Data Preprocessing

① Downloading the data sets

They can be downloaded from:
www.superdatascience.com/machine-learning
The dataset contains four columns.
Country, age, salary, purchased.

Here, country, age and salary dictate the characteristics of a ~~person~~ person who purchased a product (or not).

② Independent and dependent variables

In this database (data.csv) we will be predicting ~~an independent~~ a dependent variable based on independent variables.
Here independent variables are :- country, age, salary
dependent variable :- purchased.

③ Importing the essential libraries

The three essential libraries are:
① Numpy :- used to complete all mathematics in our code
② Matplotlib :- In this we use a sub-library called pyplot. It is used to plot charts in python.
③ Pandas :- This library is used to import and manage datasets.

__/__/____

The code to import them :-

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

④ Importing the dataset

(i) Set a folder as working directory :- Via file explorer in Spyder window, navigate to the folder where data file is located. Save your python file there

Syntax for importing :

<variable name> = pandas.read_csv('<dataset>')

∴ in this case,

```
dataset = pd.read_csv('Data.csv');
```

Indexes in python start at 0

Now, we need to separate the independent and dependent variables from our dataset. For that, we use a variable X (say) and write

```
X = dataset.iloc[:, :-1].values
```

Here : → all.

and iloc works in row, column format

∴ dataset.iloc[:, :-1].values

selects all rows (:) and leaves the last column (:-1).

Next, the dependent variable column is imported. For that we use :

```
y = dataset.iloc[:, 3].values
```

(3 is index of purchased column).

__/__/__

⑤ Handling missing data

To resolve missing data issues, we take the mean of the column, the data is missing from and replace it with that value.

To do this, we need to use sklearn.preprocessing library and import its Imputer class, which will help us handle this data.
For this we write,

→ from sklearn.preprocessing import Imputer

Then, we create an object of Imputer class. To view the parameters that constructor requires, we can press 'command + I'

⇒ Imputer class

Constructor parameters ⇒

(i) missing_values : used to check placeholder for missing values
default = "NaN"

(ii) strategy : strategy used to replace ~~default~~ missing values
default = "mean"

(iii) axis : to check direction of impute
axis = 0, along columns
axis = 1, along rows

To fit this object of imputer class on our variable X we use:

imputer.fit(X[:, 1:3])

here X[:, 1:3] is again row, column format.
':' selects all rows.
We select only columns with missing data values.
1:3 selects columns with indices 1 2 because here, upperbound is excluded.

once, imputer is fitted on X, we use the transform method of imputer class to finally fill missing data. Thus, our final code for handling missing data:

```
from sklearn.preprocessing import Imputer
imputer = Imputer(missing_values = 'NaN', strategy = 'mean', axis = 0)
imputer = imputer.fit(X[:, 1:3])
X[:, 1:3] = imputer.transform(X[:, 1:3])
```

⑥ Categorical variables

Columns which contain text can be sorted into categories.
For example, in 'data.csv' the columns country and ~~purchased~~ purchased are categorical variables.
For country, categories are ⇒ France, Spain, Germany
For purchased ⇒ Yes, No

This needs to be done because we need to make mathematical models and we simply wont be able to handle text.

⑦ Encoding categorical data

For encoding categorical data, we again use sklearn.preprocessing class ; from which we import LabelEncoder and OneHotEncoder class.

The label encoder class object when fit_transform onto our array gives each category a numerical value.
However,
when France → 0
Germany → 1
Spain → 2.
Since, country is an independent variable, our ML algorithm might mistake it for an equation in which France ≤ Germany < Spain which does not make any sense.
Hence, we use dummy encoding. Dummy encoding distributes this array and replaces countries column into one column for each category where 0 indicates presence of that category in the particular row.
The constructor for LabelEncoder requires no parameters.
The constructor for oneHotEncoder requires categorical_features which takes input of columns to be encoded.

NOTE:- We do not need to dummy encode our dependent variable column because our ML algorithm recognises it as categorical value.

The code for encoding in Python:

```
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
labelencoder_X = LabelEncoder()
X[:, 0] = labelencoder_X.fit_transform(X[:, 0])
onehotencoder = OneHotEncoder(categorical_features = [0]
```

__/__/____

X = onehotencoder.fit_transform(X).toarray().
labelencoder_y = LabelEncoder()
y = labelencoder_y.fit_transform(y).

⑧ Training set and Test set (Why split?)

our ML models are coded such that they learn from a particular set of data, but then need to be tested on a slightly different set of data to check if our machine actually learned something
∴ Split dataset into training set and test set.

⑨ Splitting

We import now train_test_split library from sklearn.model_selection.
We will split our datasets X and Y into four objects X_train, X_test and y_train, y_test

We then use the train_test_split utility.
The parameters are:
① arrays are input
② test_size = 0.2 (0.2 means 20%. Usually taken between 0.2 - 0.4).
③ random_state = 0 (can be any integer, to always split same data in test & training sets).

Code:-
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)

⑩ Feature Scaling

Say, in this database we have 2 columns salary and ~~purchased~~ age.
In ML models, many times we calculate the euclidean distance b/w two points

$$\sqrt{(y_2-y_1)^2 + (x_2-x_1)^2}$$

Here, $x \rightarrow$ age column
$y \rightarrow$ salary column.
Looking at the values, we can see that

$$(\Delta y)^2 >> (\Delta x)^2$$

$$\therefore \sqrt{\Delta y^2 + \Delta x^2} \approx \Delta y$$

Thus, our y will render x as non-existent due to its sheer size. To avoid this, we use feature scaling.
Hence, we get the scale of both x and y to the same count, i.e. b/w -1 to +1 in both the cases.
We can do this by two methods:

① Standardisation : $\frac{x - \text{mean}(x)}{\text{standard deviation}(x)}$

② Normalisation : $\frac{x - \min(x)}{\max(x) - \min(x)}$

⑪ Implementation in Python (Feature Scaling)

Again from sklearn.preprocessing library import StandardScaler class.
Fit and transform to training set and then only transform the test set.
The code is as follows:

__/__/____

```
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test).
```