Data Preprocessing means processing or importing data before making model.

A Data Set contains independent and dependent variables.

- Inputs are independent
- Outputs are dependent

In Python we need to import three libraries

1. import numpy as np - all mathematical tools

2. import matplotlib.pyplot as plt - used for plotting charts

3. import pandas as pd - data set management and import

For reading dataset

dataset=pd.read_csv("xyz.csv")

Note :: Python file should be in same folder as csv file

We will make matrix of independent variable and dependent variable differently

for this:

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

Note :: The first : before comma means that traverse all the rows from start_row:end,column_start:till second last i.e -1

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

Note :: only fetch 3rd column and all the rows

Missing Data

So our data.csv can contains rows in which one or more column data is missing so we can do is we can take the average of all the

values in that column and find mean and put it at missing place.

So to compute mean we will use an library known as **sikat learn preprocessing**

from sklearn.preprocessing import Imputer

ctrl+i to see arguments of imputer

imputer=Imputer(missing_values="NaN",strategy="mean",axis=0)

Note :: missing_values in matrix are denoted by nan axis=0 is to compute mean of column

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

Note :: fitting the object to matrix

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

Note :: transforming i.e changing the missing values in X to mean values ie assigning

Categorical data

some columns can be divided into categories like an country column data can be categorized acc to diff countries or dependent

variable can be categorized.

We will use **labelencoder** class and **onehotencoder** to create dummy set.

Country age

France 12

Germany 13

So labelencoder will encode these

Country Age

0 12

1 13

But here the problem is that algo might think and start comparing this encoding as numbers like 0>1 which means france is greater than germany which we don't want so we need to create dummy set in this case i.e

France Germany Age

1 0 12

0 1 13

This will help to filter data and categorization

from sklearn.preprocessing import LabelEncoder,OneHotEncoder

labelEncoder_x=LabelEncoder()

X[:,0]=labelEncoder.fit_transform(X[:,0])

onehot=OneHotEncoder(categorical_featues=[0])

X=onehot.fit_transform(X).toarray()

Note :: here we don't specify column in X as we already passed column to categorize or hot encode in above initialization

Now after all this we need to divide the dataset into training and test case so for this

from sklearn.cross_validation import train_test_split

X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=0)

Feature Scaling

So now we see that in our dataset some column may have higher values and some lower. since ml models are based on euclidean

distances therefore squaring large values will result in dominance over others

age salary

10 10000

15 30000

so here if we apply euclidean then (15-10)^2+(30000-10000)^2= 25+(20000)^2 so 20000 term will dominate the 25 value so it should

not happen. So we need to do feature scaling so that all values in dataset come to same scale

from sklearn.preprocessing import StandardScaler

sc_X=StandardScaler()

X_train=sc_X.fit_transform(X_train)

X_test=sc_X.transform(X_test)

we don't need to fit the object to test case as we already fitted it to training tests so if we want same scale in both training and

testing we don't fit object to train test.

Simple Linear Regression

now to start doing linear regression we import

from sklearn.linear_model import LinearRegression

regressor=LinearRegression()

regressor.fit(X_train,y_train)

Note :: let the object fits to our model i.e. find dependencies

y_pred=regressor.predict(X_test)

Note :: predict the values on basis of learned dependencies by regressor and storing in array

plt.scatter(X_train,y_train,color='red')

Note :: scatter plot to point out real values by red marks on training set

plt.plot(X_train,regressor.predict(X_train),color='blue')

Note :: plotting the regression line on basis of training set and predicted

values of training set

plt.title title of plot

plt.xlabel xlabel of plot

plt.ylabel

plt.show() ---to tell we are ready and show the plot

plt.scatter(X_test,y_test,color='red')

Note :: same as above for test case

plt.plot(X_train,regressor.predict(X_train),color='blue')

Very Important Note :: here we don't need to change the regression line as above we already found the regression line when we plotted the training set...the regression line will be same for sure.

plt.title

plt.xlabel

plt.ylabel

plt.show()