Information Visualization II

School of Information, University of Michigan

Week 2:

- Functions of interactivity

Assignment Overview

The objectives for this week are for you to:

- Understand the role of interaction in visualization
- Identify and understand various types of interaction in visualization
- Learn to implement interactive visualizations using Altair

The total score of this assignment will be

- Case study reflection: (30 points)
- Altair programming exercise (70 points)
- Extra credit (up to 10 points)

Resources:

- This article by FiveThirtyEight (https://fivethirtyeight.com). Available online (https://fivethirtyeight.com/features/women-in-comic-books/) (Hichey, 2014)
- Datasets from FiveThirtyEight, we have downloaded a subset of these datasets available in the folder for your use into ./assets (./assets)
    - The original dataset can be found on FiveThirtyEight Comic Characters (https://github.com/fivethirtyeight/data/tree/master/comic-characters)

Important notes:

1) Grading for this assignment is entirely done by manual inspection.

2) When turning in your PDF, please use the File -> Print -> Save as PDF option ***from your browser***. Do ***not*** use the File->Download as->PDF option. Complete instructions for this are under Resources in the Coursera page for this class.

If you're having trouble with printing, take a look at this video (https://youtu.be/PiO-K7AoWjk).

Part 1. Interactive visualization assesment (30 points)

Read the following article Timeless Songs (https://pudding.cool/2017/03/timeless/) on the Pudding's site (the music doesn't play anymore but you can find samples on Spotify and YouTube). Then answer the following questions:

1.1 Identify various interactions (15 points)

For the five visualizations:

- What's Remembered from the 90s
- Biggie or Tupac
- Present-day Popularity of Five Decades of Music
- XXXX Tracks: Historic Billboard Performance vs. 2014 Spotify Plays
- The Long-term Future of Hits from 2013

Identify which of the 7 interaction types are implemented and how. You don't need a long description. A short sentence will do.

- What's Remembered from the 90s
  - select is implemented by hovering a mouse over the circle mark denoting an artist or band's song to support a viewer's ability to see the name of the artist/band, the song title, the year the song came out, and the plays of the song.
  - (re)encode is implemented by the 'find a track' search feature which blurs out the artists/bands and/or songs that were not a part of the search to support the viewer being able to quickly identify the artist/band and/or song of interest. The figure is re-encoded by color by a new binary variable of whether or not the band/artist or song is in the search.
- Biggie or Tupac
  - select is implemented by hovering a mouse over the circle mark denoting a rapper's song to support a viewer's ability to see the name of the rapper, the song title, the year the song came out, and the plays of the song.

    - (re)encode is implemented by the 'find a rapper' search feature which sets to black the rapper and/or song that was not a part of the search to support the viewer being able to quickly identify the rapper and/or song of interest. The figure is re-encoded by color by a new binary variable of whether or not the rapper or song is in the search (black for not in the search, red for included). (re)encode is also implemented by the "ALL RAPPERS", "JUST BIGGIE AND TUPAC", and "JUST JAY-Z" search buttons which again set to black the rappers that were not a part of the condition to support the viewer being able to quickly identify the rapper of interest. The figure is re-encoded by color by a new binary variable of whether or not the rapper is in the condition (black for not in the condition, red for included).

- Present-day Popularity of Five Decades of Music
    - (re)configure is implemented by, regardless of filtering, having the spotify playcount always be sorted highest to lowest to support a viewer's ability to quickly determine the current popularity of artists and songs across decades.
    - filter is implemented by the "FInd an artist" search and the ALL Decade, 50s, 60s, etc. buttons to support a viewer's ability to focus on a particular decade or artist of interest to assess their current popularity.
    - select is implemented by clear labelling of the artist, song, and Spotify plays to support a viewer's ability to see this information quickly and effectively within the visualization and focus on songs or artists of interest.

- XXXX Tracks: Historic Billboard Performance vs. 2014 Spotify Plays
    - filter is implemented by changing the year in 'XXXX Tracks', which leads to the songs that the mark circles correspond to changing to that particular year. The 50s, 60s, etc. buttons also filter to the particular decade listed to support the viewer's ability to quickly assess how a song from a particular year or decade's popularity has lasted based on 2014 Spotify plays
    - (re)encode is implemented by the "Find a Track" search which blurs out the artists/bands and/or songs that were not a part of the search to support the viewer being able to quickly identify the song or artist of interest. The figure is re-encoded by color by a new binary variable of whether or not the band/artist or song is in the search. The Won a Grammy and Peaked at Number 1 buttons also re-encode the visualization as the songs that meet the condition are turned red and all others remain black again to support a viewer's ability to quickly assess a song of interest and the accolades it may have received. The figure is again re-encoded by color by a new binary variable of whether or not the song met the condition.
    - select is implemented by hovering a mouse over the circle mark denoting a song to support a viewer's ability to see the name of the artist, the song title, the year the song came out, and its position on the billboards for that particular year.

- The Long-term Future of Hits from 2013
    - filter is implemented by the "Find a Track" search which filters the listed songs, and the checked songs filter what appears in the visualization itself. This allows for a viewer to easily include the songs or artists of interest and plot their playcounts over time.
    - select is implemented by the line label denoting the song being displayed to support a viewer's ability to see this information and focus on particular songs of interest.

1.2 Critique (15 points)

For one of the five visualizations, critique the use of interaction. What works well? What could be better? You can add your own images here if it helps.

In reviewing the final visualization, "The Long-term Future of Hits from 2013", the encoding is straightforward but both effective and expressive. Time is encoded as the x-position, indexed Spotify plays as the y-position, and color encodes each individual song. The filtering functionality allows for a viewer to quickly search for songs of interest to plot in the visualization. As part of this, the visualization can be limited to just the filtered songs of interest, which allows for quick comparisons between songs and for the viewer to have a lot of control over what the visualization expresses and how effective and uncluttered it is.

In terms of improvements, while the visualization is both expressive and effective, the effectiveness can be improved. A tooltip-like functionality could be added to the visualization to denote the song name, the time period, and the indexed Spotify plays at a given point in the visualization. The x-axis is labelled quarterly and the y-axis has no tick marks which makes exact interpretations at a given point in time difficult. A tooltip encoding would assist greatly and fix this issue and allow for more accurate interpretations of the visualization. Additionally, as opposed to just including a long list of Artist - Song combinations, it likely would have been an enhancement to include some Elaborate interactive functionality. For example, a viewer could search by decade (50s, 60s, 70s, etc.), which would drill down into the artists of this time period and then could further drill into other categories like music type, etc. This would be a much easier way to work through the data and would allow the user more control and the ability to more quickly search and find the songs and artists of interest.

Part 2. Programming exercise (70 points)

Start by reading the 538 article here (https://fivethirtyeight.com/features/women-in-comic-books/). What you should know is that there are two major comic book companies: DC (Batman, Superman, Wonder Woman, etc.) and Marvel (Black Widow, Iron Man, Hulk, etc.).

We have a dataset of characters, their sex, when they were introduced, if their identify is secret, their eye and hair color, the number of appearances, etc. Lots of dimensions on which to build our visualizations.

