Minia University
Faculty of Computers & information

Artificial Neural Networks and Deep Learning

Slides By:

T.A. Sarah Osama Talaat

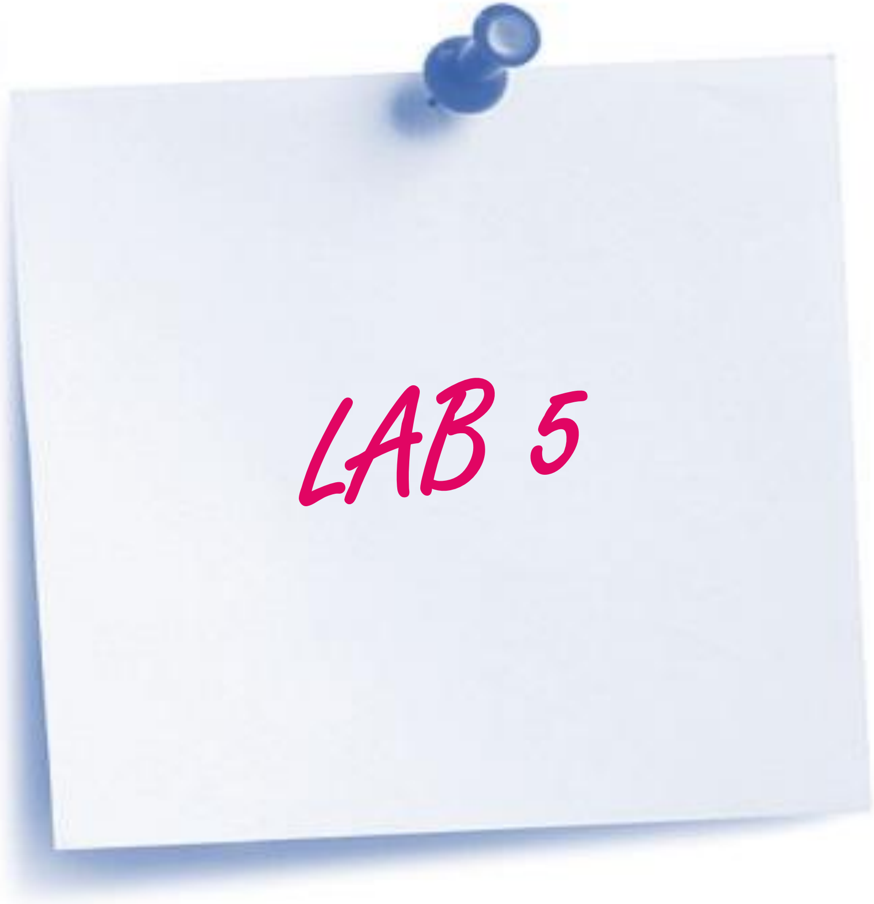
E-mail: SarahOsama.fci@gmail.com

Slides were prepared based on set of references mentioned in the last slide

Agenda

- Python Libraries
- Data Preprocessing.
  - Data cleaning
    - Dealing with missing data.
    - Dealing with categorical data.
  - Feature scaling
    - Rescaling
    - Mean normalization
    - Standardization
    - Scaling to unit length
  - Dimensionality reduction
    - Feature selection
    - Feature extraction
  - Partitioning a dataset in training and testing sets

Let’s Start



Python Libraries

Library	Uses
numpy	**numpy** is a library that contains a mathematical tools. Basically, we need this library to include any type of **mathematical operations.**
pandas	**pandas** considers as one of the best libraries that **is used to import and manage the datasets.**
matplotlib.pyplot	**matplotlib.pyplot** is a sub library which is actually used to plot a set of charts
xlsxwriter	**xlsxwriter** is a library that is used to write in Excel files.

Python Libraries

sklearn.preprocessing	This a package/library provides several common utility functions and transformer classes to change **raw feature** vectors into a representation that is more suitable for the downstream estimators.
Uses	▪ Standardization, or mean removal and variance scaling ▪ Non-linear transformation ▪ **Normalization** ▪ Binarization ▪ **Encoding categorical features** ▪ **Imputation of missing values** ▪ Generating polynomial features ▪ Custom transformers

Python Libraries

Class	**Imputer**	
Uses	The **imputer** class provides basic strategies for imputing **missing values**, either using the **mean, the median or the most frequent value of the row/column.**	
Instruction	**from** sklearn.preprocessing **import** Imputer	
Some of the Imputer parameters	▪ missing_values ▪ strategy ▪ axis	
The possible values for each parameter	**Parameter**	**Values**
	missing_values	integer or "NaN", optional (default="NaN")
	strategy	string, mean, median or most_frequent, optional (default="mean")
	axis	integer, optional (default=0) The axis along which to impute. If axis=0, then impute along columns and if axis=1, then impute along rows.

Data Preprocessing

Why we need to the data preprocessing process?

- The main objectives of data preprocessing are to **manipulate** and **transform** raw data into **cleaned** and **scaled format**.
- In addition it is important to **compress** the data onto a **smaller dimensional** subspace while retaining most of the **relevant** information.

Data Preprocessing

Types of data preprocessing?

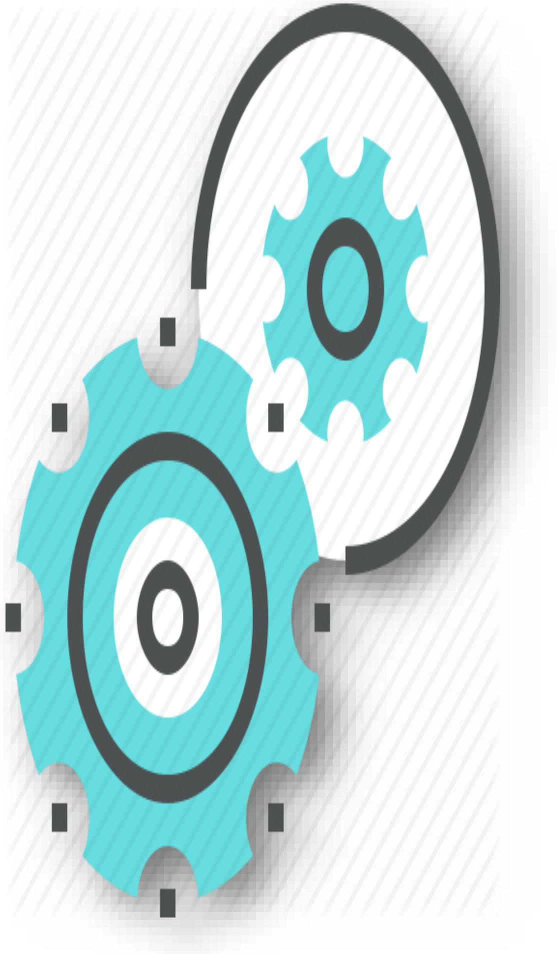
Data preprocessing

- **Data cleaning**
  - Missing data
  - Categorical data
- **Feature scaling**
  - Rescaling
  - Mean normalization
  - Standardization
  - Scaling to unit length
- **Dimensionality Reduction**
  - Features selection
  - Features extraction
- **Partitioning a dataset to training and testing sets**

Data Preprocessing

Data cleaning: missing data

- In real-world applications are **familiar** that the collected data samples contain one or more missing values for various reasons. These reasons include:
  - There could have been an error in the data collection process,
  - certain measurements are not applicable and
  - particular fields could have been simply left blank in a survey, for instance. We typically see ***missing values*** *as the* ***blank spaces*** *in our data table or as placeholder strings such as* ***NaN*** (Not A Number).



Data Preprocessing
Data cleaning: missing data

Solutions to avoid the problem of missing data

- Eliminate/Remove
- Mean
- Median
- Most frequent value

Data Preprocessing

Data cleaning: missing data

- **There are several available solutions to avoid the problem of missing data :**
  - **Eliminate/Remove:** you can simply eliminate/remove the corresponding features (columns) or samples (rows) that contain missing data from the used dataset (BUT it’s quite dangers, because we might lose too much valuable data).
  - **Mean/Median/Most frequent value:** you can take the mean/median/most frequent value from the column/row that contains missing data.
    - **Mean** is used to replace the missing values using the mean/average along the axis (i.e. column or row).
    - **Median** is used to replace missing values using the median along the axis.
    - **Most frequent value** is used to replace missing values using the most frequent value along the axis.

Data Preprocessing

Back to data cleaning: missing data

❑Example(2.1):

- In this dataset, we have 5 columns which includes 4 features (independent variables) and one response (dependent variable).

Predictor variables

Response variable

	A	B	C	D	E
1	Faculty	Name	Age	year	Score
2	FCI	Sarah	21	4	1.3
3	Medicine	Ahmed	23	6	2.5
4	Science	Mohamma	19	2	4
5	Alsun	Moataz		4	3.7
6	Alsun	Hagar	19	2	4
7	Engineerin	Emy	19	3	3
8	Medicine	Yomna		2	2.5
9	Medicine	Faisal	22	6	2.75
10	Pharmacy	Khaled		5	4.2
11	Pharmacy	Aly	18	2	3
12	Medicine	Osama	17	1	4.5

Data Preprocessing

Back to data cleaning: missing data

❑Example(2.1):

- In this dataset, we have 5 columns which includes 4 features (independent variables) and one response (dependent variable).

	A	B	C	D	E
1	Faculty	Name	Age	year	Score
2	FCI	Sarah	21	4	1.3
3	Medicine	Ahmed	23	6	2.5
4	Science	Mohamma	19	2	4
5	Alsun	Moataz		4	3.7
6	Alsun	Hagar	19	2	4
7	Engineerin	Emy	19	3	3
8	Medicine	Yomna		2	2.5
9	Medicine	Faisal	22	6	2.75
10	Pharmacy	Khaled		5	4.2
11	Pharmacy	Aly	18	2	3
12	Medicine	Osama	17	1	4.5

Data Preprocessing

Back to data cleaning: missing data

❑Example(2.1):

- In this dataset, we have 5 columns which includes 4 features (independent variables) and one response (dependent variable).

	A	B	C	D	E
1	Faculty	Name	Age	year	Score
2	FCI	Sarah	21	4	1.3
3	Medicine	Ahmed	23	6	2.5
4	Science	Mohamma	19	2	4
5	Alsun	Moataz		4	3.7
6	Alsun	Hagar	19	2	4
7	Engineerin	Emy	19	3	3
8	Medicine	Yomna		2	2.5
9	Medicine	Faisal	22	6	2.75
10	Pharmacy	Khaled		5	4.2
11	Pharmacy	Aly	18	2	3
12	Medicine	Osama	17	1	4.5

Solution by using python:

- From "**scikit-learn**" package we use "**Imputer**" class for data missing treatment. It is included in "**preprocessing**" module.

Data Preprocessing

Back to data cleaning: missing data

❑Example(2.1):

- We can solve the missing data problem in Python by using the following instructions;

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Importing the dataset
dataset = pd.read_csv('Dataset_1.csv')

# Create a matrix to store the predictor (independent) variables
X = dataset.iloc[:, :-1].values

# Create a victore to store the response (dependent) variable
y = dataset.iloc[:, 3].values
# Dealing with missing data by the eliminating method
# Remove the rows that have missing cells
dataset.dropna()
# Remove the columns that have missing cells
dataset.dropna(axis=1)
# Dealing with missing data by using mean, median or most frequent

```

Data Preprocessing

Back to data cleaning: missing data

❑Example(2.1):

- We can solve the missing data problem in Python by using the following instructions;

```
# Data Preprocessing
# Importing the libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Importing the dataset
dataset = pd.read_csv('Dataset_1.csv')
# Create a matrix to store the predictor (independent) variables
X = dataset.iloc[:, :-1].values
# Create a victore to store the response (dependent) variable
y = dataset.iloc[:, 3].values
# Dealing with missing data by using mean, median or most frequent
from sklearn.preprocessing import Imputer

# strategy can be "mean", "median"  or "most_frequent"
#If "mean", then replace missing values using the mean along the axis.
#If “median”, then replace missing values using the median along the axis.
#If “most_frequent”, then replace missing using the most frequent value along

imputer = Imputer(missing_values = 'NaN', strategy = 'mean', axis = 0)
imputer = imputer.fit(X[:, 2:3])
X[:, 2:3] = imputer.transform(X[:, 2:3])
```

Data Preprocessing

Back to data cleaning: missing data

❑Example(2.1):

- We can solve the missing data problem in Python by using the following instructions;

```python
# Data Preprocessing
# Importing the libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Importing the dataset
dataset = pd.read_csv('Dataset_1.csv')
# Create a matrix to store the predictor (independent) variables
X = dataset.iloc[:, :-1].values
# Create a victore to store the response (dependent) variable
y = dataset.iloc[:, 3].values
# Dealing with missing data by using mean, median or most frequent
from sklearn.preprocessing import Imputer

# strategy can be "mean", "median"  or "most_frequent"
#If "mean", then replace missing values using the mean along the axis.
#If “median”, then replace missing values using the median along the axis.
#If “most_frequent”, then replace missing using the most frequent value along

imputer = Imputer(missing_values = 'NaN', strategy = 'mean', axis = 0)
imputer = imputer.fit(X[:, 2:3])
X[:, 2:3] = imputer.transform(X[:, 2:3])
```

	A	B	C	D	E
1	Faculty	Name	Age	year	Score
2	FCI	Sarah	21	4	1.3
3	Medicine	Ahmed	23	6	2.5
4	Science	Mohammad	19	2	4
5	Alsun	Moataz	19.75	4	3.7
6	Alsun	Hagar	19	2	4
7	Engineering	Emy	19	3	3
8	Medicine	Yomna	19.75	2	2.5
9	Medicine	Faisal	22	6	2.75
10	Pharmacy	Khaled	19.75	5	4.2
11	Pharmacy	Aly	18	2	3
12	Medicine	Osama	17	1	4.5

Data Preprocessing

Data cleaning: categorical data

❑Example(2.2):

- In this dataset, we have three independent variables and one dependent variable.
- In this dataset all input variables consider as categorical features.

Nominal Features

Ordinal feature

	A	B	C	D
1	Country	Color	Size	Prise
2	Egypt	red	S	150.75
3	USA	black	M	200
4	KSA	blue	L	170
5	Canda	white	S	300
6	Egypt	green	M	120
7	China	green	Xl	200.5
8	Germany	blue	2XL	499.99

Data Preprocessing

Data cleaning: categorical data

❑Example(2.2):

- At the first, we deal with the ordinal feature (feature number 3 at column 2).

```
# Data Preprocessing

# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
dataset = pd.read_csv('Dataset_2.csv')
X = dataset.iloc[:, :-1].values
Y = dataset.iloc[:, 3].values

# Dealing with categorical data
# Mapping ordinal feature/independent variable

# Define a new dictionary to store the sizes and its values
size_mapping = {'S':1,'M':2,'L':3,'XL':4,'2XL':5}
dataset['Size'] = dataset['Size'].map(size_mapping)
X[:,2] = dataset.iloc[:,2].values
```

❑Example(2.2):

- At the first, we deal with the ordinal feature (feature number 3 at column 2).

```python
# Data Preprocessing

# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
dataset = pd.read_csv('Dataset_2.csv')
X = dataset.iloc[:, :-1].values
Y = dataset.iloc[:, 3].values

# Dealing with categorical data
# Mapping ordinal feature/independent variable

# Define a new dictionary to store the sizes and its values
size_mapping = {'S':1,'M':2,'L':3,'XL':4,'2XL':5}
dataset['Size'] = dataset['Size'].map(size_mapping)
X[:,2] = dataset.iloc[:,2].values
```

	Country	Color	Size	Prise
0	Egypt	red	1	150.75
1	USA	black	2	200.00
2	KSA	blue	3	170.00
3	Canda	white	1	300.00
4	Egypt	green	2	120.00
5	China	green	4	200.50
6	Germany	blue	5	499.99

Data Preprocessing

Data cleaning: categorical data

❑Example(2.2):

- If we want to transform the integer values back to the original string representation at a later stage, we can simply define a reverse-mapping dictionary.

```
# Data Preprocessing

# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
# Importing the dataset
dataset = pd.read_csv('Dataset_2.csv')
X = dataset.iloc[:, :-1].values
Y = dataset.iloc[:, 3].values
# Dealing with categorical data
# Mapping ordinal feature/independent variable
# Define a new dictionary to store the sizes and its values
size_mapping = {'S':1,'M':2,'L':3,'XL':4,'2XL':5}
dataset['Size'] = dataset['Size'].map(size_mapping)
X[:,2] = dataset.iloc[:,2].values
# If we want to transform the integer values back to the original
# string representation at a later stage,  we can simply define
#a reverse-mapping dictionary
int_size_mapping = {v: k for k, v in size_mapping.items()}
dataset['Size'] = dataset['Size'].map(int_size_mapping)
X[:,2] = dataset.iloc[:,2].values
```

Data Preprocessing

Data cleaning: categorical data

❑Example(2.2):

- If we want to transform the integer values back to the original string representation at a later stage, we can simply define a reverse-mapping dictionary.

```python
# Data Preprocessing

# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
# Importing the dataset
dataset = pd.read_csv('Dataset_2.csv')
X = dataset.iloc[:, :-1].values
Y = dataset.iloc[:, 3].values
# Dealing with categorical data
# Mapping ordinal feature/independent variable
# Define a new dictionary to store the sizes and its values
size_mapping = {'S':1,'M':2,'L':3,'XL':4,'2XL':5}
dataset['Size'] = dataset['Size'].map(size_mapping)
X[:,2] = dataset.iloc[:,2].values
# If we want to transform the integer values back to the original
# string representation at a later stage,  we can simply define
#a reverse-mapping dictionary
int_size_mapping = {v: k for k, v in size_mapping.items()}
dataset['Size'] = dataset['Size'].map(int_size_mapping)
X[:,2] = dataset.iloc[:,2].values
```

	Country	Color	Size	Prise
0	Egypt	red	S	150.75
1	USA	black	M	200.00
2	KSA	blue	L	170.00
3	Canda	white	S	300.00
4	Egypt	green	M	120.00
5	China	green	XL	200.50
6	Germany	blue	2XL	499.99

Data Preprocessing

Data cleaning: categorical data

❑Example(2.2):

- Now, we deal with the nominal features (feature number 1 and 2 at column 0 and 1).

```
# Data Preprocessing

# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
# Importing the dataset
dataset = pd.read_csv('Dataset_2.csv')
X = dataset.iloc[:, :-1].values
Y = dataset.iloc[:, 3].values
# Dealing with categorical data

# Encoding nominal features/independent variable
from sklearn.preprocessing import LabelEncoder
Country_labelencoder = LabelEncoder()
X[:, 0] = Country_labelencoder.fit_transform(X[:, 0])
Color_labelencoder = LabelEncoder()
X[:, 1] = Color_labelencoder.fit_transform(X[:, 1])
```

Data Preprocessing

Data cleaning: categorical data

❑Example(2.2):

- Now, we deal with the nominal features (feature number 1 and 2 at column 0 and 1).

In [38]:
```python
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Dealing with categorical data
# Encoding nominal features/independent variable
from sklearn.preprocessing import LabelEncoder
Country_labelencoder = LabelEncoder()
X[:, 0] = Country_labelencoder.fit_transform(X[:, 0])
Color_labelencoder = LabelEncoder()
X[:, 1] = Color_labelencoder.fit_transform(X[:, 1])

X
```

Out[38]:
```
array([[2, 3, 1],
       [5, 0, 2],
       [4, 1, 3],
       [0, 4, 1],
       [2, 2, 2],
       [1, 2, 4],
       [3, 1, 5]], dtype=object)
```

Data Preprocessing

Data cleaning: categorical data

- **Example(2.2):**

```python
# Data Preprocessing

# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
# Importing the dataset
dataset = pd.read_csv('Dataset_2.csv')
X = dataset.iloc[:, :-1].values
Y = dataset.iloc[:, 3].values
# Dealing with categorical data
# Mapping ordinal feature/independent variable
# Define a new dictionary to store the sizes and its values
size_mapping = {'S':1,'M':2,'L':3,'XL':4,'2XL':5}
dataset['Size'] = dataset['Size'].map(size_mapping)
X[:,2] = dataset.iloc[:,2].values

# Encoding nominal features/independent variable
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
Country_labelencoder = LabelEncoder()
X[:, 0] = Country_labelencoder.fit_transform(X[:, 0])
Color_labelencoder = LabelEncoder()
X[:, 1] = Color_labelencoder.fit_transform(X[:, 1])

Country_color_onehotencoder = OneHotEncoder(categorical_features = [0,1])
X = Country_color_onehotencoder.fit_transform(X).toarray()
```

Data Preprocessing

Data cleaning: categorical data

❑Example(2.2):

In [78]:
```
# Data Preprocessing
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
# Importing the dataset
dataset = pd.read_csv('Dataset_2.csv')
X = dataset.iloc[:, :-1].values
Y = dataset.iloc[:, 3].values
# Dealing with categorical data
# Mapping ordinal feature/independent variable
# Define a new dictionary to store the sizes and its values
size_mapping = {'S':1,'M':2,'L':3,'XL':4,'2XL':5}
dataset['Size'] = dataset['Size'].map(size_mapping)
X[:,2] = dataset.iloc[:,2].values
# Encoding nominal features/independent variable
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
Country_labelencoder = LabelEncoder()
X[:, 0] = Country_labelencoder.fit_transform(X[:, 0])
Color_labelencoder = LabelEncoder()
X[:, 1] = Color_labelencoder.fit_transform(X[:, 1])

Country_color_onehotencoder = OneHotEncoder(categorical_features = [0,1])
X = Country_color_onehotencoder.fit_transform(X).toarray()
X
```

Out[78]:
```
array([[ 0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,  0.,  1.],
       [ 0.,  0.,  0.,  0.,  0.,  1.,  1.,  0.,  0.,  0.,  0.,  2.],
       [ 0.,  0.,  0.,  0.,  1.,  0.,  0.,  1.,  0.,  0.,  0.,  3.],
       [ 1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,  1.],
       [ 0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,  2.],
       [ 0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,  4.],
       [ 0.,  0.,  0.,  1.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  5.]])
```

Data Preprocessing

Data cleaning: categorical data

❑Example(2.2):

```
In [78]: # Data Preprocessing
         # Importing the libraries
         import numpy as np
         import matplotlib.pyplot as plt
         import pandas as pd
         # Importing the dataset
```

Country feature after dummy encoding

Color feature after dummy encoding

```
Out[78]: array([[ 0., 0., 1., 0., 0., 0., 0., 0., 0., 1., 0., 1.],
        [ 0., 0., 0., 0., 0., 1., 1., 0., 0., 0., 0., 2.],
        [ 0., 0., 0., 0., 1., 0., 0., 1., 0., 0., 0., 3.],
        [ 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 1.],
        [ 0., 0., 1., 0., 0., 0., 0., 0., 1., 0., 0., 2.],
        [ 0., 1., 0., 0., 0., 0., 0., 0., 1., 0., 0., 4.],
        [ 0., 0., 0., 1., 0., 0., 0., 1., 0., 0., 0., 5.]])
```


```
[ 0., 0., 1., 0., 0., 0., 0., 0., 1., 0., 0., 2.],
[ 0., 1., 0., 0., 0., 0., 0., 0., 1., 0., 0., 4.],
[ 0., 0., 0., 1., 0., 0., 0., 1., 0., 0., 0., 5.]])
```

Data Preprocessing

Feature scaling: rescaling

❑Example(2.3):

- In the following dataset, we have three independent variables and one dependent variable.

	A	B	C	D
1	High temp	Solar radiation	Wind speed	Rain
2	13	192	35	18.00
3	39	178	56	12.00
4	9	199	74	23.00
5	14	100	57	12.00
6	1	187	54	25.00
7	-4	139	57	14.00
8	46	156	58	13.00
9	6	195	51	24.00
10	-4	166	38	10.00
11	11	163	68	12.00
12	29	178	42	22.00

Data Preprocessing

Feature scaling: rescaling

❑Example(2.3):

```python
# Data Preprocessing
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
# Importing the dataset
dataset = pd.read_csv('Dataset_3.csv')
X = dataset.iloc[:, :-1].values
Y = dataset.iloc[:, 3].values
# Feature scaling: rescaling
#The min-max scaling procedure is implemented in scikit-learn and can be used
# as follows:
from sklearn.preprocessing import MinMaxScaler
# Here we scale all input features
minMaxScaled_rescaling = MinMaxScaler()
X_scaled = minMaxScaled_rescaling .fit_transform(X)
```

Data Preprocessing
Feature scaling: rescaling

❑Example(2.3):

```
C:\Users\sarah\Anaconda3\lib\site-packages\sklearn\utils\validation.py:475: DataConversionWarning: Data with input dt
ype int64 was converted to float64 by MinMaxScaler.
  warnings.warn(msg, DataConversionWarning)
```

```
array([[ 0.34      ,  0.92929293,  0.        ],
       [ 0.86      ,  0.78787879,  0.53846154],
       [ 0.26      ,  1.        ,  1.        ],
       [ 0.36      ,  0.        ,  0.56410256],
       [ 0.1       ,  0.87878788,  0.48717949],
       [ 0.        ,  0.39393939,  0.56410256],
       [ 1.        ,  0.56565657,  0.58974359],
       [ 0.2       ,  0.95959596,  0.41025641],
       [ 0.        ,  0.66666667,  0.07692308],
       [ 0.3       ,  0.63636364,  0.84615385],
       [ 0.66      ,  0.78787879,  0.17948718]])
```


```
       [ 0.        ,  0.39393939,  0.56410256],
       [ 1.        ,  0.56565657,  0.58974359],
       [ 0.2       ,  0.95959596,  0.41025641],
       [ 0.        ,  0.66666667,  0.07692308],
       [ 0.3       ,  0.63636364,  0.84615385],
       [ 0.66      ,  0.78787879,  0.17948718]])
```

Data Preprocessing

Feature scaling: mean normalization

❑Example(2.3):

```python
# Data Preprocessing
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
# Importing the dataset
dataset = pd.read_csv('Dataset_3.csv')
X = dataset.iloc[:, :-1].values
Y = dataset.iloc[:, 3].values

# Feature scaling mean normalization
from sklearn.preprocessing import Normalizer
# Here we scale all input features
normailzation = Normalizer()
X_scaled = normailzation.fit_transform(X)
```

Data Preprocessing

Feature scaling: mean normalization

❑Example(2.3):

```
In [8]: runfile('C:/Users/sarah/OneDrive/Machine Learning and Pattern Recognation/Part
1 - Data Preprocessing/Feature_scaling_rescaling.py', wdir='C:/Users/sarah/OneDrive/
Machine Learning and Pattern Recognation/Part 1 - Data Preprocessing')
C:\Users\sarah\Anaconda3\lib\site-packages\sklearn\utils\validation.py:475:
DataConversionWarning: Data with input dtype int64 was converted to float64 by
MinMaxScaler.
  warnings.warn(msg, DataConversionWarning)

In [9]: X_scaled
Out[9]:
array([[ 0.06646335,  0.98161254,  0.17893979],
       [ 0.20458141,  0.93373054,  0.29375792],
       [ 0.04235212,  0.93645244,  0.34822855],
       [ 0.12073902,  0.86242154,  0.49158028],
       [ 0.0051376 ,  0.96073194,  0.27743061],
       [-0.02661585,  0.92490076,  0.37927585],
       [ 0.26639934,  0.90344123,  0.33589482],
       [ 0.02975479,  0.96703066,  0.25291571],
       [-0.02348233,  0.97451677,  0.22308215],
       [ 0.06216178,  0.92112449,  0.38427279],
       [ 0.1566104 ,  0.96126381,  0.22681506]])
```

Data Preprocessing

Feature scaling: standardization

❑Example(2.3):

```
# Data Preprocessing
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
# Importing the dataset
dataset = pd.read_csv('Dataset_3.csv')
X = dataset.iloc[:, :-1].values
Y = dataset.iloc[:, 3].values

# Feature scaling standardization
from sklearn.preprocessing import StandardScaler
# Here we scale all input features
standardization = StandardScaler()
X_scaled = standardization.fit_transform(X)
```

Data Preprocessing

Feature scaling: standardization

❑Example(2.3):

```
In [24]: runfile('C:/Users/sarah/OneDrive/Machine Learning and Pattern
Recognation/Part 1 - Data Preprocessing/Feature_scaling_rescaling.py',
wdir='C:/Users/sarah/OneDrive/Machine Learning and Pattern Recognation/
Part 1 - Data Preprocessing')
C:\Users\sarah\Anaconda3\lib\site-packages\sklearn\utils\validation.py:
475: DataConversionWarning: Data with input dtype int64 was converted
to float64 by MinMaxScaler.
  warnings.warn(msg, DataConversionWarning)

In [25]: X_scaled
Out[25]:
array([[ 0.06646335,  0.98161254,  0.17893979],
       [ 0.20458141,  0.93373054,  0.29375792],
       [ 0.04235212,  0.93645244,  0.34822855],
       [ 0.12073902,  0.86242154,  0.49158028],
       [ 0.0051376 ,  0.96073194,  0.27743061],
       [-0.02661585,  0.92490076,  0.37927585],
       [ 0.26639934,  0.90344123,  0.33589482],
       [ 0.02975479,  0.96703066,  0.25291571],
       [-0.02348233,  0.97451677,  0.22308215],
       [ 0.06216178,  0.92112449,  0.38427279],
       [ 0.1566104 ,  0.96126381,  0.22681506]])
```

❑Dimensionality Reduction:

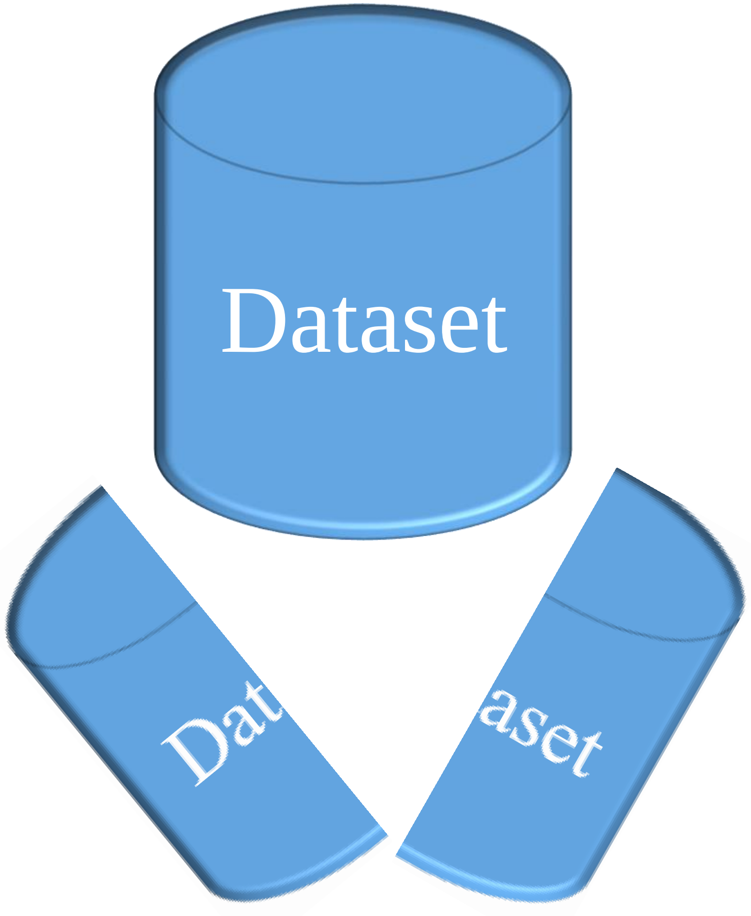
- In machine learning and statistics, dimensionality reduction or dimension reduction is the process of **reducing** the number of **random variables** under consideration.
- There are two main categories of dimensionality reduction techniques: **feature selection** and **feature extraction.**
  - **Feature Selection:** we **select** a subset of the original feature set.
  - **Feature Extraction:** we **derive** information from the feature set to construct a new feature subspace

Data Preprocessing

Partitioning a dataset in training and testing sets

❑Partitioning a dataset in training and testing sets:

- Partitioning is used to determine whether our machine learning algorithm **not only performs well** on the training set but also **generalizes** well to new data,
- We want to **randomly divide** the dataset into a separate **training and testing sets**.
- We use the **training** set to **train** and **optimize** our machine learning model, while we keep the **testing** set until the very end to **evaluate** the final model.



Data Preprocessing

Partitioning a dataset in training and testing sets

❑Partitioning methods:

- **Some of partitioning methods**
  - **Random splitting**
  - **Cross validation**
    - **K-Fold**
    - **Leave one out**

Data Preprocessing

Partitioning a dataset in training and testing sets

❑Random splitting

- Split arrays or matrices into random train and test subsets
- A convenient way to **randomly** partition the dataset into a separate **testing and training** dataset is to use the **“train_test_split”** function from **scikit-learn's “cross_validation”** submodule.
- Another way is to use the to use the **“train_test_split”** function from **scikit-learn's “model_selection”** submodule

Data Preprocessing

Partitioning a dataset in training and testing sets

❑Example(2.3):

- Back to the dataset that is used in example (2.3) to apply the data partitioning methods on it

```python
# Data Preprocessing

# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
dataset = pd.read_csv('Dataset_3.csv')
X = dataset.iloc[:, :-1].values
Y = dataset.iloc[:, 3].values

# Feature scaling: standardization
from sklearn.preprocessing import StandardScaler
# Here we scale all input features
standardization = StandardScaler()
X_scaled = standardization.fit_transform(X)

# Partitioning a dataset in training and testing sets
# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2, random_state = 0)
```

Data Preprocessing

Partitioning a dataset in training and testing sets

❑Example(2.3):

- Back to the dataset that is used in example (2.3) to apply the data partitioning methods on it

Name	Type	Size	Value
X	int64	(11, 3)	array([[13, 192, 35],
X_scaled	float64	(11, 3)	array([[-0.0970969 , 0.84888381, -1.65043736], [1.53641558
X_test	int64	(3, 3)	array([[1, 187, 54],
X_train	int64	(8, 3)	array([[29, 178, 42],
Y	float64	(11,)	array([18., 12., 23., ..., 10., 12., 22.])
dataset	DataFrame	(11, 4)	Column names: High temp, Solar radiation, Wind speed , Rain
y_test	float64	(3,)	array([25., 12., 23.])
y_train	float64	(8,)	array([22., 13., 12., 24., 10., 12., 18., 14.])

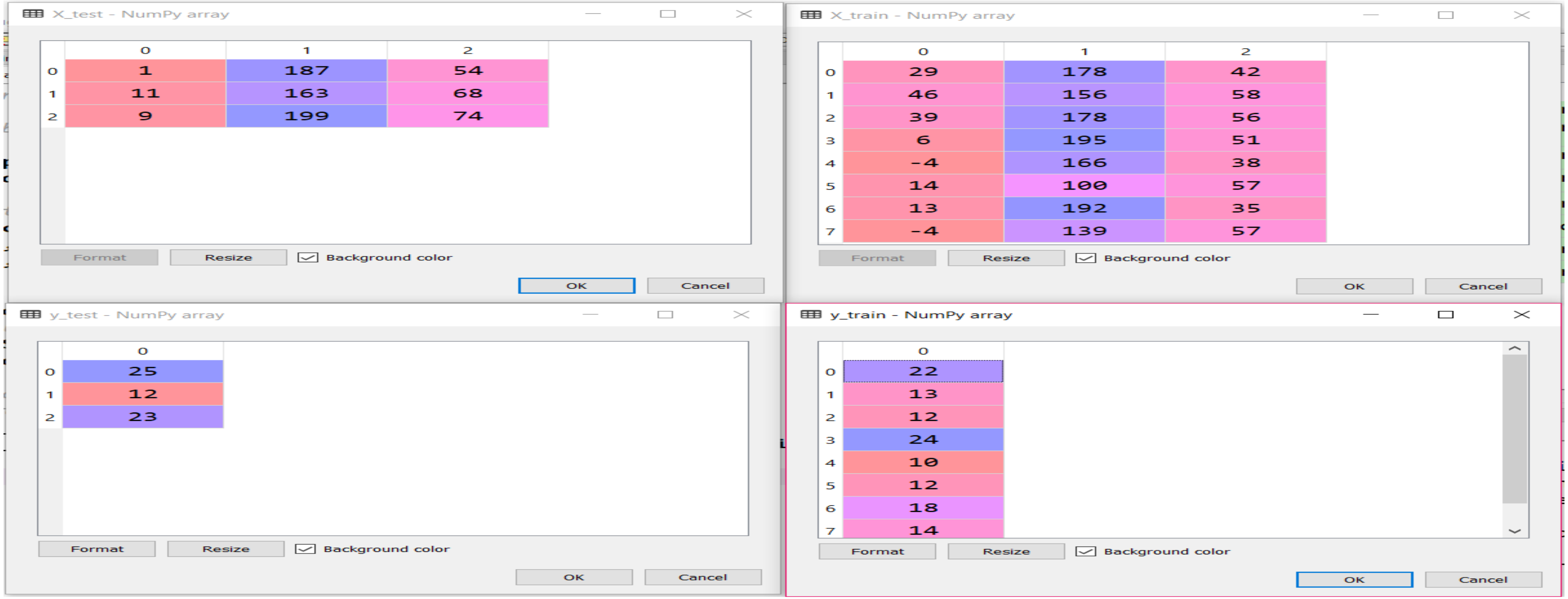
Profiler | Variable explorer | File explorer | Help

Data Preprocessing

Partitioning a dataset in training and testing sets

❑Example(2.3):

- Back to the dataset that is used in example (2.3) to apply the data partitioning methods on it



X_test - NumPy array

	0	1	2
0	1	187	54
1	11	163	68
2	9	199	74

X_train - NumPy array

	0	1	2
0	29	178	42
1	46	156	58
2	39	178	56
3	6	195	51
4	-4	166	38
5	14	100	57
6	13	192	35
7	-4	139	57

y_test - NumPy array

	0
0	25
1	12
2	23

y_train - NumPy array

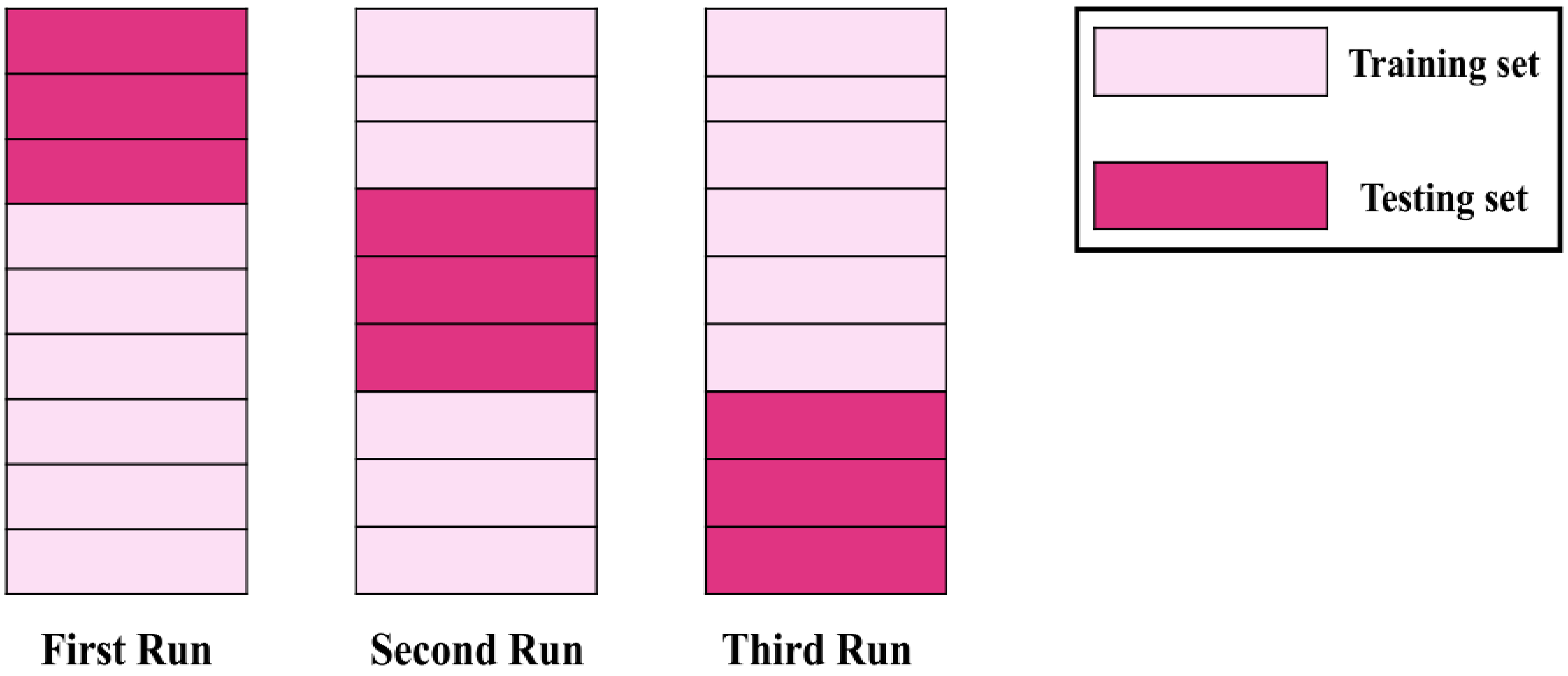
	0
0	22
1	13
2	12
3	24
4	10
5	12
6	18
7	14

Data Preprocessing

Partitioning a dataset in training and testing sets

❑K-Fold cross validation:

- The following figure shows the k-fold cross validation schema when k = 3.



Data Preprocessing

Partitioning a dataset in training and testing sets

❑Example(2.3):

- Back to the dataset that is used in example (2.3) to apply the data partitioning methods on it

```python
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
# Importing the dataset
dataset = pd.read_csv('Dataset_3.csv')
X = dataset.iloc[:, :-1].values
Y = dataset.iloc[:, 3].values
# Feature scaling: standardization
from sklearn.preprocessing import StandardScaler
standardization = StandardScaler()
X_scaled = standardization.fit_transform(X)
# Partitioning a dataset in training and testing sets
# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import KFold
# shuffle is used to whether to shuffle the data before splitting into batches.
kfold = KFold(n_splits=3, shuffle=False, random_state=None)
# Returns the number of splitting iterations in the cross-validator
k = kfold.get_n_splits(X) # or # k = kfold.get_n_splits([X,Y,3])
# Generate indices to split data into training and test set.
indices = kfold.split(X)
i =1
for train_index, test_index in kfold.split(X):
    print("The fold number:  ",i)
    i+=1
    print("TRAIN:", train_index, "TEST:", test_index)
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = Y[train_index], Y[test_index]
    print("X_train:", X[train_index], "\n Y_train:",Y[train_index])
    print("X_test:", X[test_index], "\n Y_test:",Y[test_index])
```

Data Preprocessing

Partitioning a dataset in training and testing sets

❑Leave one out(LOO) cross validation:

- LOO is a simple method of cross-validation.
- Each learning set is created by taking **all the** samples **except one**, the test set being the sample **left out**.
- Thus, for samples, we have different training sets and different tests set.
- This cross-validation procedure does not waste much data as only one sample is removed from the training set

Data Preprocessing

Partitioning a dataset in training and testing sets

❑Example(2.3):

- Back to the dataset that is used in example (2.3) to apply the data partitioning methods on it

```
# Data Preprocessing
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
# Importing the dataset
dataset = pd.read_csv('Dataset_3.csv')
X = dataset.iloc[:, :-1].values
Y = dataset.iloc[:, 3].values
# Feature scaling: standardization
from sklearn.preprocessing import StandardScaler
standardization = StandardScaler()
X_scaled = standardization.fit_transform(X)
# Partitioning a dataset in training and testing sets

from sklearn.model_selection import LeaveOneOut

LOO = LeaveOneOut()

for train, test in LOO.split(X):
    print("%s %s" % (train, test))
```

Data Preprocessing

Partitioning a dataset in training and testing sets

❑Example(2.3):

- Ba

```
In [4]: runfile('C:/Users/sarah/OneDrive/Machine Learning and Pattern Recognation/Part 1 - Data Preprocessing/Partitioning_dataset_LOO.py', wdir='C:/Users/sarah/OneDrive/Machine Learning and Pattern Recognation/Part 1 - Data Preprocessing')
[ 1  2  3  4  5  6  7  8  9 10] [0]
[ 0  2  3  4  5  6  7  8  9 10] [1]
[ 0  1  3  4  5  6  7  8  9 10] [2]
[ 0  1  2  4  5  6  7  8  9 10] [3]
[ 0  1  2  3  5  6  7  8  9 10] [4]
[ 0  1  2  3  4  6  7  8  9 10] [5]
[ 0  1  2  3  4  5  7  8  9 10] [6]
[ 0  1  2  3  4  5  6  8  9 10] [7]
[ 0  1  2  3  4  5  6  7  9 10] [8]
[ 0  1  2  3  4  5  6  7  8 10] [9]
[0 1 2 3 4 5 6 7 8 9] [10]
C:\Users\sarah\Anaconda3\lib\site-packages\sklearn\utils\validation.py:475: DataConversionWarning: Data with input dtype int64 was converted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
```

Testing sets

Training set

Any Questions!?

Thank you