G53FIV: Fundamentals of Information Visualization

Lecture 7: Visualization with R – Advanced and Visualization Tools

Ke Zhou
School of Computer Science
Ke.Zhou@nottingham.ac.uk

https://moodle.nottingham.ac.uk/course/view.php?id=96914

Last Lecture

Visualization with R

R is a tool for…

Data Manipulation

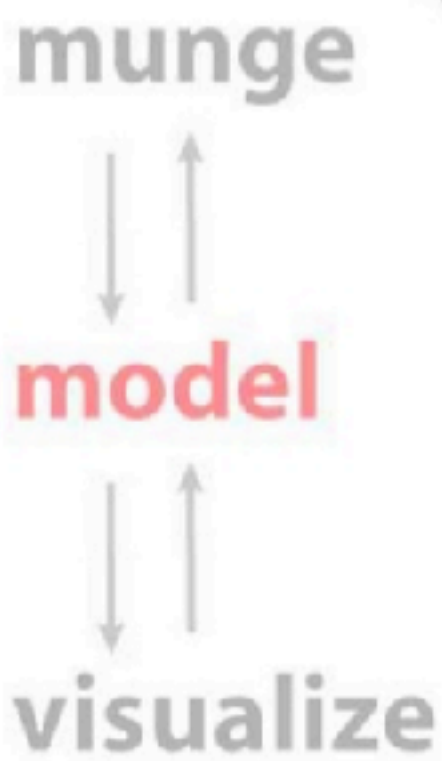
- connecting to data sources
- slicing & dicing data

Modeling & Computation

- statistical modeling
- numerical simulation

Data Visualization

- visualizing fit of models
- composing statistical graphics



Building a Plot in ggplot2

data to visualize (a data frame)
map variables to ***aes***thetic attributes
geometric objects – what you see (points, bars, etc)
scales map values from data to aesthetic space

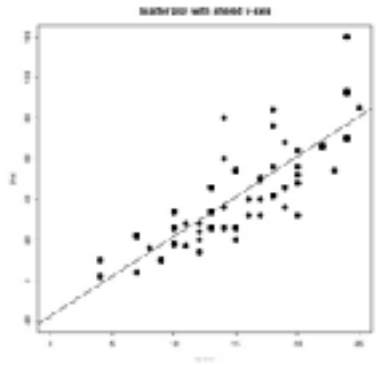
```
ggplot(iris) + geom_point(aes(x = Sepal.Length, y = Sepal.Width))
```

