assignment1

April 6, 2021

1 Information Visualization II

1.1 School of Information, University of Michigan

1.2 Week 1:

- Multivariate/Multidimensional + Temporal

1.3 Assignment Overview

1.3.1 This assignment's objectives include:

- Review, reflect on, and apply different strategies for multidimensional/multivariate/temporal datasets
- Recreate visualizations and propose new and alternative visualizations using Altair

1.3.2 The total score of this assignment will be 100 points consisting of:

- You will be producing four visualizations. Three of them will require you to follow the example closely, but the last will be fairly open-ended. For the last one, we'll also ask you to justify why you designed your visualization the way you did.

1.3.3 Resources:

- Article by FiveThirtyEight available online (Hickey, 2014)
- The associated dataset on Github
- A dataset of all the paintings from the show

1.3.4 Important notes:

1) Grading for this assignment is entirely done by manual inspection. For some of the visualizations, we'll expect you to get pretty close to our example (1-3). Problem 4 is more free-form.

2) There are a few instances where our numbers do not align exactly with those from 538. We've pre-processed our data a little bit differently.

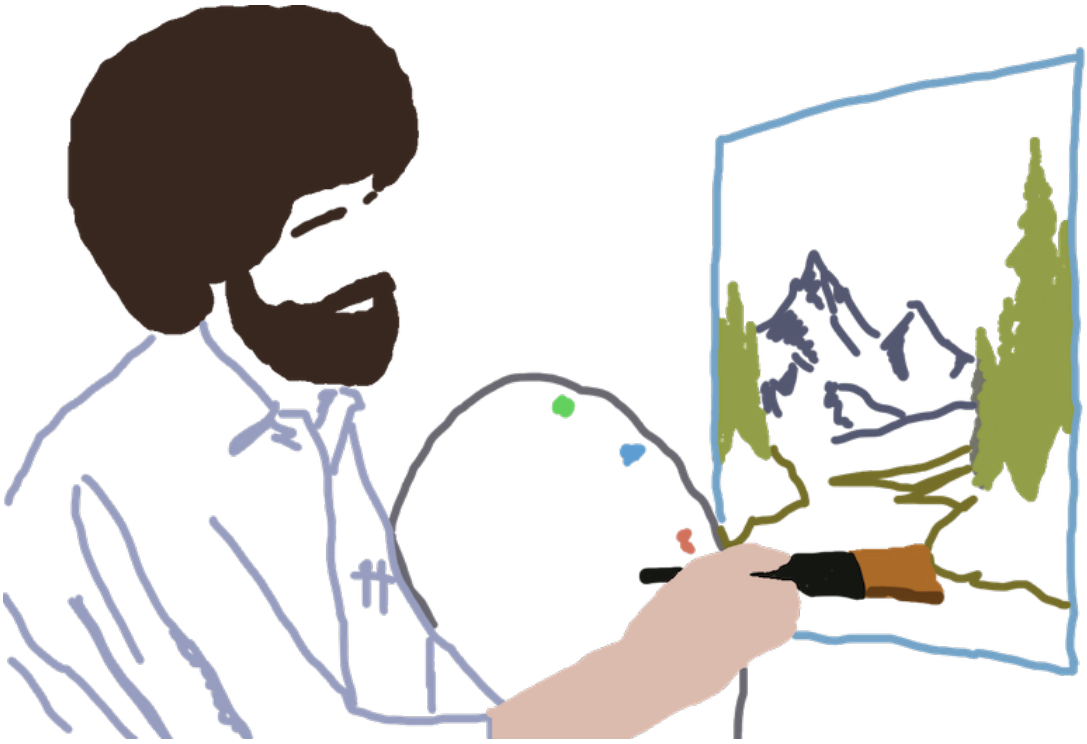
3) When turning in your PDF, please use the File -> Print -> Save as PDF option ***from your browser***. Do ***not*** use the File->Download as->PDF option. Complete instructions for this are under Resources in the Coursera page for this class.

[1]:
```
# load up the resources we need
import zipfile as zip
import urllib.request
import os.path
from os import path
import pandas as pd
import altair as alt
import numpy as np
from sklearn import manifold
from sklearn.metrics import euclidean_distances
from sklearn.decomposition import PCA
import ipywidgets as widgets
from IPython.display import display
from PIL import Image
```

1.4 Bob Ross

Today's assignment will have you working with artwork created by Bob Ross. Bob was a very famous painter who had a televised painting show from 1983 to 1994. Over 13 seasons and approximately 400 paintings, Bob would walk the audience through a painting project. Often these were landscape images. Bob was famous for telling his audience to paint "happy trees" and sayings like, "We don't make mistakes, just happy little accidents." His soothing voice and bushy hair are well known to many generations of viewers.

If you've never seen an episode, I might suggest starting with this one.



Bob Ross left a long legacy of art which makes for an interesting dataset to analyze. It's both temporally rich and has a lot of variables we can code. We'll be starting with the dataset created by 538 for their article on a Statistical Analysis of Bob Ross. The authors of the article coded each painting to indicate what features the image contained (e.g., one tree, more than one tree, what kinds of clouds, etc.).

In addition, we've downloaded a second dataset that contains the actual images. We know what kind of paint colors Bob used in each episode, and we have used that to create a dataset for you containing the color distributions. For example, we approximate how much 'burnt umber' he used by measuring the distance (in color space) from each pixel in the image to the color. This is imperfect, of course (paints don't mix this way), but it'll be close enough for our analysis.

