day-night-cycle

December 22, 2023

1 Night Day Cycle

Define the Palette of colors for a Night and Day Cycle

1.1 Seaborn Color Palette

1.1.1 Introduction

Seaborn Color Palette

Color is an utmost important aspect of figure styling because it reveals pattern in the data if used effectively; or hide those patterns if used poorly. Even professionals often assume usage of color to portray data as a solved problem. They just pick a palette from a drop-down menu (probably either a grayscale ramp or a rainbow), set start and end points & finally press apply. But it isn't that simple and thus many visualizations fail to represent the underlying data as appropriately as they could.

Primary objective with choice of color is to illuminate datapoints that are concealed in huge datasets. Quoting Robert Simmon: >Although the basics are straightforward, a number of issue complicate color choices in visualization. Among them: The relationship between the light we see and the colors we perceive is extremely complicated. There are multiple types of data, each suited to a different color scheme. A significant number of people (mostly men), are color blind. Arbitrary color choices can be confusing for viewers unfamiliar with a data set. Light colors on a dark field are perceived differently than dark colors on a bright field, which can complicate some visualization tasks, such as target detection.

One of the most fundamental and important aspects of color selection is the mapping of numbers to colors. This mapping allows us to pseudocolor an image or object based on varying numerical data. By far, the most common color map used in scientific visualization is the *rainbow* color map. Research paper on **Diverging Color Maps for Scientific Visualization** by Kenneth Moreland very well deals with the extended color concepts, if the topic interests you for further analysis.

With all that been said, let us now focus on what Seaborn has to offer BUT before doing that let me once again remind you that Seaborn runs on top of Matplotlib so any color that is supported by **Matplotlib** will be supported by Seaborn as well. So at first, let us understand what Matplotlib has to offer:

- an RGB or RGBA tuple of float values in [0, 1] (e.g., (0.1, 0.2, 0.5) or (0.1, 0.2, 0.5, 0.3))
- a hex RGB or RGBA string (e.g., '#0F0F0F' or '#0F0F0F0F')
- a string representation of a float value in [0, 1] inclusive for gray level (e.g., '0.5')
- one of {'b', 'g', 'r', 'c', 'm', 'y', 'k', 'w'}

- a X11/CSS4 color name
- a name from the xkcd color survey prefixed with 'xkcd:' (e.g., 'xkcd:sky blue')
- one of {'C0', 'C1', 'C2', 'C3', 'C4', 'C5', 'C6', 'C7', 'C8', 'C9'}
- one of {'tab:blue', 'tab:orange', 'tab:green', 'tab:red', 'tab:purple', 'tab:brown', 'tab:pink', 'tab:gray', 'tab:olive', 'tab:cyan'} which are the **Tableau** Colors from the 'T10' categorical palette (which is the default color cycle).

Note that all string specifications of color, other than "CN", are NOT case-sensitive. Let us briefly go through a couple of common supported colors here: - RGB/RGBA tuples are 4-tuples where the respective tuple components represent Red, Green, Blue, and Alpha (opacity) values for a color. Each value is a floating point number between 0.0 and 1.0. For example, the tuple (1, 0, 0, 1) represents an opaque red, while (0, 1, 0, 0.5) represents a half transparent green. - This is actually another way of representing RGBA codes and common Color Conversion Calculators can be used to translate values. Here is a **Hex to RGBA** and **RGB to Hex** Color converter for your future assistance. - Dictionary of values from {'C0', 'C1', 'C2', 'C3', 'C4', 'C5', 'C6', 'C7', 'C8', 'C9'} represent **Color Quantization**. I have attached a link in the provided notebook that shall guide you to an online book where on Page-29 you could find specifics.

1.1.2 Setup Environment

Import Library and setup global configuration

[ ]:
```
# Importing required Libraries:
import math
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

# Setting a figure size for all the plots we shall be drawing in this kernel:
sns.set(rc={"figure.figsize": (6, 6)})
```

1.1.3 Test Color Palette

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

[ ]:
```
sample_colors = ["windows blue", "amber", "greyish", "faded green", "dusty␣
↪purple", "pale red", "medium green", "denim blue"]
sns.palplot(sns.xkcd_palette(sample_colors))
```

[ ]:
```
sample_colors_rgb = ["#7e1e9c", "#0343df", "#ff81c0", "#36013f"]
sns.palplot(sns.blend_palette(sample_colors_rgb, as_cmap=False, input='rgb'))
```

[ ]:
```
sample_colors_rgb = ["#7e1e9c", "#0343df", "#15b01a"]
sns.palplot(sns.blend_palette(sample_colors_rgb,n_colors=3))
```

1.1.4 Convert Decimal to Hexadecimal

[ ]:
```
print(hex(0))
print("%02x" % 255)
print("%02x" % 0)
print("%02x" % 160)
```

```
0x0
ff
00
a0
```

1.2 Day Night Cycle

[ ]:
```
sample_colors_rgb = ["#000050", "#0000FF", "#00A0ff"]
sns.palplot(sns.blend_palette(sample_colors_rgb,n_colors=12))
```

