

Fondements du Machine Learning

Chapitre 4:
Ingénierie des caractéristiques (Features)

Mme BEDDAD Fatima
fatima.bedad@univ-temouchent.edu.dz

2025-2026

Machine Learning Vocabulary 1

- 1) **Examples:** Items or instances of data used for learning or evaluation. In our spam problem, these examples correspond to the collection of email messages we will use for learning and testing.
- 2) **Training sample:** Examples used to train a learning algorithm. In our spam problem, the training sample consists of a set of email examples along with their associated labels.
- 3) **Labels:** Values or categories assigned to examples. In classification problems, examples are assigned specific categories, for instance, the spam and not-spam categories in our binary classification problem. In regression, items are assigned real-valued labels.

Machine Learning Vocabulary 2

- 4) **Features:** The set of attributes, often represented as a vector, associated to an example. In the case of email messages, some relevant features may include the length of the message, the name of the sender, various characteristics of the header, the presence of certain keywords in the body of the message, and so on.
- 5) **Test sample:** Examples used to evaluate the performance of a learning algorithm. The test sample is separate from the training and validation data and is not made available in the learning stage. In the spam problem, the test sample consists of a collection of email examples for which the learning algorithm must predict labels based on features. These predictions are then compared with the labels of the test sample to measure the performance of the algorithm.
- 6) **Loss function:** A function that measures the difference, or loss, between a predicted label and a true label.

Main Challenges of Machine Learning

In short, since your main task is to select a learning algorithm and train it on some data, the two things that can go wrong are “**bad data**” and “**bad algorithm**”.

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.

Steps to Build a Machine Learning System

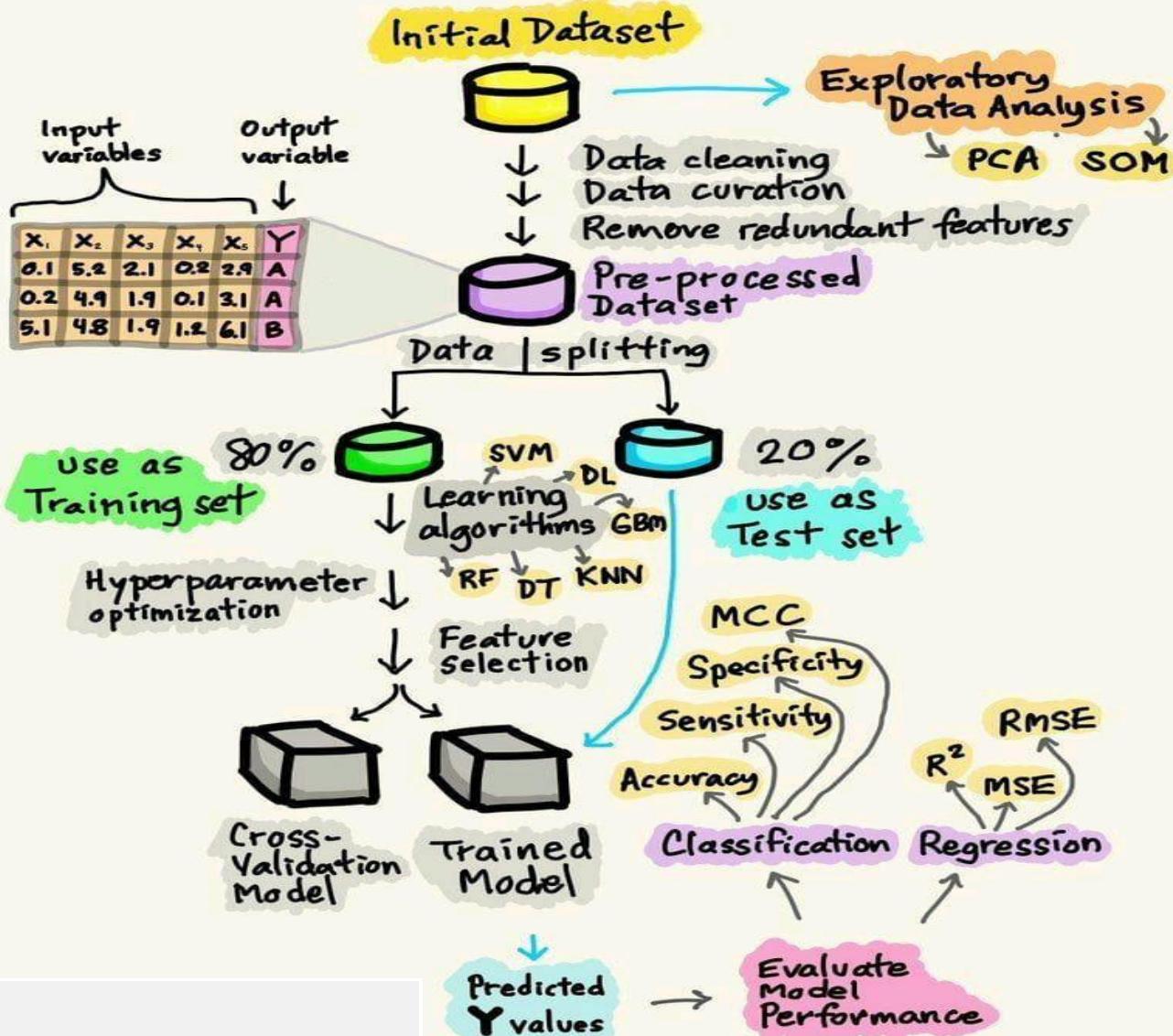
1- Database (Dataset Loading)

