Data Preprocessing

Review the dataset

Data

The above datset shows Country, Age, Salary and Purchased. The field that we need to research is called `Dependenct field` and others are called `Independent Field`

Dependent Field

A dependent field is a whose outcome or value is derived from other field. In this example, we are trying to find Whether a customer made a purchase or not. Hence, `Purchased` here becomes the dependent field. In terms of coordinate, if we map this on graph then `Purchased` will be plotted on y-axis.

Independent field

Indendent fields are values that we observe. Example - If I stand on the side of freeway and start noting down the color of each car passed, whether it is a car or a truck, is it raining that day or sunny? etc. These values are called Independent fields. In the above datasheet, `Country` , `Age` and `Salary` are independent fields.

In data science, everything is function. Hence if `Purchased` field = `Y` and `Country` , `Age` and `Salary` can be represented as `c` , `a` & `i` respectively. Then the above function can be represented as

```
Y = Xc + Ta + Zi
```

Importing the essential library

There are three essential library for most basic machine learning projects. Add the below library

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

In [2]:

```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

Import the dataset

Use pandas to import the file data.csv file

In [3]:

```
#Import the dataset
dataset = pd.read_csv("Data.csv",na_values=["$$"])
dataset
```

Out[3]:

	Country	Age	Salary	Purchased
0	France	44.0	72000.0	No
1	Spain	27.0	48000.0	Yes
2	Germany	30.0	54000.0	No
3	Spain	38.0	61000.0	No
4	Germany	40.0	NaN	Yes
5	France	35.0	58000.0	Yes
6	Spain	NaN	52000.0	No
7	France	48.0	79000.0	Yes
8	Germany	50.0	83000.0	No
9	France	37.0	67000.0	Yes

Create the dependent and Indepdent Variable matrix of features

In [4]:

```
x = dataset.iloc[:, :3].values
y = dataset.iloc[:, 3:4].values
```

Identify the missing data

In real world when you are handed a dataset. You will find that lot of data might be missing. Ex - `Row 6 Salary` is empty and `Row 8 Age` is missing in our data.csv

Country	Age	Salary	Purchased
France	44	72000	No
Spain	27	48000	Yes
Germany	30	54000	No
Spain	38	61000	No
Germany	40		Yes
France	35	58000	Yes
Spain		52000	No
France	48	79000	Yes
Germany	50	83000	No
France	37	67000	Yes

Startegies to handle missing data

- Remove the rows that have missing data. This approach is simplest but will skew our dataset if lot of fields are missing.
- Replace the missing data with Mean
- Replace the missing data with Median
- Replace the missing data with Mode or Frequency

As a general rule of thumb `mean` , `median` , `most_frequent` , `constant` and can be applied to numerical values only. It cannot be applied to alphnumeric or String values. This is where we can use Mode or Frequency.

Taking Care of missing data * (Numerical values)*

`Scikit` library in python provides a class called `Imputer` which helps in fixing the missing values using the above strategies.

In [5]:
```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(missing_values = np.nan, strategy='mean')
imputer = imputer.fit(x[:,1:3])
x[:, 1:3] = imputer.transform(x[:, 1:3])
```

In the above code we define a `SimpleImputer` class object. `imputer = SimpleImputer(missing_values = np.nan, strategy='mean')`

The next line `imputer = imputer.fit(x[:,1:3])` is telling imputer object on which matrix columns it needs to look for missing value and mark them for filling value as mean

The last line `x[:, 1:3] = imputer.transform(x[:, 1:3])` fills in the cells of X martix that were marked by the imputer.

One thing that I wanted to try was that would happen if I just used `imputer.fit(x)` and did not provide the columns which actually had the missing values. What happens is that since you have marked the startegy as `strategy='mean'` , the imputer will try to take average/mean of each row `country` , `Age` & `Salary` . Since `Country` is of type String, python will complain about it. However if the `Country` column was numeric, then this `imputer.fit(x)` would have worked. Additionally, on the left side too we have defined that data be copied into x[:, 1:3]. This is because if you write `x = imputer.transform(x[:, 1:3])` , imputer has marked the cell to be transformed is `(1,5)` as there are two columns it is looking at `Age` and `Salary` but if you put only x on the left side, then `(1,5)` refers to `Age` 40, while imputer is expecting it to be a `Salary` column.

In [6]:
```
x
```

Out[6]:
```
array([['France', 44.0, 72000.0],
       ['Spain', 27.0, 48000.0],
       ['Germany', 30.0, 54000.0],
       ['Spain', 38.0, 61000.0],
       ['Germany', 40.0, 63777.77777777778],
       ['France', 35.0, 58000.0],
       ['Spain', 38.77777777777778, 52000.0],
       ['France', 48.0, 79000.0],
       ['Germany', 50.0, 83000.0],
       ['France', 37.0, 67000.0]], dtype=object)
```

