

SI649 Altair Demo

Gabriel Grill 2024

Originally created by Liang Sie 2023



Agenda


- 1st document to open:
 - Altair2_Blank
- 2nd document
 - Debugging_Examples_Blank
- 3rd document to read:
 - Exporting from Altair
 -  We will dive deeper next week, but it will be helpful if you take a look at this document

Table of Contents

Altair_Transforms_Blank

- Warmup
- Layering
- Data Transformation

Section 1

Warmup



Load Data

- Input

```
movies_url = vega_data.movies.url
movies = pd.read_json(movies_url)
movies.columns = movies.columns.str.strip().str.replace(' ', '_').str.replace('(',
)').str.replace(')', ' ')
movies.sample(2)
```

- Output

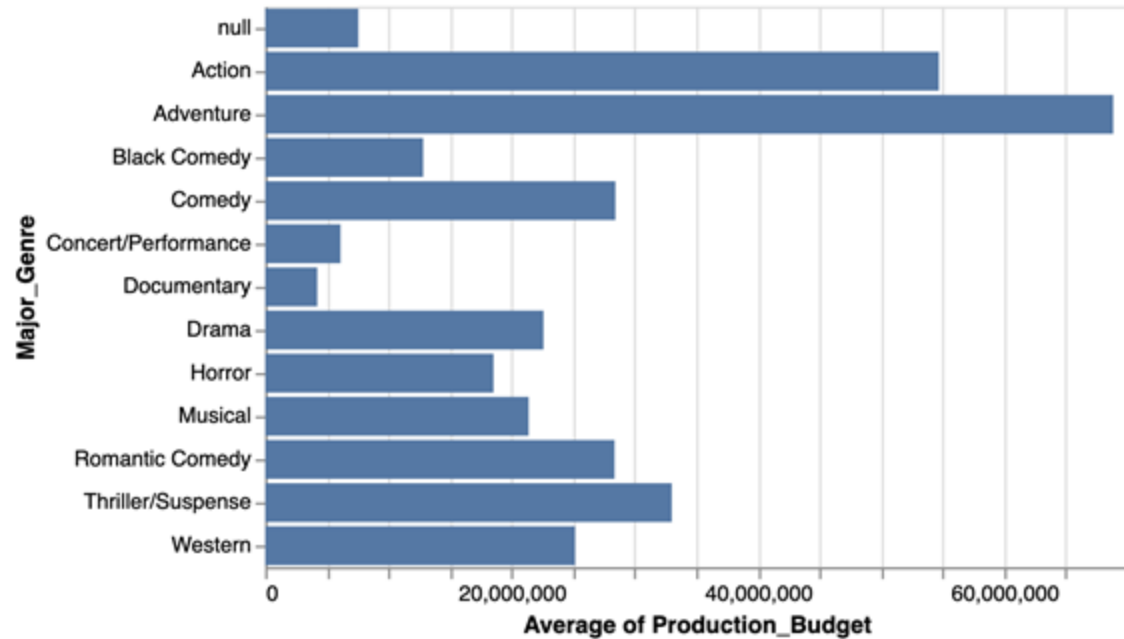
Title	US_Gross	Worldwide_Gross	US_DVD_Sales	Production_Budget	Release_Date	MPAA_Rating	Running_Time_min	Distributor	Source	Major_Genre	Creative_Type	Director
Mortal Kombat: Annihilation	35927406.0	51327406.0	NaN	30000000.0	Nov 21 1997	PG-13	91.0	New Line	Based on Game	Action	Fantasy	None
Sleeper	9408183.0	9408183.0	NaN	10000000.0	Jul 09 2004	PG	NaN	MGM	Original Screenplay	Comedy	Contemporary Fiction	None

Recap

- Mark: e.g., point, rect, bar
- Data: i.e., variables and types
- Encoding: e.g., x, y, color

Bar Chart

- A basic bar chart for **Major_Genre** and avg of **Production Budget**



Bar Chart

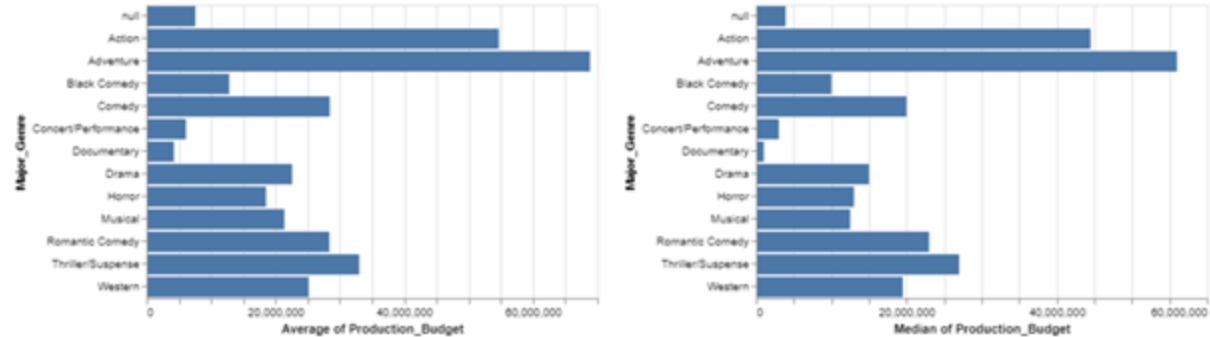
Answer

- A basic bar chart for **Major_Genre** and **avg of Production Budget**
 - ```
alt.Chart(movies).mark_bar().encode(
 alt.Y('Major_Genre:N'),
 alt.X('average(Production_Budget):Q')
)
```



