

1. Feature Engineering

Contents

1. Feature scaling.....	2
2. Data pre-processing	2
3. Why Data Pre-processing in Machine Learning?.....	2
4. What type of data we need to handle?.....	2
5. Numerical data	3
6. To handle Numerical data	4
7. To handle Categorical data	5
8. scikit-learn installation.....	6
9. Label Encoder	7
9.1. LabelEncoder class	7
9.2. fit_transform(p) method.....	7
10. MinMaxScaler.....	9
11. Transforming Features	13
12. Handling outlier	14
13. Impute Missing values	18
14. Impute missing numeric values	19

1. Feature Engineering

1. Feature scaling

- ✓ Feature Scaling is a technique to standardize the independent features present in the data in a fixed range.
- ✓ It is performed during the data pre-processing.

2. Data pre-processing

- ✓ In machine learning data pre-processing is an important step
- ✓ The purpose of this technique is cleaning and organizing the raw data to make it suitable for building and training machine learning models

3. Why Data Pre-processing in Machine Learning?

- ✓ Typically, real-world data is incomplete, inconsistent, inaccurate (contains errors or outliers), and often lacks specific attribute values/trends.
- ✓ This is where data pre-processing enters the scenario – it helps to **clean**, **format**, and **organize** the raw data, thereby making it ready to use the data for Machine Learning models.

4. What type of data we need to handle?

- ✓ Two types of data we need to handle
 - Numerical data
 - Categorical data

5. Numerical data

- ✓ Quantitative data is the measurement of something monthly sales, or student scores etc.
- ✓ The natural way to represent these quantities is numerically (e.g., 29 students, \$529,392 in sales).
- ✓ So we need to understand how to transforming raw numerical data into features, then we need to use this feature during machine learning

6. To handle Numerical data

Technique	Purpose
✓ LabelEncoder	✓ To convert all character/categorical variables to be numeric.
✓ StandardScaler	✓ To transform a feature which is mean to 0 and standard deviation to 1
✓ Transforming Features	✓ We can transform features
✓ Handling outlier	✓ To handle outlier
✓ Impute Missing Values	✓ To impute missing value with strategy

7. To handle Categorical data

Technique	Purpose
✓ Encoding nominal categories	✓ To do one hot encoding
✓ Encoding ordinal categories	✓ Ordinal categorical
✓ Imputing categorical missing values	✓ Imputing categorical missing values with most frequent strategy

8. scikit-learn installation

- ✓ To execute all these examples we need to install scikit-learn library.

```
pip install scikit-learn
```

9. Label Encoder

- ✓ By using this we can convert all character/categorical variables to be numeric.

9.1. LabelEncoder class

- ✓ **LabelEncoder** is predefined class in **sklearn.preprocessing** package
- ✓ We need to import this class from **sklearn.preprocessing** package
- ✓ Once we imported then we need to **create an object** o **LabelEncoder** class.

9.2. fit_transform(p) method

- ✓ fit_transform(p) is predefined method in **LabelEncoder** class.
- ✓ We should access this method by using **LabelEncoder** object only.
- ✓ This method converts categorical variables into numerical values.

