# Visualization (Data Transformation)

AUTHOR
Peter Ganong and Maggie Shi

PUBLISHED
October 9, 2024

# introduction

## roadmap

- putting this lecture in context
- `movies` dataset
  - load data
  - `shape`
  - `head()`

(no summary at end of this section)

## putting this lecture in context

- Fundamental problem in data visualization: in most cases, you do not want to show every single data point in your dataset.
- Instead, you want to extract patterns which you (the analyst) think are interesting.
- This lecture explores methods for *transforming* data, focusing on aggregation.
- One nice thing about Altair is that it nudges you to aggregate.
  - One example: if you try to make a plot with 10,000 dots, it will give you an error `MaxRowsError: The number of rows in your dataset is greater than the maximum allowed (5000).`
  - Help file: "This is not because Altair cannot handle larger datasets, but it is because it is important for the user to think carefully about how large datasets are handled."
  - More details here
- This lecture mostly follows Chapter 3 in the data visualization book (skip parts of section 3.2, which we will then come back to in lectures 4 and 5)

## load packages

```
import pandas as pd
import altair as alt
```

## movies dataset

```
movies_url = 'https://cdn.jsdelivr.net/npm/vega-datasets@1/data/movies.json'
```

```
movies = pd.read_json(movies_url)
```

```
movies.shape
```

```
(3201, 16)
```

With 3201 movies, we are going to need to do some transformation if we want to uncover any patterns in the data!

### head()

```
movies.head(5)
```

| | Title | US Gross | Worldwide Gross | US DVD Sales | Production Budget | Release Date | MPAA Rating | Running Time min | Distributor | Source | Major Genre | Crea Typ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | The Land Girls | 146083.0 | 146083.0 | NaN | 8000000.0 | Jun 12 1998 | R | NaN | Gramercy | None | None | Non |
| 1 | First Love, Last Rites | 10876.0 | 10876.0 | NaN | 300000.0 | Aug 07 1998 | R | NaN | Strand | None | Drama | Non |
| 2 | I Married a Strange Person | 203134.0 | 203134.0 | NaN | 250000.0 | Aug 28 1998 | None | NaN | Lionsgate | None | Comedy | Non |
| 3 | Let's Talk About Sex | 373615.0 | 373615.0 | NaN | 300000.0 | Sep 11 1998 | None | NaN | Fine Line | None | Comedy | Non |
| 4 | Slam | 1009819.0 | 1087521.0 | NaN | 1000000.0 | Oct 09 1998 | R | NaN | Trimark | Original Screenplay | Drama | Con Ficti |

# Scatter plots and binning

## Scatter plots and binning: roadmap

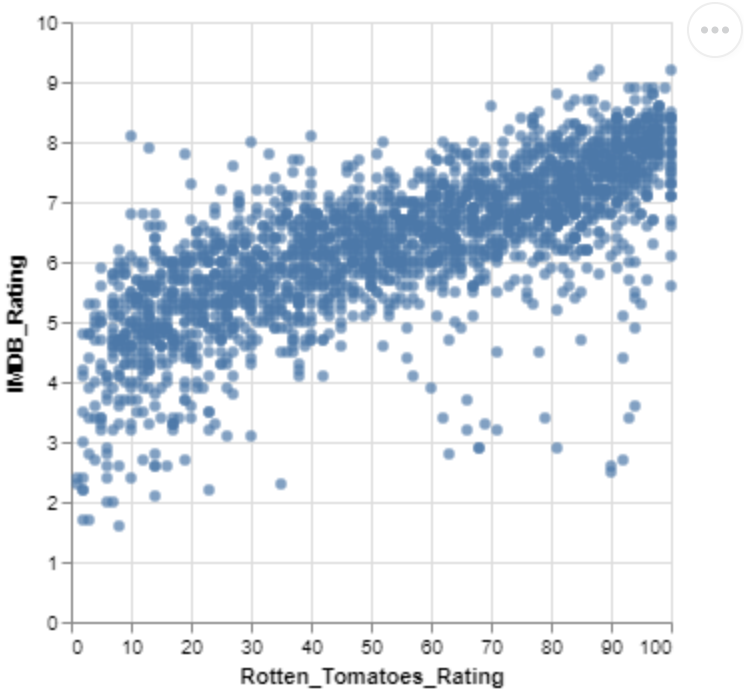
- scatter plots

- binning

## scatter plot

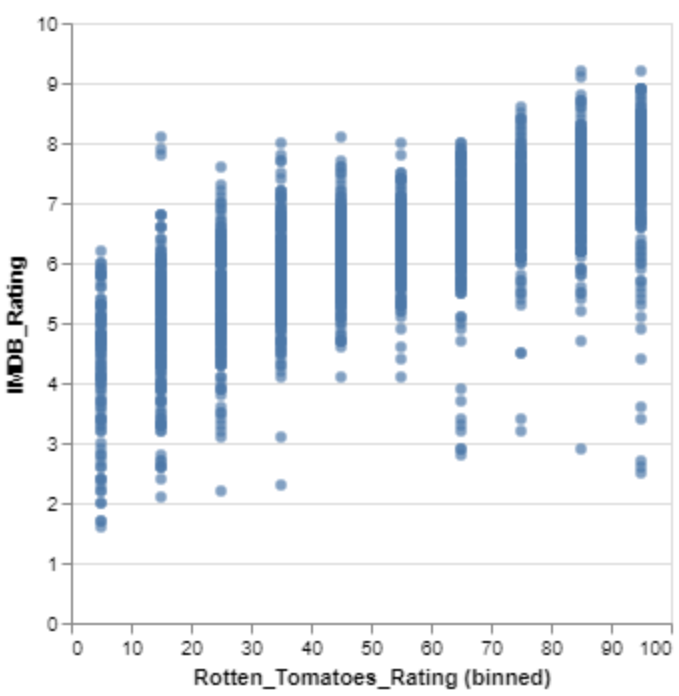
- **Rotten Tomatoes** ratings are determined by taking “thumbs up” and “thumbs down” judgments from film critics and calculating the percentage of positive reviews.
- **IMDB ratings** are formed by averaging scores (ranging from 1 to 10) provided by the site’s users.

```
alt.Chart(movies_url).mark_circle().encode(
    alt.X('Rotten_Tomatoes_Rating:Q'),
    alt.Y('IMDB_Rating:Q')
)
```



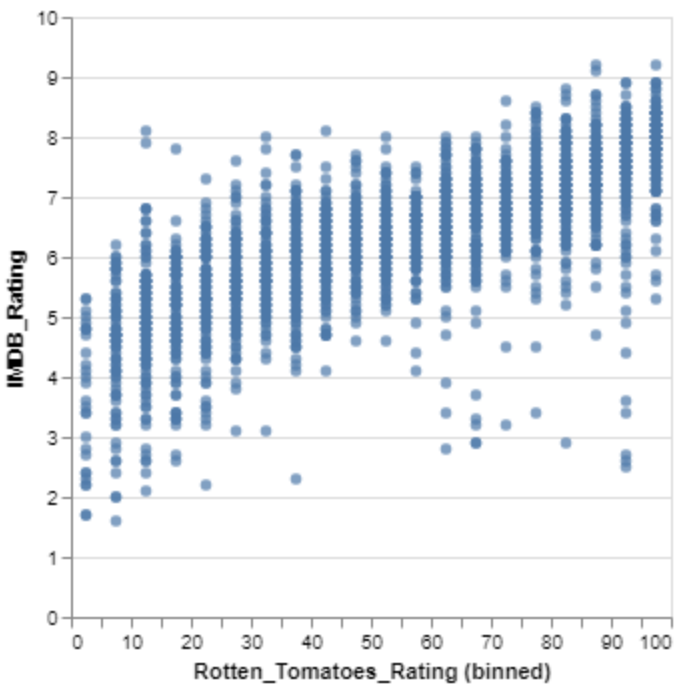

## scatter plot – add `bin=True`

```
alt.Chart(movies_url).mark_circle().encode(
    alt.X('Rotten_Tomatoes_Rating:Q', bin=True),
    alt.Y('IMDB_Rating:Q')
)
```




