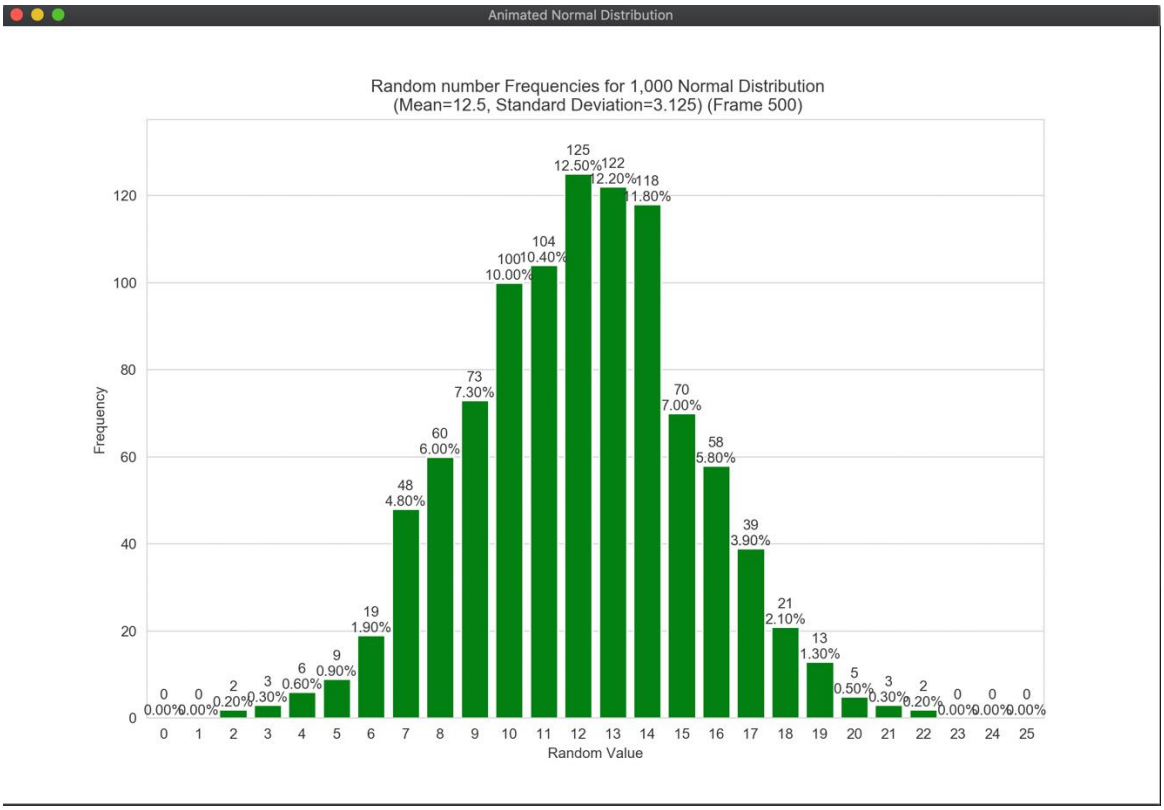
Animated Bar chart for Normal Distribution:

```
"""
Demonstrate matplotlib animation of normal values.

Use random.noramlvariate to  generator numbers
in range(1, 25) with mean value of 12.5, standard deviation of 3.125
to simulate normal distribution. Dynamically
graph frequencies of normal distribution.

"""

from matplotlib import animation
import matplotlib.pyplot as plt
import random
import seaborn as sns
import sys


def show_bar_chart(title, x_label, y_label, x_values, y_values, bar_toppers):
    """
    Display a bar chart.
    @param title the chart title.
    @param x_label the label for the x axis
    @param y_label the label for the y axis
    @param x_values the x values to plot
    @param y_values the y values to plot
    @param bar_toppers the text above each bar
    """

    plt.cla()  # clear old contents

    axes = sns.barplot(x_values, y_values, color='green')
    axes.set_title(title)
    axes.set(xlabel=x_label, ylabel=y_label)

    # Scale the y-axis by 10% to make room for text above the bars.
    axes.set_ylim(top=1.10 * max(y_values))

    # Display the topper text above each patch (bar).
    for bar, topper in zip(axes.patches, bar_toppers):
        text_x = bar.get_x() + bar.get_width() / 2
        text_y = bar.get_height()
        axes.text(text_x, text_y, topper,
                  fontsize=10, ha='center', va='bottom')


def update_frame(frame_number, count, normal_value, frequencies):
    """
    Update the bar plot contents for each animation frame.
    @param frame_number the frame number.
    @param count the count per frame.
```

```
    @param normal_value the random values.
    @param frequencies the list of random normal value frequencies.
    """

    # generate random value 'count' times and update the number frequencies.
    for _ in range(count):
        randomnumber=int(random.normalvariate(12.5,3.125))
        frequencies[randomnumber] += 1

        # Set the percentages for the bar tops.
    freq_sum = sum(frequencies)
    topper = [f'{freq:,}\n{freq / freq_sum:.2%}' for freq in frequencies]

    # Display the bar chart for this frame.
    show_bar_chart(f' Random number Frequencies for {freq_sum:,} Normal Distribution\n (Mean=12.5, Standard
Deviation=3.125) ' +
                   f'(Frame {frame_number + 2})',
                   'Random Value', 'Frequency',
                   normal_value, frequencies, topper)


# Read command-line arguments for the number of frames
# and the number of counts per frame.
number_of_frames = int(sys.argv[1])
counts_per_frame = int(sys.argv[2])

# Create the figure for the animation.
sns.set_style('whitegrid')  # white background with gray grid lines
figure = plt.figure('Animated Normal Distribution')

values = list(range(0, 26))  # random values for display on the x-axis
frequencies = [0] * 26  # 0-25 list of values frequencies

# Configure and start the animation that calls function update_frame.
normal_animation = animation.FuncAnimation(
    figure, update_frame, repeat=False,
    frames=number_of_frames - 1, interval=25,
    fargs=(counts_per_frame, values, frequencies))

plt.show()
```

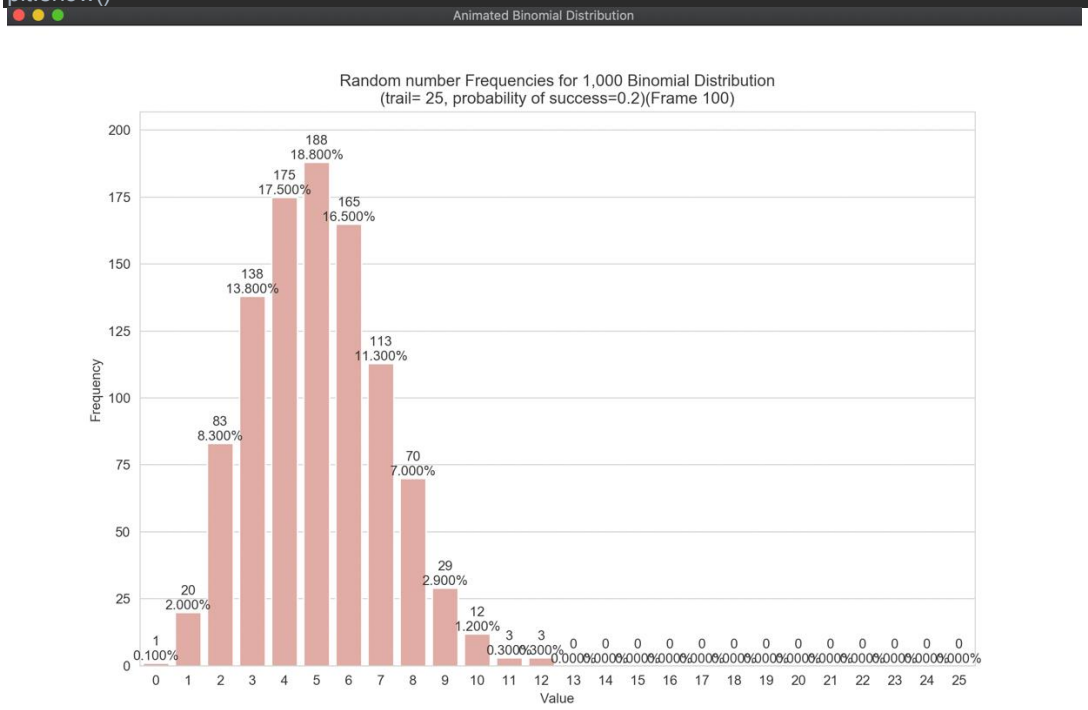


Animated Bar chart for Binomial Distribution:

```
"""
Demonstrate matplotlib animation of Binomial Distribution.