[2]:
```
# the paints Bob used
rosspaints = ['alizarin crimson','bright red','burnt umber','cadmium␣
 ↪yellow','dark sienna',
              'indian yellow','indian red','liquid black','liquid clear','black␣
 ↪gesso',
              'midnight black','phthalo blue','phthalo green','prussian␣
 ↪blue','sap green',
              'titanium white','van dyke brown','yellow ochre']

# hex values for the paints above
rosspainthex =␣
 ↪['#94261f','#c06341','#614f4b','#f8ed57','#5c2f08','#e6ba25','#cd5c5c',
               ␣
 ↪'#000000','#ffffff','#000000','#36373c','#2a64ad','#215c2c','#325fa3',
                '#364e00','#f9f7eb','#2d1a0c','#b28426']

# boolean features about what an image includes
imgfeatures = ['Apple frame', 'Aurora borealis', 'Barn', 'Beach', 'Boat',
               'Bridge', 'Building', 'Bushes', 'Cabin', 'Cactus',
               'Circle frame', 'Cirrus clouds', 'Cliff', 'Clouds',
               'Coniferous tree', 'Cumulis clouds', 'Decidious tree',
               'Diane andre', 'Dock', 'Double oval frame', 'Farm',
               'Fence', 'Fire', 'Florida frame', 'Flowers', 'Fog',
               'Framed', 'Grass', 'Guest', 'Half circle frame',
               'Half oval frame', 'Hills', 'Lake', 'Lakes', 'Lighthouse',
               'Mill', 'Moon', 'At least one mountain', 'At least two␣
 ↪mountains',
               'Nighttime', 'Ocean', 'Oval frame', 'Palm trees', 'Path',
               'Person', 'Portrait', 'Rectangle 3d frame', 'Rectangular frame',
               'River or stream', 'Rocks', 'Seashell frame', 'Snow',
               'Snow-covered mountain', 'Split frame', 'Steve ross',
               'Man-made structure', 'Sun', 'Tomb frame', 'At least one tree',
               'At least two trees', 'Triple frame', 'Waterfall', 'Waves',
               'Windmill', 'Window frame', 'Winter setting', 'Wood framed']
```

```
# load the data frame
bobross = pd.read_csv("assets/bobross.csv")

# enable correct rendering (unnecessary in later versions of Altair)
alt.renderers.enable('default')

# uses intermediate json files to speed things up
alt.data_transformers.enable('json')
```

[2]:
```
DataTransformerRegistry.enable('json')
```

We have a few variables defined for you that you might find useful for the rest of this exercise. First is the `bobross` dataframe which, has a row for every painting created by Bob (we've removed those created by guest artists).

[3]:
```
# run to see what's inside
bobross.sample(5)
```

[3]:

	EPISODE	TITLE	RELEASE_DATE	Apple frame	Aurora borealis	Barn	Beach	Boat	Bridge	Building	…	phthalo blue	phthalo green	prussian blue	sap green	titanium white	van dyke brown	yellow ochre	img_url	week_number	year
353	S29E10	"POT 'O POSIES"	10/26/93	0	0	0	0	0	0	0	…	0.000000	0.000000	0.228577	0.830542	0.000000	0.000000	0.000000	https://raw.githubusercontent.com/jwilber/Bob_…	43	1993
82	S08E01	"MISTY ROLLING HILLS"	1/2/86	0	0	0	0	0	0	0	…	0.298472	0.000000	0.000000	0.179594	0.605754	0.194742	0.263854	https://raw.githubusercontent.com/jwilber/Bob_…	1	1986
304	S25E11	"FISHERMAN'S PARADISE"	11/3/92	0	0	0	0	0	0	0	…	0.000000	0.000000	0.354522	0.265820	0.454654	0.285899	0.389736	https://raw.githubusercontent.com/jwilber/Bob_…	44	1992
201	S17E08	"VIEW FROM THE PARK"	2/22/89	1	0	0	0	0	1	1	…	0.373443	0.347124	0.392300	0.257786	0.477034	0.284331	0.272549	https://raw.githubusercontent.com/jwilber/Bob_…	8	1989
360	S30E04	"WILDERNESS TRAIL"	12/14/93	0	0	0	0	0	0	0	…	0.000000	0.000000	0.320583	0.729900	0.029397	0.783438	0.391064	https://raw.githubusercontent.com/jwilber/Bob_…	50	1993

```
353          43  1993
82            1  1986
304          45  1992
201           8  1989
360          50  1993

[5 rows x 114 columns]
```

In the dataframe you will see an episode identifier (EPISODE, which contains the season and episode number), the image title (TITLE), the release date (RELEASE_DATE as well as another column for the year). There are also a number of boolean columns for the features coded by 538. A '1' means the feature is present, a '0' means it is not. A list of those columns is available in the `imgfeatures` variable.

[4]:
```
# run to see what's inside
print(imgfeatures)
```

```
['Apple frame', 'Aurora borealis', 'Barn', 'Beach', 'Boat', 'Bridge',
'Building', 'Bushes', 'Cabin', 'Cactus', 'Circle frame', 'Cirrus clouds',
'Cliff', 'Clouds', 'Coniferous tree', 'Cumulis clouds', 'Decidious tree', 'Diane
andre', 'Dock', 'Double oval frame', 'Farm', 'Fence', 'Fire', 'Florida frame',
'Flowers', 'Fog', 'Framed', 'Grass', 'Guest', 'Half circle frame', 'Half oval
frame', 'Hills', 'Lake', 'Lakes', 'Lighthouse', 'Mill', 'Moon', 'At least one
mountain', 'At least two mountains', 'Nighttime', 'Ocean', 'Oval frame', 'Palm
trees', 'Path', 'Person', 'Portrait', 'Rectangle 3d frame', 'Rectangular frame',
'River or stream', 'Rocks', 'Seashell frame', 'Snow', 'Snow-covered mountain',
'Split frame', 'Steve ross', 'Man-made structure', 'Sun', 'Tomb frame', 'At
least one tree', 'At least two trees', 'Triple frame', 'Waterfall', 'Waves',
'Windmill', 'Window frame', 'Winter setting', 'Wood framed']
```

The columns that contain the amount of each color in the paintings are listed in `rosspaints`. There is also an analogous list variable called `rosspainthex` that has the hex values for the paints. These hex values are approximate.