## scatter plot – 20 bins

```
alt.Chart(movies_url).mark_circle().encode(
    alt.X('Rotten_Tomatoes_Rating:Q', bin=alt.BinParams(maxbins=20)),
    alt.Y('IMDB_Rating:Q')
)
```




# Aggregation

# Aggregation: roadmap

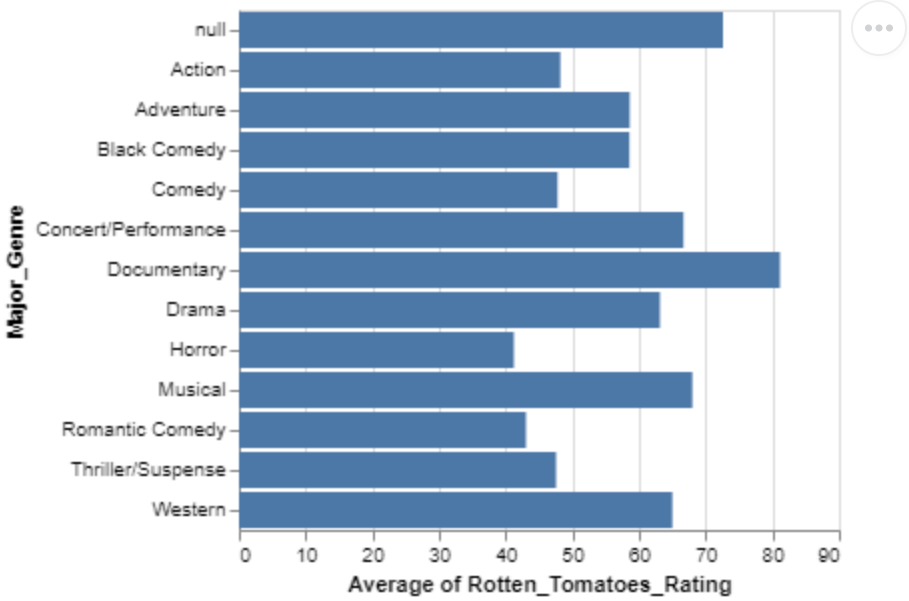
In previous lectures, we actually already saw aggregation via `average()` and `min()`. We just didn't talk explicitly about that step. Now, we examine it more carefully.

- `average()`
- interquartile range
- do-pair-share

The Altair documentation includes the full set of available aggregation functions.

## `average()`

```
alt.Chart(movies_url).mark_bar().encode(
    alt.X('average(Rotten_Tomatoes_Rating):Q'),
    alt.Y('Major_Genre:N')
)
```




This plot is fine, but hard to interpret takeaways quickly.

Discussion Question: what does the y-axis seem to be sorted on? Why?

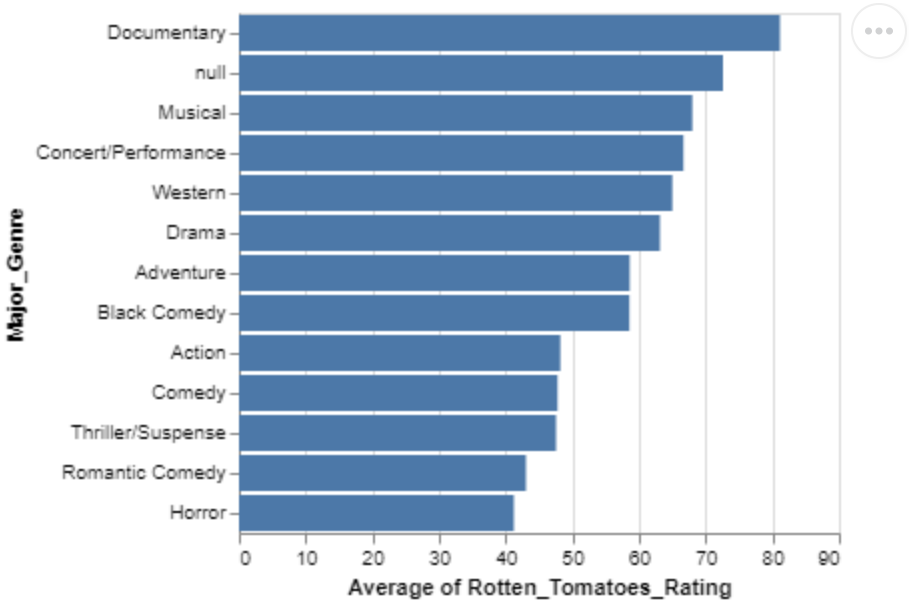
What should we do? Sort the bars vertically, but by what's on the x-axis.

But we haven't mixed X and Y encodings together, yet. What's the best way to figure out how? Ask ChatGPT.

## `average()` with `sort(...)`

Now it's clear which movie types are most and least popular

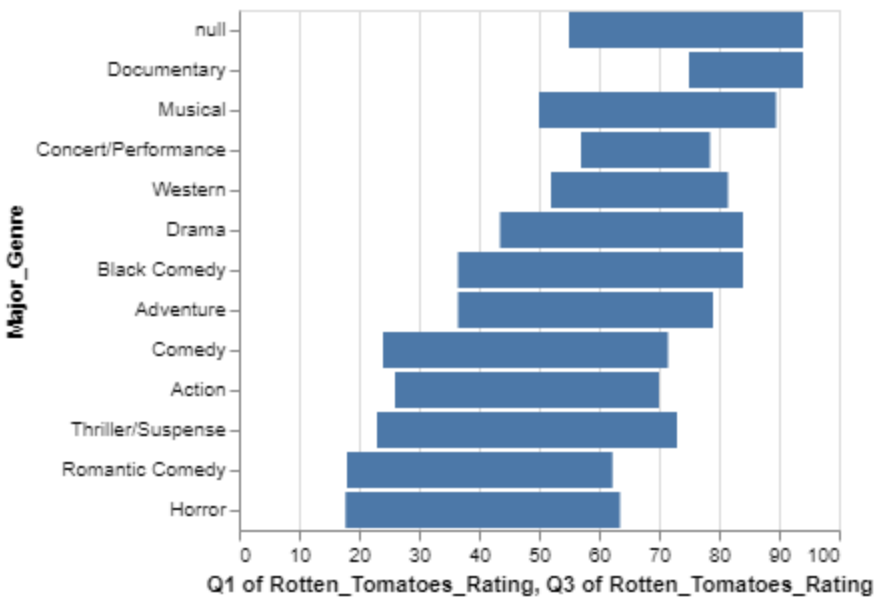
```
#Attribution: ChatGPT
#Query: I have the following bar chart code in Altair [...]
# I want to sort the bars by the X encoding (average rotten tomatoes rating). How can I do that?
alt.Chart(movies_url).mark_bar().encode(
    alt.X('average(Rotten_Tomatoes_Rating):Q'),
    alt.Y('Major_Genre:N',
        sort=alt.EncodingSortField(
            op='average',
            field='Rotten_Tomatoes_Rating',
            order='descending'
        )
    )
)
```




Discussion question – Why is "how to sorting the order of bars" such a great problem to submit to ChatGPT?

