Lecture 14, 22 September 2022

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

In [1]:
```
import pandas as pd
```

Create a series

- Convert a sequence into a series (column)

In [2]:
```
h = ('AA', '2012-02-01', 100, 10.2)
s = pd.Series(h)
s
```

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

Convert a dictionary to a series

- Keys become "row indices"

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

Out[3]:
```
name             IBM
date      2010-09-08
shares           100
price           10.2
dtype: object
```

Creating an index

In [4]:
```
f = ['FB', '2001-08-02', 90, 3.2]
fs = pd.Series(f, index = ['name','date', 'shares', 'price'])
fs
```

Out[4]:
```
name              FB
date      2001-08-02
shares            90
price            3.2
dtype: object
```

Accessing elements

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

In [5]:
```
fs['shares']
```

Out[5]:
```
90
```

In [6]:
```
fs[0]
```

Out[6]:
```
'FB'
```

In [7]:
```
fs[0:2]
```

Out[7]:
```
name            FB
date    2001-08-02
dtype: object
```

- Subset of rows

In [8]:
```
fs[[0,2]]
```

Out[8]:
```
name      FB
shares    90
dtype: object
```

- Order is important

In [9]:
```
fs[['price','name']]
```

Out[9]:
```
price    3.2
name      FB
dtype: object
```

- Slice by index or position
- If by index, both endpoints are included

In [10]:
```
fs['name':'price']
```

Out[10]:
```
name              FB
date      2001-08-02
shares            90
price            3.2
dtype: object
```

In [11]:
```
fs[0:3]
```

Out[11]:
```
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
- Each column must have the same length, otherwise an error

In [12]:
```
data1 = {'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]
}
df1 = pd.DataFrame(data1)
df1
```

Out[12]:

	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

- Like adding a key to a dictionary
- If initialized to a single value, that value is copied in each row

In [13]:
```
df1['owner'] = 'Unknown'
df1
```

Out[13]:

	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

- Can also provide an explicit sequence of values, must match column length

In [14]:
```
df2 = pd.DataFrame(data1)
df2['owner'] = ['a','b','c']
df2
```

Out[14]:

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

Add row indices

In [15]:
```
df1.index = ['one','two','three']
df1
```

Out[15]:

	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

In [16]:
```
df1.set_index(['name'])
```

Out[16]:

	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

- This returns a new data frame, does not update the existing one in place

In [17]:
```
df1
```

Out[17]:

	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

- Hence, reassign to update

In [18]:
```
df1 = df1.set_index(['name'])
df1
```

Out[18]:

	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

In [19]:
```
df1 = df1.set_index(['price'])
df1
```

Out[19]:

	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

In [20]:
```
df1 = pd.DataFrame(data1)
df1['owner'] = 'Unknown'
df1 = df1.set_index(['name','price'])
df1
```

Out[20]:

		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

- Index column may have duplicates

In [21]:
```
df2 = pd.DataFrame(data1)
df2['owner'] = 'Unknown'
df2 = df2.set_index(['owner'])
df2
```

Out[21]:

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

Accessing values in a dataframe

By column index

- Similar to projection in relational algebra
- List of columns to keep, order matters

In [22]:
```
df1[['shares','date']]
```

Out[22]:

		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

In [23]:
```
df1.loc['AA']
```

Out[23]:

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

- Row index not required to be unique valued

In [24]:
```
df2.loc['Unknown']
```

Out[24]:

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

Individual element by position

In [25]:
```
df1.loc['AA','shares']
```

Out[25]:
```
price
12.3    100
Name: shares, dtype: int64
```

Slices, etc

In [26]:
```
df1.loc[:,'shares']
```

Out[26]:
```
name  price
AA    12.3     100
IBM   10.3      30
GOOG  32.2      90
Name: shares, dtype: int64
```

In [27]:
```
df1 = pd.DataFrame(data1)
df1['owner'] = 'Unknown'
df1 = df1.set_index(['name'])
df1
```

Out[27]:

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

In [28]:
```
df1.loc['AA':'IBM','shares':'owner']
```

Out[28]:

name	shares	price	owner
AA	100	12.3	Unknown
IBM	30	10.3	Unknown

- Cannot use `loc` with position indices if "real" index exists
- Use `iloc` instead

In [29]:
```
df1.loc[0:1]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/tmp/ipykernel_74828/521513687.py in <module>
----> 1 df1.loc[0:1]

~/miniconda3/lib/python3.7/site-packages/pandas/core/indexing.py in __getitem__(self, key)
    929
    930             maybe_callable = com.apply_if_callable(key, self.obj)
--> 931             return self._getitem_axis(maybe_callable, axis=axis)
    932
    933     def _is_scalar_access(self, key: tuple):

~/miniconda3/lib/python3.7/site-packages/pandas/core/indexing.py in _getitem_axis(self, key, axis)
   1140         if isinstance(key, slice):
   1141             self._validate_key(key, axis)
-> 1142             return self._get_slice_axis(key, axis=axis)
   1143         elif com.is_bool_indexer(key):
   1144             return self._getbool_axis(key, axis=axis)

