Programming and Data Structures with Python

Lecture 18, 25 November 2021

Pandas (Python and data analysis)

- Built on top of numpy

Series and data frames

- Numpy defines homogeneous n-dimensional arrays
- Data science works with tables: 2-dimensional arrays
- Pandas has two fundamental data structures
  - Series : A column of data
  - Data Frame : A table of data

Key difference

- Numpy indices are always [0..n-1] in each dimension
- Pandas allows more flexible “named” indices for rows and columns
  - Dictionary vs list

Load pandas

- Don't need to import numpy unless one is separately using numpy arrays

```
1 import pandas as pd
```

Create a series

- Convert a sequence into a series (column)

```
1 h = ('AA', '2012-02-01', 100, 10.2)
2 s = pd.Series(h)
3 type(s)
```

```
pandas.core.series.Series
```

```
1 s
```

```
0           AA
1    2012-02-01
2           100
3          10.2
dtype: object
```

- Convert a dictionary to a series
- Keys become "row indices"

```
1 d = {'name' : 'IBM', 'date' :'2010-09-08', 'shares' : 100, 'price' : 10.2}
2 ds = pd.Series(d)
3 type(ds)
```

```
pandas.core.series.Series
```

```
1 ds
```

```
name            IBM
date     2010-09-08
shares          100
price          10.2
dtype: object
```

Creating an index

```
1 f = ['FB', '2001-08-02', 90, 3.2]
```

```
fs = pd.Series(f, index = ['name','date', 'shares', 'price'])
```

```
fs
```

```
name             FB
date     2001-08-02
shares           90
price           3.2
dtype: object
```

Accessing elements

- Use named index, or position
- Use slices, sublists

```
fs['shares']
```

```
90
```

```
fs[0]
```

```
'FB'
```

```
fs[0:2]
```

```
name           FB
date    2001-08-02
dtype: object
```

```
fs[[0,2]]
```

```
name      FB
shares    90
dtype: object
```

```
fs['name':'price']
```

```
name             FB
date     2001-08-02
shares           90
price           3.2
dtype: object
```

```
fs[0:3]
```

```
name             FB
date     2001-08-02
shares           90
dtype: object
```

Data frames

- A table is a sequence of columns
- A data frame is a sequence of series

```
data1 = [ ['AA', 'IBM', 'GOOG'],
          ['2001-12-01', '2012-02-10', '2010-04-09'],
          [100, 30, 90],
          [12.3, 10.3, 32.2]
        ]
df1 = pd.DataFrame(data1)
```

```
df1
```

	0	1	2
0	AA	IBM	GOOG
1	2001-12-01	2012-02-10	2010-04-09
2	100	30	90
3	12.3	10.3	32.2

```
data2 = {'name' : ['AA', 'IBM', 'GOOG'],
         'date' : ['2001-12-01', '2012-02-10', '2010-04-09'],
         'shares' : [100, 30, 90],
         'price' : [12.3, 10.3, 32.2]
}
df2 = pd.DataFrame(data2)
```

```
1 df2
```

	name	date	shares	price
0	AA	2001-12-01	100	12.3
1	IBM	2012-02-10	30	10.3
2	GOOG	2010-04-09	90	32.2

Add a column

```
1 df2['owner'] = 'Unknown'
2 # df2['owner'] = ['a','b']
```

```
1 df2
```

	name	date	shares	price	owner
0	AA	2001-12-01	100	12.3	Unknown
1	IBM	2012-02-10	30	10.3	Unknown
2	GOOG	2010-04-09	90	32.2	Unknown

Add row indices

```
1 df2.index = ['one','two','three']
```

```
1 df2
```

	name	date	shares	price	owner
one	AA	2001-12-01	100	12.3	Unknown
two	IBM	2012-02-10	30	10.3	Unknown
three	GOOG	2010-04-09	90	32.2	Unknown

Convert one of the columns into an index

```
1 df2 = df2.set_index(['name'])
```

```
1 df2
```

	date	shares	price	owner
name				
AA	2001-12-01	100	12.3	Unknown
IBM	2012-02-10	30	10.3	Unknown
GOOG	2010-04-09	90	32.2	Unknown

Replace an index

```
1 df2 = df2.set_index(['price'])
```

```
1 df2
```

	date	shares	owner
price			
12.3	2001-12-01	100	Unknown
10.3	2012-02-10	30	Unknown
32.2	2010-04-09	90	Unknown

Use multiple columns for indexing

```
1 df2 = pd.DataFrame(data2)
2 df2['owner'] = 'Unknown'
```

```
df2 = df2.set_index(['name','price'])
```

```
df2
```

		date	shares	owner
name	**price**			
AA	**12.3**	2001-12-01	100	Unknown
IBM	**10.3**	2012-02-10	30	Unknown
GOOG	**32.2**	2010-04-09	90	Unknown

Accessing values in a dataframe

By column index

```
df2[['shares','date']]   # Similar to the projection operation in relational algebra / RDBMS
```

		shares	date
name	**price**		
AA	**12.3**	100	2001-12-01
IBM	**10.3**	30	2012-02-10
GOOG	**32.2**	90	2010-04-09

By row index

```
df2.loc['AA']
```

	date	shares	owner
price			
12.3	2001-12-01	100	Unknown

Individual element by position

```
df2.loc['AA','shares']
```

```
price
12.3    100
Name: shares, dtype: int64
```

Slices, etc

```
df2.loc[:,'shares']
```

```
name  price
AA    12.3     100
IBM   10.3      30
GOOG  32.2      90
Name: shares, dtype: int64
```

```
df2 = pd.DataFrame(data2)
df2['owner'] = 'Unknown'
df2 = df2.set_index(['name'])
df2
```

	date	shares	price	owner
name				
AA	2001-12-01	100	12.3	Unknown
IBM	2012-02-10	30	10.3	Unknown
GOOG	2010-04-09	90	32.2	Unknown

```
df2.loc['AA':'IBM','shares':'owner']
```

name	shares	price	owner

- Unlike series, cannot use position indices if "real" index exists

```
df2 = pd.DataFrame(data2)
df2['owner'] = 'Unknown'
df2
```

	name	date	shares	price	owner
0	AA	2001-12-01	100	12.3	Unknown
1	IBM	2012-02-10	30	10.3	Unknown
2	GOOG	2010-04-09	90	32.2	Unknown

```
df2.loc[0:1]
```

	name	date	shares	price	owner
0	AA	2001-12-01	100	12.3	Unknown
1	IBM	2012-02-10	30	10.3	Unknown

```
df2 = df2.set_index(['name'])
df2
```

name	date	shares	price	owner
AA	2001-12-01	100	12.3	Unknown
IBM	2012-02-10	30	10.3	Unknown
GOOG	2010-04-09	90	32.2	Unknown

```
df2.loc[0:2]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-63-8c062d306d54> in <module>()
----> 1 df2.loc[0:1]

 7 frames
/usr/local/lib/python3.7/dist-packages/pandas/core/indexes/base.py in _invalid_indexer(self, form, key)
   3269         """
   3270         raise TypeError(
-> 3271             f"cannot do {form} indexing on {type(self).__name__} with these "
   3272             f"indexers [{key}] of type {type(key).__name__}"
   3273         )

TypeError: cannot do slice indexing on Index with these indexers [0] of type int
```

SEARCH STACK OVERFLOW

Reading csv files

```
import io
from google.colab import files
uploaded = files.upload()
```

Choose files 3 files

- **cast.csv**(text/csv) - 4289363 bytes, last modified: 19/05/2018 - 100% done
- **housing.csv**(text/csv) - 1423529 bytes, last modified: 04/02/2021 - 100% done
- **titles.csv**(text/csv) - 1098719 bytes, last modified: 19/05/2018 - 100% done

```
Saving cast.csv to cast.csv
Saving housing.csv to housing.csv
Saving titles.csv to titles.csv
```

```
casts = pd.read_csv(io.BytesIO(uploaded['cast.csv']),index_col=None)
titles = pd.read_csv(io.BytesIO(uploaded['titles.csv']),index_col=None)
```

```
# casts = pd.read_csv('cast.csv',index_col=None)
# titles = pd.read_csv('titles.csv',index_col=None)
```

```
casts.head()
```

	title	year	name	type	character	n
0	Closet Monster	2015	Buffy #1	actor	Buffy 4	31.0
1	Suuri illusioni	1985	Homo $	actor	Guests	22.0
2	Battle of the Sexes	2017	$hutter	actor	Bobby Riggs Fan	10.0
3	Secret in Their Eyes	2015	$hutter	actor	2002 Dodger Fan	NaN

```
1 casts.head(7)
```

	title	year	name	type	character	n
0	Closet Monster	2015	Buffy #1	actor	Buffy 4	31.0
1	Suuri illusioni	1985	Homo $	actor	Guests	22.0
2	Battle of the Sexes	2017	$hutter	actor	Bobby Riggs Fan	10.0
3	Secret in Their Eyes	2015	$hutter	actor	2002 Dodger Fan	NaN
4	Steve Jobs	2015	$hutter	actor	1988 Opera House Patron	NaN
5	Straight Outta Compton	2015	$hutter	actor	Club Patron	NaN
6	Straight Outta Compton	2015	$hutter	actor	Dopeman	NaN

```
1 titles.tail()
```

	title	year
49995	Rebel	1970
49996	Suzanne	1996
49997	Bomba	2013
49998	Aao Jao Ghar Tumhara	1984
49999	Mrs. Munck	1995

Filtering data

- Movies after 1985
- Analogous to `select` in relational algebra or SQL

```
1 after85 = titles[titles['year'] > 1985]
2 after85
```

	title	year
0	The Rising Son	1990
2	Crucea de piatra	1993
3	Country	2000
4	Gaiking II	2011
5	Medusa (IV)	2015
...	...	...
49990	Junebug	2005
49993	Corruption.Gov	2010
49996	Suzanne	1996
49997	Bomba	2013
49999	Mrs. Munck	1995

29814 rows × 2 columns

- Movies in years 1990 - 1999

```
1 t = titles
2 movies90 = t[(t['year'] >= 1990) &(t['year'] < 2000)]
3 movies90
```

	title	year
0	The Rising Son	1990
2	Crucea de piatra	1993
12	Poka Makorer Ghar Bosoti	1996
19	Maa Durga Shakti	1999
24	Conflict of Interest	1993
...	...	...
49969	Chi mei wang liang	1998

Sorting

- All movies named 'Macbeth'
- Sort by year

```
1 macbeth = t[t['title'] == 'Macbeth']
```

```
1 macbeth
```

	title	year
4226	Macbeth	1913
9322	Macbeth	2006
11722	Macbeth	2013
17166	Macbeth	1997
25847	Macbeth	1998

```
1 macbeth = macbeth.sort_values('year')
```

```
1 macbeth
```

	title	year
4226	Macbeth	1913
17166	Macbeth	1997
25847	Macbeth	1998
9322	Macbeth	2006
11722	Macbeth	2013

```
1 macbeth = macbeth.sort_index()
```

```
1 macbeth
```

	title	year
4226	Macbeth	1913
9322	Macbeth	2006
11722	Macbeth	2013
17166	Macbeth	1997
25847	Macbeth	1998

Summaries and descriptive statistics

```
1 titles.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   title   50000 non-null  object
 1   year    50000 non-null  int64
dtypes: int64(1), object(1)
memory usage: 781.4+ KB
```

```
1 casts.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 75001 entries, 0 to 75000
Data columns (total 6 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   title      75001 non-null  object
 1   year       75001 non-null  int64
 2   name       75001 non-null  object
 3   type       75001 non-null  object
 4   character  75001 non-null  object
 5   n          46035 non-null  float64
dtypes: float64(1), int64(1), object(4)
memory usage: 3.4+ MB
```

```
titles.describe()
```

	year
count	50000.000000
mean	1986.106120
std	29.293942
min	1900.000000
25%	1967.000000
50%	1996.000000
75%	2011.000000
max	2024.000000

```
casts.describe()
```

	year	n
count	75001.000000	46035.000000
mean	1990.536473	16.814359
std	26.748233	24.695616
min	1912.000000	1.000000
25%	1974.000000	4.000000
50%	2002.000000	10.000000
75%	2012.000000	21.000000
max	2023.000000	701.000000

Descriptive statistics for categorical data

```
casts['name'].describe()
```

```
count           75001
unique          29319
top       Ernie Adams
freq              431
Name: name, dtype: object
```

```
# housing = pd.read_csv('housing.csv', index_col=None)
housing = pd.read_csv(io.BytesIO(uploaded['housing.csv']), index_col=None)
```

```
housing.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   longitude           20640 non-null  float64
 1   latitude            20640 non-null  float64
 2   housing_median_age  20640 non-null  float64
 3   total_rooms         20640 non-null  float64
 4   total_bedrooms      20433 non-null  float64
 5   population          20640 non-null  float64
 6   households          20640 non-null  float64
 7   median_income       20640 non-null  float64
 8   median_house_value  20640 non-null  float64
 9   ocean_proximity     20640 non-null  object
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

```
housing.describe()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_va
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20640.000000	20640.000000	20640.000000	20640.00
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	1425.476744	499.539680	3.870671	206855.81
std	2.003532	2.135952	12.585558	2181.615252	421.385070	1132.462122	382.329753	1.899822	115395.61
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	1.000000	0.499900	14999.00
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	787.000000	280.000000	2.563400	119600.00
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	1166.000000	409.000000	3.534800	179700.00
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	1725.000000	605.000000	4.743250	264725.00
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.000000	6082.000000	15.000100	500001.00

✓ 0s completed at 2:49 PM