Jaya Aiyappan
Nov 25, 2019 · 5 min read

Exploratory Data Analysis and Data Pre-Processing

Getting the data ready

Data Pre-Processing is a technique that transforms raw real-world data into a format. that can be used by the machine learning algorithms.

Let's continue with the data set that we saw in my post Supervised Learning and perform EDA on it.

The data we have is that for Customer Churn Problem:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
0	1	15634602	Hargrave	619	France	Female	42	2	0.00	1	1	1	101348.88	1
1	2	15647311	Hill	608	Spain	Female	41	1	83807.86	1	0	1	112542.58	0
2	3	15619304	Onio	502	France	Female	42	8	159660.80	3	1	0	113931.57	1
3	4	15701354	Boni	699	France	Female	39	1	0.00	2	0	0	93826.63	0
4	5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	1	1	79084.10	0

Data Source : https://github.com/var97/Customer-Churn-Problem/blob/master/Churn_Modelling.csv

First, let's take a look at the number of data samples and the number of features that are data set contains

```
In [10]: data.shape
Out[10]: (10000, 14)
```

So, there are 10000 data points with 14 features.

Further, let's find out what the columns/features are

```
data.columns
```

```
Index(['RowNumber', 'CustomerId', 'Surname', 'CreditScore', 'Geography',
       'Gender', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'HasCrCard',
       'IsActiveMember', 'EstimatedSalary', 'Exited'],
      dtype='object')
```

'Exited' is the target column and we need to find out how this column is related and influenced by other columns.

From the above , we can remove a few columns based on the knowledge we have on the use case. In the example we have , we can remove the following columns because we believe that these columns will not have any impact on the target ('Exit')column.

1) RowNumber 2) Surname 3) CustomerId

```
data = data.drop(["RowNumber","Surname","CustomerId"],axis = 1)
data.columns
```

```
Index(['CreditScore', 'Geography', 'Gender', 'Age', 'Tenure', 'Balance',
       'NumOfProducts', 'HasCrCard', 'IsActiveMember', 'EstimatedSalary',
       'Exited'],
      dtype='object')
```

Next, let's find out more about each feature.

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 11 columns):
CreditScore        10000 non-null int64
Geography          10000 non-null object
Gender             10000 non-null object
Age                10000 non-null int64
Tenure             10000 non-null int64
Balance            10000 non-null float64
NumOfProducts      10000 non-null int64
HasCrCard          10000 non-null int64
IsActiveMember     10000 non-null int64
EstimatedSalary    10000 non-null float64
Exited             10000 non-null int64
dtypes: float64(2), int64(7), object(2)
memory usage: 859.5+ KB
```

The above output shows that there are 10000 samples and 11 features. It also gives the data type for each feature along with non-null count. We understand there is no missing value in the data set.

Missing Value Imputation:

```
data = pd.read_csv('Data.csv')
data[(data.isnull().any(axis=1))]
```

The above code will return a dataframe with all those rows containing nan values.

As you can see above, in our example data set , there are no null values.

When there are null values identified in a data set/column values, we need to first analyse the reason. Depending on the reason and analysis ,we might want to replace the null values with substitute values. One of the common practice to handle null value is to replace the null value with the median value of the column. We can use the following syntax to replace the null value with median value of that column.

```
df = df.apply(lambda x: x.fillna(x.median()),axis = 0)
```

We can also use mean and mode to replace missing values. This decision depends on the data.

Alternate to the above code, Sklearn also provides implementation to replace missing values with either mean, median or mode.

Handle Duplicate Values:

Finally, we can check if there are duplicate rows in the dataset

```
data.duplicated().any()
```

The above line returns 'True' if there are duplicate rows in the dataset.

There are no duplicate rows in our example dataset. If there are, we can remove the duplicate rows using Pandas *DataFrame.drop_duplicates*

Categorical Features:

We notice that there are two columns (Geography and Gender)with 'Object' data type. This means the value in them could be non-numerical or mixed numerical values

We will need to transform these feature values to numerical formats so as to include them in analysis. Let's take a look at these features values first.

```
data["Geography"].value_counts()
```

```
France     5014
Germany    2509
Spain      2477
Name: Geography, dtype: int64
```

```
data["Gender"].value_counts()
```

```
Male       5457
Female     4543
Name: Gender, dtype: int64
```

So, there are three categories(France, Germany, Spain) in 'Geography' and two categories(male, female) in 'Gender'. As we see, the values are represented in non-numerical format. Let's convert them to numerical format using one hot encoding and not by Label Encoding. To understand why one-hot encoding is better than label encoding, read

Choosing the right Encoding method-Label vs OneHot Encoder

how can your choice of encoding play a major role in your prediction model

towardsdatascience.com

Pandas get_dummies can be used for one hot encoding of your values. Here is the result of one-hot encoding.