Use np.random.binomial to  generator numbers
with sample of trail of 25, probability of success of 0.2
to simulate binomial distribution. Dynamically
graph frequencies of binomial distribution.

"""


from matplotlib import animation
import matplotlib.pyplot as plt
import random
import seaborn as sns
import sys
import numpy as np


def show_bar_chart(title, x_label, y_label, x_values, y_values, bar_toppers):
    """
    Display a bar chart.
    @param title the chart title.
    @param x_label the label for the x axis
    @param y_label the label for the y axis
    @param x_values the x values to plot
```

```
    @param y_values the y values to plot
    @param bar_toppers the text above each bar
    """

    plt.cla()  # clear old contents

    axes = sns.barplot(x_values, y_values, color='salmon',saturation=.5)
    axes.set_title(title)
    axes.set(xlabel=x_label, ylabel=y_label)

    # Scale the y-axis by 10% to make room for text above the bars.
    axes.set_ylim(top=1.10 * max(y_values))

    # Display the topper text above each patch (bar).
    for bar, topper in zip(axes.patches, bar_toppers):
        text_x = bar.get_x() + bar.get_width() / 2
        text_y = bar.get_height()
        axes.text(text_x, text_y, topper,
                  fontsize=10, ha='center', va='bottom')


def update_frame(frame_number, count, binomial_value, frequencies):
    """
    Update the bar plot contents for each animation frame.
    @param frame_number the frame number.
    @param count the count of random binomial values frame.
    @param binomial_value the random values.
    @param frequencies the list of random value frequencies.
    """

    # Generate random value 'count' times and update the random binomial value frequencies.
    for _ in range(count):
        frequencies[np.random.binomial(25,0.2)] += 1

        # Set the percentages for the bar tops.
    freq_sum = sum(frequencies)
    topper = [f'{freq:,}\n{freq / freq_sum:.3%}' for freq in frequencies]

    # Display the bar chart for this frame.
    show_bar_chart(f'Random number Frequencies for {freq_sum:,} Binomial Distribution\n(trail= 25, probability of
success=0.2)' +
                   f'(Frame {frame_number + 2})',
                   'Value', 'Frequency',
                   binomial_value, frequencies, topper)


# Read command-line arguments for the number of frames
# and the number of counts per frame.
number_of_frames = int(sys.argv[1])
counts_per_frame = int(sys.argv[2])

# Create the figure for the animation.
sns.set_style('whitegrid')  # white background with gray grid lines
```

```
figure = plt.figure('Animated Binomial Distribution')

values = list(range(0,26))  # random values for display on the x-axis
frequencies = [0] * 26  # 0-25 list of value frequencies

# Configure and start the animation that calls function update_frame.
binomial_animation = animation.FuncAnimation(
    figure, update_frame, repeat=False,
    frames=number_of_frames - 1, interval=25,
    fargs=(counts_per_frame, values, frequencies))

plt.show()
```

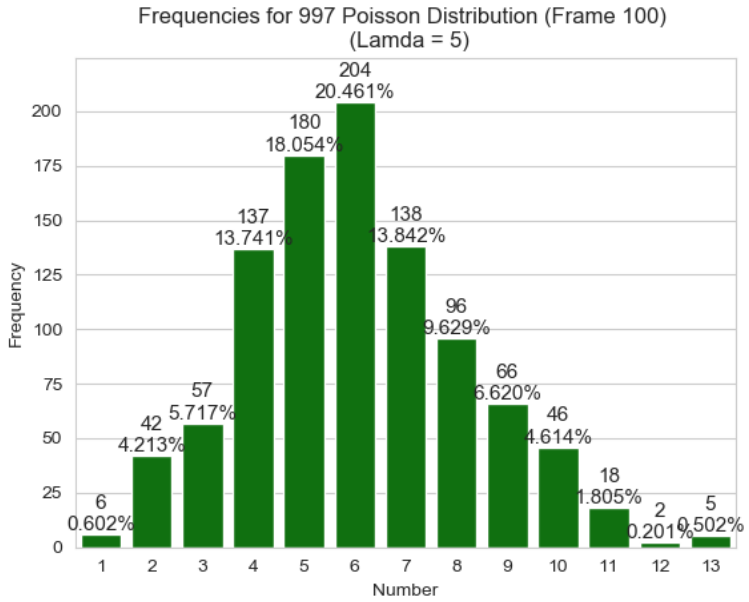


Animated Bar chart for Poisson Distribution:

```
"""
Demonstrate matplotlib animation of random values.

Demonstrate a randomly generated Poisson Distribution. Dynamically
graph frequencies of each number.

"""

from matplotlib import animation
import matplotlib.pyplot as plt
import random
import seaborn as sns
import sys
import numpy as np


def show_bar_chart(title, x_label, y_label, x_values, y_values, bar_toppers):
```

```
    """
    Display a bar chart.
    @param title the chart title.
    @param x_label the label for the x axis
    @param y_label the label for the y axis
    @param x_values the x values to plot
    @param y_values the y values to plot
    @param bar_toppers the text above each bar
    """

    plt.cla()  # clear old contents

    axes = sns.barplot(x_values, y_values, color='green')
    axes.set_title(title)
    axes.set(xlabel=x_label, ylabel=y_label)

    # Scale the y-axis by 10% to make room for text above the bars.
    axes.set_ylim(top=1.10 * max(y_values))

    # Display the topper text above each patch (bar).
    for bar, topper in zip(axes.patches, bar_toppers):
        text_x = bar.get_x() + bar.get_width() / 2
        text_y = bar.get_height()
        axes.text(text_x, text_y, topper,
                  fontsize=11, ha='center', va='bottom')


def update_frame(frame_number, counts, numbers, frequencies):
    """
    Update the bar plot contents for each animation frame.
    @param frame_number the frame number.
    @param counts the numbers per frame.
    @param numbers the generated numbers.
    @param frequencies the list of number occurrence frequencies.
    """

    # generate random numbers and update the frequencies.
    for _ in range(counts):
        try:
            frequencies[np.random.poisson(5)] += 1
        except: # if the number is out the frequency list, it will pass
            pass

    # Set the percentages for the bar tops.
    freq_sum = sum(frequencies)
    topper = [f'{freq:,}\n{freq / freq_sum:.3%}' for freq in frequencies]

    # Display the bar chart for this frame.
    show_bar_chart(f'Frequencies for {freq_sum:,} Poisson Distribution ' +
                   f'(Frame {frame_number + 2}) \n (Lamda = 5)',
                   'Number', 'Frequency',
                   numbers, frequencies, topper)


# Read command-line arguments for the number of frames
```

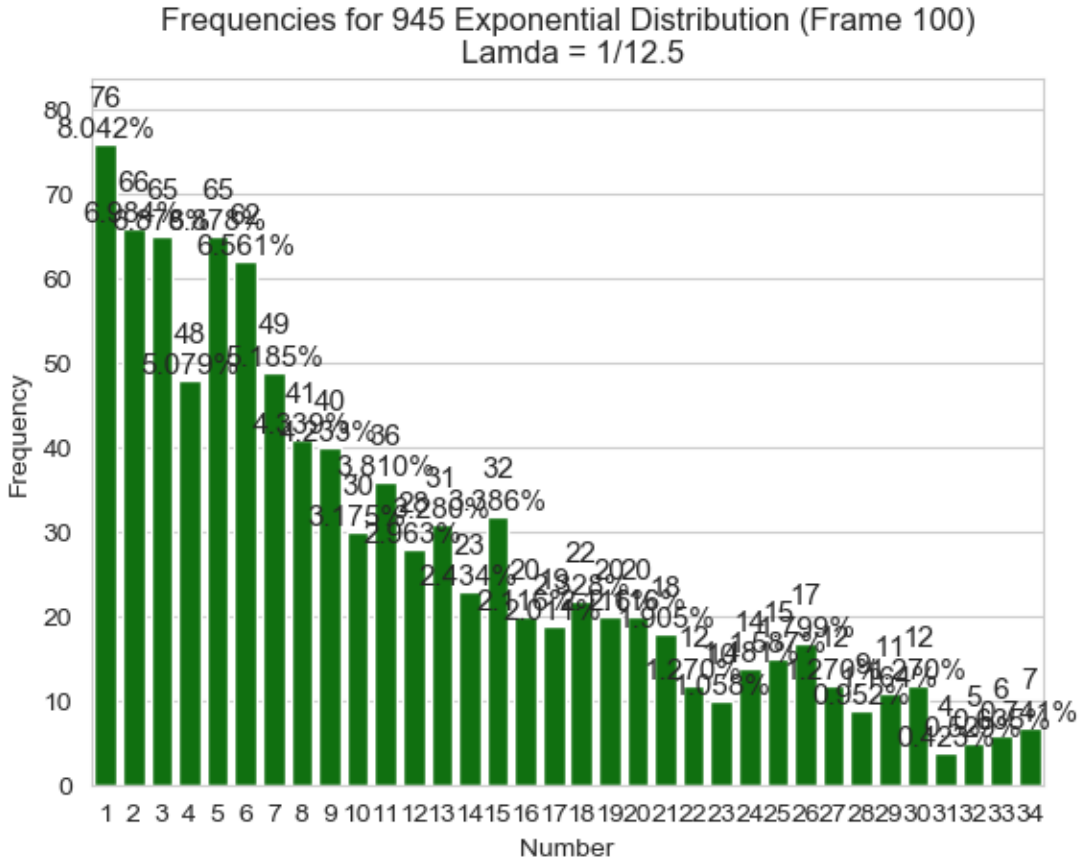
```
# and the number of rolls per frame.
number_of_frames = int(sys.argv[1])
rolls_per_frame = int(sys.argv[2])

# Create the figure for the animation.
sns.set_style('whitegrid')  # white background with gray grid lines
figure = plt.figure('Poisson Distribution')

values = list(range(1, 14))  # numbers which display on the x-axis
frequencies = [0] * 13  # number frequencies which we will count

# Configure and start the animation that calls function update_frame.
die_animation = animation.FuncAnimation(
    figure, update_frame, repeat=False,
    frames=number_of_frames - 1, interval=25,
    fargs=(rolls_per_frame, values, frequencies))

plt.show()

# Adapted from:
# **************************************************************************
# * (C) Copyright 1992-2018 by Deitel & Associates, Inc. and               *
# * Pearson Education, Inc. All Rights Reserved.                           *
# *                                                                        *
# * DISCLAIMER: The authors and publisher of this book have used their     *
# * best efforts in preparing the book. These efforts include the          *
# * development, research, and testing of the theories and programs        *
# * to determine their effectiveness. The authors and publisher make       *
# * no warranty of any kind, expressed or implied, with regard to these    *
# * programs or to the documentation contained in these books. The authors *
# * and publisher shall not be liable in any event for incidental or       *
# * consequential damages in connection with, or arising out of, the       *
# * furnishing, performance, or use of these programs.                     *
# **************************************************************************
```



Animated Bar chart for Exponential Distribution:

```
"""
Demonstrate matplotlib animation of random values.

Demonstrate a randomly generated Exponential Distribution. Dynamically
graph frequencies of each number.

"""

from matplotlib import animation
import matplotlib.pyplot as plt
import random
import seaborn as sns
import sys

def show_bar_chart(title, x_label, y_label, x_values, y_values, bar_toppers):
    """
    Display a bar chart.
    @param title the chart title.
    @param x_label the label for the x axis
    @param y_label the label for the y axis
    @param x_values the x values to plot
    @param y_values the y values to plot
    @param bar_toppers the text above each bar
    """

    plt.cla()  # clear old contents

    axes = sns.barplot(x_values, y_values, color='green')
    axes.set_title(title)
    axes.set(xlabel=x_label, ylabel=y_label)


    # Scale the y-axis by 10% to make room for text above the bars.
    axes.set_ylim(top=1.10*max(y_values))

    # Display the topper text above each patch (bar).
    for bar, topper in zip(axes.patches, bar_toppers):
        text_x = bar.get_x() + bar.get_width()/2
        text_y = bar.get_height()
        axes.text(text_x, text_y, topper,
                  fontsize=11, ha='center', va='bottom')

def update_frame(frame_number, counts, numbers, frequencies):
    """
    Update the bar plot contents for each animation frame.
    @param frame_number the frame number.
    @param counts the numbers per frame.
    @param numbers the generated numbers.
    @param frequencies the list of number occurrence frequencies.
    """
```

```
    # generate random numbers and update the frequencies.
    for _ in range(counts):
        try:
            frequencies[int(random.expovariate(1/12.5))] += 1
        except: pass  # if the number is out the frequency list, it will pass

    # Set the percentages for the bar tops.
    freq_sum = sum(frequencies)
    topper = [f'{freq:,}\n{freq/freq_sum:.3%}' for freq in frequencies]

    # Display the bar chart for this frame.
    show_bar_chart(f'Frequencies for {freq_sum:,} Normal Distribution ' +
                   f'(Frame {frame_number + 2})',
                   'Number', 'Frequency'+'\n Lamda = 1/12.5',
                   numbers, frequencies, topper)

# Read command-line arguments for the number of frames
# and the number of rolls per frame.
number_of_frames = int(sys.argv[1])
rolls_per_frame  = int(sys.argv[2])

# Create the figure for the animation.
sns.set_style('whitegrid')  # white background with gray grid lines
figure = plt.figure('Exponential Distribution')

values = list(range(1,26))  # numbers which display on the x-axis
frequencies = [0]*25  # number frequencies which we will count

# Configure and start the animation that calls function update_frame.
die_animation = animation.FuncAnimation(
    figure, update_frame, repeat=False,
    frames=number_of_frames - 1, interval=25,
    fargs=(rolls_per_frame, values, frequencies))

plt.show()

# Adapted from:

```


Frequencies for 945 Exponential Distribution (Frame 100)
Lamda = 1/12.5
Frequency
Number
0
10
20
30
40
50
60
70
80
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34
76
8.042%
66
48
5.079%
65
6.561%
49
5.185%
41
40
4.233%
36
3.810%
30
3.175%
31
3.280%
32
3.386%
23
2.434%
20
22
20
20
18
1.905%
12
1.270%
14
15
17
1.799%
12
1.270%
11
12
1.270%
7