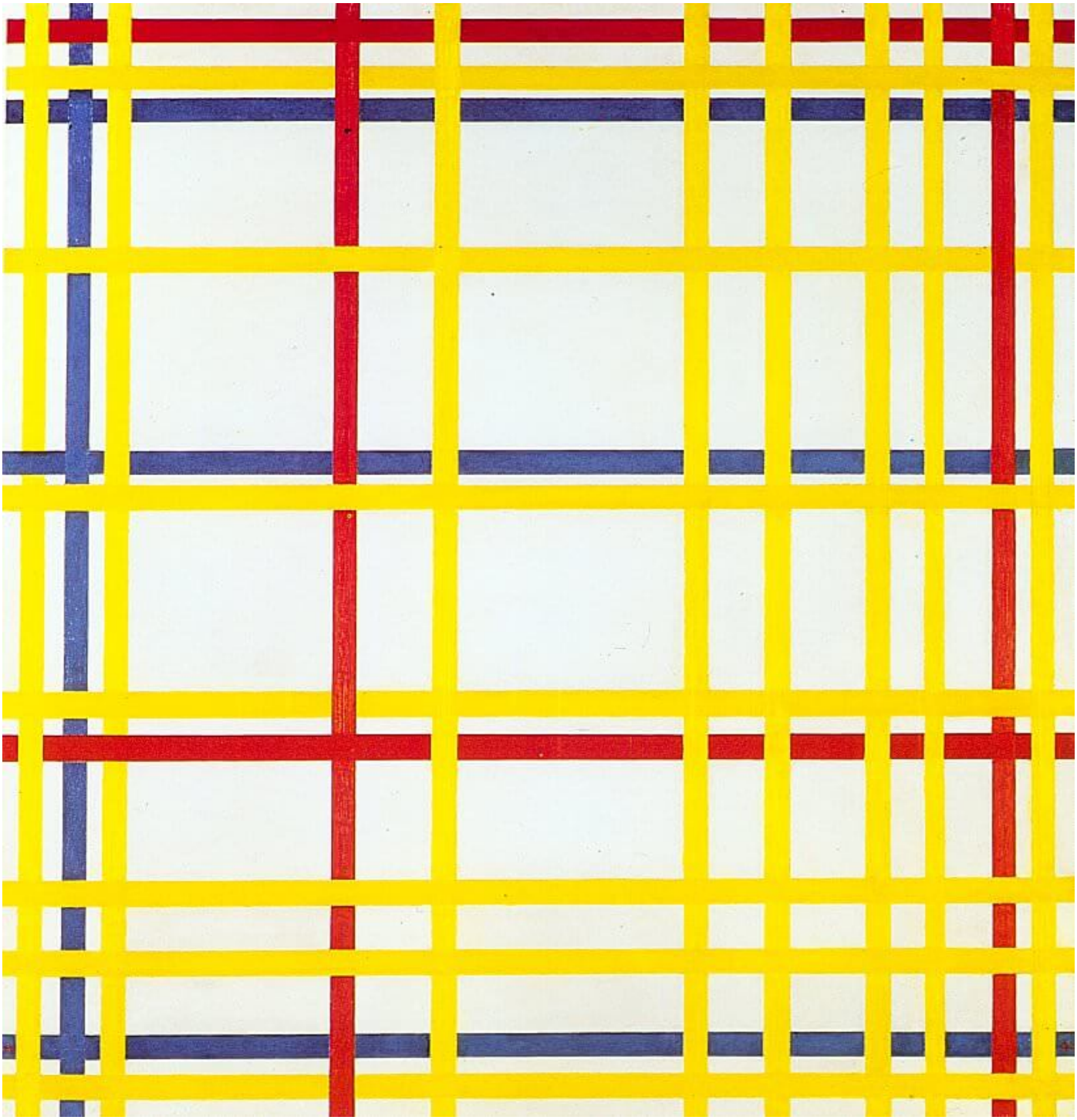
Mondrian is one of the most influential abstract artist.\ Many of his work is made of artistic combinations of lines and squares, such as in his famous "New York" piece below.\ In this Jupyter Notebook, I used Python libraries and functions to make plots that mimick his artwork, and recorded my thought process.



First thing first - let's import some useful python libraries!\ The artwork is consist of 3 interwined layers of lines, with the yellow lines as the topmost layer.\ I plan to randomly generate some offsets to determine the positions of the lines on the plots, and use matplotlib to draw the lines.\ Here, numpy and matplotlib are used.

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

Optional step: use random seed to make the figures reproducible. Otherwise, they will change each time they are run.

In [2]:
```
np.random.seed(5)
```

Then, define two functions that would roll the dice for the positions of lines on the plot.

- The ramdom_offsets samples numbers from a random distribution. Those numbers are later used to generate the offsets of vertical lines in the plots.
- The uniform_offsets samples numbers from a uniform distribution, which are later used as offsets of the horizontal lines in the plots.

In [3]:
```
# num is the total number of offsets.
def random_offsets(num):
    # Draw random samples from a normal (Gaussian) distribution.
    # The the mean is 20 and the standard deviation is 60.
    x = 20 + np.random.normal(0, 60, num)
    return x
```

In [4]:
```
def uniform_offsets(num,umin,umax):
    # Draw samples from a uniform distribution.
    u = np.random.uniform(umin,umax,num)
    return u
```

Test whether the functions work as expected.

