

2. Feature Engineering

Contents

1. Handling Categorical Data	2
2. Encoding Categorical Data	3
2.1. Ordinal encoding.....	4
2.2. One hot encoding.....	6
2.3. Dummy variable encoding	9
2.4. Imputing Missing Class Values	11

2. Feature Engineering

1. Handling Categorical Data

Categorical data

- ✓ Categorical data are variables that contain label values rather than numeric values.

Types of categorical data

- ✓ Nominal Variable
- ✓ Ordinal Variable

Nominal Variable

- ✓ The variables which are having **no-order** those are called as **Nominal** Variable.
- ✓ Examples:
 - Pet variables values : cat, dog
 - Color variables values : blue, green, red

Ordinal Variable

- ✓ The variables which are having an **order** those are called as **ordinal** Variable.
- ✓ Examples:
 - Score variables values : low, medium, high

Kind note

- ✓ In real time mostly we do have nominal variable scenarios.
- ✓ So, please understand the below scenarios

2. Encoding Categorical Data

- ✓ There are 3 ways to convert categorical variables to numerical values.
 - Ordinal encoding
 - One hot encoding
 - Dummy variable encoding

2.1. Ordinal encoding

✓ In ordinal encoding every nominal value is assigned an integer value.

✓ Example

- blue : 0
- green : 1
- red : 2

Program Ordinal encoding
Name demo1.py

```
from numpy import asarray
from sklearn.preprocessing import OrdinalEncoder

data = asarray(['blue'], ['green'], ['red']))

encoder = OrdinalEncoder()
result = encoder.fit_transform(data)

print(data)
print(result)
```

Output

```
['blue']
['green']
['red']]

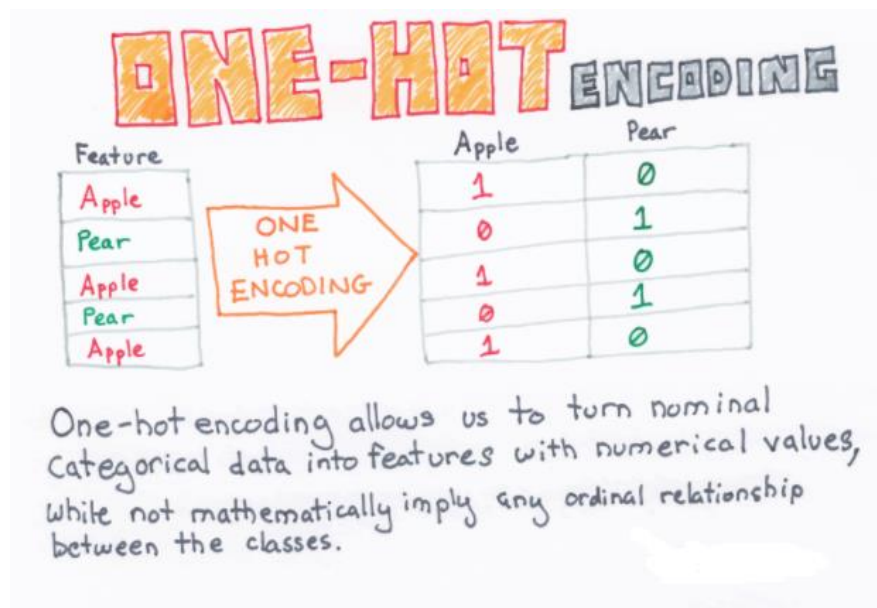
[[0.]
 [1.]
 [2.]]
```

Problem with ordinal encoding

- ✓ If we have applied ordinal encoding on nominal values then it will be an order and having relationship but actually there is no relationship in between the nominal variables.
- ✓ Machine learning algorithm understands like there is an order in between nominal values.
- ✓ So it causes a problem like machine learning algorithm will produce poor performance.
- ✓ We can solve this problem by using **one hot encoding**.

2.2. One hot encoding

- ✓ For **nominal** values integer encoding may not be enough and even it is misleading the model.
- ✓ Here one hot encoding helps, it is technique where each of the nominal variables will be represented with binary values.



✓ Example

- blue : 1 0 0
- green : 0 1 0
- red : 0 0 1

Program Name One hot encoding
demo2.py

```
from numpy import asarray
from sklearn.preprocessing import OneHotEncoder

a = [['apple'], ['peer'], ['apple'], ['peer'], ['apple']]
data = asarray(a)

encoder = OneHotEncoder(sparse = False)
onehot = encoder.fit_transform(data)

print(data)
print()
print(onehot)
```

output

```
[[ 'apple']
 ['peer']
 ['apple']
 ['peer']
 ['apple']]

[[1.  0.]
 [0.  1.]
 [1.  0.]
 [0.  1.]
 [1.  0.]]
```

Program Name One hot encoding
demo3.py

```
from numpy import asarray
from sklearn.preprocessing import OneHotEncoder

data = asarray(['blue'], ['green'], ['red']))
encoder = OneHotEncoder(sparse = False)
onehot = encoder.fit_transform(data)

print(data)
print(onehot)
```

Output

```
['blue']
['green']
['red']]

[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```


2.3. Dummy variable encoding

- ✓ The one hot encoding creates one binary variable for each category.
- ✓ The problem is that this representation includes **redundancy**.
- ✓ For example, if we know that **[1, 0, 0]** represents for **first value** and **[0, 1, 0]** represents for second value then we don't need another binary variable to represent **third value**, instead we could use 0 values alone like **[0, 0]**.

One hot encoding example

- ✓ Example

○ blue :	1	0	0
○ green :	0	1	0
○ red :	0	0	1

Dummy variable encoding example

- ✓ Example

○ blue :	0	0
○ green :	1	0
○ red :	0	1

Conclusion

- ✓ If we drop first column from the result of one hot encoding then we will get dummy variable encoding

Program Name Dummy variable encoding
demo4.py

```
from numpy import asarray
from sklearn.preprocessing import OneHotEncoder

data = asarray(['blue'], ['green'], ['red']))
encoder = OneHotEncoder(drop = 'first', sparse = False)

onehot = encoder.fit_transform(data)

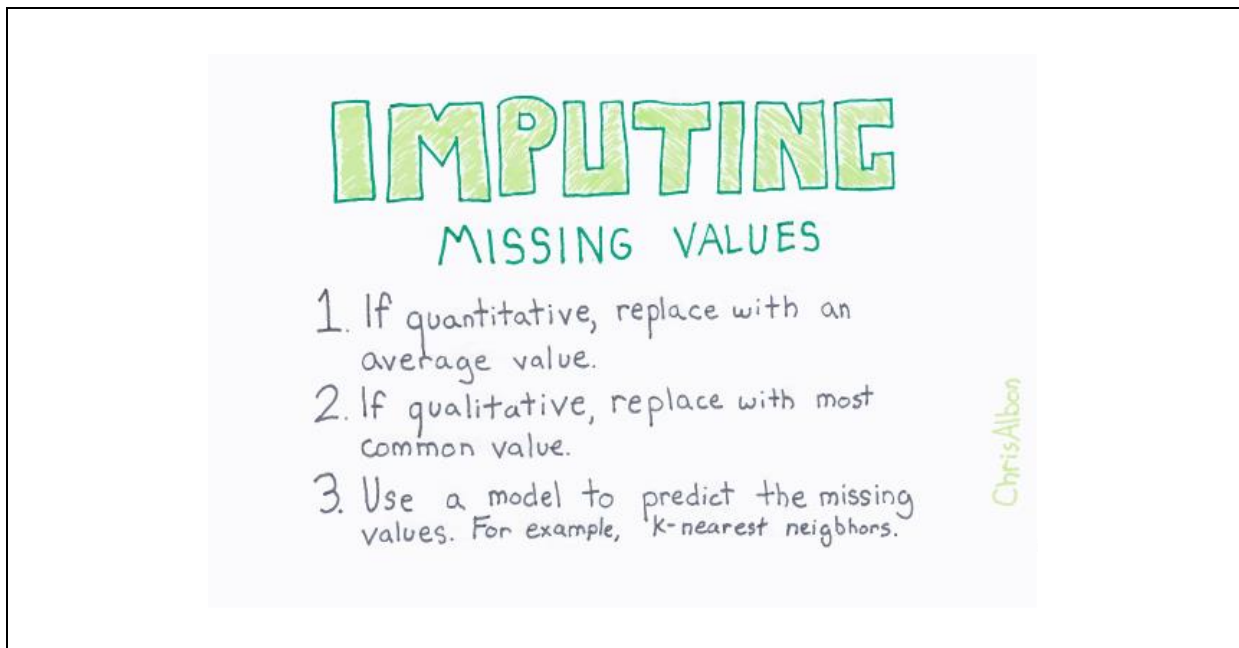
print(data)
print(onehot)
```

Output

```
['blue']
['green']
['red']

[[0. 0.]
 [1. 0.]
 [0. 1.]]
```

2.4. Imputing Missing Class Values



- ✓ Categorical feature may have missing values
- ✓ These we can impute with most frequent strategy

Program Name Imputing categorical values with most frequent strategy
demo5.py

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

students = [
    [85, 'M', 'verygood'],
    [95, 'F', 'excellent'],
    [75, np.NaN, 'good'],
    [np.NaN, 'M', 'average'],
    [70, 'M', 'good'],
    [np.NaN, np.NaN, 'verygood'],
    [92, 'F', 'verygood'],
    [98, 'M', 'excellent']
]

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

print(df)

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

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

df.gender = imputer.fit_transform(result)

print()
print(df)
```

output

```
   marks gender  result
0   85.0      M  verygood
1   95.0      F  excellent
2   75.0     NaN    good
3    NaN      M  average
4   70.0      M    good
5    NaN     NaN  verygood
6   92.0      F  verygood
7   98.0      M  excellent
```

```
   marks gender  result
0   85.0      M  verygood
1   95.0      F  excellent
2   75.0      M    good
3    NaN      M  average
4   70.0      M    good
5    NaN      M  verygood
6   92.0      F  verygood
7   98.0      M  excellent
```