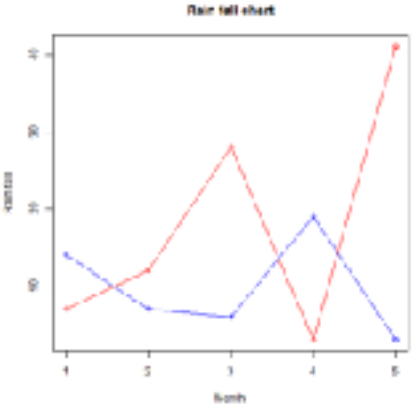
Data

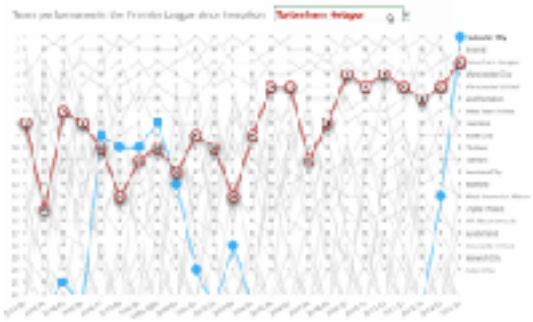
Geometric objects to display

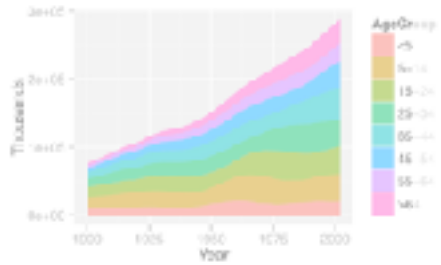
Aesthetics map variables to scales

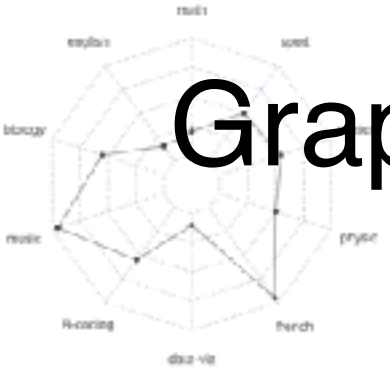
Graphs

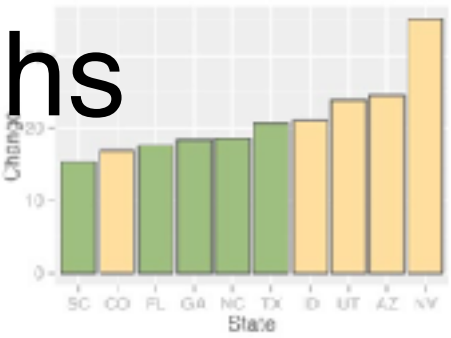


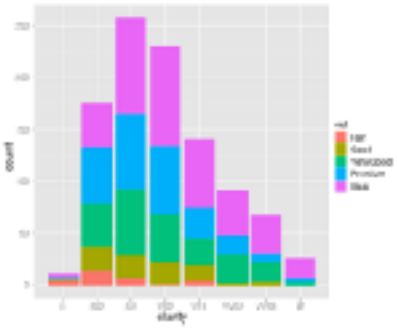


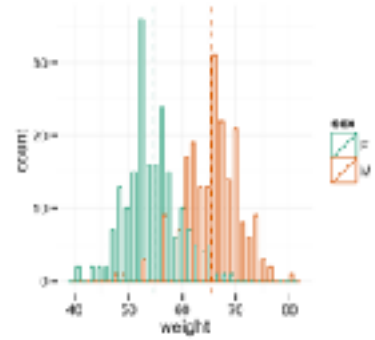


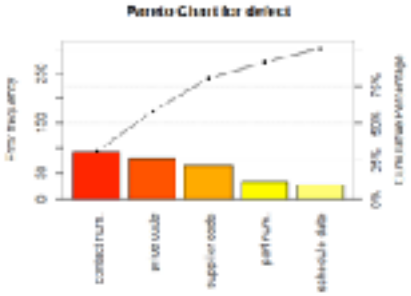


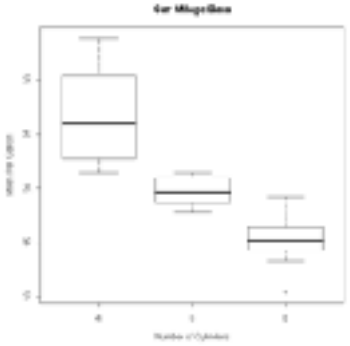


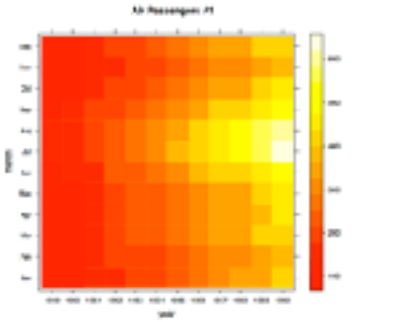




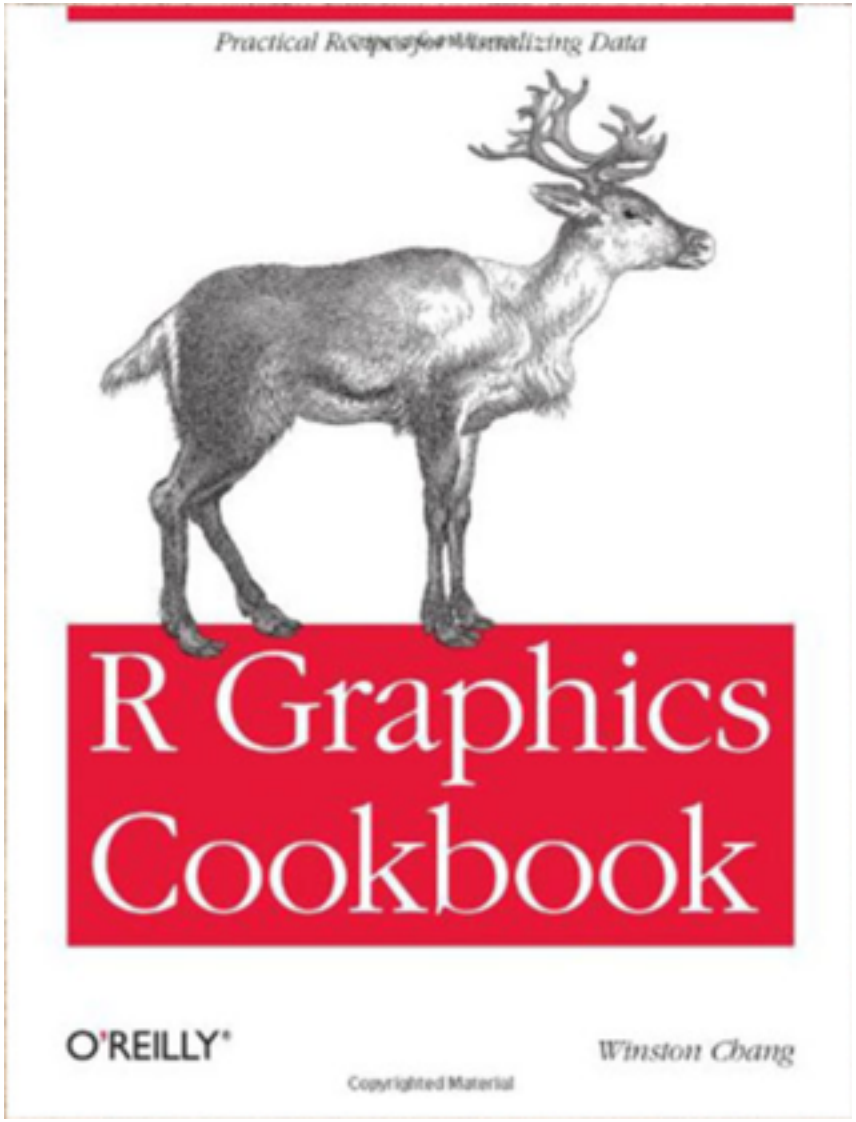




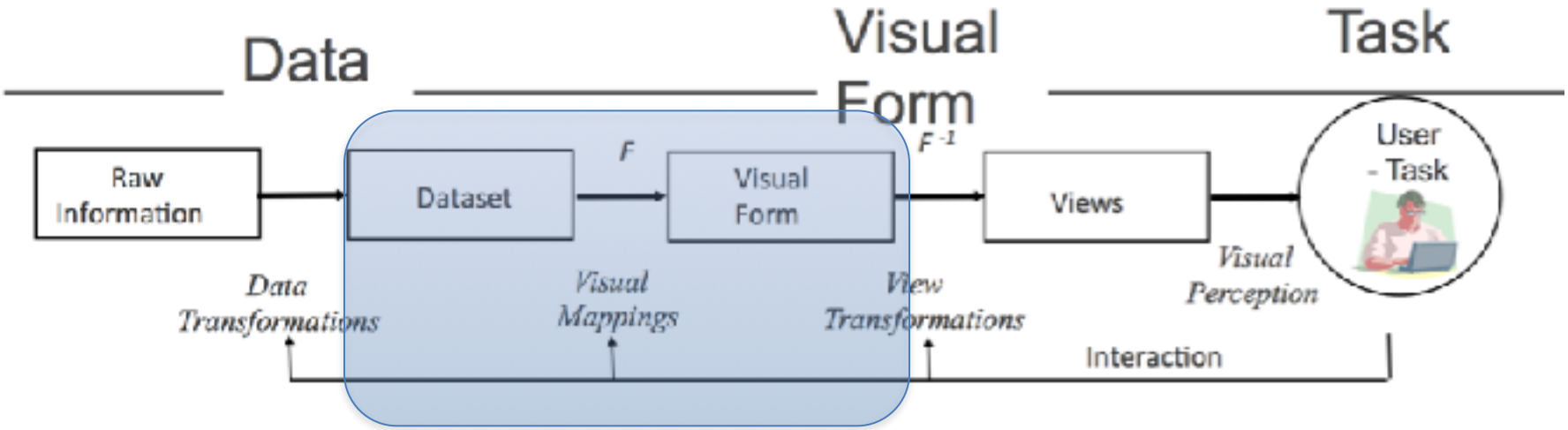




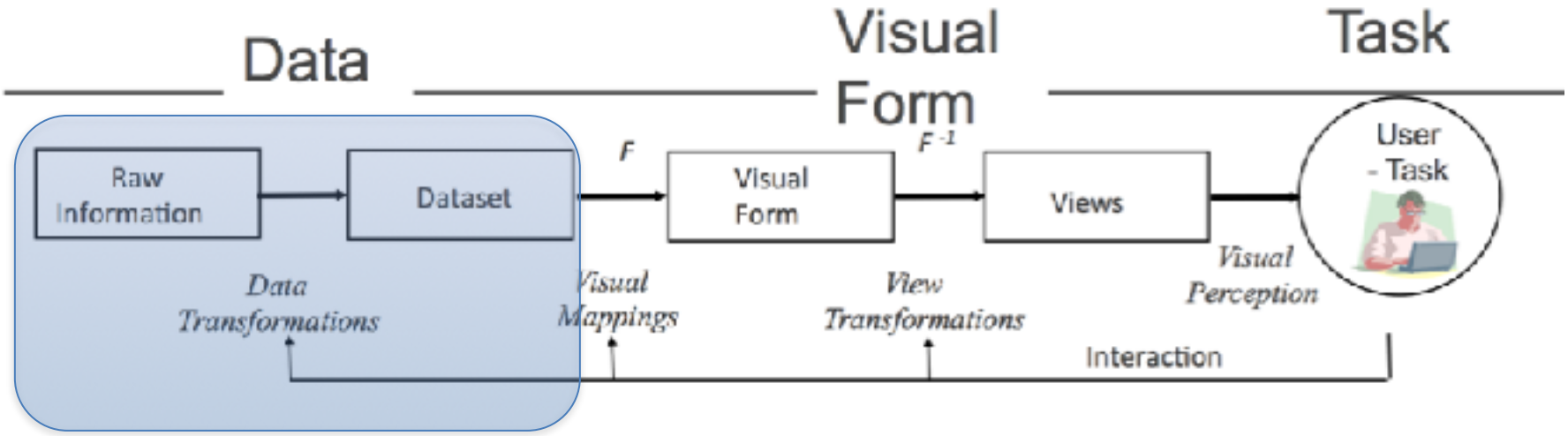
Dr. Ke Zhou (http://www.cs.nott.ac.uk/~pszkz/)


