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.

YOUR ANSWER HERE

- What's Remembered from the 90s *Select is implemented by hovering over selected individual points to show the artist, the song, the year and how many times it has been played on Spotify, and clicking selected individual points plays a bit of the song.
- Biggie or Tupac *Select is implemented by hovering over selected individual points to show the artist, the song, the year and how many times it has been played on Spotify.
- Present-day Popularity of Five Decades of Music *Reconfiguration is implemented by clicking any of the top buttons (ALL DECADES, 90s, 80s, etc.), and the playcounts of top 51 songs on Spotify reorder starting from most played to least played.
- XXXX Tracks: Historic Billboard Performance vs. 2014 Spotify Plays *Reconfiguration is implemented by clicking any of the top buttons (ALL SONGS, WON GRAMMY, (PEAKED AT BILLBOARD) and the buttons below those(ALL DECADES, 90s, 80s, etc.), and the Spotify playcounts of those songs. *Select is implemented by hovering over selected individual points to show the artist, the song, the year, how many times it has been played on Spotify in 2014, and its billboard yearly rank.
- The Long-term Future of Hits from 2013 *Filter is implemented by showing the growth and decay rate of 2013-released tracks on Spotify through graphs by clicking on specific tracks and comparing different tracks by clicking and unclicking different tracks.

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.

For the Biggie or Tupac visualization, while I did like the idea of visualization for the most-played rap hits from 1986 to 1999, I felt that it was too crowded and confusing. For example, even when I clicked on the 'Just Biggie and Tupac' button, it was still a bit hard to find just Biggie's and/or Tupac's most-played hits, due to the fact that the other rapper's circles were still there and it was hard to see only Biggie and/or Tupac's hits. Also, when I typed in 'Biggie' or 'Tupac' in the 'Find a rapper' search bar, they did not show up due to the fact that they were called by their full rapper names in the visualization. 'Tupac' did show up for two songs, but the rest were kept under the name '2 Pac', which is confusing and will not ever show all of Tupac's songs when searched. The button should say their full rapper names, because people will try to search for 'Biggie' or 'Tupac', and nothing will show up. Also the three random titles in the visualization, such as 'Jump, Kriss Kross' are unnecessary, as the reader can just hover over the circle and see who the rapper is and what song it is that is being presented. Overall, it was a good start for a visualization and did have a lot of valuable information, but needed a lot of clean up.

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
    # 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	
10005	299011	Carl (Mutant) (Earth-616)	\/Carl_(Mutant)_(Earth-616)	Secret Identity	Neutral Characters	NaN	White Hair	Cl
664	1094	Philippa Sontag (Earth-616)	\/Philippa_Sontag_(Earth-616)	Secret Identity	Bad Characters	Violet Eyes	Black Hair	Cl
12085	658425	Susan Storm (Counter-Earth) (Earth-616)	\/Susan_Storm_(Counter-Earth)_(Earth-616)	Public Identity	Good Characters	Blue Eyes	Blond Hair	Cl
9963	474805	Waldo Dini (Earth-616)	\/Waldo_Dini_(Earth-616)	No Dual Identity	Bad Characters	Brown Eyes	Brown Hair	Cl
1250	68368	Vernon Questor (New Earth)	\/wiki\/Vernon_Questor_(New_Earth)	NaN	NaN	NaN	Black Hair	Cl

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/)

