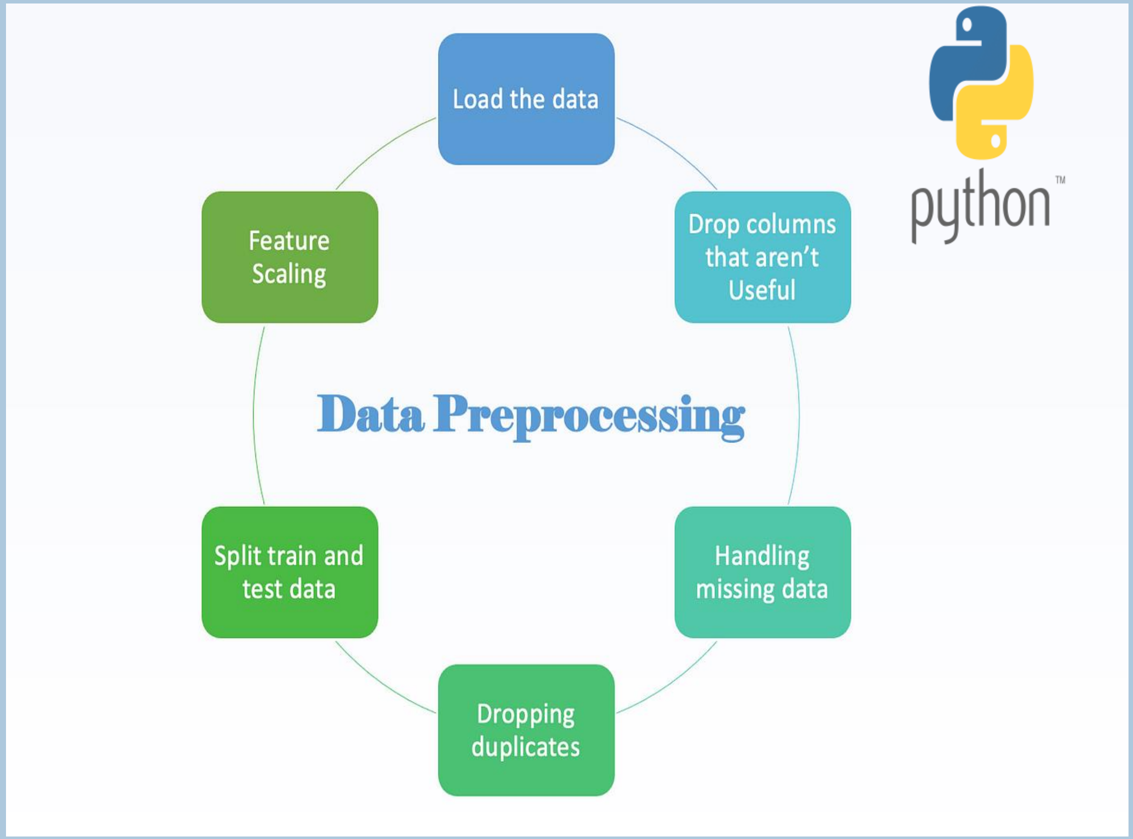

Load the data
Drop columns that aren't Useful
Handling missing data
Dropping duplicates
Split train and test data
Feature Scaling
Data Preprocessing
python

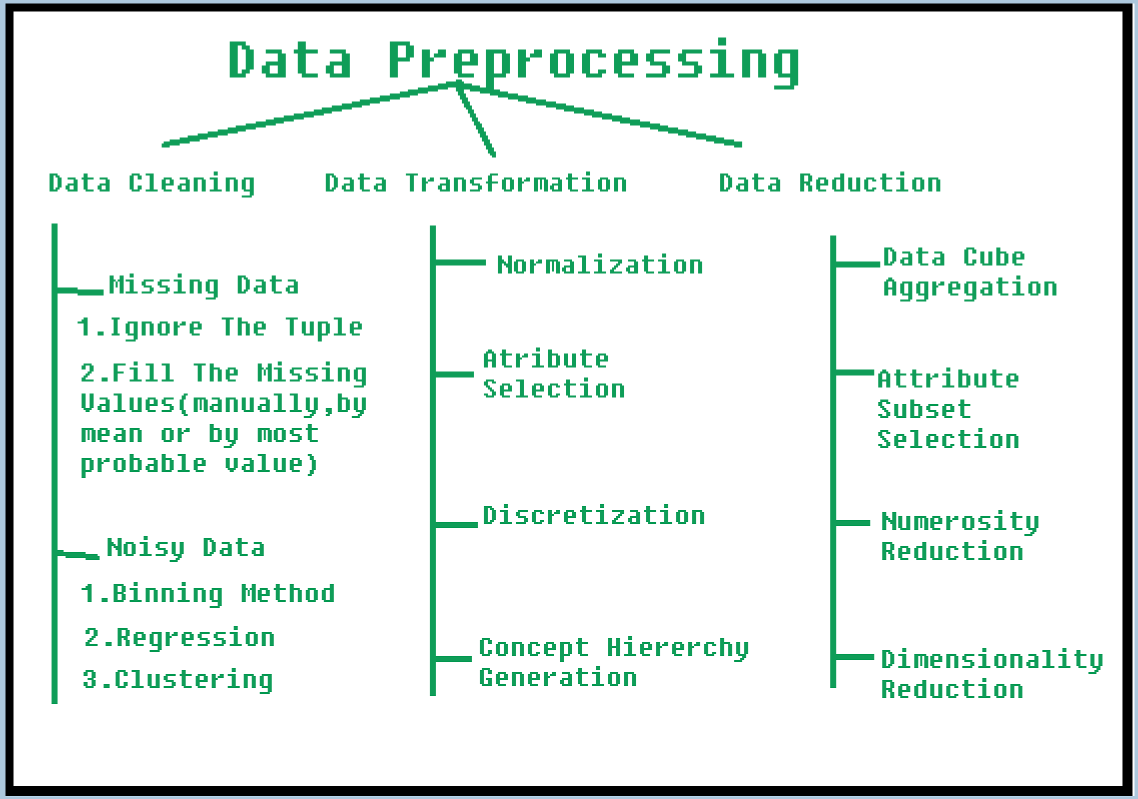
What Is Data Preprocessing?

Data preprocessing is a step in the data mining and data analysis process that takes raw data and transforms it into a format that can be understood and analyzed by computers and machine learning.
Raw, real-world data in the form of text, images, video, etc., is messy. Not only may it contain errors and inconsistencies, but it is often incomplete, and doesn't have a regular, uniform design.

Data Preprocessing Importance

When using data sets to train machine learning models, you'll often hear the phrase **"garbage in, garbage out"** This means that if you use bad or "dirty" data to train your model, you'll end up with a bad, improperly trained model that won't actually be relevant to your analysis.

Good, preprocessed data is even more important than the most powerful algorithms, to the point that machine learning models trained with bad data could actually be harmful to the analysis you're trying to do – giving you "garbage" results.


Data Preprocessing
Data Cleaning
Data Transformation
Data Reduction
Missing Data
1.Ignore The Tuple
2.Fill The Missing Values(manually,by mean or by most probable value)
Noisy Data
1.Binning Method
2.Regression
3.Clustering
Normalization
Atribute Selection
Discretization
Concept Hiererchy Generation
Data Cube Aggregation
Attribute Subset Selection
Numerosity Reduction
Dimensionality Reduction

الجامعة الألمانية الأردنية
German Jordanian University

Data Preprocessing

Data preprocessing is an important step in the data mining process that involves cleaning and transforming raw data to make it suitable for analysis. Some common steps in data preprocessing include:

Data Cleaning: This involves identifying and correcting errors or inconsistencies in the data, such as missing values, outliers, and duplicates.

Data Integration: This involves combining data from multiple sources to create a unified dataset. Data integration can be challenging as it requires handling data with different formats, structures, and semantics.

Data Transformation: This involves converting the data into a suitable format for analysis. Common techniques used in data transformation include normalization, standardization, and discretization. Normalization is used to scale the data to a common range, while standardization is used to transform the data to have zero mean and unit variance. Discretization is used to convert continuous data into discrete categories.

Data Reduction: This involves reducing the size of the dataset while preserving the important information.

Data Discretization: This involves dividing continuous data into discrete categories or intervals.

Data Normalization: This involves scaling the data to a common range, such as between 0 and 1 or -1 and 1. Normalization is often used to handle data with different units and scales.

Data preprocessing plays a crucial role in ensuring the quality of data and the accuracy of the analysis results. The specific steps involved in data preprocessing may vary depending on the nature of the data and the analysis goals.

By performing these steps, the data mining process becomes more efficient and the results become more accurate.

Features in machine learning

Individual independent variables that operate as an input in our machine learning model are referred to as features. They can be thought of as representations or attributes that describe the data and help the models to predict the classes/labels.

Data sets can be explained with or communicated as the “features” that make them up. This can be by size, location, age, time, color, etc. Features appear as columns in datasets and are also known as attributes, variables, fields, and characteristics.

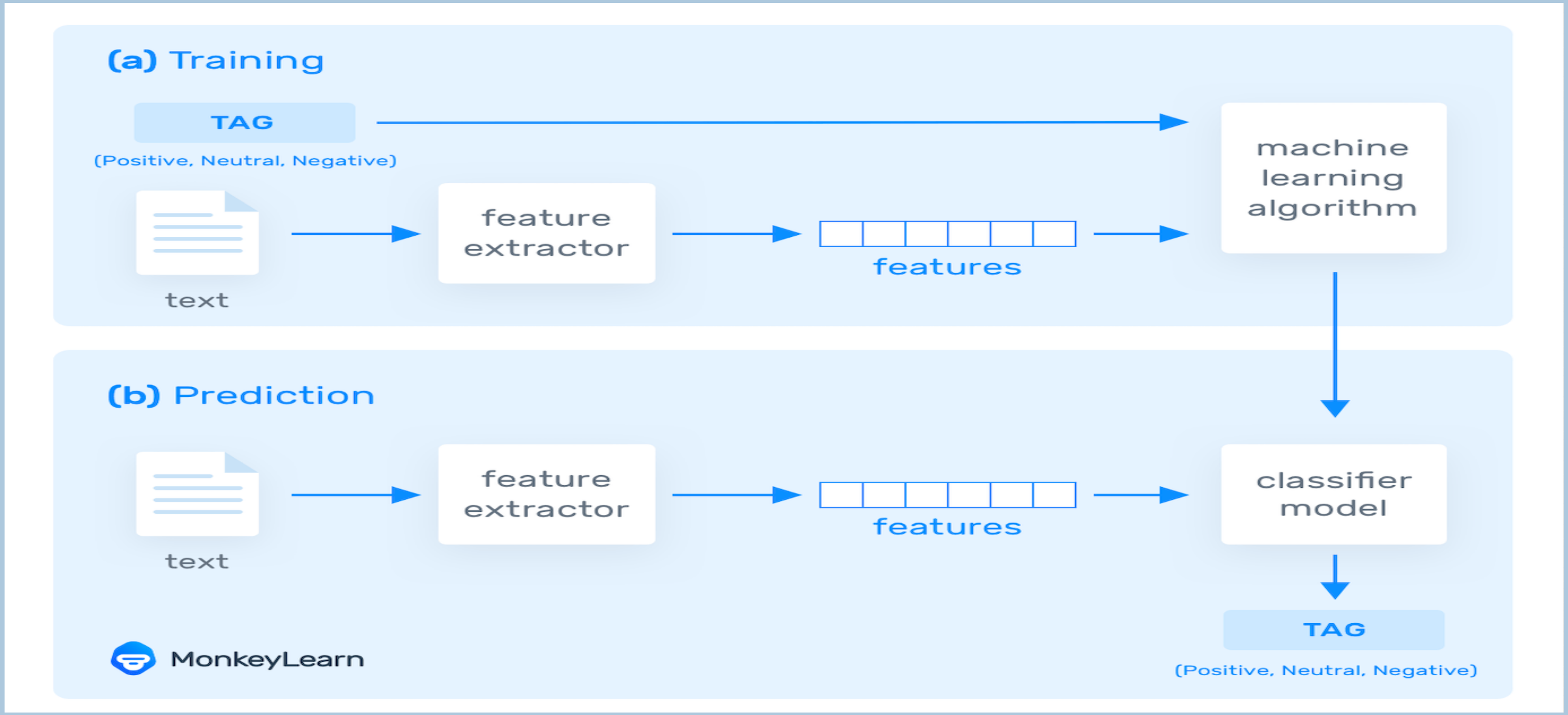
Types of features

•**Categorical features:** Features whose explanations or values are taken from a defined set of possible explanations or values. Categorical values can be colors of a house; types of animals; months of the year; True/False; positive, negative, neutral, etc. The set of possible categories that the features can fit into is predetermined.

•**Numerical features:** Features with values that are continuous on a scale, statistical, or integer-related. Numerical values are represented by whole numbers, fractions, or percentages. Numerical features can be house prices, word counts in a document, time it takes to travel somewhere, etc.

Example

The diagram below shows how features are used to train machine learning text analysis models. Text is run through a feature extractor (to pull out or highlight words or phrases) and these pieces of text are classified or tagged by their features. Once the model is properly trained, text can be run through it, and it will make predictions on the features of the text or "tag" the text itself.



Step 1: Data Collection

Suppose we have all the necessary data; we can proceed with creating a dataset.

Suppose we have all the necessary data; we can proceed with creating a dataset.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
# load the data
df = pd.read_csv('credit_scoring_eng.csv')
```

Data Description

df.head(5)

	children	days_employed	dob_years	education	education_id	family_status	family_status_id	gender	income_type	debt	total_income	purpose
0	1	-8437.673028	42	bachelor's degree	0	married	0	F	employee	0	40620.102	purchase of the house
1	1	-4024.803754	36	secondary education	1	married	0	F	employee	0	17932.802	car purchase
2	0	-5623.422610	33	Secondary Education	1	married	0	M	employee	0	23341.752	purchase of the house
3	3	-4124.747207	32	secondary education	1	married	0	M	employee	0	42820.568	supplementary education
4	0	340266.072047	53	secondary education	1	civil partnership	1	F	retiree	0	25378.572	to have a wedding

children - number of children in the family
days_employed - number of days employed
dob_years - client's age in years
education - client's education level
education_id - education identifier
family_status - marital status
family_status_id - marital status identifier
gender - client's gender
income_type - type of employment
debt - whether the client has a loan debt
total_income - monthly income
purpose - purpose of the loan application

Step 2: Data Cleaning

This involves identifying and correcting errors or inconsistencies in the data, such as missing values, outliers and duplicates. Various techniques can be used for data cleaning, such as imputation, removal or transformation.

a: Handling missing values

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21525 entries, 0 to 21524
Data columns (total 12 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   children          21525 non-null  int64  
 1   days_employed     19351 non-null  float64
 2   dob_years         21525 non-null  int64  
 3   education         21525 non-null  object 
 4   education_id      21525 non-null  int64  
 5   family_status     21525 non-null  object 
 6   family_status_id  21525 non-null  int64  
 7   gender            21525 non-null  object 
 8   income_type       21525 non-null  object 
 9   debt              21525 non-null  int64  
 10  total_income      19351 non-null  float64
 11  purpose           21525 non-null  object 
dtypes: float64(2), int64(5), object(5)
memory usage: 2.0+ MB
```

Findings:

There are missing values in the columns days_employed and total_income, because the number of rows should be 21,525

# check the percentage
df.isna().sum() / len(df)

```
children              0.000000
days_employed         0.100999
dob_years             0.000000
education             0.000000
education_id          0.000000
family_status         0.000000
family_status_id      0.000000
gender                0.000000
income_type           0.000000
debt                  0.000000
total_income          0.100999
purpose               0.000000
dtype: float64
```

Findings:
The missing value percentage for both columns are around 10%

```
plt.figure(figsize=(10,6))
sns.heatmap(df.isna().transpose(),
        cmap="YlGnBu",
        cbar_kws={'label': 'Missing Data'})
```

Findings:

1. Missing values form a pattern. The missing values are caused by job types where clients with the job types ‘student’ and ‘unemployed’ do not have any income, leading them to leave the ‘days_employed’ and ‘total_income’ columns empty.

2. This conclusion is reinforced by the pattern shown in the seaborn heatmap, indicating that when the value in the ‘days_employed’ column is missing, the data in the same row for ‘total_income’ is also missing (symmetrical).

3. Since the missing values are only present in the ‘days_employed’ and ‘total_income’ columns, and both of these columns have float data types, which fall under the Numeric/Ratio category, the missing data will be filled using statistical calculations (such as Mean, Median).

4. Median is chosen to fill in missing values because it can prevent the occurrence of outliers

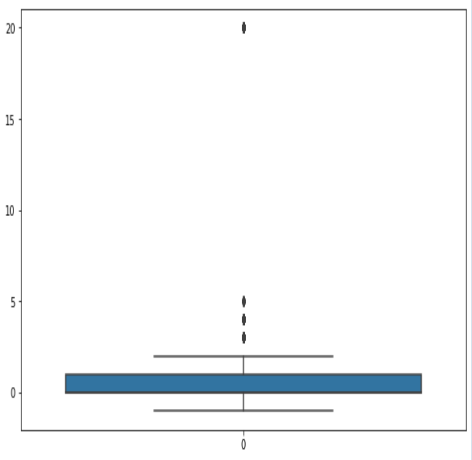
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
# load the data
df = pd.read_csv('credit_scoring_eng.csv')

print(df.info())


df['days_employed'] = df['days_employed'].fillna(df['days_employed'].median())
df['total_income'] = df['total_income'].fillna(df['total_income'].median())

print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21453 entries, 0 to 21452
Data columns (total 17 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   children                  21453 non-null  int64
 1   days_employed             21453 non-null  float64
 2   dob_years                 21453 non-null  int64
 3   education                 21453 non-null  object
 4   education_id              21453 non-null  int64
 5   family_status             21453 non-null  object
 6   family_status_id          21453 non-null  int64
 7   gender                    21453 non-null  object
 8   income_type               21453 non-null  object
 9   debt                      21453 non-null  int64
 10  total_income              21453 non-null  float64
 11  purpose                   21453 non-null  object
 12  debt_status               21453 non-null  object
 13  employee_length_category  21453 non-null  object
 14  age_category              21453 non-null  object
 15  purpose_category          21453 non-null  object
 16  income_category           21453 non-null  object
dtypes: float64(2), int64(5), object(10)
memory usage: 2.8+ MB
```

German Jordanian University

Handling outliers

```
# check outlier in children column
sns.boxplot(df['children'])
```




```
# check statistical data in children column
df['children'].describe()
```

```
count    21525.000000
mean         0.538908
std          1.381587
min         -1.000000
25%          0.000000
50%          0.000000
75%          1.000000
max         20.000000
Name: children, dtype: float64
```

Findings:

1. Based on the statistical data, I will replace the value 20 with the value 2, assuming it was an input error.
2. I will remove the minus sign (-), assuming it was an input error

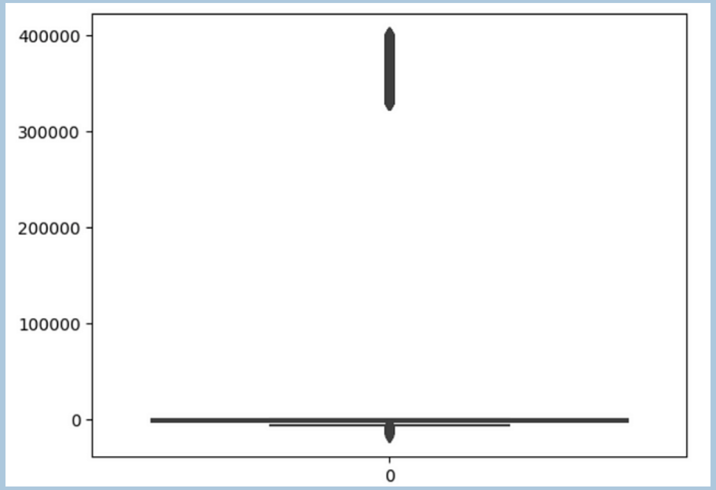
```
# replace the value 20 with the value 2
df['children'].values[df['children'].values ==20] = 2
# remove minus sign
df['children'] = abs(df['children'])
```

```
# Verify the data
sorted(df['children'].unique())
```

```
[0, 1, 2, 3, 4, 5]
```

```
# check outliers in days_employed column
sns.boxplot(df['days_employed'])

# check percentage
len(df.loc[(df['days_employed'] < 0 ) | (df['days_employed'] > 200000)]) / len(df)
```

Result:
0.8990011614401858



Findings:

There are 2 issues identified in the ‘days_employed’ column:
Too many digits after the decimal point.
Existence of negative values and outliers, with a high percentage of rows having these conditions, approximately 89%.

2. The steps to solve these issues are as follows:
Remove the minus sign (-).
Perform rounding.
Replace the outlier values.

```
# remove minus sign (-), assuming it was an input error
df['days_employed'] = abs(df['days_employed'])
# round
df['days_employed'] = round(df['days_employed'],0)
# check data distribution
df['days_employed'].describe()
```

Findings:

The mean value does not represent the data as it is mixed with outliers. Therefore, the replacement of outliers will be done using the median value.

```
count     19351.000000
mean      66914.727973
std      139030.879631
min          24.000000
25%         927.000000
50%        2194.000000
75%        5538.000000
max      401755.000000
Name: days_employed, dtype: float64
```

Replace outlier with median

```
# Replace outlier with median (values with outlier values)

df['days_employed'].values[df['days_employed'].values >
200000] = df['days_employed'].median()
```

```
# Replace outlier with median (values without outlier values)
indexAge = df[(df['days_employed'] > 200000) | (df['days_employed'].isnull() )].index
x=df['days_employed']
x.drop(indexAge , inplace=True)
df['days_employed'].values[(df['days_employed'].values > 200000) | (df['days_employed'].isnull())] =
x.median()
```

Handling Duplicates

```
# check duplicate data
df.duplicated().sum()
```

```
72
```

Findings

1. There are 72 identified duplicate data entries.
2. These duplicate data entries will be removed, and the index will be reset.

remove duplicate data and do reset index

df = df.drop_duplicates().reset_index(drop=True)

```
# verify the data
df.duplicated().sum()
```

```
0
```

Data Normalization

Normalization refers to rescaling real-valued numeric attributes into a 0 to 1 range.

Data normalization is used in machine learning to make model training less sensitive to the scale of features. This allows our model to converge to better weights and, in turn, leads to a more accurate model.

Normalization makes the features more consistent with each other, which allows the model to predict outputs more accurately.

Normalization

- An attribute is **normalized** by scaling its values so that they fall within a small specified range.
- A larger range of an attribute gives a greater effect (weight) to that attribute.
  - This means that an attribute with a larger range can have greater weight at data minining tasks than an attribute with a smaller range.
- Normalizing the data attempts to give all attributes an equal weight.
  - Normalization is particularly useful for classification algorithms involving neural networks or distance measurements such as nearest-neighbor classification and clustering.

Some Normalization Methods:

- **Min-max normalization**
- **Z-score normalization**
- **Normalization by decimal scaling**

Normalize

```
from sklearn import preprocessing
import numpy as np
x_array = np.array([2,3,5,6,7,4,8,7,6])
normalized_arr = preprocessing.normalize([x_array])
print(normalized_arr)
```

```
[[0.11785113 0.1767767  0.29462783 0.35355339 0.41247896 0.23570226
  0.47140452 0.41247896 0.35355339]]
```

Min-Max Normalization

- **Min-max normalization** performs a linear transformation on the original data.
- Suppose that $\mathbf{min_A}$ and $\mathbf{max_A}$ are minimum and maximum values of an attribute A.
- **Min-max normalization** maps a value, $\boldsymbol{v_i}$ of an attribute A to $\boldsymbol{v'_i}$ in the range [$\mathbf{new\_min_A}$,$\mathbf{new\_max_A}$] by computing:

$$v'_i = \frac{v_i - min_A}{max_A - min_A}(new\_max_A - new\_min_A) + new\_min_A$$

- Min-max normalization preserves the relationships among the original data values.
- We can standardize the range of all the numerical attributes to [0,1] by applying ***min-max normalization*** with *newmin=0* and *newmax=1* to all the numeric attributes.

Min-Max Normalization: Example

- Suppose that the range of the attribute *income* is \$12,000 to \$98,000. We want to normalize *income* to range [0.0, 1.0].
- Then \$73,000 is mapped to

$$\text{newvalue}(73000) = \frac{73000 - 12000}{98000 - 12000}(1.0 - 0.0) + 0 = 0.716$$

- Suppose that the range of the attribute *income* is \$12,000 to \$98,000. We want to normalize *income* to range[1.0, 5.0].
- Then \$73,000 is mapped to

$$\text{newvalue}(73000) = \frac{73000 - 12000}{98000 - 12000}(5.0 - 1.0) + 1.0 = 3.864$$

GJU
الجامعة الألمانية الأردنية
German Jordanian University

Min-Max Normalization

There is another way of data scaling, where the minimum of feature is made equal to zero and the maximum of feature equal to one. MinMax Scaler shrinks the data within the given range, usually of 0 to 1. It transforms data by scaling features to a given range. It scales the values to a specific value range without changing the shape of the original distribution.

```
# import module
from sklearn.preprocessing import MinMaxScaler

# create data
data = [[11, 2], [3, 7], [0, 10], [11, 8]]

# scale features
scaler = MinMaxScaler()
model=scaler.fit(data)
scaled_data=model.transform(data)

# print scaled features
print(scaled_data)

print(scaler.transform([[2, 2]]))
```

Output:

```
[[1.         0.        ]
 [0.27272727 0.625     ]
 [0.         1.        ]
 [1.         0.75      ]]

[[0.18181818 0.        ]]
```

```
scaler = MinMaxScaler(feature_range=(5,10))
model=scaler.fit(data)
scaled_data=model.transform(data)

# print scaled features
print(scaled_data)
```

```
[[10.          5.        ]
 [ 6.36363636  8.125     ]
 [ 5.         10.        ]
 [10.          8.75      ]]

[[5.90909091 5.        ]]
```

Z-score Normalization

- The z-score (or ***zero-mean normalization***), is a score that measures how many standard deviations a data point is away from the mean. The z-score allows us to determine how usual or unusual a data point is in a distribution. The z-score allows us more easily compare data points for a record across features, especially when the different features have significantly different ranges.

In statistics, a **z-score** tells us how many standard deviations away a value is from the mean. We use the following formula to calculate a z-score:

$$\mathbf{z} = (X - \mu) / \sigma$$

where:
X is a single raw data value
μ is the population mean
σ is the population standard deviation

```
import pandas as pd
import numpy as np
import scipy.stats as stats
```

Step 2: Create an array of values.

```
data = np.array([6, 7, 7, 12, 13, 13, 15, 16, 19, 22])
```

Step 3: Calculate the z-scores for each value in the array.

```
stats.zscore(data)

[-1.394, -1.195, -1.195, -0.199, 0, 0, 0.398, 0.598, 1.195, 1.793]
```

Each z-score tells us how many standard deviations away an individual value is from the mean. For example:

- The first value of “6” in the array is **1.394** standard deviations *below* the mean.
- The fifth value of “13” in the array is **0** standard deviations away from the mean, i.e. it is equal to the mean.
- The last value of “22” in the array is **1.793** standard deviations *above* the mean.

Detecting outliers

```
import pandas as pd
import numpy as np
import scipy.stats as stats

data = np.array([3,9, 23, 43,53, 4, 5,30, 35, 50, 70, 150, 6, 7, 8, 9, 10])

print(stats.zscore(data))
```

```
[-0.7574907  -0.59097335 -0.20243286  0.35262498  0.6301539  -0.72973781
 -0.70198492 -0.00816262  0.13060185  0.54689523  1.10195307  3.32218443
 -0.67423202 -0.64647913 -0.61872624 -0.59097335 -0.56322046]
```

The outliers in the dataset is[150]

•Z-scores can be used to identify outliers in a dataset. Data points with Z-scores beyond a certain threshold (usually 3 standard deviations from the mean) may be considered outliers.

•Z-scores can be used in anomaly detection algorithms to identify instances that deviate significantly from the expected behavior.

Discretization

Discretization: To transform a numeric (continuous) attribute into a categorical attribute.

- Some data mining algorithms require that data be in the form of categorical attributes.
- In discretization:
  - The range of a continuous attribute is divided into intervals.
  - Then, interval labels can be used to replace actual data values to obtain a categorical attribute.

Simple Discretization Example: *income* attribute is discretized into a categorical attribute.

- Target categories (low, medium, high).
- Calculate average *income:* AVG.
  - If income> 2* AVG, new_income_value = “high”.
  - If income < 0.5* AVG, new_income_value = “low”.
  - Otherwise, new_income_value = “medium”.

Discretization Methods

- A basic distinction between discretization methods for classification is whether class information is used (**supervised**) or not (**unsupervised**).

- Some of discretization methods are as follows:

Unsupervised Discretization: If class information is not used, then relatively simple approaches are common.

- Binning
- Clustering analysis

Supervised Discretization:

- Classification (e.g., decision tree analysis)
- Correlation (e.g., χ^2) analysis

Discretization by Binning

- Attribute values can be discretized by applying **equal-width** or **equal-frequency binning**.
- Binning aproaches sorts the atribute values first, then partition them into the bins.
  - **equal width approach** divides the range of the attribute into a user-specified number of intervals each having the same width.
  - **equal frequency (equal depth) approach** tries to put the same number of objects into each interval.
- After bins are determined, all values are replaced by **bin labels** to discretize that attribute.
  - Instead of bin labels, values may be replaced by bin means (or medians).

- Binning does not use class information and is therefore an unsupervised discretization technique.

Discretization by Binning: Example

equal-width approach

- Suppose a group of 12 values of *price* attribute has been sorted as follows:

price	5	10	11	13	15	35	50	55	72	89	204	215

equal-width partitioning: The width of each interval is (215-5)/3 = 70.

- Partition them into ***three bins***

bin1	5, 10, 11, 13, 15, 35, 50, 55, 72
bin2	89
bin3	204, 215

- Replace each value with its bin label to discretize.

price	5	10	11	13	15	35	50	55	72	89	204	215
categorical attr.	1	1	1	1	1	1	1	1	1	2	3	3

German Jordanian University

Discretization by Binning: Example equal-width approach Python

```
import pandas as pd
import numpy as np

x=pd.cut([5,10,11,13,15,35,50,55,72,89,204,215], bins=3,labels=[1, 2, 3])
print(x)

for i in x:
    print(i,end=' ')
```

1 1 1 1 1 1 1 1 1 2 3 3

Discretization by Binning: Example

equal-frequency approach

- Suppose a group of 12 values of *price* attribute has been sorted as follows:

price	5	10	11	13	15	35	50	55	72	89	204	215

equal-frequency partitioning:

- Partition them into ***three bins:*** each interval contains 4 values

bin1	5, 10, 11, 13
bin2	15, 35, 50, 55
bin3	72, 89, 204, 215

- Replace each value with its bin label to discretize.

price	5	10	11	13	15	35	50	55	72	89	204	215
categorical attr.	1	1	1	1	2	2	2	2	3	3	3	3

Discretization by Binning: Example
equal-frequency approach
Python

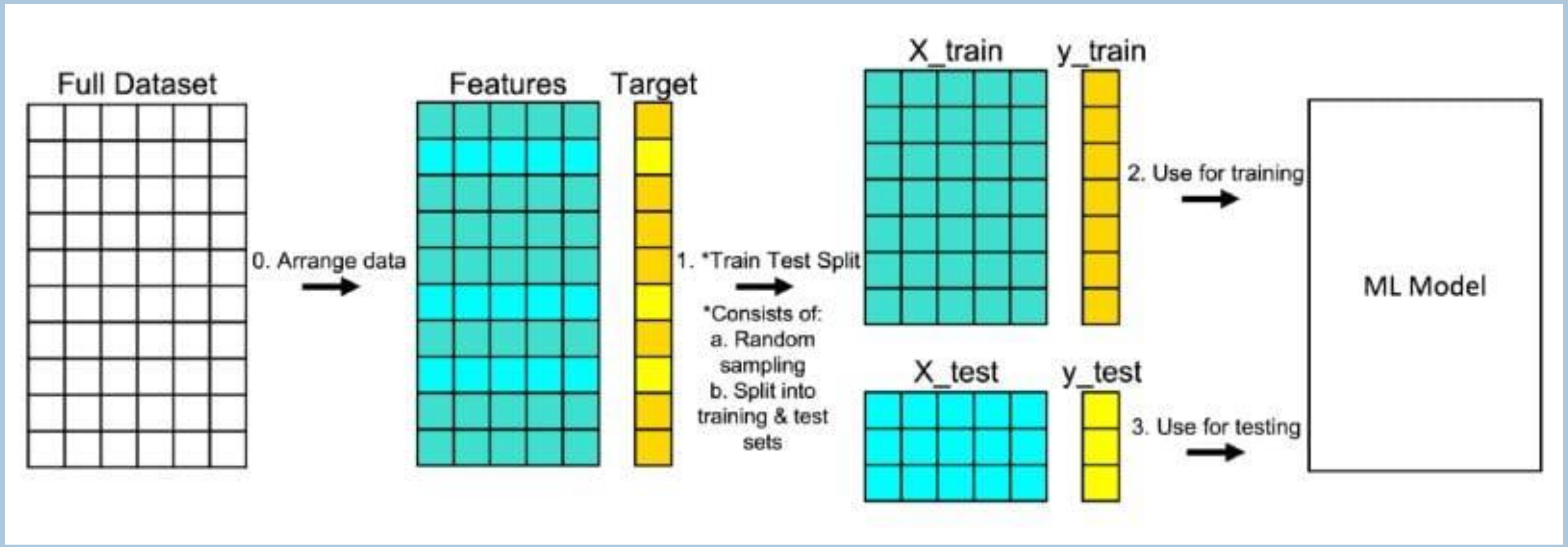
Quantile-based discretization function.

```
import pandas as pd
import numpy as np


x=pd.qcut([5,10,11,13,15,35,50,55,72,89,204,215],
q=3,labels=[1, 2, 3])
print(x)

for i in x:
    print(i,end=' ')
```

Train Test Split

What Is the Train Test Split Procedure?

Train test split is a model validation procedure that allows you to simulate how a model would perform on new/unseen data. Here is how the procedure works:



Example of Using Machine Learning

- Example of Golf Dataset
  - The aim of this machine learning application is to predict whether to play golf or not based on the Weather conditions.
- Steps:
  - **Step 1:** Import the required libraries
  - **Step 2:** Create or read the data file
  - **Step 3:** Apply machine learning algorithms and select the best one

Import the required Libraries

- There are two main libraries that you will need to install, which are:
  - **pandas**
  - **sklearn (scikit-learn)**
- **Import Dependencies:**
  - `import pandas as pd`
  - `from sklearn.model_selection import train_test_split`
  - `from sklearn import metrics`

Create or Read the Data file (Dataset)

- Creating an empty Pandas data frame, and inputting data into every column/feature/attribute.
- **Data Description:**
  - **Outlook** = The outlook of the weather
    - **Outlook values:** sunny, overcast, rainy
  - **Temperature** = The temperature of the weather
    - **Temperature values:** hot, mild, cold
  - **Humidity** = The humidity of the weather
    - **Humidity values:** high, normal
  - **Windy** = A variable if it is windy that day or not
    - **Windy values:** true, false
  - **Play** = The target variable, tells if the golfer played golf that day or not
    - **Play values:** yes, no

GJU
الجامعة الألمانية الأردنية
German Jordanian University

Create or Read the Data file (Dataset)

DATA MINING PATTERNS

	Outlook	Temperature	Humidity	Windy	Play
0	sunny	hot	high	false	no
1	sunny	hot	high	true	no
2	overcast	hot	high	false	yes
3	rainy	mild	high	false	yes
4	rainy	cool	normal	false	yes
5	rainy	cool	normal	true	no
6	overcast	cool	normal	true	yes
7	sunny	mild	high	false	no
8	sunny	cool	normal	false	yes
9	rainy	mild	normal	false	yes
10	sunny	mild	normal	true	yes
11	overcast	mild	high	true	yes
12	overcast	hot	normal	false	yes
13	rainy	mild	high	true	no

```python
# Create an empty data frame:
golf_df = pd.DataFrame()

# Add outlook column:
golf_df['Outlook'] = ['sunny','sunny','overcast','rainy','rainy',
                      'rainy', 'overcast', 'sunny','sunny','rainy',
                      'sunny','overcast','overcast','rainy']

# Add Temperature column:
golf_df['Temperature'] = ['hot','hot','hot','mild','cool','cool',
              'cool','mild','cool','mild','mild','mild','hot','mild']

# Add Humidity column:
golf_df['Humidity'] = ['high','high','high','high','normal','normal',
    'normal','high','normal','normal','normal','high','normal','high']

# Add Windy column:
golf_df['Windy'] = ['false','true','false','false','false','true',
         'true','false','false','false','true','true','false','true']

# Finally Add Play column:
golf_df['Play'] = ['no','no','yes','yes','yes','no','yes',
                   'no','yes','yes','yes','yes','yes','no']

#Print/show the new data:
print(golf_df)

y = golf_df['Play']
```

GJU
الجامعة الألمانية الأردنية
German Jordanian University

Create or Read the Data file (Dataset)

- OR you can read the dataset from excel file:

```
golf_df = pd.read_excel('weather.xlsx', 0)
print(golf_df)

data=golf_df.iloc[:,0:4]
print(data)

y = golf_df['Play']
```

	A	B	C	D	E
1	Outlook	Temperature	Humidity	Windy	Play
2	sunny	hot	high	FALSE	no
3	sunny	hot	high	TRUE	no
4	overcast	hot	high	FALSE	yes
5	rainy	mild	high	FALSE	yes
6	rainy	cool	normal	FALSE	yes
7	rainy	cool	normal	TRUE	no
8	overcast	cool	normal	TRUE	yes
9	sunny	mild	high	FALSE	no
10	sunny	cool	normal	FALSE	yes
11	rainy	mild	normal	FALSE	yes
12	sunny	mild	normal	TRUE	yes
13	overcast	mild	high	TRUE	yes
14	overcast	hot	normal	FALSE	yes
15	rainy	mild	high	TRUE	no
16					

data | numeric

GJU
الجامعة الألمانية الأردنية
German Jordanian University

Convert string values to numbers

- **Original weather.xlsx dataset:**

	A	B	C	D	E
1	Outlook	Temperature	Humidity	Windy	Play
2	sunny	hot	high	FALSE	no
3	sunny	hot	high	TRUE	no
4	overcast	hot	high	FALSE	yes
5	rainy	mild	high	FALSE	yes
6	rainy	cool	normal	FALSE	yes
7	rainy	cool	normal	TRUE	no
8	overcast	cool	normal	TRUE	yes
9	sunny	mild	high	FALSE	no
10	sunny	cool	normal	FALSE	yes
11	rainy	mild	normal	FALSE	yes
12	sunny	mild	normal	TRUE	yes
13	overcast	mild	high	TRUE	yes
14	overcast	hot	normal	FALSE	yes
15	rainy	mild	high	TRUE	no
16					

data numeric

- **weather.xlsx Converted to numbers:**

Outlook	Temperature	Humidity	Windy	Play
2	1	0	0	0
2	1	0	1	0
0	1	0	0	1
1	2	0	0	1
1	0	1	0	1
1	0	1	1	0
0	0	1	1	1
2	2	0	0	0
2	0	1	0	1
1	2	1	0	1
2	2	1	1	1
0	2	0	1	1
0	1	1	0	1
1	2	0	1	0

Label Encoding

Label Encoding is a technique that is used to convert categorical columns into numerical ones so that they can be fitted by machine learning models which only take numerical data.

```
from sklearn.preprocessing import LabelEncoder
import pandas as pd
golf_df=pd.read_excel('Wheather.xlsx')

le=LabelEncoder()
Outlook=le.fit_transform(golf_df.Outlook)
Temperature=le.fit_transform(golf_df.Temperature)
Humidity=le.fit_transform(golf_df.Humidity)
Windy=le.fit_transform(golf_df.Windy)
Play=le.fit_transform(golf_df.Play)

df=pd.DataFrame()
df["Outlook"]=Outlook
df["Temperature"]=Temperature
df["Humidity"]=Humidity
df["Windy"]=Windy
df["Play"]=Play

df.to_excel('WheatherTF.xlsx',index=False)
```

Split Train-Test

- The **train-test split** procedure is used to estimate the performance of machine learning algorithms when they are used to make predictions on data not used to train the model.
- **train_test_split** is a function in Sklearn model selection for splitting data arrays into two subsets: for training data and for testing data. With this function, you don't need to divide the dataset manually.

Is the object that controls randomization during splitting. It can be either an int or an instance of RandomState.

```
X_train, X_test, y_train, y_test =
train_test_split(data, y, test_size=0.3,
random_state=0)
```

This parameter specifies the size of the testing dataset.

The procedure involves taking a dataset and dividing it into two subsets. The first subset is used to fit the model and is referred to as the training dataset. The second subset is not used to train the model; instead, the input element of the dataset is provided to the model, then predictions are made and compared to the expected values. This second dataset is referred to as the test dataset.

Train Dataset: Used to fit the machine learning model.

Test Dataset: Used to evaluate the fit machine learning model.

Learning and predicting

Fitting your model to (i.e. using the .fit() method on) the training data is essentially the training part of the modeling process. The default value is None.

Then, for a classifier, you can classify incoming data points (from a test set, or otherwise) using the **predict** method.

```
rf = RandomForestClassifier()
X_train, X_test, y_train, y_test = train_test_split(data, y, test_size=0.3, 
random_state=0)

rf.fit(X_train, y_train);# train themodel
y_pred = rf.predict(X_test)
```