Matplotlib

March 7, 2023

#

Matplotlib Practice

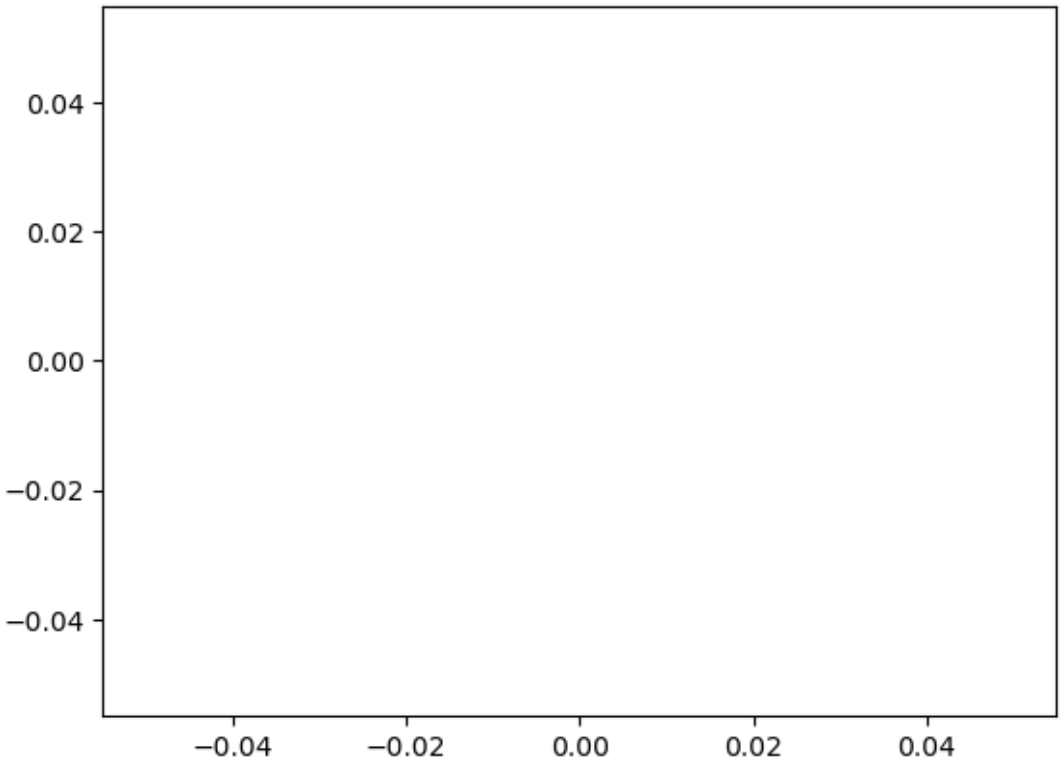
This notebook offers a set of exercises to different tasks with Matplotlib.

For further reference and resources, it's advised to check out the Matplotlib documentation.

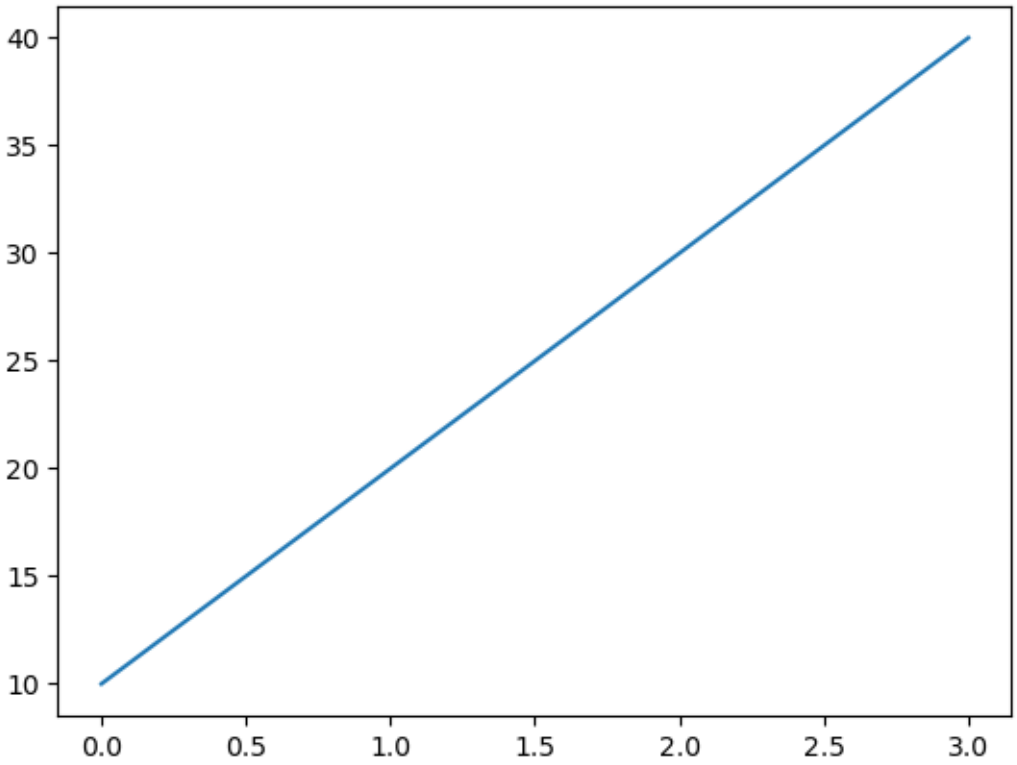
[1]:
```
# Import the pyplot module from matplotlib as plt and make sure
# plots appear in the notebook using '%matplotlib inline'
%matplotlib inline
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
```

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

[2]:
```
[]
```

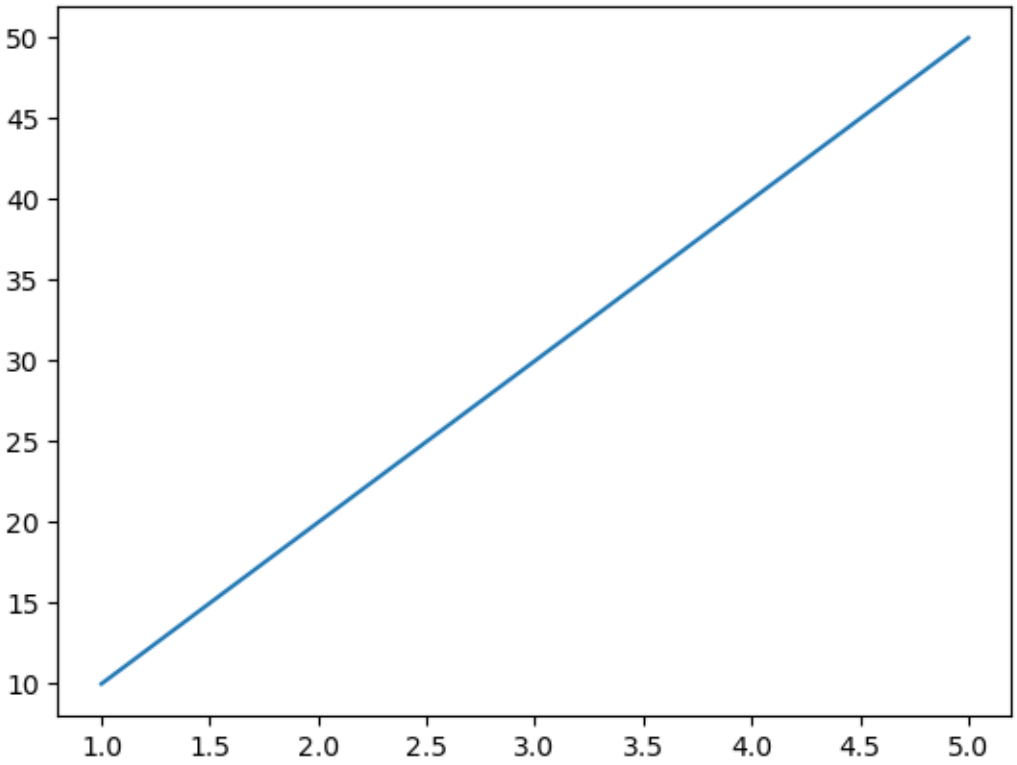


[7]:
```
# Plot a single Python list
a=[10,20,30,40]
plt.plot(a);
```



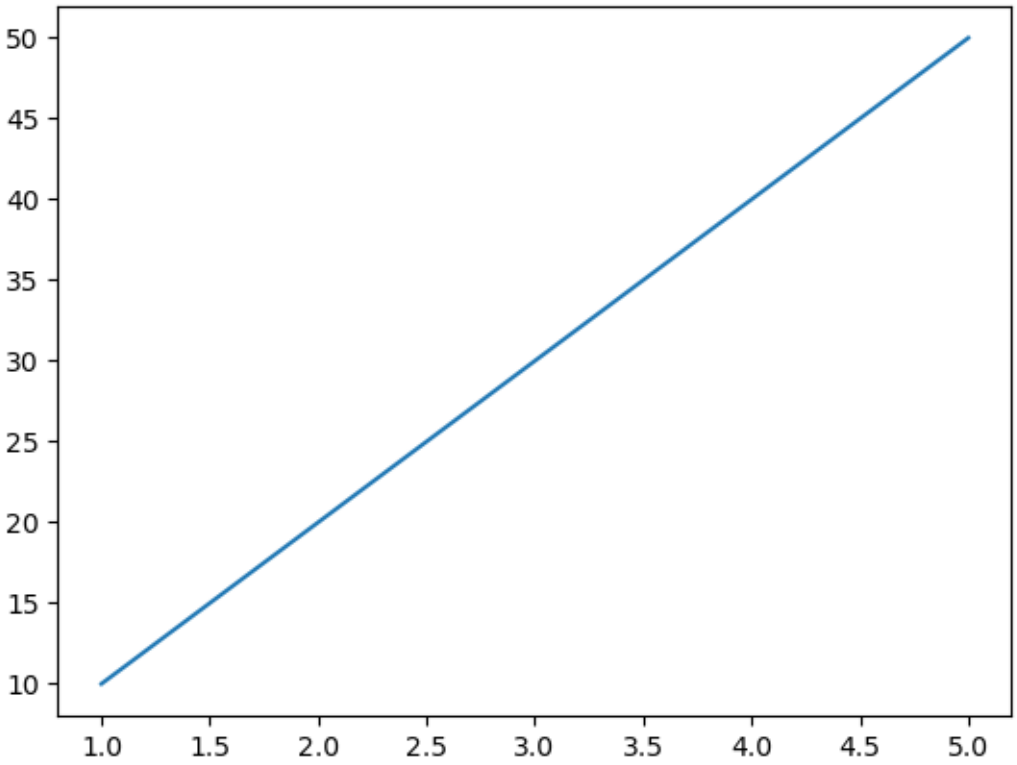
[9]:
```
# Create two lists, one called X, one called y, each with 5 numbers in them
x=[1,2,3,4,5]
y=[10,20,30,40,50]
```

[10]:
```
# Plot X & y (the lists you've created)
plt.plot(x,y);
```

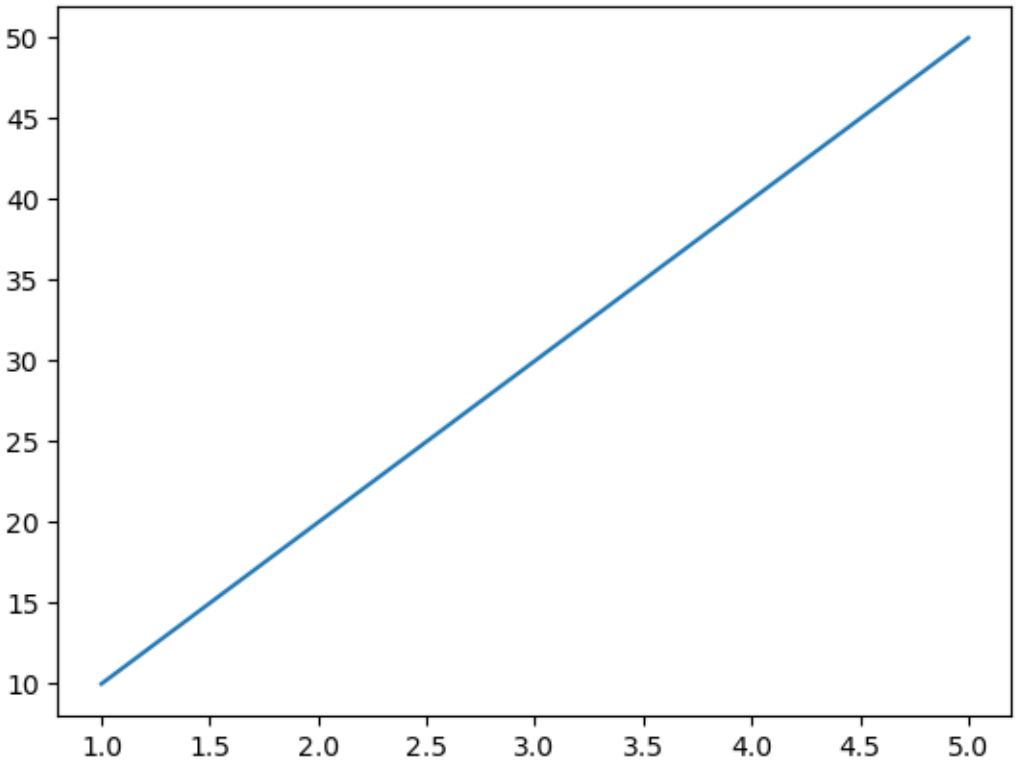


There's another way to create plots with Matplotlib, it's known as the object-orientated (OO) method. Let's try it.

