Visualization (Data Transformation)

Peter Ganong and Maggie Shi

January 21, 2026

Introduction: roadmap

- Putting this lecture in context
- Introducing the `movies` dataset
  - load data
  - `shape`
  - `head()`

Putting this lecture in context

- This lecture explores methods for *transforming* data, focusing on aggregation.
  - We will be mostly following Chapter 3 in the data visualization (Heer et al.) book
- 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.

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

Load packages and data

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

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


```
[
  {
    "Title": "The Land Girls",
    "US_Gross": 146083,
    "Worldwide_Gross": 146083,
    "US_DVD_Sales": null,
    "Production_Budget": 8000000,
    "Release_Date": "Jun 12 1998",
    "MPAA_Rating": "R",
    "Running_Time_min": null,
    "Distributor": "Gramercy",
    "Source": null,
    "Major_Genre": null,
    "Creative_Type": null,
    "Director": null,
    "Rotten_Tomatoes_Rating": null,
    "IMDB_Rating": 6.1,
    "IMDB_Votes": 1071
  },
  {
    "Title": "First Love, Last Rites",
    "US_Gross": 10876,
    "Worldwide_Gross": 10876,
    "US_DVD_Sales": null,
    "Production_Budget": 300000,
    "Release_Date": "Aug 07 1998",
    "MPAA_Rating": "R",
    "Running_Time_min": null,
    "Distributor": "Strand",
    "Source": null,
    "Major_Genre": "Drama",
    "Creative_Type": null,
    "Director": null,
    "Rotten_Tomatoes_Rating": null,
    "IMDB_Rating": 6.9,
    "IMDB_Votes": 207
  },
```

An aside on JSON

- Movies database is stored as a `.json` at this URL
- Recall `altair` that writes Vega-lite, which is also recorded in JSON!
  - JSON is just a “syntax” to store text, numbers, etc. in a human-readable way
  - In spatial lectures, we will also encounter the “geojson” format, which can store geographic features

head()

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

	Title	US_Gross	Worldwide_Gross	US_DVD_Sales	Production_Budget	Release_Date	MPAA_Rating	Runni
0	The Land Girls	146083.0	146083.0	NaN	8000000.0	Jun 12 1998	R	NaN
1	First Love, Last Rites	10876.0	10876.0	NaN	300000.0	Aug 07 1998	R	NaN
2	I Married a Strange Person	203134.0	203134.0	NaN	250000.0	Aug 28 1998	None	NaN
3	Let's Talk About Sex	373615.0	373615.0	NaN	300000.0	Sep 11 1998	None	NaN
4	Slam	1009819.0	1087521.0	NaN	1000000.0	Oct 09 1998	R	NaN

shape

```python
movies.shape
```

```
(3201, 16)
```

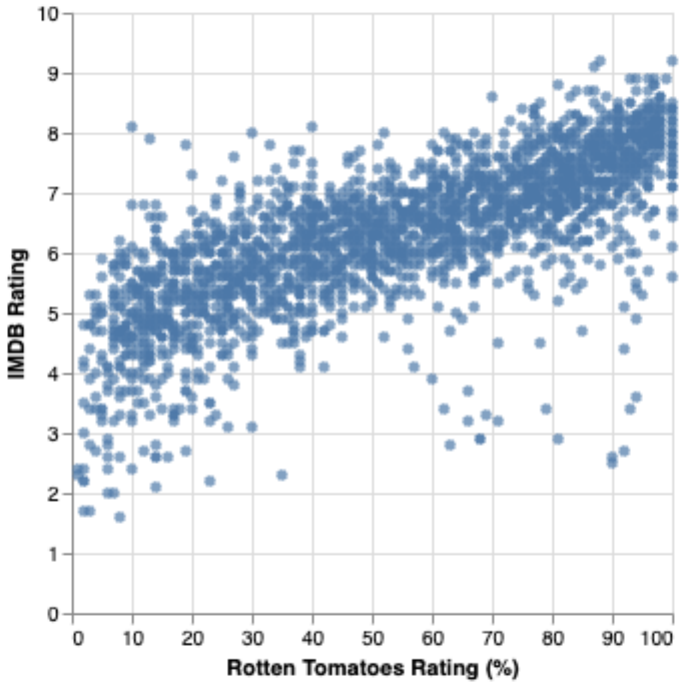
With 3201 movies, we are going to need to do some transformation if we want to uncover any patterns in the data!

Variables of interest

- **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.

Exploring the raw data

```
alt.Chart(movies_url).mark_circle().encode(
    alt.X('Rotten_Tomatoes_Rating:Q', title = "Rotten Tomatoes Rating (%)")
    alt.Y('IMDB_Rating:Q', title = "IMDB Rating")
)
```



Recall from last lecture: label when scale is %!

Aggregation

Aggregation: roadmap

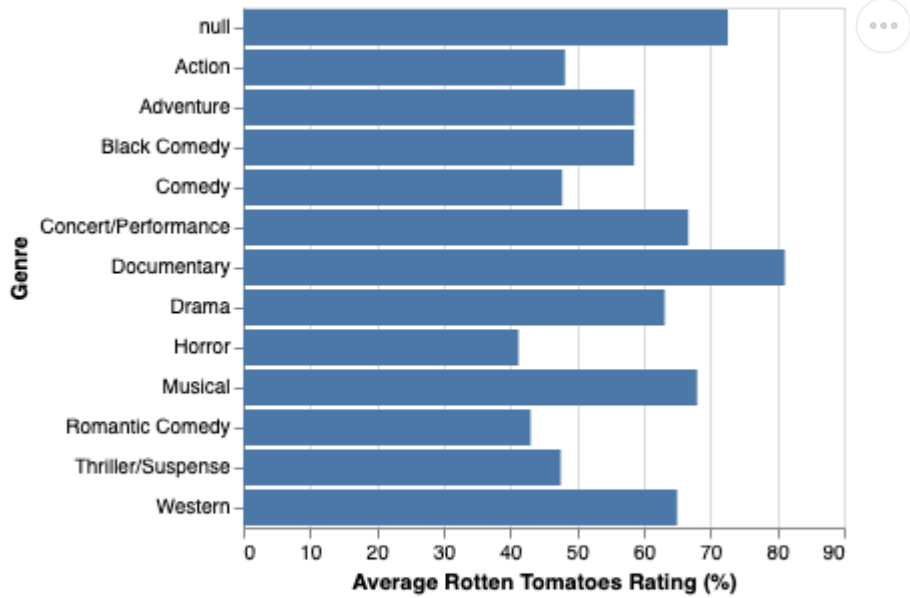
In previous lectures, we actually already saw aggregation via `average()` and `min()`. We just didn't talk explicitly about that step. Now, we examine it more carefully.

- `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', title = "Average Rotten Tomatoes Rating (%)"),
    alt.Y('Major_Genre:N', title = "Genre")
)
```

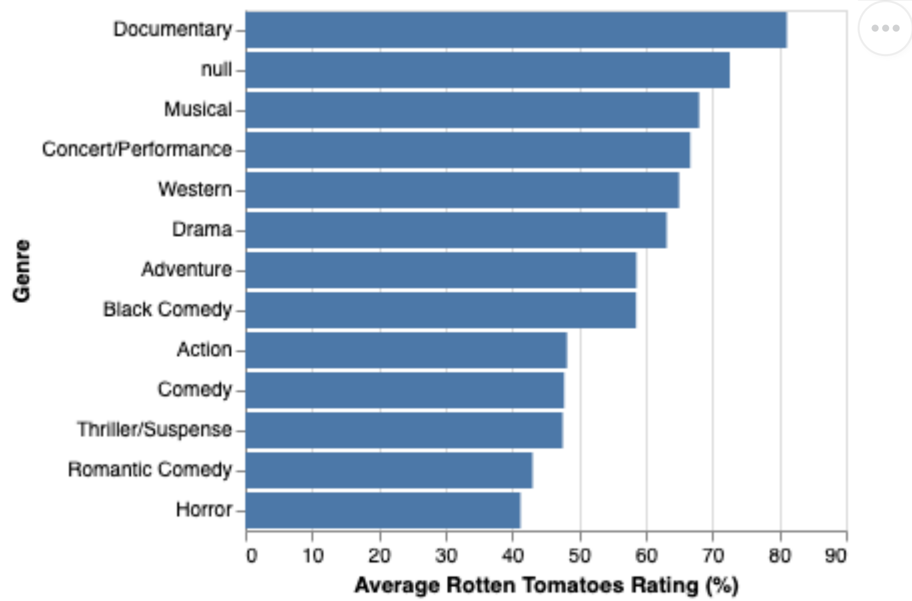


- This plot is fine, but hard to interpret takeaways quickly.
- Discussion Question: what does the y-axis seem to be sorted on? Why?

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

More useful: sort the bars vertically, based on x-axis encoding

