HyperionDev

Data Visualisation – Python Libraries

Task

Visit our website

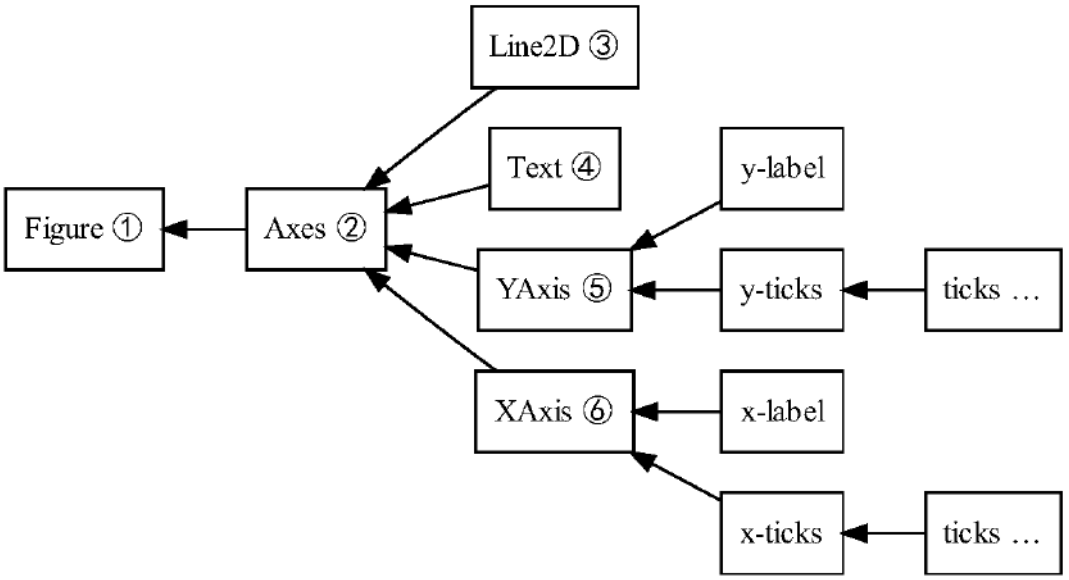
Introduction

There are many packages in Python that allow one to perform powerful data analysis. In this task, you will learn about matplotlib and seaborn, and practice using them to visualise data.

Matplotlib

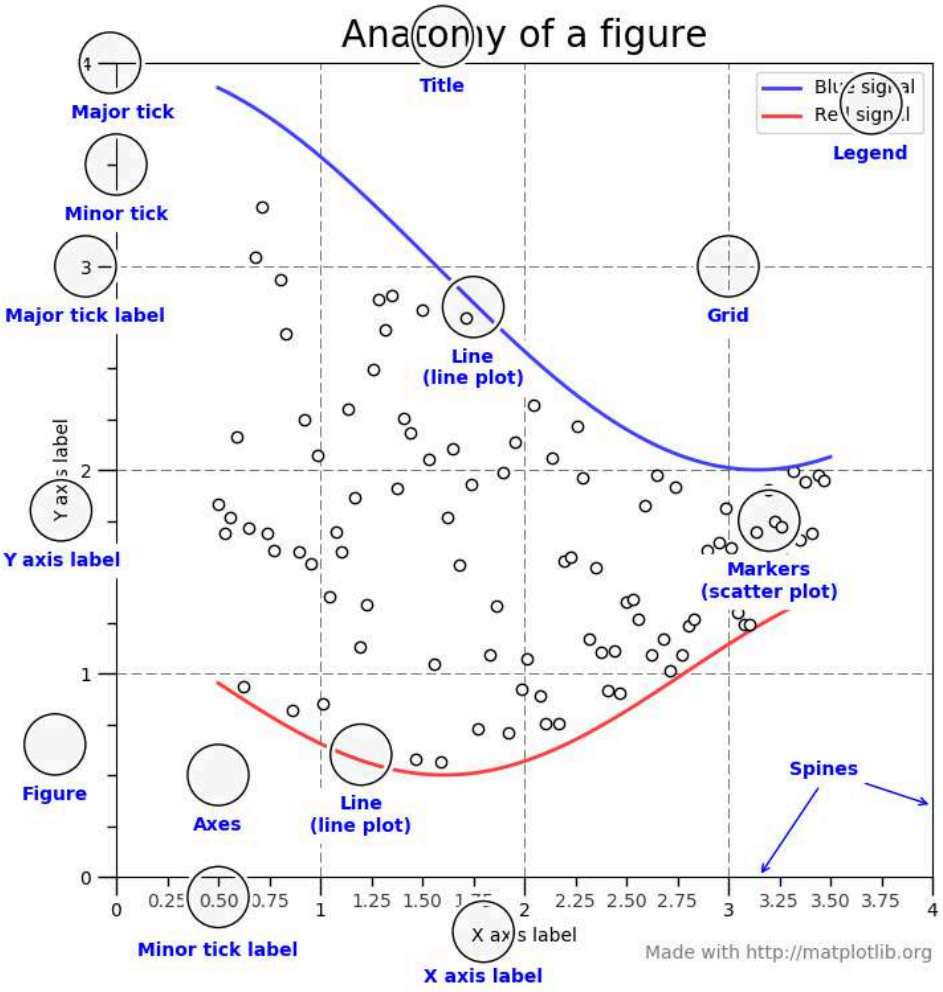
Matplotlib is a powerful visualisation tool that creates both exploratory and deployment visualisations. In this task, you will learn about how matplotlib is structured and how to create a variety of plots.

