Pandas

---

import pandas **as** pd

df **=** pd**.**read_csv('data/survey_results_public.csv') >> if I want to read data from a csv file.

Now If I printed that (df) >> it will show me a data frame consists of only 20 columns even if the data frame has more than 20 columns.

pd**.**set_option('display.max_columns', 85) >> if I want to see all the columns not just 20 , 85 is the number of columns in my data frame.

pd**.**set_option('display.max_rows', 85) >> if I want to see all the rows.

df.head() >> show me only the first 5 rows , we also can pass any number we want to see.

df.tail(number) >> show me only the last 5 rows , we also can pass any number we want to see.

df.shape >> returns the number of (rows , columns) , shape is an attribute not a method so we don’t use parentheses () with it.

df.info() >> returns the number of (rows , columns) and the data type of each column.

schema_df **=** pd**.**read_csv('data/survey_results_schema.csv') >> if I want to know the meaning of each column name.

```
people = {

   "first": ["Corey", 'Jane', 'John'],
   "last": ["Schafer", 'Doe', 'Doe'],
   "email": ["CoreyMSchafer@gmail.com", 'JaneDoe@email.com',
'JohnDoe@email.com']

}
```

If I want to create a data frame from that dictionary:
df = pd.DataFrame(people)

df['email'] or df.email >> if I want to get the email column from the data frame.

df[['last', 'email']] >> to get the “last” and “email” columns I pass them as a list.

df.iloc[rows, columns]>> integer location: I use iloc to access rows and columns by their index.

df.loc[rows,columns]>> here we can access columns by their names “search by labels”.

df.columns >> returns all column names in our data frame.

df.index >> returns all indexes names in our data frame

df = pd.read_csv('data/survey_results_public.csv', index_col='Respondent') >> if I want to set a specific column to be index column.

schema_df = pd.read_csv('data/survey_results_schema.csv', index_col='Column') >> if I want to set a specific column to be index column in schema.

schema_df.sort_index() >> to sort the schema.

To change a specific column name:
df.rename(columns={‘old_value’ : ‘new_value’ , ‘old_value2’:’ new_value2’,... }) >> that will make a copy of new changes but the original data frame will not be affected. If I want the original data frame to change so i should write it like that: df.rename(columns={‘old_value’ : ‘new_value’ , ‘old_value2’:’ new_value2’,... },inplace = True)

if I had this data frame:

	first_name	last_name	email
0	Corey	Schafer	CoreyMSchafer@gmail.com
1	Jane	Doe	JaneDoe@email.com
2	John	Doe	JohnDoe@email.com

and I want to change a single raw: let’s say the last raw:

we can write rhat df.loc[2] = ['John', 'Smith', 'JohnSmith@email.com']
but we have to fill each column value.

So , as a better solution we can write that:
df.loc[2, ['last', 'email']] = ['Doe', 'JohnDoe@email.com']
we use loc and chose the columns we want to change.

To change a single value:
df.loc[2, 'last'] = 'Smith'
or we can use **at()** method >> we can only use it if it is a single value that we want to change: df.at[2, 'last'] = 'Doe'

another way is to use filter:

filt = (df[‘email] == 'JohnDoe@email.com')
df.loc(filt , ‘last’) = ‘smith’

now if I want to change every email in the data frame to a lower case:
df[‘email’].str.lower()

APPLY() >> apply a function to every single row or column
If I wrote: df[‘email’]=df[‘email’].apply(len) >> that will return the length ot every email in data frame.

APPLYMAP() >> ONLY WORKS ON DATA FRAMES NOT SERIES OBJECTS. Apply a function to every individual element of the data frame.

MAP() >> ONLY WORKS ON SERIES OBJECTS AND USED TO SUBSTITUTE A VALUE WITH ANOTHER ONE.

df['first']**.**map({'Corey': 'Chris', 'Jane': 'Mary'}) >> here we just change two values so it will change the two values and make all other values = nan

so if we want to change only two values and keep the rest of the other values we just use **replace() :**
df['first'] **=** df['first']**.**replace({'Corey': 'Chris', 'Jane': 'Mary'})

---

df**.**rename(columns**=**{'ConvertedComp': 'SalaryUSD'}, inplace**=True**) >> if I want to change a column name.

If I want to add a column to my data frame:

df['full_name'] = df['first'] + ' ' + df['last']

now if I want to delete the ‘first name’ and ‘last name’ columns: **drop()**

df.drop(columns=['first', 'last'], inplace=**True**)

if I want to return the deleted columns:
df[['first', 'last']] = df['full_name'].**str.split**(' ', **expand=**True)

if I want to add a new row:
df.append({'first': 'Tony'}, **ignore_index=True**)

if I want to merge to data frames or to add a data frame to another:
df.append(df2, ignore_index=**True**, sort=**False**)

if I want to delete a row knowing its index:
df.drop(index=4)
if I don’t know the index but I know his last name so we can use filter:
filt = df['last'] == 'Doe'
df.drop(index=df[filt].index)

---

df.sort_values(by='last') >> If I want to sort the data frame by the ‘last; column in ascending order.

df.sort_values(by='last', ascending=**False**) >> If I want to sort the data frame by the ‘last; column in descending order.

df.sort_values(by=['last', 'first'], ascending=**False**) >> sorting by more than column

df.sort_values(by=['last', 'first'], ascending=[**False**, **True**], inplace=**True**) >> sorting by last in descending order and by first in ascending order.

df.sort_index() >> sorting by indexes [oldest to newest].

df['ConvertedComp']**.nlargest(10) >> to get the largest of a row.**

df.**nsmallest**(10, 'ConvertedComp') **>> to get the smallest of a row.**