## Interquartile range

```
alt.Chart(movies_url).mark_bar().encode(
    alt.X('q1(Rotten_Tomatoes_Rating):Q'),
    alt.X2('q3(Rotten_Tomatoes_Rating):Q'),
    alt.Y('Major_Genre:N', sort=alt.EncodingSortField(
        op='median', field='Rotten_Tomatoes_Rating', order='descending')
    )
)
```




Discussion question – What can you learn from the interquartile range plot that you could not learn from the plot with just `average()`?

## Case study: when are the highest grossing films?

```
movies_gross = movies[['US Gross', 'Release Date']]
movies_gross.head()
```

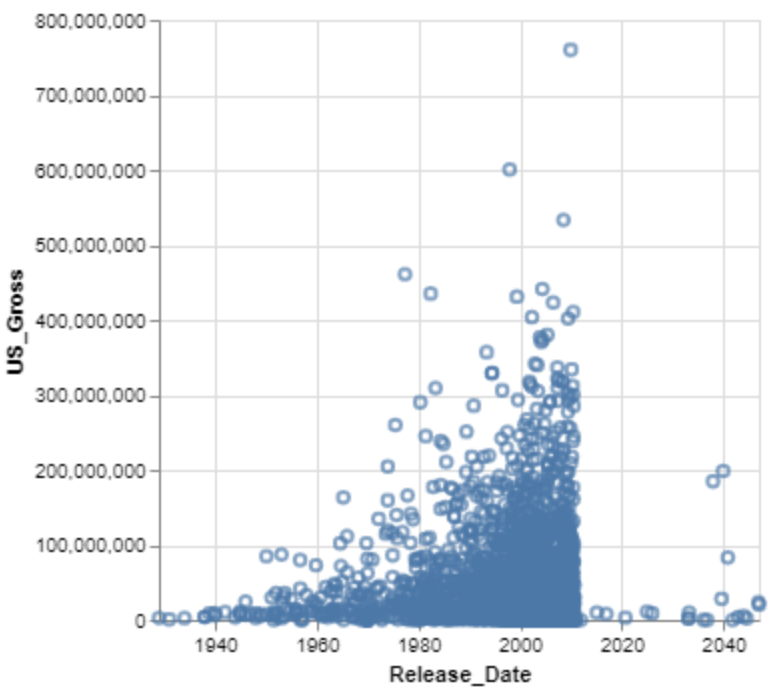
| | US Gross | Release Date |
|---|---|---|
| 0 | 146083.0 | Jun 12 1998 |
| 1 | 10876.0 | Aug 07 1998 |
| 2 | 203134.0 | Aug 28 1998 |
| 3 | 373615.0 | Sep 11 1998 |
| 4 | 1009819.0 | Oct 09 1998 |

## a first pass

obviously we need to aggregate.

also: what bug in the data does this plot reveal?

```
alt.Chart(movies_url).mark_point().encode(
    alt.X('Release_Date:T'),
    alt.Y('US_Gross:Q')
)
```




## do-pair-share

What time of year are the highest grossing films released? Aggregate both the x- and the y-variables.

Hint: if you aren't sure how to aggregate the x-variable, go back to the first visualization lecture as an example.

1. *Do* – make a plot on your own
2. *Pair* – compare your results with person next to you
3. *Share* – discuss results as a class

## Aggregation: summary

- Quantitative beyond `count()` and `average()`
  - Distribution: `min()`, `q1()`, `median()`, `q3()`, `max()`
  - Dispersion: `variance()`, `stdev()`, `distinct()`
  - Bootstrap confidence intervals: `ci0()`, `ci1()`
- Dates: see prior slide

# Advanced data transformation

## Advanced data transformation: roadmap

- Two ways to aggregate data in Altair: within the encoding itself, or using a top level aggregate transform.
- Doing it in the encoding is fine for simple transformations, but for advanced transformations, we'll have to define it separately
- `transform_calculate()`
- `transform_filter()`
- do-pair-share
- `transform_aggregate()`
- `transform_window()`

These are all written in the Vega expression language.

## Advanced data transformation: connection to packages you might already know

| Purpose | Vega | `pandas` equivalent |
|---|---|---|
| Define a new variable | `transform_calculate()` | `df['new_col']` |
| Filter to subset of rows | `transform_filter(cond)` | `df.loc[cond]` |
| Aggregate function - reduces number of rows down to one per group | `transform_aggregate(groupby(...))` | `df.groupby('A').agg('mean')` |
| Window function - transform across multiple rows, keeps same num. of rows) | `transform_window(sum())` | `df['values'].cumsum()` |

One way to think of these verbs is that they are fundamental to any data analysis project and so in any/every language you learn, you need to know how to do these.

## connection to prior material

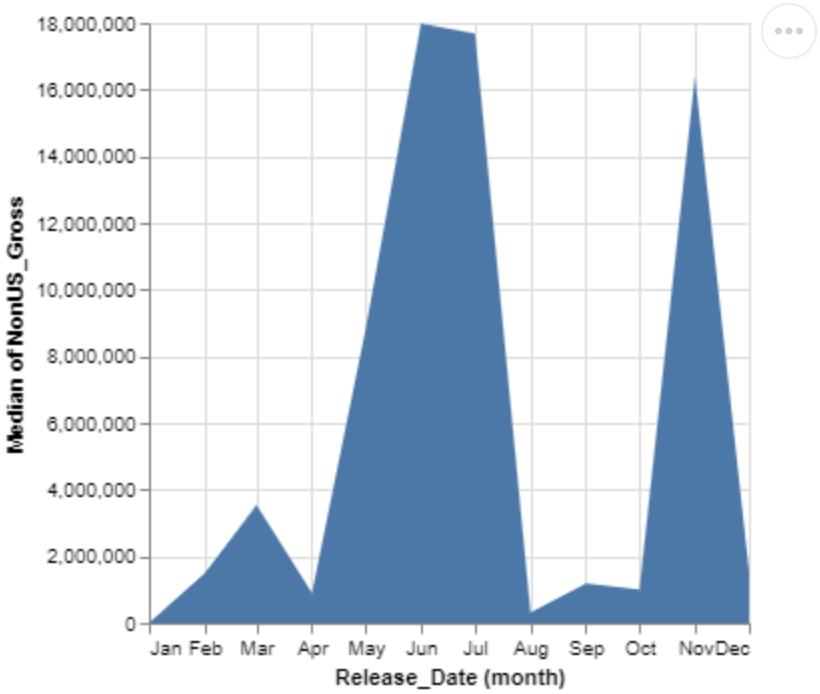
You already know how to do these all in `pandas` and in `dplyr` so it is not conceptually new.

Why bother doing it in Altair/Vega?

**Exploratory data analysis** can be done faster in Altair: manipulate data and plot simulataneously

## `transform_calculate` case study redux: what time of year do US movies make money abroad?

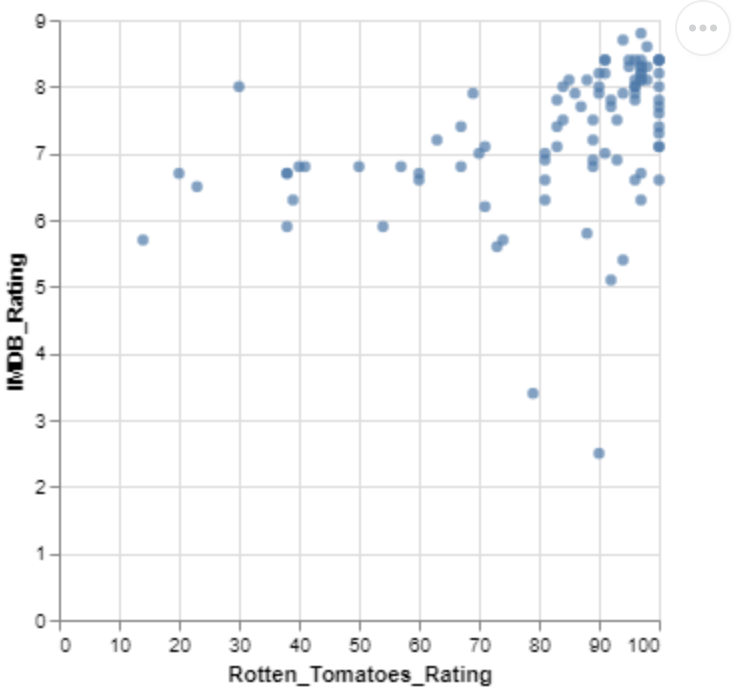
```
alt.Chart(movies_url).mark_area().transform_calculate(
    NonUS_Gross='datum.Worldwide_Gross - datum.US_Gross'
).encode(
    alt.X('month(Release_Date):T'),
    alt.Y('median(NonUS_Gross):Q')
)
```




- `datum` is how you reference the underlying dataset within a transformation expression
- `transform_calculate()` uses expressions for writing basic formulas
  - Math functions: `min()`, `random()`, `round()`
  - Statistical functions: `sampleNormal()`, `sampleUniform()`
  - Date-time functions: `date()`, `year()`, `month()`
  - String functions: `length()`, `lower()`, `substring()`
  - Full list here

## `transform_filter` show just movies before 1970

```
alt.Chart(movies_url).mark_circle().encode(
    alt.X('Rotten_Tomatoes_Rating:Q'),
    alt.Y('IMDB_Rating:Q')
).transform_filter('year(datum.Release_Date) < 1970')
```

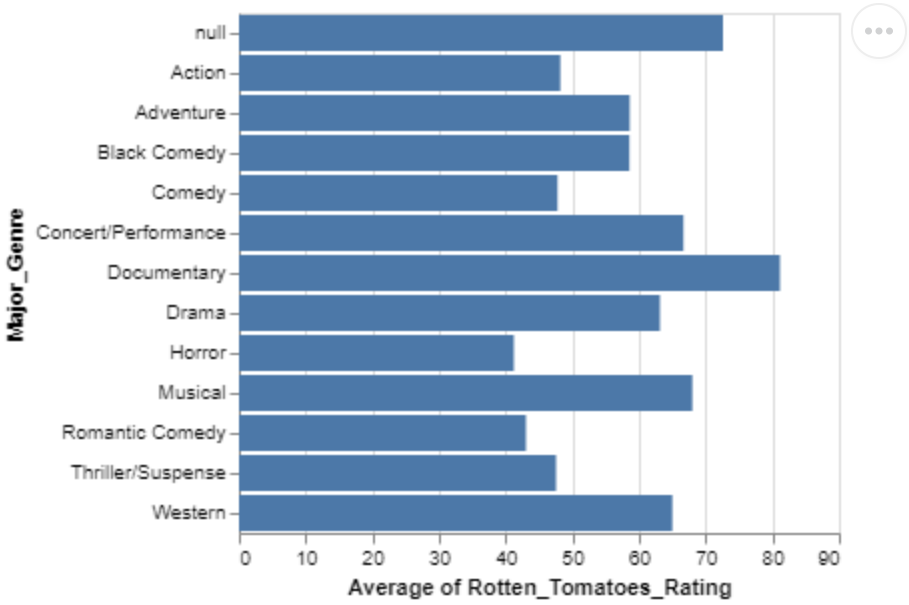



## Do-pair-share

- Make two plots that compare ratings before and after 1970
  - Plot before and after 1970 on one plot
    - Create a categorical variable to indicate whether an observation is from before or after 1970.
    - Encode the color of the mark depending on the value of that categorical variable
  - Append together two plots side by side: a scatter plot of ratings before 1970 and after 1970
- Which do you prefer and why?

## `transform_aggregate` recap from earlier in lecture

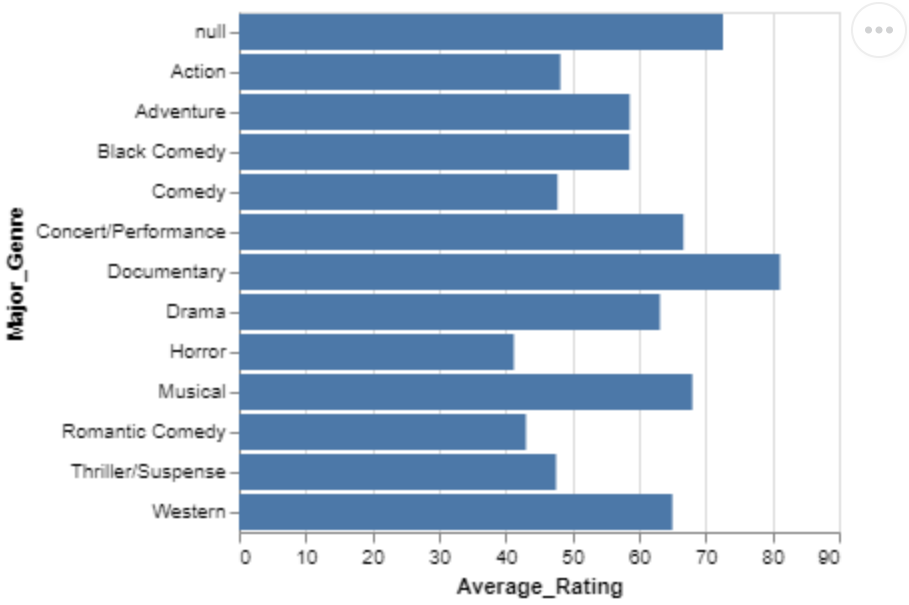
```
alt.Chart(movies_url).mark_bar().encode(
    alt.X('average(Rotten_Tomatoes_Rating):Q'),
    alt.Y('Major_Genre:N')
)
```




## `transform_aggregate` what's happening under the hood

The previous example did the aggregation directly in the encoding. We can also express this in a more verbose way, with `transform_aggregate()`

```
alt.Chart(movies_url).mark_bar().transform_aggregate(
    groupby=['Major_Genre'],
    Average_Rating='average(Rotten_Tomatoes_Rating)'
).encode(
    alt.X('Average_Rating:Q'),
    alt.Y('Major_Genre:N')
)
```

