Dataset Loading

Scikit Learn

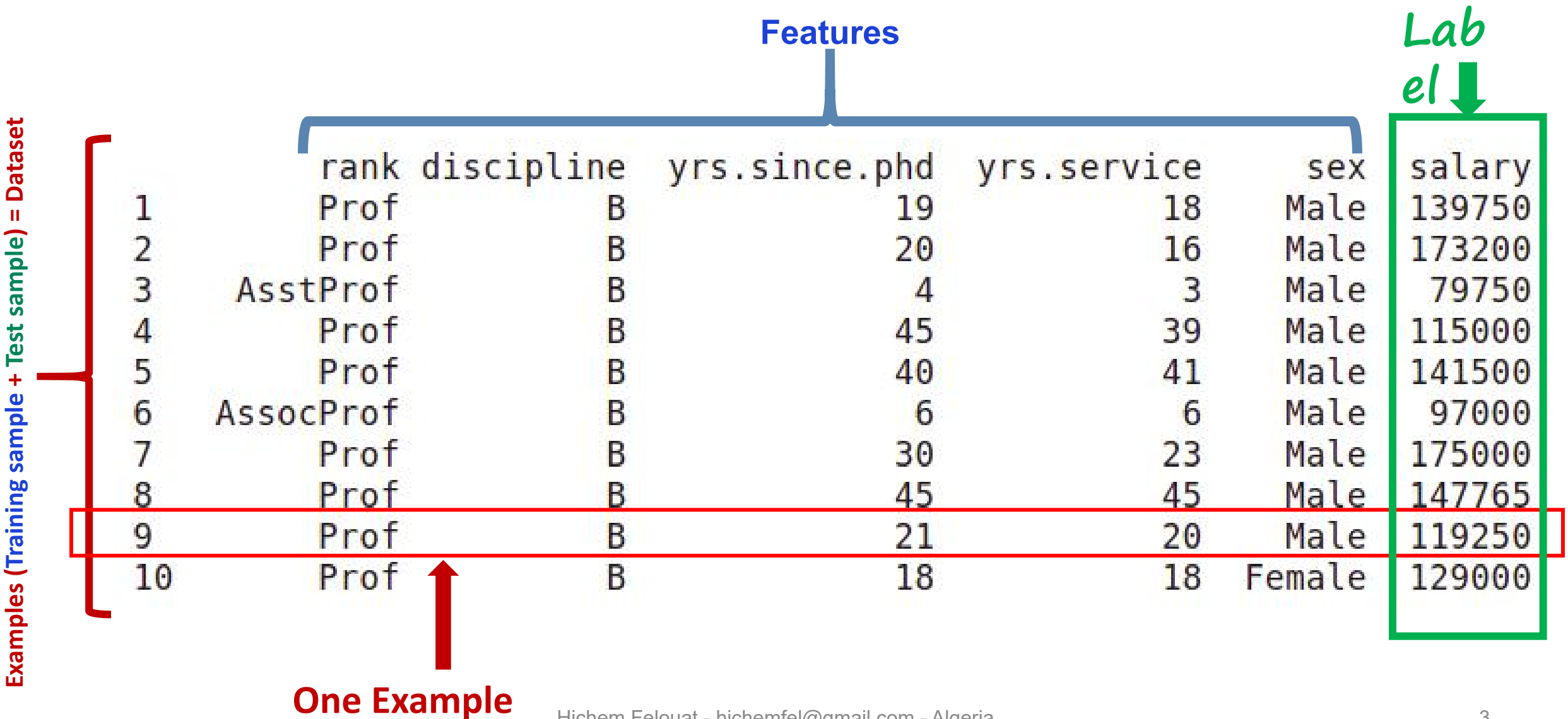
Hichem Felouat
hichemfel@gmail.com

https://www.linkedin.com/in/hichemfelouat

Steps to Build a Machine Learning System

1. Data collection.
2. Improving data quality (data preprocessing: drop duplicate rows, handle missing values and outliers).
3. Feature engineering (feature extraction and selection, dimensionality reduction).
4. Splitting data into training (and evaluation) and testing sets.
5. Algorithm selection (Regression, Classification, Clustering ...).
6. Training.
7. Evaluation + Hyperparameter tuning.
8. Testing.
9. Deployment

1. Dataset Loading



	rank	discipline	yrs.since.phd	yrs.service	sex	salary
1	Prof	B	19	18	Male	139750
2	Prof	B	20	16	Male	173200
3	AsstProf	B	4	3	Male	79750
4	Prof	B	45	39	Male	115000
5	Prof	B	40	41	Male	141500
6	AssocProf	B	6	6	Male	97000
7	Prof	B	30	23	Male	175000
8	Prof	B	45	45	Male	147765
9	Prof	B	21	20	Male	119250
10	Prof	B	18	18	Female	129000

1. Dataset Loading

Places to Find Free Datasets :

- Google Dataset Search
- Kaggle
- GitHub
- OpenML
- Data.Gov
- Datahub.io
- UCI Machine Learning Repository

1. Dataset Loading: Pandas

```
import pandas as pd

df = pd.DataFrame(
     {"a" : [4, 5, 6],
      "b" : [7, 8, 9],
      "c" : [10, 11, 12]},
     index = [1, 2, 3])

print(df)
```

	a	b	c
1	4	7	10
2	5	8	11
3	6	9	12

1. Dataset Loading: Pandas

Read data from file 'filename.csv'

```
import pandas as pd
data = pd.read_csv("filename.csv")
print (data)
```

Select only the feature_1 and feature_2 columns

```
my_data = pd.DataFrame(data, columns= ['feature_1',' feature_2 '])
print (my_data)
```

Data Exploration

```
# Using head() method with an argument which helps us to restrict the number of initial records
that should be displayed
data.head(n=2)

# Using .tail() method with an argument which helps us to restrict the number of initial records
that should be displayed
data.tail(n=2)
```

1. Dataset Loading: Pandas

Training Set & Test Set

```
columns = ['  ', ... , '  ']  # n -1
my_data = data[columns ]
# information about the data
print(my_data.describe())

# assigning the 'col_i ' column as target
target = data['col_i ' ]
data.head(n=2)
```

Read and Write to CSV & Excel

```
df = pd.read_csv('file.csv')
df.to_csv('myDataFrame.csv')

df = pd.read_excel('file.xlsx')
df.to_excel('myDataFrame.xlsx')
```

1. Dataset Loading: Pandas

Split training and test sets

```
data_copy = data.copy()
train_set = data_copy.sample(frac=0.80, random_state=0)
test_set  = data_copy.drop(train_set.index)

# Use ‘pop’ to extract the labels
train_set_labels = train_set.pop('col_i ')
test_set_labels  = test_set.pop('col_i ')

print("train_set : \n",train_set)
print("train_set_labels : \n",train_set_labels)
```

1. Dataset Loading: Pandas

pandas.DataFrame.from_dict

By default the keys of the dict become the DataFrame columns:

```
>>> data = {'col_1': [3, 2, 1, 0], 'col_2': ['a', 'b', 'c', 'd']}
>>> pd.DataFrame.from_dict(data)
   col_1 col_2
0      3     a
1      2     b
2      1     c
3      0     d
```

Specify `orient='index'` to create the DataFrame using dictionary keys as rows:

```
>>> data = {'row_1': [3, 2, 1, 0], 'row_2': ['a', 'b', 'c', 'd']}
>>> pd.DataFrame.from_dict(data, orient='index')
       0  1  2  3
row_1  3  2  1  0
row_2  a  b  c  d
```

When using the 'index' orientation, the column names can be specified manually:

```
>>> pd.DataFrame.from_dict(data, orient='index',
...                        columns=['A', 'B', 'C', 'D'])
       A  B  C  D
row_1  3  2  1  0
row_2  a  b  c  d
```

1. Dataset Loading: Files txt

Files

storing data on disk, and reading it back

```
f = open("file.txt","w",encoding="utf8")
```

- file **variable** for operations
- **name** of file on disk (+path…)
- opening **mode**
  - `'r'` read
  - `'w'` write
  - `'a'` append
  - …`'+'` `'x'` `'b'` `'t'`
- **encoding** of chars for *text files*: `utf8` `ascii` `latin1` …

cf. modules `os`, `os.path` and `pathlib`

writing

```
f.write("coucou")
f.writelines(list of lines)
```

reading

read empty string if end of file

```
f.read([n])          → next chars
```
if n not specified, read up to end !
```
f.readlines([n])     → list of next lines
f.readline()         → next line
```

text mode `t` *by default (read/write* `str`*), possible binary mode* `b` *(read/write* `bytes`*). **Convert from/to required type !***

```
f.close()
```
dont forget to **close the file** after use !

```
f.flush()
```
write cache

```
f.truncate([size])
```
resize

reading/writing progress sequentially in the file, modifiable with:

```
f.tell() →position
f.seek(position[,origin])
```

Very common: opening with a guarded block (automatic closing) and reading loop on lines of a text file:

```
with open(…) as f:
    for line in f :
        # processing of line
```

1. Dataset Loading: Numpy

Saving & Loading Text Files

```
import numpy as np

In [1]: a = np.array([1, 2, 3, 4])
In [2]: np.savetxt('test1.txt', a, fmt='%d')
In [3]: b = np.loadtxt('test1.txt', dtype=int)
In [4]: a == b
Out[4]: array([ True,  True,  True,  True], dtype=bool)

# write and read binary files

In [5]: a.tofile('test2.dat')
In [6]: c = np.fromfile('test2.dat', dtype=int)
In [7]: c == a
Out[7]: array([ True,  True,  True,  True], dtype=bool)
```

1. Dataset Loading: Numpy

Saving & Loading On Disk

```
import numpy as np

 # .npy extension is added if not given
In  [8]: np.save('test3.npy', a)
In  [9]: d = np.load('test3.npy')
In [10]: a == d
Out[10]: array([ True,  True,  True,  True], dtype=bool)
```

1. Dataset Loading: glob

The glob module finds **all the pathnames** matching a specified pattern.

```
from glob import glob

# Returns a list of pathnames in list files
pathnames = glob("/home/Desktop/my_images*/*")
# pathnames = glob("/home/Desktop/my_images/*.png")
for path in pathnames:
    print(path)
```

1. Dataset Loading: Scikit Learn

```
from sklearn import datasets

dat = datasets.load_breast_cancer()
print("Examples = ",dat.data.shape ," Labels = ", dat.target.shape)
```

load_boston ([return_X_y])	Load and return the boston house-prices dataset (regression).
load_iris ([return_X_y])	Load and return the iris dataset (classification).
load_diabetes ([return_X_y])	Load and return the diabetes dataset (regression).
load_digits ([n_class, return_X_y])	Load and return the digits dataset (classification).
load_linnerud ([return_X_y])	Load and return the linnerud dataset (multivariate regression).
load_wine ([return_X_y])	Load and return the wine dataset (classification).
load_breast_cancer ([return_X_y])	Load and return the breast cancer wisconsin dataset (classification).

1. Dataset Loading: Scikit Learn

```
from sklearn import datasets

dat = datasets.fetch_20newsgroups(subset='train')
from pprint import pprint
 pprint(list(dat.target_names))
```

fetch_olivetti_faces([data_home, shuffle, ...])	Load the Olivetti faces data-set from AT&T (classification).
fetch_20newsgroups([data_home, subset, ...])	Load the filenames and data from the 20 newsgroups dataset (classification).
fetch_20newsgroups_vectorized([subset, ...])	Load the 20 newsgroups dataset and vectorize it into token counts (classification).
fetch_lfw_people([data_home, funneled, ...])	Load the Labeled Faces in the Wild (LFW) people dataset (classification).
fetch_lfw_pairs([subset, data_home, ...])	Load the Labeled Faces in the Wild (LFW) pairs dataset (classification).
fetch_covtype([data_home, ...])	Load the covertype dataset (classification).
fetch_rcv1([data_home, subset, ...])	Load the RCV1 multilabel dataset (classification).
fetch_kddcup99([subset, data_home, shuffle, ...])	Load the kddcup99 dataset (classification).
fetch_california_housing([data_home, ...])	Load the California housing dataset (regression).

1. Dataset Loading: Scikit Learn

scikit-learn includes utility functions for loading datasets in the **svmlight / libsvm** format. In this format, each line takes the form
<label> <feature-id>:<feature-value> <feature-id>:<feature-value> ....
This format is especially suitable for sparse datasets. In this module, scipy sparse CSR matrices are used for **X** and numpy arrays are used for **Y**.

You may load a dataset like as follows:

```
from sklearn.datasets import load_svmlight_file
X_train, Y_train = load_svmlight_file("/path/to/train_dataset.txt")
```

You may also load two (or more) datasets at once:

```
X_train, y_train, X_test, y_test = load_svmlight_files(("/path/to/train_dataset.txt",
"/path/to/test_dataset.txt"))
```

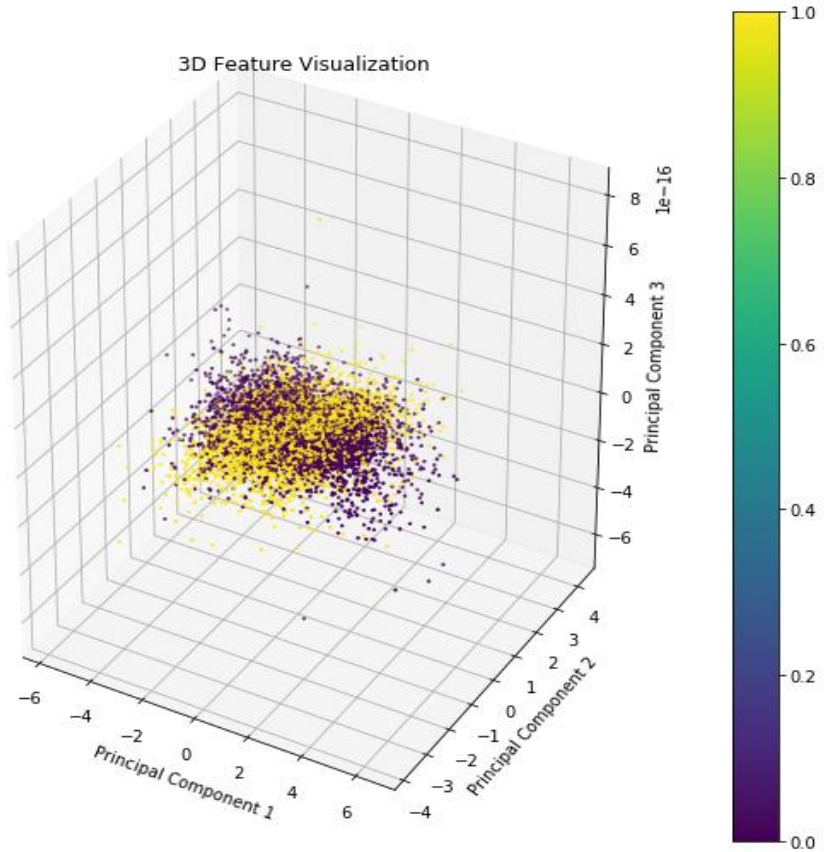
1. Dataset Loading: Scikit Learn

Downloading datasets from the openml.org repository

```
>>> from sklearn.datasets import fetch_openml
>>> mice = fetch_openml(name='miceprotein', version=4)

>>> mice.data.shape
(1080, 77)
>>> mice.target.shape
(1080,)
>>> np.unique(mice.target)
array(['c-CS-m', 'c-CS-s', 'c-SC-m', 'c-SC-s', 't-CS-m', 't-CS-s', 't-SC-m', 't-SC-s'],
dtype=object)
>>> mice.url
'https://www.openml.org/d/40966'
>>> mice.details['version']
'1'
```

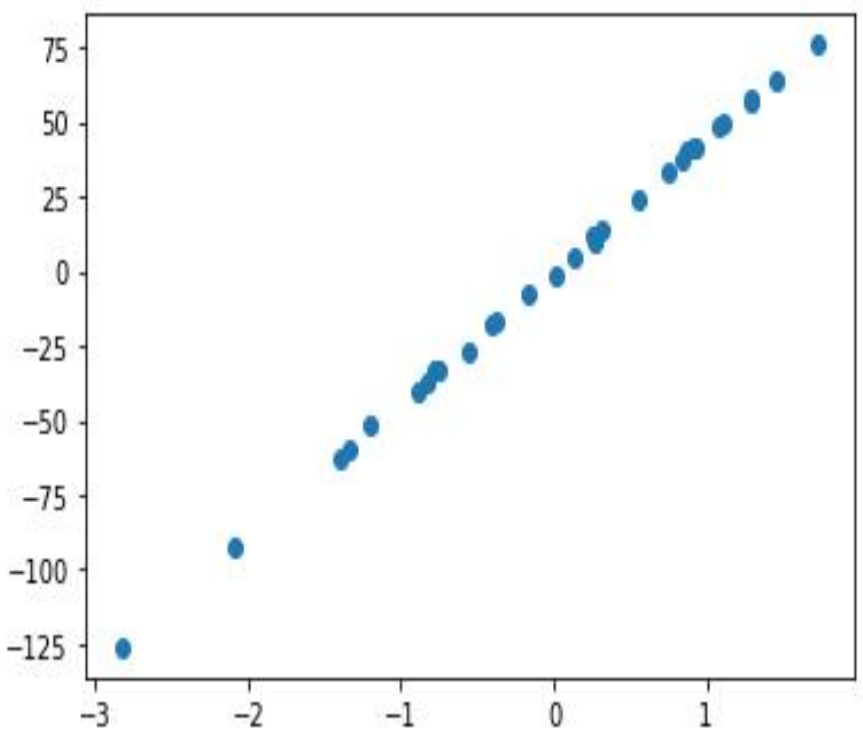
1. Dataset Loading: Generated Datasets - classification

```
from sklearn.datasets import
make_classification
X, y = make_classification( n_samples=10000,
#   n_features=3,
    flip_y=0.1, class_sep=0.1)

print("X shape = ",X.shape)
print("len y   = ", len(y))
print(y)

from mirapy.visualization import visualize_2d,
visualize_3d
visualize_3d(X,y)
```



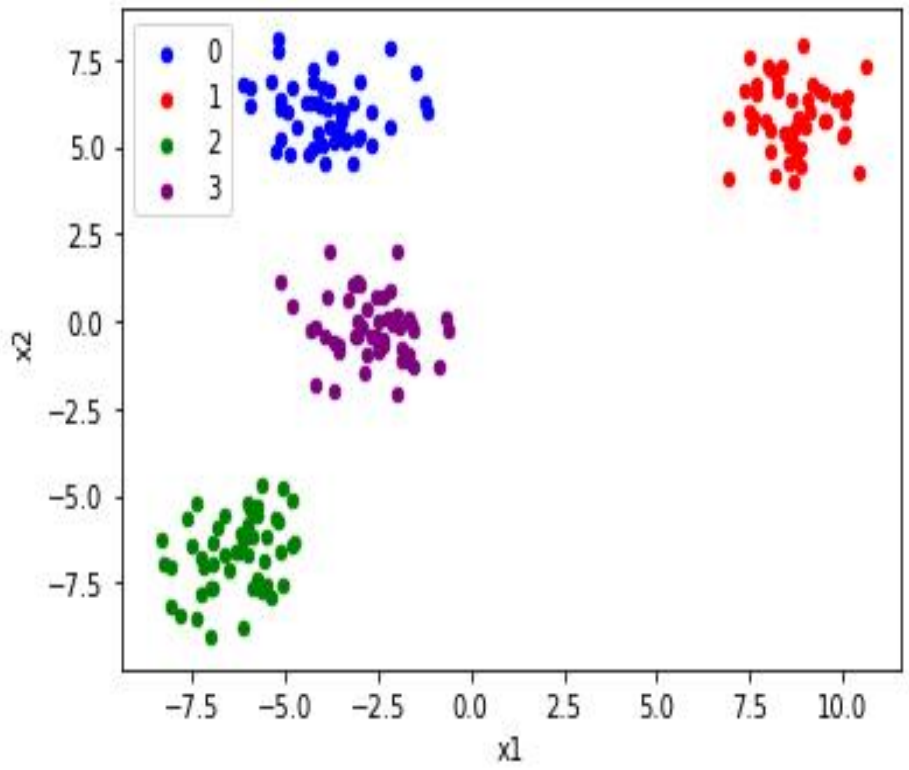
1. Dataset loading: Generated Datasets - Regression

```
from sklearn import datasets
from matplotlib import pyplot as plt

x, y = datasets.make_regression(
   n_samples=30,
   n_features=1,
   noise=0.8)

plt.scatter(x,y)
plt.show()
```



1. Dataset Loading: Generated Datasets - Clustering

```
from sklearn.datasets.samples_generator import
make_blobs
from matplotlib import pyplot as plt
import pandas as pd

X, y = make_blobs(n_samples=200, centers=4,
n_features=2)

Xy = pd.DataFrame(dict(x1=X[:,0], x2=X[:,1], label=y))
groups = Xy.groupby('label')

fig, ax = plt.subplots()
colors = ["blue", "red", "green", "purple"]
for idx, classification in groups:
    classification.plot(ax=ax, kind='scatter', x='x1', y='x2',
label=idx,    color=colors[idx])
plt.show()
```



2. Data Preprocessing: missing values

```
import pandas as pd
import numpy as np
# dictionary of lists
dictionary = {"Name":["Alex", "Mike", "John", "Dave", "Joey"],
        "Height(m)": [1.75, 1.65, 1.73, np.nan, 1.82],
        "Test Score":[70, np.nan, 8, 62, 73]}
# creating a dataframe from dict
df = pd.DataFrame(dictionary)

# using isnull() function this function return dataframe of Boolean values
# which are True for NaN values.
print(df.isnull())
print("SUM : \n",df.isnull().sum())
```

	Name	Height(m)	Test Score
0	False	False	False
1	False	False	True
2	False	False	False
3	False	True	False
4	False	False	False

SUM :

Name	0
Height(m)	1
Test Score	1

dtype: int64

Dealing with missing values :

```
df = df.fillna('*')
df['Test Score'] = df['Test Score'].fillna('*')
df['Test Score'] = df['Test Score'].fillna(df['Test Score'].mean())
df['Test Score'] = df['Test Score'].fillna(df['Test Score'].interpolate())
df= df.dropna()  #delete the missing rows of data
df['Height(m)']= df['Height(m)'].dropna()
```

2. Data Preprocessing: missing values

```
# Dealing with Non-standard missing values:
# dictionary of lists
dictionary_1 = {"Name":["Alex", "Mike", "John", "Dave", "Joey"],
       "Height(m)": [1.75, 1.65, "-", "na", 1.82],
       "Test Score":[70, np.nan, 8, 62, 73]}

# creating a dataframe from list
df_1 = pd.DataFrame(dictionary_1)
print("df_1 : \n",df_1)
print("isnull : \n",df_1.isnull())

df_1 = df_1.replace(["-","na"], np.nan)
print("replace non-standard missing values : \n",df_1)

df_1 = df_1.fillna(0)
print("fillna : \n",df_1)
```

Out[27]:

	Name	Height(m)	Test Score
0	Alex	1.75	70.0
1	Mike	1.65	NaN
2	John	-	8.0
3	Dave	na	62.0
4	Joey	1.82	73.0

2. Data Preprocessing: missing values

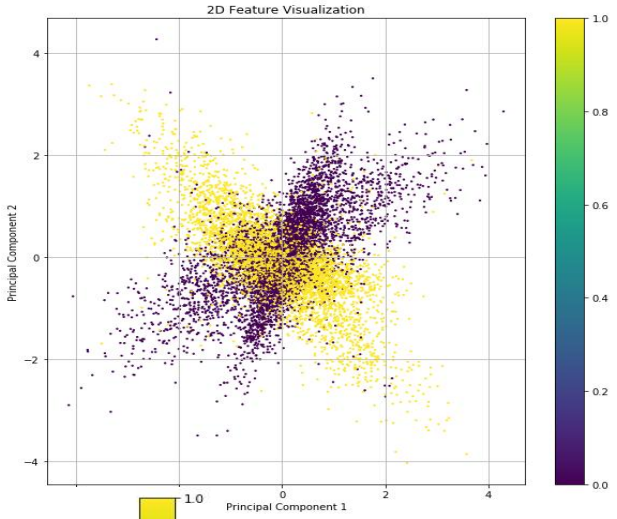
```
import numpy as np
from sklearn.impute import SimpleImputer
X = [[np.nan, 2], [6, np.nan], [7, 6]]
# mean, median, most_frequent, constant(fill_value = )
imp = SimpleImputer(missing_values = np.nan, strategy='mean')
data = imp.fit_transform(X)
print(data)
```

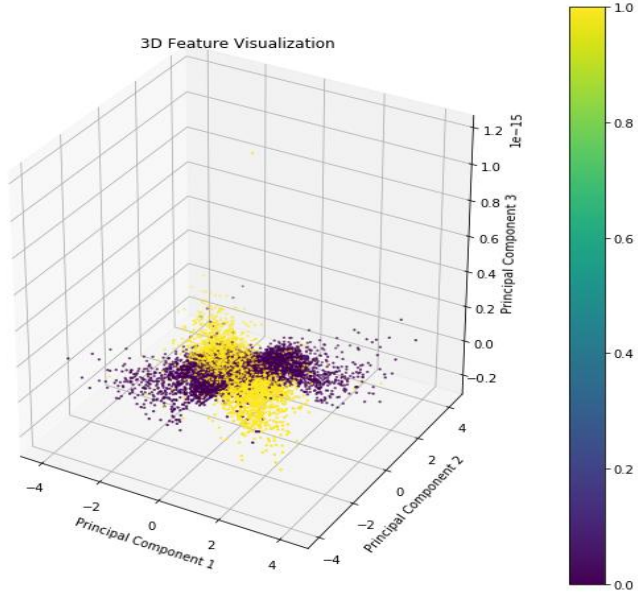
Multivariate feature imputation :
IterativeImputer
Nearest neighbors imputation :
KNNImputer
Marking imputed values :
MissingIndicator

```
>>> import pandas as pd
>>> df = pd.DataFrame([["a", "x"],
...                    [np.nan, "y"],
...                    ["a", np.nan],
...                    ["b", "y"]], dtype="category")
...
>>> imp = SimpleImputer(strategy="most_frequent")
>>> print(imp.fit_transform(df))
[['a' 'x']
 ['a' 'y']
 ['a' 'y']
 ['b' 'y']]
```

2. Data Preprocessing: Data Projection

```
from sklearn.datasets import make_classification

X, y = make_classification( n_samples=10000,
   n_features=4,
    flip_y=0.1, class_sep=0.1)

print("X shape = ",X.shape)
print("len y   = ", len(y))
print(y)

from mirapy.visualization import visualize_2d,
visualize_3d
visualize_2d(X,y)
visualize_3d(X,y)
```





Thank you for your attention

Hichem Felouat ...