Lucas Wong Mang A01639032

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
import matplotlib.pyplot as plt
```

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/My·Drive/Herramientas·Computacionales/"

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

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

```
# url string that hosts our .csv file
Ruta_General = "/content/drive/My Drive/Herramientas Computacionales/"
url = "iris.csv"
url_iris=Ruta_General+url
# Read the .csv file and store it as a pandas Data Frame
header = ['s_length', 's_width', 'p_length', 'p_width', 'Class']
ds_iris = pd.read_csv(url_iris, names=header)
```

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

```
type(ds_iris)
```

```
pandas.core.frame.DataFrame
```

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.

```
ds_iris.shape
```

```
(150, 5)
```

```
R=ds_iris.shape[0]
V=ds_iris.shape[1]

print("# of Data: "+str(R))
print("# of Vars: "+str(V))
```

```
# of Data: 150
# of Vars: 5
```

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

```
ds_iris
```

	s_length	s_width	p_length	p_width	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 5 columns

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

```
ds_iris.head()
```

	s_length	s_width	p_length	p_width	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

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

```
ds_iris.tail()
```

	s_length	s_width	p_length	p_width	class
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

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:

```
ds_iris.columns
```

```
Index(['s_length', 's_width', 'p_length', 'p_width', 'class'], dtype='object')
```

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.

```
ds_iris.dtypes
```

```
s_length    float64
s_width     float64
p_length    float64
p_width     float64
class        object
dtype: object
```

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
ds_iris.describe()
```

	s_length	s_width	p_length	p_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

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

```
print(ds_iris.mean())
print(ds_iris.corr())
print(ds_iris.count())
print(ds_iris.max())
print(ds_iris.min())
print(ds_iris.median())
print(ds_iris.std())
```

```
s_length    5.843333
s_width     3.057333
p_length    3.758000
p_width     1.199333
dtype: float64
             s_length   s_width  p_length   p_width
s_length     1.000000 -0.117570  0.871754  0.817941
s_width     -0.117570  1.000000 -0.428440 -0.366126
p_length     0.871754 -0.428440  1.000000  0.962865
p_width      0.817941 -0.366126  0.962865  1.000000
s_length    150
s_width     150
p_length    150
p_width     150
class       150
dtype: int64
s_length               7.9
s_width                4.4
p_length               6.9
p_width                2.5
class       Iris-virginica
dtype: object
s_length            4.3
s_width             2.0
p_length            1.0
p_width             0.1
class       Iris-setosa
dtype: object
s_length    5.80
s_width     3.00
p_length    4.35
p_width     1.30
dtype: float64
s_length    0.828066
s_width     0.435866
p_length    1.765298
p_width     0.762238
dtype: float64
<ipython-input-19-78906132eb50>:1: FutureWarning: Dropping of nuisance columns in DataFrame reductions (with 'numeric_on]
  print(ds_iris.mean())
<ipython-input-19-78906132eb50>:6: FutureWarning: Dropping of nuisance columns in DataFrame reductions (with 'numeric_on]
  print(ds_iris.median())
<ipython-input-19-78906132eb50>:7: FutureWarning: Dropping of nuisance columns in DataFrame reductions (with 'numeric_on]
  print(ds_iris.std())
```

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')

```
ds_iris.to_csv('myDataFrame.csv')
```

Rename columns

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

Example:

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

df.head()

```
ds_iris=ds_iris.rename(columns={4:"Especie"})
ds_iris.head()
```

	s_length	s_width	p_length	p_width	Class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

```
# 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 = ds_iris.Class
b = ds_iris["Class"]
c = ds_iris.loc[:, "Class"]
d = ds_iris.iloc[:, 4]

print(d)

ds_iris[["s_length", "s_width"]]
```

```
0         Iris-setosa
1         Iris-setosa
2         Iris-setosa
3         Iris-setosa
4         Iris-setosa
            ...
145    Iris-virginica
146    Iris-virginica
147    Iris-virginica
148    Iris-virginica
149    Iris-virginica
Name: Class, Length: 150, dtype: object
```

	s_length	s_width
0	5.1	3.5
1	4.9	3.0
2	4.7	3.2
3	4.6	3.1
4	5.0	3.6
...	...	...
145	6.7	3.0
146	6.3	2.5
147	6.5	3.0
148	6.2	3.4
149	5.9	3.0

150 rows × 2 columns

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
ds_iris.loc[:,"Class"]
```

```
# Return a subset of observations of CWDistance
ds_iris.loc[:9, "Class"]

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

# Select multiple columns, ["Gender", "GenderGroup"]me
#keep = ['Gender', 'GenderGroup']
#df_gender = df[keep]

# Select few rows for multiple columns, ["CWDistance", "Height", "Wingspan"]
#df.loc[4:9, ["CWDistance", "Height", "Wingspan"]]

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

```
0    Iris-setosa
1    Iris-setosa
2    Iris-setosa
3    Iris-setosa
4    Iris-setosa
5    Iris-setosa
6    Iris-setosa
7    Iris-setosa
8    Iris-setosa
9    Iris-setosa
Name: Class, dtype: object
```

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

```
# .
ds_iris.iloc[:, :4]

# .
#df.iloc[:4, :]

# .
ds_iris.iloc[:, 3:7]

# .
#df.iloc[4:8, 2:4]

# This is incorrect:
#df.iloc[1:5, ["Gender", "GenderGroup"]]
```

	p_width	Class
0	0.2	Iris-setosa
1	0.2	Iris-setosa
2	0.2	Iris-setosa
3	0.2	Iris-setosa
4	0.2	Iris-setosa
...	...	...
145	2.3	Iris-virginica
146	1.9	Iris-virginica
147	2.0	Iris-virginica
148	2.3	Iris-virginica
149	1.8	Iris-virginica

150 rows × 2 columns

Get unique existing values

List unique values in the one of the columns

df.Gender.unique()

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

```
array(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'], dtype=object)
```

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]

```
ds_iris[ds_iris['s_width']>=3.1]
```

	s_length	s_width	p_length	p_width	Class
0	5.1	3.5	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
5	5.4	3.9	1.7	0.4	Iris-setosa
...	...	...	...	...	...
140	6.7	3.1	5.6	2.4	Iris-virginica
141	6.9	3.1	5.1	2.3	Iris-virginica
143	6.8	3.2	5.9	2.3	Iris-virginica
144	6.7	3.3	5.7	2.5	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica

67 rows × 5 columns

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)

```
ds_iris.sort_values("s_length")
```

	s_length	s_width	p_length	p_width	Class
13	4.3	3.0	1.1	0.1	Iris-setosa
42	4.4	3.2	1.3	0.2	Iris-setosa
38	4.4	3.0	1.3	0.2	Iris-setosa
8	4.4	2.9	1.4	0.2	Iris-setosa
41	4.5	2.3	1.3	0.3	Iris-setosa
...	...	...	...	...	...
122	7.7	2.8	6.7	2.0	Iris-virginica
118	7.7	2.6	6.9	2.3	Iris-virginica
117	7.7	3.8	6.7	2.2	Iris-virginica
135	7.7	3.0	6.1	2.3	Iris-virginica
131	7.9	3.8	6.4	2.0	Iris-virginica

150 rows × 5 columns

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'])

```
ds_iris.groupby(["Class"]).first()
```

Class	s_length	s_width	p_length	p_width
Iris-setosa	5.1	3.5	1.4	0.2
Iris-versicolor	7.0	3.2	4.7	1.4
Iris-virginica	6.3	3.3	6.0	2.5

Size of each group

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

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

```
ds_iris.groupby(["Class"]).size()
```

```
Class
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
dtype: int64
```

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.

```
ds_iris.isnull().sum()
```

```
s_length    0
s_width     0
p_length    0
p_width     0
Class       0
dtype: int64
```

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

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())

```
ds_iris.Class.notnull().sum()
```

```
150
```

```
# 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

```
ds_iris.head()
```

	s_length	s_width	p_length	p_width	Class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

```
# Add a new column with new data

# Create a column data
NewColumnData = ds_iris.Class

# Insert that column in the data frame
ds_iris.insert(5, "Test", NewColumnData, True)

ds_iris.head()
```

	s_length	s_width	p_length	p_width	Class	Test
0	5.1	3.5	1.4	0.2	Iris-setosa	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa	Iris-setosa

```
# # Eliminate inserted column
ds_iris.drop("Test", 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)
#
ds_iris.head()
```

	s_length	s_width	p_length	p_width	Class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

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

	s_length	s_width	p_length	p_width	Class	p_width*12
0	5.1	3.5	1.4	0.2	Iris-setosa	2.4
1	4.9	3.0	1.4	0.2	Iris-setosa	2.4
2	4.7	3.2	1.3	0.2	Iris-setosa	2.4
3	4.6	3.1	1.5	0.2	Iris-setosa	2.4
4	5.0	3.6	1.4	0.2	Iris-setosa	2.4

```
# # Eliminate inserted column
ds_iris.drop("p_width*12", axis=1, inplace = True)
```

```
ds_iris.head()
```

	s_length	s_width	p_length	p_width	Class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

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




## Eliminate inserted row
#df.drop([28], inplace = True )
#
#df.tail()
```

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

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

#vars = ["Age", "Gender", "Glasses", "Height", "Wingspan", "CWDistance", "Complete", "Score"]
#df = df[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"]
#df = df[vars].dropna()

#df
```

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

```
# 1 -- Calculate
ds_irisSum=ds_iris.iloc[:,0:4].copy()
ds_irisSum.describe()
```

```
# 2 -- Missing data?
ds_iris.dropna()
```

	s_length	s_width	p_length	p_width	Class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 5 columns

```
# 3 -- New Dataset 1
ds_iris[["Class", "p_width", "p_length"]]
```

	Class	p_width	p_length
0	Iris-setosa	0.2	1.4
1	Iris-setosa	0.2	1.4
2	Iris-setosa	0.2	1.3
3	Iris-setosa	0.2	1.5
4	Iris-setosa	0.2	1.4
...	...	...	...
145	Iris-virginica	2.3	5.2
146	Iris-virginica	1.9	5.0
147	Iris-virginica	2.0	5.2
148	Iris-virginica	2.3	5.4
149	Iris-virginica	1.8	5.1

150 rows × 3 columns

```
# 4 -- New Dataset 2
ds_iris[["Class", "s_width", "s_length"]]
```

	Class	s_width	s_length
0	Iris-setosa	3.5	5.1
1	Iris-setosa	3.0	4.9
2	Iris-setosa	3.2	4.7
3	Iris-setosa	3.1	4.6
4	Iris-setosa	3.6	5.0
...	...	...	...
145	Iris-virginica	3.0	6.7
146	Iris-virginica	2.5	6.3
147	Iris-virginica	3.0	6.5
148	Iris-virginica	3.4	6.2
149	Iris-virginica	3.0	5.9

150 rows × 3 columns

```
# 5 -- New Dataset 3
ds_iris[["Class", "p_width", "p_length"]]
New_ds_iris=ds_iris
```

```
New_ds_iris["Class"]=New_ds_iris["Class"].replace({"Iris-setosa":0,"Iris-virginica":1,"Iris-versicolo
New_ds_iris
```

	s_length	s_width	p_length	p_width	Class
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	1
146	6.3	2.5	5.0	1.9	1
147	6.5	3.0	5.2	2.0	1
148	6.2	3.4	5.4	2.3	1
149	5.9	3.0	5.1	1.8	1

150 rows × 5 columns

✓ 0s completed at 10:41 PM