Python and Datascience Workshop

Python Tutorial

Author: Mohammad Akradi [1]

[1] Institute of Medical Science and Technology, Shahid Beheshti University, Tehran, Iran

Session 2b

Now, it's time taht we go through pandas library to learn how to deal with tabular data.

---

1. Pandas and Dataframe

pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language. it can be easily installed using pip.

A DataFrame is a data structure that organizes data into a 2-dimensional table of rows and columns, much like a spreadsheet. DataFrames are one of the most common data structures used in modern data analytics because they are a flexible and intuitive way of storing and working with data.

in a dataframe, each column shows a feature or label and each row represents a sample. it's template is like below table:

sample-id	**feature_1**	**...**	**feature_n**	**label**
id_1	value_11	...	value_n1	classA
id_2	value_12	...	value_n2	classB
...	...	...	...	...
id_m	value_1m	...	value_nm	classA

---

2. Creating Dataframe

There are lots of ways to create a dataframe with pandas. we've showed some methods in the example below:

- Example #1:

```
import pandas as pd

# creating an empty dataframe
df = pd.DataFrame()

# creating a dataframe from a list of values with arbitrary colnames:
lst = [["sub1", 20], ["sub2", 22], ["sub3", 24]]
pd.DataFrame(lst, columns = ["sub-id", "age"])

# creating a dataframe using a dictionary:
dct = {"sub-id": ["sub1", "sub2", "sub3"], "age": [20, 22, 21]}
pd.DataFrame(dct)

# creating a dataframe using a dictionary with arbitrary colnames and
index:
lst = [["sub1", 20], ["sub2", 22], ["sub3", 24]]
pd.DataFrame(lst, columns = ["sub-id", "age"], index = ["gold", "silver",
"bronze"])
```

---

3. Load and Save Data

know, we will load a data from seaborn module.

- Example #1:

```
import seaborn as sns

iris = sns.load_dataset('iris')
iris.head()
```

however, if we have data file on our device, we can load it with pandas library:

- Example #2:

```
import pandas as pd

iris = pd.read_csv("Materials/iris.csv")
iris.head()
```

if we want to save `iris` dataframe, we use `pd.to_csv` function:

- Example #3:

```
# save file in the current directory
iris.to_csv("iris.csv")

# save file in other directory
iris.to_csv("<directory_path>/iris.csv")
```

in a dataframe, some columns may be categorical, which have more than one value, but repeated; we call its values as their level:

- Example #4:

```
# to select a column from dataframe:
iris['species']

# or
iris.species

# for displaying a categorical column's different levels:
print(iris['species'].unique())
```

if we want to make a copy from dataframe, we use `df.copy()` function:

- Example #5:

```
df = iris.copy()
```

---

4. Delete data from a dataframe

to drop a column from dataframe we use `df.drop` function:

- Example #4:

```
iris.drop(columns = "sepal_length")

# if we want to drop more than one column, we should put col-names in the
[]

iris.drop(columns = ["sepal_length", "sepal_width"])

# to save changes in the dataframe we set inplace arguman value to True
iris.drop(columns = "sepal_length", inplace = True)
```

if we want to drop a row from dataframe, again we use `drop` function:

- Example #5:

```
'''
labels : single label or list-like
    Index or column labels to drop.
axis : {0 or 'index', 1 or 'columns'}, default 0
    Whether to drop labels from the index (0 or 'index') or
    columns (1 or 'columns').
index : single label or list-like
    Alternative to specifying axis (``labels, axis=0``
    is equivalent to ``index=labels``).
columns : single label or list-like
    Alternative to specifying axis (``labels, axis=1``
    is equivalent to ``columns=labels``).
'''

iris.drop(label=2, axis=0)
# or
iris.drop(label=2, axis='index')

iris.drop(index=2)
```

---

5. Update data in a dataframe

we can select a column by its name in `[ ]` or after ".". for example, if we want to choose column "sepal_length" from iris dataset, we write `iris['sepal_length']` or `iris.sepal_length`.

for selecting a row, we can use both `loc` or `iloc` functions.

- Example #1:

```
sepal_length = iris['sepal_length']

# or
sepal_length = iris.filter(items=["sepal_length"])

print(sepal_length)

first_row = iris.loc[0]

# If you want to see a selected row of a data frame as a data frame, you
must put the row index in an additional []
```

```
first_row = iris.loc[[0]]

first_row = iris.iloc[[0]]
```

there is some differences between `loc` and `iloc`.

- Example #2:

```
iris.set_index('sepal_length', inplace=True)

iris.loc[[5.1]]

iris.iloc[[1]]
```

if we want to have a look at a specific column from a row, we can put the column name in `loc` function, however, in `iloc` function we should put the column's location

- Example #3:

```
iris.reset_index(inplace=True)

iris.loc[[1], "sepal_width"]

iris.iloc[[1], 2]

# if we want to use iloc, it is very difficult to find location of a
column in a dataframe with lots of columns
iris.iloc[[1], iris.columns.get_loc("sepal_width")]
```

Now that we have learned how to access a value from a specific sample in a dataframe, we can easily update the data.

- Example #4:

```
print(iris.loc[[10], "sepal_length"])
iris.loc[[10], "sepal_length"] = 2
print(iris.loc[[10, "sepal_length"]])
```

in the last example, we want to select samples with `sepal_length==5.1` and update their sepal_length data with 5.11.

- Example #5:

```
iris["sepal_length"][iris["sepal_length"]==5.1] = 5.11

# we can do it with "loc"
iris.loc[iris.sepal_length==5.1, "sepal_length"] = 5.11

iris
```

---

6. Missing values

First Look at the Dataset

- Example #1:

```
import pandas as pd

iris_missing = pd.read_csv("Materials/iris-miss.csv")
iris_missing.head()

iris_missing.info()

# counting missing values
print(iris_missing.isnull().sum())
```

- **Deleting columns with missing data**

we can delete columns which have missing values

- Example #2:

```
iris_updated = iris_missing.dropna(axis = 1)

# we can set a threshold for dropping columns:
iris_updated = iris_missing.dropna(thresh = 150-10, axis = 1)
```

- **Deleting rows with missing data**

another approach is to delete samples with missing values, again we can use threshold settings:

- Example #3:

```
iris_updated = iris_missing.dropna(axis = 0)

# we can set a threshold for dropping columns:
iris_updated = iris_missing.dropna(thresh = 4, axis = 0)
```

- **Filling the missing values (Imputation)**

In this case, we will be filling the missing values with a certain number.

The possible ways to do this are:

1. Filling the missing data with the mean or median value if it's a numerical variable.
2. Filling the missing data with mode if it's a categorical value.
3. Filling the numerical value with 0 or -999, or some other number that will not occur in the data. This can be done so that the machine can recognize that the data is not real or is different.
4. Filling the categorical value with a new type for the missing values.

You can use the `fillna()` function to fill the null values in the dataset.

- Example #4:

```
iris_updated = iris_missing.copy()
iris_updated["petal_length"] =
iris_updated["petal_length"].fillna(iris_updated["petal_length"].mean())

iris_updated
```

---

7. Replace

sometimes we need to replace values in a column with new values. in this case, we use `replace` function. in this function the input is a dictionary in which the keywords are old values and values are the new values.

- Example #1:

```
print(iris["species"].unique())

iris_missing["species"].replace({'setosa': 0, 'versicolor': 1,
'virginica': 2}, inplace=True)
print(iris["species"].unique())
```

another way to replace values in a columns, is to use `loc`.

- Example #2:

```
import numpy as np

iris.loc[iris["petal_length"]>iris["petal_length"].mean(), "petal_length"]
= np.nan
```

---

8. Grouping and Summarizing data

In real data science projects, you'll be dealing with large amounts of data and trying things over and over, so for efficiency, we use `Groupby` concept. Groupby concept is really important because it's ability to aggregate data efficiently, both in performance and the amount code is magnificent.

- Example #1:

```
import pandas as pd

iris = pd.read_csv("Materials/iris.csv")

iris.groupby(by="species").mean()

# to add more functions we use "agg"
iris.groupby(by="species").agg(["mean", "std", "median"])

# if we want to look at specific features:
iris[["sepal_length", "sepal_width",
"species"]].groupby(by="species").agg(["mean", "std", "median"])
```

other way to summarize data is using `describe` function. this function gives some basic information for each feature of dataframe. also, sometimes we need to count number of an specific value in a column. in this case, we use `value_counts` function.

- Example #2:

```
iris.describe()

# describe specific columns
iris[["sepal_length", "sepal_width"]].describe()
```

```
# value_counts
iris["species"].value_counts()
```

---

9. Merge and Concatenate

in some cases, we have two or more seprated dataframes that we want to merge them together. so we use one of the `merge` or `concat` functions. firstly, we create 3 dataframes and concat them:

- Example #1:

```
import pandas as pd

df1 = pd.DataFrame(
    {
        "A": ["A0", "A1", "A2", "A3"],
        "B": ["B0", "B1", "B2", "B3"],
        "C": ["C0", "C1", "C2", "C3"],
        "D": ["D0", "D1", "D2", "D3"],
    },
    index=[0, 1, 2, 3],
)

df2 = pd.DataFrame(
    {
        "A": ["A4", "A5", "A6", "A7"],
        "B": ["B4", "B5", "B6", "B7"],
        "C": ["C4", "C5", "C6", "C7"],
        "D": ["D4", "D5", "D6", "D7"],
    },
    index=[4, 5, 6, 7],
)

df3 = pd.DataFrame(
    {
        "A": ["A8", "A9", "A10", "A11"],
        "B": ["B8", "B9", "B10", "B11"],
        "C": ["C8", "C9", "C10", "C11"],
        "D": ["D8", "D9", "D10", "D11"],
    },
    index=[8, 9, 10, 11],
)

frames = [df1, df2, df3]
df = pd.concat(frames, axis=0, join="outer")
```

Suppose we wanted to associate specific keys with each of the pieces of the chopped up DataFrame. We can do this using the keys argument:

- Example #2:

```
result = pd.concat(frames, keys=["x", "y", "z"])

result.loc["y"]
```

if we set `join` arguman to `inner` and `axis` to 1, then rows with same indexes will be concat together:

- Example #3:

```
df4 = pd.DataFrame(
    {
        "B": ["B2", "B3", "B6", "B7"],
        "D": ["D2", "D3", "D6", "D7"],
        "F": ["F2", "F3", "F6", "F7"],
    },
    index=[2, 3, 6, 7],
)

result = pd.concat([df1, df4], axis=1)
result

# inner join:
result = pd.concat([df1, df4], axis=1, join="inner")
result
```

pandas provides a single function, `merge()`, as the entry point for all standard database join operations between DataFrame or named Series objects:

- Example #4:

```
df_left = pd.DataFrame(
    {
        "key": ["K0", "K1", "K2", "K3"],
        "A": ["A0", "A1", "A2", "A3"],
        "B": ["B0", "B1", "B2", "B3"],
    }
)


df_right = pd.DataFrame(
    {
```

```
        "key": ["K0", "K1", "K2", "K3"],
        "C": ["C0", "C1", "C2", "C3"],
        "D": ["D0", "D1", "D2", "D3"],
    }
)

df_left.merge(df_right, on = "key") # there are more options like
{left_on, right_on, how, ...}
```