# Compound Chart I

- A compound chart of **Major\_Genre** and avg of Production Budget and **Major\_Genre** and median of Production Budget



# Compound Chart I

## Answer

- A compound chart of **Major\_Genre** and avg of Production Budget and **Major\_Genre** and median of Production Budget

```
○ chart1=alt.Chart(movies).mark_bar().encode(
 alt.Y('Major_Genre:N'),
 alt.X('average(Production_Budget):Q')
)
```

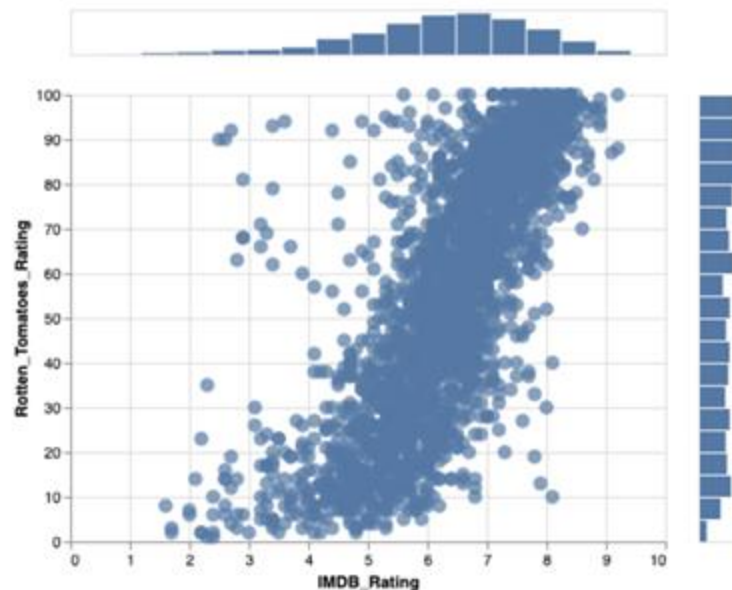
```
chart2=alt.Chart(movies).mark_bar().encode(
 alt.Y('Major_Genre:N'),
 alt.X('median(Production_Budget):Q')
)
```

```
chart1|chart2
```

# Compound Chart II

## Hint

- A scatter plot of **Rotten Tomatoes Rating** and **IMDB Rating** with the distribution histogram of both axis



# Compound Chart II

## Hint

- A scatter plot of **Rotten Tomatoes Rating** and **IMDB Rating** with the distribution histogram of both axis
  - Create a scatter plot first (filled=True, size=90)
  - Create bar charts for both sides
    - Remove axis title (`axis=None`)
    - Create bins and set bins to `alt.BinParams(maxbins=20)`
    - Set property height or width as 30
  - `distribtop & (points | distribright)`

# Compound Chart II

## Answer

- A scatter plot of **Rotten Tomatoes Rating** and **IMDB Rating** with the distribution histogram of both axis

```
○ points = alt.Chart(movies).mark_point(filled=True, size=90).encode(
 x=alt.X('IMDB_Rating'),
 y=alt.Y('Rotten_Tomatoes_Rating')
)
```

# Compound Chart II

## Answer

- A scatter plot of **Rotten Tomatoes Rating** and **IMDB Rating** with the distribution histogram of both axis
  - `distribright= alt.Chart(movies).mark_bar().encode(
 y =alt.Y(
 'Rotten_Tomatoes_Rating',bin=alt.BinParams(maxbins=20),axis=None),
 x = alt.X('count()',axis=None),
 ).properties(width=30)`
  - `distribtop = alt.Chart(movies).mark_bar().encode(
 x = alt.X('IMDB_Rating',bin=alt.BinParams(maxbins=20),axis=None),
 y = alt.Y('count()',axis=None),
 ).properties(height=30)`
  - `distribtop & (points | distribright)`

# Interaction

- Basic interaction (zoom-in and out)
  - `chart.interactive()`
- Tooltip
  - `chart.encode(  
 tooltip='Field' # Can be multiple fields (list)  
)`.`interactive()`

# Interaction

- Tooltip
  - `points.encode(  
 tooltip=['Title:N','Release_Date:T']  
)`.`interactive()`



# Section 2

## Layering

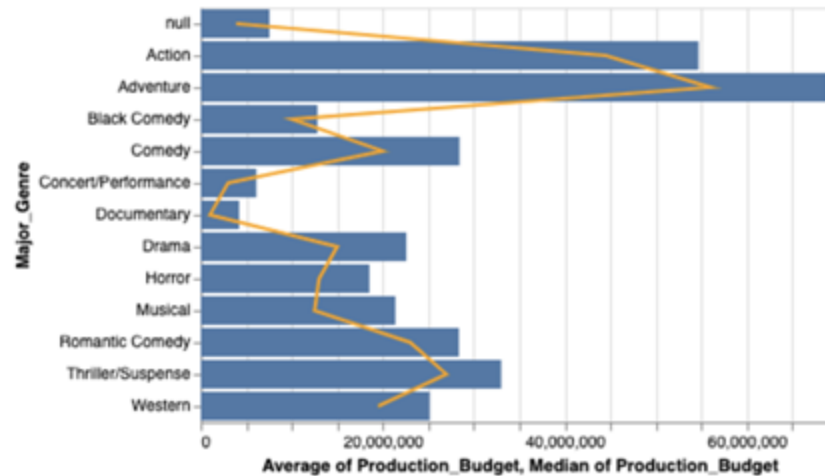


# Layering

- Layering is a very useful compounding method that allows you to overlay two different charts on the same set of axes. You can layer charts using the "+" operator.

