# Machine Learning

# LAB

**Lab #4**

**Feature Scaling and Transformation Pipeline**

**Instructor: Saad Rashad**

**Course Code: AL3002**

**Semester Fall 2024**

**Department of Computer Science,**
**National University of Computer and Emerging Sciences FAST**
**Peshawar Campus**

1. **Feature Scaling:**
   - Is the process of adjusting the range and distribution of feature values so that they are comparable across different features.
   - One of the most important transformations you need to apply to your data is feature scaling.
   - With few exceptions, Machine Learning algorithms don't perform well when the input numerical attributes have very different scales

**1.1. Types of Feature Scaling:**

**1.1.1. Standardization (z-score):**

$$X = (X' - \mu)/\sigma$$

- This method scales the feature so that it has a mean of 0 and a standard deviation of 1.

**Example:**

**Data = [5,10,15,25]**

$\mu$=15

$\sigma$=7.07 ------> $\sigma = \sqrt{(variance)^2}/N$ ------> $\sigma = \sqrt{(X' - \mu)^2}/N$

- For $x_i = 5$:

$$z_i = \frac{5-15}{7.07} \approx -1.42$$

- For $x_i = 10$:

$$z_i = \frac{10-15}{7.07} \approx -0.71$$

- For $x_i = 15$:

$$z_i = \frac{15-15}{7.07} = 0$$

- For $x_i = 20$:

$$z_i = \frac{20-15}{7.07} \approx 0.71$$

- For $x_i = 25$:

$$z_i = \frac{25-15}{7.07} \approx 1.42$$

**Transformed Dataset**

After standardization, the dataset becomes:

$$\text{Standardized Data} = [-1.42, -0.71, 0, 0.71, 1.42]$$

**Key Points**

- **Mean of Transformed Data**: The mean of the standardized data is 0:

$$\text{Mean} = \frac{-1.42 - 0.71 + 0 + 0.71 + 1.42}{5} = 0$$

$$\text{Variance} = \frac{(-1.42)^2 + (-0.71)^2 + 0^2 + 0.71^2 + 1.42^2}{5}$$

$$\text{Variance} = \frac{5.041}{5} = 1.0082$$

**Calculate the Standard Deviation**:

The standard deviation is the square root of the variance:

$$\sigma = \sqrt{1.0082} \approx 1.004$$

**When to Use?**

- Useful when data follows a Gaussian (normal) distribution or when algorithms assume normally distributed data (e.g., **Linear Regression**, **Logistic Regression**, **Support Vector Machines**).

#### 1.1.2. Normalization Scaling (MinMax):

- Min-max scaling is similar to z-score normalization in that it will replace every value in a column with a new value using a formula. In this case, that formula is:

$$m = (x - x_{min}) / (x_{max} - x_{min})$$

Where:

- $m$ is our new value
- $x$ is the original cell value
- $x_{min}$ is the minimum value of the column
- $x_{max}$ is the maximum value of the column

- It rescales the range of numerical features to a predetermined range, typically between 0 and 1 or -1 and 1, to ensure that all features are on the same scale.
- ideal for algorithms that use distances (e.g., **K-Nearest Neighbors**), and when features need to be within a bounded range.

# AL_3002_ML_Lab_4

September 9, 2024

[1]:
```python
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

[2]:
```python
import pandas as pd
```

[3]:
```python
csv_file='/content/drive/MyDrive/Colab Notebooks/housing.csv'
df = pd.read_csv(csv_file)
```

[4]:
```python
import numpy as np
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy='median')
```

[5]:
```python
housing_data=df.select_dtypes(include=[np.number])
```

[15]:
```python
housing_data_mm=housing_data
housing_data_mm
```

[15]:
```
       longitude  latitude  housing_median_age  total_rooms  total_bedrooms  \
0        -122.23     37.88                41.0        880.0           129.0
1        -122.22     37.86                21.0       7099.0          1106.0
2        -122.24     37.85                52.0       1467.0           190.0
3        -122.25     37.85                52.0       1274.0           235.0
4        -122.25     37.85                52.0       1627.0           280.0
...          ...       ...                 ...          ...             ...
20635    -121.09     39.48                25.0       1665.0           374.0
20636    -121.21     39.49                18.0        697.0           150.0
20637    -121.22     39.43                17.0       2254.0           485.0
20638    -121.32     39.43                18.0       1860.0           409.0
20639    -121.24     39.37                16.0       2785.0           616.0

       population  households  median_income  median_house_value
0           322.0       126.0         8.3252            452600.0
1          2401.0      1138.0         8.3014            358500.0
2           496.0       177.0         7.2574            352100.0
3           558.0       219.0         5.6431            341300.0
4           565.0       259.0         3.8462            342200.0
```

```
...          ...       ...          ...           ...
20635      845.0     330.0       1.5603       78100.0
20636      356.0     114.0       2.5568       77100.0
20637     1007.0     433.0       1.7000       92300.0
20638      741.0     349.0       1.8672       84700.0
20639     1387.0     530.0       2.3886       89400.0

[20640 rows x 9 columns]
```

[6]:
```
imputer.fit(housing_data)
```

[6]:
```
SimpleImputer(strategy='median')
```

[7]:
```
imputer.statistics_
housing_data.median().values
```

[7]:
```
array([-1.1849e+02,  3.4260e+01,  2.9000e+01,  2.1270e+03,  4.3500e+02,
        1.1660e+03,  4.0900e+02,  3.5348e+00,  1.7970e+05])
```

[8]:
```
X=imputer.transform(housing_data)
print(X)
```

```
[[-1.2223e+02  3.7880e+01  4.1000e+01 ...  1.2600e+02  8.3252e+00
   4.5260e+05]
 [-1.2222e+02  3.7860e+01  2.1000e+01 ...  1.1380e+03  8.3014e+00
   3.5850e+05]
 [-1.2224e+02  3.7850e+01  5.2000e+01 ...  1.7700e+02  7.2574e+00
   3.5210e+05]
 ...
 [-1.2122e+02  3.9430e+01  1.7000e+01 ...  4.3300e+02  1.7000e+00
   9.2300e+04]
 [-1.2132e+02  3.9430e+01  1.8000e+01 ...  3.4900e+02  1.8672e+00
   8.4700e+04]
 [-1.2124e+02  3.9370e+01  1.6000e+01 ...  5.3000e+02  2.3886e+00
   8.9400e+04]]
```

[9]:
```
housing_tr = pd.DataFrame(X, columns=housing_data.columns,
index=housing_data.index)

print(housing_tr)
```

```
       longitude  latitude  housing_median_age  total_rooms  total_bedrooms  \
0        -122.23     37.88                41.0        880.0           129.0
1        -122.22     37.86                21.0       7099.0          1106.0
2        -122.24     37.85                52.0       1467.0           190.0
3        -122.25     37.85                52.0       1274.0           235.0
4        -122.25     37.85                52.0       1627.0           280.0
...          ...       ...                 ...          ...             ...
20635    -121.09     39.48                25.0       1665.0           374.0
```

```
20636   -121.21   39.49              18.0      697.0            150.0
20637   -121.22   39.43              17.0     2254.0            485.0
20638   -121.32   39.43              18.0     1860.0            409.0
20639   -121.24   39.37              16.0     2785.0            616.0

       population  households  median_income  median_house_value
0           322.0       126.0         8.3252            452600.0
1          2401.0      1138.0         8.3014            358500.0
2           496.0       177.0         7.2574            352100.0
3           558.0       219.0         5.6431            341300.0
4           565.0       259.0         3.8462            342200.0
...           ...         ...            ...                 ...
20635       845.0       330.0         1.5603             78100.0
20636       356.0       114.0         2.5568             77100.0
20637      1007.0       433.0         1.7000             92300.0
20638       741.0       349.0         1.8672             84700.0
20639      1387.0       530.0         2.3886             89400.0

[20640 rows x 9 columns]
```

[10]:
```python
housing_categorical=df[["ocean_proximity"]]
housing_categorical.head(500)
```

[10]:
```
    ocean_proximity
0          NEAR BAY
1          NEAR BAY
2          NEAR BAY
3          NEAR BAY
4          NEAR BAY
..              ...
495          INLAND
496       <1H OCEAN
497       <1H OCEAN
498       <1H OCEAN
499      NEAR OCEAN

[500 rows x 1 columns]
```

[ ]:
```python
from sklearn.preprocessing import OrdinalEncoder
ordinal_encoder = OrdinalEncoder()
housing_cat_encoded = ordinal_encoder.fit_transform(housing_categorical)
```

[ ]:
```python
housing_cat_encoded[:200]
```

[ ]:
```
array([[3.],
       [3.],
       [3.],
       [3.],
```

```
       [0.],
       [4.],
       [1.],
       [3.],
       [3.],
       [0.],
       [1.],
       [0.]])
```

[ ]:
```
ordinal_encoder.categories_
```

[ ]:
```
[array(['<1H OCEAN', 'INLAND', 'ISLAND', 'NEAR BAY', 'NEAR OCEAN'],
       dtype=object)]
```

#**Label Encoding**#

[ ]:
```
from sklearn.preprocessing import LabelEncoder
label_encode=LabelEncoder()
housing_cat_label=label_encode.fit_transform(housing_categorical)
print(housing_cat_label)
```

```
[3 3 3 ... 1 1 1]
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_label.py:114:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  y = column_or_1d(y, warn=True)
```

[ ]:
```
from sklearn.preprocessing import LabelEncoder
label_encode = LabelEncoder()
# If housing_categorical is a DataFrame or a 2D array, convert it to a 1D array
housing_categorical = housing_categorical.values.ravel() if␣
↪hasattr(housing_categorical, 'values') else housing_categorical.ravel()
housing_cat_label = label_encode.fit_transform(housing_categorical)
print(housing_cat_label.tolist())
```

```
[3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 0, 1, 4, 0, 0, 0, 4,
0, 0, 0, 1, 4, 0, 0, 4, 0, 3, 1, 1, 4, 0, 4, 3, 4, 0, 4, 0, 1, 0, 3, 0, 1, 4, 0,
3, 0, 1, 4, 0, 1, 1, 0, 1, 1, 0, 0, 3, 0, 0, 0, 1, 0, 0, 0, 0, 4, 4, 0, 1, 4, 1,
0, 1, 3, 0, 1, 1, 4, 1, 1, 0, 4, 0, 4, 1, 0, 0, 4, 4, 1, 0, 0, 0, 4, 0, 0, 0, 0,
1, 0, 3, 1, 0, 4, 1, 3, 3, 0, 1, 0, 0, 1, 0, 0, 1, 0, 3, 3, 0, 0, 0, 0, 0, 0, 0,
3, 0, 1, 3, 3, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 3, 1, 3, 1, 0, 1, 0, 1, 0, 4, 0, 4,
1, 0, 0, 0, 0, 1, 1, 0, 4, 0, 0, 3, 1, 0, 0, 1, 0, 0, 4, 0, 4, 1, 0, 0, 0, 4, 1,
0, 0, 1, 1, 0, 3, 3, 4, 1, 3, 0, 1, 1, 0, 0, 0, 0, 4, 0, 1, 0, 1, 0, 0, 1, 4, 4,
0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 3, 1, 1, 1, 1, 4, 1, 0, 0, 0,
```

```
4, 0, 0, 0, 3, 0, 0, 0, 0, 3, 3, 0, 1, 0, 1, 1, 0, 3, 0, 4, 4, 3, 4, 1, 4, 1, 0,
4, 4, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 3, 0, 1, 1, 0, 4, 1, 1, 0,
0, 4, 0, 1, 0, 1, 3, 1, 0, 0, 1, 3, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 3, 0, 0, 0,
0, 1, 1, 0, 0, 1, 4, 0, 4, 0, 1, 1, 0, 1, 3, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0,
1, 4, 4, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 3, 1, 4, 1, 4, 1, 1, 1, 1, 4, 1, 0, 1, 1,
3, 1, 3, 4, 0, 1, 0, 3, 3, 0, 4, 3, 0, 0, 1, 1, 0, 4, 4, 4, 0, 0, 4, 1, 1, 1, 1,
0, 3, 1, 1, 3, 0, 0, 0, 0, 3, 3, 0, 3, 0, 4, 1, 0, 1, 0, 4, 1, 0, 0, 1, 3, 0, 1,
4, 4, 3, 0, 4, 1, 1, 4, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 4, 3, 3, 1, 1, 1,
4, 0, 3, 1, 0, 1, 0, 4, 0, 0, 0, 1, 4, 0, 1, 0, 0, 0, 0, 0, 4, 0, 4, 0, 0, 1, 1,
1, 1, 0, 0, 1, 1, 1, 0, 4, 1, 0, 4, 3, 1, 0, 1, 0, 0, 1, 1, 1, 3, 1, 0, 1, 3, 1,
0, 0, 0, 0, 1, 1, 0, 1, 0, 3, 1, 0, 0, 1, 0, 0, 0, 3, 4, 4, 0, 0, 3, 1, 1, 3, 1,
0, 0, 1, 4, 4, 0, 0, 4, 0, 0, 0, 1, 1, 0, 0, 3, 1, 0, 1, 0, 3, 0, 3, 3, 1, 1, 3,
4, 1, 0, 0, 0, 0, 4, 1, 0, 4, 0, 0, 0, 0, 1, 1, 1, 4, 4, 0, 4, 3, 0, 0, 0, 0, 1,
1, 4, 1, 1, 1, 0, 0, 0, 1, 0, 1, 3, 0, 4, 4, 4, 3, 4, 0, 1, 0, 1, 1, 3, 4, 0, 0,
1, 0, 0, 0, 3, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 3, 4, 3, 0, 4, 0, 1, 1, 0,
4, 1, 0, 1, 0, 4, 3, 0, 0, 4, 0, 4, 1, 0, 0, 0, 1, 0, 0, 1, 4, 1, 0, 1, 1, 0, 0,
0, 0, 1, 0, 0, 1, 0, 1, 0, 3, 1, 0, 1, 0, 3, 4, 3, 4, 0, 0, 0, 1, 1, 1, 0, 1, 1,
1, 1, 0, 0, 0, 1, 1, 3, 0, 0, 3, 4, 1, 0, 3, 4, 0, 4, 3, 0, 0, 4, 1, 4, 1, 1, 0,
0, 3, 0, 4, 1, 4, 1, 0, 3, 4, 4, 4, 0, 0, 3, 4, 3, 1, 3, 0, 4, 4, 0, 1, 0, 4, 1,
3, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 3, 3, 0, 1, 0, 1, 0, 1, 0, 1, 0,
0, 0, 3, 4, 1, 0, 1, 1, 4, 1, 0, 0, 3, 0, 0, 3, 0, 0, 1, 1, 3, 1, 0, 0, 4, 0, 0,
3, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 3, 3, 0, 3, 4, 1, 1, 4, 0, 1, 0, 0, 1, 0, 3,
1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 4, 1, 0, 0, 0, 4, 0, 3, 3, 3,
0, 1, 1, 0, 1, 4, 0, 1, 0, 4, 1, 1, 1, 1, 1, 0, 1, 4, 1, 4, 4, 1, 1, 0, 0, 0, 0,
0, 0, 1, 0, 1, 0, 0, 4, 0, 4, 0, 1, 0, 1, 0, 3, 0, 1, 4, 0, 1, 1, 0, 0, 1, 0, 3,
0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 3, 1, 4, 0, 1, 0, 4, 1, 3, 1, 3, 0, 4, 3, 0, 4, 1,
1, 4, 1, 1, 1, 0, 3, 3, 3, 0, 0, 0, 0, 1, 1, 0, 3, 1, 0, 0, 0, 0, 0, 1, 0, 4, 0,
4, 0, 0, 3, 0, 0, 0, 1, 0, 1, 3, 0, 0, 0, 0, 4, 0, 0, 3, 0, 3, 0, 1, 4, 1, 0, 1,
0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 3, 1, 0, 0, 1, 3, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```

## 0.1 Feature Scaling

### 0.1.1 Standardization(StandardScaler)

[13]:
```python
from sklearn.preprocessing import StandardScaler
std_scaler = StandardScaler()
housing_num_std_scaled = std_scaler.fit_transform(housing_data)
housing_num_std_scaled
```

[13]:
```
array([[-1.32783522,  1.05254828,  0.98214266, ..., -0.97703285,
         2.34476576,  2.12963148],
       [-1.32284391,  1.04318455, -0.60701891, ...,  1.66996103,
         2.33223796,  1.31415614],
       [-1.33282653,  1.03850269,  1.85618152, ..., -0.84363692,
```

```
  1.7826994 ,  1.25869341],
...,
[-0.8237132 ,  1.77823747, -0.92485123, ..., -0.17404163,
 -1.14259331, -0.99274649],
[-0.87362627,  1.77823747, -0.84539315, ..., -0.39375258,
 -1.05458292, -1.05860847],
[-0.83369581,  1.75014627, -1.00430931, ...,  0.07967221,
 -0.78012947, -1.01787803]])
```

### 0.1.2 Normalization(MinMax Scaler)

[20]:
```python
from sklearn.preprocessing import MinMaxScaler
min_max_scaler = MinMaxScaler(feature_range=(-1, 1))
housing_num_min_max_scaled = min_max_scaler.fit_transform(housing_data_mm)
housing_num_min_max_scaled
```

[20]:
```
array([[-0.57768924,  0.13496281,  0.56862745, ..., -0.95888834,
         0.07933684,  0.80453276],
       [-0.57569721,  0.13071201, -0.21568627, ..., -0.62604835,
         0.07605412,  0.41649313],
       [-0.57968127,  0.12858661,  1.        , ..., -0.94211478,
        -0.06794389,  0.39010148],
       ...,
       [-0.37649402,  0.46439957, -0.37254902, ..., -0.85791811,
        -0.83447125, -0.6812343 ],
       [-0.39641434,  0.46439957, -0.33333333, ..., -0.88554514,
        -0.8114095 , -0.71257438],
       [-0.38047809,  0.45164718, -0.41176471, ..., -0.82601546,
        -0.73949325, -0.69319302]])
```