# Feature scaling

- **Feature scaling** refers to the methods or techniques used to normalize the range of independent variables in our data, or in other words, the methods to set the feature value range within a similar scale.
- Variables with bigger magnitude / larger value range dominate over those with smaller magnitude / value range
- Scale of the features is an important consideration when building machine learning models.
- Feature scaling is generally the last step in the data preprocessing pipeline, performed **just before training the machine learning algorithms**.

## Feature Scaling importance in some ML Algorithms like

- Linear and Logistic Regression
    - The regression coefficients of linear models are directly influenced by the scale of the variable.
- Gradient descent converges faster when features are on similar scales
- Support Vector Machines
    - Feature scaling helps decrease the time to find support vectors for SVMs
- K-means clustering
    - Euclidean distances are sensitive to feature magnitude.
- Principal Component Analysis (PCA)
    - PCA require the features to be centered at 0.

## Various Feature Scaling Techniques

There are several Feature Scaling techniques, which we will discuss throughout this section:

- Standardisation
- Normalisation
    - Scaling to minimum and maximum values - MinMaxScaling
    - Scaling to maximum value - MaxAbsScaling
    - Scaling to quantiles and median - RobustScaling

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

df=pd.read_csv('titanic.csv', usecols=['Age'])
df.head()
```

Out[1]:

| | Age |
|---|---|
| 0 | 22.0 |
| 1 | 38.0 |
| 2 | 26.0 |
| 3 | 35.0 |
| 4 | 35.0 |

In [2]:
```
df.isnull().sum()
```

Out[2]:
```
Age    177
dtype: int64
```

In [3]:
```
df['Age'].fillna(df.Age.median(),inplace=True)
```

In [4]:
```
df.isnull().sum()
```

Out[4]:
```
Age    0
dtype: int64
```

## Standardisation

Standardisation involves centering the variable at zero, and standardising the variance to 1.

**z = (x - x_mean) / std**

The result of the above transformation is **z**, which is called the z-score, and represents how many standard deviations a given observation deviates from the mean. A z-score specifies the location of the observation within a distribution (in numbers of standard deviations respect to the mean of the distribution). The sign of the z-score (+ or - ) indicates whether the observation is above (+) or below ( - ) the mean.

The shape of a standardised (or z-scored normalised) distribution will be identical to the original distribution of the variable. If the original distribution is normal, then the standardised distribution will be normal. But, if the original distribution is skewed, then the standardised distribution of the variable will also be skewed.

standardisation:

- centers the mean at 0
- scales the variance at 1
- preserves the shape of the original distribution
- the minimum and maximum values of the different variables ma var

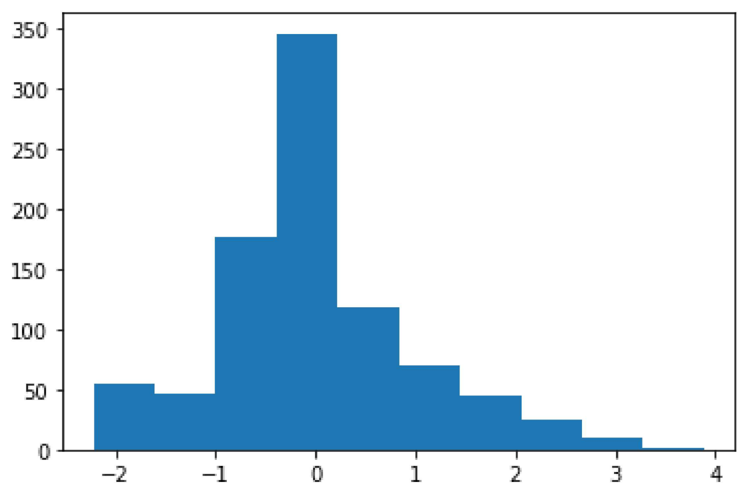
,

- preserves outliers

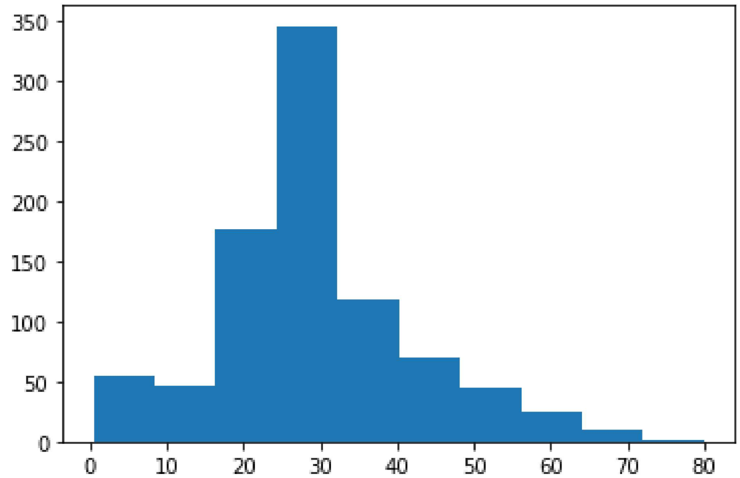
In [5]:
```
### standarisation: We use the Standardscaler from sklearn library
from sklearn.preprocessing import StandardScaler

### Call the function
sc=StandardScaler()

### fit_transform
df['Age_sc']=sc.fit_transform(df[["Age"]])

df['Age_sc']
```

Out[5]:
```
0     -0.565736
1      0.663861
2     -0.258337
3      0.433312
4      0.433312
         ...
886   -0.181487
887   -0.796286
888   -0.104637
889   -0.258337
890    0.202762
Name: Age_sc, Length: 891, dtype: float64
```

In [6]:
```
plt.hist(df['Age_sc']) # sc
plt.show()
```



In [7]:
```
plt.hist(df['Age']) # original
plt.show()
```




### Min Max Scaling (CNN --Deep Learning Techniques)

Min Max Scaling scales the values between 0 to 1. X_scaled = (X - X.min / (X.max - X.min)

In [8]:
```
### Normalization: We use the MinMaxScaler from sklearn library
from sklearn.preprocessing import MinMaxScaler

### Call the function
min_max=MinMaxScaler()

### fit_transform
df['Age_mm']=min_max.fit_transform(df[["Age"]])

df['Age_mm']
```

Out[8]:
```
0      0.271174
1      0.472229
2      0.321438
3      0.434531
4      0.434531
         ...
886    0.334004
887    0.233476
888    0.346569
889    0.321438
890    0.396833
Name: Age_mm, Length: 891, dtype: float64
```

***Robust Scaler***

It is used to scale the feature to median and quantiles Scaling using median and quantiles consists of substracting the median to all the observations, and then dividing by the interquantile difference. The interquantile difference is the difference between the 75th and 25th quantile:

IQR = 75th quantile - 25th quantile

X_scaled = (X - X.median) / IQR

0,1,2,3,4,5,6,7,8,9,10

9-90 percentile---90% of all values in this group is less than 9 1-10 precentile---10% of all values in this group is less than 1 4-40%

In [9]:
```
### Normalization: We use the RobustScaler from sklearn library
from sklearn.preprocessing import RobustScaler

### Call the function
rs = RobustScaler()

### fit_transform
df['Age_rs']=rs.fit_transform(df[["Age"]])

df['Age_rs']
```

Out[9]:
```
0     -0.461538
1      0.769231
2     -0.153846
3      0.538462
4      0.538462
         ...
886   -0.076923
887   -0.692308
888    0.000000
889   -0.153846
890    0.307692
Name: Age_rs, Length: 891, dtype: float64
```

### MaxAbsScaling

- The MaxAbsScaler from scikit-learn re-scales features to their maximum value, so that the new maximum value is 1.
- When performing maximum absolute scaling on the data set, we need to first identify the maximum values of the variables.

```
X_scaled = X/Xmax
```

In [10]:
```python
### Normalization: We use the MaxAbsScaler from sklearn library
from sklearn.preprocessing import MaxAbsScaler

### Call the function
mas = MaxAbsScaler()

### fit_transform
df['Age_mas']=mas.fit_transform(df[["Age"]])

df['Age_mas']
```

Out[10]:
```
0      0.2750
1      0.4750
2      0.3250
3      0.4375
4      0.4375
        ...
886    0.3375
887    0.2375
888    0.3500
889    0.3250
890    0.4000
Name: Age_mas, Length: 891, dtype: float64
```

In [11]:
```python
plt.hist(df['Age_mas']) # mas
plt.show()
```