In [1]:
```
# start with the setup
import pandas as pd
import numpy as np
import altair as alt
```

In [2]:
```
# enable correct rendering
alt.renderers.enable('default')
```

Out[2]:
```
RendererRegistry.enable('default')
```

In [3]:
```
# uses intermediate json files to speed things up
alt.data_transformers.enable('json')

# use the 538 theme
alt.themes.enable('fivethirtyeight')
```

Out[3]:
```
ThemeRegistry.enable('fivethirtyeight')
```

In [4]:
```
def createComicFrame(DCDataFile='assets/dc-wikia-data.csv',marvelDatafile='assets/marvel-wikia-data.csv'):
    # load up the two datasets, one for Marvel and one for DC
    dc = pd.read_csv(DCDataFile)
    marvel = pd.read_csv(marvelDatafile)

    # label the publisher
    dc['publisher'] = 'DC'
    marvel['publisher'] = 'Marvel'

    # rename some columns
    marvel.rename(columns={'Year': 'YEAR'}, inplace=True)

    # create the concatenated table with everything
    comic = pd.concat([dc, marvel])

    # drop years with na values
    comic.dropna(subset=['YEAR'], inplace=True)

    return(comic)
```

In [5]:
```
# let's get the comic data:

comic = createComicFrame()
```

In [6]:
```
# let's look inside
comic.sample(5)

# this next line sub-samples the data if you want to experiment with
# a smaller dataset. This should only be used for testing. and should
# be commented back in after (otherwise your results won't match the
# images)
# comic = comic[comic.index % 5 == 0]
```

Out[6]:

	page_id	name	urlslug	ID	ALIGN	EYE	HAIR	SEX	GSM	ALIVE	APPEARAN
16163	522132	Rot's Creations (Earth-616)	\/Rot%27s_Creations_(Earth-616)	NaN	Bad Characters	NaN	No Hair	NaN	NaN	Living Characters	
3561	1200	Amanda Mueller (Earth-616)	\/Amanda_Mueller_(Earth-616)	Secret Identity	Bad Characters	Brown Eyes	White Hair	Female Characters	NaN	Deceased Characters	
4204	85958	Jefferson Skeevers (New Earth)	\/wiki\/Jefferson_Skeevers_(New_Earth)	Public Identity	Bad Characters	Brown Eyes	Black Hair	Male Characters	NaN	Living Characters	
7471	284330	Harry Palmer (Earth-616)	\/Harry_Palmer_(Earth-616)	NaN	Bad Characters	NaN	Blond Hair	Male Characters	NaN	Deceased Characters	
15532	62353	Juliette D'Angelo (Earth-616)	\/Juliette_D%27Angelo_(Earth-616)	NaN	NaN	NaN	NaN	Female Characters	NaN	Living Characters	

Comic Books Are Still Made By Men, For Men And About Men

Original article available at [FiveThirtyEight (https://fivethirtyeight.com/features/women-in-comic-books/)](https://fivethirtyeight.com/features/women-in-comic-books/)

By [Walt Hickey (https://fivethirtyeight.com/contributors/walt-hickey/)](https://fivethirtyeight.com/contributors/walt-hickey/)

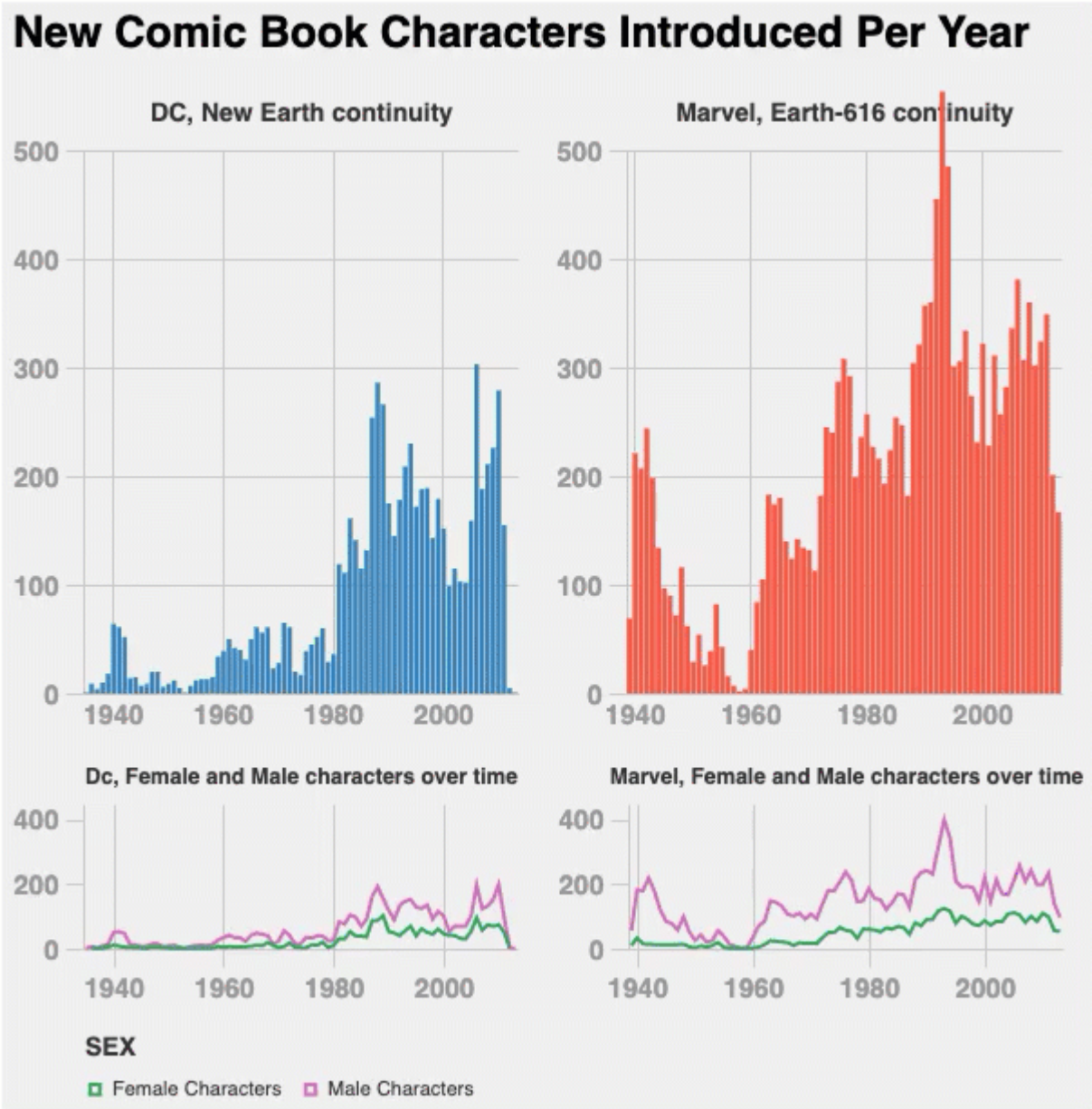
Get the data on GitHub (https://github.com/fivethirtyeight/data/tree/master/comic-characters)

We are going to be revising and adding to the visualizations for this article. While they're nice, we think we can do better by adding some interactivity.

Because many of these visualizations are interactive, we will be recording short clips to demonstrate the desired behavior of the systems. Unlike some of your previous assignments, we will give you portions of the Altair code and ask you to complete the interactive elements.

Problem 2.1 (35 Points)

We'd like to build an interactive visualization that allows us to compare the distributions of characters over time as well. The top two charts will represent the total characters over time (as bar charts). The bottom two will be a line chart with separate lines for female and male characters.



As ranges are selected or moved in the top charts, the bottom charts will automatically update (and the selection will be visible). Selection in one of the top two will also cause the appropriate data to be selected in the other (i.e., select on the left, and the right gets changed (highlighting); select on the right, and the left changes).

You can watch a narrated version of the visualization here (https://www.youtube.com/watch?v=NKCk97yBlJM)

In [7]:

```
def genStaticBars(comicDF):
    # input: comicDF -- dataframe of characters as described above (e.g., comic)
    # return: a static version of the visualization pieces described above

    # we're going to generate the 2 visualizations at once to make things concise, but you might break this up
    # into sub-functions in a different implementation

    # we're largely going to use the same "base" visualization here for the bar. We can start with a bar
    # chart and then change the details. The Y axis will be the count()
    p1_bar_base = alt.Chart(comicDF).mark_bar(size=2.5).encode(
        alt.Y('count():Q',
              axis=alt.Axis(values=[0, 100, 200, 300, 400, 500],
                            title=None,
                            labelFontWeight="bold",
                            labelFontSize=15),
              scale=alt.Scale(domain=[0, 500]))).properties(
                    width=240,
                    height=300
    )



    # let's create the bar chart for DC. We'll take the "base" chart
    bar_dc = p1_bar_base.encode(alt.X('YEAR:N',  # create the X axis based on year and fix the look of the axes
                                      axis=alt.Axis(values=[1940, 1960, 1980, 2000], labels=True, ticks=False,grid=
                                                    title="DC, New Earth continuity",
                                                    titlePadding=-347,
                                                    labelAngle=360,
                                                    labelFontWeight="bold",
                                                    labelFontSize=15,)),
            ).transform_filter(
                # we will use Altair's filter to only keep DC for this chart
                alt.datum.publisher == 'DC'
            )

    # let's do the same thing for marvel
    bar_marvel = p1_bar_base.mark_bar(color='#f6573f').encode(alt.X('YEAR:N', # create the X axis based on year
                                    # fix the look of the axes
                                   axis=alt.Axis(values=[1940, 1960, 1980, 2000], labels=True, ticks=False,grid=True
                                                 title="Marvel, Earth-616 continuity",
                                                 titlePadding=-347,
                                                 labelAngle=360,
```

```
                                        labelFontWeight="bold",
                                        labelFontSize=15)),
            ).transform_filter(
                # we will use Altair's filter to only keep DC for this chart
                alt.datum.publisher == 'Marvel'
            )

    return(bar_dc,bar_marvel)
```

In [8]:

```
def genStaticLines(comicDF):
    # input: comicDF -- dataframe of characters as described above (e.g., comic)
    # return: a static version of the visualization pieces described above

    # we're going to generate the 2 visualizations at once to make things concise, but you might break this up
    # into sub-functions in a different implementation

    # let's create a new "base" chart for the two line charts. We'll take the bar chart base above
    # and modify it to use a line chart

    p1_line_base = alt.Chart(comicDF).mark_line().encode(
         # the X axis will be year
         alt.X('YEAR:N'),
         # the Y axis will be the count (the number of points that year)
         alt.Y('count():Q', axis=alt.Axis(grid=False,
                                          labelFontWeight="bold",
                                          labelFontSize=15,
                                          title=None)),
         # let's split the data and color by SEX
         alt.Color('SEX',
                   scale = alt.Scale(domain=['Female Characters', 'Male Characters'], range=['#31a354', '#ce6dbd'
                   legend=alt.Legend(orient="bottom"))
        ).properties(
                    width=240, height=80
         )


    line_dc = p1_line_base.encode(alt.X('YEAR:N',
                                        axis=alt.Axis(values=[1940, 1960, 1980, 2000],
                                                      grid=True,
                                                      labelAngle=360,
                                                      labelFontWeight="bold",
                                                      labelFontSize=15,
                                                      title = 'Dc, Female and Male characters over t
                                                      titlePadding=-130,
                                                      titleFontSize = 12
                                                     )
                                       )
                ).transform_filter(
                    # this is the DC line chart, so we only want DC
                    alt.datum.publisher == 'DC'
                )
```

```
    line_marvel = p1_line_base.encode(alt.X('YEAR:N',
                                            axis=alt.Axis(values=[1940, 1960, 1980, 2000],
                                                          grid=True,
                                                          labelAngle=360,
                                                          labelFontWeight="bold",
                                                          labelFontSize=15,
                                                          title = 'Marvel, Female and Male characters over t
                                                          titlePadding=-130,
                                                          titleFontSize = 12
                                                         )
                                            )
                      ).transform_filter(
                          # this is the Marvel line chart, so we only want Marvel
                          alt.datum.publisher == 'Marvel'
                      )



    return(line_dc,line_marvel)
```

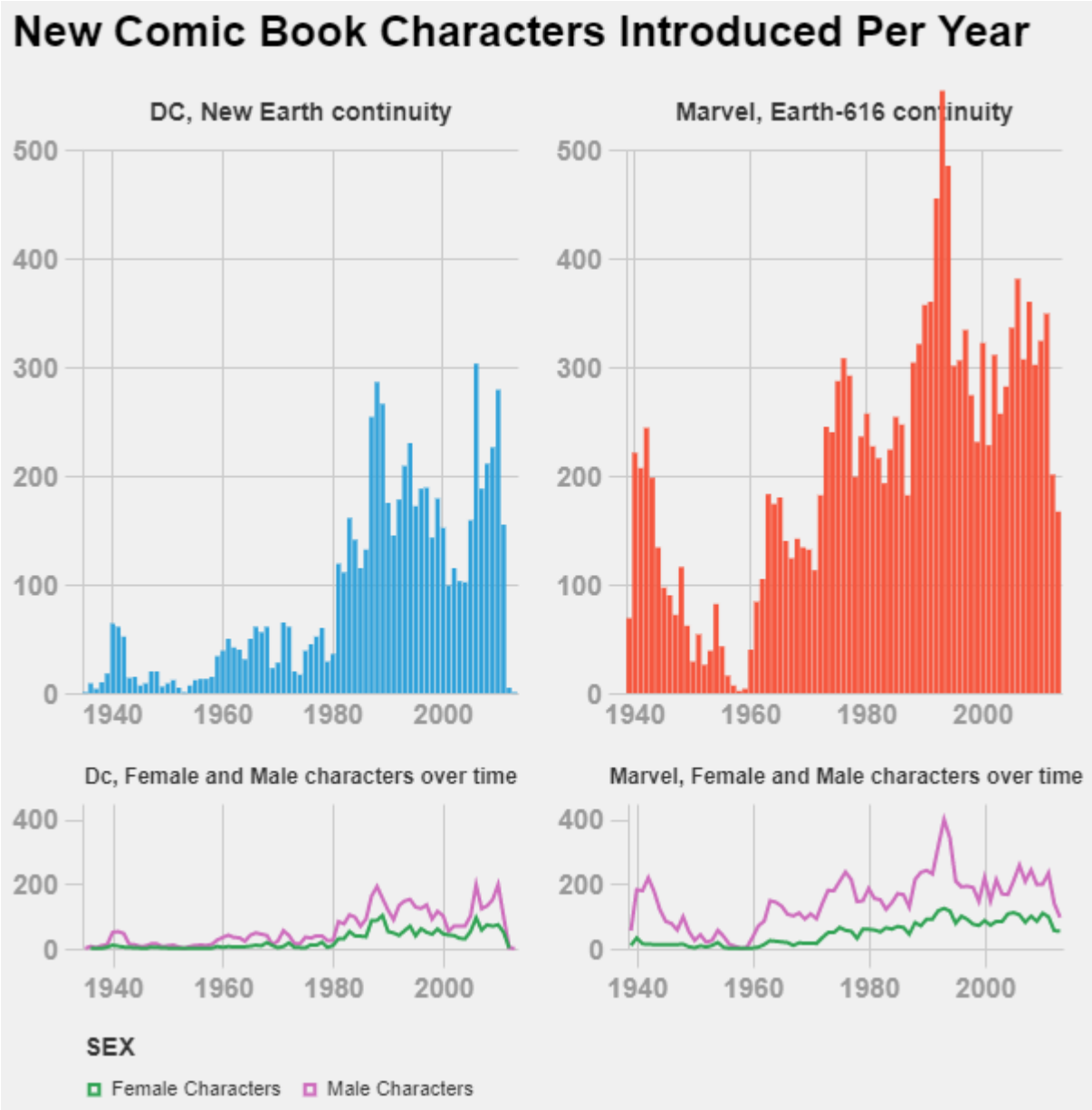
In [9]:
```
# we're going to get all the little subpiece visualizations

# the top two bar charts
bar_dc, bar_marvel = genStaticBars(comic)

# and the bottom two line charts
line_dc, line_marvel = genStaticLines(comic)
```

In [10]:

```
# let's make a function that puts it all together
def genIntroVis(b_dc, b_marvel, l_dc, l_marvel):
    # let's put everything together

    # top piece
    top_charts = alt.hconcat(b_dc,b_marvel).resolve_scale(y='shared'
                ).properties(
                        title='New Comic Book Characters Introduced Per Year'
                )

    # bottom piece
    bottom_charts = alt.hconcat(l_dc,l_marvel).resolve_scale(y='shared')

    return alt.vconcat(top_charts,bottom_charts).configure_view(
        strokeWidth=0
    )
```

In [11]:
```
# and now that we have them, let's put them together
genIntroVis(bar_dc, bar_marvel, line_dc, line_marvel)
```

Out[11]:



Now we have the chart we need, but here is where you have to start doing some work. For this problem, we'll do this a little bit at a time.

Problem 2.1.1

First, modify the code below to create a "brush" object (a "selection" in Altair speak) that will let us select a time range. For all these, you should take a look at the examples on this page (https://altair-viz.github.io/user_guide/interactions.html) to identify the right (and the lab).

In [12]:
```
# modify this cell to create the brush object
brush = alt.selection_interval(encodings=['x'])

#raise NotImplementedError()
```

Problem 2.1.2

The next step is to create the condition for the DC chart. Look at the documentation for the condition. We specifically want things selected by the "brush" object to stay the same color (#2182bd) and the unselected content to turn gray.

In [13]:
```
# modify this cell to create the brush object
colorConditionDC = alt.condition(brush, alt.value('#2182bd'),alt.value('gray'))

#raise NotImplementedError()
```

Problem 2.1.3

Finally, we need to add both the condition and selection to the `bar_dc` chart. We'll call this new chart `i_bar_dc` (i for interactive). Remember that you can "override" or modify a chart by simply taking the original chart and adding an encode or some other function to it. For example the line:

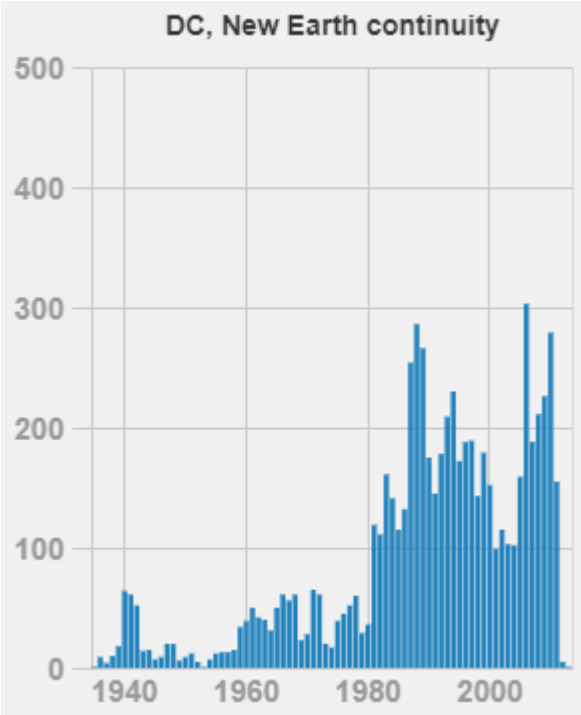
```
i_bar_dc = bar_dc.encode(color = 'TEST')
```

will take the original chart with all its original settings and make the color encoding based on the TEST column (which doesn't exist in this case). If there was a color encoding in `bar_dc`, it will be overridden by TEST. If there wasn't one, it will be added.

In [14]:
```
# modify this cell to create the brush object
i_bar_dc = bar_dc.add_selection(brush) \
                 .encode(color=colorConditionDC)

#raise NotImplementedError()
```

In [15]:
```
# if you did the last step correctly, you should be able to see the selection work for the DC bar chart
i_bar_dc
```

Out[15]:



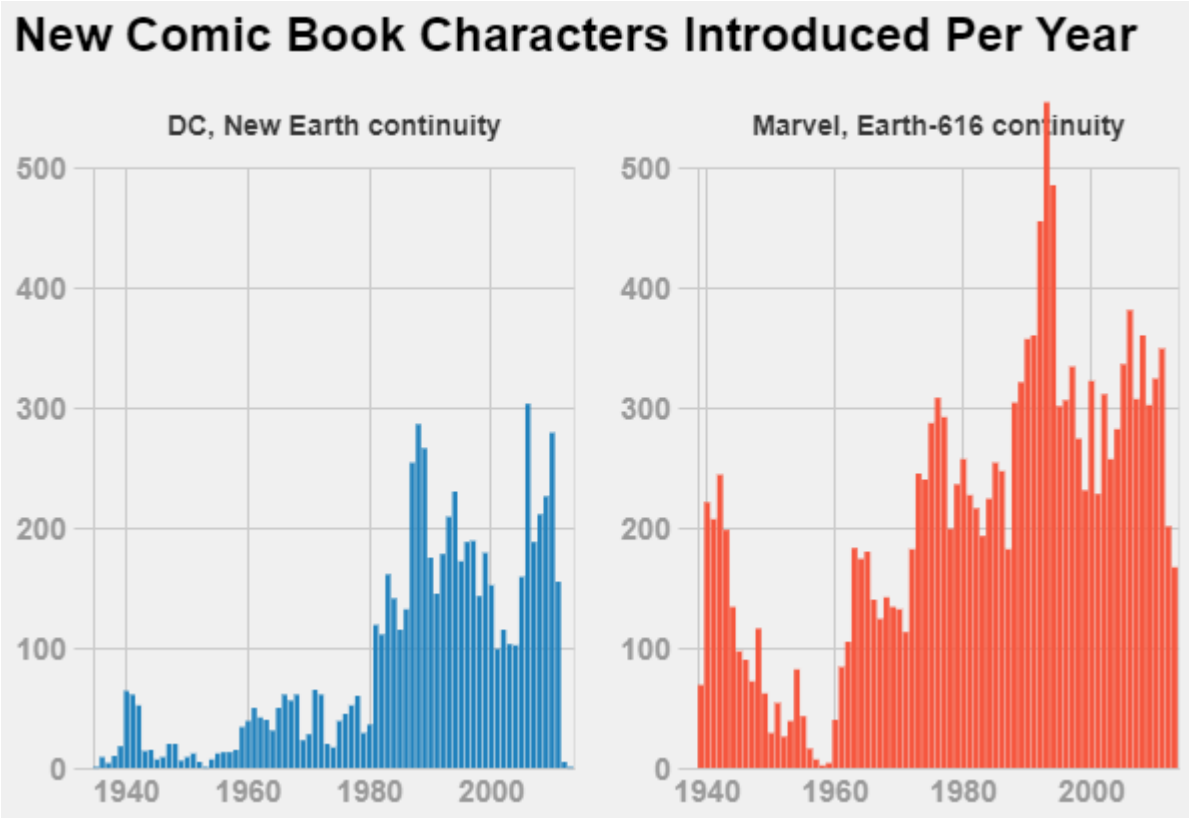
Problem 2.1.4

Do the same thing for the marvel chart. Create the color condition for marvel (selected should be #f6573f, unselected should be gray). Then add the brush and condition to the `bar_marvel` to create `i_bar_marvel`

In [16]:
```
# modify the following two lines
colorConditionMarvel = alt.condition(brush, alt.value('#f6573f'),alt.value('gray'))
i_bar_marvel = bar_marvel.add_selection(brush) \
                         .encode(color=colorConditionMarvel)

#raise NotImplementedError()
```

In [17]:
```
# top piece
top_charts = alt.hconcat(i_bar_dc,i_bar_marvel).resolve_scale(y='shared'
           ).properties(
                    title='New Comic Book Characters Introduced Per Year'
           )

# if you did the two bar charts correctly, you should now be able to interactively select
# (and the selections should be linked)
top_charts
```

Out[17]:



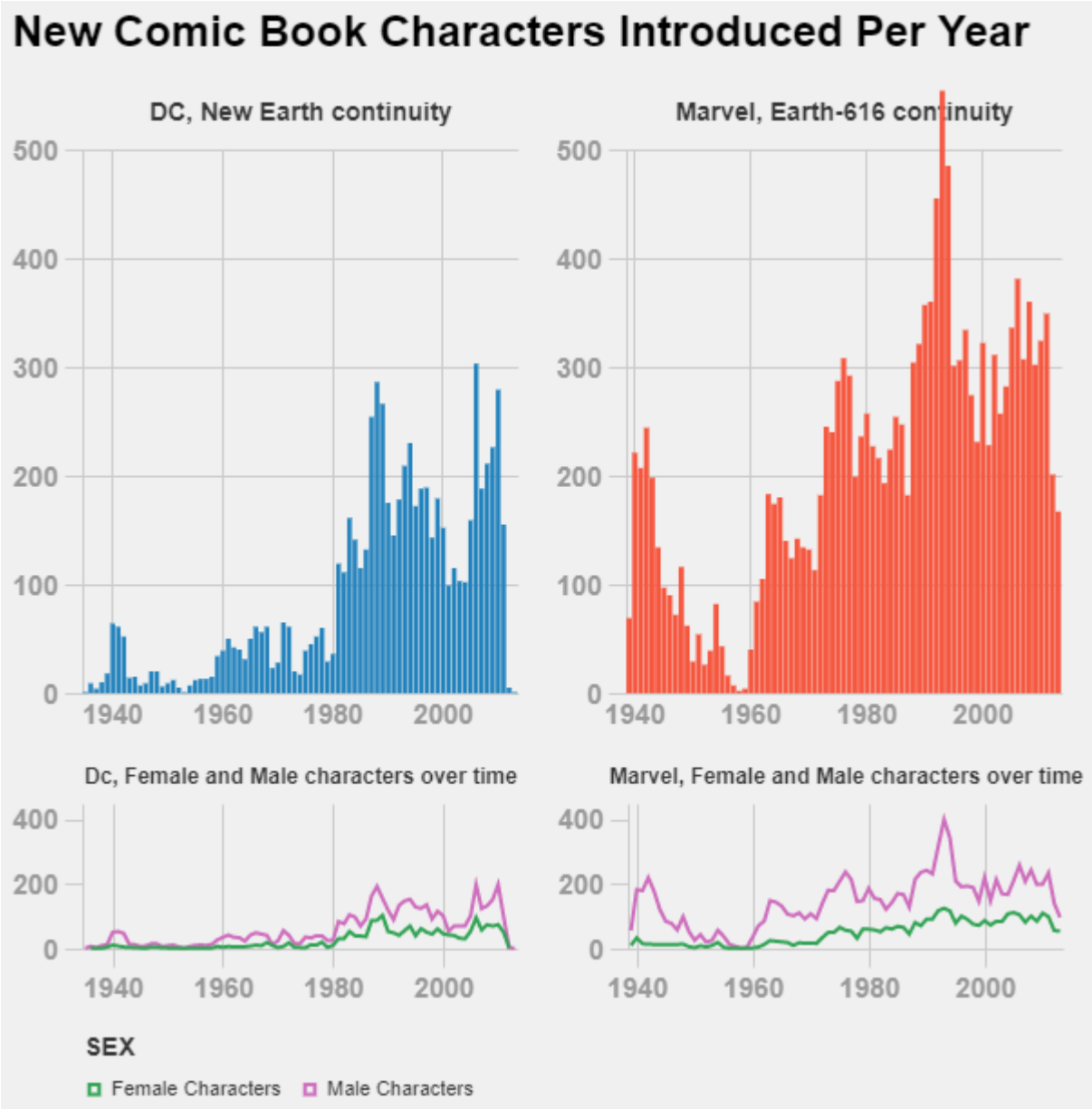
Problem 2.1.5

The last step is to modify the two line charts. Again, you'll want to start with `line_dc` and `line_marvel` to create the new charts.

In [18]:
```
# modify the code below
i_line_dc = line_dc.transform_filter(brush)
i_line_marvel = line_marvel.transform_filter(brush)

#raise NotImplementedError()
```

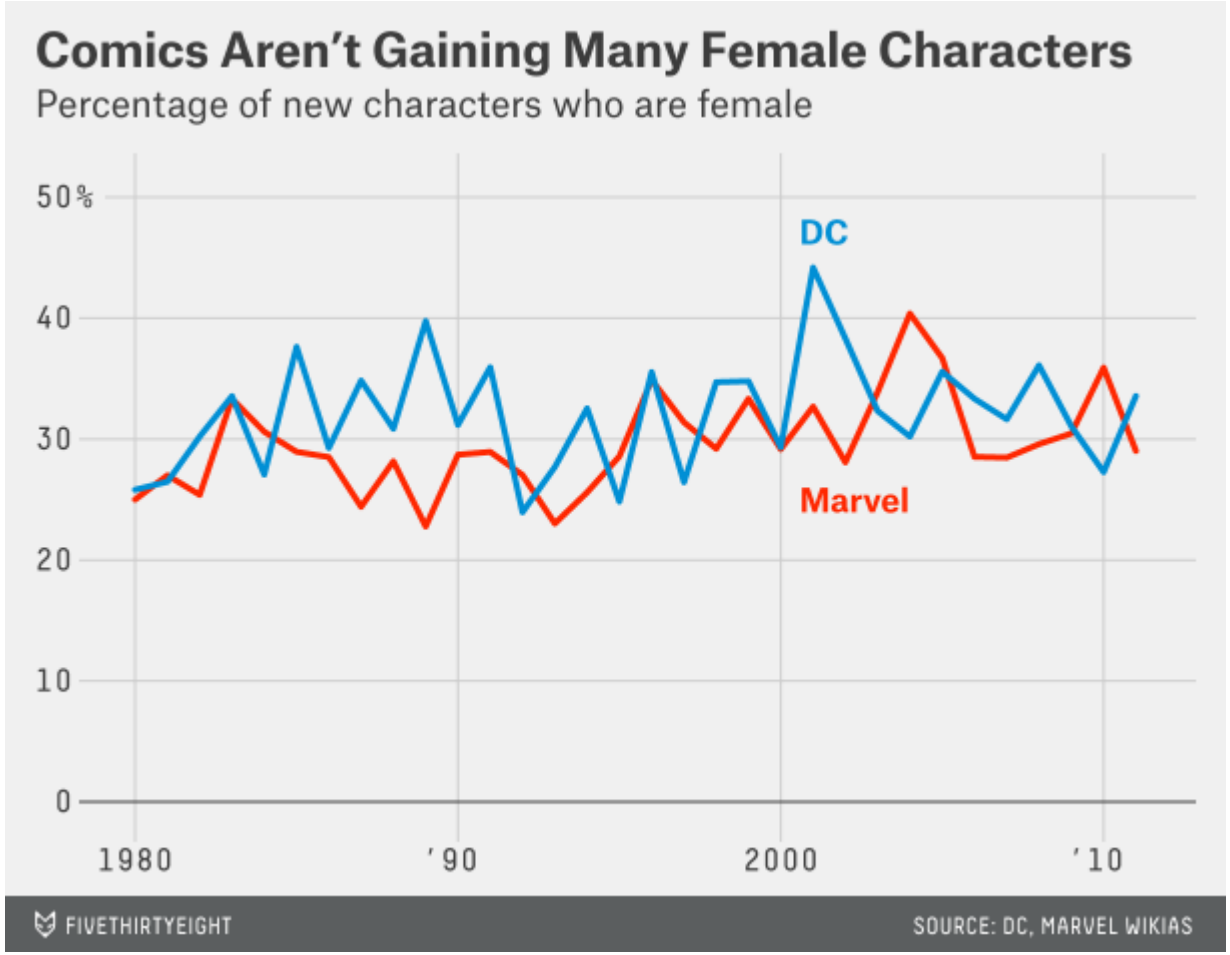
In [19]:
```
# let's put everything together with your new interactive charts. If you did everything correctly this part
# should generate the visualizations we want

# the same function we wrote above should work for the new subplots
genIntroVis(i_bar_dc, i_bar_marvel, i_line_dc, i_line_marvel)
```

Out[19]:



Problem 2.2 (35 Points)

One of the issues discussed in the article is that the comics aren't gaining many female characters.



This visualization is ok, but we can enhance it with some interactivity. Let's start by dealing with the fact that the chart only presents one interesting: Percent female in any given year. It might help us understand the claim that there's a relatively trending change in this percent by plotting year-over-year percent changes. Also, it's possible that there are more characters being introduced in later years. So even one or two good years in the 2000's may make up for lots of bad years in the past (it turns out that this is not the case, but it is a question we might ask).

We're going to create the table with all the necessary statistics for you next:

In [20]:
```
def generatePercentTable(comicDF, publisher):
    # input: comicDF -- dataframe of characters as described above (e.g., comic)
    # input: publisher -- a string, either DC or Marvel
    # return: a procesed percent table

    _df = comicDF[comicDF.publisher == publisher]
    _df = _df[['SEX','YEAR']]
    _df = pd.get_dummies(_df)
    _df.YEAR = _df.YEAR.astype('int')
    _df = _df.groupby(['YEAR']).sum()

    _df['total'] = 0
    _df['total'] = _df['total'].astype('int')
    for col in list(comicDF[comicDF.publisher == publisher].SEX.unique()):
        col = str(col)
        if (col != 'nan'):
            _df['total'] = _df['total'].astype('int') + _df["SEX_"+col].astype('int')

    _df['% Female'] = _df['SEX_Female Characters'] / _df.total
    _df = _df.reset_index()
    _df = _df[['YEAR','% Female','SEX_Female Characters','SEX_Male Characters','total']]
    _df['publisher'] = publisher
    _df = _df[(_df.YEAR >= 1979)]
    _df['Year-over-year change in % Female'] = _df['% Female'].pct_change()
    toret = _df[(_df.YEAR > 1980) & (_df.YEAR < 2013)].copy()
    t2 = toret.cumsum()
    toret['% Female characters to date'] = list(t2['SEX_Female Characters'] / t2['total'])
    return(toret)
```

In [21]:
```
# let's create the table we need

# first, make the percent table for Marvel and DC and concat them
changedata = pd.concat([generatePercentTable(comic,"Marvel"),generatePercentTable(comic,"DC")])

# now, let's restructure it to the right long form
changedata = pd.melt(changedata,id_vars=['YEAR','publisher'],value_vars=['% Female',
                                                                         'Year-over-year change in % Female',
                                                                         '% Female characters to date'])
```

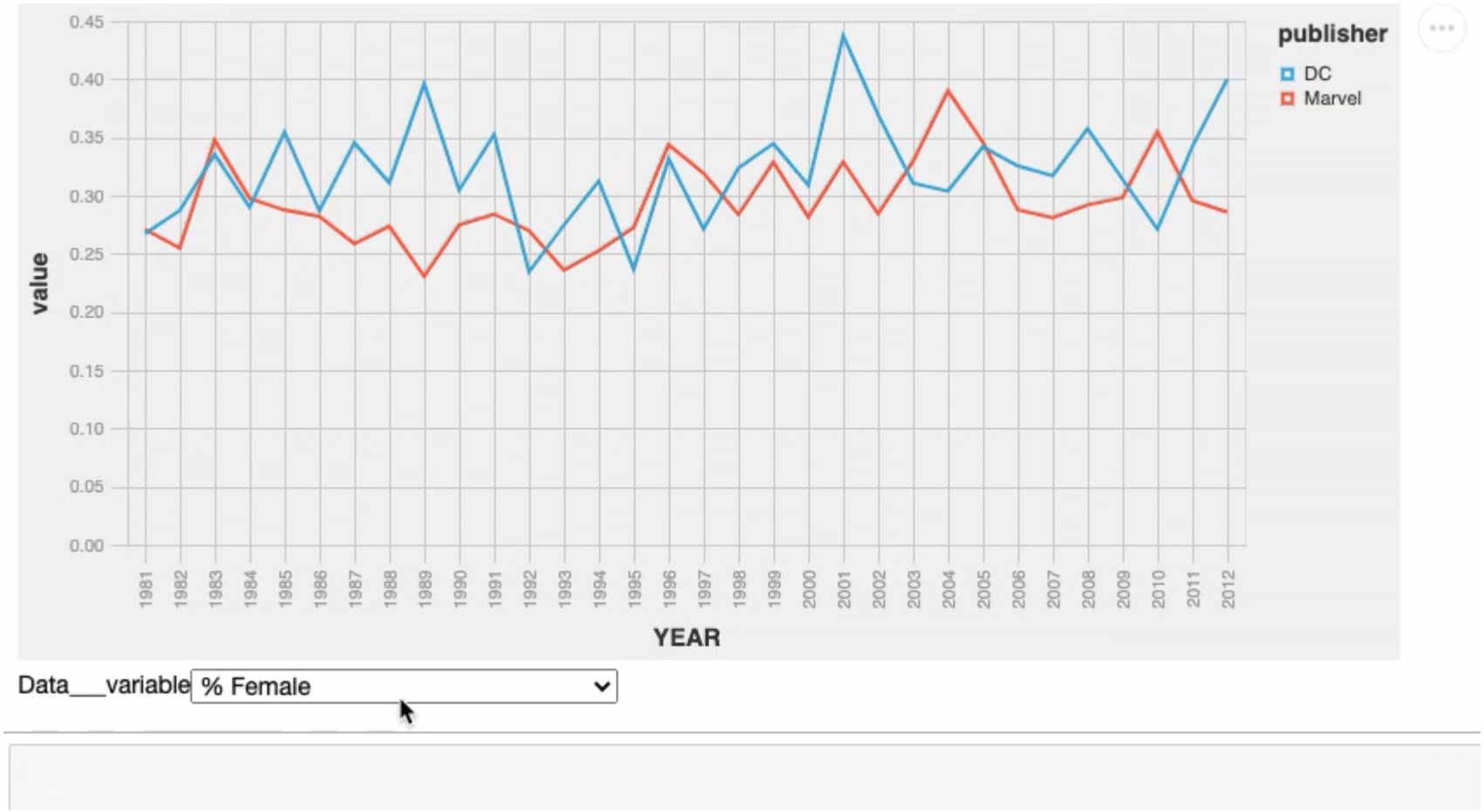
In [22]:
```
# let's see what's inside
changedata.sample(5)
```

Out[22]:

	YEAR	publisher	variable	value
13	1994	Marvel	% Female	0.252700
60	2009	DC	% Female	0.313901
158	2011	Marvel	% Female characters to date	0.291694
51	2000	DC	% Female	0.308725
182	2003	DC	% Female characters to date	0.316178

Problem 2.2.1

Your first job will be to create an interactive chart that has a drop-down box that allows us to select the variable of interest. Here's our target in action:



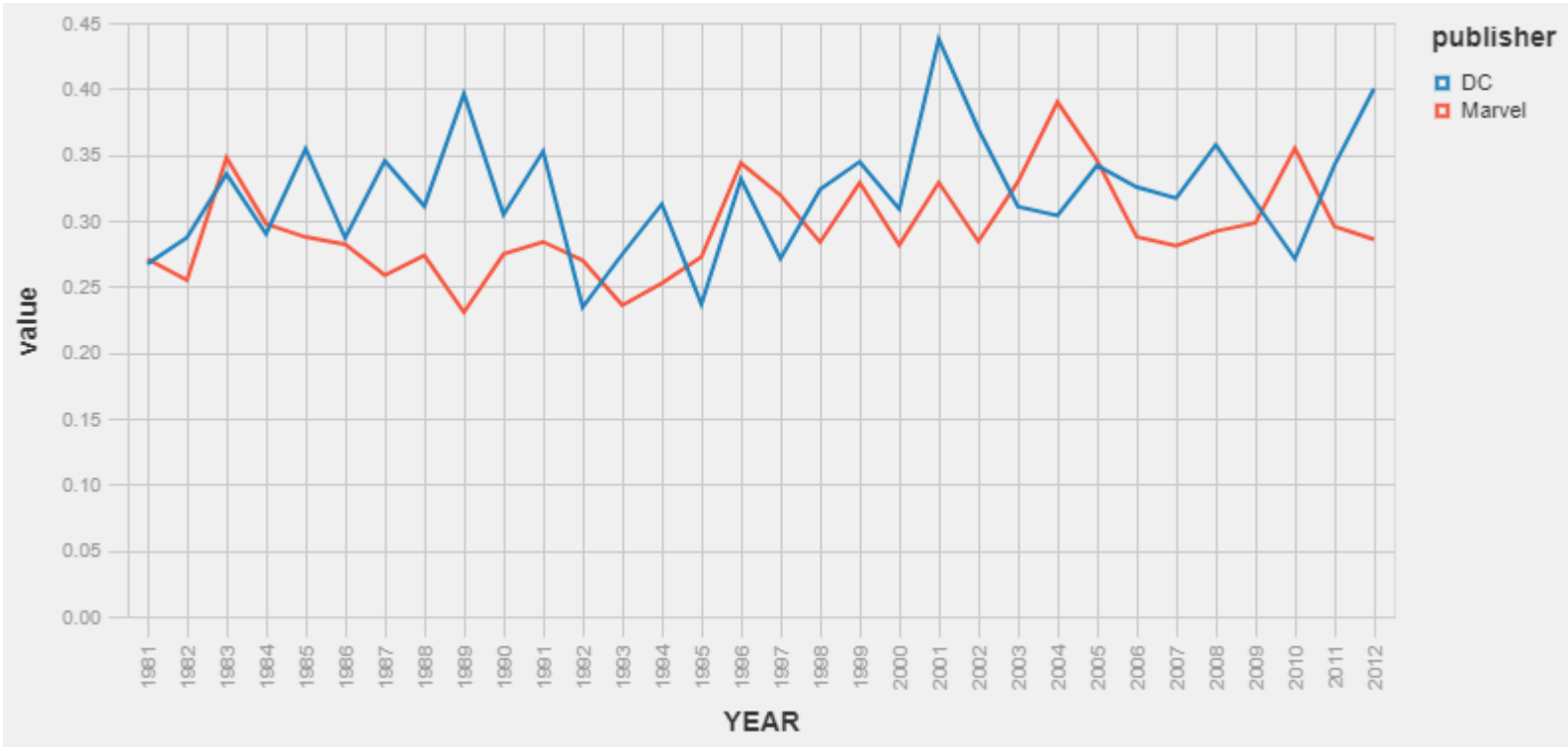
Modify `generateLineChartP21` below to generate this chart. If you haven't already, you'll want to take a look at the binding_select examples. Make sure you can get the chart working without interactivity first (hint: see if you can figure out how to filter to specific variables of interest).

In [23]:

```
def generateLineChartP21(changeDF):
    # input: changeDF -- the data frame, formatted as changedata above
    # return: an altar chart as described above

    metricOptions = ['% Female','Year-over-year change in % Female','% Female characters to date']
    input_dropdown = alt.binding_select(options= metricOptions, name="Data___variable: ")

    dropdown_selection = alt.selection_single(fields = ['variable'], init={'variable':'% Female'}, bind = input_

    line = alt.Chart(changeDF) \
              .mark_line() \
              .encode(
                     x = alt.X('YEAR:N'),
                     y = alt.Y('value:Q'),
                     color  = alt.Color('publisher',
                                        scale = alt.Scale(domain=['DC', 'Marvel'],
                                                          range=['#2182bd', '#f6573f'])
                                       )
                     ) \
              .add_selection(dropdown_selection) \
              .transform_filter(dropdown_selection)

    #raise NotImplementedError()
    return(line)
```

In [24]:
```
# if you did everything correctly, this should generate the visualization:
generateLineChartP21(changedata)
```

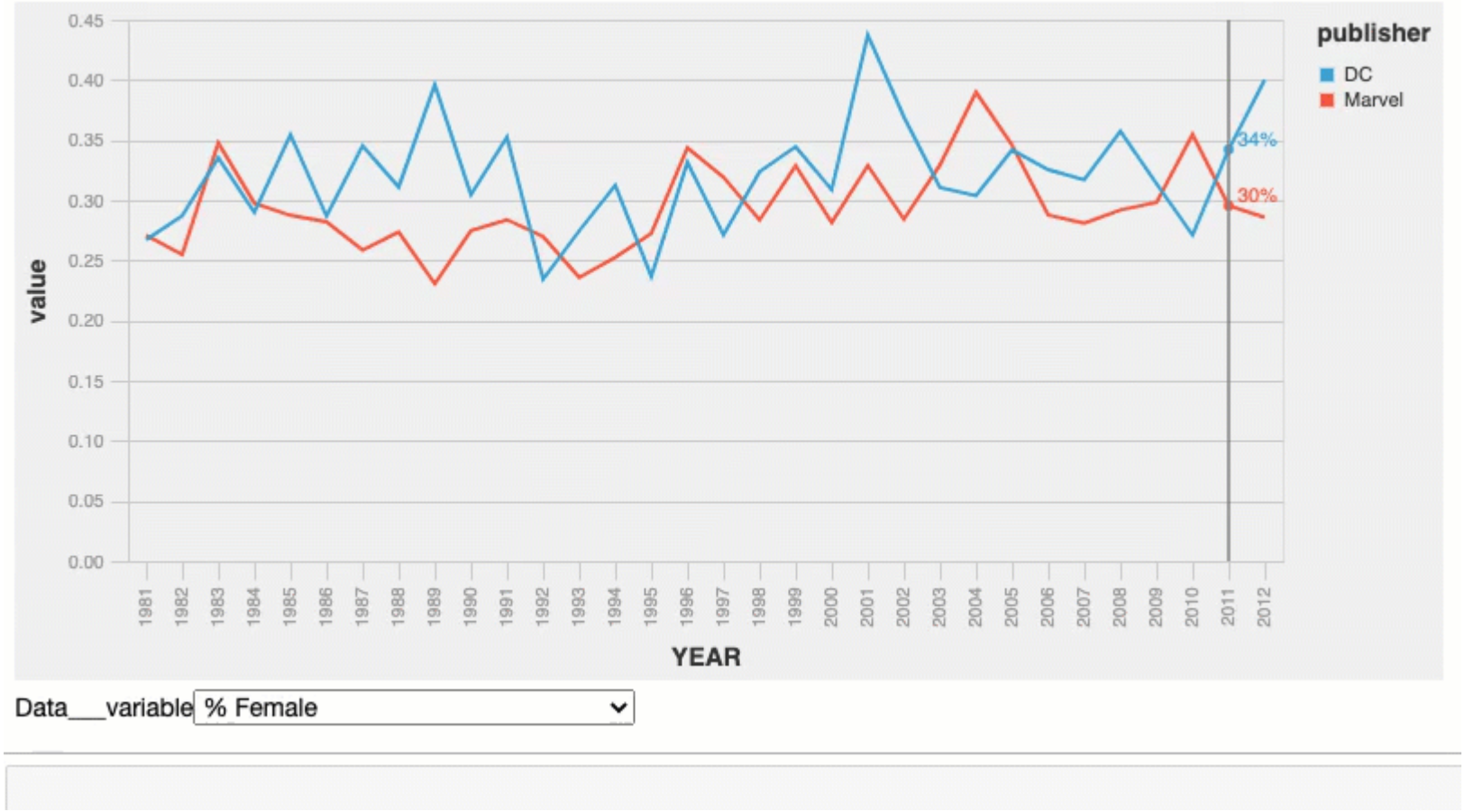
Out[24]:



Data___variable: % Female

Problem 2.2.2

The next thing we're going to do is modify this example to give us a useful line that gives us the actual values (an effectiveness boost if we want to know the numbers). Here's an example:



Notice that the dropdown functionality still works. Your task is to build `generateLineChartP22` below to return this modified line chart. The good news is there's an example that's really close to what you need (https://altair-viz.github.io/gallery/multiline_tooltip.html). But you'll need to understand what's going on and modify it.

Some hints:

- You probably want to copy your code for `generateLineChartP21` into the new function. There are pieces of code you defined (e.g., the selection) that you'll need to use again.

- The example relies a lot on overloading Altair charts from a common base (e.g., `line = alt.Chart(...` and then `newline = line.encode...` so newline overloads/extends line). Our experience is that it's easy to get errors when doing this here because you'll be using multiple selections and conditions (another hint). We recommend defining the Altair charts (selectors, points, etc.) from scratch. It's more repeated code, but it'll save you the same headaches.

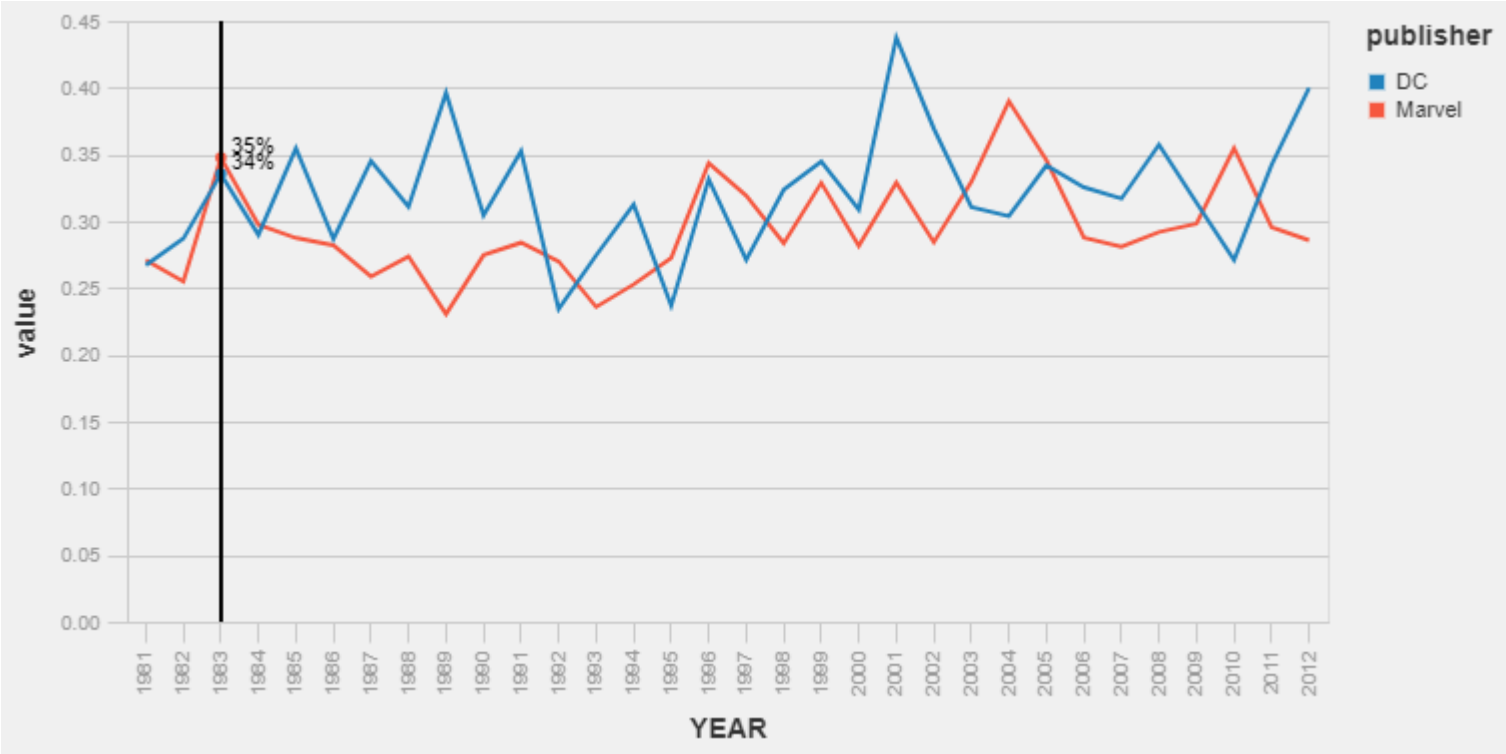
In [25]:

```
def generateLineChartP22(changeDF):
    # input: changeDF -- the data frame, formatted as changedata above
    # return: an altar chart as described above

    nearest = alt.selection(type='single', nearest=True, on='mouseover', fields=['YEAR'], empty='none')

    metricOptions = ['% Female','Year-over-year change in % Female','% Female characters to date']
    input_dropdown = alt.binding_select(options= metricOptions, name="Data___variable: ")
    dropdown_selection = alt.selection_single(fields = ['variable'], init={'variable':'% Female'}, bind = input_

    line = alt.Chart(changeDF) \
              .mark_line() \
              .encode(
                     x = alt.X('YEAR:N'),
                     y = alt.Y('value:Q'),
                     color  = alt.Color('publisher',
                                        scale = alt.Scale(domain=['DC', 'Marvel'],
                                                          range=['#2182bd', '#f6573f'])
                                       )
                     ) \
              .add_selection(dropdown_selection) \
              .transform_filter(dropdown_selection)

    selectors = alt.Chart(changeDF) \
                   .mark_point() \
                   .encode(
                            x='YEAR:N',
                            opacity=alt.value(0),
                           )

    points = alt.Chart(changeDF) \
              .mark_point() \
              .encode(
                     x = alt.X('YEAR:N'),
                     y = alt.Y('value:Q'),
                     color  = alt.Color('publisher',
                                        scale = alt.Scale(domain=['DC', 'Marvel'],
                                                          range=['#2182bd', '#f6573f'])
                                       ),
                     opacity=alt.condition(nearest, alt.value(1), alt.value(0))
                     ) \
              .add_selection(nearest) \
```

```
            .transform_filter(dropdown_selection)

    text = alt.Chart(changeDF) \
            .mark_text(align='left', dx=5, dy=-5) \
            .encode(
                    x = alt.X('YEAR:N'),
                    y = alt.Y('value:Q'),
                    text= alt.condition(nearest, alt.Text('value:Q', format = '.0%'), alt.value(' '))
                    ) \
            .transform_filter(dropdown_selection)

    # Draw a rule at the location of the selection
    rules = alt.Chart(changeDF) \
            .mark_rule() \
            .encode(
                    x = alt.X('YEAR:N')
                    ) \
            .transform_filter(nearest)

# Put the five layers into a chart and bind the data
    final_chart = alt.layer(line, selectors, points, rules, text) \
                    .properties(width=600, height=300)

    #raise NotImplementedError()
    return final_chart
```

In [26]:
```
# let's try it out
generateLineChartP22(changedata)
```

Out[26]:



Data___variable: % Female

Extra Credit (up to 10 points)

As an extra credit exercise, you can create a new interactive visualization that either replaces/extends one of the 538 examples OR invent a new one that fits with the article.

The interaction should be well thought out and appropriate (so just turning on `.interactive()` on a static chart won't really cut it). Please give us 1-2 sentences about what your interactivity adds.

In [27]:
```
# YOUR ANSWER HERE
```