Lecture 19, 21 October 2025

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 = ['AMA', 'IBM', 'GOOG', 'META']
s = pd.Series(h)
s
```

Out[2]:
```
0     AMA
1     IBM
2    GOOG
3    META
dtype: object
```

In [3]:
```
type(s)
```

Out[3]:
```
pandas.core.series.Series
```

- The intention is that a series represents a column, so it should be of a uniform type, like a numpy array
- However, pandas does permit non-uniform series! Best avoided in practice.

In [4]:
```
hbad = [1,'Two',True]
sbad = pd.Series(hbad)
sbad
```

Out[4]:
```
0       1
1     Two
2    True
dtype: object
```

Convert a dictionary to a series

- Keys become "row indices"

In [5]:
```
d = {'A':'AMA', 'B':'IBM', 'C':'GOOG', 'D':'META'}
ds = pd.Series(d)
ds
```

Out[5]:
```
A     AMA
B     IBM
C    GOOG
D    META
dtype: object
```

In [6]:
```
type(ds)
```

Out[6]:
```
pandas.core.series.Series
```

Creating an index

- Provide a separate sequence of index headers, same length as values

In [7]:
```
f = ['AMA', 'IBM', 'GOOG', 'META']
fs = pd.Series(f, index = ['A','B', 'C', 'D'])
fs
```

Out[7]:
```
A     AMA
B     IBM
C    GOOG
D    META
dtype: object
```

Accessing elements

- Using named index

In [8]:
```
fs['B']
```

Out[8]:
```
'IBM'
```

- Using position
  - Note: `fs[3]` also works but is *deprecated* -- may be disallowed in future versions

In [9]:
```
fs.iloc[3], fs['D'], fs[3]
```

```
/tmp/ipykernel_88733/1188991964.py:1: FutureWarning: Series.__getitem__ treating keys as positio
ns is deprecated. In a future version, integer keys will always be treated as labels (consistent
with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  fs.iloc[3], fs['D'], fs[3]
```

Out[9]:
```
('META', 'META', 'META')
```

- Using a slice of positions
  - Here the indices behave like positions in a list
  - Slice `[i:j]` runs from `i` to `j-1`

In [10]:
```
fs.iloc[0:2]
```

Out[10]:
```
A    AMA
B    IBM
dtype: object
```

- Using a sequence of positions
- Can be out of order

In [11]:
```
fs.iloc[[0,2,1]]
```

Out[11]:
```
A     AMA
C    GOOG
B     IBM
dtype: object
```

- Can do the same with named indices
  - Slice uses position of indices
  - However, includes last index in the slice, unlike positional slice

In [12]:
```
fs['B':'D']
```

Out[12]:
```
B     IBM
C    GOOG
D    META
dtype: object
```

- An invalid slice range produces an empty output, as usual, not an error

In [13]:
```
fs['C':'A']
```

Out[13]:
```
Series([], dtype: object)
```

In [14]:
```
fs[['C','B']]
```

Out[14]:
```
C    GOOG
B     IBM
dtype: object
```

Data frames

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

In [15]:
```
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)
df2
```

Out[15]:

	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

- Table (data frame) is specified by column (series)
- Each column should have the same length, else error

In [16]:
```
data2bad = {'name' : ['AA', 'IBM', 'GOOG','META'],
        'date' : ['2001-12-01', '2012-02-10', '2010-04-09'],
        'shares' : [100, 30, 90],
        'price' : [12.3, 10.3, 32.2]
}
df2bad = pd.DataFrame(data2bad)
df2bad
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[16], line 6
      1 data2bad = {'name' : ['AA', 'IBM', 'GOOG','META'],
      2         'date' : ['2001-12-01', '2012-02-10', '2010-04-09'],
      3         'shares' : [100, 30, 90],
      4         'price' : [12.3, 10.3, 32.2]
      5 }
----> 6 df2bad = pd.DataFrame(data2bad)
      7 df2bad

File ~/python-venv/lib/python3.13/site-packages/pandas/core/frame.py:778, in DataFrame.__init__
(self, data, index, columns, dtype, copy)
    772     mgr = self._init_mgr(
    773         data, axes={"index": index, "columns": columns}, dtype=dtype, copy=copy
    774     )
    776 elif isinstance(data, dict):
    777     # GH#38939 de facto copy defaults to False only in non-dict cases
--> 778     mgr = dict_to_mgr(data, index, columns, dtype=dtype, copy=copy, typ=manager)
    779 elif isinstance(data, ma.MaskedArray):
    780     from numpy.ma import mrecords

File ~/python-venv/lib/python3.13/site-packages/pandas/core/internals/construction.py:503, in di
ct_to_mgr(data, index, columns, dtype, typ, copy)
    499     else:
    500         # dtype check to exclude e.g. range objects, scalars
    501         arrays = [x.copy() if hasattr(x, "dtype") else x for x in arrays]
--> 503 return arrays_to_mgr(arrays, columns, index, dtype=dtype, typ=typ, consolidate=copy)

File ~/python-venv/lib/python3.13/site-packages/pandas/core/internals/construction.py:114, in ar
rays_to_mgr(arrays, columns, index, dtype, verify_integrity, typ, consolidate)
    111 if verify_integrity:
    112     # figure out the index, if necessary
    113     if index is None:
--> 114         index = _extract_index(arrays)
    115     else:
    116         index = ensure_index(index)

File ~/python-venv/lib/python3.13/site-packages/pandas/core/internals/construction.py:677, in _e
xtract_index(data)
    675 lengths = list(set(raw_lengths))
    676 if len(lengths) > 1:
--> 677     raise ValueError("All arrays must be of the same length")
    679 if have_dicts:
    680     raise ValueError(
    681         "Mixing dicts with non-Series may lead to ambiguous ordering."
    682     )

ValueError: All arrays must be of the same length
```

In [17]:
```
type(df2)
```

Out[17]:
```
pandas.core.frame.DataFrame
```

- We can create a data frame from an anonymous sequence of sequences
  - In this case, each inner sequence is interpreted as a *row*, not a *column*
- Both rows and columns are indexed by position

In [18]:
```
data3 = (['AA', 'IBM', 'GOOG'],
         ['2001-12-01', '2012-02-10', '2010-04-09'],
         [100, 30, 90],
         [12.3, 10.3, 32.2])
df3 = pd.DataFrame(data3)
df3
```

Out[18]:

	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

Add a column

- We can add a column
  - Provide a default value for all rows, or
  - Provide a sequence of values

In [19]:
```
df2['owner'] = 'Unknown'
# df2['owner'] = ['a','b','c']
df2
```

Out[19]:

	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

In [20]:
```
#df2['owner'] = 'Unknown'
df2['owner2'] = ['a','b','c']
df2
```

Out[20]:

	name	date	shares	price	owner	owner2
0	AA	2001-12-01	100	12.3	Unknown	a
1	IBM	2012-02-10	30	10.3	Unknown	b
2	GOOG	2010-04-09	90	32.2	Unknown	c

- If we provide a sequence of values, it must cover all rows

In [21]:
```
#df2['owner'] = 'Unknown'
df2['owner3'] = ['a','b']
df2
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[21], line 2
      1 #df2['owner'] = 'Unknown'
----> 2 df2[        ] = ['a','b']
      3 df2

File ~/python-venv/lib/python3.13/site-packages/pandas/core/frame.py:4316, in DataFrame.__setitem__(self, key, value)
   4313     self._setitem_array([key], value)
   4314 else:
   4315     # set column
-> 4316     self._set_item(key, value)

File ~/python-venv/lib/python3.13/site-packages/pandas/core/frame.py:4529, in DataFrame._set_item(self, key, value)
   4519 def _set_item(self, key, value) -> None:
   4520     """
   4521     Add series to DataFrame in specified column.
   4522
   (...)   4527     ensure homogeneity.
   4528     """
-> 4529     value, refs = self._sanitize_column(value)
   4531     if (
   4532         key in self.columns
   4533         and value.ndim == 1
   4534         and not isinstance(value.dtype, ExtensionDtype)
   4535     ):
   4536         # broadcast across multiple columns if necessary
   4537         if not self.columns.is_unique or isinstance(self.columns, MultiIndex):

File ~/python-venv/lib/python3.13/site-packages/pandas/core/frame.py:5273, in DataFrame._sanitize_column(self, value)
   5270     return _reindex_for_setitem(value, self.index)
   5272 if is_list_like(value):
-> 5273     com.require_length_match(value, self.index)
   5274 arr = sanitize_array(value, self.index, copy=True, allow_2d=True)
   5275 if (
   5276     isinstance(value, Index)
   5277     and value.dtype == "object"
   (...)   5280     # TODO: Remove kludge in sanitize_array for string mode when enforcing
   5281     # this deprecation

File ~/python-venv/lib/python3.13/site-packages/pandas/core/common.py:573, in require_length_match(data, index)
    569 """
    570 Check the length of data matches the length of the index.
    571 """
    572 if len(data) != len(index):
--> 573     raise ValueError(
    574         "Length of values "
    575         f"({len(data)}) "
    576         "does not match length of index "
    577         f"({len(index)})"
    578     )

ValueError: Length of values (2) does not match length of index (3)
```

Add row indices

In [22]:
```
df2.index = ['one','two','three']
df2
```

Out[22]:

	name	date	shares	price	owner	owner2
one	AA	2001-12-01	100	12.3	Unknown	a
two	IBM	2012-02-10	30	10.3	Unknown	b
three	GOOG	2010-04-09	90	32.2	Unknown	c

Convert one of the columns into an index

- Note that we lose the previous indices `(one, two, three)`

In [23]:
```
df2 = df2.set_index(['name'])
df2
```

Out[23]:

name	date	shares	price	owner	owner2
AA	2001-12-01	100	12.3	Unknown	a
IBM	2012-02-10	30	10.3	Unknown	b
GOOG	2010-04-09	90	32.2	Unknown	c

Replace an index

- Again, we lose the previous index, `name`

In [24]:
```
df2 = df2.set_index(['price'])
df2
```

Out[24]:

price	date	shares	owner	owner2
12.3	2001-12-01	100	Unknown	a
10.3	2012-02-10	30	Unknown	b
32.2	2010-04-09	90	Unknown	c

Use multiple columns for indexing

In [25]:
```
df2 = pd.DataFrame(data2)  # Reset data frame to original
df2['owner'] = 'Unknown'
df2 = df2.set_index(['name','price'])
df2
```

Out[25]:

name	price	date	shares	owner
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

- Similar to projection in relational algebra

In [26]:
```
df2[['shares','date']]
```

Out[26]:

name	price	shares	date
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 [27]:
```
df2.loc['AA']
```

Out[27]:

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

Individual element - specify row and column index

In [28]:
```
df2.loc['AA','shares']
```

Out[28]:
```
price
12.3    100
Name: shares, dtype: int64
```

- What happens if the index column does not have unique values?

In [29]:
```
data3dup = {'name' : ['AA', 'IBM', 'GOOG','AA'],
        'date' : ['2001-12-01', '2012-02-10', '2010-04-09', '2025-10-21'],
        'shares' : [100, 30, 90, 100],
        'price' : [12.3, 10.3, 32.2, 33.3]
}
df3dup = pd.DataFrame(data3dup)
df3dup
```

Out[29]:

	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
3	AA	2025-10-21	100	33.3

In [30]:
```
df3dup = df3dup.set_index('name')
df3dup
```

Out[30]:

name	date	shares	price
AA	2001-12-01	100	12.3
IBM	2012-02-10	30	10.3
GOOG	2010-04-09	90	32.2
AA	2025-10-21	100	33.3

- So far so good

In [31]:
```
df3dup.loc['AA']
```

Out[31]:

name	date	shares	price
AA	2001-12-01	100	12.3
AA	2025-10-21	100	33.3

- So there is no requirement that the index column have unique values

Slices, etc

In [32]:
```
df2.loc[:,'shares']  # All rows, column 'shares'
```

Out[32]:
```
name  price
AA    12.3     100
IBM   10.3      30
GOOG  32.2      90
Name: shares, dtype: int64
```

In [33]:
```
df2 = pd.DataFrame(data2)  # Reset data frame to original
df2['owner'] = 'Unknown'
df2 = df2.set_index(['name'])
df2
```

Out[33]:

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

- An arbitrary subtable of rows and columns

In [34]:
```
df2.loc['AA':'IBM','shares':'owner']
```

Out[34]:

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

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

In [35]:
```
df2 = pd.DataFrame(data2) # Reset data frame to original
df2['owner'] = 'Unknown'
df2
```

Out[35]:

	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

- We can slice the rows
  - Note that for data frames, `0`, `1`, ... are treated as labels, so slice works like for named indices

In [36]:
```
df2.loc[0:1]
```

Out[36]:

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

- Now create a row index

In [37]:
```
df2 = df2.set_index(['name'])
```

```
df2
```

Out[37]:

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

- Can no longer slice rows by position
  - This is because the row numbers are really labels, not positions
  - The behaviour for `Series` is inconsistent with this interpretation for `DataFrane`

In [38]:
```
df2.loc[0:2]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[38], line 1
----> 1 df2.loc[0:2]

File ~/python-venv/lib/python3.13/site-packages/pandas/core/indexing.py:1191, in _LocationIndexer.__getitem__(self, key)
   1189 maybe_callable = com.apply_if_callable(key, self.obj)
   1190 maybe_callable = self._check_deprecated_callable_usage(key, maybe_callable)
-> 1191 return self._getitem_axis(maybe_callable, axis=axis)

File ~/python-venv/lib/python3.13/site-packages/pandas/core/indexing.py:1411, in _LocIndexer._getitem_axis(self, key, axis)
   1409 if isinstance(key, slice):
   1410     self._validate_key(key, axis)
-> 1411     return self._get_slice_axis(key, axis=axis)
   1412 elif com.is_bool_indexer(key):
   1413     return self._getbool_axis(key, axis=axis)

File ~/python-venv/lib/python3.13/site-packages/pandas/core/indexing.py:1443, in _LocIndexer._get_slice_axis(self, slice_obj, axis)
   1440     return obj.copy(deep=False)
   1442 labels = obj._get_axis(axis)
-> 1443 indexer = labels.slice_indexer(slice_obj.start, slice_obj.stop, slice_obj.step)
   1445 if isinstance(indexer, slice):
   1446     return self.obj._slice(indexer, axis=axis)

File ~/python-venv/lib/python3.13/site-packages/pandas/core/indexes/base.py:6708, in Index.slice_indexer(self, start, end, step)
   6664 def slice_indexer(
   6665     self,
   6666     start: Hashable | None = None,
   6667     end: Hashable | None = None,
   6668     step: int | None = None,
   6669 ) -> slice:
   6670     """
   6671     Compute the slice indexer for input labels and step.
   6672
   (...)   6706     slice(1, 3, None)
   6707     """
-> 6708     start_slice, end_slice = self.slice_locs(start, end, step=step)
   6710     # return a slice
   6711     if not is_scalar(start_slice):

File ~/python-venv/lib/python3.13/site-packages/pandas/core/indexes/base.py:6934, in Index.slice_locs(self, start, end, step)
   6932 start_slice = None
   6933 if start is not None:
-> 6934     start_slice = self.get_slice_bound(start,        )
   6935 if start_slice is None:
   6936     start_slice = 0

File ~/python-venv/lib/python3.13/site-packages/pandas/core/indexes/base.py:6849, in Index.get_slice_bound(self, label, side)
   6845 original_label = label
   6847 # For datetime indices label may be a string that has to be converted
   6848 # to datetime boundary according to its resolution.
-> 6849 label = self._maybe_cast_slice_bound(label, side)
   6851 # we need to look up the label
   6852 try:

File ~/python-venv/lib/python3.13/site-packages/pandas/core/indexes/base.py:6782, in Index._maybe_cast_slice_bound(self, label, side)
   6780 # reject them, if index does not contain label
   6781 if (is_float(label) or is_integer(label)) and label not in self:
-> 6782     self._raise_invalid_indexer(       , label)
   6784 return label

File ~/python-venv/lib/python3.13/site-packages/pandas/core/indexes/base.py:4308, in Index._raise_invalid_indexer(self, form, key, reraise)
   4306 if reraise is not lib.no_default:
   4307     raise TypeError(msg) from reraise
-> 4308 raise TypeError(msg)

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

- However, `iloc` works, like for `Series`

In [39]:
```
df2.iloc[0:2]
```

Out[39]:

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

Reading csv files

- By convention, the first line of a csv file is interpreted to be column names
- `index_col=None` says do not create a row index from any of the given columns

In [40]:
```
casts = pd.read_csv('cast.csv',index_col=None)
titles = pd.read_csv('titles.csv',index_col=None)
```

- Examine the first few rows using `head()`
  - Default is 5 rows
  - Can ask for `n` rows

In [41]:
```
casts.head()
```

Out[41]:

	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 [42]:
```
casts.head(7)
```

Out[42]:

	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

- Likewise, `tail()` gives last few lines

In [43]:
```
titles.tail()
```

Out[43]:

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

In [44]:
```
titles.tail(8)
```

Out[44]:

	title	year
49992	Legend of Horror	1972
49993	Corruption.Gov	2010
49994	Lille Fridolf blir morfar	1957
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 [45]:
```
after85 = titles[titles['year'] > 1985]
# select * from titles where year > 1985
after85
```

Out[45]:

	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

- Project the output of select on column `title`

In [46]:
```
after85titles = titles[titles['year'] > 1985]['title']
# Select title from titles where year > 1985
```

```
after85titles
```

Out[46]:
```
0          The Rising Son
2        Crucea de piatra
3                 Country
4              Gaiking II
5             Medusa (IV)
                 ...
49990             Junebug
49993      Corruption.Gov
49996             Suzanne
49997               Bomba
49999          Mrs. Munck
Name: title, Length: 29814, dtype: object
```

- Boolean combinations of conditions
  - `&` for and, `|` for or, `~` for not

- Movies in years 1990 - 1999

In [47]:
```
t = titles
movies90 = t[(t['year'] >= 1990) & (t['year'] < 2000)]
movies90
```

Out[47]:

	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

- Complement of the previous condition, using negation

In [48]:
```
t = titles
notmovies90 = t[~((t['year'] >= 1990) & (t['year'] < 2000))]
notmovies90
```

Out[48]:

	title	year
1	The Thousand Plane Raid	1969
3	Country	2000
4	Gaiking II	2011
5	Medusa (IV)	2015
6	The Fresh Air Will Do You Good	2008
...	...	...
49993	Corruption.Gov	2010
49994	Lille Fridolf blir morfar	1957
49995	Rebel	1970
49997	Bomba	2013
49998	Aao Jao Ghar Tumhara	1984

45197 rows × 2 columns

- Complement of the previous condition, using or

In [49]:
```
t = titles
notmovies90or = t[(t['year'] < 1990) | (t['year'] >= 2000)]
notmovies90or
```

Out[49]:

	title	year
1	The Thousand Plane Raid	1969
3	Country	2000
4	Gaiking II	2011
5	Medusa (IV)	2015
6	The Fresh Air Will Do You Good	2008
...	...	...
49993	Corruption.Gov	2010
49994	Lille Fridolf blir morfar	1957
49995	Rebel	1970
49997	Bomba	2013
49998	Aao Jao Ghar Tumhara	1984

45197 rows × 2 columns

Sorting

- All movies named 'Macbeth'

In [50]:
```
macbeth = t[t['title'] == 'Macbeth']
macbeth
```

Out[50]:

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

- Sort by year
  - Note that sort is in-place

In [51]:
```
macbeth = macbeth.sort_values('year')
macbeth
```

Out[51]:

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

- To restore original order, sort by index

In [52]:
```
macbeth = macbeth.sort_index()
macbeth
```

Out[52]:

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

Summaries and descriptive statistics

- `info()` gives overall summary of a data frame

In [53]:
```
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 [54]:
```
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      75000 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
```

- `describe()` gives statistical summary of numeric columns

In [55]:
```
titles.describe()
```

Out[55]:

	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 [56]:
```
casts.describe()
```

Out[56]:

	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

- Can also get summary for a non-numeric column

In [57]:
```
casts['name'].describe()
```

Out[57]:
```
count           75001
unique          29319
top       Ernie Adams
freq              431
Name: name, dtype: object
```

In [58]:
```
casts.groupby(['name'])['name'].count()
```

Out[58]:
```
name
$hutter             5
'Babe' Agamenoni    1
'Babe' Agaminono    1
'El Guisa'          1
'El Viti'           1
                   ..
Zura Abesadze       1
Zuri Alexander      1
Zuzu Abu            1
Zvone Agrez         2
gregg Alexander     1
Name: name, Length: 29319, dtype: int64
```

In [59]:
```
casts['character'].describe()
```

Out[59]:
```
count       75001
unique      50299
top       Himself
freq          405
Name: character, dtype: object
```

- Another example, housing data by locality in California

In [60]:
```
housing = pd.read_csv('housing.csv', index_col=None)
```

In [61]:
```
housing.head()
```

Out[61]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_i
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	

In [62]:
```
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 [63]:
```
housing.describe()
```

Out[63]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	hou
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20640.000000	20640
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	1425.476744	499
std	2.003532	2.135952	12.585558	2181.615252	421.385070	1132.462122	382
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	1
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	787.000000	280
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	1166.000000	409
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	1725.000000	605
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.000000	6082

- Only one non-numeric column, `ocean_proximity`

In [64]:
```
housing['ocean_proximity'].describe()
```

Out[64]:
```
count         20640
unique            5
top       <1H OCEAN
freq           9136
Name: ocean_proximity, dtype: object
```

In [65]:
```
housing['ocean_proximity'].values
```

Out[65]:
```
array(['NEAR BAY', 'NEAR BAY', 'NEAR BAY', ..., 'INLAND', 'INLAND',
       'INLAND'], shape=(20640,), dtype=object)
```