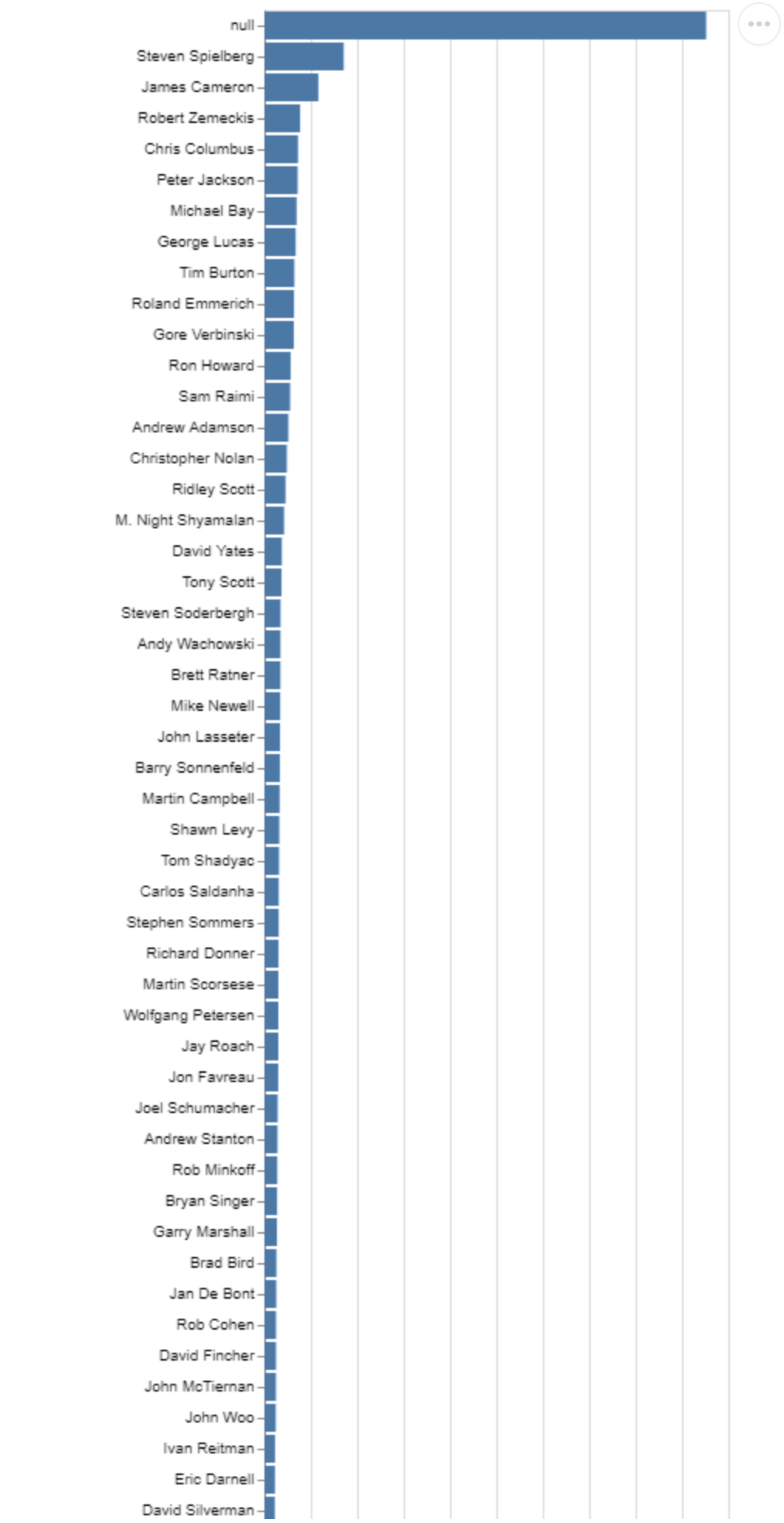


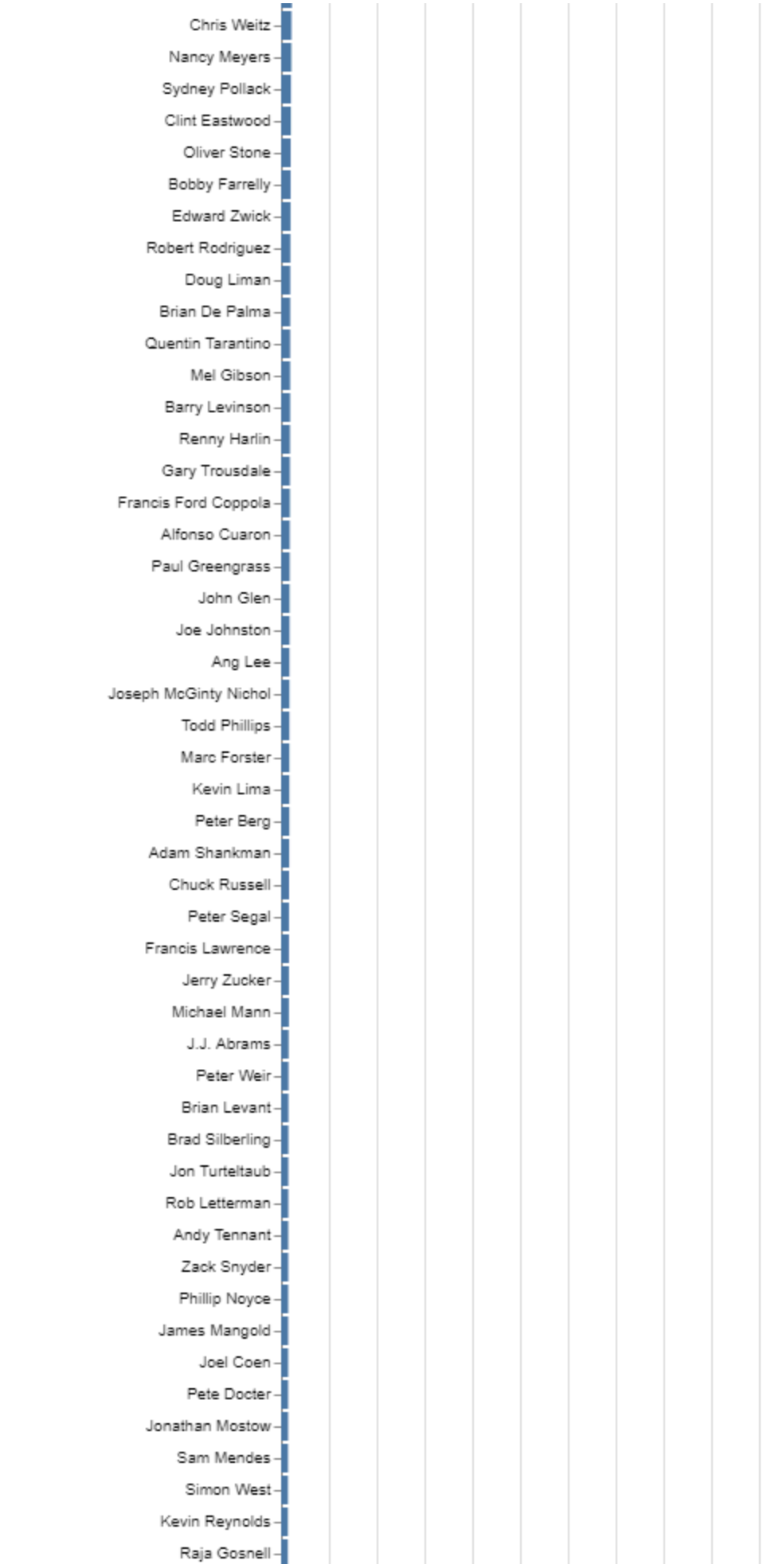

Discussion question – If prior two code blocks have identical output, which version is better (and why)?