```
alt.Chart(movies_url).mark_bar().encode(
    alt.X('average(Rotten_Tomatoes_Rating):Q', title = "Average Rotten Tomatoes Rating (%)"),
    alt.Y('Major_Genre:N', title = "Genre",
        sort=alt.EncodingSortField(op='average', field='Rotten_Tomatoes_Rating', order='descen
    )
)
```

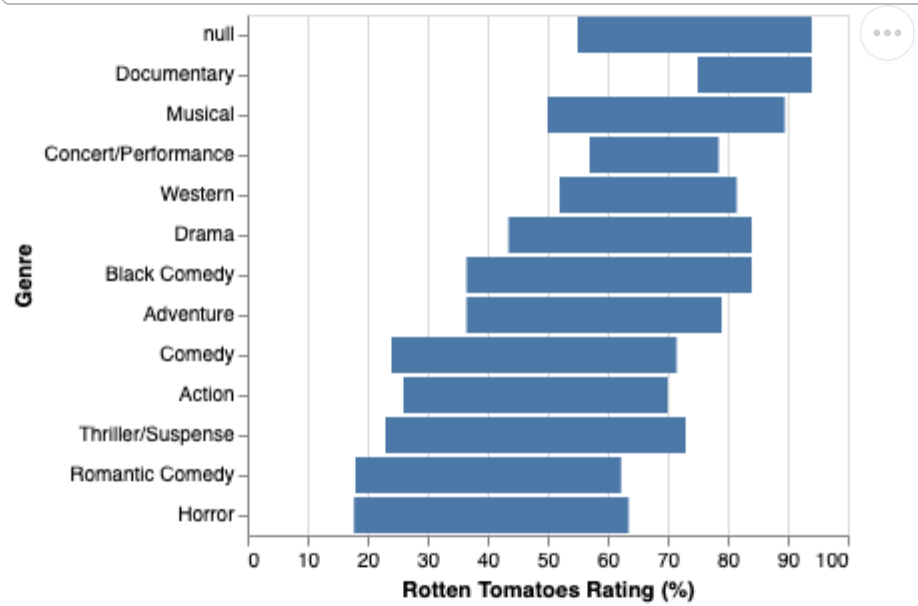


This focuses the viewer's attention on which movie types are most and least popular

Interquartile range

Plot 1st and 3rd quartiles, then sort by median.

