

(32)

pandas

allows us to work with the data structured in row, column format

python module for data manipulation

cleaning data, add rows, add columns,

remove them,

managing data in a

way similar to excel.

making data more

accessible or

in other words,

operation ready

import pandas as pd

pd.read_csv()

columns separated by commas

to load data

path of the csv file

similarly read_json()

read_html()

read_excel()

functions

URL

local device as well

example, iris = pd.read_csv()

iris

* iris csv had

150 rows x 5 columns

the pandas module, takes the first row as header automatically

since this csv did not have proper header row

the 1st data row became header

and pandas reported 149 rows x 5 columns in dataset

type (iris)

(return type of read-csv)

→ pandas dataframe

↓
2D dataholder with certain rows and columns.

* `df = iris` (assigns same pointer, changes made using one, will reflect in the other as well).

↙
assigns reference to the variable

↓
shallow copy

↓
`df = iris.copy()`

↘ relates df to a new copy made of iris and referenced using df

also have a corresponding `df.tail()`

* `df.head()` → brief look at the data set

↓
prints few starting elements of the dataset

argument sends how many elements we want to see

↓
by default = 5

↓
allows us to do basic analysis of the dataset wrt types and values.

* `df.shape` → (the resolution of the dataframe is no. of rows and no. of columns in total)

↓
`df.shape`

* `df.columns` → to look at current headers

↓
to change headers, we need to send in a list of apt. size.

↓
`df.columns = ['sl', 'sw', 'pl', 'pw', 'flower-type']`

* lost the row which was used for columns

34

* df.dtypes

↓
returns the type of values stored in each column

↓
i.e. sl : Float 64
sw : Float 64
pl : Float 64
pw : Float 64
flower type : object

* df.describe()

→ describes our data frame,
column wise

valid no of values
in column
(ignores NaN)

Not valid value

spread of
the data

← i) count

ii) mean

iii) std → standard deviation

iv) min

v) 25%

vi) 50%

vii) 75%

viii) max

percentile

↓
value having 25%
values less than it

to access a column of data frame

df.sl

df["sl"]

* df.isnull()

→ wherever null values are
present, it returns true,
otherwise false elementwise
in the data

* df.isnull().sum()

→ sum columnwise

→ accessing part of data. → slice of the data.

* `df.iloc [1:4, 2:3]`

↓

pd

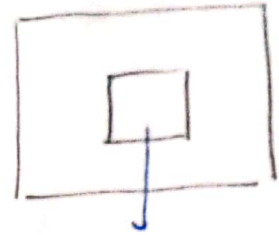
1 1.3

2 1.5

3 2.4

rows 1-3,

cols 2



to select a part of our data.

↓

direct slicing is not allowed.

* scikit → modelling data

↓

not on loading, manipulating and summarizing data features.

* to delete rows in dataframe

↓
`df.drop()`

label of

the row to be excluded as argument

makes a copy without the specified row
↓
to delete,

indexing doesn't change automatically

↓
difference b/w label and position

`df = df.drop(0)`
or

`df.drop(0, inplace = True)`

↓
(by default, inplace is false)

(*) `df.index` → to look at labels of the dataframe

↓

`df.index[0]`

→ gives label of element / row with position 0.

to drop by position

`df.drop(df.index[0])`

36

* to drop multiple rows at once, use comma

↓
`df.drop(df.index[[0,1]], inplace=True)`

↓
gives list of labels to drop

* `df.sl > 5`

↓
runs condition on column
and returns a set of bools
based on the condition
↓
(returns a column)

set operations on
dataframes

↗ like a boolean array
* `df[df.sl > 5]`

↓
returns complete rows of dataframe
in which the specified condition,
in this case the column
sl of df satisfied.

to use multiple
conditions, we use &

3 different kinds of flowers in our data set Iris

- a) Iris - setosa
- b) Iris - versicolor
- c) Iris - virginica

* `df[column].value_counts()`
↓
frequency of different
unique values.

* to rename single
pandas column, we
use

`df.rename(
columns = {3,`

`inplace=True)` `df[df.flower_type == 'Iris - setosa'].describe()`

↓
provides interesting insights
based on dataframe selected
using describe function of pandas

↓
mean of different flower type comparison

by storing in different classes

≠ add a row

index based

* `df.iloc()`
 ↳ position based

* inserting using position

* if already present

`df.loc()`
 ↳ label based

`df.loc[0] = [1, 2, 3, 4, "Iris-setosa"]`

↓
 adds a row at the end, with label 0

↓
 Using `iloc`, we can insert on position basis

↓
 overwrites if already present.

* based on dropping operations or data manipulation operations, the dataframes may have unordered indices/labels, which may cause **referencing inconsistency**.

↓
 `df.reset_index()`

named index

* not making changes in df

* giving a new dataframe

↓
 to place changes in the same dataframe

↓
 adds a new column with old labels by putting in indices reset to 0.

↓
 avoid adding that by adding a drop argument

↓
 `df.reset_index(drop=True, inplace=True)`

`df.index` → describes the label of the dataframe

38) # deleting columns in dataframe

`df.drop('sl', axis=1)` → drops 'sl' column

axis → in arguments refers to the axis about which we operate in MD arrays.

axis = 1 refers to column space

↓
think of it as the **coordinate**

axis

0 → x axis

1 → y axis

2 → z axis

⋮

(based on index)

`df.drop('sl', axis=1, inplace=True)`

* alternate method to drop

`del df['sl']`

↓
will make changes in df itself

adding columns in dataframe

↓
for ex. if we want column with diff of pw and pl

↓
we used using

↓
`df["diff-pl-pw"] = df["pl"] - df["pw"]`

similar to dictionary

Reordering columns

`df = df[['sl', 'sn', 'pl', 'pw', 'diff', 'type']]`

df["abc"] = 1

→ column of all ones

Handling NAN

↓
we can do two things to handle NAN values, i.e.

- (a) filling them with certain **defined values**.
- (b) or removing them

* trying indexing
using iloc. to fetch
certain indexed row

- ↓
i.e. (a) drop na
(b) ~~remove~~ na
fill

* inserting NAN in dataframe (for demonstration purposes)

df.iloc[2:4, 1:3] = **np.nan** → constant in
numpy

* describe with
ignore these values

import numpy as np.

* Handling methods.

to drop entire columns with
NA, axis=1.

(i) df.dropna() → drops rows with any one or more
values = NaN.

↓
again df doesn't
change, without
using inplace.

→ df.dropna(inplace=True)
↓
and then reset

(ii) filling na → this can be done in many ways i.e.
different values can be

mean of the
specific column

inserted based on our
requirements

df ~~codes~~ **.sw.fillna(df** ~~codes~~ **.sw.mean())**

the column whose nan
needs to be

* again not inplace *

handled

(40)

```
df.sw.fillna(df.sw.mean(), inplace=True)
```

* to insert mean of that flower class to maintain a sense of distribution

find the replacement class i.e

```
a = df[df.flower_type == '_']
```

then

```
df.sw.fillna(a.sw.mean(), inplace=True)
```

```
df["Age"] = df.groupby("Survived").transform
```

```
(lambda x: x.fillna(x.mean()))
```

Handling strings in data.

↓
it is much more easier to operate on numerical data as compared to string data
↓
for example, the facebook or advertisement example of reg. male and female data, and how easy it would be to perform computations on that data if numerical
↓
so we change ~~numeric~~ string based data to numeric data.

string based data

```
df["Gender"] = "Male"  
df.iloc[0:10, 6] = "Female"  
def f  
df["sex"] = df.Gender.apply(f)  
def f(s):  
    if s == "Male":  
        return 0  
    else:  
        return 1  
del df["gender"]  
del df["gender"]
```

inplace doesn't work
↑
applying function f on the selected column and puts it into a new column by the name of sex
↓
and then delete gender column
This puts comparative association based on magnitude 0 and 1
better to use one-hot encoding

(92)

* Titanic Dataset

(i) useful columns.

(ii) data processing requirements.

↓
useless columns drops
adding new columns.

* understanding what different features mean

↓
useful columns
← to our final output

analysing
preprocessing
required about
the data.

↓
drop anything else
↓
adding new columns

→ Data cleaning and analysis - Titanic dataset

* i) understanding meaning of each column

ii) analyse columns which can be deleted

iii) replace columns with string (if any) with null values for analysts

iv) Fill the null values of the age column

↓
Fill mean survived age (mean age of
survived people)

in the column where the person has survived
and mean not survived age (mean age of
people who didn't survive)

v) draw graph for analysis

C = Cherbourg

Q = Queenstown

S = Southampton