Practical Recipes for Visualizing Data
R Graphics Cookbook
O'REILLY
Winston Chang
Copyrighted Material

Seven Stages of Visualization



Seven Stages of Visualization



G53FIV Early Module Feedback

- Survey (on Moodle)
  - Anonymous
  - Seeking for constructive feedback
  - Two compulsory multiple questions
  - A few open-ended questions
    - If you have specific feedback or suggestions for improvement.

- A summary of the survey (and action points for improvements) will be presented.

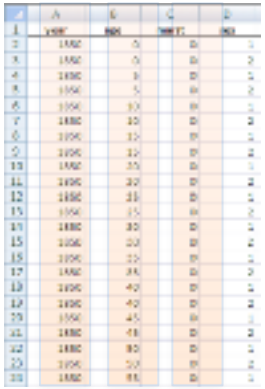
Overview

- Data Manipulations with R
- Visualization Tools

Data Manipulations with R

Dr. Ke Zhou (http://www.cs.nott.ac.uk/~pszkz/)

Transform Data: A Swiss-Army Knife

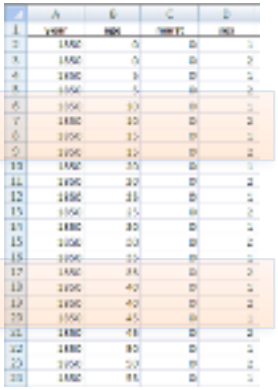


- Indexing
- Three ways to index into a data frame
  - Array of integer indices
  - Array of character names
  - Array of logical Booleans
- Examples:
  - **df[1:3,]**
  - **df[c("New York", "Chicago"),]**
  - **df[c(TRUE, FALSE, TRUE, TRUE),]**

Transform Data: A Swiss-Army Knife

- subset – extract subsets meeting some criteria

```
subset(Insurance, District==1)
subset(Insurance, Claims < 20)
```



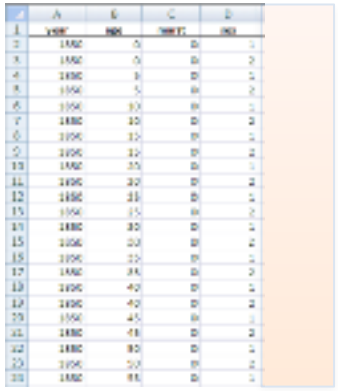
Transform Data: A Swiss-Army Knife

- subset – extract subsets meeting some criteria

```
subset(Insurance, District==1)
subset(Insurance, Claims < 20)
```

- transform – add or alter a column of a data frame

```
transform(Insurance, Propensity=Claims/Holders)
```



Transform Data: A Swiss-Army Knife

- subset – extract subsets meeting some criteria

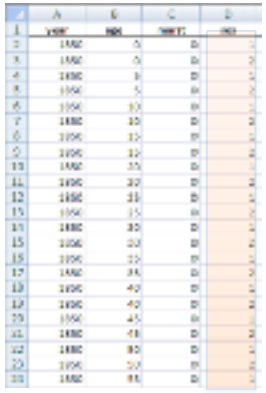
```
subset(Insurance, District==1)
subset(Insurance, Claims < 20)
```

- transform – add or alter a column of a data frame

```
transform(Insurance, Propensity=Claims/Holders)
```

- cut – cut a continuous value into groups

```
cut(Insurance$Claims, breaks=c(-
1,100,Inf), labels=c('lo','hi'))
```



Transform Data: A Swiss-Army Knife

- subset – extract subsets meeting some criteria

```
subset(Insurance, District==1)
subset(Insurance, Claims < 20)
```

- transform – add or alter a column of a data frame

```
transform(Insurance, Propensity=Claims/Holders)
```

- cut – cut a continuous value into groups

```
cut(Insurance$Claims, breaks=c(-
1,100,Inf), labels=c('lo','hi'))
```

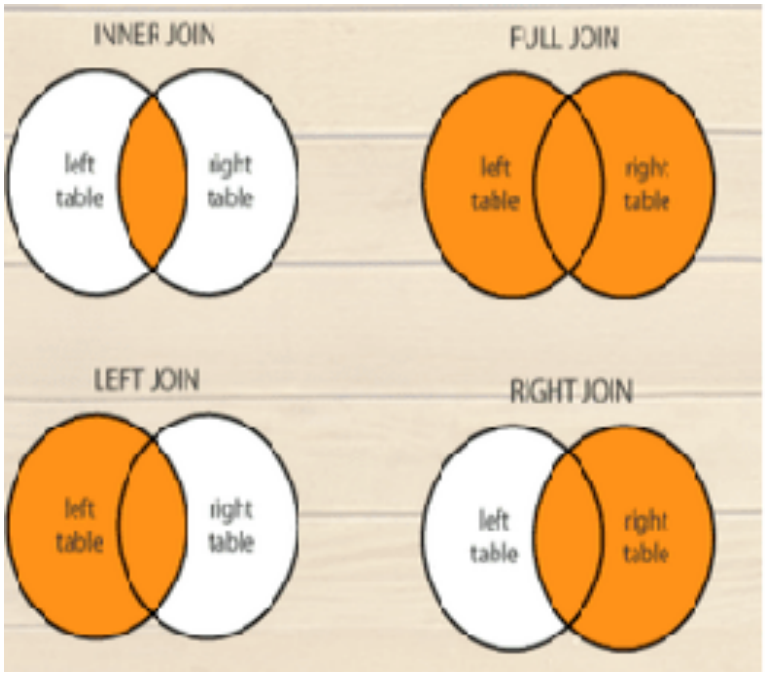
- Put it all together: create a new, transformed data frame

```
transform(subset(Insurance, District==1),
   ClaimLevel=cut(Claims, breaks=c(-1,100,Inf),
   labels=c('lo','hi')))
```

Joining Two Data Frames

- inner_join(df1, df2, by = "common_column")
- ?join
  - Left_join, right_join
  - Inner_join, outer_join

- merge(x=df1, y=df2, by.x="id", by.y="bid")



