

Information Visualization II

School of Information, University of Michigan

Week 1:

- Multivariate/Multidimensional + Temporal

Assignment Overview

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](https://altair-viz.github.io/) (<https://altair-viz.github.io/>)

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.

Resources:

- Article by [FiveThirtyEight](https://fivethirtyeight.com) (<https://fivethirtyeight.com>) available [online](https://fivethirtyeight.com/features/a-statistical-analysis-of-the-work-of-bob-ross/) (<https://fivethirtyeight.com/features/a-statistical-analysis-of-the-work-of-bob-ross/>) (Hickey, 2014)
- The associated dataset on [Github](https://github.com/fivethirtyeight/data/tree/master/bob-ross) (<https://github.com/fivethirtyeight/data/tree/master/bob-ross>)
- A dataset of all the [paintings from the show](https://github.com/jwilber/Bob_Ross_Paintings) (https://github.com/jwilber/Bob_Ross_Paintings)

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.

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

```
In [1]: # load up the resources we need
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
```

Bob Ross

Today's assignment will have you working with artwork created by [Bob Ross](https://en.wikipedia.org/wiki/Bob_Ross) (https://en.wikipedia.org/wiki/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](https://www.youtube.com/watch?v=Fw6odlNp7_8) (https://www.youtube.com/watch?v=Fw6odlNp7_8).



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 \(https://fivethirtyeight.com/features/a-statistical-analysis-of-the-work-of-bob-ross/\)](https://fivethirtyeight.com/features/a-statistical-analysis-of-the-work-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.

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

# hex values for the paints above
rosspainthex = ['#94261f', '#c06341', '#614f4b', '#f8ed57', '#5c2f08', '#e6ba25',
                '#000000', '#ffffff', '#000000', '#36373c', '#2a64ad', '#215c2c',
                '#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 mount',
               'Nighttime', 'Ocean', 'Oval frame', 'Palm trees', 'Path',
               'Person', 'Portrait', 'Rectangle 3d frame', 'Rectangular fra',
               'River or stream', 'Rocks', 'Seashell frame', 'Snow',
               'Snow-covered mountain', 'Split frame', 'Steve ross',
               'Man-made structure', 'Sun', 'Tomb frame', 'At least one tre',
               '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')
```

```
Out[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).

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

Out[3]:

	EPISODE	TITLE	RELEASE_DATE	Apple frame	Aurora borealis	Barn	Beach	Boat	Bridge	Building
355	S29E12	"AURORA'S DANCE"	11/9/93	0	1	0	0	0	0	0
52	S05E07	"BUBBLING BROOK"	2/13/85	0	0	0	0	0	0	0
31	S03E07	"QUIET INLET"	2/15/84	0	0	0	0	0	0	0
128	S11E10	"SUNSET OVER THE WAVES"	3/4/87	0	0	0	1	0	0	0
371	S31E03	"WINDING STREAM"	3/8/94	0	0	0	0	0	0	0

5 rows x 114 columns

```
In [4]: # see all the columns
[print(i) for i in bobross.columns];
Black_Gesso
Bright_Red
Burnt_Umber
Cadmium_Yellow
Dark_Sienna
Indian_Red
Indian_Yellow
Liquid_Black
Liquid_Clear
Midnight_Black
Phthalo_Blue

Phthalo_Green
Prussian_Blue
Sap_Green
Titanium_White
Van_Dyke_Brown
Yellow_Ochre
Alizarin_Crimson
alizarin crimson
```

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.

```
In [5]: # 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', 'Cumulus clouds', 'Deciduous 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', 'Rectangular 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.

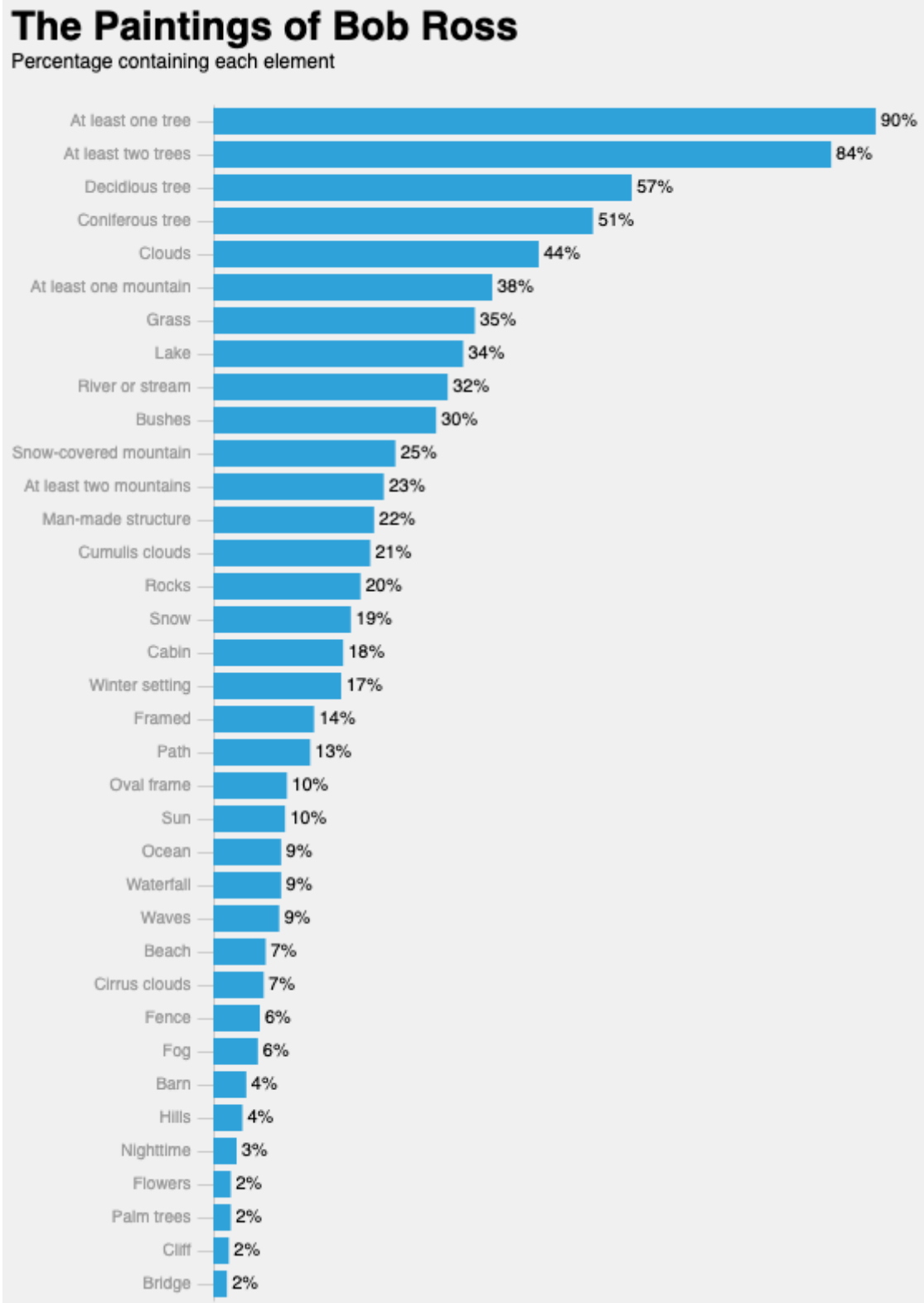
```
In [6]: # run to see what's inside
print("paint names",rosspaints)
print("")
print("hex values", rosspainthex)
```

```
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']
```

Problem 1 (20 points)

As a warmup, we're going to have you recreate the [first chart from the Bob Ross article \(assets/bob_ross_538.png\)](#) (source: [Statistical Analysis of Bob Ross \(https://fivethirtyeight.com/features/a-statistical-analysis-of-the-work-of-bob-ross/\)](#)). 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. Also, **note the order of the bars. It's not arbitrary, please re-create it.**

```

In [7]: def makeBobRossBar():
import math

# ==== prep dataframe ====
# use only the features columns
features_df = bobross[imgfeatures]

# count the number of times each feature appears among the paintings and
features_count = pd.DataFrame(features_df.sum(axis = 0)).rename({0: 'count'})
features_count = features_count[features_count['count'] > 5]

# calculate the percent of paintings a feature appears in
features_count['perc'] = (features_count['count'] / 381)
# features_count['perc'] = features_count['perc'].round()

# reset index
features_count = features_count.reset_index()

# sort order
sort_order = list(features_count['index'])

# create the bars
bars = alt.Chart(features_count).mark_bar(size=20).encode(
    x=alt.X(
        'perc:Q',
        axis=None
    ),
    y=alt.Y(
        'index:N',
        sort=sort_order,
        axis=alt.Axis(labelPadding=5, title='')
    )
)

# perc text
text = bars.mark_text(
    align='left',
    baseline='middle',
    dx=3
).encode(
    text=alt.Text('perc:Q', format='0.0%')
)

# create and customize figure
painting_ftrs = (text + bars).properties(
    title={"text": "The Paintings of Bob Ross", "subtitle": "Percentage"}
).configure_title(
    fontSize=25,
    anchor='start',
    offset=20
).configure_axis(
    grid=False
).configure_view(
    strokeWidth=0

```



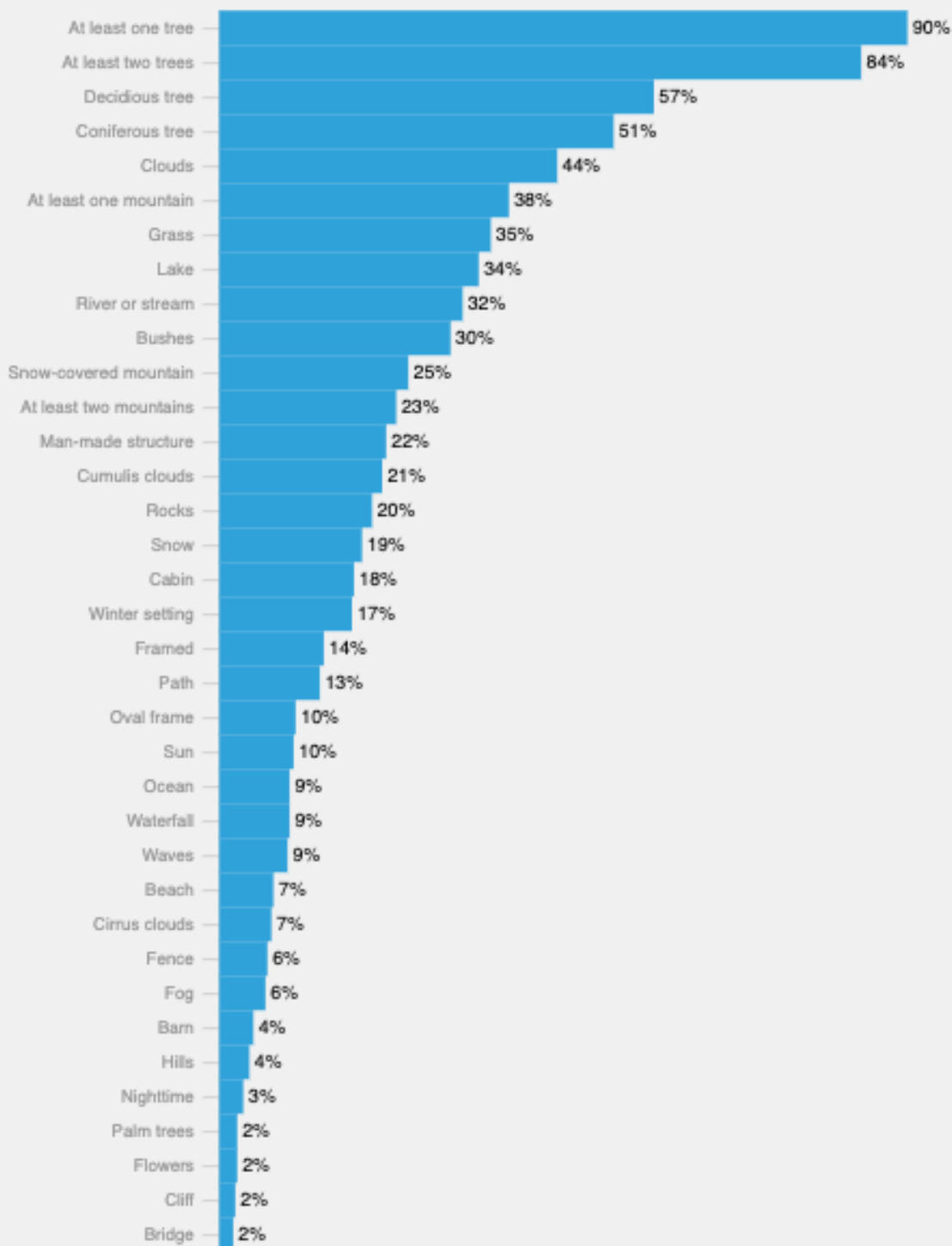
```
).configure_scale(  
    bandPaddingInner=0.2  
)  
  
return painting_fts
```

```
In [8]: # run this code to validate  
alt.themes.enable('fivethirtyeight')  
makeBobRossBar()
```

Out[8]:

The Paintings of Bob Ross

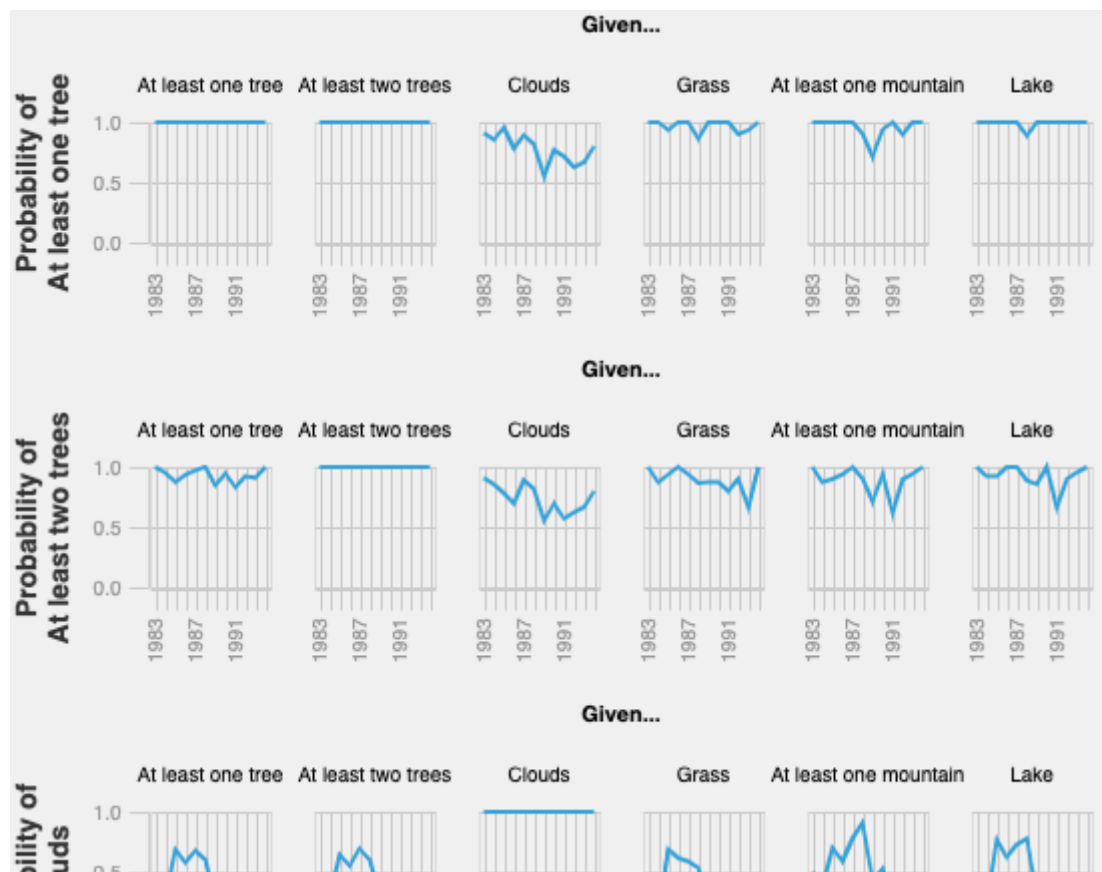
Percentage containing each element



Problem 2 (25 points)

The 538 article ([Statistical Analysis of Bob Ross](https://fivethirtyeight.com/features/a-statistical-analysis-of-the-work-of-bob-ross/) (<https://fivethirtyeight.com/features/a-statistical-analysis-of-the-work-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 \(assets/matrix_full.png\)](#). 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 Conditional Probability/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 check 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.

```

In [9]: def condprobability(df,pair_df):
        """Calculate conditional probabilities of features

        df: original dataframe with 1/0 data, containing only columns of interest
        pair_df: dataframe of the pairwise comparisons and years

        Returns: the input pair_df with an additional column for pairwise conditional probabilities
        """

        probs = []
        for index,row in pair_df.iterrows():

            # filter bobross dataframe by the year in the current row
            year_df = df[df['year']==row['year']]

            # count number of paintings created in that year, used in prob calculation
            num_paint = len(year_df)

            # calculate the conditional probability for the two columns in question
            # calculate the conditional probability for each pair in pair_df (e.g., Grass and cloud)
            # equation:  $p(\text{col}_1|\text{col}_2) = p(\text{col}_1 \text{ and } \text{col}_2)/p(\text{col}_1)$ 

            # two features of comparison
            key1 = row['key1']
            key2 = row['key2']

            # k1_prob: probability that key1 (e.g., Grass) appears in a painting
            # k1k2_prob: probability that key1 and key 2 (e.g., Grass and cloud) appear together
            # cond_prob: probability that key1 appears given that key 2 appears
            k1_prob = year_df[key1].sum()/num_paint
            k1k2_prob = len(year_df[(year_df[key1]==1) & (year_df[key2]==1)])/num_paint
            cond_prob = k1k2_prob/k1_prob

            # save the cond prob to pair_df prob column
            probs.append(cond_prob)

        # create a new column with conditional probabilities
        pair_df['prob'] = probs

        return pair_df

```

```

In [10]: def makeBobRossCondProb(bobross,totest=['At least one tree','At least two t
        """
        bobross: the original bobross dataframe.
        totest: columns of interest for pairwise comparisons

        Returns a small multiples chart using the totest columns.
        """

        # ==== prepare dataframe ====

        # the years we would like to see trends among
        years = bobross['year'].unique()

        # filter the original bob ross dataframe to contain only columns of int
        df = bobross[['At least one tree','At least two trees','Clouds','Grass']

        # create the new dataframe with cols key1, key2, year (all the unique c
        data = []
        for tag1 in totest:
            for tag2 in totest:
                for year in years:
                    data.append([tag1,tag2,year])
        pair_df = pd.DataFrame(data,columns=['key1','key2','year'])

        # add a conditional probability column to the new dataframe,
        # cond prob of the occurence key1 in a painting of that year given the
        pair_df = condprobability(df=df,pair_df=pair_df)

        # ==== create chart ====

        for key1 in totest:
            filtered_df = pair_df[pair_df['key1']==key1]

            temp_chart = alt.Chart(filtered_df).mark_line().encode(
                x=alt.X(
                    'year:O',
                    axis=alt.Axis(title='')
                ),
                y=alt.Y(
                    'prob:Q',
                    axis=alt.Axis(title=['Probability of',str(key1)])
                ),
                facet=alt.Facet(
                    'key2:N',
                    columns=6,
                    title='Given...'
                )
            ).properties(
                width=100,
                height=100
            )

            try:
                chart = alt.vconcat(chart,temp_chart)
            except Exception as e:
                chart = temp_chart

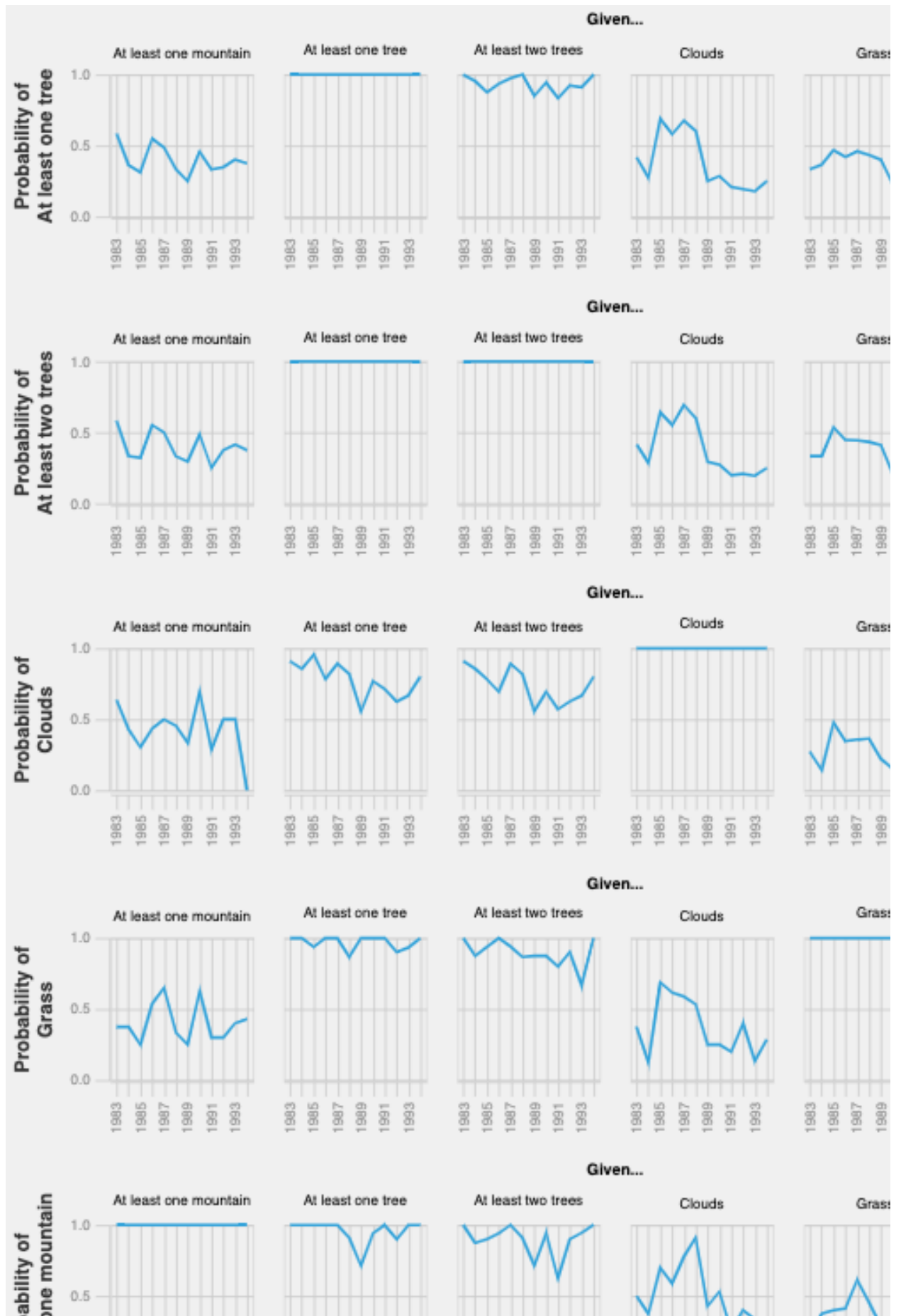
```

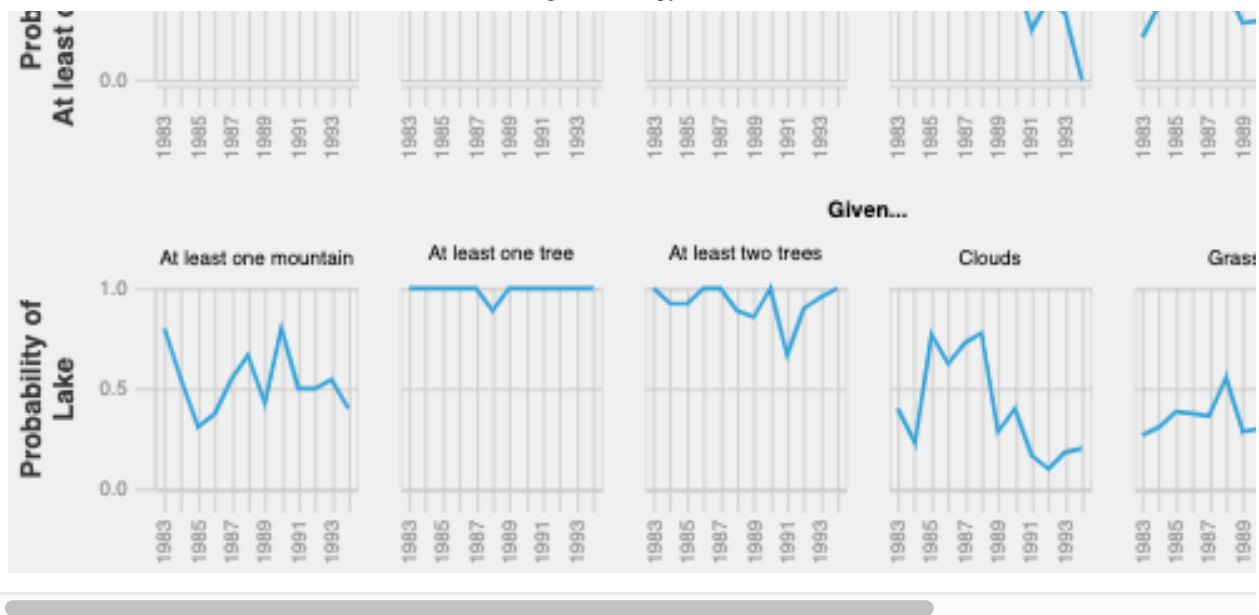
```
return chart
```



```
In [11]: # run this cell to test your code
makeBobRossCondProb(bobross)
```

```
Out[11]:
```





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.

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](https://scikit-learn.org/stable/modules/generated/sklearn.manifold.MDS.html) (<https://scikit-learn.org/stable/modules/generated/sklearn.manifold.MDS.html>)

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


```
In [12]: # create the seed
seed = np.random.RandomState(seed=3)

# generate the MDS configuration, we want 2 components, etc. You can tweak
# the settings change the layout
mds = manifold.MDS(n_components=2, max_iter=3000, eps=1e-9, random_state=seed)

# 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
bobross['x'] = [x[0] for x in pos]
bobross['y'] = [x[1] for x in pos]
```

```
In [13]: bobross
```

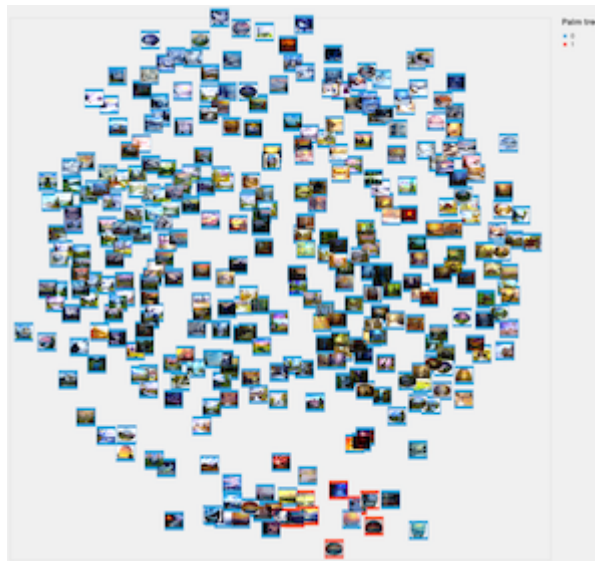
```
Out[13]:
```

	EPISODE	TITLE	RELEASE_DATE	Apple frame	Aurora borealis	Barn	Beach	Boat	Bridge	Buildir
0	S01E01	"A WALK IN THE WOODS"	1/11/83	0	0	0	0	0	0	
1	S01E02	"MT. MCKINLEY"	1/11/83	0	0	0	0	0	0	
2	S01E03	"EBONY SUNSET"	1/18/83	0	0	0	0	0	0	
3	S01E04	"WINTER MIST"	1/25/83	0	0	0	0	0	0	
4	S01E05	"QUIET STREAM"	2/1/83	0	0	0	0	0	0	
...
376	S31E08	"TRAIL'S END"	4/12/94	0	0	0	0	0	0	
377	S31E09	"EVERGREEN VALLEY"	4/19/94	0	0	0	0	0	0	
378	S31E10	"BALMY BEACH"	4/26/94	0	0	0	1	0	0	
379	S31E12	"IN THE MIDST OF WINTER"	5/10/94	0	0	1	0	0	0	
380	S31E13	"WILDERNESS DAY"	5/17/94	0	0	0	0	0	0	

381 rows × 116 columns

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 \(https://altair-viz.github.io/user_guide/marks.html#user-guide-image-mark\)](https://altair-viz.github.io/user_guide/marks.html#user-guide-image-mark). 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")` :

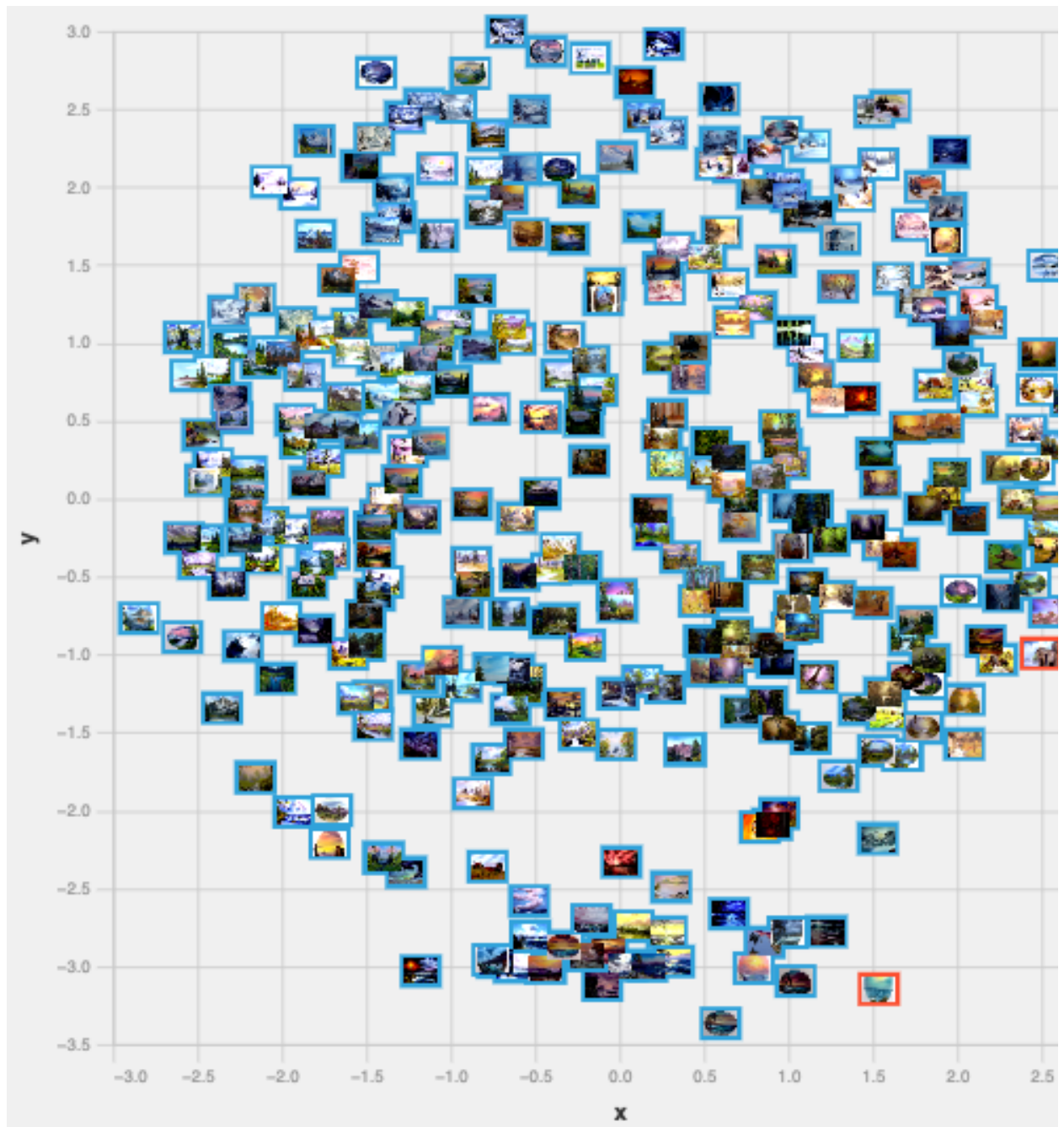


Click [here \(assets/mds_large.png\)](#) for a large version of this image. Notice the orange boxes indicating where the Palm tree images are. Note that we have styled the MDS plot to not have axes. Recall that these are meaningless in MDS 'space' (this is not a scatterplot, it's a projection).

```
In [14]: def genMDSPlot(key):  
    '''  
    key is a string indicating which images should be visually highlighted  
    '''  
  
    # ==== images ====  
  
    img = alt.Chart(bobross).mark_image(  
        width=20,  
        height=20  
    ).encode(  
        x='x',  
        y='y',  
        url='img_url'  
    ).properties(  
        width=600,  
        height=600  
    )  
  
    # ==== rectangles ====  
  
    rect = alt.Chart(bobross).mark_rect(  
        width=25,  
        height=20  
    ).encode(  
        x='x',  
        y='y',  
        color=str(key)+' :N'  
    )  
  
    # ==== layer charts ====  
  
    chart=rect+img  
  
    return chart
```

```
In [15]: genMDSPlot('Boat')
```

```
Out[15]:
```



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:

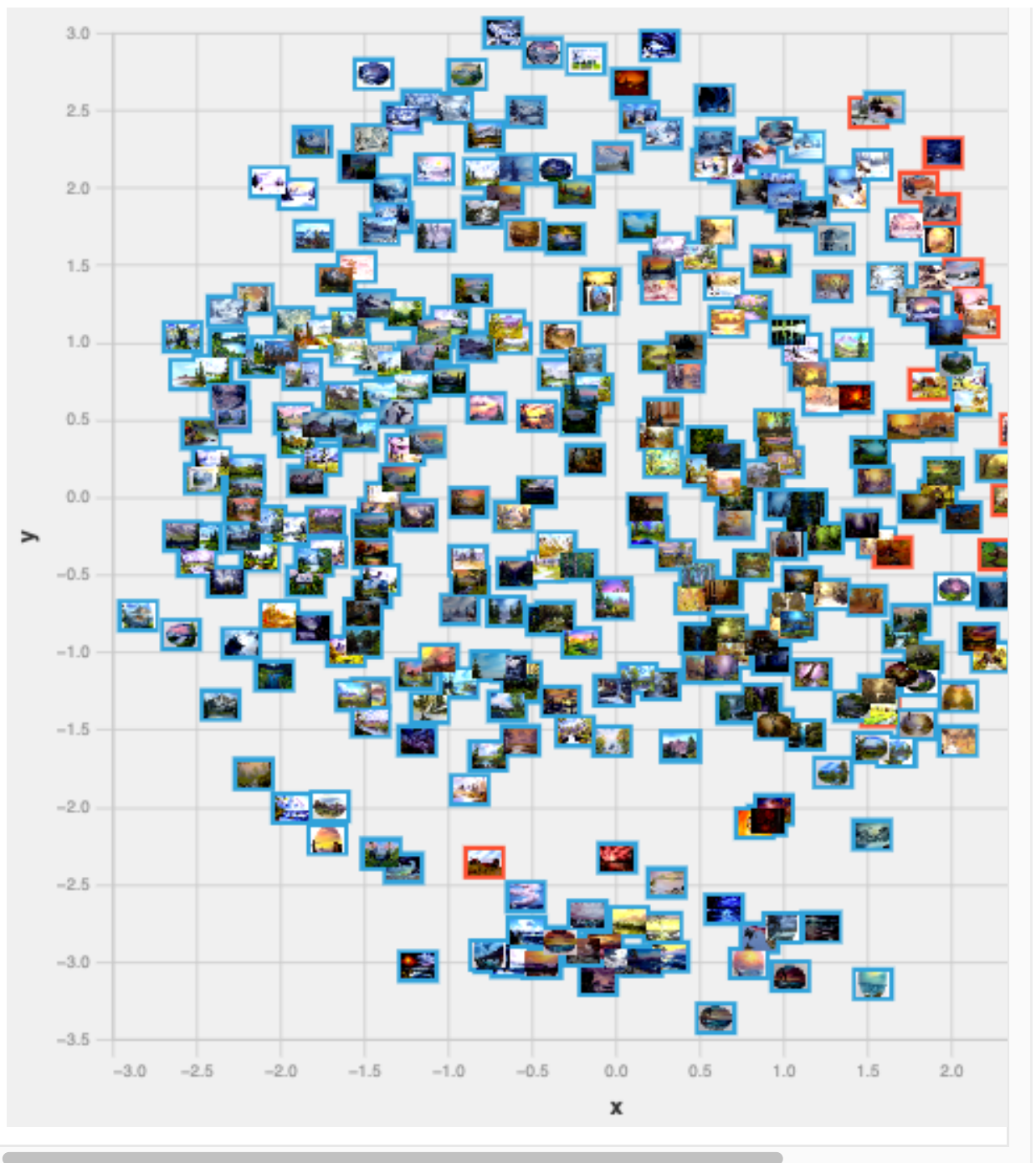


It may take a few seconds the first time you run this to download all the images.

```
In [16]: # 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()
```

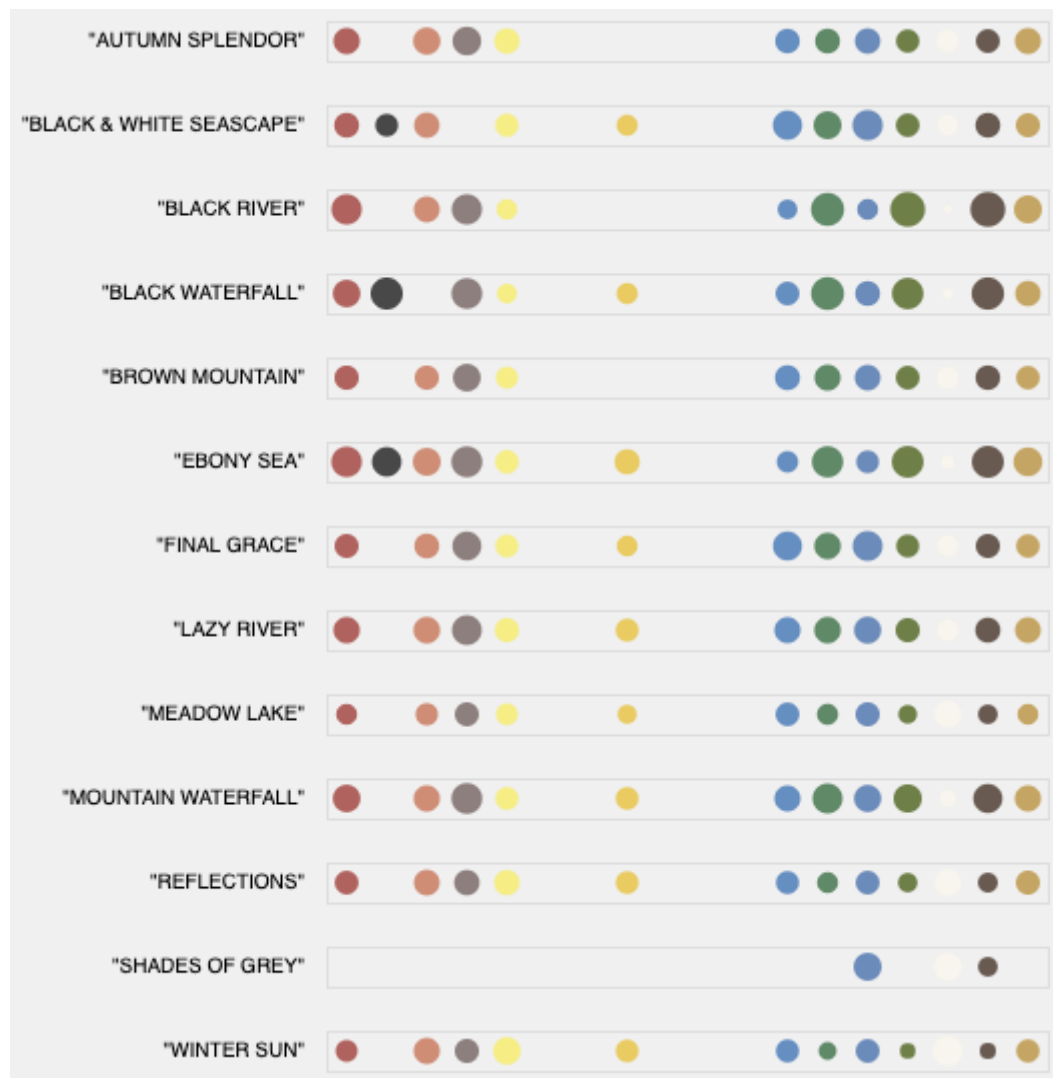
Highlight:



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

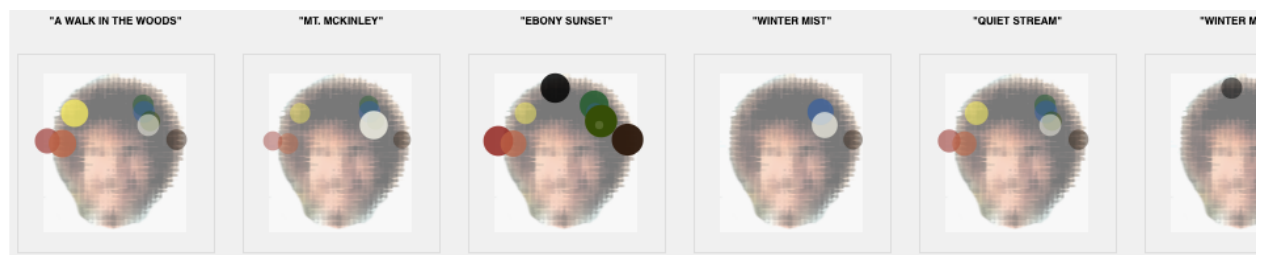
Your last problem is fairly open-ended in terms of visualization. We would like to analyze the colors used in different images for a given season as a small multiples plot. You can pick how you represent your small multiples, but we will ask you to defend your choices below. You must implement the function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

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.

Make sure your visualization is actually a small multiple approach. There should be "mini" visualizations for each painting.


```
In [17]: # the paints Bob used
rosspaints = ['alizarin crimson', 'bright red', 'burnt umber', 'cadmium yellow',
              'indian yellow', 'indian red', 'liquid black', 'liquid clear', 'b',
              'midnight black', 'phthalo blue', 'phthalo green', 'prussian blu',
              'titanium white', 'van dyke brown', 'yellow ochre']

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

```
In [18]: def colorSmallMultiples(season):

    # simplify bobross df to only cols of interest
    cols = [
        'TITLE',
        'season',
        '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'
    ]
    colors_df = bobross[cols]

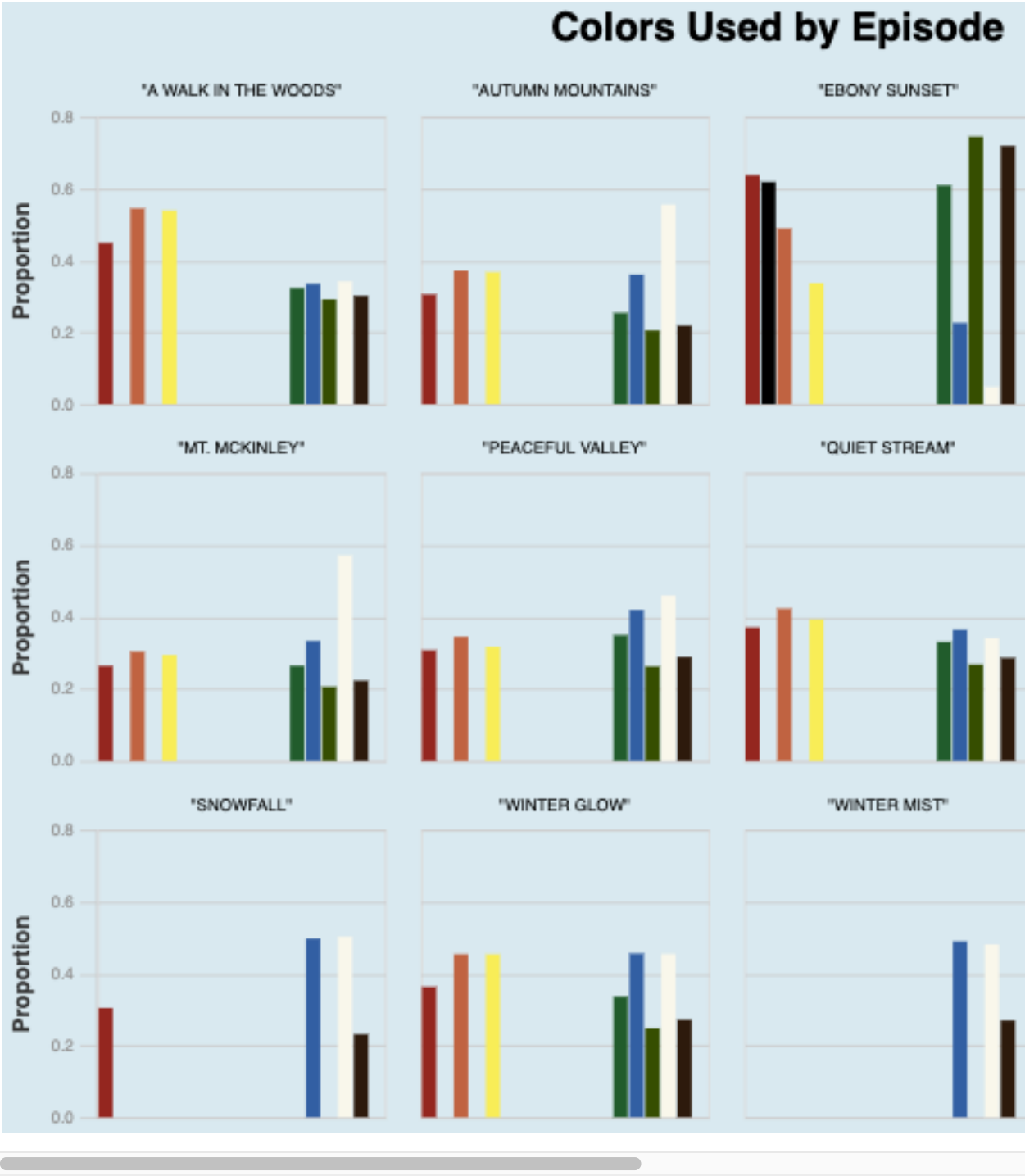
    # filter paintings (i.e., rows) from a particular season of interest an
    season_df = colors_df[colors_df['season']==season].melt(id_vars=['seaso

    plot = alt.Chart(season_df).mark_bar().encode(
        y=alt.Y('value:Q', axis=alt.Axis(title='Proportion')),
        x=alt.X('Color:N', axis=None),
        color=alt.Color('Color:N', scale=alt.Scale(domain=rosspaints, range=r
        facet=alt.Facet('TITLE:N', columns=4, title='')
    ).properties(
        width=175,
        height=175,
        title='Colors Used by Episode'
    ).configure_axis(
        grid=False
    ).configure(background='#D9E9F0').configure_title(anchor='middle')

    return plot
```

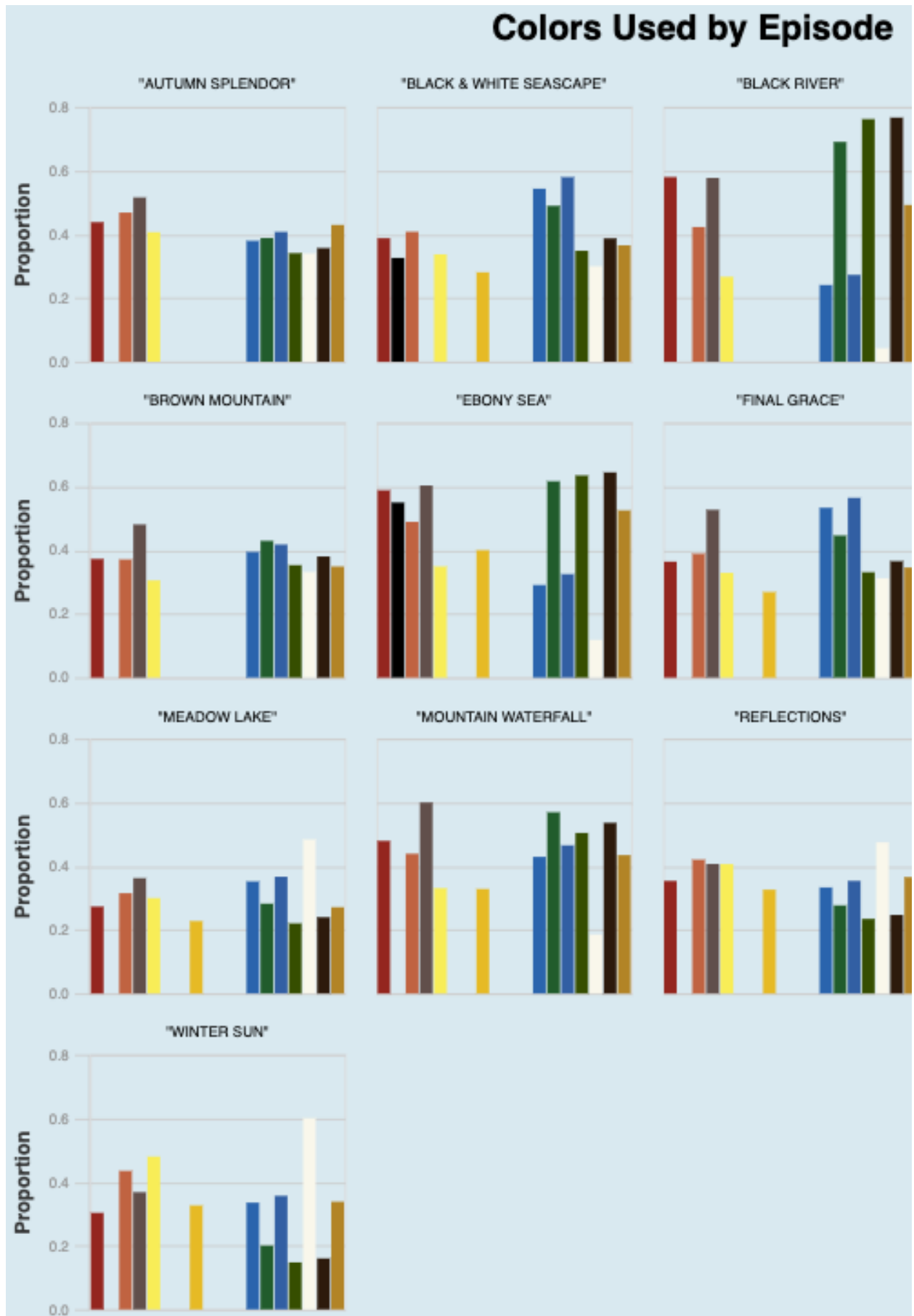
```
In [19]: # run this to test your code for season 1
colorSmallMultiples(1)
```

Out[19]:



```
In [20]: # run this to test your code for season 2
colorSmallMultiples(2)
```

Out[20]:



Explain your choices

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

Given the constraint that the multiple of the small multiples chart should be the painting (i.e., the episode), I decided to create a facet chart for each episode in the season. This decision would allow the reader to compare the distribution of colors across the paintings quickly.

I also decided on an x and y plot by considering Mackinlay's APT algorithm. Here we have the proportion of color, a quantitative variable, and the colors used, a nominal variable. According to the algorithm, the optimal choice would be to have both variables represented by position, hence the x and y plot.

Lastly, I decided to use bars rather than other encodings because the bars emphasize the proportion value and make comparison across the bars easier. Using circles, for example, would have required some additional scanning and processing to compare the heights of the positions of the various circles.