1.3 Day Night Functions

1.3.1 Functions

Base Function $a \in \mathbb{R}$ is 'height' of the curve

$b \in \mathbb{R}$ is the x-position of the peak of the curve

$c \in \mathbb{R}$ is the the width of the curve

$f(x) = a \exp\left(-\frac{(x-b)^2}{2c^2}\right)$

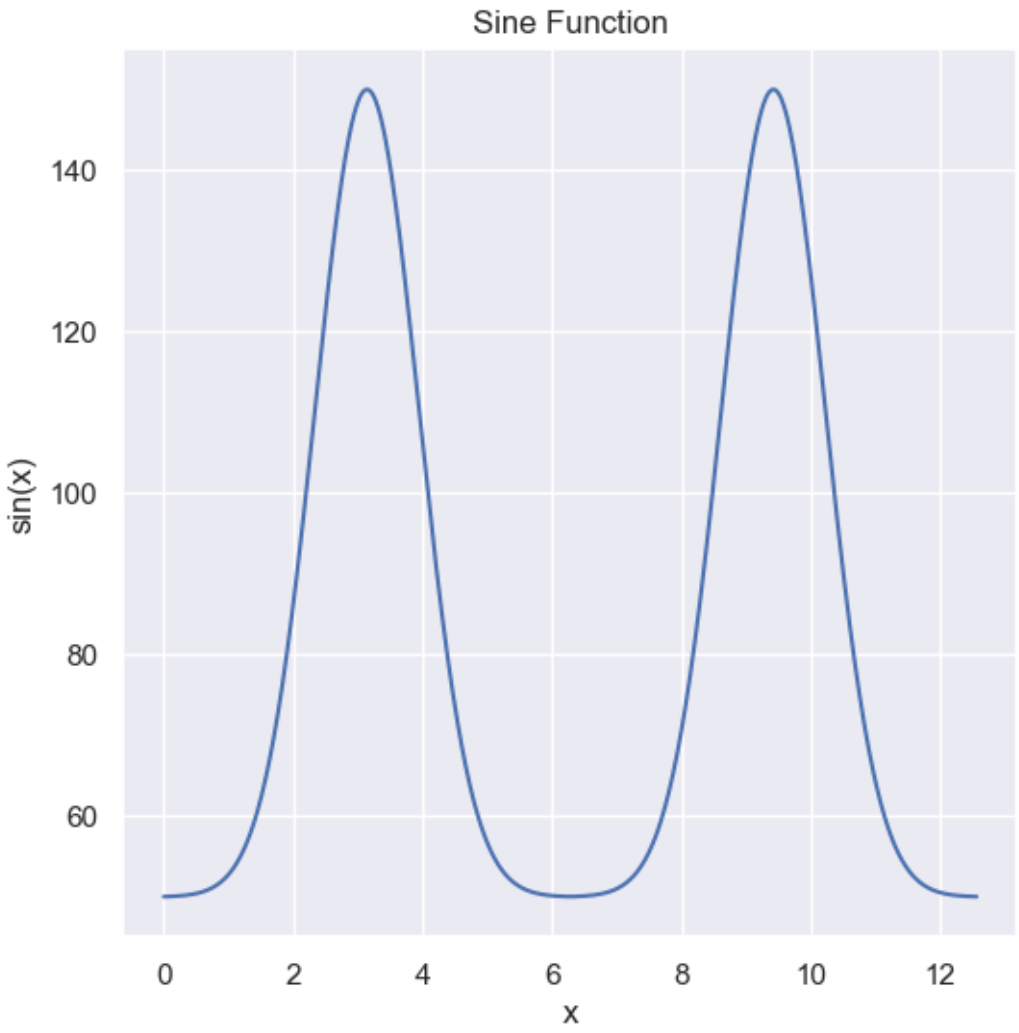
[ ]:
```
def gaussian(x, height, position, length):
    """Gaussian (normal) distribution function."""
    return height * np.exp(-(x - position)**2 / (2 * length**2))
```

Sky Blue $f(x) = \left(100 * \exp\left(-\frac{(x-\pi)^2}{2*(0.8^2)}\right)\right) + 50$

[ ]:
```
# Function Blu Day Nigth Cycle

def blue_DayNightCycle(x):
    """This function takes an angle in radians (x) and returns the maximum␣
 ↪between sin(x) and 0."""
    shift = 50;
    #return max(((150-shift) * np.sin(x)) + shift, 0)
    cycle = math.floor(x / (2 * np.pi))
    return gaussian(x, 100, (cycle * 2 * np.pi) + (np.pi), 0.8) + 50
```

[ ]:
```
# Generate x values from 0 to 2*pi with small intervals
x = np.linspace(0, 4 * np.pi, 1000)

# Calculate y values for the sine function
y = np.vectorize(blue_DayNightCycle)(x)

# Plot the sine function
plt.plot(x, y)
```

```
# Add labels and title
plt.title('Sine Function')
plt.xlabel('x')
plt.ylabel('sin(x)')

# Show the plot
plt.show()
```

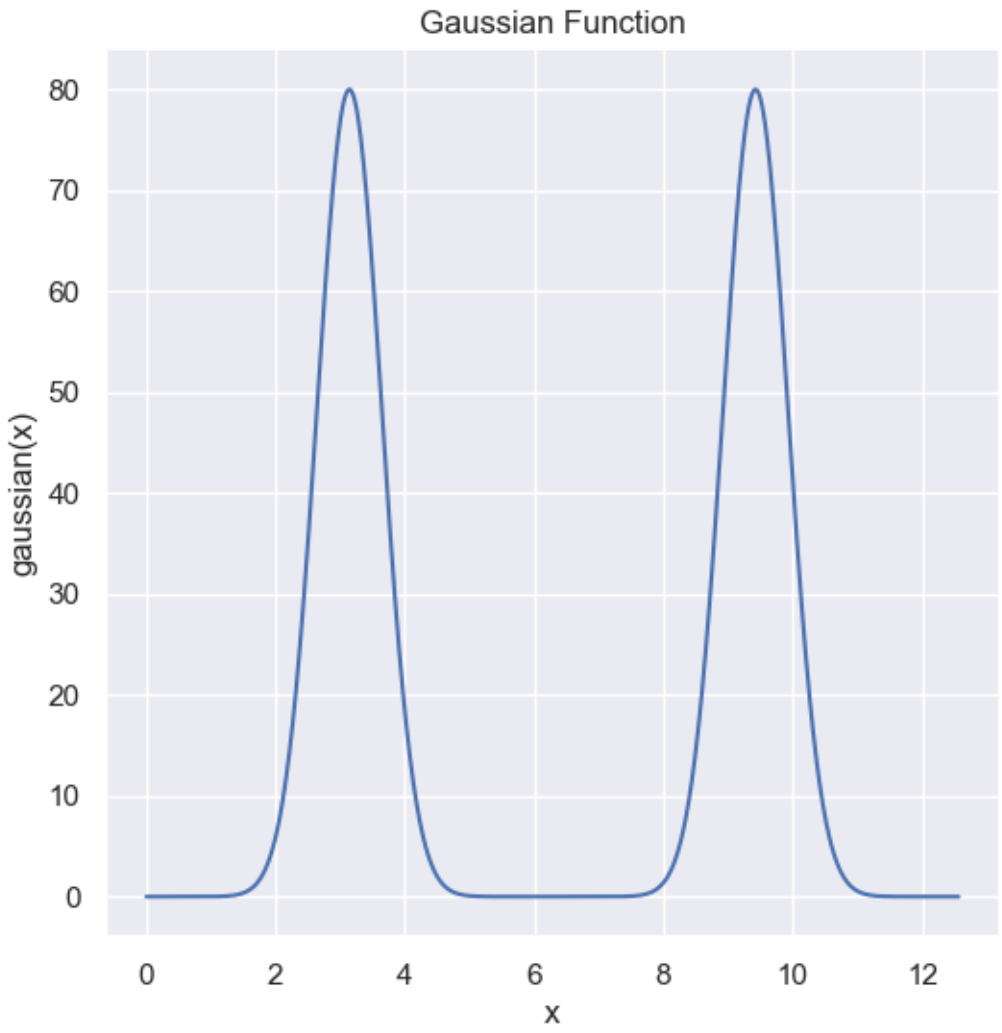


Sky Green $f(x) = \left(80 * \exp\left(-\frac{(x-\pi)^2}{2*(0.5^2)}\right)\right)$

[ ]:
```
def green_DayNightCycle(x):
    """This function takes an angle in radians (x) and returns the maximum␣
 ↪between sin(x) and 0."""
    cycle = math.floor(x / (2 * np.pi))
```

```
    return gaussian(x, 80, (cycle * 2 * np.pi) + (np.pi), 0.5)
```

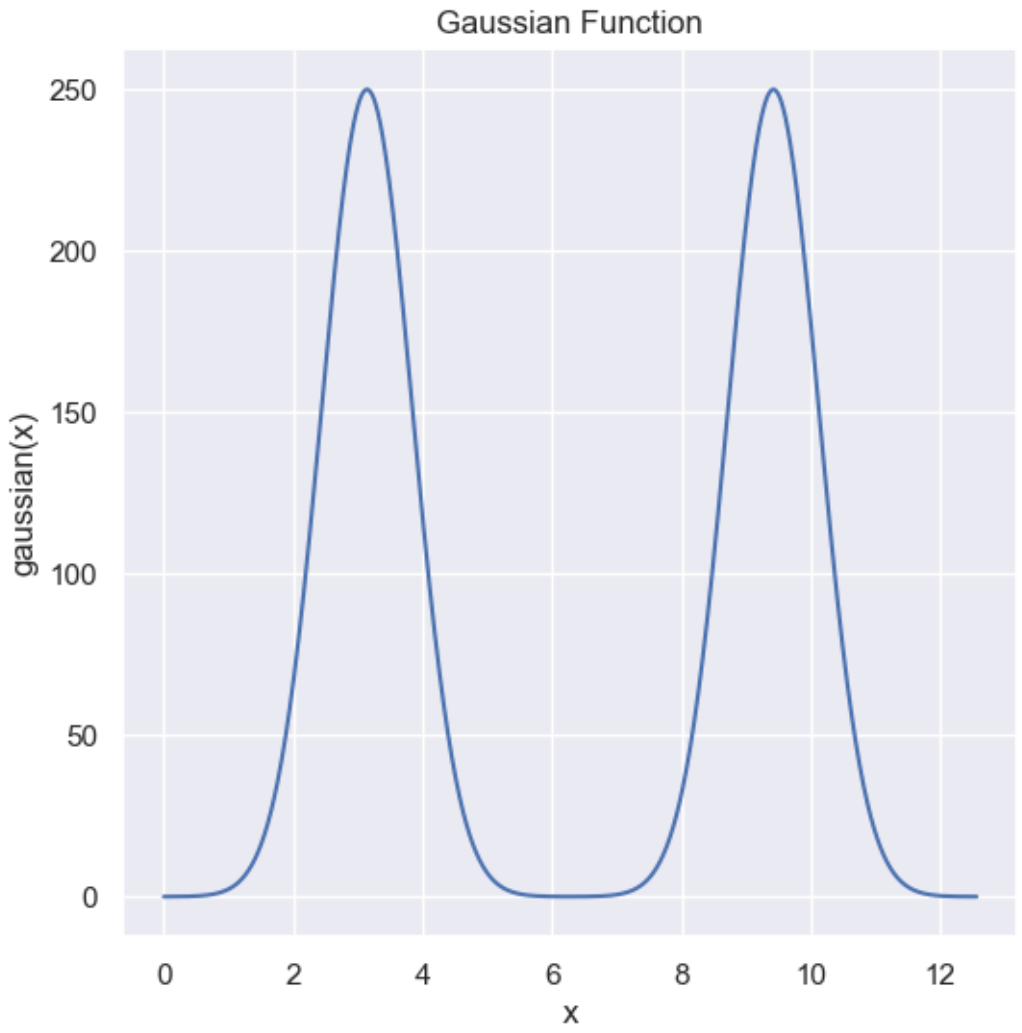
[ ]:
```
# Generate x values from 0 to 2*pi with small intervals
x = np.linspace(0, 4 * np.pi, 1000)

# Calculate y values for the sine function
y = np.vectorize(green_DayNightCycle)(x)

# Plot the sine function
plt.plot(x, y)

# Add labels and title
plt.title('Gaussian Function')
plt.xlabel('x')
plt.ylabel('gaussian(x)')

# Show the plot
plt.show()
```



Sun Red $f(x) = \left(250 * \exp\left(-\frac{(x-\pi)^2}{2*(0.7^2)}\right)\right)$

[ ]:
```
def red_SunDayNightCycle(x):
    """This function takes an angle in radians (x) and returns the maximum between sin(x) and 0."""
    cycle = math.floor(x / (2 * np.pi))
    #cycle_quarter = math.floor(x / (np.pi /2 ))
    #shift = 95;
    #return max(((255-shift) * np.sin(x)), 0)
    #return gaussian(x,230, (cycle * 2 * np.pi) + (np.pi / 16), 0.1) + gaussian(x,230, (cycle * 2 * np.pi) + (np.pi * 15 / 16), 0.1)
    return gaussian(x,250, (cycle * 2 * np.pi) + (np.pi), 0.7)
```

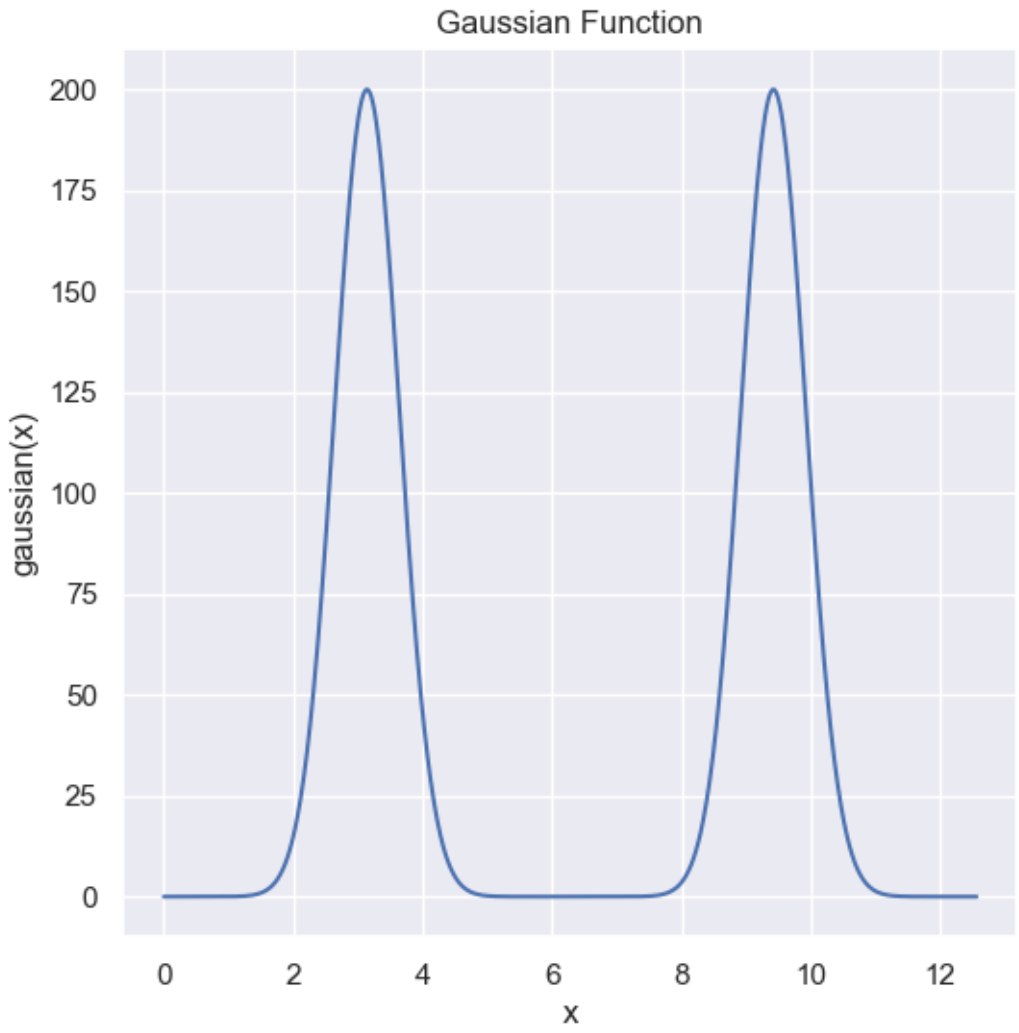
[ ]:
```
# Generate x values from 0 to 2*pi with small intervals
x = np.linspace(0, 4 * np.pi, 1000)

# Calculate y values for the sine function
y = np.vectorize(red_SunDayNightCycle)(x)

# Plot the sine function
plt.plot(x, y)

# Add labels and title
plt.title('Gaussian Function')
plt.xlabel('x')
plt.ylabel('gaussian(x)')

# Show the plot
plt.show()
```



Sun Green $f(x) = \left(200 * \exp\left(-\frac{(x-\pi)^2}{2*(0.5^2)}\right)\right)$

[ ]:
```
def green_SunDayNightCycle(x):
    """This function takes an angle in radians (x) and returns the maximum␣
 ↪between sin(x) and 0."""
    cycle = math.floor(x / (2 * np.pi))
    #cycle_quarter = math.floor(x / (np.pi /2 ))
    #shift = 95;
    #return max(((255-shift) * np.sin(x)), 0)
    #return gaussian(x,230, (cycle * 2 * np.pi) + (np.pi / 16), 0.1) +␣
 ↪gaussian(x,230, (cycle * 2 * np.pi) + (np.pi * 15 / 16), 0.1)
    return gaussian(x,200, (cycle * 2 * np.pi) + (np.pi), 0.5)
```

[ ]:
```
# Generate x values from 0 to 2*pi with small intervals
x = np.linspace(0, 4 * np.pi, 1000)

# Calculate y values for the sine function
y = np.vectorize(green_SunDayNightCycle)(x)

# Plot the sine function
plt.plot(x, y)

# Add labels and title
plt.title('Gaussian Function')
plt.xlabel('x')
plt.ylabel('gaussian(x)')

# Show the plot
plt.show()
```



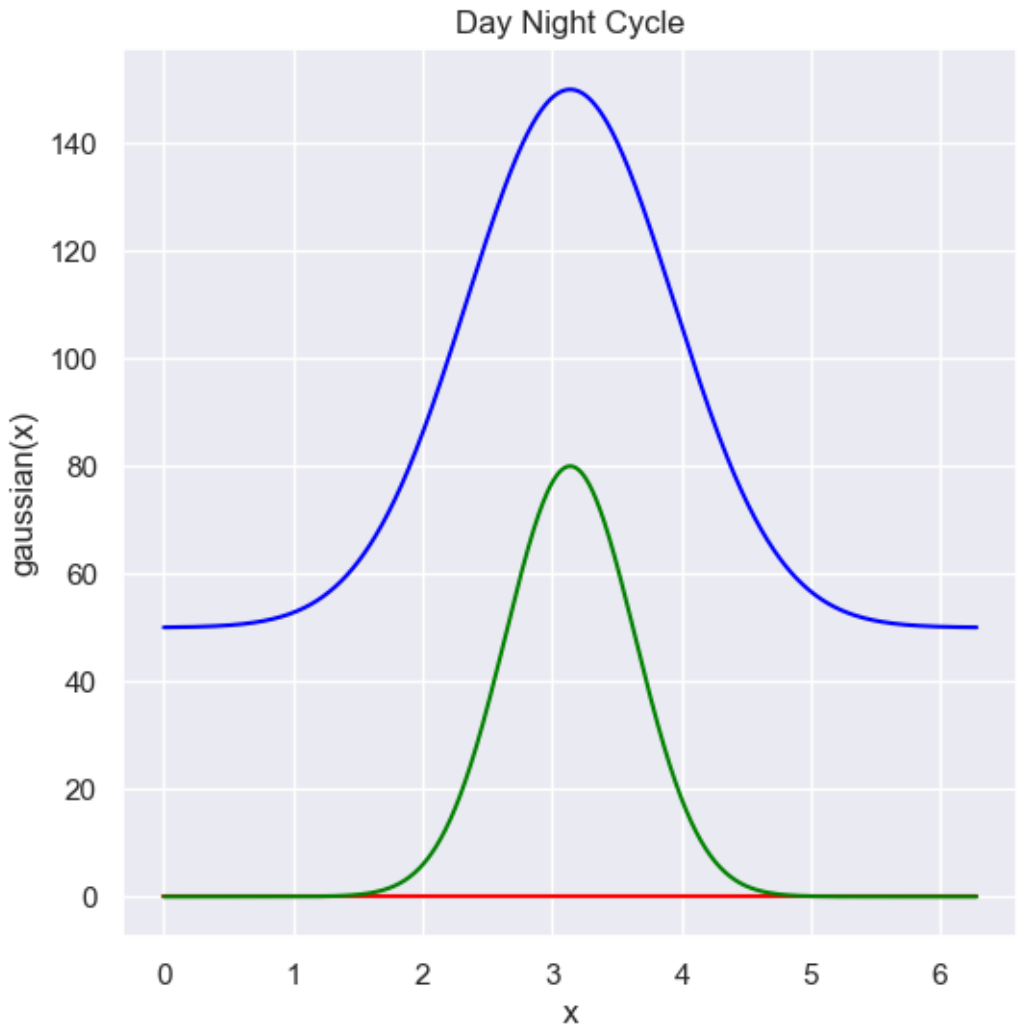
Sky + Sun Green $f(x) = \min\left(200 * \exp\left(-\frac{(x-\pi)^2}{2*(0.5^2)}\right)\right) + \left(80 * \exp\left(-\frac{(x-\pi)^2}{2*(0.5^2)}\right), 250\right)$

[ ]:
```
def green_min_of_sum(x):
    return min(green_DayNightCycle(x) + green_SunDayNightCycle(x), 220)
```

1.3.2 Sky Day Night Cycle

[ ]:
```
def constant_zero_function(x):
    return 0

# Generate x values from 0 to 2*pi with small intervals
x = np.linspace(0, 2 * np.pi, 1000)

# Calculate y values for the sine function
```

```
y = np.vectorize(blue_DayNightCycle)(x)
# Plot the sine function
plt.plot(x, y, color='blue')

y = np.vectorize(constant_zero_function)(x)
plt.plot(x, y, color='red')

# Calculate y values for the sine function
y = np.vectorize(green_DayNightCycle)(x)
# Plot the sine function
plt.plot(x, y, color='green')



# Add labels and title
plt.title('Day Night Cycle')
plt.xlabel('x')
plt.ylabel('gaussian(x)')

# Show the plot
plt.show()
```



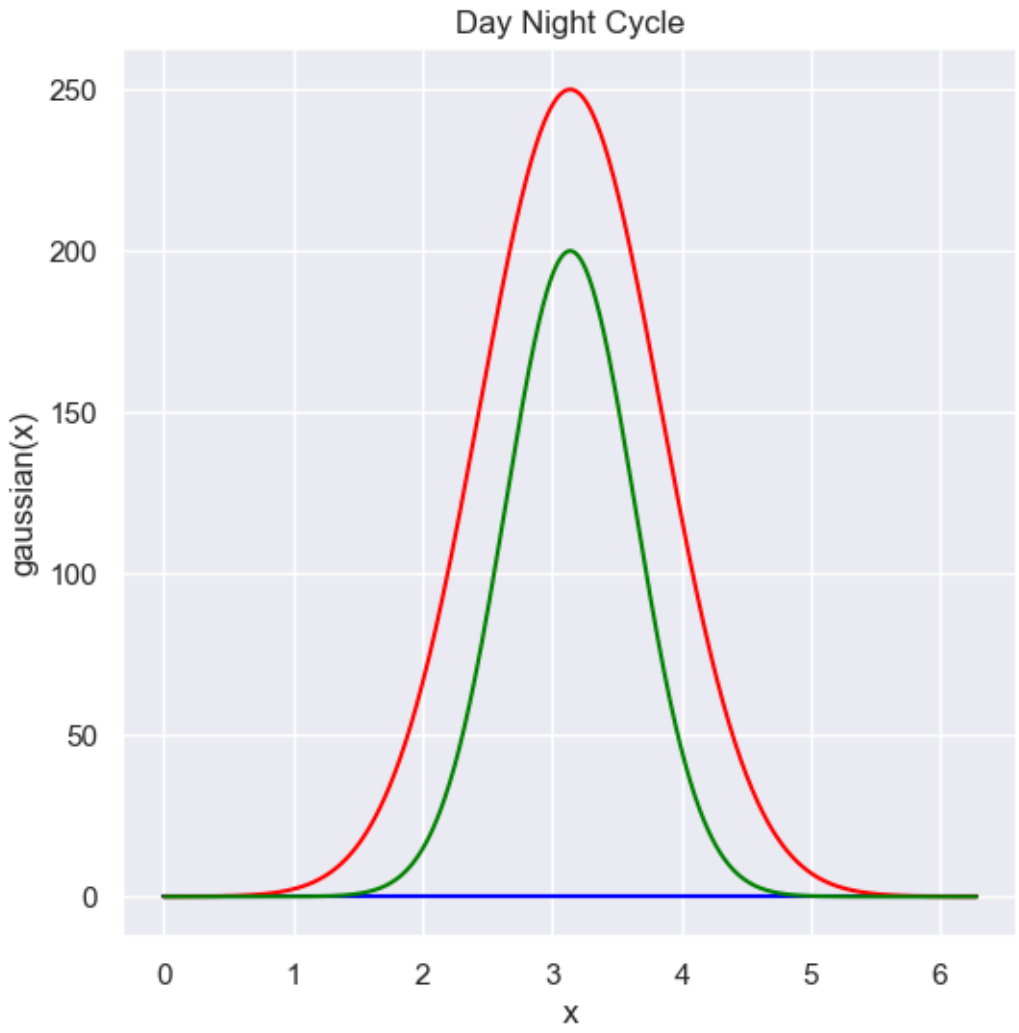
Color Palette

[ ]:
```
# Generate x values from 0 to 2*pi with large intervals
interval = 40
x = np.linspace(0, 2 * np.pi, interval)
y = [""] * interval

for i in range(interval):
    # red = red_DayNightCycle(x[i])
    green = green_DayNightCycle(x[i])
    blu = blue_DayNightCycle(x[i])

    #rgb = "#" + "%02x" % int(red) + "%02x" % int(green) + "%02x" % int(blu)
    rgb = "#00" + "%02x" % int(green) + "%02x" % int(blu)
    # print(rgb)
```

```
    y[i] = rgb

sns.palplot(sns.blend_palette(y,n_colors=interval))
```