```
alt.Chart(movies_url).mark_bar().encode(
    alt.X('q1(Rotten_Tomatoes_Rating):Q', title = "Rotten Tomatoes Rating (%)"),
    alt.X2('q3(Rotten_Tomatoes_Rating):Q'),
    alt.Y('Major_Genre:N', sort=alt.EncodingSortField(op='median', field='Rotten_Tomatoes_Rati
        title = "Genre"
    )
)
```



Discussion question: what can you learn from the IQR plot that you could not learn from the plot with just `average()`?

Aggregation functions

- Distribution: `min()`, `q1()`, `median()`, `mean()`, `q3()`, `max()`
- Dispersion: `variance()`, `stdev()`
- Bootstrap confidence intervals: `ci0()`, `ci1()`
- Full list here

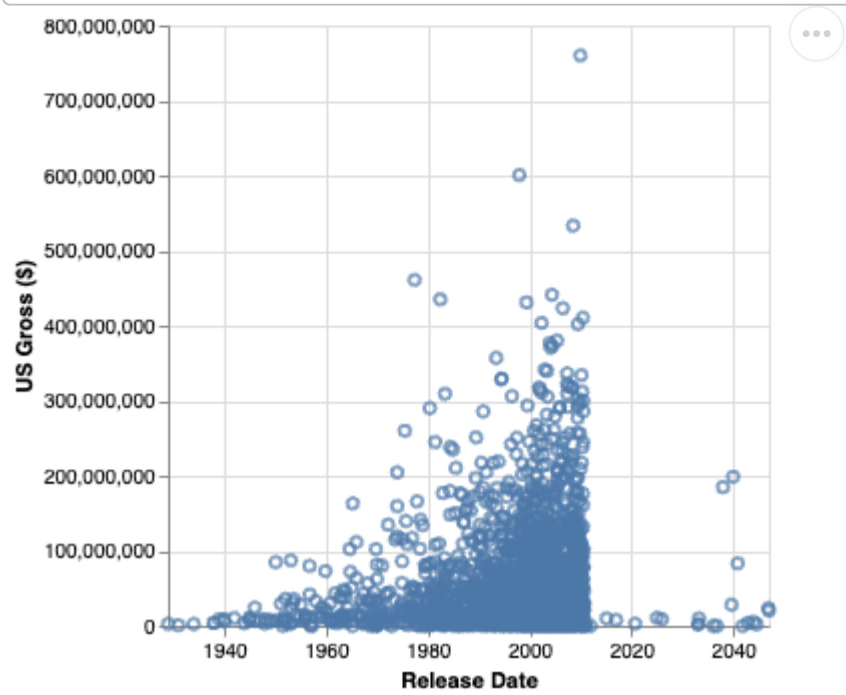
Case study: when are the highest grossing films?

```python
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

```
alt.Chart(movies_url).mark_point().encode(
    alt.X('Release_Date:T', title = "Release Date"),
    alt.Y('US_Gross:Q', title = "US Gross ($)")
)
```



Obviously we need to aggregate.

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

Do-pair-share

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

- **Question**: What time of year are the highest grossing films released? Aggregate both the x- and the y-variables.
- There are several ways to approach answering this question. What seems most reasonable to you?

Data aggregation: Do-pair-share

Starter code in lecture `dps_highest_grossing_film.qmd` file:

```python
import pandas as pd
import altair as alt
movies_url = 'https://cdn.jsdelivr.net/npm/vega-datasets@1/data/movies.json'
movies = pd.read_json(movies_url)

# unaggregated scatter plot
alt.Chart(movies_url).mark_point().encode(
    alt.X('Release_Date:T', title = "Release Date"),
    alt.Y('US_Gross:Q', title = "US Gross ($)")
)
```

Documentation on working with time units here

More on time units

Temporal variables can be transformed into a variety of other time units

- `year`
- `quarter`
- `month`
- `date` (numeric day in month)
- `day` (day of the week)
- `hours`
- `yearmonth`
- `hoursminutes`

Aggregation: summary

- Many built-in aggregation functions to quantify distribution, dispersion, and characterize data
- Dates: see prior slide

Advanced data transformation

Advanced data transformation: introduction

- Two ways to aggregate data in `altair`
  - Within the encoding itself: `alt.Y('median(US_Gross):Q')`
  - Separately 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

Advanced data transformation: roadmap

- `transform_calculate()`
- `transform_timeunit()`
- `transform_filter()`
- do-pair-share
- `transform_aggregate()`
- `transform_window()`

These are all written in the Vega expression language.

