



Intro to Visualization in Python - Static Plots - 3

One should look for what is and not what he thinks should be. (Albert Einstein)

Module completion checklist

Objective	Complete
Create violin plots	
Create compound visualizations	

What we have learned so far...

- Visualizing data with matplotlib
- Creating histograms, boxplots, and bar charts
- Creating scatterplots
- Customizing graphs for impact

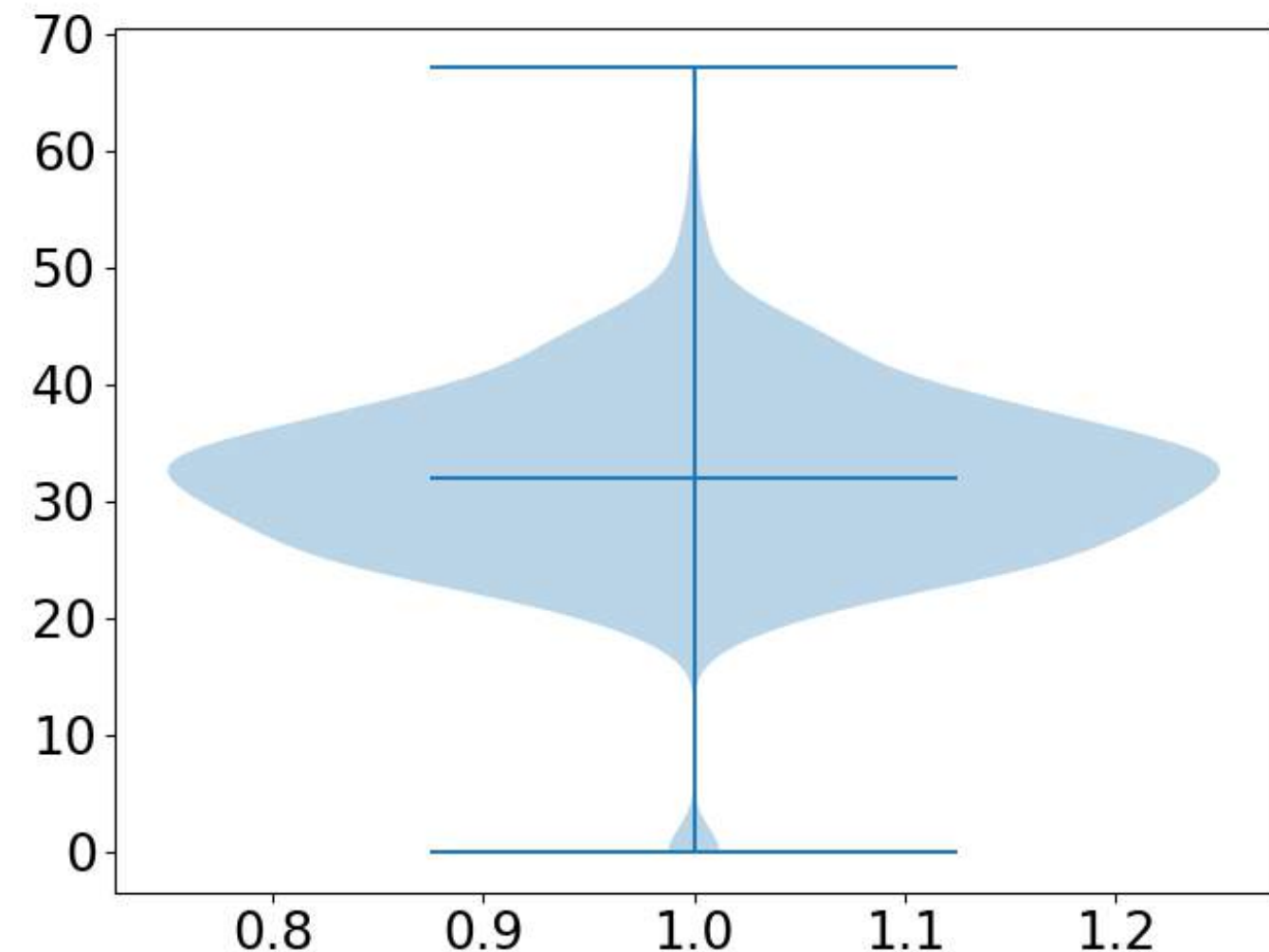
In this module, we'll explore complex visualizations, saving plots in the plot directory, and best data visualization practices

Complex univariate plots: violin plots

- Violin plots are primarily used to look at the **variations in the data**
- The characteristics of violin plots are similar to the box plot, except they **visualize the probability density** of the entire data
- Just like box plots, they include a marker that shows the median
- The violin plot has elongated projections when the density is high and flat projections when the probability density is low
- The attributes `showmeans` and `showmedians` can be set to true or false to show the mean/median and vice versa

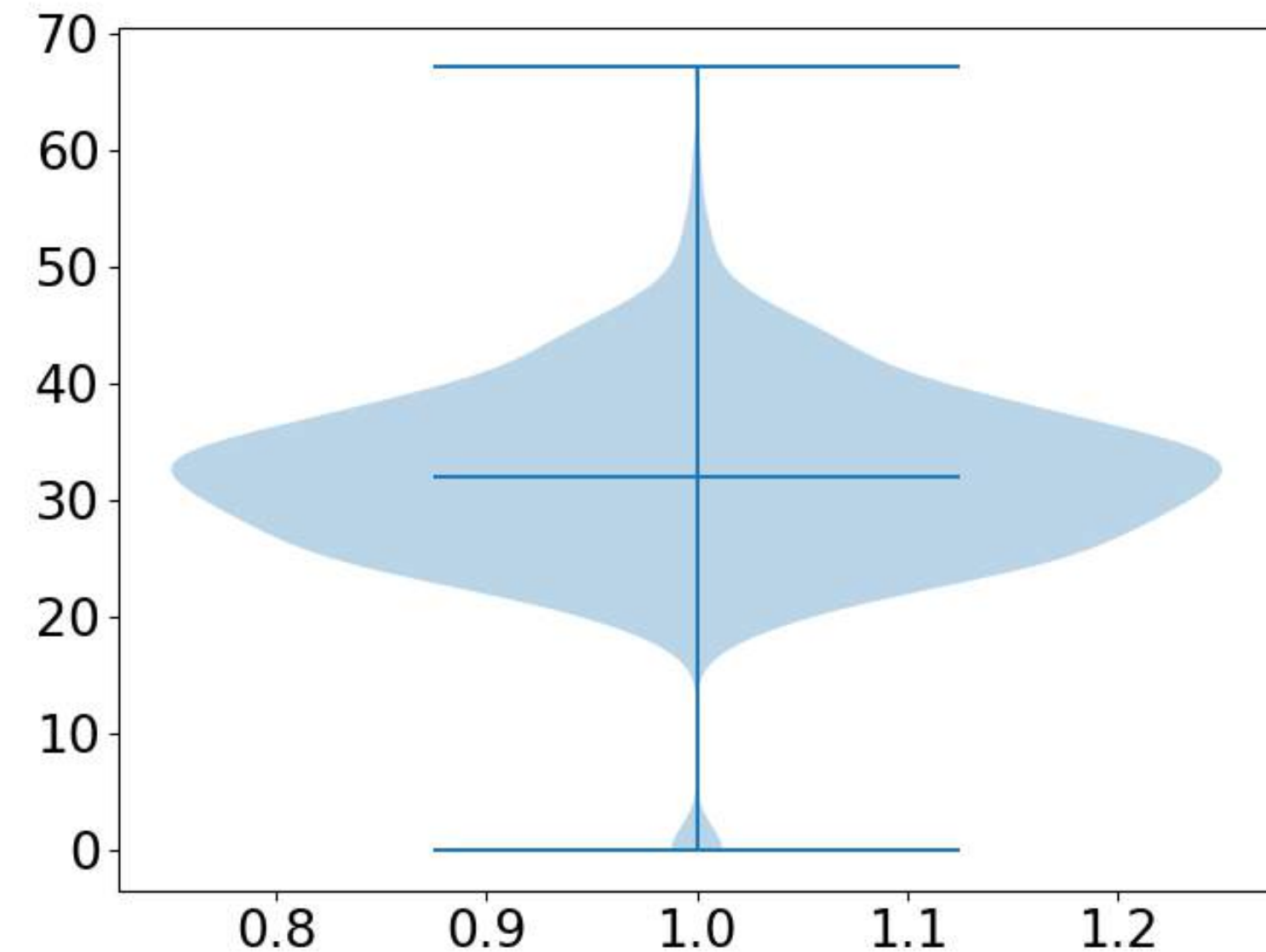
```
plt.violinplot(df_subset['BMI'],  
               showmeans=False,  
               showmedians=True)
```

```
plt.show()
```



Univariate plots: violin plot interpretation

- The blue line in the middle shows the median of 'BMI'
- The **immediate areas around the median of the violin plot where the probability density is higher represent approximately the 25th and 75th percentile**
- By comparing the box plot with the violin plot, we understand that the violin plot is a lot more helpful in understanding the exact probability distribution of data

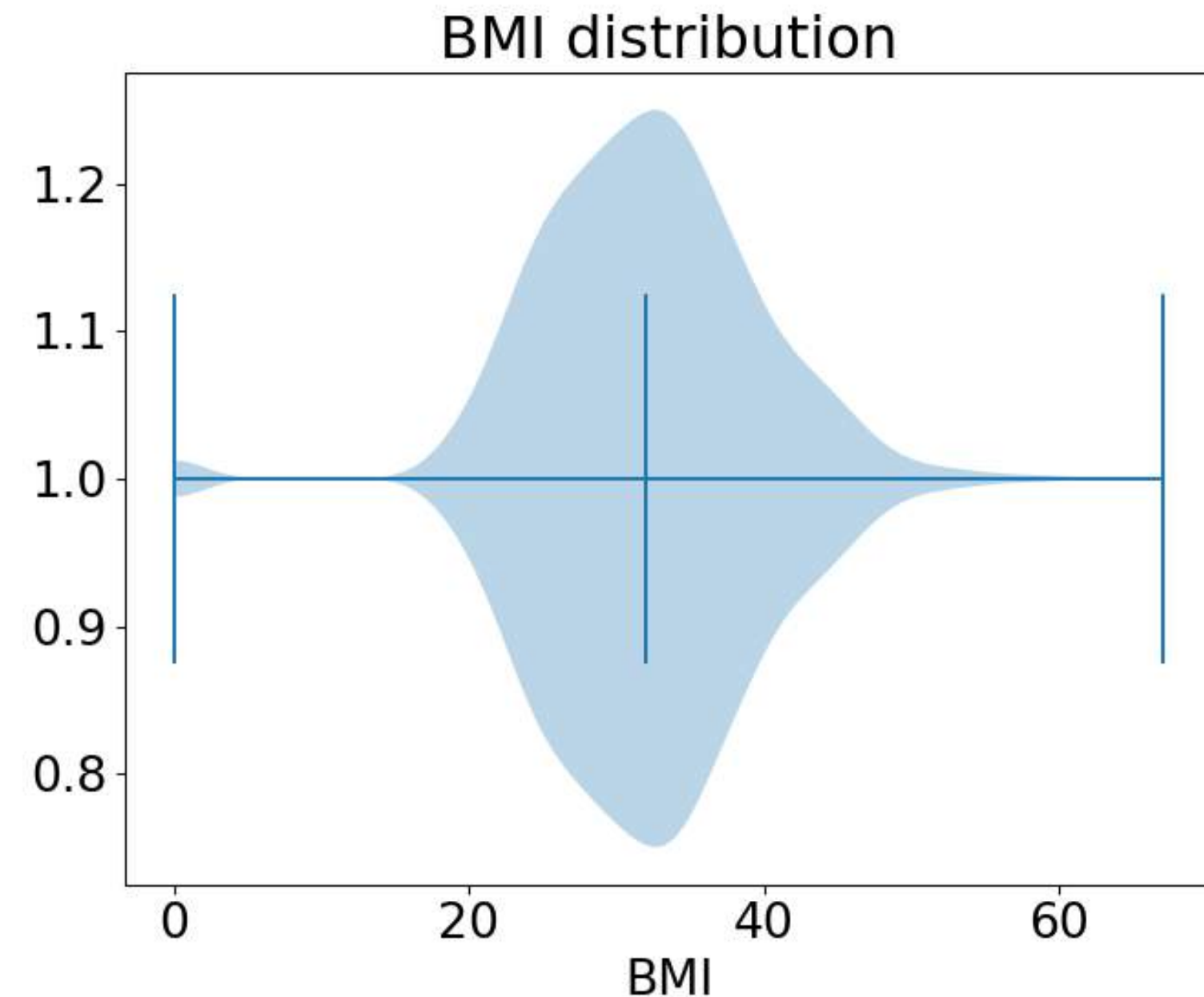


Univariate plots: violin plot (cont'd)

- You can change the orientation of the plot to horizontal by setting `vert = False`
- In chat, share your thoughts on: Looking at this violin plot, what can you tell about the '**BMI**' distribution in our data?

```
plt.violinplot(df_subset['BMI'], vert = False,  
showmeans=False, showmedians=True)
```

```
plt.xlabel('BMI')  
plt.title('BMI distribution')  
plt.show()
```



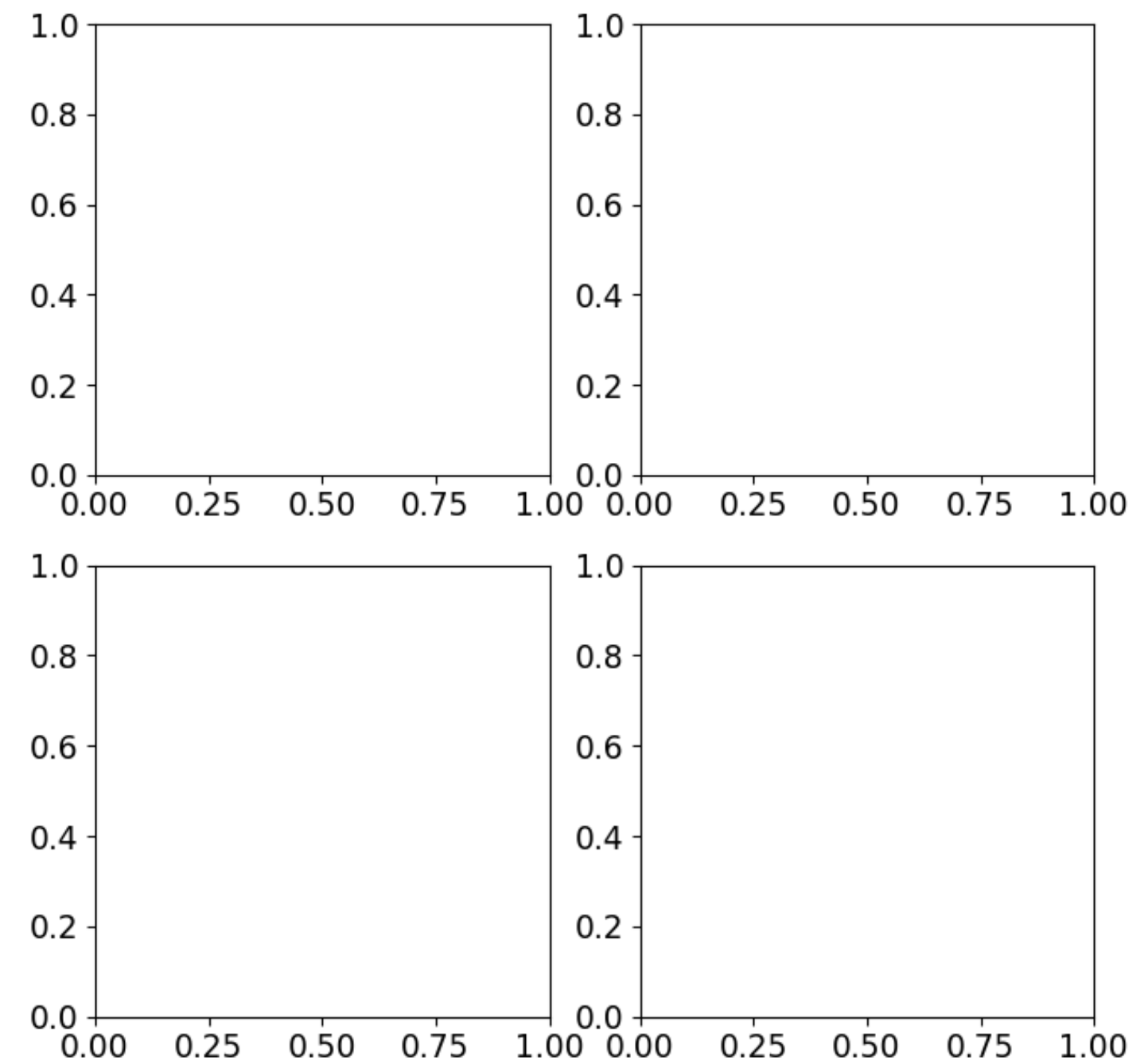
Module completion checklist

Objective	Complete
Create violin plots	✓
Create compound visualizations	

Compound visualizations: grids

- We can create figures containing multiple plots laid out in a **grid** using `plt.subplots()`
- The `subplots` function returns two values, a **Figure** object, and an **Axes** object
 - Figure contains the entire grid and all of the elements inside
 - Axes is an array, where each member contains a particular subplot
- *Why do you think grid or compound visualizations are helpful?*
- *Where would you use such visualizations in your work?*

```
# Create a 2 x 2 figure and axes grid.  
fig, axes = plt.subplots(2, 2)  
plt.show()
```



Compound visualizations: axes

- Axes is just an array

```
print(axes)
```

```
[ [<AxesSubplot:> <AxesSubplot:>]  
  [<AxesSubplot:> <AxesSubplot:>] ]
```

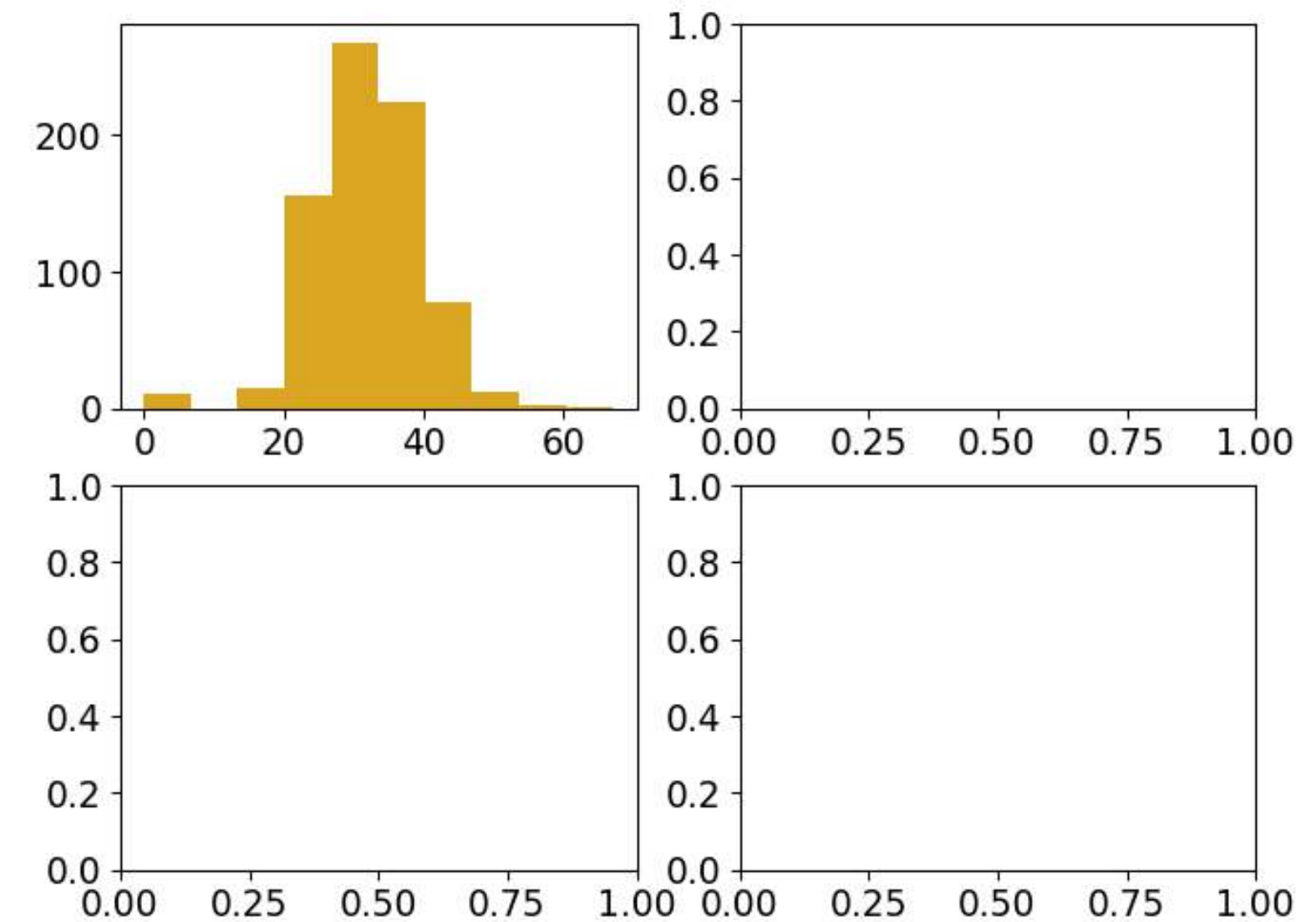
- Since it's a 2×2 grid, we have a 2D array with four entries that we will “fill” with values - that is, **plots**

Compound visualizations: axes (cont'd)

- To access each element of the array, use a simple 2D array subsetting style
`[row_id, col_id]`
- Instead of attaching a particular plot like a histogram, for example, to a `plt` object, we will attach it to the
`axes[row_id, col_id]`

```
plt.clf()
plt.figure(figsize = (8, 8))
plt.rcParams.update({'font.size': 14})
fig, axes = plt.subplots(2, 2)

axes[0, 0].hist(df_subset['BMI'],
                facecolor = 'goldenrod') #<- set
color
```



Compound visualizations: axes (cont'd)

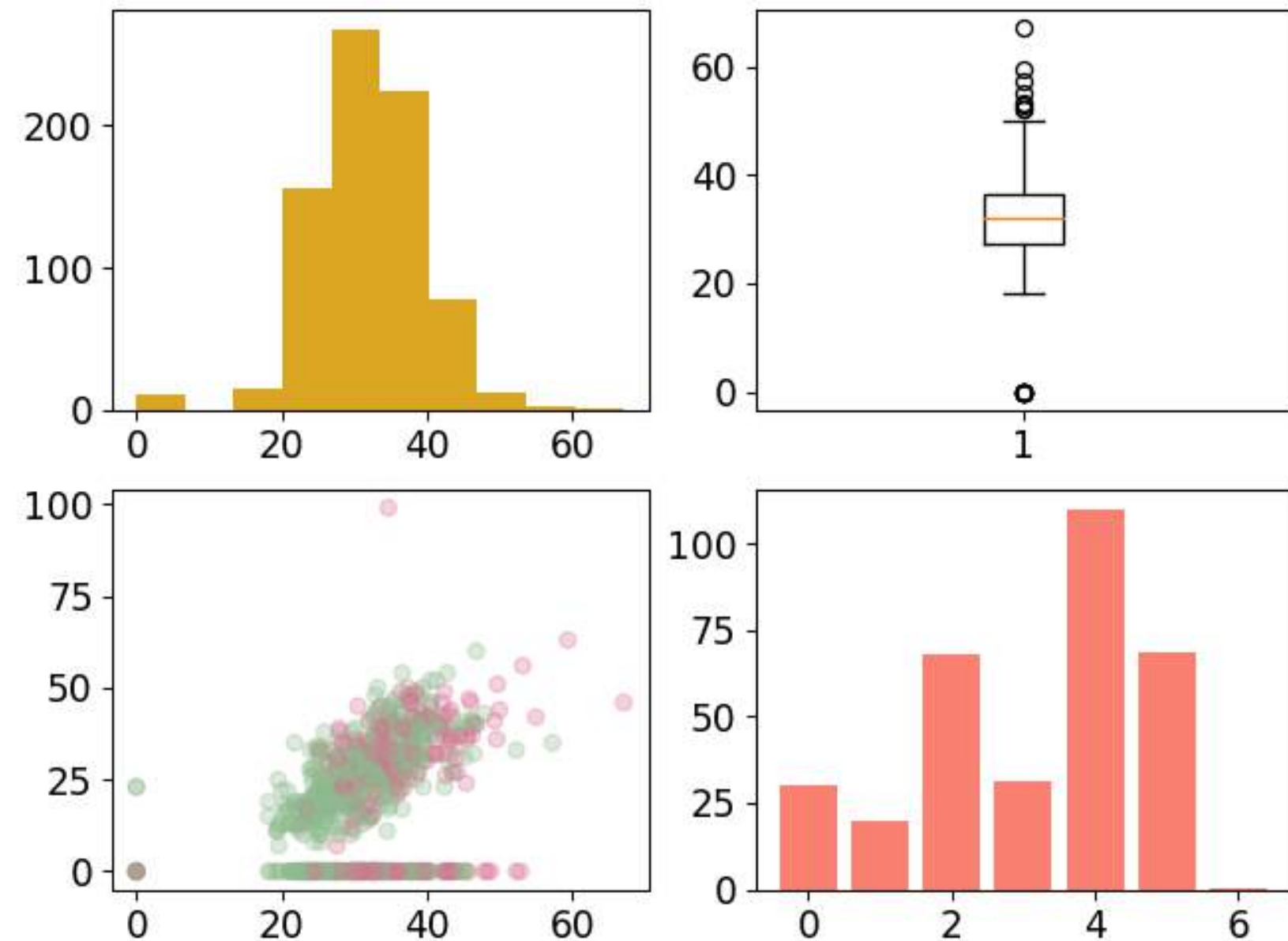
- Let's fill out three remaining plots

```
plt.figure(figsize = (12, 8))
fig, axes = plt.subplots(2, 2)
color_dict = {int('0'): 'darkseagreen',
              int('1'): 'palevioletred'}
color = df_subset['Outcome'].map(color_dict)
axes[0, 0].hist(df_subset['BMI'],
               facecolor = 'goldenrod') #<-
set color
```

```
axes[0, 1].boxplot(df_subset['BMI'])
```

```
axes[1, 0].scatter(df_subset['BMI'],
                  df_subset['SkinThickness'],
                  c = color,
                  alpha = 0.3)
axes[1, 1].bar(bar_positions, bar_heights,
               color = "salmon")
```

```
plt.show()
```



Compound visualizations: labeling axes

- To label each plot's axis, use `axes[row_id, col_id].set_xlabel` format

```
# Histogram.
axes[0, 0].set_ylabel('BMI distribution')
axes[0, 0].set_xlabel('BMI')

# Boxplot.
axes[0, 1].set_ylabel('BMI')

# Scatterplot.
axes[1, 0].set_xlabel('BMI')
axes[1, 0].set_ylabel('SkinThickness')

# Mean values of categories of variable means.
axes[1, 1].set_ylabel('Mean values')
```

Compound visualizations: labeling ticks

- To set ticks on each axis, use `axes[row_id, col_id].xaxis.set_ticks` format

```
# No labels for ticks for boxplot.  
axes[0, 1].xaxis.set_ticklabels([""])
```

```
# Tick positions set to bar positions in bar chart.  
axes[1, 1].xaxis.set_ticks(bar_positions)  
  
# Tick labels set to bar categories in bar chart.  
axes[1, 1].xaxis.set_ticklabels(bar_labels, rotation = 18)
```

Compound visualizations: figure adjustments

- Now let's make a few final adjustments to how our figure outputs

```
plt.rcParams['axes.labelsize'] = 20
plt.rcParams['figure.titlesize'] = 25
fig.set_size_inches(18, 7.5)
fig.suptitle('Data Summary')
```

Compound visualizations: putting it all together

- **Note:** The entire code block will be visible in your notebook

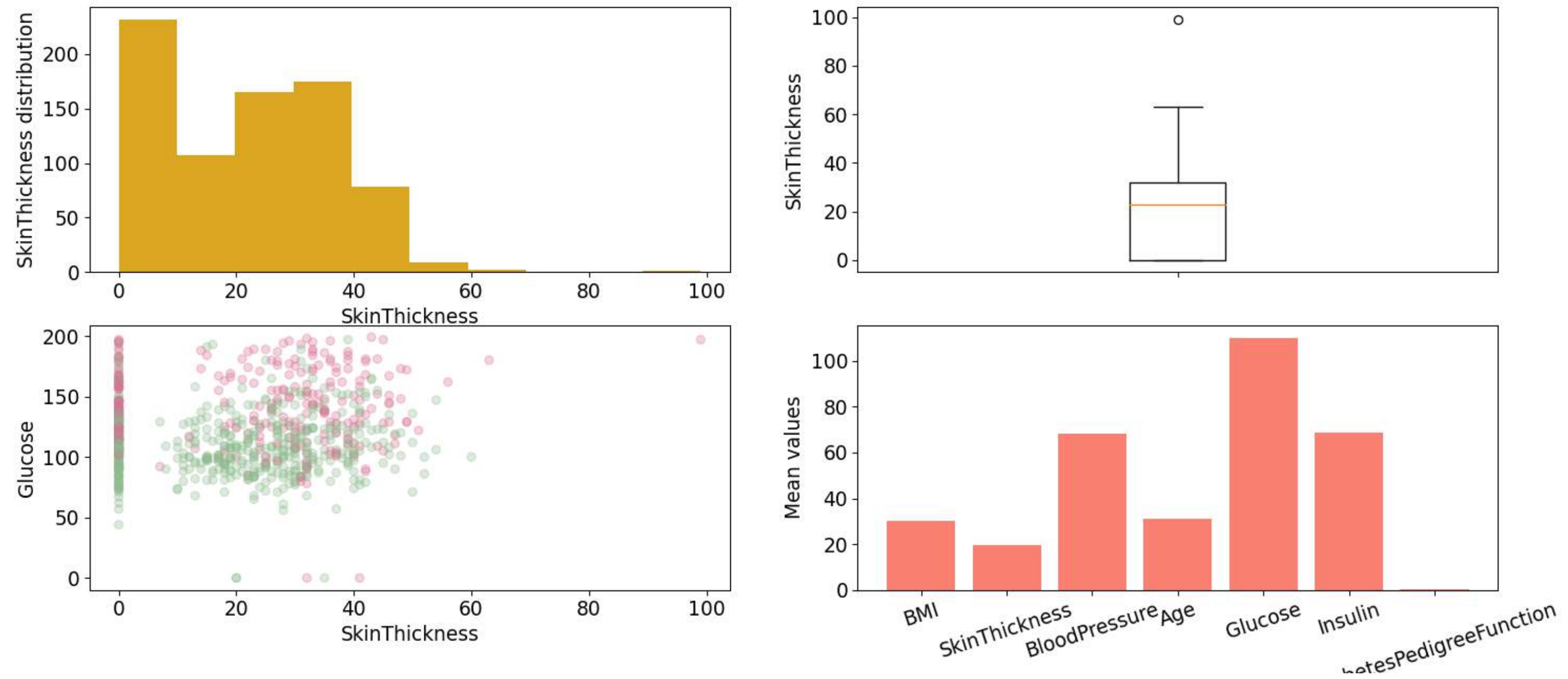
```
plt.clf()
plt.figure(figsize = (8, 8))
plt.rcParams.update({'font.size': 14})
fig, axes = plt.subplots(2, 2)

color_dict = {int('0'): 'darkseagreen',
              int('1'): 'palevioletred'}
color = df_subset['Outcome'].map(color_dict)
axes[0, 0].hist(df_subset['SkinThickness'],
               facecolor = 'goldenrod') #<- set color
axes[0, 1].boxplot(df_subset['SkinThickness'])
axes[1, 0].scatter(df_subset['SkinThickness'],
                  df_subset['Glucose'],
                  c = color,
                  alpha = 0.3)
axes[1, 1].bar(bar_positions, bar_heights,
               color = "salmon")

# Histogram
```

Compound visualizations: display the figure

Data Summary



Compound visualizations: layered plots

- We can create figures containing multiple plots layered on top of each other using the same plotting area `plt.subplots()`
- Layered plots allow any number of plotting layers, making them very flexible - especially in those datasets where looking at patterns across multiple categories is essential

Compound visualizations: layered plots (cont'd)

- We'll now create a layered plot based on the scatterplot we created earlier
- **Note:** The entire code block will be visible in your notebook

```
plt.clf() #<- clear plotting area
fig, axes = plt.subplots() #<- create a new figure and axes objects for plotting

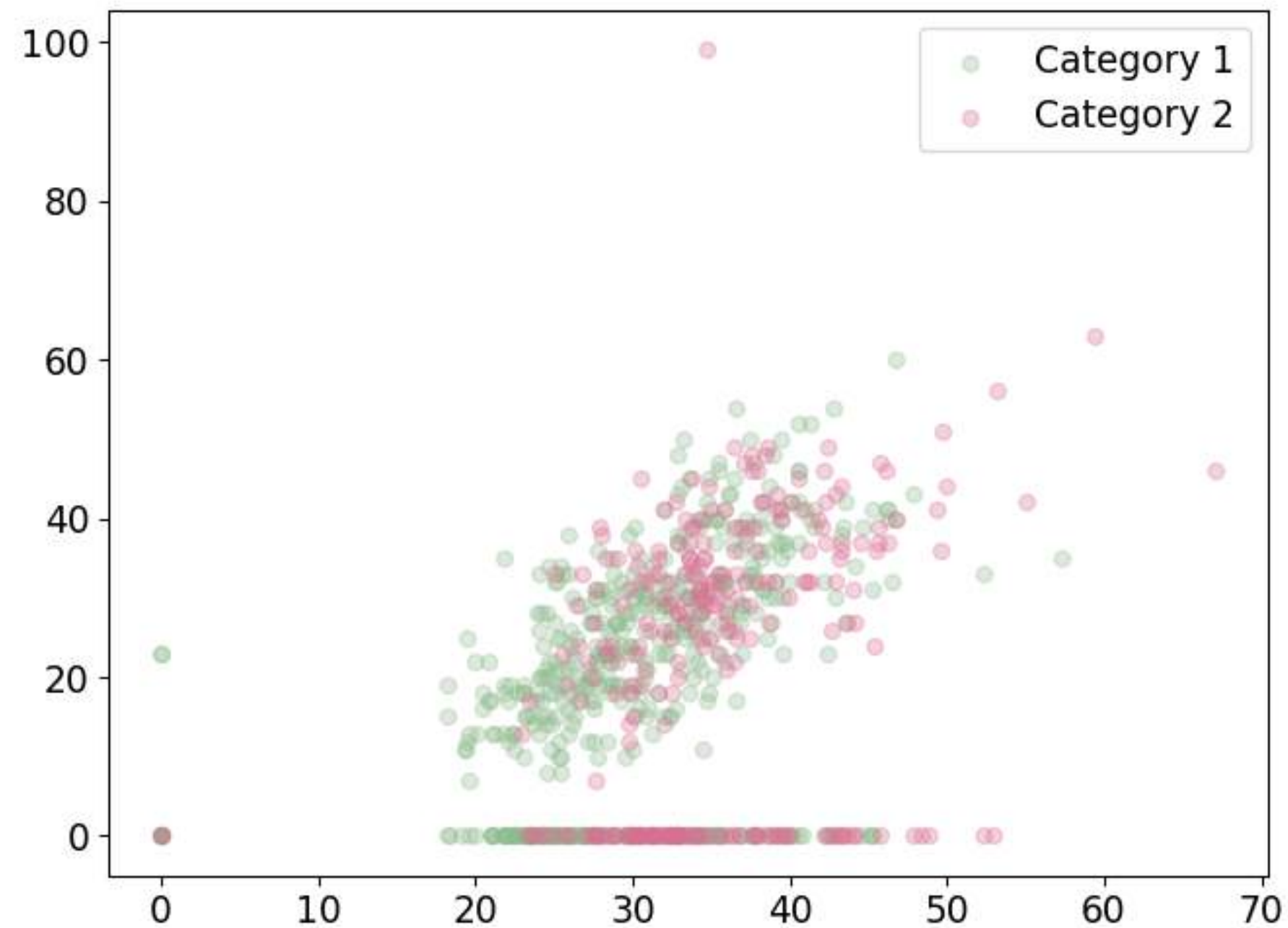
grouping_col_levels = list(df_grouped_mean_long[grouping_col].unique())
grouping_category_1 = grouping_col_levels[0]
grouping_category_2 = grouping_col_levels[1]

for key, value in color_dict.items():
    query = str('Outcome') + '==' + str(key)
    sc_col_1 = df_subset.query(query) ['BMI']
    sc_col_2 = df_subset.query(query) ['SkinThickness']

    if key == int(grouping_category_1):
        Flag = "Category 1"
    else:
        Flag = "Category 2"

    axes.scatter(sc_col_1,
                 sc_col_2,
```

Compound visualizations: layered plots (cont'd)



Compound visualizations: layered plots (cont'd)

- Now let's create a layered bar chart to visualize the mean values for each of our variables, based on both the `True` and `False` mean data

```
# We already have `Outcome` = `0` mean data.  
print(df_true_means.head())
```

	metric	mean
0	BMI	30.3042
2	SkinThickness	19.6640
4	BloodPressure	68.1840
6	Age	31.1900
8	Glucose	109.9800

```
# Let's get the `Outcome` = `1` mean data.  
query = str('Outcome') + '==' + str('1')  
df_false_means = df_grouped_mean_long.query(query)[['metric', 'mean']]  
print(df_false_means)
```

	metric	mean
1	BMI	35.142537
3	SkinThickness	22.164179
5	BloodPressure	70.824627
7	Age	37.067164
9	Glucose	141.257463
11	Insulin	100.335821
13	DiabetesPedigreeFunction	0.550500

Compound visualizations: layered plots (cont'd)

```
# Mean values for `Outcome` = `0` data.  
category_1_bar_heights = df_true_means['mean']  
# Mean values for `Outcome` = `1` data.  
category_2_bar_heights = df_false_means['mean']  
# Labels of bars, their width, and positions are shared for both categories.  
bar_labels = df_false_means['metric']  
num_bars = len(bar_labels)  
bar_positions = np.arange(num_bars)  
width = 0.35
```

Compound visualizations: layered plots (cont'd)

```
# Clear the plotting area for the new plot.
plt.clf()
# Create the figure and axes objects.
fig, axes = plt.subplots()
```

```
category_1_bar_chart = axes.bar(bar_positions,          #<- set bar positions
                                category_1_bar_heights, #<- set bar heights
                                width,                  #<- set width of the bars
                                color = color_dict[0])  #<- set color corresponding to '0' in dictionary
```

```
category_2_bar_chart = axes.bar(bar_positions + width, #<- set bar positions
                                category_2_bar_heights, #<- set bar heights
                                width,                  #<- set width of the bars
                                color = color_dict[1])  #<- set color corresponding to '1' in dictionary
```

Compound visualizations: layered plots (cont'd)

```
# Add text for labels, title and axes ticks.  
axes.set_ylabel('Mean values')  
axes.set_title('Data metrics summary')  
axes.set_xticks(bar_positions + width/2)
```

```
axes.set_xticklabels(bar_labels)
```

Compound visualizations: layered plots (cont'd)

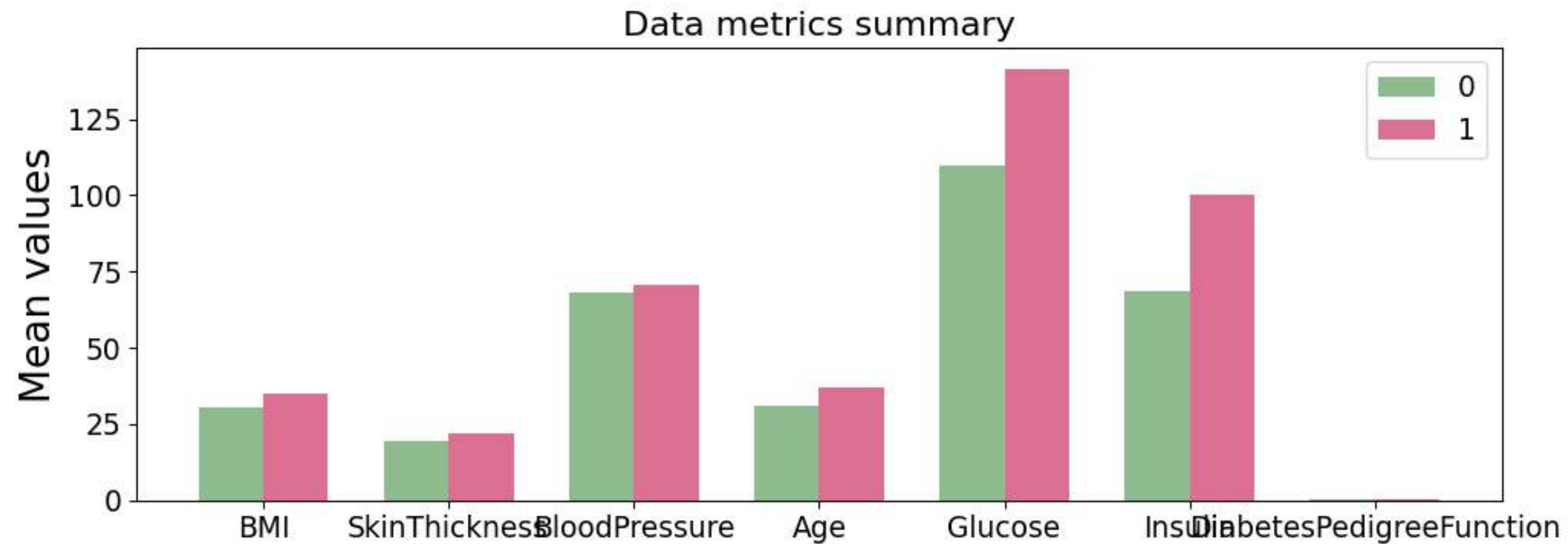
- **Note:** The entire code block will be visible in your notebook

```
# Clear the plotting area for the new plot.
plt.clf()
# Create the figure and axes objects.
fig, axes = plt.subplots()

category_1_bar_chart = axes.bar(bar_positions,          #<- set bar positions
                                category_1_bar_heights, #<- set bar heights
                                width,                  #<- set width of the bars
                                color = color_dict[0])  #<- set color corresponding to '0' in dictionary
category_2_bar_chart = axes.bar(bar_positions + width, #<- set bar positions
                                category_2_bar_heights, #<- set bar heights
                                width,                  #<- set width of the bars
                                color = color_dict[1])  #<- set color corresponding to '1' in dictionary

# Add text for labels, title and axes ticks.
axes.set_ylabel('Mean values')
axes.set_title('Data metrics summary')
axes.set_xticks(bar_positions + width/2)
axes.set_xticklabels(bar_labels)
```


Compound visualizations: layered plots (cont'd)



Module completion checklist

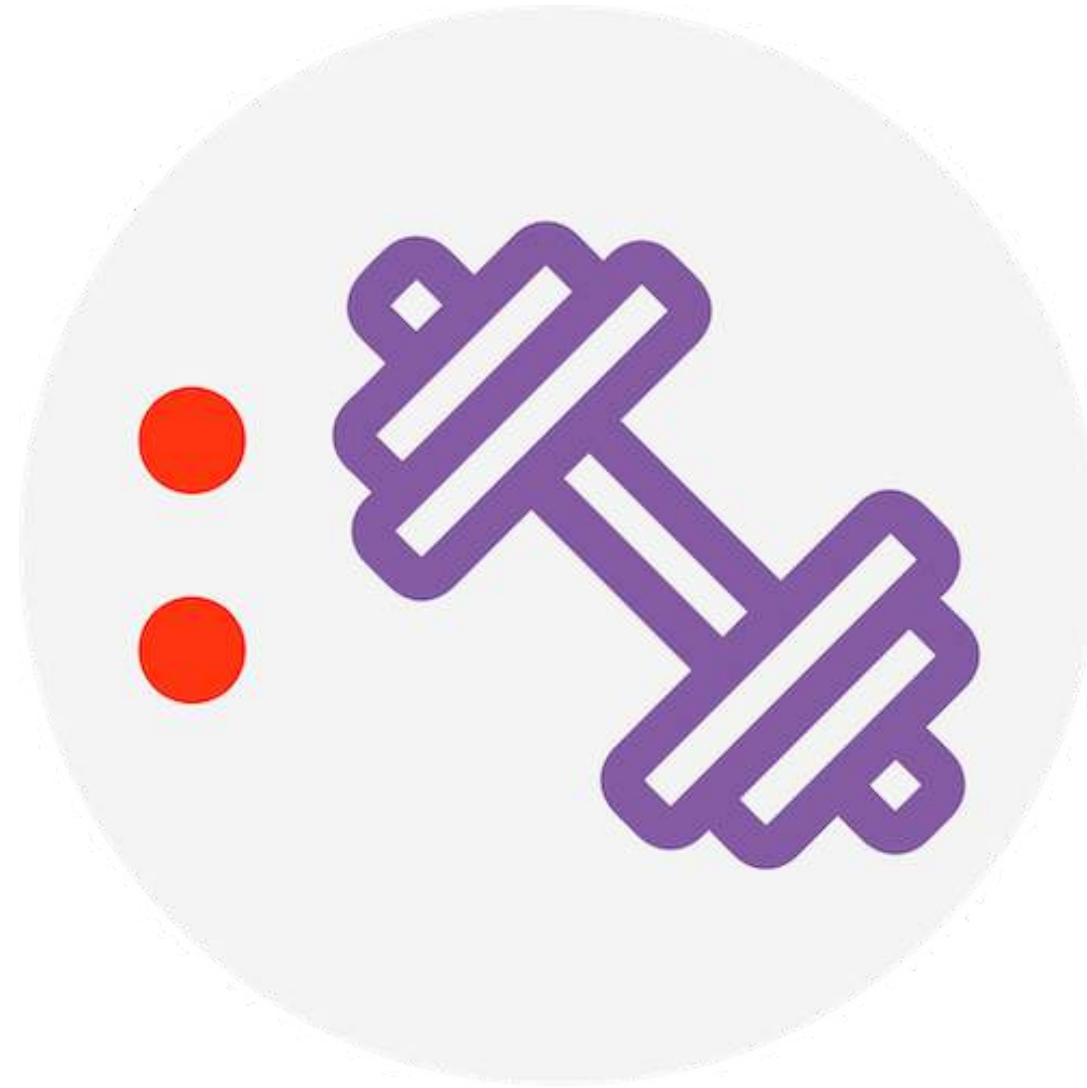
Objective	Complete
Create violin plots	✓
Create compound visualizations	✓

Knowledge check



Link: [*Click here to complete the knowledge check*](#)

Exercise



You are now ready to try tasks 19-24 in the Exercise for this topic.

Module completion checklist

Objective	Complete
Create violin plots	✓
Create compound visualizations	✓

Congratulations on completing this module!