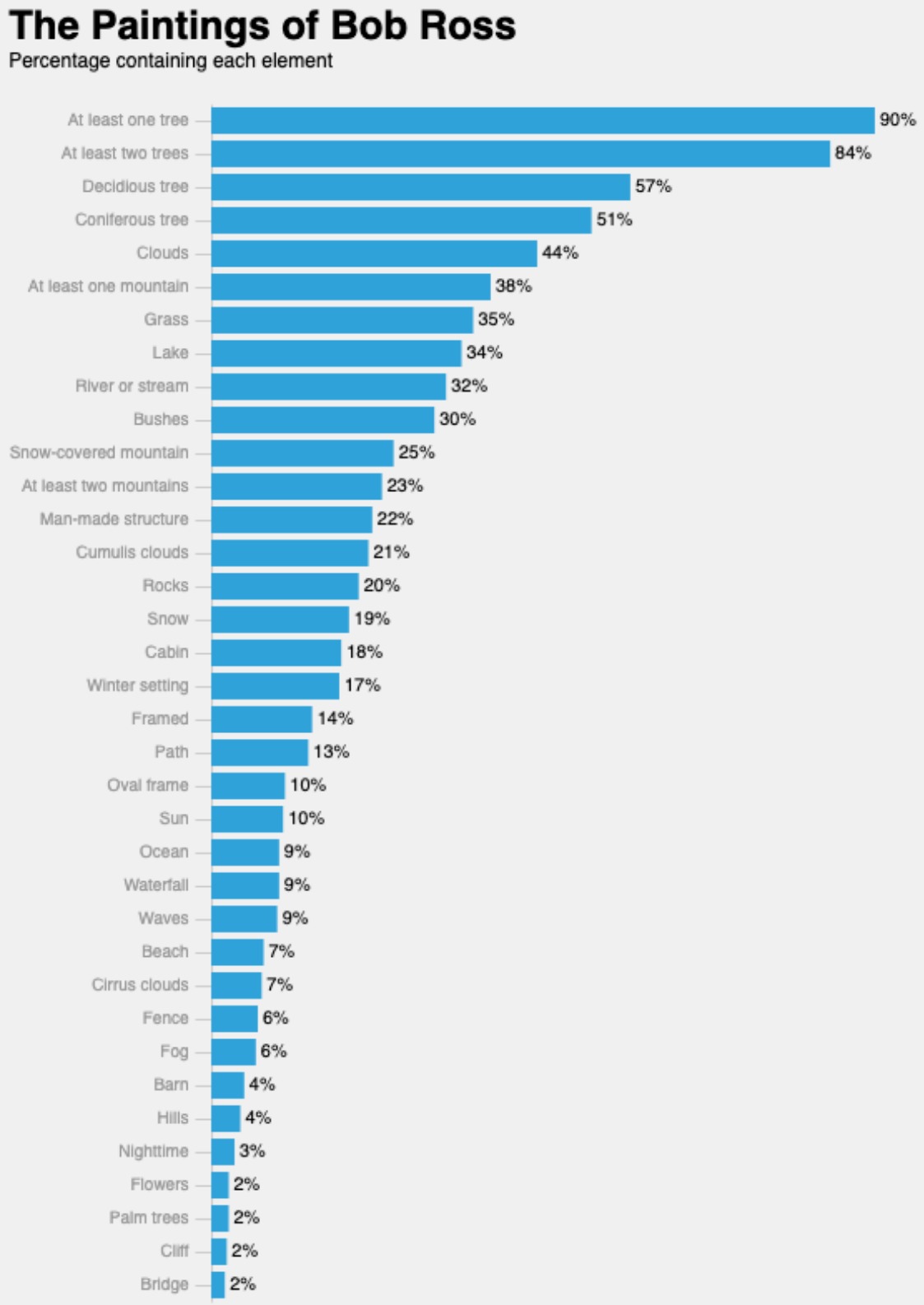
[5]:
```
# run to see what's inside
print("paint names",rosspaints)
print("")
print("hex values", rosspainthex)
```

```
paint names ['alizarin crimson', 'bright red', 'burnt umber', 'cadmium yellow',
'dark sienna', 'indian yellow', 'indian red', 'liquid black', 'liquid clear',
'black gesso', 'midnight black', 'phthalo blue', 'phthalo green', 'prussian
blue', 'sap green', 'titanium white', 'van dyke brown', 'yellow ochre']

hex values ['#94261f', '#c06341', '#614f4b', '#f8ed57', '#5c2f08', '#e6ba25',
'#cd5c5c', '#000000', '#ffffff', '#000000', '#36373c', '#2a64ad', '#215c2c',
'#325fa3', '#364e00', '#f9f7eb', '#2d1a0c', '#b28426']
```

1.4.1 Problem 1 (20 points)

As a warmup, we're going to have you recreate the first chart from the Bob Ross article (source: Statistical Analysis of Bob Ross). This one simply shows a bar chart for the percent of images that have certain features. The Altair version is:



We'll be using the 538 theme for styling, so you don't have to do much beyond creating the chart (but do note that we want to see the percents, titles, and modifications to the axes).

