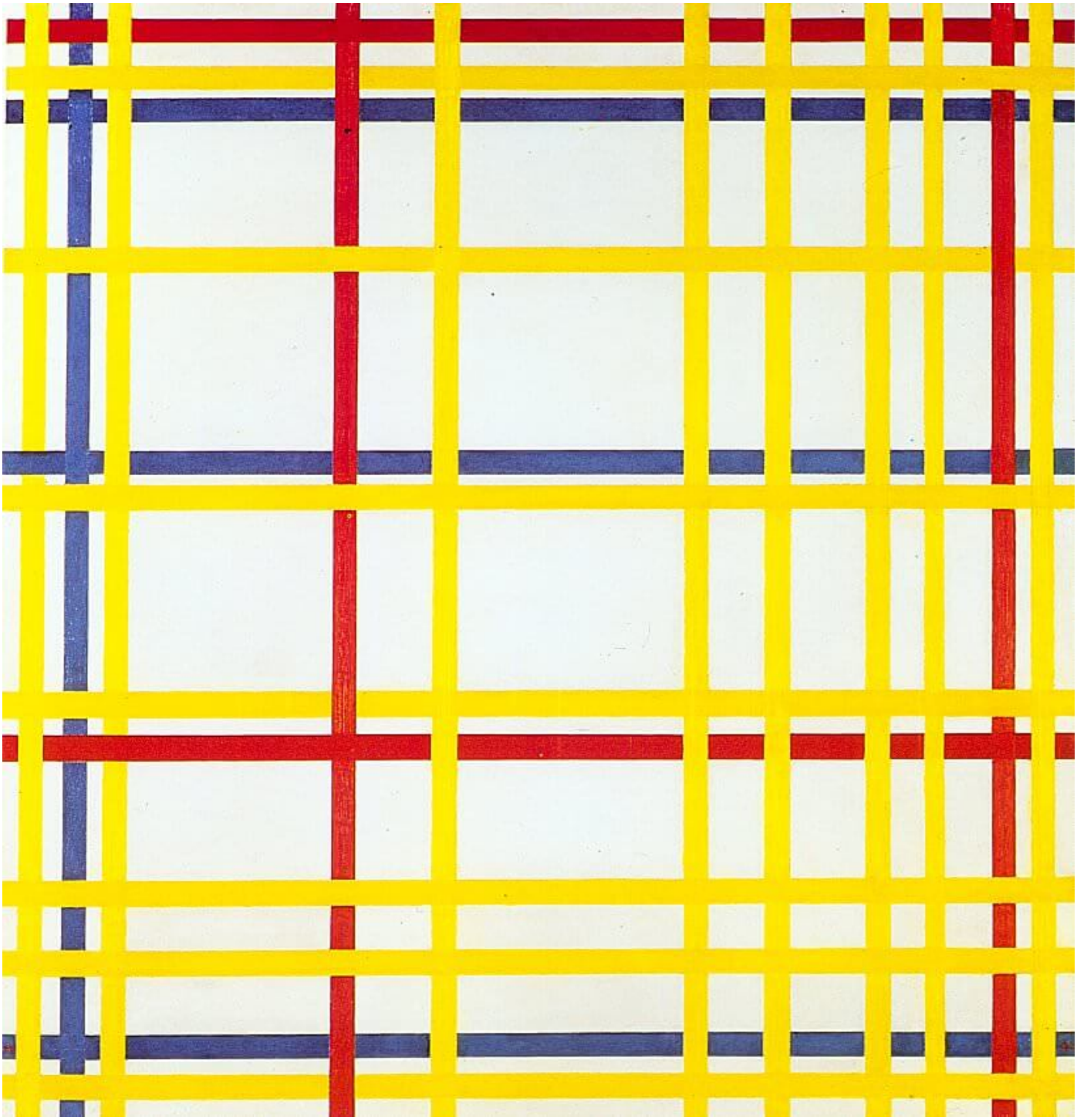
Mondrian is one of the most influential abstract artist. Many of his work is made of artistic combinations of lines and squares, such as in his famous "New York" piece below. In this Jupyter Notebook, I used Python libraries and functions to make plots that mimick his artwork.



