Lab: Python for Data Science

ACTL3143 & ACTL5111 Deep Learning for Actuaries

Data Science Libraries

A couple of fundamental data science packages in Python are NumPy and Pandas. NumPy is a package for handling matrices and vector math, while Pandas handles dataframes and data wrangling.

Libraries are imported using the `import` keyword:

```
import numpy
```

You can set an alias to the libraries you are importing. Usually this is done to simplify the name of a long library.

```
import numpy as np
import pandas as pd
```

You can also import specific functions from a library by using the `from` keyword:

```
from sklearn.preprocessing import StandardScaler
```

In this lab, we will be working with two libraries used for data processing, NumPy and Pandas.

A Note on Installing Libraries

If you have successfully installed Anaconda onto your system, you should already have NumPy and Pandas installed as well. However, if for some reason you do not have a particular library installed, or you would like to update a particular library, you can use the command line to install new packages.

You can either open up Command Prompt/Terminal and type:

```
pip install numpy
```

The `pip` method will also work on Anaconda Prompt. This will install the libraries onto your machine. When installing libraries, it is highly recommended that you create a Conda **environment**, as this allows you to install and manage separate sets of libraries for each Python project you are working on.

For a tutorial on how to set up your own environments, see https://docs.conda.io/projects/conda/en/latest/user-guide/concepts/environments.html

NumPy

NumPy is a package used for scientific computing in Python, with the ability to perform advanced mathematical operations, linear algebra, and vectorisation. Core to the NumPy package is the NumPy array.

NumPy 1D arrays

Unlike lists in base Python, NumPy arrays can only work with numerical data. NumPy arrays are also faster and consumes less memory than Python lists (source: numpy.org/doc/stable/user/absolute_beginners.html).

```
l1 = [1,1,1]
l2 = [2,2,2]

a1 = np.array(l1)
a2 = np.array(l2)

#What do you notice?
print(l1 + l2)
print(a1 + a2)
```

```
[1, 1, 1, 2, 2, 2]
[3 3 3]
```

As you can see in the above code snippet, NumPy arrays are designed for linear algebra operations.

Other operations you can do include adding and multiplying arrays by a constant, calculating determinants of matrices, and even calculating eigenvalues and eigenvectors:

```
a1 + 3 #adds 3 to each element of the array, returns an error if done to a list
a1 * 3 #multiplies each element by 3
```

```
array([3, 3, 3])
```

```
m1 = np.array([[2,4],[1,3]]) #creating a 2D array, i.e. a matrix
print(m1)

print(np.linalg.det(m1)) #Determinant
print(np.linalg.eig(m1)) #Eigenvalues and eigenvectors
```

```
[[2 4]
 [1 3]]
2.0
EigResult(eigenvalues=array([0.43844719, 4.56155281]), eigenvectors=array([[-0.93153209, -0.8
       [ 0.36365914, -0.5392856 ]]))
```

You can create arrays using ranges or linearly spaced sequences:

```
array_range = np.arange(5)
array_lin = np.linspace(start = 0, stop = 1, num = 6)

print(array_range)
print(array_lin)
```

```
[0 1 2 3 4]
[0.  0.2 0.4 0.6 0.8 1. ]
```

NumPy 2D arrays

As mentioned beforehand, you can create a matrix by feeding a list of lists into `np.array()`:

```
m1 = np.array([[2,4],[1,3]])
```

You can also create matrices of zeroes and identity matrices:

```
m_zero = np.zeros([3,3]) #3 x 3 matrix
print(m_zero)

m_ones = np.ones([3,3])
print(m_ones)

m_id = np.identity(3)
print(m_id)
```

```
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

Pandas

Pandas is a Python library used for working with tabular data. It contains tools for data manipulation, time series, and data visualisation. Pandas can be considered a Python equivalent to `dplyr`, and core to Pandas is the `DataFrame` object, which is analogous to R's `data.frame` type.

```
import pandas as pd
```

DataFrames

For this lab we will be working with the Titanic machine learning dataset - a legendary dataset in the data science community. It is available at https://www.kaggle.com/competitions/titanic/data, and we will specifically be using `train.csv`.

To use the dataset in Google Colab, we need to upload and then import it. To see which datasets are available in Google Colab, click the folder icon on the sidebar. Here, you can see

the datasets you have uploaded, as well as any sample datasets that are already built into Google Colab. To upload files, click the upload icon that appears and select the file that you want to upload.

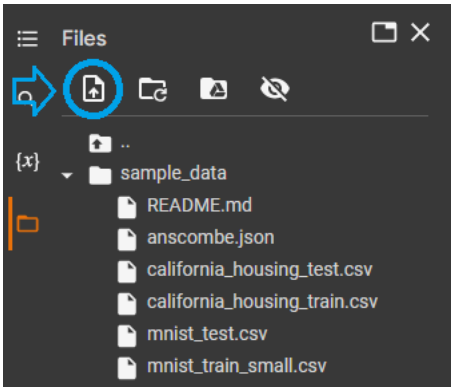

Figure 1: Google Colab Files

We will import the dataset using Pandas' `read_csv()` function.

```
titanic = pd.read_csv("train.csv")
```

This creates a `DataFrame` object, which is a 2-dimensional, tabular data structure.

There are a number of methods available in Pandas to inspect your data, including `.head()` and `.info()`.

```
titanic.head() #much like the head() function in R, this method prints the first 5 rows of t
```

	PassengerId	Survived	Pclass	Name	Sex	Age	Sib
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1
4	5	0	3	Allen, Mr. William Henry	male	35.0	0

```
titanic.tail(10) # Prints last 10 rows
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Pa
881	882	0	3	Markun, Mr. Johann	male	33.0	0	0
882	883	0	3	Dahlberg, Miss. Gerda Ulrika	female	22.0	0	0
883	884	0	2	Banfield, Mr. Frederick James	male	28.0	0	0
884	885	0	3	Sutehall, Mr. Henry Jr	male	25.0	0	0
885	886	0	3	Rice, Mrs. William (Margaret Norton)	female	39.0	0	5
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0

```
titanic.info() # Gives a list of columns, their counts and their types, akin to the str() fu
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

Selecting columns of a Pandas DataFrame is done using square brackets notation:

```
titanic["Age"] # Selecting "Age" column from dataset
```

```
0      22.0
1      38.0
```

```
2      26.0
3      35.0
4      35.0
       ...
886    27.0
887    19.0
888     NaN
889    26.0
890    32.0
Name: Age, Length: 891, dtype: float64
```

```
titanic[["Sex","Age"]] # Selecting multiple columns
```

	Sex	Age
0	male	22.0
1	female	38.0
2	female	26.0
3	female	35.0
4	male	35.0
...	...	...
886	male	27.0
887	female	19.0
888	female	NaN
889	male	26.0
890	male	32.0

There are several ways of selecting rows in a DataFrame, including selecting by row number using the square bracket notation or the `.iloc` method, or selecting by row name using the `.loc` method.

```
titanic[4:9] # Selecting rows by the index (can be different to row number)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0
5	6	0	3	Moran, Mr. James	male	NaN	0
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0

```
titanic.iloc[4:9] # Selecting rows by their row numbers
```

	PassengerId	Survived	Pclass	Name	Sex	Age	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0
5	6	0	3	Moran, Mr. James	male	NaN	0
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0

```
titanic.set_index("Name", inplace=True) #sets the "Name" column as the index
# By setting inplace = True, we modify the existing DataFrame rather than creating a new one
# In other words, we do not need to assign it back to the titanic variable.
```

```
# Selecting rows using .loc
titanic.loc[["Allen, Mr. William Henry", "Moran, Mr. James"]]
```

Name	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Ticket	Fare
Allen, Mr. William Henry	5	0	3	male	35.0	0	0	373450	8.0500
Moran, Mr. James	6	0	3	male	NaN	0	0	330877	8.4583

When selecting both rows and columns, using `.loc` or `.iloc` is necessary:

```
titanic.iloc[4:9, [0, 3]] # Selecting rows 4 to 8, and columns 0 and 3
```

Name	PassengerId	Sex
Allen, Mr. William Henry	5	male
Moran, Mr. James	6	male
McCarthy, Mr. Timothy J	7	male
Palsson, Master. Gosta Leonard	8	male
Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	9	female

```
titanic.loc[["McCarthy, Mr. Timothy J", "Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)"]
```

```
Name
McCarthy, Mr. Timothy J                          54.0
Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)    27.0
Name: Age, dtype: float64
```

You can use the bracket notation to filter the dataset:

```
titanic[titanic["Age"] >= 18]
```

Name	PassengerId	Survived	Pclass	Sex	Age	Sib
Braund, Mr. Owen Harris	1	0	3	male	22.0	1
Cumings, Mrs. John Bradley (Florence Briggs Thayer)	2	1	1	female	38.0	1
Heikkinen, Miss. Laina	3	1	3	female	26.0	0
Futrelle, Mrs. Jacques Heath (Lily May Peel)	4	1	1	female	35.0	1
Allen, Mr. William Henry	5	0	3	male	35.0	0
...	...	...	...	...	...	...
Rice, Mrs. William (Margaret Norton)	886	0	3	female	39.0	0
Montvila, Rev. Juozas	887	0	2	male	27.0	0
Graham, Miss. Margaret Edith	888	1	1	female	19.0	0
Behr, Mr. Karl Howell	890	1	1	male	26.0	0
Dooley, Mr. Patrick	891	0	3	male	32.0	0

This has reduced the dataset from 891 rows to 601.