Connection to pandas operations

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

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 - collapse 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()`

Connection to pandas operations

- You already know how to do these all in `pandas` so it is not conceptually new.
- Why bother doing it in `altair`?
  - **Exploratory data analysis** can be done faster in `altair`: manipulate data and plot simultaneously
  - Aggregation and transformations are **temporary** – don't need to define and keep track of new aggregated dataframes

`transform_calculate` and `transform_timeunit`

- `transform_calculate()` uses expressions for writing basic formulas
  - Math functions: `min()`, `random()`, `round()`
  - Statistical functions: `sampleNormal()`, `sampleUniform()`
  - String functions: `length()`, `lower()`, `substring()`
- Use `transform_timeunit()` when working with Temporal variables
  - `month()`, `quarter()`, `yearmonth()`

`transform_calculate` case study

Question: 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', title = "Release Month"),
    alt.Y('median(NonUS_Gross):Q', title = "Median Non-US Gross ($)")
)
```

`transform_calculate` case study

Question: 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', title = "Release Month"),
    alt.Y('median(NonUS_Gross):Q', title = "Median Non-US Gross ($)")
)
```

- `NonUS_Gross` is a variable we're defining *temporarily*

`transform_calculate` case study

Question: 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', title = "Release Month"),
    alt.Y('median(NonUS_Gross):Q', title = "Median Non-US Gross ($)")
)
```

- `datum` is how you reference the underlying dataset within a transformation expression
- Here, `datum` means `movies_url`

`transform_calculate` case study

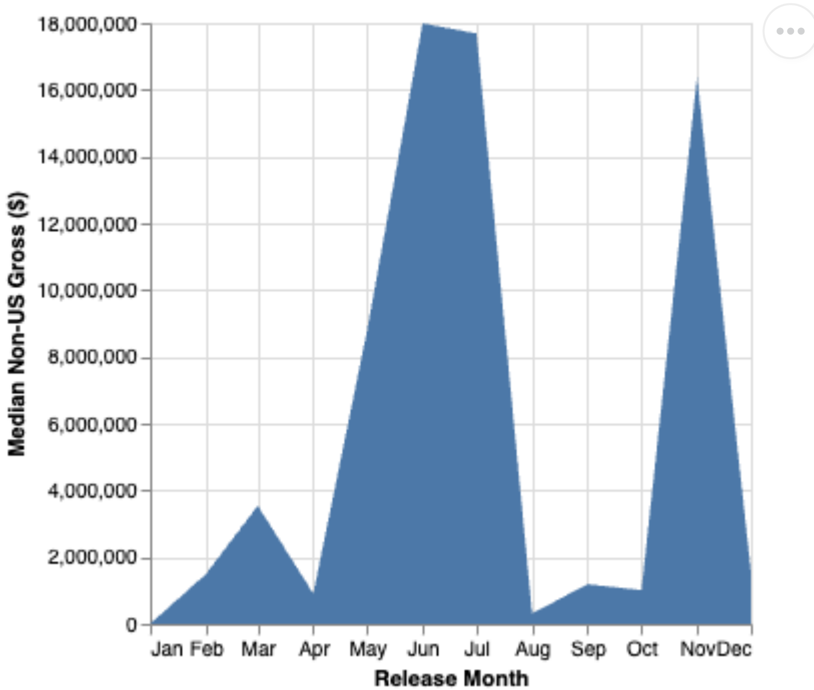
Question: what time of year do US movies make money abroad?

```
alt.Chart(movies_url).mark_area().transform_calculate(
    NonUS_Gross='datum.Worldwide_Gross - datum.US_Gross'
).encode(
    alt.X('month(Release_Date):T', title = "Release Month"),
    alt.Y('median(NonUS_Gross):Q', title = "Median Non-US Gross ($)")
)
```

- After defining `NonUS_Gross`, we can use like any other variable within `movies_url`
- It can be combined with other aggregation methods

`transform_calculate` case study

Question: what time of year do US movies make money abroad?



transform_filter

- Goal: show just movies before 1970

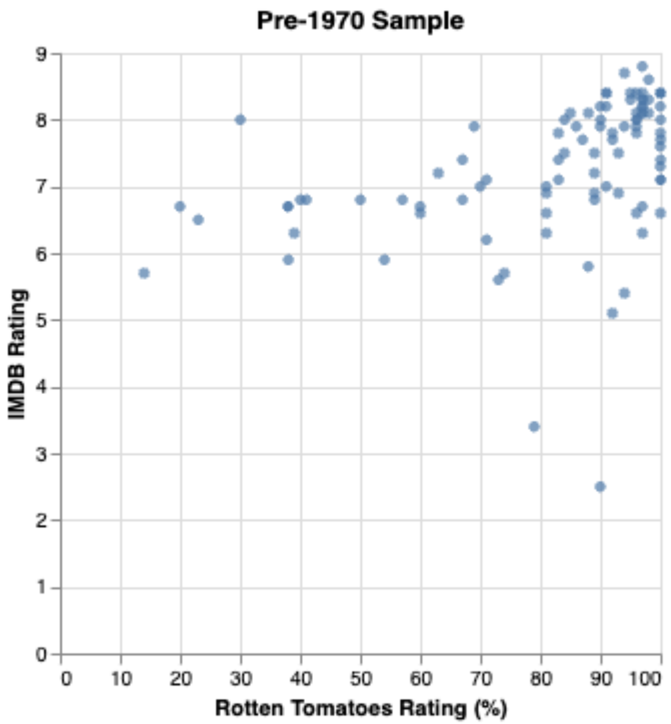
```
alt.Chart(movies_url).mark_circle().transform_filter(
    'year(datum.Release_Date) < 1970').encode(
    alt.X('Rotten_Tomatoes_Rating:Q', title = "Rotten Tomatoes Rating (%)")
    alt.Y('IMDB_Rating:Q', title = "IMDB Rating"),
).properties(title = "Pre-1970 Sample")
```

- `transform_filter` filters the dataset based on an expression
- Like `transform_aggregate`, this filtering is *temporary*

transform_filter

- Goal: show just movies before 1970

```
alt.Chart(movies_url).mark_circle().transform_filter(
    'year(datum.Release_Date) < 1970').encode(
    alt.X('Rotten_Tomatoes_Rating:Q', title = "Rotten Tomatoes Rating (%)"),
    alt.Y('IMDB_Rating:Q', title = "IMDB Rating")
).properties(title = "Pre-1970 Sample")
```



Do-pair-share

- Make two plots that compare ratings before and after 1970
  - Use `transform_filter()` to create a plot of ratings before vs. after 1970, then append side-by-side
  - Plot before and after 1970 on one plot
    - Use `transform_aggregate()` to 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
- These plots show equivalent information. Which do you prefer and why?

Do-pair-share

Starter code in lecture `dps_1970.qmd` file:

```python
import pandas as pd
import altair as alt
movies_url = 'https://cdn.jsdelivr.net/npm/vega-datasets@1/data/movies.json'
movies = pd.read_json(movies_url)

# scatter plot, filtered to < 1970
alt.Chart(movies_url).mark_circle().encode(
    alt.X('Rotten_Tomatoes_Rating:Q', title = "Rotten Tomatoes Rating (%)"),
    alt.Y('IMDB_Rating:Q', title = "IMDB Rating")
).transform_filter('year(datum.Release_Date) < 1970'
).properties(title = "Pre-1970 Sample")
```

Hint: recall `graphA | graphB` plots `graphA` next to `graphB`

`transform_window`: case study

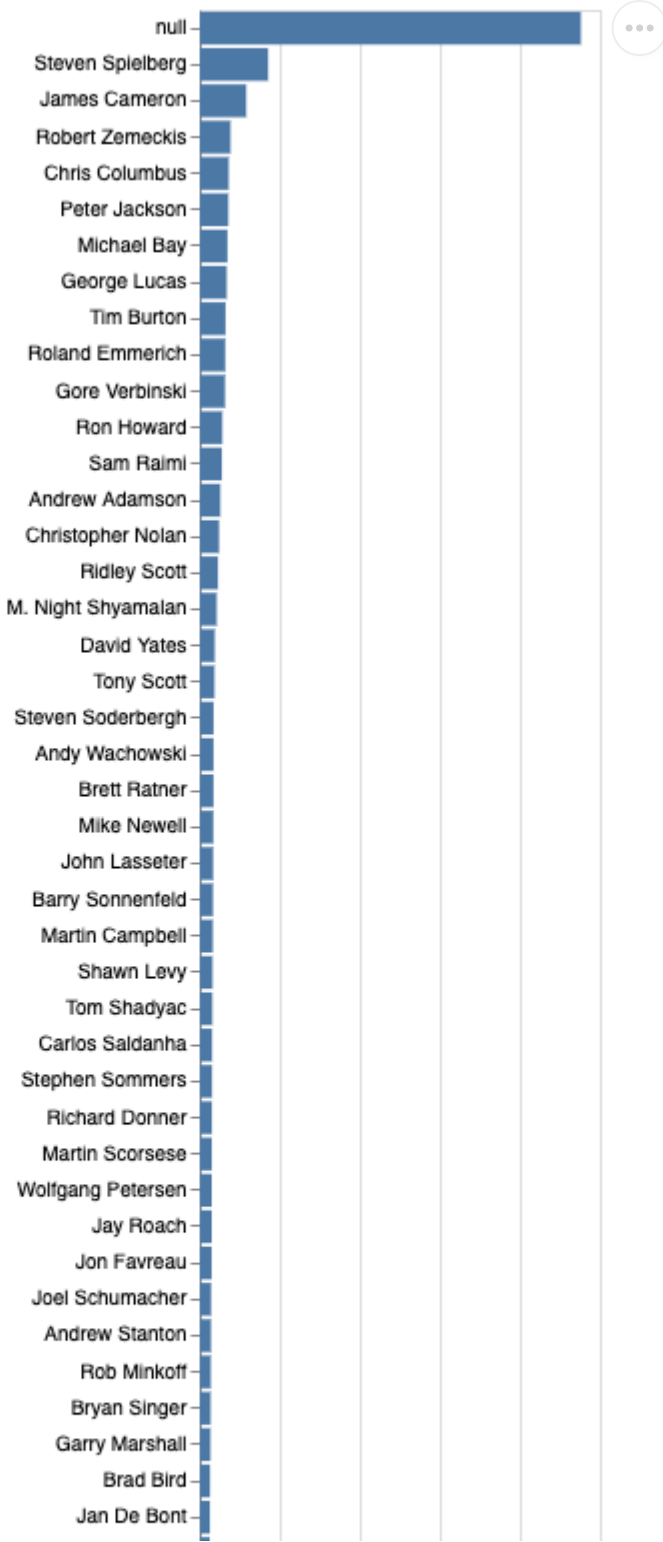
Question: who are the top grossing directors of all time?

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

- First, sum `Worldwide_Gross` for each director to make `Gross`
- Then, plot in descending order of `Gross`

`transform_window`: case study

Question: who are the top grossing directors of all time?


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

`transform_window`: case study

That's a lot of directors! Let's restrict to the **top 10**

```python
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 <= 10'
).encode(
    alt.X('Gross:Q', title = "Worldwide Gross ($)"),
    alt.Y('Director:N', sort=alt.EncodingSortField(
        op='max', field='Gross', order='descending'
    ), title = "Director")
)
```

- Use `transform_window()` when we want to add a variable and keep the *same* number of rows
- Opposed to `transform_aggregate()` which *collapses* the number of rows

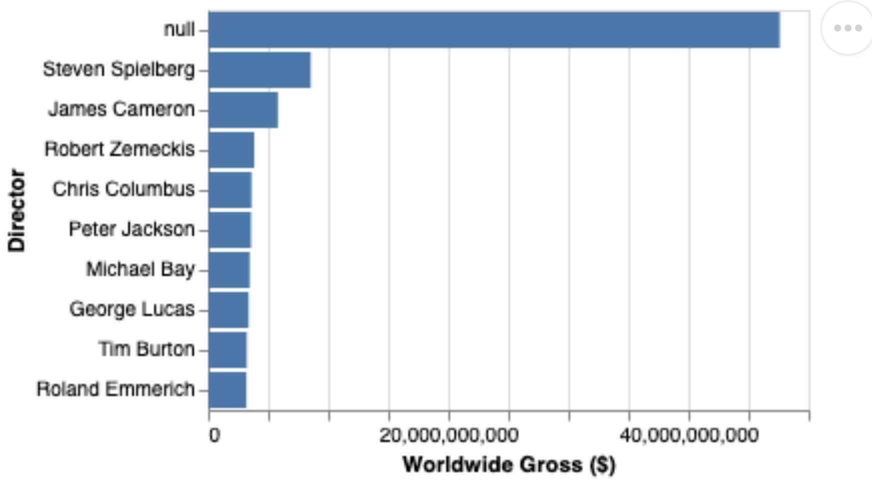
`transform_window`: case study

That's a lot of directors! Let's restrict to the **top 10**

```
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 <= 10'
).encode(
    alt.X('Gross:Q', title = "Worldwide Gross ($)"),
    alt.Y('Director:N', sort=alt.EncodingSortField(
        op='max', field='Gross', order='descending'
    ), title = "Director")
)
```

After ranking, use `transform_filter()` to restrict to the top 10 highest-ranked

`transform_window`: case study



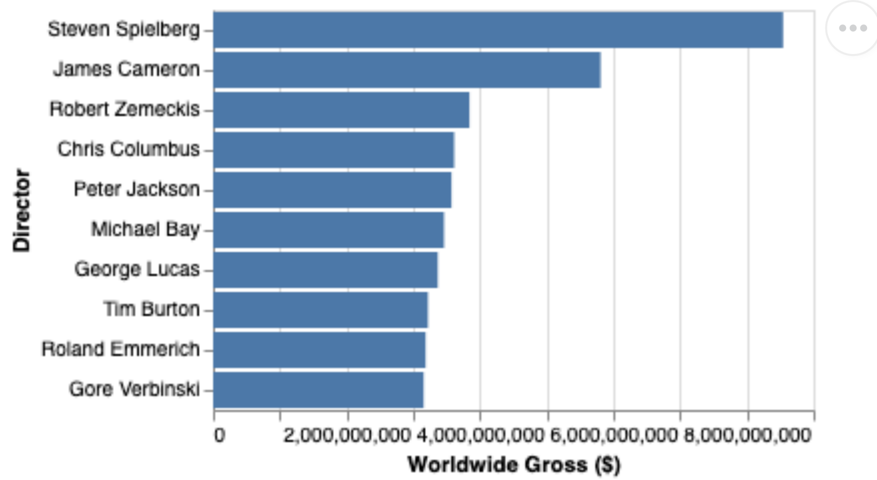
- `null` is not a director, and we certainly don't want to say they're the highest-grossing director.
- So let's remove that -> `transform_filter()` again

`transform_window`: case study

```
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 <= 11'
).transform_filter(
    'datum.Director != null'
).encode(
    alt.X('Gross:Q', title = "Worldwide Gross ($)"),
    alt.Y('Director:N', sort=alt.EncodingSortField(
        op='max', field='Gross', order='descending'
    ), title = "Director")
)
```

Note that the `transform_filter()` is now `<=11`. Why?

`transform_window`: case study



Steven Spielberg has been quite successful in his career!

`transform_window`: do-pair-share

- 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?

`transform_window`: do-pair-share

Starter code in `dps_directors.qmd`:

```
import pandas as pd
import altair as alt
movies_url = 'https://cdn.jsdelivr.net/npm/vega-datasets@1/data/movies.json
movies = pd.read_json(movies_url)

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 < 10'
).encode(
    alt.X('Gross:Q', title = "Worldwide Gross ($)"),
    alt.Y('Director:N', sort=alt.EncodingSortField(
```

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()`

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