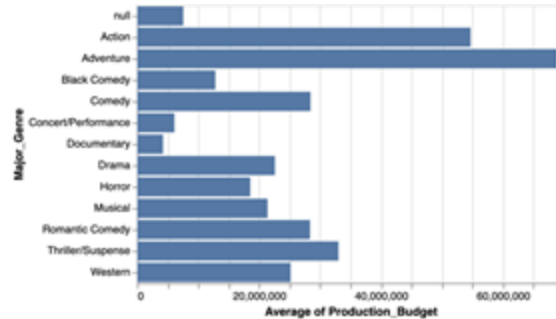
# Layering

- Put two layers together

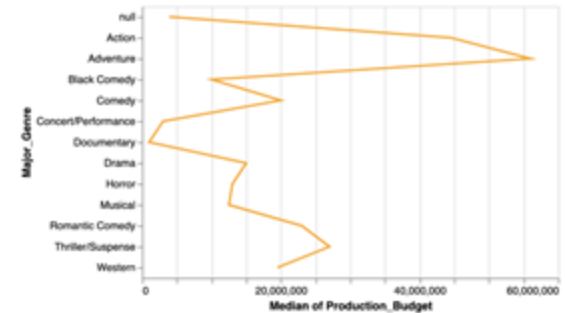


# Layering

- Layer 1: chart1  
(Average - bar)



- Layer 2: chart2  
(Median - line)



# Layering

## Answer

- `chart1=alt.Chart(movies).mark_bar().encode(  
 alt.Y('Major_Genre:N'),  
 alt.X('average(Production_Budget):Q')  
)`
- `chart2=alt.Chart(movies).mark_line(color="orange").encode(  
 alt.Y('Major_Genre:N'),  
 alt.X('median(Production_Budget):Q')  
)`
- **`chart1+chart2`**

# Layering Shortcut

- ```
chart1=alt.Chart(movies).mark_bar().encode(  
    alt.Y('Major_Genre:N'),  
    alt.X('average(Production_Budget):Q')  
)
```

take the first chart and **override** the values

bar --> line

average on X --> median

everything else stays the same (data, Y, etc.)
- ```
chart2=chart1.mark_line(color="orange").encode(
 alt.X('median(Production_Budget):Q')
)
```
- ```
chart1+chart2
```

Layering

Shortcut

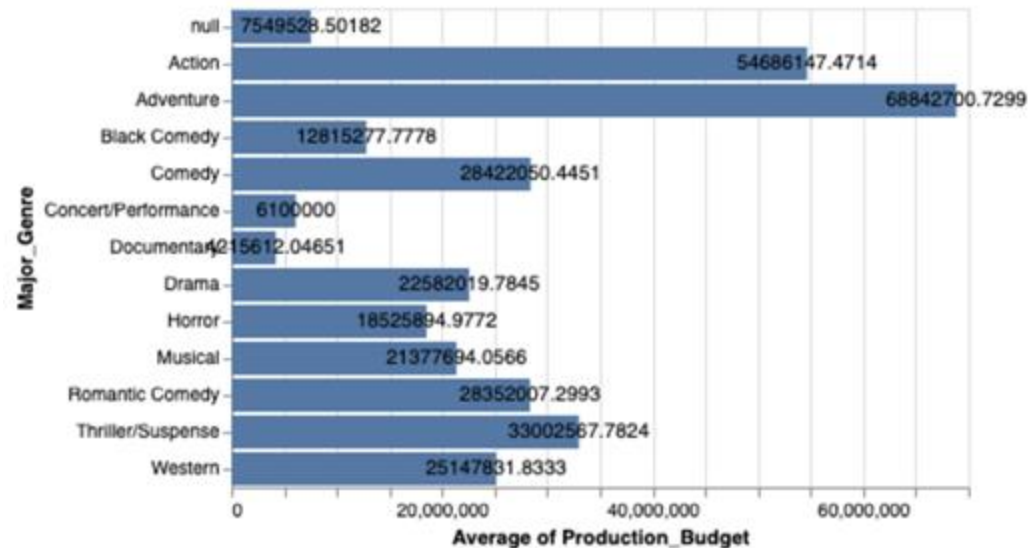
- ```
base = alt.Chart(movies).encode(
 alt.Y('Major_Genre:N')
)
```

### # overridden versions

- ```
chart1=base.mark_bar().encode(  
    alt.X('average(Production_Budget):Q')  
)
```
- ```
chart2=base.mark_line(color="orange").encode(
 alt.X('median(Production_Budget):Q')
)
```
- ```
chart1+chart2
```

Layering Text

- Add text label to the bar chart (Hint: mark_text())



