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

In [65]:
```
# Import the packages that we will be using
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

In [66]:
```
from sklearn import datasets
iris = datasets.load_iris()
```

In [67]:
```
# Define the col names for the iris dataset
col_names = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'Flower']
```

```
# Dataset url
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"

# Load the dataset from URL
iris_df = pd.read_csv(url, header=None, names=col_names)
print(iris_df.head())
```

```
   sepal_length  sepal_width  petal_length  petal_width       Flower
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
```

Importing data

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

In [68]:
```
type(iris_df)
```

Out[68]:
```
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.

In [69]:
```
iris_df.shape
```

Out[69]:
```
(150, 5)
```

In [70]:
```
# Get number of rows
Nrows = iris_df.shape[0]

print('The Number of rows is ' + str(Nrows))
```

```
The Number of rows is 150
```

In [71]:
```
# Get number of cols
Ncols = iris_df.shape[1]

print('The Number of columns is ' + str(Ncols))
```

```
The Number of columns is 5
```

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

In [72]:
```
iris_df
```

Out[72]:

	sepal_length	sepal_width	petal_length	petal_width	Flower
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

In [73]:
```
iris_df.head(5)
```

Out[73]:

	sepal_length	sepal_width	petal_length	petal_width	Flower
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

In [74]:
```
iris_df.tail(5)
```

Out[74]:

	sepal_length	sepal_width	petal_length	petal_width	Flower
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:

In [75]:
```
iris_df.columns
```

Out[75]:
```
Index(['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'Flower'], 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.

In [76]:
```
iris_df.dtypes
```

Out[76]:
```
sepal_length    float64
sepal_width     float64
petal_length    float64
petal_width     float64
Flower           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.

In [77]:
```
# Summary statistics for the quantitative variables
iris_df.describe()
```

Out[77]:

	sepal_length	sepal_width	petal_length	petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
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

In [78]:
```
# Drop observations with NaN values
iris_df.dropna()
iris_df.shape
```

Out[78]:
```
(150, 5)
```

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

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

In [79]:
```
iris_df.sepal_length.mean()
```

Out[79]:
```
5.843333333333334
```

In [80]:
```
iris_df.max()
```

Out[80]:
```
sepal_length               7.9
sepal_width                4.4
petal_length               6.9
petal_width                2.5
Flower          Iris-virginica
dtype: object
```

In [81]:
```
iris_df.min()
```

Out[81]:
```
sepal_length            4.3
sepal_width             2.0
petal_length            1.0
petal_width             0.1
Flower          Iris-setosa
dtype: object
```

In [82]:
```
iris_df.median(numeric_only=True)
```

Out[82]:
```
sepal_length    5.80
sepal_width     3.00
petal_length    4.35
petal_width     1.30
dtype: float64
```

In [83]:
```
iris_df.corr(numeric_only=True)
```

Out[83]:

	sepal_length	sepal_width	petal_length	petal_width
sepal_length	1.000000	-0.109369	0.871754	0.817954
sepal_width	-0.109369	1.000000	-0.420516	-0.356544
petal_length	0.871754	-0.420516	1.000000	0.962757
petal_width	0.817954	-0.356544	0.962757	1.000000

In [84]:
```
iris_df.std(numeric_only=True)
```

Out[84]:
```
sepal_length    0.828066
sepal_width     0.433594
petal_length    1.764420
petal_width     0.763161
dtype: float64
```

In [85]:
```
iris_df.count()
```

Out[85]:
```
sepal_length    150
sepal_width     150
petal_length    150
petal_width     150
Flower          150
dtype: int64
```

How to write a data frame to a File

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

Examples:

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

In [86]:
```
iris_df.to_csv('IrisOut.csv')
```

Rename columns

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

Example:

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

iris_df.head()

In [87]:
```
iris_df = iris_df.rename(columns={"SepalLengthCm":"SepalLCM","SepalWidthCm":"SepalWCM"
iris_df.head()
```

Out[87]:

	sepal_length	sepal_width	petal_length	petal_width	Flower
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

In [88]:
```
# Back to the original name
iris_df = iris_df.rename(columns={"SepalLCM":"SepalLengthCm","SepalWCM":"SepalWidthCm"
iris_df.head()
```

Out[88]:

	sepal_length	sepal_width	petal_length	petal_width	Flower
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

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.

In [89]:
```
# Select specific columns
selected_columns = iris_df[['sepal_length', 'sepal_width', 'Flower']]
selected_columns.head()
```

Out[89]:

	sepal_length	sepal_width	Flower
0	5.1	3.5	Iris-setosa
1	4.9	3.0	Iris-setosa
2	4.7	3.2	Iris-setosa
3	4.6	3.1	Iris-setosa
4	5.0	3.6	Iris-setosa

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.

In [90]:
```
# Return all observations of CWDistance
# Select few rows for multiple columns, ["CWDistance", "Height", "Wingspan"]
iris_df.loc[4:9, ['sepal_length', 'sepal_width', 'Flower']]
```

Out[90]:

	sepal_length	sepal_width	Flower
4	5.0	3.6	Iris-setosa
5	5.4	3.9	Iris-setosa
6	4.6	3.4	Iris-setosa
7	5.0	3.4	Iris-setosa
8	4.4	2.9	Iris-setosa
9	4.9	3.1	Iris-setosa

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

In [91]:
```
iris_df.iloc[:, 3:5]
```

Out[91]:

	petal_width	Flower
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

iris_df.Gender.unique()

In [92]:
```
# List unique values in the iris_df['Gender'] column
iris_df.Flower.unique()
```

Out[92]:
```
array(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'], dtype=object)
```

In [93]:
```
# Lets explore iris_df["GenderGroup] as well
iris_df.petal_length.unique()
```

Out[93]:
```
array([1.4, 1.3, 1.5, 1.7, 1.6, 1.1, 1.2, 1. , 1.9, 4.7, 4.5, 4.9, 4. ,
       4.6, 3.3, 3.9, 3.5, 4.2, 3.6, 4.4, 4.1, 4.8, 4.3, 5. , 3.8, 3.7,
       5.1, 3. , 6. , 5.9, 5.6, 5.8, 6.6, 6.3, 6.1, 5.3, 5.5, 6.7, 6.9,
       5.7, 6.4, 5.4, 5.2])
```

Filter, Sort and Groupby

With **Filter** you can use different conditions to filter columns. For example, iris_df[iris_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.

iris_df[iris_df["Height"] >= 70]

In [94]:
```
iris_df[iris_df["petal_length"] >= 3]
```

Out[94]:

	sepal_length	sepal_width	petal_length	petal_width	Flower
50	7.0	3.2	4.7	1.4	Iris-versicolor
51	6.4	3.2	4.5	1.5	Iris-versicolor
52	6.9	3.1	4.9	1.5	Iris-versicolor
53	5.5	2.3	4.0	1.3	Iris-versicolor
54	6.5	2.8	4.6	1.5	Iris-versicolor
...	...	...	...	...	...
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

100 rows × 5 columns

With **Sort** is possible to sort values in a certain column in an ascending order using `iris_df.sort_values("ColumnName")` or in descending order using `iris_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 `iris_df.sort_values([Column1Name,Column2Name],ascending=[True,False])`

iris_df.sort_values("Height")

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

In [95]:
```
iris_df.sort_values("petal_length",ascending=True)
```

Out[95]:

	sepal_length	sepal_width	petal_length	petal_width	Flower
22	4.6	3.6	1.0	0.2	Iris-setosa
13	4.3	3.0	1.1	0.1	Iris-setosa
14	5.8	4.0	1.2	0.2	Iris-setosa
35	5.0	3.2	1.2	0.2	Iris-setosa
36	5.5	3.5	1.3	0.2	Iris-setosa
...	...	...	...	...	...
131	7.9	3.8	6.4	2.0	Iris-virginica
105	7.6	3.0	6.6	2.1	Iris-virginica
117	7.7	3.8	6.7	2.2	Iris-virginica
122	7.7	2.8	6.7	2.0	Iris-virginica
118	7.7	2.6	6.9	2.3	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.
iris_df.groupby(col) returns a groupby object for values from one column while iris_df.groupby([col1,col2]) returns a groupby object for values from multiple columns.

iris_df.groupby(['Gender'])

In [96]:
```
iris_df.groupby(['Flower']).size()
```

Out[96]:
```
Flower
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
dtype: int64
```

Size of each group

In [97]:
```
iris_df.groupby(['sepal_width','Flower']).size()
```

Out[97]:

sepal_width	Flower	
2.0	Iris-versicolor	1
2.2	Iris-versicolor	2
	Iris-virginica	1
2.3	Iris-setosa	1
	Iris-versicolor	3
2.4	Iris-versicolor	3
2.5	Iris-versicolor	4
	Iris-virginica	4
2.6	Iris-versicolor	3
	Iris-virginica	2
2.7	Iris-versicolor	5
	Iris-virginica	4
2.8	Iris-versicolor	6
	Iris-virginica	8
2.9	Iris-setosa	1
	Iris-versicolor	7
	Iris-virginica	2
3.0	Iris-setosa	6
	Iris-versicolor	8
	Iris-virginica	12
3.1	Iris-setosa	5
	Iris-versicolor	3
	Iris-virginica	4
3.2	Iris-setosa	5
	Iris-versicolor	3
	Iris-virginica	5
3.3	Iris-setosa	2
	Iris-versicolor	1
	Iris-virginica	3
3.4	Iris-setosa	9
	Iris-versicolor	1
	Iris-virginica	2
3.5	Iris-setosa	6
3.6	Iris-setosa	2
	Iris-virginica	1
3.7	Iris-setosa	3
3.8	Iris-setosa	4
	Iris-virginica	2
3.9	Iris-setosa	2
4.0	Iris-setosa	1
4.1	Iris-setosa	1
4.2	Iris-setosa	1
4.4	Iris-setosa	1

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.

In [98]:
```
missing= iris_df.isnull().sum()
notMissing= iris_df.notnull().sum()

print("missing:", missing)
```

```
missing: sepal_length    0
sepal_width     0
petal_length    0
petal_width     0
Flower          0
dtype: int64
```

Add and eliminate columns

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

In [99]:
```
# Add a new column with new data

# Create a column data
NewColumnData = iris_df.sepal_width/iris_df.sepal_length

# Insert that column in the data frame
iris_df.insert(3, "Ratio width to length", NewColumnData, True)

iris_df.head()
```

Out[99]:

	sepal_length	sepal_width	petal_length	Ratio width to length	petal_width	Flower
0	5.1	3.5	1.4	0.686275	0.2	Iris-setosa
1	4.9	3.0	1.4	0.612245	0.2	Iris-setosa
2	4.7	3.2	1.3	0.680851	0.2	Iris-setosa
3	4.6	3.1	1.5	0.673913	0.2	Iris-setosa
4	5.0	3.6	1.4	0.720000	0.2	Iris-setosa

In [100…
```
# Eliminate inserted column
iris_df.drop("Ratio width to length", axis=1, inplace = True)
```

```
iris_df.head()
```

Out[100]:

	sepal_length	sepal_width	petal_length	petal_width	Flower
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

In [101…

```
# Add new column derived from existing columns

# The new column is a function of another column
iris_df["SepalLengthMm"] = iris_df["sepal_length"] * 10

iris_df.head()
```

Out[101]:

	sepal_length	sepal_width	petal_length	petal_width	Flower	SepalLengthMm
0	5.1	3.5	1.4	0.2	Iris-setosa	51.0
1	4.9	3.0	1.4	0.2	Iris-setosa	49.0
2	4.7	3.2	1.3	0.2	Iris-setosa	47.0
3	4.6	3.1	1.5	0.2	Iris-setosa	46.0
4	5.0	3.6	1.4	0.2	Iris-setosa	50.0

In [102…

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

Out[102]:

	sepal_length	sepal_width	petal_length	petal_width	Flower
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

In [103…

```
# Drop several "unused" columns
```

Add and eliminate rows

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

In [104…

```
# Print tail
iris_df.tail()
```

Out[104]:

	sepal_length	sepal_width	petal_length	petal_width	Flower
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

In [105…

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

Out[105]:

	sepal_length	sepal_width	petal_length	petal_width	Flower
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

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

1. Are there missing data? If so, create a new dataset containing only the rows with the non-missing data

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

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

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

In [107…
```
# Define the col names for the iris dataset
col_names = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'Flower']

# Dataset url
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"

# Load the dataset from URL
iris_df = pd.read_csv(url, header=None, names=col_names)
print(iris_df.head())
```

```
   sepal_length  sepal_width  petal_length  petal_width        Flower
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
```

In [108…
```
iris_df.describe()
```

Out[108]:

	sepal_length	sepal_width	petal_length	petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
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

In [109…
```
iris_df.columns
```

Out[109]:
```
Index(['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'Flower'], dtype
='object')
```

In [110…
```
iris_df.dtypes
```

Out[110]:
```
sepal_length    float64
sepal_width     float64
petal_length    float64
petal_width     float64
Flower           object
dtype: object
```

In [111…
```
missing= iris_df.isnull().sum()
print("Missing values: ", missing)
```

```
Missing values:  sepal_length    0
sepal_width     0
petal_length    0
petal_width     0
Flower          0
dtype: int64
```

In [113…
```
iris_df1 = iris_df[['petal_length','petal_width','Flower']]
iris_df1
```

Out[113]:

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

150 rows × 3 columns

In [114…
```
iris_df2 = iris_df[['sepal_length','sepal_width','Flower']]
iris_df2
```

Out[114]:

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

150 rows × 3 columns

In [115…

```
categories = iris_df['Flower'].unique()
print('Numerical categories', categories)
```

```
Numerical categories ['Iris-setosa' 'Iris-versicolor' 'Iris-virginica']
```

In [155…

```
iris_df['Species Encoded'] = iris_df['Species'].astype('category').cat.codes

iris_df3 = iris_df[['SepalWidthCm','SepalLengthCm','Species Encoded']]
iris_df3
```

Out[155]:

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

150 rows × 3 columns