



NOTRE DAME UNIVERSITY BANGLADESH

Machine Learning Lab Report-01

Course Code: CSE4214

Course Title: Machine Learning Lab

Lab Task Topic: Data Loading and Preprocessing Using Pandas

Submitted by:

Name: Istiak Alam

ID: 0692230005101005

Batch: CSE-20

Submission Date: April 23, 2026

Submitted to:

A. H. M. Saiful Islam

Chairman, Dept of CSE

Notre Dame University Bangladesh

Table of Contents

1	Importing Required Libraries	1
2	Loading CSV Data	2
3	Loading the Dataset and Displaying Records	2
4	Displaying the Last Records of the Dataset	3
5	Encoding Marital Status Attribute	3
6	Encoding Housing Loan Attribute	4
7	Encoding Loan Attribute	5
8	Checking Unique Job Categories	5
9	Encoding Job Attribute	6
10	Checking Unique Values of Month Attribute	7
11	Encoding Month Attribute	7
12	Inspecting Unique Values of Education Attribute	8
13	Encoding Education Attribute	9
14	Unique Values of the Outcome of Previous Marketing Campaign (poutcome)	10
15	Encoding Poutcome Attribute	10
16	Normalizing the Balance Attribute	11
17	Normalization of pdays Attribute	12
18	Feature Scaling using Min-Max Scaler	13

Objective

The objective of this experiment is to familiarize with basic data handling and preprocessing techniques required before applying Machine Learning algorithms. This lab focuses on loading a dataset, inspecting its structure, handling categorical variables, and converting them into numerical representations suitable for model training.

Dataset Description

The dataset used in this experiment is `bank.csv`, which contains customer information related to a banking institution. The dataset includes demographic details, financial attributes, and campaign-related information. The target variable indicates whether a customer subscribed to a term deposit. The main objective is to load a CSV dataset into a Pandas DataFrame and inspect the initial records.

1 Importing Required Libraries

```
[1]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Explanation

Machine Learning workflows rely on multiple Python libraries, each serving a specific purpose. NumPy provides efficient data structures and mathematical functions for numerical computation. Pandas enables loading, cleaning, and manipulating structured datasets. Matplotlib is used to generate basic plots and graphs, while Seaborn builds on Matplotlib to produce more informative and visually appealing statistical visualizations. Importing these libraries prepares the environment for data analysis and model development.

Output

No visible output is produced if the libraries are successfully imported. If any library is missing from the environment, an error message such as `ModuleNotFoundError` is displayed.

```
-----
ModuleNotFoundError                                Traceback (most recent call last)
Cell In[2], line 4
      2 import pandas as pd
      3 import matplotlib.pyplot as plt
----> 4 import seaborn as sns

ModuleNotFoundError: No module named 'seaborn'
```

2 Loading CSV Data

```
[2]: import pandas as pd

df = pd.read_csv("bank.csv")
df.head()
```

Explanation

Pandas provides the `read_csv()` function to load structured data into a DataFrame. Initially, the dataset was loaded without specifying the delimiter, which resulted in improper column parsing. This step helps identify formatting issues in raw data.

Output

The output displays the first five records of the dataset, revealing incorrect column separation due to the delimiter mismatch.

```
[2]: age;"job";"marital";"education";"default";"balance";"housing";"loan";
"contact";"day";"month";"duration";"campaign";"pdays";"previous";"poutcome";
"y"
0 30;"unemployed";"married";"primary";"no";1787;...
1 33;"services";"married";"secondary";"no";4789;...
2 35;"management";"single";"tertiary";"no";1350;...
3 30;"management";"married";"tertiary";"no";1476...
4 59;"blue-collar";"married";"secondary";"no";0;...
```

3 Loading the Dataset and Displaying Records

```
[3]: df = pd.read_csv("bank.csv", sep = ';')
df.head()
```

Explanation

The dataset `bank.csv` uses a semicolon (;) as its field separator instead of the default comma. Therefore, the delimiter is explicitly specified while loading the file. The `read_csv()` function stores the data in a Pandas DataFrame, which is a tabular data structure. The `head()` function is then used to display the first five records of the dataset, allowing quick verification of data correctness and column structure.

Output

The output displays the first five rows of the dataset along with all column names, confirming that the data has been loaded correctly into the DataFrame.

```
[3]:   age      job  marital  education  default  balance  housing  loan  \
0   30  unemployed  married   primary     no     1787     no    no
1   33   services  married  secondary     no     4789     yes   yes
2   35  management  single   tertiary     no     1350     yes   no
3   30  management  married   tertiary     no     1476     yes   yes
4   59  blue-collar  married  secondary     no         0     yes   no
```

	contact	day	month	duration	campaign	pdays	previous	poutcome	y
0	cellular	19	oct	79	1	-1	0	unknown	no
1	cellular	11	may	220	1	339	4	failure	no
2	cellular	16	apr	185	1	330	1	failure	no
3	unknown	3	jun	199	4	-1	0	unknown	no
4	unknown	5	may	226	1	-1	0	unknown	no

4 Displaying the Last Records of the Dataset

```
[4]: df.tail()
```

Explanation

The `tail()` function in Pandas is used to display the last few entries of a DataFrame. By default, it returns the final five rows. This operation is useful for verifying that the dataset has been loaded completely and for checking any anomalies or missing values at the end of the dataset.

Output

The output displays the last five rows of the dataset along with all corresponding column values.

```
[4]:      age      job  marital  education  default  balance  housing  loan  \
4516  33  services  married  secondary     no     -333     yes   no
4517  57 self-employed  married  tertiary     yes    -3313     yes  yes
4518  57  technician  married  secondary     no      295     no   no
4519  28 blue-collar  married  secondary     no     1137     no   no
4520  44  entrepreneur   single  tertiary     no     1136     yes  yes

      contact  day month  duration  campaign  pdays  previous  poutcome  y
4516  cellular  30  jul     329         5     -1         0  unknown  no
4517  unknown   9  may     153         1     -1         0  unknown  no
4518  cellular  19  aug     151        11     -1         0  unknown  no
4519  cellular   6  feb     129         4    211         3   other  no
4520  cellular   3  apr     345         2    249         7   other  no
```

5 Encoding Marital Status Attribute

```
[5]: def replace_marital(val):
      if val == 'single':
          return 0
      else:
          return 1

df['marital'] = df['marital'].apply(replace_marital)
df.head()
```

Explanation

In this step, the categorical attribute `marital` is converted into a numerical format to make it suitable for Machine Learning algorithms. A user-defined function named `replace_marital` is created, which assigns the value 0 to customers with marital status `single` and the value 1 to all other categories. The `apply()` function is then used to apply this transformation to the entire `marital` column of the dataset. This process simplifies categorical data while preserving relevant information.

Output

The output displays the first five rows of the dataset where the `marital` column is successfully transformed into numerical values. Customers with marital status `single` are represented by 0, while all other marital statuses are represented by 1.

```
[5]:   age      job  marital  education  default  balance  housing  loan  \
0   30  unemployed      1   primary     no     1787      no   no
1   33   services      1  secondary     no     4789     yes  yes
2   35  management      0   tertiary     no     1350     yes  no
3   30  management      1   tertiary     no     1476     yes  yes
4   59 blue-collar      1  secondary     no         0     yes  no

      contact  day month  duration  campaign  pdays  previous  poutcome  y
0  cellular   19  oct      79         1     -1         0  unknown  no
1  cellular   11  may     220         1    339         4  failure  no
2  cellular   16  apr     185         1    330         1  failure  no
3   unknown    3  jun     199         4     -1         0  unknown  no
4   unknown    5  may     226         1     -1         0  unknown  no
```

6 Encoding Housing Loan Attribute

```
[6]: df["housing"] = df["housing"].map({"yes": 1, "no": 0}).get()
df.head()
```

Explanation

In this step, the categorical attribute `housing` is converted into a numerical format using the `map()` function. The values `yes` and `no` are mapped to 1 and 0 respectively. This conversion is necessary because Machine Learning algorithms require numerical input features. The `get` method ensures safe mapping of values within the column.

Output

The output displays the first five rows of the dataset where the `housing` column has been successfully transformed into numerical values. A value of 1 indicates the presence of a housing loan, while 0 indicates the absence of a housing loan.

```
[6]:   age      job  marital  education  default  balance  housing  loan  \
0   30  unemployed      1   primary     no     1787      0   no
1   33   services      1  secondary     no     4789      1  yes
2   35  management      0   tertiary     no     1350      1   no
3   30  management      1   tertiary     no     1476      1  yes
```

```

4  59  blue-collar      1  secondary      no      0      1  no

   contact  day month  duration  campaign  pdays  previous  poutcome  y
0  cellular  19  oct     79         1     -1        0  unknown  no
1  cellular  11  may    220         1    339        4  failure  no
2  cellular  16  apr    185         1    330        1  failure  no
3  unknown   3  jun    199         4     -1        0  unknown  no
4  unknown   5  may    226         1     -1        0  unknown  no

```

7 Encoding Loan Attribute

```
[7]: df["loan"] = df["loan"].replace({"yes": 1, "no": 0})
df.head()
```

Explanation

In this step, the categorical attribute `loan` is converted into numerical form to ensure compatibility with Machine Learning algorithms. The `replace()` function is used to map the value `yes` to 1 and `no` to 0. This binary encoding simplifies the data representation while retaining the original meaning of the attribute.

Output

The output displays the first five rows of the dataset, where the `loan` column is successfully transformed into numerical values. A value of 1 indicates the presence of a personal loan, while 0 indicates the absence of a loan.

```
[7]:   age      job  marital  education  default  balance  housing  loan  \
0   30  unemployed      1   primary     no    1787      0      0
1   33   services      1  secondary     no    4789      1      1
2   35  management      0  tertiary     no    1350      1      0
3   30  management      1  tertiary     no    1476      1      1
4   59  blue-collar      1  secondary     no      0      1      0

   contact  day month  duration  campaign  pdays  previous  poutcome  y
0  cellular  19  oct     79         1     -1        0  unknown  no
1  cellular  11  may    220         1    339        4  failure  no
2  cellular  16  apr    185         1    330        1  failure  no
3  unknown   3  jun    199         4     -1        0  unknown  no
4  unknown   5  may    226         1     -1        0  unknown  no

```

8 Checking Unique Job Categories

```
[8]: df["job"].unique()
```

Explanation

The `unique()` function is used to identify all distinct values present in the `job` column of the dataset. This helps in understanding the different job categories of the customers and is useful for preprocessing and encoding categorical variables before applying Machine Learning algorithms.

Output

The output displays an array of unique job categories present in the dataset, such as unemployed, services, management, blue-collar, self-employed, technician, entrepreneur, admin., student, housemaid, retired, and unknown.

```
[8]: array(['unemployed', 'services', 'management', 'blue-collar',
          'self-employed', 'technician', 'entrepreneur', 'admin.', 'student',
          'housemaid', 'retired', 'unknown'], dtype=object)
```

9 Encoding Job Attribute

```
[9]: df["job"] = df["job"].replace({'unemployed': 0,
                                   'services': 0,
                                   'management': 1,
                                   'blue-collar': 0,
                                   'self-employed': 0,
                                   'technician': 1,
                                   'entrepreneur': 1,
                                   'admin.': 0,
                                   'student': 1,
                                   'housemaid': 0,
                                   'retired': 0,
                                   'unknown': np.nan})
df.head()
```

Explanation

The categorical attribute job is transformed into a numerical format to make it suitable for Machine Learning algorithms. The `replace()` function is used to map specific job categories to binary values: jobs considered as professional or managerial (e.g., management, technician, entrepreneur, student) are assigned 1, while other jobs (e.g., unemployed, services, blue-collar, self-employed, admin., housemaid, retired) are assigned 0. Any unknown job entries are replaced with NaN. This encoding simplifies categorical data for model training.

Output

The output displays the first five rows of the dataset with the job column converted to numerical values. Job categories are represented as 0 or 1, and any unknown values are represented as NaN.

```
[9]:
```

	age	job	marital	education	default	balance	housing	loan	contact	\
0	30	0.0	1	1.0	no	1787	0	0	cellular	
1	33	0.0	1	2.0	no	4789	1	1	cellular	
2	35	1.0	0	3.0	no	1350	1	0	cellular	
3	30	1.0	1	3.0	no	1476	1	1	unknown	
4	59	0.0	1	2.0	no	0	1	0	unknown	

	day	month	duration	campaign	pdays	previous	poutcome	y
0	19	10	79	1	-1	0	unknown	no
1	11	5	220	1	339	4	failure	no

2	16	4	185	1	330	1	failure	no
3	3	6	199	4	-1	0	unknown	no
4	5	5	226	1	-1	0	unknown	no

10 Checking Unique Values of Month Attribute

```
[10]: df["month"].unique()
```

Explanation

The `unique()` function is used to identify all distinct values present in the `month` column of the dataset. This operation helps to understand the range of months represented in the data and is useful for preprocessing, such as converting categorical month names into numerical values for Machine Learning algorithms.

Output

The output displays an array of unique month names in the dataset. For example:

```
[10]: array(['oct', 'may', 'apr', 'jun', 'feb', 'aug', 'jan', 'jul', 'nov',  
          'sep', 'mar', 'dec'], dtype=object)
```

11 Encoding Month Attribute

```
[11]: df.month = df.month.map({  
      'oct': 10,  
      'may': 5,  
      'apr': 4,  
      'jun': 6,  
      'feb': 2,  
      'aug': 8,  
      'jan': 1,  
      'jul': 7,  
      'nov': 11,  
      'sep': 9,  
      'mar': 3,  
      'dec': 12  
    })  
df.head(10)
```

Explanation

The `month` column contains the names of months as categorical string values. To convert this categorical data into a numerical format suitable for Machine Learning models, a mapping is applied where each month name is replaced with its corresponding integer value (e.g., 'jan' = 1, 'feb' = 2, ..., 'dec' = 12). The `map()` function is used to transform the entire `month` column according to this mapping. This preprocessing step ensures that the month attribute can be interpreted quantitatively by algorithms.

Output

The output displays the first ten rows of the dataset after transformation. The month column now contains integer values representing the months, while all other columns remain unchanged.

```
[11]:  age  job  marital  education  default  balance  housing  loan  contact  \
0   30  0.0      1         1.0      no     1787      0      0  cellular
1   33  0.0      1         2.0      no     4789      1      1  cellular
2   35  1.0      0         3.0      no     1350      1      0  cellular
3   30  1.0      1         3.0      no     1476      1      1  unknown
4   59  0.0      1         2.0      no         0      1      0  unknown
5   35  1.0      0         3.0      no     747      0      0  cellular
6   36  0.0      1         3.0      no     307      1      0  cellular
7   39  1.0      1         2.0      no     147      1      0  cellular
8   41  1.0      1         3.0      no     221      1      0  unknown
9   43  0.0      1         1.0      no     -88      1      1  cellular

      day  month  duration  campaign  pdays  previous  poutcome  y
0   19   NaN      79         1      -1         0  unknown  no
1   11   NaN     220         1     339         4  failure  no
2   16   NaN     185         1     330         1  failure  no
3    3   NaN     199         4      -1         0  unknown  no
4    5   NaN     226         1      -1         0  unknown  no
5   23   NaN     141         2     176         3  failure  no
6   14   NaN     341         1     330         2   other  no
7    6   NaN     151         2      -1         0  unknown  no
8   14   NaN      57         2      -1         0  unknown  no
9   17   NaN     313         1     147         2  failure  no
```

12 Inspecting Unique Values of Education Attribute

```
[12]: df["education"].unique()
```

Explanation

The `unique()` function is used on the `education` column to identify all distinct categories present in the dataset. This step helps in understanding the range of values for the attribute and is useful before encoding categorical variables into numerical format for Machine Learning models.

Output

The output is an array of unique values in the `education` column. For example: `['primary', 'secondary', 'tertiary', 'unknown']`, which shows all education levels present in the dataset.

```
[12]: array(['primary', 'secondary', 'tertiary', 'unknown'], dtype=object)
```

13 Encoding Education Attribute

```
[13]: df.education = df.education.map({
      'primary': 1,
      'secondary': 2,
      'tertiary': 3,
      'unknown': np.nan
    })
df.head(10)
```

Explanation

The education column contains categorical values representing the education level of customers. To prepare the data for Machine Learning models, these categorical values are converted to numerical values using the `map()` function. The mapping is as follows: 'primary' = 1, 'secondary' = 2, 'tertiary' = 3, and 'unknown' is replaced with NaN to indicate missing data. This transformation allows algorithms to process the education levels quantitatively.

Output

The output displays the first ten rows of the dataset after the transformation. The education column now contains numerical values (1, 2, 3) corresponding to education levels, with NaN for unknown entries.

```
[13]:   age  job  marital  education  default  balance  housing  loan  contact  \
0   30  0.0         1         1.0       no    1787         0     0  cellular
1   33  0.0         1         2.0       no    4789         1     1  cellular
2   35  1.0         0         3.0       no    1350         1     0  cellular
3   30  1.0         1         3.0       no    1476         1     1  unknown
4   59  0.0         1         2.0       no         0         1     0  unknown
5   35  1.0         0         3.0       no     747         0     0  cellular
6   36  0.0         1         3.0       no     307         1     0  cellular
7   39  1.0         1         2.0       no     147         1     0  cellular
8   41  1.0         1         3.0       no     221         1     0  unknown
9   43  0.0         1         1.0       no     -88         1     1  cellular

   day  month  duration  campaign  pdays  previous  poutcome  y
0   19     10         79         1      -1         0  unknown  no
1   11      5        220         1    339         4  failure  no
2   16      4        185         1    330         1  failure  no
3    3      6        199         4     -1         0  unknown  no
4    5      5        226         1     -1         0  unknown  no
5   23      2        141         2    176         3  failure  no
6   14      5        341         1    330         2   other  no
7    6      5        151         2     -1         0  unknown  no
8   14      5         57         2     -1         0  unknown  no
9   17      4        313         1    147         2  failure  no
```

14 Unique Values of the Outcome of Previous Marketing Campaign (poutcome)

```
[14]: df["poutcome"].unique()
```

Explanation

The `unique()` function is used to identify all distinct values present in the `poutcome` column of the dataset. This column represents the result of the previous marketing campaign for each customer. Determining unique values helps in understanding the different categories present and assists in further preprocessing or encoding steps required for Machine Learning.

Output

The output is an array of all unique values in the `poutcome` column, for example:

```
array(['unknown', 'failure', 'other', 'success'], dtype=object)
```

This shows that the column contains four distinct categories indicating the previous campaign outcome.

```
[14]: array(['unknown', 'failure', 'other', 'success'], dtype=object)
```

15 Encoding Poutcome Attribute

```
[15]: df.poutcome = df.poutcome.map({
    'unknown': np.nan,
    'failure': 1,
    'other': 2,
    'success': 3
})
df.head(10)
```

Explanation

The `poutcome` column represents the outcome of the previous marketing campaign and is a categorical variable. To make it compatible with Machine Learning algorithms, it is mapped to numerical values using the `map()` function. The mapping assigns `failure` to 1, `other` to 2, `success` to 3, and replaces `unknown` values with `NaN` to handle missing information. This encoding simplifies the dataset while preserving meaningful distinctions between campaign outcomes.

Output

The output displays the first ten rows of the dataset with the `poutcome` column transformed into numerical values. Entries that were `unknown` are now shown as `NaN`, while other outcomes are represented by 1, 2, or 3 accordingly.

```
[15]:   age  job  marital  education  default  balance  housing  loan  contact \
0   30  0.0         1         1.0       no    1787         0     0  cellular
1   33  0.0         1         2.0       no    4789         1     1  cellular
2   35  1.0         0         3.0       no    1350         1     0  cellular
```

3	30	1.0	1	3.0	no	1476	1	1	unknown
4	59	0.0	1	2.0	no	0	1	0	unknown
5	35	1.0	0	3.0	no	747	0	0	cellular
6	36	0.0	1	3.0	no	307	1	0	cellular
7	39	1.0	1	2.0	no	147	1	0	cellular
8	41	1.0	1	3.0	no	221	1	0	unknown
9	43	0.0	1	1.0	no	-88	1	1	cellular

	day	month	duration	campaign	pdays	previous	poutcome	y
0	19	NaN	79	1	-1	0	NaN	no
1	11	NaN	220	1	339	4	1.0	no
2	16	NaN	185	1	330	1	1.0	no
3	3	NaN	199	4	-1	0	NaN	no
4	5	NaN	226	1	-1	0	NaN	no
5	23	NaN	141	2	176	3	1.0	no
6	14	NaN	341	1	330	2	2.0	no
7	6	NaN	151	2	-1	0	NaN	no
8	14	NaN	57	2	-1	0	NaN	no
9	17	NaN	313	1	147	2	1.0	no

16 Normalizing the Balance Attribute

```
[16]: df["balance"] = df["balance"].apply(lambda v: (v - df["balance"].min()) /
      ↪ (df["balance"].max() - df["balance"].min()))
df.head(10)
```

Explanation

The balance column is a numerical feature representing the account balance of customers. To scale this feature between 0 and 1, min-max normalization is applied. The formula used is:

$$\text{normalized_value} = \frac{v - \min(\text{balance})}{\max(\text{balance}) - \min(\text{balance})}$$

where v is the original balance value. This transformation ensures that all values of balance lie within the range $[0, 1]$, improving the performance and convergence of many Machine Learning algorithms.

Output

The output displays the first ten rows of the dataset after normalization. The balance column values are now scaled between 0 and 1 while all other columns remain unchanged.

```
[16]:   age  job  marital  education  default  balance  housing  loan  contact \
0   30  0.0         1         1.0       no  0.068455         0         0  cellular
1   33  0.0         1         2.0       no  0.108750         1         1  cellular
2   35  1.0         0         3.0       no  0.062590         1         0  cellular
3   30  1.0         1         3.0       no  0.064281         1         1  unknown
4   59  0.0         1         2.0       no  0.044469         1         0  unknown
5   35  1.0         0         3.0       no  0.054496         0         0  cellular
6   36  0.0         1         3.0       no  0.048590         1         0  cellular
7   39  1.0         1         2.0       no  0.046442         1         0  cellular
```

```

8  41  1.0      1      3.0    no  0.047436      1    0  unknown
9  43  0.0      1      1.0    no  0.043288      1    1  cellular

```

```

   day  month  duration  campaign  pdays  previous  poutcome  y
0   19   NaN       79         1      -1         0       NaN  no
1   11   NaN      220         1     339         4       1.0  no
2   16   NaN      185         1     330         1       1.0  no
3    3   NaN      199         4      -1         0       NaN  no
4    5   NaN      226         1      -1         0       NaN  no
5   23   NaN      141         2     176         3       1.0  no
6   14   NaN      341         1     330         2       2.0  no
7    6   NaN      151         2      -1         0       NaN  no
8   14   NaN       57         2      -1         0       NaN  no
9   17   NaN      313         1     147         2       1.0  no

```

17 Normalization of pdays Attribute

```
[17]: df["pdays"] = df["pdays"].apply(lambda v: (v - df["pdays"].min())/
↳(df["pdays"].max() - df["pdays"].min()))
df.head(10)
```

Explanation

The pdays column, which represents the number of days since a client was last contacted, is normalized using the Min-Max scaling technique. This transformation scales all values to a range between 0 and 1, which helps improve the performance and convergence of Machine Learning algorithms.

Output

The first ten rows of the dataset are displayed. The pdays column now contains values scaled between 0 and 1, while the other columns remain unchanged.

```
[17]:   age  job  marital  education  default  balance  housing  loan  contact \
0   30  0.0      1      1.0      no  0.068455      0    0  cellular
1   33  0.0      1      2.0      no  0.108750      1    1  cellular
2   35  1.0      0      3.0      no  0.062590      1    0  cellular
3   30  1.0      1      3.0      no  0.064281      1    1  unknown
4   59  0.0      1      2.0      no  0.044469      1    0  unknown
5   35  1.0      0      3.0      no  0.054496      0    0  cellular
6   36  0.0      1      3.0      no  0.048590      1    0  cellular
7   39  1.0      1      2.0      no  0.046442      1    0  cellular
8   41  1.0      1      3.0      no  0.047436      1    0  unknown
9   43  0.0      1      1.0      no  0.043288      1    1  cellular

   day  month  duration  campaign  pdays  previous  poutcome  y
0   19   NaN       79         1  0.000000         0       NaN  no
1   11   NaN      220         1  0.389908         4       1.0  no
2   16   NaN      185         1  0.379587         1       1.0  no
3    3   NaN      199         4  0.000000         0       NaN  no

```

4	5	NaN	226	1	0.000000	0	NaN	no
5	23	NaN	141	2	0.202982	3	1.0	no
6	14	NaN	341	1	0.379587	2	2.0	no
7	6	NaN	151	2	0.000000	0	NaN	no
8	14	NaN	57	2	0.000000	0	NaN	no
9	17	NaN	313	1	0.169725	2	1.0	no

18 Feature Scaling using Min-Max Scaler

```
[18]: from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
df["duration"] = scaler.fit_transform(df[["duration"]])
df["pdays"] = scaler.fit_transform(df[["pdays"]])
df.head()
```

Explanation

Feature scaling is performed to normalize the numerical attributes `duration` and `pdays` into a fixed range, usually between 0 and 1, which helps in faster convergence and better performance of Machine Learning algorithms. The `MinMaxScaler` from `sklearn.preprocessing` is used. The `fit_transform()` method calculates the minimum and maximum values of each feature and scales all values accordingly. This ensures that the features contribute proportionally to the model training.

Output

The output displays the first five rows of the dataset where the `duration` and `pdays` columns have been scaled to values between 0 and 1, while all other columns remain unchanged.

```
[18]:   age  job  marital  education  default  balance  housing  loan  contact  \
0   30  0.0         1         1.0      no  0.068455      0     0  cellular
1   33  0.0         1         2.0      no  0.108750      1     1  cellular
2   35  1.0         0         3.0      no  0.062590      1     0  cellular
3   30  1.0         1         3.0      no  0.064281      1     1  unknown
4   59  0.0         1         2.0      no  0.044469      1     0  unknown

   day  month  duration  campaign  pdays  previous  poutcome  y
0   19   NaN  0.024826         1  0.000000         0     NaN  no
1   11   NaN  0.071500         1  0.389908         4     1.0  no
2   16   NaN  0.059914         1  0.379587         1     1.0  no
3    3   NaN  0.064548         4  0.000000         0     NaN  no
4    5   NaN  0.073486         1  0.000000         0     NaN  no
```