DATA SOCIETY:

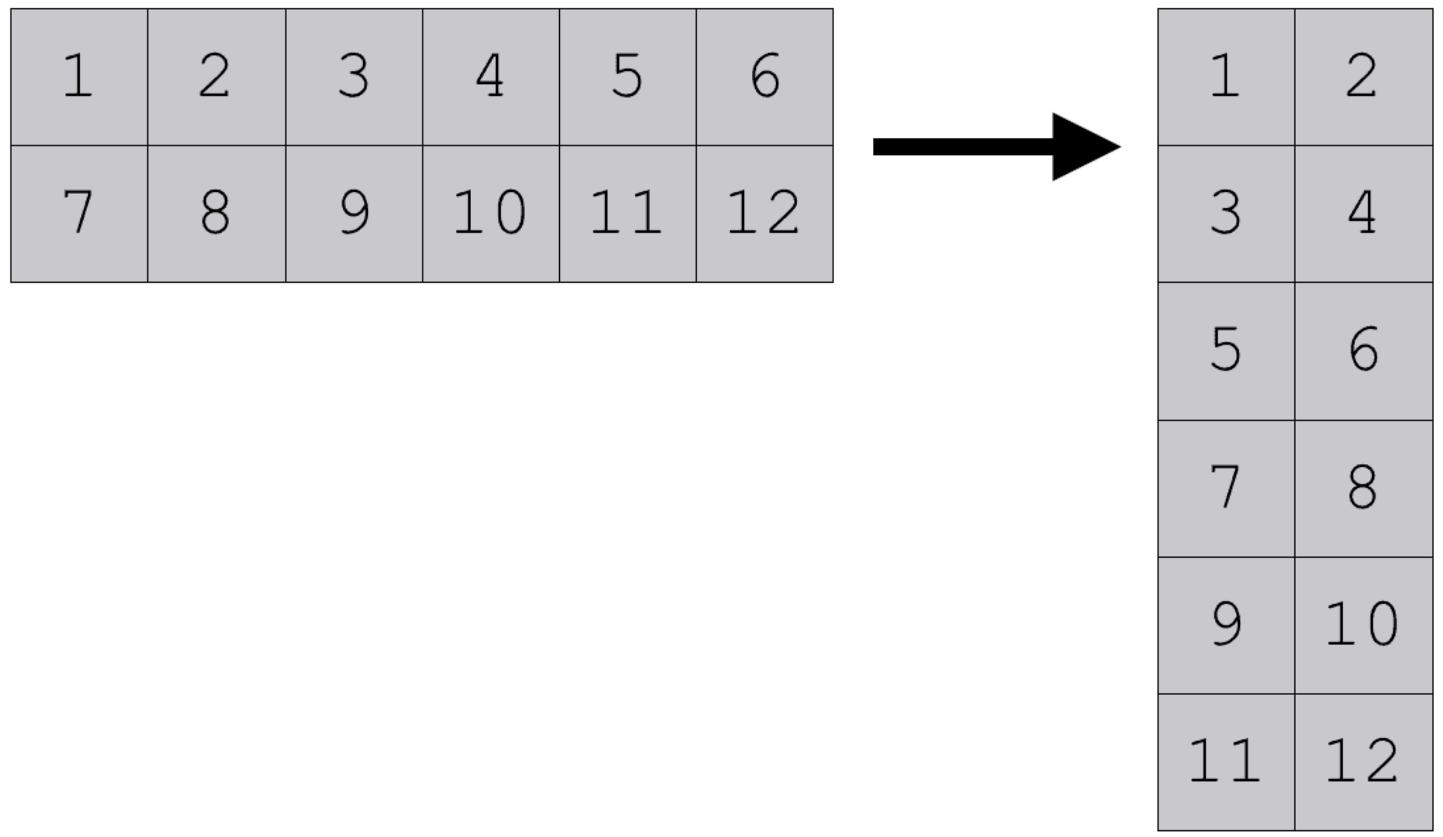
Intro to Visualization in Python - Static Plots - 2

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

Recap Quiz: True or False

- In chat, answer whether the following statement is true or false:

- **Data reshaping involves two types of data: large data and big data**



Recap Quiz: True or False explanation

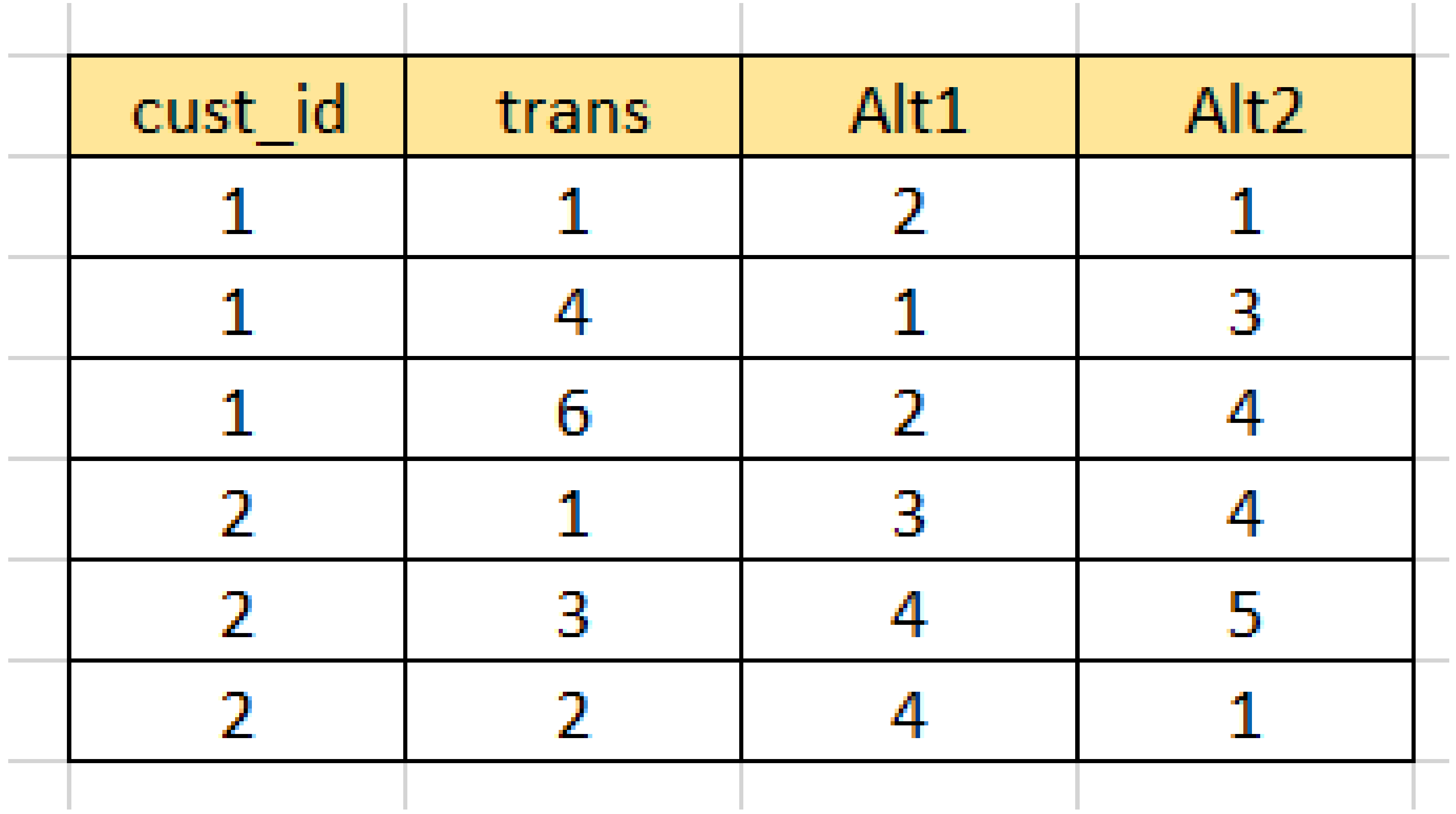
- False

- Data reshaping involves two types of data: **Wide data and Long data**

Recap Quiz: True or False

- In chat, answer whether the following statement is true or false:

- **A dataframe is referred to as wide because each variable has its own column**



cust_id	trans	Alt1	Alt2
1	1	2	1
1	4	1	3
1	6	2	4
2	1	3	4
2	3	4	5
2	2	4	1

Recap Quiz: True or False explanation

- True

- A dataframe is referred to as wide because each variable has **its own column**

Recap Quiz: True or False

- In chat, answer whether the following statement is true or false:

- **We can convert long data to wide format with the `melt` function, and convert the wide data to a long format with `.pivot()` method**