You will replace the code for `makeBobRossBar()` and have it return an Altair chart. We suggest you first create a table that contains the names of the features and the percents. Something like this:

	index	value
0	Barn	0.044619
1	Beach	0.070866
2	Bridge	0.018373
3	Bushes	0.301837
4	Cabin	0.175853
5	Cirrus clouds	0.068241
6	Cliff	0.020997
7	Clouds	0.440945

Recall that this is the 'long form' representation of the data, which will make it easier to create a visualization with.

[6]:
```
new = bobross[['Apple frame', 'Aurora borealis', 'Barn', 'Beach', 'Boat',␣
↪'Bridge', 'Building', 'Bushes', 'Cabin', 'Cactus', 'Circle frame', 'Cirrus␣
↪clouds', 'Cliff', 'Clouds', 'Coniferous tree', 'Cumulis clouds', 'Decidious␣
↪tree', 'Diane andre', 'Dock', 'Double oval frame', 'Farm', 'Fence', 'Fire',␣
↪'Florida frame', 'Flowers', 'Fog', 'Framed', 'Grass', 'Guest', 'Half circle␣
↪frame', 'Half oval frame', 'Hills', 'Lake', 'Lakes', 'Lighthouse', 'Mill',␣
↪'Moon', 'At least one mountain', 'At least two mountains', 'Nighttime',␣
↪'Ocean', 'Oval frame', 'Palm trees', 'Path', 'Person', 'Portrait',␣
↪'Rectangle 3d frame', 'Rectangular frame', 'River or stream', 'Rocks',␣
↪'Seashell frame', 'Snow', 'Snow-covered mountain', 'Split frame', 'Steve␣
↪ross', 'Man-made structure', 'Sun', 'Tomb frame', 'At least one tree', 'At␣
↪least two trees', 'Triple frame', 'Waterfall', 'Waves', 'Windmill', 'Window␣
↪frame', 'Winter setting', 'Wood framed']].copy()
```

```
new2 = pd.DataFrame({'Index':['Apple frame', 'Aurora borealis', 'Barn',␣
 ↪'Beach', 'Boat', 'Bridge', 'Building', 'Bushes', 'Cabin', 'Cactus', 'Circle␣
 ↪frame', 'Cirrus clouds', 'Cliff', 'Clouds', 'Coniferous tree', 'Cumulis␣
 ↪clouds', 'Decidious tree', 'Diane andre', 'Dock', 'Double oval frame',␣
 ↪'Farm', 'Fence', 'Fire', 'Florida frame', 'Flowers', 'Fog', 'Framed',␣
 ↪'Grass', 'Guest', 'Half circle frame', 'Half oval frame', 'Hills', 'Lake',␣
 ↪'Lakes', 'Lighthouse', 'Mill', 'Moon', 'At least one mountain', 'At least␣
 ↪two mountains', 'Nighttime', 'Ocean', 'Oval frame', 'Palm trees', 'Path',␣
 ↪'Person', 'Portrait', 'Rectangle 3d frame', 'Rectangular frame', 'River or␣
 ↪stream', 'Rocks', 'Seashell frame', 'Snow', 'Snow-covered mountain', 'Split␣
 ↪frame', 'Steve ross', 'Man-made structure', 'Sun', 'Tomb frame', 'At least␣
 ↪one tree', 'At least two trees', 'Triple frame', 'Waterfall', 'Waves',␣
 ↪'Windmill', 'Window frame', 'Winter setting', 'Wood framed'], 'Value':new.
 ↪apply(np.sum, axis=0)})
new2.reset_index(inplace=True)
new2.drop(columns='index', inplace=True)
new2['Percent'] = new2['Value'] / 381
new2.sort_values('Percent', ascending=False, inplace=True)
new2.drop(columns='Value', inplace=True)
new2 = new2[new2['Percent'] >= 0.018]


bars = alt.Chart(new2).mark_bar(size=15).encode(
    x=alt.X(
        'Percent',
        axis=None
    ),
    y=alt.Y(
        'Index:N',
        axis=alt.Axis(tickCount=5, title=' '),
        sort=new2['Index'].tolist()
    )
)

text = bars.mark_text(
    align='left',
    baseline='middle',
    dx=3
).encode(
    text=alt.Text('Percent:Q', format='.0%')
)

alt.themes.enable('fivethirtyeight')

paintings = (text + bars).configure_view(
    strokeWidth=0
).properties(
```

