGGREGATE boxplot2pdf (sfunc = plr_array_accum,
basetype = float8,
stype = _float8,
finalfunc = r_boxplot2);
CREATE AGGREGATE

kalinina=# SELECT boxplot2pdf(fx_ddg)
FROM clinivar WHERE clinicalsignificance = 'Pathogenic';
boxplot2pdf

(1 row)
More data (real-life)

- **Structural annotation** of the human proteome

<table>
<thead>
<tr>
<th>#AC</th>
<th>Mut</th>
<th>Species</th>
<th>Tags</th>
<th>Surface/Core</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>P30613</td>
<td>R498</td>
<td>HUMAN</td>
<td>None</td>
<td>Surface</td>
<td>Ligand</td>
</tr>
<tr>
<td>P30613</td>
<td>G411</td>
<td>HUMAN</td>
<td>None</td>
<td>Core</td>
<td>Core</td>
</tr>
<tr>
<td>P30613</td>
<td>R559</td>
<td>HUMAN</td>
<td>None</td>
<td>None</td>
<td>Disorder</td>
</tr>
</tbody>
</table>

- Every protein position is classified as **Surface**, **Core**, **Ligand**, **Metal**, **Protein**, **DNA**, **RNA**, or **Disorder** (8 categories)

- 23,095,049 rows (1.9 GB)
Pie chart (R)

```r
> p <- read.table("proteome.classification.tsv", sep="\t")
> p[ p == "None" ] <- NA
> pp <- p[p$Class <> 'Disorder', ]
> piedata <- aggregate(pp$AC, by=list(Category=pp$Class), FUN=length)
> piedataOrdered <- piedata[ order(-piedata$x), ]
> piedataOrdered
     Category  x
    7  Surface 6411178
    1    Core 4519347
    5   Protein 2228705
    3   Ligand  934970
    4   Metal  830419
    2     DNA  265432
    6    RNA  69701
> pie(piedataOrdered$x/nrow(pp),
    labels=piedataOrdered$Category)
```
Pie chart (R)

```r
> p <- read.table("proteome.classification.tsv", sep="\t")
> p[ p == "None" ] <- NA
> pp <- p[p$Class <> 'Disorder',]
> piedata <- aggregate(pp$AC, by=list(Category=pp$Class), FUN=length)
> piedataOrdered <- piedata[ order(-piedata$x),]
> piedataOrdered

<table>
<thead>
<tr>
<th>Category</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>Surface</td>
<td>6411178</td>
</tr>
<tr>
<td>Core</td>
<td>4519347</td>
</tr>
<tr>
<td>Protein</td>
<td>2228705</td>
</tr>
<tr>
<td>Ligand</td>
<td>934970</td>
</tr>
<tr>
<td>Metal</td>
<td>830419</td>
</tr>
<tr>
<td>DNA</td>
<td>265432</td>
</tr>
<tr>
<td>RNA</td>
<td>69701</td>
</tr>
</tbody>
</table>

> pie(piedataOrdered$x/nrow(pp), labels=piedataOrdered$Category)
```
Pie chart (PL/R)

```sql
kalinina=# CREATE VIEW piechart AS SELECT class, CAST(count(ac) AS float)/(SELECT count(ac) FROM structman WHERE class <> 'Disorder') AS percentage FROM structman WHERE class <> 'Disorder' GROUP BY class ORDER BY percentage DESC;
CREATE VIEW

kalinina=# CREATE or REPLACE function r_pie(_float8) RETURNS void AS '
pdf("~/Work/ddG/testpie.pdf")
pie(arg1)
dev.off()'
' language 'plr';
CREATE FUNCTION

kalinina=# CREATE AGGREGATE pie2pdf (sfunc = plr_array_accum,
basetype = float8,
stype = _float8,
finalfunc = r_pie);
CREATE AGGREGATE

kalinina=# SELECT pie2pdf(percentage) FROM piechart;
pie2pdf
---------
34
(1 row)
```
Pie chart (PL/R)

Kalinina=# CREATE VIEW piechart AS SELECT class, CAST(count(ac) AS float)/(SELECT count(ac) FROM structman WHERE class <> 'Disorder') AS percentage FROM structman WHERE class <> 'Disorder' GROUP BY class ORDER BY percentage DESC;
CREATE VIEW

Kalinina=# CREATE or REPLACE function r_pie(_float8) RETURNS void AS 'pdf("~/Work/ddG/testpie.pdf")
pie(arg1)
dev.off();'
' language 'plr';
CREATE FUNCTION

Kalinina=# CREATE AGGREGATE pie2pdf ( sfunc = plr_array_accum, basetype = float8, stype = _float8, finalfunc = r_pie );
CREATE AGGREGATE

Kalinina=# SELECT pie2pdf(percentage) FROM piechart;
pie2pdf
---------

(1 row)
Pie chart (PL/R)

```sql
kalinina=# CREATE VIEW piechart AS SELECT class, CAST(count(ac) AS float)/(SELECT count(ac) FROM structman WHERE class <> 'Disorder') AS percentage FROM structman WHERE class <> 'Disorder' GROUP BY class ORDER BY percentage DESC;
CREATE VIEW

kalinina=# CREATE or REPLACE function r_pie(_float8) RETURNS void AS '
pdf("~/Work/ddG/testpie.pdf")
pie(arg1)
dev.off()
' language 'plr';
CREATE FUNCTION

kalinina=# CREATE AGGREGATE pie2pdf (sfunc = plr_array_accum, basetype = float8, stype = _float8, finalfunc = r_pie);
CREATE AGGREGATE

kalinina=# SELECT pie2pdf(percentage) FROM piechart;
pie2pdf
---------

(1 row)
```

No clean solution to pass the names of the categories
Now it starts to pay off
Now it starts to pay off

- \( pp \) (all rows except ‘Disorder’) has 15,259,752 rows
Now it starts to pay off

- \( pp \) (all rows except ‘Disorder’) has 15,259,752 rows

- The most expensive command in R:
  
  ```r
  aggregate(pp$AC, by=list(Category=pp$Class), FUN=length)
  ```
  
  takes \(~6.3\) sec to execute
Now it starts to pay off

- \( pp \) (all rows except ‘Disorder’) has 15,259,752 rows

- The most expensive command in R:
  
  ```
  aggregate(pp$AC, by=list(Category=pp$Class), FUN=length)
  ```
  
  takes \( \sim 6.3 \) sec to execute

- Selection from `piechart` in the database takes 1.97 sec
Now it starts to pay off

- \( pp \) (all rows except ‘Disorder’) has 15,259,752 rows

- The most expensive command in R:
  
  \[
  \text{aggregate}(pp\text{$AC$, by=list(Category=pp\text{$Class$)}, FUN=length)}
  \]
  
  takes \( \sim 6.3 \) sec to execute

- Selection from \texttt{piechart} in the database takes 1.97 sec

- On the other hand, running \texttt{median} grouped by \texttt{Class} will never finish: full table scan
Statistical significance

- R has implementations of a variety of statistical tests, e.g. Wilcoxon test:
Statistical significance

- R has implementations of a variety of statistical tests, e.g. Wilcoxon test:

  ```r
  > wilcox.test(x[x$ClinicalSignificance==‘Pathogenic’,]$FX_ddG,
  x[x$ClinicalSignificance==‘Benign’,]$FX_ddG)
  
  Wilcoxon rank sum test with continuity correction
  
  data:  x[x$ClinicalSignificance == "Pathogenic", ]$FX_ddG and
  x[x$ClinicalSignificance == "Benign", ]$FX_ddG
  W = 4419800, p-value < 2.2e-16
  alternative hypothesis: true location shift is not equal to 0
  ```
Statistical significance

- R has implementations of a variety of statistical tests, e.g. Wilcoxon test:

```r
> wilcox.test(x[x$ClinicalSignificance=='Pathogenic',]$FX_ddG, x[x$ClinicalSignificance=='Benign',]$FX_ddG)

Wilcoxon rank sum test with continuity correction
data:  x[x$ClinicalSignificance == "Pathogenic", ]$FX_ddG and x[x$ClinicalSignificance == "Benign", ]$FX_ddG
W = 4419800, p-value < 2.2e-16
alternative hypothesis: true location shift is not equal to 0
```

```r
> wilcox.test(x[x$ClinicalSignificance=='Pathogenic',]$FX_ddG, x[x$ClinicalSignificance=='Benign',]$FX_ddG)$p.value
[1] 1.033810e-167
```
Passing two arrays of datapoint

```sql
CREATE TABLE ddg (pathogenic float, benign float);

INSERT INTO ddg(pathogenic) SELECT fx_ddg FROM clinvar WHERE clinicalsignificance = 'Pathogenic';
INSERT 0 20336

INSERT INTO ddg(benign) SELECT fx_ddg FROM clinvar WHERE clinicalsignificance = 'Benign';
INSERT 0 64090

CREATE TABLE ddg_all (ddg float);

INSERT INTO ddg_all(ddg) SELECT pathogenic FROM ddg;
INSERT 0 84426

INSERT INTO ddg_all(ddg) SELECT benign FROM ddg;
INSERT 0 84426
```
...and calculating statistical significance

```
kalinina=# CREATE OR REPLACE FUNCTION r_wilcox(_float8) RETURNS float AS 'x<-arg1[1:length(arg1)/2] y<-arg1[length(arg1)/2+1:length(arg1)] wilcox.test(x,y)$p.value' language 'plr';
CREATE FUNCTION

kalinina=# CREATE AGGREGATE wilcox ( sfunc = plr_array_accum, basetype = float8, stype = _float8, finalfunc = r_wilcox );
CREATE AGGREGATE

kalinina=# SELECT wilcox(ddg) FROM ddg_all;
   wilcox
--------------
1.0338096840586e-167
(1 row)
```
...draw plots with two series

```plsql
kalinina=# CREATE OR REPLACE FUNCTION r_plottwo(_float8) RETURNS float AS
'pdf("testtwo.pdf")
x<-arg1[1:length(arg1)/2]
y<-arg1[length(arg1)/2+1:length(arg1)]
boxplot(x,y)
dev.off()
' language 'plr';
CREATE FUNCTION

kalinina=# CREATE AGGREGATE plottwo (
sfunc = plr_array_accum,
basetype = float8,
stype = _float8,
finalfunc = r_plottwo
);
CREATE AGGREGATE

kalinina=# SELECT plottwo(ddg) FROM ddg_all;
  plottwo
-----------------------
   (1 row)
```
...draw plots with two series

calinina=# CREATE OR REPLACE FUNCTION r_plottwo(_float8) RETURNS float AS
'
  pdf("testtwo.pdf")
x<-arg1[1:length(arg1)/2]
y<-arg1[length(arg1)/2+1:length(arg1)]
boxplot(x,y)
dev.off()
' language 'plr';
CREATE FUNCTION

calinina=# CREATE AGGREGATE plottwo ( sfunc = plr_array_accum,
    basetype = float8,
    stype = _float8,
    finalfunc = r_plottwo
);
CREATE AGGREGATE

calinina=# SELECT plottwo(ddg) FROM ddg_all;

---------------------

plottwo

---------------------

plottwo

---------------------

(1 row)
Joins (R)

- Theoretically, you can join in R
Joins (R)

• Theoretically, you can join in R

• Let’s do an inner join:

```r
x: chr Gene ClinicalSignificance uniprot_ac uniprot_pos aa1 aa2 FX_ddG
p: AC Mut Species Tags Surface/Core Class
```
Joins (R)

• Theoretically, you can join in R

• Let’s do an inner join:

x: chr Gene ClinicalSignificance uniprot_ac uniprot_pos aa1 aa2 FX_ddG

p: AC Mut Species Tags Surface/Core Class

> library (dplyr)
> joined_data <- t %>% inner_join(p, by = c(c(x$uniprot_ac == p$AC)),
c(x$uniprot_pos == p$Mut)))
Error in Ops.factor(x$uniprot_ac, p$AC) : level sets of factors are different

• You have to have the same set of identifiers in both tables!
Joins (PL/R)

kalinina=# SELECT DISTINCT structman.ac AS ac, clinicalsignificance, fx_ddg INTO core FROM clinvar INNER JOIN structman ON structman.ac = clinvar.uniprot_ac AND structman.mut = clinvar.aal||clinvar.uniprot_pos WHERE structman.class = 'Core';
SELECT 6637

kalinina=# SELECT DISTINCT structman.ac AS ac, clinicalsignificance, fx_ddg INTO notcore FROM clinvar INNER JOIN structman ON structman.ac = clinvar.uniprot_ac AND structman.mut = clinvar.aal||clinvar.uniprot_pos WHERE structman.class <> 'Core';
SELECT 13430
### Joins (PL/R)

kalinina=# SELECT clinicalsignificance, median(fx_ddg) FROM clinvar GROUP BY clinicalsignificance;

<table>
<thead>
<tr>
<th>clinicalsignificance</th>
<th>median</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pathogenic</td>
<td>1.7756</td>
</tr>
<tr>
<td>Benign</td>
<td>0.6220875</td>
</tr>
</tbody>
</table>

(2 rows)

kalinina=# SELECT clinicalsignificance, median(fx_ddg) FROM core GROUP BY clinicalsignificance;

<table>
<thead>
<tr>
<th>clinicalsignificance</th>
<th>median</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pathogenic</td>
<td>3.4113</td>
</tr>
<tr>
<td>Benign</td>
<td>1.55485</td>
</tr>
</tbody>
</table>

(2 rows)

kalinina=# SELECT clinicalsignificance, median(fx_ddg) FROM notcore GROUP BY clinicalsignificance;

<table>
<thead>
<tr>
<th>clinicalsignificance</th>
<th>median</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pathogenic</td>
<td>1.003565</td>
</tr>
<tr>
<td>Benign</td>
<td>0.424089</td>
</tr>
</tbody>
</table>

(2 rows)
Summary
Summary

- Data analysis can be done with PL/R (almost) as easily as in the R environment
Summary

- Data analysis can be done with PL/R (almost) as easily as in the R environment

- Syntax is more cumbersome
Summary

• Data analysis can be done with PL/R (almost) as easily as in the R environment

• Syntax is more cumbersome

• Passing two arrays of datapoints is a problem
Summary

- Data analysis can be done with PL/R (almost) as easily as in the R environment
- Syntax is more cumbersome
- Passing two arrays of datapoints is a problem
- However, one can benefit from data handling in the database
Summary

• Data analysis can be done with PL/R (almost) as easily as in the R environment

• Syntax is more cumbersome

• Passing two arrays of datapoints is a problem

• However, one can benefit from data handling in the database

• **Feedback:** [https://2019.fosdempgday.org/f](https://2019.fosdempgday.org/f)