Import python libraries.

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

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

Define two functions. 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
```

Testing the functions.

In [5]:
```
voffsets = random_offsets(10)
print(voffsets)
# vmin and vmax are 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-4,vmax-5)
print(hoffsets)
# hmin and hmax are used to set the boundaries of the vertical lines
hmin = min(hoffsets)
hmax = max(hoffsets)
print(hmin,hmax)
```

```
[ 49.44320986  -6.98692255 136.90568937  16.0989553   44.04460779
  20.4844581   86.82650025  77.06584002  81.07206681  14.45044503]
-6.986922551820776 136.90568936701032
```

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 0x116f1c550>
```

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 0x116f75d30>
```

Now the first layer 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 number 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 that 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 of the boundaries. So here is an updated version of the 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 = []
    # Sample from the uppermost layer first to define the boundaries of the lines.
    voffsets3 = random_offsets(set3)
    left = min(voffsets3)
    right = max(voffsets3)
    hoffsets3 = uniform_offsets(set3,left-4,right-5)
    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 as vline/hline arguments.
    # Define the bounds as a tuple for easy unpacking.
    bounds = (left,right,bottom,up)
    return voffsets_list,hoffsets_list,bounds
```

Except for the offset list, this function now also returns the limits of the lines.

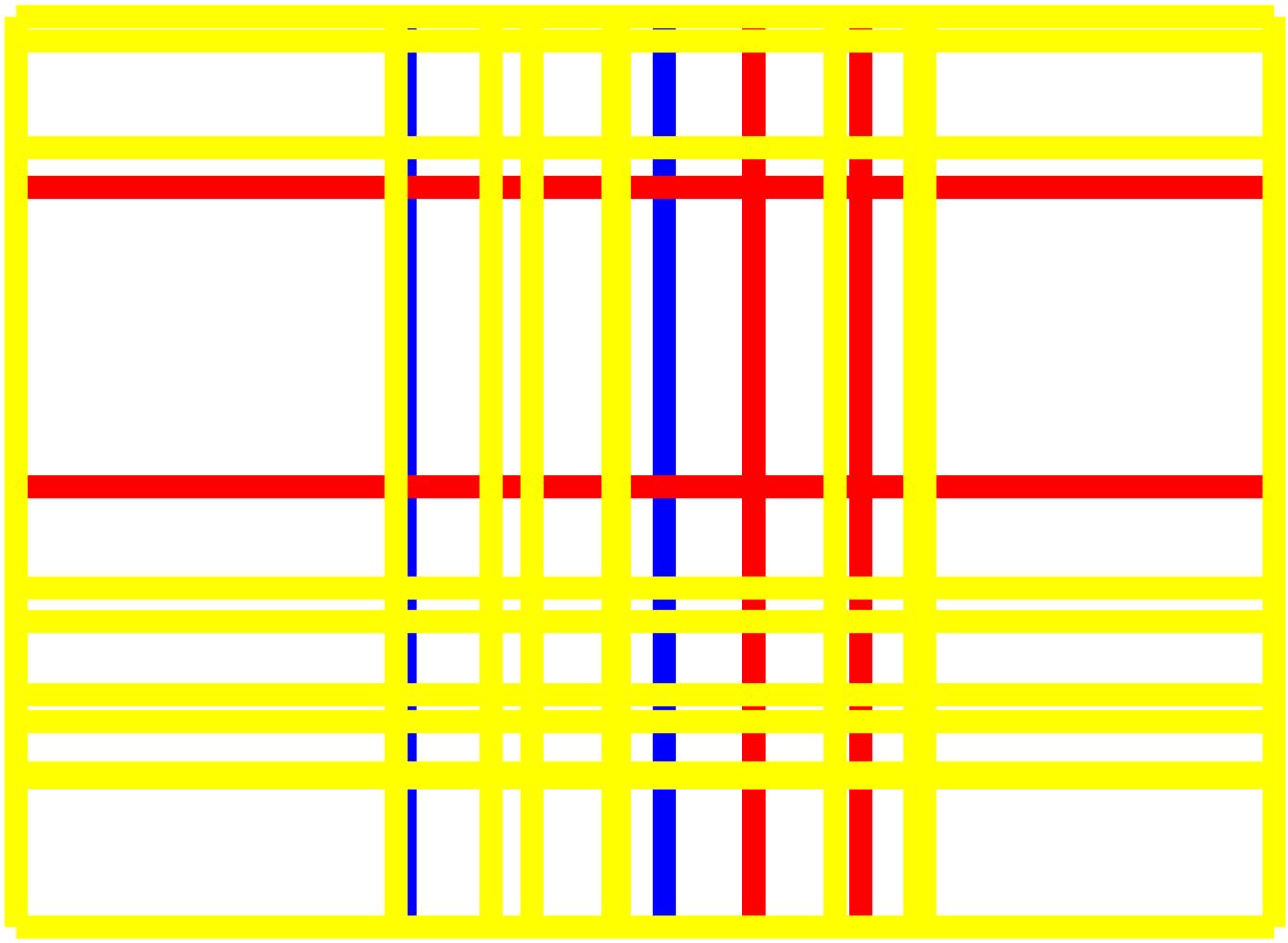
In [12]:
```
mondrian_offsets(set1=2,set2=2,set3=10)
```

