Pandas

import pandas as pd

pd.__version__

series

pd.Series(['San Francisco', 'San Jose', 'Sacramento'])

pd.series(data) — look like exel sheet

we can add two series then the same index's data will add up and different index's will show as NaN

d = pd.DataFrame(data,index,column)

in data we can pass array it will create a spreadsheet

column

d[column_name] — to access data from dataframe, we can also pass multiple column as list

to create a column

d[name of new column] = d[] + d[] … or a list

to remove

d.drop(column name,axis=1 , inplace = True) — for row axis =0. and inplace is imp as it make changes permanent

Row

d.loc[row name] or d.iloc[row index] — to access data as row we can pass list also

d.loc[row name,column name] — we can also pass list

compare d>0 return an dataframe of bool

d[d[column_name]>0] — return all the rows where it is true (first inner bracket return bool and second take that bool and return rows where bool is true

we can also pass multiple condition

df.reset_index() — to reset the index to 0,1 (in bracket we can pass column too to set as index)

Data frame

python frame - zip(list1, list2) - tuple list - eg ((a, b), (c, d), (), ….)

```
city_names = pd.Series(['San Francisco', 'San Jose', 'Sacramento'])
population = pd.Series([852469, 1015785, 485199])
pd.DataFrame({ 'City name': city_names, 'Population': population })
```

	City Name	Population
0	San Francisco	852469
1	San Jose	1015785
2	Sancrameto	9038943

df.head() - to print only few top entries since df can be too large and print all will just gonna fill the terminal

we can pass tuple to create multipy level index in dataframe

eg -

	1
G1	2
	3
	1
G2	2
	3

df.hist('graph_title') - graphing lets you quickly study the distribution of values in a column
cities = pd.DataFrame({ 'City name': city_names, 'Population': population })
print(type(cities['City name']))
cities['City name']
to acess - d.loc[g1] or d.loc[g1][1]
to give name to index - d.index.name = [' ', ' ', ….]
d.xs(,level = index_name) - to grab inner index
d.dropna() - it gonna drop all row with missing value and for column put axis=1
d.fillna(value=' ') - replace all nan value
d.groupby('column_name').fun()
eg mean(), sum(), max(), min(), describe()
d.transpose()
pd.concat([d1, d2]) - to concatinate
pd.merge(d1, d2, how="inner", on="on which row u want to merge") - for along column put axis =1
d.join(d2) - to join based on index key
d.[].unique() - only unique value
.value_counts() — count unique values
.apply(function() or lambda)
.sort_value(column_name)
.drop(column_name, axis=1, inplace=true)