```
[['France' 44.0 72000.0]
 ['Spain' 27.0 48000.0]
 ['Germany' 30.0 54000.0]
 ['Spain' 38.0 61000.0]
 ['Germany' 40.0 63777.77777777778]
 ['France' 35.0 58000.0]
 ['Spain' 38.77777777777778 52000.0]
 ['France' 48.0 79000.0]
 ['Germany' 50.0 83000.0]
 ['France' 37.0 67000.0]]
```

As you can see, the imputer has filled in the missing values for `Salary` as $63777.78 and `Age` as 38.7.

Taking care of Categorical data

Categorical data is any data that is not numeric. Think of it as attributes of an object. Ex - If I am observing cars on a freeway and noting down the speed at which the car is driving and the color of the car. The cars may have different colors such as red, green, blue etc. I can only observe them but cannot do anything else with them like add, subtract, find mean or anything. However, a machine learning algorithm may see value in finding a relation between dependent and independent feature. The algorithm can only make use of it, if it is defined in numbers.

One way to define them is by assigning them numbers ex- Red =1, Blue =2 and so on. In our current example we have a categorical data - `Country` . To convert or encode as it is called in ML, we will again use a library from `Scikit`

In [7]:
```
from sklearn.preprocessing import LabelEncoder
labelEncoder_x = LabelEncoder()
x[: ,0] =labelEncoder_x.fit_transform(x[:, 0])
x
```

Out[7]:
```
array([[0, 44.0, 72000.0],
       [2, 27.0, 48000.0],
       [1, 30.0, 54000.0],
       [2, 38.0, 61000.0],
       [1, 40.0, 63777.77777777778],
       [0, 35.0, 58000.0],
       [2, 38.77777777777778, 52000.0],
       [0, 48.0, 79000.0],
       [1, 50.0, 83000.0],
       [0, 37.0, 67000.0]], dtype=object)
```

The above code uses `LabelEncoder` class to encode the values of country. Here, France =0, Spain = 2, Germany=1 There is however a simple problem here.

```
[[0 44.0 72000.0]
 [2 27.0 48000.0]
 [1 30.0 54000.0]
 [2 38.0 61000.0]
 [1 40.0 63777.77777777778]
 [0 35.0 58000.0]
 [2 38.77777777777778 52000.0]
 [0 48.0 79000.0]
 [1 50.0 83000.0]
 [0 37.0 67000.0]]
```

#####Problem: The modesl are based on equation. Since the LabelEncoder here, has assigned them values 0,1,2, the equation would think that Germany has highre value than France and Spain has higher value than Germany. This certainly is not the case. These are supposed to be treated as observational values. Example - Picking on our earlier example of observing car speeds and color. If we use LabelEncoder for encoding car colors, the model may comeback and say that A `red Prius` will always be faster than a `White Ferari`

To get over this problem and use category as markers, we will use another class which will create dummy encoding and this give equal value to all categorical data. It does that by creating a sparse matrix.

Country
France
Spain
Germany
Spain
Germany
France
Spain
France
Germany

France	Germany	Spain
1	0	0
0	0	1
0	1	0
0	0	1
0	1	0
1	0	0
0	0	1
1	0	0
0	1	0

In [10]:
```python
from sklearn.preprocessing import OneHotEncoder
oneHotEncoder = OneHotEncoder(categories='auto')
x= oneHotEncoder.fit_transform(x).toarray()
```

Out[10]:
```
array([[0., 1., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0.,
        1., 0., 1., 0., 0., 1., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0.,
        1., 0., 1., 0., 1., 0., 1., 0., 0., 1., 1., 0., 1., 0.],
       [1., 0., 1., 0., 0., 1., 0., 1., 1., 0., 1., 0., 1., 0., 1., 0.,
        1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 0., 1., 1., 0., 1., 0.,
        1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0.],
       [1., 0., 0., 1., 1., 0., 1., 0., 0., 1., 1., 0., 1., 0., 1., 0.,
        1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 0., 1.,
        1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0.],
       [1., 0., 1., 0., 0., 1., 1., 0., 1., 0., 1., 0., 1., 0., 0., 1.,
        1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0.,
        1., 0., 0., 1., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0.],
       [1., 0., 0., 1., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0.,
        1., 0., 0., 1., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0.,
        1., 0., 1., 0., 0., 1., 1., 0., 1., 0., 1., 0., 1., 0.],
       [0., 1., 1., 0., 1., 0., 1., 0., 1., 0., 0., 1., 1., 0., 1., 0.,
        1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0.,
        0., 1., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0.],
       [1., 0., 1., 0., 0., 1., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0.,
        0., 1., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 0., 1., 1., 0.,
        1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0.],
       [0., 1., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0.,
        1., 0., 1., 0., 1., 0., 0., 1., 1., 0., 1., 0., 1., 0., 1., 0.,
        1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 0., 1., 1., 0.],
       [1., 0., 0., 1., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0.,
        1., 0., 1., 0., 1., 0., 1., 0., 0., 1., 1., 0., 1., 0., 1., 0.,
        1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 0., 1.],
       [0., 1., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 0., 1., 1., 0.,
        1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0.,
        1., 0., 1., 0., 1., 0., 0., 1., 1., 0., 1., 0., 1., 0.]])
```

So after dummy encoding, the complete sparse matrix of `x` looks like this

dummyEncodingResult.PNG

So now let's convert the dependent coloumn `Purchased` as well. However, we do not need to use dummy encoding as there are only two

In [11]:
```
labelEncoder_y = LabelEncoder()
y =labelEncoder_y.fit_transform(y)
```

Splitting dataset into training and test set

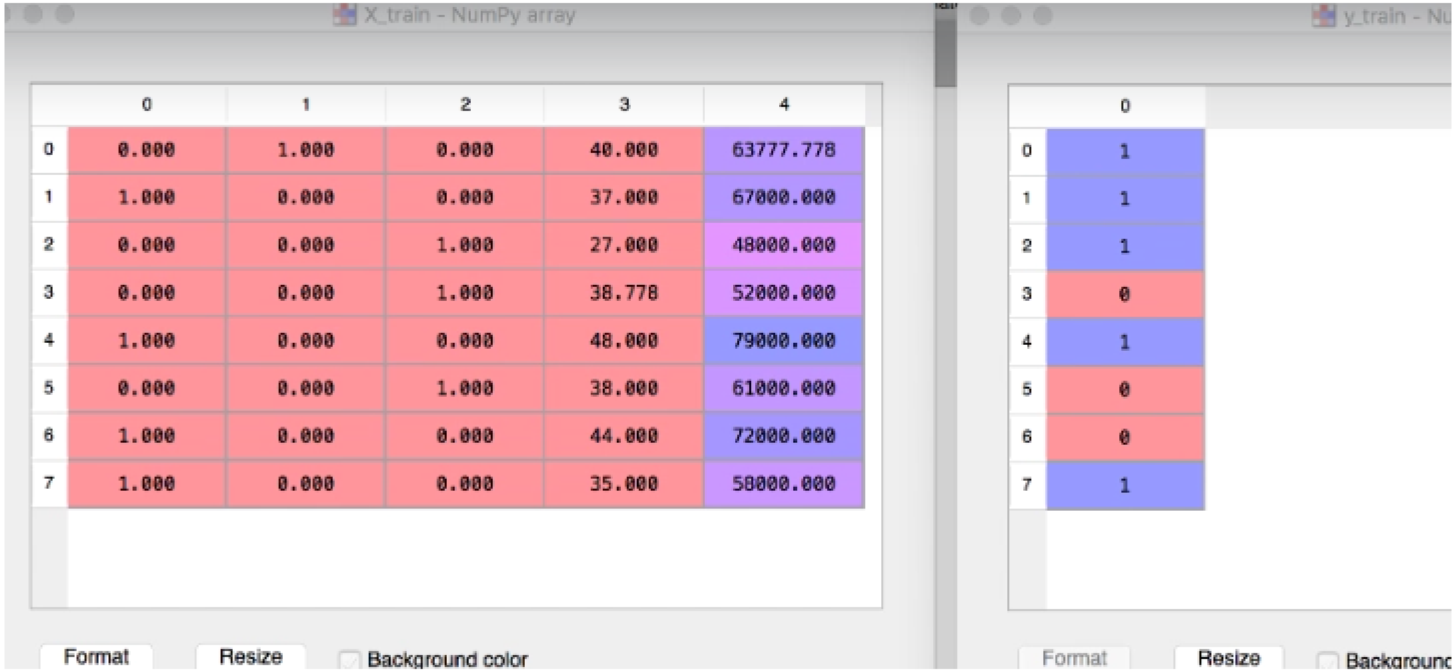
- **Why do we need to split the data into training and test set?**

    1. This is about algorithm that will create the equation to predict the result of new information based on history. If you let it run on all of the data then it will learn too much and will have correlation value or in other words overfitting. A simple example in real world is of a boy who memorizes the book word by word but fails in the actual exam, because instead of asking what is 2 +2 like he read in the book, the exam asked what is 1+3. The student learnt too much but cannot relate or imply the same knowledge on a new data set.
    2. Sometimes you may have limited data to build the model and may not additional data to test your model.
- ** What is a good ratio to split the data?** Usually 80/20 or upto 70/30 is a good number. Going beyond 70/30 is not recommended.

To split a dataset into taining and test set. We will use another class called `train_test_split` , which returns 4 different values - Traiing_X, Test_X,Traiing_Y, Test_Y

In [12]:
```
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)
```

The data is now spllit into 8 and 2 observation.



	0	1	2	3	4
0	0.000	1.000	0.000	40.000	63777.778
1	1.000	0.000	0.000	37.000	67000.000
2	0.000	0.000	1.000	27.000	48000.000
3	0.000	0.000	1.000	38.778	52000.000
4	1.000	0.000	0.000	48.000	79000.000
5	0.000	0.000	1.000	38.000	61000.000
6	1.000	0.000	0.000	44.000	72000.000
7	1.000	0.000	0.000	35.000	58000.000

	0
0	1
1	1
2	1
3	0
4	1
5	0
6	0
7	1

Feature Scaling/ Normalization/Standardization
If you look your dataset and pay attention to independent features `Age` and `Salary`, the range varies for `Age` between `27 - 50` and `Salary` between `48000 - 83000`. When an equation is created, the distance between two datapoints is huge and the values can be skewed because one of the columns have 27 for age and 83000 for salary.

`![ecd](ecd.PNG)`
The models are usualy based on Euclidean distance. In our case since Salary has a higher range, it will dominate the age values which means when we do distance between observation (27, 48000) and (48, 79000) then (x2-x1) ^2 vs (y2-y1) ^2 is
441 vs 961000000. Hence Age is oversahdowed by Salary.

`![zstd](zstd.PNG)`

If you are from statstics field, you probably are already familiar with standardization or Z-index etc. Above is the two way to calculate the standard range which will always give us a value between **0 and 1**

In [13]:
```
from sklearn.preprocessing import StandardScaler
sc_x = StandardScaler()
x_train = sc_x.fit_transform(x_train)
x_test = sc_x.transform(x_test)
```

NOTE : We are not fitting and transforming the `x_test` because it is already fitted based on `xtrain` so that they are now on same scale. If we would have used fit_transform on both test and train then their scale would have been different say one could between -1 and +1 while the other could be on -3 and +3

	0	1	2	3	4
0	-1.000	2.646	-0.775	0.263	0.124
1	1.000	-0.378	-0.775	-0.254	0.462
2	-1.000	-0.378	1.291	-1.975	-1.531
3	-1.000	-0.378	1.291	0.053	-1.111
4	1.000	-0.378	-0.775	1.641	1.720
5	-1.000	-0.378	1.291	-0.081	-0.168
6	1.000	-0.378	-0.775	0.952	0.986
7	1.000	-0.378	-0.775	-0.598	-0.482

- **Why did we not apply scaling on `y` or dependent variable?** In this case the values are 0 and 1 only or in other words this is a classification problem whcih has two choices, whether a customer bought the product or he did not buy the product. In other scenario such as that realted to multiple regression , we may have sitituation where we may have to do feature scaling on `y` as well.

Constant Factor Normalization (or) Decimal Scaling :

This technique scales the values of a feature by dividing the values of a feature by a power of 10.

$$v' = \frac{v}{\quad}$$

In [ ]:

```

```