Machine Learning

Lab : 2

Data Preprocessing and Feature Selection

December 2024

Perform Date: December 09-13, 2024

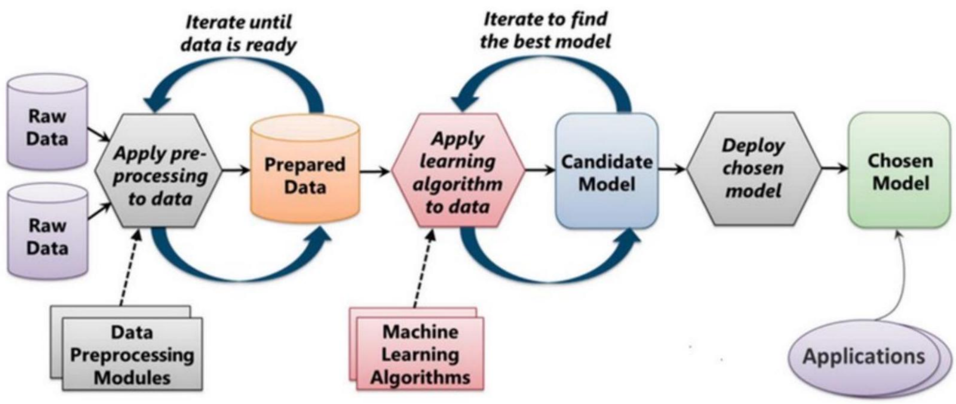
1 Objective

To perform various pre-processing operations on data.

2 Description

Data preprocessing is a data mining technique that involves transforming raw data into an understandable format. Real-world data is often incomplete, inconsistent, and/or lacking in certain behaviours or trends, and is likely to contain many errors. Data preprocessing is a proven method of resolving such issues.

In Real-world data are generally incomplete: lacking attribute values, lacking certain attributes of interest, or containing only aggregate data. **Noisy:** containing errors or outliers. **Inconsistent:** containing discrepancies in codes or names.



From "Introduction to Microsoft Azure" by David Chappell

3 Few Data Preprocessing Techniques

1. Handling the missing value
2. Data Transformation
   (a) Scaling (Min-Max Normalization)
   (b) Mean - Normalization
   (c) Standardization (Z score)
   (d) Binarize Data (Make Binary)
3. Handling Categorical Data
   (a) Lable Encoding
   (b) One Hot Encoding

3.1 Handling the missing values:

1) **Removing data (Row or Column):** This method is commonly used to handle the null values. Here, we either delete a particular row if it has a null value for a particular feature or a particular column if it has missing values more than some threshold. This method is advised only when there are enough samples in the data set. One has to make sure that after we have deleted the data, there is no addition of bias. Removing the data will lead to a loss of information which may not give the expected results while predicting the output.

2) **Imputation:** This strategy can be applied to a feature that has numeric data. We can calculate the mean, median, or mode of the feature and replace it with the missing values. This is an approximation that can add variance to the data set. But the loss of the data can be negated by this method which yields better results compared to the removal of rows and columns.

Replacing with any of the above three approximations is a statistical approach to handling the missing values. This method is also called leaking the data while training. Another way is to approximate it with the deviation of neighboring values. This works better if the data is linear.

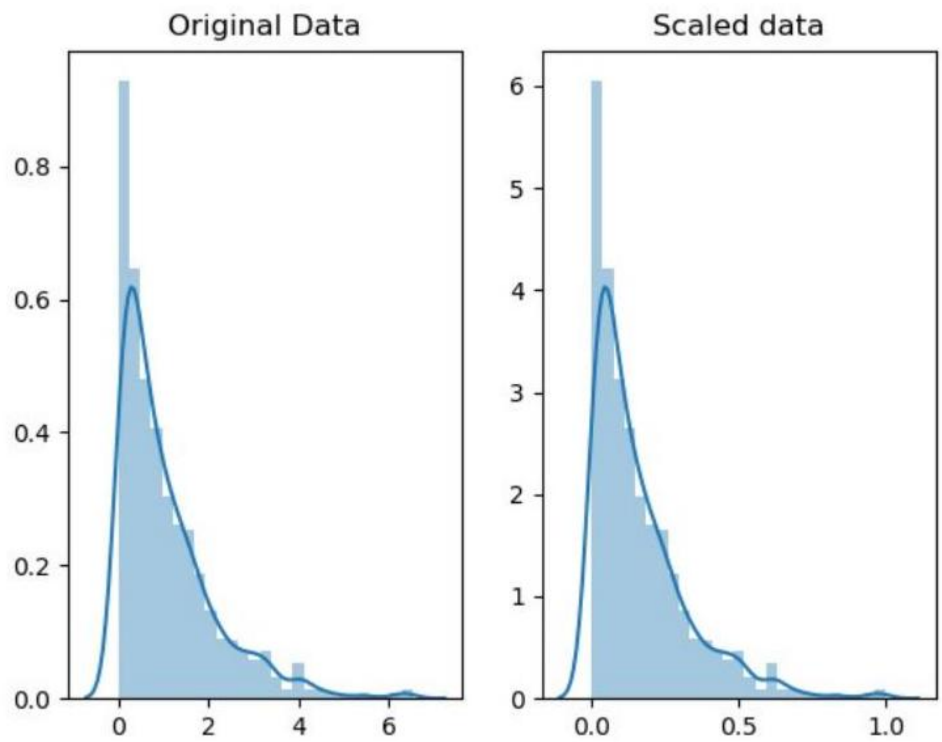
3.2 Data Transformation

1) **Scaling (Min-Max normalization):** When your data is comprised of attributes with varying scales, many machine learning algorithms can benefit from rescaling the attributes to all have the same scale.

Often this is referred to as normalization and attributes are often rescaled into the range between 0 and 1. This is useful for optimization algorithms used in the core of machine learning algorithms like gradient descent. It is also useful for algorithms that weight inputs like regression and neural networks and algorithms that use distance measures like K-Nearest Neighbours.

The Formula:

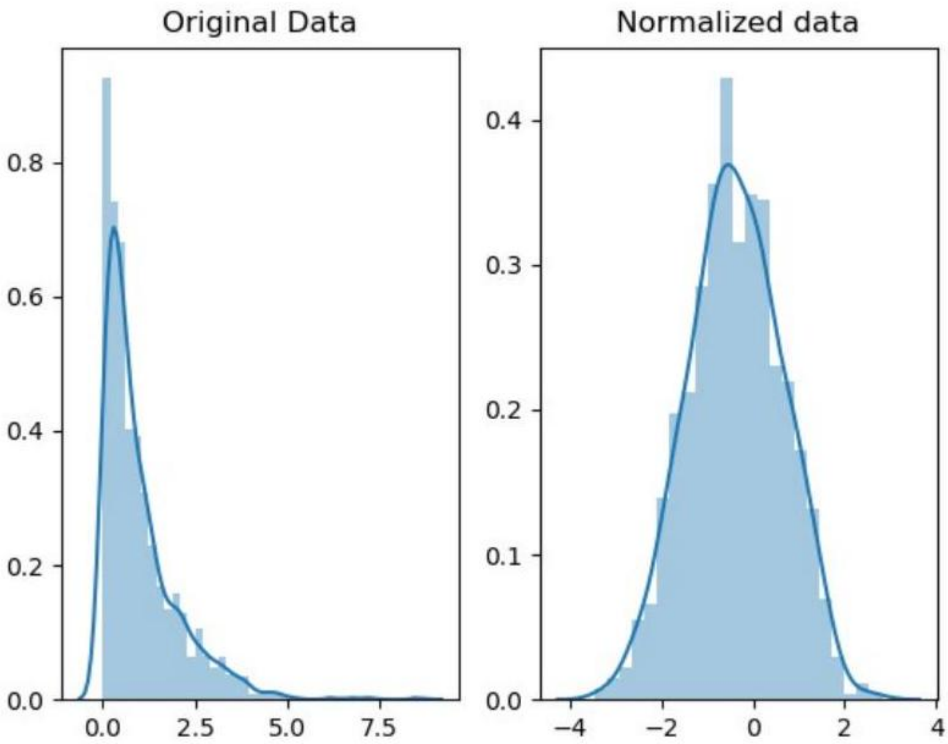
$$x' = \frac{x - x_{\text{min}}}{x_{\text{max}} - x_{\text{min}}}$$



2) **Mean-Normalization:** Mean Normalization is a way to implement Feature Scaling. What Mean normalization does is that it calculates and subtracts the mean for every feature. A common practice is also to divide this value by the range or the standard deviation.

The Formula is:

$$x' = \frac{x - x_{\text{mean}}}{x_{\text{max}} - x_{\text{min}}}$$



In scaling, the range of data is changed while in normalization the shape of the distribution of data is changed.

Data needs to be normalized if a machine learning or statistics technique that assumes a normal distribution of data, is going to be used. e.g. linear regression, linear discriminant analysis (LDA), and Gaussian Naive Bayes.