If we wanted to combine multiple conditions together, we can use conditional operators. However, Python's usual conditional operators (`and`, `or`, `not`) will not work here, and instead we will need to use symbols (`&`, `|`, `!`).

```
# Selecting passengers whose ages are 18 and above and are in passenger class 3.
titanic[(titanic["Age"] >= 18) & (titanic["Pclass"] == 3)] #Note that we need to wrap each c
```

Name	PassengerId	Survived	Pclass	Sex	Age	SibSp
Braund, Mr. Owen Harris	1	0	3	male	22.0	1
Heikkinen, Miss. Laina	3	1	3	female	26.0	0
Allen, Mr. William Henry	5	0	3	male	35.0	0
Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	9	1	3	female	27.0	0
Saundercock, Mr. William Henry	13	0	3	male	20.0	0

Name	PassengerId	Survived	Pclass	Sex	Age	SibSp
...	...	...	...	...	...	...
Markun, Mr. Johann	882	0	3	male	33.0	0
Dahlberg, Miss. Gerda Ulrika	883	0	3	female	22.0	0
Sutehall, Mr. Henry Jr	885	0	3	male	25.0	0
Rice, Mrs. William (Margaret Norton)	886	0	3	female	39.0	0
Dooley, Mr. Patrick	891	0	3	male	32.0	0

That line of code is quite longwinded, so if you wanted to filter your DataFrame in a more concise way, you can use the `.query()` method:

```
titanic.query("Age >= 18 & Pclass == 3")
```

Name	PassengerId	Survived	Pclass	Sex	Age	SibSp
Braund, Mr. Owen Harris	1	0	3	male	22.0	1
Heikkinen, Miss. Laina	3	1	3	female	26.0	0
Allen, Mr. William Henry	5	0	3	male	35.0	0
Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	9	1	3	female	27.0	0
Saundercock, Mr. William Henry	13	0	3	male	20.0	0
...	...	...	...	...	...	...
Markun, Mr. Johann	882	0	3	male	33.0	0
Dahlberg, Miss. Gerda Ulrika	883	0	3	female	22.0	0
Sutehall, Mr. Henry Jr	885	0	3	male	25.0	0
Rice, Mrs. William (Margaret Norton)	886	0	3	female	39.0	0
Dooley, Mr. Patrick	891	0	3	male	32.0	0

In Pandas you can aggregate datasets using the `.groupby()` method:

```
titanic.groupby("Pclass").sum()["Survived"]
```

```
Pclass
1    136
2     87
3    119
Name: Survived, dtype: int64
```

Notice in the above line of code, we combined two methods. In Pandas, you can chain multiple methods together, much like dplyr's pipline operator (`%>%`) in R.

```
# Select the names of passengers in class 3 who are 65 years of age or older.
titanic.reset_index().query("Pclass == 3 & Age >= 65")["Name"]
```

```
116    Connors, Mr. Patrick
280        Duane, Mr. Frank
851     Svensson, Mr. Johan
Name: Name, dtype: object
```

Exercises

1. Filter the dataset to people where `Embarked` is Q.
2. Filter the dataset to people 18 years or older, and `Fare` is less than 10
3. Filter the dataset to people with an above-median age.
4. What is the highest value of `Fare` for female passengers in class 2?

Series

Let's select the `Ticket` column:

```
titanic["Ticket"]
```

```
Name
Braund, Mr. Owen Harris                                     A/5 21171
Cumings, Mrs. John Bradley (Florence Briggs Thayer)          PC 17599
Heikkinen, Miss. Laina                                STON/O2. 3101282
Futrelle, Mrs. Jacques Heath (Lily May Peel)                  113803
Allen, Mr. William Henry                                      373450
                                                              ...
Montvila, Rev. Juozas                                         211536
Graham, Miss. Margaret Edith                                  112053
Johnston, Miss. Catherine Helen "Carrie"                    W./C. 6607
Behr, Mr. Karl Howell                                         111369
Dooley, Mr. Patrick                                           370376
Name: Ticket, Length: 891, dtype: object
```

When selecting a single column of the DataFrame, Pandas returns what is known as a **Series**. This is a data structure used to represent one-dimensional data, much like a list or NumPy array. They are more flexible than NumPy arrays because they can hold non-numeric data types. However, they are not as flexible as lists because they can only hold one datatype at a time. If you try to create a Series with values of different data types, Pandas will convert all the elements of the Series into strings.

```
# You can create series from lists, tuples, and NumPy arrays
l = ["The", "quick", "brown", "fox"]
t = (3,1,4,1,5,9)
a = np.array(t)
mix = ["this", 3, "will", True, "convert"]

print(pd.Series(l))
print(pd.Series(t))
print(pd.Series(a))
print(pd.Series(mix)) #converted into strings
```

```
0      The
1    quick
2    brown
3      fox
dtype: object
0    3
1    1
2    4
3    1
4    5
5    9
dtype: int64
0    3
1    1
2    4
3    1
4    5
5    9
dtype: int64
0       this
1          3
2       will
3       True
4    convert
```

```
dtype: object
```