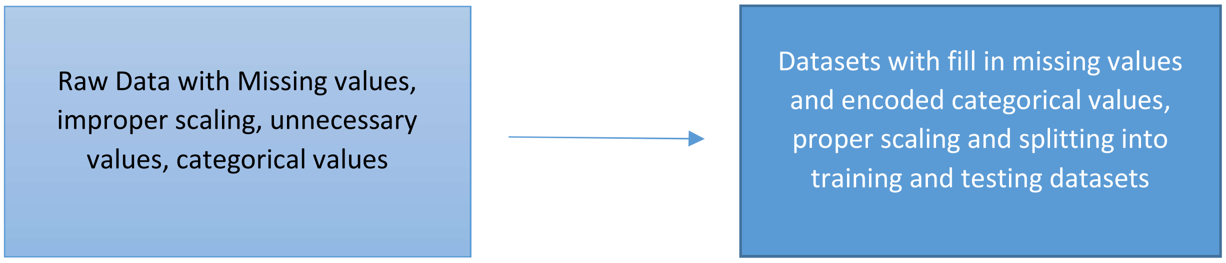
Data Preprocessing

Data Preprocessing: Data preprocessing is an important step of solving every machine learning problem. Most of the datasets used with Machine Learning problems need to be processed / cleaned / transformed so that a Machine Learning algorithm can be trained on it.

Data Preprocessing is processing of raw data into Datasets (meaningful data)


Raw Data with Missing values, improper scaling, unnecessary values, categorical values
Datasets with fill in missing values and encoded categorical values, proper scaling and splitting into training and testing datasets

Why Data Preprocessing?

Real world data are generally

1. Incomplete: lacking attribute values, lacking certain attributes of interest, or containing only aggregate data
2. Noisy: containing errors or outliers
3. Inconsistent: containing discrepancies in codes or names

Steps for Data Preprocessing:

1. Modeling Overview
2. Introduce Data
   I. Import Libraries
   II. Import Datasets
3. Basic Data Cleaning
   I. Dealing with Data Types
   II. Handling Missing Data
   III. Handling Categorical data
4. Scaling the Features
5. Assessing the model performance

Check out the complete Implementation at: https://github.com/KoushikMuthakana/MachineLearning

@Koushik Muthakana

Data Preprocessing

1. **Modeling Overview:** Statistical technique to predict the unknown outcome
   - Credit card transaction is fraud or not
   - Whether or not person will be diagnosed with a given disease in next year

   A. **Data Terminology:**
   - **Features:** Independent Variables, also called Inputs. These are the predictors
   - **Outcomes:** Dependent variables for predictions also called ad output.

2. **Introduce Data:**

   I. **Import Libraries:**
   - In python we have many libraries available to preprocess the raw data in to Datasets
   - Import appropriate libraries for data preprocessing, here we use scientific libraries Pandas, Numpy, matplotlib, Sklearn

   Steps to import libraries:

   i. pip install pandas numpy matplotlib sklearn
   ii. import pandas as pd
   iii. import numpy as np
   iv. import matplotlib as plt
   v. import sklearn (Import needed classes)

   II. **Import DataSets:**
   - Pandas library is used to import the datasets from csv, excel format data and splitting the data in to dependent and independent datasets
     i. Datasets=pd.read_csv('data_file_name.csv')
        #Independent variables
     ii. X=datasets.iloc[:,:-1].values
        #Dependent variables
     iii. Y=datasets.iloc[:,:3].values

3. **Basic Data Cleaning:** There are three main data types

   I. **Dealing with Datatypes**
   - There are three main datatypes
     1. Numeric : eg income, age
     2. Categorical : eg: Gender, nationality
     3. Ordinal eg: high/low/medium
   - Models can only handle numeric features
   - Must convert categorical and ordinal features
   - Create dummy variables.

- Transform categorical features into set of dummy features ,each represents a unique categorical

Imp Notes:

- Decide which categorical variables want to use in model
- Convert low frequency categories into others, if there are many categories.
- Drop the original and replace with dummy variables

II. Handling Missing Data:

- Models cannot handle missing data.
  Simplest solution: Remove observations/features that have missing data but removing data can introduce a lot of issues.
  Alternate Solution: Use imputation, replace missing values with another values by using any strategies 'MEAN/MEDIAN/HighFrequency'
- There are many reasons for missing data such as data is not continuously collected, a mistake in data entry, technical problems with biometrics and much more.
- Imputer method in sklearn.preprocessing is used to fill the missing data with mean, median, most frequent values.
  Note: If more than 70% attributes contains missing values, it's better to drop the columns rather than filling the data.

Steps to handle the missing Data:

from sklearn.preprocessing import Imputer

imputer=**Imputer(missing_values='**NaN**',strategy='**mean/median/
most_frequent**',axis=**0/1**)**

0=impute along columns

1=impute along with rows

X [:,:-1]=imputer.fit_transpose(X[:,:-1])

Y [: 3]=imputer.fit_transpose(Y[:,:3])

Note: When column doesn't have missing values - It's possible that a column doesn't have any null values in the train dataset, but it's very possible that it might have null values in test dataset. Hence, it's important to review the columns/data and perform missing value imputation of all columns that can possibly have missing values, even if the train dataset doesn't have any missing values

III. **Handling Categorical data:**

- Converts the categorical data into numbers like country, purchased into numbers
- As ML deals with the equations, so strings need to convert in to numbers.

Steps to handle Categorical Data:

```
from sklearn.preprocessing import LabelEncoder,OneHotEncoder

X_le=LabelEncode()

X [: 1] =X_le.fit_transpose(X [:,1])
```

Eg:

Yes	No
1	0
0	1

- If categorical data contains more than 2, need to use dummy encoders, which can be achieved with OneHotEncoder

```
Onehot_x=OneHotEncoder(categorical_feature=[0])

x=Onehot_x.fit_transform(X)
```

Eg:

India	USA	Russia
1	0	0
0	1	0
0	0	1

4. **Feature Scaling:**

- This is the important feature because the values are not in same range. Feature scaling is the method to limit the range of variables so that they can be compared on common grounds. It is performed on continuous variables.

Eg: consider Salary and Age

Age=25 salary=23,300

Age=35 salary =1,23,000

Steps for scaling:

```
from sklearn.preprocessing import StandardScaler

Sd=StandardScaler()

X_train=sd(X_train)

Y_train=Sd(Y_train)
```

Note: Most of the libraries take care scaling features

5. **Assessing model performance:**

- ML is learning from data and predicting the data or other ML goals
- The Datasets is split into two sets, Train and Test sets. ML models/Algorithms learns from train sets and ML tests on test sets

Steps to split the Data:

```
frP a g e | 5om 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)
```

Note: test_size should be always <0.5

Reference: Kirill Eremenko @Udemy Instructor, YouTube, Google.