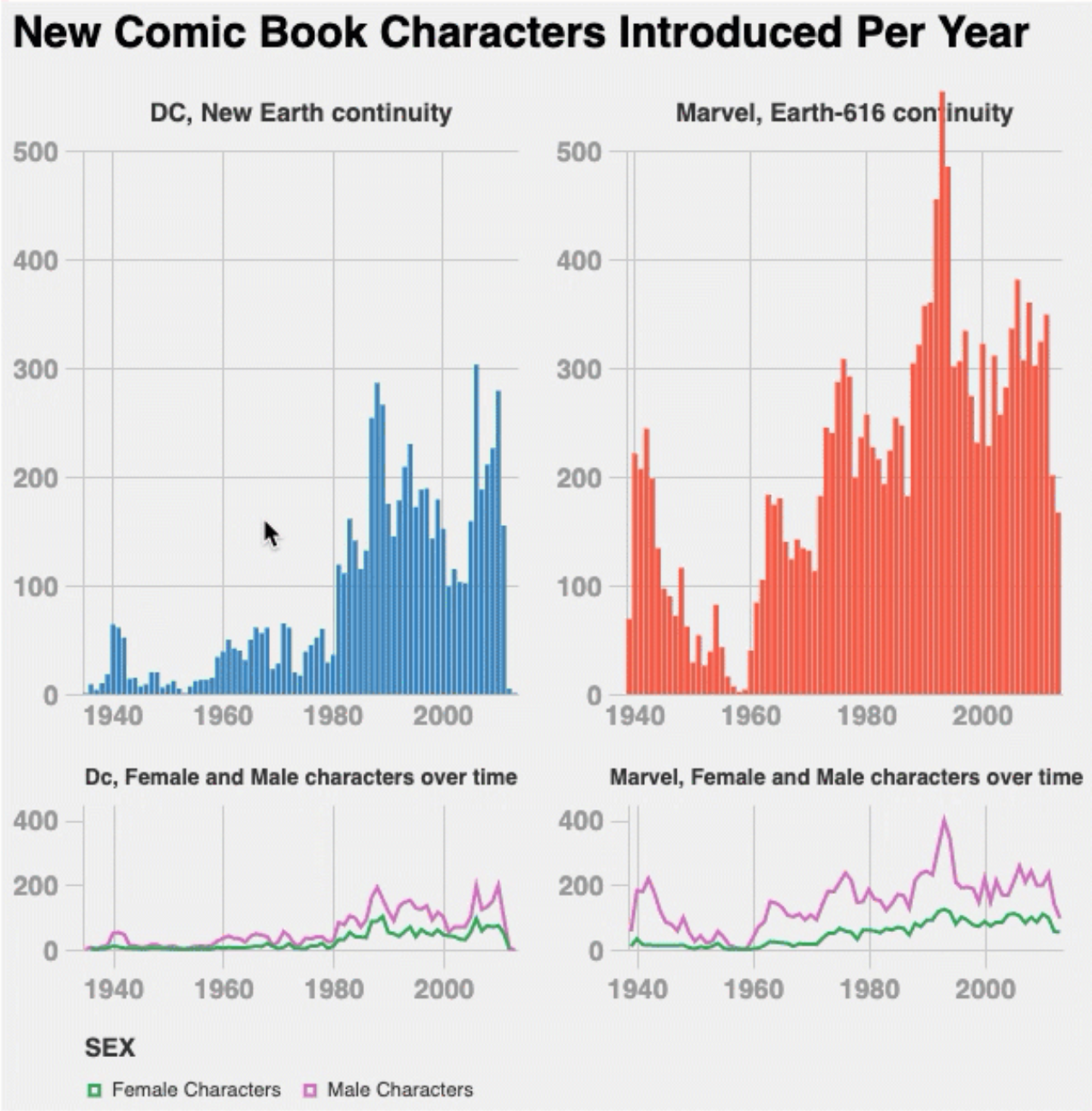
Get the data on [GitHub (https://github.com/fivethirtyeight/data/tree/master/comic-characters)](https://github.com/fivethirtyeight/data/tree/master/comic-characters)

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.
    # return: a static version of the visualization pieces described abo
```

```
    # we're going to generate the 2 visualizations at once to make things
    # into sub-functions in a different implementation

    # we're largely going to use the same "base" visualization here for
    # 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 base
                                      axis=alt.Axis(values=[1940, 1960, 198
                                                    title="DC, New Earth co
                                                    titlePadding=-347,
                                                    labelAngle=360,
                                                    labelFontWeight="bold",
                                                    labelFontSize=15,)),
            ).transform_filter(
                # we will use Altair's filter to only keep DC for this c
                alt.datum.publisher == 'DC'
            )

    # let's do the same thing for marvel
    bar_marvel = p1_bar_base.mark_bar(color='#f6573f').encode(alt.X('YEA
                                # fix the look of the axes
                                axis=alt.Axis(values=[1940, 1960, 1980, 2
                                                    title="Marvel, Earth-616 co
                                                    titlePadding=-347,
                                                    labelAngle=360,
                                                    labelFontWeight="bold",
                                                    labelFontSize=15)),
            ).transform_filter(
                # we will use Altair's filter to only keep DC for this c
                alt.datum.publisher == 'Marvel'
            )

    return(bar_dc,bar_marvel)
```

In [8]:
```
def genStaticLines(comicDF):
    # input: comicDF -- dataframe of characters as described above (e.g.
    # return: a static version of the visualization pieces described abo
```

```
# we're going to generate the 2 visualizations at once to make things
# into sub-functions in a different implementation

# let's create a new "base" chart for the two line charts. We'll take
# 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 Cl
               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, 19
                                                          grid=Tr
                                                          labelAn
                                                          labelFo
                                                          labelFo
                                                          title =
                                                          titlePa
                                                          titleFo
                                                         )
                                    )
            ).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,
                                                              grid=True,
                                                              labelAngle=
                                                              labelFontWe
                                                              labelFontSi
                                                              title = 'Ma
                                                              titlePaddin
                                                              titleFontSi
```

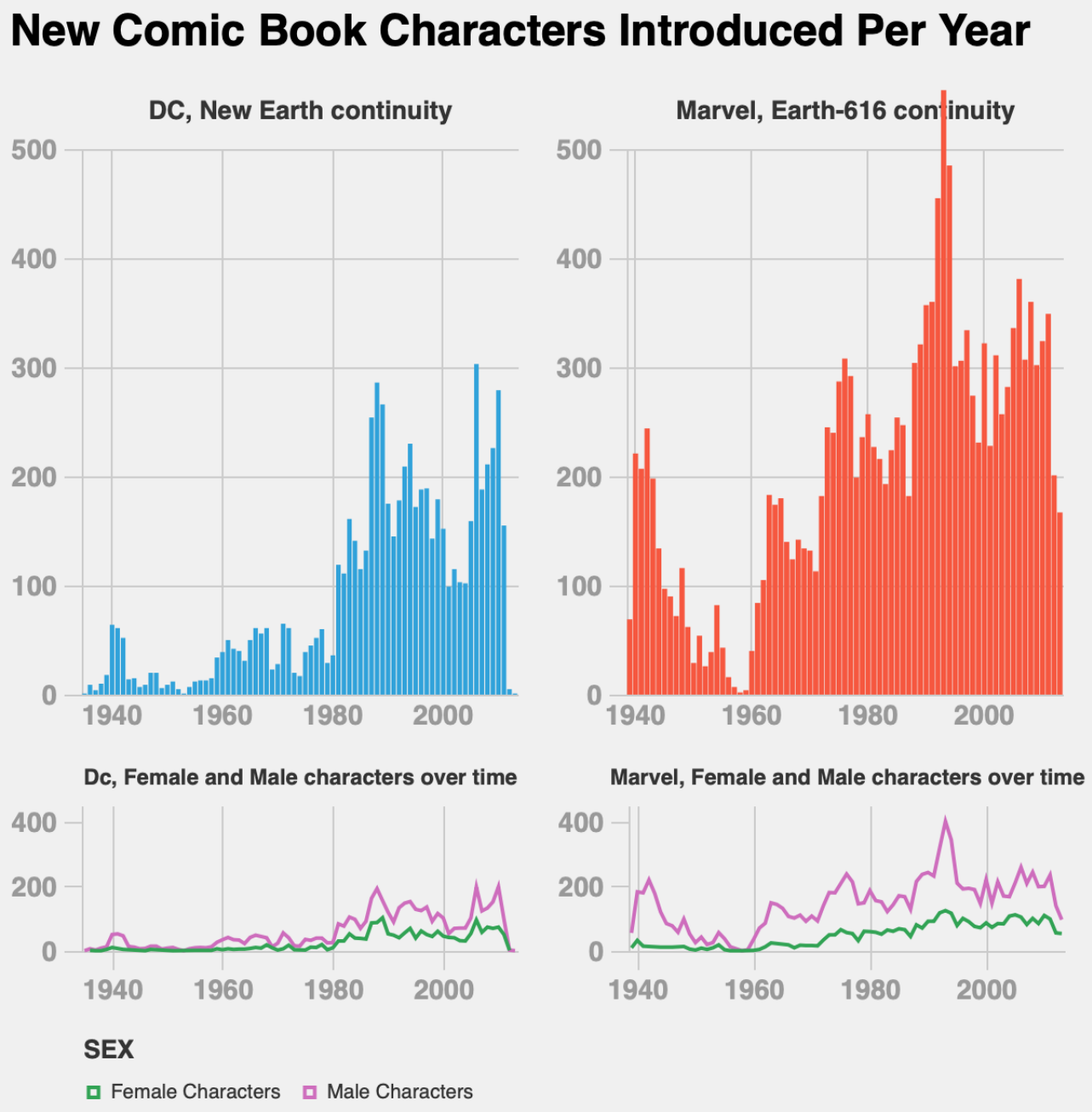
```
                                                                    )
                                                )
                    ).transform_filter(
                        # this is the Marvel line chart, so we only want Mar
                        alt.datum.publisher == 'Marvel'
                    )



    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
                )

    # 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'])

# YOUR CODE HERE
#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("g

# YOUR CODE HERE
#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:

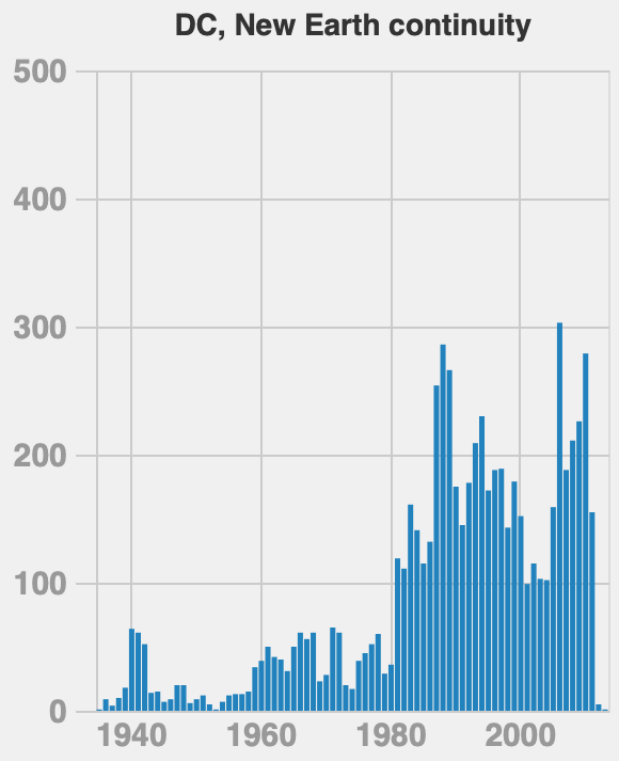
```
i_bar_dc = bar_dc.encode(color = 'TEST')
```

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.encode(
    color=colorConditionDC
).add_selection(
    brush
)

