data preprocessing

###############################################################

**#       Step 1 : Importing the libraries            #**

###############################################################

# NumPy is module for Python. The name is an acronym for "Numeric Python" or "Numerical Python".

# This makes sure that the precompiled mathematical and numerical functions

# and functionalities of Numpy guarantee great execution speed.

import numpy as np

# Pandas is an open-source Python Library providing high-performance data manipulation

# and analysis tool using its powerful data structures.

# The name Pandas is derived from the word Panel Data – an Econometrics from Multidimensional data.

import pandas as pd

# The OS module in Python provides a way of using operating system dependent functionality.

# The functions that the OS module provides allows you to interface with the underlying operating system

# that Python is running on – be that Windows, Mac or Linux.

import os

###############################################################

#       Step 2 : Importing the Dataset                        #

###############################################################

#Read the 'Data.csv' and store the data in the vairable dataset.

dataset = pd.read\_csv("../input/Data.csv")

print('Our the datasets......')

# ......................................................................

# If the dataset is in the Google Drive, You Can write the following:

#Connecting/ Mounting with Google Drive

from google.colab import drive

drive.mount('/content/drive/')

#Assigning/ Defining the Root's Path

root\_path= '/content/drive/MyDrive/Colab Notebooks/ML Lab/ml\_lab\_task\_01\_data\_C201032.csv'

dataset = pd.read\_csv(root\_path)

print('Our the datasets......')

# ......................................................................

# Print the shape of the dataset

print ('dataset: %s'%(str(dataset.shape)))

# print the dataset

dataset

# Separate the dependent and independent variables

# Independent variable

# iloc[rows,columns]

# Take all rows

# Take last but one column from the dataset (:-1)

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

(

* dataset.iloc[:,:-1] selects all rows and all columns except the last one from the dataset.
* .values converts the selected data into a NumPy array.

)

# Dependent variable

# iloc[rows,columns]

# Take all rows

# Take last column from the dataset (:-1)

Y = dataset.iloc[:,3].values

(

* (dataset.iloc[:,3] selects all rows and only the fourth column (index 3) from the dataset.
* .values converts the selected data into a NumPy array.

)

# Print the X and Y

print ('X: %s'%(str(X)))

print ('-----------------------------------')

print ('Y: %s'%(str(Y)))

#### 1. Handle Missing Data

There are few missing data in the Age and salary columns (NaN values).

#### i. Deleting Rows:

\* We cannot remove the rows with the missing data as it will affect the output of the machine learning algorithm.

\* However we can delete a particular row if it has a null value for a particular feature and a particular column if it has more than 70-75% of missing values.

#### ii. Replacing With Mean/Median/Mode:

\* This strategy can be applied on a feature which has numeric data like the age of a person.

\* We can calculate the mean, median or mode of the feature and replace it with the missing values.

\* The loss of the data can be negated by this method which yields better results compared to removal of rows and

\* columns.

\* Replacing with the above three approximations are a statistical approach of handling the missing values.

\* This method is also called as leaking the data while training.

\* Another way is to approximate it with the deviation of neighbouring values.

\* This works better if the data is linear.

###############################################################

#       Step 3 : Missing Data                                 #

###############################################################

# Scikit-learn provides a range of supervised and unsupervised learning algorithms via a consistent interface in Python.

# The sklearn.preprocessing package provides several common utility functions and transformer classes

# to change raw feature vectors into a representation that is more suitable for the downstream estimators.

from sklearn.preprocessing import Imputer

# Imputer Class takes the follwing parameters:

#     missing\_values : The missing values in our dataset are called as NaN (Not a number).Default is NaN

#     strategy       : replace the missing values by mean/median/mode. Default is mean.

#     axis           : if axis = 0, we take we of the column and if axis = 1, we take mean value of row.

imputer = Imputer(missing\_values = 'NaN',strategy = 'mean', axis = 0)

# Fit the imputer on X.

# Take all rows and columns only with the missing values.

# Note: Index starts with 0. Upper bound (3) is not included.

# Fit imputer for columns 1 and 2 of X matrix.

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

#Replace missing data with mean of column

#### 2. Encode the Categorical data

Categorical data are variables that contain label values rather than numeric values.

Some algorithms can work with categorical data directly.

For example, a decision tree can be learned directly from categorical data with no data transform required (this depends on the specific implementation).

Many machine learning algorithms cannot operate on label data directly. They require all input variables and output variables to be numeric.

This means that categorical data must be converted to a numerical form.

In our dataset there are 2 columns with categorical data.

The First column which contains the country and the last column purchased.

#### i. Label Encoder:

\* It is used to transform non-numerical labels to numerical labels (or nominal categorical variables).

\* Numerical labels are always between 0 and n\_classes-1.

#### ii. OneHotEncoder:

\* Encode categorical integer features using a one-hot aka one-of-K scheme.

\* The input to this transformer should be a matrix of integers, denoting the values taken on by categorical (discrete)

features.

\* The output will be a sparse matrix where each column corresponds to one possible value of one feature.

\* It is assumed that input features take on values in the range [0, n\_values]

\* This encoding is needed for feeding categorical data to many scikit-learn estimators, notably linear models and SVMs

with the standard kernels.

###############################################################

#       Step 4 : Categorical variables                        #

###############################################################

from sklearn.preprocessing import LabelEncoder,OneHotEncoder

labelencoder\_X = LabelEncoder()

X[:,0] = labelencoder\_X.fit\_transform(X[:,0])

X[:,0]

……….

Now the categorical data of the country value is changed to numerical value.

| Country | Value |

|:--------|:------|

| China | 0 |

| India | 1 |

| Srilanka| 2 |

………

#### Dummy Encoding

\* The above encoding will result in a problem.

\* The label encoding transforms the data as shown in the table above.

\* The Machine learning algorithm will assume that China>India>Sri Lanka.

\* But this is not the case. We just converted the categorical value and assigned it to a numeric value.

\* Hence there is a need to apply Dummy encoding to the above dataset.

| Country | China | India | Sri Lanka |

|:--------|:------|:------|:----------|

| China | 1 | 0 | 0 |

| India | 0 | 1 | 0 |

| Srilanka| 0 | 0 | 1 |

| India | 0 | 1 | 0 |

| Srilanka| 0 | 0 | 1 |

| China | 1 | 0 | 0 |