Program Name LabelEncoder
demo1.py

```
from sklearn.preprocessing import LabelEncoder
import pandas as pd

d = {
    "Company": ["Google", "Twitter", "Google", "LinkedIn"],
    "Role": ["Data Scientist", "Sales manager", "HR", "HR"]
}

df = pd.DataFrame(d)
label_encoder = LabelEncoder()

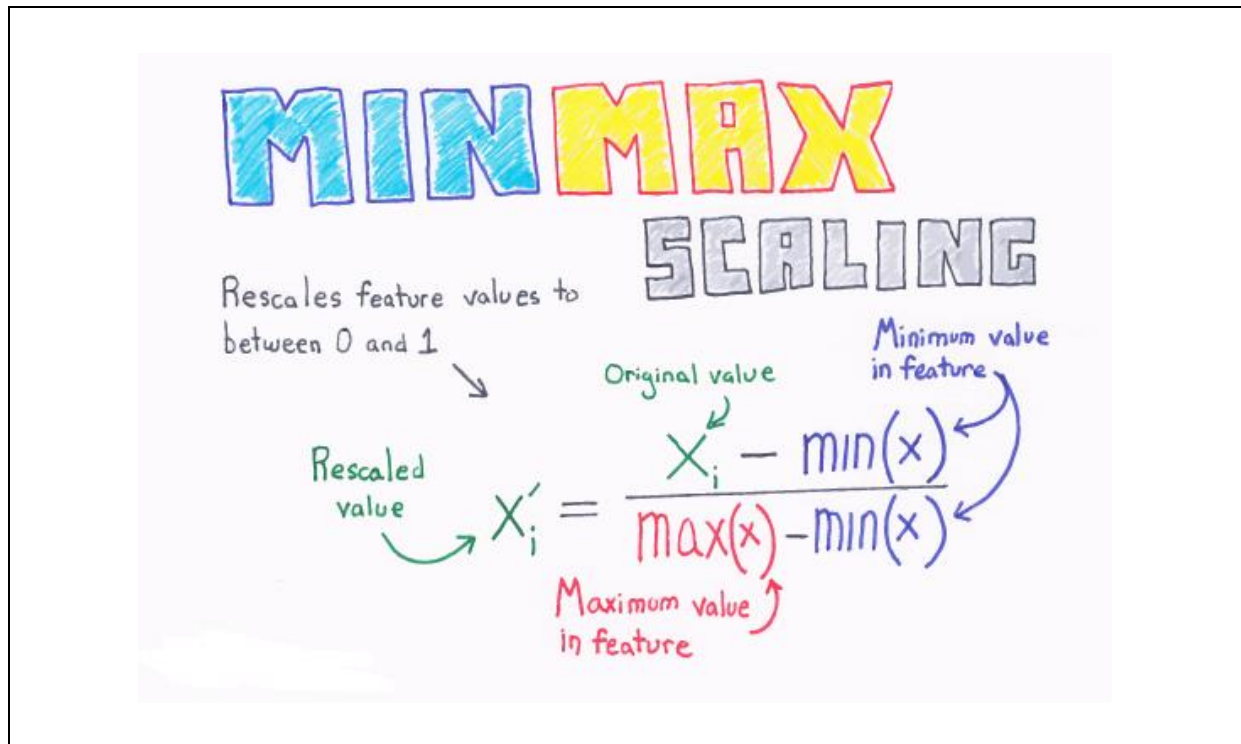
df["Company_n"] = label_encoder.fit_transform(df['Company'])
df["Role_n"] = label_encoder.fit_transform(df['Role'])

print()
print(df)
```

Output

```
   Company  Role  Company_n  Role_n
0   Google  Data Scientist      0      0
1  Twitter  Sales manager      2      2
2   Google      HR          0      1
3  LinkedIn      HR          1      1
```


10. MinMaxScaler



- ✓ Min-max scaling is a common feature pre-processing technique which results in scaled data values that fall in the range 0 and 1.
 - 0 is minimum value
 - 1 is maximum value
- ✓ If we **rescale** the value of numeric feature then it is in between two values.

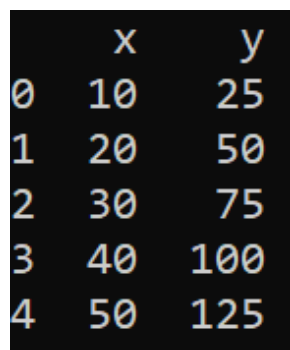
Program Name MinMaxScaler
demo2.py

```
import pandas as pd

d = {
    "x" : [10, 20, 30, 40, 50],
    "y" : [25, 50, 75, 100, 125]
}

df = pd.DataFrame(d)
print(df)
```

Output



	x	y
0	10	25
1	20	50
2	30	75
3	40	100
4	50	125

Program Name MinMaxScaler: single column
demo3.py

```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler

d = {
    "x" : [10, 20, 30, 40, 50],
    "y" : [25, 50, 75, 100, 125]
}

df = pd.DataFrame(d)
mm_scale = MinMaxScaler(feature_range = (0, 1))

print(df)

one_col = mm_scale.fit_transform(df[["x"]])

print(one_col)
```

Output

```
   x    y
0  10   25
1  20   50
2  30   75
3  40  100
4  50  125

[[0.  ]
 [0.25]
 [0.5 ]
 [0.75]
 [1.  ]]
```

Program Name MinMaxScaler: two columns
demo4.py

```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler

d = {
    "x" : [10, 20, 30, 40, 50],
    "y" : [25, 50, 75, 100, 125]
}

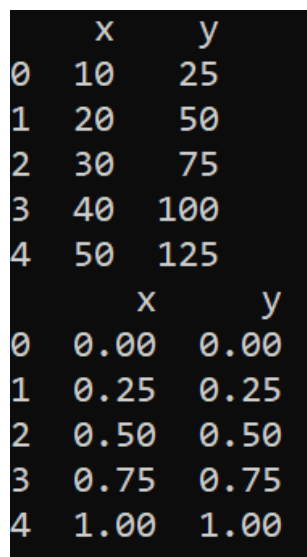
df = pd.DataFrame(d)
mm_scale = MinMaxScaler(feature_range = (0, 1))

print(df)

df[["x", "y"]] = mm_scale.fit_transform(df[["x", "y"]])

print(df)
```

Output



```
   x    y
0  10   25
1  20   50
2  30   75
3  40  100
4  50  125

   x    y
0  0.00  0.00
1  0.25  0.25
2  0.50  0.50
3  0.75  0.75
4  1.00  1.00
```

11. Transforming Features

- ✓ By using **FunctionTransformer** we can transform the features.

Program Name FunctionTransformer
demo5.py

```
import numpy as np
from sklearn.preprocessing import FunctionTransformer

a = [[10, 20], [30, 40], [50, 60]]

f = np.array(a)

def add_ten(x):
    return x + 10

obj = FunctionTransformer(add_ten)

result = obj.transform(f)

print(f)
print()
print(result)
```

Output

```
[[10 20]
 [30 40]
 [50 60]]

[[20 30]
 [40 50]
 [60 70]]
```

12. Handling outlier

HANDLING OUTLIERS

1. If due to an error: drop, mark as missing value, mark as possible error.
2. If a legitimate but extreme value: decide if it is genuinely a member of the population we are trying to address with our model.

- ✓ Outlier means a large value compared with other values
- ✓ Some values are also out of the range of the feature, so they are also considered as outliers.
- ✓ Outliers affect our model's efficiency because they influence the model very much.
- ✓ Three ways to handle these,
 - Drop outlier or filter outlier
 - Marking them using boolean condition
 - Transform into feature

Program Name A dataframe with outliers
demo6.py

```
import numpy as np
import pandas as pd

houses = pd.DataFrame()

houses['Price'] = [534433, 392333, 293222, 4322032]
houses['rooms'] = [2, 3, 2, 116]
houses['Square_Feet'] = [1500, 2500, 1500, 48000]

print(houses)
```

Output

	Price	rooms	Square_Feet
0	534433	2	1500
1	392333	3	2500
2	293222	2	1500
3	4322032	116	48000

Program Name Filtering outlier
demo7.py

```
import numpy as np
import pandas as pd

houses = pd.DataFrame()

houses['Price'] = [534433, 392333, 293222, 4322032]
houses['rooms'] = [2, 3, 2, 116]
houses['Square_Feet'] = [1500, 2500, 1500, 48000]

con1 = houses['rooms'] < 20
new = houses[con1]

print(new)
```

Output

	Price	rooms	Square_Feet
0	534433	2	1500
1	392333	3	2500
2	293222	2	1500

Program Name Mark them as outliers
demo8.py

```
import numpy as np
import pandas as pd

houses = pd.DataFrame()

houses['Price'] = [534433, 392333, 293222, 4322032]
houses['rooms'] = [2, 3, 2, 116]
houses['Square_Feet'] = [1500, 2500, 1500, 48000]

houses["Outlier"] = np.where(houses["rooms"] < 20, 0, 1)

print(houses)
```

Output

	Price	rooms	Square_Feet	Outlier
0	534433	2	1500	0
1	392333	3	2500	0
2	293222	2	1500	0
3	4322032	116	48000	1

13. Impute Missing values

- ✓ We can impute missing values with different strategy.
- ✓ There are two types of missing values
 - Numeric missing values
 - Categorical missing values

IMPUTING MISSING VALUES

1. If quantitative, replace with an average value.
2. If qualitative, replace with most common value.
3. Use a model to predict the missing values. For example, k-nearest neighbors.

ChrisAlben

14. Impute missing numeric values

- ✓ Missing numeric values we can impute with different strategy
 - mean
 - median
 - most_frequent
 - constant

Program Name Creating DataFrame
demo9.py

```
import numpy as np
import pandas as pd

students = [
    [85, 'M', 'verygood'],
    [95, 'F', 'excellent'],
    [60, None, 'good'],
    [np.NaN, 'M', 'average'],
    [70, 'M', 'good'],
    [np.NaN, None, 'verygood'],
    [60, 'F', 'verygood'],
    [98, 'M', 'excellent']
]

cols = ['marks', 'gender', 'result']
df = pd.DataFrame(students, columns = cols)
print(df)
```

output

	marks	gender	result
0	85.0	M	verygood
1	95.0	F	excellent
2	60.0	None	good
3	NaN	M	average
4	70.0	M	good
5	NaN	None	verygood
6	60.0	F	verygood
7	98.0	M	excellent

Program Name Imputing missing numeric values with mean strategy
demo10.py

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

students = [
    [85, 'M', 'verygood'],
    [95, 'F', 'excellent'],
    [60, None, 'good'],
    [np.NaN, 'M', 'average'],
    [70, 'M', 'good'],
    [np.NaN, None, 'verygood'],
    [60, 'F', 'verygood'],
    [98, 'M', 'excellent']
]

cols = ['marks', 'gender', 'result']
df = pd.DataFrame(students, columns = cols)

print(df)

imputer = SimpleImputer(missing_values = np.nan, strategy =
'mean')

result = df['marks'].values.reshape(-1, 1)

df.marks = imputer.fit_transform(result)

print()
print(df)
```

output

```
      marks gender  result
0    85.0      M  verygood
1    95.0      F  excellent
2    60.0    None    good
3     NaN      M  average
4    70.0      M    good
5     NaN    None  verygood
6    60.0      F  verygood
7    98.0      M  excellent
```

```
      marks gender  result
0    85.0      M  verygood
1    95.0      F  excellent
2    60.0    None    good
3    78.0      M  average
4    70.0      M    good
5    78.0    None  verygood
6    60.0      F  verygood
7    98.0      M  excellent
```

Program Name Imputing missing numeric values with median strategy
demo11.py

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

students = [
    [85, 'M', 'verygood'],
    [95, 'F', 'excellent'],
    [60, None, 'good'],
    [np.NaN, 'M', 'average'],
    [70, 'M', 'good'],
    [np.NaN, None, 'verygood'],
    [60, 'F', 'verygood']
]

cols = ['marks', 'gender', 'result']
df = pd.DataFrame(students, columns = cols)

print(df)

imputer = SimpleImputer(missing_values = np.nan, strategy = '
median')

result = df['marks'].values.reshape(-1, 1)

df.marks = imputer.fit_transform(result)

print()
print(df)
```

output

```
   marks gender  result
0   85.0      M  verygood
1   95.0      F  excellent
2   60.0   None    good
3    NaN      M  average
4   70.0      M    good
5    NaN   None  verygood
6   60.0      F  verygood
```

```
   marks gender  result
0   85.0      M  verygood
1   95.0      F  excellent
2   60.0   None    good
3   70.0      M  average
4   70.0      M    good
5   70.0   None  verygood
6   60.0      F  verygood
```


Program Name Imputing missing numeric values with most_frequent strategy
demo12.py

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

students = [
    [85, 'M', 'verygood'],
    [95, 'F', 'excellent'],
    [60, None, 'good'],
    [np.NaN, 'M', 'average'],
    [70, 'M', 'good'],
    [np.NaN, None, 'verygood'],
    [60, 'F', 'verygood'],
    [98, 'M', 'excellent']
]

cols = ['marks', 'gender', 'result']
df = pd.DataFrame(students, columns = cols)

imputer = SimpleImputer(missing_values = np.nan, strategy = '
most_frequent')

result = df['marks'].values.reshape(-1, 1)

df.marks = imputer.fit_transform(result)

print(df)
```

output

```
   marks gender  result
0   85.0      M verygood
1   95.0      F  excellent
2   60.0    None    good
3   60.0      M  average
4   70.0      M    good
5   60.0    None verygood
6   60.0      F verygood
7   98.0      M  excellent
```

Program Name Imputing missing numeric values with constant strategy
demo13.py

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

students = [
    [85, 'M', 'verygood'],
    [95, 'F', 'excellent'],
    [60, None, 'good'],
    [np.NaN, 'M', 'average'],
    [70, 'M', 'good'],
    [np.NaN, None, 'verygood'],
    [60, 'F', 'verygood'],
    [98, 'M', 'excellent']
]

cols = ['marks', 'gender', 'result']
df = pd.DataFrame(students, columns = cols)

print(df)

imputer = SimpleImputer(missing_values = np.nan, strategy =
'constant', fill_value = 80)

result = df['marks'].values.reshape(-1, 1)

df.marks = imputer.fit_transform(result)

print()
print(df)
```

output

```

    marks gender  result
0   85.0      M  verygood
1   95.0      F  excellent
2   60.0    None    good
3    NaN      M  average
4   70.0      M    good
5    NaN    None  verygood
6   60.0      F  verygood
7   98.0      M  excellent

    marks gender  result
0   85.0      M  verygood
1   95.0      F  excellent
2   60.0    None    good
3   80.0      M  average
4   70.0      M    good
5   80.0    None  verygood
6   60.0      F  verygood
7   98.0      M  excellent
```