Layering Text

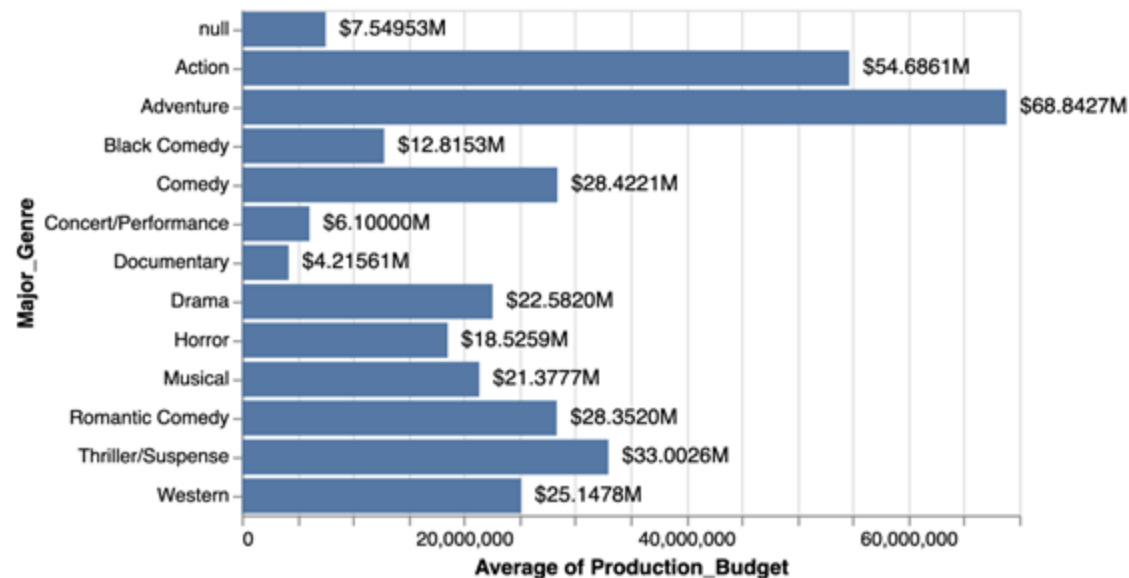
Answer

- ```
chart1=alt.Chart(movies).mark_bar().encode(
 alt.Y('Major_Genre:N'),
 alt.X('average(Production_Budget):Q')
)
```

**# Our text chart "overrides" elements of the base**  
**# in this case we are replacing the mark type and**  
**# adding an encoding**
- ```
text_annotation_for_mean = chart1.mark_text().encode(
    text=alt.Text('mean(Production_Budget)')
)
```
- ```
chart1+text_annotation_for_mean
```

# Layering Text

- Add text label to the bar chart (Hint: mark\_text())



# Layering Text

## Answer

- ```
text_annotation_for_mean = chart1.mark_text(
    align='left',
    dx=7, # horizontal distance between end of bar to the start of
    text
).encode(
    alt.Text(
        'mean(Production_Budget)',
        formatType="number", # necessary for formatting numbers
        format="$s" # formatting numbers into millions
    ),
)
```
- ```
chart1+text_annotation_for_mean
```
- [FormatType documents](#)
- [Format documents](#)

# Section 3 Part I

## Transform/Filter/Aggregate



# Altair Data Transformation

- Data Transformation

| Transform                | Description                                                                     |
|--------------------------|---------------------------------------------------------------------------------|
| Aggregate Transforms     | Create a new data column by aggregating an existing column.                     |
| Bin transforms           | Create a new data column by binning an existing column.                         |
| Calculate Transform      | Create a new data column using an arithmetic calculation on an existing column. |
| Filter Transform         | Select a subset of data based on a condition.                                   |
| Flatten Transform        | Flatten array data into columns.                                                |
| Fold Transform           | Convert wide-form data into long-form data.                                     |
| Impute Transform         | Impute missing data.                                                            |
| Join Aggregate Transform | Aggregate transform joined to original data.                                    |
| Lookup Transform         | One-sided join of two datasets based on a lookup key.                           |
| Sample Transform         | Random sub-sample of the rows in the dataset.                                   |
| Stack Transform          | Compute stacked version of values.                                              |
| TimeUnit Transform       | Discretize/group a date by a time unit (day, month, year, etc.)                 |
| Window Transform         | Compute a windowed aggregation                                                  |

# Aggregate and Join Aggregate

- Aggregate vs. Join Aggregate

| Transform                | Method                                 | Description                                                                     |
|--------------------------|----------------------------------------|---------------------------------------------------------------------------------|
| Aggregate Transforms     | <code>transform_aggregate()</code>     | Create a new data column by aggregating an existing column.                     |
| Bin transforms           | <code>transform_bin()</code>           | Create a new data column by binning an existing column.                         |
| Calculate Transform      | <code>transform_calculate()</code>     | Create a new data column using an arithmetic calculation on an existing column. |
| Density Transform        | <code>transform_density()</code>       | Create a new data column with the kernel density estimate of the input.         |
| Filter Transform         | <code>transform_filter()</code>        | Select a subset of data based on a condition.                                   |
| Flatten Transform        | <code>transform_flatten()</code>       | Flatten array data into columns.                                                |
| Fold Transform           | <code>transform_fold()</code>          | Convert wide-form data into long-form data (opposite of pivot).                 |
| Impute Transform         | <code>transform_impute()</code>        | Impute missing data.                                                            |
| Join Aggregate Transform | <code>transform_joinaggregate()</code> | Aggregate transform joined to original data.                                    |

# Aggregate and Join Aggregate

- Aggregate

- `.transform_aggregate(`  
`groupby=['Major_Genre'],`  
`mean_production_budget='mean(Production_Budget)'`  
`)`

| Major_Genre | mean_production_budget |
|-------------|------------------------|
| x           | 1.5                    |
| y           | 10                     |