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])],
 (-58.64638800278473,
  104.80388107530703,
  -46.266970395837845,
  90.63195220665969))
```

In [27]:
```
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()
```

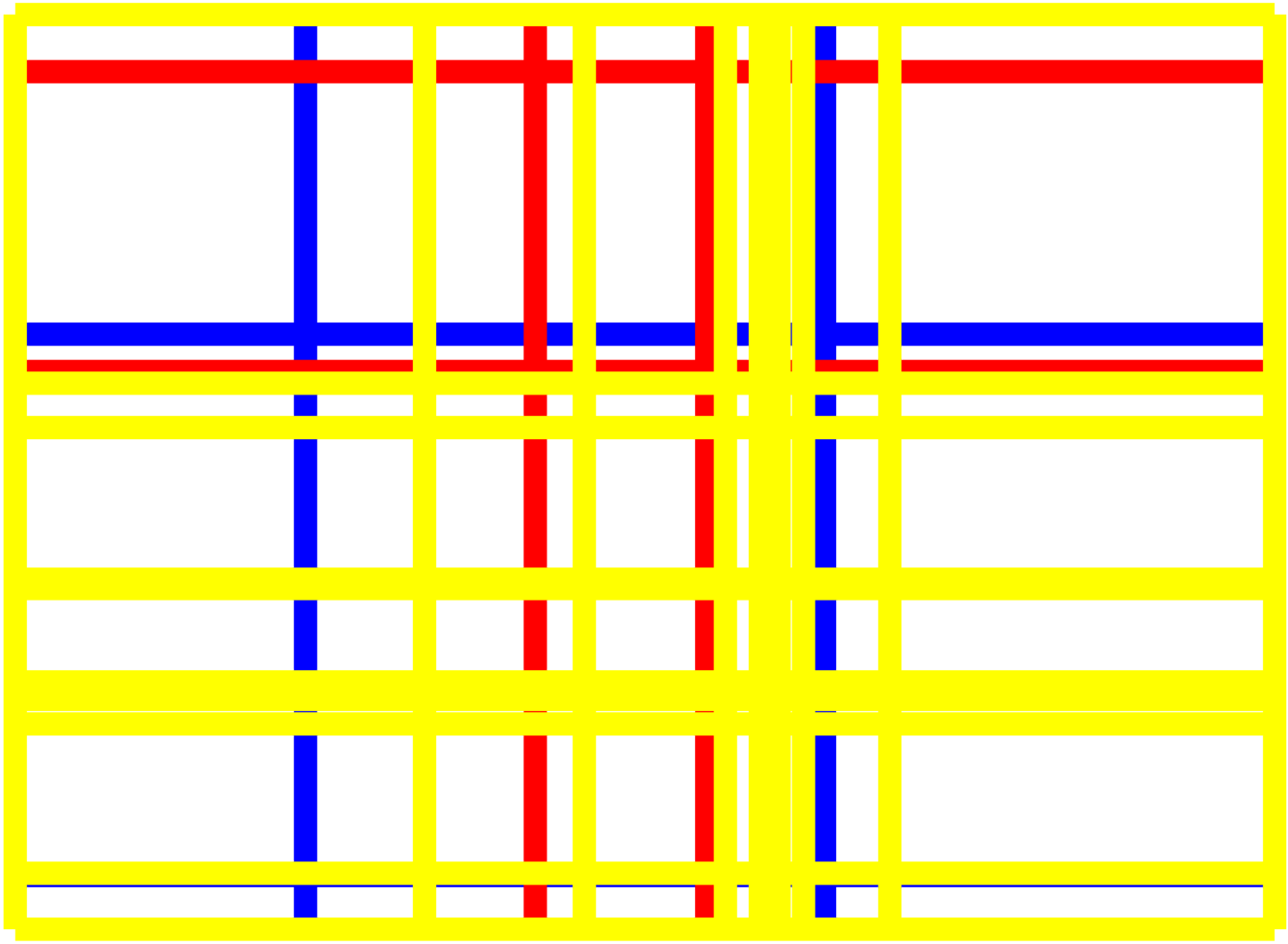
In [22]:

```
imitate_new_york()
```



In [31]:

```
imitate_new_york()
```



Pick 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 [33]:
```
imitate_new_york(['#fada00 ','#d61d18','#413f6f'], 20)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
/var/folders/pc/7mhp__sx37j2068rj4qnsq5w0000gn/T/ipykernel_2889/52244748.py in <module>
----> 1 imitate_new_york(['#fada00 ','#d61d18','#413f6f'], 20)

/var/folders/pc/7mhp__sx37j2068rj4qnsq5w0000gn/T/ipykernel_2889/1178277162.py in imitate_n
ew_york(palette, linewidth)
     10     for i, c in enumerate(palette):
     11         # bottom,up: Respective beginning and end of each verticle line.
---> 12         plt.vlines(voffsets_list[i],bottom,up, linewidth=linewidth, color=c)
     13         # left,right: Respective beginning and end of each vhorizontal line.
```

```
     14         plt.hlines(hoffsets_list[i],left,right, linewidth=linewidth, color=c)

~/opt/anaconda3/lib/python3.9/site-packages/matplotlib/pyplot.py in vlines(x, ymin, ymax,
 colors, linestyles, label, data, **kwargs)
   3259         x, ymin, ymax, colors=None, linestyles='solid', label='', *,
   3260         data=None, **kwargs):
-> 3261     return gca().vlines(
   3262         x, ymin, ymax, colors=colors, linestyles=linestyles,
   3263         label=label, **({"data": data} if data is not None else {}),

~/opt/anaconda3/lib/python3.9/site-packages/matplotlib/__init__.py in inner(ax, data, *arg
s, **kwargs)
   1359     def inner(ax, *args, data=None, **kwargs):
   1360         if data is None:
-> 1361             return func(ax, *map(sanitize_sequence, args), **kwargs)
   1362
   1363         bound = new_sig.bind(ax, *args, **kwargs)

