MANIPULATING DATAFRAMES WITH PANDAS

Pivoting DataFrames

Clinical trials data

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

In [2]: trials = pd.read_csv('trials_01.csv')

In [3]: print(trials)
   id treatment gender  response
0   1         A      F         5
1   2         A      M         3
2   3         B      F         8
3   4         B      M         9
```

Reshaping by pivoting

```
In [4]: trials.pivot(index='treatment',
   ...:              columns='gender',
   ...:              values='response')

Out[4]:
gender       F  M
treatment
A            5  3
B            8  9
```

Pivoting multiple columns

```
In [5]: trials.pivot(index='treatment', columns='gender')
Out[5]:
             id          response
gender       F   M          F   M
treatment
A            1   2          5   3
B            3   4          8   9
```

MANIPULATING DATAFRAMES WITH PANDAS

Let’s practice!

MANIPULATING DATAFRAMES WITH PANDAS

Stacking & unstacking DataFrames

Creating a multi-level index

```
In [1]: print(trials)
   id treatment gender  response
0   1         A      F         5
1   2         A      M         3
2   3         B      F         8
3   4         B      M         9

In [2]: trials = trials.set_index(['treatment', 'gender'])

In [3]: print(trials)
                   id  response
treatment gender
A         F         1         5
          M         2         3
B         F         3         8
          M         4         9
```

Unstacking a multi-index (1)

```
In [4]: print(trials)
                    id  response
treatment gender
A         F          1         5
          M          2         3
B         F          3         8
          M          4         9

In [5]: trials.unstack(level='gender')
Out[5]:
           id    response
gender      F  M        F  M
treatment
A           1  2        5  3
B           3  4        8  9
```

Unstacking a multi-index (2)

```
In [6]: print(trials)
                    id  response
treatment gender
A         F          1         5
          M          2         3
B         F          3         8
          M          4         9

In [7]: trials.unstack(level=1)
Out[7]:
           id    response
gender      F  M        F  M
treatment
A           1  2        5  3
B           3  4        8  9
```

Stacking DataFrames

```
In [8]: trials_by_gender = trials.unstack(level='gender')

In [9]: trials_by_gender
Out[9]:
           id    response
gender      F  M        F  M
treatment
A           1  2        5  3
B           3  4        8  9

In [10]: trials_by_gender.stack(level='gender')
Out[10]:
                   id  response
treatment gender
A         F         1         5
          M         2         3
B         F         3         8
          M         4         9
```

Stacking DataFrames

```
In [11]: stacked = trials_by_gender.stack(level='gender')

In [12]: stacked
Out[12]:
                    id  response
treatment gender
A         F          1         5
          M          2         3
B         F          3         8
          M          4         9
```

Swapping levels

```
In [13]: swapped = stacked.swaplevel(0, 1)

In [14]: print(swapped)

                  id  response
gender treatment
F      A           1         5
M      A           2         3
F      B           3         8
M      B           4         9
```

Sorting rows

```
In [15]: sorted_trials = swapped.sort_index()

In [16]: print(sorted_trials)
                    id  response
gender treatment
F      A             1         5
       B             3         8
M      A             2         3
       B             4         9
```

MANIPULATING DATAFRAMES WITH PANDAS

Let’s practice!

MANIPULATING DATAFRAMES WITH PANDAS

Melting DataFrames

Clinical trials data

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

In [2]: trials = pd.read_csv('trials_01.csv')

In [3]: print(trials)
   id treatment gender  response
0   1         A      F         5
1   2         A      M         3
2   3         B      F         8
3   4         B      M         9
```

Clinical trials after pivoting

```
In [4]: trials.pivot(index='treatment',
   ...:              columns='gender',
   ...:              values='response')
Out[4]:
gender      F  M
treatment
A           5  3
B           8  9
```

Clinical trials data

```
In [5]: new_trials = pd.read_csv('trials_02.csv')

In [6]: print(new_trials)
  treatment  F  M
0         A  5  3
1         B  8  9
```

Melting DataFrame

```
In [7]: pd.melt(new_trials)
Out[7]:
    variable value
0  treatment     A
1  treatment     B
2          F     5
3          F     8
4          M     3
5          M     9
```

Specifying id_vars

```
In [8]: pd.melt(new_trials, id_vars=['treatment'])
Out[8]:
  treatment variable  value
0         A        F      5
1         B        F      8
2         A        M      3
3         B        M      9
```

Specifying value_vars

```
In [9]: pd.melt(new_trials, id_vars=['treatment'],
   ...:         value_vars=['F', 'M'])
Out[9]:
  treatment variable  value
0         A        F      5
1         B        F      8
2         A        M      3
3         B        M      9
```

Specifying value_name

```
In [10]: pd.melt(new_trials, id_vars=['treatment'],
    ...:          var_name='gender', value_name='response')
Out[10]:
  treatment gender  response
0         A      F         5
1         B      F         8
2         A      M         3
3         B      M         9
```

MANIPULATING DATAFRAMES WITH PANDAS

Let’s practice!

MANIPULATING DATAFRAMES WITH PANDAS

Pivot tables

More clinical trials data

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

In [2]: more_trials = pd.read_csv('trials_03.csv')

In [3]: print(more_trials)
   id treatment gender  response
0   1         A      F         5
1   2         A      M         3
2   3         A      M         8
3   4         A      F         9
4   5         B      F         1
5   6         B      M         8
6   7         B      F         4
7   8         B      F         6
```

Rearranging by pivoting

```
In [4]: more_trials.pivot(index='treatment',
   ...:                   columns='gender',
   ...:                   values='response')
---------------------------------------------------------------------------
ValueError: Index contains duplicate entries, cannot reshape
```

Pivot table

```
In [5]: more_trials.pivot_table(index='treatment',
   ...:                         columns='gender',
   ...:                         values='response')
Out[5]:
gender                F    M
treatment
A             7.000000  5.5
B             3.666667  8.0
```

Other aggregations

```
In [6]: more_trials.pivot_table(index='treatment',
   ...:                         columns='gender',
   ...:                         values='response',
   ...:                         aggfunc='count')
Out[6]:
gender      F  M
treatment
A           2  2
B           3  1
```

MANIPULATING DATAFRAMES WITH PANDAS

Let’s practice!

MANIPULATING DATAFRAMES WITH PANDAS

Categoricals and groupby

Sales data

```
In [1]: sales = pd.DataFrame(
     ...:  {
     ...:   'weekday': ['Sun', 'Sun', 'Mon', 'Mon'],
     ...:   'city': ['Austin', 'Dallas', 'Austin', 'Dallas'],
     ...:   'bread': [139, 237, 326, 456],
     ...:   'butter': [20, 45, 70, 98]
     ...:  }
     ...:   )

In [2]: sales
Out[2]:
   bread  butter    city weekday
0    139      20  Austin     Sun
1    237      45  Dallas     Sun
2    326      70  Austin     Mon
3    456      98  Dallas     Mon
```

Boolean filter and count

```
In [3]: sales.loc[sales['weekday'] == 'Sun'].count()
Out[3]:
bread      2
butter     2
city       2
weekday    2
dtype: int64
```

Groupby and count

```
In [4]: sales.groupby('weekday').count()
Out[4]:
                bread  butter  city
weekday
Mon                 2       2     2
Sun                 2       2     2
```

Split-apply-combine

- sales.groupby('weekday').count()
  - split by ‘weekday’
  - apply count() function on each group
  - combine counts per group

Aggregation/Reduction

- Some reducing functions
  - mean()
  - std()
  - sum()
  - first(), last()
  - min(), max()

Groupby and sum

```
In [5]: sales.groupby('weekday')['bread'].sum()
Out[5]:
weekday
Mon    782
Sun    376
Name: bread, dtype: int64
```

Groupby and sum: multiple columns

```
In [6]: sales.groupby('weekday')[['bread','butter']].sum()
Out[6]:
                bread  butter
weekday
Mon               782     168
Sun               376      65
```

Groupby and mean: multi-level index

```
In [7]: sales.groupby(['city','weekday']).mean()
Out[7]:
                bread  butter
city   weekday
Austin Mon        326      70
       Sun        139      20
Dallas Mon        456      98
       Sun        237      45
```

Customers

```
In [8]: customers = pd.Series(['Dave','Alice','Bob','Alice'])

In [9]: customers
Out[9]:
0     Dave
1    Alice
2      Bob
3    Alice
dtype: object
```

Groupby and sum: by series

```
In [10]: sales.groupby(customers)['bread'].sum()
Out[10]:
Alice    693
Bob      326
Dave     139
Name: bread, dtype: int64
```

Categorical data

```
In [11]: sales['weekday'].unique()
Out[11]: array(['Sun', 'Mon'], dtype=object)

In [12]: sales['weekday'] = sales['weekday'].astype('category')

In [13]: sales['weekday']
Out[13]:
0    Sun
1    Sun
2    Mon
3    Mon
Name: weekday, dtype: category
Categories (2, object): [Mon, Sun]
```

Categorical data

- Advantages
  - Uses less memory
  - Speeds up operations like groupby()

MANIPULATING DATAFRAMES WITH PANDAS

Let’s practice!

MANIPULATING DATAFRAMES WITH PANDAS

Groupby and aggregation

Sales data

```
In [1]: sales = pd.DataFrame(
     ...:  {
     ...:   'weekday': ['Sun', 'Sun', 'Mon', 'Mon'],
     ...:   'city': ['Austin', 'Dallas', 'Austin', 'Dallas'],
     ...:   'bread': [139, 237, 326, 456],
     ...:   'butter': [20, 45, 70, 98]
     ...:  }
     ...:   )

In [2]: sales
Out[2]:
   bread  butter    city weekday
0    139      20  Austin     Sun
1    237      45  Dallas     Sun
2    326      70  Austin     Mon
3    456      98  Dallas     Mon
```

Review: groupby

```
In [3]: sales.groupby('city')[['bread','butter']].max()
Out[3]:
         bread  butter
         city
         Austin     326       70
         Dallas     456       98
```

Multiple aggregations

```
In [4]: sales.groupby('city')[['bread','butter']].agg(['max','sum'])
Out[4]:
        bread       butter
          max  sum     max  sum
city
Austin    326  465      70   90
Dallas    456  693      98  143
```

Aggregation functions

- string names
  - ‘sum’
  - ‘mean’
  - ‘count’

Custom aggregation

```
In [5]: def data_range(series):
   ...:     return series.max() - series.min()
```

Custom aggregation

```
In [6]: sales.groupby('weekday')[['bread', 'butter']].agg(data_range)
Out[6]:
         bread  butter
weekday
Mon        130      28
Sun         98      25
```

Custom aggregation: dictionaries

```
In [7]: sales.groupby(customers)[['bread', 'butter']]
   ...:    .agg({'bread':'sum', 'butter':data_range})
Out[7]:
        butter  bread
Alice       53    693
Bob          0    326
Dave         0    139
```

MANIPULATING DATAFRAMES WITH PANDAS

Let’s practice!

MANIPULATING DATAFRAMES WITH PANDAS

Groupby and transformation

The z-score

```
In [1]: def zscore(series):
   ...:       return (series - series.mean()) / series.std()
```

The automobile dataset

```
In [2]: auto = pd.read_csv('auto-mpg.csv')

In [3]: auto.head()
Out[3]:
    mpg  cyl  displ   hp  weight  accel  yr origin                       name
0  18.0    8  307.0  130    3504   12.0  70     US  chevrolet chevelle malibu
1  15.0    8  350.0  165    3693   11.5  70     US          buick skylark 320
2  18.0    8  318.0  150    3436   11.0  70     US         plymouth satellite
3  16.0    8  304.0  150    3433   12.0  70     US             amc rebel sst
4  17.0    8  302.0  140    3449   10.5  70     US               ford torino
```

MPG z-score

```
In [4]: zscore(auto['mpg']).head()
Out[4]:
         mpg
0 -0.697747
1 -1.082115
2 -0.697747
3 -0.953992
4 -0.825870
```

MPG z-score by year

```
In [5]: auto.groupby('yr')['mpg'].transform(zscore).head()
Out[5]:
         mpg
0  0.058125
1 -0.503753
2  0.058125
3 -0.316460
4 -0.129168
```

Apply transformation and aggregation

```
In [6]: def zscore_with_year_and_name(group):
   ...:     df = pd.DataFrame(
   ...:             {'mpg': zscore(group['mpg']),
   ...:              'year': group['yr'],
   ...:              'name': group['name']})
   ...:     return df
```

Apply transformation and aggregation

```
In [7]: auto.groupby('yr').apply(zscore_with_year_and_name).head()
Out[7]:
         mpg                        name  year
0  0.058125  chevrolet chevelle malibu  70
1 -0.503753          buick skylark 320  70
2  0.058125         plymouth satellite  70
3 -0.316460              amc rebel sst  70
4 -0.129168                ford torino  70
```

Apply transformation and aggregation

```
In [8]: def zscore_with_year_and_name(group):
     ...:     df = pd.DataFrame(
     ...:             {'mpg': zscore(group['mpg']),
     ...:              'year': group['yr'],
     ...:              'name': group['name']})
     ...:     return df

In [9]:auto.groupby('yr').apply(zscore_with_year_and_name).head()
Out[9]:
        mpg                        name  year
0  0.058125  chevrolet chevelle malibu  70
1 -0.503753          buick skylark 320  70
2  0.058125         plymouth satellite  70
3 -0.316460              amc rebel sst  70
4 -0.129168                ford torino  70
```

MANIPULATING DATAFRAMES WITH PANDAS

Let’s practice!

MANIPULATING DATAFRAMES WITH PANDAS

Groupby and filtering

The automobile dataset

```
In [1]: auto = pd.read_csv('auto-mpg.csv')

In [2]: auto.head()
Out[2]:
    mpg  cyl  displ   hp  weight  accel  yr origin                       name
0  18.0    8  307.0  130    3504   12.0  70     US  chevrolet chevelle malibu
1  15.0    8  350.0  165    3693   11.5  70     US          buick skylark 320
2  18.0    8  318.0  150    3436   11.0  70     US         plymouth satellite
3  16.0    8  304.0  150    3433   12.0  70     US             amc rebel sst
4  17.0    8  302.0  140    3449   10.5  70     US               ford torino
```

Mean MPG by year

```
In [3]: auto.groupby('yr')['mpg'].mean()
Out[3]:
yr
70    17.689655
71    21.111111
72    18.714286
73    17.100000
74    22.769231
75    20.266667
76    21.573529
77    23.375000
78    24.061111
79    25.093103
80    33.803704
81    30.185714
82    32.000000
Name: mpg, dtype: float64
```

groupby object

```
In [4]: splitting = auto.groupby('yr')

In [4]: type(splitting)
Out[4]: pandas.core.groupby.DataFrameGroupBy

In [5]: type(splitting.groups)
Out[5]: dict

In [6]: print(splitting.groups.keys())
Out[6]: dict_keys([70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81,
82])
```

groupby object: iteration

```
In [7]: for group_name, group in splitting:
   ...:     avg = group['mpg'].mean()
   ...:     print(group_name, avg)
Out[7]:
70 17.6896551724
71 21.1111111111
72 18.7142857143
73 17.1
74 22.7692307692
75 20.2666666667
76 21.5735294118
77 23.375
78 24.0611111111
79 25.0931034483
80 33.8037037037
81 30.1857142857
82 32.0
```

groupby object: iteration and filtering

```
In [8]: for group_name, group in splitting:
   ...:     avg = group.loc[group['name'].str.contains('chevrolet'), 'mpg'].mean()
   ...:     print(group_name, avg)
Out[8]:
70 15.6666666667
71 20.25
72 15.3333333333
73 14.8333333333
74 18.6666666667
75 17.6666666667
76 23.25
77 20.25
78 23.2333333333
79 21.6666666667
80 30.05
81 23.5
82 29.0
```

groupby object: comprehension

```
In [9]: chevy_means = {year:group.loc[group['name'].str.contains('chevrolet'),'mpg'].mean()
   ...:                for year,group in splitting}

In [10]: pd.Series(chevy_means)
Out[10]:
70    15.666667
71    20.250000
72    15.333333
73    14.833333
74    18.666667
75    17.666667
76    23.250000
77    20.250000
78    23.233333
79    21.666667
80    30.050000
81    23.500000
82    29.000000
dtype: float64
```

Boolean groupby

```
In [11]: chevy = auto['name'].str.contains('chevrolet')

In [12]: auto.groupby(['yr', chevy])['mpg'].mean()
Out[12]:
yr  name
70  False    17.923077
    True     15.666667
71  False    21.260870
    True     20.250000
72  False    19.120000
    True     15.333333
73  False    17.500000
    True     14.833333
74  False    23.304348
    True     18.666667
75  False    20.555556
    True     17.666667
76  False    21.350000
    True     23.250000
```

MANIPULATING DATAFRAMES WITH PANDAS

Let’s practice!

MANIPULATING DATAFRAMES WITH PANDAS

Case Study: Olympic Medals

Olympic medals dataset

	City	Edition	Sport	Discipline	Athlete	NOC	Gender	Event	Event_gender	Medal
0	Athens	1896	Aquatics	Swimming	HAJOS, Alfred	HUN	Men	100m freestyle	M	Gold
1	Athens	1896	Aquatics	Swimming	HERSCHMANN, Otto	AUT	Men	100m freestyle	M	Silver
2	Athens	1896	Aquatics	Swimming	DRIVAS, Dimitrios	GRE	Men	100m freestyle for sailors	M	Bronze
3	Athens	1896	Aquatics	Swimming	MALOKINIS, Ioannis	GRE	Men	100m freestyle for sailors	M	Gold
4	Athens	1896	Aquatics	Swimming	CHASAPIS, Spiridon	GRE	Men	100m freestyle for sailors	M	Silver
5	Athens	1896	Aquatics	Swimming	CHOROPHAS, Efstathios	GRE	Men	1200m freestyle	M	Bronze
6	Athens	1896	Aquatics	Swimming	HAJOS, Alfred	HUN	Men	1200m freestyle	M	Gold
7	Athens	1896	Aquatics	Swimming	ANDREOU, Joannis	GRE	Men	1200m freestyle	M	Silver
8	Athens	1896	Aquatics	Swimming	CHOROPHAS, Efstathios	GRE	Men	400m freestyle	M	Bronze
9	Athens	1896	Aquatics	Swimming	NEUMANN, Paul	AUT	Men	400m freestyle	M	Gold

Reminder: indexing & pivoting

- Filtering and indexing
  - One-level indexing
  - Multi-level indexing
- Reshaping DataFrames with pivot()
- pivot_table()

Reminder: groupby

- Useful DataFrame methods
  - unique()
  - value_counts()
- Aggregations, transformations, filtering

MANIPULATING DATAFRAMES WITH PANDAS

Let’s practice!

MANIPULATING DATAFRAMES WITH PANDAS

Understanding the column labels

“Gender” and “Event_gender”

	NOC	Gender	Event	Event_gender	Medal
145	GRE	Men	heavyweight - two hand lift	M	Bronze
146	DEN	Men	heavyweight - two hand lift	M	Gold
147	GBR	Men	heavyweight - two hand lift	M	Silver
148	GRE	Men	open event	M	Bronze
149	GER	Men	open event	M	Gold
150	GRE	Men	open event	M	Silver
151	HUN	Men	1500m freestyle	M	Bronze
152	GBR	Men	1500m freestyle	M	Gold
153	AUT	Men	1500m freestyle	M	Silver
154	NED	Men	200m backstroke	M	Bronze

Reminder: slicing & filtering

- Indexing and slicing
  - .loc[] and .iloc[] accessors
- Filtering
  - Selecting by Boolean Series
  - Filtering null/non-null and zero/non-zero values

Reminder: Handling categorical data

- Useful DataFrame methods for handling categorical data:
  - value_counts()
  - unique()
  - groupby()
- groupby() aggregations:
  - mean(), std(), count()

MANIPULATING DATAFRAMES WITH PANDAS

Let’s practice!

MANIPULATING DATAFRAMES WITH PANDAS

Constructing alternative country rankings

Counting distinct events

```
In [1]: medals[‘Sport'].unique() # 42 distinct events
Out[1]:
array(['Aquatics', 'Athletics', 'Cycling', 'Fencing', 'Gymnastics',
       'Shooting', 'Tennis', 'Weightlifting', 'Wrestling', 'Archery',
       'Basque Pelota', 'Cricket', 'Croquet', 'Equestrian', 'Football',
       'Golf', 'Polo', 'Rowing', 'Rugby', 'Sailing', 'Tug of War',
       'Boxing', 'Lacrosse', 'Roque', 'Hockey', 'Jeu de paume', 'Rackets',
       'Skating', 'Water Motorsports', 'Modern Pentathlon', 'Ice Hockey',
       'Basketball', 'Canoe / Kayak', 'Handball', 'Judo', 'Volleyball',
       'Table Tennis', 'Badminton', 'Baseball', 'Softball', 'Taekwondo',
       'Triathlon'], dtype=object)
```

Ranking of distinct events

- Top five countries that have won medals in the most sports
- Compare medal counts of USA and USSR from 1952 to 1988

Two new DataFrame methods

- idxmax(): Row or column label where maximum value is located
- idxmin(): Row or column label where minimum value is located

idxmax() Example

```
In [2]: weather = pd.read_csv('monthly_mean_temperature.csv',
   ...:                       index_col='Month')

In [3]: weather # DataFrame with single column
Out[3]:
               Mean TemperatureF
       Month
       Apr             53.100000
       Aug             70.000000
       Dec             34.935484
       Feb             28.714286
       Jan             32.354839
       Jul             72.870968
       Jun             70.133333
       Mar             35.000000
       May             62.612903
       Nov             39.800000
       Oct             55.451613
       Sep             63.766667
```

Using idxmax()

```
In [4]: weather.idxmax()  # Returns month of highest temperature
Out[4]:
Mean TemperatureF    Jul
dtype: object
```

Using idxmax() along columns

```
In [5]: weather.T  # Returns DataFrame with single row, 12 columns
Out[5]:
Month                 Apr   Aug        Dec         Feb         Jan         Jul  \
Mean TemperatureF  53.1  70.0  34.935484  28.714286  32.354839  72.870968

Month                      Jun   Mar        May   Nov        Oct        Sep
Mean TemperatureF  70.133333  35.0  62.612903  39.8  55.451613  63.766667

In [6]: weather.T.idxmax(axis='columns')
Out[6]:
Mean TemperatureF    Jul
dtype: object
```

Using idxmin()

```
In [7]: weather.T.idxmin(axis='columns')
Out[7]:
Mean TemperatureF    Feb
dtype: object
```

MANIPULATING DATAFRAMES WITH PANDAS

Let’s practice!

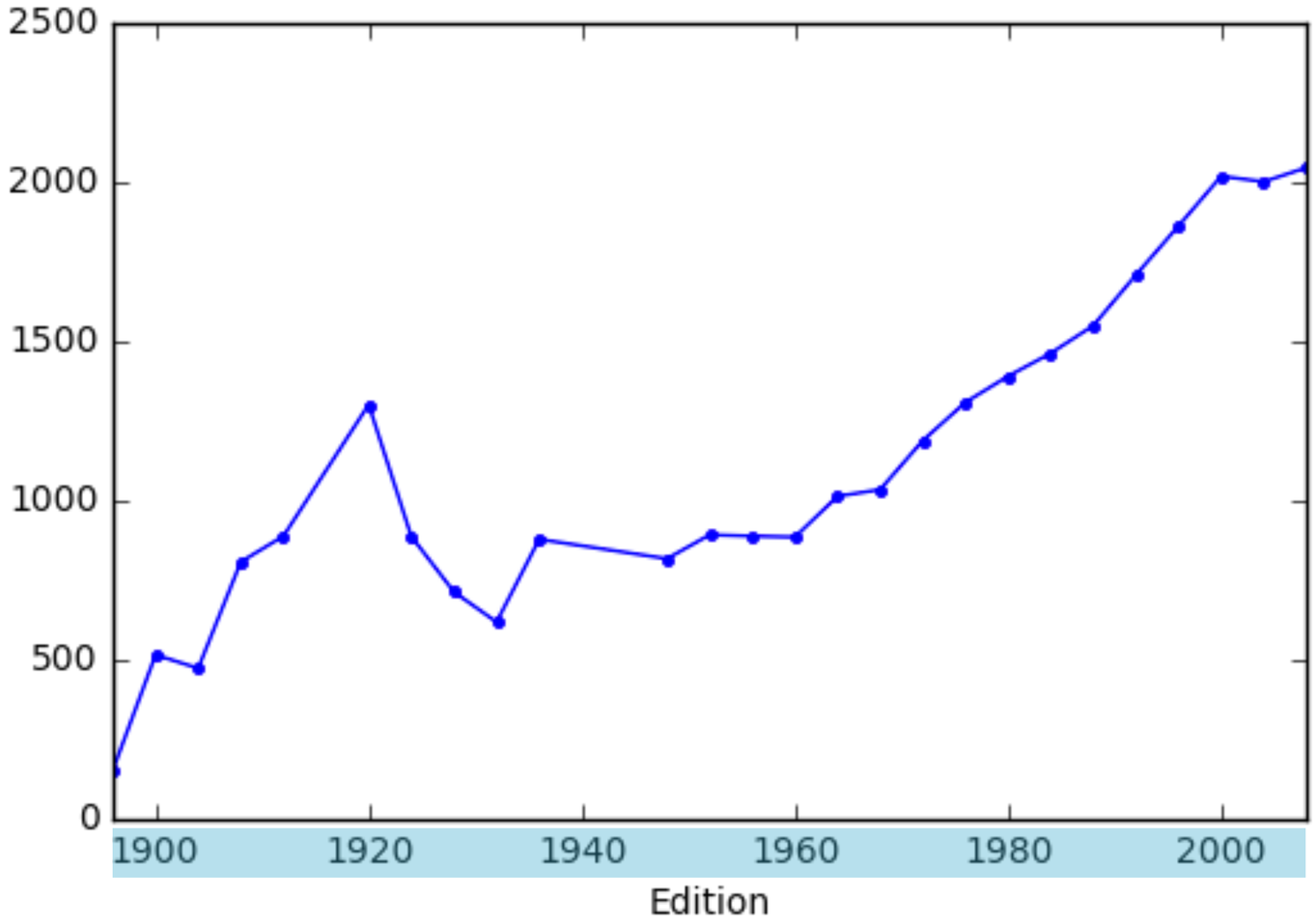
MANIPULATING DATAFRAMES WITH PANDAS

Reshaping DataFrames for visualization

Reminder: plotting DataFrames

```
In [1]: all_medals = medals.groupby('Edition')['Athlete'].count()

In [2]: all_medals.head(6) # Series for all medals, all years
Out[2]:
Edition
1896     151
1900     512
1904     470
1908     804
1912     885
1920    1298
Name: Athlete, dtype: int64

In [3]: all_medals.plot(kind='line', marker='.')

In [4]: plt.show()
```

Plotting DataFrames



Grouping the data

```
In [5]: france = medals.NOC == 'FRA' # Boolean Series for France

In [6]: france_grps = medals[france].groupby(['Edition', 'Medal'])

In [7]: france_grps['Athlete'].count().head(10)
Out[7]:
Edition  Medal
1896     Bronze     2
         Gold       5
         Silver     4
1900     Bronze    53
         Gold      46
         Silver    86
1908     Bronze    21
         Gold       9
         Silver     5
1912     Bronze     5
Name: Athlete, dtype: int64
```

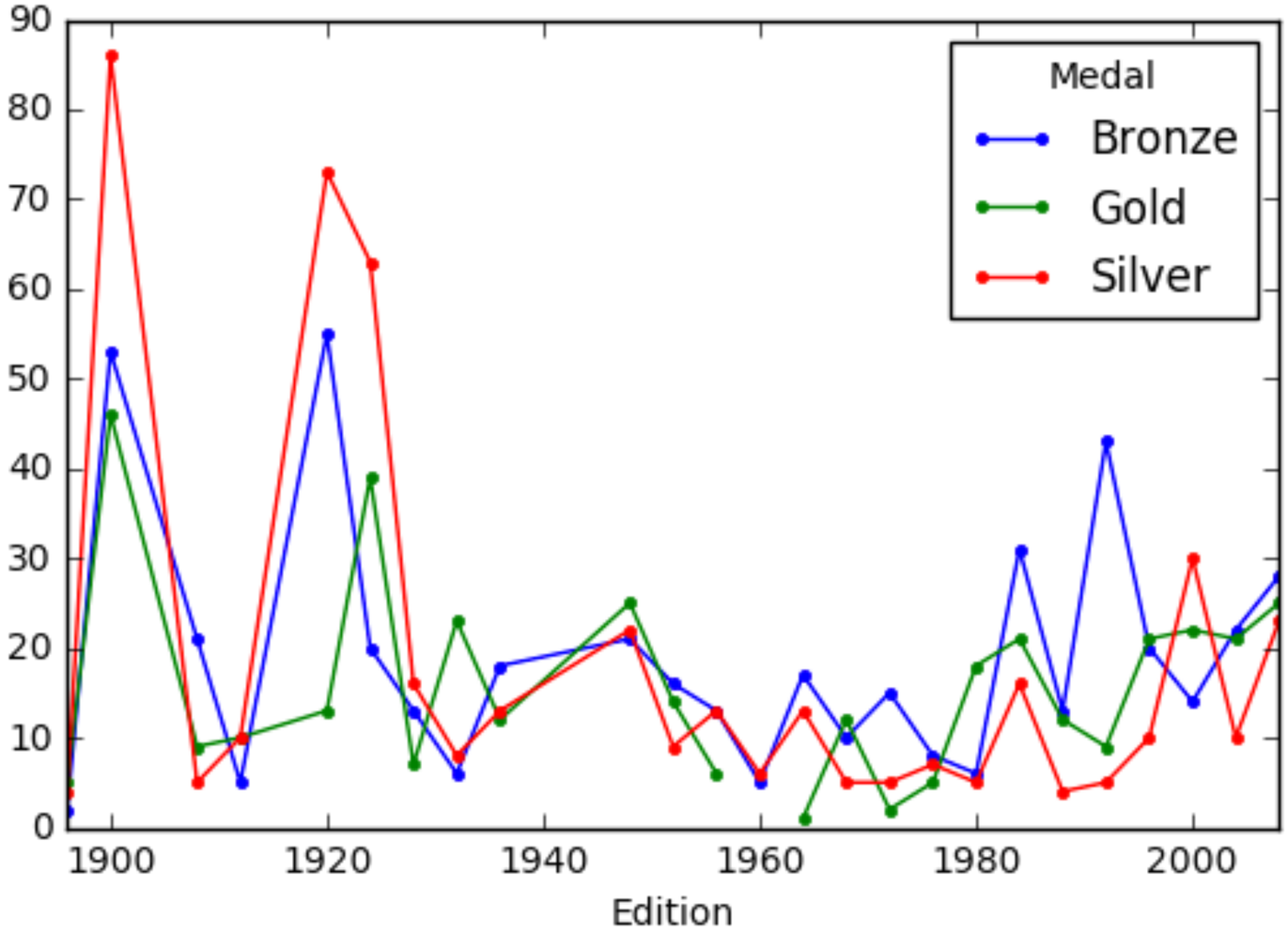
Reshaping the data

```
In [8]: france_medals = france_grps['Athlete'].count().unstack()

In [9]: france_medals.head(12)    # Single level index
Out[9]:
Medal    Bronze  Gold  Silver
Edition
1896        2.0   5.0     4.0
1900       53.0  46.0    86.0
1908       21.0   9.0     5.0
1912        5.0  10.0    10.0
1920       55.0  13.0    73.0
1924       20.0  39.0    63.0
1928       13.0   7.0    16.0
1932        6.0  23.0     8.0
1936       18.0  12.0    13.0
1948       21.0  25.0    22.0
1952       16.0  14.0     9.0
1956       13.0   6.0    13.0
```

Plotting the result

```
In [10]: france_medals.plot(kind='line', marker='.')

In [11]: plt.show()
```

Plotting the result



MANIPULATING DATAFRAMES WITH PANDAS

Let’s practice!

MANIPULATING DATAFRAMES WITH PANDAS

Final thoughts

You can now...

- Transform, extract, and filter data from DataFrames
- Work with pandas indexes and hierarchical indexes
- Reshape and restructure your data
- Split your data into groups and categories

MANIPULATING DATAFRAMES WITH PANDAS

See you in the next course!

MANIPULATING DATAFRAMES WITH PANDAS

Manipulating DataFrames with pandas

What you will learn

- Extracting, filtering, and transforming data from DataFrames
- Advanced indexing with multiple levels
- Tidying, rearranging and restructuring your data
- Pivoting, melting, and stacking DataFrames
- Identifying and splitting DataFrames by groups

MANIPULATING DATAFRAMES WITH PANDAS

See you in the course!

MANIPULATING DATAFRAMES WITH PANDAS

Indexing DataFrames

A simple DataFrame

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

In [2]: df = pd.read_csv('sales.csv', index_col='month')

In [3]: df
Out[3]:
       eggs  salt  spam
month
Jan      47  12.0    17
Feb     110  50.0    31
Mar     221  89.0    72
Apr      77  87.0    20
May     132   NaN    52
Jun     205  60.0    55
```

Indexing using square brackets

```
In [4]: df
Out[4]:
       eggs  salt  spam
month
Jan      47  12.0    17
Feb     110  50.0    31
Mar     221  89.0    72
Apr      77  87.0    20
May     132   NaN    52
Jun     205  60.0    55

In [5]: df['salt']['Jan']
Out[5]: 12.0
```

Using column attribute and row label

```
In [6]: df
Out[6]:
       eggs  salt  spam
month
Jan      47  12.0    17
Feb     110  50.0    31
Mar     221  89.0    72
Apr      77  87.0    20
May     132   NaN    52
Jun     205  60.0    55

In [7]: df.eggs['Mar']
Out[7]: 221
```

Using the .loc accessor

```
In [8]: df
Out[8]:
       eggs  salt  spam
month
Jan      47  12.0    17
Feb     110  50.0    31
Mar     221  89.0    72
Apr      77  87.0    20
May     132   NaN    52
Jun     205  60.0    55

In [9]: df.loc['May', 'spam']
Out[9]: 52.0
```

Using the .iloc accessor

```
In [10]: df
Out[10]:
       eggs  salt  spam
month
Jan      47  12.0    17
Feb     110  50.0    31
Mar     221  89.0    72
Apr      77  87.0    20
May     132   NaN    52
Jun     205  60.0    55

In [11]: df.iloc[4, 2]
Out[11]: 52.0
```

Selecting only some columns

```
In [12]: df_new = df[['salt','eggs']]

In [13]: df_new
Out[13]:
       salt  eggs
month
Jan    12.0    47
Feb    50.0   110
Mar    89.0   221
Apr    87.0    77
May     NaN   132
Jun    60.0   205
```

MANIPULATING DATAFRAMES WITH PANDAS

Let’s practice!

MANIPULATING DATAFRAMES WITH PANDAS

Slicing DataFrames

sales DataFrame

```
In [1]: df
Out[1]:
       eggs  salt  spam
month
Jan      47  12.0    17
Feb     110  50.0    31
Mar     221  89.0    72
Apr      77  87.0    20
May     132   NaN    52
Jun     205  60.0    55
```

Selecting a column (i.e., Series)

```
In [2]: df['eggs']
Out[2]:
month
Jan     47
Feb    110
Mar    221
Apr     77
May    132
Jun    205
Name: eggs, dtype: int64

In [3]: type(df['eggs'])
Out[3]: pandas.core.series.Series
```

Slicing and indexing a Series

```
In [4]: df['eggs'][1:4] # Part of the eggs column
Out[4]:
month
Feb    110
Mar    221
Apr     77
Name: eggs, dtype: int64

In [5]: df['eggs'][4]   # The value associated with May
Out[5]: 132
```

Using .loc[] (1)

```
In [6]: df.loc[:, 'eggs':'salt'] # All rows, some columns
Out[6]:
       eggs  salt
month
Jan      47  12.0
Feb     110  50.0
Mar     221  89.0
Apr      77  87.0
May     132   NaN
Jun     205  60.0
```

Using .loc[] (2)

```
In [7]: df.loc['Jan':'Apr',:] # Some rows, all columns
Out[7]:
       eggs  salt  spam
month
Jan      47  12.0    17
Feb     110  50.0    31
Mar     221  89.0    72
Apr      77  87.0    20
```

Using .loc[] (3)

```
In [8]: df.loc['Mar':'May', 'salt':'spam']
Out[8]:
       salt  spam
month
Mar    89.0    72
Apr    87.0    20
May     NaN    52
```

Using .iloc[]

```
In [9]: df.iloc[2:5, 1:] # A block from middle of the DataFrame
Out[9]:
       salt  spam
month
Mar    89.0    72
Apr    87.0    20
May     NaN    52
```

Using lists rather than slices (1)

```
In [10]: df.loc['Jan':'May', ['eggs', 'spam']]
Out[10]:
       eggs  spam
month
Jan      47    17
Feb     110    31
Mar     221    72
Apr      77    20
May     132    52
```

Using lists rather than slices (2)

```
In [11]: df.iloc[[0,4,5], 0:2]
Out[11]:
       eggs  salt
month
Jan      47  12.0
May     132   NaN
Jun     205  60.0
```

Series versus 1-column DataFrame

```
# A Series by column name
In [13]: df['eggs']
Out[13]:
month
Jan     47
Feb    110
Mar    221
Apr     77
May    132
Jun    205
Name: eggs, dtype: int64

In [14]: type(df['eggs'])
Out[14]:
pandas.core.series.Series
```

```
# A DataFrame w/ single column
In [15]: df[['eggs']]
Out[15]:
       eggs
month
Jan      47
Feb     110
Mar     221
Apr      77
May     132
Jun     205

In [16]: type(df[['eggs']])
Out[16]:
pandas.core.frame.DataFrame
```

MANIPULATING DATAFRAMES WITH PANDAS

Let’s practice!

MANIPULATING DATAFRAMES WITH PANDAS

Filtering DataFrames

Creating a Boolean Series

```
In [1]: df.salt > 60
Out[1]:
month
Jan    False
Feb    False
Mar     True
Apr     True
May    False
Jun    False
Name: salt, dtype: bool
```

Filtering with a Boolean Series

```
In [2]: df[df.salt > 60]
Out[2]:
       eggs  salt  spam
month
Mar     221  89.0    72
Apr      77  87.0    20

In [3]: enough_salt_sold = df.salt > 60

In [4]: df[enough_salt_sold]
Out[4]:
       eggs  salt  spam
month
Mar     221  89.0    72
Apr      77  87.0    20
```

Combining filters

```
In [5]: df[(df.salt >= 50) & (df.eggs < 200)] # Both conditions
Out[5]:
       eggs  salt  spam
month
Feb     110  50.0    31
Apr      77  87.0    20

In [6]: df[(df.salt >= 50) | (df.eggs < 200)] # Either condition
Out[6]:
       eggs  salt  spam
month
Jan      47  12.0    17
Feb     110  50.0    31
Mar     221  89.0    72
Apr      77  87.0    20
May     132   NaN    52
Jun     205  60.0    55
```

DataFrames with zeros and NaNs

```
In [7]: df2 = df.copy()

In [8]: df2['bacon'] = [0, 0, 50, 60, 70, 80]

In [9]: df2
Out[9]:
       eggs  salt  spam  bacon
month
Jan      47  12.0    17      0
Feb     110  50.0    31      0
Mar     221  89.0    72     50
Apr      77  87.0    20     60
May     132   NaN    52     70
Jun     205  60.0    55     80
```

Select columns with all nonzeros

```
In [10]: df2.loc[:, df2.all()]
Out[10]:
       eggs  salt  spam
month
Jan      47  12.0    17
Feb     110  50.0    31
Mar     221  89.0    72
Apr      77  87.0    20
May     132   NaN    52
Jun     205  60.0    55
```

Select columns with any nonzeros

```
In [11]: df2.loc[:, df2.any()]
Out[11]:
       eggs  salt  spam  bacon
month
Jan      47  12.0    17      0
Feb     110  50.0    31      0
Mar     221  89.0    72     50
Apr      77  87.0    20     60
May     132   NaN    52     70
Jun     205  60.0    55     80
```

Select columns with any NaNs

```
In [12]: df.loc[:, df.isnull().any()]
Out[12]:
       salt
month
Jan    12.0
Feb    50.0
Mar    89.0
Apr    87.0
May     NaN
Jun    60.0
```

Select columns without NaNs

```
In [13]: df.loc[:, df.notnull().all()]
Out[13]:
       eggs  spam
month
Jan      47    17
Feb     110    31
Mar     221    72
Apr      77    20
May     132    52
Jun     205    55
```

Drop rows with any NaNs

```
In [14]: df.dropna(how='any')
Out[14]:
       eggs  salt  spam
month
Jan      47  12.0    17
Feb     110  50.0    31
Mar     221  89.0    72
Apr      77  87.0    20
Jun     205  60.0    55
```

Filtering a column based on another

```
In [15]: df.eggs[df.salt > 55]
Out[15]:
month
Mar    221
Apr     77
Jun    205
Name: eggs, dtype: int64
```

Modifying a column based on another

```
In [16]: df.eggs[df.salt > 55] += 5

In [17]: df
Out[17]:
       eggs  salt  spam
month
Jan      47  12.0    17
Feb     110  50.0    31
Mar     226  89.0    72
Apr      82  87.0    20
May     132   NaN    52
Jun     210  60.0    55
```

MANIPULATING DATAFRAMES WITH PANDAS

Let’s practice!

MANIPULATING DATAFRAMES WITH PANDAS

Transforming DataFrames

DataFrame vectorized methods

```
In [1]: df.floordiv(12)   # Convert to dozens unit
Out[1]:
       eggs  salt  spam
month
Jan       3   1.0     1
Feb       9   4.0     2
Mar      18   7.0     6
Apr       6   7.0     1
May      11   NaN     4
Jun      17   5.0     4
```

NumPy vectorized functions

```
In [2]: import numpy as np

In [3]: np.floor_divide(df, 12)  # Convert to dozens unit
Out[3]:
       eggs  salt  spam
month
Jan     3.0   1.0   1.0
Feb     9.0   4.0   2.0
Mar    18.0   7.0   6.0
Apr     6.0   7.0   1.0
May    11.0   NaN   4.0
Jun    17.0   5.0   4.0
```

Plain Python functions (1)

```
In [4]: def dozens(n):
   ....:     return n//12

In [5]: df.apply(dozens)  # Convert to dozens unit
Out[5]:
       eggs  salt  spam
month
Jan       3   1.0     1
Feb       9   4.0     2
Mar      18   7.0     6
Apr       6   7.0     1
May      11   NaN     4
Jun      17   5.0     4
```

Plain Python functions (2)

```
In [6]: df.apply(lambda n: n//12)
Out[6]:
       eggs  salt  spam
month
Jan       3   1.0     1
Feb       9   4.0     2
Mar      18   7.0     6
Apr       6   7.0     1
May      11   NaN     4
Jun      17   5.0     4
```

Storing a transformation

```
In [7]: df['dozens_of_eggs'] = df.eggs.floordiv(12)

In [8]: df
Out[8]:
       eggs  salt  spam  dozens_of_eggs
month
Jan      47  12.0    17               3
Feb     110  50.0    31               9
Mar     221  89.0    72              18
Apr      77  87.0    20               6
May     132   NaN    52              11
Jun     205  60.0    55              17
```

The DataFrame index

```
In [9]: df
Out[9]:
       eggs  salt  spam  dozens_of_eggs
month
Jan      47  12.0    17               3
Feb     110  50.0    31               9
Mar     221  89.0    72              18
Apr      77  87.0    20               6
May     132   NaN    52              11
Jun     205  60.0    55              17

In [10]: df.index
Out[10]: Index(['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun'],
dtype='object', name='month')
```

Working with string values (1)

```
In [11]: df.index = df.index.str.upper()

In [12]: df
Out[12]:
       eggs  salt  spam  dozens_of_eggs
month
JAN      47  12.0    17               3
FEB     110  50.0    31               9
MAR     221  89.0    72              18
APR      77  87.0    20               6
MAY     132   NaN    52              11
JUN     205  60.0    55              17
```

Working with string values (2)

```
In [13]: df.index = df.index.map(str.lower)

In [14]: df
Out[14]:
     eggs  salt  spam  dozens_of_eggs
jan    47  12.0    17               3
feb   110  50.0    31               9
mar   221  89.0    72              18
apr    77  87.0    20               6
may   132   NaN    52              11
jun   205  60.0    55              17
```

Defining columns using other columns

```
In [15]: df['salty_eggs'] = df.salt + df.dozens_of_eggs

In [16]: df
Out[16]:
     eggs  salt  spam  dozens_of_eggs  salty_eggs
jan    47  12.0    17               3        15.0
feb   110  50.0    31               9        59.0
mar   221  89.0    72              18       107.0
apr    77  87.0    20               6        93.0
may   132   NaN    52              11         NaN
jun   205  60.0    55              17        77.0
```

MANIPULATING DATAFRAMES WITH PANDAS

Let’s practice!

MANIPULATING DATAFRAMES WITH PANDAS

Index objects and labeled data

pandas Data Structures

- Key building blocks
  - Indexes: Sequence of labels
  - Series: 1D array with Index
  - DataFrames: 2D array with Series as columns
- Indexes
  - Immutable (Like dictionary keys)
  - Homogenous in data type (Like NumPy arrays)

Creating a Series

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

In [2]: prices = [10.70, 10.86, 10.74, 10.71, 10.79]

In [3]: shares = pd.Series(prices)

In [4]: print(shares)
0    10.70
1    10.86
2    10.74
3    10.71
4    10.79
dtype: float64
```

Creating an index

```
In [5]: days = ['Mon', 'Tue', 'Wed', 'Thur', 'Fri']

In [6]: shares = pd.Series(prices, index=days)

In [7]: print(shares)
Mon     10.70
Tue     10.86
Wed     10.74
Thur    10.71
Fri     10.79
dtype: float64
```

Examining an index

```
In [8]: print(shares.index)
Index(['Mon', 'Tue', 'Wed', 'Thur', 'Fri'], dtype='object')

In [9]: print(shares.index[2])
Wed

In [10]: print(shares.index[:2])
Index(['Mon', 'Tue'], dtype='object')

In [11]: print(shares.index[-2:])
Index(['Thur', 'Fri'], dtype='object')

In [12]: print(shares.index.name)
None
```

Modifying index name

```
In [13]: shares.index.name = 'weekday'

In [14]: print(shares)
weekday
Monday        10.70
Tuesday       10.86
Wednesday     10.74
Thursday      10.71
Friday        10.79
dtype: float64
```

Modifying index entries

```
In [15]: shares.index[2] = 'Wednesday'
---------------------------------------------------------------------------
TypeError: Index does not support mutable operations

In [16]: shares.index[:4] = ['Monday', 'Tuesday',
    ...:                     'Wednesday', 'Thursday']
---------------------------------------------------------------------------
TypeError: Index does not support mutable operations
```

Modifying all index entries

```
In [17]: shares.index = ['Monday', 'Tuesday', 'Wednesday',
    ...:                 'Thursday', 'Friday']

In [18]: print(shares)
Monday       10.70
Tuesday      10.86
Wednesday    10.74
Thursday     10.71
Friday       10.79
dtype: float64
```

Unemployment data

```
In [19]: unemployment = pd.read_csv('Unemployment.csv')

In [20]: unemployment.head()
Out[20]:
    Zip  unemployment  participants
0  1001          0.06         13801
1  1002          0.09         24551
2  1003          0.17         11477
3  1005          0.10          4086
4  1007          0.05         11362
```

Unemployment data

```
In [21]: unemployment.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 33120 entries, 0 to 33119
Data columns (total 3 columns):
Zip             33120 non-null int64
unemployment    32556 non-null float64
particpants     33120 non-null int64
dtypes: float64(1), int64(2)
memory usage: 776.3 KB
```

Assigning the index

```
In [22]: unemployment.index = unemployment['Zip']

In [23]: unemployment.head()
Out[23]:
        Zip  unemployment  participants
Zip
1001   1001          0.06         13801
1002   1002          0.09         24551
1003   1003          0.17         11477
1005   1005          0.10          4086
1007   1007          0.05         11362
```

Removing extra column

```
In [24]: unemployment.head(3)
Out[24]:
       Zip  unemployment  participants
Zip
1001  1001           0.06         13801
1002  1002           0.09         24551
1003  1003           0.17         11477

In [25]: del unemployment['Zip']

In [26]: unemployment.head(3)
Out[26]:
       unemployment  participants
Zip
1001           0.06         13801
1002           0.09         24551
1003           0.17         11477
```

Examining index & columns

```
In [27]: print(unemployment.index)
Int64Index([1001, 1002, 1003, 1005, 1007, 1008, 1009, 1010, 1011, 1012,
             ...
             966,  968,  969,  971,  976,  979,  982,  983,  985,  987],
            dtype='int64', name='Zip', length=33120)

In [28]: print(unemployment.index.name)
Zip

In [29]: print(type(unemployment.index))
<class ‘pandas.indexes.numeric.Int64Index'>

In [30]: print(unemployment.columns)
Index(['unemployment', 'participants'], dtype='object')
```

read_csv() with index_col

```
In [31]: unemployment = pd.read_csv('Unemployment.csv',
    ...:                            index_col='Zip')

In [32]: unemployment.head()
Out[32]:
      unemployment  participants
Zip
1001          0.06         13801
1002          0.09         24551
1003          0.17         11477
1005          0.10          4086
1007          0.05         11362
```

MANIPULATING DATAFRAMES WITH PANDAS

Let’s practice!

MANIPULATING DATAFRAMES WITH PANDAS

Hierarchical Indexing

Stock data

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

In [2]: stocks = pd.read_csv('datasets/stocks.csv')

In [3]: print(stocks)
         Date   Close     Volume Symbol
0  2016-10-03   31.50   14070500   CSCO
1  2016-10-03  112.52   21701800   AAPL
2  2016-10-03   57.42   19189500   MSFT
3  2016-10-04  113.00   29736800   AAPL
4  2016-10-04   57.24   20085900   MSFT
5  2016-10-04   31.35   18460400   CSCO
6  2016-10-05   57.64   16726400   MSFT
7  2016-10-05   31.59   11808600   CSCO
8  2016-10-05  113.05   21453100   AAPL
```

Repeated values

Repeated values

Setting index

```
In [4]: stocks = stocks.set_index(['Symbol', 'Date'])

In [5]: print(stocks)
                    Close    Volume
Symbol Date
CSCO   2016-10-03   31.50  14070500
AAPL   2016-10-03  112.52  21701800
MSFT   2016-10-03   57.42  19189500
AAPL   2016-10-04  113.00  29736800
MSFT   2016-10-04   57.24  20085900
CSCO   2016-10-04   31.35  18460400
MSFT   2016-10-05   57.64  16726400
CSCO   2016-10-05   31.59  11808600
AAPL   2016-10-05  113.05  21453100
```

MultiIndex on DataFrame

```
In [6]: print(stocks.index)
MultiIndex(levels=[['AAPL', 'CSCO', 'MSFT'], ['2016-10-03',
'2016-10-04', '2016-10-05']], labels=[[1, 0, 2, 0, 2, 1, 2, 1, 0],
[0, 0, 0, 1, 1, 1, 2, 2, 2]], names=['Symbol', 'Date'])

In [7]: print(stocks.index.name)
None

In [8]: print(stocks.index.names)
['Symbol', 'Date']
```

Sorting index

```
In [9]: stocks = stocks.sort_index()

In [10]: print(stocks)
                    Close    Volume
Symbol Date
AAPL   2016-10-03  112.52  21701800
       2016-10-04  113.00  29736800
       2016-10-05  113.05  21453100
CSCO   2016-10-03   31.50  14070500
       2016-10-04   31.35  18460400
       2016-10-05   31.59  11808600
MSFT   2016-10-03   57.42  19189500
       2016-10-04   57.24  20085900
       2016-10-05   57.64  16726400
```

Indexing (individual row)

```
In [11]: stocks.loc[('CSCO', '2016-10-04')]
Out[11]:
Close            31.35
Volume    18460400.00
Name: (CSCO, 2016-10-04), dtype: float64

In [12]: stocks.loc[('CSCO', '2016-10-04'), 'Volume']
Out[12]: 18460400.0
```

Slicing (outermost index)

```
In [13]: stocks.loc['AAPL']
Out[13]:
              Close    Volume
Date
2016-10-03  112.52  21701800
2016-10-04  113.00  29736800
2016-10-05  113.05  21453100
```

Slicing (outermost index)

```
In [14]: stocks.loc['CSCO':'MSFT']
Out[14]:
                    Close    Volume
Symbol Date
CSCO   2016-10-03  31.50  14070500
       2016-10-04  31.35  18460400
       2016-10-05  31.59  11808600
MSFT   2016-10-03  57.42  19189500
       2016-10-04  57.24  20085900
       2016-10-05  57.64  16726400
```

Fancy indexing (outermost index)

```
In [15]: stocks.loc[(['AAPL', 'MSFT'], '2016-10-05'), :]
Out[15]:
                     Close    Volume
Symbol Date
AAPL   2016-10-05  113.05  21453100
MSFT   2016-10-05   57.64  16726400

In [16]: stocks.loc[(['AAPL', 'MSFT'], '2016-10-05'), 'Close']
Out[16]:
Symbol  Date
AAPL    2016-10-05    113.05
MSFT    2016-10-05     57.64
Name: Close, dtype: float64
```

Fancy indexing (innermost index)

```
In [17]: stocks.loc[('CSCO', ['2016-10-05', '2016-10-03']), :]
Out[17]:
                    Close    Volume
Symbol Date
CSCO   2016-10-03  31.50  14070500
       2016-10-05  31.59  11808600
```

Slicing (both indexes)

```
In [18]: stocks.loc[(slice(None), slice('2016-10-03', '2016-10-04')),:]
Out[18]:
                     Close    Volume
Symbol Date
AAPL   2016-10-03  112.52  21701800
       2016-10-04  113.00  29736800
CSCO   2016-10-03   31.50  14070500
       2016-10-04   31.35  18460400
MSFT   2016-10-03   57.42  19189500
       2016-10-04   57.24  20085900
```

MANIPULATING DATAFRAMES WITH PANDAS

Let’s practice!