Course: Artificial Intelligence

Assistant Lecturer : Yasmin Alsakar

Information Technology Department

Faculty of Computer & Information Sciences - Mansoura University.

OUTLINE

- Reading an Excel File
- Preprocessing
- Rescale Data
- Normalization Data
- Binarize Data (Make Binary)
- Standardize Data
- Label Encoding
- Scikit-learn

OUTLINE

- **Data Imputation**
- **How to impute missing values with means in Python?**
- **Examples for data imputation**
- **Exploratory Data Analysis**

Reading an Excel File

- The **read_excel** function of the pandas library is used read the content of an Excel file into the python environment as a pandas DataFrame. The function can read the files from the OS by using proper path to the file. By default, the function will read Sheet1.

```
# Reading an Excel File
import pandas as pd
data = pd.read_excel('presidents_names.xlsx')
print (data)
```

```
              Name       born  in office  in office.1
0    Jimmy C@rter 2024-01-10        1977       1981.0
1    Donald Trump 1946-06-14        2017          NaN
2  George W. Bush 1946-07-06        2001       2009.0
3    Bill Clinton 1946-08-19        1993       2001.0
4    Barack Obama 1961-08-04        2009       2017.0
```

- **Reading Specific Columns and Rows:**
- Similar to what we have already seen in the previous chapter to read the CSV file, the **read_excel** function of the pandas library can also be used to read some specific columns and specific rows. We use the multi-axes indexing method called **.loc()** for this purpose. We choose to display the salary and name column for some of the rows.

```
# Reading Specific Columns and Rows
import pandas as pd
data = pd.read_excel('presidents_names.xlsx')

# Use the multi-axes indexing funtion
print (data.loc[[1,2],['Name','born']])
```

```
              Name        born
1     Donald Trump  1946-06-14
2  George W. Bush  1946-07-06
```

- **Reading Multiple Excel Sheets**
- Multiple sheets with different Data formats can also be read by using read_excel function with help of a wrapper class named **ExcelFile**. It will read the multiple sheets into memory only once. In the below example we read sheet1 and sheet2 into two data frames and print them out individually.

```
# Reading Multiple Excel Sheets
import pandas as pd
with pd.ExcelFile('presidents_names.xlsx') as xls:
    df1 = pd.read_excel(xls, 'Sheet1')
    df2 = pd.read_excel(xls, 'Sheet2')

print("****Result Sheet 1****")
print (df1[0:3]['Name'])
print("")
print("***Result Sheet 2****")
print (df2[0:3]['Name'])
```

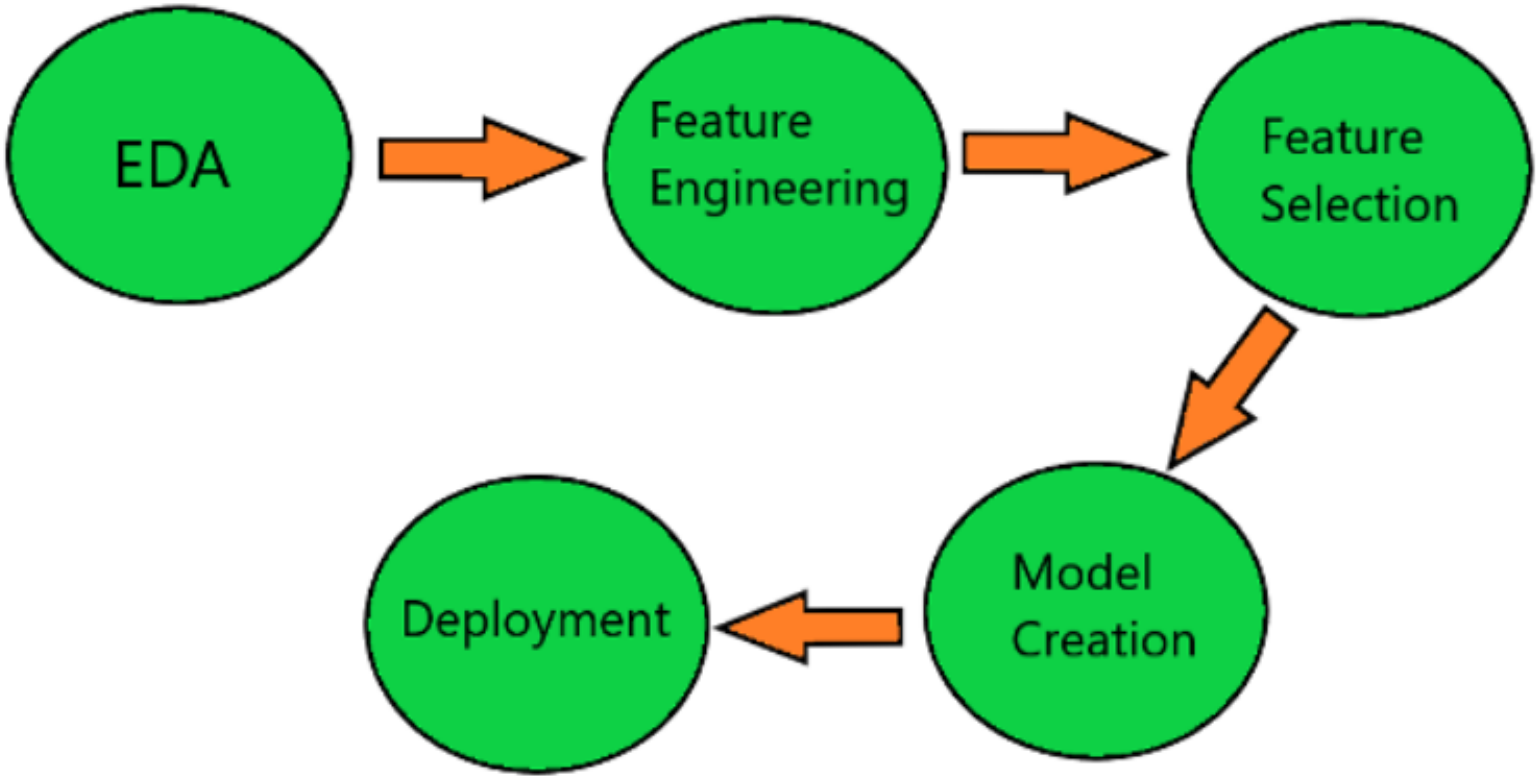
```
****Result Sheet 1****
0      Jimmy C@rter
1      Donald Trump
2    George W. Bush
Name: Name, dtype: object

***Result Sheet 2****
0     yasmin
1      ahmed
2    mohamed
Name: Name, dtype: object
```

DATA SCIENCE PROJECT LIFE CYCLE

- This will give us a better idea of the steps involved in developing any data science project and the importance and usage of these functions. Let's discuss these steps in points:

1. **Exploratory Data Analysis (EDA)** is used to analyze the datasets using pandas, numpy, matplotlib, etc., and dealing with missing values. By doing EDA, we summarize their main importance.
2. **Feature Engineering** is the process of extracting features from raw data with some domain knowledge.
3. **Feature Selection** is where we select those features from the dataframe that will give a high impact on the estimator.
4. **Model creation** in this, we create a machine learning model using suitable algorithms, e.g., regressor or classifier.
5. **Deployment** where we deploy our ML model on the web.


EDA
Feature Engineering
Feature Selection
Model Creation
Deployment

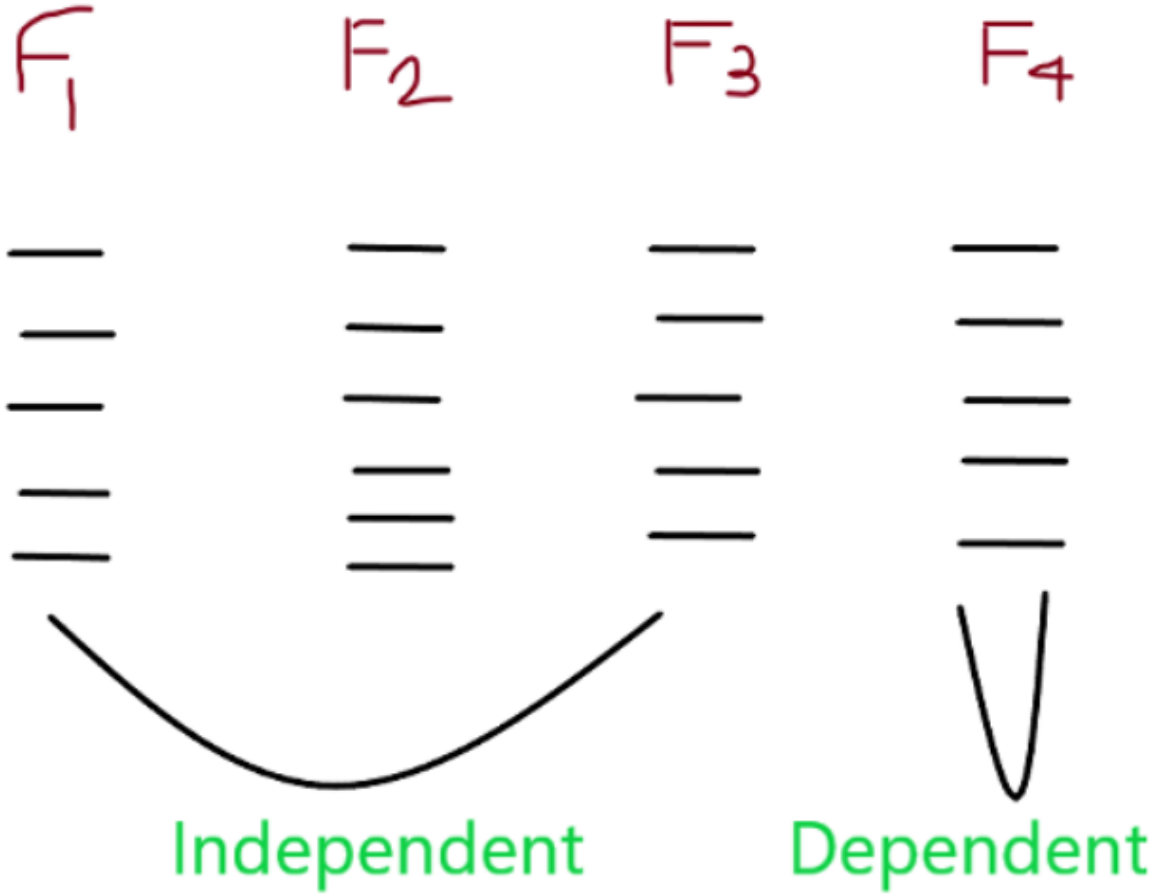
If we consider the first 3 steps, then it will probably be more towards Data Preprocessing, and Model Creation is more towards Model Training. So these are the two most important steps whenever we want to deploy any machine learning application.

TRANSFORMER IN SKLEARN

Scikit-learn has an object, usually, something called a **Transformer.** The use of a transformer is that it will be performing data preprocessing and feature transformation, but in the case of model training, we have learning algorithms like linear regression, logistic regression, knn, etc., if we talk about the examples of Transformer-like **StandardScaler,** which helps us to do feature transformation where it converts the feature with mean =**0** and standard deviation =**1**, **PCA**, **Imputer**, **MinMaxScaler,** etc. then all these particular techniques have seen that we are doing some preprocessing on the input data will change the format of training dataset, and that data will be used for model training.

Suppose we take **f1, f2, f3,** and **f4** features where f1, f2, and f3 are independent features, and f4 is our dependent feature. We apply a standardization process in which it takes a feature **F** and converts it into **F′** by applying a formula of standardization. If you notice, at this stage, we take one input feature F and convert it into another input feature F′ itself So, in this condition, we do three different operations:

1. **fit()**
2. **transform()**
3. **fit_transform()**


F1
F2
F3
F4
Independent
Dependent

Difference Between fit and fit_transform Sklearn

Method	Purpose	Syntax Example	
fit()	Learn and estimate the parameters of the transformation	`estimator.fit(X)`	`estimator.fit(train_data)`
transform()	Apply the learned transformation to new data	`transformed_data = estimator.transform(X)`	`transformed_data = estimator.transform(test_data)`
fit_transform()	Learn the parameters and apply the transformation to new data	`transformed_data = estimator.fit_transform(X)`	`transformed_data = estimator.fit_transform(data)`

Preprocessing

- **Rescale Data**
- **Normalization Data**
- **Binarize Data (Make Binary)**
- **Standardize Data**
- **Label Encoding**
- **Data Imputation**

1. Rescale Data

- When our data is comprised of attributes with varying scales, many machine learning algorithms can benefit from rescaling the attributes to all have the same scale.
- This is useful for optimization algorithms used in the core of machine learning algorithms like gradient descent.
- It is also useful for algorithms that weight inputs like regression and neural networks and algorithms that use distance measures like K-Nearest Neighbors.
- We can rescale your data using scikit-learn using the MinMaxScaler class.

- **MinMax Scaler equation:**

$$X_{new} = \frac{X_i - \min(X)}{\max(x) - \min(X)}$$

```
# Rescale Data
from pandas import read_csv
from numpy import set_printoptions
from sklearn import preprocessing
path = "C:\\Users\\lenovo\\ML codes\\pima-indians-diabetes.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(path, names=names)
array = dataframe.values

#In this example we will rescale the data of Pima Indians Diabetes dataset which we used earlier.
#First, the CSV data will be loaded  and then with the help of MinMaxScaler class,
# it will be rescaled in the range of 0 and 1

## initialising the MinMaxScaler
data_scaler = preprocessing.MinMaxScaler(feature_range=(0,1))

# learning the statistical parameters for each of the data and transforming
data_rescaled = data_scaler.fit_transform(array)

# summarize transformed data
set_printoptions(precision=1)

# data_rescaled[rows, columns]
# in the next line ..... this code return 5 rows and all columns
print ("\nScaled data:\n", data_rescaled[0:5, :])
```

2. Normalization Data

- Using the scikit-learn preprocessing.normalize() Function to Normalize Data

You can use the scikit-learn `preprocessing.normalize()` function to normalize an array-like dataset.

The `normalize()` function scales vectors individually to a unit norm so that the vector has a length of one. The default norm for `normalize()` is L2, also known as the Euclidean norm. The L2 norm formula is the square root of the sum of the squares of each value. Although using the `normalize()` function results in values between 0 and 1, it's not the same as simply scaling the values to fall between 0 and 1.

- Firstly, read csv file:

```
# read csv file
from pandas import read_csv
from numpy import set_printoptions
from sklearn.preprocessing import Normalizer
path = r'C:\Users\lenovo\ML codes\pima-indians-diabetes.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv (path, names=names)
array = dataframe.values
array
```

```
array([[  6. , 148. ,  72. , ...,   0.6,  50. ,   1. ],
       [  1. ,  85. ,  66. , ...,   0.4,  31. ,   0. ],
       [  8. , 183. ,  64. , ...,   0.7,  32. ,   1. ],
       ...,
       [  5. , 121. ,  72. , ...,   0.2,  30. ,   0. ],
       [  1. , 126. ,  60. , ...,   0.3,  47. ,   1. ],
       [  1. ,  93. ,  70. , ...,   0.3,  23. ,   0. ]])
```

L1 Normalization

```
#L1 Normalization
from pandas import read_csv
from numpy import set_printoptions
from sklearn.preprocessing import Normalizer
path = r'C:\Users\lenovo\ML codes\pima-indians-diabetes.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv (path, names=names)
array = dataframe.values

Data_normalizer = Normalizer(norm='l1').fit(array)
Data_normalized = Data_normalizer.transform(array)
```

```
Data_normalized
```

```
array([[0.  , 0.4, 0.2, ..., 0.  , 0.1, 0. ],
       [0.  , 0.4, 0.3, ..., 0.  , 0.1, 0. ],
       [0.  , 0.6, 0.2, ..., 0.  , 0.1, 0. ],
       ...,
       [0.  , 0.3, 0.2, ..., 0.  , 0.1, 0. ],
       [0.  , 0.5, 0.2, ..., 0.  , 0.2, 0. ],
       [0.  , 0.4, 0.3, ..., 0.  , 0.1, 0. ]])
```

L2 Normalization

```
#L2 normalization
from pandas import read_csv
from numpy import set_printoptions
from sklearn.preprocessing import Normalizer
path = r'C:\Users\lenovo\ML codes\pima-indians-diabetes.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv (path, names=names)
array = dataframe.values

Data_normalizer = Normalizer(norm='l2').fit(array)
Data_normalized = Data_normalizer.transform(array)
```

```
Data_normalized
```

```
array([[0.  , 0.8, 0.4, ..., 0.  , 0.3, 0. ],
       [0.  , 0.7, 0.6, ..., 0.  , 0.3, 0. ],
       [0.  , 0.9, 0.3, ..., 0.  , 0.2, 0. ],
       ...,
       [0.  , 0.7, 0.4, ..., 0.  , 0.2, 0. ],
       [0.  , 0.8, 0.4, ..., 0.  , 0.3, 0. ],
       [0.  , 0.7, 0.6, ..., 0.  , 0.2, 0. ]])
```

3. Binarize Data (Make Binary)

- We can transform our data using a binary threshold. All values above the threshold are marked 1 and all equal to or below are marked as 0.
- This is called binarizing your data or threshold your data. It can be useful when you have probabilities that you want to make crisp values. It is also useful when feature engineering and you want to add new features that indicate something meaningful.
- We can create new binary attributes in Python using scikit-learn with the Binarizer class.

```python
#Binarization.....................................

from pandas import read_csv
from sklearn.preprocessing import Binarizer
path = r"C:\Users\lenovo\ML codes\pima-indians-diabetes.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(path, names=names)
array = dataframe.values

binarizer = Binarizer(threshold=0.5).fit(array)
Data_binarized = binarizer.transform(array)
#Here, we are showing the first 5 rows in the output.
print ("\nBinary data:\n", Data_binarized [0:5])
```

```
Binary data:
 [[1. 1. 1. 1. 0. 1. 1. 1. 1.]
 [1. 1. 1. 1. 0. 1. 0. 1. 0.]
 [1. 1. 1. 0. 0. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 0. 1. 0.]
 [0. 1. 1. 1. 1. 1. 1. 1. 1.]]
```

4. Standardize Data

- Standardization is a useful technique to transform attributes with a Gaussian distribution and differing means and standard deviations to a standard Gaussian distribution with a mean of 0 and a standard deviation of 1.
- We can standardize data using scikit-learn with the StandardScaler class.

- Standard Scaler equation:

$$z = \frac{x_i - \mu}{\sigma}$$

```python
#Standrization .....................
from sklearn.preprocessing import StandardScaler
from pandas import read_csv
from numpy import set_printoptions
path = r"C:\Users\lenovo\ML codes\pima-indians-diabetes.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(path, names=names)
array = dataframe.values
#Now, we can use StandardScaler class to rescale the data.
data_scaler = StandardScaler().fit(array)
data_rescaled = data_scaler.transform(array)
# We can also summarize the data for output as per our choice.
# Here, we are setting the precision to 2 and showing the first 5 rows in the output.
set_printoptions(precision=2)
print ("\nRescaled data:\n", data_rescaled [0:5])
```

```
Rescaled data:
 [[ 0.64  0.85  0.15  0.91 -0.69  0.2   0.47  1.43  1.37]
 [-0.84 -1.12 -0.16  0.53 -0.69 -0.68 -0.37 -0.19 -0.73]
 [ 1.23  1.94 -0.26 -1.29 -0.69 -1.1   0.6  -0.11  1.37]
 [-0.84 -1.   -0.16  0.15  0.12 -0.49 -0.92 -1.04 -0.73]
 [-1.14  0.5  -1.5   0.91  0.77  1.41  5.48 -0.02  1.37]]
```

5. label encoding

- **Label Encoding** refers to converting the labels into a numeric form so as to convert them into the machine-readable form. Machine learning algorithms can then decide in a better way how those labels must be operated. It is an important pre-processing step for the structured dataset in supervised learning.

Example :

Suppose we have a column *Height* in some dataset.

Height
Tall
Medium
Short

After applying label encoding, the Height column is converted into:

Height
0
1
2

where 0 is the label for tall, 1 is the label for medium, and 2 is a label for short height.

```python
#Label encoding
import numpy as np
from sklearn import preprocessing
#Now, we need to provide the input labels as follows –
input_labels = ['red','black','red','green','black','yellow','white']
#The next line of code will create the label encoder and train it.
encoder = preprocessing.LabelEncoder()
encoder.fit(input_labels)

#The next lines of script transform the labels into numeric
encoded_values = encoder.transform(input_labels)
print("\nLabels =", input_labels)
print("Encoded values =", list(encoded_values))

decoded_lst = encoder.inverse_transform(encoded_values)
#We can get the list of encoded values with the help of following python script –
print("\nEncoded values =", encoded_values)
print("\nDecoded labels =", list(decoded_lst))
```

```
Labels = ['red', 'black', 'red', 'green', 'black', 'yellow', 'white']
Encoded values = [2, 0, 2, 1, 0, 4, 3]

Encoded values = [2 0 2 1 0 4 3]

Decoded labels = ['red', 'black', 'red', 'green', 'black', 'yellow', 'white']
```

Scikit-learn

- **What is Scikit-learn?**
  - Scikit-learn, or sklearn, is a machine learning package for Python built on top of SciPy, Matplotlib and NumPy.
- **Why Use Sklearn?**
  - One of the major advantages of Scikit-learn is that it can be used for many different applications such as Classification, Regression, Clustering, NLP and more.
  - Sklearn not only has a vast range of functionalities but also has very thorough documentation, making the package easy to learn and use.

Data Imputation

- **What is Imputation?**
- Imputation is a technique used for replacing the missing data with some substitute value to retain most of the data/information of the dataset. These techniques are used because removing the data from the dataset every time is not feasible and can lead to a reduction in the size of the dataset to a large extend, which not only raises concerns for biasing the dataset but also leads to incorrect analysis.

Why Imputation is Important?

1. **Incompatible with most of the Python libraries used in Machine Learning:-** Yes, you read it right. While using the libraries for ML(the most common is skLearn), they don't have a provision to automatically handle these missing data and can lead to errors.
2. **Distortion in Dataset:-** A huge amount of missing data can cause distortions in the variable distribution i.e it can increase or decrease the value of a particular category in the dataset.
3. **Affects the Final Model:-** the missing data can cause a bias in the dataset and can lead to a faulty analysis by the model.

- A popular approach for data imputation is to calculate a statistical value for each column (such as a mean) and replace all missing values for that column with the statistic. It is a popular approach because the statistic is easy to calculate using the training dataset and because it often results in good performance.
- Common statistics calculated include:
  - The column mean value.
  - The column median value.
  - The column mode value.
  - A constant value.

How to impute missing values with means in Python?

1. Import libraries
2. Load dataset for code running
3. Using imputer to fill the nun values with mean

- **Using Imputer to fill the nun values with the Mean:**
- We know that we have few nun values in column C1 so we have to fill it with the mean of remaining values of the column. So for this we will be using Imputer function, so let us first look into the parameters.
  - **missing_values** : In this we have to place the missing values and in pandas it is 'NaN'.
  - **strategy** : In this we have to pass the strategy that we need to follow to impute in missing value it can be mean, median, most_frequent or constant. By default it is mean.
  - **fill_value** : By default it is set as none. It is used when the strategy is set to constant then we have to pass the value that we want to fill as a constant in all the nun places.
  - **axis** : In this we have to pass 0 for columns and 1 for rows.

So we have created an object and called Imputer with the desired parameters. Then we have fit our dataframe and transformed its nun values with the mean and stored it in imputed_df. Then we have printed the final dataframe.

```
miss_mean_imputer = Imputer(missing_values='NaN', strategy='mean', axis=0)

miss_mean_imputer = miss_mean_imputer.fit(df)

imputed_df = miss_mean_imputer.transform(df.values)

print(imputed_df)
```

Example 1:

- Write the code using imputer to fill the nun values with mean in the following matrix:

```
data = [[1,2,np.nan],
        [3,np.nan,1],
        [5,np.nan,0],
        [np.nan,4,6 ],
        [5,0,np.nan],
        [4,5,5]]
```

Code:

```
import numpy as np

from sklearn.impute import SimpleImputer

data = [[1,2,np.nan],
        [3,np.nan,1],
        [5,np.nan,0],
        [np.nan,4,6 ],
        [5,0,np.nan],
        [4,5,5]]

data = np.array(data)
print(data)


imp = SimpleImputer(missing_values=np.nan, strategy='mean')
imp = imp.fit(data)

print("----------------------")

modifieddata = imp.transform(data)
print(modifieddata)
```

Output:

```
[[ 1.  2. nan]
 [ 3. nan  1.]
 [ 5. nan  0.]
 [nan  4.  6.]
 [ 5.  0. nan]
 [ 4.  5.  5.]]
-----------------------
[[1.   2.   3.  ]
 [3.   2.75 1.  ]
 [5.   2.75 0.  ]
 [3.6  4.   6.  ]
 [5.   0.   3.  ]
 [4.   5.   5.  ]]
```

Example2:

- Write the code using imputer to fill the nun values with median in the following matrix:

```
data = [[1,2,np.nan],
        [3,np.nan,1],
        [5,np.nan,0],
        [np.nan,4,6 ],
        [5,0,np.nan],
        [4,5,5]]
```

Code:

```
import numpy as np

from sklearn.impute import SimpleImputer

data = [[1,2,np.nan],
        [3,np.nan,1],
        [5,np.nan,0],
        [np.nan,4,6 ],
        [5,0,np.nan],
        [4,5,5]]

data = np.array(data)
print(data)

imp = SimpleImputer(missing_values=np.nan, strategy='median')
imp = imp.fit(data)

print("-----------------------")

modifieddata = imp.transform(data)
print(modifieddata)
```

Output:

```
[[ 1.  2. nan]
 [ 3. nan  1.]
 [ 5. nan  0.]
 [nan  4.  6.]
 [ 5.  0. nan]
 [ 4.  5.  5.]]
------------------------
[[1. 2. 3.]
 [3. 3. 1.]
 [5. 3. 0.]
 [4. 4. 6.]
 [5. 0. 3.]
 [4. 5. 5.]]
```

Example3:

- Write the code using imputer to fill the nun values with mean in the following matrix:

```
data = [[1,2,0],
        [3,0,1],
        [5,0,0],
        [0,4,6],
        [5,0,0],
        [4,5,5]]
```

Code:

```
from sklearn.impute import SimpleImputer


data = [[1,2,0],
        [3,0,1],
        [5,0,0],
        [0,4,6],
        [5,0,0],
        [4,5,5]]

data = np.array(data)
print(data)

imp = SimpleImputer(missing_values=0, strategy='mean')
imp = imp.fit(data)


modifieddata = imp.transform(data)
print("----------------------")
print(modifieddata)
```

Output:

```
[[1 2 0]
 [3 0 1]
 [5 0 0]
 [0 4 6]
 [5 0 0]
 [4 5 5]]
----------------------
[[1.         2.         4.        ]
 [3.         3.66666667 1.        ]
 [5.         3.66666667 4.        ]
 [3.6        4.         6.        ]
 [5.         3.66666667 4.        ]
 [4.         5.         5.        ]]
```

Example4: Write the code using imputer to load a breast cancer dataset and replace nun values with mean in the following matrix:

```
# Import Libraries
from sklearn.datasets import load_breast_cancer
from sklearn.impute import SimpleImputer
import numpy as np
#-----------------------------------------------------
#load breast cancer data

BreastData = load_breast_cancer()

#X Data
X = BreastData.data
#y Data
y = BreastData.target

#-----------------------------------------------------
# Cleaning data
'''
impute.SimpleImputer(missing_values=nan, strategy='mean', fill_value=None, verbose=0, copy=True)
'''

ImputedModule = SimpleImputer(missing_values = np.nan, strategy ='mean')
ImputedX = ImputedModule.fit(X)

#X Data
print('X Data is \n' , X[:10])

#y Data
print('y Data is \n' , y[:10])
```

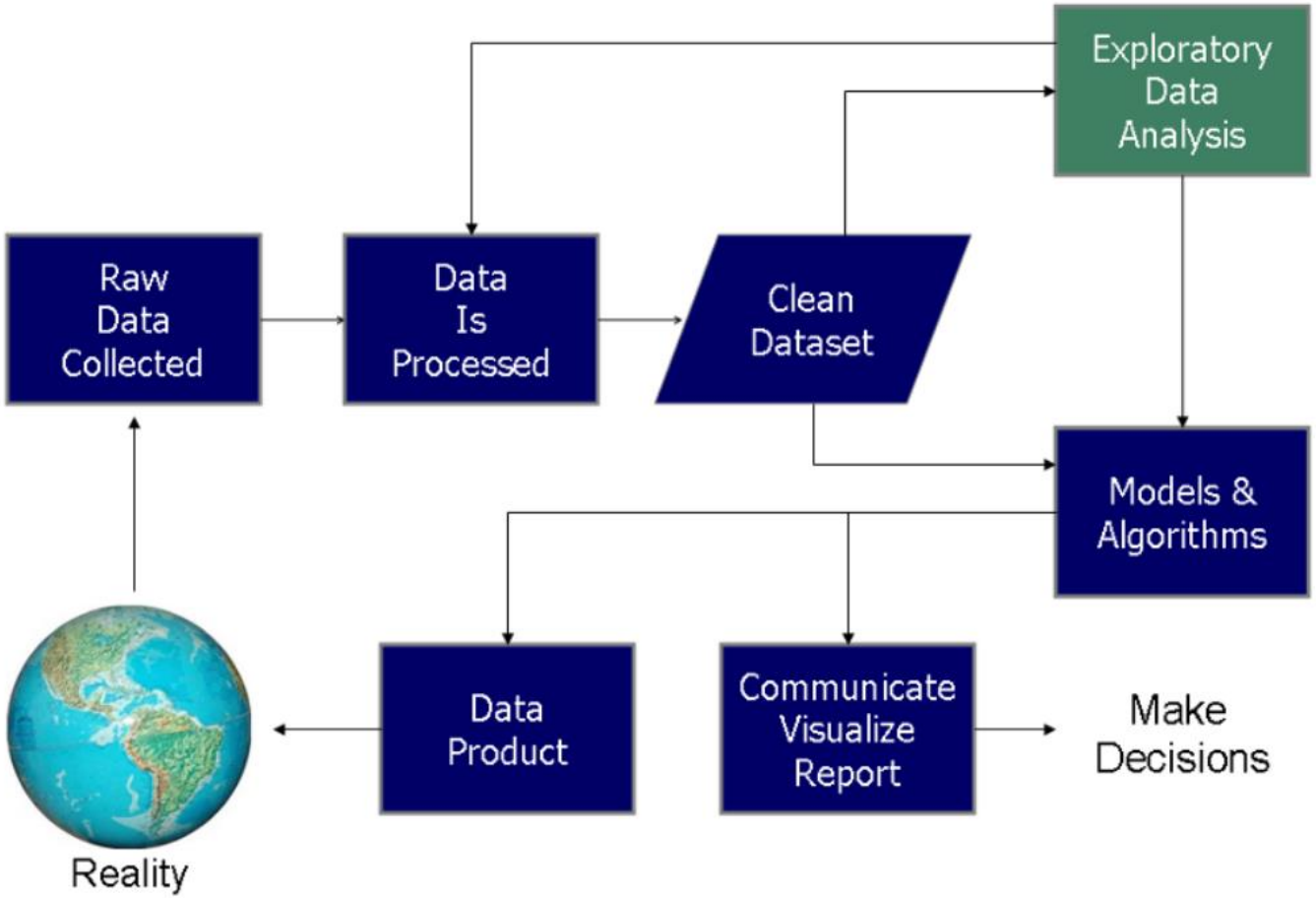
Output:

```
X Data is
 [[1.799e+01 1.038e+01 1.228e+02 1.001e+03 1.184e-01 2.776e-01 3.001e-01
  1.471e-01 2.419e-01 7.871e-02 1.095e+00 9.053e-01 8.589e+00 1.534e+02
  6.399e-03 4.904e-02 5.373e-02 1.587e-02 3.003e-02 6.193e-03 2.538e+01
  1.733e+01 1.846e+02 2.019e+03 1.622e-01 6.656e-01 7.119e-01 2.654e-01
  4.601e-01 1.189e-01]
 [2.057e+01 1.777e+01 1.329e+02 1.326e+03 8.474e-02 7.864e-02 8.690e-02
  7.017e-02 1.812e-01 5.667e-02 5.435e-01 7.339e-01 3.398e+00 7.408e+01
  5.225e-03 1.308e-02 1.860e-02 1.340e-02 1.389e-02 3.532e-03 2.499e+01
  2.341e+01 1.588e+02 1.956e+03 1.238e-01 1.866e-01 2.416e-01 1.860e-01
  2.750e-01 8.902e-02]
 [1.969e+01 2.125e+01 1.300e+02 1.203e+03 1.096e-01 1.599e-01 1.974e-01
  1.279e-01 2.069e-01 5.999e-02 7.456e-01 7.869e-01 4.585e+00 9.403e+01
  6.150e-03 4.006e-02 3.832e-02 2.058e-02 2.250e-02 4.571e-03 2.357e+01
  2.553e+01 1.525e+02 1.709e+03 1.444e-01 4.245e-01 4.504e-01 2.430e-01
  3.613e-01 8.758e-02]
 [1.142e+01 2.038e+01 7.758e+01 3.861e+02 1.425e-01 2.839e-01 2.414e-01
  1.052e-01 2.597e-01 9.744e-02 4.956e-01 1.156e+00 3.445e+00 2.723e+01
  9.110e-03 7.458e-02 5.661e-02 1.867e-02 5.963e-02 9.208e-03 1.491e+01
  2.650e+01 9.887e+01 5.677e+02 2.098e-01 8.663e-01 6.869e-01 2.575e-01
  6.638e-01 1.730e-01]
```

Exploratory Data Anlysis

- Exploratory Data Analysis: is an approach in analyzing data sets to summarize their main characteristics, often using statistical graphics and other data visualization methods. EDA assists Data science professionals in various ways:-
  - Getting a better understanding of data.
  - Identifying various data patterns

Data Science Process



Thank you