Exploratory Data Analysis

Exploratory data analysis is the process of analyzing and interpreting datasets while summarizing their particular characteristics with the help of data visualization methods.

EDA assist in determining the best possible ways to manipulate data resources to obtain required interferences, making data easier to study and discover hidden trends, test a hypothesis and check assumptions. Moreover, the method scrutinizes data in order to deliver

Optimal interpretation into a dataset,

Unearth promising structures,

Identify outliers and anomalies,

Determine optimal factor settings,

Detect significant data variables and many more.

In 1970, originally created by John Tukey, an American mathematician, the EDA technique also helps in deciding where the selected statistical techniques are suitable or not for data analysis. It is a widely used method employed for data discovery processes in the present time.

Let's take an example to know more about EDA. I have taken two datasets, one from the Kaggle website which is called the Pima Indian diabetes database and another from UCI Machine Learning Repository that is the Iris dataset. Let us do EDA on both datasets.

1. Importing the datasets

import pandas as pd import matplotlib.pyplot as plt import seaborn as sns import warnings warnings.filterwarnings('ignore')

import pandas as pd pima_df = pd.read_csv('pima.csv') iris_df= pd.read_csv('iris.csv')

After downloading the dataset you can import your dataset using a function in pandas called pd.read_csv. You can read the full documentation of pandas here.

2. Printing the first 5 rows of the dataset to see the first view of the dataset

pima_df.head(5)

	Preg	Plas	Pres	skin	test	mass	pedi	age	class
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

printing 5 rows of the dataset to see the first view of the dataset

	Sepal Length (in cm)	Sepal Width in (cm)	Petal length (in cm)	Petal width (in cm)	Class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

iris_df.head(5)

printing 5 rows of the dataset to see the first view of the dataset

3. Shape of a dataset

The shape of the dataset is basically a representation of total rows and columns present in the dataset. You can explore .shape() function present in the pandas' package here. In the Pima diabetic dataset, we have 768 rows and 9 columns, similarly, in the iris dataset, we have around 150 rows and 5 columns.

print(pima_df.shape) print(iris_df.shape) shape of dataset

```
(768, 9)
(150, 5)
```

4. Descriptive statistics of the data-sets

In pandas, describe() function is used to view central tendency, mean, median, standard deviation, percentile & many other things to give you the idea about the data.

pima_df.describe().transpose()

	count	mean	std	min	25%	50%	75%	max
Preg	768.0	3.845052	3.369578	0.000	1.00000	3.0000	6.00000	17.00
Plas	768.0	120.894531	31.972618	0.000	99.00000	117.0000	140.25000	199.00
Pres	768.0	69.105469	19.355807	0.000	62.00000	72.0000	80.00000	122.00
skin	768.0	20.536458	15.952218	0.000	0.00000	23.0000	32.00000	99.00
test	768.0	79.799479	115.244002	0.000	0.00000	30.5000	127.25000	846.00
mass	768.0	31.992578	7.884160	0.000	27.30000	32.0000	36.60000	67.10
pedi	768.0	0.471876	0.331329	0.078	0.24375	0.3725	0.62625	2.42
age	768.0	33.240885	11.760232	21.000	24.00000	29.0000	41.00000	81.00
class	768.0	0.348958	0.476951	0.000	0.00000	0.0000	1.00000	1.00

.describe() function to view central tendency

iris_df.describe().transpose()

	count	mean	std	min	25%	50%	75%	max
Sepal Length (in cm)	150.0	5.843333	0.828066	4.3	5.1	5.80	6.4	7.9
Sepal Width in (cm)	150.0	3.054000	0.433594	2.0	2.8	3.00	3.3	4.4
Petal length (in cm)	150.0	3.758667	1.764420	1.0	1.6	4.35	5.1	6.9
Petal width (in cm)	150.0	1.198667	0.763161	0.1	0.3	1.30	1.8	2.5

Descriptive statistics of the data-sets

5. Checking about the correlation between features in a dataset

There is a function in the pandas's package which allows you to check about the correlation between features which is pd.DataFrame.corr(). It calculates the correlation between features pairwise excluding null values. I have used this function to compute the correlation between features in the Pima dataset which is shown in the below image.

pima_df.corr()

	Preg	Plas	Pres	skin	test	mass	pedi	age	class
Preg	1.000000	0.129459	0.141282	-0.081672	-0.073535	0.017683	-0.033523	0.544341	0.221898
Plas	0.129459	1.000000	0.152590	0.057328	0.331357	0.221071	0.137337	0.263514	0.466581
Pres	0.141282	0.152590	1.000000	0.207371	0.088933	0.281805	0.041266	0.239528	0.065068
skin	-0.081672	0.057328	0.207371	1.000000	0.436783	0.392573	0.183928	-0.113970	0.074752
test	-0.073535	0.331357	0.088933	0.436783	1.000000	0.197859	0.185071	-0.042163	0.130548
mass	0.017683	0.221071	0.281805	0.392573	0.197859	1.000000	0.140647	0.036242	0.292695
pedi	-0.033523	0.137337	0.041266	0.183928	0.185071	0.140647	1.000000	0.033561	0.173844
age	0.544341	0.263514	0.239528	-0.113970	-0.042163	0.036242	0.033561	1.000000	0.238356
class	0.221898	0.466581	0.065068	0.074752	0.130548	0.292695	0.173844	0.238356	1.000000

correlation between features in pima diabetic data-set

6. Checking about data types and more information about the data

There is a function present in the pandas' package known as pd.dataFrame.info() which returns the data type of each column present in the dataset. Also, it tells you about null and not null values present. So, in our dataset, we have even int64 data types values and also float64 data type values.

pima_df.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   Preg        768 non-null    int64  
 1   Plas        768 non-null    int64  
 2   Pres        768 non-null    int64  
 3   skin        768 non-null    int64  
 4   test        768 non-null    int64  
 5   mass        768 non-null    float64
 6   pedi        768 non-null    float64
 7   age         768 non-null    int64  
 8   class       768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

information about pima-india diabetic dataset using .info() function

7. Checking about missing values in the data

Missing values in the data can be checked by using isnull() function present in pandas documentation. It returns the boolean values that are true and false. If you want to calculate how many missing values are present in each column in the data set you can make use of the function isnull().sum(). This function returns the total number of missing values in each column.

In our case, in both the data sets we did not get any of the missing values in any of the columns.

pima_df.isnull().sum()

```
Preg     0
Plas     0
Pres     0
skin     0
test     0
mass     0
pedi     0
age      0
class    0
dtype: int64
```

Output of the isnull() function present in pandas to check about missing values

7.1 If missing values are present then how to impute them?

For various scenarios, while dealing with data you will come across real-world data which will have missing values like nan values, -, blanks. The basic approach to deal with such a situation is to drop/ remove the entire row or column which contains missing values.

But dropping is not advisable because there will be a loss of data as well which can result in important parts of the data being removed. So, to deal with such things there are different methods used to impute the missing values.

There are two ways by which missing values can be imputed: the first is called univariate imputation and the other one is multivariate imputation.

Univariate imputation is a type of imputation which imputes missing values considering only the non-missing values in that feature dimension. (e.g. impute.SimpleImputer). On the other hand, a multivariate imputer imputes the missing values considering all available features dimensions (e.g. impute.IterativeImputer).

8. Encoding categorical features

Often it is seen that we do not have continuous values in our features. There are sometimes categorical values. And the system cannot understand such values so there is a need to convert them to continuous numerical values.

As seen in the below iris data frame we have classes as categorical features which are - 'iris-setosa', 'iris-versicolor', 'iris-virginica'.

How to encode them?

There are several different techniques that are used to encode categorical values which are stated below:

a) LabelEncoder() -

It is a function present in the scikit- learn library of python which is used to convert categorical values in numerical values.

from sklearn.preprocessing import LabelEncoder,OneHotEncoder LE = LabelEncoder() OE= OneHotEncoder() iris_df['Class'] = LE.fit_transform(iris_df['Class']) iris_df['Class'].unique()

```
array([0, 1, 2], dtype=int64)
```

hand-on implementation of encoding using LabelEncoder

Here we have imported LabelEncoder from sklearn.preprocessing followed by initialising of the object through which we will use the label encoder. We have made an object called "LE". Then we have transformed our class column by using the LE.fit_transform function & printed the transformed class which is now [0,1,2]. It has given the values to Iris-setosa - 0, Iris-versicolor - 1 , Iris-virginica - 2.

b) get_dummies() -

Converts categorical features into dummy variables.

c) OneHotEncoder() -

Array-like of integers or strings is the required input for this encoder. The features are encoded using a one-hot encoding scheme. The result is a binary column for each category and reverts a sparse matrix.

9. Standardization of data

Standardization of data is a major important step that is required for machine learning algorithms to give good results. There are different scaling functions present in the preprocessing module of sci-kit learn. If data is not scaled and is passed to the algorithm the result might be wrong due to wrongly distributed data.

Why is it important to scale the data?

It is usually seen that we ignore checking the shape of the data distribution and change the data to be centred. That is done by removing the mean values of each column and then scaling it by dividing non-constant columns by their standard deviation.

Different functions are used by algorithms to learn to assume that all the desired features are centred as zero and also their variance is in the same structure. If any of the features have a higher proportion than all other features it may dominate the function for learning algorithm and does not allow learning from other features as required.

a) Scale:

present in the pre-processing module gives a fast and effective way to do this operation on a single array-like data:

X= np.array([1. -1. 3], [5. 0. 0], [0. 2. -1])

X_scaled = preprocessing.scale(X_train) X_scaled

```
array([[ 0.        , -1.22474487,  1.13630621],
       [ 1.22474487,  0.        , -0.26726124],
       [-1.22474487,  1.22474487, -1.06904497]])
```

X_scaled has now unit variance and zero mean as you can see in the below image.

X_scaled.mean(axis=0)

array([0., 0., 0.])

b) The pre-processing module

it is also has different other classes like StandardScaler that are used in scaling the data that is converting the mean to be zero and standard deviation to be united on training data which can be further used in test data as well. Such a class can also be used in building pipelines also.

The code implementation of the standard scaler is shown below.

Y = np.array([1. -1. 2.], [2. 0. 0.], [0. 1. -1.]) Std = preprocessing.StandardScaler() Y_scaled = Std.fit_transform(Y) Y_scaled.mean(axis=0)

array([0., 0., 0.])

c) Scaling features to a range:

There are other methods also to scale data within a respective range that is a min value and max value. It mainly ranges between 0 and 1. You can use MinMaxScaler or MaxAbsScaler for scaling the data respectively.

d) Scaling sparse data:

Centering the scatter data would result in knock-down of sparsity structure of data thus it is not advisable to do. MinMaxScaler and MaxAbs scaler were introduced to scale the sparse data. Scalar often accepts both CSR (Compressed Sparse Rows) & also CSC (Compressed Sparse Columns).

If there is any other different sparse input then it is converted to Compressed Sparse Rows. To take care of the memory it is advisable to convert it in CSR and CSC representation.

e) Scaling data with the presence of outliers:

If the data has outliers in it then scaling that sort of data using mean and variance is not a good approach. You can use robust_scale & Robust_Scaler as drop-in substitution.

10. Normalization of data

It is the process of scaling each sample to have a unit standard. These types of techniques are much more effective if you are computing the similarity between different pairs of samples or using a quadratic form like a dot product. This is the base of models used in text classifications. As discussing text classification, learn more about text mining and text mining techniques.

There is a function in the pre-processing module that is normalized which provides a good way to execute such operations on single array-like data by using L1 or L2 standards. Implementation of normalizing data using normalize is shown in the below image.

X = [[2. -1. 1], [5. 1. 0], [0. 1. -1]]

X_normalized = preprocessing.normalize(X, norm='l2') X_normalized

```
array([[ 0.81649658, -0.40824829,  0.40824829],
       [ 0.98058068,  0.19611614,  0.        ],
       [ 0.        ,  0.70710678, -0.70710678]])
```

code to normalize the data using Normalize Function

Pre-processing module also has another class that is called a normalizer that executes similar operations using the transformer API. This class can also be used in the initial stage of the pipeline. Implementation of the normalizer is shown below in the image.

X = [[2. -1. 1], [5. 1. 0], [0. 1. -1]] norm= preprocessing.Normalizer() norm.fit_transform(X)

```
array([[ 0.81649658, -0.40824829,  0.40824829],
       [ 0.98058068,  0.19611614,  0.        ],
       [ 0.        ,  0.70710678, -0.70710678]])
```

code to normalize the data using Normalize Function

If you want to look for code implementation of EDA discussed above you can refer to the Github link here. It contains a jupyter file and both the datasets which are used.

Exploratory Data Analysis refers to the critical process of performing initial investigations on data so as to discover patterns,to spot anomalies, to test hypothesis and to check assumptions with the help of summary statistics and graphical representations.

Objective of EDA :

1. To check for missing data and other anamolies.
2. To gain maximum insight into the dat set and its underlying structure.
3. To check the distribution of the data.
4. Identify the most influential variables.

Conclusion

In this blog, I have tried to explain some operations which are done in exploratory data analysis to get a better understanding of the data. Techniques like missing values, standardization, normalization, shape, correlation between independent features also descriptive statistics of the data are discussed.

There can be various other things that can be done in EDA to get a better understanding that is dependent on what type of data we have. EDA in textual data or image data is entirely different which will be covered in different blogs dedicated to the image or textual data.

In [ ]:

reference by:https://www.analyticssteps.com/blogs/how-do-exploratory-data-analysis-building-machine-learning-models

In [ ]: