seaborn-tut-choosing-color-palettes

December 25, 2015

1 Choosing color palettes

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

```
/Users/bartev/.virtualenvs/sbrn/lib/python2.7/site-packages/matplotlib/__init__.py:872: UserWarning: axes
  warnings.warn(self.msg_depr % (key, alt_key))
```

```
In [2]: sns.set(rc = {'figure.figsize': (6, 6)})
        np.random.seed(sum(map(ord, 'palettes')))
```

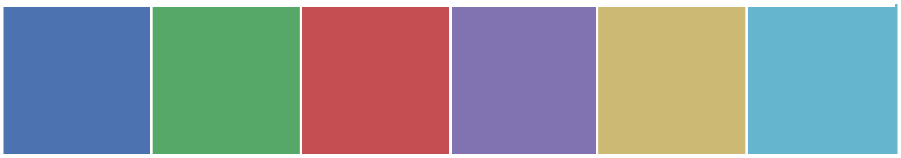
1.1 Building color palettes with color_palette()

2 Qualitative color palettes

2.1 Plot current color palette

```
In [3]: current_palette = sns.color_palette()
        sns.palplot(current_palette)
```

```
/Users/bartev/.virtualenvs/sbrn/lib/python2.7/site-packages/matplotlib/__init__.py:892: UserWarning: axes
  warnings.warn(self.msg_depr % (key, alt_key))
```

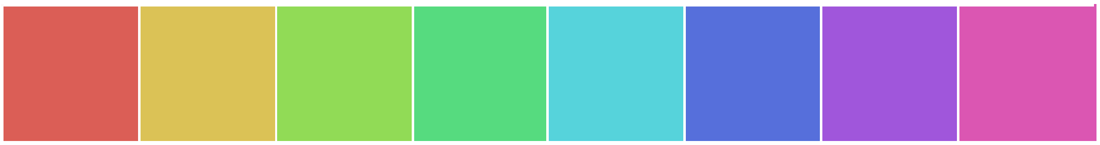


Six variations of default theme 1. deep 2. muted 3. pastel 4. bright 5. dark 6. colorblind

2.1.1 Using circular colors

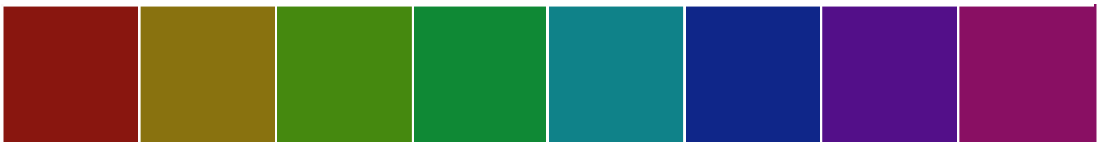
hls Draw evenly spaced colors in a circular color space

```
In [4]: sns.palplot(sns.color_palette('hls', 8))
```



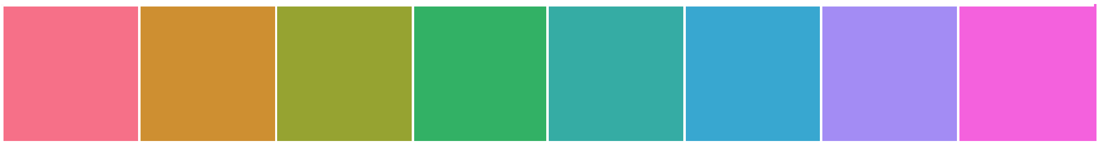
’hls_palette‘ allows control of lightness and saturation

```
In [5]: sns.palplot(sns.hls_palette(8, l = 0.3, s = 0.8))
```

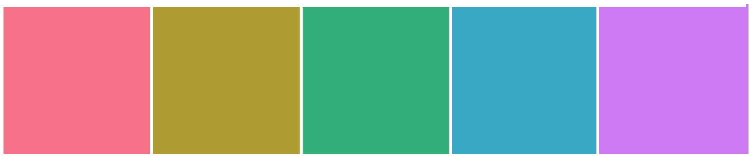


husl_palette select evenly spaced hues while keeping apparent brightness and saturation more uniform

```
In [6]: sns.palplot(sns.color_palette('husl', 8))
```

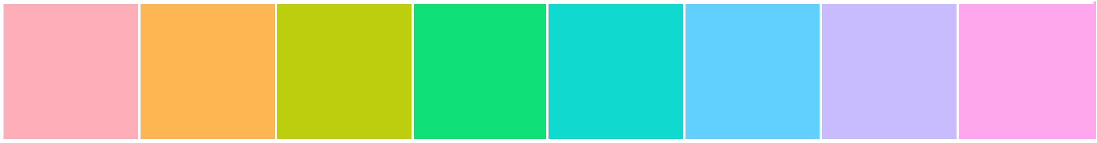


```
In [7]: sns.palplot(sns.color_palette('husl', 5))
```

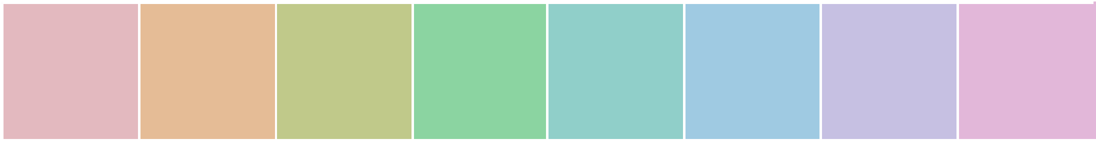


husl_palette

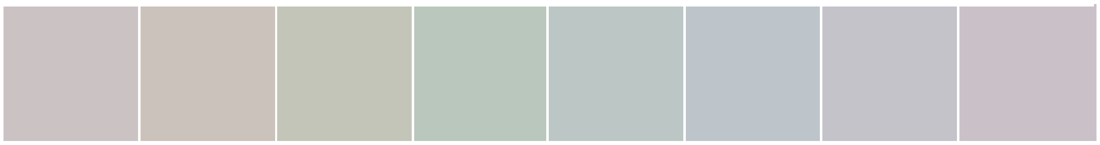
```
In [8]: sns.palplot(sns.husl_palette(8, l = 0.8, s = 1))
```



```
In [9]: sns.palplot(sns.husl_palette(8, l = 0.8, s = 0.5))
```

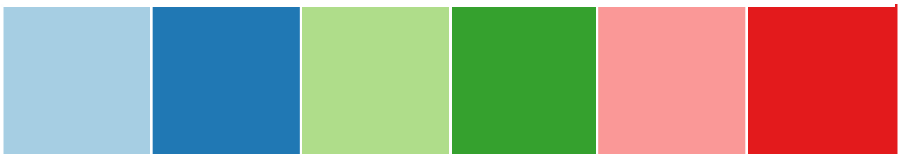


```
In [10]: sns.palplot(sns.husl_palette(8, l = 0.8, s = .1))
```