# YOUR CODE HERE
#raise NotImplementedError()
```

In [15]:
```
# if you did the last step correctly, you should be able to see the sele
i_bar_dc
```

Out[15]:



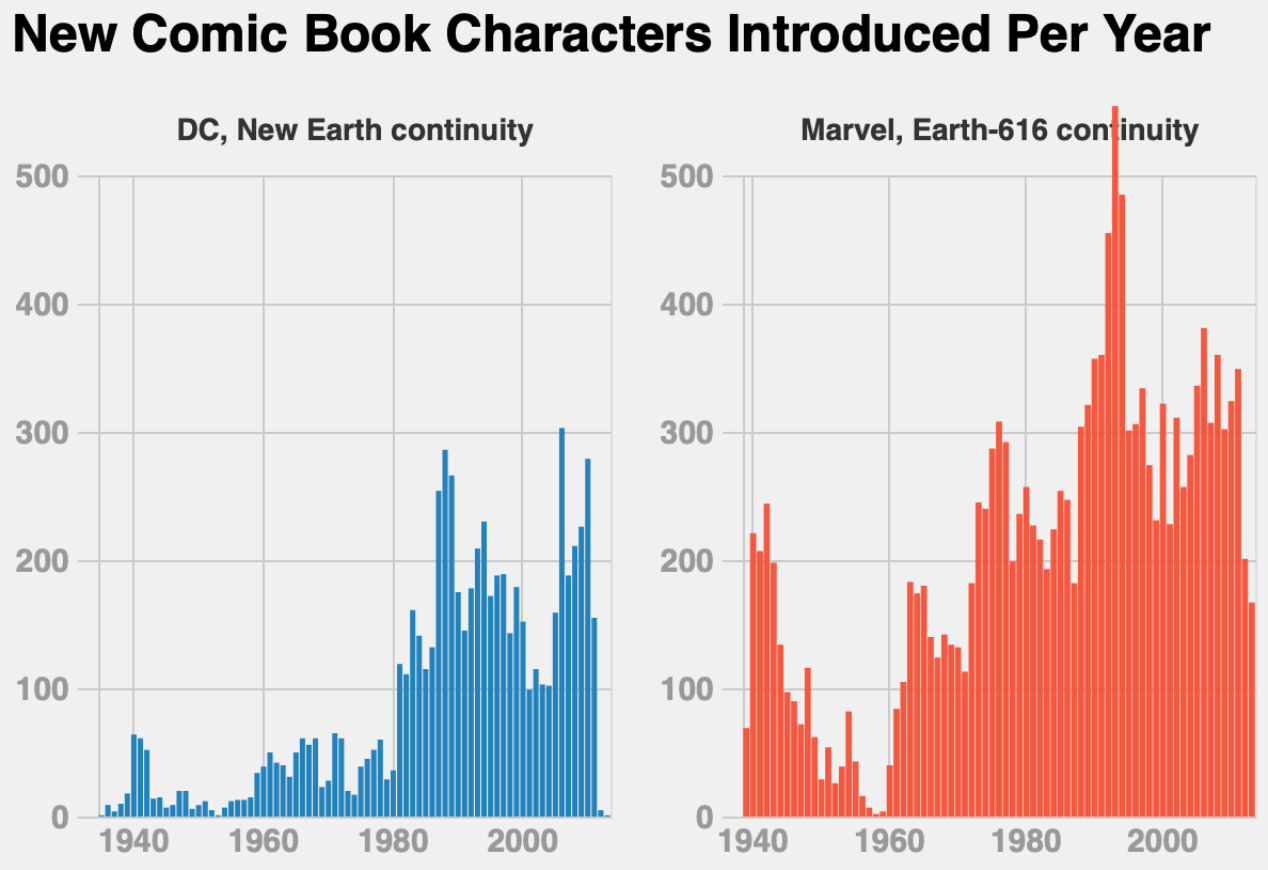
Problem 2.1.4

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

i_bar_marvel = bar_marvel.encode(
    color=colorConditionMarvel
).add_selection(
    brush
)

# YOUR CODE HERE
#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 int
# (and the selections should be linked)
top_charts
```

Out[17]:



