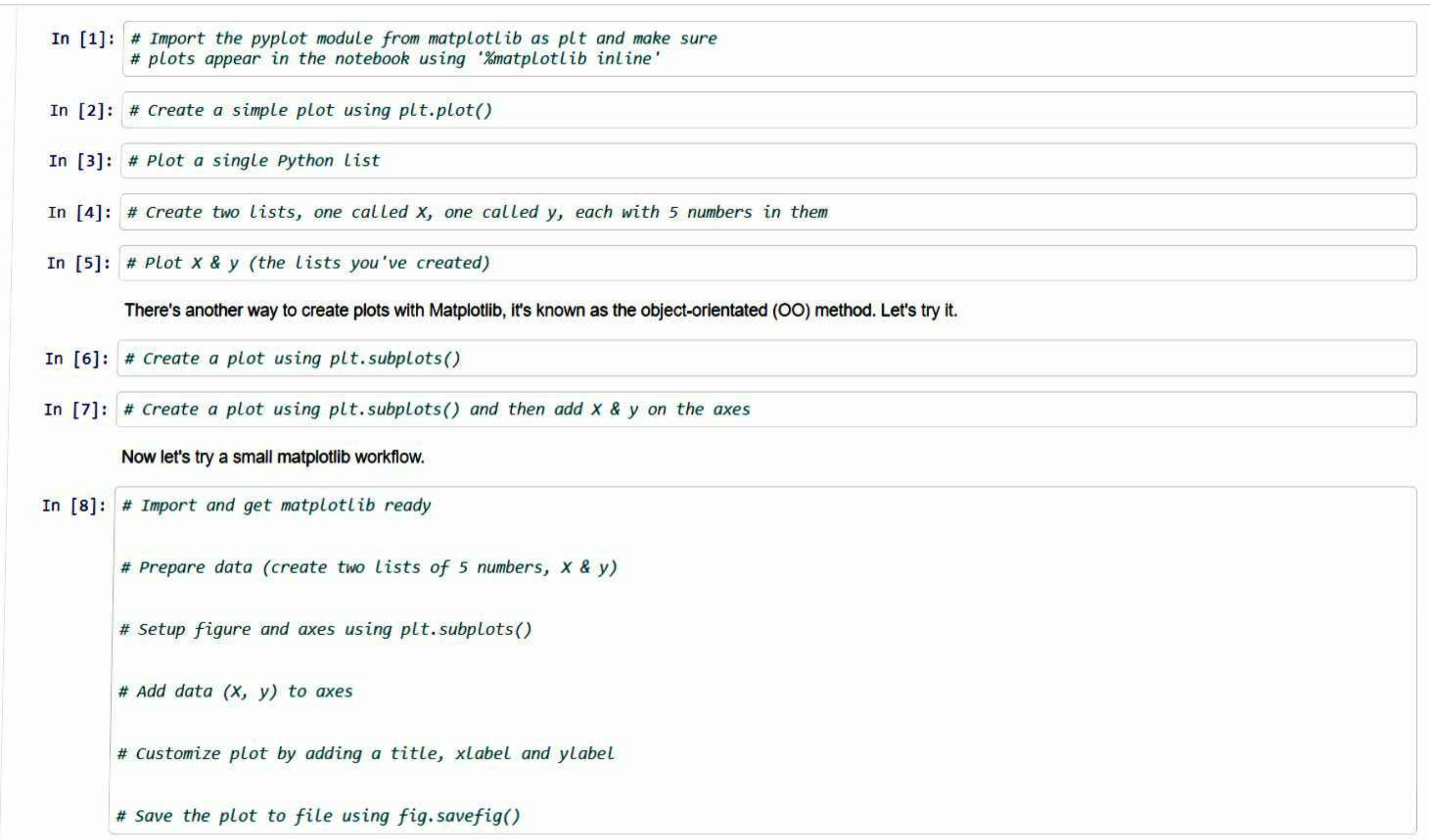


```
In [1]: # Import the pyplot module from matplotlib as plt and make sure
        # plots appear in the notebook using '%matplotlib inline'
```

```
In [2]: # Create a simple plot using plt.plot()
```

```
In [3]: # Plot a single Python list
```

```
In [4]: # Create two lists, one called X, one called y, each with 5 numbers in them
```

```
In [5]: # Plot X & y (the lists you've created)
```

There's another way to create plots with Matplotlib, it's known as the object-orientated (OO) method. Let's try it.

```
In [6]: # Create a plot using plt.subplots()
```

```
In [7]: # Create a plot using plt.subplots() and then add X & y on the axes
```

Now let's try a small matplotlib workflow.

```
In [8]: # Import and get matplotlib ready


        # Prepare data (create two lists of 5 numbers, X & y)


        # Setup figure and axes using plt.subplots()


        # Add data (X, y) to axes


        # Customize plot by adding a title, xlabel and ylabel


        # Save the plot to file using fig.savefig()
```

In [9]:
```
# Import NumPy as np
```

In [10]:
```
# Create an array of 100 evenly spaced numbers between 0 and 100 using NumPy and save it to variable X
```

In [11]:
```
# Create a plot using plt.subplots() and plot X versus X^2 (X squared)
```

We'll start with scatter plots.