~/miniconda3/lib/python3.7/site-packages/pandas/core/indexing.py in _get_slice_axis(self, slice_obj, axis)
   1174
   1175         labels = obj._get_axis(axis)
-> 1176         indexer = labels.slice_indexer(slice_obj.start, slice_obj.stop, slice_obj.step)
   1177
   1178         if isinstance(indexer, slice):

~/miniconda3/lib/python3.7/site-packages/pandas/core/indexes/base.py in slice_indexer(self, start, end, step, kind)
   5683         slice(1, 3, None)
   5684         """
-> 5685         start_slice, end_slice = self.slice_locs(start, end, step=step)
   5686
   5687         # return a slice

~/miniconda3/lib/python3.7/site-packages/pandas/core/indexes/base.py in slice_locs(self, start, end, step, kind)
   5885         start_slice = None
   5886         if start is not None:
-> 5887             start_slice = self.get_slice_bound(start, "left")
   5888         if start_slice is None:
   5889             start_slice = 0

~/miniconda3/lib/python3.7/site-packages/pandas/core/indexes/base.py in get_slice_bound(self, label, side, kind)
   5795         # For datetime indices label may be a string that has to be converted
   5796         # to datetime boundary according to its resolution.
-> 5797         label = self._maybe_cast_slice_bound(label, side)
   5798
   5799         # we need to look up the label

~/miniconda3/lib/python3.7/site-packages/pandas/core/indexes/base.py in _maybe_cast_slice_bound(self, label, side, kind)
   5747         # reject them, if index does not contain label
   5748         if (is_float(label) or is_integer(label)) and label not in self._values:
-> 5749             raise self._invalid_indexer("slice", label)
   5750
   5751         return label

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

In [30]:
```
df1.iloc[0:2]
```

Out[30]:

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

Reading csv files

In [31]:
```
casts = pd.read_csv('cast.csv',index_col=None)
titles = pd.read_csv('titles.csv',index_col=None)
```

In [32]:
```
casts.head()
```

Out[32]:

	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

In [33]:
```
casts.head(7)
```

Out[33]:

	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

In [34]:
```
titles.tail()
```

Out[34]:

	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
- Like `select` in relational algebra

In [35]:
```
after85 = titles[titles['year'] > 1985]
after85
```

Out[35]:

	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

In [36]:
```
t = titles
movies90 = t[(t['year'] >= 1990) &(t['year'] < 2000)]
movies90
```

Out[36]:

	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
49979	Gagay: Prinsesa ng brownout	1993
49987	I Won't Dance	1992
49996	Suzanne	1996
49999	Mrs. Munck	1995

4803 rows × 2 columns

Sorting

- All movies named 'Macbeth'
- Sort by year

In [37]:
```
macbeth = t[t['title'] == 'Macbeth']
macbeth
```

Out[37]:

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

In [38]:
```
macbeth = macbeth.sort_values('year')
macbeth
```

Out[38]:

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

In [39]:
```
macbeth = macbeth.sort_index()
macbeth
```

Out[39]:

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

Summaries and descriptive statistics

In [40]:
```
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
```

In [41]:
```
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
```

In [42]:
```
titles.describe()
```

Out[42]:

	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

In [43]:
```
casts.describe()
```

Out[43]:

	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

In [44]:
```
casts['name'].describe()
```

Out[44]:
```
count           75001
unique          29319
top       Ernie Adams
freq              431
Name: name, dtype: object
```

- California housing dataset

In [45]:
```
housing = pd.read_csv('housing.csv', index_col=None)
```

In [46]:
```
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
```

In [47]:
```
housing.describe()
```

Out[47]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20640.000000	20640.000000	20640.000000	20640.000000
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	1425.476744	499.539680	3.870671	206855.816909
std	2.003532	2.135952	12.585558	2181.615252	421.385070	1132.462122	382.329753	1.899822	115395.615874
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	1.000000	0.499900	14999.000000
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	787.000000	280.000000	2.563400	119600.000000
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	1166.000000	409.000000	3.534800	179700.000000
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	1725.000000	605.000000	4.743250	264725.000000
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.000000	6082.000000	15.000100	500001.000000

In [48]:
```
housing['ocean_proximity'].describe()
```

Out[48]:
```
count         20640
unique            5
top       <1H OCEAN
freq           9136
Name: ocean_proximity, dtype: object
```

In [49]:
```
housing['ocean_proximity']
```

Out[49]:
```
0        NEAR BAY
1        NEAR BAY
2        NEAR BAY
3        NEAR BAY
4        NEAR BAY
           ...
20635      INLAND
20636      INLAND
20637      INLAND
20638      INLAND
20639      INLAND
Name: ocean_proximity, Length: 20640, dtype: object
```