In [5]:
```
voffsets = random_offsets(10)
print(voffsets)
# vmin and vmax are later used to set the boundaries of the horizontal lines.
vmin = min(voffsets)
vmax = max(voffsets)
print(vmin,vmax)
```

```
[ 4.64736492e+01  1.47790886e-01  1.65846271e+02  4.87447222e+00
  2.65765905e+01  1.14948867e+02 -3.45539443e+01 -1.54981995e+01
  3.12561936e+01  2.07802533e-01]
-34.55394429137451 165.8462712204668
```

In [6]:
```
hoffsets = uniform_offsets(10,vmin,vmax)
print(hoffsets)
# hmin and hmax are used to set the boundaries of the vertical lines.
hmin = min(hoffsets)
hmax = max(hoffsets)
print(hmin,hmax)
```

```
[ 53.88451908  -2.82861268 141.7856264   20.37304176  48.45884281
  24.78053804  91.45528816  81.64567783  85.671996    18.71626415]
-2.8286126841081227 141.7856263982116
```

The two dice rolling functions look good.\ So now I can start making the plot. \ First, plot the vertical lines in the first layer:

In [7]:
```
fig, ax = plt.subplots(figsize=(20,15))
ax.axis('off')
plt.vlines(voffsets,hmin,hmax, linewidth=20, color='yellow')
```

Out[7]:
```
<matplotlib.collections.LineCollection at 0x117e547f0>
```

Then, add the horizontal lines in the first layer.

In [8]:
```
fig, ax = plt.subplots(figsize=(20,15))
ax.axis('off')
plt.vlines(voffsets,hmin,hmax, linewidth=20, color='yellow')
plt.hlines(hoffsets,vmin,vmax, linewidth=20, color='yellow')
```

Out[8]:
```
<matplotlib.collections.LineCollection at 0x117eba7c0>
```

Now the first layer of the image is done!\ Define a function to generate offsets for all 3 layers.

In [9]:
```
def mondrian_offsets(num1,num2,num3):
    voffsets_list = []
    hoffsets_list = []
    for num in [num1,num2,num3]:
        voffsets = random_offsets(num)
        hoffsets = uniform_offsets(num,min(voffsets)-4,max(voffsets)-5)
        voffsets_list.append(voffsets)
        hoffsets_list.append(hoffsets)
    return voffsets_list,hoffsets_list
```

Test the function. \ It returns 2 lists, each have 3 arrays in them, corresponding to offsets in each layer.\ The numbers of elements in the array correspond to the number of lines.

In [10]:
```
mondrian_offsets(2,2,10)
```

Out[10]:
```
([array([-38.83647311, -31.41118928]),
  array([79.78638961, 62.74527625]),
  array([ 20.19733058,  13.64417348,  67.58319917, -17.89429779,
          19.62830549,  13.93594329,  16.86151095,  34.95305951,
          31.85960546, 100.09091445])],
 [array([-40.73178178, -41.91017656]),
  array([59.11314859, 68.01163164]),
  array([ 68.2842312 ,  34.83546694, -18.48867533, -11.77207177,
          -8.85584989,   7.49766493,  90.9865225 ,  52.01298798,
          73.64286935,  44.32892467])])
```

I realize despite the above function looks clean, when making the plots I need to set the beginning and the end of each line, as well as xlim and ylim. \ So I decide to first generate the offsets of the top layer and use

that as references to determine the boundaries.\ Here is an updated version of the mondrian_offsets function:

In [11]:
```
def mondrian_offsets(set1=2,set2=2,set3=10):
    # Make empty list to store the 3 sets of randomly sampled offsets.
    voffsets_list = []
    hoffsets_list = []
    # Roll the dice for voffsets in the uppermost layer.
    voffsets3 = random_offsets(set3)
    # Define some boundaries by voffsets.
    left = min(voffsets3)
    right = max(voffsets3)
    # Roll the dice for hoffsets.
    hoffsets3 = uniform_offsets(set3,left-4,right-5)
    # Determine the upper and lower boundaries by hoffsets.
    bottom = min(hoffsets3)
    up = max(hoffsets3)
    # Sample from the bottom two layers and record the offsets in the lists.
    for num in [set1,set2]:
        voffsets = random_offsets(num)
        hoffsets = uniform_offsets(num,left,right)
        voffsets_list.append(voffsets)
        hoffsets_list.append(hoffsets)
    # Append the offsets of the uppermost layer in the end.
    voffsets_list.append(voffsets3)
    hoffsets_list.append(hoffsets3)
    # Record the boundaries of the lines, later used in vline/hline arguments.
    # Define the bounds as a tuple for easy unpacking.
    bounds = (left,right,bottom,up)
    return voffsets_list,hoffsets_list,bounds
```

In addition to the offset list, this function now also returns the limits of the lines.

In [12]:
```
mondrian_offsets(set1=2,set2=2,set3=10)
```

Out[12]:
```
([array([54.08796306, 20.765499  ]),
  array([91.34364355, 15.58440088]),
  array([ 88.36057873,  49.66642389,  -0.18017555,  13.96313922,
         104.80388108,  33.27524737, -58.646388  , -21.37391392,
         -14.65079399,  89.13228622])],
 [array([  2.07744894, -56.62460435]),
  array([25.48767121, 14.35178164]),
  array([ 41.13946336,  17.50901995,  32.14330237,  89.94304447,
          90.63195221, -44.50192315,  74.38934322,  -6.43402164,
         -46.2669704 ,  -0.36148205])],
 (-58.646388002784 73,
  104.80388107530703,
  -46.266970395837845,
  90.63195220665969))
```

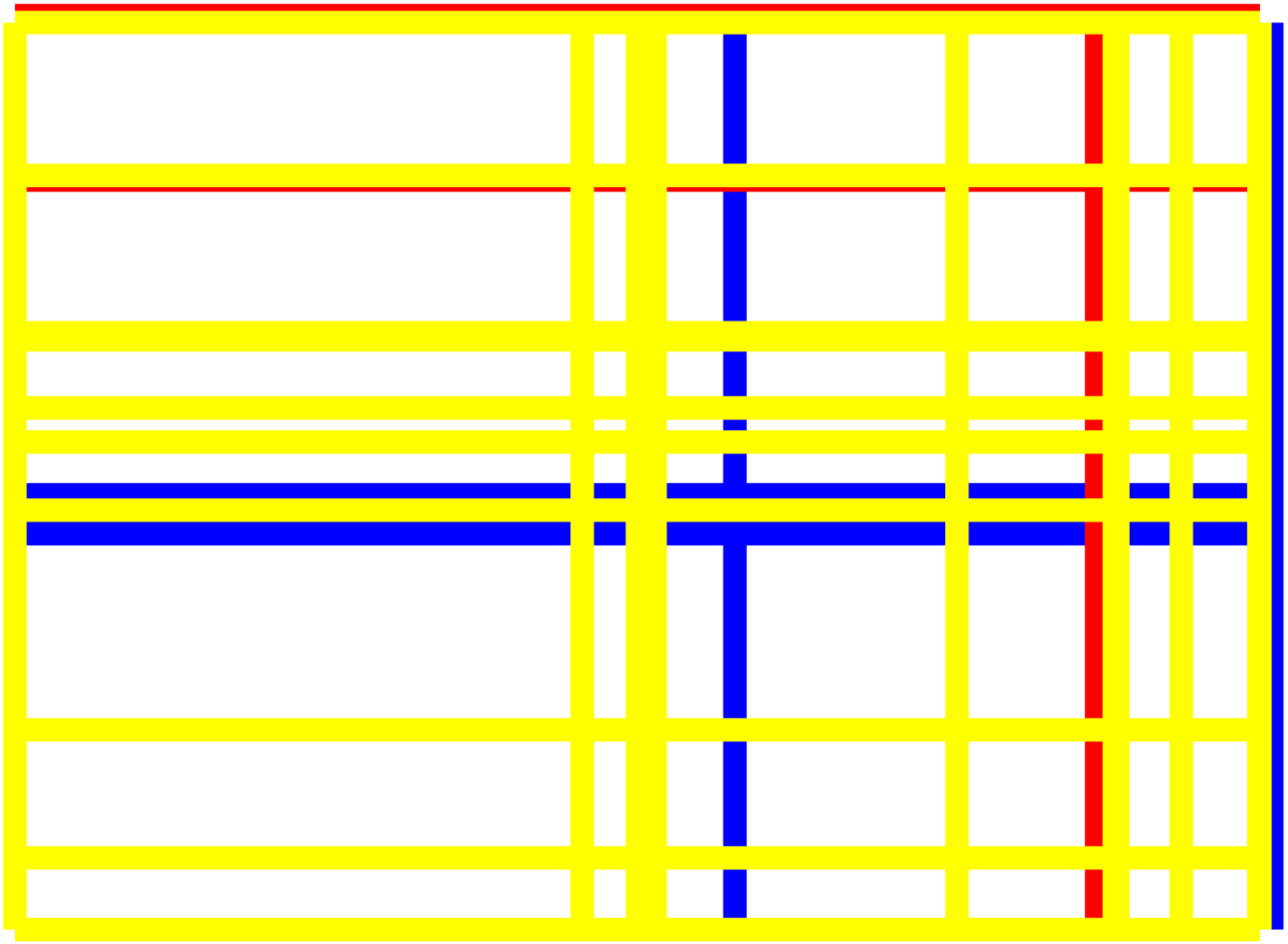
Seems like I'm getting there. \ Define the final function needed to make the plots.

In [13]:
```
def imitate_new_york(palette = ['blue','red','yellow'], linewidth=20):
    # roll the dice
    mondrian_offsets(set1=2,set2=2,set3=10)
    # assign values to variables
    voffsets_list, hoffsets_list, bounds = mondrian_offsets(set1=2,set2=2,set3=10)
    left,right,bottom,up = bounds
    # make the plot
    fig, ax = plt.subplots(figsize=(20,15))
    ax.axis('off')
    for i, c in enumerate(palette):
```

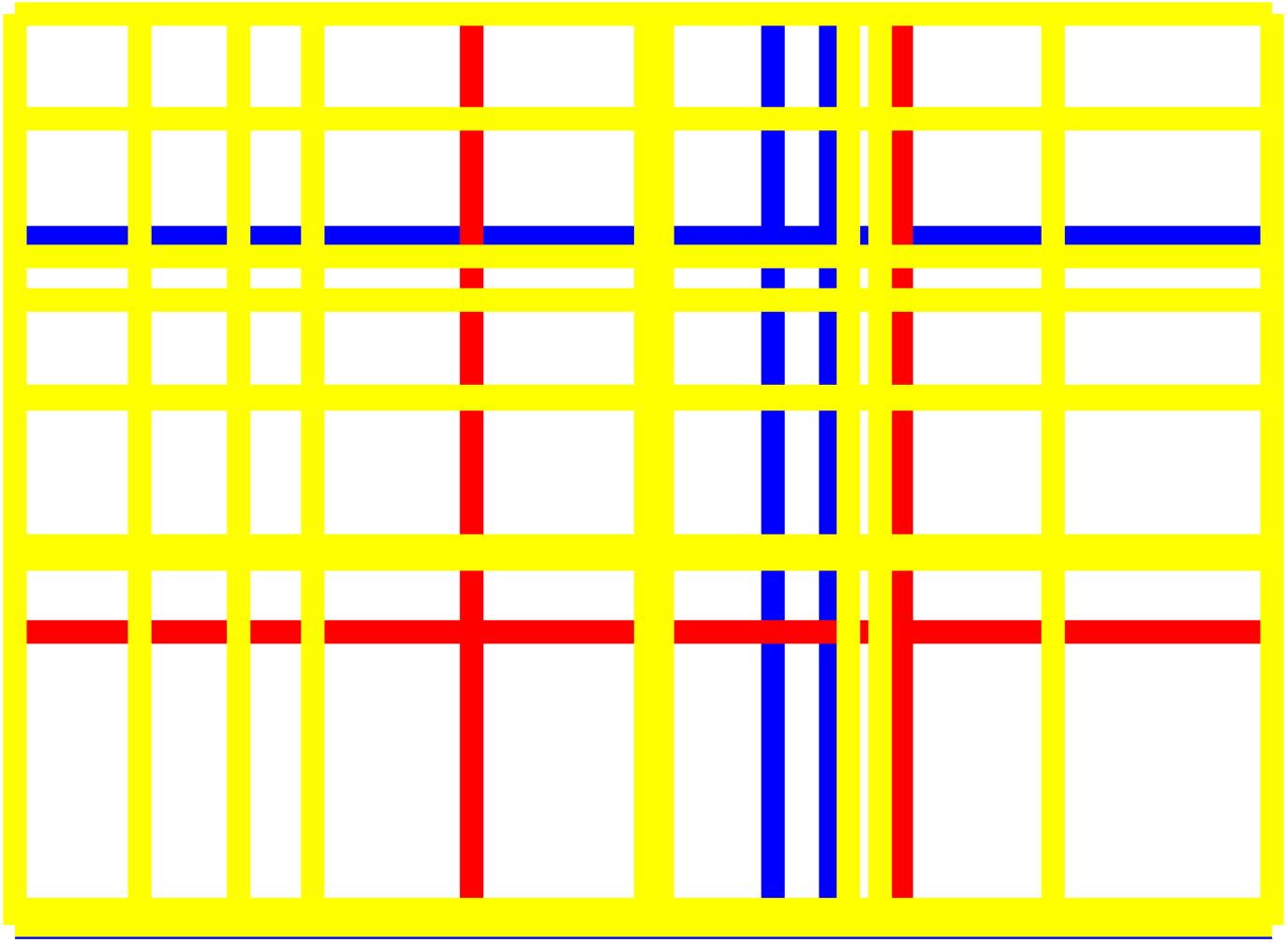
```
        # bottom,up: Respective beginning and end of each verticle line.
        plt.vlines(voffsets_list[i],bottom,up, linewidth=linewidth, color=c)
        # left,right: Respective beginning and end of each vhorizontal line.
        plt.hlines(hoffsets_list[i],left,right, linewidth=linewidth, color=c)
    ax.set_xlim([left-4,right+4])
    ax.set_ylim([bottom-4,up+4])
    plt.show()
```

Drumroll please -- test the imitate_new_york function to hopefully get some good image!

In [14]:
```
imitate_new_york()
```



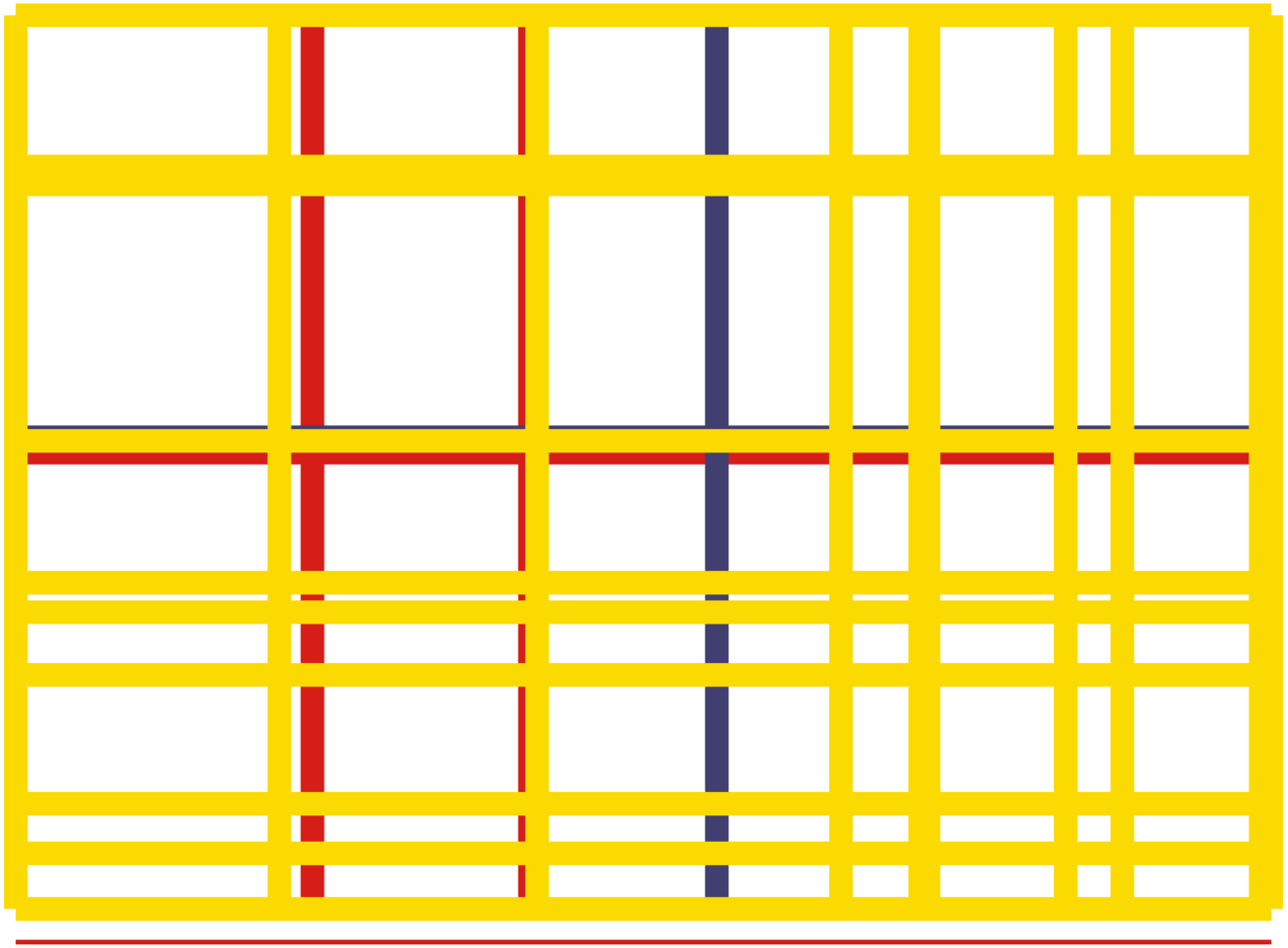
In [15]:
```
imitate_new_york()
```



The above images look quite satisfying.\ I want to use colors closer to the original artwork to make it look more subtle.\ I pickd RGB colors from the original New York image using this website: https://imagecolorpicker.com/en

- #d61d18 rgba(214,29,24,255) RED
- #fada00 rgba(250,218,0,255) YELLOW
- #413f6f rgba(65,63,111,255) BLUE
- #f0f5f1 rgba(240,245,241,255) WHITE

In [16]:
```
imitate_new_york(['#d61d18','#413f6f','#fada00'], 20)
```



It looks pretty good, but I still want to further refine them:

- Make the bounds of the image bigger.
- Add a background color.

That means some changes to my functions:

- Adjust the "bounds" and "offsets" variables.
- Define facecolor in matplotlib syntax.

In [17]:

```
def mondrian_offsets(set1=2,set2=2,set3=8):
    # Make empty list to store the 3 sets of randomly sampled offsets.
    voffsets_list = []
    hoffsets_list = []
    # Roll the dice for voffsets in the uppermost layer.
    voffsets3 = random_offsets(set3)
    # Define some boundaries by voffsets.
    left = min(voffsets3)+ 30
    right = max(voffsets3) + 30
    # Roll the dice for hoffsets.
    hoffsets3 = uniform_offsets(set3,left-34,right-35)
    # Determine the upper and lower boundaries by hoffsets.
    bottom = min(hoffsets3) + 30
    up = max(hoffsets3) + 30
    # Sample from the bottom two layers and record the offsets in the lists.
    for num in [set1,set2]:
        voffsets = random_offsets(num)
        hoffsets = uniform_offsets(num,left,right)
        voffsets_list.append(voffsets)
        hoffsets_list.append(hoffsets)
```

```
    # Append the offsets of the uppermost layer in the end.
    voffsets_list.append(voffsets3)
    hoffsets_list.append(hoffsets3)
    # Record the boundaries of the lines, later used in vline/hline arguments.
    # Define the bounds as a tuple for easy unpacking.
    bounds = (left,right,bottom,up)
    return voffsets_list,hoffsets_list,bounds
```

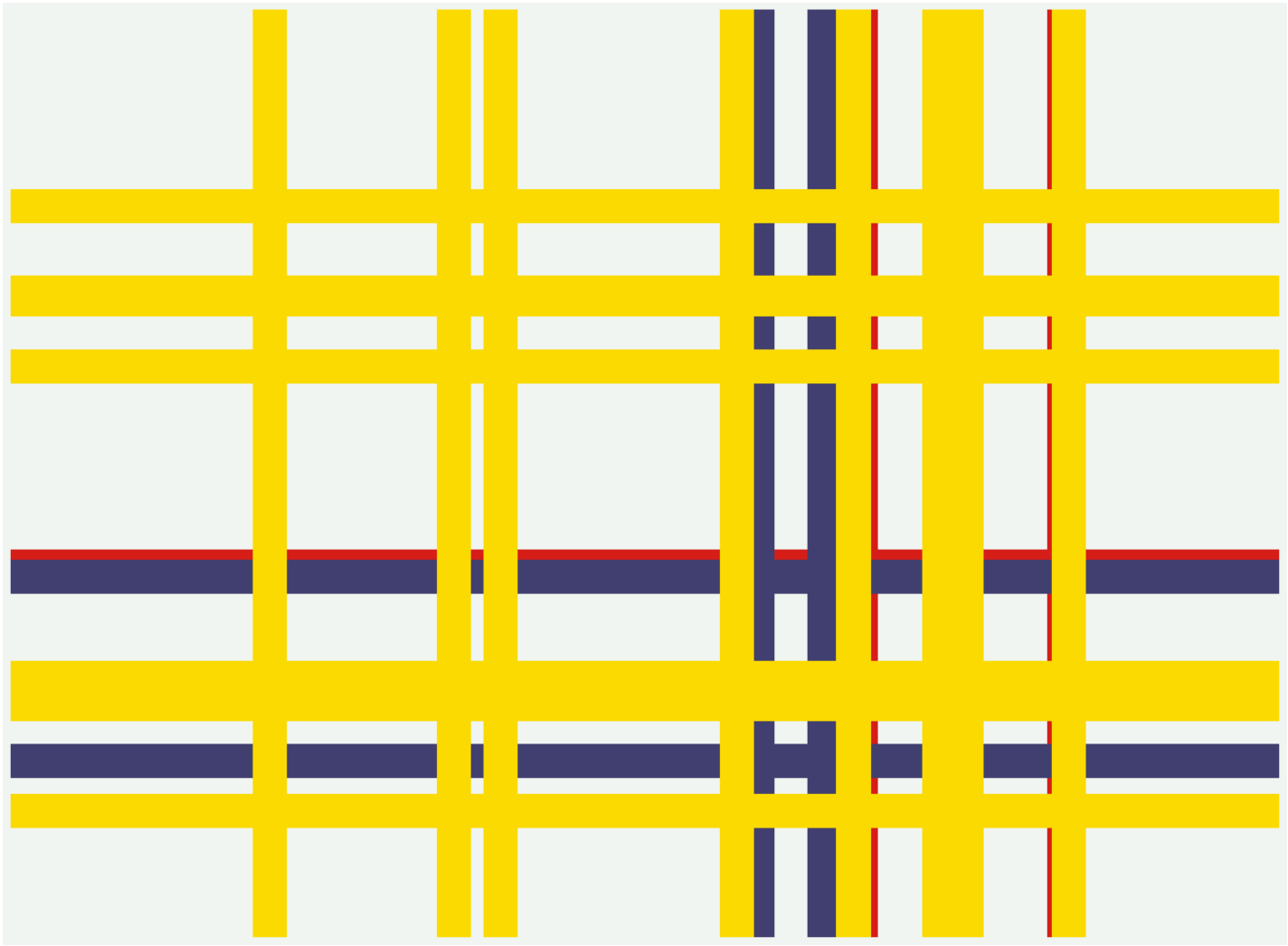
Added facecolor and updated set_xlim and set_ylim in imitate_new_york function.

In [18]:

```
def imitate_new_york(palette = ['#d61d18','#413f6f','#fada00'], linewidth = 30):
    # roll the dice
    mondrian_offsets(set1=2,set2=2,set3=10)
    # assign values to variables
    voffsets_list, hoffsets_list, bounds = mondrian_offsets(set1=2,set2=2,set3=10)
    left,right,bottom,up = bounds
    # make the plot
    fig, ax = plt.subplots(figsize=(20,15), facecolor = '#f0f5f1')
    ax.axis('off')
    for i, c in enumerate(palette):
        # bottom,up: Respective beginning and end of each verticle line.
        plt.vlines(voffsets_list[i],bottom,up, linewidth=linewidth, color=c)
        # left,right: Respective beginning and end of each vhorizontal line.
        plt.hlines(hoffsets_list[i],left,right, linewidth=linewidth, color=c)
    ax.set_xlim([left,right])
    ax.set_ylim([bottom,up])
    plt.show()
```

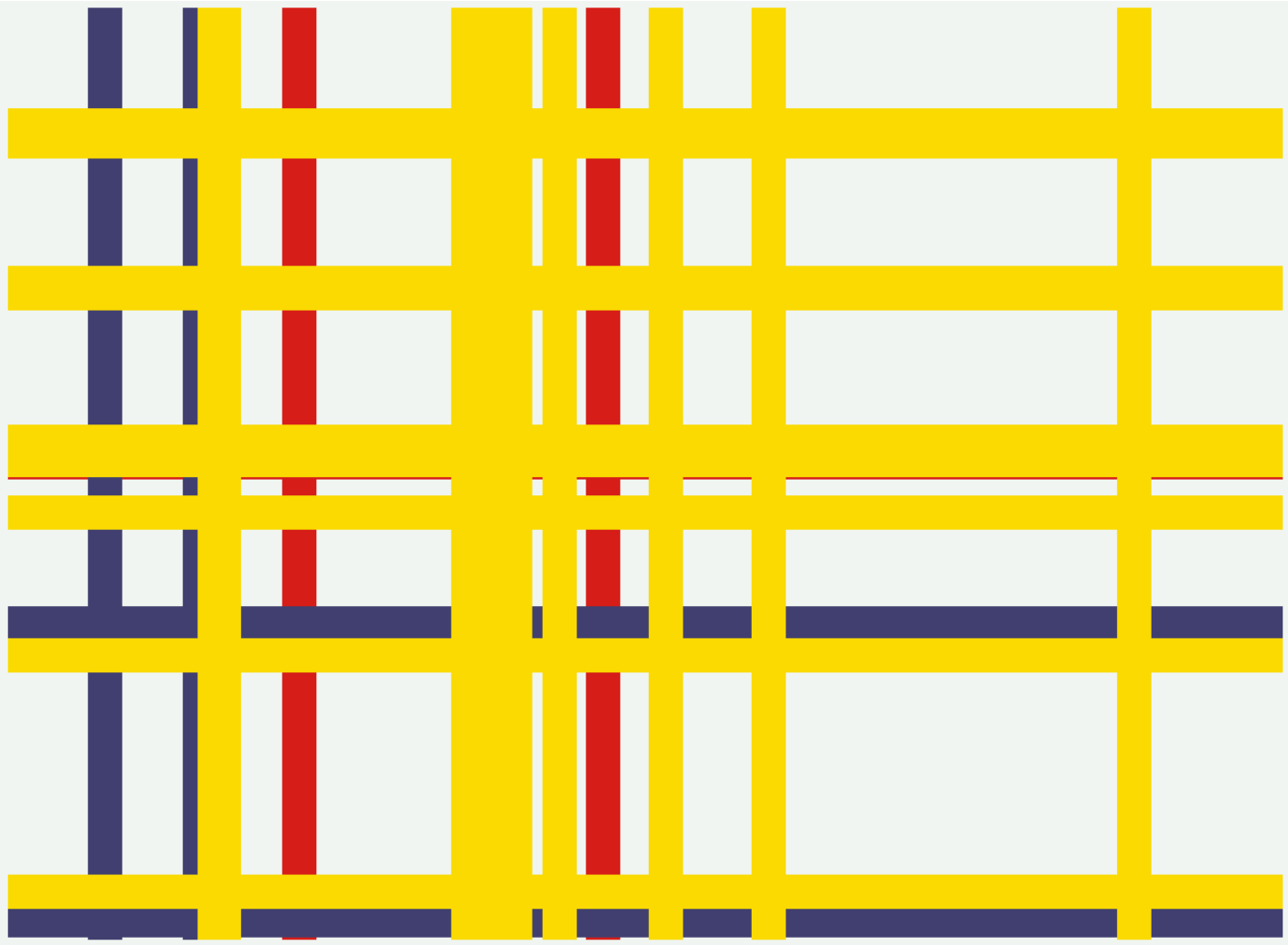
In [19]:

```
imitate_new_york()
```



In [20]:

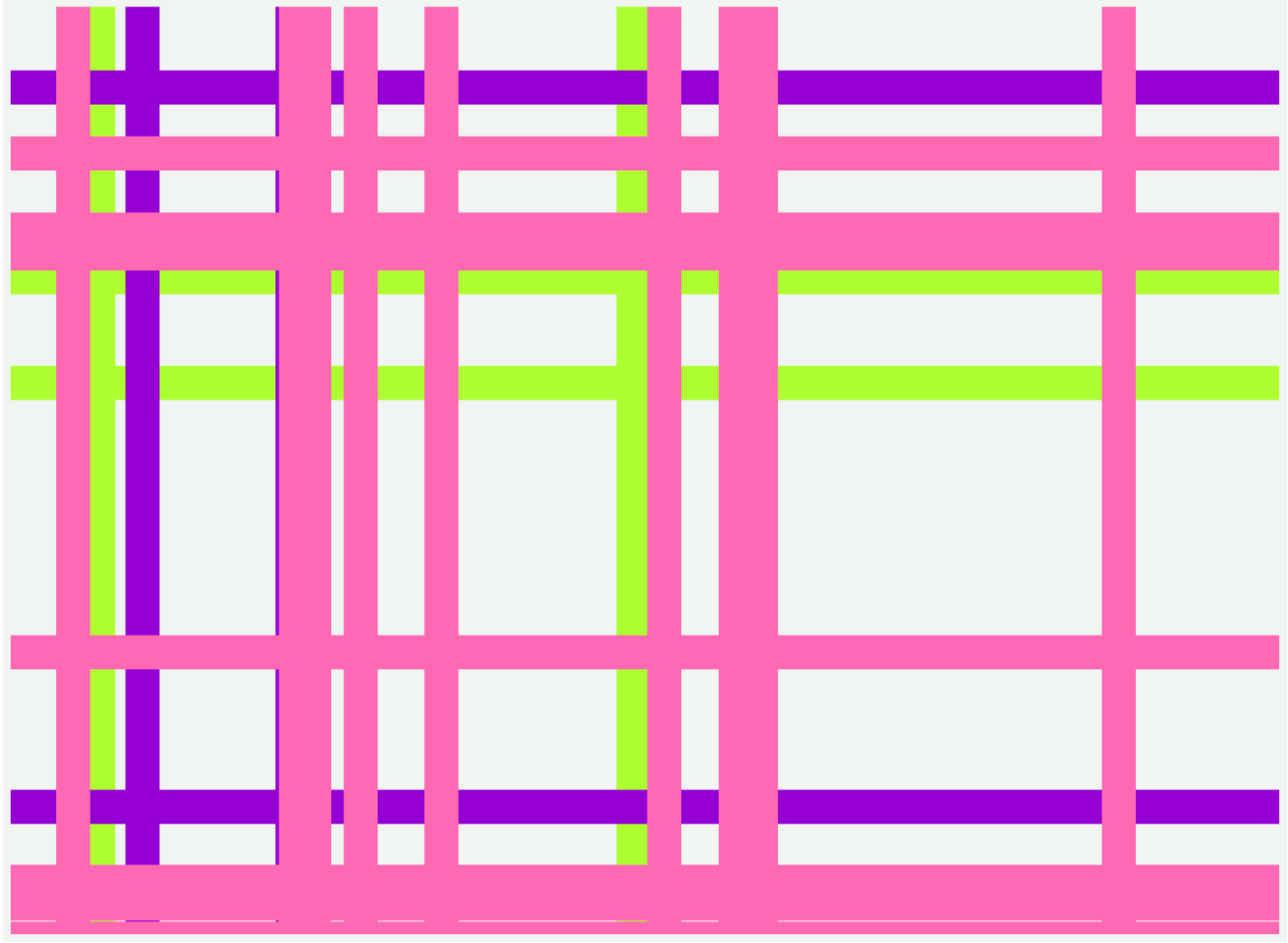
```
imitate_new_york()
```



Yay! They look quite satisfying to me.\ I got a new idea -- what if I change the colors and use 3 colors to represent each of the 4 seasons?

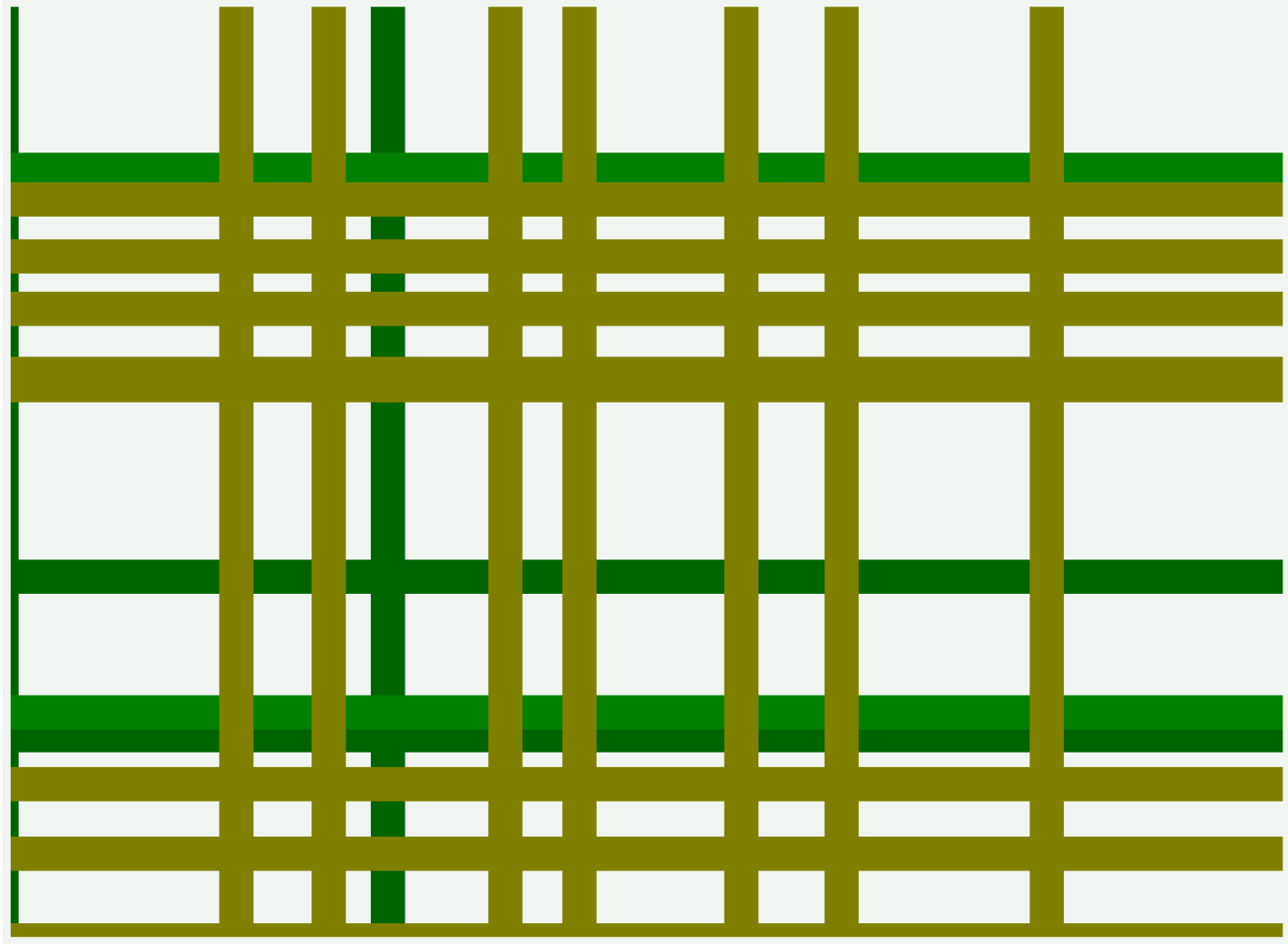
In [21]:

```
# Spring
imitate_new_york(['greenyellow','darkviolet','hotpink'], 30)
```



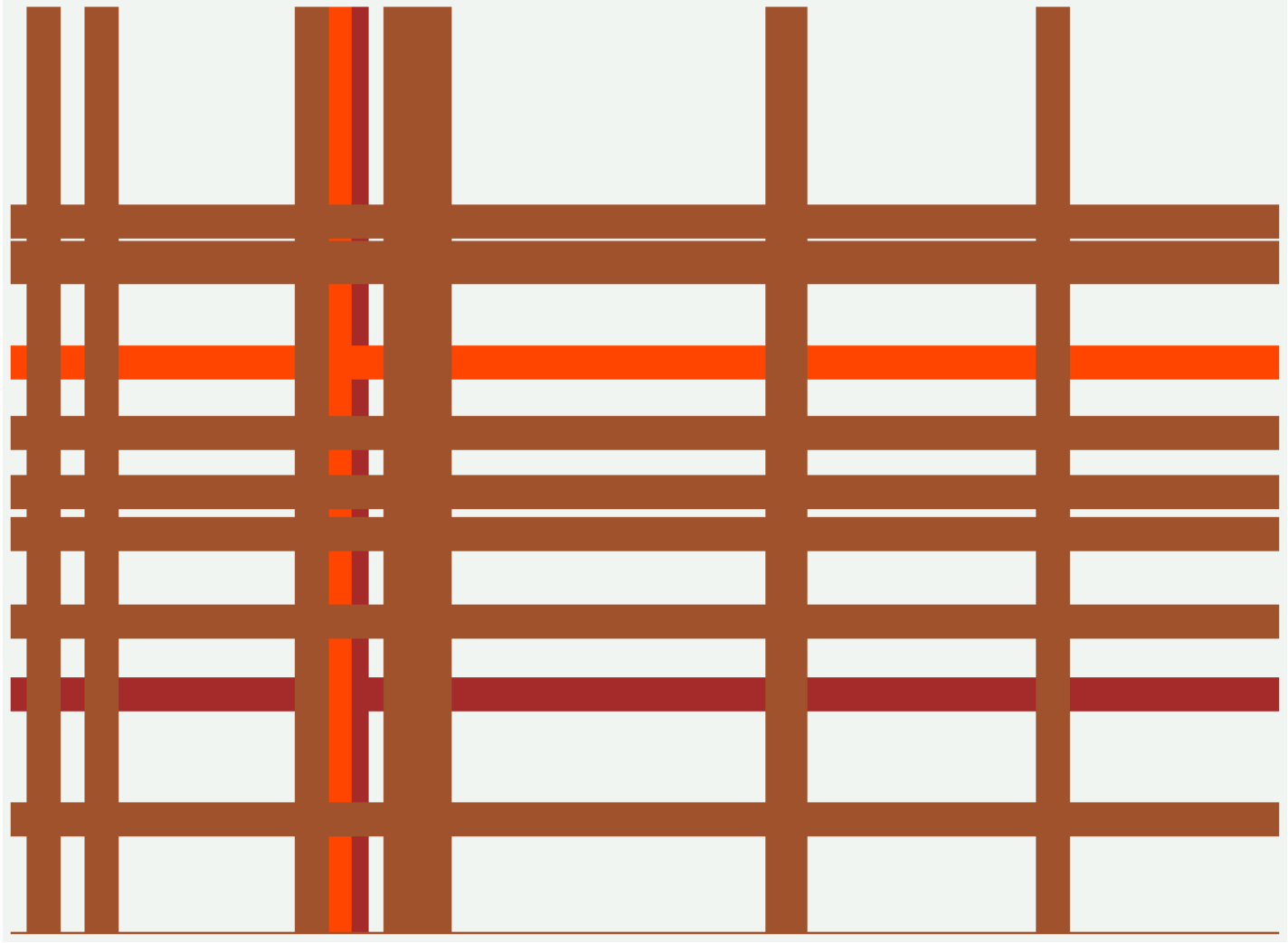
In [22]:

```
# Summer
imitate_new_york(['darkgreen','green','olive'], 30)
```



In [23]:

```
# Fall
imitate_new_york(['brown','orangered','sienna'], 30)
```



In [24]:

```
# Winter
imitate_new_york(['grey','navy','azure'], 30)
```