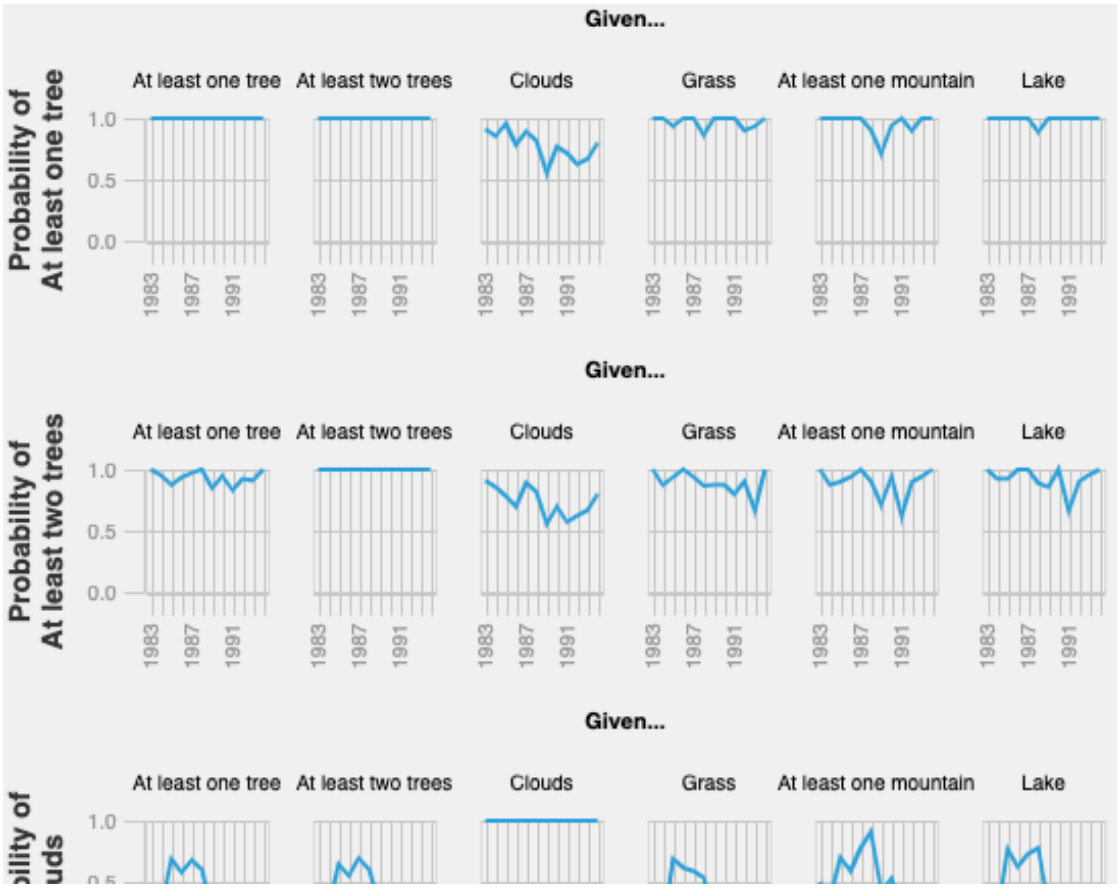
```
    title={
        "text": "The Paintings of Bob Ross",
        "subtitle":"Percentage containing each element"}
)
paintings
```

[6]: `alt.LayerChart(…)`

1.5 Problem 2 (25 points)

The 538 article (Statistical Analysis of Bob Ross) has a long analysis of conditional probabilities. Essentially, we want to know the probability of one feature given another (e.g., what is the probability of Snow given Trees?). The article calculates this over the entire history of the show, but we would like to visualize these probabilities over time. Have they been constant? or evolving? We will only be doing this for a few variables (otherwise, we'll have a matrix of over 3000 small charts). Specifically, we care about images that contain: 'At least one tree','At least two trees','Clouds','Grass','At least one mountain','Lake.' Each small multiple plot will be a line chart corresponding to the conditional probability over time. The matrix "cell" indicates which pairs of variables are being considered (e.g., probability of at least two trees given the probability of at least one tree is the 2nd row, first column in our example).

Your task will be to generate the small multiples plot below:



The full image is available here. While your small multiples visualization should contain all this data, you can ***feel free to style it as you think is appropriate***. We will be grading (minimally) on aesthetics. Implement the code for the function: `makeBobRossCondProb()` to return this chart.

Some notes on doing this exercise:

- If you don't remember how to calculate conditional probabilities, take a look at the article. Remember, we want the conditional probabilities given the images in a specific year. This is simply an implementation of Bayes' Theorem. We implemented a function called `condprobability(...)` as you can see below. You can do the same or pick your own strategy for this.
- We suggest creating a long-form representation of the table for this data. For example, here's a sample of ours (you can use this to double this sample for your calculations):

	key1	key2	year	prob
392	Lake	Clouds	1991	0.142857
60	At least one tree	Lake	1983	1.000000
417	Lake	At least one mountain	1992	0.500000
264	Grass	At least one mountain	1983	0.214286
318	At least one mountain	Clouds	1989	0.333333
85	At least two trees	At least two trees	1984	1.000000
69	At least one tree	Lake	1992	1.000000
387	Lake	Clouds	1986	0.217391
278	Grass	Lake	1985	0.384615
68	At least one tree	Lake	1991	1.000000

- There are a number of strategies to build the small-multiple plots. Some are easier than others. You will find in this case that some combinations of repeated charts and faceting will not work. However, you should be able to use the standard concatenation approaches in combination with repeated charts or faceting.

[7]:
```
def condprobability(frame,column1,column2,year):
    # we suggest you implement this function to make your life easier. It␣
↪should take a datafame as input,
    # the two columns we want the conditional probability for, and the year for␣
↪which we want to compare
    # you can make variants of this function as you see fit
```

```
    df = frame[frame['year'] == year]
    cond_prob = (len(df.loc[(df[column1] == 1) & (df[column2] == 1)]) /␣
 ↪df[column2].sum())
    return cond_prob
    # YOUR CODE HERE
    # raise NotImplementedError()
cols = ['Lake','Clouds','Grass','At least one tree','At least two trees','At␣
 ↪least one mountain']
years = [1983,1984,1985,1986,1987,1988,1989,1990,1991,1992,1993]
cond_prob = []
for column1 in cols:
    for column2 in cols:
        for year in years:
            cond_prob.append([column1, column2, year, condprobability(bobross,␣
 ↪column1, column2, year)])

cond_prob_df = pd.DataFrame(cond_prob, columns=['key1','key2','year','prob'])
```

[8]:
```
df = cond_prob_df[cond_prob_df['key1'] == 'At least one tree']
chart1 = alt.Chart(df).mark_line().encode(
            alt.X('year', title=None, axis=alt.Axis(tickCount=8)),
            alt.Y('prob', title=["Probability of", "At least one tree"]),
            facet=alt.Facet('key2', title="Given...")
        ).properties(width=100, height=100
        )
```

[9]:
```
df = cond_prob_df[cond_prob_df['key1'] == 'At least two trees']
chart2 = alt.Chart(df).mark_line().encode(
            alt.X('year', title=None, axis=alt.Axis(tickCount=8)),
            alt.Y('prob', title=["Propability of", "At least two trees"]),
            facet=alt.Facet('key2', title="Given...")
        ).properties(width=100, height=100
        )
```

[10]:
```
df = cond_prob_df[cond_prob_df['key1'] == 'Clouds']
chart3 = alt.Chart(df).mark_line().encode(
            alt.X('year', title=None, axis=alt.Axis(tickCount=8)),
            alt.Y('prob', title=["Probability of", "Clouds"]),
            facet=alt.Facet('key2', title="Given...")
        ).properties(width=100, height=100
        )
```

[11]:
```
df = cond_prob_df[cond_prob_df['key1'] == 'Grass']
chart4 = alt.Chart(df).mark_line().encode(
            alt.X('year', title=None, axis=alt.Axis(tickCount=8)),
            alt.Y('prob', title=["Probability of", "Grass"]),
            facet=alt.Facet('key2', title="Given...")
```

```
        ).properties(width=100, height=100
        )
```

[12]:
```
df = cond_prob_df[cond_prob_df['key1'] == 'At least one mountain']
chart5 = alt.Chart(df).mark_line().encode(
            alt.X('year', title=None, axis=alt.Axis(tickCount=8)),
            alt.Y('prob', title=["Probability of", "At least one mountain"]),
            facet=alt.Facet('key2', title="Given...")
        ).properties(width=100, height=100
        )
```

[13]:
```
df = cond_prob_df[cond_prob_df['key1'] == 'Lake']
chart6 = alt.Chart(df).mark_line().encode(
            alt.X('year', title=None, axis=alt.Axis(tickCount=8)),
            alt.Y('prob', title=["Probability of", "Lake"]),
            facet=alt.Facet('key2', title="Given...")
        ).properties(width=100, height=100
        )
```

[14]:
```
chart = alt.vconcat(chart1, chart2, chart3, chart4, chart5, chart6).
↪configure_header(labelFontSize=12)
chart
```

[14]:
```
alt.VConcatChart(…)
```

1.5.1 Additional comments

If you deviated from our example, please use this cell to give us additional information about your design choices and why you think they are an improvement.

1.6 Problem 3 (25 points)

Recall that in some cases of multidimensional data a good strategy is to use dimensionality reduction to visualize the information. Here, we would like to understand how images are similar to each other in 'feature' space. Specifically, how similar are they based on the image features? Are images that have beaches close to those with waves?

We are going to create a 2D MDS plot using the scikit learn package. We're going to do most of this for you in the next cell. Essentially we will use the euclidean distance between two images based on their image feature array to create the image. Your plot may look slightly different than ours based on the random seed (e.g., rotated or reflected), but in the end, it should be close. If you're interested in how this is calculated, we suggest taking a look at this documentation

Note that the next cell may take a minute or so to run, depending on the server.

[15]:
```
# create the seed
seed = np.random.RandomState(seed=3)
```

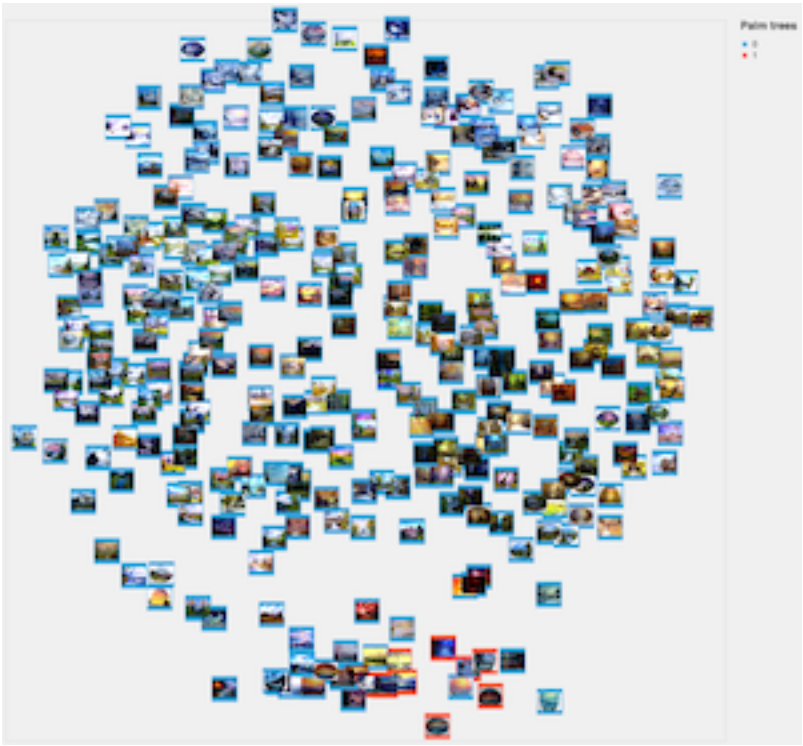
```
# generate the MDS configuration, we want 2 components, etc. You can tweak this␣
 ↪if you want to see how
# the settings change the layout
mds = manifold.MDS(n_components=2, max_iter=3000, eps=1e-9, random_state=seed,␣
 ↪n_jobs=1)

# fit the data. At the end, 'pos' will hold the x,y coordinates
pos = mds.fit(bobross[imgfeatures]).embedding_

# we'll now load those values into the bobross data frame, giving us a new x␣
 ↪column and y column
bobross['x'] = [x[0] for x in pos]
bobross['y'] = [x[1] for x in pos]
```

Your task is to implement the visualization for the MDS layout. We will be using a new mark, `mark_image`, for this. You can read all about this mark on the Altair site here. Note that we all already saved the images for you. They are accessible in the img_url column in the bobross table. You will use the `url` encode argument to mark_image to make this work.

In this case, we would also like to emphasize all the images that *have* a specific feature. So when you define your `genMDSPlot()` function below, it should take a key string as an argument (e.g., 'Beach') and visually highlight those images. A simple way to do this is to use a second mark underneath the image (e.g., a rectangle) that is a different color based on the absence or presence of the image. Here's an example output for `genMDSPlot("Palm trees")`:



Click here for a large version of this image. Notice the orange boxes indicating where the Palm

tree images are. Note that we have styled the MDS plot to not have axes. Recall that these are meaningless in MDS 'space' (this is not a scatterplot, it's a projection).

[16]:
```
image = alt.Chart(bobross).mark_image(
        width=15,
        height=15,
    ).encode(
        alt.X('x', title=None, axis=None),
        alt.Y('y', title=None, axis=None),
        url='img_url'
    )
back = alt.Chart(bobross).mark_square(size=300).encode(
        alt.X('x', title=None, axis=None),
        alt.Y('y', title=None, axis=None),
        color='Palm trees:N'
    ).properties(
        width=500,
        height=500
    )
alt.layer(
    back,
    image
).configure_view(
    strokeWidth=0
)
```

[16]:
```
alt.LayerChart(…)
```

[17]:
```
def genMDSPlot(key):
    # return an altair chart (e.g., return alt.Chart(...))
    # key is a string indicating which images should be visually highlighted (i.
 ↪e., images containing the feature
    # should be made salient)
    image = alt.Chart(bobross).mark_image(
        width=15,
        height=15,
    ).encode(
        alt.X('x', title=None, axis=None),
        alt.Y('y', title=None, axis=None),
        url='img_url'
    )
    back = alt.Chart(bobross).mark_square(size=300).encode(
            alt.X('x', title=None, axis=None),
            alt.Y('y', title=None, axis=None),
            color=alt.Color(key, type='nominal', legend=alt.Legend(title=key))
        ).properties(
            width=500,
```

```
            height=500
        )
    return alt.layer(back, image).configure_view(strokeWidth=0)

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

We are going to create an interactive widget that allows you to select the feature you want to be highlighted. If you implemented your `genMDSPlot` code correctly, the plot should change when you select new items from the list. We would ordinarily do this directly in Altair, but because we don't have control over the way you created your visualization, it's easiest for us to use the widgets built into Jupyter.

It should look something like this:

[18]:
```
# note that it might take a few seconds for the images to download
# depending on your internet connection

output = widgets.Output()

def clicked(b):
    output.clear_output()
    with output:
        highlight = filterdrop.value
        if (highlight == ""):
            print("please enter a query")
        else:
            genMDSPlot(highlight).display()


featurecount = bobross[imgfeatures].sum()

filterdrop = widgets.Dropdown(
    options=list(featurecount[featurecount > 2].keys()),
    description='Highlight:',
    disabled=False,
)
```

```
filterdrop.observe(clicked)

display(filterdrop,output)

with output:
    genMDSPlot('Barn').display()
```

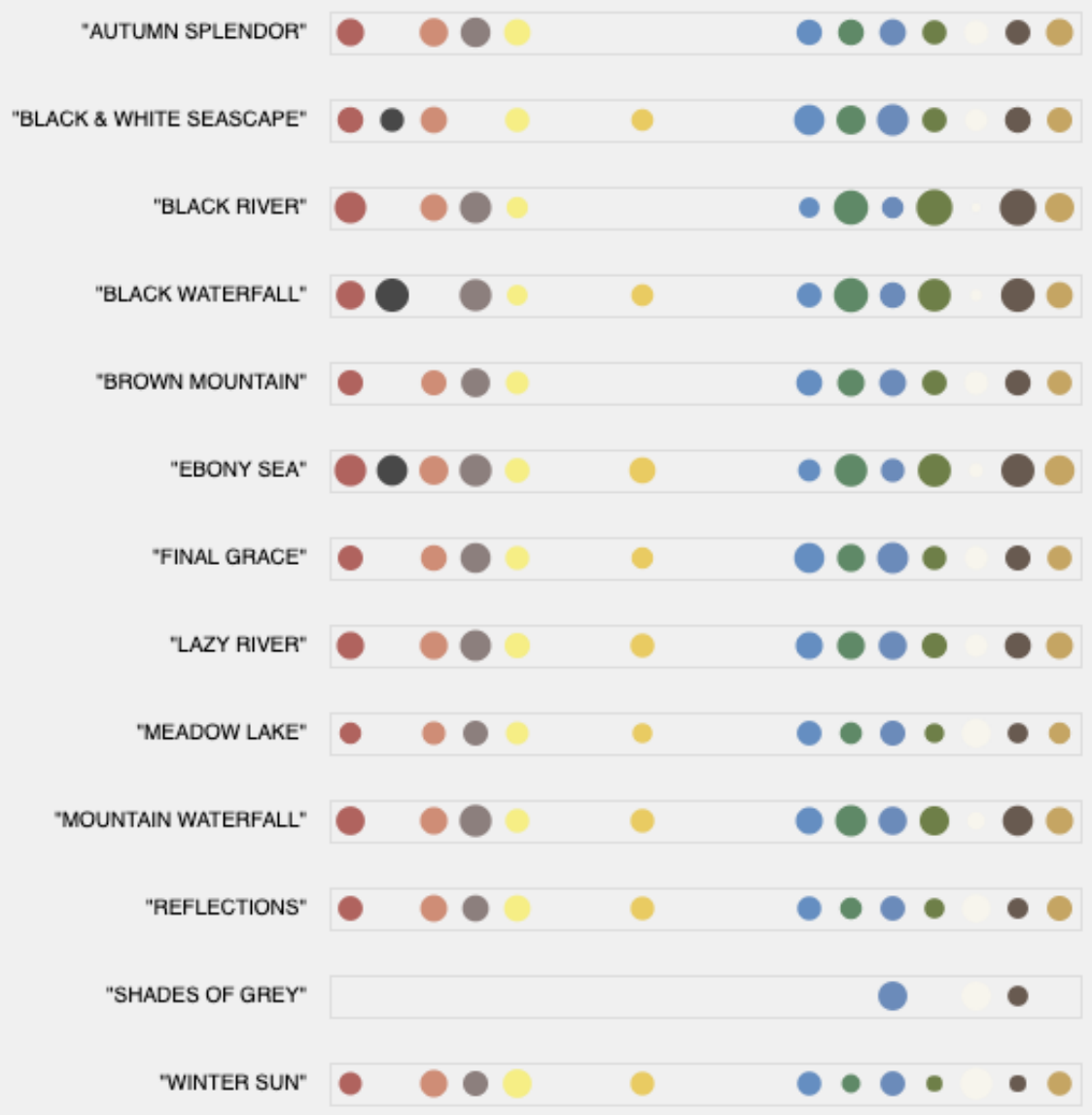
```
Dropdown(description='Highlight:', options=('Barn', 'Beach', 'Bridge', 'Bushes',␣
↪'Cabin', 'Cactus', 'Cirrus cl…
```

```
Output()
```

1.7 Problem 4 (30 points: 25 for solution, 5 for explanation)

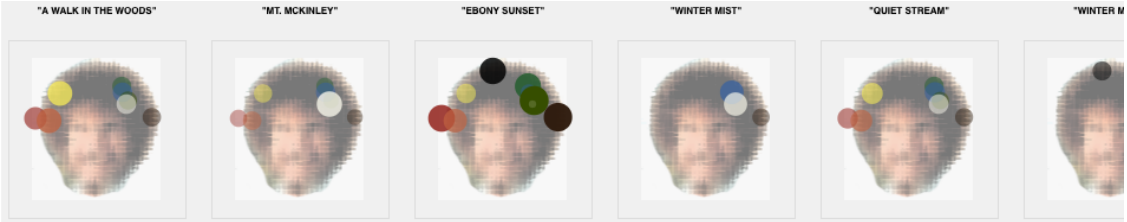
Your last problem is fairly open-ended in terms of visualization. We would like to analyze the colors used in different images for a given season as a small multiples plot. You can pick how you represent your small multiples, but we will ask you to defend your choices below. You must implement the function `colorSmallMultiples(season)` that takes a season number as input (e.g., 2) and returns an Altair chart.

You can go something as simple as this:



This visualization has a row for every painting and a colored circle (in the color of the paint). The circle is sized based on the amount of the corresponding paint that is used in the image.

You can also go to something as crazy as this:



Here, we've overlaid circles as curls in Bob's massive hair. We're not claiming this is an effective solution, but you're welcome to do this (or anything else) as long as you describe the pros and cons

of your choices. And, yes, we generated both examples using Altair.

Again, the relevant columns are available are listed in `rosspaints` (there are 18 of them). The values range from 0 to 1 based on the fraction of pixel color allocated to that specific paint. The `rosspainthex` has the corresponding hex values for the paint color.

[19]:
```
season_list = []
for i in bobross.index:
    season_list.append(bobross.iloc[i]['EPISODE'][1])

color_df = bobross[rosspaints].copy()
color_df['TITLE'] = bobross['TITLE']
color_df['Season'] = season_list
color_df.set_index(['Season'], inplace=True)

def colorSmallMultiples(season):
    df = color_df.filter(like=season, axis=0)
    new_df = pd.melt(df, id_vars=['TITLE'], var_name='Paints',␣
 ↪value_name='Value')

    return alt.Chart(new_df).mark_circle().encode(
        alt.X('Paints', axis=None),
        alt.Y('TITLE:N', axis=alt.Axis(ticks=False, grid=False)),
        alt.Size('Value:Q', legend=None),
        alt.Color('Paints', scale=alt.Scale(domain=rosspaints,␣
 ↪range=rosspainthex), legend=None)
    ).configure_view(strokeWidth=0)
    # return an Altair chart
    # season is the integer representing the season of the show are interested␣
 ↪in. Limit your images
    # to that season in the small multiples display.

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

[20]:
```
# run this to test your code for season 1
colorSmallMultiples("1")
```

[20]: alt.Chart(…)

[21]:
```
# run this to test your code for season 2
colorSmallMultiples("2")
```

[21]: alt.Chart(…)

1.7.1 Explain your choices

Explain your design here. Describe the pros and cons in terms of visualization principles.

I don't have much time to create a new kind/style of this question, so I just reproduce the example provided by the instruction. I believe the dot/point mark visualization does the great job to show the proportion of colors which Bob Ross is using in his paintings. The size of dots shows the values of each color and the color of dots means the color Bob Ross used. The only disadvantage of this visualization is that the reader cannot figure out the exact values of dots. For example, I can read in the paint of "Winter at the farm", Bob Ross used more blue color than others. However, I can't tell how much more he used the blue color. If I have more time, I will definitely create a creative visualization containing more information for this question.