1.3.3 Sun Day Night Cycle

[ ]:
```
def constant_zero_function(x):
    return 0

# Generate x values from 0 to 2*pi with small intervals
x = np.linspace(0, 2 * np.pi, 1000)

# Calculate y values for the sine function
y = np.vectorize(constant_zero_function)(x)
# Plot the sine function
plt.plot(x, y, color='blue')

y = np.vectorize(red_SunDayNightCycle)(x)
plt.plot(x, y, color='red')

# Calculate y values for the sine function
y = np.vectorize(green_SunDayNightCycle)(x)
# Plot the sine function
plt.plot(x, y, color='green')



# Add labels and title
plt.title('Day Night Cycle')
plt.xlabel('x')
plt.ylabel('gaussian(x)')

# Show the plot
plt.show()
```



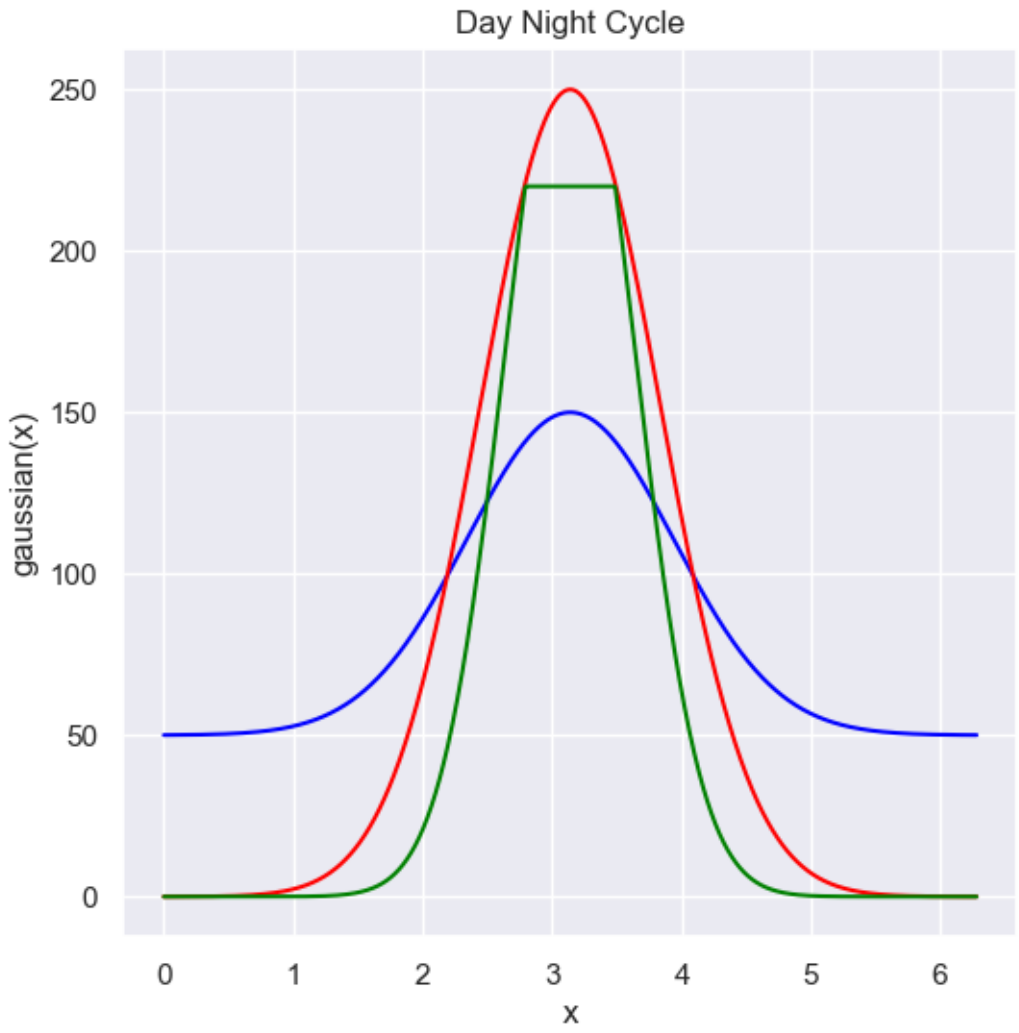
Color Palette

[ ]:
```
interval = 40
x = np.linspace(0, 2 * np.pi, interval)
y = [""] * interval

for i in range(interval):
    red = red_SunDayNightCycle(x[i])
    green = green_SunDayNightCycle(x[i])
    # blu = blue_DayNightCycle(x[i])

    rgb = "#" + "%02x" % int(red) + "%02x" % int(green) + "00"
    # print(rgb)

    y[i] = rgb
```

```
sns.palplot(sns.blend_palette(y,n_colors=interval))
```

1.3.4 Sky + Sun Day Night Cycle

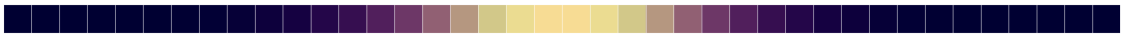
[ ]:
```
# Generate x values from 0 to 2*pi with small intervals
x = np.linspace(0, 2 * np.pi, 1000)

# Calculate y values for the sine function
y = np.vectorize(blue_DayNightCycle)(x)
# Plot the sine function
plt.plot(x, y, color='blue')

y = np.vectorize(red_SunDayNightCycle)(x)
plt.plot(x, y, color='red')

# Calculate y values for the sine function
y = np.vectorize(green_min_of_sum)(x)
# Plot the sine function
plt.plot(x, y, color='green')



# Add labels and title
plt.title('Day Night Cycle')
plt.xlabel('x')
plt.ylabel('gaussian(x)')

# Show the plot
plt.show()
```



Day Night Cycle Sun

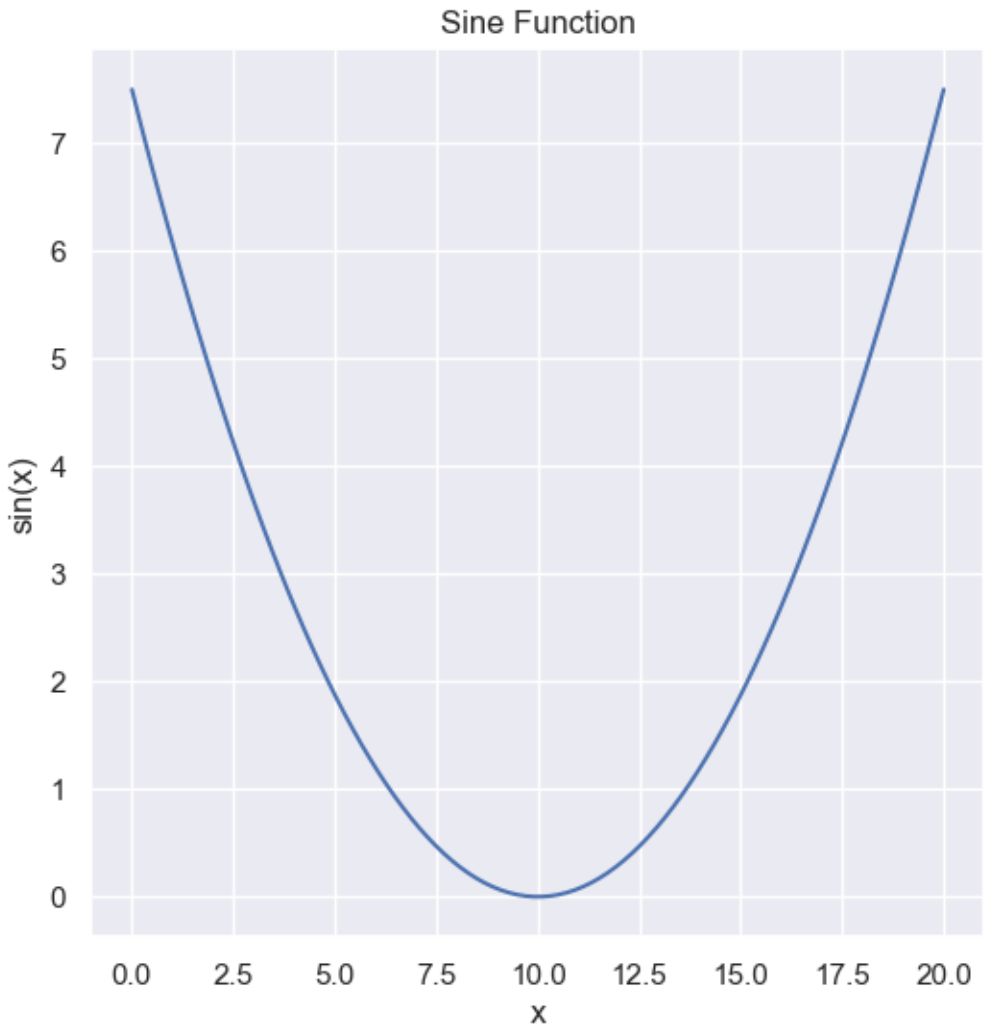
[ ]:
```
interval = 40
x = np.linspace(0, 2 * np.pi, interval)
y = [""] * interval

for i in range(interval):
    red = red_SunDayNightCycle(x[i])
    green = green_min_of_sum(x[i])
    blu = blue_DayNightCycle(x[i])

    rgb = "#" + "%02x" % int(red) + "%02x" % int(green) + "%02x" % int(blu)
    # print(rgb)

    y[i] = rgb
```

```
sns.palplot(sns.blend_palette(y,n_colors=interval))
```



1.4 Test

[ ]:
```
NUM_LED = 20
interval = 100

def quadratic(x):
    """Gaussian (normal) distribution function."""
    return ((x - (NUM_LED / 2)) ** 2) / (NUM_LED / 1.5)

x = np.linspace(0, NUM_LED, interval)

# Calculate y values for the sine function
y = np.vectorize(quadratic)(x)

# Plot the sine function
plt.plot(x, y)

# Add labels and title
plt.title('Sine Function')
plt.xlabel('x')
plt.ylabel('sin(x)')

# Show the plot
plt.show()
```



$f(x) = a * \frac{(x-b)^2}{c} + d$

[ ]:
```
NUM_LED = 60
interval = 100

def quadratic(x):
    """Gaussian (normal) distribution function."""
    return 10 * ((x - np.pi) ** 2) + 10 #/ np.pi

x = np.linspace(np.pi / 2, 3 / 2 * np.pi, interval)

# Calculate y values for the sine function
y = np.vectorize(quadratic)(x)

# Plot the sine function
```

```
plt.plot(x, y)

# Add labels and title
plt.title('Quadratic Function')
plt.xlabel('x')
plt.ylabel('f(x)')

# Show the plot
plt.show()
```