## `transform_window`: case study: who are the top 20 grossing directors of all time?

Start by summing `Worldwide_Gross` for each director, and then plotting in descending order.

```
alt.Chart(movies_url).mark_bar().transform_aggregate(
    Gross='sum(Worldwide_Gross)',
    groupby=['Director']
).encode(
    alt.X('Gross:Q'),
    alt.Y('Director:N', sort=alt.EncodingSortField(
        op='max', field='Gross', order='descending'
    ))
)
```


null
Steven Spielberg
James Cameron
Robert Zemeckis
Chris Columbus
Peter Jackson
Michael Bay
George Lucas
Tim Burton
Roland Emmerich
Gore Verbinski
Ron Howard
Sam Raimi
Andrew Adamson
Christopher Nolan
Ridley Scott
M. Night Shyamalan
David Yates
Tony Scott
Steven Soderbergh
Andy Wachowski
Brett Ratner
Mike Newell
John Lasseter
Barry Sonnenfeld
Martin Campbell
Shawn Levy
Tom Shadyac
Carlos Saldanha
Stephen Sommers
Richard Donner
Martin Scorsese
Wolfgang Petersen
Jay Roach
Jon Favreau
Joel Schumacher
Andrew Stanton
Rob Minkoff
Bryan Singer
Garry Marshall
Brad Bird
Jan De Bont
Rob Cohen
David Fincher
John McTiernan
John Woo
Ivan Reitman
Eric Darnell
David Silverman


Chris Weitz
Nancy Meyers
Sydney Pollack
Clint Eastwood
Oliver Stone
Bobby Farrelly
Edward Zwick
Robert Rodriguez
Doug Liman
Brian De Palma
Quentin Tarantino
Mel Gibson
Barry Levinson
Renny Harlin
Gary Trousdale
Francis Ford Coppola
Alfonso Cuaron
Paul Greengrass
John Glen
Joe Johnston
Ang Lee
Joseph McGinty Nichol
Todd Phillips
Marc Forster
Kevin Lima
Peter Berg
Adam Shankman
Chuck Russell
Peter Segal
Francis Lawrence
Jerry Zucker
Michael Mann
J.J. Abrams
Peter Weir
Brian Levant
Brad Silberling
Jon Turteltaub
Rob Letterman
Andy Tennant
Zack Snyder
Phillip Noyce
James Mangold
Joel Coen
Pete Docter
Jonathan Mostow
Sam Mendes
Simon West
Kevin Reynolds
Raja Gosnell


Tim Story
David Slade
Robert Luketic
Paul Anderson
Keenen Ivory Wayans
Paul Verhoeven
Adrian Lyne
Chris Wedge
George Miller
Guy Ritchie
Wes Craven
Jonathan Demme
Lee Tamahori
Steve Carr
Len Wiseman
William Friedkin
Andrew Davis
Cameron Crowe
Richard Marquand
Alex Proyas
Mark Waters
Antoine Fuqua
Paul Weitz
John Singleton
Danny Boyle
Dennis Dugan
F. Gary Gray
Baz Luhrmann
Donald Petrie
Peyton Reed
Nora Ephron
Sylvester Stallone
Harold Ramis
Kevin Costner
Guy Hamilton
Steve Hickner
Frank Oz
Frank Coraci
Guillermo Del Toro
Curtis Hanson
Mark Dindal
Rob Reiner
Anthony Minghella
Gabriele Muccino
Rob Marshall
Robert Wise
Judd Apatow
George Cukor
Catherine Hardwicke


Darren Lynn Bousman
Martin Brest
Mimi Leder
Roger Spottiswoode
David Zucker
Woody Allen
Luc Besson
Roger Donaldson
Mike Nichols
Jason Reitman
Mark Steven Johnson
Spike Lee
Anne Fletcher
Gus Van Sant
Hayao Miyazaki
Michael Apted
Joel Zwick
Nick Park
Frank Darabont
Jean-Pierre Jeunet
Gavin Hood
Louis Leterrier
Tim Hill
Terry Gilliam
Betty Thomas
Leonard Nimoy
David R. Ellis
P.J. Hogan
Peter Hyams
Timur Bekmambetov
James L. Brooks
Frank Marshall
George P. Cosmatos
Ben Stiller
Neil Jordan
John Madden
Yimou Zhang
Taylor Hackford
Steve Oedekerk
David Dobkin
Mel Brooks
Penny Marshall
John Landis
Michael Moore
Jonathan Frakes
Joe Dante
Jon Amiel
Les Mayfield
David Frankel


D.J. Caruso
Stephen Daldry
Lasse Hallstrom
John Pasquin
Stephen Hopkins
Stephen Norrington
Stephen Frears
Roman Polanski
Peter Hewitt
Hugh Wilson
George Roy Hill
Nick Cassavetes
Jason Friedberg
Jean-Jacques Annaud
Joe Wright
Stanley Kubrick
Bruce Beresford
Andrzej Bartkowiak
Chris Noonan
Tom Dey
Larry Charles
James Wong
Peter Cattaneo
Sam Fell
John Musker
Don Bluth
Adam McKay
Wayne Wang
Joe Pytka
Michael Caton-Jones
Brian Robbins
Gary Winick
Kevin Smith
Joe Carnahan
Warren Beatty
Randal Kleiser
Steve Barron
Pierre Morel
John Guillermin
Tim Johnson
David Lean
Alexander Payne
Robert Redford
John G. Avildsen
Matthew Vaughn
Alejandro Gonzalez Inarritu
David Goyer
Milos Forman
Neill Blomkamp


Director
Robert Altman
Norman Jewison
Andy Fickman
Peter Chelsom
Richard Linklater
Iain Softley
Tom Tykwer
John Carpenter
Tyler Perry
Alan Parker
Oren Peli
Phil Alden Robinson
Stephen Chow
Russell Mulcahy
James McTeigue
Joseph Ruben
Penelope Spheeris
David Twohy
Michael Lembeck
Matt Reeves
Bill Condon
Tobe Hooper
Peter Billingsley
Wes Anderson
Sofia Coppola
Ronny Yu
Greg Mottola
Seth Gordon
Dominic Sena
Stephen Herek
Jay Russell
Ted Demme
Barbra Streisand
Steve Miner
Spike Jonze
Gil Kenan
Edgar Wright
Gary Ross
Ron Shelton
Carl Franklin
Jared Hess
Eli Roth
Kenneth Branagh
Scott Derrickson
Angela Robinson
Roger Kumble
Alan J. Pakula
David Koepp
Kevin MacDonald

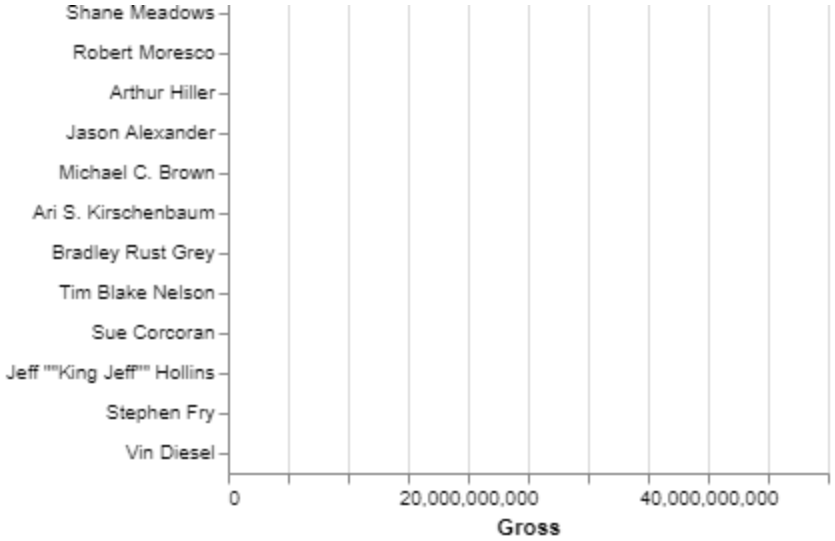

Phil Lord
Lawrence Kasdan
Boaz Yakin
Harold Becker
Malcolm D. Lee
Jay Chandrasekhar
Neil LaBute
Sir Richard Attenborough
George Clooney
Rupert Wainwright
Franklin J. Schaffner
Pedro Almodovar
Ted Kotcheff
Paul Haggis
David O. Russell
John Frankenheimer
Mick Jackson
Paul Thomas Anderson
George A. Romero
Ken Kwapis
Michael Lehmann
Shekhar Kapur
Howard Deutch
Kathryn Bigelow
Jon Avnet
Kelly Asbury
Paul Weiland
Michael Ritchie
Gurinder Chadha
Trey Parker
Peter Webber
Amy Heckerling
David Gordon Green
Bille Woodruff
Christophe Gans
Ruben Fleischer
Alfred Hitchcock
John Schultz
William Wyler
John Schlesinger
Fernando Meirelles
Tony Gilroy
Forest Whitaker
Richard Fleischer
David Cronenberg
Joan Chen
Dennie Gordon
Dwight H. Little
Michel Gondry