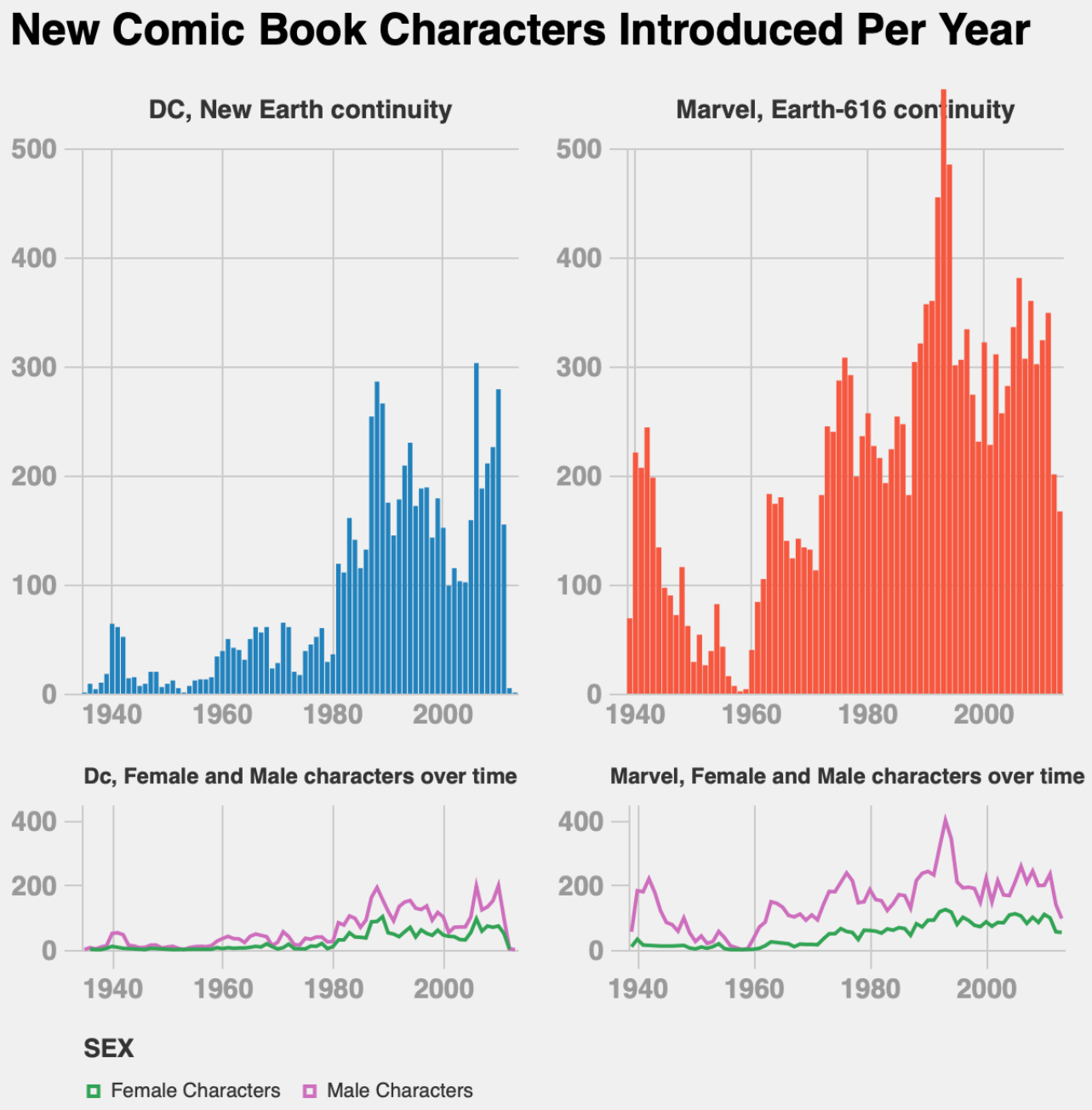
Problem 2.1.5

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.add_selection(
    brush
).transform_filter(
    brush
)
i_line_marvel = line_marvel.add_selection(
    brush
).transform_filter(
    brush
)

# YOUR CODE HERE
#raise NotImplementedError()
```

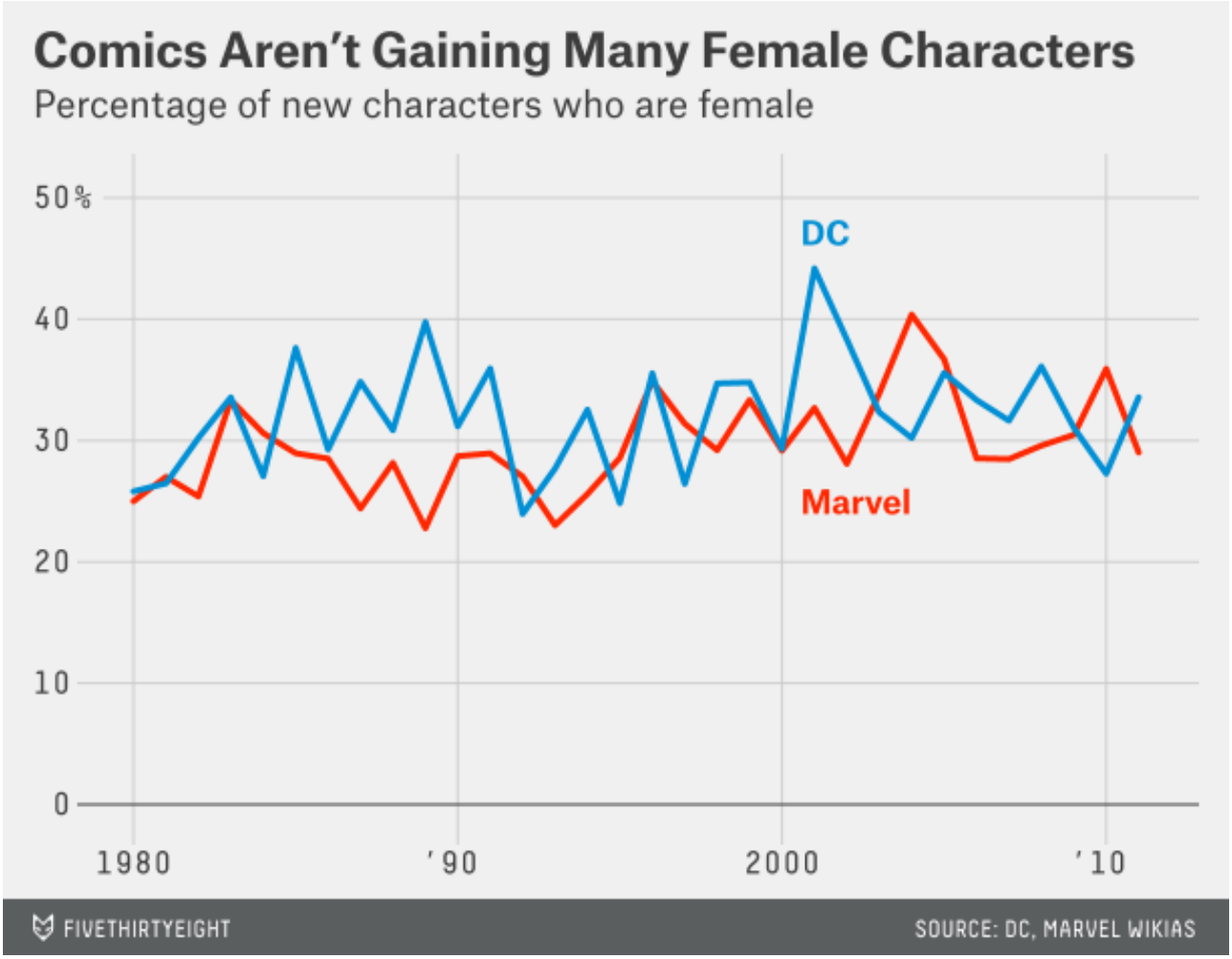
In [19]:
```
# let's put everything together with your new interactive charts. If you
# should generate the visualizations we want

# 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.
    # 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].a

    _df['% Female'] = _df['SEX_Female Characters'] / _df.total
    _df = _df.reset_index()
    _df = _df[['YEAR','% Female','SEX_Female Characters','SEX_Male Charac
    _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
    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"),generatePerc

# now, let's restructure it to the right long form
changedata = pd.melt(changedata,id_vars=['YEAR','publisher'],value_vars=
                                                            'Year-over-y
                                                            '% Female cl
```

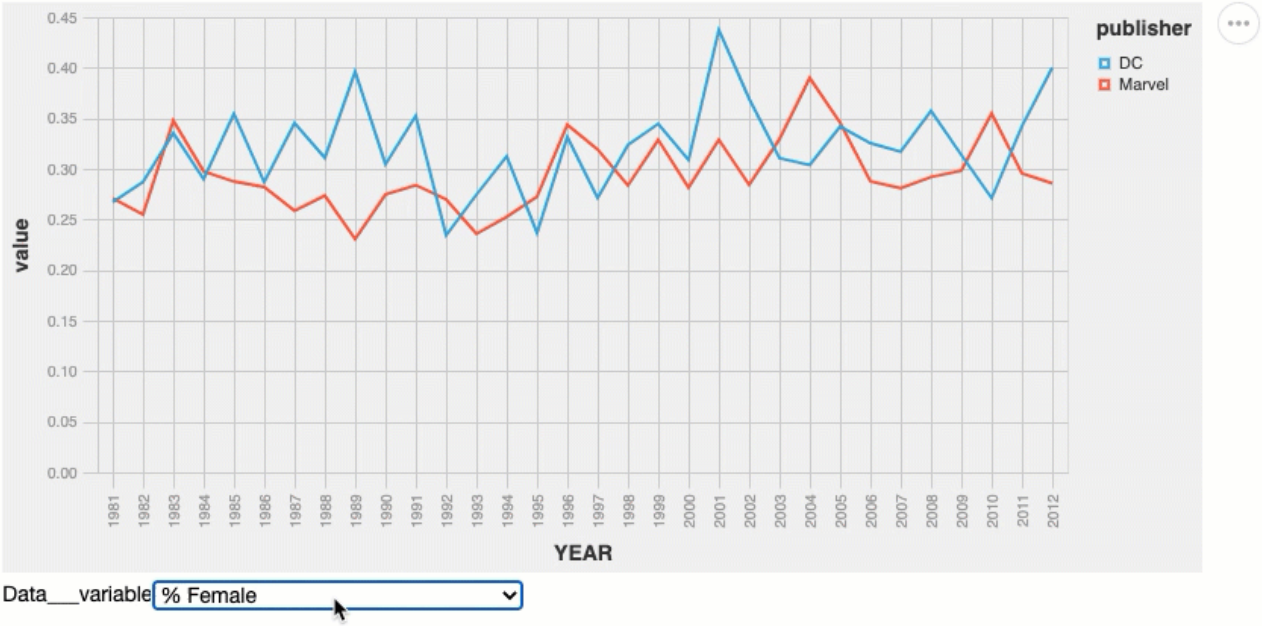
In [22]:
```
# let's see what's inside
changedata.sample(5)
```

Out[22]:

	YEAR	publisher	variable	value
54	2003	DC	% Female	0.310680
103	1988	DC	Year-over-year change in % Female	-0.099305
138	1991	Marvel	% Female characters to date	0.276369
108	1993	DC	Year-over-year change in % Female	0.171688
45	1994	DC	% Female	0.312500

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:

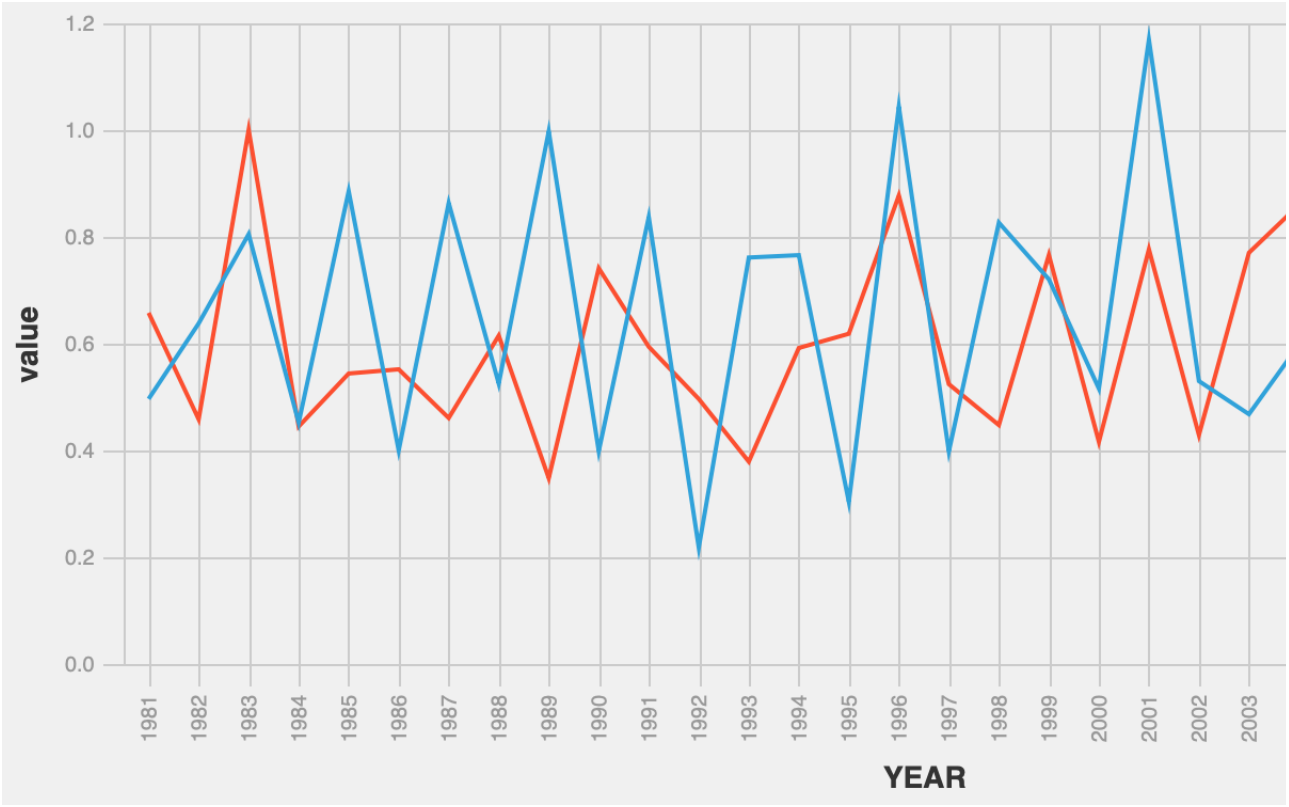


Modify `generateLineChartP21` below to generate this chart. If you haven't already, you'll want to take a look at the binding_select examples. Make sure you can get the chart working without interactivity first (hint: see if you can figure out how to filter to specific variables of interest).

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','% Fe
    input_dropdown = alt.binding_select(options=metricOptions)

    dropdown_selection = alt.selection_single(fields=['variable'], bind=

    line = alt.Chart(changedata).mark_line().encode(
        x=alt.X('YEAR:N'),
        y=alt.Y("sum(value)", title='value'),
        color='publisher'
    ).add_selection(
        dropdown_selection
    ).transform_filter(
        dropdown_selection
    ).properties(width=750, height=300)
    #raise NotImplementedError()
    return(line)
```