- Join Aggregate

- `.transform_joinaggregate(`  
`groupby=['Major_Genre'],`  
`mean_production_budget='mean(Production_Budget)'`  
`)`

| Title | Major_Genre | Value | mean_production_budget |
|-------|-------------|-------|------------------------|
| A     | x           | 1     | 1.5                    |
| B     | x           | 2     | 1.5                    |
| C     | y           | 10    | 10                     |

# Aggregate and Join Aggregate

- Calculate the average production budget per genre
  - ```
alt.Chart(movies).mark_bar().encode(  
    alt.Y('Major_Genre:N'),  
    alt.X('average(Production_Budget):Q')  
)
```
- A longer form
 - ```
alt.Chart(movies).mark_bar().encode(
 alt.Y('Major_Genre:N'),
 alt.X(field='Production_Budget',
 aggregate="mean", type="quantitative")
 # can't use Q here
)
```



# Aggregate

transform\_aggregate()

- Calculate the average production budget per genre (Aggregate)

```
○ alt.Chart(movies).transform_aggregate(
 groupby=['Field'], # can be multiple columns
 # calculating mean and storing in a new variable
 New Column ='aggregation(Field) '
).mark_bar().encode(
 alt.Y(Field used above: DataType),
 alt.X(Field used above: DataType)
)
```

- Data Type must be specified, otherwise

`ValueError: xxx encoding field is specified without a type; the type cannot be inferred because it does not match any column in the data.`

# Aggregate

- Calculate the average production budget per genre (Aggregate)
  - ```
alt.Chart(movies).transform_aggregate(  
    groupby=['Major_Genre'], # can be multiple columns  
    # calculating mean and storing in a new variable  
    mean_production_budget='mean(Production_Budget) '  
) .mark_bar().encode(  
    alt.Y('Major_Genre:N'),  
    alt.X('mean_production_budget:Q')  
)
```

Join Aggregate

- Calculate the average production budget per genre (Aggregate)
 - ```
alt.Chart(movies).mark_bar().transform_joinaggregate(
 groupby=['Major_Genre'],
 mean_production_budget='mean(Production_Budget) '
) .encode(
 alt.Y('Major_Genre:N'),
 alt.X('Production_Budget:Q') # you can use both
 mean_production_budget AND Production_Budget
)
```

# Bin Transformation

transform\_bin()

- Create a barchart with binned IMDB\_rating and mean Production\_Budget
  - ```
alt.Chart(movies).mark_bar().encode(  
    alt.X('IMDB_Rating:Q', bin=True),  
    alt.Y('mean(Production_Budget)')  
)
```
- A longer form
 - ```
alt.Chart(movies).mark_bar().transform_bin(
 'Binned_IMDB_Rating', 'IMDB_Rating'
) .encode(
 alt.X('Binned_IMDB_Rating:Q'),
 alt.Y('mean(Production_Budget)')
)
```

**Almost there**  
Let's take a break



# Section 3 Part II

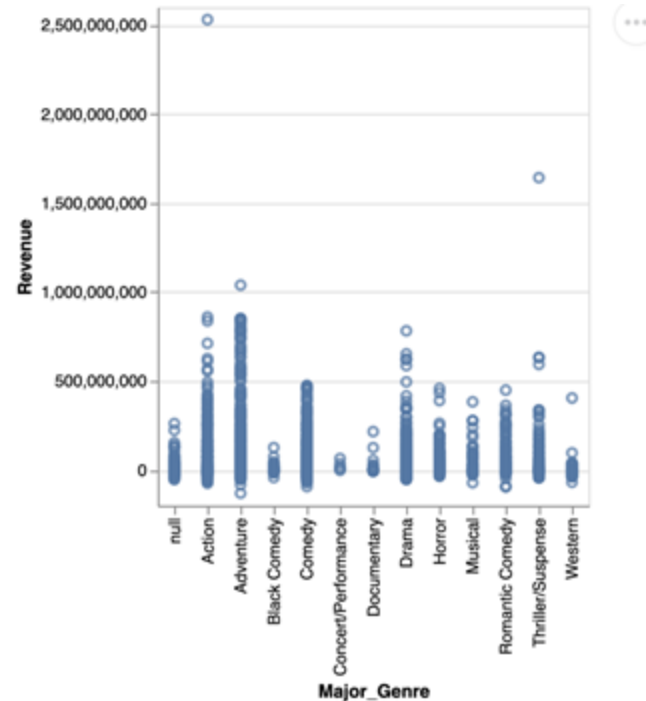
## Transform/Filter/Aggregate



# Calculate Transform

transform\_calculate()

- Plot a scatter plot of **revenue** and genre



# Calculate Transform Answer

- Plot a scatter plot of **revenue** and genre

- ```
alt.Chart(movies).transform_calculate(  
    Revenue=alt.datum.Worldwide_Gross - alt.datum.Production_Budget  
    #alt.datum is a way of describing every row  
) .mark_point().encode(  
    alt.X('Major_Genre:N'),  
    alt.Y('Revenue:Q')  
)
```

- Data Type must be specified, otherwise

ValueError: xxx encoding field is specified without a type; the type cannot be inferred because it does not match any column in the data.

- alt.datum

`alt.value()` specifies a constant range value, while **alt.datum** specifies a domain value.

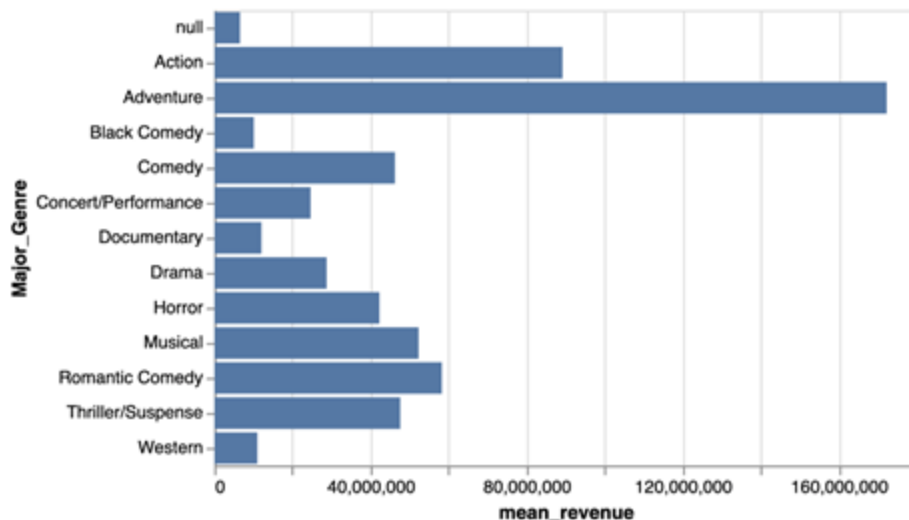
Calculate Transform Answer

- Plot a scatter plot of **revenue** and genre (an alternative way)
 - ```
alt.Chart(movies).transform_calculate(
 Revenue="datum.Worldwide_Gross - datum.Production_Budget" #
 can use datum instead of alt.datum
) .mark_point().encode(
 alt.X('Major_Genre:N'),
 alt.Y('Revenue:Q')
)
```

# Calculate Transform + Aggregate Transform

`transform_calculate()`  
`transform_aggregate()`

- Create a bar chart for **mean revenue** of each **major\_genre**



# Calculate Transform + Aggregate Transform Answer

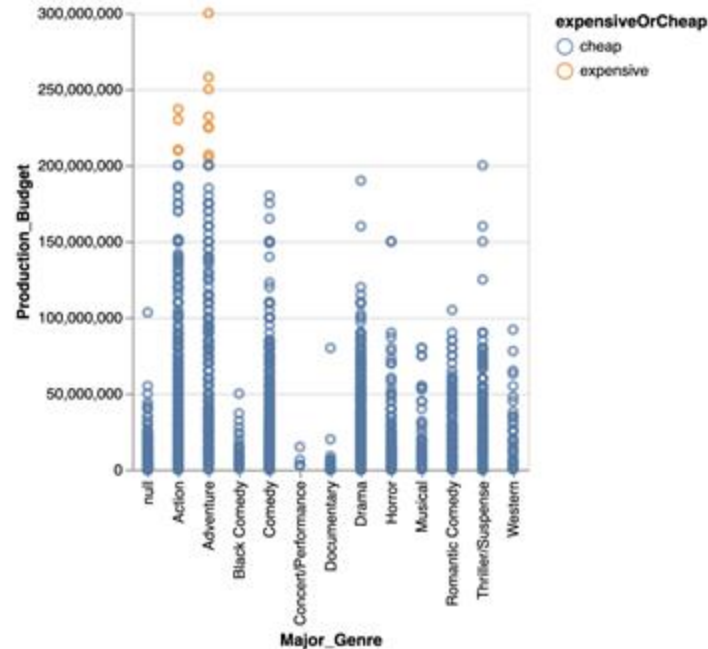
- Create a bar chart for **mean revenue** of each major\_genre
  - `alt.Chart(movies).transform_calculate(  
 Revenue= alt.datum.Worldwide_Gross -  
 alt.datum.Production_Budget  
) .transform_aggregate(  
 mean_revenue='mean(Revenue)',  
 groupby=['Major_Genre']  
) .mark_bar().encode(  
 alt.Y('Major_Genre:N'),  
 alt.X('mean_revenue:Q')  
)`

# Calculate Transform

transform\_calculate()

condition ? expression\_1 (if true) : expression\_2 (if false)

- Color movies by whether they are cheap or expensive in the Budget vs. Genre scatterplot



# Calculate Transform Answer

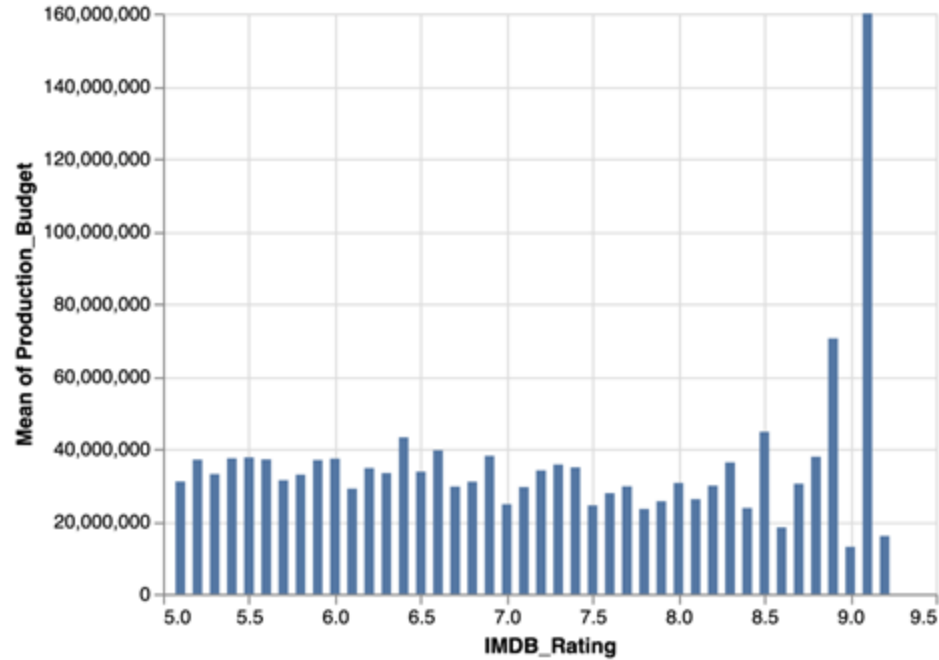
- Color movies by whether they are cheap or expensive in the Budget vs. Genre scatterplot

```
o alt.Chart(movies).transform_calculate(
 # uses the javascript shorthand to write the condition
 # condition ? expression_1 (if true) : expression_2 (if false)
 # if production budget>200000000 then 'expensive', else 'cheap'
 expensiveOrCheap="datum.Production_Budget > 200000000 ?
 'expensive' : 'cheap'
)
.mark_point().encode(
 alt.X('Major_Genre:N'),
 alt.Y('Production_Budget:Q'),
 alt.Color('expensiveOrCheap:N')
)
```

# Filter Transform

`transform_filter()`

- What's the mean production budget for movies with more than # 500 votes and a rating > 5



# Filter Transform Answer

- What's the mean production budget for movies with more than # 500 votes and a rating > 5

```
○ alt.Chart(movies).transform_filter(
 # & for and, | of or, > < = !=
 (alt.datum.IMDB_Votes >= 500) &
 (alt.datum.IMDB_Rating > 5)
) .mark_bar().encode(
 alt.X('IMDB_Rating:Q'),
 alt.Y('mean(Production_Budget)')
)
```

# Filter Transform Answer

- What's the mean production budget for movies with more than # 500 votes and a rating > 5 (an alternative way)

```
○ alt.Chart(movies).transform_filter(
 # & for and, | of or, > < = !=
 # javascript way
 "datum.IMDB_Votes >= 500 & datum.IMDB_Rating > 5"
) .mark_bar().encode(
 alt.X('IMDB_Rating:Q'),
 alt.Y('mean(Production_Budget)')
)
```



# Filter Transform Answer

- What's the mean production budget for movies with more than # 500 votes and a rating > 5 (an alternative way)
  - ```
alt.Chart(movies).transform_filter(  
    # equal, lt, gt, lte, gte  
    alt.FieldGTEPredicate(field='IMDB_Votes', gte=500)  
)  
.transform_filter(  
    #can't put two filters in one transform.  
    alt.FieldGTPredicate(field='IMDB_Rating', gt=5)  
)  
.mark_bar().encode(  
    alt.X('IMDB_Rating:Q'),  
    alt.Y('mean(Production_Budget)')  
)
```

Useful Predicate

- **FieldOneOfPredicate:** evaluates whether a field is among a list of specified values.

```
alt.FieldOneOfPredicate(field='__',  
oneOf=['value1', 'value2'])
```

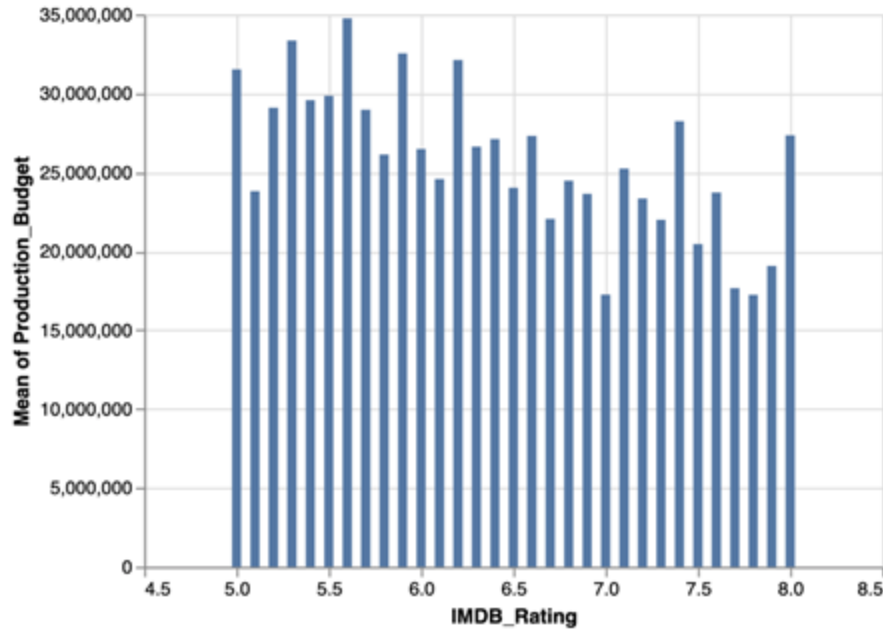
- **FieldRangePredicate:** evaluates whether a continuous field is within a range of values.

```
alt.FieldRangePredicate(field='__', range=[0, N])
```

Filter Transform

`transform_filter()`

- Create a bar chart includes only **Drama** and **Comedy** movies that have IMDB ratings between 5 and 7.



