Data Preprocessing

Process summarizes necessary steps required to process data.

Importing Required Libraries

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

- Import numpy to work with Arrays
- Import matplotlib to plot data
- Import pandas which allow us to matrix of features and import datasets

Importing Dataset

```
dataset = pd.read_csv('Data.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
```

- Creates the data frame object called `dataset`, which contains the dataset including columns and rows with respective data.
- Features are the columns of data which we are to use and predict the outcome
- Dependent variable is the last column which shows the actual outcome values
- X in this case would be all the columns except last, using `iloc` which extracts indexes of rows/columns. To do this, lower bound = empty and upper bound is -1.
- Y in this case is the dependent variable and is in most cases the last columns, to get that we need the all the rows and the last column, to do so we add the value of -1.

Take care of Missing Data

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

- It is important to deal with missing values in datasets. If you dataset is large, you can opt to delete that entire row which contains the missing data. But if dataset is smaller, this may

lead to errors in model learning.

- Another fix for this issue, we import the `SimpleImputer` class from `sklearn.impute` in which we replace missing values with the average of all the values.
- We create the instance of this class and specify the strategy, in this case the mean or average. We also specify which missing values we want to replace, so we use `np.nan`.
- We use the `fit` method to connect with matrix of features(also known as `X`, meaning if will get the row/column and calculate mean. First it requires all columns with numerical values (not strings). So we pass in `X[: , 1:3]`, meaning it checks all rows and for a test dataset, the 2nd and 3r column.
- Lastly we use the transform method replace the missing values with the mean value in each respective column. The transform method replaces the values and returns the new values for the columns affected. So it is crucial to remember to update the value of the columns with the new values from input function.

Encoding Categorical Data

Encoding Independent Variable

- To further improve the learning of the model, replace the strings in the independent variable with numerical values. To do this we assign each categorical data with a numerical value.
  For example:

  ```
  France > 1
  Spain > 2
  Canada > 3
  ...
  ```

  But now the learning model will interpret that this is a categorical hierarchy between Spain, Canada and France. Which is not the case, hence we have to encode this in such way where the model does not falsely interpret such nuances.
  To do this, we use `OneHotEncoder`, a encoding practice widely and commonly used.

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [0])],
remainder='passthrough')
X = np.array(ct.fit_transform(X))
```

- We use the `ColumnTransformer` class and pass in the two parameters `transformers` and `remainders`. `transformers` parameter is used to distinguish the type of transformation, in

our case `encoding`. We also pass in the class used for encoding, `OneHotEncoder()`, and lastly the column we want to apply this transformation to, lets say the first column, `[0]`. Next parameter is `remainder`, which is used to specify that the remaining columns don't get wiped out after this transformation is applied, which is why we pass in `passthrough`; otherwise pass in `drop`.
- Next apply transformation and get the updated value as a numpy array object back and store as the independent matrix of features.

Encoding Dependent Variable

- Useful to encode the dependent variable if they are in `(yes/no)` or `(true/false)` string formats to numerical values such as `(1/0)`.

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y = le.fit_transform(y)
```

- We don't need to set any parameters for `LabelEncoder`, as it understands there is only two values, similar to binary data where `Yes == 1` and `No == 0`.

Split Data into Training/Test Sets

- It is important to split the dataset into `Training` and `Test` sets. Process is fairly simple, split current set into two sets, one for training the model with existing values, and the other set is used for testing and evaluating the model on new observations.

```
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 = 1)
```

- We use `train_test_split` as a way to split up the training and testing datasets, where we have 4 variables:
  `X_train` > independent matrix of features for the training set
  `X_test` > independent matrix of features for the test set
  `y_train` > dependent variable for the training set
  `y_test` > dependent variable for the testing set
- It is common to divide the sets with a ratio of 8:2, where 80% of the dataset is used for training the model, and 20% is used to test the model. The goal is, the more training dataset data we have, the smarter and better the model will be.
- Random state is used to simply set the seed to the random generator, so that training and testing sets are similar each time, otherwise the two sets will be different every time we run

this.

Apply Feature Scaling

- It is crucial to apply feature scaling after splitting data into training and test sets. Reasoning is fairly simple, the test set is supposed to be a new set which we use to evaluate the trained model on. Meaning it should not have any work done on it, and feature scaling applies standard deviations or normalization.
- Two feature scaling techniques, which ensure that all features are on the same scale.
  - Standardization = Process is taking the average μ of all the values in the column, and subtracting each value by the average and dividing it by the standard deviation σ
    - `x_stand` = `x - mean(x)` / `standard deviation (x)`
    - All values will be ranging from `[-3:3]`
  - Normalization = Process of taking the Minimum of the column and subtracting it from every single value in that column and than dividing by the difference of the Maximum and Minimum.
    - `x_norm` = `x - min(x)` / `max(x) - min(x)`
    - All values will be ranging from `[-1:1]`
  - So the question is, which should we use? The answer is, when you have a normal distribution in most features; you use `Normalization`. But `Standardization` can be used anytime as it is always effective.

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train[:, 3:] = sc.fit_transform(X_train[:, 3:])
X_test[:, 3:] = sc.transform(X_test[:, 3:])
```

- Import the `StandardScaler` class, which will help us perform `Standardization` on both matrix of features of test and training sets.
- It is important to note, we do not apply feature scaling on the dummy values created by encoding the categorical data as they are already between (-3, 3) and in fact it will make the encoding worthless.
- Only apply feature scaling on the columns which have numerical values and not the dummy variables created by encoding the columns which were non-numerical.
- Take all rows and for columns take the all columns which are of numerical of origin. Since `OneHotEncoder` has creates 3 variables for encoding, usually we take the 4rth column and so on, which in python is index of 3. Hence `X_train[:, 3:]` is used.
- We take those columns and apply `Standardization` on all the values in the specified ranges of rows and columns, including the `mean`, `standard deviation`, `min` and `max`. Using the `fit_transform` which will get the (functionality of fit:) `mean` and `standard`

`deviation` of the features, (functionality of transform) and will apply transformation on your values so that they are all on same scale.

- Now for the test data, we must treat it as new data. Meaning we cannot use the `fit` function to calculate the new `mean`, `standard deviation & etc`, as we must use the same which was used for the training model set. Hence all we need to do is apply the transform function, which will give us the dataset scaled using the same scaler which was used for training.

Next Steps -> Regression Topics

Regression Topics