In [12]:
```
# Create a scatter plot of X versus the exponential of X (np.exp(X))
```

In [13]:
```
# Create a scatter plot of X versus np.sin(X)
```

How about we try another type of plot? This time let's look at a bar plot. First we'll make some data.

In [14]:
```
# Create a Python dictionary of 3 of your favourite foods with
# The keys of the dictionary should be the food name and the values their price
```

In [15]:
```
# Create a bar graph where the x-axis is the keys of the dictionary
# and the y-axis is the values of the dictionary


# Add a title, xlabel and ylabel to the plot
```

In [16]:
```
# Make the same plot as above, except this time make the bars go horizontal
```

All this food plotting is making me hungry. But we've got a couple of plots to go.

Let's see a histogram.

In [17]:
```
# Create a random NumPy array of 1000 normally distributed numbers using NumPy and save it to X


# Create a histogram plot of X
```

In [18]:
```
# Create a NumPy array of 1000 random numbers and save it to X


# Create a histogram plot of X
```

In [19]:
```
# Create an empty subplot with 2 rows and 2 columns (4 subplots total)
```

Notice how the subplot has multiple figures. Now let's add data to each axes.

In [20]:
```
# Create the same plot as above with 2 rows and 2 columns and figsize of (10, 5)


# Plot X versus X/2 on the top left axes


# Plot a scatter plot of 10 random numbers on each axis on the top right subplot


# Plot a bar graph of the favourite food keys and values on the bottom left subplot


# Plot a histogram of 1000 random normally distributed numbers on the bottom right subplot
```

Woah. There's a lot going on there.

Now we've seen how to plot with Matplotlib and data directly. Let's practice using Matplotlib to plot with pandas.

First we'll need to import pandas and create a DataFrame work with.

In [21]:
```
# Import pandas as pd
```

In [22]:
```
# Import the '../data/car-sales.csv' into a DataFame called car_sales and view
```

In [23]:
```
# Try to plot the 'Price' column using the plot() function
```

Why doesn't it work?

Hint: It's not numeric data.

In the process of turning it to numeric data, let's create another column which adds the total amount of sales and another one which shows what date the car was sold.

Hint: To add a column up cumulatively, look up the cumsum() function. And to create a column of dates, look up the date_range() function.

In [24]:
```
# Remove the symbols, the final two numbers from the 'Price' column and convert it to numbers
```

In [25]:
```
# Add a column called 'Total Sales' to car_sales which cumulatively adds the 'Price' column


# Add a column called 'Sale Date' which lists a series of successive dates starting from today (your today)

# View the car_sales DataFrame
```

Now we've got a numeric column (`Total Sales`) and a dates column (`Sale Date`), let's visualize them.

In [26]:
```
# Use the plot() function to plot the 'Sale Date' column versus the 'Total Sales' column
```

In [27]:
```
# Convert the 'Price' column to the integers


# Create a scatter plot of the 'Odometer (KM)' and 'Price' column using the plot() function
```

In [28]:
```
# Create a NumPy array of random numbers of size (10, 4) and save it to X


# Turn the NumPy array X into a DataFrame with columns called ['a', 'b', 'c', 'd']


# Create a bar graph of the DataFrame
```

In [29]:
```
# Create a bar graph of the 'Make' and 'Odometer (KM)' columns in the car_sales DataFrame
```

In [30]:
```
# Create a histogram of the 'Odometer (KM)' column
```

In [31]:
```
# Create a histogram of the 'Price' column with 20 bins
```

Now we've seen a few examples of plotting directly from DataFrames using the `car_sales` dataset.

Let's try using a different dataset.

In [32]:
```
# Import "../data/heart-disease.csv" and save it to the variable "heart_disease"
```

In [33]:
```
# View the first 10 rows of the heart_disease DataFrame
```

In [34]:
```
# Create a histogram of the "age" column with 50 bins
```

In [35]:
```
# Call plot.hist() on the heart_disease DataFrame and toggle the
# "subplots" parameter to True
```

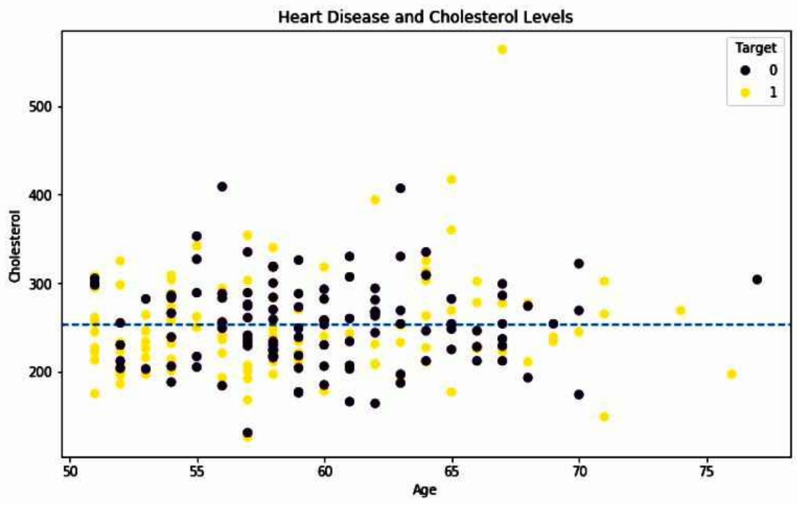
That plot looks pretty squished. Let's change the figsize.

In [36]:
```
# Call the same line of code from above except change the "figsize" parameter
# to be (10, 30)
```

Now let's try comparing two variables versus the target variable.

More specifially we'll see how age and cholesterol combined effect the target in **patients over 50 years old**.

For this next challenge, we're going to be replicating the following plot:



In [37]:
```
# Replicate the above plot in whichever way you see fit

# Note: The method below is only one way of doing it, yours might be
# slightly different
```

In [37]:
```
# Replicate the above plot in whichever way you see fit

# Note: The method below is only one way of doing it, yours might be
# slightly different

# Create DataFrame with patients over 50 years old


# Create the plot


# Plot the data


# Customize the plot


# Add a meanline
```

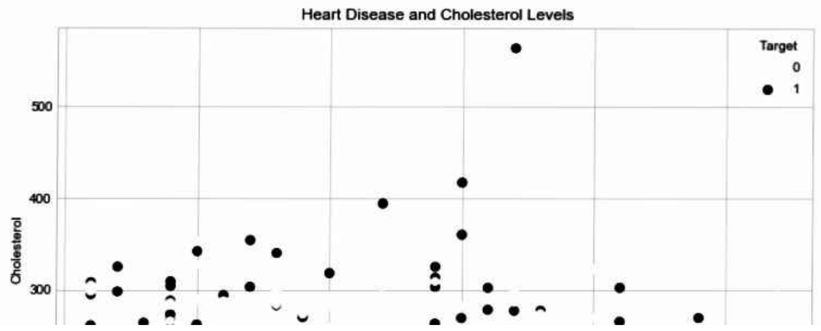
Beatiful, now you've created a plot of two different variables, let's change the style.

In [38]:
```
# Check what styles are available under plt
```

In [39]:
```
# Change the style to use "seaborn-whitegrid"
```

Now the style has been changed, we'll replot the same figure from above and see what it looks like.

If you've changed the style correctly, it should look like the following:

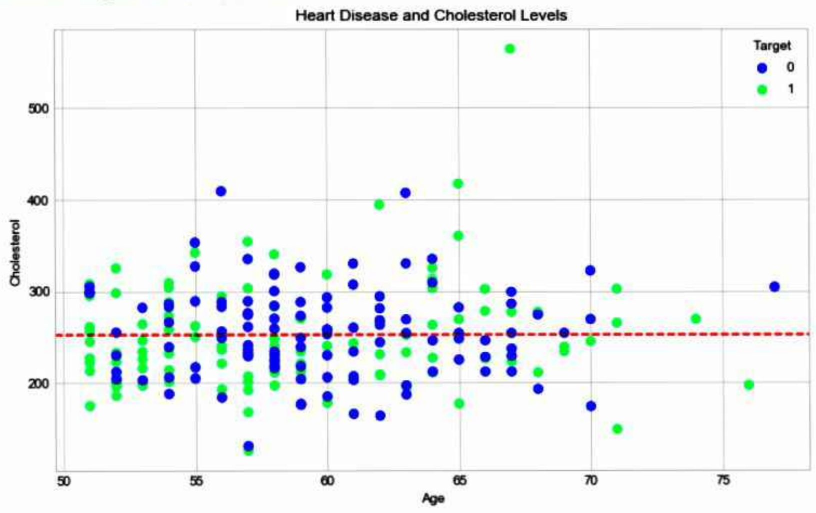


In [40]:
```python
# Reproduce the same figure as above with the "seaborn-whitegrid" style

# Create the plot


# Plot the data


# Customize the plot


# Add a meanline
```

Wonderful, you've changed the style of the plots and the figure is looking different but the dots aren't a very good colour.

Let's change the `cmap` parameter of `scatter()` as well as the `color` parameter of `axhline()` to fix it.

Completing this step correctly should result in a figure which looks like this:



In [41]:
```python
# Replot the same figure as above except change the "cmap" parameter
```

In [41]:
```python
# Replot the same figure as above except change the "cmap" parameter
# of scatter() to "winter"
# Also change the "color" parameter of axhline() to "red"

# Create the plot


# Plot the data


# Customize the plot


# Add a meanline
```

Beautiful! Now our figure has an upgraded color scheme let's save it to file.

In [42]:
```python
# Save the current figure using savefig(), the file name can be anything you want
```

In [43]:
```python
# Reset the figure by calling plt.subplots()
```