Wide Format

Team	Points	Assists	Rebounds
A	88	12	22
B	91	17	28
C	99	24	30
D	94	28	31

Long Format

Team	Variable	Value
A	Points	88
A	Assists	12
A	Rebounds	22
B	Points	91
B	Assists	17
B	Rebounds	28
C	Points	99
C	Assists	24
C	Rebounds	30
D	Points	94
D	Assists	28
D	Rebounds	31

Recap Quiz: True or False explanation

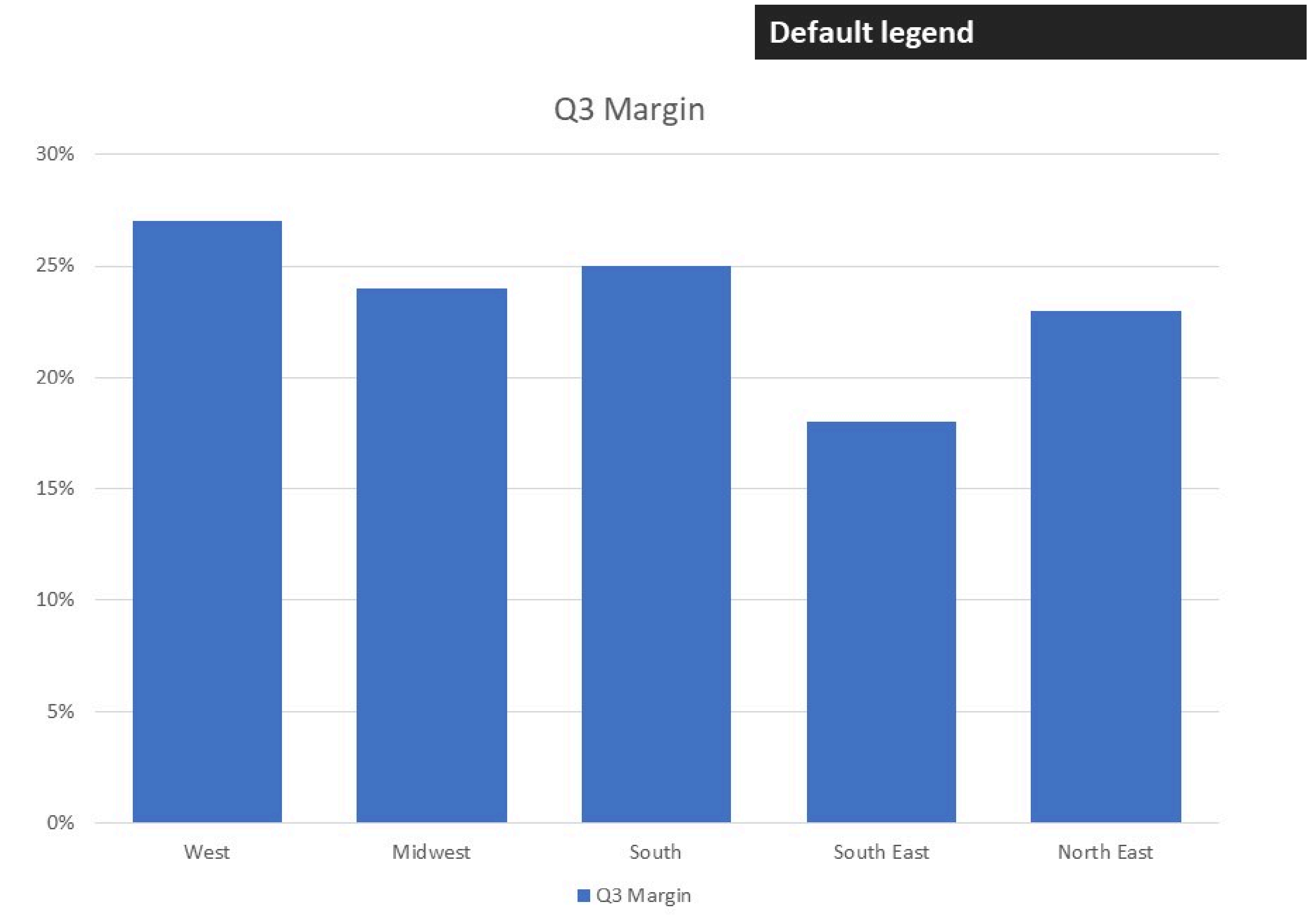
- False

- We can convert **wide data to long format with the `melt` function**, and convert the **long data to a wide format with `.pivot()`method**

Recap Quiz: True or False

- In chat, answer whether the following statement is true or false:

- **For plotting bar charts of any complexity, it is better to use wide data**



Recap Quiz: True or False explanation

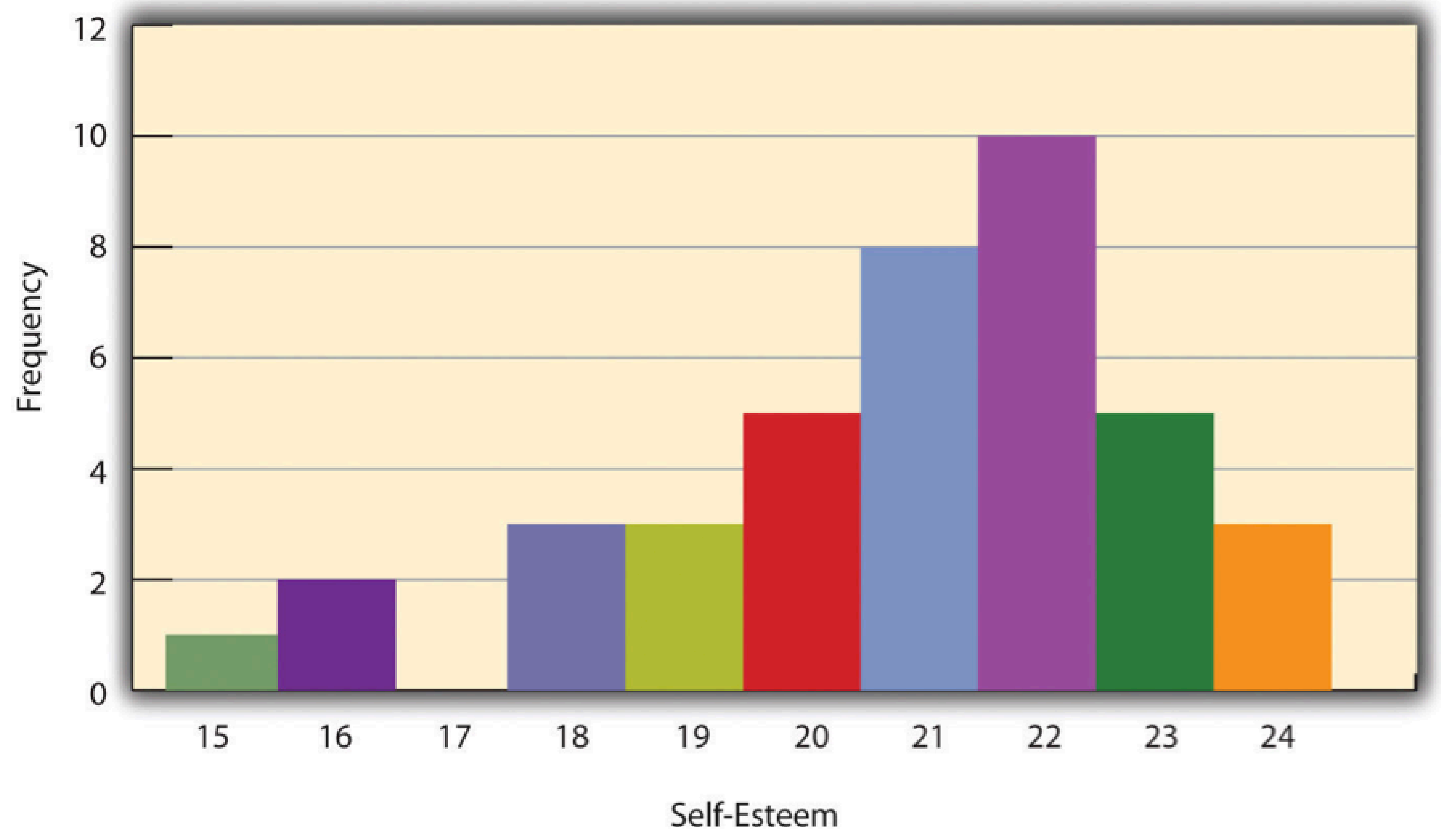
- False

- When plotting bar charts of any complexity, the best type of data to use is long data

Recap Quiz: True or False

- In chat, answer whether the following statement is true or false:

- **Univariate plots are used to visualize distribution of two variables**



Recap Quiz: True or False explanation

- False

- Univariate plots are used to visualize distribution of a **single variables**

Module completion checklist

Objective	Complete
Define bivariate plots and create scatterplots	
Construct customized graphs	

Bivariate plots

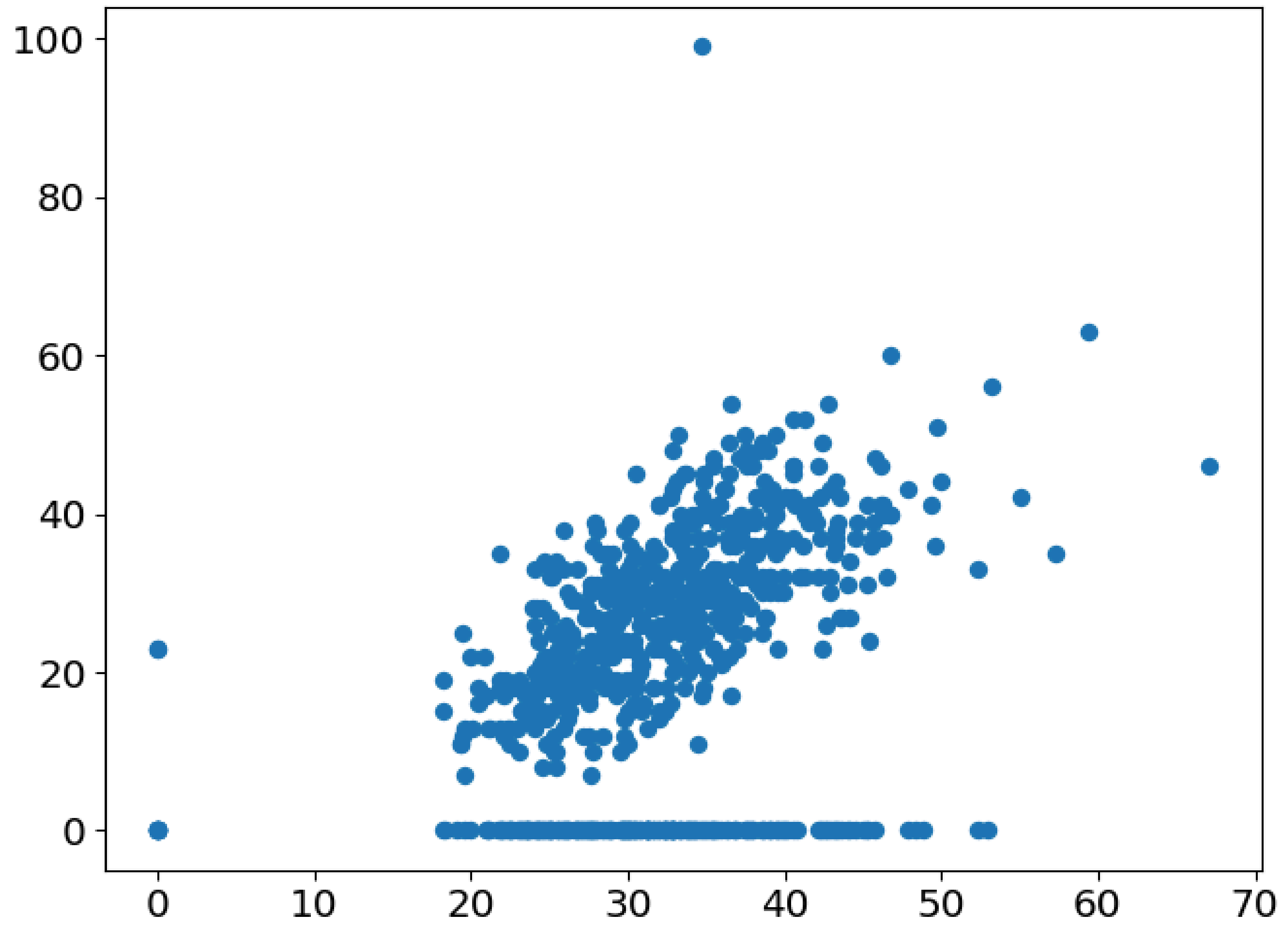
- **Bivariate plots** are used to visualize data **distribution and relationships between two variables**
- They are used to a great extent throughout different stages of EDA **to learn more about how one variable relates to another**
- They are also used **in combination with other bivariate plots to compare relationships between different pairs of variables**
- Bivariate plots include `scatterplots` and `line graphs`

Bivariate plots: scatterplot

- Scatterplot is the most **common bivariate plot** type
- It is one of the most popular plots in scientific computing, machine learning, and data analysis
- It is great for showing **patterns between 2 variables** (hence *bivariate*)
- Plot `'BMI'` against `'SkinThickness'` for each observation
- Takes an array of x values and an array of y values

```
plt.scatter(df_subset['BMI'],
            df_subset['SkinThickness'])
```

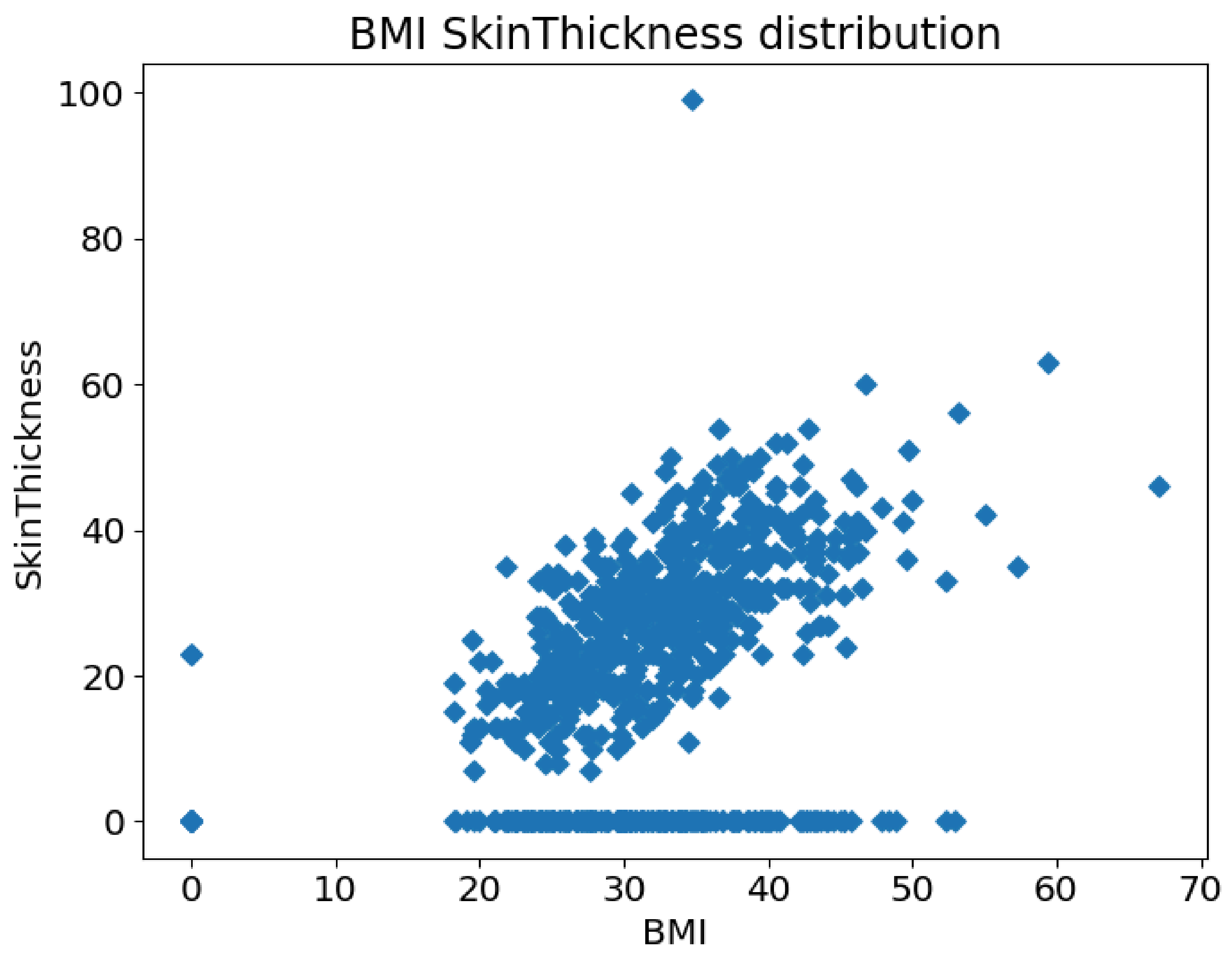
```
plt.show()
```



Bivariate plots: scatterplot (cont'd)

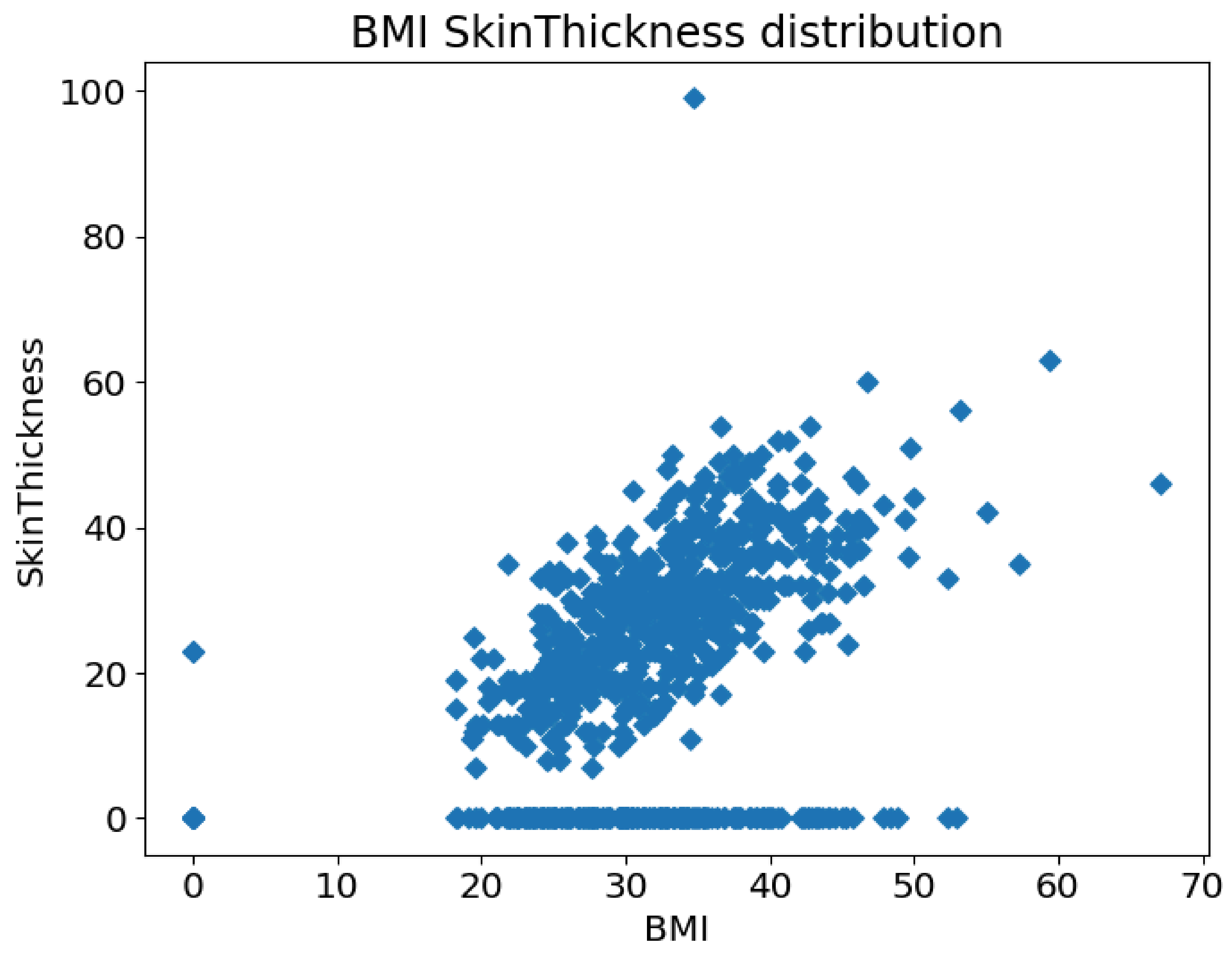
- The marker type can be changed to a shape other than a point
- For a list of marker and line types, refer to ***the matplotlib documentation (link)***

```
plt.scatter(df_subset['BMI'],
            df_subset['SkinThickness'],
            marker = "D") #<- set marker type to
diamond
plt.xlabel('BMI')
plt.ylabel('SkinThickness')
plt.title('BMI SkinThickness distribution')
plt.show()
```



Chat question

- In this scatterplot, what **patterns** do you see in the relationship between the two variables?

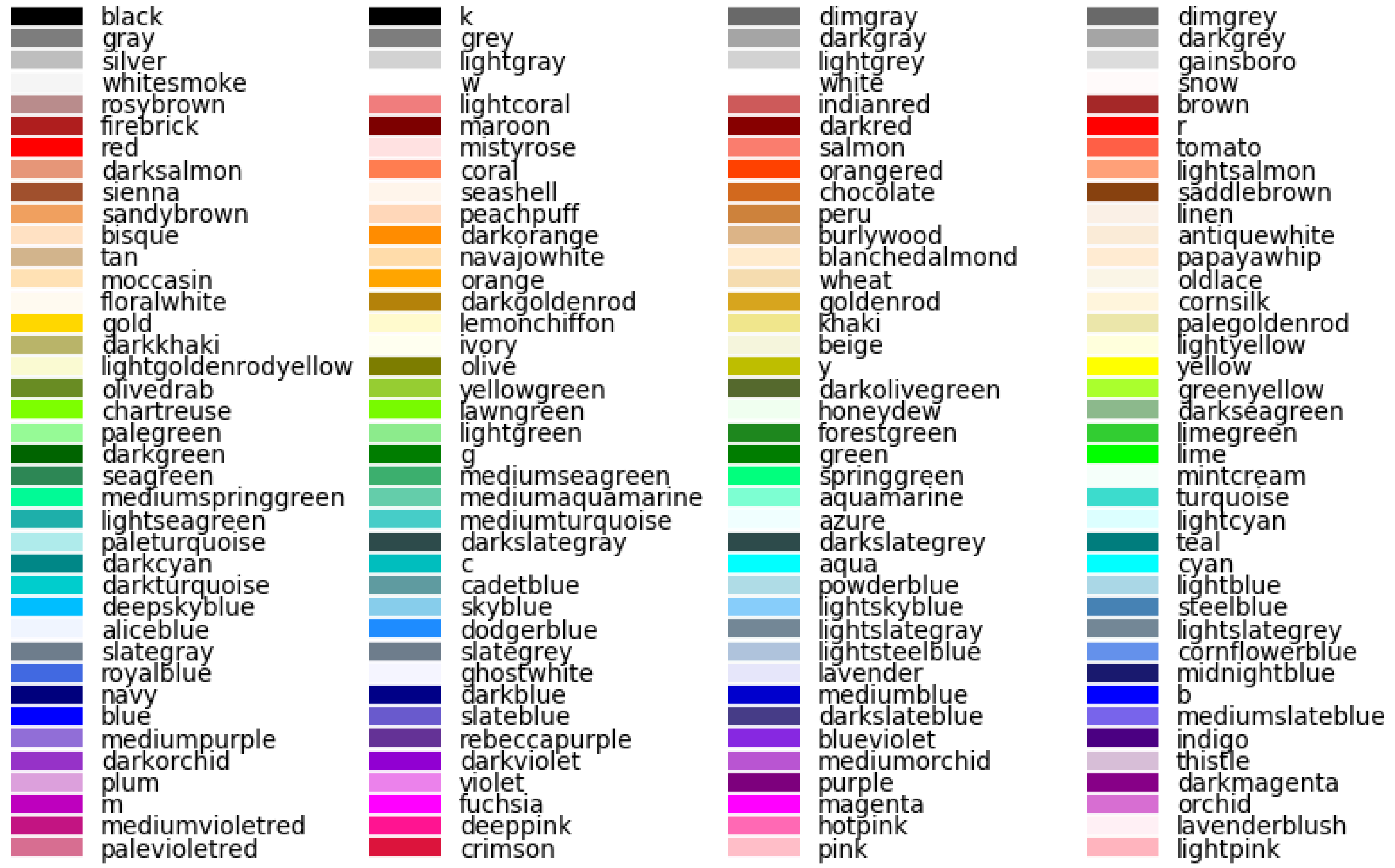


Module completion checklist

Objective	Complete
Define bivaiate plots and create scatterplots	✔
Construct customized graphs	

Customize colors

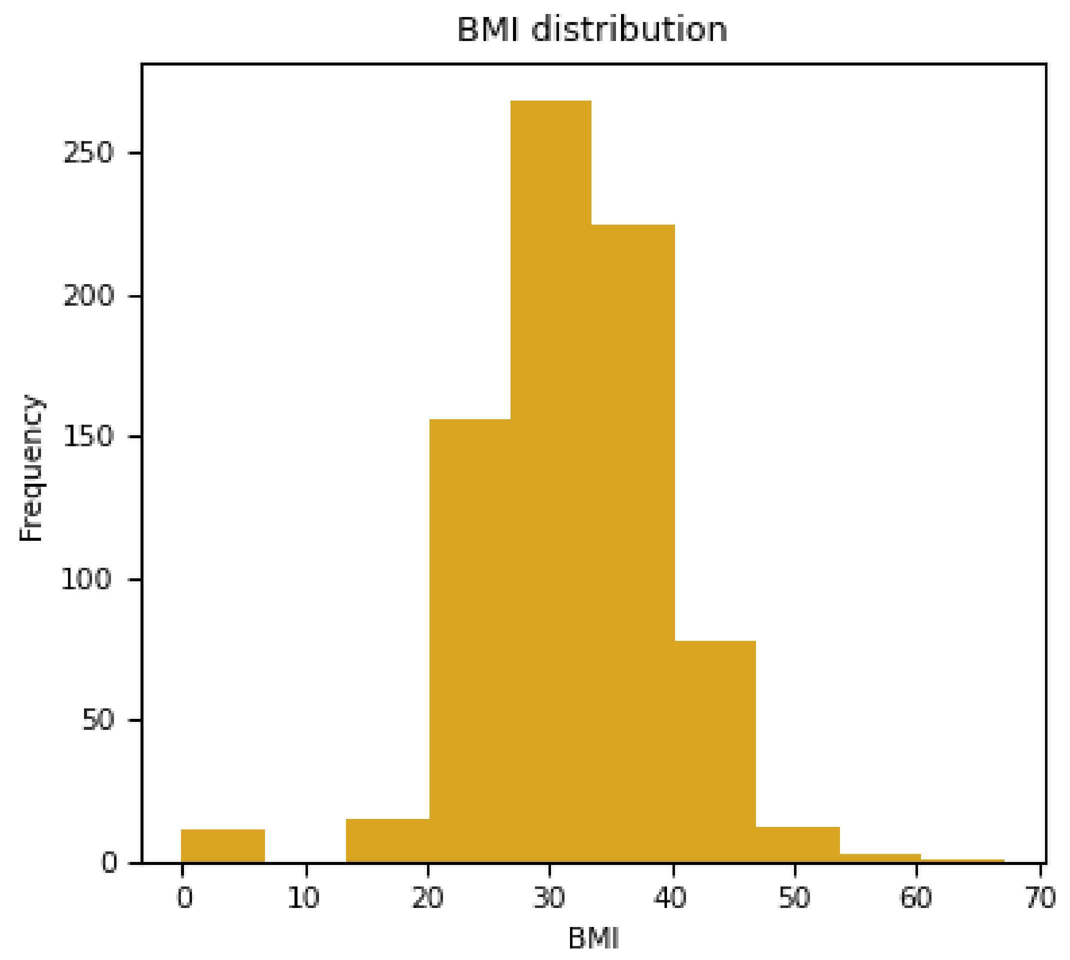
- Change the color of the marker by setting an argument specific to the visualization type
- The basic options are `b` (blue), `g` (green), `r` (red), `c` (cyan), `m` (magenta), `y` (yellow), `k` (black), and `w` (white)
- Use any color by providing its ***RGB code (link)***
- The list of named colors in `matplotlib` is also available in this ***reference table / color map visualization (link)***



Customize colors (cont’d)

- Add an argument `facecolor` to change the color of a histogram

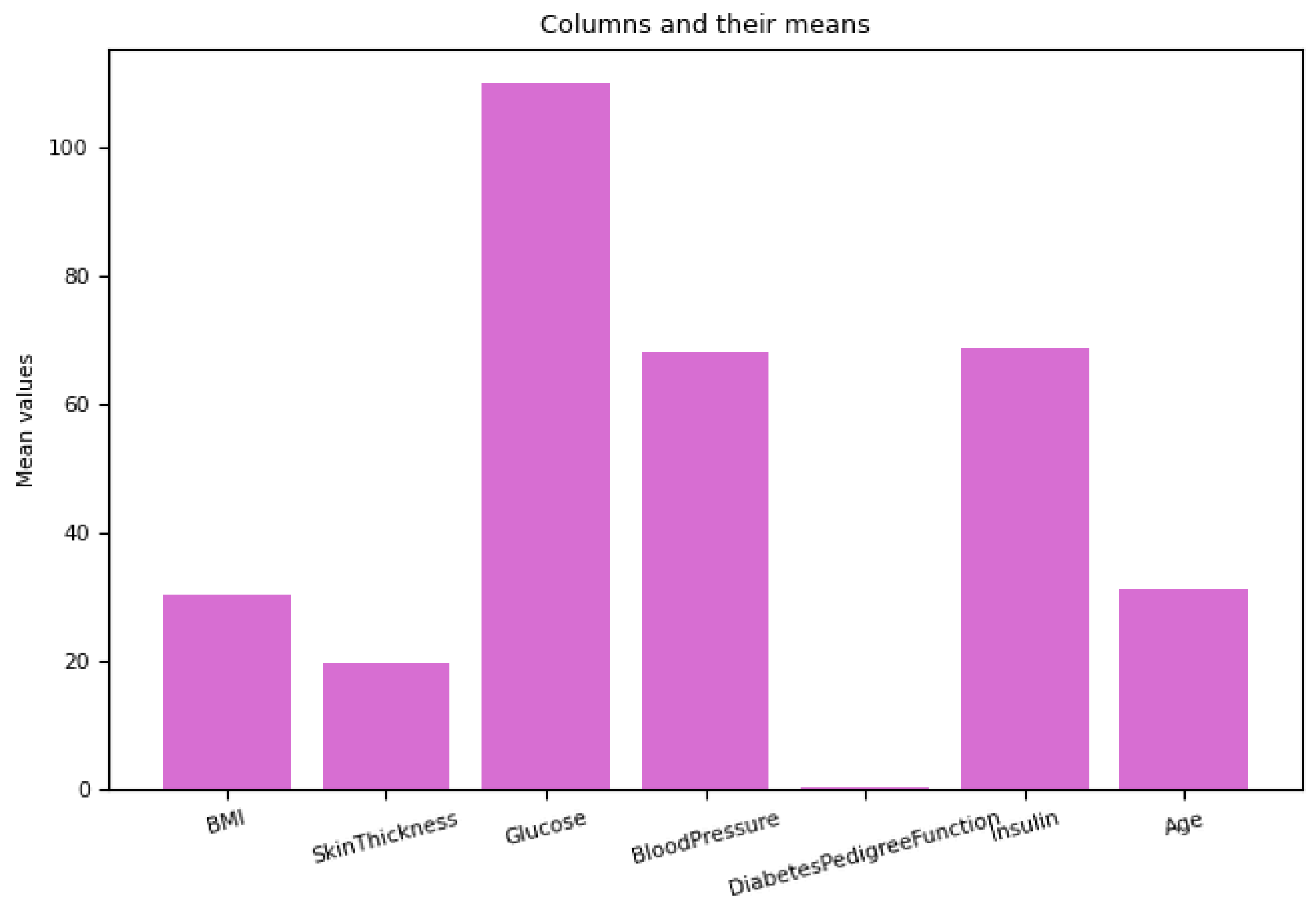
```
plt.hist(df_subset['BMI'],
            facecolor = 'goldenrod') #<- set
color
plt.xlabel('BMI')
plt.ylabel('Frequency')
plt.title('BMI distribution')
plt.show()
```



Customize colors (cont'd)

- Add an argument `color` to change the color of a bar chart

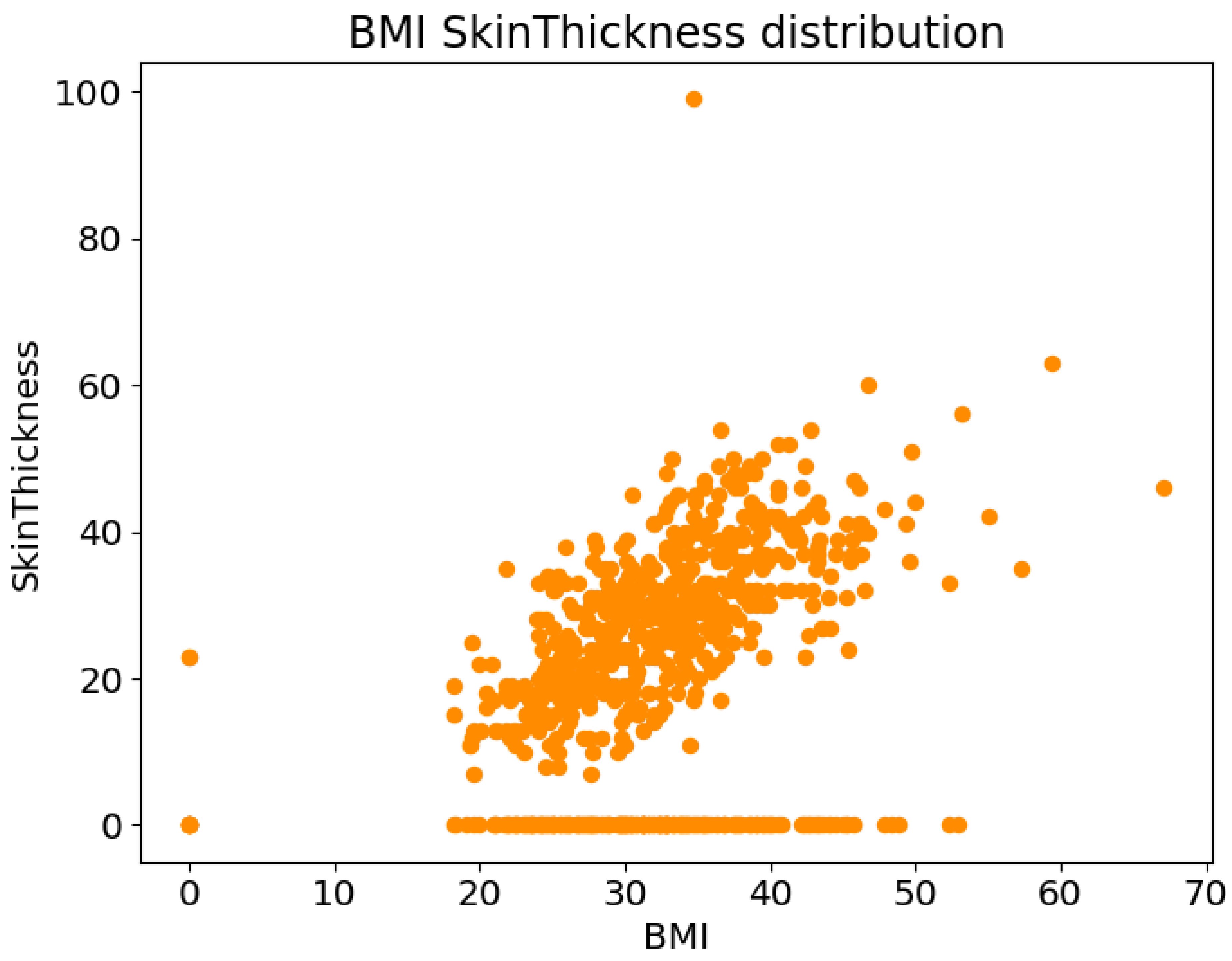
```
plt.bar(bar_positions,
        bar_heights,
        color = "orchid")
plt.xticks(bar_positions,
           bar_labels,
           rotation=15)
plt.ylabel('Mean values')
plt.title('Column Means')
plt.show()
```



Customize color: scatterplot

- Add an argument c to change the color of a scatterplot

```
plt.scatter(df_subset['BMI'],
            df_subset['SkinThickness'],
            c = 'darkorange') #<- set
marker type to diamond
plt.xlabel('BMI')
plt.ylabel('SkinThickness')
plt.title('BMI SkinThickness
distribution')
plt.show()
```



Customize color: map colors

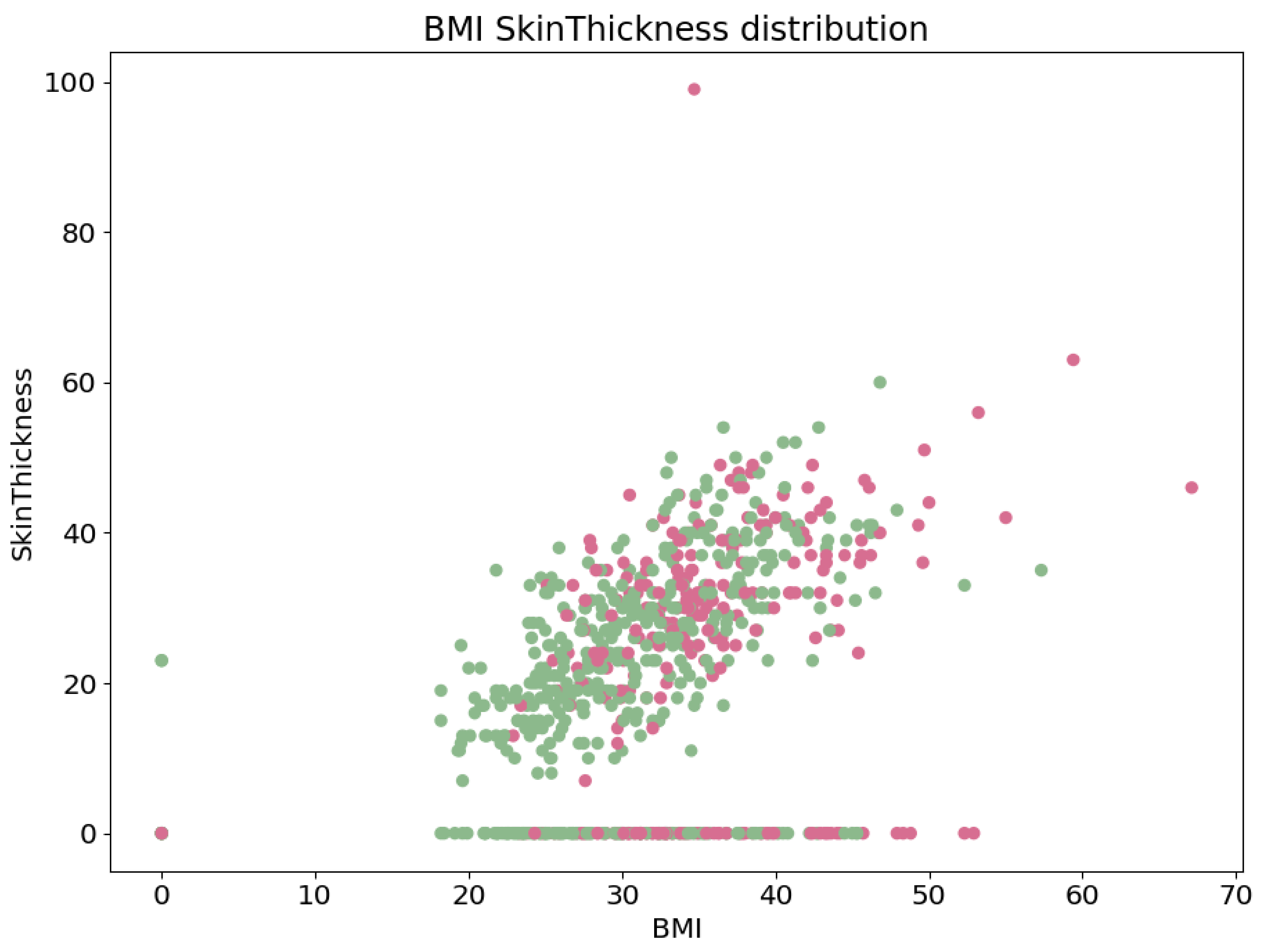
- When plotting data using scatterplots, there is the capability to see values corresponding to two or more distinct categories
- It is achieved by coloring observations that belong to different categories
- In this example, color the observations based on `'Outcome'` variable
- Add a new column to the dataframe called `color` with:
  - `'0'` corresponding to `darkseagreen` color, and
  - `'1'` corresponding to `palevioletred` color

Customize color: map colors (cont'd)

- Scatterplot points colored based on categories

```
color_dict = {int('0'): 'darkseagreen',
              int('1'):
'palevioletred'}
color =
df_subset['Outcome'].map(color_dict)
print(color.head())
```

```
plt.scatter(df_subset['BMI'],
            df_subset['SkinThickness'],
            c = color)
plt.xlabel('BMI')
plt.ylabel('SkinThickness')
plt.title('BMI SkinThickness
distribution')
plt.show()
```



Customize color: opacity

- When plotting many data points on one graph, lots of them get overplotted on top of each other

- That makes it difficult to discern how many observations are in the “clumps”

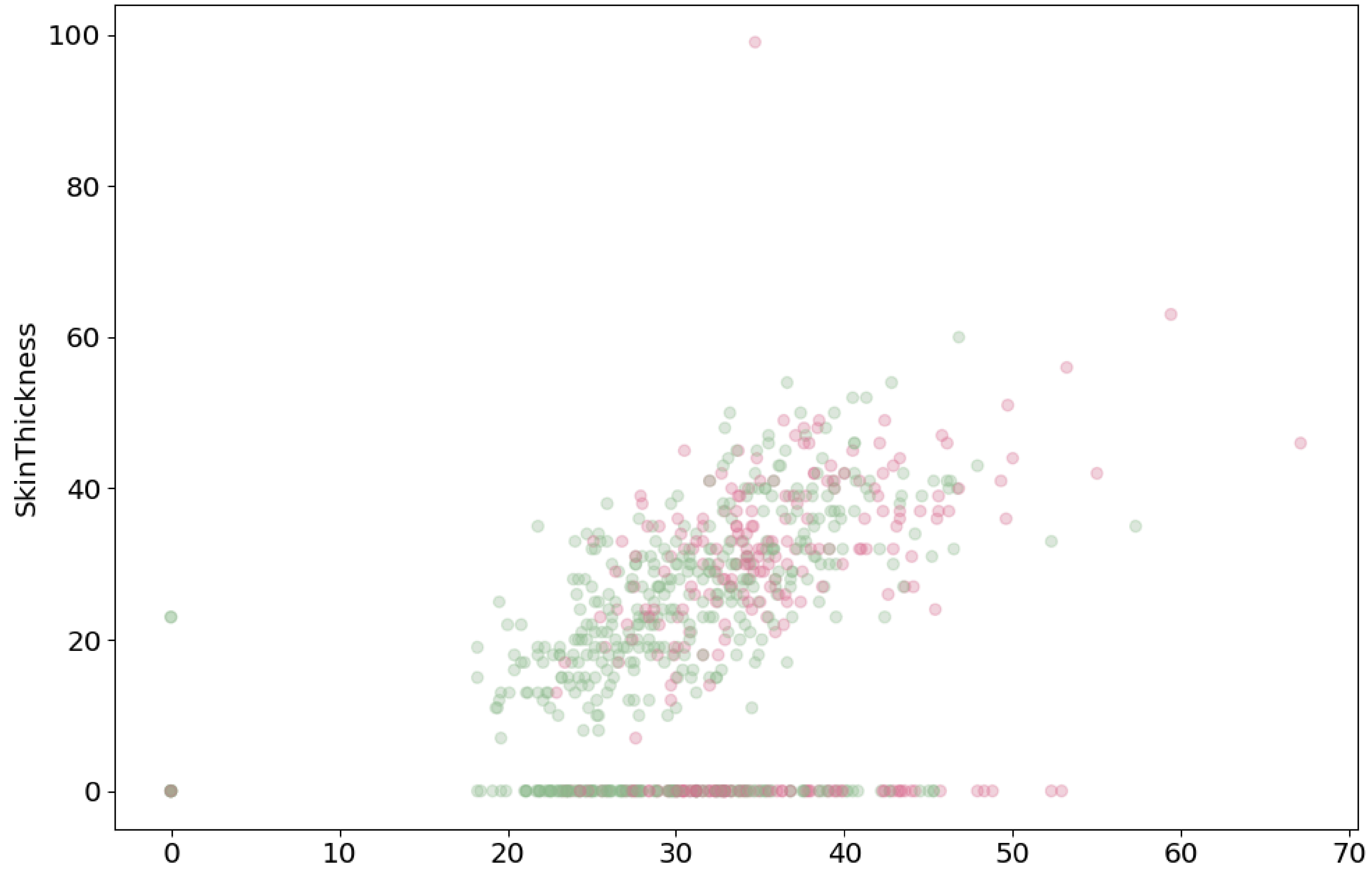
- The next slide address overplotting

Customize color: opacity (cont'd)

- One way to address overplotting is by setting the `alpha` parameter, which is responsible for regulating the **opacity** of the color

- It must be a value between `0` and `1`, where `0` is **transparent** and `1` is **opaque**

```
plt.scatter(df_subset['BMI'],df_subset['SkinThickness'],
            c = color, alpha = 0.3)
plt.xlabel('BMI')
plt.ylabel('SkinThickness')
plt.show()
```



Customize plot settings: available styles

- There are a number of pre-defined styles provided by `matplotlib`
- Preview available styles by running the following command:

```
print(plt.style.available)
```

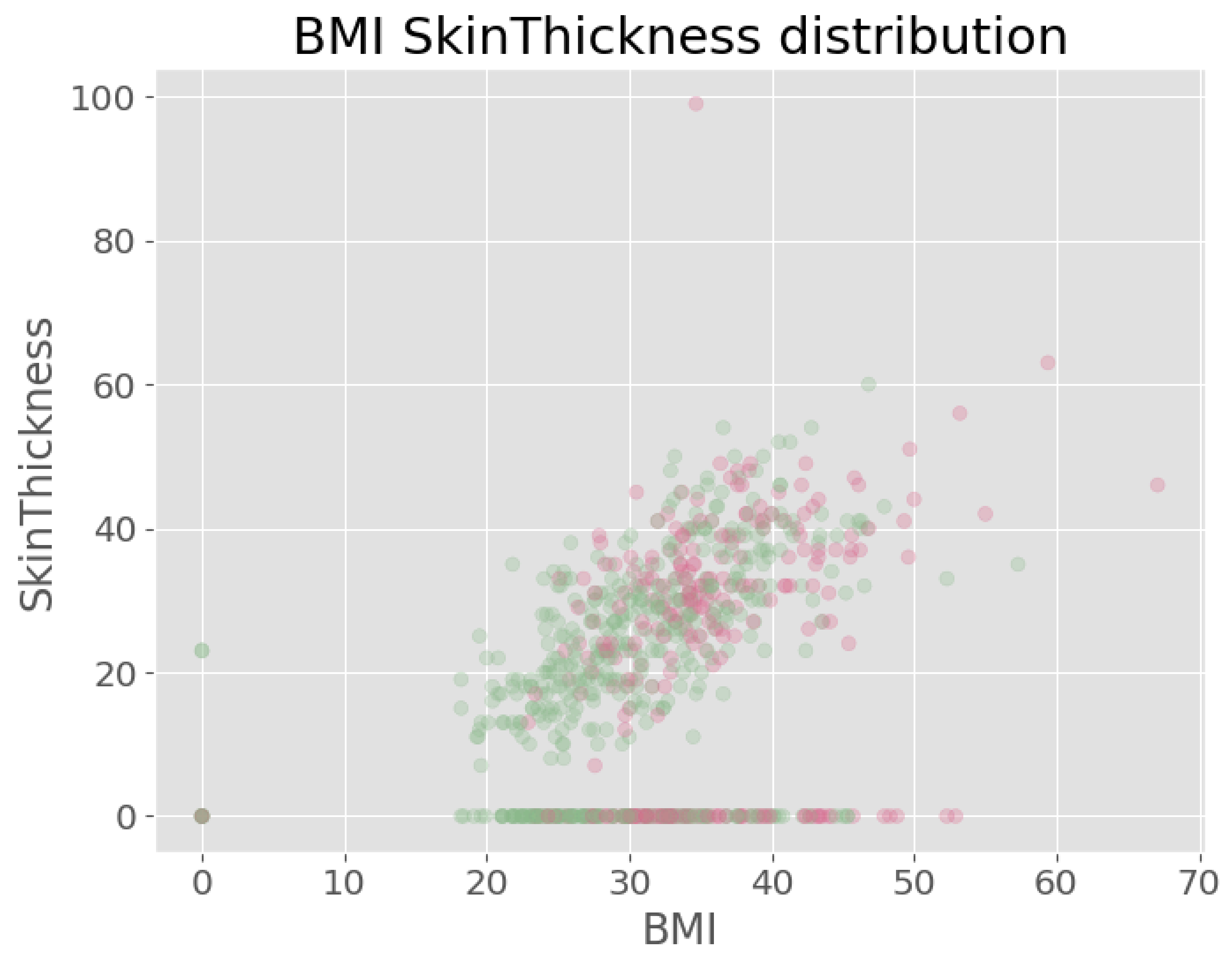
- One of the styles available is called “ggplot”, which emulates the aesthetics of `ggplot2`, one of the most widely used plotting libraries in R
- To use this style, run the following command:

```
plt.style.use('ggplot')
```

Customize plot settings: test ggplot style

- Plot using ggplot style

```
plt.scatter(df_subset['BMI'],
            df_subset['SkinThickness'],
            c = color,
            alpha = 0.3)
plt.xlabel('BMI')
plt.ylabel('SkinThickness')
plt.title('BMI SkinThickness distribution')
plt.show()
```



Customize plot settings: changing other presets

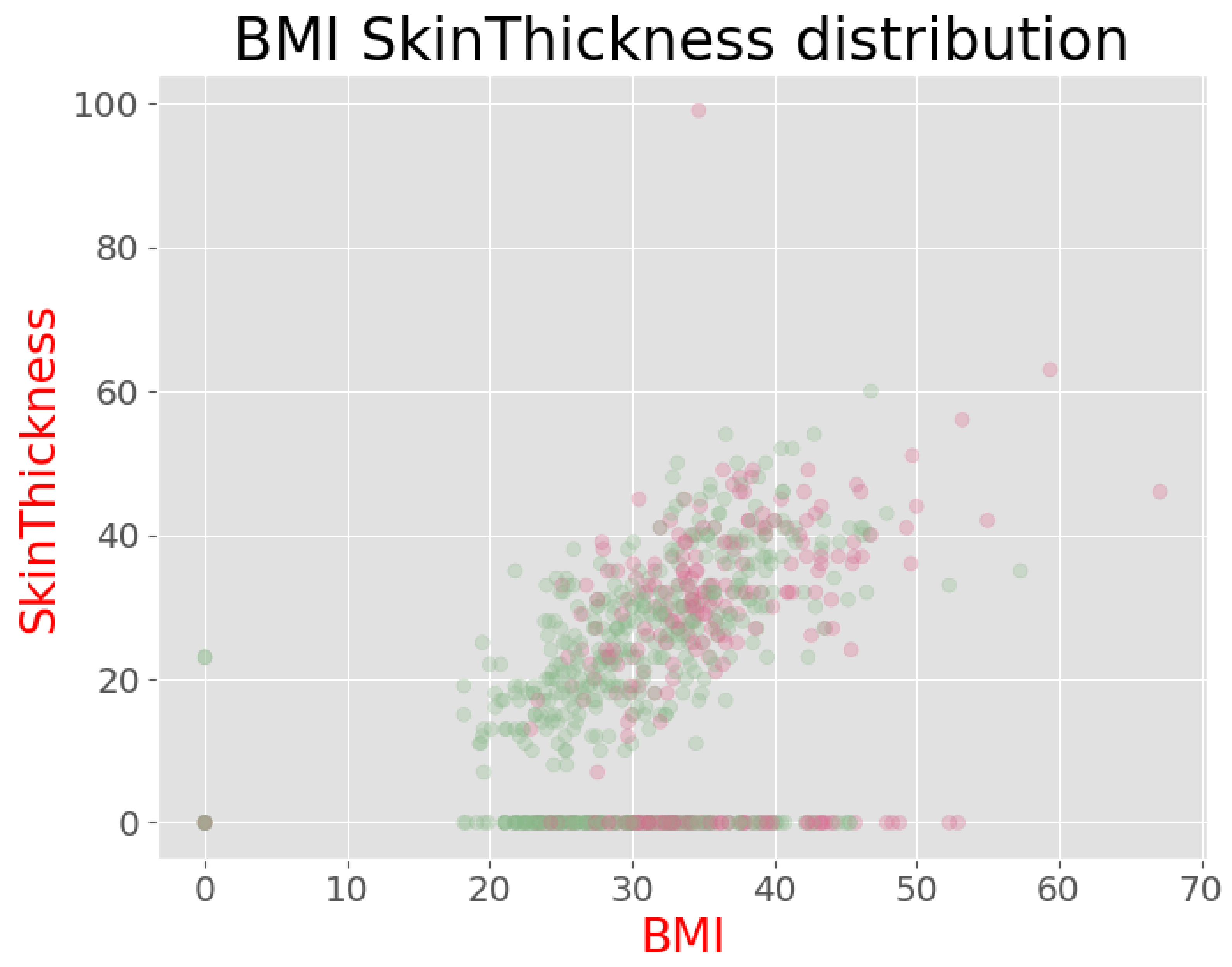
- As with all other plotting libraries, `matplotlib` comes with preset defaults for everything you see in your plot
- To adjust any preset defaults, use `plt.rcParams` variable, which is a dictionary-like object
- You can either set those parameters on a one-off basis or create a file with your presets and save it for your use for every project you work on
- Note: We will not cover it in class, but you ***click here***) for more information about it

Customize plot settings: labels

- The most common thing you would adjust is the **label appearance** for the following
  - x- and y-axis
  - x- and y-axis ticks
  - title

```
plt.rcParams['axes.labelsize'] = 20
plt.rcParams['axes.labelcolor'] = 'red'
plt.rcParams['axes.titlesize'] = 25
```

```
plt.scatter(df_subset['BMI'],
            df_subset['SkinThickness'],
            c = color,
            alpha = 0.3)
plt.xlabel('BMI')
plt.ylabel('SkinThickness')
plt.title('BMI SkinThickness distribution')
plt.show()
```

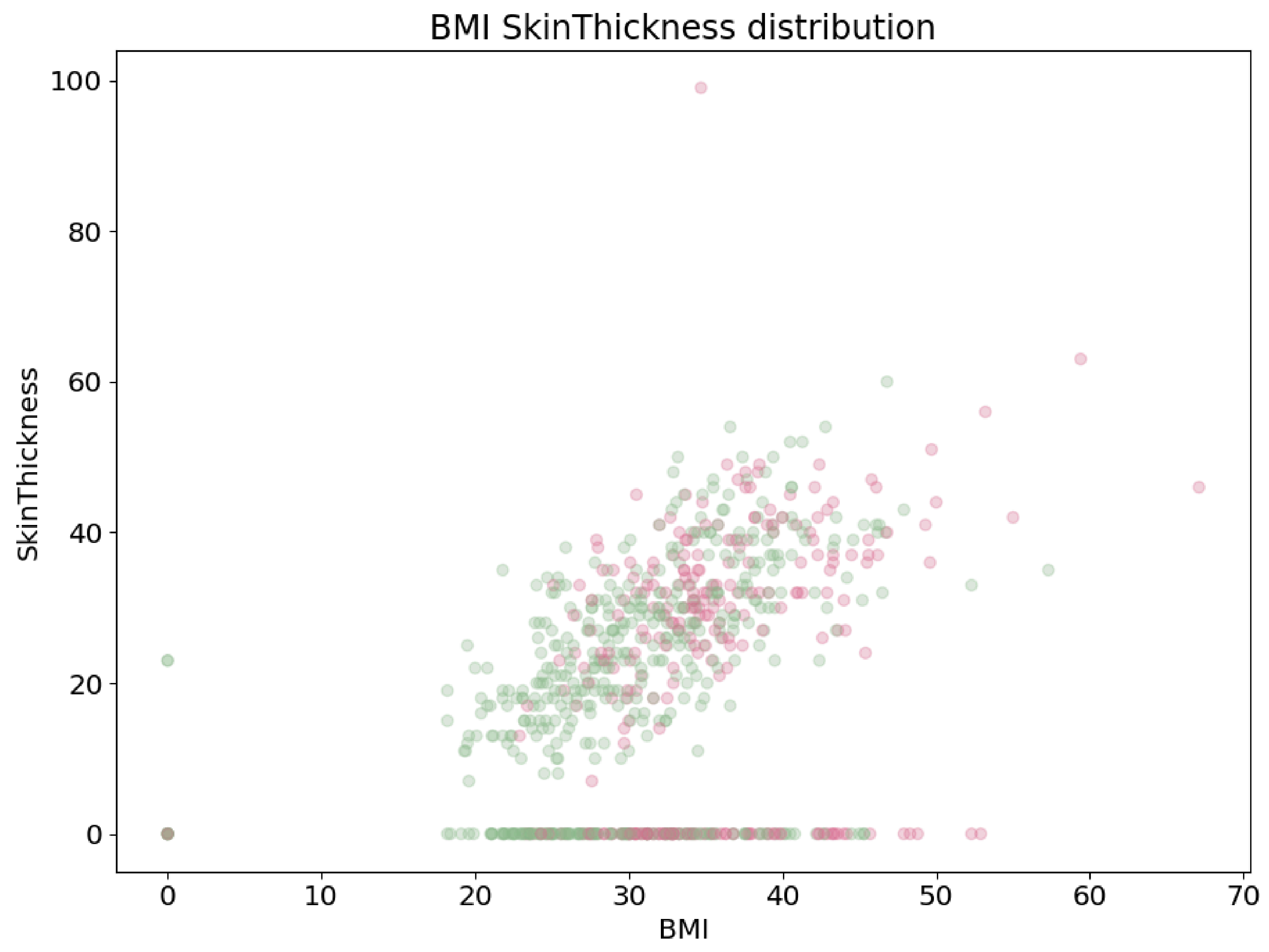


Customize plot settings: reset defaults

- The labels have been updated, but not necessarily in a good way
- Use the following function when we need to reset the `rcParams` to default:

```
plt.rcdefaults()
```

```
plt.scatter(df_subset['BMI'],
            df_subset['SkinThickness'],
            c = color,
            alpha = 0.3)
plt.xlabel('BMI')
plt.ylabel('SkinThickness')
plt.title('BMI SkinThickness distribution')
plt.show()
```



Customize anything

- All possible style customizations are available in a `matplotlibrc` file
- ***The customizing Matplotlib with style sheets and rcParams (link)*** contains all of them and any of those parameters can be passed to `rcParams` variable as we did earlier
- This sample includes a script of parameters and their default values
- Here's a part of the file. It contains a sample of all parameters for modifying the style of the `axes`

```
## ***************************************************************************
## * AXES                                                                    *
## ***************************************************************************
## Following are default face and edge colors, default tick sizes,
## default font sizes for tick labels, and so on.  See
## https://matplotlib.org/api/axes_api.html#module-matplotlib.axes
#axes.facecolor:     white   # axes background color
#axes.edgecolor:     black   # axes edge color
#axes.linewidth:     0.8     # edge line width
#axes.grid:          False   # display grid or not
#axes.grid.axis:     both    # which axis the grid should apply to
#axes.grid.which:    major   # grid lines at {major, minor, both} ticks
#axes.titlelocation: center  # alignment of the title: {left, right, center}
```

Knowledge check

Module completion checklist

Objective	Complete
Define bivariate plots and create scatterplots	✔
Construct customized graphs	✔

Congratulations on completing this module!

You are now ready to try Tasks 14-18 in the Exercise for this topic