3) **Standardization (Z score):** Standardization is a useful technique to transform attributes with a Gaussian distribution and differing means and standard deviations to a standard Gaussian distribution with a mean of 0 and a standard deviation of 1.

It is most suitable for techniques that assume a Gaussian distribution in the input variables and work better with rescaled data, such as linear regression, logistic regression, and linear discriminate analysis.

The formula is:

$$x' = \frac{x - x_{\text{mean}}}{\sigma}$$

where x is the original feature vector, x_{mean} is the mean of that feature vector, and σ is its standard deviation.

4) **Binarize Data (Make Binary):** You can transform your data using a binary threshold. All values above the threshold are marked as 1 and all equal to or below are marked as 0.

This is called binarizing your data or threshold your data. It can be useful when you have probabilities that you want to make crisp values. It is also useful when feature engineering and you want to add new features that indicate something meaningful.

3.3 Handling Categorical Data

There are mainly two types of encoders - Label Encoder and One Hot Encoder. They are used to convert categorical data, or text data, into numbers, which our predictive models can better understand.

1. **Label Encoding:** Let's consider the following data:

Country	Age	Salary	Purchased
France	44	72000	No
Spain	27	48000	Yes
Germany	30	54000	No
Spain	38	61000	No
Germany	40	nan	Yes
France	35	58000	Yes
Spain	nan	52000	No
France	48	79000	Yes
Germany	50	83000	No
France	37	67000	Yes

Data from SuperDataScience

In this example, the first column is the country column, which is all text. As you might know by now, we can't have text in our data if we're going to run any kind of model on it. So before we can run a model, we need to make this data ready for the model. And to convert this kind of categorical text data into model-understandable numerical data, we use the Label Encoder class.

Once data is encoded it will be changed as below

```
array([[0, 44.0, 72000.0],
       [2, 27.0, 48000.0],
       [1, 30.0, 54000.0],
       [2, 38.0, 61000.0],
       [1, 40.0, nan],
       [0, 35.0, 58000.0],
       [2, nan, 52000.0],
       [0, 48.0, 79000.0],
       [1, 50.0, 83000.0],
       [0, 37.0, 67000.0]], dtype=object)
```

That's all label encoding is about. But depending on the data, label encoding introduces a new problem. For example, we have encoded a set of country names into numerical data. This is actually categorical data and there is no relation, of any kind, between the rows. The problem here is, since there are different numbers in the same column, the model will misunderstand the data to be in some kind of order, $0 < 1 < 2$. But this isn't the case at all. To overcome this problem, we use One Hot Encoder.

2. **One Hot Encoding**

Now, as we already discussed, depending on the data we have, we might run into situations where, after label encoding, we might confuse our model into thinking that a column has data with some kind of order or hierarchy when we clearly don't have it. To avoid this, we 'OneHotEncode' in that column.

What one hot encoding does is, it takes a column that has categorical data, which has been label encoded and then splits the column into multiple columns. The numbers are replaced by 1 s and 0s, depending on which column has what value. In our example, we'll get three new columns, one for each country-France, Germany, and Spain.

For rows that have the first column value as France, the 'France' column will have a ' 1 ' and the other two columns will have ' 0 's. Similarly, for rows that have the first column value as Germany, the 'Germany' column will have a ' 1 ' and the other two columns will have ' 0 's.

	0	1	2	3	4
0	1	0	0	44	72000
1	0	0	1	27	48000
2	0	1	0	30	54000
3	0	0	1	38	61000
4	0	1	0	40	63777.8
5	1	0	0	35	58000
6	0	0	1	38.7778	52000
7	1	0	0	48	79000
8	0	1	0	50	83000
9	1	0	0	37	67000

As you can see, we have three new columns with 1 s and 0s, depending on the country that the rows represent.

3.4 Feature Selection

All of the features we find in the dataset might not be useful in building a machine-learning model to make the necessary prediction. Using some of the features might even make the predictions worse. So, feature selection plays a huge role in building a machine-learning model.

3.4.1 What is correlation?

Correlation is a statistical term that in common usage refers to how close two variables are to having a linear relationship with each other.

Features with high correlation are more linearly dependent and hence have almost the same effect on the dependent variable (Y).

So, when two features have a high correlation, we can drop one of the two features.

4 Implementation Guidelines

4.1 Steps for Data Transformation

1. Import Libraries

2. Load Data
3. Seprate Input and Output attributes
4. Perform scaling (Min-Max Normalization)
5. Perform Standardization

[3]:
```
# Step 1: Import Libraries

import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler, StandardScaler

# Step 2: Load Data

datasets = pd.read_csv('Data_for_Transformation.csv')
print("\nData :\n",datasets)
#print("\nData statistics\n",datasets.describe())

# Step 3: Seprate Input and Output attributes

# All rows, all columns except last
X = datasets.iloc[:, :-1].values

# Only last column
Y = datasets.iloc[:, -1].values

#print("\n\nInput : \n", X)
#print("\n\nOutput: \n", Y)

X_new = datasets.iloc[:,1:3].values
print("\n\nX for transformation : \n", X_new)
```

```
Data :
    Country  Age  Salary Purchased
0    France   44   72000        No
1     Spain   27   48000       Yes
2   Germany   30   54000        No
3     Spain   38   61000        No
4   Germany   40   68000       Yes
5    France   35   58000       Yes
6     Spain   39   52000        No
7    France   48   79000       Yes
8   Germany   50   83000        No
9    France   37   67000       Yes
10    Spain   45   55000        No


X for transformation :
 [[   44 72000]
 [   27 48000]
 [   30 54000]
 [   38 61000]
 [   40 68000]
 [   35 58000]
 [   39 52000]
 [   48 79000]
 [   50 83000]
 [   37 67000]
 [   45 55000]]
```

[8]:

```
# Step 4 : Perform scaling on age and salary


scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X_new)
```

```
print("\n\nScaled X : \n", X_scaled)
```

```
Scaled X :
 [[0.73913043 0.68571429]
 [0.         0.        ]
 [0.13043478 0.17142857]
 [0.47826087 0.37142857]
 [0.56521739 0.57142857]
 [0.34782609 0.28571429]
 [0.52173913 0.11428571]
 [0.91304348 0.88571429]
 [1.         1.        ]
 [0.43478261 0.54285714]
 [0.7826087  0.2       ]]
```

[5]:
```
# Step 5 : Perform standardization on age and salary

std = StandardScaler()
X_std = std.fit_transform(X_new)
print("\n\nStandardized X : \n", X_std)
```

```
Standardized X :
 [[ 0.68188156  0.79548755]
 [-1.81835082 -1.41513049]
 [-1.37713334 -0.86247598]
 [-0.2005534  -0.21771238]
 [ 0.09359159  0.42705121]
 [-0.64177088 -0.49403964]
 [-0.05348091 -1.04669415]
```

```
[ 1.27017153  1.44025115]
[ 1.56431652  1.80868749]
[-0.34762589  0.33494213]
[ 0.82895405 -0.77036689]]
```

4.2 Steps for Handling Categorical Data

1. Import Libraries
2. Load Data
3. Seprate Input and Output attributes
4. Convert the categorical data into numerical data

[5]:
```
# Step 1: Import Libraries

import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder,OneHotEncoder

# Step 2: Load Data

datasets = pd.read_csv('Data_for_Categorical_Values.csv')
print("\nData :\n",datasets)
print("\nData statistics\n",datasets.describe())
```

```
Data :
     Country  Age  Salary Purchased
0    France   44   72000        No
1     Spain   27   48000       Yes
2   Germany   30   54000        No
3     Spain   38   61000        No
4   Germany   40   68000       Yes
5    France   35   58000       Yes
6     Spain   39   52000        No
```

```
7     France   48   79000        Yes
8    Germany   50   83000         No
9     France   37   67000        Yes
10     Spain   45   55000         No

Data statistics
              Age        Salary
count   11.000000     11.000000
mean    39.363636  63363.636364
std      7.131237  11386.594989
min     27.000000  48000.000000
25%     36.000000  54500.000000
50%     39.000000  61000.000000
75%     44.500000  70000.000000
max     50.000000  83000.000000
```

[2]:
```
# Step 3: Seprate Input and Output attributes

# All rows, all columns except last
X = datasets.iloc[:, :-1].values

# Only last column
Y = datasets.iloc[:, -1].values

print("\n\nInput : \n", X)
print("\n\nOutput: \n", Y)
```

```
Input :
 [['France' 44 72000]
 ['Spain' 27 48000]
 ['Germany' 30 54000]
```

```
 ['Spain' 38 61000]
 ['Germany' 40 68000]
 ['France' 35 58000]
 ['Spain' 39 52000]
 ['France' 48 79000]
 ['Germany' 50 83000]
 ['France' 37 67000]
 ['Spain' 45 55000]]


Output:
 ['No' 'Yes' 'No' 'No' 'Yes' 'Yes' 'No' 'Yes' 'No' 'Yes' 'No']
```

[3]:
```
# Step 4a: Apply LabelEncoder on the data
#           to convert country names into numeric values

le = LabelEncoder()
X[ : ,0] = le.fit_transform(X[ : ,0])
print("\n\nInput : \n", X)
```

```
Input :
 [[0 44 72000]
 [2 27 48000]
 [1 30 54000]
 [2 38 61000]
 [1 40 68000]
 [0 35 58000]
 [2 39 52000]
 [0 48 79000]
 [1 50 83000]
 [0 37 67000]
```

```
 [2 45 55000]]
```

[4]:
```
# Step 4b: Use dummy variables from pandas library
#          to create one column for each country

dummy = pd.get_dummies(datasets['Country'])
print("\n\nDummy :\n",dummy)
datasets = datasets.drop(['Country','Purchased'],axis=1)
datasets = pd.concat([dummy,datasets],axis=1)
print("\n\nFinal Data :\n",datasets)
```

```
Dummy :
     France  Germany  Spain
0        1        0      0
1        0        0      1
2        0        1      0
3        0        0      1
4        0        1      0
5        1        0      0
6        0        0      1
7        1        0      0
8        0        1      0
9        1        0      0
10       0        0      1


Final Data :
     France  Germany  Spain  Age  Salary
0        1        0      0   44   72000
1        0        0      1   27   48000
2        0        1      0   30   54000
```

```
3          0          0        1    38    61000
4          0          1        0    40    68000
5          1          0        0    35    58000
6          0          0        1    39    52000
7          1          0        0    48    79000
8          0          1        0    50    83000
9          1          0        0    37    67000
10         0          0        1    45    55000
```

[13]:
```
#Use One Hot Encoder from scikit learn
onehotencoder = OneHotEncoder()
#reshape the 1-D country array to 2-D as fit_transform␣
↪expects 2-D and finally fit the object
x = onehotencoder.fit_transform(datasets.Country.values.
↪reshape(-1,1)).toarray()
```

[14]:
```
x
```

[14]:
```
array([[1., 0., 0.],
       [0., 0., 1.],
       [0., 1., 0.],
       [0., 0., 1.],
       [0., 1., 0.],
       [1., 0., 0.],
       [0., 0., 1.],
       [1., 0., 0.],
       [0., 1., 0.],
       [1., 0., 0.],
       [0., 0., 1.]])
```

[17]:
```
dfOneHot = pd.DataFrame(x, columns = ["Country_"+str(int(i))␣
↪for i in range(datasets.shape[1]-1)])
df = pd.concat([datasets, dfOneHot], axis=1) #column
#droping the country column
```

```
df= df.drop(['Country'], axis=1)
#printing to verify
print(df.head())
```

```
   Age  Salary Purchased  Country_0  Country_1  Country_2
0   44   72000        No        1.0        0.0        0.0
1   27   48000       Yes        0.0        0.0        1.0
2   30   54000        No        0.0        1.0        0.0
3   38   61000        No        0.0        0.0        1.0
4   40   68000       Yes        0.0        1.0        0.0
```

4.3 Steps for Handling the missing value

1. Import Libraries
2. Load data
3. Seprate Input and Output attributes
4. Find the missing values and handle it in either way
   a. Removing data
   b. Imputation

[27]:
```
# Step 1: Import Libraries

import numpy as np
import pandas as pd
from sklearn.impute import SimpleImputer

# Step 2: Load Data

datasets = pd.read_csv('Data_for_Missing_Values.csv')
print("\nData :\n",datasets)
print("\nData statistics\n",datasets.describe())
```

```
Data :
```

```
     Country   Age    Salary Purchased
0     France  44.0   72000.0        No
1      Spain  27.0   48000.0       Yes
2    Germany  30.0   54000.0        No
3      Spain  38.0   61000.0        No
4        NaN   NaN       NaN       NaN
5    Germany  40.0       NaN       Yes
6     France  35.0   58000.0       Yes
7      Spain   NaN   52000.0        No
8     France  48.0   79000.0       Yes
9    Germany  50.0   83000.0        No
10    France  37.0   67000.0       Yes
11     Spain  45.0   55000.0        No

Data statistics
              Age        Salary
count  10.000000     10.000000
mean   39.400000  62900.000000
std     7.515909  11892.574714
min    27.000000  48000.000000
25%    35.500000  54250.000000
50%    39.000000  59500.000000
75%    44.750000  70750.000000
max    50.000000  83000.000000
```

[28]:
```
# Step 3: Seprate Input and Output attributes

# All rows, all columns except last
X = datasets.iloc[:, :-1].values

# Only last column
Y = datasets.iloc[:, -1].values
```

```
print("\n\nInput : \n", X)
print("\n\nOutput: \n", Y)
```

```
Input :
 [['France' 44.0 72000.0]
 ['Spain' 27.0 48000.0]
 ['Germany' 30.0 54000.0]
 ['Spain' 38.0 61000.0]
 [nan nan nan]
 ['Germany' 40.0 nan]
 ['France' 35.0 58000.0]
 ['Spain' nan 52000.0]
 ['France' 48.0 79000.0]
 ['Germany' 50.0 83000.0]
 ['France' 37.0 67000.0]
 ['Spain' 45.0 55000.0]]


Output:
 ['No' 'Yes' 'No' 'No' nan 'Yes' 'Yes' 'No' 'Yes' 'No' 'Yes'␣
 ↪'No']
```

[29]:
```
# Step 4: Find the missing values and handle it in either way

# 4a. Removing the row with all null values

datasets.dropna(axis=0,how='all',inplace=True)
print("\nNew Data :",datasets)
```

```
#4b. Removing the row with any one null values

#datasets.dropna(axis=0,how='any',inplace=True)
```

```
New Data :      Country   Age   Salary Purchased
0     France  44.0  72000.0        No
1      Spain  27.0  48000.0       Yes
2    Germany  30.0  54000.0        No
3      Spain  38.0  61000.0        No
5    Germany  40.0      NaN       Yes
6     France  35.0  58000.0       Yes
7      Spain   NaN  52000.0        No
8     France  48.0  79000.0       Yes
9    Germany  50.0  83000.0        No
10    France  37.0  67000.0       Yes
11     Spain  45.0  55000.0        No
```

[30]:
```
updated_df = datasets;
updated_df['Age']=updated_df['Age'].fillna(updated_df['Age'].
 ↪mean())
updated_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 11 entries, 0 to 11
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Country    11 non-null     object
 1   Age        11 non-null     float64
 2   Salary     10 non-null     float64
 3   Purchased  11 non-null     object
dtypes: float64(2), object(2)
```

```
memory usage: 440.0+ bytes
```

[31]:
```
datasets
```

[31]:
```
    Country   Age   Salary Purchased
0    France  44.0  72000.0        No
1     Spain  27.0  48000.0       Yes
2   Germany  30.0  54000.0        No
3     Spain  38.0  61000.0        No
5   Germany  40.0      NaN       Yes
6    France  35.0  58000.0       Yes
7     Spain  39.4  52000.0        No
8    France  48.0  79000.0       Yes
9   Germany  50.0  83000.0        No
10   France  37.0  67000.0       Yes
11    Spain  45.0  55000.0        No
```

[32]:
```
updated_df = datasets;
updated_df['Salary']=updated_df['Salary'].
 ↪fillna(updated_df['Salary'].mean())
updated_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 11 entries, 0 to 11
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Country    11 non-null     object
 1   Age        11 non-null     float64
 2   Salary     11 non-null     float64
 3   Purchased  11 non-null     object
dtypes: float64(2), object(2)
memory usage: 440.0+ bytes
```

[33]:
```
datasets
```

[33]:

	Country	Age	Salary	Purchased
0	France	44.0	72000.0	No
1	Spain	27.0	48000.0	Yes
2	Germany	30.0	54000.0	No
3	Spain	38.0	61000.0	No
5	Germany	40.0	62900.0	Yes
6	France	35.0	58000.0	Yes
7	Spain	39.4	52000.0	No
8	France	48.0	79000.0	Yes
9	Germany	50.0	83000.0	No
10	France	37.0	67000.0	Yes
11	Spain	45.0	55000.0	No

[22]:
```
new_X = datasets.iloc[:, :-1].values
# Only last column
new_Y = datasets.iloc[:, -1].values
```

[ ]:
```
#Using SimpleImputer from Scikit-Learn Library
```

[37]:
```
# Step 1: Import Libraries

import numpy as np
import pandas as pd
from sklearn.impute import SimpleImputer

# Step 2: Load Data

datasets = pd.read_csv('Data_for_Missing_Values.csv')
print("\nData :\n",datasets)
print("\nData statistics\n",datasets.describe())
```

```
Data :
```

```
    Country   Age   Salary Purchased
0    France  44.0  72000.0        No
1     Spain  27.0  48000.0       Yes
2   Germany  30.0  54000.0        No
3     Spain  38.0  61000.0        No
4       NaN   NaN      NaN       NaN
5   Germany  40.0      NaN       Yes
6    France  35.0  58000.0       Yes
7     Spain   NaN  52000.0        No
8    France  48.0  79000.0       Yes
9   Germany  50.0  83000.0        No
10   France  37.0  67000.0       Yes
11    Spain  45.0  55000.0        No

Data statistics
             Age        Salary
count  10.000000     10.000000
mean   39.400000  62900.000000
std     7.515909  11892.574714
min    27.000000  48000.000000
25%    35.500000  54250.000000
50%    39.000000  59500.000000
75%    44.750000  70750.000000
max    50.000000  83000.000000
```

[38]:

```
# Step 3: Seprate Input and Output attributes

# All rows, all columns except last
X = datasets.iloc[:, :-1].values

# Only last column
Y = datasets.iloc[:, -1].values
```

```
print("\n\nInput : \n", X)
print("\n\nOutput: \n", Y)
```

```
Input :
 [['France' 44.0 72000.0]
 ['Spain' 27.0 48000.0]
 ['Germany' 30.0 54000.0]
 ['Spain' 38.0 61000.0]
 [nan nan nan]
 ['Germany' 40.0 nan]
 ['France' 35.0 58000.0]
 ['Spain' nan 52000.0]
 ['France' 48.0 79000.0]
 ['Germany' 50.0 83000.0]
 ['France' 37.0 67000.0]
 ['Spain' 45.0 55000.0]]


Output:
 ['No' 'Yes' 'No' 'No' nan 'Yes' 'Yes' 'No' 'Yes' 'No' 'Yes'
 'No']
```

[39]:
```
# Step 4: Find the missing values and handle it in either way

# 4a. Removing the row with all null values

datasets.dropna(axis=0,how='all',inplace=True)
print("\nNew Data :",datasets)
```

```
#4b. Removing the row with any one null values

#datasets.dropna(axis=0,how='any',inplace=True)
```

```
New Data :      Country   Age    Salary Purchased
0     France  44.0  72000.0        No
1      Spain  27.0  48000.0       Yes
2    Germany  30.0  54000.0        No
3      Spain  38.0  61000.0        No
5    Germany  40.0      NaN       Yes
6     France  35.0  58000.0       Yes
7      Spain   NaN  52000.0        No
8     France  48.0  79000.0       Yes
9    Germany  50.0  83000.0        No
10    France  37.0  67000.0       Yes
11     Spain  45.0  55000.0        No
```

[40]:
```
# 4b. Imputation (Replacing null values with mean value of␣
↪that attribute)

# All rows, all columns except last
new_X = datasets.iloc[:, :-1].values

# Only last column
new_Y = datasets.iloc[:, -1].values



updated_df['Age'].fillna(updated_df['Age'].mean())

# Using Imputer function to replace NaN values with mean of␣
↪that parameter value
```

```
imputer = SimpleImputer(missing_values = np.nan,strategy =␣
 ↪"mean")

# Fitting the data, function learns the stats
imputer = imputer.fit(new_X[:, 1:3])

# fit_transform() will execute those stats on the input ie.␣
 ↪X[:, 1:3]
new_X[:, 1:3] = imputer.transform(new_X[:, 1:3])

# filling the missing value with mean
print("\n\nNew Input with Mean Value for NaN : \n\n", new_X)
```

```
New Input with Mean Value for NaN :

 [['France' 44.0 72000.0]
 ['Spain' 27.0 48000.0]
 ['Germany' 30.0 54000.0]
 ['Spain' 38.0 61000.0]
 ['Germany' 40.0 62900.0]
 ['France' 35.0 58000.0]
 ['Spain' 39.4 52000.0]
 ['France' 48.0 79000.0]
 ['Germany' 50.0 83000.0]
 ['France' 37.0 67000.0]
 ['Spain' 45.0 55000.0]]
```

4.4 Correlation

4.4.1 How does correlation help in feature selection?

Features with high correlation are more linearly dependent and hence have almost the same effect on the dependent variable. So, when two features have high correlation, we can drop one of the two features.

4.4.2 Import the necessary libraries

[1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
```

4.4.3 Loading the dataset

[2]:
```
data = pd.read_csv('Data_for_Correlation.csv')
```

[3]:
```
data.head()
```

[3]:

	X1	X2	X3	X4	Y
0	1	1	4	-2	1
1	2	4	5	-4	1
2	3	9	6	3	0
3	4	16	7	4	0
4	5	25	8	25	1

Removing the Class Label entry (Y)

[4]:
```
data = data.iloc[:,:-1]
data.head()
```

[4]:

	X1	X2	X3	X4
0	1	1	4	-2
1	2	4	5	-4
2	3	9	6	3
3	4	16	7	4
4	5	25	8	25

[5]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14 entries, 0 to 13
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   X1      14 non-null     int64
 1   X2      14 non-null     int64
 2   X3      14 non-null     int64
 3   X4      14 non-null     int64
dtypes: int64(4)
memory usage: 576.0 bytes
```

Selecting features based on correlation Generating the correlation matrix

[6]:
```
corr = data.corr()
corr.head()
```

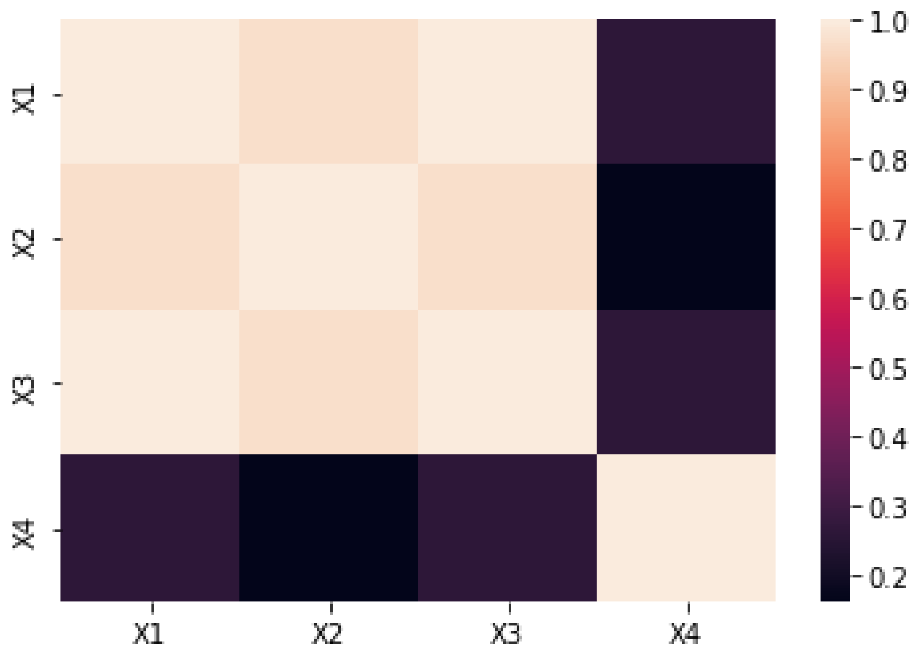
[6]:

	X1	X2	X3	X4
X1	1.000000	0.972714	1.000000	0.263266
X2	0.972714	1.000000	0.972714	0.163575
X3	1.000000	0.972714	1.000000	0.263266
X4	0.263266	0.163575	0.263266	1.000000

Generating the correlation heatmap

[7]:
```
sns.heatmap(corr)
```

[7]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fceb1f4be50>
```



Next, we compare the correlation between features and remove one of two features that have a correlation higher than 0.9

[8]:
```
columns = np.full((corr.shape[0],), True, dtype=bool)
for i in range(corr.shape[0]):
    for j in range(i+1, corr.shape[0]):
        if corr.iloc[i,j] >= 0.9:
            if columns[j]:
                columns[j] = False
```

[9]:
```
selected_columns = data.columns[columns]
selected_columns.shape
```

[9]: (2,)

[10]:
```
data = data[selected_columns]
print(data)
```

```
   X1  X4
0   1  -2
```

```
1    2   -4
2    3    3
3    4    4
4    5   25
5    6   76
6    7   34
7    8  346
8    9   67
9   10    3
10  11  355
11  12   88
12  13    2
13  14    1
```

5 Exercise:

1) Perform all data preprocessing tasks and feature selection on "Exercise-CarData.csv"

6 Exercise 1.1

1. Describe the dataset used in this lab exercise.
2. What were the various pre-processesing steps performed on Exercise-CarData.csv dataset. Mention the appropriate functions used to perform them.
3. Write the correlation matrix generated for Iris Dataset. Mention the features which have the hightest correlation.

7 References

1. https://proclusacademy.com/blog/robust-scaler-outliers/