```
In [14]: data = pd.get_dummies(data, columns = ['Gender', 'Geography'])
         data.head()
```

Out[14]:

NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited	Gender_Female	Gender_Male	Geography_France	Geography_Germany	Geography_Spain
1	1	1	101348.88	1	1	0	1	0	0
1	0	1	112542.58	0	1	0	0	0	1
3	1	0	113931.57	1	1	0	1	0	0
2	0	0	93826.63	0	1	0	1	0	0
1	1	1	79084.10	0	1	0	0	0	1

We now don't see the 'Gender' or 'Geography' columns. Instead get_dummies created columns for each class in the original column. So in our case, it would be

Gender_Female, Gender_Male, Geography_France, Geography_Germany, Geography_Spain.

Alternate to the pandas.get_dummies, we can achieve one hot encoding using `sklearn.preprocessing`.OneHotEncoder

One-hot encoding may not be a good idea when we have a lot a classes in a feature because it creates that many columns.

After encoding, let's run the data.info() one more time to ensure all our column data types are numeric.

```
In [67]: data.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 14 columns):
CreditScore          10000 non-null int64
Age                  10000 non-null int64
Tenure               10000 non-null int64
Balance              10000 non-null float64
NumOfProducts        10000 non-null int64
HasCrCard            10000 non-null int64
IsActiveMember       10000 non-null int64
EstimatedSalary      10000 non-null float64
Exited               10000 non-null int64
Gender_Female        10000 non-null uint8
Gender_Male          10000 non-null uint8
Geography_France     10000 non-null uint8
Geography_Germany    10000 non-null uint8
Geography_Spain      10000 non-null uint8
dtypes: float64(2), int64(7), uint8(5)
memory usage: 752.0 KB
```

Identifying Categorical Feature Columns:

There could be columns with numerical values but columns could be categorical. For eg, Exited column has values in the range of 0 and 1. This column could likely be categorical. To validate let's check the distinct values of this column

```
In [24]: data["Exited"].value_counts()

Out[24]: 0    7963
         1    2037
         Name: Exited, dtype: int64
```

Yes. Exited column consists of 0 and 1. In other words it is categorical. To convert column to categorical ones.

```
In [25]: data['Exited_1'] = data['Exited'].astype("category")
         data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 15 columns):
CreditScore          10000 non-null int64
Age                  10000 non-null int64
Tenure               10000 non-null int64
Balance              10000 non-null float64
NumOfProducts        10000 non-null int64
HasCrCard            10000 non-null int64
IsActiveMember       10000 non-null int64
EstimatedSalary      10000 non-null float64
Exited               10000 non-null int64
Gender_Female        10000 non-null uint8
Gender_Male          10000 non-null uint8
Geography_France     10000 non-null uint8
Geography_Germany    10000 non-null uint8
Geography_Spain      10000 non-null uint8
Exited_1             10000 non-null category
dtypes: category(1), float64(2), int64(7), uint8(5)
memory usage: 761.9 KB
```

Split Data to Train and Test:

X = data.drop(['Exited'],axis=1)
y = data['Exited']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=42)

Feature Scaling:

If we observe the data, each independent variable is in different range. The process of transforming all the features in the given data set to a fixed range is known as 'Scaling'

```
X_train.head()
```

.1]:

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Gender_Female	Gender_Male	Geography_Franc
9069	619	32	4	175406.13	2	1	1	172792.43	1	0	
2603	643	34	7	160426.07	1	0	1	188533.11	1	0	
7738	561	33	6	0.00	2	0	0	173680.39	0	1	
1579	618	41	8	37702.79	1	1	1	195775.48	0	1	
5058	714	37	9	148466.93	2	0	1	151280.96	0	1	

Let's standardize the data.

```
In [15]: from sklearn.preprocessing import StandardScaler
         sc = StandardScaler()
         X_train_norm = sc.fit_transform(X_train)
         X_test_norm = sc.transform(X_test)
```

```
In [16]: X_train_norm

Out[16]: array([[-0.34459497, -0.65674999, -0.34217046, ..., -1.00171576,
                 -0.57559072,  1.73073215],
                [-0.09518109, -0.46637979,  0.69816249, ..., -1.00171576,
                  1.73734559, -0.57779016],
                [-0.94734518, -0.56156489,  0.35138484, ...,  0.99828718,
                 -0.57559072, -0.57779016],
                ...,
                [ 0.86090545, -0.08563939, -1.38250341, ...,  0.99828718,
                 -0.57559072, -0.57779016],
                [ 0.15423279,  0.39028611,  1.04494014, ...,  0.99828718,
                 -0.57559072, -0.57779016],
                [ 0.46600014,  1.1517669 , -1.38250341, ..., -1.00171576,
                  1.73734559, -0.57779016]])
```

With this our data is ready to be fed to the machine learning algorithms!

Data Science Eda Jaya N Aiyappan