[19]:
```
# Create a plot using plt.subplots()
fig, ax=plt.subplots(nrows=1, ncols=1)
ax.plot(x,y);
```

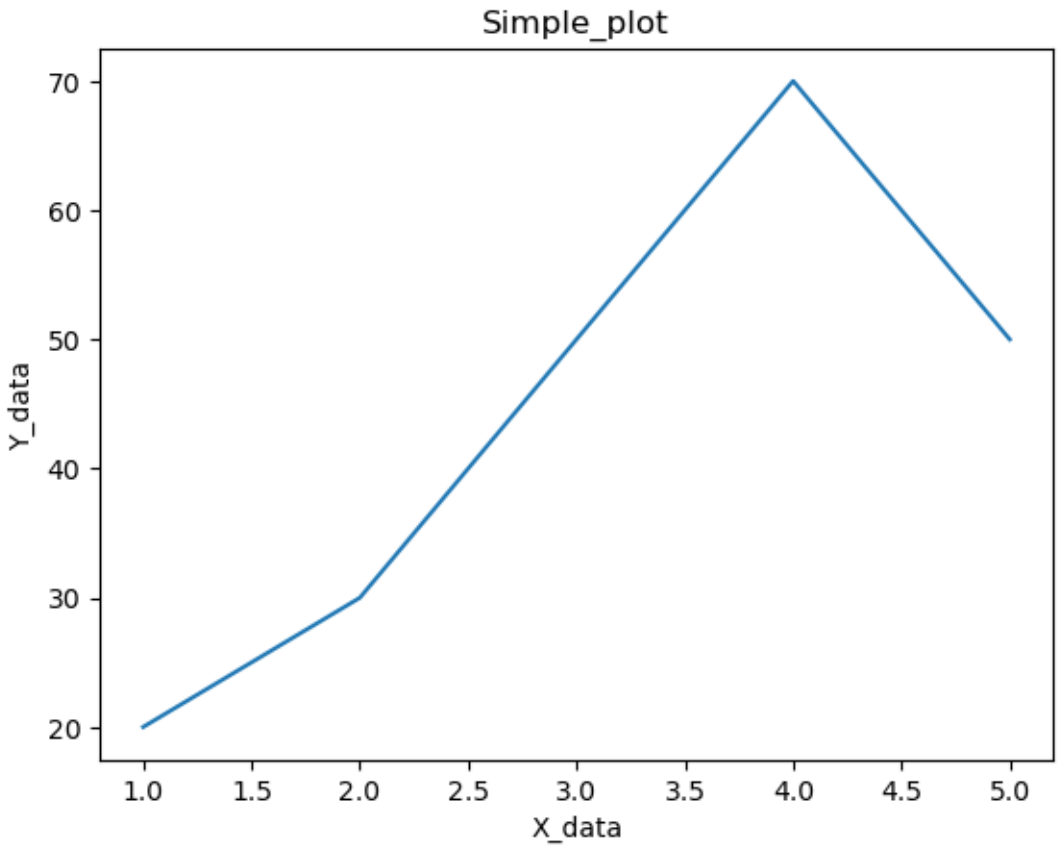


[20]:
```
# Create a plot using plt.subplots() and then add X & y on the axes
fig, ax=plt.subplots(nrows=1, ncols=1,
                     )
ax.plot(x,y);
```



Now let's try a small matplotlib workflow.

[27]:
```
# Import and get matplotlib ready
%matplotlib inline
import matplotlib.pyplot as plt
# Prepare data (create two lists of 5 numbers, X & y)
x=[1,2,3,4,5]
y=[20,30,50,70,50]

# Setup figure and axes using plt.subplots()
fig, ax= plt.subplots(nrows=1,
                      ncols=1)

# Add data (X, y) to axes
ax.plot(x,y);

# Customize plot by adding a title, xlabel and ylabel
ax.set(title="Simple_plot", xlabel="X_data", ylabel="Y_data");

# Save the plot to file using fig.savefig()
fig.savefig(r'D:\Study\Complete Machine Learning & Data Science Bootcamp␣
 ↪2022\08 - Matplotlib Plotting and Data Visualization')
```



Okay, this is a simple line plot, how about something a little different?

To help us, we'll import NumPy.

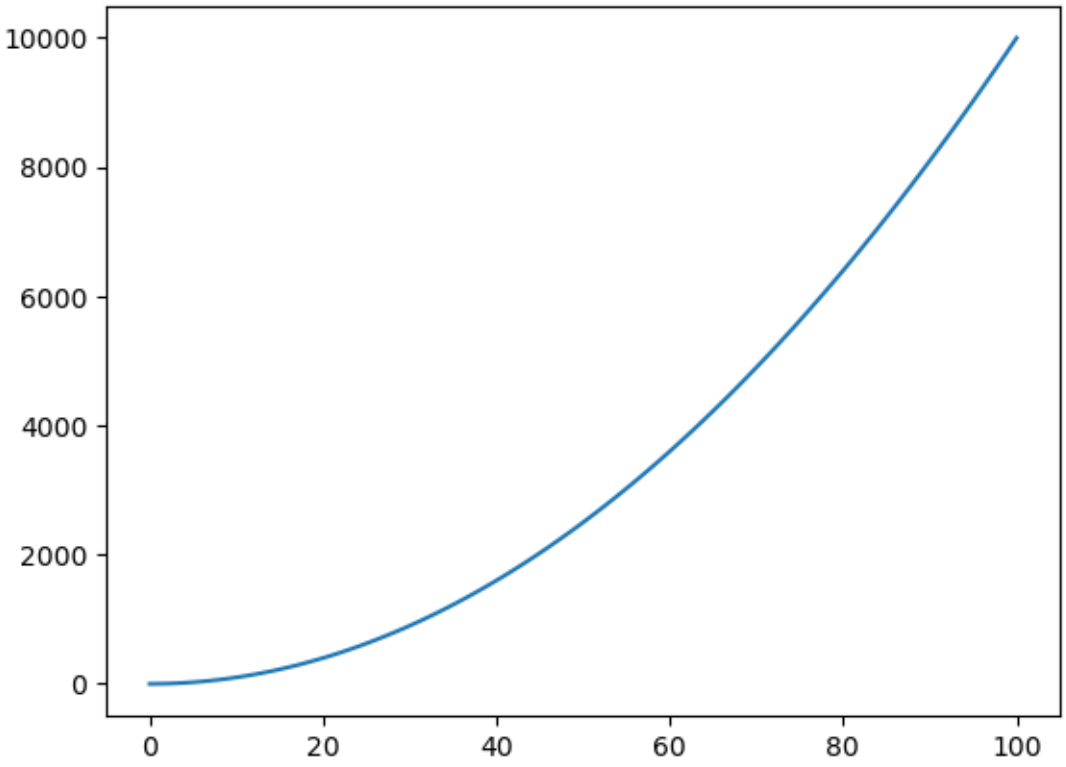
[58]:
```
# Import NumPy as np
import numpy as np
```

[59]:
```
# Create an array of 100 evenly spaced numbers between 0 and 100 using NumPy␣
  ↪and save it to variable X
x=np.linspace(0,100,100)
x
```

[59]:
```
array([  0.        ,   1.01010101,   2.02020202,   3.03030303,
         4.04040404,   5.05050505,   6.06060606,   7.07070707,
         8.08080808,   9.09090909,  10.1010101 ,  11.11111111,
        12.12121212,  13.13131313,  14.14141414,  15.15151515,
        16.16161616,  17.17171717,  18.18181818,  19.19191919,
        20.2020202 ,  21.21212121,  22.22222222,  23.23232323,
        24.24242424,  25.25252525,  26.26262626,  27.27272727,
        28.28282828,  29.29292929,  30.3030303 ,  31.31313131,
        32.32323232,  33.33333333,  34.34343434,  35.35353535,
```

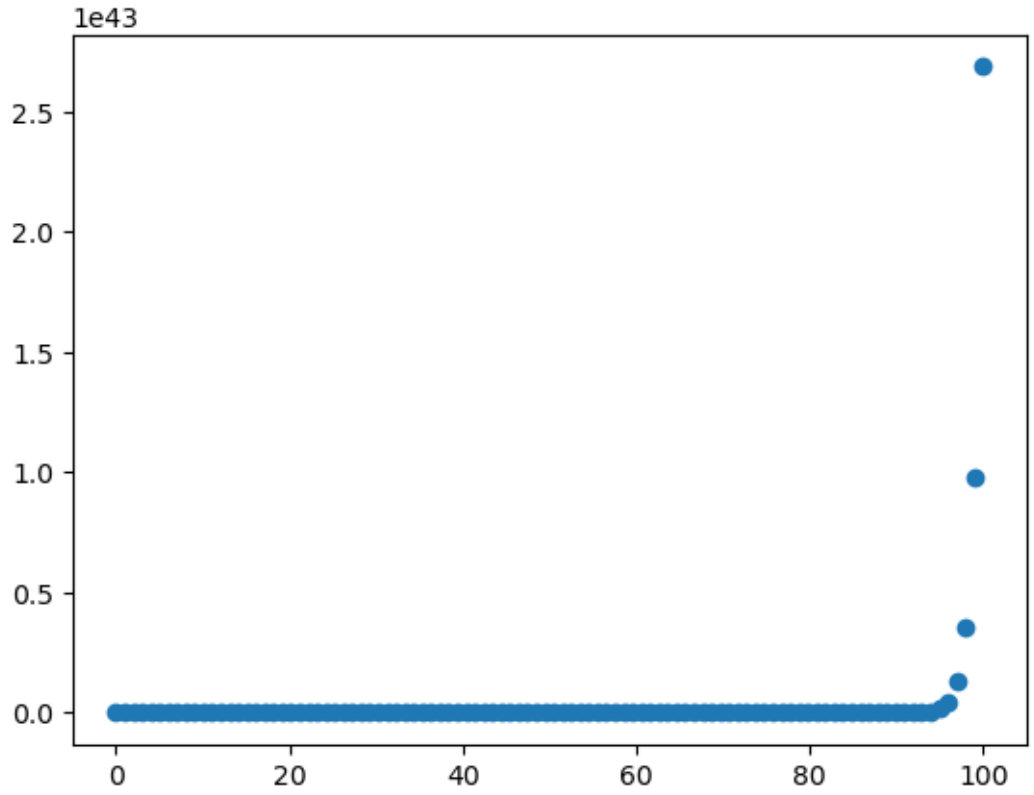
```
36.36363636,  37.37373737,  38.38383838,  39.39393939,
40.4040404 ,  41.41414141,  42.42424242,  43.43434343,
44.44444444,  45.45454545,  46.46464646,  47.47474747,
48.48484848,  49.49494949,  50.50505051,  51.51515152,
52.52525253,  53.53535354,  54.54545455,  55.55555556,
56.56565657,  57.57575758,  58.58585859,  59.5959596 ,
60.60606061,  61.61616162,  62.62626263,  63.63636364,
64.64646465,  65.65656566,  66.66666667,  67.67676768,
68.68686869,  69.6969697 ,  70.70707071,  71.71717172,
72.72727273,  73.73737374,  74.74747475,  75.75757576,
76.76767677,  77.77777778,  78.78787879,  79.7979798 ,
80.80808081,  81.81818182,  82.82828283,  83.83838384,
84.84848485,  85.85858586,  86.86868687,  87.87878788,
88.88888889,  89.8989899 ,  90.90909091,  91.91919192,
92.92929293,  93.93939394,  94.94949495,  95.95959596,
96.96969697,  97.97979798,  98.98989899, 100.        ])
```

[60]:
```
# Create a plot using plt.subplots() and plot X versus X^2 (X squared)
fig, ax= plt.subplots()
ax.plot(x, x**2);
```

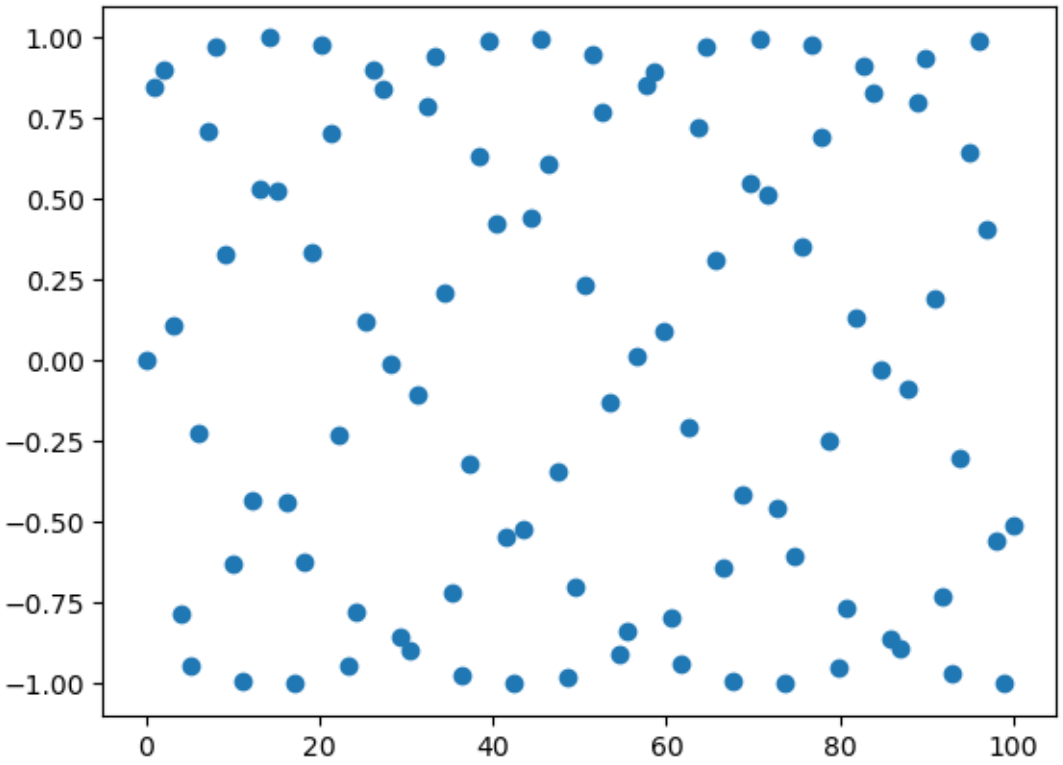


We'll start with scatter plots.

[61]:
```
# Create a scatter plot of X versus the exponential of X (np.exp(X))
fig, ax= plt.subplots()
ax.scatter(x,(np.exp(x)));
```



[62]:
```
# Create a scatter plot of X versus np.sin(X)
fig, ax= plt.subplots()
ax.scatter(x, np.sin(x));
```

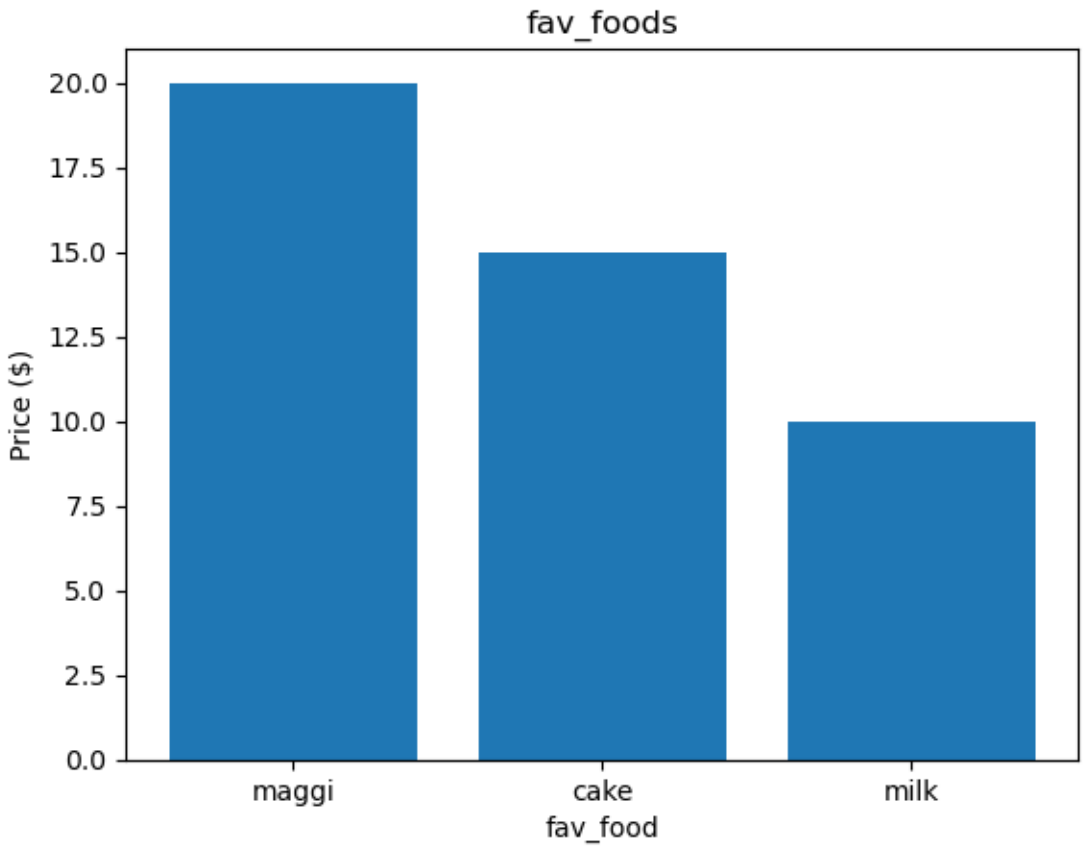


How about we try another type of plot? This time let's look at a bar plot. First we'll make some data.

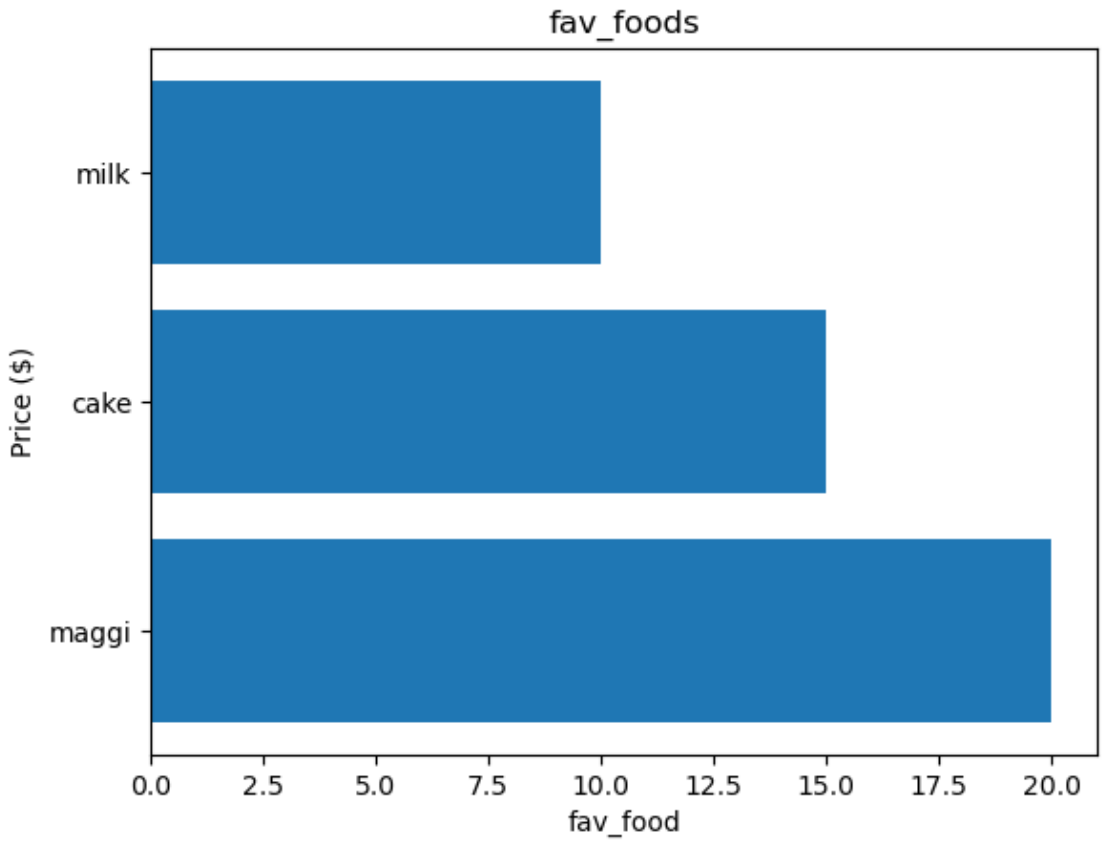
[64]:
```
# Create a Python dictionary of 3 of your favourite foods with
# The keys of the dictionary should be the food name and the values their price
fav_foods={'maggi':20,'cake':15, 'milk':10}
fav_foods
```

[64]:
```
{'maggi': 20, 'cake': 15, 'milk': 10}
```

[82]:
```
# Create a bar graph where the x-axis is the keys of the dictionary
# and the y-axis is the values of the dictionary
fav_foods={'maggi':20,'cake':15, 'milk':10}
fig, ax= plt.subplots()
ax.bar(fav_foods.keys(), fav_foods.values())
# Add a title, xlabel and ylabel to the plot
ax.set(title="fav_foods",
       xlabel='fav_food',
       ylabel= 'Price ($)');
```



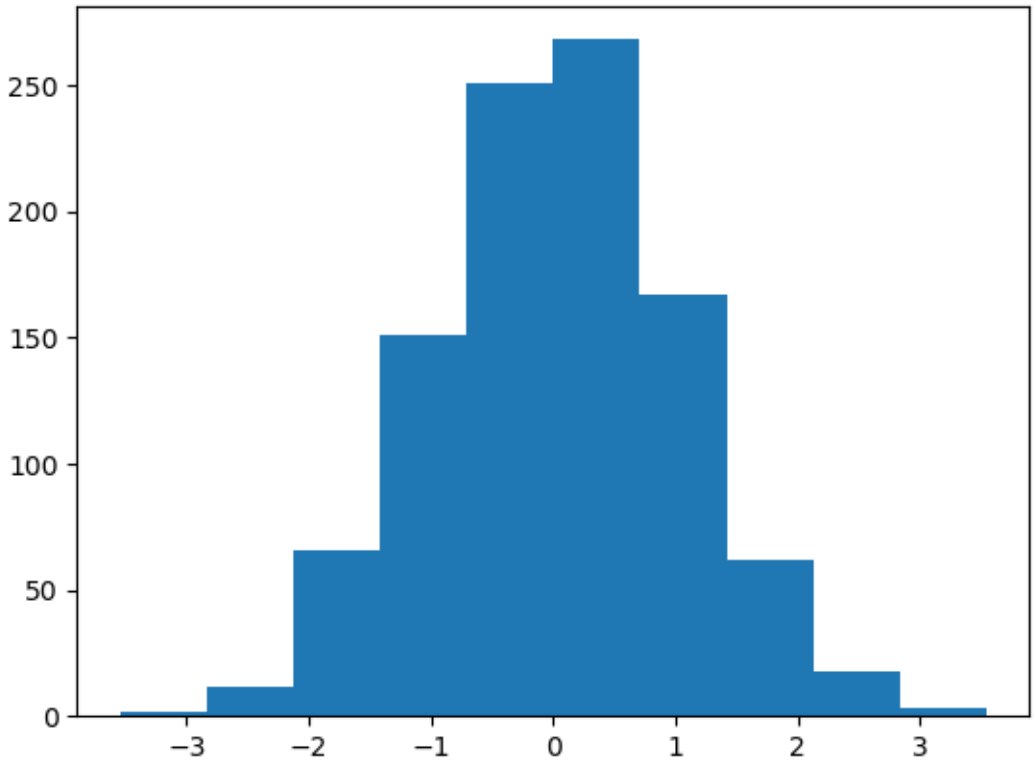
[87]:
```
# Make the same plot as above, except this time make the bars go horizontal
fig, ax= plt.subplots()
ax.barh(list(fav_foods.keys()), list(fav_foods.values()))
# Add a title, xlabel and ylabel to the plot
ax.set(title="fav_foods",
       xlabel='fav_food',
       ylabel= 'Price ($)');
```



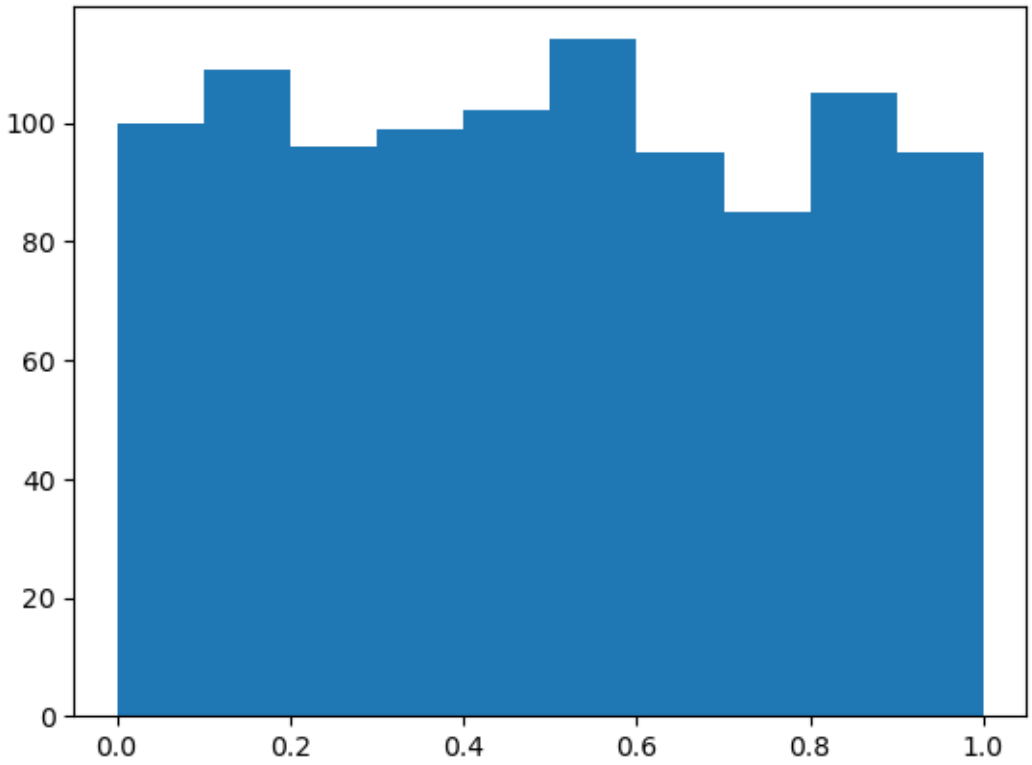
All this food plotting is making me hungry. But we've got a couple of plots to go.

Let's see a histogram.

[100]:
```
# Create a random NumPy array of 1000 normally distributed numbers using NumPy␣
↪and save it to X
X=np.random.randn(1000)
fig, ax=plt.subplots()
# Create a histogram plot of X
ax.hist(X);
```

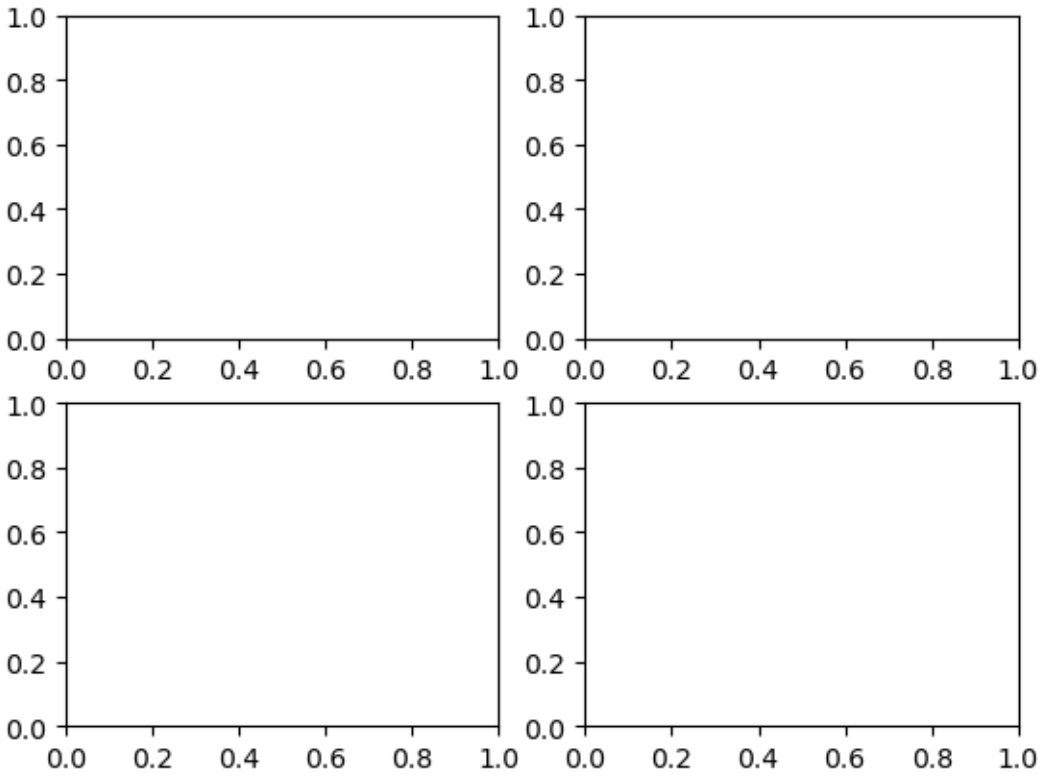


[112]:
```
# Create a NumPy array of 1000 random numbers and save it to X
X=np.random.random(1000)

# Create a histogram plot of X
fig, ax=plt.subplots()

ax.hist(X);
```



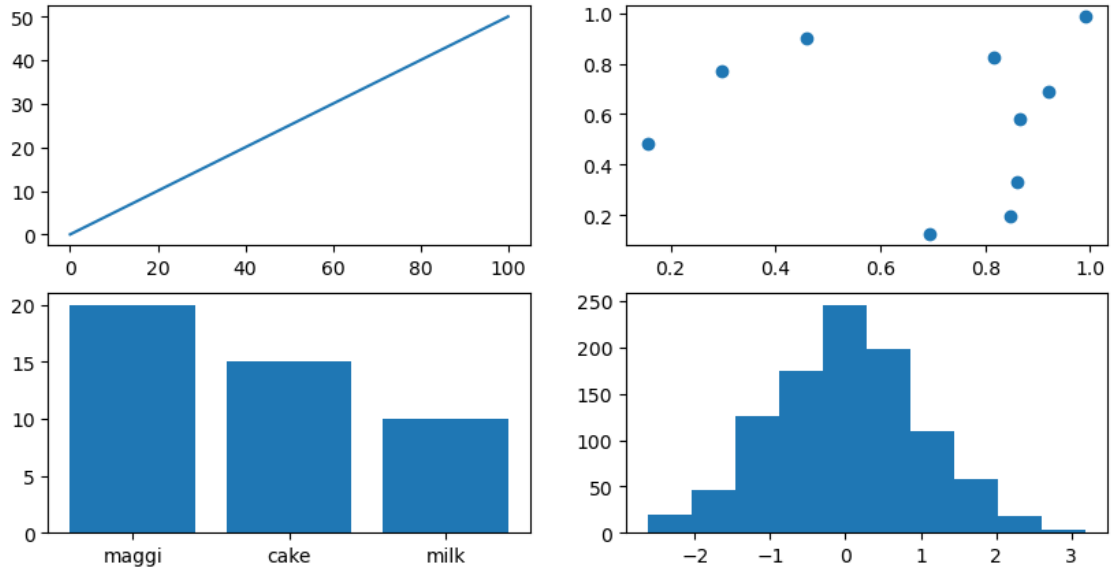
Notice how the distributions (spread of data) are different. Why do they differ?

[113]:
```
# Create an empty subplot with 2 rows and 2 columns (4 subplots total)
fig, ax=plt.subplots(nrows=2,
                     ncols=2)
```



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

[135]:
```
# Create the same plot as above with 2 rows and 2 columns and figsize of (10, 5)
fig, ((ax0,ax1), (ax2,ax3))=plt.subplots(nrows=2,
                    ncols=2,
                    figsize=(10,5))

# Plot X versus X/2 on the top left axes
ax0.plot(x,x/2);

# Plot a scatter plot of 10 random numbers on each axis on the top right subplot
ax1.scatter(np.random.random(10), np.random.random(10));

# Plot a bar graph of the favourite food keys and values on the bottom left␣
 ↪subplot
ax2.bar(fav_foods.keys(),fav_foods.values());

# Plot a histogram of 1000 random normally distributed numbers on the bottom␣
 ↪right subplot
ax3.hist(np.random.randn(1000));
```



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.