Matplotlib is organised into a hierarchy. At the top of the hierarchy, there is a module called a pyplot. We create a pyplot to create a figure, which keeps track of all the child axes, artists (titles, figure legends, etc.), and the canvas.



Matplotlib hierarchy (Hunter, 2007)

An artist in matplotlib is a fundamental object representing any visual element within a plot. This encompasses a wide range of components, from the overall figure and its axes to specific elements like lines, text labels, and shapes. When a plot is generated, these artist objects are rendered onto the canvas, creating the final visualisation. It's important to note that most artists are linked to a particular axis and cannot be shared or moved between different axes (Hunter, 2007).



Anatomy of a matplotlib figure (Hunter, 2007)

All of the plotting functions expect `np.array` or `np.ma.masked_array` as input, so it is best to convert any pandas (or similar array-like data structures) to arrays before using them with matplotlib.

Installing matplotlib

First, let's check if you have matplotlib installed. Open up your terminal, input the following, and then hit enter:

```
pip3 show matplotlib
```

Before proceeding with the installation, remember to create and activate a virtual environment to manage your project dependencies.

Then, use the following command to install matplotlib:

```
python -m pip install -U matplotlib
```

For further information on installing Matplotlib, see the **official documentation**.

Creating a figure

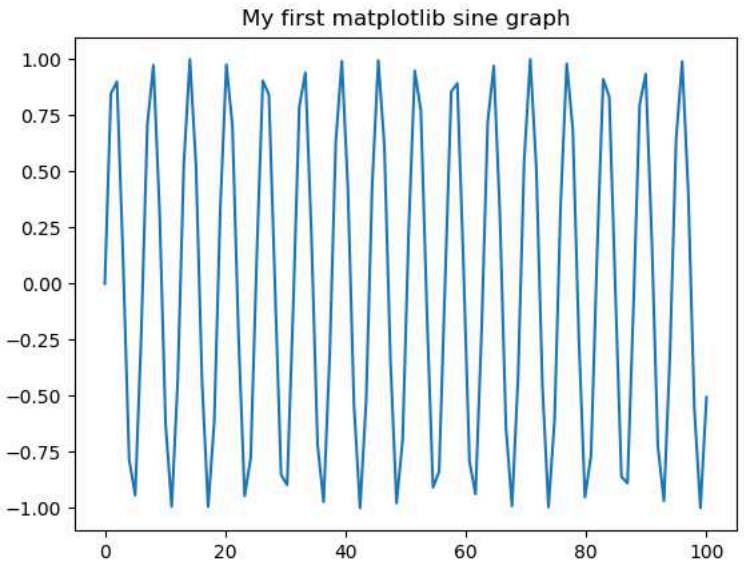
Remember to import packages or libraries at the top of your Python file:

```
import matplotlib.pyplot as plt
import numpy as np
```