Libraries for Data Manipulations

- Packages
  - plyr
  - data.table
  - reshape2
  - doBY
  - sqldf
  - and many more

dplyr: A Grammar of Data Manipulation

- Very intuitive, once you understand the basics
- Very fast
  - Created with execution times in mind
- Easy for those migrating from the SQL world
- When written well, your code reads like a "recipe"
- "Code the way you think"

https://cran.rstudio.com/web/packages/dplyr/vignettes/introduction.html

Pipe Operator

- Library(maggritr)
  - A R package launched on Jan 2014
  - A "magic" operator called the PIPE was introduced
  - %>%
  - i.e. "AND THEN", "PIPE TO"

```
round(sqrt(1000), 3)

library(magrittr)
1000 %>% sqrt %>% round()
1000 %>% sqrt %>% round(.,3)
```

dplyr

- dplyr takes the %>% operator and uses it to great effect for manipulating data frames
  - Works only with data frames
  - 5 basic "verbs" work for 90% of data

Verbs	What does it do?
`filter()`	Select a subset of ROWS by conditions
`arrange()`	Reorders ROWS in a data frame
`select()`	Select the COLUMNS of interest
`mutate()`	Create new columns based on existing columns (mutations!)
`summarise()`	Aggregate values for each group, reduces to single value

5 Basic Verbs

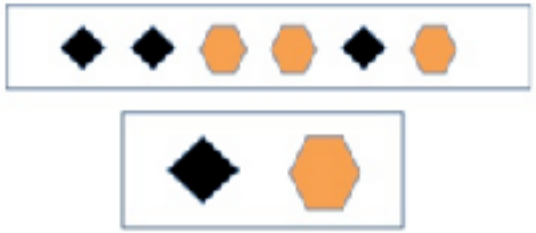
- FILTERows

- SELECT Column Types

- ArRange Rows (SORT)

- Mutate (into something new)

- **Summarize** by Groups



Movies dataset

title	year	budget	votes	length	Documentary	rating	…
Titanic	1997	200,000,000	1000	195	0	7.8	…
Leon	1994	16,000,000	500	90	0	8.6	…
McQueen	2018	52,000,000	200	91	1	7.9	…

Filter()

- Usage: `filter(data, condition)`
  - Returns a subset of rows
  - Multiple conditions can be supplied.
  - They are combined with an AND

```
movies_with_budgets <- filter(movies_df, !is.na(budget))
filter(movies, Documentary==1)
filter(movies, Documentary==1) %>% nrow()
```

Filter()

- Usage: `filter(data, condition)`
  - Returns a subset of rows
  - Multiple conditions can be supplied.
  - They are combined with an AND

```
movies_with_budgets <- filter(movies_df, !is.na(budget))
filter(movies, Documentary==1)
filter(movies, Documentary==1) %>% nrow()
good_comedies <- filter(movies, rating > 9, Comedy==1)
dim(good_comedies) #171 movies
```

Filter()

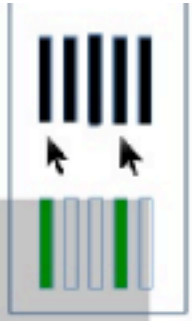
- Usage: `filter(data, condition)`
  - Returns a subset of rows
  - Multiple conditions can be supplied.
  - They are combined with an AND

```
movies_with_budgets <- filter(movies_df, !is.na(budget))
filter(movies, Documentary==1)
filter(movies, Documentary==1) %>% nrow()
good_comedies <- filter(movies, rating > 9, Comedy==1)
dim(good_comedies) #171 movies

#' Let us  say we only want highly rated comdies, which a lot
of people have watched, made after year 2000.
movies %>%
  filter(rating >8, Comedy==1, votes > 100, year > 2000)
```

Select()

- Usage:

```
select(data, columns)
```

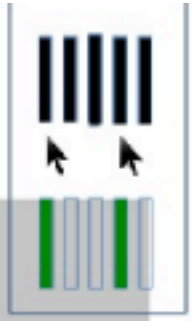


```
movies_df <- tbl_df(movies)
select(movies_df, title, year, rating) #Just the columns we want to see
select(movies_df, -c(r1:r10)) #we don't want certain columns
```

Select()

- Usage:

```
select(data, columns)
```

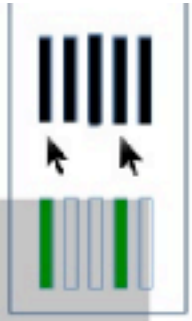


```
movies_df <- tbl_df(movies)
select(movies_df, title, year, rating) #Just the columns we want to see
select(movies_df, -c(r1:r10)) #we don't want certain columns

#You can also select a range of columns from start:end
select(movies_df, title:votes)   # All the columns from title to votes
select(movies_df, -c(budget, r1:r10, Animation, Documentary, Short, Romance))
```

Select()

- Usage:

```
select(data, columns)
```

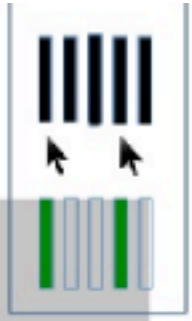


```
movies_df <- tbl_df(movies)
select(movies_df, title, year, rating) #Just the columns we want to see
select(movies_df, -c(r1:r10)) #we don't want certain columns

#You can also select a range of columns from start:end
select(movies_df, title:votes)   # All the columns from title to votes
select(movies_df, -c(budget, r1:r10, Animation, Documentary, Short, Romance))

select(movies_df, contains("r"))  # Any column that contains 'r' in its name
select(movies_df, ends_with("t")) # All vars ending with "t"
```

Select()

- Usage:

```
select(data, columns)
```



```
movies_df <- tbl_df(movies)
select(movies_df, title, year, rating) #Just the columns we want to see
select(movies_df, -c(r1:r10)) #we don't want certain columns

#You can also select a range of columns from start:end
select(movies_df, title:votes)   # All the columns from title to votes
select(movies_df, -c(budget, r1:r10, Animation, Documentary, Short, Romance))

select(movies_df, contains("r"))  # Any column that contains 'r' in its name
select(movies_df, ends_with("t")) # All vars ending with "t"

select(movies_df, starts_with("r"))  # Gets all vars staring with "r"
#The above is not quite what we want. We don't want the Romance column
select(movies_df, matches("r[0-9]"))  # Columns that match a regex.
```

Arrange()

Usage: `arrange(data, column_to_sort_by)`

- Returns a reordered set of rows
- Multiple inputs are arranged from left-to-right

```
movies_df <- tbl_df(movies)
arrange(movies_df, rating) #but this is not what we want
arrange(movies_df, desc(rating))
```

Arrange()

Usage: `arrange(data, column_to_sort_by)`

- Returns a reordered set of rows
- Multiple inputs are arranged from left-to-right

```
movies_df <- tbl_df(movies)
arrange(movies_df, rating) #but this is not what we want
arrange(movies_df, desc(rating))
#Show the highest ratings first and the latest year…
#Sort by Decreasing Rating and Year
arrange(movies_df, desc(rating), desc(year))
```

Arrange()

Usage: `arrange(data, column_to_sort_by)`

- Returns a reordered set of rows
- Multiple inputs are arranged from left-to-right

```
movies_df <- tbl_df(movies)
arrange(movies_df, rating) #but this is not what we want
arrange(movies_df, desc(rating))
#Show the highest ratings first and the latest year…
#Sort by Decreasing Rating and Year
arrange(movies_df, desc(rating), desc(year))
```

What's the difference between these two?

```
arrange(movies_df, desc(rating), desc(year))
arrange(movies_df, desc(year), desc(rating))
```

Mutate()

- Usage:

```
mutate(data, new_col = func(oldcolumns)
```

- Creates new columns, that are functions of existing variables

```
movies_with_budgets <- filter(movies_df, !is.na(budget))
mutate(movies_with_budgets, costPerMinute = budget/length) %>%
  select(title, costPerMinute)
```

Group_by() and Summarize()

```
group_by(data, column_to_group) %>%
  summarize(function_of_variable)
```

- Group_by creates groups of data
- Summarize aggregates the data for each group

```
by_rating <- group_by(movies_df, rating)

by_rating %>% summarize(n())
```

Group_by() and Summarize()

```
group_by(data, column_to_group) %>%
  summarize(function_of_variable)
```

- Group_by creates groups of data
- Summarize aggregates the data for each group

```
by_rating <- group_by(movies_df, rating)

by_rating %>% summarize(n())

avg_rating_by_year <-
   group_by(movies_df, year) %>%
   summarize(avg_rating = mean(rating))
```

Chain the “Verbs” Together

- Chain them together

```
producers_nightmare <-
  filter(movies_df, !is.na(budget)) %>%
  mutate(costPerMinute = budget/length) %>%
  arrange(desc(costPerMinute)) %>%
  select(title, costPerMinute)
```

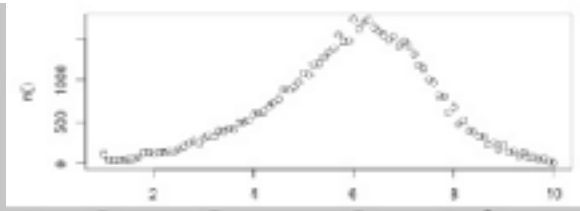
Chain the "Verbs" Together

- Chain them together

```
producers_nightmare <-
  filter(movies_df, !is.na(budget)) %>%
  mutate(costPerMinute = budget/length) %>%
  arrange(desc(costPerMinute)) %>%
  select(title, costPerMinute)
```

- Can also be fed to a "plot" command

```
movies %>%
  group_by(rating) %>%
  summarize(n()) %>%
  plot() # plots the histogram of movies by Each value of rating
```



Practice

- Find all the post-2000 comedy movies with budget of over $1,000,000, rank them by rating in the decreasing order, and output their title and rating

Practice

- Find all the post-2000 comedy movies with budget of over $1,000,000, rank them by rating in the decreasing order, and output their title and rating

- **comedies <- filter(movies_df, year > 2000, Comedy = 1, budget > 1000000) %>%**

Practice

- Find all the post-2000 comedy movies with budget of over $1,000,000, rank them by rating in the decreasing order, and output their title and rating

- **comedies <- filter(movies_df, year > 2000, Comedy = 1, budget > 1000000) %>%**
- **arrange(desc(rating)) %>%**
- **select(title, rating)**

Visualization Tools

Dr. Ke Zhou (http://www.cs.nott.ac.uk/~pszkz/)

Visualization Tools

Chart Typologies
Excel, Many Eyes, Google Charts

Visual Analysis Grammars
VizQL, ggplot2

Visualization Grammars
Protovis, D3.js

Component Architectures
Prefuse, Flare, Improvise, VTK

Graphics APIs
Processing, OpenGL, Java2D

Visualization Tools

Chart Typologies
Excel, Many Eyes, Google Charts

Charting Tools

Visual Analysis Grammars
VizQL, ggplot2

Visualization Grammars
Protovis, D3.js

Declarative Languages

Component Architectures
Prefuse, Flare, Improvise, VTK

Graphics APIs
Processing, OpenGL, Java2D

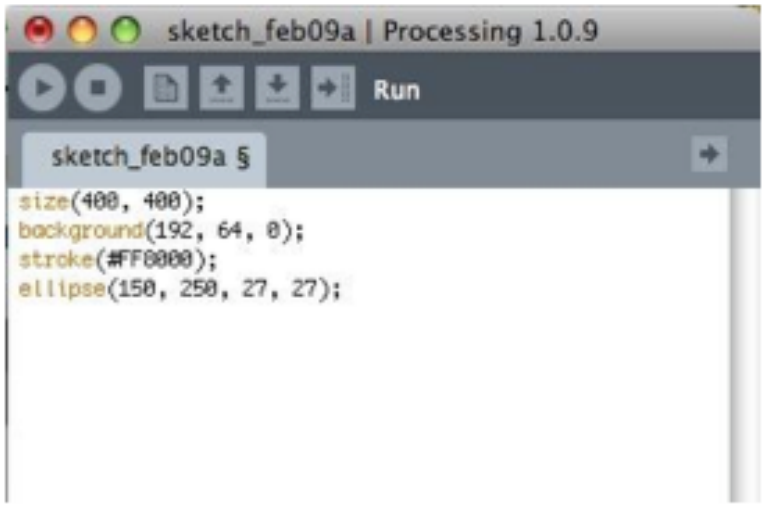
Programming Toolkits

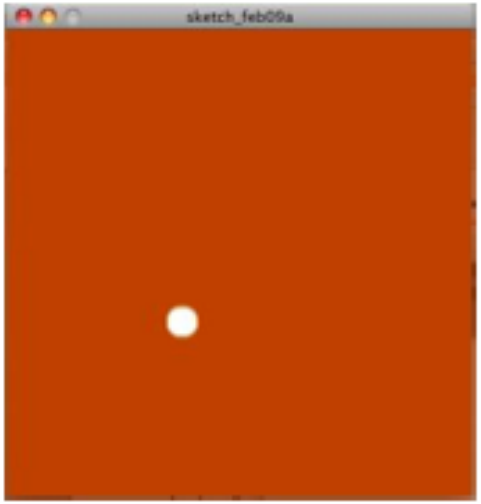
Chart Typology (Charting Tools)

- Pick from a stock of templates
- Easy-to-use but limited expressiveness
- Prohibits novel designs, new data types

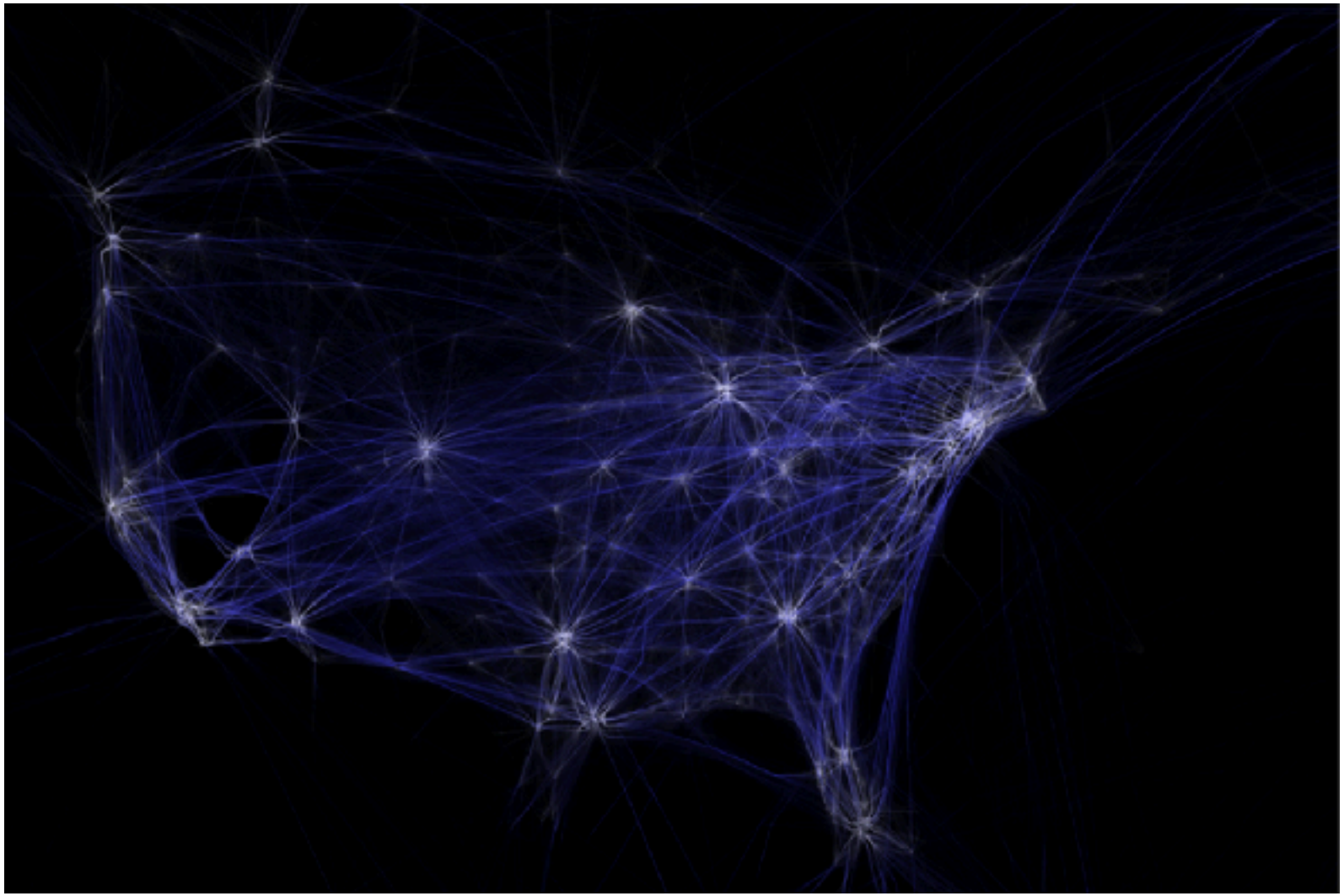
Graphics APIs (Programming Toolkits)

- Processing.org
  - Java based
  - not specifically designed for InfoVis
  - Well documented, lots of tutorials (even books)





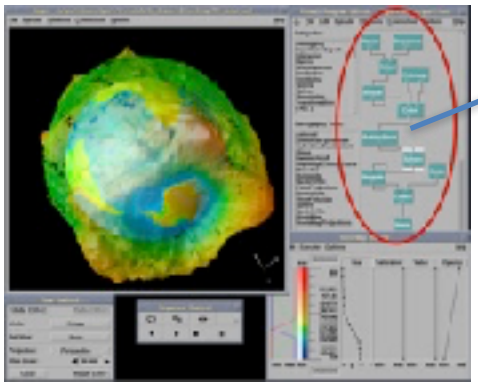
Graphics APIs can be very powerful

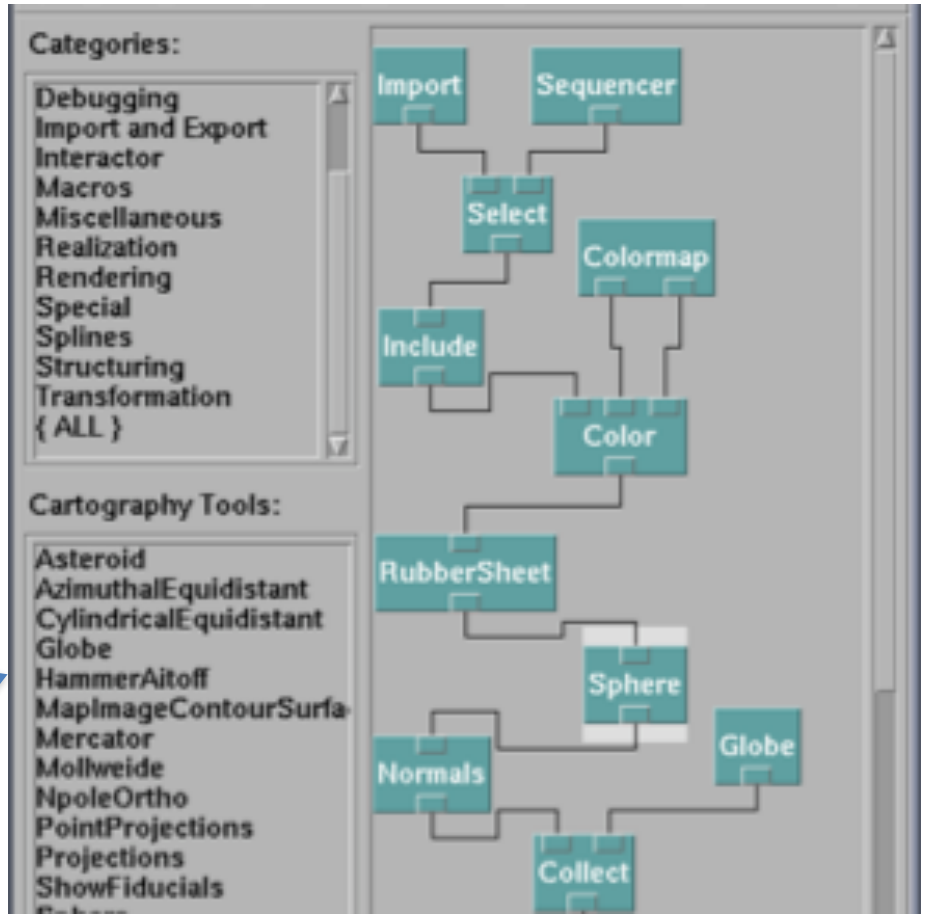


US Air Traffic Visualization Dr. Ke Zhou (http://www.cs.nott.ac.uk/~pszkz/)

Component Architectures (Programming Toolkits)

- Permits more combinatorial possibilities
- Novel views require new operators, which requires software engineering.

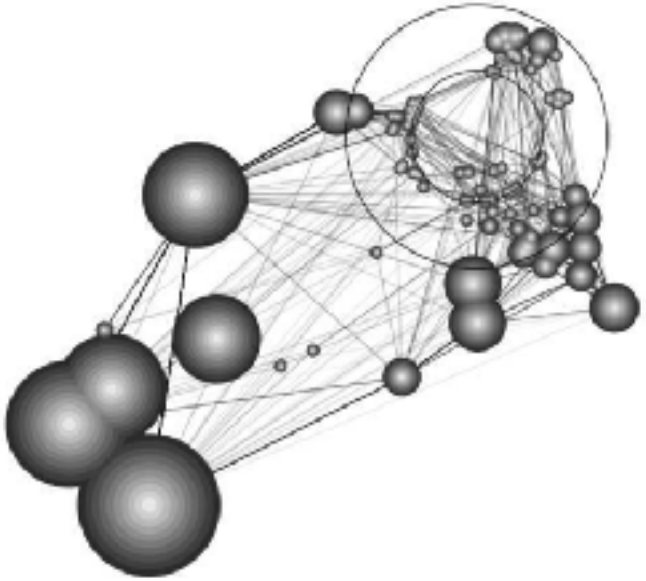




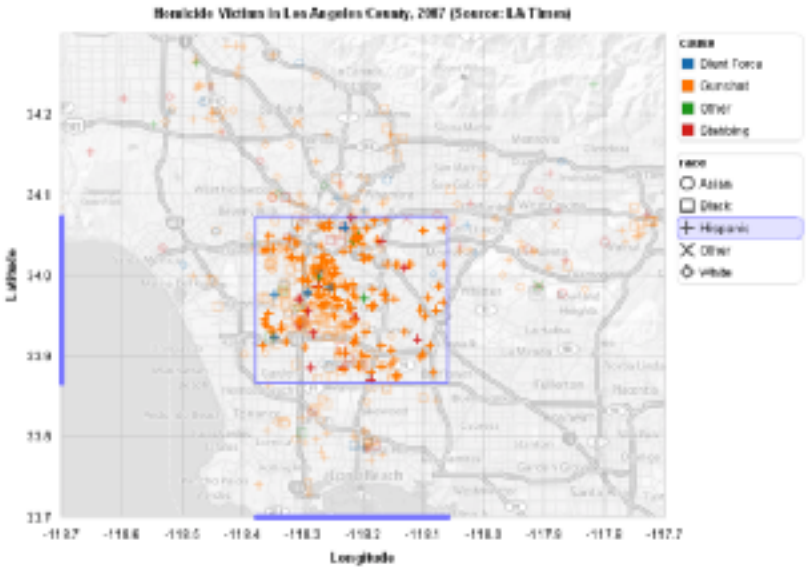
Dr. Ke Zhou (http://www.cs.nott.ac.uk/~pszkz/)

Prefuse & Flare

- Operator-based toolkits for visualization design Vis = (Input Data -> Visual Objects) + Operators



Prefuse (http://prefuse.org)



Flare (http://flare.prefuse.org)

Comparison

- **Chart Typology**
  - Pick from a stock of templates
  - Easy-to-use but limited expressiveness
  - Prohibits novel designs, new data types
- **Component Architecture**
  - Permits more combinatorial possibilities
  - Novel views require new operators, which requires software engineering.

The Grammar of Graphics (Declarative Languages)

- Programming by describing what, not how
- Separate specification (what you want) from execution (how it should be computed)
- In contrast to imperative programming, where you must give explicit steps.

```
d3.selectAll("rect")
  .data(my_data)
 .enter().append("rect")
  .attr("x", function(d) { return xscale(d.foo); })
  .attr("y", function(d) { return yscale(d.bar); })
```

Building a Plot in ggplot2

data to visualize (a data frame)

map variables to ***aes***thetic attributes

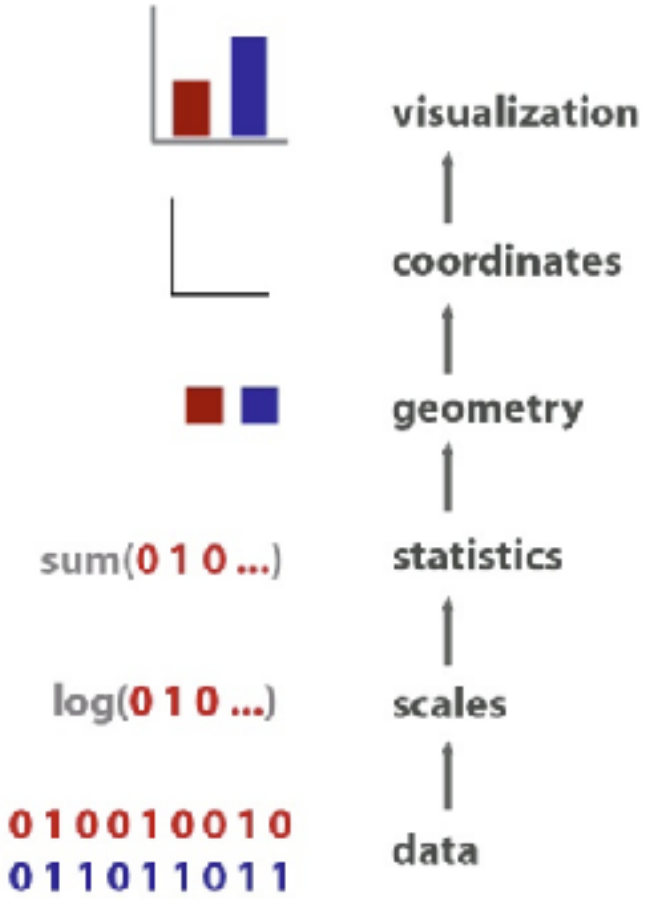
geometric objects – what you see (points, bars, etc)

scales map values from data to aesthetic space

faceting subsets the data to show multiple plots

statistical transformations – summarize data

coordinate systems put data on plane of graphic



Wickham 2009

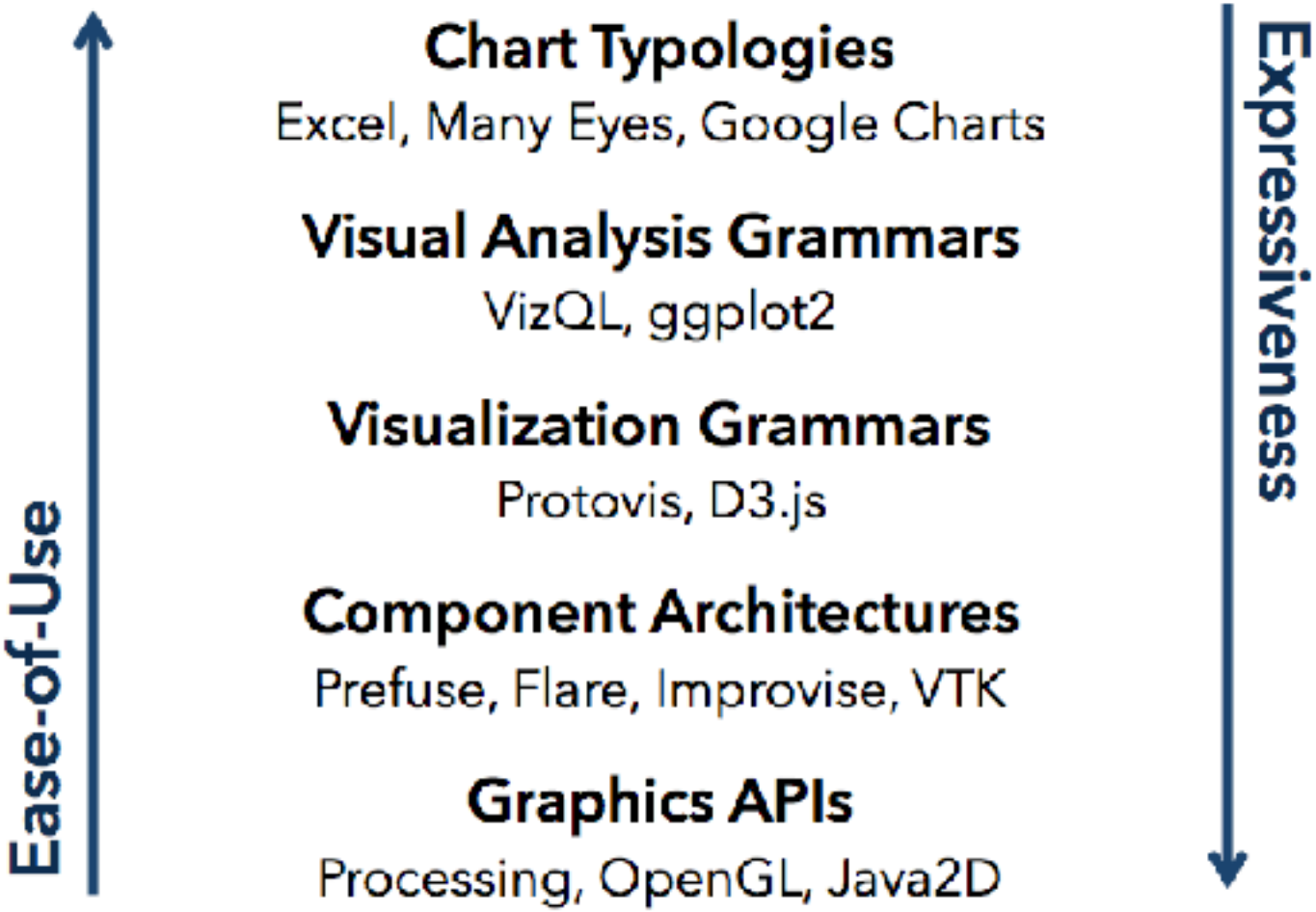
The Advantages of Declarative Languages

- **Faster iteration**. Less code. Larger user base.
- **Better visualization**. Smart defaults.
- **Reuse**. Write-once, then re-apply.
- **Performance**. Optimization, scalability.
- **Portability**. Multiple devices, renderers, inputs.
- **Programmatic generation**. Write programs which output visualizations. Automated search & recommendation.

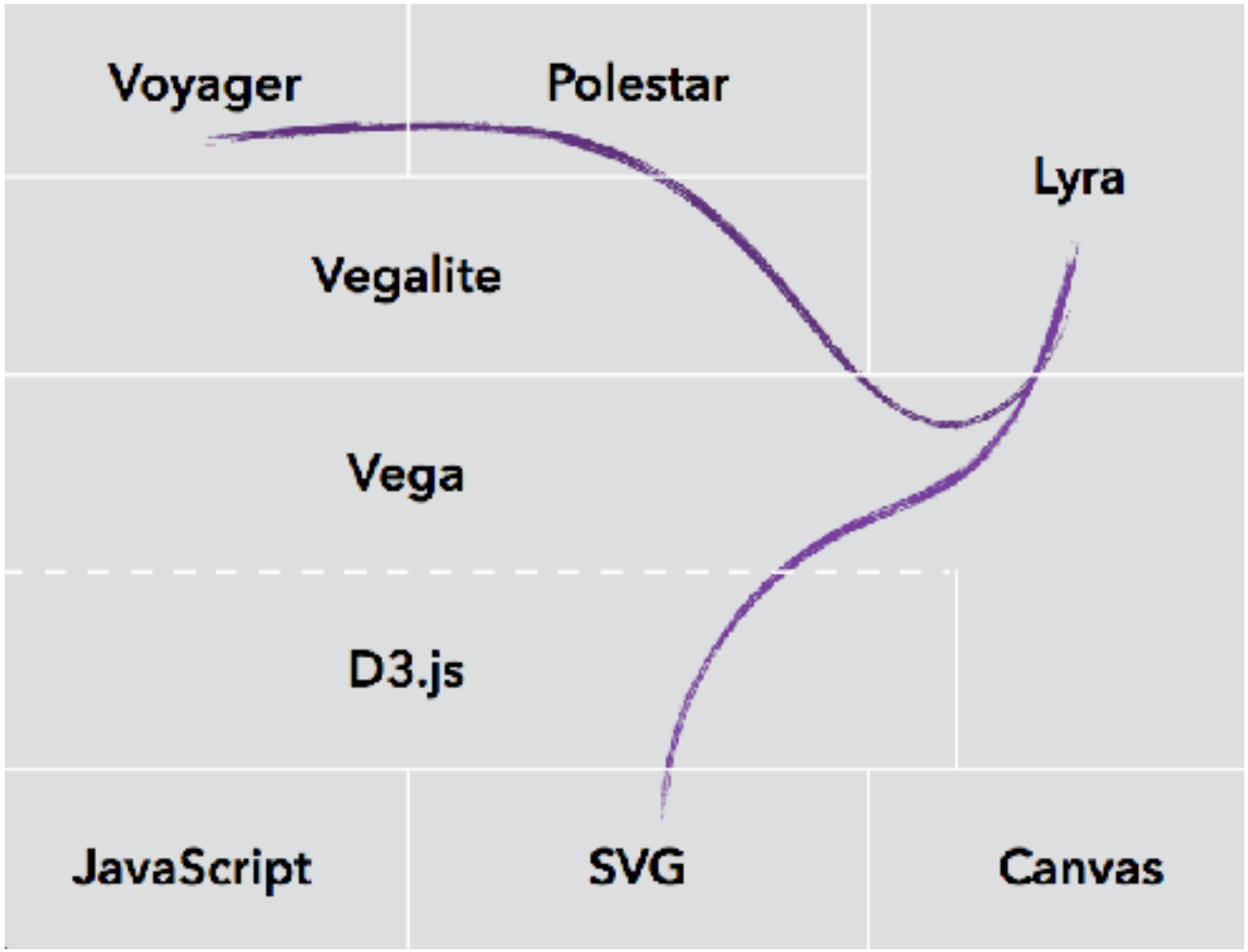
Tools Tradeoffs

- InfoVis-focused
  - Many fundamental techniques built-in
  - Can be faster to get something going
  - Often more difficult to implement something “different”
  - Documentation?
- Generic Graphics
  - More flexible
  - Can customize better
  - Big learning curve
  - Doc is often better
  - Can take a long time to (re)implement basic techniques

Visualization Tools



Many Tools Developed by Prof. Jeffrey Heer, University of Washington



This is just a reference point.
You should try those information visualization tools out
(optional for those who don't take G53IVP)!

(Optional) Resources

- D3 tutorial: https://uwdata.github.io/d3-tutorials/
- Vega tutorial: https://github.com/vega/vega/wiki/Tutorial
- Please start working on the course work using R.

Next Lecture

- Topic:
  - Visual Perception