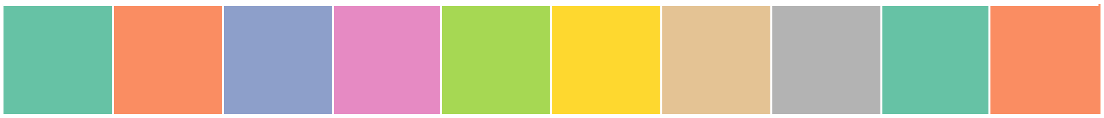
2.2 Using categorical Color Brewer palettes

```
In [11]: sns.palplot(sns.color_palette('Paired'))
```



Note: Set2 begins repeating colors after the 8th color

```
In [12]: sns.palplot(sns.color_palette('Set2', 10))
```



2.2.1 Help choosing a palette from the Color Brewer library

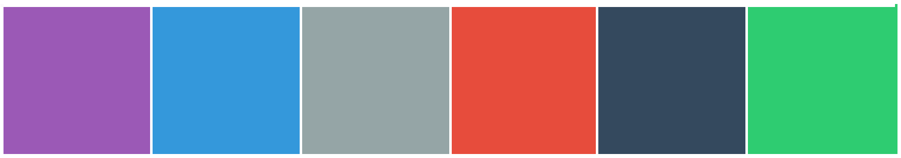
```
In [13]: # sns.choose_colorbrewer_palette(data_type = 'sequential', as_cmap = False)

In [14]: # sns.choose_colorbrewer_palette('diverging')

In [15]: # sns.choose_colorbrewer_palette('qualitative')
```

2.2.2 Choose your own colors

```
In [16]: flatui = ["#9b59b6", "#3498db", "#95a5a6", "#e74c3c",
                   "#34495e", "#2ecc71"]
         sns.palplot(sns.color_palette(flatui))
```



2.2.3 Use named colors from xkcd color survey

Get a sample of 10 items in the `xkcd_rgb` dict

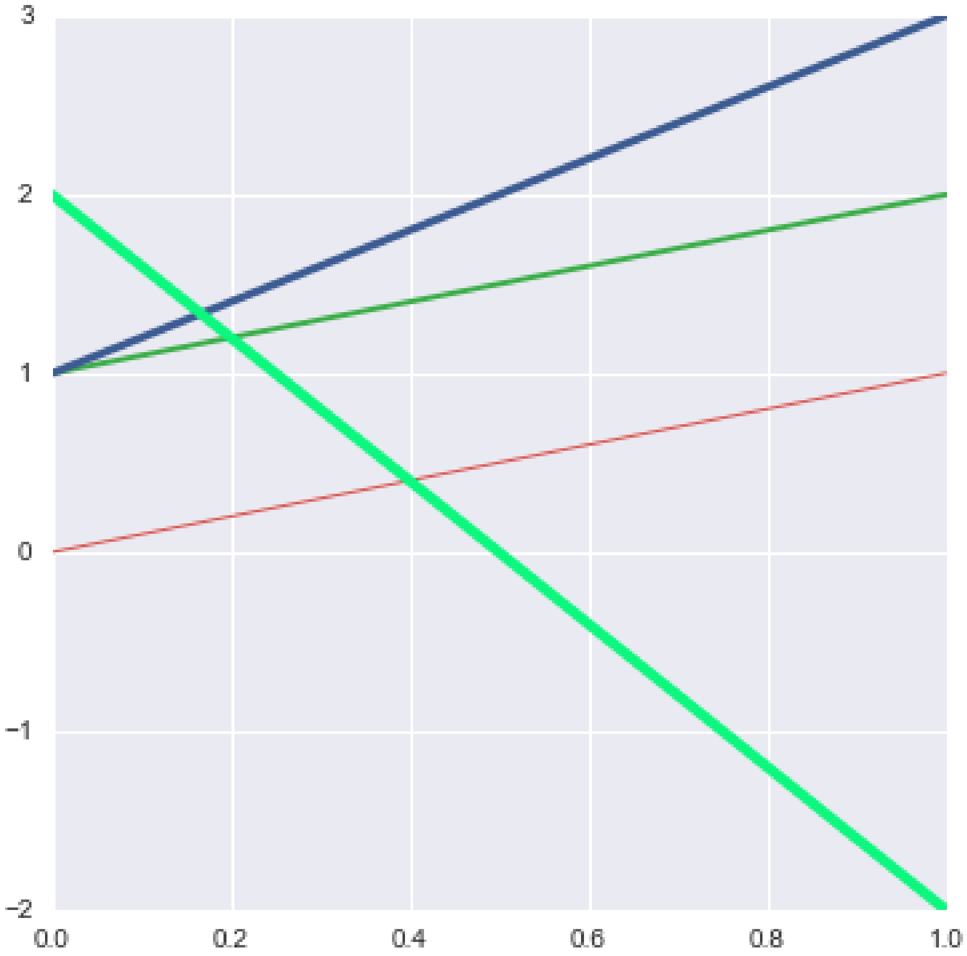
```
In [17]: sns.xkcd_rgb.items()[:10]
```

```
Out[17]: [(’fawn’, ’#cfaf7b’),
          (’light grey blue’, ’#9dbcd4’),
          (’dirty orange’, ’#c87606’),
          (’clay brown’, ’#b2713d’),
          (’yellow’, ’#ffff14’),
          (’minty green’, ’#0bf77d’),
          (’dull red’, ’#bb3f3f’),
          (’apple green’, ’#76cd26’),
          (’clear blue’, ’#247afd’),
          (’windows blue’, ’#3778bf’)]
```

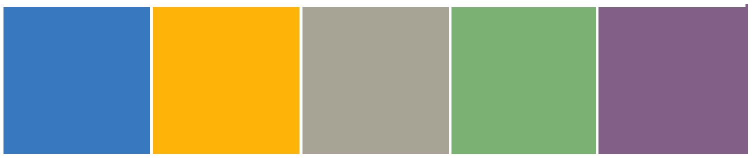
```
plt.plot([x1, x2, ...], [y1, y2, ...])
```

```
In [18]: plt.plot([0, 1], [0, 1], sns.xkcd_rgb[’pale red’], lw = 1)
         plt.plot([0, 1], [1, 2], sns.xkcd_rgb[’medium green’], lw = 2)
         plt.plot([0, 1], [1, 3], sns.xkcd_rgb[’denim blue’], lw = 3)
         plt.plot([0, 1], [2, -2], sns.xkcd_rgb[’minty green’], lw = 4)
```

```
Out[18]: [<matplotlib.lines.Line2D at 0x112393b10>]
```



```
In [19]: colors = ["windows blue", "amber", "greyish",
                   "faded green", "dusty purple"]
         sns.palplot(sns.xkcd_palette(colors))
```



Interactive visualization for picking colors http://www.luminoso.com/colors/

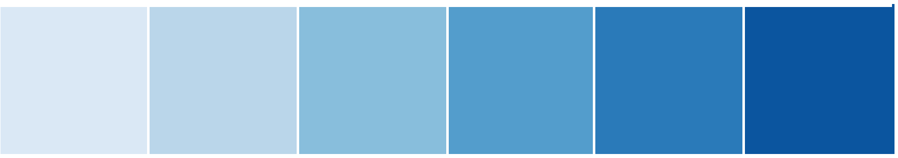
2.3 Sequential color palettes

- Often useful for colormap in functions like `kdeplot` or `corrplot`
- For sequential data, good to use palettes with a subtle shift in hues and a large shift in brightness or saturation

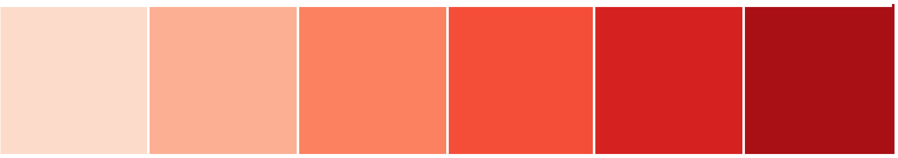
- Color Brewer has a set of these palettes named after the dominant color(s)

e.g.

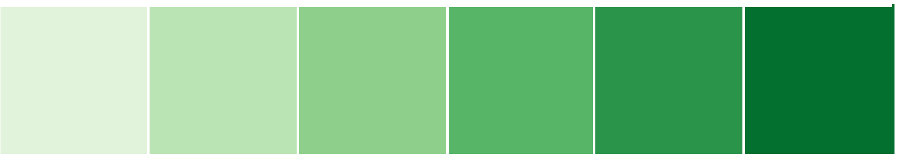
```
In [20]: sns.palplot(sns.color_palette('Blues'))
```



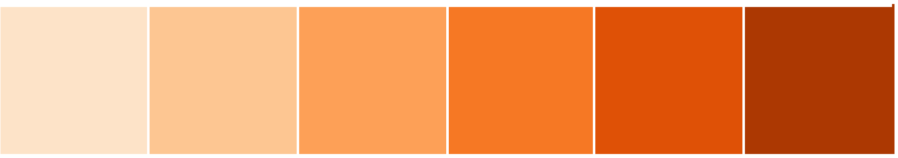
```
In [21]: sns.palplot(sns.color_palette('Reds'))
```



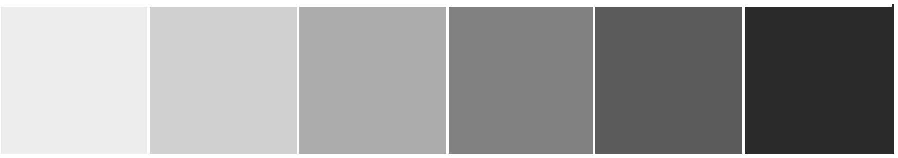
```
In [22]: sns.palplot(sns.color_palette('Greens'))
```



```
In [23]: sns.palplot(sns.color_palette('Oranges'))
```

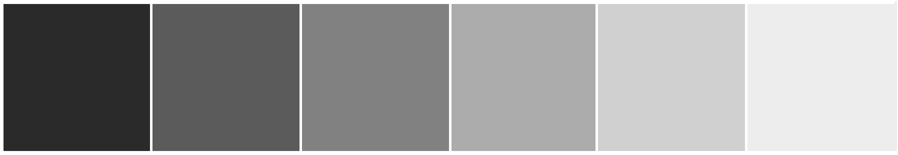


```
In [24]: sns.palplot(sns.color_palette('Greys'))
```

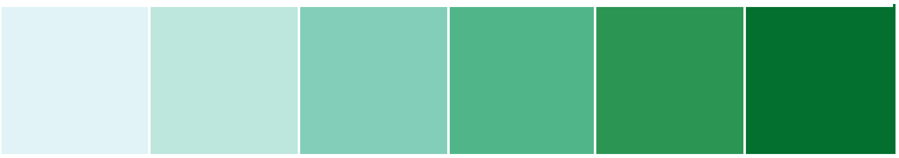


Reverse lightness map - append '_r' after the name

```
In [25]: sns.palplot(sns.color_palette('Greys_r'))
```

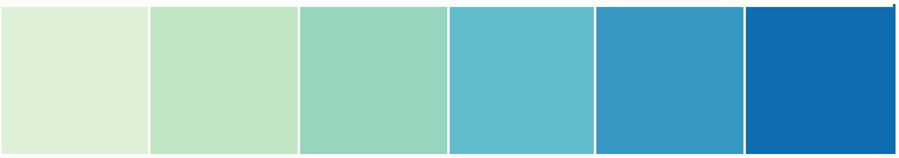


```
In [26]: sns.palplot(sns.color_palette('BuGn'))
```

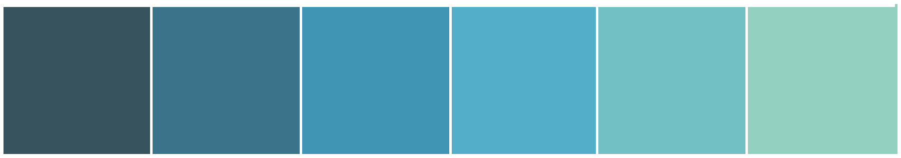


'Dark' palettes - add '_d'

```
In [27]: sns.palplot(sns.color_palette('GnBu'))
```



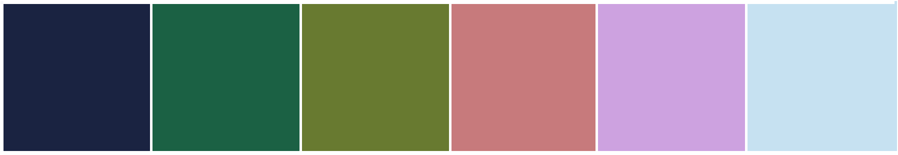
```
In [28]: sns.palplot(sns.color_palette('GnBu_d'))
```



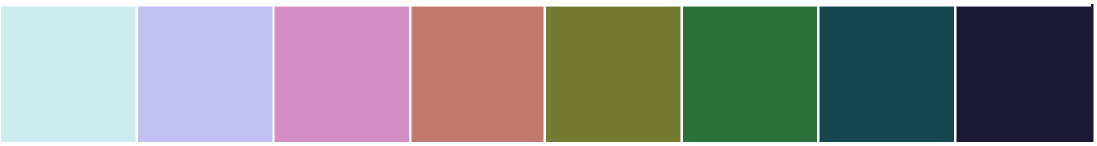
2.4 Sequential palettes with cubehelix_palette

Make Sequential palettes with a linear increase/decrease in brightness and some variation in hue

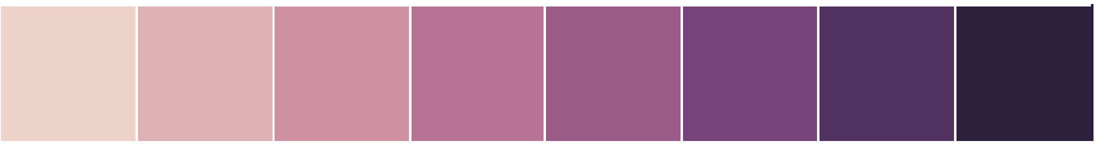
```
In [29]: sns.palplot(sns.color_palette('cubehelix', 6))
```



```
In [30]: sns.palplot(sns.color_palette('cubehelix_r', 8))
```