~/opt/anaconda3/lib/python3.9/site-packages/matplotlib/axes/_axes.py in vlines(self, x, ym
in, ymax, colors, linestyles, label, **kwargs)
   1118                                       linestyles=linestyles, label=label)
   1119         self.add_collection(lines, autolim=False)
-> 1120         lines.update(kwargs)
   1121
   1122         if len(x) > 0:

~/opt/anaconda3/lib/python3.9/site-packages/matplotlib/artist.py in update(self, props)
   1062                         raise AttributeError(f"{type(self).__name__!r} object "
   1063                                              f"has no property {k!r}")
-> 1064                     ret.append(func(v))
   1065         if ret:
   1066             self.pchanged()

~/opt/anaconda3/lib/python3.9/site-packages/matplotlib/collections.py in set_color(self,
 c)
   1544             cycle through the sequence.
   1545         """
-> 1546         self.set_edgecolor(c)
   1547
   1548     set_colors = set_color

~/opt/anaconda3/lib/python3.9/site-packages/matplotlib/collections.py in set_edgecolor(sel
f, c)
    837             c = c.lower()
    838         self._original_edgecolor = c
--> 839         self._set_edgecolor(c)
    840
    841     def set_alpha(self, alpha):

~/opt/anaconda3/lib/python3.9/site-packages/matplotlib/collections.py in _set_edgecolor(se
lf, c)
    816             self.stale = True
    817             return
--> 818         self._edgecolors = mcolors.to_rgba_array(c, self._alpha)
    819         if set_hatch_color and len(self._edgecolors):
    820             self._hatch_color = tuple(self._edgecolors[0])

~/opt/anaconda3/lib/python3.9/site-packages/matplotlib/colors.py in to_rgba_array(c, alph
a)
    365
    366     if isinstance(c, str):
--> 367         raise ValueError("Using a string of single character colors as "
    368                          "a color sequence is not supported. The colors can "
    369                          "be passed as an explicit list instead.")
```

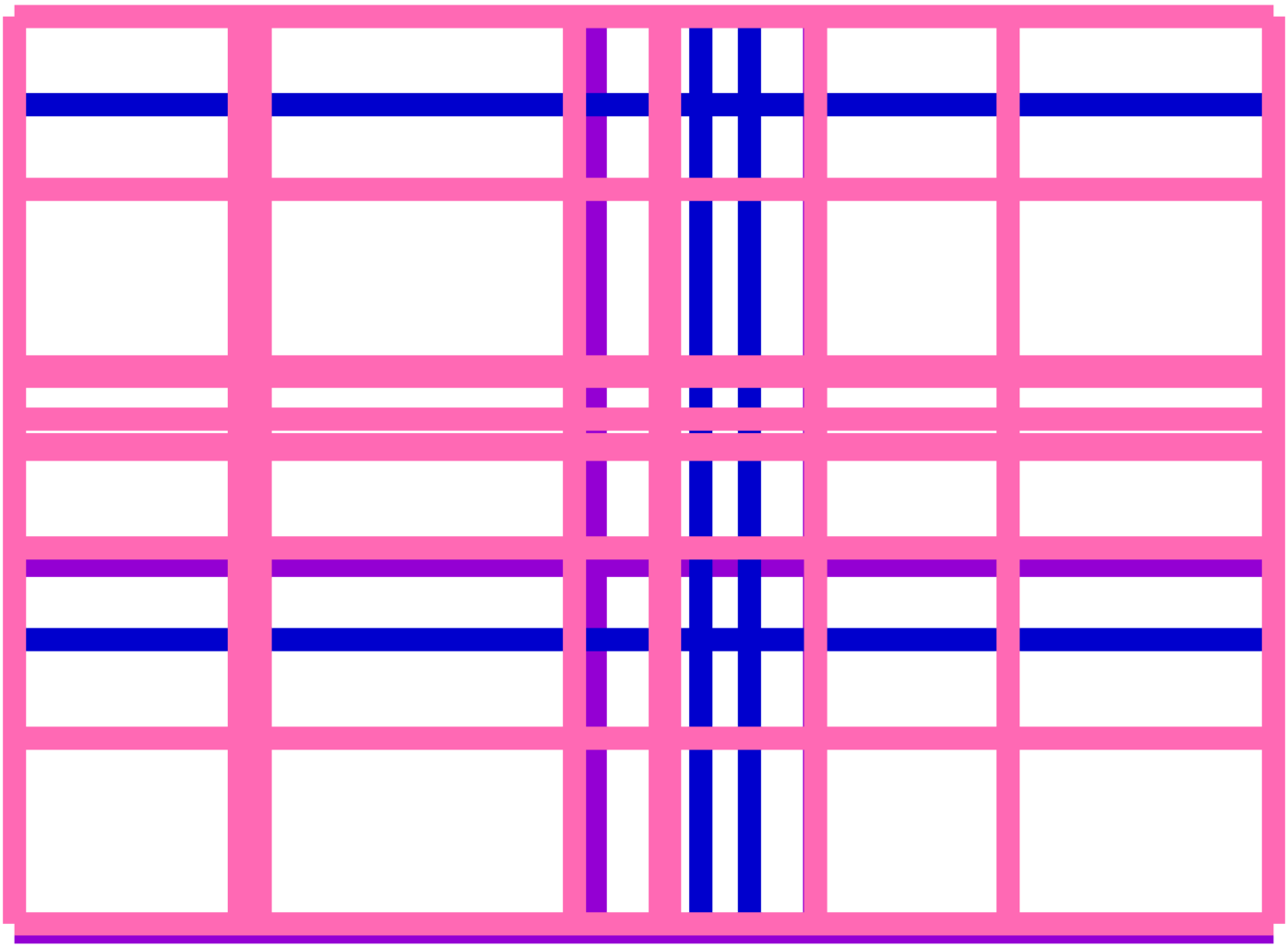
```
ValueError: Using a string of single character colors as a color sequence is not supporte
d. The colors can be passed as an explicit list instead.
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~/opt/anaconda3/lib/python3.9/site-packages/IPython/core/formatters.py in __call__(self, o
bj)
    339                 pass
    340             else:
--> 341                 return printer(obj)
    342             # Finally look for special method names
    343             method = get_real_method(obj, self.print_method)

~/opt/anaconda3/lib/python3.9/site-packages/IPython/core/pylabtools.py in print_figure(fi
g, fmt, bbox_inches, base64, **kwargs)
    149         FigureCanvasBase(fig)
    150
--> 151     fig.canvas.print_figure(bytes_io, **kw)
    152     data = bytes_io.getvalue()
    153     if fmt == 'svg':

~/opt/anaconda3/lib/python3.9/site-packages/matplotlib/backend_bases.py in print_figure(se
lf, filename, dpi, facecolor, edgecolor, orientation, format, bbox_inches, pad_inches, bbo
x_extra_artists, backend, **kwargs)
   2228                         else suppress())
   2229                 with ctx:
-> 2230                     self.figure.draw(renderer)
   2231
   2232             if bbox_inches:

~/opt/anaconda3/lib/python3.9/site-packages/matplotlib/artist.py in draw_wrapper(artist, r
enderer, *args, **kwargs)
     72     @wraps(draw)
     73     def draw_wrapper(artist, renderer, *args, **kwargs):
---> 74         result = draw(artist, renderer, *args, **kwargs)
     75         if renderer._rasterizing:
     76             renderer.stop_rasterizing()

~/opt/anaconda3/lib/python3.9/site-packages/matplotlib/artist.py in draw_wrapper(artist, r
enderer, *args, **kwargs)
     49                 renderer.start_filter()
     50
---> 51             return draw(artist, renderer, *args, **kwargs)
     52         finally:
     53             if artist.get_agg_filter() is not None:

~/opt/anaconda3/lib/python3.9/site-packages/matplotlib/figure.py in draw(self, renderer)
   2788
   2789             self.patch.draw(renderer)
-> 2790             mimage._draw_list_compositing_images(
   2791                 renderer, self, artists, self.suppressComposite)
   2792

~/opt/anaconda3/lib/python3.9/site-packages/matplotlib/image.py in _draw_list_compositing_
images(renderer, parent, artists, suppress_composite)
    130     if not_composite or not has_images:
    131         for a in artists:
--> 132             a.draw(renderer)
    133     else:
    134         # Composite any adjacent images together

~/opt/anaconda3/lib/python3.9/site-packages/matplotlib/artist.py in draw_wrapper(artist, r
enderer, *args, **kwargs)
     49                 renderer.start_filter()
     50
---> 51             return draw(artist, renderer, *args, **kwargs)
```

```
     52         finally:
     53             if artist.get_agg_filter() is not None:

~/opt/anaconda3/lib/python3.9/site-packages/matplotlib/_api/deprecation.py in wrapper(*inn
er_args, **inner_kwargs)
    429                             else deprecation_addendum,
    430                 **kwargs)
--> 431         return func(*inner_args, **inner_kwargs)
    432
    433     return wrapper

~/opt/anaconda3/lib/python3.9/site-packages/matplotlib/axes/_base.py in draw(self, rendere
r, inframe)
   2919             renderer.stop_rasterizing()
   2920
-> 2921         mimage._draw_list_compositing_images(renderer, self, artists)
   2922
   2923         renderer.close_group('axes')

~/opt/anaconda3/lib/python3.9/site-packages/matplotlib/image.py in _draw_list_compositing_
images(renderer, parent, artists, suppress_composite)
    130     if not_composite or not has_images:
    131         for a in artists:
--> 132             a.draw(renderer)
    133     else:
    134         # Composite any adjacent images together

~/opt/anaconda3/lib/python3.9/site-packages/matplotlib/artist.py in draw_wrapper(artist, r
enderer, *args, **kwargs)
     49                 renderer.start_filter()
     50
---> 51             return draw(artist, renderer, *args, **kwargs)
     52         finally:
     53             if artist.get_agg_filter() is not None:

~/opt/anaconda3/lib/python3.9/site-packages/matplotlib/collections.py in draw(self, render
er)
    348         renderer.open_group(self.__class__.__name__, self.get_gid())
    349
--> 350         self.update_scalarmappable()
    351
    352         transform, transOffset, offsets, paths = self._prepare_points()

~/opt/anaconda3/lib/python3.9/site-packages/matplotlib/collections.py in update_scalarmapp
able(self)
    933             self._edgecolors = self._mapped_colors
    934         else:
--> 935             self._set_edgecolor(self._original_edgecolor)
    936         self.stale = True
    937

~/opt/anaconda3/lib/python3.9/site-packages/matplotlib/collections.py in _set_edgecolor(se
lf, c)
    816             self.stale = True
    817             return
--> 818         self._edgecolors = mcolors.to_rgba_array(c, self._alpha)
    819         if set_hatch_color and len(self._edgecolors):
    820             self._hatch_color = tuple(self._edgecolors[0])

~/opt/anaconda3/lib/python3.9/site-packages/matplotlib/colors.py in to_rgba_array(c, alph
a)
    365
    366     if isinstance(c, str):
--> 367         raise ValueError("Using a string of single character colors as "
    368                          "a color sequence is not supported. The colors can "
    369                          "be passed as an explicit list instead.")
```

```
ValueError: Using a string of single character colors as a color sequence is not supporte
d. The colors can be passed as an explicit list instead.
<Figure size 1440x1080 with 1 Axes>
```

In [30]:
```
imitate_new_york(['darkviolet','mediumblue','hotpink'], 20)
```



In [ ]: