Normalize data using numpy

In [1]:
```
import numpy as np

# Example data
data = np.array([1, 2, 3, 4, 5])

# Normalizing the data
normalized_data = (data - np.min(data)) / (np.max(data) - np.min(data))

print(normalized_data)
```

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

Standardize data using numpy

In [2]:
```
import numpy as np

# Example data
data = np.array([1, 2, 3, 4, 5])

# Standardizing the data
standardized_data = (data - np.mean(data)) / np.std(data)

print(standardized_data)
```

```
[-1.41421356 -0.70710678  0.          0.70710678  1.41421356]
```

Normalize using MinMaxScaler

In [3]:
```
from sklearn.preprocessing import MinMaxScaler

# Example data
data = [[1], [2], [3], [4], [5]]

# Initialize the scaler
scaler = MinMaxScaler()

# Fit and transform the data
normalized_data = scaler.fit_transform(data)

print(normalized_data)
```

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

Standardize using StandardScaler

In [4]:
```
from sklearn.preprocessing import StandardScaler

# Example data
data = [[1], [2], [3], [4], [5]]

# Initialize the scaler
scaler = StandardScaler()

# Fit and transform the data
standardized_data = scaler.fit_transform(data)

print(standardized_data)
```

```
[[-1.41421356]
 [-0.70710678]
 [ 0.        ]
 [ 0.70710678]
 [ 1.41421356]]
```

Normalize on dataframe

In [5]:
```
import pandas as pd

# Example DataFrame
df = pd.DataFrame({'A': [1, 2, 3, 4, 5], 'B': [10, 20, 30, 40, 50]})

# Normalizing the DataFrame
df_normalized = (df - df.min()) / (df.max() - df.min())

print(df_normalized)
```

```
      A     B
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
```

Standardize on DataFrame

In [6]:
```
import pandas as pd

# Example DataFrame
df = pd.DataFrame({'A': [1, 2, 3, 4, 5], 'B': [10, 20, 30, 40, 50]})

# Standardizing the DataFrame
df_standardized = (df - df.mean()) / df.std()

print(df_standardized)
```

```
          A         B
0 -1.264911 -1.264911
1 -0.632456 -0.632456
2  0.000000  0.000000
3  0.632456  0.632456
4  1.264911  1.264911
```

Feature Encoding

Label encoding on list

In [7]:
```
from sklearn.preprocessing import LabelEncoder

# Example data
data = ['red', 'green', 'blue', 'green', 'blue', 'red']

# Initialize the encoder
encoder = LabelEncoder()

# Fit and transform the data
encoded_data = encoder.fit_transform(data)

print(encoded_data)
```

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

Label encoding on DataFrame

In [8]:
```
import pandas as pd
from sklearn.preprocessing import LabelEncoder

# Example DataFrame
df = pd.DataFrame({
    'color': ['red', 'green', 'blue', 'green', 'blue', 'red'],
    'size': ['S', 'M', 'L', 'M', 'S', 'L']
})

# Initialize the LabelEncoder
label_encoder = LabelEncoder()

# Apply label encoding to the 'color' column
df['color_encoded'] = label_encoder.fit_transform(df['color'])

print(df)
```

```
   color size  color_encoded
0    red    S              2
1  green    M              1
2   blue    L              0
3  green    M              1
4   blue    S              0
5    red    L              2
```

One Hot Encoding on DataFrame

In [9]:
```
import pandas as pd

# Example data
df = pd.DataFrame({'color': ['red', 'green', 'blue', 'green', 'blue', 'red']})

# One-hot encoding
one_hot_encoded_data = pd.get_dummies(df, columns=['color'])

print(one_hot_encoded_data)
```

```
   color_blue  color_green  color_red
0           0            0          1
1           0            1          0
2           1            0          0
3           0            1          0
4           1            0          0
5           0            0          1
```

One Hot Encoding using sklearn

In [10]:
```
from sklearn.preprocessing import OneHotEncoder

# Example data
data = [['red'], ['green'], ['blue'], ['green'], ['blue'], ['red']]

# Initialize the encoder
encoder = OneHotEncoder(sparse_output=False)

# Fit and transform the data
one_hot_encoded_data = encoder.fit_transform(data)

print(one_hot_encoded_data)
```

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

Ordinal Encoding

In [11]:
```
from sklearn.preprocessing import OrdinalEncoder

# Example data
data = [['low'], ['medium'], ['high'], ['medium'], ['high'], ['low']]

# Initialize the encoder
encoder = OrdinalEncoder(categories=[['low', 'medium', 'high']])

# Fit and transform the data
ordinal_encoded_data = encoder.fit_transform(data)

print(ordinal_encoded_data)
```

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

In [12]:
```
#pip install category_encoders
```

Binary Encoding

In [13]:
```
import category_encoders as ce

# Example data
data = pd.DataFrame({'city': ['New York', 'Los Angeles', 'Chicago', 'Houston', 'Phoenix']})

# Initialize the encoder
encoder = ce.BinaryEncoder(cols=['city'])

# Fit and transform the data
binary_encoded_data = encoder.fit_transform(data)

print(binary_encoded_data)
```

```
   city_0  city_1  city_2
0       0       0       1
1       0       1       0
2       0       1       1
3       1       0       0
4       1       0       1
```

Log Transformation

In [14]:
```
import numpy as np

# Example data
data = np.array([1, 10, 100, 1000, 10000])

# Logarithmic transformation
log_transformed = np.log(data)

print(log_transformed)
```

```
[0.         2.30258509 4.60517019 6.90775528 9.21034037]
```

Log Transformation on single column

In [15]:
```
import pandas as pd
import numpy as np

# Example DataFrame
df = pd.DataFrame({
    'A': [1, 10, 100, 1000, 10000],
    'B': [5, 15, 25, 35, 45]
})

# Apply log transformation to column 'A'
df['A_log'] = np.log(df['A'])

print(df)
```

```
       A   B     A_log
0      1   5  0.000000
1     10  15  2.302585
2    100  25  4.605170
3   1000  35  6.907755
4  10000  45  9.210340
```

log Transformation on Multiple Columns

In [16]:
```
import pandas as pd
import numpy as np

# Example DataFrame
df = pd.DataFrame({
    'A': [1, 10, 100, 1000, 10000],
    'B': [5, 15, 25, 35, 45],
    'C': [2, 20, 200, 2000, 20000]
})

# Apply log transformation to multiple columns
columns_to_transform = ['A', 'B','C']
df_log_transformed = df.copy()

for col in columns_to_transform:
    df_log_transformed[col + '_log'] = np.log(df_log_transformed[col])

print(df_log_transformed)
```

```
       A   B      C     A_log     B_log     C_log
0      1   5      2  0.000000  1.609438  0.693147
1     10  15     20  2.302585  2.708050  2.995732
2    100  25    200  4.605170  3.218876  5.298317
3   1000  35   2000  6.907755  3.555348  7.600902
4  10000  45  20000  9.210340  3.806662  9.903488
```

Square root transformation

In [17]:
```
import numpy as np

# Example data
data = np.array([1, 4, 9, 16, 25])

# Square root transformation
sqrt_transformed = np.sqrt(data)

print(sqrt_transformed)
```

```
[1. 2. 3. 4. 5.]
```

Exponential transformation

In [18]:
```
import numpy as np

# Example data
data = np.array([0.1, 0.2, 0.3, 0.4, 0.5])

# Exponential transformation
exp_transformed = np.exp(data)

print(exp_transformed)
```

```
[1.10517092 1.22140276 1.34985881 1.4918247  1.64872127]
```