Using Datasets for Machine Learning tasks

Overview

In this tutorial, we are going to briefly explore the basics of the following

1. Datasets, thier formats and ways of gathering datasets for building ML models.
2. Loading the datasets.
3. Summarizing the datasets.
4. Visualizing the datasets.
5. Dataset preparation for building machine learning models.
6. Dealing with the text files
    - Reading from text files and writing to text files, Tokenization of text etc.

Introduction

- Dataset is a collection of data.
- A single row of data in the dataset is called an **instance**.
- Datasets are a collection of instances that all share common attributes.
- Data can come in many forms.
- But, ML models rely mostly on four primary data types namely **numerical data, categorical data, time series data, and text data**.
- Dataset files can be in different formats like .csv, .txt, .xlsx etc.
- A **file format** defines the structure and encoding of the data stored in it and it is typically identified by its file extension.

- Datasets can be obtained from various resources like UCI Machine Learning Repository, Kaggle, Google Dataset Search (https://datasetsearch.research.google.com/) etc.
- The choice of data entirely depends on the problem we are trying to solve.
- For the purpose of this presentation, we are going to use the **Iris flowers dataset**.
- The Iris flowers dataset is in **.csv** format and it contains **150 observations** of iris flowers.
- There are four columns of measurements of the flowers in centimeters. The fifth column is the species of the flower observed. All observed flowers belong to one of **three species**.
- Datasets can be manipulated using several library tools and methods.
- In this presentation we are going to use python library tools like **Pandas and matplotlib** etc.

Loading the Datasets

- Loading the Iris data set from an URL using pandas

```
import pandas
from pandas import read_csv

# Loading the dataset directly from its URL
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/iris.csv"
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']
dataset = read_csv(url, names=names)
```

Summarizing the dataset

- Checking the dimensions of the data

```
print(dataset.shape)
```

```
(150, 5)
```

- Peek at the data

```
# Want to see first few (assume 5) rows of the dataset? Then do this.
print(dataset.head(5))
```

```
   sepal-length  sepal-width  petal-length  petal-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
```

- Finding statistical summary of each attribute

```
# descriptions
print(dataset.describe())
```

```
       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
```

- Finding the number of instances (rows) that belong to each class

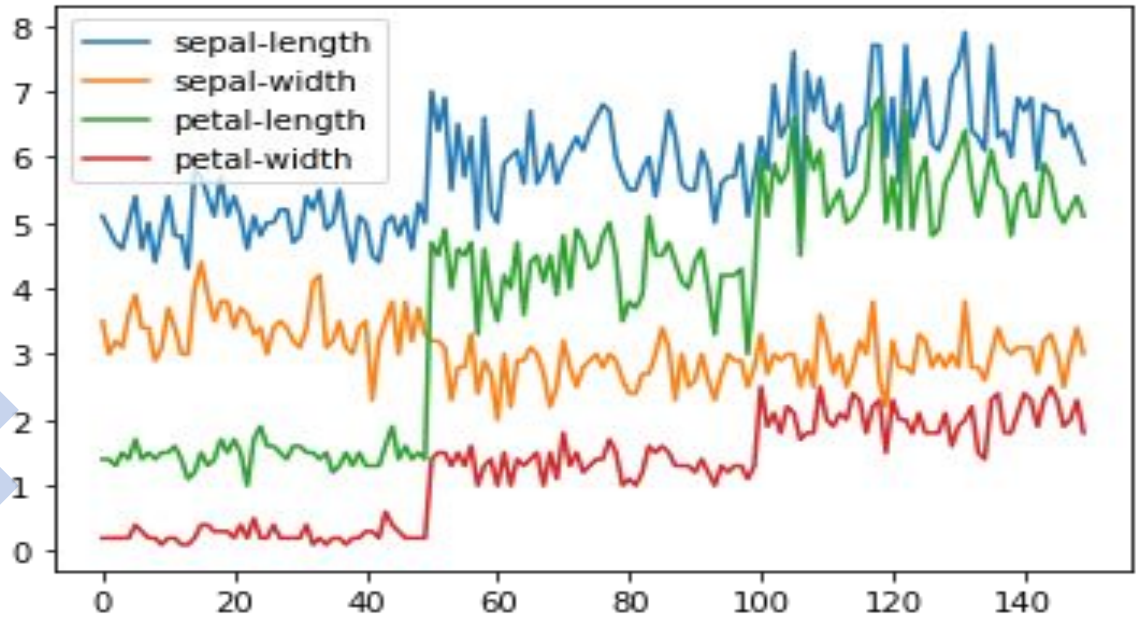
```
# class distribution
print(dataset.groupby('class').size())
```

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

Data Visualization

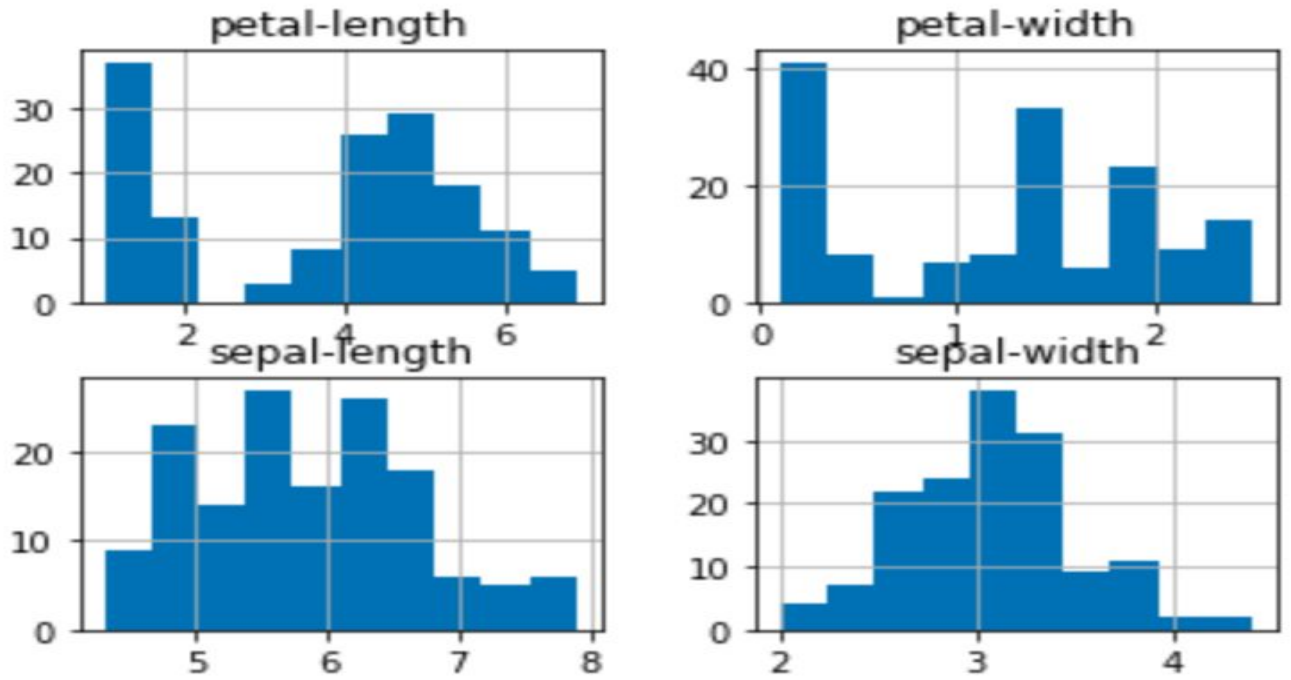
- Univariate plots - plots of each individual attribute

```
import matplotlib
from matplotlib import pyplot
dataset.plot()
```



- Creating histogram of each individual variable to get an idea of its distribution

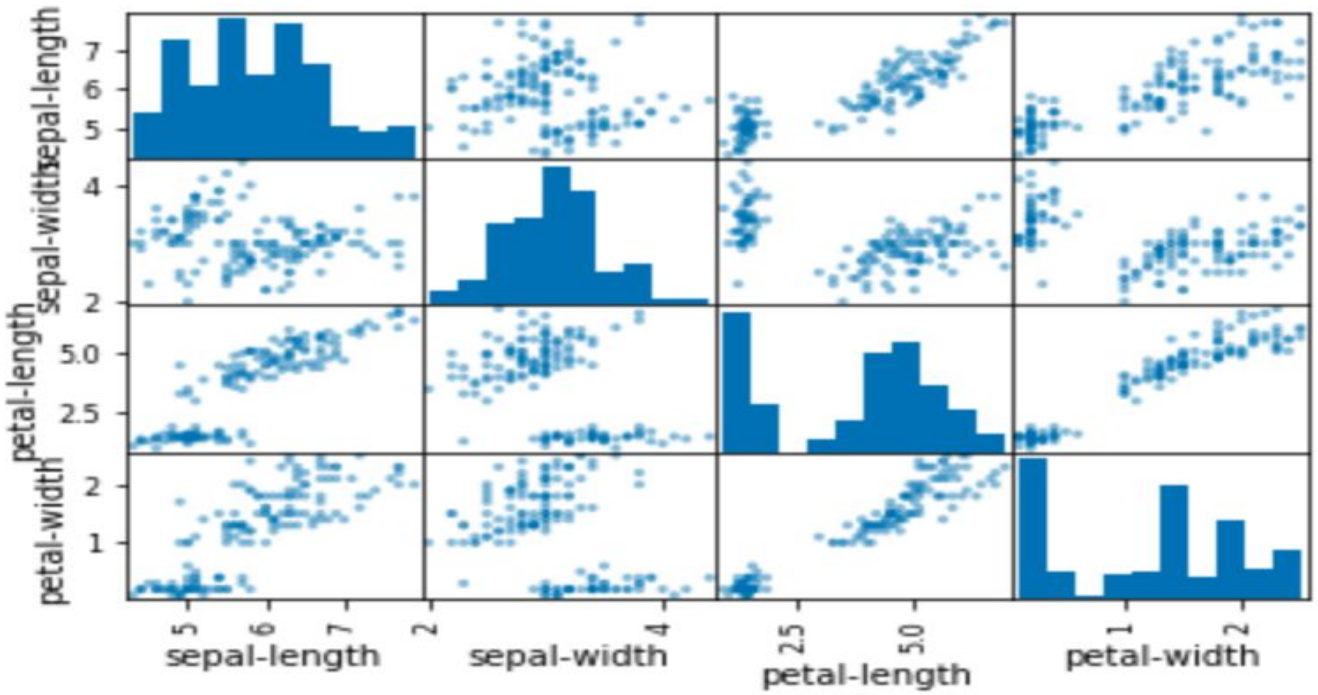
```
# histograms
dataset.hist()
pyplot.show()
```



- Multi variate plots – To look at the structured relationships between the variables
- Plotting scatterplots of all pairs of attributes – this helps us know correlation and a predictable relationship between pairs of attributes.

```
from pandas.plotting import scatter_matrix

# scatter plot matrix
scatter_matrix(dataset)
pyplot.show()
```



How to prepare your Dataset for ML tasks?

- We just have seen basics of loading dataset, summarizing and visualizing the data in it.
- Most of the times, we do not have the correct data to work with.
- If the data is not processed correctly, we need to prepare it and then start building ML models.
- Preparing the data can include the steps like following
    1. Getting the dataset and importing all the required libraries.
    2. Handling the missing data.
    3. Taking your data further with feature extraction
    4. Encoding the categorical data.
    5. Feature Scaling, if all the columns are not scaled correctly.
    6. Splitting the dataset into the Training set and Test set

Dealing with text files

- Create a text file with the following text and store it as "myfile.txt" in your local directory.

- Opening the text file

```
myfile = open("myfile.txt")
```

- Let's now see what is stored in the myfile variable:

```
print(myfile)
```

```
<_io.TextIOWrapper name='myfile.txt' mode='r' encoding='UTF-8'>
```

- The output reads that myfile variable is a wrapper to the myfile.txt file and opens the file in read-only mode.

- Now, let's read the contents of the file. To do so, you need to call the read() function on the myfile variable, as shown below:

```
myfile = open("myfile.txt")
print(myfile.read())
```

```
Welcome to Machine Learning Fundamentals and Applications.
Machine Learning is one of the most exciting research fields today.
Let us see how python can be used to work with text files.
Happy Learning!
```

- Now if you try to call the read method again, you will see that nothing will be printed since after calling the read method, the cursor is moved to the end of the file. So if you call read method again, nothing will be printed since there is no text to print at the end of the file.

```
print(myfile.read())
```

- A solution to this problem is that after calling the read() method, call the seek() method and pass 0 as the argument. This will move the cursor back to the start of the text file.

```
myfile = open("myfile.txt")
print(myfile.read())
myfile.seek(0)
print(myfile.read())
```

```
Welcome to Machine Learning Fundamentals and Applications.
Machine Learning is one of the most exciting research fields today.
Let us see how python can be used to work with text files.
Happy Learning!

Welcome to Machine Learning Fundamentals and Applications.
Machine Learning is one of the most exciting research fields today.
Let us see how python can be used to work with text files.
Happy Learning!
```

- It is important to close the file once you are done so that other applications can use the file. To do so, you need to call the close() method.

```
myfile.close()
```

- Instead of reading all the contents of the file at once, we can also read the file contents line by line using readlines() method.

```
myfile = open("myfile.txt")
print(myfile.readlines())
```

```
['Welcome to Machine Learning Fundamentals and Applications.\n', 'Machine Learning is one of the most exciting research fields today.\n', 'Let us see how python can be used to work with text files.\n', 'Happy Learning!\n']
```

- In the output, you can see each line in the text file as a list item
- In many cases this makes the text easier to work with.
- For example, we can now easily iterate through each line and print the first word in the line.

```
myfile = open("myfile.txt")
for lines in myfile:
    print(lines.split()[0])
```

```
Welcome
Machine
Let
Happy
```

- Till now, we have seen how to read from a text file. Now let's see how to write to text files.
- To write to a text file, we have to open it with its mode set to 'w' or 'w+'.
- Use 'w' to open the file in write only mode.
- Use 'w+' to open the file in both read and write mode.
- If the file doesn't exist while opening, the file will be created.
- Content can be written to the file using write() method.
- It is important to notice that if you open a file that already contains some text with w or w+ mode, all the existing file contents will be removed. (observe that no output printed for second line of code)

```
myfile = open("myfile.txt", 'w+')
print(myfile.read())
myfile.write("The file has been rewritten")
myfile.seek(0)
print(myfile.read())
```

```
The file has been rewritten
```

- But most of the times, you don’t want to remove the existing contents of the file. In this case, we can open the file in 'a+' mode instead of 'w' or 'w+' mode. So that new text can be appended to the existing text.

```
myfile = open("myfile.txt", 'a+')
myfile.write("\nThis is a new line")
myfile.seek(0)
print(myfile.read())
```

```
The file has been rewritten
This is a new line
```

- Using the with keyword, as shown below, you don't need to explicitly close the file. Rather, the below script opens the file, reads its contents, and then closes it automatically

```
with open("myfile.txt") as myfile:
    print(myfile.read())
```

```
The file has been rewritten
This is a new line
```

Dealing with the Text data

- We have seen how to read from and write to text files.
- In this tutorial, we are going to learn sentence tokenization and word tokenization using NLTK library.
- Make sure that NLTK library is installed.
- **Sentence tokenization:** Sentence tokenization (also called sentence segmentation) is the problem of dividing a string of written language into its component sentences.

```
import nltk
text = "Welcome to MLFA class. Now we are about to learn sentence and word tokenizations "
sentences = nltk.sent_tokenize(text)
for sentence in sentences:
    print(sentence)
    print()
```

```
Welcome to MLFA class.

Now we are about to learn sentence and word tokenizations
```

- We can see 2 component sentences seperately in the output.

- **Word Tokenization:** Word tokenization (also called word segmentation) is the problem of dividing a string of written language into its component words.
- Let's use the sentences from the previous output and see how we can apply word tokenization on them. We can use the nltk.word_tokenize() function.

```
for sentence in sentences:
    words = nltk.word_tokenize(sentence)
    print(words)
    print()
```

```
['Welcome', 'to', 'MLFA', 'class', '.']

['Now', 'we', 'are', 'about', 'to', 'learn', 'sentence', 'and', 'word', 'tokenizations', '.']
```


Thank You!