We can create a quick dataset using NumPy and visualise the data using matplotlib:

```
# Prepare the data
x = np.linspace(0, 100, 100) # x axis
y = np.sin(x) # y values

# Plot the data
plt.plot(x, y)

# Create a title
plt.title("My first matplotlib sine graph")

# Show the plot
plt.show()
```

And if you run the program, you will get something like this:



Try to play around with the sine graph code provided in the example file to get a better understanding.

Extra resource

For more information about working with matplotlib, please consult the fourth chapter ("**Visualization with matplotlib**") in the book ***Python Data Science Handbook*** (2016), by Jake VanderPlas. You can also explore the **matplotlib webpage**.

Now that you are familiar with some of the basic concepts and techniques associated with creating data visualisations with matplotlib, we will begin to explore creating some more advanced visualisations.

Stacked area chart

Below is an example of a stacked area chart. These graphs are often used to compare multiple variables that change over time.

Example of a stacked area graph (The Data Visualisation Catalogue, n.d.)

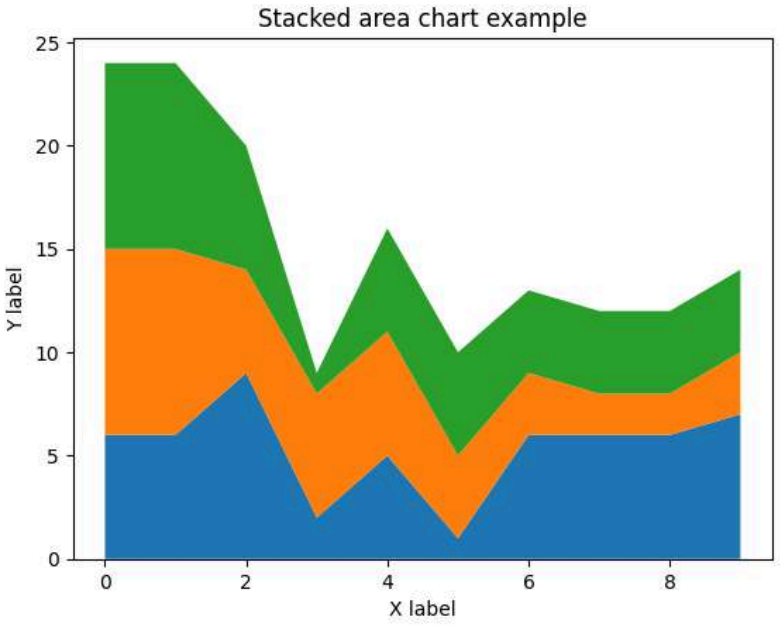
Stacked area graphs can be easily created with matplotlib, as shown in the example below:

```
import numpy as np
import matplotlib.pyplot as plt

# Generate random data
group_one = np.random.randint(1,10,10)
group_two = np.random.randint(1,10,10)
group_three = np.random.randint(1,10,10)

# Create the stacked area
y = np.row_stack((group_one, group_two, group_three))
x = np.arange(10)
y1, y2, y3 = (group_one, group_two, group_three)

fig, ax = plt.subplots()
ax.stackplot(x,y)

# Labels
plt.title("Stacked area chart example")
plt.xlabel("X label")
plt.ylabel("Y label")
plt.show()
```

In the code above, we have created three arrays (`group_one`, `group_two`, and `group_three`) that contain the data that we want to represent on our stacked area graph. The instruction `y = np.row_stack((group_one,group_two,group_three))` takes a sequence of arrays and stacks them vertically to make a single array. The code `x = np.arange(10)` returns an array with evenly spaced elements as per the interval.

With a stacked area graph, we want to be able to view different data together in the same graph. To do this we use subplots. A subplot is defined as follows: “Groups of smaller axes that can exist together within a single figure. These subplots might be insets, grids of plots, or other more complicated layouts” (VanderPlas, 2016). Notice that we use the **`plt.subplots()`** function. This function returns a figure, which is the top-level container for all the plot elements, and an array of axes objects. Furthermore, the code `ax.stackplot(x,y)` creates a stackplot, which is generated by plotting different datasets vertically on top of one another rather than overlapping with one another.

When you run the program, you will get something like this:



Keep the **matplotlib documentation** in mind when you get stuck or want to try a new type of visualisation.

Plotting with pandas

Although pandas is not a data visualisation package, it can be used to create basic plots to simplify the pipeline during data exploration. The pandas **`plot()` method** uses matplotlib as the default backend package and can be used in conjunction with matplotlib.

Seaborn

Seaborn is a data visualisation library that has been built on top of matplotlib. It's well suited for statistical visualisations and is designed to work with pandas data structures. It also allows you to switch easily between different types of graphs to look at the data from different angles. You can explore the **seaborn documentation** now or later when we introduce seaborn in more detail.

Some commonly used seaborn plots include:

- **`histplot()`**
- **`barplot()`**
- **`boxplot()`**

And there are many others! You will get accustomed to many of these methods during the course of this bootcamp.

Install seaborn via your terminal using pip:

```
pip install -U seaborn
```

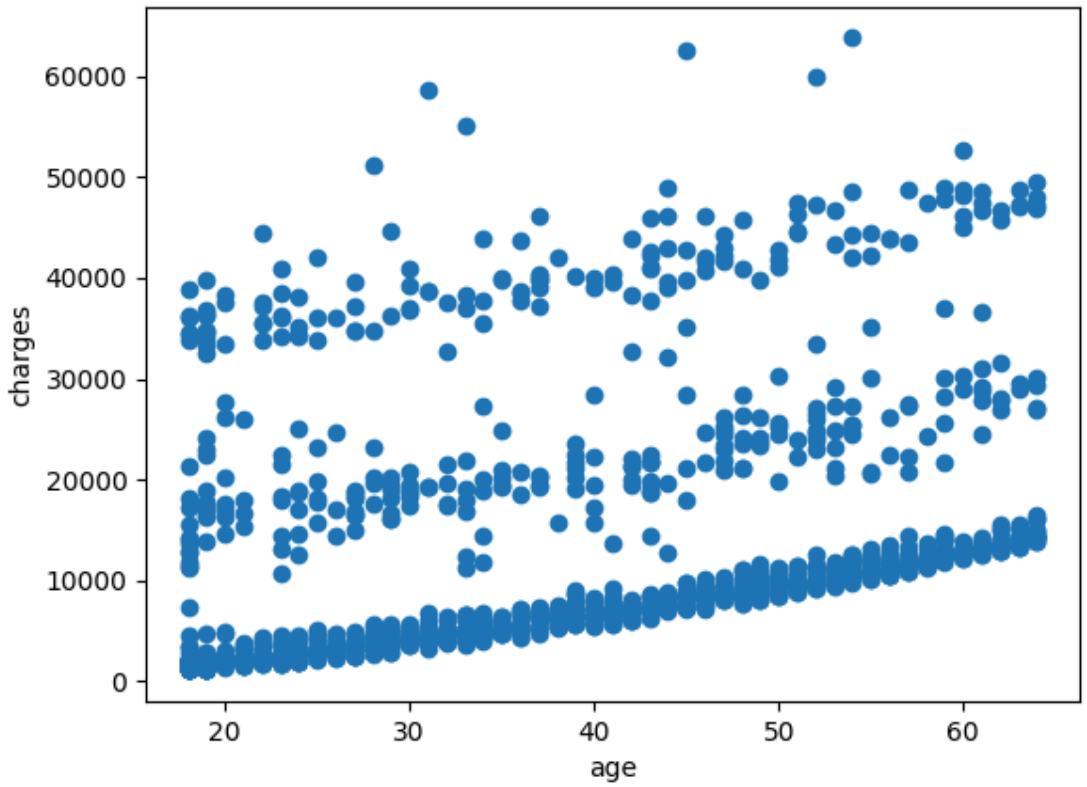
Let's say that we are reading insurance data that contains a column for age and a column for the insurance charge. We would like to understand the relationship that exists between these two columns.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load data
ins_df = pd.read_csv('insurance.csv')
```

In matplotlib, the `scatter()` method would be most appropriate. This can be achieved as follows:

```
# Plot scatterplot
plt.figure()
plt.scatter(ins_df['age'], ins_df['charges'])
plt.xlabel('age')
plt.ylabel('charges')
plt.show()
plt.close()
```

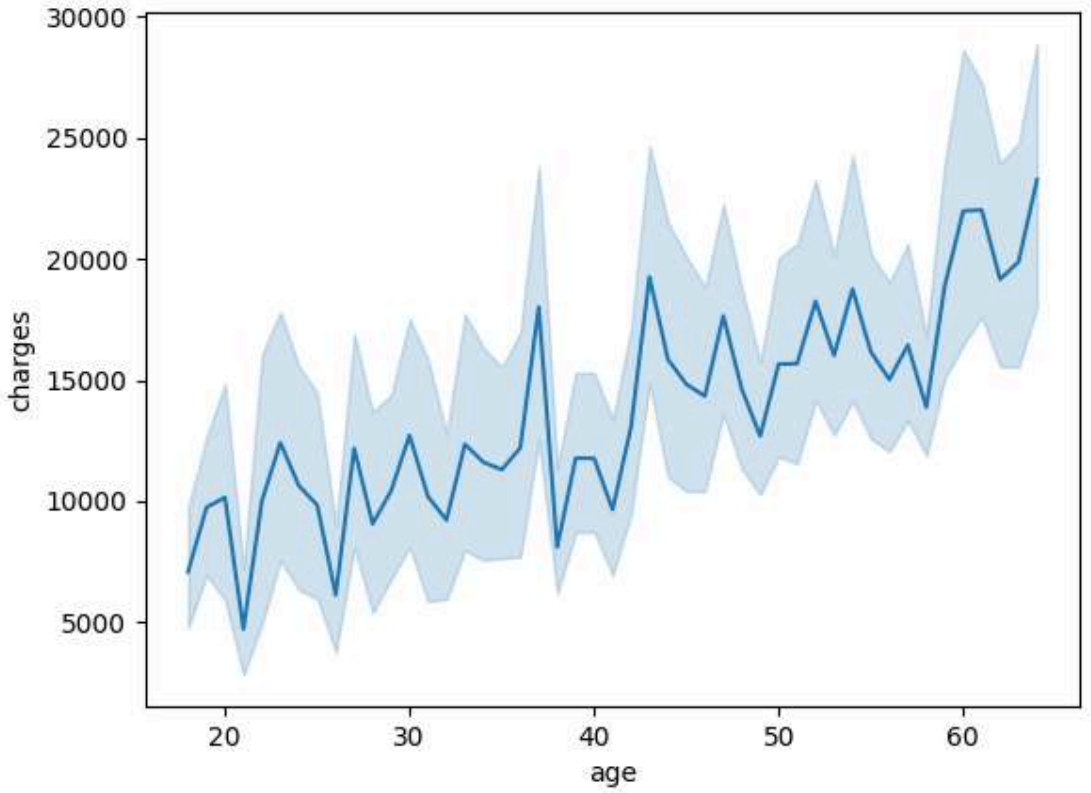
The graph would look something like this:



You can get a general sense of this data, but it may take time to understand the overall trend. In seaborn, the `lineplot()` method automatically plots averages and standard deviations for ease of reading. To do this, use the following:

```
# Plot lineplot
plt.figure()
sns.lineplot(x='age', y='charges', data=ins_df)
plt.show()
plt.savefig('sns_lineplot.png') # save as a png image file
plt.close()
```

You will end up with something that looks like this:



This makes it a lot easier to see the overall trend that exists in the data: the higher your age, the larger your insurance charges.

Multi-plot grids

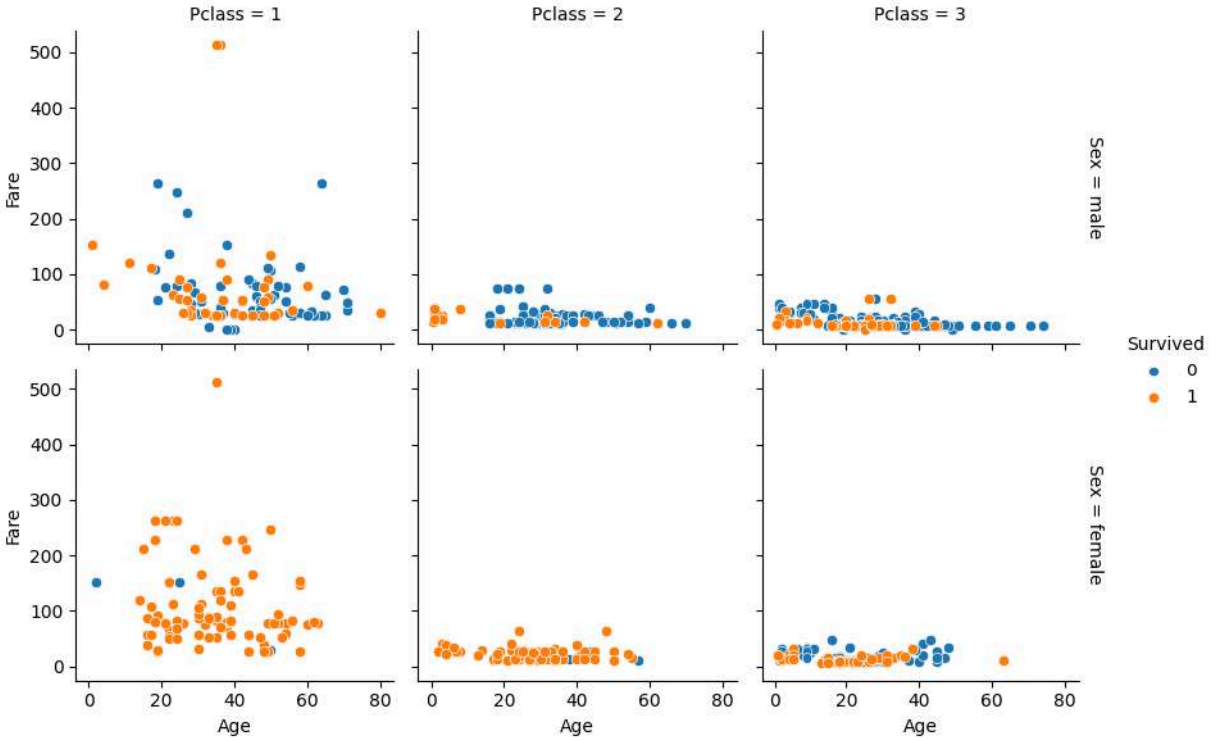
More often than not, we have to visualise relationships between four or more dimensions. As you can imagine, thinking in four dimensions can be difficult. Five dimensions? Even more so. Imagine a dataset containing 20 columns: that's 20 dimensions that we have to visualise! Luckily, there are workarounds for this.

Seaborn contains a `FacetGrid`, which is essentially a grid containing multiple plots. While this can easily be done in matplotlib, the interface presented in seaborn is much easier to use.

Let's use the Titanic dataset as an example here. We have multiple factors contributing to whether someone survived or not. Let's take a look at the relationship between a person's sex, passenger class, age, fare, and whether they survived or not:

```
import pandas as pd
import seaborn as sns

# Load data
titanic_df = pd.read_csv('Titanic.csv')

# Plot FacetGrid plot
plt.figure()
fg = sns.FacetGrid(titanic_df, row="Sex", col="Pclass", hue="Survived",
margin_titles=True)
fg.map(sns.scatterplot, "Age", "Fare")
fg.add_legend()
plt.show()
plt.close()
```

This produces the following diagram:



This tells us a lot. The first (and most obvious) observation is that Pclass 1 has the most expensive fare. Also, by comparing the number of orange and blue dots in the first and second rows of the graphs, we see that mostly females survived, especially in Pclass 1 and 2.

What other insights can you observe from these plots? Are there any outliers? Why do you think there is variability in the fare for each class?

Correlation heatmaps

A correlation matrix is a very important tool to have as a data scientist. This is quite possibly the best way to get a basic understanding of all of the features in a single plot.

Before we talk about a correlation matrix, let's talk about correlation, and more specifically, the correlation coefficient. In short, the correlation coefficient is a single number that tells you the relationship between two variables. If one variable increases consistently as another variable increases, this means that they have a positive correlation. If one variable decreases as another increases, this is a negative correlation. If two variables have no effect on each other, this means that they have zero correlation.

The correlation coefficient has a value between -1 and 1. A correlation of 1 is basically a straight line going upwards from left to right. A correlation of -1 is a straight line going down from left to right.

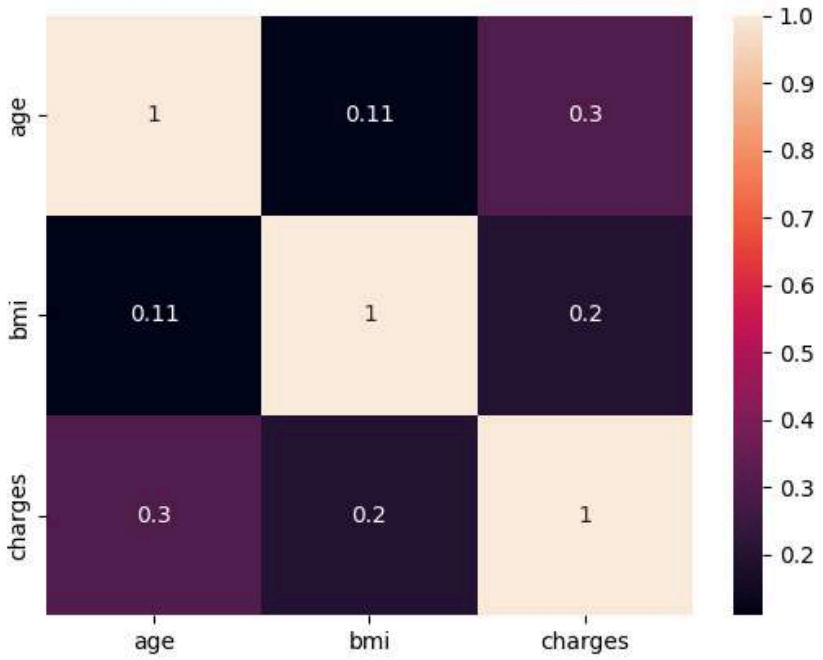
Now that we understand correlation, what is a correlation matrix? It's essentially a 2D square matrix, which means that there are as many rows as there are columns. The value at position (0, 1) in the correlation matrix will show you the correlation coefficient between features 0 and 1.

The correlation matrix can be viewed as a set of numbers, but numbers have never been very easy to read. The best way to view this matrix is with a heatmap. Thankfully, seaborn (as always) makes this easy.

Let's take a look at the insurance dataset. Specifically, we want to see the relationship between age, BMI, and insurance charges:

```
import pandas as pd
import seaborn as sns

# Load data
insurance_df = pd.read_csv('insurance.csv')

# Select columns of interest
age_bmi_charges = insurance_df[['age', 'bmi', 'charges']]

# Plot correlation matrix
plt.figure()
corr_coeff_mat = age_bmi_charges.corr()
sns.heatmap(corr_coeff_mat, annot=True)
plt.show()
plt.close()
```

This produces the following graph:



What can we tell from this graph? Well, keep in mind that a correlation close to zero means that there's little to be seen in terms of a relationship and values closer to one indicate a strong relationship. The range of values from zero to one are assigned different shades on the heatmap. A more purple colour indicates a weak positive relationship and red to beige indicates a stronger positive relationship. In this example, the relationships between features are weak as the colour blocks showing relationships between different features are all shades of purple. Nevertheless, there are some observations that are interesting. For example, being older has a slightly higher impact on insurance charges than your BMI.

Extra resource

You can find plenty of Python packages related to data visualisation with a quick search. For geographic or map visualisations you can use **geoplotlib**, or you can create a **word cloud** for fun.

Take note

First, read and run the **example files** provided. Feel free to write and run your own example code before doing the auto-graded tasks to become more comfortable with the concepts covered in this task.

Take note

The tasks below are **auto-graded**. An auto-graded task still counts towards your progression and graduation. Give it your best attempt and submit it when you are ready.

When you select “Request Review”, the task is automatically complete, and you do not need to wait for it to be reviewed by a mentor.

You will then receive an email with a link to a model answer, as well as an overview of the approach taken to reach this answer.

Take some time to review and compare your work against the model answer. This exercise will help solidify your understanding and provide an opportunity for reflection on how to apply these concepts in future projects.

In the same email, you will also receive a link to a survey, which you can use to self-assess your submission.

Once you’ve done that, feel free to progress to the next task.

Auto-graded task 1

- Open the Jupyter notebook named **data_viz_task.ipynb**.
- Explore the metadata for the **93CARS dataset**.

- Generate the following graphs from the **Cars93.csv** (93CARS) dataset. Then, answer the accompanying questions in the markdown cells in the notebook:

    i. A box plot for the revs per mile for the Audi, Hyundai, Suzuki, and Toyota car manufacturers.

        1. Which of these manufacturers has the car with the highest revs per mile?

    ii. A histogram of MPG in the city. On the same axis, show a histogram of MPG on the highway.

        1. Is it generally more fuel-efficient to drive in the city or on the highway?

    iii. A line plot showing the relationship between the “wheelbase” and “turning circle”.

        1. What is this relationship? What happens when the wheelbase gets larger?

    iv. A bar plot showing the mean horsepower for each car type (small, midsize, etc.).

        1. Does a larger car mean more horsepower?

Auto-graded task 2

- Create a new Jupyter notebook in this task’s folder and name it **wine.ipynb**.
- Read in the **wine.csv** file, which is a sample of **wine reviews** collected in 2017.
- Explore the dataset using the `.head()`, `.info()`, and `unique()` **pandas** functions.
- Use the variety column to filter the dataset to only contain “Cabernet Sauvignon”, “Pinot Noir”, and “Chardonnay” wines using the pandas `isin()` function.
- Create a multi-plot grid with a histogram plot of the “points” column for each of these three wine varieties.

Important: Be sure to upload all files required for the task submission inside your task folder and then click "Request review" on your dashboard.

Share your thoughts

Please take some time to complete this short feedback **form** to help us ensure we provide you with the best possible learning experience.

Reference list

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, *9*(3), 90–95. **https://ieeexplore.ieee.org/document/4160265**

The Data Visualisation Catalogue. (n.d.). Stacked area graph. **https://datavizcatalogue.com/methods/stacked_area_graph.html**

VanderPlas, J. (2016). *Python data science handbook*. O'Reilly Media. **https://jakevdp.github.io/PythonDataScienceHandbook/04.08-multiple-subplots.html**