Mike Judge
Tim Robbins
Simon Wincer
Sean Penn
Herbert Ross
Mira Nair
Fred Zinnemann
David Lynch
Rob Zombie
Billy Wilder
Andrew Niccol
John Badham
Luke Greenfield
Sidney Lumet
King Vidor
Kevin Bray
Barbet Schroeder
John Huston
Darren Aronofsky
Fred Wolf
Mathieu Kassovitz
James Ivory
Alexandre Aja
Paul McGuigan
Terry Zwigoff
James Gray
Griffin Dunne
Ben Affleck
Grant Heslov
Andrew Fleming
Walter Salles
Steven Zaillian
Hugh Hudson
Lee Daniels
Andrei Konchalovsky
Rick Rosenthal
Terrence Malick
Philip Kaufman
Kevin Greutert
John Dahl
Albert Hughes
Todd Field
Richard LaGravenese
William Malone
Denzel Washington
Michael Cimino
John Milius
John Boorman
Chris Rock


Mike Figgis
Michael Spierig
Christopher Guest
Mike Binder
Shane Acker
Corey Yuen
Oxide Pang Chun
Jim Sheridan
Bennett Miller
Walter Hill
Bob Rafelson
David Bowers
Bernardo Bertolucci
Alex Kendrick
Tony Goldwyn
Christian Duguay
Alan Alda
Phil Joanou
Richard Brooks
Todd Haynes
Steven Seagal
Joss Whedon
Rod Lurie
Paul Michael Glaser
Chris Nahon
Ed Harris
John Erick Dowdle
Scott Hicks
Andrew Bergman
Charles Shyer
Peter Sollett
Jane Campion
Ricky Gervais
Hal Ashby
Zach Braff
Damien Wayans
Kimberly Peirce
Tom Hanks
Sammo Hung Kam-Bo
Roberto Benigni
Patrick Read Johnson
Richard Kelly
Jim Abrahams
Morgan Spurlock
Martha Coolidge
Sheldon Lettich
Katia Lund
Mary Harron
David Mamet


Gina Prince-Bythewood
Charles Martin Smith
Edward Burns
James Foley
Frank Capra
Richard Benjamin
Michael Curtiz
Elia Kazan
Frank Perry
Atom Egoyan
Uwe Boll
Sylvain White
Chan-wook Park
Catherine Owens
Julian Schnabel
Adrienne Shelly
Michael Winterbottom
Robert Duvall
Emilio Estevez
John Ford
Mike Leigh
Sergio Leone
Wong Kar-wai
Michael Radford
Jack Lee Thompson
Danny De Vito
Billy Bob Thornton
John Sayles
E. Elias Merhige
Sam Peckinpah
Karey Kirkpatrick
Bill Paxton
Werner Herzog
Neal Brennan
Kasi Lemmons
Tony Bill
Mark DiSalle
Franco Zeffirelli
Michael Polish
Michael O. Sajbel
Drew Barrymore
Tom Vaughan
Martin Ritt
John Sturges
Albert Brooks
Sidney J. Furie
Noah Baumbach
Costa-Gavras
Bill Duke


Todd Solondz
Roland Joffe
John Waters
Howard Hawks
Mark Duplass
Ronald Neame
Deepa Mehta
Bob Fosse
John Wayne
James Toback
Kevin Spacey
Charles S. Dutton
Jim Jarmusch
Hal Needham
Tamara Jenkins
Brad Anderson
Blake Edwards
Bille August
Vondie Curtis-Hall
Tom Hooper
Jeffrey W. Byrd
Sam Firstenberg
Justin Lin
Andy Garcia
Stanley Donen
John Turturro
Sally Field
Vincent Gallo
Vincente Minnelli
Whit Stillman
Paul Schrader
Fred Schepisi
Emile Ardolino
Paul Mazursky
Thomas Vinterberg
Antonio Banderas
Abel Ferrara
Jeff Burr
Ryan Little
Lars Von Trier
Matt Dillon
Richard E. Grant
Michael Crichton
Akira Kurosawa
Alan Rudolph
Steve Buscemi
Luke Wilson
Joey Lauren Adams
Zak Penn

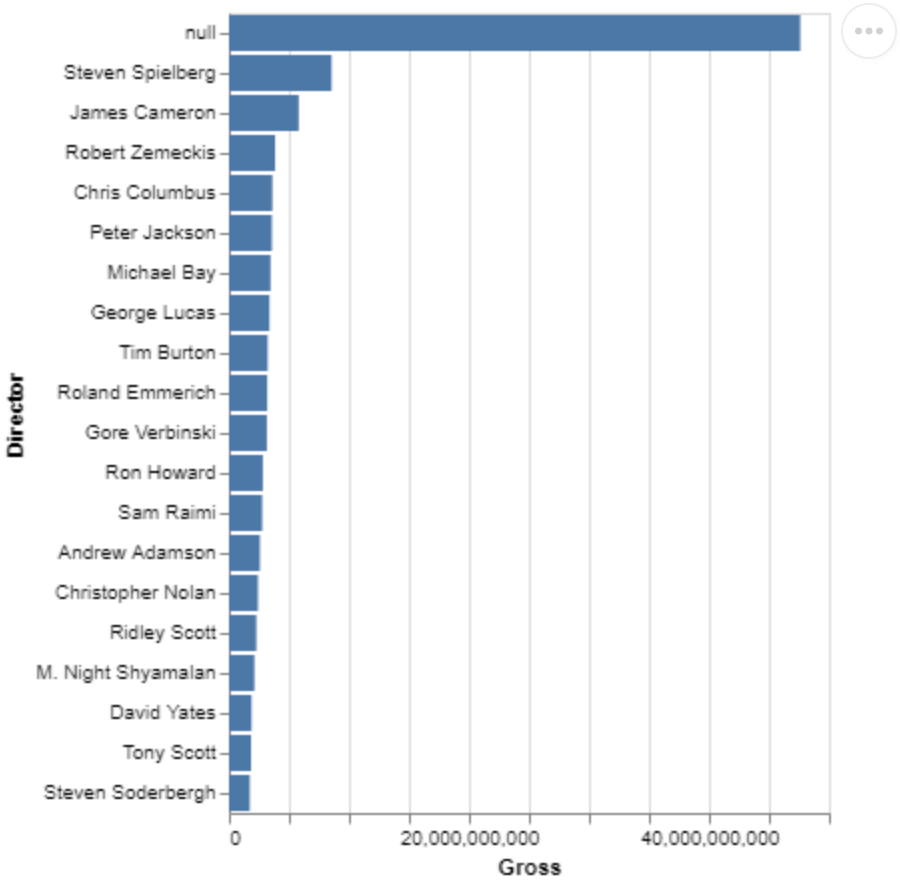



That's a lot of directors! Let's restrict to the top 20 -> `transform_window()` + `transform_filter`

*Side note: when do we know when to use `transform_window()` vs. `transform_aggregate()`?*
`transform_aggregate()`: we want to reduce the number of rows (e.g., one row per group) *
`transform_window()`: we want to keep the same number of rows

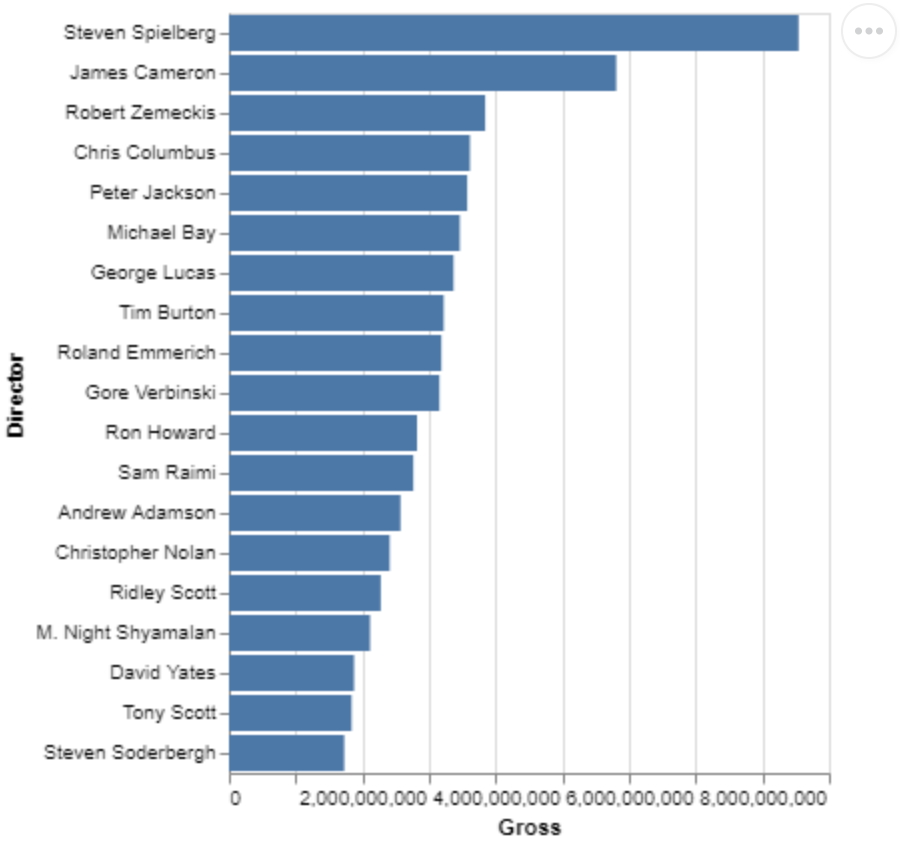
In this case, we want to know the *ranking* of each row, but not reduce the total number of rows. So, use `transform_window()`.

```
alt.Chart(movies_url).mark_bar().transform_aggregate(
    Gross='sum(Worldwide_Gross)',
    groupby=['Director']
).transform_window(
    Rank='rank()',
    sort=[alt.SortField('Gross', order='descending')]
).transform_filter(
    'datum.Rank <= 20'
).encode(
    alt.X('Gross:Q'),
    alt.Y('Director:N', sort=alt.EncodingSortField(
        op='max', field='Gross', order='descending'
    ))
)
```




`null` is not a director, and we certainly don't want to say they're the highest-grossing director. So let's remove that -> `transform_filter()` again

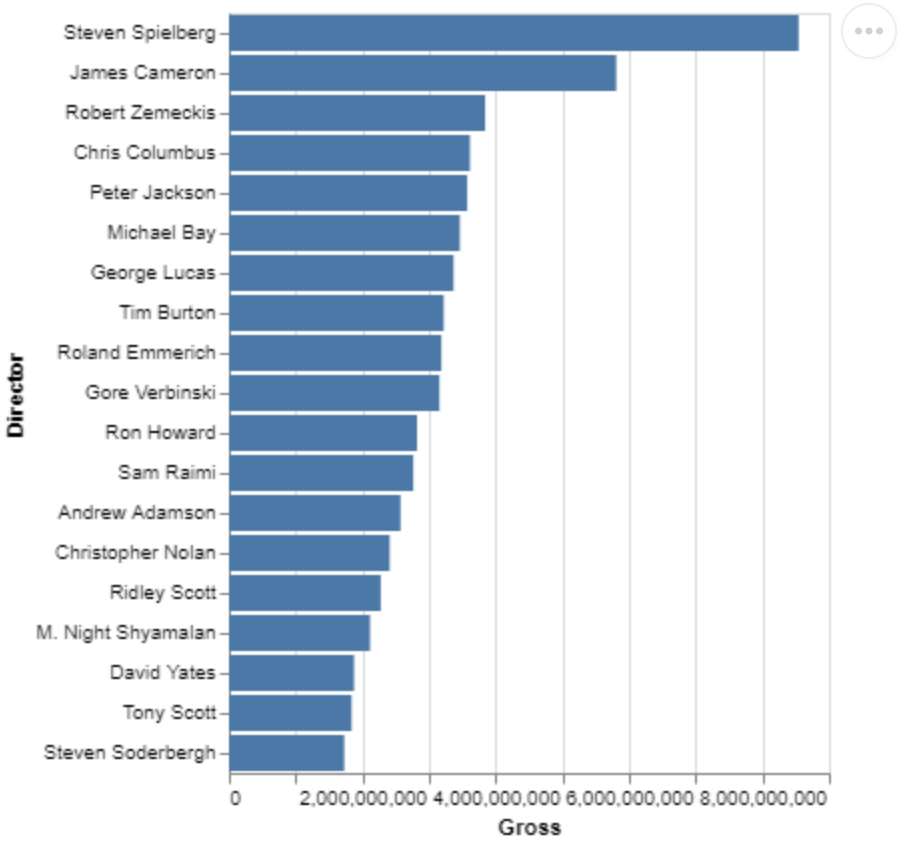
```
alt.Chart(movies_url).mark_bar().transform_aggregate(
    Gross='sum(Worldwide_Gross)',
    groupby=['Director']
).transform_window(
    Rank='rank()',
    sort=[alt.SortField('Gross', order='descending')]
).transform_filter(
    'datum.Rank <= 20'
).transform_filter(
    'datum.Director != null'
).encode(
    alt.X('Gross:Q'),
    alt.Y('Director:N', sort=alt.EncodingSortField(
        op='max', field='Gross', order='descending'
    ))
)
```




**Question**: How many directors are displayed now? Why?

We need to remove `null` *before* ranking and filtering. Our final graph:

```
alt.Chart(movies_url).mark_bar().transform_filter(
    'datum.Director != null'
).transform_aggregate(
    Gross='sum(Worldwide_Gross)',
    groupby=['Director']
).transform_window(
    Rank='rank()',
    sort=[alt.SortField('Gross', order='descending')]
).transform_filter(
    'datum.Rank < 20'
).encode(
    alt.X('Gross:Q'),
    alt.Y('Director:N', sort=alt.EncodingSortField(
        op='max', field='Gross', order='descending'
    ))
)
```




## `transform_window`: in-class exercise

Steven Spielberg has been quite successful in his career! However, showing sums might favor directors who have had longer careers, and so have made more movies and thus more money.

What happens if we change the choice of aggregate operation? Who is the most successful director in terms of `average` gross per film? Modify the aggregate transform above!

## Advanced data transformation: summary

| Purpose | Vega | `pandas` equivalent |
|---|---|---|
| Define a new variable | `transform_calculate()` | `df['new_col']` |
| Filter to subset of rows | `transform_filter(cond)` | `df.loc[cond]` |
| Aggregate function - reduces number of rows down to one per group | `transform_aggregate(groupby(...))` | `df.groupby('A').agg('mean')` |
| Window function - transform across multiple rows, keeps same num. of rows) | `transform_window(sum())` | `df['values'].cumsum()` |

Finally, Altair actually has 19 transformation methods (and counting...) and we have only covered four of them. Read about the rest of them here.