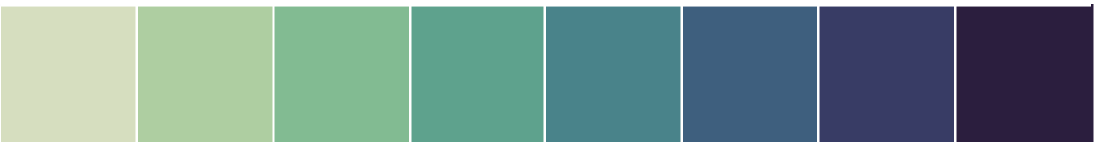


```
In [31]: sns.palplot(sns.cubehelix_palette(8))
```

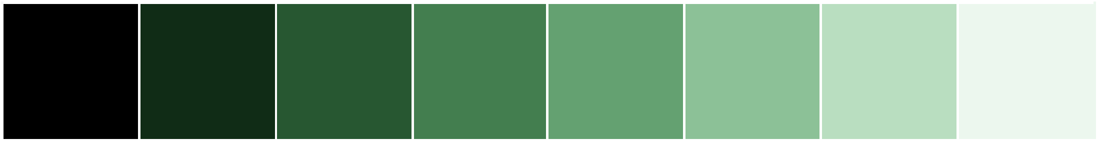


params: * start : value between 0 and 3 * rot: number of rotations (probably btw -1 and 1)

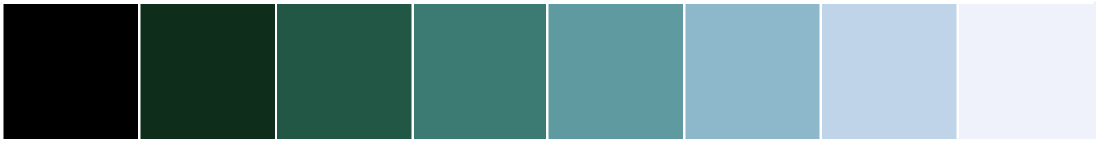
```
In [32]: sns.palplot(sns.cubehelix_palette(8, start = 0.5, rot = -.75))
```



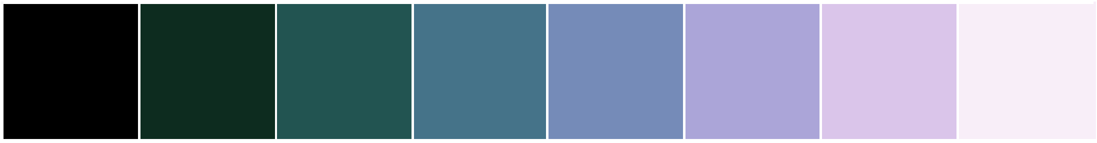
```
In [33]: sns.palplot(sns.cubehelix_palette(
             8, start = 2, rot = 0,
             dark = 0, light = 0.95,
             reverse = True))
```



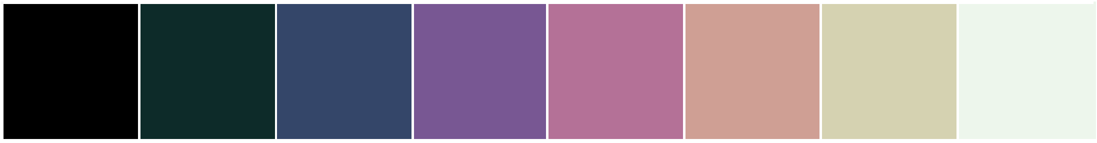
```
In [34]: sns.palplot(sns.cubehelix_palette(
                8, start = 2, rot = 0.3,
                dark = 0, light = 0.95,
                reverse = True))
```



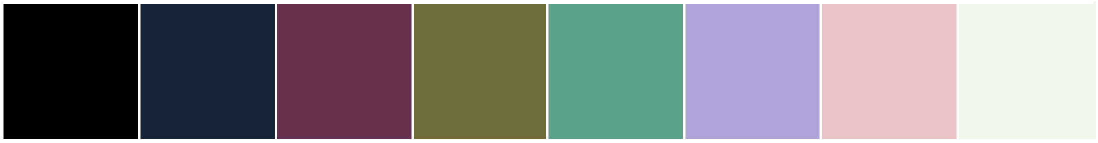
```
In [35]: sns.palplot(sns.cubehelix_palette(
                8, start = 2, rot = 0.5,
                dark = 0, light = 0.95,
                reverse = True))
```



```
In [36]: sns.palplot(sns.cubehelix_palette(
                8, start = 2, rot = 1,
                dark = 0, light = 0.95,
                reverse = True))
```



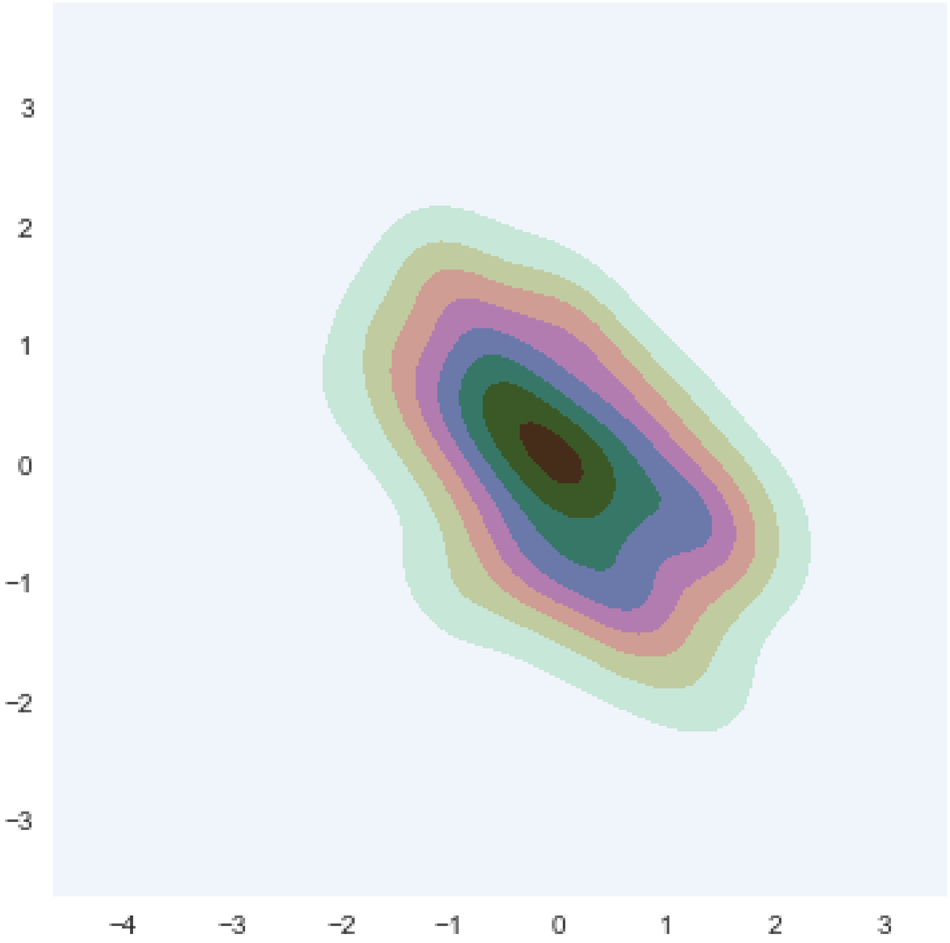
```
In [37]: sns.palplot(sns.cubehelix_palette(
                8, start = 2, rot = 2,
                dark = 0, light = 0.95,
                reverse = True))
```



as_cmap = True Get a colormap object

```
In [38]: x, y = np.random.multivariate_normal(
             [0, 0], [[1, -0.5], [-0.5, 1]],
             size = 300).T
         cmap = sns.cubehelix_palette(start = 3, rot = 2,
                                      light = 1, as_cmap = True)
         sns.kdeplot(x, y, cmap = cmap, shade = True)
```

```
Out[38]: <matplotlib.axes._subplots.AxesSubplot at 0x1121ea150>
```

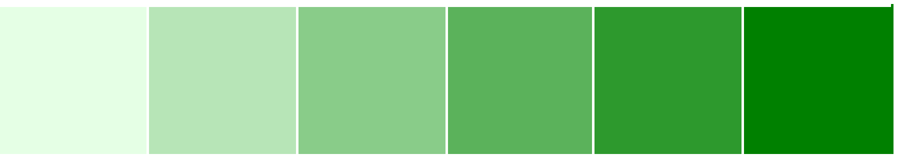


2.5 Custom Sequential palettes with light_palette

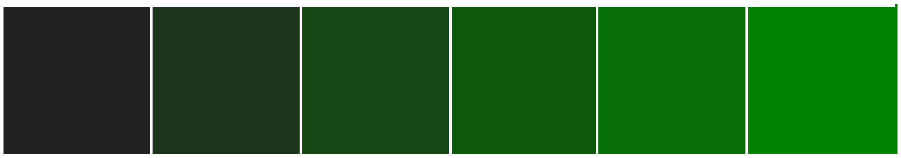
and dark_palette

- choose_light_palette()
- choose_dark_palette()

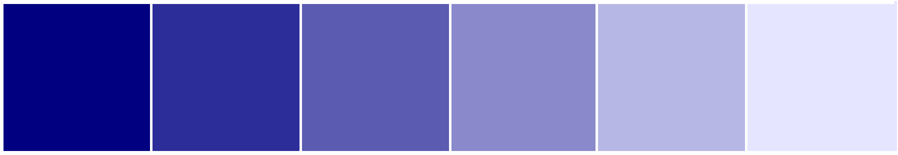
```
In [39]: sns.palplot(sns.light_palette('green'))
```



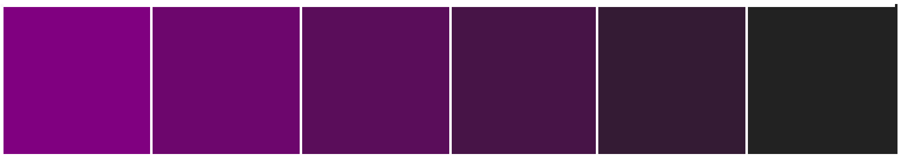
```
In [40]: sns.palplot(sns.dark_palette('green'))
```



```
In [41]: sns.palplot(sns.light_palette('navy', reverse = True))
```

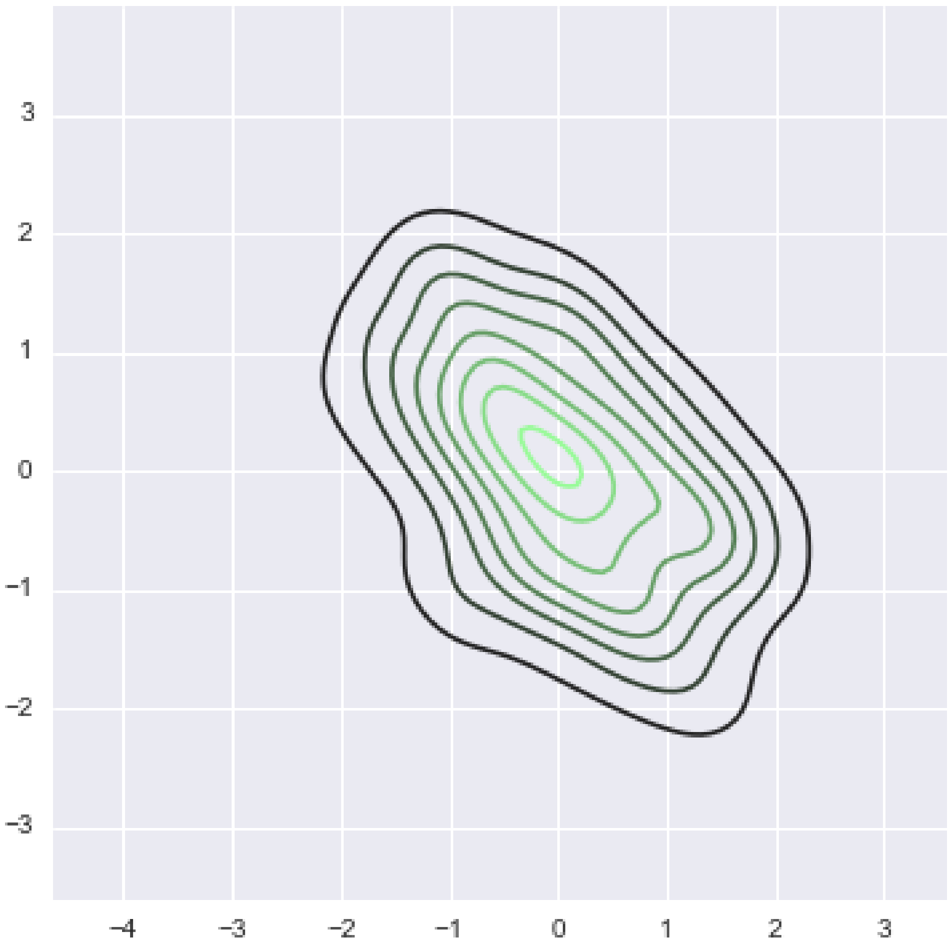


```
In [42]: sns.palplot(sns.dark_palette('purple', reverse = True))
```

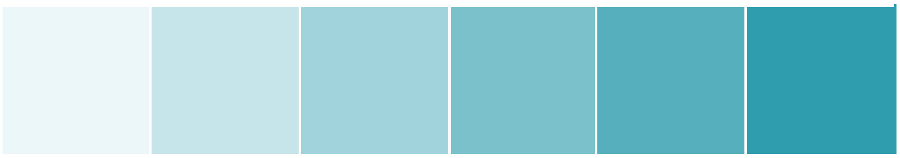


```
In [43]: pal = sns.dark_palette('palegreen', as_cmap = True)
         sns.kdeplot(x, y, cmap = pal)
```

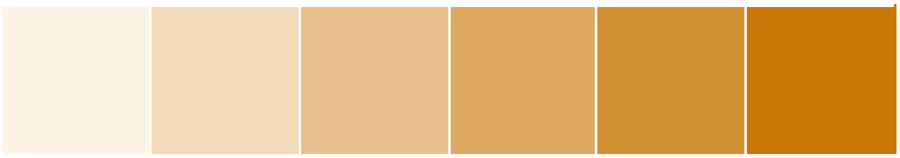
```
Out[43]: <matplotlib.axes._subplots.AxesSubplot at 0x112696690>
```



```
In [44]: sns.palplot(sns.light_palette((210, 90, 60), input = 'husl'))
```



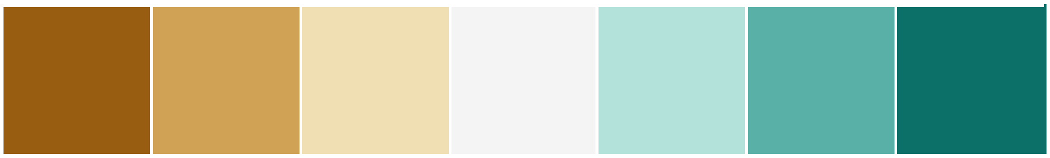
```
In [45]: sns.palplot(sns.light_palette('dirty orange', input = 'xkcd'))
```



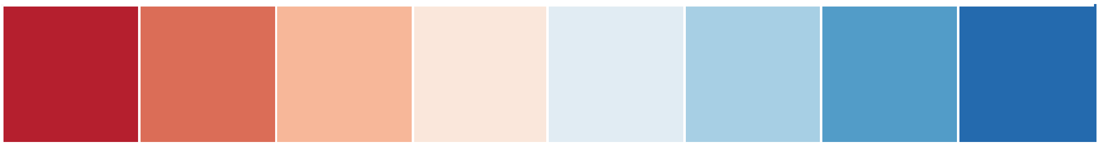
2.6 Diverging color palettes

- Used for data where both large low and high values are interesting
- Data usually has a well-defined midpoint
- Avoid Red/Green (indistinguishable for color-blind)

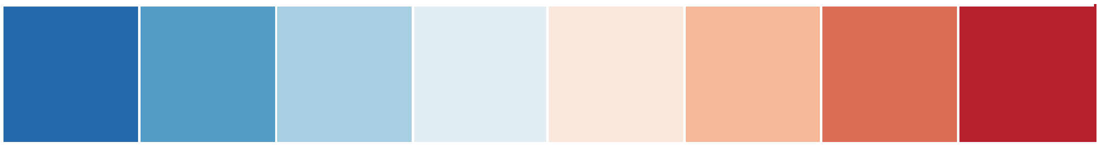
```
In [46]: sns.palplot(sns.color_palette('BrBG', 7))
```



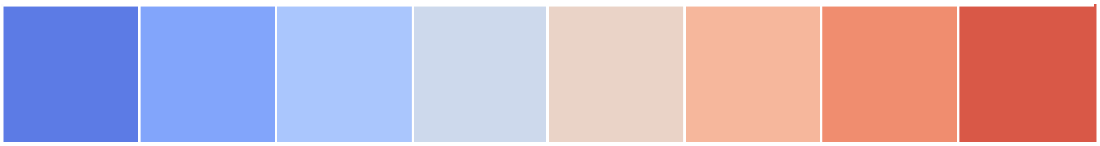
```
In [47]: sns.palplot(sns.color_palette('RdBu', 8))
```



```
In [48]: sns.palplot(sns.color_palette('RdBu_r', 8))
```



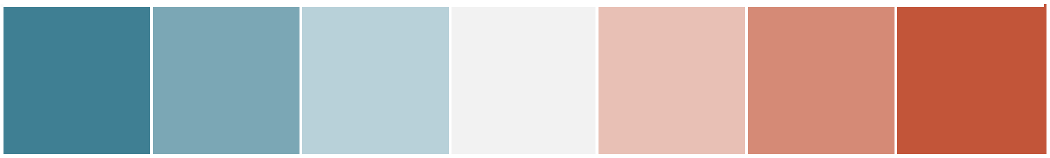
```
In [49]: sns.palplot(sns.color_palette('coolwarm', 8))
```



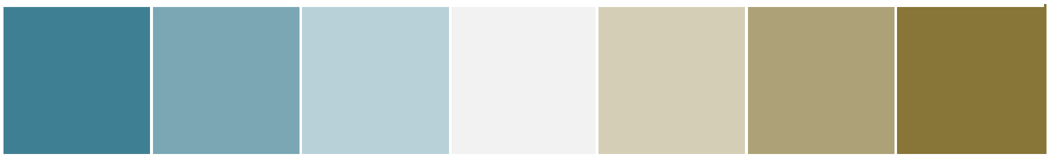
2.6.1 Custom diverging palettes with diverging_palette()

- interactive function:choose_diverging_palette‘

```
In [50]: sns.palplot(sns.diverging_palette(220, 20, n=7))
```

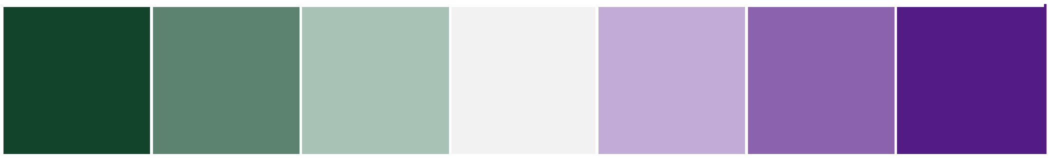


```
In [51]: sns.palplot(sns.diverging_palette(220, 70, n=7))
```



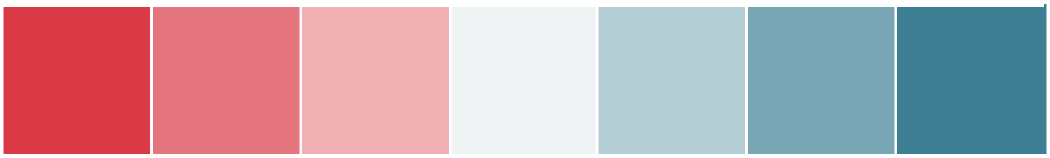
Specify lightness and saturation (husl)

```
In [52]: sns.palplot(sns.diverging_palette(145, 280, s = 85, l = 25, n = 7))
```

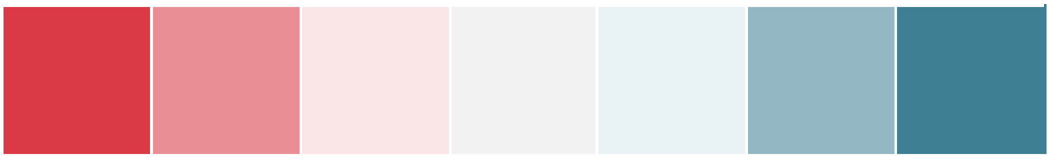


sep controls width of separation between 2 ramps in the middle of the region

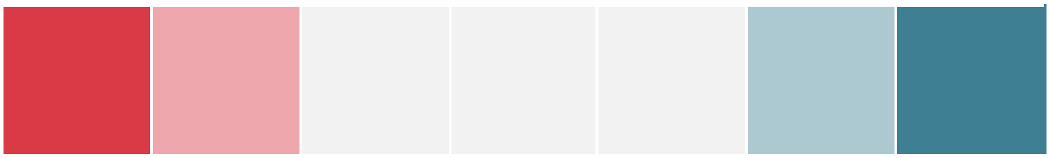
```
In [53]: sns.palplot(sns.diverging_palette(10, 220, sep = 1, n=7))
```



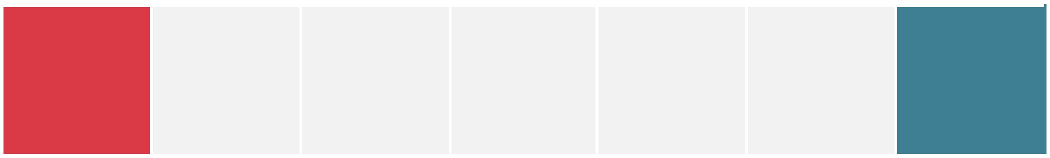
```
In [54]: sns.palplot(sns.diverging_palette(10, 220, sep = 80, n=7))
```



```
In [55]: sns.palplot(sns.diverging_palette(10, 220, sep = 120, n=7))
```

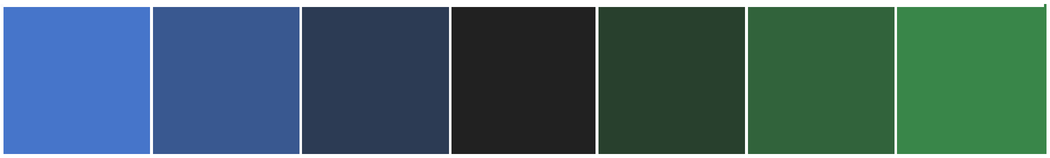


```
In [56]: sns.palplot(sns.diverging_palette(10, 220, sep = 180, n=7))
```

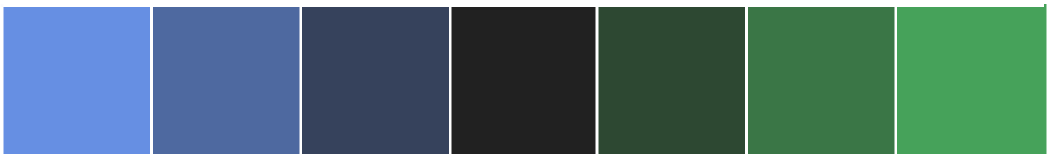


Make palette with dark midpoint

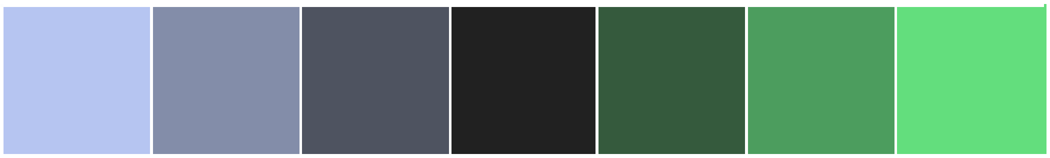
```
In [57]: sns.palplot(sns.diverging_palette(255, 133, n=7, center = ’dark’))
```



```
In [58]: sns.palplot(sns.diverging_palette(
             255, 133, l = 60, n=7, center = ’dark’))
```



```
In [59]: sns.palplot(sns.diverging_palette(
             255, 133, l = 80, n=7, center = 'dark'))
```

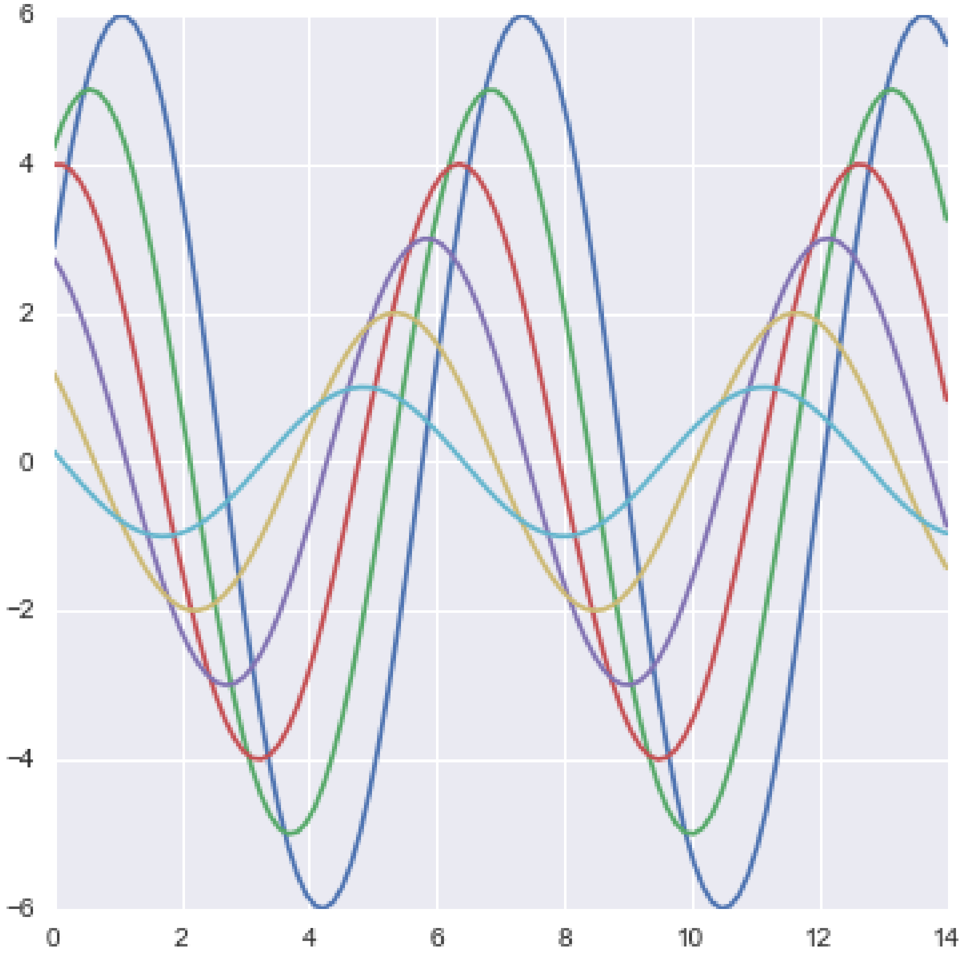


2.7 Changing default palettes with `set_palette`

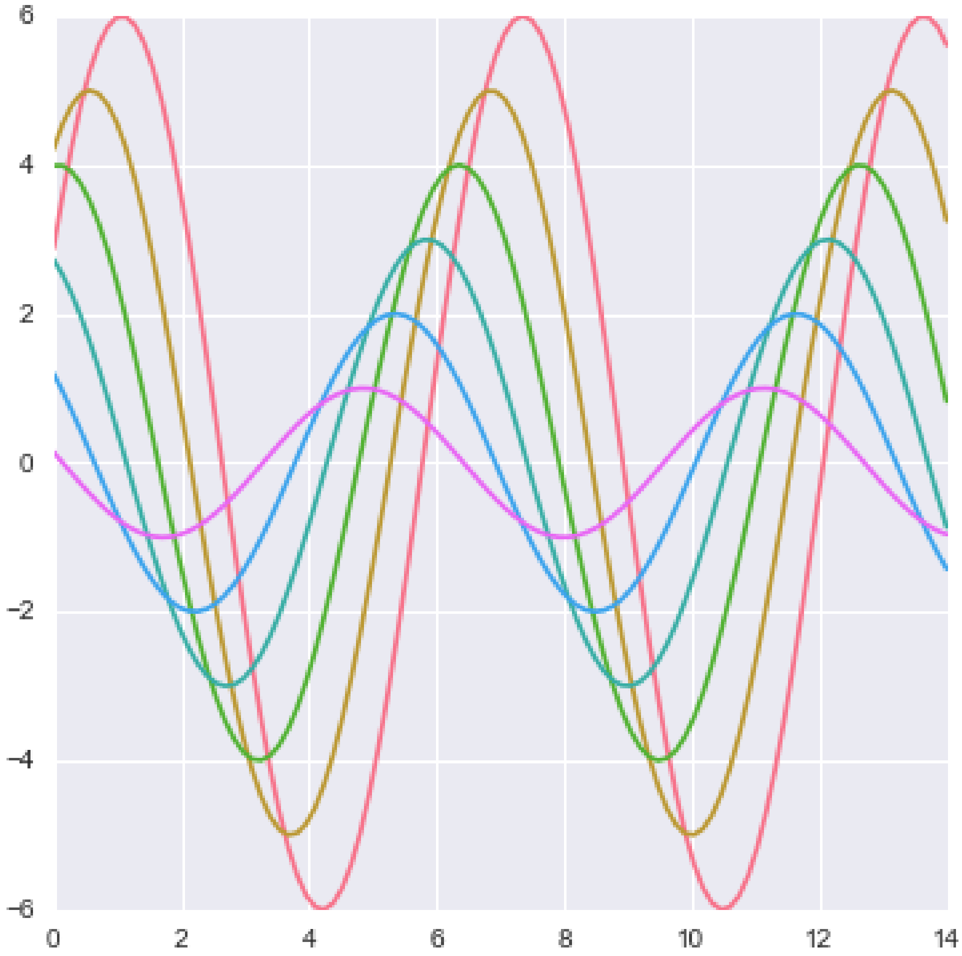
- `set_palette` & `color_palette` are companion functions
- Accept the same arguments
- `set_palette` changes the default matplotlib parameters so that the palette is used for all plots

```
In [60]: def sinplot(flip = 1):
             x = np.linspace(0, 14, 100)
             for i in range(1, 7):
                 plt.plot(x, np.sin(x + i * 0.5) * (7 - i) * flip)
```

```
In [61]: sinplot()
```

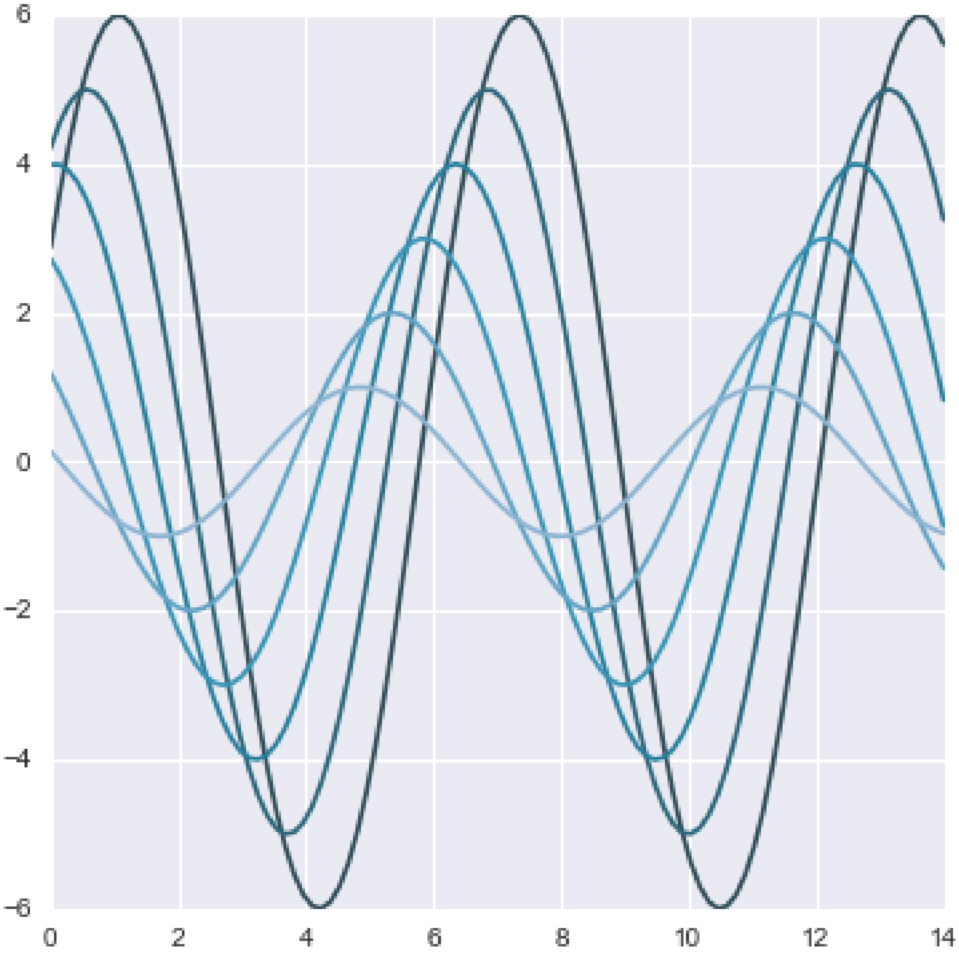


```
In [62]: sns.set_palette('husl')
         sinplot()
```



Temporarily change the color palette

```
In [63]: with sns.color_palette('PuBuGn_d'):
             sinplot()
```



```
In [ ]:
```