	feature 1	f 2	f 3	f 4	f 5	Label
1	rank 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

Examples (Training sample + Test sample)

One Example

BUILDING THE MACHINE LEARNING MODEL



1- Database (Dataset Loading)

Places to Find Free Datasets :

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

1- Database (Dataset Loading)

- Pandas
- Files txt
- Numpy
- Scikit Learn
- Glob

1- Database (Dataset Loading): glob

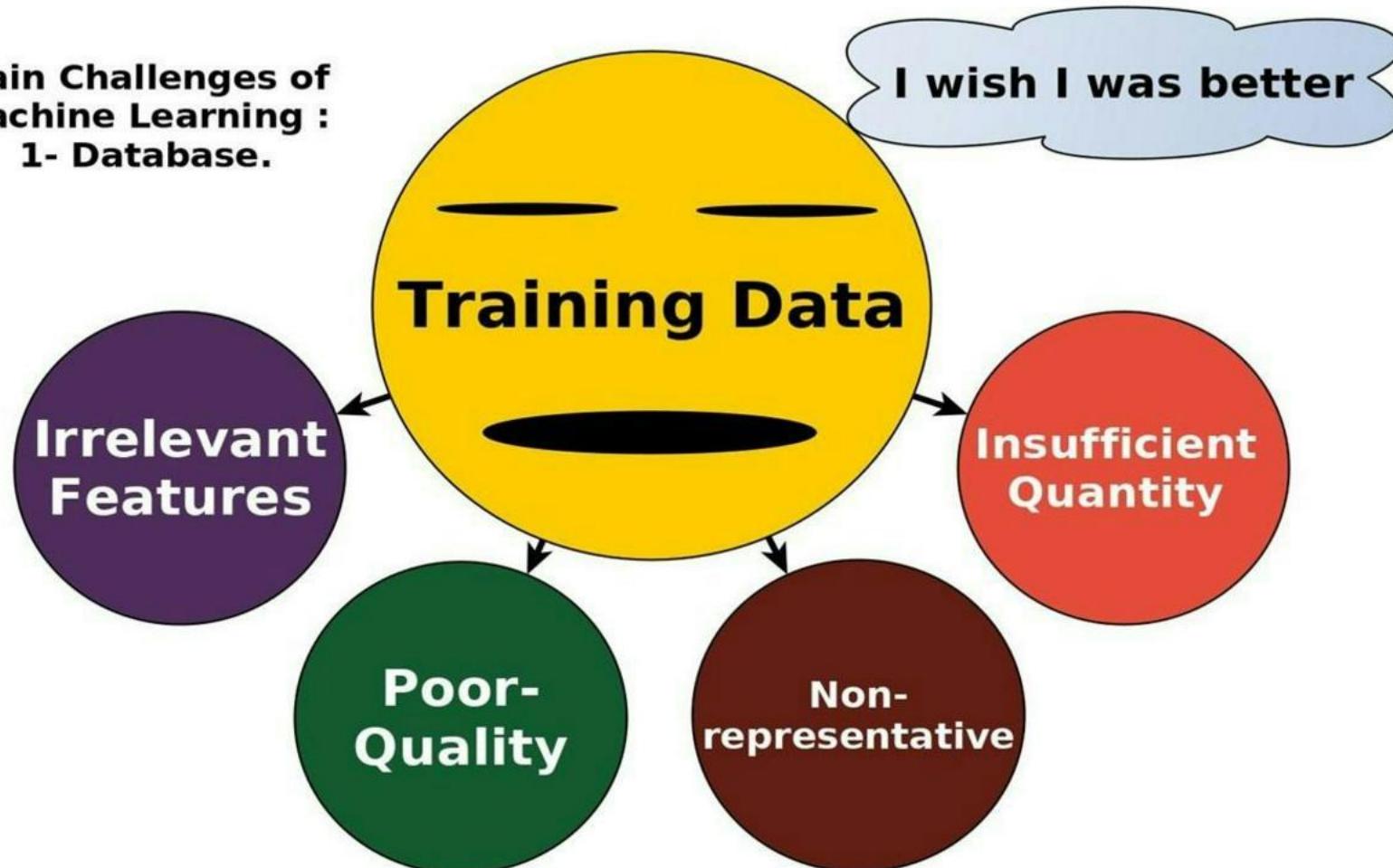
The **glob** module finds **all the pathnames** matching a specified pattern.

```
from glob import glob
```

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

1- Database

Main Challenges of
Machine Learning :
1- Database.



2- Improving data quality

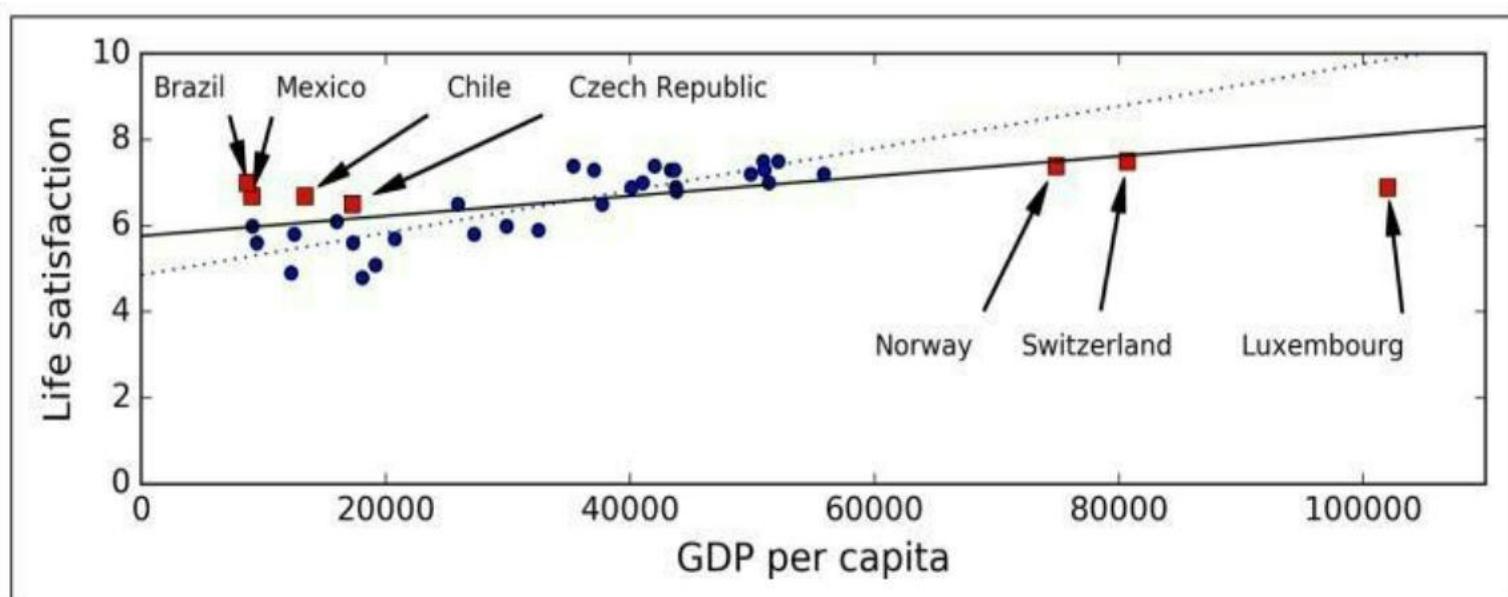
1- Insufficient Quantity of Training Data :

Machine Learning takes a lot of data for most Machine Learning algorithms to work properly. Even for very simple problems you typically need thousands of examples, and for complex problems such as image or speech recognition you may need millions of examples (unless you can reuse parts of an existing model).

2- Improving data quality

2) Non-representative Training Data:

In order to generalize well, it is crucial that your training data be representative of the new cases you want to generalize to. This is true whether you use instance-based learning or model-based learning.



2- Improving data quality

3) Poor-Quality Data:

If your training data is full of errors, outliers, and noise (e.g., due to poor quality measurements), it will make it harder for the system to detect the underlying patterns, so your system is less likely to perform well. It is often well worth the effort to spend time cleaning up your training data. The truth is, most data scientists spend a significant part of their time doing just that. For example:

- 1) If some instances are clearly outliers, it may help to simply discard them or try to fix the errors manually.
- 2) If some instances are missing a few features (e.g., 5% of your customers did not specify their age), you must decide whether you want to ignore this attribute altogether, ignore these instances, fill in the missing values (e.g., with the median age), or train one model with the feature and one model without it, and so on.

2- Improving data quality

4) Irrelevant Features:

Your system will only be capable of learning if the **training data contains enough relevant features** and not too many irrelevant ones. A critical part of the success of a Machine Learning project is coming up with a good set of features to train on. This process, called **feature engineering**, involves:

- 1) **Feature selection:** selecting the most useful features to train on among existing features.
- 2) **Feature extraction:** combining existing features to produce a more useful one (dimensionality reduction algorithms can help).
- 3) **Creating new features** by gathering new data.

2- Improving data quality :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()) [ for each column ]
```

Dealing with missing values :

```
df = df.fillna('*') : exmaple * = 0 or carachter X [missing values = 0]
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()
```

	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
```

```
# Filling missing values using fillna(), replace() and interpolate()
print("Filling missing values : \n")

# print("fillna : \n",df.fillna(99)) # char : "*" or "X" ...
# Filling null values with the previous ones
print("pad : \n",df.fillna(method ="pad"))

# Filling null values with the next ones
# print("bfill : \n",df.fillna(method ="bfill"))

# filling the null value for each column independently
# df["Height(m)"] = df["Height(m)"].fillna(0)
# df["Test Score"] = df["Test Score"].fillna(999)

# mean and interpolate
# df["Height(m)"] = df["Height(m)"].fillna(df["Height(m)"].mean())
# df["Test Score"] = df["Test Score"].fillna(df["Test Score"].mean())
```

2- Improving data quality :missing values (example 2)

```
# 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()) « to display the missing  
values »
```

```
df_1 = df_1.replace(["-", "na"], np.nan) « replace unrecognized values  
with nan »
```

```
print("replace non-standard missing values : \n", df_1)
```

```
df_1 = df_1.fillna(0) « use  
a method for missing  
values like exemple 1 «  
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

Exercice :

1- Uploading Dataset Employees.Csv Use Pandas .

```
import pandas as pd  
data = pd.read_csv("/content/employees.csv")  
print (data)  
print("-----")
```

2- Print The Data

3- Use Df.IsNotNull().Sum()

4 –use other méthod

from matplotlib import pyplot as plt

```
# To check Missing Values  
import seaborn as sns  
sns.heatmap(data.isnull(), cbar=False, yticklabels=False, cmap="viridis")
```

```
plt.show()
```

5-Fulling nul values

```
print("-----")  
from sklearn.impute import SimpleImputer  
  
imp = SimpleImputer(missing_values = np.nan, strategy="most_frequent")  
data = imp.fit_transform(data)  
print(data)
```

2- Improving data quality :missing values with sklearn

```
import numpy as np
from sklearn.impute import SimpleImputer # this
class for missing values #
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

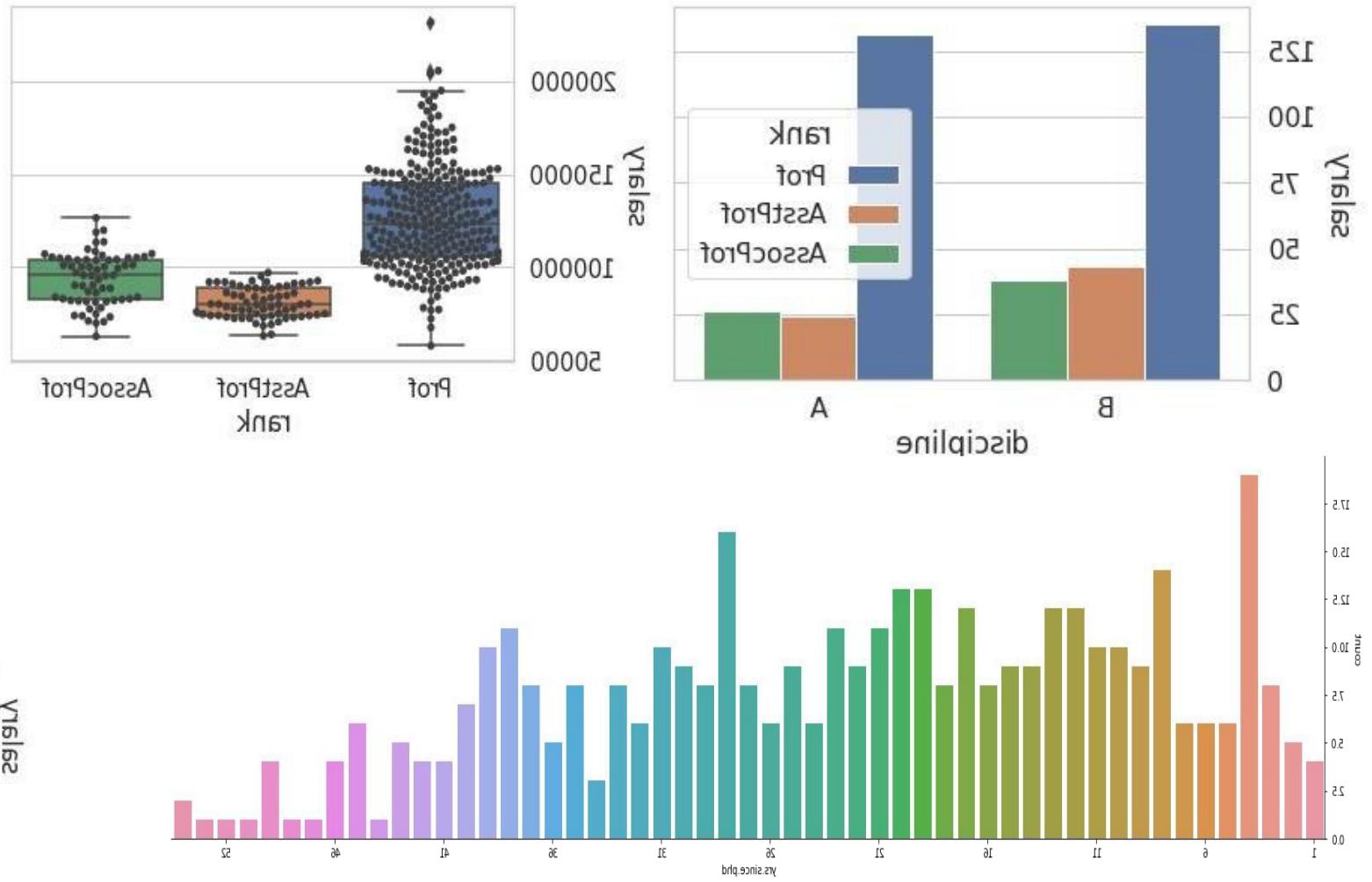
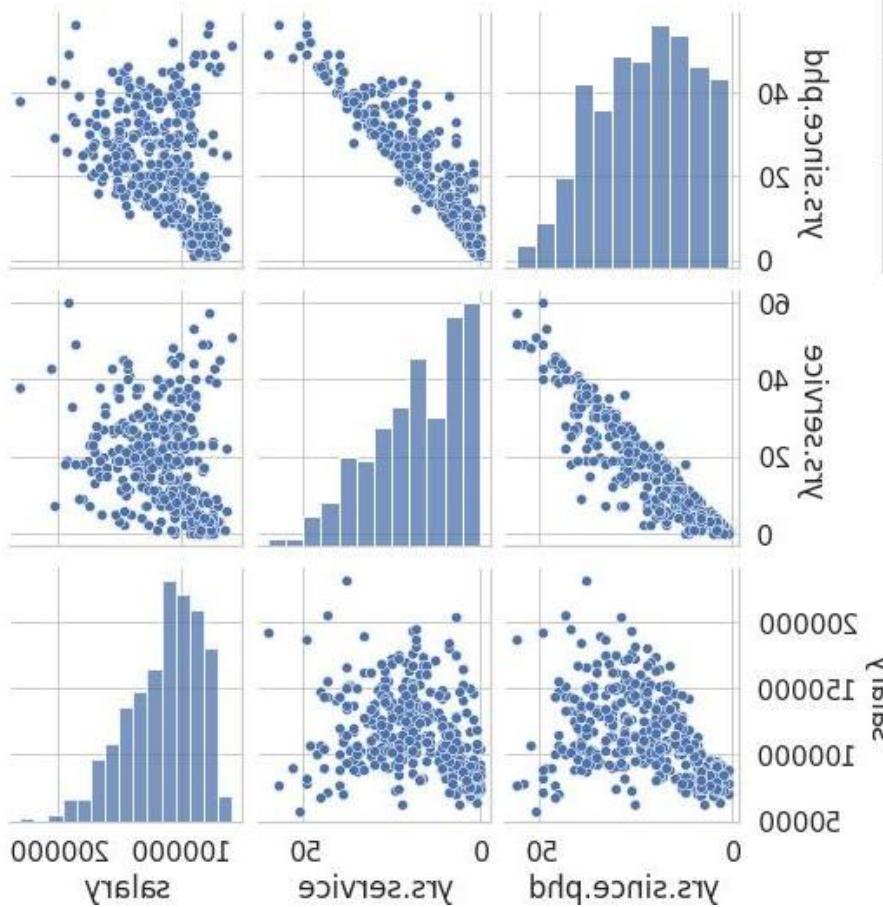
```
>>> import pandas as pd
>>> df = pd.DataFrame([["a", "x"],
...                     [np.nan, "y"],
...                     ["a", np.nan],
...                     ["b", "y"]], dtype="category")
...
>>> imp = SimpleImputer(strategy="most_frequent")
>>> print(imp.fit_transform(df))
[[['a' 'x']
  ['a'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

2. Data Preprocessing: Exploratory Data Analysis

Exploratory Data Analysis or (EDA) is **understanding the data sets** by summarizing their main characteristics often plotting them visually. This step is very important especially when we arrive at modeling the data **in order to apply Machine learning**.

- Plotting in EDA consists of Histograms, Box plot, Scatter plot and many more. It often **takes much time** to explore the data. Through the process of EDA, we can ask to define **the problem statement** or definition on our data set which is very important.

2. Data Preprocessing: Exploratory Data Analysis



2. Data Preprocessing: Description

```
# Load dataset
df = pd.read_csv("/content/Salaries.csv", index_col=0)

# Display dimensions of dataframe
print(df.shape)
print(df.info())

print("-----")
# Display first 10 records
print(df.head(10))

print("-----")
print("-----")
# Display last 10 records
print(df.tail(10))

print("-----")
# List types of all columns
print(df.dtypes)
```

```
# Display statistics for numeric columns
print(df.describe())

print("-----")
print("\n")
print("yrs.since.phd :")           I
print(df["yrs.since.phd"].describe()) I

print("-----")
df["yrs.since.phd"].hist(figsize=(8, 10), bins=100, legend=True, xlabelsize=8, ylab
```

```
# Numerical data distribution
# First list all the types of our data from our dataset and take only the numerical
print("All types : ",list(set(df.dtypes.tolist())))
I
print("-----")
df_num = df.select_dtypes(include = ["int64"]) # 'float64', 'int64'
print(df_num.head(5))

# Now lets plot them all
df_num.hist(figsize=(12, 16), bins=50, xlabelsize=8, ylabelsize=8)
# df.hist(figsize=(12, 16), bins=50, xlabelsize=8, ylabelsize=8)
```

```
import seaborn as sns

# Find the pairwise correlation of all columns in the dataframe.
print(df.corr())

print("-----")
# Heatmap
print("\n")
corrMatrix = df.corr()
# sns.heatmap(corrMatrix, annot=True)
sns.heatmap(corrMatrix, annot=True, linewidth=0.01, square=True, cmap="RdBu", linecolor="black")
```

```
# value_counts() function : Return a Series containing counts of unique values.
print("sex      : \n",df["sex"].value_counts())
print("\n")
print("discipline : \n",df["discipline"].value_counts())

print("-----")
# (normalize=True) means the object returned will contain the relative frequencies
print(df["sex"].value_counts(normalize=True))
print("\n")
print(df["discipline"].value_counts(normalize=True))
print("\n")

print("-----")
# plot.bar(title='') function is used to plot bargraph
df["sex"].value_counts().plot.bar(title="Sex")

df_copy = df.copy()

# Drop features
df_copy = df_copy.drop(["rank" , "sex"] , axis = 1)
print(df_copy.head(5))
```

```
df_copy = df.copy()
print(df_copy.head())
print("-----")
# By replace() function we replace the value "B" with "C" in discipline column
print("\n")
df_copy["discipline"] = df_copy["discipline"].replace("B", "C")
print(df_copy.head())
```

```
# Dropping the duplicate rows
duplicate_rows = df_copy[df_copy.duplicated()]
print("number of duplicate rows (before) : ",duplicate_rows.shape)

df_copy.drop_duplicates(inplace=True)
duplicate_rows = df_copy[df_copy.duplicated()]
print("number of duplicate rows (after) : ",duplicate_rows.shape)
print("shape : ",df_copy.shape)
```

```
number of duplicate rows (before) : (4, 6)
number of duplicate rows (after) : (0, 6)
shape : (393, 6)
```

```
# Sort the data frame by salary and create a new data frame
# ascending=False, na_position="first"
df_sorted = df.sort_values(by = "salary", ascending=False)
print(df_sorted.head(10))
```

```
# Mapping of data feature values
```

```
"""
```

If a certain column in dataset has categorical values and for any model training we want to convert it into numeric type then we must use the .map() function and allocate the desired numeric values to the categorical values as you can see from the above example. It will help you in predicting better results.

```
"""
```

```
df_copy = df.copy()
```

```
print(df.head(10))
```

```
print("-----")
```

```
df_copy["sex"] = df_copy["sex"].map({"Male":0,"Female":1})
```

```
print("sex : \n",df_copy["sex"].head(10))
```

```
print("-----")
```

```
print("rank : \n",df["rank"].value_counts())
```

```
print("\n")
```

```
df_copy["rank"] = df_copy["rank"].map({"Prof":0, "AsstProf":1, "AssocProf":2 })
```

```
print("rank : \n",df_copy["rank"].head(10))
```