Filter Transform Answer

- Create a bar chart includes only Drama and Comedy movies that have ratings between 5 and 7.

```
○ alt.Chart(movies).transform_filter(  
    alt.FieldOneOfPredicate(field='Major_Genre',  
    oneOf=['Drama', 'Comedy'])  
).transform_filter(  
    alt.FieldRangePredicate(field='IMDB_Rating',  
    range=[5, 8])  
).mark_bar().encode(  
    alt.X('IMDB_Rating:Q'),  
    alt.Y('mean(Production_Budget)')  
)
```

Filter Transform Answer

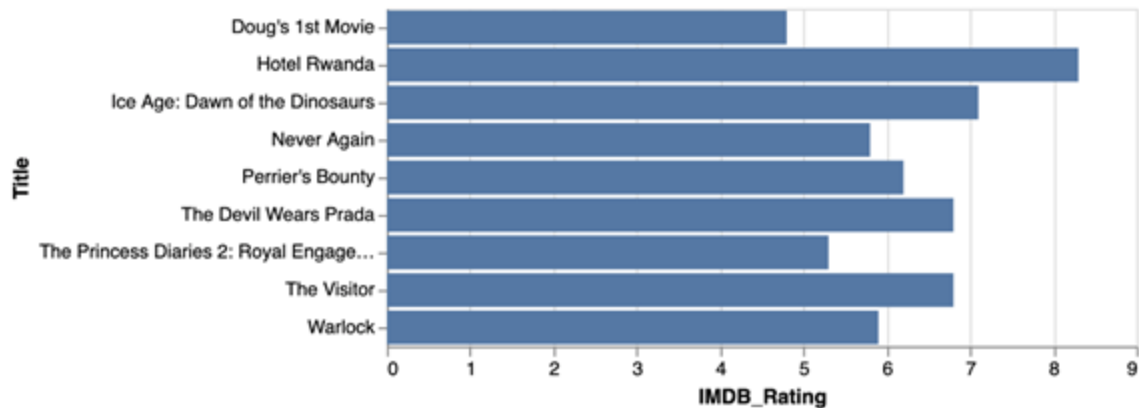
- Create a bar chart includes only Drama and Comedy movies that have ratings between 5 and 7. (json format, not recommended)

```
○ alt.Chart(movies).transform_filter(  
    {  
        "and": [  
            alt.FieldOneOfPredicate(field='Major_Genre',  
oneOf=['Drama', 'Comedy'])  
            ,  
            alt.FieldRangePredicate(field='IMDB_Rating',  
range=[5,7])  
        ]  
    }  
) .mark_bar().encode(  
    alt.X('IMDB_Rating:Q'),  
    alt.Y('mean(Production_Budget)')
```

Sample Transform

transform_sample()

```
○ alt.Chart(movies).mark_bar().encode(  
    alt.X('IMDB_Rating:Q'),  
    alt.Y('Title:N')  
) .transform_sample(10)
```



Window Transform

transform_sample()

- Window transformation calculates over sorted groups of data objects. These calculations include ranking, lead/lag analysis, and aggregates such as cumulative sums and averages.

Aggregate	Parameter	Description
row_number	None	Assigns each data object a consecutive row number, starting from 1.
rank	None	Assigns a rank order value to each data object in a window, starting from 1. Peer values are assigned the same rank. Subsequent rank scores incorporate the number of prior values. For example, if the first two values tie for rank 1, the third value is assigned rank 3.
dense_rank	None	Assigns dense rank order values to each data object in a window, starting from 1. Peer values are assigned the same rank. Subsequent rank scores do not incorporate the number of prior values. For example, if the first two values tie for rank 1, the third value is assigned rank 2.
percent_rank	None	Assigns a percentage rank order value to each data object in a window. The percent is calculated as $(\text{rank} - 1) / (\text{group_size} - 1)$.
cume_dist	None	Assigns a cumulative distribution value between 0 and 1 to each data object in a window.
ntile	Number	Assigns a quantile (e.g., percentile) value to each data object in a window. Accepts an integer parameter indicating the number of buckets to use (e.g., 100 for percentiles, 5 for quintiles).
lag	Number	Assigns a value from the data object that precedes the current object by a specified number of positions. If no such object exists, assigns <code>null</code> . Accepts an offset parameter (default: 1) that indicates the number of positions. This operation must have a corresponding entry in the fields parameter array.
lead	Number	Assigns a value from the data object that follows the current object by a specified number of positions. If no such object exists, assigns <code>null</code> . Accepts an offset parameter (default: 1) that indicates the number of positions. This operation must have a corresponding entry in the fields parameter array.

Window Transform

transform_sample()

- A bar chart of top 10 IMDB rating movies
 - ```
alt.Chart(movies,width=600).transform_window(
 sort=[alt.SortField('IMDB_Rating', order='descending')], # ordering
 the data
 imdb_rank='rank(*)' #creates a new column, decides rank relative to
 the list
).transform_filter(
 alt.datum.imdb_rank < 10
).mark_circle().encode(
 alt.X('imdb_rank:O'),
 alt.Y('Title:N',sort=alt.EncodingSortField(# need an additional
 sort field here because we are sorting based on titles
 field="imdb_rank",order="ascending"))
)
```



## Section 4

# Debugging Example

Please open the second document:

Debugging Example blank - the filled version will be released after lab



# Section 5

## Exporting from Altair

Please play around with the ipynb file on your own, and reach out for help if needed - we will learn this later in the course



Information  
changes  
everything.

# Thank You

