Nombre Y matricula

Andres Benjamin Antelis Moreno A01637683

Data management using Pandas

Data management is a crucial component to statistical analysis and data science work.

This notebook will show you how to import, view, undertand, and manage your data using the Pandas data processing library, i.e., the notebook will demonstrates how to read a dataset into Python, and obtain a basic understanding of its content.

Note that **Python** by itself is a general-purpose programming language and does not provide high-level data processing capabilities. The **Pandas** library was developed to meet this need. **Pandas** is the most popular Python library for data manipulation, and we will use it extensively in this course. **Pandas** provides high-performance, easy-to-use data structures and data analysis tools.

The main data structure that **Pandas** works with is called a **Data Frame**. This is a two-dimensional table of data in which the rows typically represent cases and the columns represent variables (e.g. data used in this tutorial). Pandas also has a one-dimensional data structure called a **Series** that we will encounter when accesing a single column of a Data Frame.

Pandas has a variety of functions named `read_xxx` for reading data in different formats. Right now we will focus on reading `csv` files, which stands for comma-separated values. However the other file formats include `excel`, `json`, and `sql`.

There are many other options to `read_csv` that are very useful. For example, you would use the option `sep='\t'` instead of the default `sep=','` if the fields of your data file are delimited by tabs instead of commas. See here for the full documentation for `read_csv`.

Acknowledgments

- The dataset used in this tutorial is from https://www.coursera.org/ from the course "Understanding and Visualizing Data with Python" by University of Michigan

Importing libraries

```
# Import the packages that we will be using
import pandas as pd
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Importing data

```
# Define where you are running the code: colab or local
RunInColab          = True     # (False: no  | True: yes)

# If running in colab:
if RunInColab:
    # Mount your google drive in google colab
    from google.colab import drive
    drive.mount('/content/drive')

    # Find location
    #!pwd
    #!ls
    #!ls "/content/drive/My Drive/Colab Notebooks/MachineLearningWithPython/"

    # Define path del proyecto
    Ruta            = "/content/drive/MyDrive/!Tec stuff/!Uni/Semestre 2/Semana Tec1/TC1002S/NotebooksProfessor/datasets/cartwheel/cartwheel.

else:
    # Define path del proyecto
    Ruta            = ""
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
# url string that hosts our .csv file
Dataset="/content/drive/MyDrive/!Tec stuff/!Uni/Semestre 2/Semana Tec1/TC1002S/NotebooksProfessor/datasets/cartwheel/cartwheel.csv"

# Read the .csv file and store it as a pandas Data Frame

db=pd.read_csv(Dataset)
```

If we want to print the information about th output object type we would simply type the following: type(df)

```
db
```

	ID	Age	Gender	GenderGroup	Glasses	GlassesGroup	Height	Wingspan	CWDista
0	1	56.0	F	1	Y	1	62.00	61.0	
1	2	26.0	F	1	Y	1	62.00	60.0	
2	3	33.0	F	1	Y	1	66.00	64.0	
3	4	39.0	F	1	N	0	64.00	63.0	
4	5	27.0	M	2	N	0	73.00	75.0	
5	6	24.0	M	2	N	0	75.00	71.0	
6	7	28.0	M	2	N	0	75.00	76.0	
7	8	22.0	F	1	N	0	65.00	62.0	
8	9	29.0	M	2	Y	1	74.00	73.0	
9	10	33.0	F	1	Y	1	63.00	60.0	
10	11	30.0	M	2	Y	1	69.50	66.0	
11	12	28.0	F	1	Y	1	62.75	58.0	
12	13	25.0	F	1	Y	1	65.00	64.5	
13	14	23.0	F	1	N	0	61.50	57.5	

```
len(db.index)
```

```
52
```

```
print(len(db.columns))
```

```
12
```

19	20	24.0	F	1	Y	1	68.00	66.0	

```
db.head(11)
```

	ID	Age	Gender	GenderGroup	Glasses	GlassesGroup	Height	Wingspan	CWDista
0	1	56.0	F	1	Y	1	62.0	61.0	
1	2	26.0	F	1	Y	1	62.0	60.0	
2	3	33.0	F	1	Y	1	66.0	64.0	
3	4	39.0	F	1	N	0	64.0	63.0	
4	5	27.0	M	2	N	0	73.0	75.0	
5	6	24.0	M	2	N	0	75.0	71.0	
6	7	28.0	M	2	N	0	75.0	76.0	
7	8	22.0	F	1	N	0	65.0	62.0	
8	9	29.0	M	2	Y	1	74.0	73.0	
9	10	33.0	F	1	Y	1	63.0	60.0	
10	11	30.0	M	2	Y	1	69.5	66.0	

```
db.tail(11)
```

	ID	Age	Gender	GenderGroup	Glasses	GlassesGroup	Height	Wingspan	CWDista
41	42	26.0	M	2	Y	1	73.5	72.0	

```
db.columns
```

```
Index(['ID', 'Age', 'Gender', 'GenderGroup', 'Glasses', 'GlassesGroup',
       'Height', 'Wingspan', 'CWDistance', 'Complete', 'CompleteGroup',
       'Score'],
      dtype='object')
```

46	47	27.0	M	2	N	0	78.0	75.0	

```
db.dtypes
```

```
ID                int64
Age             float64
Gender           object
GenderGroup       int64
Glasses          object
GlassesGroup      int64
Height          float64
Wingspan        float64
CWDistance        int64
Complete         object
CompleteGroup   float64
Score             int64
dtype: object
```

Exploring the content of the data set

Use the `shape` method to determine the numbers of rows and columns in a data frame. This can be used to confirm that we have actually obtained the data the we are expecting.

Based on what we see below, the data set being read here has N_r rows, corresponding to N_r observations, and N_c columns, corresponding to N_c variables in this particular data file.

If we want to show the entire data frame we would simply write the following:

As you can see, we have a 2-Dimensional object where each row is an independent observation and each coloum is a variable.

Now, use the the `head()` function to show the first 5 rows of our data frame

Also, you can use the the `tail()` function to show the last 5 rows of our data frame

The columns in a Pandas data frame have names, to see the names, use the `columns` method:

To gather more information regarding the data, we can view the column names with the following function:

Be aware that every variable in a Pandas data frame has a data type. There are many different data types, but most commonly you will encounter floating point values (real numbers), integers, strings (text), and date/time values. When Pandas reads a text/csv file, it guesses the

data types based on what it sees in the first few rows of the data file. Usually it selects an appropriate type, but occasionally it does not. To confirm that the data types are consistent with what the variables represent, inspect the `dtypes` attribute of the data frame.

Summary statistics, which include things like the mean, min, and max of the data, can be useful to get a feel for how large some of the variables are and what variables may be the most important.

```
# Summary statistics for the quantitative variables
db.describe()
```

	ID	Age	GenderGroup	GlassesGroup	Height	Wingspan	CWDist
count	52.000000	51.000000	52.000000	52.000000	51.000000	51.000000	52.00
mean	26.500000	28.411765	1.500000	0.500000	68.971569	67.313725	85.57
std	15.154757	5.755611	0.504878	0.504878	5.303812	5.624021	14.35
min	1.000000	22.000000	1.000000	0.000000	61.500000	57.500000	63.00
25%	13.750000	25.000000	1.000000	0.000000	64.500000	63.000000	72.00
50%	26.500000	27.000000	1.500000	0.500000	69.000000	66.000000	85.00
75%	39.250000	30.000000	2.000000	1.000000	73.000000	72.000000	96.50
max	52.000000	56.000000	2.000000	1.000000	79.500000	76.000000	115.00

```
# Drop observations with NaN values
db.Age.dropna().describe()
#df.Wingspan.dropna().describe()
```

```
count    51.000000
mean     28.411765
std       5.755611
min      22.000000
25%      25.000000
50%      27.000000
75%      30.000000
max      56.000000
Name: Age, dtype: float64
```

It is also possible to get statistics on the entire data frame or a column as follows

- `df.mean()` Returns the mean of all columns
- `df.corr()` Returns the correlation between columns in a data frame
- `df.count()` Returns the number of non-null values in each data frame column
- `df.max()` Returns the highest value in each column
- `df.min()` Returns the lowest value in each column
- `df.median()` Returns the median of each column
- `df.std()` Returns the standard deviation of each column

```
db.std()
```

```
<ipython-input-16-c6ec16309607>:1: FutureWarning: Dropping of nuisance columns in DataFrame reductions (with 'numeric_only=None') is dep
  db.std()
ID               15.154757
Age               5.755611
GenderGroup       0.504878
GlassesGroup      0.504878
Height            5.303812
Wingspan          5.624021
CWDistance       14.353173
CompleteGroup     0.367290
Score             2.211566
dtype: float64
```

How to write a data frame to a File

To save a file with your data simply use the `to_csv` attribute

Examples:

- df.to_csv('myDataFrame.csv')
- df.to_csv('myDataFrame.csv', sep='\t')

Rename columns

To change the name of a colum use the `rename` attribute

Example:

df = df.rename(columns={"Age": "Edad"})

df.head()

```
df = db.rename(columns={"Age": "Edad"})
df.head()
```

	ID	Edad	Gender	GenderGroup	Glasses	GlassesGroup	Height	Wingspan	CWDistar
0	1	56.0	F	1	Y	1	62.0	61.0	
1	2	26.0	F	1	Y	1	62.0	60.0	
2	3	33.0	F	1	Y	1	66.0	64.0	
3	4	39.0	F	1	N	0	64.0	63.0	
4	5	27.0	M	2	N	0	73.0	75.0	

```
# Back to the original name
```

Selection of colums

As discussed above, a Pandas data frame is a rectangular data table, in which the rows represent observations or samples and the columns represent variables. One common manipulation of a data frame is to extract the data for one case or for one variable. There are several ways to do this, as shown below.

To extract all the values for one column (variable), use one of the following alternatives.

```
a = db.Age
b = db["Age"]
c = db.loc[:, "Age"]
d = db.iloc[:, 1]

#df[["Gender", "GenderGroup"]]
```

```
c
```

```
0     56.0
1     26.0
2     33.0
3     39.0
4     27.0
5     24.0
6     28.0
7     22.0
8     29.0
9     33.0
10    30.0
11    28.0
12    25.0
13    23.0
14    31.0
15    26.0
16    26.0
```

```
17    27.0
18    23.0
19    24.0
20    23.0
21    29.0
22    25.0
23    26.0
24    23.0
25    28.0
26    24.0
27    25.0
28    32.0
29    38.0
30    27.0
31    33.0
32    38.0
33    27.0
34    24.0
35    27.0
36    25.0
37    26.0
38    31.0
39    30.0
40    23.0
41    26.0
42    28.0
43    26.0
44    30.0
45    39.0
46    27.0
47    24.0
48    28.0
49    30.0
50     NaN
51    27.0
Name: Age, dtype: float64
```

Slicing a data set

As discussed above, a Pandas data frame is a rectangular data table, in which the rows represent cases and the columns represent variables. One common manipulation of a data frame is to extract the data for one observation or for one variable. There are several ways to do this, as shown below.

Lets say we would like to splice our data frame and select only specific portions of our data. There are three different ways of doing so.

1. .loc()
2. .iloc()
3. .ix()

We will cover the .loc() and .iloc() splicing functions.

The attibute **.loc()** uses labels/column names, in specific, it takes two single/list/range operator separated by ',', the first one indicates the rows and the second one indicates columns.

```
# Return all observations of CWDistance
db.loc[0:4,"CWDistance"]

# Return a subset of observations of CWDistance
db.loc[:9, "CWDistance"]

# Select all rows for multiple columns, ["Gender", "GenderGroup"]
db.loc[:,["Gender", "GenderGroup"]]

# Select multiple columns, ["Gender", "GenderGroup"]me
keep = ['Gender', 'GenderGroup']
db_gender = df[keep]
db_gender
# Select few rows for multiple columns, ["CWDistance", "Height", "Wingspan"]
db.loc[4:9, ["CWDistance", "Height", "Wingspan"]]

# Select range of rows for all columns
#df.loc[10:15,:]
```

	CWDistance	Height	Wingspan
4	72	73.0	75.0
5	81	75.0	71.0
6	107	75.0	76.0
7	98	65.0	62.0
8	106	74.0	73.0
9	65	63.0	60.0

The attribute **iloc()** is an integer based slicing.

```
# Prints all the values of the first four set of variables
db.iloc[:, :4]

# Prints the first four set of values of all variables
db.iloc[:4, :]

# Prints all values of the sets 3 until 6
db.iloc[:, 3:7]

# Prints values 4 until 8 from sets 2,3
db.iloc[4:8, 2:4]

# This is incorrect: We had to use the "loc" instead of iloc
db.loc[1:5, ["Gender", "GenderGroup"]]
```

	Gender	GenderGroup
1	F	1
2	F	1
3	F	1
4	M	2
5	M	2

Get unique existing values

List unique values in the one of the columns

df.Gender.unique()

```
# List unique values in the df['Gender'] column
db.Gender.unique()
```

```
array(['F', 'M'], dtype=object)
```

```
# Lets explore df["GenderGroup] as well
db.GenderGroup.unique()
```

```
array([1, 2])
```

Filter, Sort and Groupby

With **Filter** you can use different conditions to filter columns. For example, df[df[year] > 1984] would give you only the column year is greater than 1984. You can use & (and) or | (or) to add different conditions to your filtering. This is also called boolean filtering.

df[df["Height"] >= 70]

With **Sort** is possible to sort values in a certain column in an ascending order using `df.sort_values("ColumnName")` or in descending order using `df.sort_values(ColumnName, ascending=False)`.

Furthermore, it's possible to sort values by Column1Name in ascending order then Column2Name in descending order by using `df.sort_values([Column1Name,Column2Name],ascending=[True,False])`

df.sort_values("Height")

df.sort_values("Height",ascending=False)

The attribute **Groupby** involves splitting the data into groups based on some criteria, applying a function to each group independently and combining the results into a data structure. df.groupby(col) returns a groupby object for values from one column while df.groupby([col1,col2]) returns a groupby object for values from multiple columns.

df.groupby(['Gender'])

Size of each group

df.groupby(['Gender']).size()

df.groupby(['Gender','GenderGroup']).size()

This output indicates that we have two types of combinations.

- Case 1: Gender = F & Gender Group = 1
- Case 2: Gender = M & GenderGroup = 2.

This validates our initial assumption that these two fields essentially portray the same information.

Data Cleaning: handle with missing data

Before getting started to work with your data, it's a good practice to observe it thoroughly to identify missing values and handle them accordingly.

When reading a dataset using Pandas, there is a set of values including 'NA', 'NULL', and 'NaN' that are taken by default to represent a missing value. The full list of default missing value codes is in the '`read_csv`' documentation here. This document also explains how to change the way that '`read_csv`' decides whether a variable's value is missing.

Pandas has functions called `isnull` and `notnull` that can be used to identify where the missing and non-missing values are located in a data frame.

Below we use these functions to count the number of missing and non-missing values in each variable of the datasetr.

Unfortunately, our output indicates that some of our columns contain missing values so we are no able to continue on doing analysis with those colums

```
#df.isnull().sum()
#df.notnull().sum()
```

Now we use these functions to count the number of missing and non-missing values in a single variable in the dataset

print(df.Height.notnull().sum())

print(pd.isnull(df.Height).sum())

```
# Extract all non-missing values of one of the columns into a new variable
#x = df.Age.dropna().describe()
#x.describe()
```

Add and eliminate columns

In some cases it is useful to create or eiminate new columns

```
#df.head()


# Add a new column with new data

# Create a column data
NewColumnData = db.Age/db.Age

# Insert that column in the data frame
db.insert(12, "ColumnInserted", NewColumnData, True)

#df.head()


# # Eliminate inserted column
# df.drop("ColumnInserted", axis=1, inplace = True)
# #df.drop(columns=['ColumnInserted'], inplace = True)
# # Remove three columns as index base
# #df.drop(df.columns[[12]], axis = 1, inplace = True)
#
# df.head()


# # Add new column derived from existing columns
#
# # The new column is a function of another column
# df["AgeInMonths"] = df["Age"] * 12
#
# df.head()


# # Eliminate inserted column
# df.drop("AgeInMonths", axis=1, inplace = True)
#
# df.head()


# Add a new column with text labels reflecting the code's meaning

# df["GenderGroupNew"] = df.GenderGroup.replace({1: "Female", 2: "Male"})

# Show the first 5 rows of the created data frame



## Eliminate inserted column
# df.drop("GenderGroupNew", axis=1, inplace = True)
##df.drop(['GenderGroupNew'],vaxis='columns',vinplace=True)




## Add a new column with strata based on these cut points
#
## Create a column data
#NewColumnData = df.Age/df.Age
#
## Insert that column in the data frame
#df.insert(1, "ColumnStrata", NewColumnData, True)
#
#df["ColumnStrata"] = pd.cut(df.Height, [60., 63., 66., 69., 72., 75., 78.])
#
## Show the first 5 rows of the created data frame
#df.head()
```

```
## Eliminate inserted column
#df.drop("ColumnStrata", axis=1, inplace = True)
#
#df.head()
```

```
# Drop several "unused" columns
#vars = ["ID", "GenderGroup", "GlassesGroup", "CompleteGroup"]
#df.drop(vars, axis=1, inplace = True)
```

Add and eliminate rows

In some cases it is requiered to add new observations (rows) to the data set

```
# Print tail
df.tail()
```

	ID	Edad	Gender	GenderGroup	Glasses	GlassesGroup	Height	Wingspan	CWDista
47	48	24.0	M	2	N	0	79.5	75.0	
48	49	28.0	M	2	N	0	77.8	76.0	
49	50	30.0	F	1	N	0	74.6	NaN	
50	51	NaN	M	2	N	0	71.0	70.0	
51	52	27.0	M	2	N	0	NaN	71.5	

```
df.loc[len(df.index)] = [26, 24, 'F', 1, 'Y', 1, 66, 'NaN', 68, 'N', 0, 3]

df.tail()
```

	ID	Edad	Gender	GenderGroup	Glasses	GlassesGroup	Height	Wingspan	CWDista
48	49	28.0	M	2	N	0	77.8	76.0	
49	50	30.0	F	1	N	0	74.6	NaN	
50	51	NaN	M	2	N	0	71.0	70.0	
51	52	27.0	M	2	N	0	NaN	71.5	
52	26	24.0	F	1	Y	1	66.0	NaN	

```
# Eliminate inserted row
df.drop([20], inplace = True )

df.tail()
```

	ID	Edad	Gender	GenderGroup	Glasses	GlassesGroup	Height	Wingspan	CWDista
48	49	28.0	M	2	N	0	77.8	76.0	
49	50	30.0	F	1	N	0	74.6	NaN	
50	51	NaN	M	2	N	0	71.0	70.0	
51	52	27.0	M	2	N	0	NaN	71.5	
52	26	24.0	F	1	Y	1	66.0	NaN	

Cleaning your data: drop out unused columns and/or drop out rows with any missing values

```
# Drop unused columns
vars = ["ID", "GenderGroup", "GlassesGroup", "CompleteGroup"] #Dejar estar variables
#db.drop(vars, axis=1, inplace = True)


vars = ["Age", "Gender", "Glasses", "Height", "Wingspan", "CWDistance", "Complete", "Score"] #Quedarnos con estas variables
df = db[vars]

# Drop rows with any missing values
df = df.dropna()

# Drop unused columns and drop rows with any missing values
vars = ["Age", "Gender", "Glasses", "Height", "Wingspan", "CWDistance", "Complete", "Score"]
dc = df[vars].dropna()

dc
```

	Age	Gender	Glasses	Height	Wingspan	CWDistance	Complete	Score
0	56.0	F	Y	62.00	61.0	79	Y	7
1	26.0	F	Y	62.00	60.0	70	Y	8
2	33.0	F	Y	66.00	64.0	85	Y	7
3	39.0	F	N	64.00	63.0	87	Y	10
4	27.0	M	N	73.00	75.0	72	N	4
5	24.0	M	N	75.00	71.0	81	N	3
6	28.0	M	N	75.00	76.0	107	Y	10
7	22.0	F	N	65.00	62.0	98	Y	9
8	29.0	M	Y	74.00	73.0	106	N	5
9	33.0	F	Y	63.00	60.0	65	Y	8
10	30.0	M	Y	69.50	66.0	96	Y	6
11	28.0	F	Y	62.75	58.0	79	Y	10
12	25.0	F	Y	65.00	64.5	92	Y	6
13	23.0	F	N	61.50	57.5	66	Y	4

```
dc.shape
```

```
(49, 8)
```

```
db.isnull().sum()
```

```
ID                0
Age               1
Gender            0
GenderGroup       0
Glasses           0
GlassesGroup      0
Height            1
Wingspan          1
CWDistance        0
Complete          0
CompleteGroup     1
Score             0
ColumnInserted    1
dtype: int64
```

```
x=db.Age.dropna().describe()
x
```

```
count    51.000000
mean     28.411765
std       5.755611
min      22.000000
25%      25.000000
50%      27.000000
75%      30.000000
max      56.000000
Name: Age, dtype: float64
```

Final remarks

- The understanding of your dataset is essential
  - Number of observations
  - Variables
  - Data types: numerical or categorial
  - What are my variables of interest
- There are several ways to do the same thing
- Cleaning your dataset (dropping out rows with any missing values) is a good practice
- The **Pandas** library provides fancy, high-performance, easy-to-use data structures and data analysis tools

Activity: work with the iris dataset

Repeat this tutorial with the iris data set and respond to the following inquiries

1. Calculate the statistical summary for each quantitative variables. Explain the results
    - Identify the name of each column
    - Identify the type of each column
    - Minimum, maximum, mean, average, median, standar deviation
2. Are there missing data? If so, create a new dataset containing only the rows with the non-missing data
3. Create a new dataset containing only the petal width and length and the type of Flower
4. Create a new dataset containing only the setal width and length and the type of Flower
5. Create a new dataset containing the setal width and length and the type of Flower encoded as a categorical numerical column

```
Iris= "/content/drive/MyDrive/!Tec stuff/!Uni/Semestre 2/Semana Tec1/TC1002S/NotebooksProfessor/datasets/iris/iris.csv"
```

```
df=pd.read_csv(Iris)
```

```
df.head()
```

	5.1	3.5	1.4	0.2	Iris-setosa
0	4.9	3.0	1.4	0.2	Iris-setosa
1	4.7	3.2	1.3	0.2	Iris-setosa
2	4.6	3.1	1.5	0.2	Iris-setosa
3	5.0	3.6	1.4	0.2	Iris-setosa
4	5.4	3.9	1.7	0.4	Iris-setosa

```
df1 = df.rename(columns={"5.1": "sepal length in cm"})
df2 = df1.rename(columns={"3.5": "sepal width in cm"})
df3 = df2.rename(columns={"1.4": "petal length in cm"})
df4 = df3.rename(columns={"0.2": "petal width in cm"})
df5 = df4.rename(columns={"Iris-setosa": "Class, Iris type"})
```

```
df5.head()
```

```
df5.dtypes
```

```
df5.describe()
```

```
df5.max()
```

```
df5.min()
```

```
df5.mean()
```

```
df5.median()
```

```
df5.std()
```

```
df5.isnull()
```

```
df5.notnull()
```

```
df5.isnull().sum()
df5.notnull().sum()
```

```
Column1        149
Column2        149
Column3        149
Column4        149
Flower type    149
dtype: int64
```

As we can see there are no values missing

Create a new dataset containing only the petal width and length and the type of Flower

```
# Eliminate inserted column
df5.drop("sepal length in cm", axis=1, inplace = True)
df5.drop("sepal width in cm", axis=1, inplace = True)
df5.drop("petal length in cm", axis=1, inplace = True)
df5.head()
```

	width in cm	Class, Iris type
0	0.2	Iris-setosa
1	0.2	Iris-setosa
2	0.2	Iris-setosa
3	0.2	Iris-setosa
4	0.4	Iris-setosa

Create a new dataset containing only the setal width and length and the type of Flower

```
df=pd.read_csv(Iris)
df1 = df.rename(columns={"5.1": "sepal length in cm"})
df2 = df1.rename(columns={"3.5": "sepal width in cm"})
df3 = df2.rename(columns={"1.4": "petal length in cm"})
df4 = df3.rename(columns={"0.2": "petal width in cm"})
df5 = df4.rename(columns={"Iris-setosa": "Class, Iris type"})
df5.head()
```

```
# Eliminate inserted column
df5.drop("sepal length in cm", axis=1, inplace = True)
df5.drop("petal width in cm", axis=1, inplace = True)
df5.drop("petal length in cm", axis=1, inplace = True)
df5.head()
```

	sepal width in cm	Class, Iris type
0	3.0	Iris-setosa
1	3.2	Iris-setosa
2	3.1	Iris-setosa
3	3.6	Iris-setosa
4	3.9	Iris-setosa

Create a new dataset containing the setal width and length and the type of Flower encoded as a categorical numerical column

```
df=pd.read_csv(Iris)
df1 = df.rename(columns={"5.1": "sepal length in cm"})
df2 = df1.rename(columns={"3.5": "sepal width in cm"})
df3 = df2.rename(columns={"1.4": "petal length in cm"})
df4 = df3.rename(columns={"0.2": "petal width in cm"})
df5 = df4.rename(columns={"Iris-setosa": "Class, Iris type"})
df5.head()
```

	sepal length in cm	sepal width in cm	petal length in cm	petal width in cm	Class, Iris type
0	4.9	3.0	1.4	0.2	Iris-setosa
1	4.7	3.2	1.3	0.2	Iris-setosa
2	4.6	3.1	1.5	0.2	Iris-setosa
3	5.0	3.6	1.4	0.2	Iris-setosa
4	5.4	3.9	1.7	0.4	Iris-setosa

```
df5.dtypes
```

```
sepal length in cm    float64
sepal width in cm     float64
petal length in cm    float64
petal width in cm     float64
Class, Iris type       object
dtype: object
```

```
obj_df = df5.select_dtypes(include=['object']).copy()
obj_df
```

	Class, Iris type
0	Iris-setosa
1	Iris-setosa
2	Iris-setosa
3	Iris-setosa
4	Iris-setosa
...	...
144	Iris-virginica
145	Iris-virginica
146	Iris-virginica
147	Iris-virginica
148	Iris-virginica

149 rows × 1 columns

```
obj_df["Class, Iris type"].value_counts()
```

```
Iris-versicolor    50
Iris-virginica     50
Iris-setosa        49
Name: Class, Iris type, dtype: int64
```

```
cleanup_nums = {"Class, Iris type":     {"Iris-setosa": 1, "Iris-versicolour": 2, "Iris-virginica":3}}
```

```
obj_df = obj_df.replace(cleanup_nums)
obj_df
```