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

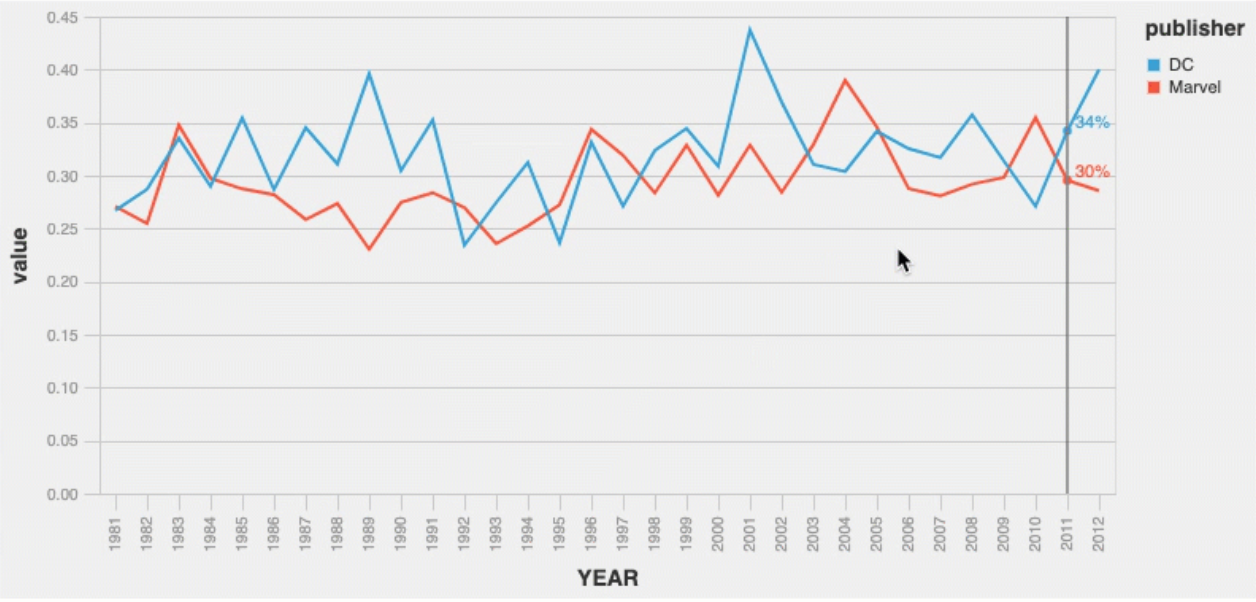
Out[24]:



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.

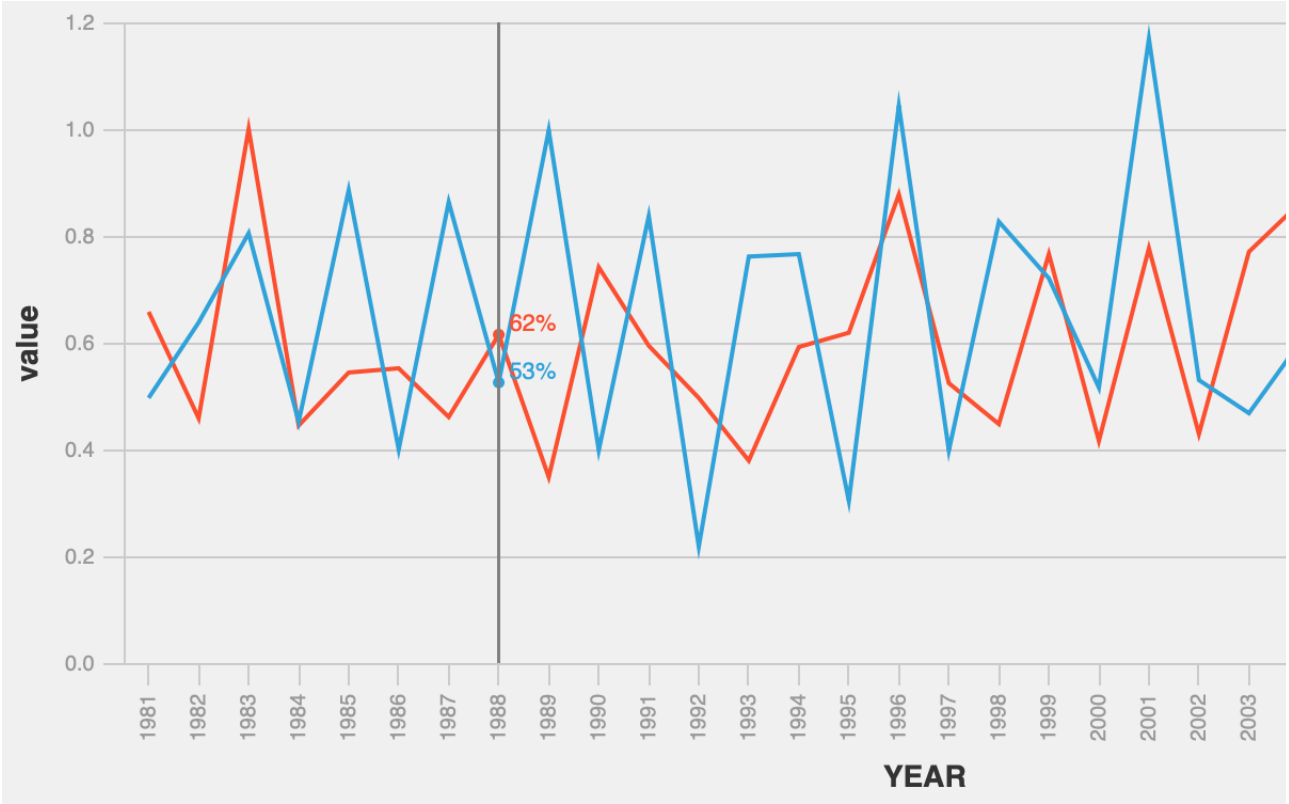
In [25]:
```
def generateLineChartP22(changeDF):
    metricOptions = ['% Female','Year-over-year change in % Female','% Fe
```

```
    input_dropdown = alt.binding_select(options=metricOptions)

    dropdown_selection = alt.selection_single(fields=['variable'], bind=

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

    line2 = alt.Chart(changedata).mark_line().encode(
        x=alt.X('YEAR:N'),
        y=alt.Y("sum(value):Q", title='value'),
        color='publisher:N'
    )

    selectors = alt.Chart(changedata).mark_point().encode(
        x=alt.X('YEAR:N',axis=alt.Axis(labels=True)),
        opacity=alt.value(0)
    ).add_selection(
        nearest
    )

    points = line2.mark_point().encode(
        opacity=alt.condition(
            nearest,
            alt.value(1),
            alt.value(0)
        )
    )

    text = line2.mark_text(align='left', dx=5, dy=-5).encode(
        text=alt.condition(
            nearest,
            "sum(value):Q",
            alt.value(' '),
            format='.0%'
        )
    )

    rules = alt.Chart(changedata).mark_rule(color='gray').encode(
        x=alt.X('YEAR:N',),
    ).transform_filter(
        nearest
    )

    final = alt.layer(line2, selectors, points, rules, text).add_selecti
        dropdown_selection
    ).transform_filter(
        dropdown_selection
    ).properties(width=750,height=300)

    return(final)
```

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

    # YOUR CODE HERE
    #raise NotImplementedError()
```

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
```

In [ ]: