20/4/23

Pandas Python

(from corey schafer)

used to
read tabular data.

Introduction

* import Pandas as pd — General Convention of importing

* df = pd.read_csv('path/to/file.txt') — read into the file

* df (short for Dataframe). — Prints the file.

* df.shape — columns x rows of the file (or) Shape of the file.

* df.info() — information of all the datatypes of the columns along with the shape of the dataframe

datatypes → int64, float64, object
(some of 'em) — object: string type. (generally)

pd.set_option('display.max_columns', 36) — a max of '36' columns is displayed.

* pd.set_option('display.max_rows', 20) — a max of 20 rows is displayed.

df.head() → get first 5 rows.

* df.head(7) → get first 7 rows.

df.tail() → get ~~first~~ last 5 rows.

df.tail(7) → get the last 7 rows.

Dataframe & Series Datatypes

Dataframe → 2-D Datastructure.

Dataframe is |||^{er} to a Dictionary.

ex people = {

```
        "name" : ["x", "y", "z"]
        "role" : ["A", "B", "C"]
}
```

X → A
Y → B
Z → C

people["name"] → ["X", "Y", "Z"]

to convert this Dictionary into Dataframe we can do

~~df = pd. DataFrame~~ df = pd.DataFrame(people)

*

print(df) →

	name	role
0	X	A
1	Y	B
2	Z	C

→ indexes * — INDEXES

* type(df) = pandas.core.frame.DataFrame

print(df["name"])

0	X
1	Y
2	Z

type(df["name"])

↓

pandas.core.series.Series

Series → 1-D Datastructure |||^{er} to Lists but a lot more functionality. (or) Array.

We can Also write df.name (or) df.role but

* df["name"] (or) df["role"] is more preferred.
(✓) (✓)

Printing out Multiple columns

df[ ]

↑ pass in '1' column (or) a list of columns.

* df[["name", "role"]]

Now, we've seen how to access columns,

How to Access rows?

→ for that we use iloc & loc.

df.iloc[( ),( )]

→ pass in a list (or) single index (row indexes) of indexes (necessary argument) *

→ pass in the column index (or) list of column indexes (optional argument) *

→ Used for slicing the Dataframe with integer-indexes.

* df.iloc[0]

prints out all the columns of row index 0.

i.e.

	name	role
0	X	A

* df.iloc[[0,1], 1]

↓

0	A
1	B

* df.iloc[[1,2], [0,1]]

↓

	name	role
1	Y	B
2	Z	C

But we can use custom-named indexes

	0	1	2
0			
1			
2			

regular

	name	role
0		
1		
2		

custom-named

df.loc → $||^{er}$ to df.iloc

* df.iloc[[1,2],["role, "name"]]

↓

	role	name
1	B	y
2	Z	Z

* df.columns → shows (or) [illegible] returns a list of all the columns in the DataFrame df.

↓

["name", "role"]

We can Also use slicing

```
df.loc[[0,1,2], ["", ""]]
                 name  role
```

can be written as → * df.loc[0:2, "name": "role"]

&

```
df.iloc[0:2, 0:1]
```

* While slicing in pandas → 0:2 → * Both are inclusive.

(✓) pandas → 0:2 → 0,1,2

generally → 0:2 → 0,1

Indexes

We can set a column to act as a Index [*Also Indexes need not be unique according to pandas]

we can do it by

```
* df.set_index("column_name")
  df
```

But this does not actually change df to set index as 'column_name' we can do that by

```
df = df.set_index("column_name")
      (or)
df.set_index("column_name", inplace = True).
```

df.index → shows or displays the index column of the DataFrame df.

*

→ When we do this we are setting custom indexes

the use of this feature is

Imagine we have fields like "name" & "email" we have unique emails then that can act as a field

=> we do df= df.set_index ("email")

& we can search based on email using 'loc'

* df.loc ("johndoe@gmail.com")

↳ gets all info relating to the email "johndoe@gmail.com" the name & stuff.

* Note

We cannot use loc[0] once we change the indexes

since indexes 0, 1, .. do not exist since we changed it

But, we can still use iloc[0] & stuff

with iloc it still works.

We can reset indexes by doing

* df = df.reset_index()

(or)

df.reset-index (inplace = True)

Now, all the integer indexes are restored.

We can use * schema_df.sort_index()

↳ sorts all the index

ascending=False

inplace=True

(in pandas) can be used in many places where the changes are temporary

*

df.shape, df.index, df.columns.

Note* df.loc & df.iloc return the smaller Dataframes wrt to the values we put in them. The returned Dataframes are of the type 'pandas.core.frame.DataFrame'

Same as all the Dataframe.

(A new sub-Dataframe of is returned, of the Original 'df'.

Filtering

Consider Dataframe

	Name	last	email
0	John	Doe	johndoe@gmail.com
1	Jane	Doe	Janedoe@gmail.com.
2	James	Brad	JamesB@gmail.com.
3	Jennifer	Anniston	JenniferAnniston@gmail.com.

*We use conditionals such as `'=='`, `'&'`, `'|'` to filter our DataFrame.

→ (df["last"] == "Doe") → filt

↳ this will return a Series (Array) of True/False values.

[True, True, False, False] → df[[True, True, False, False]]

→ df[filt] will give indexes 0, 1

↳ it is a sub-DataFrame to the original DF (it is filtered.)

We can do df.loc[filt] instead → this has Advantage

*(df.loc processes the Array of T/F values)

We can do

→ df.loc[filt, ["Name", "email"]]

* to get only Name & email fields.

Using Conditionals. '&' and '|'

'and'

filt = (df["last"] == "Doe") & (df["name"] == "John")

df.loc[filt]

returns Dataframe with index '0'.

Negation → '~'
(Not)

df.loc[~filt]

returns Dataframe with indexes '1', '2', '3'

||ly we use '|' for or.

We can also use '>', '<', '>=', '<=' for — filtering.

Examples

```
→ high_salary = (df["salary"] > 70000)
→ df[high_salary]
→ df.loc[high_salary, ["Country", "prog_lang",
                        "salary"]]
```

Now let's say we want to see if the "Country" is in a set of our preferred countries

```
→ countries = ["USA", "UK", "India", "Germany", "Canada"]
```

We use ".isin()" function to check if "Country" is in the array
(python-pandas fun())

```
→ filt1 = df["Country"].isin(countries)
```

Now let's say we want to see how many use "python" in their work.

consider "prog_lang" format as (HTML, CSS; Java; python ...)

We use a python-pandas func() ".str.Contains()" to check if substring is present inside a main string.

we need to put a additional parameter 'na = False' to deal with NaN values in the DataFrame. *

(Null or None)

NaN = None = NULL

Basically NULL values in Pandas DataFrame.

~~filt2 = df["language"~~

```
filt2 = df["prog_lang"].str.contains("python", na=False)
df.loc[filt2].
```

filtering is one of the first things we do with Data