[136]:
```
# Import pandas as pd
import pandas as pd
```

[189]:
```
# Import the '../data/car-sales.csv' into a DataFame called car_sales and view
car_sales=pd.read_csv('007 car-sales.csv')
car_sales
# removing extra columns
car_sales=car_sales.drop('Unnamed: 0', axis=1)
car_sales
```

[189]:

	Make	Colour	Odometer (KM)	Doors	Price
0	Toyota	White	150043	4	$4,000.00
1	Honda	Red	87899	4	$5,000.00
2	Toyota	Blue	32549	3	$7,000.00
3	BMW	Black	11179	5	$22,000.00
4	Nissan	White	213095	4	$3,500.00
5	Toyota	Green	99213	4	$4,500.00
6	Honda	Blue	45698	4	$7,500.00
7	Honda	Blue	54738	4	$7,000.00
8	Toyota	White	60000	4	$6,250.00
9	Nissan	White	31600	4	$9,700.00

[190]:
```
# Try to plot the 'Price' column using the plot() function
car_sales['Price'].plot()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[190], line 2
      1 # Try to plot the 'Price' column using the plot() function
----> 2 car_sales['Price'].plot()

File ~\Desktop\sample_project_1\env\lib\site-packages\pandas\plotting\_core.py:
 ↪1000, in PlotAccessor.__call__(self, *args, **kwargs)
    997             label_name = label_kw or data.columns
    998             data.columns = label_name
-> 1000 return plot_backend.plot(data, kind=kind, **kwargs)

File␣
 ↪~\Desktop\sample_project_1\env\lib\site-packages\pandas\plotting\_matplotlib\__init__.
 ↪py:71, in plot(data, kind, **kwargs)
     69         kwargs["ax"] = getattr(ax, "left_ax", ax)
     70 plot_obj = PLOT_CLASSES[kind](data, **kwargs)
---> 71 plot_obj.generate()
     72 plot_obj.draw()
     73 return plot_obj.result

File␣
 ↪~\Desktop\sample_project_1\env\lib\site-packages\pandas\plotting\_matplotlib\core.
 ↪py:450, in MPLPlot.generate(self)
    448 def generate(self) -> None:
    449     self._args_adjust()
--> 450     self._compute_plot_data()
    451     self._setup_subplots()
    452     self._make_plot()

File␣
 ↪~\Desktop\sample_project_1\env\lib\site-packages\pandas\plotting\_matplotlib\core.
 ↪py:635, in MPLPlot._compute_plot_data(self)
    633 # no non-numeric frames or series allowed
    634 if is_empty:
--> 635     raise TypeError("no numeric data to plot")
    637 self.data = numeric_data.apply(self._convert_to_ndarray)

TypeError: no numeric data to plot
```

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.

[194]:
```
# Remove the symbols, the final two numbers from the 'Price' column and convert␣
 ↪it to numbers
car_sales['Price']=car_sales['Price'].str.replace('[\$\,]|\.\d*', '').
 ↪astype(int)
car_sales
```

```
C:\Users\LALIT\AppData\Local\Temp\ipykernel_6276\1454954990.py:2: FutureWarning:
The default value of regex will change from True to False in a future version.
  car_sales['Price']=car_sales['Price'].str.replace('[\$\,]|\.\d*',
'').astype(int)
```

[194]:

	Make	Colour	Odometer (KM)	Doors	Price
0	Toyota	White	150043	4	4000
1	Honda	Red	87899	4	5000
2	Toyota	Blue	32549	3	7000
3	BMW	Black	11179	5	22000
4	Nissan	White	213095	4	3500
5	Toyota	Green	99213	4	4500
6	Honda	Blue	45698	4	7500
7	Honda	Blue	54738	4	7000
8	Toyota	White	60000	4	6250
9	Nissan	White	31600	4	9700

[202]:
```
# Add a column called 'Total Sales' to car_sales which cumulatively adds the␣
 ↪'Price' column

car_sales['Total_sales ($)']=car_sales['Price'].cumsum()

# Add a column called 'Sale Date' which lists a series of successive dates␣
 ↪starting from today (your today)
car_sales['Sale Date']=pd.date_range('2023-03-01', periods=len(car_sales))

# View the car_sales DataFrame
car_sales
```

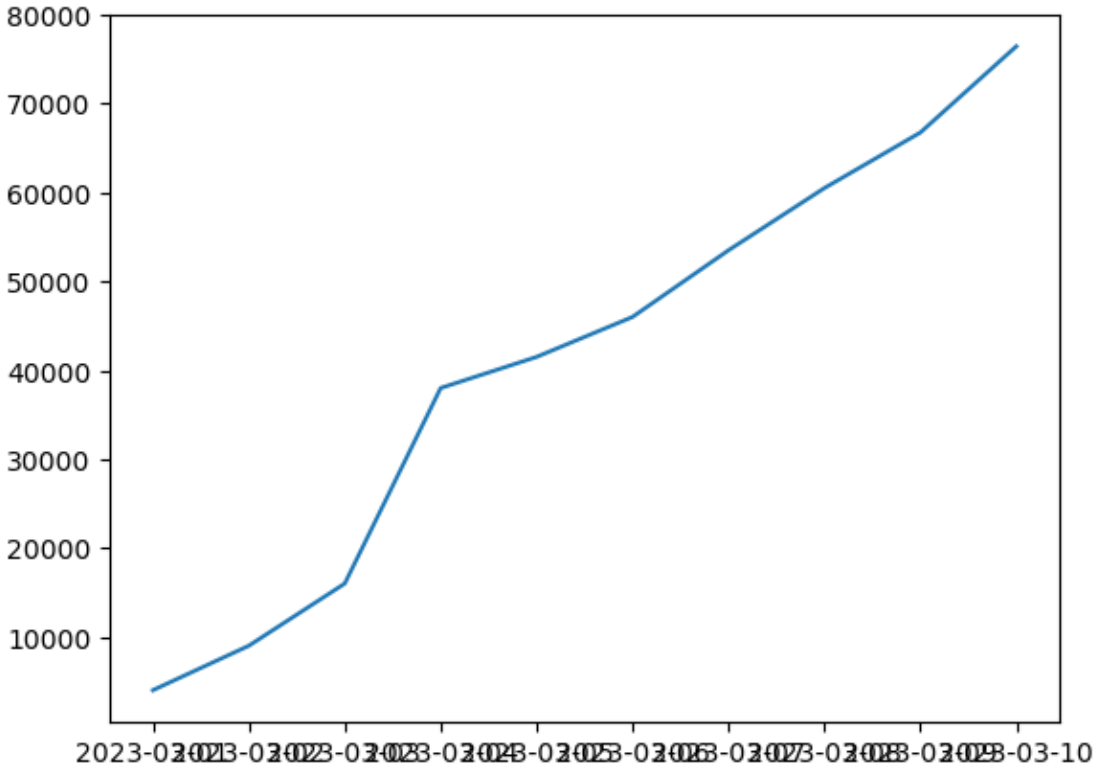
[202]:

	Make	Colour	Odometer (KM)	Doors	Price	Total_sales ($)	Sale Date
0	Toyota	White	150043	4	4000	4000	2023-03-01
1	Honda	Red	87899	4	5000	9000	2023-03-02
2	Toyota	Blue	32549	3	7000	16000	2023-03-03
3	BMW	Black	11179	5	22000	38000	2023-03-04
4	Nissan	White	213095	4	3500	41500	2023-03-05
5	Toyota	Green	99213	4	4500	46000	2023-03-06
6	Honda	Blue	45698	4	7500	53500	2023-03-07

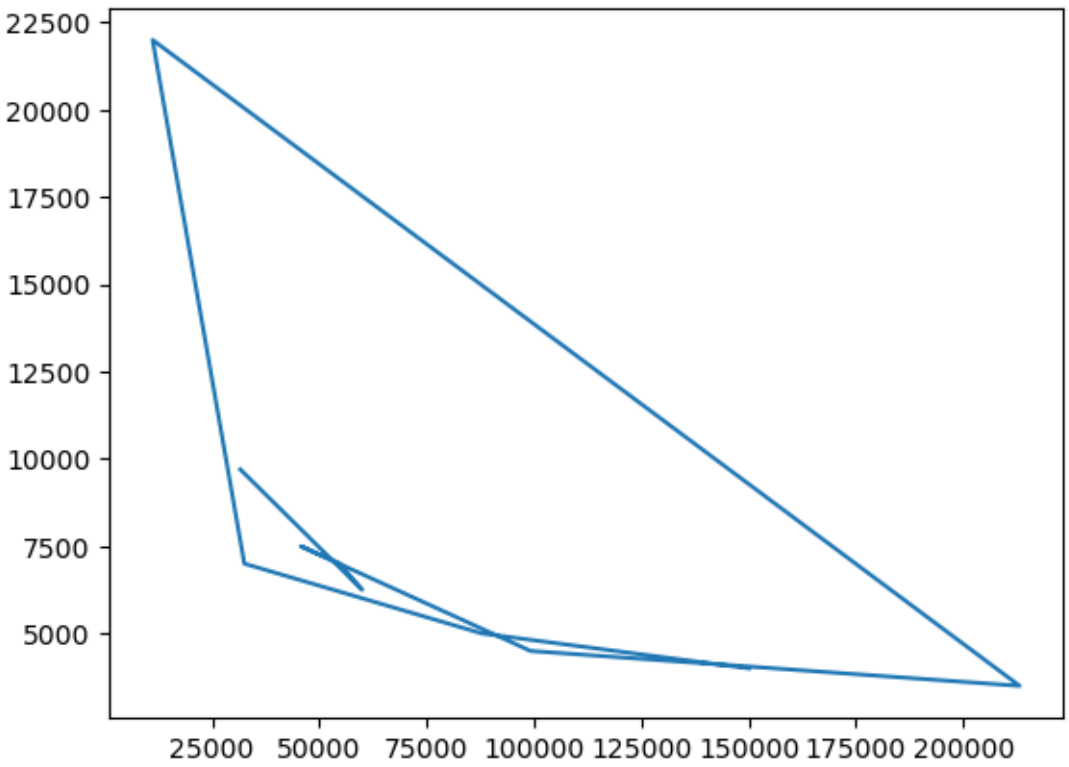
```
7   Honda   Blue          54738      4   7000          60500 2023-03-08
8  Toyota  White          60000      4   6250          66750 2023-03-09
9  Nissan  White          31600      4   9700          76450 2023-03-10
```

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

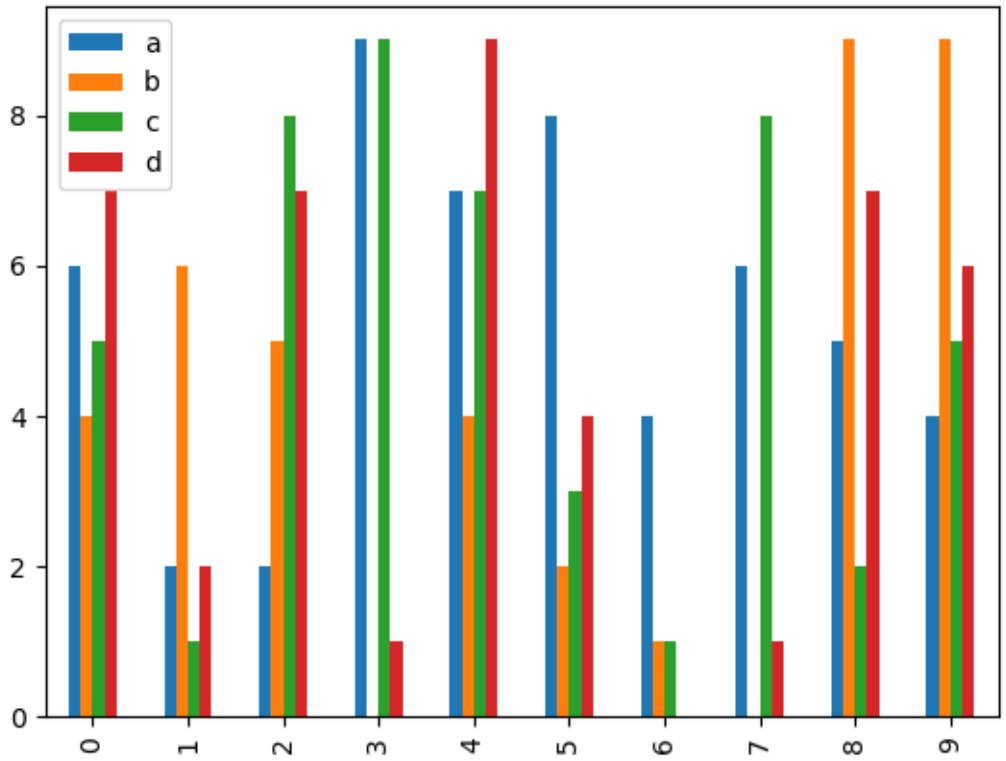
[205]:
```
# Use the plot() function to plot the 'Sale Date' column versus the 'Total␣
↪Sales' column
plt.plot(car_sales['Sale Date'], car_sales['Total_sales ($)']);
```



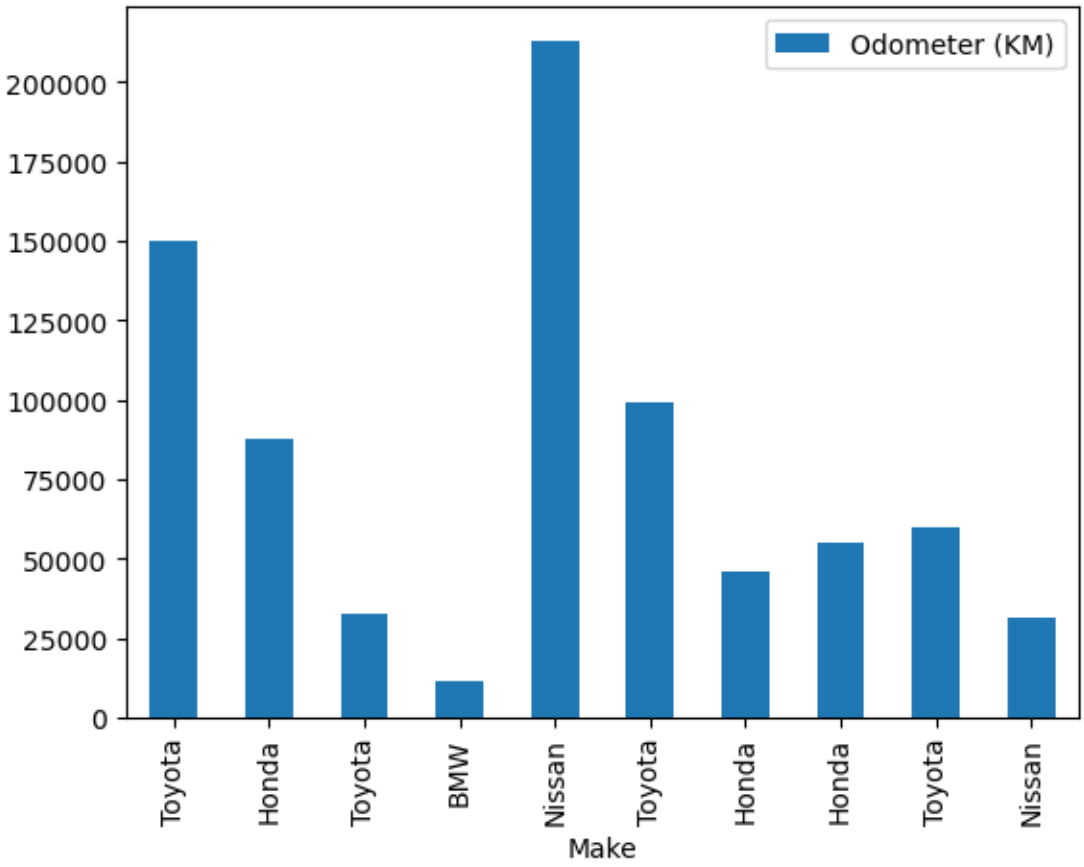
[209]:
```
# Convert the 'Price' column to the integers
#car_sales.dtypes

# Create a scatter plot of the 'Odometer (KM)' and 'Price' column using the␣
↪plot() function
plt.plot(car_sales['Odometer (KM)'],car_sales['Price']);
```



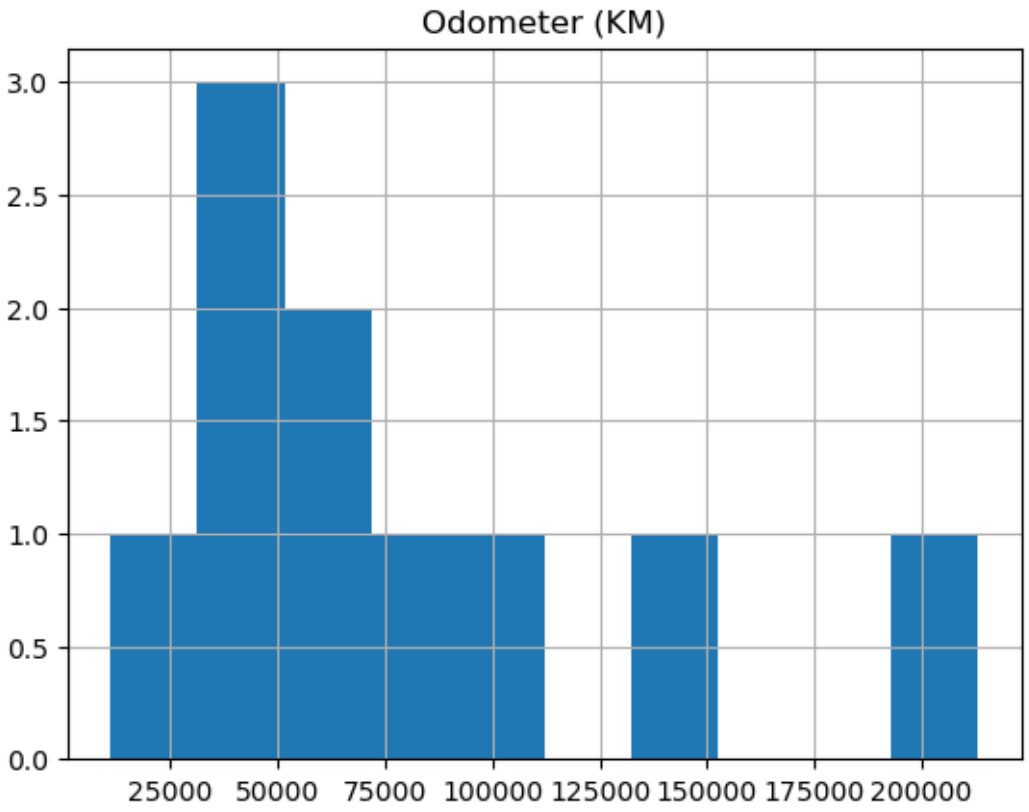
[227]:
```
# Create a NumPy array of random numbers of size (10, 4) and save it to X
X=np.random.randint(10,size=(10,4))
X
# Turn the NumPy array X into a DataFrame with columns called ['a', 'b', 'c',␣
↪'d']
df=pd.DataFrame(X,columns=['a', 'b', 'c', 'd'] )

# Create a bar graph of the DataFrame
df.plot(kind='bar');
```



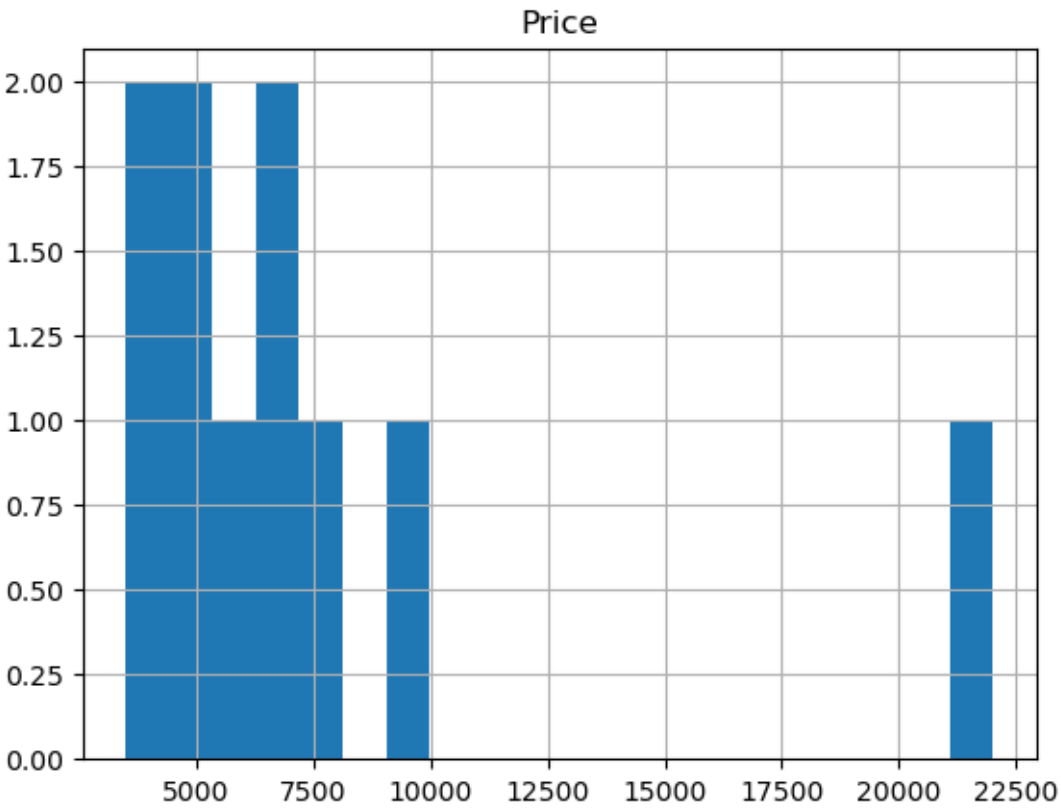
[239]:
```
# Create a bar graph of the 'Make' and 'Odometer (KM)' columns in the car_sales␣
↪DataFrame
car_sales.plot(x="Make", y="Odometer (KM)", kind="bar");
```



[242]:
```
# Create a histogram of the 'Odometer (KM)' column
car_sales.hist('Odometer (KM)');
```



[245]:
```
# Create a histogram of the 'Price' column with 20 bins
car_sales.hist('Price', bins=20);
```



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

Let's try using a different dataset.

[247]:
```
# Import "../data/heart-disease.csv" and save it to the variable "heart_disease"
heart_disease=pd.read_csv(r'D:\Study\Complete Machine Learning & Data Science␣
↪Bootcamp 2022\08 - Matplotlib Plotting and Data Visualization\013␣
↪heart-disease.csv')
```

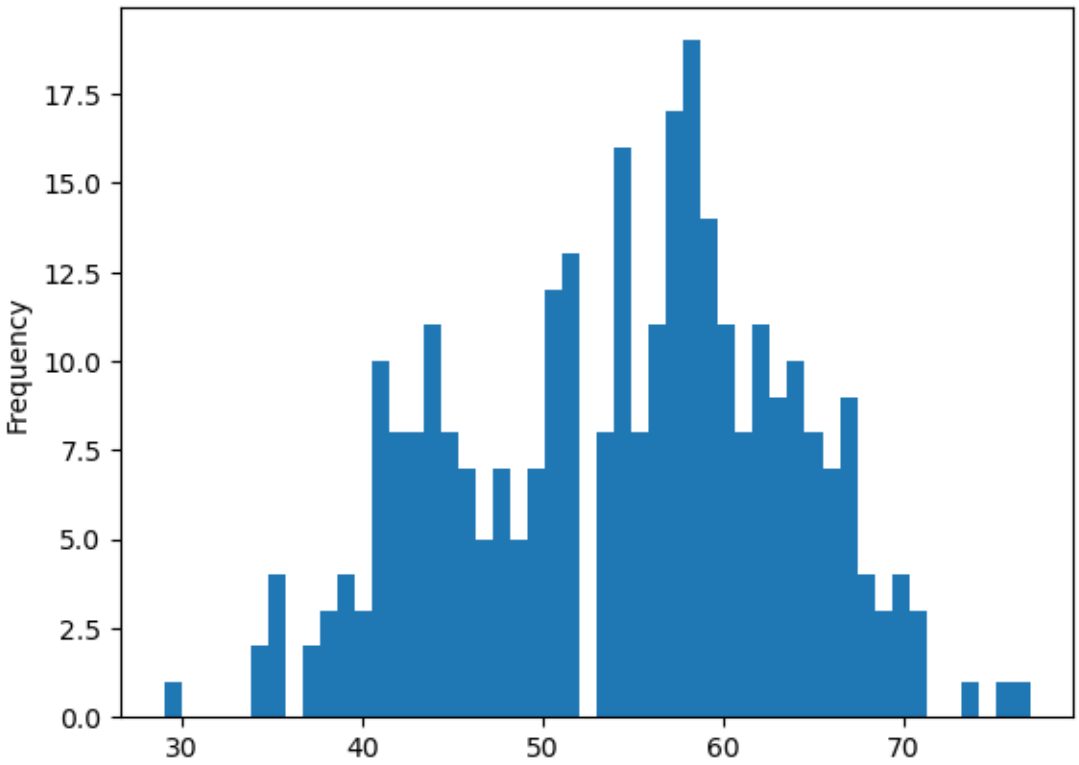
[249]:
```
# View the first 10 rows of the heart_disease DataFrame
heart_disease.head(10)
```

[249]:

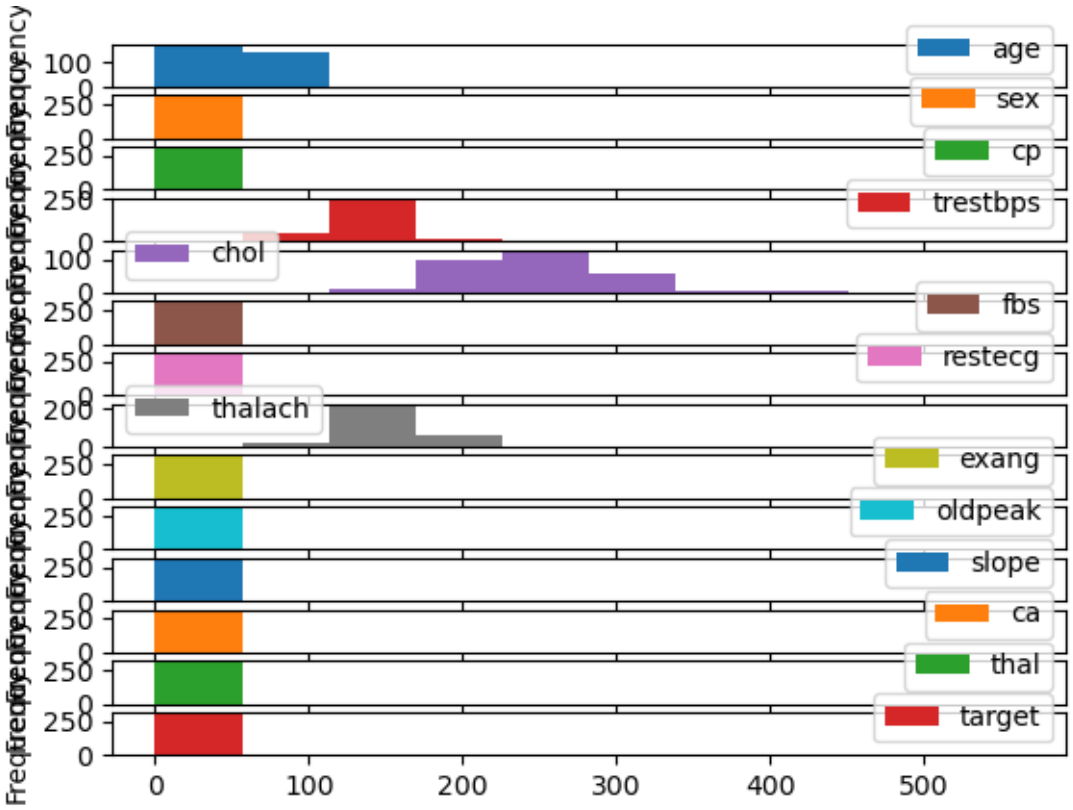
	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	\
0	63	1	3	145	233	1	0	150	0	2.3	0	
1	37	1	2	130	250	0	1	187	0	3.5	0	
2	41	0	1	130	204	0	0	172	0	1.4	2	
3	56	1	1	120	236	0	1	178	0	0.8	2	
4	57	0	0	120	354	0	1	163	1	0.6	2	
5	57	1	0	140	192	0	1	148	0	0.4	1	
6	56	0	1	140	294	0	0	153	0	1.3	1	
7	44	1	1	120	263	0	1	173	0	0.0	2	
8	52	1	2	172	199	1	1	162	0	0.5	2	

```
9   57    1   2        150   168    0         1       174       0       1.6       2

    ca  thal  target
0    0     1       1
1    0     2       1
2    0     2       1
3    0     2       1
4    0     2       1
5    0     1       1
6    0     2       1
7    0     3       1
8    0     3       1
9    0     2       1
```

[251]:
```
# Create a histogram of the "age" column with 50 bins
heart_disease['age'].plot(kind='hist', bins=50);
```

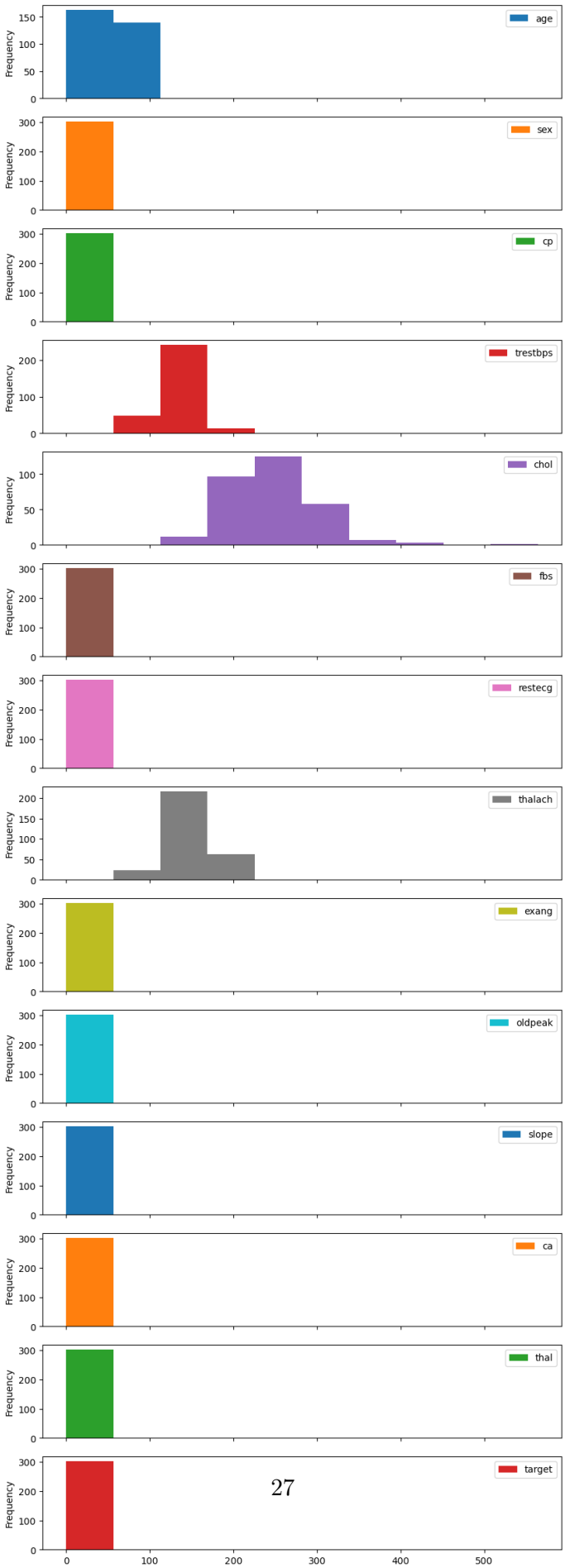


[252]:
```
# Call plot.hist() on the heart_disease DataFrame and toggle the
# "subplots" parameter to True
heart_disease.plot.hist(subplots=True);
```



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

[253]:
```
# Call the same line of code from above except change the "figsize" parameter
# to be (10, 30)
heart_disease.plot.hist(subplots=True, figsize=(10,30));
```

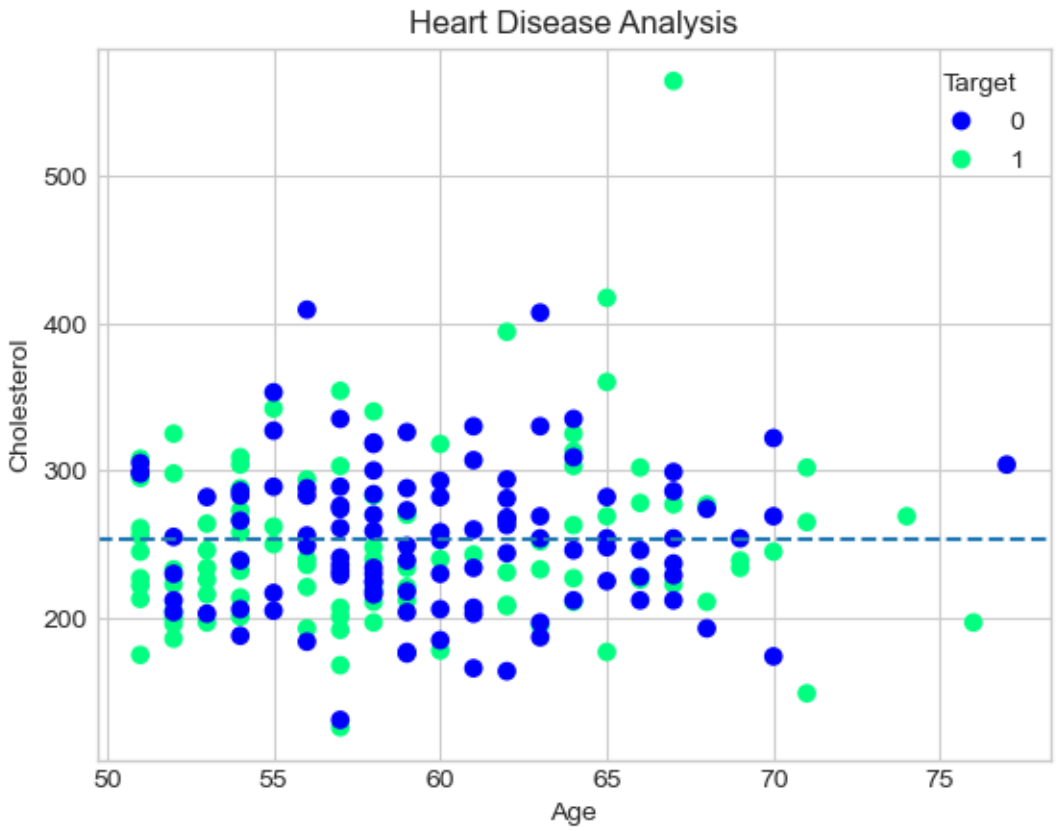

age
sex
cp
trestbps
chol
fbs
restecg
thalach
exang
oldpeak
slope
ca
thal
target
Frequency
0
50
100
150
200
300
400
500

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:

[277]:
```
# 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
over_50=heart_disease[heart_disease['age']>50];

# Create the plot
fig, ax=plt.subplots();

# Plot the data
scatter= ax.scatter(over_50['age'],
                    over_50['chol'],
                    c=over_50["target"],
                   cmap="winter");

# Customize the plot
ax.set(title='Heart Disease Analysis',
      xlabel= 'Age',
      ylabel='Cholesterol')
ax.legend(*scatter.legend_elements(),  title="Target")

# Add a meanline
ax.axhline((over_50['chol']).mean(), linestyle='--');
```



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

[273]:
```
# Check what styles are available under plt
plt.style.available
```

[273]:
```
['Solarize_Light2',
 '_classic_test_patch',
 '_mpl-gallery',
 '_mpl-gallery-nogrid',
 'bmh',
 'classic',
 'dark_background',
 'fast',
 'fivethirtyeight',
 'ggplot',
 'grayscale',
 'seaborn-v0_8',
 'seaborn-v0_8-bright',
 'seaborn-v0_8-colorblind',
 'seaborn-v0_8-dark',
 'seaborn-v0_8-dark-palette',
```

```
 'seaborn-v0_8-darkgrid',
 'seaborn-v0_8-deep',
 'seaborn-v0_8-muted',
 'seaborn-v0_8-notebook',
 'seaborn-v0_8-paper',
 'seaborn-v0_8-pastel',
 'seaborn-v0_8-poster',
 'seaborn-v0_8-talk',
 'seaborn-v0_8-ticks',
 'seaborn-v0_8-white',
 'seaborn-v0_8-whitegrid',
 'tableau-colorblind10']
```

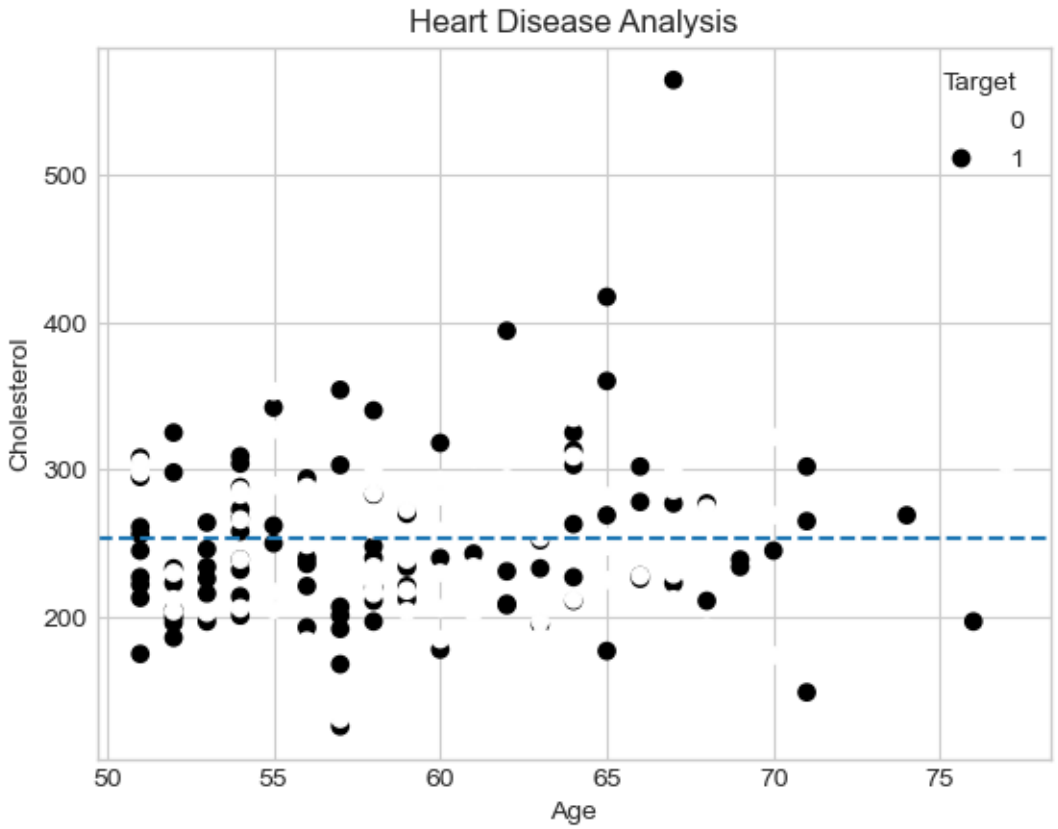
[275]:
```
# Change the style to use "seaborn-whitegrid"
plt.style.use("seaborn-whitegrid");
```

```
C:\Users\LALIT\AppData\Local\Temp\ipykernel_6276\560332574.py:2:
MatplotlibDeprecationWarning: The seaborn styles shipped by Matplotlib are
deprecated since 3.6, as they no longer correspond to the styles shipped by
seaborn. However, they will remain available as 'seaborn-v0_8-<style>'.
Alternatively, directly use the seaborn API instead.
  plt.style.use("seaborn-whitegrid");
```

[281]:
```
# Reproduce the same figure as above with the "seaborn-whitegrid" style
# 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
over_50=heart_disease[heart_disease['age']>50];

# Create the plot
fig, ax=plt.subplots();

# Plot the data
scatter= ax.scatter(over_50['age'],
                    over_50['chol'],
                    c=over_50["target"]);

# Customize the plot
ax.set(title='Heart Disease Analysis',
       xlabel= 'Age',
       ylabel='Cholesterol')
ax.legend(*scatter.legend_elements(),  title="Target")

# Add a meanline
ax.axhline((over_50['chol']).mean(), linestyle='--');
```

```
plt.style.use("seaborn-whitegrid");
```

```
C:\Users\LALIT\AppData\Local\Temp\ipykernel_6276\174329778.py:26:
MatplotlibDeprecationWarning: The seaborn styles shipped by Matplotlib are
deprecated since 3.6, as they no longer correspond to the styles shipped by
seaborn. However, they will remain available as 'seaborn-v0_8-<style>'.
Alternatively, directly use the seaborn API instead.
  plt.style.use("seaborn-whitegrid");
```

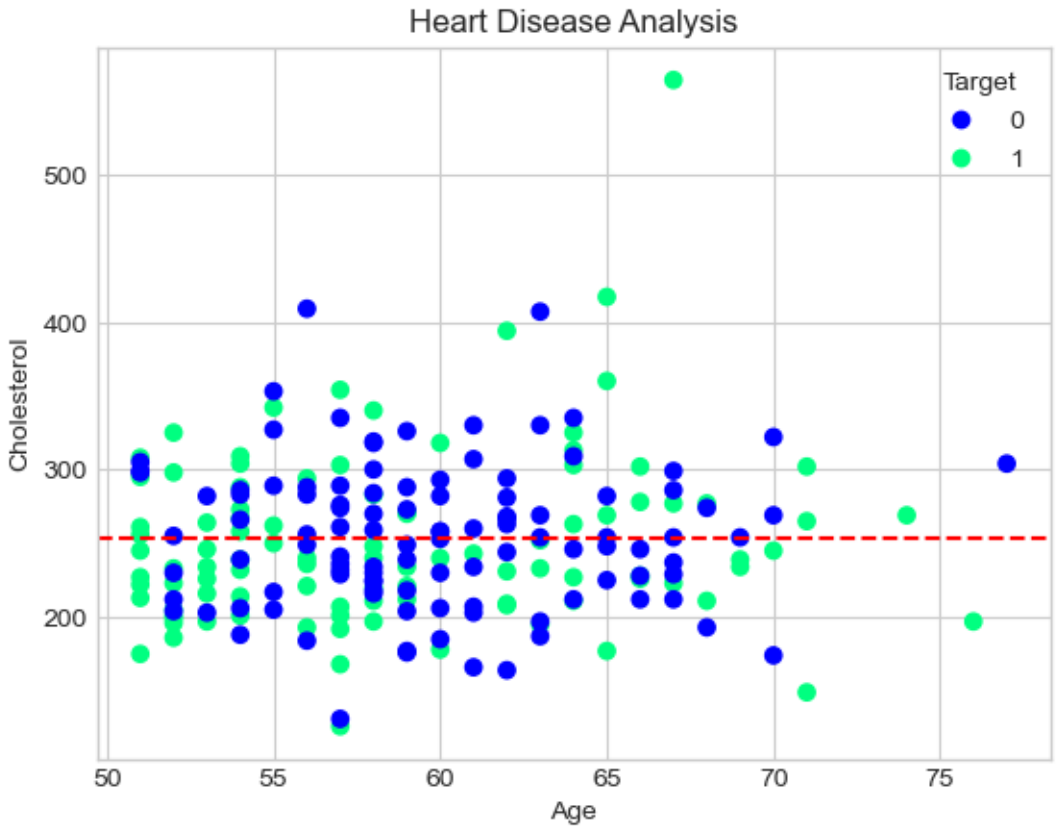


Wonderful, you've changed the style of the plots and the figure is looking different but the dots aren't a very good colour.

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

[285]:
```
# 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 DataFrame with patients over 50 years old
```

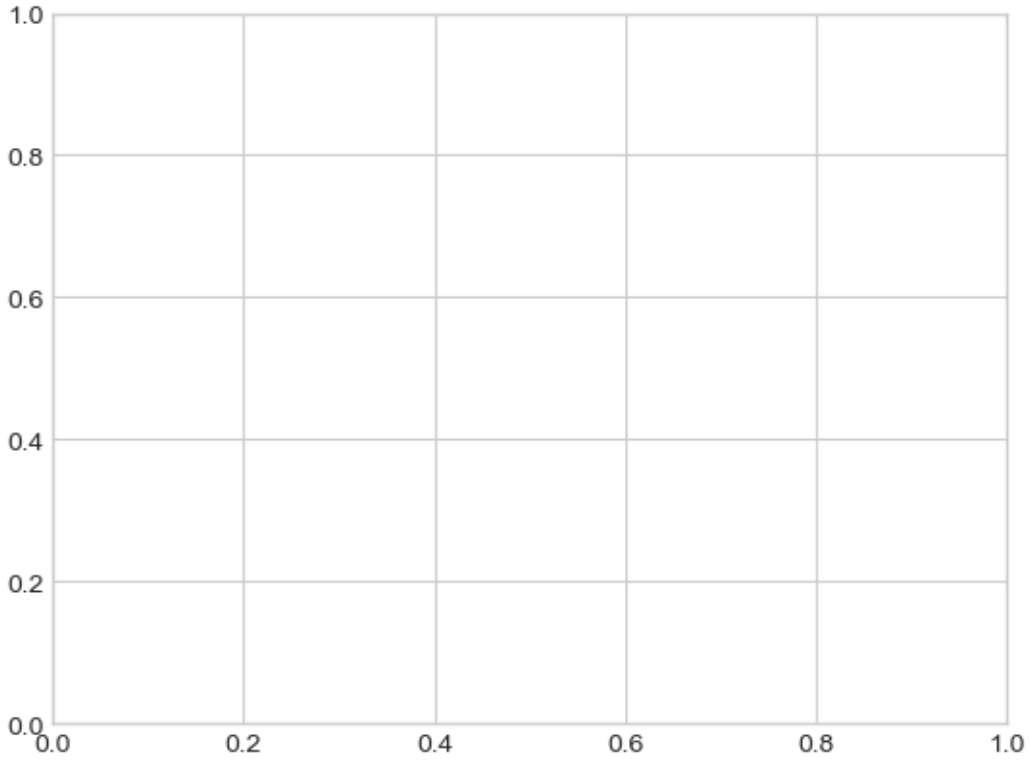
```
over_50=heart_disease[heart_disease['age']>50];

# Create the plot
fig, ax=plt.subplots();

# Plot the data
scatter= ax.scatter(over_50['age'],
                    over_50['chol'],
                    c=over_50["target"],
                   cmap='winter');

# Customize the plot
ax.set(title='Heart Disease Analysis',
      xlabel= 'Age',
      ylabel='Cholesterol')
ax.legend(*scatter.legend_elements(),  title="Target")

# Add a meanline
ax.axhline(over_50['chol'].mean(), linestyle='--',
          color='red');
```



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

[287]:
```
# Save the current figure using savefig(), the file name can be anything you␣
 ↪want
fig.savefig(r"D:\Study\Complete Machine Learning & Data Science Bootcamp␣
 ↪2022\08 - Matplotlib Plotting and Data␣
 ↪Visualization\matplotlib-heart-disease-chol-age-plot-saved.png")
```

[288]:
```
# Reset the figure by calling plt.subplots()
fig, ax = plt.subplots()
```



By: Abhishek Kumar