Reading a csv file

In [3]:

```
import pandas as pd

df = pd.read_csv\
(r'C:\Users\lenovo\anaconda3\pkgs\bokeh-3.3.4-py311h746a85d_0\Lib\site-
df.head()
```

Out[3]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

In [5]:

```
type(df)
```

Out[5]:

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

In [6]:

```
df.shape
```

Out[6]:

```
(150, 5)
```

In [8]:

```
df.columns
```

Out[8]:

```
Index(['sepal_length', 'sepal_width', 'petal_length', 'petal_width',
       'species'],
      dtype='object')
```

In [13]:

```
df['species'].unique()
```

Out[13]:

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

In [16]:

```
df['species'].nunique()
```

Out[16]:

```
3
```

In [18]:

```python
d = {'Iris-setosa': 0,
     'Iris-versicolor': 1,
     'Iris-virginica': 2}

df['species'] = df['species'].map(d)
df.head()
```

Out[18]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	NaN
1	4.9	3.0	1.4	0.2	NaN
2	4.7	3.2	1.3	0.2	NaN
3	4.6	3.1	1.5	0.2	NaN
4	5.0	3.6	1.4	0.2	NaN

In [19]:

```python
df.tail()
```

Out[19]:

	sepal_length	sepal_width	petal_length	petal_width	species
145	6.7	3.0	5.2	2.3	NaN
146	6.3	2.5	5.0	1.9	NaN
147	6.5	3.0	5.2	2.0	NaN
148	6.2	3.4	5.4	2.3	NaN
149	5.9	3.0	5.1	1.8	NaN

In [13]:

```python
import numpy as np
np.unique(df['Species'])
```

Out[13]:

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

Reading a tab delimited file

In [22]:

```python
# reading a tab delimited file
import pandas as pd
df = pd.read_csv(r"C:\Users\lenovo\Downloads\kriti.txt", sep="\t")
df
```

Out[22]:

	Name	Marks
0	Kriti	100
1	Neha	89

In [ ]:

```python
# If you dont want to consider the firt row as column name then make he
```

In [24]:

```
df = pd.read_csv(r"C:\Users\lenovo\Downloads\kriti.txt", sep="\t",heade
df
```

Out[24]:

```
       0      1
0   Name  Marks
1  Kriti    100
2   Neha     89
```

Make a comma seperated file. Manually make a comma seperated file (text file) and read it

In [30]:

```
d = pd.read_csv(r'C:\Users\lenovo\Downloads\hi.txt', header = None)
d.head()
```

Out[30]:

```
     0     1
0   hi   how
1  are   you
2  iam  fine
```

In [31]:

```
# Read a semi-colon seperated file. Make your own semi colon text file
df = pd.read_csv(r'C:\Users\lenovo\Downloads\semi_colon.txt', \
                  sep=";", header=None)
df
```

Out[31]:

```
     0     1
0   hi   how
1  are   you
2  iam  fine
```

Reading the .csv file from a http link

In [32]:

```
# reading the .csv file from a http link
df = pd.read_csv('https://\
raw.githubusercontent.com/scpike/us-state-county-zip/master/geo-data.cs
```

In [33]:

```
df
```

Out[33]:

```
       state_fips     state  state_abbr  zipcode      county         city
    0           1   Alabama          AL    35004    St. Clair        Acmar
    1           1   Alabama          AL    35005    Jefferson   Adamsville
    2           1   Alabama          AL    35006    Jefferson        Adger
    3           1   Alabama          AL    35007       Shelby     Keystone
    4           1   Alabama          AL    35010   Tallapoosa     New site
  ...         ...       ...         ...      ...          ...          ...
33098          56   Wyoming          WY    83126      Lincoln        Smoot
33099          56   Wyoming          WY    83127      Lincoln       Thayne
33100          56   Wyoming          WY    83128      Lincoln       Alpine
33101          56   Wyoming          WY   831HH       Lincoln   Zcta 831hh
33102          56   Wyoming          WY   831XX       Lincoln   Zcta 831xx

33103 rows × 6 columns
```

Reading an inbuilt dataset:

Scikit-learn is probably the most useful library for machine learning in Python. The sklearn library contains a lot of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction.

Inbuld dataset using sklearn: There are many inbuild datasets such as: load_iris(): Loads the Iris flower dataset. load_digits(): Loads the handwritten digits dataset. load_boston(): (Deprecated) Loads the Boston housing dataset. load_breast_cancer(): Loads the breast cancer dataset. load_wine(): Loads the wine recognition dataset. load_diabetes(): Loads the diabetes dataset. load_linnerud() Load and return the linnerud dataset (multivariate regression).

In [40]:

```python
#In the sklearn (scikit-learn) library, the datasets module provides a
# load_iris is a function from sklearn
from sklearn.datasets import load_iris
```

In Python, particularly when using the sklearn.datasets module to load toy datasets (like Iris, Digits, Wine, etc.), the data is typically stored in a special data structure known as a Bunch object.

What is a Bunch Object? A Bunch is a dictionary-like object that allows access to its keys as attributes. It's similar to a dictionary but provides a more convenient, object-oriented way of accessing the data. It is a custom data structure provided by scikit-learn to encapsulate the data, target labels, and metadata for toy datasets.

Structure of a Bunch Object When you load a dataset using functions like load_iris() or load_wine(), the returned Bunch object typically contains the following attributes:

data: The main feature matrix (usually a NumPy array). target: The target labels (usually a NumPy array). feature_names: The names of the features (if available). target_names: The names of the target classes (if available). DESCR: A full description of the dataset. filename:

In [47]:

```
iris = load_iris()
iris
```

Out[47]:

```
{'data': array([[5.1, 3.5, 1.4, 0.2],
        [4.9, 3. , 1.4, 0.2],
        [4.7, 3.2, 1.3, 0.2],
        [4.6, 3.1, 1.5, 0.2],
        [5. , 3.6, 1.4, 0.2],
        [5.4, 3.9, 1.7, 0.4],
        [4.6, 3.4, 1.4, 0.3],
        [5. , 3.4, 1.5, 0.2],
        [4.4, 2.9, 1.4, 0.2],
        [4.9, 3.1, 1.5, 0.1],
        [5.4, 3.7, 1.5, 0.2],
        [4.8, 3.4, 1.6, 0.2],
        [4.8, 3. , 1.4, 0.1],
        [4.3, 3. , 1.1, 0.1],
        [5.8, 4. , 1.2, 0.2],
        [5.7, 4.4, 1.5, 0.4],
        [5.4, 3.9, 1.3, 0.4],
        [5.1, 3.5, 1.4, 0.3],
        [5.7, 3.8, 1.7, 0.3],
        [5.1, 3.8, 1.5, 0.3],
```

In [56]:

```
iris.DESCR
```

Out[56]:

```
'.. _iris_dataset:\n\nIris plants dataset\n--------------------\n\n**Data
Set Characteristics:**\n\n    :Number of Instances: 150 (50 in each of thr
ee classes)\n    :Number of Attributes: 4 numeric, predictive attributes a
nd the class\n    :Attribute Information:\n        - sepal length in cm\n
- sepal width in cm\n        - petal length in cm\n        - petal width i
n cm\n        - class:\n                - Iris-Setosa\n                - I
ris-Versicolour\n                - Iris-Virginica\n                \n    :
Summary Statistics:\n\n    ============== ==== ==== ======= ===== ========
============\n                    Min  Max   Mean    SD   Class Correlatio
n\n    ============== ==== ==== ======= ===== ====================\n    se
pal length:   4.3  7.9   5.84   0.83    0.7826\n    sepal width:    2.0
4.4   3.05   0.43   -0.4194\n    petal length:   1.0  6.9   3.76   1.76
0.9490  (high!)\n    petal width:    0.1  2.5   1.20   0.76    0.9565  (hi
gh!)\n    ============== ==== ==== ======= ===== ====================\n\n
:Missing Attribute Values: None\n    :Class Distribution: 33.3% for each o
f 3 classes.\n    :Creator: R.A. Fisher\n    :Donor: Michael Marshall (MAR
SHALL%PLU@io.arc.nasa.gov)\n    :Date: July, 1988\n\nThe famous Iris datab
ase, first used by Sir R.A. Fisher. The dataset is taken\nfrom Fisher\'s p
aper. Note that it\'s the same as in R, but not as in the UCI\nMachine Lea
rning Repository, which has two wrong data points.\n\nThis is perhaps the
best known database to be found in the\npattern recognition literature.  F
isher\'s paper is a classic in the field and\nis referenced frequently to
this day.  (See Duda & Hart, for example.)  The\ndata set contains 3 class
es of 50 instances each, where each class refers to a\ntype of iris plant.
One class is linearly separable from the other 2; the\nlatter are NOT line
arly separable from each other.\n\n.. topic:: References\n\n   - Fisher,
R.A. "The use of multiple measurements in taxonomic problems"\n     Annual
Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to\n     Math
ematical Statistics" (John Wiley, NY, 1950).\n   - Duda, R.O., & Hart, P.
E. (1973) Pattern Classification and Scene Analysis.\n     (Q327.D83) John
Wiley & Sons.  ISBN 0-471-22361-1.  See page 218.\n   - Dasarathy, B.V. (1
980) "Nosing Around the Neighborhood: A New System\n     Structure and Cla
ssification Rule for Recognition in Partially Exposed\n     Environments".
IEEE Transactions on Pattern Analysis and Machine\n     Intelligence, Vol.
PAMI-2, No. 1, 67-71.\n   - Gates, G.W. (1972) "The Reduced Nearest Neighb
or Rule".  IEEE Transactions\n     on Information Theory, May 1972, 431-43
3.\n   - See also: 1988 MLC Proceedings, 54-64.  Cheeseman et al"s AUTOCLA
SS II\n     conceptual clustering system finds 3 classes in the data.\n
- Many, many more ...'
```

In [57]:

```
iris.target_names
```

Out[57]:

```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

In [58]:

```
# retreive features  names
iris.feature_names
```

Out[58]:

```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

In [59]:

```
iris.target
```

Out[59]:

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

In [52]:

```
iris.data
```

Out[52]:

```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1],
       [5.4, 3.7, 1.5, 0.2],
       [4.8, 3.4, 1.6, 0.2],
       [4.8, 3. , 1.4, 0.1],
       [4.3, 3. , 1.1, 0.1],
       [5.8, 4. , 1.2, 0.2],
       [5.7, 4.4, 1.5, 0.4],
       [5.4, 3.9, 1.3, 0.4],
       [5.1, 3.5, 1.4, 0.3],
       [5.7, 3.8, 1.7, 0.3],
```

In [63]:

```
# Load iris into a dataframe and set the field names
df = pd.DataFrame(iris['data'], columns=iris['feature_names'])
df.head()
```

Out[63]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

Attach the target to the dataframe

The pd.Categorical.from_codes() method in pandas is used to create a Categorical object from integer codes, where each integer represents a category. This is particularly useful when you have data encoded as integers, but you want to associate those integers with specific category labels.

In [66]:

```
# Change target to target_names & merge with main dataframe
df['species'] = pd.Categorical.from_codes(iris.target, iris.target_name
df
```

Out[66]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

150 rows × 5 columns

In [72]:

```
# adding target column to dataframe
df1 = pd.DataFrame(iris['data'], columns=iris['feature_names'])
df1.head()
```

Out[72]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

In [77]:

```
species=iris.target
species
```

Out[77]:

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

In [78]:

```
# adding target column to dataframe
df1["species"] = species
df1
```

Out[78]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
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	2
146	6.3	2.5	5.0	1.9	2
147	6.5	3.0	5.2	2.0	2
148	6.2	3.4	5.4	2.3	2
149	5.9	3.0	5.1	1.8	2

150 rows × 5 columns

In [69]:

```
x = df1.iloc[:, 1:5]
```

In [70]:

```
x
```

Out[70]:

	sepal width (cm)	petal length (cm)	petal width (cm)
0	3.5	1.4	0.2
1	3.0	1.4	0.2
2	3.2	1.3	0.2
3	3.1	1.5	0.2
4	3.6	1.4	0.2
...	...	...	...
145	3.0	5.2	2.3
146	2.5	5.0	1.9
147	3.0	5.2	2.0
148	3.4	5.4	2.3
149	3.0	5.1	1.8

150 rows × 3 columns

In [70]:

```
#Just as a quick check, we show the first 5 rows
x[:5]
```

Out[70]:

```
   sepal width (cm)  petal length (cm)  petal width (cm)  species
0               3.5                1.4               0.2        0
1               3.0                1.4               0.2        0
2               3.2                1.3               0.2        0
3               3.1                1.5               0.2        0
4               3.6                1.4               0.2        0
```

In [ ]: