

# Introduction to Flat Files

STREAMLINED DATA INGESTION WITH PANDAS

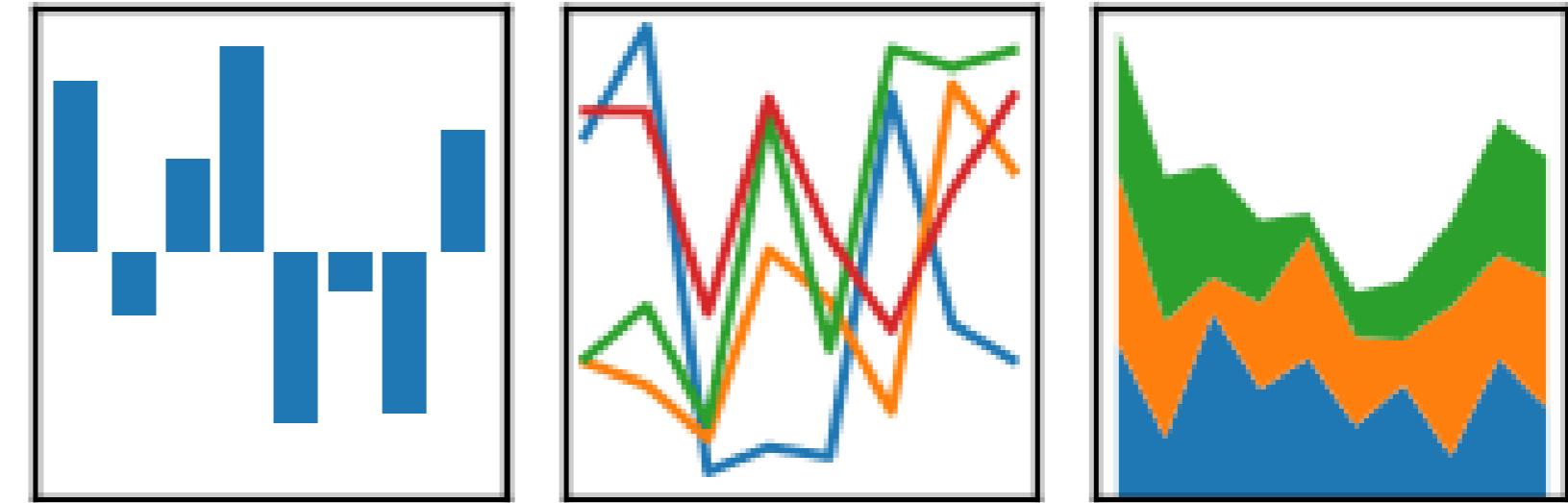


Amany Mahfouz  
Instructor

# pandas

# pandas

$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$



# Data Frames

- pandas -specific structure for two-dimensional data

	Country	Population	Area (sq. mi.)
0	Afghanistan	31056997	647500
1	Albania	3581655	28748
2	Algeria	32930091	2381740
3	American Samoa	57794	199
4	Andorra	71201	468
5	Angola	12127071	1246700
6	Anguilla	13477	102
7	Antigua & Barbuda	69108	443
8	Argentina	39921833	2766890
9	Armenia	2976372	29800
10	Aruba	71891	193

# Data Frames

- pandas -specific structure for two-dimensional data

	Country	Population	Area (sq. mi.)
0	Afghanistan	31056997	647500
1	Albania	3581655	28748
2	Algeria	32930091	2381740
3	American Samoa	57794	199
4	Andorra	71201	468
5	Angola	12127071	1246700
6	Anguilla	13477	102
7	Antigua & Barbuda	69108	443
8	Argentina	39921833	2766890
9	Armenia	2976372	29800
10	Aruba	71891	193

Column Labels

# Data Frames

- pandas -specific structure for two-dimensional data

	Country	Population	Area (sq. mi.)
0	Afghanistan	31056997	647500
1	Albania	3581655	28748
2	Algeria	32930091	2381740
3	American Samoa	57794	199
4	Andorra	71201	468
5	Angola	12127071	1246700
6	Anguilla	13477	102
7	Antigua & Barbuda	69108	443
8	Argentina	39921833	2766890
9	Armenia	2976372	29800
10	Aruba	71891	193

**Row Labels (Index)**

**Column Labels**

# Flat Files

- Simple, easy-to-produce format
- Data stored as plain text (no formatting)
- One row per line
- Values for different fields are separated by a delimiter
- Most common flat file type: comma-separated values
- One `pandas` function to load them all: `read_csv()`

# Loading CSVs

- Sample of `us_tax_data_2016.csv`

```
STATEFIPS,STATE,zipcode,agi_stub,...,N11901,A11901,N11902,A11902  
1,AL,0,1,...,63420,51444,711580,1831661
```

```
import pandas as pd  
  
tax_data = pd.read_csv("us_tax_data_2016.csv")  
tax_data.head(4)
```

	STATEFIPS	STATE	zipcode	agi_stub	...	N11901	A11901	N11902	A11902
0	1	AL	0	1	...	63420	51444	711580	1831661
1	1	AL	0	2	...	74090	110889	416090	1173463
2	1	AL	0	3	...	64000	143060	195130	543284
3	1	AL	0	4	...	45020	128920	117410	381329

# Loading Other Flat Files

- Specify a different delimiter with `sep`
- Sample of `us_tax_data_2016.tsv`

	STATEFIPS	STATE	zipcode	agi_stub	...	N11901	A11901	N11902	A11902
1	AL	0	1	...	63420	51444	711580	1831661	

```
import pandas as pd  
  
tax_data = pd.read_csv("us_tax_data_2016.tsv", sep="\t")  
  
tax_data.head(3)
```

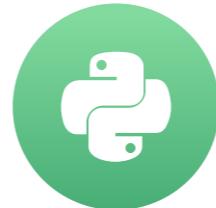
	STATEFIPS	STATE	zipcode	agi_stub	...	N11901	A11901	N11902	A11902
0	1	AL	0	1	...	63420	51444	711580	1831661
1	1	AL	0	2	...	74090	110889	416090	1173463
2	1	AL	0	3	...	64000	143060	195130	543284

# Let's practice!

**STREAMLINED DATA INGESTION WITH PANDAS**

# Modifying flat file imports

STREAMLINED DATA INGESTION WITH PANDAS



Amany Mahfouz  
Instructor

# U.S. Tax Data

```
tax_data = pd.read_csv('us_tax_data_2016.csv')  
  
print(tax_data.shape)
```

```
(179796, 147)
```

# Limiting Columns

- Choose columns to load with the `usecols` keyword argument
- Accepts a list of column numbers or names, or a function to filter column names

```
col_names = ['STATEFIPS', 'STATE', 'zipcode', 'agi_stub', 'N1']
col_nums = [0, 1, 2, 3, 4]
# Choose columns to load by name
tax_data_v1 = pd.read_csv('us_tax_data_2016.csv',
                          usecols=col_names)
# Choose columns to load by number
tax_data_v2 = pd.read_csv('us_tax_data_2016.csv',
                          usecols=col_nums)
print(tax_data_v1.equals(tax_data_v2))
```

True

# Limiting Rows

- Limit the number of rows loaded with the `nrows` keyword argument

```
tax_data_first1000 = pd.read_csv('us_tax_data_2016.csv', nrows=1000)  
print(tax_data_first1000.shape)
```

```
(1000, 147)
```

# Limiting Rows

- Use `nrows` and `skiprows` together to process a file in chunks
- `skiprows` accepts a list of row numbers, a number of rows, or a function to filter rows
- Set `header=None` so `pandas` knows there are no column names

```
tax_data_next500 = pd.read_csv('us_tax_data_2016.csv',  
                               nrows=500,  
                               skiprows=1000,  
                               header=None)
```

# Limiting Rows

```
print(tax_data_next500.head(1))
```

```
    0   1      2   3   4   5   6   7   8   9   10  ...  136  137  138  139  140  141
0   1   AL  35565   4  270   0  250   0  210  790  280  ...  1854  260  1978   0   0   0
```

```
[1 rows x 147 columns]
```

# Assigning Column Names

- Supply column names by passing a list to the `names` argument
- The list **MUST** have a name for every column in your data
- **If you only need to rename a few columns, do it after the import!**

# Assigning Column Names

```
col_names = list(tax_data_first1000)

tax_data_next500 = pd.read_csv('us_tax_data_2016.csv',
                               nrows=500,
                               skiprows=1000,
                               header=None,
                               names=col_names)

print(tax_data_next500.head(1))
```

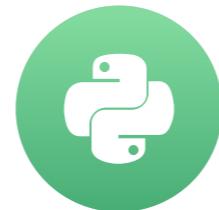
```
STATEFIPS STATE zipcode agi_stub ... N11901 A11901 N11902 A11902
0 1 AL 35565 4 ... 50 222 210 794
[1 rows x 147 columns]
```

# Let's practice!

**STREAMLINED DATA INGESTION WITH PANDAS**

# Handling errors and missing data

STREAMLINED DATA INGESTION WITH PANDAS



Amany Mahfouz

Instructor

# Common Flat File Import Issues

- Column data types are wrong
- Values are missing
- Records that cannot be read by `pandas`

# Specifying Data Types

- pandas automatically infers column data types

```
print(tax_data.dtypes)
```

```
STATEFIPS      int64
STATE          object
zipcode        int64
agi_stub       int64
N1             int64
...
N11902        int64
A11902        int64
Length: 147, dtype: object
```

# Specifying Data Types

- Use the `dtype` keyword argument to specify column data types
- `dtype` takes a dictionary of column names and data types

```
tax_data = pd.read_csv("us_tax_data_2016.csv", dtype={"zipcode": str})  
print(tax_data.dtypes)
```

```
STATEFIPS      int64  
STATE         object  
zipcode       object  
agi_stub      int64  
N1            int64  
...  
N11902        int64  
A11902        int64  
Length: 147, dtype: object
```

# Customizing Missing Data Values

- pandas automatically interprets some values as missing or NA

```
print(tax_data.head())
```

```
STATEFIPS STATE zipcode agi_stub N1 ... A85300 N11901 A11901 N11902 A11902
0 1 AL 0 1 815440 ... 0 63420 51444 711580 1831661
1 1 AL 0 2 495830 ... 0 74090 110889 416090 1173463
2 1 AL 0 3 263390 ... 0 64000 143060 195130 543284
3 1 AL 0 4 167190 ... 0 45020 128920 117410 381329
4 1 AL 0 5 217440 ... 19 82940 423629 126130 506526
[ 5 rows x 147 columns]
```

# Customizing Missing Data Values

- Use the `na_values` keyword argument to set custom missing values
- Can pass a single value, list, or dictionary of columns and values

```
tax_data = pd.read_csv("us_tax_data_2016.csv",
                       na_values={"zipcode" : 0})

print(tax_data[tax_data.zipcode.isna()])
```

	STATEFIPS	STATE	zipcode	agi_stub	N1	...	A85300	N11901	A11901	N11902	A11902
0	1	AL	NaN	1	815440	...	0	63420	51444	711580	183
1	1	AL	NaN	2	495830	...	0	74090	110889	416090	117
2	1	AL	NaN	3	263390	...	0	64000	143060	195130	54
...	...	...	...	...	...	...	...	...	...	...	...
179034	56	WY	NaN	5	38030	...	121	13230	73326	22250	9
179035	56	WY	NaN	6	8480	...	53835	3630	128149	2250	12

# Lines with Errors

Sample of `us_tax_data_2016_corrupt.csv`

```
STATEFIPS,STATE,zipcode,agi_stub,...,N11901,A11901,N11902,A11902  
1,AL,0,1,...,63420,51444,711580,1831661  
1,AL,0, ,2,...,74090,110889,416090,1173463
```

```
tax_data = pd.read_csv("us_tax_data_2016_corrupt.csv")
```

```
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
    data = pd.read_csv('us_tax_data_2016_corrupt.csv')
  File "<stdin>", line 697, in parser_f
    return _read(filepath_or_buffer, kwds)
  File "<stdin>", line 430, in _read
    data = parser.read(nrows)
  File "<stdin>", line 1134, in read
    ret = self._engine.read(nrows)
  File "<stdin>", line 1990, in read
    data = self._reader.read(nrows)
  File "<stdin>", line 899, in pandas._libs.parsers.TextReader.read
  File "<stdin>", line 914, in pandas._libs.parsers.TextReader._read_low_memory
  File "<stdin>", line 968, in pandas._libs.parsers.TextReader._read_rows
  File "<stdin>", line 955, in pandas._libs.parsers.TextReader._tokenize_rows
  File "<stdin>", line 2172, in pandas._libs.parsers.raise_parser_error
pandas.errors.ParserError: Error tokenizing data. C error: Expected 147 fields in line 3, saw 148
```

# Lines with Errors

- Set `error_bad_lines=False` to skip unparsable records
- Set `warn_bad_lines=True` to see messages when records are skipped

```
tax_data = pd.read_csv("us_tax_data_2016_corrupt.csv",
                       error_bad_lines=False,
                       warn_bad_lines=True)
```

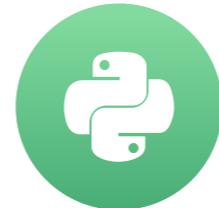
```
b'Skipping line 3: expected 147 fields, saw 148\n'
```

# Let's practice!

**STREAMLINED DATA INGESTION WITH PANDAS**

# Introduction to spreadsheets

STREAMLINED DATA INGESTION WITH PANDAS



Amany Mahfouz  
Instructor

# Spreadsheets

- Also known as Excel files
- Data stored in tabular form, with cells arranged in rows and columns
- Unlike flat files, can have formatting and formulas
- Multiple spreadsheets can exist in a workbook

# Loading Spreadsheets

- Spreadsheets have their own loading function in `pandas` : `read_excel()`

	A	B	C	D	E	F	G	H	I
1	Age	AttendedBootcamp	BootcampFinish	BootcampLoanYesNo	BootcampName	BootcampRecommend	ChildrenNumber	CityPopulation	CodeEventConfere
2	28	0						between 100,000 and 1 million	
3	22	0						between 100,000 and 1 million	
4	19	0						more than 1 million	
5	26	0						more than 1 million	
6	20	0						between 100,000 and 1 million	
7	34	0						more than 1 million	
8	23	0						more than 1 million	
9	35	0						between 100,000 and 1 million	
10	33	0						between 100,000 and 1 million	
11	33	0						more than 1 million	
12	57	0						less than 100,000	
13	23	0						more than 1 million	
14	47	0						more than 1 million	
15		0						between 100,000 and 1 million	
16	37	0						1 between 100,000 and 1 million	
17	31	0						more than 1 million	
18	27	0						more than 1 million	
19	29	0						less than 100,000	
20	30	0						more than 1 million	
21	30	0						less than 100,000	
22	32	0						1 more than 1 million	
23	25	0						between 100,000 and 1 million	
24	29	0						between 100,000 and 1 million	

# Loading Spreadsheets

```
import pandas as pd

# Read the Excel file
survey_data = pd.read_excel("fcc_survey.xlsx")

# View the first 5 lines of data
print(survey_data.head())
```

```
   Age AttendedBootcamp ... SchoolMajor StudentDebtOwe
0  28.0           0.0    ...          NaN        20000
1  22.0           0.0    ...          NaN          NaN
2  19.0           0.0    ...          NaN          NaN
3  26.0           0.0    ...  Cinematography And Film      7000
4  20.0           0.0    ...          NaN          NaN

[5 rows x 98 columns]
```

# Loading Select Columns and Rows

A	B	C	D		
1	FreeCodeCamp New Developer Survey Responses, 2016				
2	Source: <a href="https://www.kaggle.com/freecodecamp/2016-new-coder-survey">https://www.kaggle.com/freecodecamp/2016-new-coder-survey</a>				
3	Age	AttendedBootcamp	BootcampFinish	BootcampLoanYesNo	BootcampName
4	28	0			
5	22	0			
6	19	0			
7	26	0			
8	20	0			
9	34	0			
10	23	0			
11	35	0			
12	33	0			
13	33	0			
14	57	0			
15	23	0			
16	47	0			
17		0			
18	37	0			
19	31	0			
20	27	0			
21	29	0			

	A	B	C	D	E	F	G	H
8			<h1>Invoice</h1>					
9			<b>Submitted on 05/30/2019</b>					
10								
11	<b>Invoice for</b>		<b>Payable to</b>		<b>Invoice #</b>			
12			Local Driving School		36299241			
13								
14					<b>Due date</b>			
15					6/30/2019			
16								
17								
18	<b>Description</b>		<b>Hours</b>	<b>Hourly Rate</b>	<b>Total price</b>			
19	Driving classes		7.5	\$40.00	\$300.00			
20	Driver safety class		5	\$10.00	\$50.00			
21	Road test appointment		n/a	\$20.00	\$20.00			
22	Car for road test		n/a	\$70.00	\$70.00			
23								
24	Notes:			Subtotal	\$440.00			
25				Adjustments	\$0.00			
26						<b>\$440.00</b>		

# Loading Select Columns and Rows

- `read_excel()` has many keyword arguments in common with `read_csv()`
  - `nrows` : limit number of rows to load
  - `skiprows` : specify number of rows or row numbers to skip
  - `usecols` : choose columns by name, positional number, or letter (e.g. "A:P")

# Loading Select Columns and Rows

	W	X	Y	Z	AA	AB	AR
1							
2							
3	CommuteTime	CountryCitizen	CountryLive	EmploymentField	EmploymentFieldOther	EmploymentStatus	Income
4	35	United States of America	United States of America	office and administrative support		Employed for wages	32000
5	90	United States of America	United States of America	food and beverage		Employed for wages	15000
6	45	United States of America	United States of America	finance		Employed for wages	48000
7	45	United States of America	United States of America	arts, entertainment, sports, or media		Employed for wages	43000
8	10	United States of America	United States of America	education		Employed for wages	6000
9	45	United States of America	United States of America	finance		Self-employed freelancer	40000
10	60	Singapore	Singapore	software development		Employed for wages	32000

# Loading Select Columns and Rows

```
# Read columns W-AB and AR of file, skipping metadata header
survey_data = pd.read_excel("fcc_survey_with_headers.xlsx",
                            skiprows=2,
                            usecols="W:AB, AR")

# View data
print(survey_data.head())
```

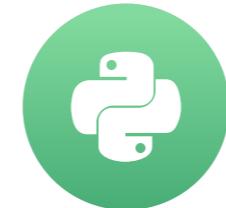
```
CommuteTime      CountryCitizen ... EmploymentFieldOther EmploymentStatus Income
0            35.0 United States of America ...           NaN Employed for wages  32000.0
1            90.0 United States of America ...           NaN Employed for wages  15000.0
2            45.0 United States of America ...           NaN Employed for wages  48000.0
3            45.0 United States of America ...           NaN Employed for wages  43000.0
4            10.0 United States of America ...           NaN Employed for wages   6000.0
[5 rows x 7 columns]
```

# Let's practice!

**STREAMLINED DATA INGESTION WITH PANDAS**

# Getting data from multiple worksheets

STREAMLINED DATA INGESTION WITH PANDAS



Amany Mahfouz

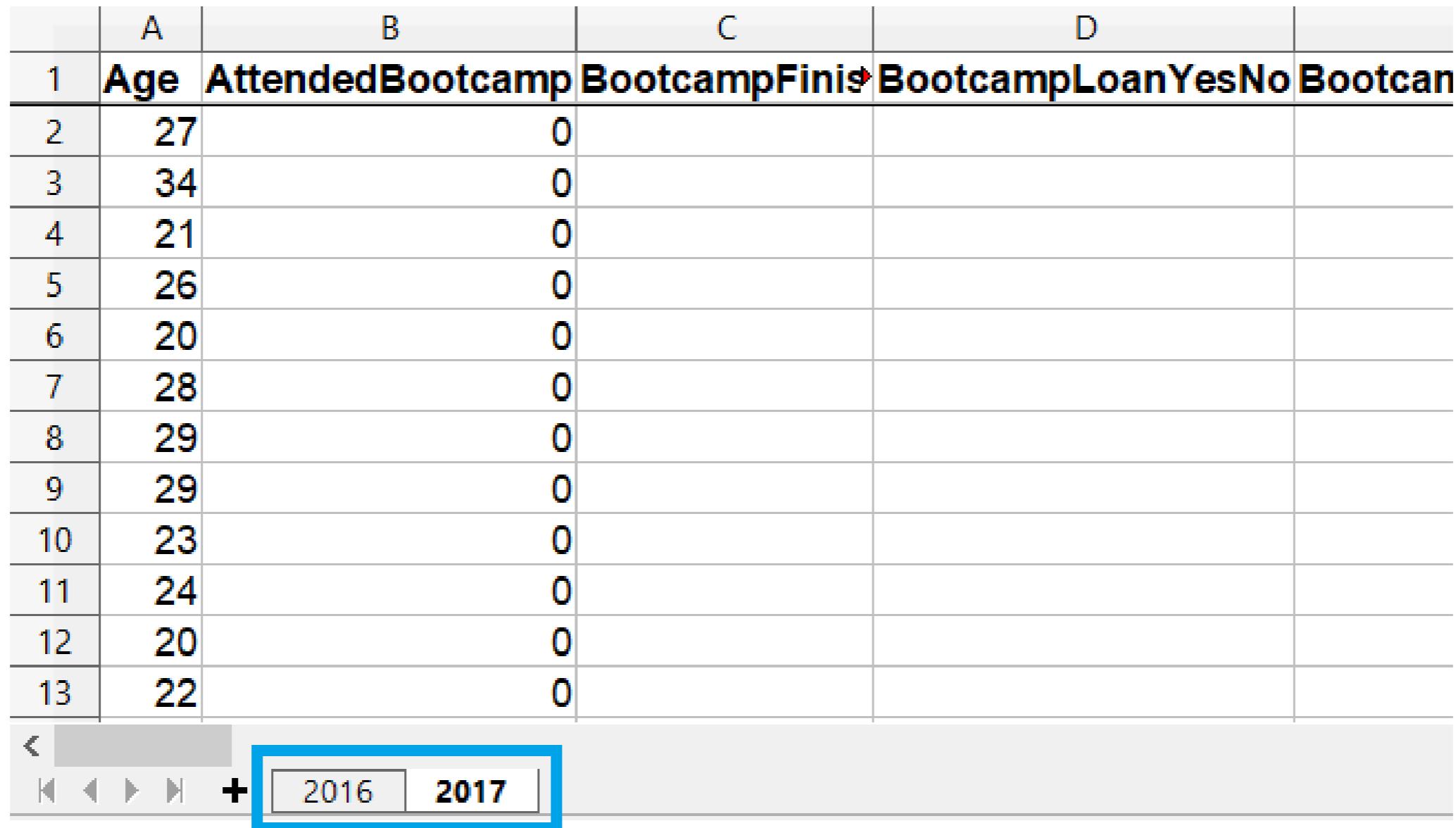
Instructor

# Selecting Sheets to Load

- `read_excel()` loads the first sheet in an Excel file by default
- Use the `sheet_name` keyword argument to load other sheets
- Specify spreadsheets by name and/or (zero-indexed) position number
- Pass a list of names/numbers to load more than one sheet at a time
- Any arguments passed to `read_excel()` apply to all sheets read

# Selecting Sheets to Load

	A	B	C	D	
1	Age	AttendedBootcamp	BootcampFinis	BootcampLoan	YesNo Bootcan
2	27		0		
3	34		0		
4	21		0		
5	26		0		
6	20		0		
7	28		0		
8	29		0		
9	29		0		
10	23		0		
11	24		0		
12	20		0		
13	22		0		



2016 2017

# Loading Select Sheets

```
# Get the second sheet by position index  
survey_data_sheet2 = pd.read_excel('fcc_survey.xlsx',  
                                    sheet_name=1)  
  
# Get the second sheet by name  
survey_data_2017 = pd.read_excel('fcc_survey.xlsx',  
                                 sheet_name='2017')  
  
print(survey_data_sheet2.equals(survey_data_2017))
```

```
True
```

# Loading All Sheets

- Passing `sheet_name=None` to `read_excel()` reads all sheets in a workbook

```
survey_data = pd.read_excel("fcc_survey.xlsx", sheet_name=None)

print(type(survey_data))
```

```
<class 'collections.OrderedDict'>
```

```
for key, value in survey_data.items():
    print(key, type(value))
```

```
2016 <class 'pandas.core.frame.DataFrame'>
2017 <class 'pandas.core.frame.DataFrame'>
```

# Putting It All Together

```
# Create empty data frame to hold all loaded sheets
all_responses = pd.DataFrame()

# Iterate through data frames in dictionary
for sheet_name, frame in survey_responses.items():
    # Add a column so we know which year data is from
    frame["Year"] = sheet_name

    # Add each data frame to all_responses
    all_responses = all_responses.append(frame)

# View years in data
print(all_responses.Year.unique())
```

```
['2016' '2017']
```

# Let's practice!

**STREAMLINED DATA INGESTION WITH PANDAS**

# Modifying imports: true/false data

STREAMLINED DATA INGESTION WITH PANDAS



Amany Mahfouz

Instructor

# Boolean Data

- True / False data

	A	B	C	D	E	F	G
1	ID.x	AttendedBootcamp	AttendedBootCampYesNo	AttendedBootcampTF	BootcampLoan	LoanYesNo	LoanTF
89	6ca993739cf368a8b764ecb355359da2		0 No		FALSE		
90	48439bea8554956d8a577b5ad63f9524		0 No		FALSE		
91	79aebaf36d9ccd10d0f1b2a9dff9543c		0 No		FALSE		
92	ea0319686c422efc9fe9c0364a6fb117		0 No		FALSE		
93	915f2ed898947d610e3b41c10bed72fe		0 No		FALSE		
94	24b64d38e5025f28bd5c0be8fd6ae9be		0 No		FALSE		
95	1a124244c3f5501bc0a5c96ff2387cc0		1 Yes		TRUE	0 No	FALSE
96	fe4b00562e4aaa53b4b6956d0631f021		0 No		FALSE		
97	9cc94bb3a1e6a029c54e1baaad346055		0 No		FALSE		
98	16e7110386a7c024adcb4753cdd042b8		0 No		FALSE		
99	f78cf5785eba1985f5bdb9de8dfdda69		1 Yes		TRUE	0 No	FALSE
100	65bb23364ae1581e38e35b166d47ef1e		0 No		FALSE		
101	ae712b0271669b79479c8051e56956cc		0 No		FALSE		
102	3aaaae9b5b7a39f4a6b4febedc5152c2f		0 No		FALSE		
103	50eb0912d0efb00dee1b0590a48c8668		0 No		FALSE		
104	8a4040d2531281194752475dc2c53609		0 No		FALSE		
105	5aaa2d5e9596cccc55ca93a8d7de6127		0 No		FALSE		
106	b20068a41d1199ada2e55b5fdfd254f2		0 No		FALSE		
107	e90cb86f2b59212724bce3b2dad53276		0 No		FALSE		
108	7c196c58dbe549119218158b2b28d8d		0 No		FALSE		

# Boolean Data

	A	B	C	D	E	F	G
1	ID.x	AttendedBootcamp	AttendedBootCampYesNo	AttendedBootcampTF	BootcampLoan	LoanYesNo	LoanTF
89	6ca993739cf368a8b764ecb355359da2	0 No		FALSE			
90	48439bea8554956d8a577b5ad63f9524	0 No		FALSE			
91	79aebaf36d9ccd10d0f1b2a9dff9543c	0 No		FALSE			
92	ea0319686c422efc9fe9c0364a6fb117	0 No		FALSE			
93	915f2ed898947d610e3b41c10bed72fe	0 No		FALSE			
94	24b64d38e5025f28bd5c0be8fd6ae9be	0 No		FALSE			
95	1a124244c3f5501bc0a5c96ff2387cc0	1 Yes		TRUE	0 No		FALSE
96	fe4b00562e4aaa53b4b6956d0631f021	0 No		FALSE			
97	9cc94bb3a1e6a029c54e1baaad346055	0 No		FALSE			
98	16e7110386a7c024adcb4753cdd042b8	0 No		FALSE			
99	f78cf5785eba1985f5bdb9de8dfdda69	1 Yes		TRUE	0 No		FALSE
100	65bb23364ae1581e38e35b166d47ef1e	0 No		FALSE			
101	ae712b0271669b79479c8051e56956cc	0 No		FALSE			
102	3aaae9b5b7a39f4a6b4febedc5152c2f	0 No		FALSE			
103	50eb0912d0efb00dee1b0590a48c8668	0 No		FALSE			
104	8a4040d2531281194752475dc2c53609	0 No		FALSE			
105	5aaa2d5e9596cccc55ca93a8d7de6127	0 No		FALSE			
106	b20068a41d1199ada2e55b5fdfd254f2	0 No		FALSE			
107	e90cb86f2b59212724bce3b2dad53276	0 No		FALSE			
108	7c196c58dbe549119218158b2b28d8d	0 No		FALSE			
109	bc28535824b91a4a5b7cce99bfe8d4f	0 No		FALSE			

# Boolean Data

	A	B	C	D	E	F	G
1	ID.x	AttendedBootcamp	AttendedBootCampYesNo	AttendedBootcampTF	BootcampLoan	LoanYesNo	LoanTF
89	6ca993739cf368a8b764ecb355359da2		0 No		FALSE		
90	48439bea8554956d8a577b5ad63f9524		0 No		FALSE		
91	79aebaf36d9ccd10d0f1b2a9dff9543c		0 No		FALSE		
92	ea0319686c422efc9fe9c0364a6fb117		0 No		FALSE		
93	915f2ed898947d610e3b41c10bed72fe		0 No		FALSE		
94	24b64d38e5025f28bd5c0be8fd6ae9be		0 No		FALSE		
95	1a124244c3f5501bc0a5c96ff2387cc0		1 Yes		TRUE	0 No	FALSE
96	fe4b00562e4aaa53b4b6956d0631f021		0 No		FALSE		
97	9cc94bb3a1e6a029c54e1baaad346055		0 No		FALSE		
98	16e7110386a7c024adcb4753cdd042b8		0 No		FALSE		
99	f78cf5785eba1985f5bdb9de8dfdda69		1 Yes		TRUE	0 No	FALSE
100	65bb23364ae1581e38e35b166d47ef1e		0 No		FALSE		
101	ae712b0271669b79479c8051e56956cc		0 No		FALSE		
102	3aaae9b5b7a39f4a6b4febedc5152c2f		0 No		FALSE		
103	50eb0912d0efb00dee1b0590a48c8668		0 No		FALSE		
104	8a4040d2531281194752475dc2c53609		0 No		FALSE		
105	5aaa2d5e9596cccc55ca93a8d7de6127		0 No		FALSE		
106	b20068a41d1199ada2e55b5fdfd254f2		0 No		FALSE		
107	e90cb86f2b59212724bce3b2dad53276		0 No		FALSE		
108	7c196c58dbe549119218158b2b28d8d		0 No		FALSE		
109	bc28535824b91a4a5b7cce99bfe8d4f		0 No		FALSE		

# Boolean Data

	A	B	C	D	E	F	G
1	ID.x	AttendedBootcamp	AttendedBootCampYesNo	AttendedBootcampTF	BootcampLoan	LoanYesNo	LoanTF
89	6ca993739cf368a8b764ecb355359da2		0 No		FALSE		
90	48439bea8554956d8a577b5ad63f9524		0 No		FALSE		
91	79aebaf36d9ccd10d0f1b2a9dff9543c		0 No		FALSE		
92	ea0319686c422efc9fe9c0364a6fb117		0 No		FALSE		
93	915f2ed898947d610e3b41c10bed72fe		0 No		FALSE		
94	24b64d38e5025f28bd5c0be8fd6ae9be		0 No		FALSE		
95	1a124244c3f5501bc0a5c96ff2387cc0		1 Yes		TRUE	0 No	FALSE
96	fe4b00562e4aaa53b4b6956d0631f021		0 No		FALSE		
97	9cc94bb3a1e6a029c54e1baaad346055		0 No		FALSE		
98	16e7110386a7c024adcb4753cdd042b8		0 No		FALSE		
99	f78cf5785eba1985f5bdb9de8dfdda69		1 Yes		TRUE	0 No	FALSE
100	65bb23364ae1581e38e35b166d47ef1e		0 No		FALSE		
101	ae712b0271669b79479c8051e56956cc		0 No		FALSE		
102	3aaae9b5b7a39f4a6b4febedc5152c2f		0 No		FALSE		
103	50eb0912d0efb00dee1b0590a48c8668		0 No		FALSE		
104	8a4040d2531281194752475dc2c53609		0 No		FALSE		
105	5aaa2d5e9596cccc55ca93a8d7de6127		0 No		FALSE		
106	b20068a41d1199ada2e55b5fdfd254f2		0 No		FALSE		
107	e90cb86f2b59212724bce3b2dad53276		0 No		FALSE		
108	7c196c58dbe549119218158b2b28d8d		0 No		FALSE		
109	bc28535824b91a4a5b7cce99bfe8d4f		0 No		FALSE		

# Boolean Data

	A	B	C	D	E	F	G
1	ID.x	AttendedBootcamp	AttendedBootCampYesNo	AttendedBootcampTF	BootcampLoan	LoanYesNo	LoanTF
89	6ca993739cf368a8b764ecb355359da2	0 No		FALSE			
90	48439bea8554956d8a577b5ad63f9524	0 No		FALSE			
91	79aebaf36d9ccd10d0f1b2a9dff9543c	0 No		FALSE			
92	ea0319686c422efc9fe9c0364a6fb117	0 No		FALSE			
93	915f2ed898947d610e3b41c10bed72fe	0 No		FALSE			
94	24b64d38e5025f28bd5c0be8fd6ae9be	0 No		FALSE			
95	1a124244c3f5501bc0a5c96ff2387cc0	1 Yes		TRUE	0 No		FALSE
96	fe4b00562e4aaa53b4b6956d0631f021	0 No		FALSE			
97	9cc94bb3a1e6a029c54e1baaad346055	0 No		FALSE			
98	16e7110386a7c024adcb4753cdd042b8	0 No		FALSE			
99	f78cf5785eba1985f5bdb9de8dfdda69	1 Yes		TRUE	0 No		FALSE
100	65bb23364ae1581e38e35b166d47ef1e	0 No		FALSE			
101	ae712b0271669b79479c8051e56956cc	0 No		FALSE			
102	3aaae9b5b7a39f4a6b4febedc5152c2f	0 No		FALSE			
103	50eb0912d0efb00dee1b0590a48c8668	0 No		FALSE			
104	8a4040d2531281194752475dc2c53609	0 No		FALSE			
105	5aaa2d5e9596cccc55ca93a8d7de6127	0 No		FALSE			
106	b20068a41d1199ada2e55b5fdfd254f2	0 No		FALSE			
107	e90cb86f2b59212724bce3b2dad53276	0 No		FALSE			
108	7c196c58dbe549119218158b2b28d8d	0 No		FALSE			
109	bc28535824b91a4a5b7cce99bfe8d4f	0 No		FALSE			

# Boolean Data

	A	B	C	D	E	F	G
1	ID.x	AttendedBootcamp	AttendedBootCampYesNo	AttendedBootcampTF	BootcampLoan	LoanYesNo	LoanTF
89	6ca993739cf368a8b764ecb355359da2		0 No	FALSE			
90	48439bea8554956d8a577b5ad63f9524		0 No	FALSE			
91	79aebaf36d9ccd10d0f1b2a9dff9543c		0 No	FALSE			
92	ea0319686c422efc9fe9c0364a6fb117		0 No	FALSE			
93	915f2ed898947d610e3b41c10bed72fe		0 No	FALSE			
94	24b64d38e5025f28bd5c0be8fd6ae9be		0 No	FALSE			
95	1a124244c3f5501bc0a5c96ff2387cc0		1 Yes	TRUE	0 No		FALSE
96	fe4b00562e4aaa53b4b6956d0631f021		0 No	FALSE			
97	9cc94bb3a1e6a029c54e1baaad346055		0 No	FALSE			
98	16e7110386a7c024adcb4753cdd042b8		0 No	FALSE			
99	f78cf5785eba1985f5bdb9de8dfdda69		1 Yes	TRUE	0 No		FALSE
100	65bb23364ae1581e38e35b166d47ef1e		0 No	FALSE			
101	ae712b0271669b79479c8051e56956cc		0 No	FALSE			
102	3aaae9b5b7a39f4a6b4febedc5152c2f		0 No	FALSE			
103	50eb0912d0efb00dee1b0590a48c8668		0 No	FALSE			
104	8a4040d2531281194752475dc2c53609		0 No	FALSE			
105	5aaa2d5e9596cccc55ca93a8d7de6127		0 No	FALSE			
106	b20068a41d1199ada2e55b5fdfd254f2		0 No	FALSE			
107	e90cb86f2b59212724bce3b2dad53276		0 No	FALSE			
108	7c196c58dbe549119218158b2b28d8d		0 No	FALSE			
109	bc28535824b91a4a5b7cce99bfe8d4f		0 No	FALSE			

# Boolean Data

	A	B	C	D	E	F	G
1	ID.x	AttendedBootcamp	AttendedBootCampYesNo	AttendedBootcampTF	BootcampLoan	LoanYesNo	LoanTF
89	6ca993739cf368a8b764ecb355359da2		0 No		FALSE		
90	48439bea8554956d8a577b5ad63f9524		0 No		FALSE		
91	79aebaf36d9ccd10d0f1b2a9dff9543c		0 No		FALSE		
92	ea0319686c422efc9fe9c0364a6fb117		0 No		FALSE		
93	915f2ed898947d610e3b41c10bed72fe		0 No		FALSE		
94	24b64d38e5025f28bd5c0be8fd6ae9be		0 No		FALSE		
95	1a124244c3f5501bc0a5c96ff2387cc0		1 Yes		TRUE	0 No	FALSE
96	fe4b00562e4aaa53b4b6956d0631f021		0 No		FALSE		
97	9cc94bb3a1e6a029c54e1baaad346055		0 No		FALSE		
98	16e7110386a7c024adcb4753cdd042b8		0 No		FALSE		
99	f78cf5785eba1985f5bdb9de8dfdda69		1 Yes		TRUE	0 No	FALSE
100	65bb23364ae1581e38e35b166d47ef1e		0 No		FALSE		
101	ae712b0271669b79479c8051e56956cc		0 No		FALSE		
102	3aaae9b5b7a39f4a6b4febedc5152c2f		0 No		FALSE		
103	50eb0912d0efb00dee1b0590a48c8668		0 No		FALSE		
104	8a4040d2531281194752475dc2c53609		0 No		FALSE		
105	5aaa2d5e9596cccc55ca93a8d7de6127		0 No		FALSE		
106	b20068a41d1199ada2e55b5fdfd254f2		0 No		FALSE		
107	e90cb86f2b59212724bce3b2dad53276		0 No		FALSE		
108	7c196c58dbe549119218158b2b28d8d		0 No		FALSE		
109	bc28535824b91a4a5b7cce99bfe8d4f		0 No		FALSE		

# pandas and Booleans

```
bootcamp_data = pd.read_excel("fcc_survey_booleans.xlsx")
print(bootcamp_data.dtypes)
```

```
ID.x          object
AttendedBootcamp    float64
AttendedBootCampYesNo   object
AttendedBootcampTF    float64
BootcampLoan        float64
LoanYesNo          object
LoanTF              float64
dtype: object
```

# pandas and Booleans

```
# Count True values  
print(bootcamp_data.sum())
```

```
AttendedBootcamp           38  
AttendedBootcampTF         38  
BootcampLoan                14  
LoanTF                      14  
dtype: object
```

```
# Count NAs  
print(bootcamp_data.isna().sum())
```

```
ID.x                         0  
AttendedBootcamp               0  
AttendedBootCampYesNo          0  
AttendedBootcampTF              0  
BootcampLoan                   964  
LoanYesNo                     964  
LoanTF                        964  
dtype: int64
```

```
# Load data, casting True/False columns as Boolean
bool_data = pd.read_excel("fcc_survey_booleans.xlsx",
                           dtype={"AttendedBootcamp": bool,
                                  "AttendedBootCampYesNo": bool,
                                  "AttendedBootcampTF":bool,
                                  "BootcampLoan": bool,
                                  "LoanYesNo": bool,
                                  "LoanTF": bool})

print(bool_data.dtypes)
```

```
ID.x          object
AttendedBootcamp    bool
AttendedBootCampYesNo    bool
AttendedBootcampTF    bool
BootcampLoan        bool
LoanYesNo          bool
LoanTF             bool
dtype: object
```

```
# Count True values  
print(bool_data.sum())
```

```
AttendedBootcamp           38  
AttendedBootCampYesNo     1000  
AttendedBootcampTF         38  
BootcampLoan               978  
LoanYesNo                 1000  
LoanTF                     978  
dtype: object
```

```
# Count NA values  
print(bool_data.isna().sum())
```

```
ID.x                      0  
AttendedBootcamp           0  
AttendedBootCampYesNo      0  
AttendedBootcampTF         0  
BootcampLoan               0  
LoanYesNo                 0  
LoanTF                     0  
dtype: int64
```

# pandas and Booleans

- pandas loads True / False columns as float data by default
- Specify a column should be bool with read\_excel()'s dtype argument
- Boolean columns can *only* have True and False values
- NA/missing values in Boolean columns are changed to True
- pandas automatically recognizes some values as True / False in Boolean columns
- Unrecognized values in a Boolean column are also changed to True

# Setting Custom True/False Values

- Use `read_excel()`'s `true_values` argument to set custom `True` values
- Use `false_values` to set custom `False` values
- Each takes a list of values to treat as `True` / `False`, respectively
- Custom `True` / `False` values are only applied to columns set as Boolean

# Setting Custom True/False Values

```
# Load data with Boolean dtypes and custom T/F values
bool_data = pd.read_excel("fcc_survey_booleans.xlsx",
                           dtype={"AttendedBootcamp": bool,
                                   "AttendedBootCampYesNo": bool,
                                   "AttendedBootcampTF":bool,
                                   "BootcampLoan": bool,
                                   "LoanYesNo": bool,
                                   "LoanTF": bool},
                           true_values=[ "Yes" ],
                           false_values=[ "No" ])
```

# Setting Custom True/False Values

```
print(bool_data.sum())
```

```
AttendedBootcamp           38
AttendedBootCampYesNo      38
AttendedBootcampTF         38
BootcampLoan               978
LoanYesNo                 978
LoanTF                     978
dtype: object
```

# Boolean Considerations

- Are there missing values, or could there be in the future?
- How will this column be used in analysis?
- What would happen if a value were incorrectly coded as `True` ?
- Could the data be modeled another way (e.g., as floats or integers)?

# Let's practice!

**STREAMLINED DATA INGESTION WITH PANDAS**

# Modifying imports: parsing dates

STREAMLINED DATA INGESTION WITH PANDAS



Amany Mahfouz  
Instructor

# Date and Time Data

- Dates and times have their own data type and internal representation
- Datetime values can be translated into string representations
- Common set of codes to describe datetime string formatting

# pandas and Datetimes

- Datetime columns are loaded as objects (strings) by default
- Specify that columns have datetimes with the `parse_dates` argument (not `dtype` !)
- `parse_dates` can accept:
  - a list of column names or numbers to parse
  - a list containing lists of columns to combine and parse
  - a dictionary where keys are new column names and values are lists of columns to parse together

# pandas and Datetimes

	BG	BH	BI	BJ	BK
1	Part1StartTime	Part1EndTime	Part2StartDate	Part2StartTime	Part2EndTime
2	2016-03-29 21:23:13	2016-03-29 21:24:53	2016-03-29	21:24:57	03292016 21:27:25
3	2016-03-29 21:24:59	2016-03-29 21:27:09	2016-03-29	21:27:14	03292016 21:29:10
4	2016-03-29 21:25:37	2016-03-29 21:27:11	2016-03-29	21:27:13	03292016 21:28:21
5	2016-03-29 21:21:37	2016-03-29 21:28:47	2016-03-29	21:28:51	03292016 21:30:51
6	2016-03-29 21:26:22	2016-03-29 21:29:27	2016-03-29	21:29:32	03292016 21:31:54
7	2016-03-29 21:29:33	2016-03-29 21:30:40	2016-03-29	21:30:44	03292016 21:32:19
8	2016-03-29 21:24:58	2016-03-29 21:31:49	2016-03-29	21:31:51	03292016 21:33:08
9	2016-03-29 21:30:44	2016-03-29 21:33:58	2016-03-29	21:34:04	03292016 21:37:32
10	2016-03-29 21:33:05	2016-03-29 21:34:21	2016-03-29	21:34:25	03292016 21:35:40
11	2016-03-29 21:34:52	2016-03-29 21:36:17	2016-03-29	21:36:23	03292016 21:39:18
12	2016-03-29 21:32:59	2016-03-29 21:36:26	2016-03-29	21:36:29	03292016 21:39:27

# pandas and Datetimes

	BG	BH	BI	BJ	BK
1	Part1StartTime	Part1EndTime	Part2StartDate	Part2StartTime	Part2EndTime
2	2016-03-29 21:23:13	2016-03-29 21:24:53	2016-03-29	21:24:57	03292016 21:27:25
3	2016-03-29 21:24:59	2016-03-29 21:27:09	2016-03-29	21:27:14	03292016 21:29:10
4	2016-03-29 21:25:37	2016-03-29 21:27:11	2016-03-29	21:27:13	03292016 21:28:21
5	2016-03-29 21:21:37	2016-03-29 21:28:47	2016-03-29	21:28:51	03292016 21:30:51
6	2016-03-29 21:26:22	2016-03-29 21:29:27	2016-03-29	21:29:32	03292016 21:31:54
7	2016-03-29 21:29:33	2016-03-29 21:30:40	2016-03-29	21:30:44	03292016 21:32:19
8	2016-03-29 21:24:58	2016-03-29 21:31:49	2016-03-29	21:31:51	03292016 21:33:08
9	2016-03-29 21:30:44	2016-03-29 21:33:58	2016-03-29	21:34:04	03292016 21:37:32
10	2016-03-29 21:33:05	2016-03-29 21:34:21	2016-03-29	21:34:25	03292016 21:35:40
11	2016-03-29 21:34:52	2016-03-29 21:36:17	2016-03-29	21:36:23	03292016 21:39:18
12	2016-03-29 21:32:59	2016-03-29 21:36:26	2016-03-29	21:36:29	03292016 21:39:27

# pandas and Datetimes

	BG	BH	BI	BJ	BK
1	Part1StartTime	Part1EndTime	Part2StartDate	Part2StartTime	Part2EndTime
2	2016-03-29 21:23:13	2016-03-29 21:24:53	2016-03-29	21:24:57	03292016 21:27:25
3	2016-03-29 21:24:59	2016-03-29 21:27:09	2016-03-29	21:27:14	03292016 21:29:10
4	2016-03-29 21:25:37	2016-03-29 21:27:11	2016-03-29	21:27:13	03292016 21:28:21
5	2016-03-29 21:21:37	2016-03-29 21:28:47	2016-03-29	21:28:51	03292016 21:30:51
6	2016-03-29 21:26:22	2016-03-29 21:29:27	2016-03-29	21:29:32	03292016 21:31:54
7	2016-03-29 21:29:33	2016-03-29 21:30:40	2016-03-29	21:30:44	03292016 21:32:19
8	2016-03-29 21:24:58	2016-03-29 21:31:49	2016-03-29	21:31:51	03292016 21:33:08
9	2016-03-29 21:30:44	2016-03-29 21:33:58	2016-03-29	21:34:04	03292016 21:37:32
10	2016-03-29 21:33:05	2016-03-29 21:34:21	2016-03-29	21:34:25	03292016 21:35:40
11	2016-03-29 21:34:52	2016-03-29 21:36:17	2016-03-29	21:36:23	03292016 21:39:18
12	2016-03-29 21:32:59	2016-03-29 21:36:26	2016-03-29	21:36:29	03292016 21:39:27

# pandas and Datetimes

	BG	BH	BI	BJ	BK
1	Part1StartTime	Part1EndTime	Part2StartDate	Part2StartTime	Part2EndTime
2	2016-03-29 21:23:13	2016-03-29 21:24:53	2016-03-29	21:24:57	03292016 21:27:25
3	2016-03-29 21:24:59	2016-03-29 21:27:09	2016-03-29	21:27:14	03292016 21:29:10
4	2016-03-29 21:25:37	2016-03-29 21:27:11	2016-03-29	21:27:13	03292016 21:28:21
5	2016-03-29 21:21:37	2016-03-29 21:28:47	2016-03-29	21:28:51	03292016 21:30:51
6	2016-03-29 21:26:22	2016-03-29 21:29:27	2016-03-29	21:29:32	03292016 21:31:54
7	2016-03-29 21:29:33	2016-03-29 21:30:40	2016-03-29	21:30:44	03292016 21:32:19
8	2016-03-29 21:24:58	2016-03-29 21:31:49	2016-03-29	21:31:51	03292016 21:33:08
9	2016-03-29 21:30:44	2016-03-29 21:33:58	2016-03-29	21:34:04	03292016 21:37:32
10	2016-03-29 21:33:05	2016-03-29 21:34:21	2016-03-29	21:34:25	03292016 21:35:40
11	2016-03-29 21:34:52	2016-03-29 21:36:17	2016-03-29	21:36:23	03292016 21:39:18
12	2016-03-29 21:32:59	2016-03-29 21:36:26	2016-03-29	21:36:29	03292016 21:39:27

# Parsing Dates

```
# List columns of dates to parse  
date_cols = ["Part1StartTime", "Part1EndTime"]  
  
# Load file, parsing standard datetime columns  
survey_df = pd.read_excel("fcc_survey.xlsx",  
                           parse_dates=date_cols)
```

# Parsing Dates

```
# Check data types of timestamp columns
print(survey_df[["Part1StartTime",
                 "Part1EndTime",
                 "Part2StartDate",
                 "Part2StartTime",
                 "Part2EndTime"]].dtypes)
```

```
Part1StartTime      datetime64[ns]
Part1EndTime        datetime64[ns]
Part2StartDate      object
Part2StartTime      object
Part2EndTime        object
dtype: object
```

# Parsing Dates

```
# List columns of dates to parse
date_cols = ["Part1StartTime",
             "Part1EndTime",
             [ ["Part2StartDate", "Part2StartTime"]]]

# Load file, parsing standard and split datetime columns
survey_df = pd.read_excel("fcc_survey.xlsx",
                           parse_dates=date_cols)

print(survey_df.head(3))
```

```
Part2StartDate_Part2StartTime    Age      ...
0      2016-03-29 21:24:57    28.0      ...
1      2016-03-29 21:27:14    22.0      ...
2      2016-03-29 21:27:13    19.0      ...

[3 rows x 98 columns]
```

# Parsing Dates

```
# List columns of dates to parse
date_cols = {"Part1Start": "Part1StartTime",
             "Part1End": "Part1EndTime",
             "Part2Start": ["Part2StartDate",
                           "Part2StartTime"]}
# Load file, parsing standard and split datetime columns
survey_df = pd.read_excel("fcc_survey.xlsx",
                          parse_dates=date_cols)

print(survey_df.Part2Start.head(3))
```

```
0    2016-03-29 21:24:57
1    2016-03-29 21:27:14
2    2016-03-29 21:27:13
Name: Part2Start, dtype: datetime64[ns]
```

# Non-Standard Dates

- `parse_dates` doesn't work with non-standard datetime formats
- Use `pd.to_datetime()` after loading data if `parse_dates` won't work
- `to_datetime()` arguments:
  - Data frame and column to convert
  - `format` : string representation of datetime format

# Datetime Formatting

- Describe datetime string formatting with codes and characters
- Refer to [strftime.org](http://strftime.org) for the full list

# Datetime Formatting

Code	Meaning	Example
%Y	Year (4-digit)	1999
%m	Month (zero-padded)	03
%d	Day (zero-padded)	01
%H	Hour (24-hour clock)	21
%M	Minute (zero-padded)	09
%S	Second (zero-padded)	05

# Parsing Non-Standard Dates

	BG	BH	BI	BJ	BK
1	Part1StartTime	Part1EndTime	Part2StartDate	Part2StartTime	Part2EndTime
2	2016-03-29 21:23:13	2016-03-29 21:24:53	2016-03-29	21:24:57	03292016 21:27:25
3	2016-03-29 21:24:59	2016-03-29 21:27:09	2016-03-29	21:27:14	03292016 21:29:10
4	2016-03-29 21:25:37	2016-03-29 21:27:11	2016-03-29	21:27:13	03292016 21:28:21
5	2016-03-29 21:21:37	2016-03-29 21:28:47	2016-03-29	21:28:51	03292016 21:30:51
6	2016-03-29 21:26:22	2016-03-29 21:29:27	2016-03-29	21:29:32	03292016 21:31:54
7	2016-03-29 21:29:33	2016-03-29 21:30:40	2016-03-29	21:30:44	03292016 21:32:19
8	2016-03-29 21:24:58	2016-03-29 21:31:49	2016-03-29	21:31:51	03292016 21:33:08
9	2016-03-29 21:30:44	2016-03-29 21:33:58	2016-03-29	21:34:04	03292016 21:37:32
10	2016-03-29 21:33:05	2016-03-29 21:34:21	2016-03-29	21:34:25	03292016 21:35:40
11	2016-03-29 21:34:52	2016-03-29 21:36:17	2016-03-29	21:36:23	03292016 21:39:18
12	2016-03-29 21:32:59	2016-03-29 21:36:26	2016-03-29	21:36:29	03292016 21:39:27

```
format_string = "%m%d%Y %H:%M:%S"  
  
survey_df["Part2EndTime"] = pd.to_datetime(survey_df["Part2EndTime"],  
                                         format=format_string)
```

# Parsing Non-Standard Dates

```
print(survey_df.Part2EndTime.head())
```

```
0    2016-03-29 21:27:25  
1    2016-03-29 21:29:10  
2    2016-03-29 21:28:21  
3    2016-03-29 21:30:51  
4    2016-03-29 21:31:54  
Name: Part2EndTime, dtype: datetime64[ns]
```

# Let's practice!

**STREAMLINED DATA INGESTION WITH PANDAS**

# Introduction to databases

STREAMLINED DATA INGESTION WITH PANDAS



Amany Mahfouz  
Instructor

# Relational Databases

- Data about entities is organized into tables
- Each row or record is an instance of an entity
- Each column has information about an attribute
- Tables can be linked to each other via unique keys
- Support more data, multiple simultaneous users, and data quality controls
- Data types are specified for each column
- SQL (Structured Query Language) to interact with databases

# Common Relational Databases



- SQLite databases are computer files

# Connecting to Databases

- Two-step process:
  1. Create way to connect to database
  2. Query database



# Creating a Database Engine



- `sqlalchemy`'s `create_engine()` makes an engine to handle database connections
  - Needs string URL of database to connect to
  - SQLite URL format: `sqlite:///filename.db`

# Querying Databases

- `pd.read_sql(query, engine)` to load in data from a database
- Arguments
  - `query` : String containing SQL query to run or table to load
  - `engine` : Connection/database engine object

# SQL Review: SELECT

- Used to query data from a database
- Basic syntax:

```
SELECT [column_names] FROM [table_name];
```

- To get all data in a table:

```
SELECT * FROM [table_name];
```

- Code style: keywords in ALL CAPS, semicolon (;) to end a statement

# Getting Data from a Database

```
# Load pandas and sqlalchemy's create_engine
import pandas as pd
from sqlalchemy import create_engine

# Create database engine to manage connections
engine = create_engine("sqlite:///data.db")

# Load entire weather table by table name
weather = pd.read_sql("weather", engine)
```

```
# Create database engine to manage connections
engine = create_engine("sqlite:///data.db")

# Load entire weather table with SQL
weather = pd.read_sql("SELECT * FROM weather", engine)

print(weather.head())
```

```
      station          name  latitude    ...  prcp  snow  tavg  tmax  tmin
0  USW00094728  NY CITY CENTRAL PARK, NY US  40.77898  ...   0.00   0.0     52    42
1  USW00094728  NY CITY CENTRAL PARK, NY US  40.77898  ...   0.00   0.0     48    39
2  USW00094728  NY CITY CENTRAL PARK, NY US  40.77898  ...   0.00   0.0     48    42
3  USW00094728  NY CITY CENTRAL PARK, NY US  40.77898  ...   0.00   0.0     51    40
4  USW00094728  NY CITY CENTRAL PARK, NY US  40.77898  ...   0.75   0.0     61    50

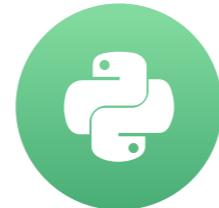
[5 rows x 13 columns]
```

# Let's practice!

**STREAMLINED DATA INGESTION WITH PANDAS**

# Refining imports with SQL queries

STREAMLINED DATA INGESTION WITH PANDAS



Amany Mahfouz  
Instructor

# SELECTing Columns

- `SELECT [column names] FROM [table name];`
- Example:

```
SELECT date, tavg  
FROM weather;
```

# WHERE Clauses

- Use a `WHERE` clause to selectively import records

```
SELECT [column_names]  
FROM [table_name]  
WHERE [condition];
```

# Filtering by Numbers

- Compare numbers with mathematical operators
  - =
  - > and >=
  - < and <=
  - <> (not equal to)
  - Example:

```
SELECT *
FROM weather
WHERE tmax > 32;
```

# Filtering Text

- Match exact strings with the `=` sign and the text to match
- String matching is case-sensitive
- Example:

```
/* Get records about incidents in Brooklyn */  
SELECT *  
FROM hpd311calls  
WHERE borough = 'BROOKLYN' ;
```

# SQL and pandas

```
# Load libraries
import pandas as pd
from sqlalchemy import create_engine

# Create database engine
engine = create_engine("sqlite:///data.db")

# Write query to get records from Brooklyn
query = """SELECT *
            FROM hpd311calls
            WHERE borough = 'BROOKLYN';"""

# Query the database
brooklyn_calls = pd.read_sql(query, engine)
print(brooklyn_calls.borough.unique())
```

```
[ 'BROOKLYN' ]
```

# Combining Conditions: AND

- WHERE clauses with AND return records that meet all conditions

```
# Write query to get records about plumbing in the Bronx
and_query = """SELECT *
                  FROM hp311calls
                 WHERE borough = 'BRONX'
                   AND complaint_type = 'PLUMBING' ;"""

# Get calls about plumbing issues in the Bronx
bx_plumbing_calls = pd.read_sql(and_query, engine)

# Check record count
print(bx_plumbing_calls.shape)
```

```
(2016, 8)
```

# Combining Conditions: OR

- WHERE clauses with OR return records that meet at least one condition

```
# Write query to get records about water leaks or plumbing
or_query = """SELECT *
              FROM hp311calls
              WHERE complaint_type = 'WATER LEAK'
                    OR complaint_type = 'PLUMBING' ;"""

# Get calls that are about plumbing or water leaks
leaks_or_plumbing = pd.read_sql(or_query, engine)

# Check record count
print(leaks_or_plumbing.shape)
```

```
(10684, 8)
```

# Let's practice!

**STREAMLINED DATA INGESTION WITH PANDAS**

# More complex SQL queries

STREAMLINED DATA INGESTION WITH PANDAS



Amany Mahfouz  
Instructor

# Getting DISTINCT Values

- Get unique values for one or more columns with `SELECT DISTINCT`
- Syntax:

```
SELECT DISTINCT [column names] FROM [table];
```

- Remove duplicate records:

```
SELECT DISTINCT * FROM [table];
```

```
/* Get unique street addresses and boroughs */
SELECT DISTINCT incident_address,
               borough
        FROM hpd311calls;
```

# Aggregate Functions

- Query a database directly for descriptive statistics
- Aggregate functions
  - SUM
  - AVG
  - MAX
  - MIN
  - COUNT

# Aggregate Functions

- `SUM` , `AVG` , `MAX` , `MIN`
  - Each takes a single column name

```
SELECT AVG(tmax) FROM weather;
```

- `COUNT`
  - Get number of rows that meet query conditions

```
SELECT COUNT(*) FROM [table_name];
```

- Get number of unique values in a column

```
SELECT COUNT(DISTINCT [column_names]) FROM [table_name];
```

# GROUP BY

- Aggregate functions calculate a single summary statistic by default
- Summarize data by categories with `GROUP BY` statements
- Remember to also select the column you're grouping by!

```
/* Get counts of plumbing calls by borough */
SELECT borough,
       COUNT(*)
FROM hpd311calls
WHERE complaint_type = 'PLUMBING'
GROUP BY borough;
```

# Counting by Groups

```
# Create database engine
engine = create_engine("sqlite:///data.db")

# Write query to get plumbing call counts by borough
query = """SELECT borough, COUNT(*)
            FROM hpd311calls
            WHERE complaint_type = 'PLUMBING'
            GROUP BY borough"""

# Query database and create data frame
plumbing_call_counts = pd.read_sql(query, engine)
```

# Counting by Groups

```
print(plumbing_call_counts)
```

```
      borough  COUNT(*)  
0      BRONX      2016  
1    BROOKLYN      2702  
2   MANHATTAN      1413  
3     QUEENS       808  
4  STATEN ISLAND      178
```

# Let's practice!

**STREAMLINED DATA INGESTION WITH PANDAS**

# Loading multiple tables with joins

STREAMLINED DATA INGESTION WITH PANDAS



Amany Mahfouz  
Instructor

# Keys

- Database records have unique identifiers, or keys

unique_key	created_date	agency	complaint_type	incident_zip	incident_address	community_board	borough
Filter	Filter	Fi...	Filter	Filter	Filter	Filter	Filter
38070822	01/01/2018	HPD	HEAT/HOT WATER	10468	2786 JEROME AVEN...	07 BRONX	BRONX
38065299	01/01/2018	HPD	PLUMBING	10003	323 EAST 12 STRE...	03 MANHATTAN	MANHA...
38066653	01/01/2018	HPD	HEAT/HOT WATER	10452	1235 GRAND CONC...	04 BRONX	BRONX
38070264	01/01/2018	HPD	HEAT/HOT WATER	10032	656 WEST 171 STR...	12 MANHATTAN	MANHA...
38072466	01/01/2018	HPD	HEAT/HOT WATER	11213	1030 PARK PLACE	08 BROOKLYN	BROOK...

# Keys

- Database records have unique identifiers, or keys

course_code	course_title	department	faculty	credits	level	term	instructor_id
Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
ANAT212	Molecular Me...	Anatomy and ...	Medicine	3	Undergra...	Spring	199157185
ANAT214	Systemic Hu...	Anatomy and ...	Medicine	3	Undergra...	Fall	199157185
ANAT261	Introduction t...	Anatomy and ...	Medicine	4	Undergra...	Fall	175319914
ANAT69D1	Cell and Deve...	Anatomy and ...	Medicine	3	Graduate	Fall	193644393
ANTH201	Prehistoric Ar...	Anthropology	Arts	3	Undergra...	Spring	182832540

# Keys

- Database records have unique identifiers, or keys

course_code	course_title	department	faculty	credits	level	term	instructor_id
Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
ANAT212	Molecular Me...	Anatomy and ...	Medicine	3	Undergra...	Spring	199157185
ANAT214	Systemic Hu...	Anatomy and ...	Medicine	3	Undergra...	Fall	199157185
ANAT261	Introduction t...	Anatomy and ...	Medicine	4	Undergra...	Fall	175319914
ANAT69D1	Cell and Deve...	Anatomy and ...	Medicine	3	Graduate	Fall	193644393
ANTH201	Prehistoric Ar...	Anthropology	Arts	3	Undergra...	Spring	182832540

# Keys

<a href="#">id</a>	<a href="#">name</a>	<a href="#">title</a>	<a href="#">degree_institution</a>	<a href="#">faculty</a>
Filter	Filter	Filter	Filter	Filter
<a href="#">112013131</a>	<a href="#">Isabel Morales</a>	<a href="#">Assistant Professor</a>	<a href="#">Complutense Uni...</a>	<a href="#">Science</a>
<a href="#">184601849</a>	<a href="#">Abraham Waldman</a>	<a href="#">Assistant Professor</a>	<a href="#">LMU Munich</a>	<a href="#">Science</a>
<a href="#">182832540</a>	<a href="#">Henry Jones</a>	<a href="#">Professor</a>	<a href="#">University of Chi...</a>	<a href="#">Arts</a>
<a href="#">146661411</a>	<a href="#">Anna Ng</a>	<a href="#">Associate Professor</a>	<a href="#">Queen's University</a>	<a href="#">Arts</a>
<a href="#">199157185</a>	<a href="#">Vanessa Hawkins</a>	<a href="#">Professor</a>	<a href="#">Queen's University</a>	<a href="#">Medicine</a>

# Keys

course_code	course_title	department	faculty	credits	level	term	instructor_id	name
ANAT212	Molecular Mechanisms of Cell Function	Anatomy and Cell Biology	Medicine	3	Undergraduate	Spring	199157185	Vanessa Hawkins
ANAT214	Systemic Human Anatomy	Anatomy and Cell Biology	Medicine	3	Undergraduate	Fall	199157185	Vanessa Hawkins
ANAT261	Introduction to Dynamic Histology	Anatomy and Cell Biology	Medicine	4	Undergraduate	Fall	175319914	Arun Agarwal
ANAT69D1	Cell and Developmental Biology	Anatomy and Cell Biology	Medicine	3	Graduate	Fall	193644393	Alexis Cook
ANTH201	Prehistoric Archaeology	Anthropology	Arts	3	Undergraduate	Spring	182832540	Henry Jones

# Joining Tables

station	name	latitude	longitude	elevation	date	month	awnd	prcp	snow	tavg	tmax	tmin
Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	...	Filter	Filter
USW00094728	NY CITY CEN...	40.77898	-73.96925	42.7	01/01/2018	January	7.83	0	0	19	7	
USW00094728	NY CITY CEN...	40.77898	-73.96925	42.7	01/02/2018	January	8.05	0	0	26	13	
USW00094728	NY CITY CEN...	40.77898	-73.96925	42.7	01/03/2018	January	3.13	0	0	30	16	
USW00094728	NY CITY CEN...	40.77898	-73.96925	42.7	01/04/2018	January	12.53	0.76	9.8	29	19	
USW00094728	NY CITY CEN...	40.77898	-73.96925	42.7	01/05/2018	January	12.97	0	0	19	9	
USW00094728	NY CITY CEN...	40.77898	-73.96925	42.7	01/06/2018	January	10.96	0	0	13	6	
USW00094728	NY CITY CEN...	40.77898	-73.96925	42.7	01/07/2018	January	6.49	0	0	18	5	

unique_key	created_date	agency	complaint_type	incident_zip	incident_address	community_board	borough
Filter	Filter	Fi...	Filter	Filter	Filter	Filter	Filter
38070822	01/01/2018	HPD	HEAT/HOT WATER	10468	2786 JEROME AVEN...	07 BRONX	BRONX
38065299	01/01/2018	HPD	PLUMBING	10003	323 EAST 12 STRE...	03 MANHATTAN	MANHATTAN
38066653	01/01/2018	HPD	HEAT/HOT WATER	10452	1235 GRAND CONC...	04 BRONX	BRONX
38070264	01/01/2018	HPD	HEAT/HOT WATER	10032	656 WEST 171 STR...	12 MANHATTAN	MANHATTAN
38072466	01/01/2018	HPD	HEAT/HOT WATER	11213	1030 PARK PLACE	08 BROOKLYN	BROOKLYN

# Joining Tables

```
SELECT *
FROM hpd311calls
```

# Joining Tables

```
SELECT *
  FROM hpd311calls
    JOIN weather
  ON hpd311calls.created_date = weather.date;
```

- Use dot notation ( `table.column` ) when working with multiple tables
- Default join only returns records whose key values appear in both tables
- Make sure join keys are the same data type or nothing will match

# Joining and Filtering

```
/* Get only heat/hot water calls and join in weather data */  
SELECT *  
  FROM hpd311calls  
    JOIN weather  
      ON hpd311calls.created_date = weather.date  
WHERE hpd311calls.complaint_type = 'HEAT/HOT WATER';
```

# Joining and Aggregating

```
/* Get call counts by borough */  
SELECT hpd311calls.borough,  
        COUNT(*)  
    FROM hpd311calls  
GROUP BY hpd311calls.borough;
```

# Joining and Aggregating

```
/* Get call counts by borough  
and join in population and housing counts */  
SELECT hpd311calls.borough,  
        COUNT(*),  
        boro_census.total_population,  
        boro_census.housing_units  
FROM hpd311calls  
GROUP BY hpd311calls.borough
```

# Joining and Aggregating

```
/* Get call counts by borough  
and join in population and housing counts */  
SELECT hpd311calls.borough,  
        COUNT(*),  
        boro_census.total_population,  
        boro_census.housing_units  
FROM hpd311calls  
    JOIN boro_census  
    ON hpd311calls.borough = boro_census.borough  
GROUP BY hpd311calls.borough;
```

```
query = """SELECT hpd311calls.borough,  
                 COUNT(*),  
                 boro_census.total_population,  
                 boro_census.housing_units  
            FROM hpd311calls  
              JOIN boro_census  
                ON hpd311calls.borough = boro_census.borough  
        GROUP BY hpd311calls.borough;"""
```

```
call_counts = pd.read_sql(query, engine)  
print(call_counts)
```

	borough	COUNT(*)	total_population	housing_units
0	BRONX	29874	1455846	524488
1	BROOKLYN	31722	2635121	1028383
2	MANHATTAN	20196	1653877	872645
3	QUEENS	11384	2339280	850422
4	STATEN ISLAND	1322	475948	179179

# Review

- SQL order of keywords

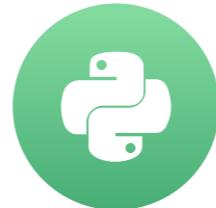
- **SELECT**
- **FROM**
- **JOIN**
- **WHERE**
- **GROUP BY**

# Let's practice!

**STREAMLINED DATA INGESTION WITH PANDAS**

# Introduction to JSON

STREAMLINED DATA INGESTION WITH PANDAS



Amany Mahfouz  
Instructor

# Javascript Object Notation (JSON)

- Common web data format
- Not tabular
  - Records don't have to all have the same set of attributes
- Data organized into collections of objects
- Objects are collections of attribute-value pairs
- Nested JSON: objects within objects

# Reading JSON Data

- `read_json()`
  - Takes a string path to JSON or JSON data as a string
  - Specify data types with `dtype` keyword argument
  - `orient` keyword argument to flag uncommon JSON data layouts
    - possible values in `pandas` documentation

# Data Orientation

- JSON data isn't tabular
  - `pandas` guesses how to arrange it in a table
- `pandas` can automatically handle common orientations

# Record Orientation

- Most common JSON arrangement

```
[  
  {  
    "age_adjusted_death_rate": "7.6",  
    "death_rate": "6.2",  
    "deaths": "32",  
    "leading_cause": "Accidents Except Drug Positioning (V01-X39, X43, X45-X59, Y85-Y86)",  
    "race_ethnicity": "Asian and Pacific Islander",  
    "sex": "F",  
    "year": "2007"  
  },  
  {  
    "age_adjusted_death_rate": "8.1",  
    "death_rate": "8.3",  
    "deaths": "87",  
    ...  
  }]
```

# Column Orientation

- More space-efficient than record-oriented JSON

```
{  
  "age_adjusted_death_rate": {  
    "0": "7.6",  
    "1": "8.1",  
    "2": "7.1",  
    "3": ".",  
    "4": ".",  
    "5": "7.3",  
    "6": "13",  
    "7": "20.6",  
    "8": "17.4",  
    "9": ".",  
    "10": ".",  
    "11": "19.8",  
    ...  
  }  
}
```

# Specifying Orientation

- Split oriented data - `nyc_death_causes.json`

```
{  
  "columns": [  
    "age_adjusted_death_rate",  
    "death_rate",  
    "deaths",  
    "leading_cause",  
    "race_ethnicity",  
    "sex",  
    "year"  
  ],  
  "index": [...],  
  "data": [  
    [  
      "7.6",  

```

# Specifying Orientation

```
import pandas as pd  
  
death_causes = pd.read_json("nyc_death_causes.json",  
                             orient="split")  
  
print(death_causes.head())
```

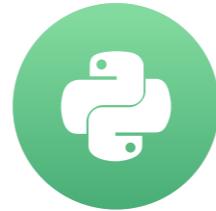
```
age_adjusted_death_rate  death_rate  deaths  leading_cause  race_ethnicity  sex  
0                      7.6        6.2      32  Accidents Except Drug...  Asian and Pacific Islander  M  
1                      8.1        8.3      87  Accidents Except Drug...  Black Non-Hispanic  F  
2                      7.1        6.1      71  Accidents Except Drug...  Hispanic            F  
3                      .          .       .  Accidents Except Drug...  Not Stated/Unknown  M  
4                      .          .       .  Accidents Except Drug...  Other Race/ Ethnicity  M  
  
[ 5 rows x 7 columns]
```

# Let's practice!

**STREAMLINED DATA INGESTION WITH PANDAS**

# Introduction to APIs

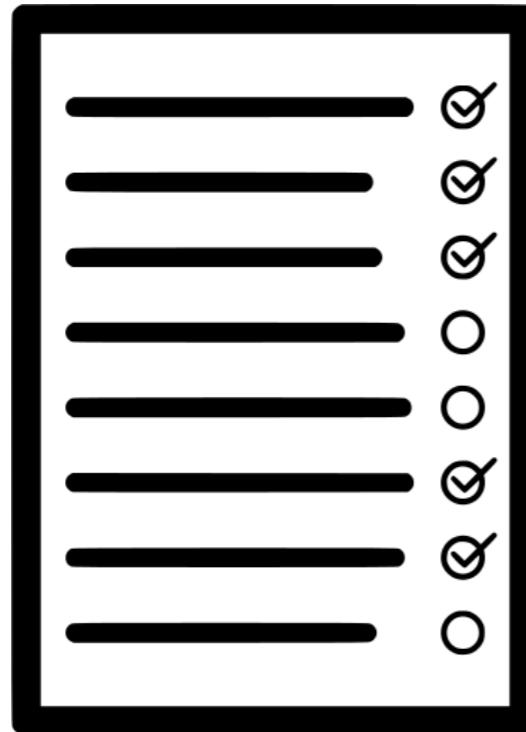
STREAMLINED DATA INGESTION WITH PANDAS



Amany Mahfouz  
Instructor

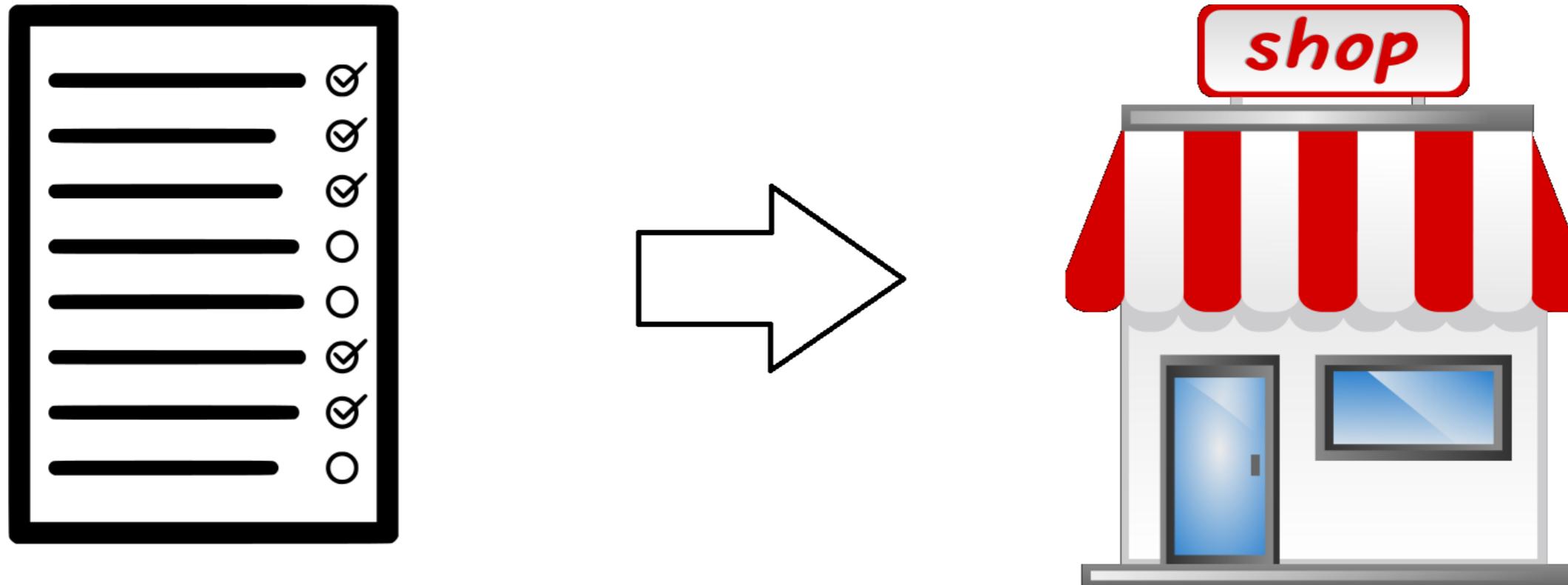
# Application Programming Interfaces

- Defines how a application communicates with other programs
- Way to get data from an application without knowing database details



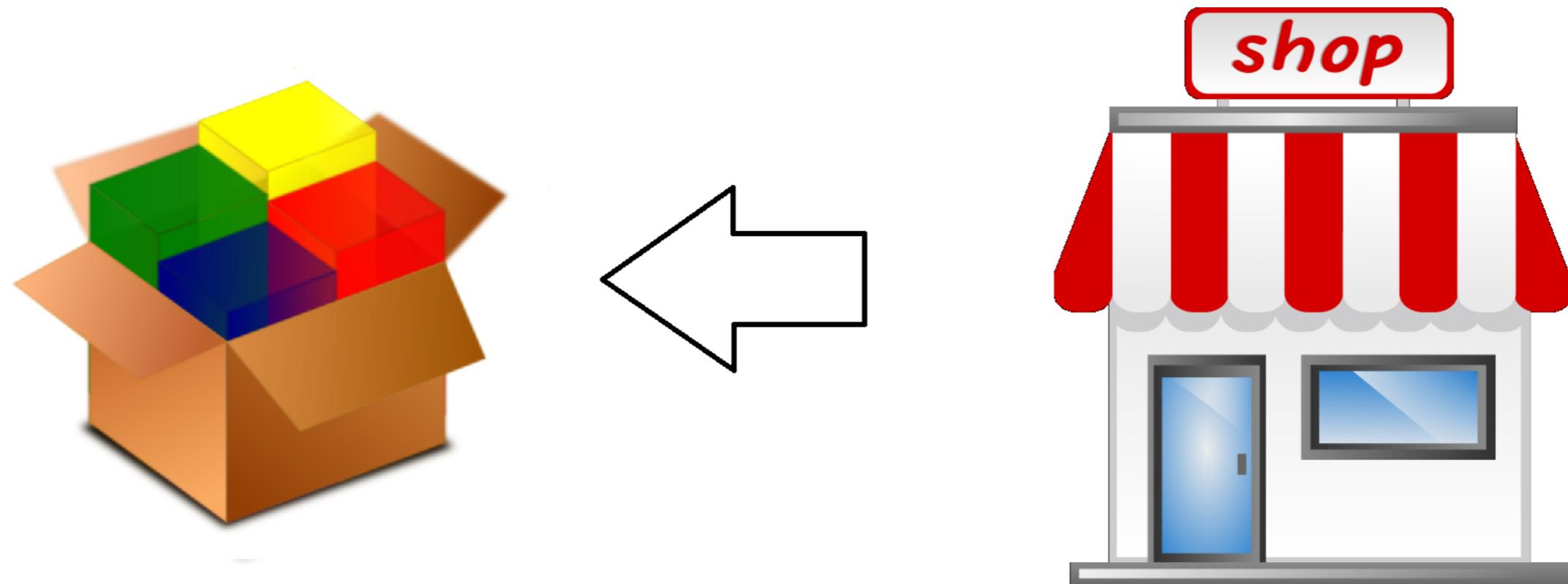
# Application Programming Interfaces

- Defines how a application communicates with other programs
- Way to get data from an application without knowing database details



# Application Programming Interfaces

- Defines how a application communicates with other programs
- Way to get data from an application without knowing database details



# Requests

- Send and get data from websites
- Not tied to a particular API
- `requests.get()` to get data from a URL



**Requests**  
*http for humans*

# requests.get()

- `requests.get(url_string)` to get data from a URL
- Keyword arguments
  - `params` keyword: takes a dictionary of parameters and values to customize API request
  - `headers` keyword: takes a dictionary, can be used to provide user authentication to API
- Result: a `response` object, containing data and metadata
  - `response.json()` will return just the JSON data

# response.json() and pandas

- `response.json()` returns a dictionary
- `read_json()` expects strings, not dictionaries
- Load the response JSON to a data frame with `pd.DataFrame()`
  - `read_json()` will give an error!

# Yelp Business Search API



# Yelp Business Search API

## Request

```
GET https://api.yelp.com/v3/businesses/search
```

## Parameters

These parameters should be in the query string.

Name	Type	Description
term	string	Optional. Search term, for example "food" or "restaurants". The term may also be business names, such as "Starbucks". If term is not included the endpoint will default to searching across businesses from a small number of popular categories.
location	string	Required if either latitude or longitude is not provided. This string indicates the geographic area to be used when searching for businesses. Examples: "New York City", "NYC", "350 5th Ave, New York, NY 10118". Businesses returned in the response may not be strictly within the specified location.
latitude	decimal	Required if location is not provided. Latitude of the location you want to search nearby.

# Yelp Business Search API

## Request

GET <https://api.yelp.com/v3/businesses/search>

## Parameters

These parameters should be in the query string.

Name	Type	Description
term	string	Optional. Search term, for example "food" or "restaurants". The term may also be business names, such as "Starbucks". If term is not included the endpoint will default to searching across businesses from a small number of popular categories.
location	string	Required if either latitude or longitude is not provided. This string indicates the geographic area to be used when searching for businesses. Examples: "New York City", "NYC", "350 5th Ave, New York, NY 10118". Businesses returned in the response may not be strictly within the specified location.
latitude	decimal	Required if location is not provided. Latitude of the location you want to search nearby.

# Yelp Business Search API

## Request

```
GET https://api.yelp.com/v3/businesses/search
```

## Parameters

These parameters should be in the query string.

Name	Type	Description
term	string	Optional. Search term, for example "food" or "restaurants". The term may also be business names, such as "Starbucks". If term is not included the endpoint will default to searching across businesses from a small number of popular categories.
location	string	Required if either latitude or longitude is not provided. This string indicates the geographic area to be used when searching for businesses. Examples: "New York City", "NYC", "350 5th Ave, New York, NY 10118". Businesses returned in the response may not be strictly within the specified location.
latitude	decimal	Required if location is not provided. Latitude of the location you want to search nearby.

# Yelp Business Search API

## Request

```
GET https://api.yelp.com/v3/businesses/search
```

## Parameters

These parameters should be in the query string.

Name	Type	Description
term	string	Optional. Search term, for example "food" or "restaurants". The term may also be business names, such as "Starbucks". If term is not included the endpoint will default to searching across businesses from a small number of popular categories.
location	string	Required if either latitude or longitude is not provided. This string indicates the geographic area to be used when searching for businesses. Examples: "New York City", "NYC", "350 5th Ave, New York, NY 10118". Businesses returned in the response may not be strictly within the specified location.
latitude	decimal	Required if location is not provided. Latitude of the location you want to search nearby.

# Yelp Business Search API

## Response Body

```
{  
    "total": 8228,  
    "businesses": [  
        {  
            "rating": 4,  
            "price": "$",  
            "phone": "+14152520800",  
            "id": "E8RJkjfdcwgtyoPMjQ_olg",  
            "alias": "four-barrel-coffee-san-francisco",  
            "is_closed": false,  
            "categories": [  
                {  
                    "alias": "coffee",  
                    "title": "Coffee & Tea"  
                }  
            ],  
            "review_count": 1738,  
            "name": "Four Barrel Coffee",  
            "url": "https://www.yelp.com/biz/four-barrel-coffee-san-francisco",  
            "...": "..."  
        }  
    ]  
}
```

# Making Requests

```
import requests
import pandas as pd

api_url = "https://api.yelp.com/v3/businesses/search"
# Set up parameter dictionary according to documentation
params = {"term": "bookstore",
           "location": "San Francisco"}
# Set up header dictionary w/ API key according to documentation
headers = {"Authorization": "Bearer {}".format(api_key)}

# Call the API
response = requests.get(api_url,
                        params=params,
                        headers=headers)
```

# Parsing Responses

```
# Isolate the JSON data from the response object  
data = response.json()  
print(data)
```

```
{'businesses': [{  
    'id': '_rbF2ooLcMRA7Kh8neIr4g',  
    'alias': 'city-lights-bookstore-san-francisco',  
    'name': 'City Lights Bookstore',  
    'phone': '(415) 821-7575',  
    'address': '102 Columbus Avenue, San Francisco, CA 94133',  
    'city': 'San Francisco',  
    'state': 'CA',  
    'zip_code': '94133',  
    'country': 'US',  
    'latitude': 37.7749, 'longitude': -122.421}, {  
    'id': 'CJLjyLcMRA7Kh8neIr4g',  
    'alias': 'alexander-book-company-san-francisco',  
    'name': 'Alexander Book Company',  
    'phone': '(415) 821-7575',  
    'address': '102 Columbus Avenue, San Francisco, CA 94133',  
    'city': 'San Francisco',  
    'state': 'CA',  
    'zip_code': '94133',  
    'country': 'US',  
    'latitude': 37.7749, 'longitude': -122.421}], 'meta': {  
    'total': 1, 'page': 1, 'per_page': 1, 'next_page': null, 'prev_page': null}}
```

```
# Load businesses data to a data frame  
bookstores = pd.DataFrame(data["businesses"])  
print(bookstores.head(2))
```

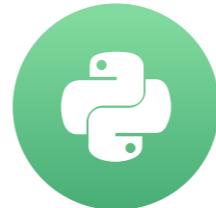
```
          alias      ...  
0  city-lights-bookstore-san-francisco  ...  https://www.yelp.com/biz/city-lights-bookstore-san-francisco  
1  alexander-book-company-san-francisco  ...  https://www.yelp.com/biz/alexander-book-company-san-francisco  
  
[2 rows x 16 columns]
```

# Let's practice!

**STREAMLINED DATA INGESTION WITH PANDAS**

# Working with nested JSONs

STREAMLINED DATA INGESTION WITH PANDAS



Amany Mahfouz  
Instructor

# Nested JSONs

- JSONs contain objects with attribute-value pairs
- A JSON is nested when the value itself is an object

## Response Body

```
{  
    "total": 8228,  
    "businesses": [  
        {  
            "rating": 4,  
            "price": "$",  
            "phone": "+14152520800",  
            "id": "E8RJkjfdcwgtyoPMjQ_olg",  
            "alias": "four-barrel-coffee-san-francisco",  
            "is_closed": false,  
            "categories": [  
                {  
                    "alias": "coffee",  
                    "title": "Coffee & Tea"  
                }  
            ],  
            "review_count": 1738,  
            "name": "Four Barrel Coffee",  
            "url": "https://www.yelp.com/biz/four-barrel-coffee-san-francisco",  
            "coordinates": {  
                "latitude": 37.7670169511878,  
                "longitude": -122.42184275  
            },  
            "image_url": "http://s3-media2.fl.yelpcdn.com/bphoto/MmgtASP3l_t4tPCL1iAsCg/o.jpg",  
            "location": {  
                "city": "San Francisco",  
                "country": "US",  
            }  
        }  
    ]  
}
```

## Response Body

```
{  
    "total": 8228,  
    "businesses": [  
        {  
            "rating": 4,  
            "price": "$",  
            "phone": "+14152520800",  
            "id": "E8RJkjfdcwgtyoPMjQ_olg",  
            "alias": "four-barrel-coffee-san-francisco",  
            "is_closed": false,  
            "categories": [  
                {  
                    "alias": "coffee",  
                    "title": "Coffee & Tea"  
                }  
            ],  
            "review_count": 1738,  
            "name": "Four Barrel Coffee",  
            "url": "https://www.yelp.com/biz/four-barrel-coffee-san-francisco",  
            "coordinates": {  
                "latitude": 37.7670169511878,  
                "longitude": -122.42184275  
            },  
            "image_url": "http://s3-media2.fl.yelpcdn.com/bphoto/MmgtaSP3l_t4tPCL1iAsCg/o.jpg",  
            "location": {  
                "city": "San Francisco",  
                "country": "US",  
            }  
        }  
    ]  
}
```

## Response Body

```
{  
    "total": 8228,  
    "businesses": [  
        {  
            "rating": 4,  
            "price": "$",  
            "phone": "+14152520800",  
            "id": "E8RJkjfdcwgtyoPMjQ_olg",  
            "alias": "four-barrel-coffee-san-francisco",  
            "is closed": false,  
            "categories": [  
                {  
                    "alias": "coffee",  
                    "title": "Coffee & Tea"  
                }  
            ],  
            "review_count": 1738,  
            "name": "Four Barrel Coffee",  
            "url": "https://www.yelp.com/biz/four-barrel-coffee-san-francisco",  
            "coordinates": {  
                "latitude": 37.7670169511878,  
                "longitude": -122.42184275  
            },  
            "image_url": "http://s3-media2.fl.yelpcdn.com/bphoto/MmgtASP3l_t4tPCL1iAsCg/o.jpg",  
            "location": {  
                "city": "San Francisco",  
                "country": "US",  
            }  
        }  
    ]  
}
```

## Response Body

```
{  
  "total": 8228,  
  "businesses": [  
    {  
      "rating": 4,  
      "price": "$",  
      "phone": "+14152520800",  
      "id": "E8RJkjfdcwgtyoPMjQ_olg",  
      "alias": "four-barrel-coffee-san-francisco",  
      "is_closed": false,  
      "categories": [  
        {  
          "alias": "coffee",  
          "title": "Coffee & Tea"  
        }  
      ],  
      "review_count": 1738,  
      "name": "Four Barrel Coffee",  
      "url": "https://www.yelp.com/biz/four-barrel-coffee-san-francisco",  
      "coordinates": {  
        "latitude": 37.7670169511878,  
        "longitude": -122.42184275  
      },  
      "image_url": "http://s3-media2.fl.yelpcdn.com/bphoto/MmgtaSP3l_t4tPCL1iAsCg/o.jpg",  
      "location": {  
        "city": "San Francisco",  
        "country": "US",  
      }  
    }  
  ]  
}
```

```
# Print columns containing nested data
print(bookstores[["categories", "coordinates", "location"]].head(3))
```

```
          categories \
0  [ {'alias': 'bookstores', 'title': 'Bookstores'}]
1  [ {'alias': 'bookstores', 'title': 'Bookstores'...}
2  [ {'alias': 'bookstores', 'title': 'Bookstores'}]

          coordinates \
0  {'latitude': 37.7975997924805, 'longitude': -1...
1  {'latitude': 37.7885846793652, 'longitude': -1...
2  {'latitude': 37.7589836120605, 'longitude': -1...

          location
0  {'address1': '261 Columbus Ave', 'address2': '...
1  {'address1': '50 2nd St', 'address2': '', 'add...
2  {'address1': '866 Valencia St', 'address2': ''...}
```

# pandas.io.json

- `pandas.io.json` submodule has tools for reading and writing JSON
  - Needs its own `import` statement
- `json_normalize()`
  - Takes a dictionary/list of dictionaries (like `pd.DataFrame()` does)
  - Returns a flattened data frame
  - Default flattened column name pattern: `attribute.nestedattribute`
  - Choose a different separator with the `sep` argument

# Loading Nested JSON Data

```
import pandas as pd
import requests

from pandas.io.json import json_normalize

# Set up headers, parameters, and API endpoint
api_url = "https://api.yelp.com/v3/businesses/search"
headers = {"Authorization": "Bearer {}".format(api_key)}
params = {"term": "bookstore",
          "location": "San Francisco"}

# Make the API call and extract the JSON data
response = requests.get(api_url,
                        headers=headers,
                        params=params)
data = response.json()
```

```
# Flatten data and load to data frame, with _ separators
bookstores = json_normalize(data["businesses"], sep="_")
print(list(bookstores))
```

```
['alias',
 'categories',
 'coordinates_latitude',
 'coordinates_longitude',
 ...
 'location_address1',
 'location_address2',
 'location_address3',
 'location_city',
 'location_country',
 'location_display_address',
 'location_state',
 'location_zip_code',
 ...
 'url']
```

# Deeply Nested Data

```
print(bookstores.categories.head())
```

```
0    [ {'alias': 'bookstores', 'title': 'Bookstores'}]
1    [ {'alias': 'bookstores', 'title': 'Bookstores'...]
2    [ {'alias': 'bookstores', 'title': 'Bookstores'}]
3    [ {'alias': 'bookstores', 'title': 'Bookstores'}]
4    [ {'alias': 'bookstores', 'title': 'Bookstores'...]
Name: categories, dtype: object
```

# Deeply Nested Data

- `json_normalize()`
  - `record_path` : string/list of string attributes to nested data
  - `meta` : list of other attributes to load to data frame
  - `meta_prefix` : string to prefix to meta column names

# Deeply Nested Data

```
# Flatten categories data, bring in business details
df = json_normalize(data["businesses"],
                     sep="_",
                     record_path="categories",
                     meta=[ "name",
                            "alias",
                            "rating",
                            [ "coordinates", "latitude" ],
                            [ "coordinates", "longitude" ]],
                     meta_prefix="biz_")
```

```
print(df.head(4))
```

```
      alias          title        biz_name \
0  bookstores    Bookstores  City Lights Bookstore
1  bookstores    Bookstores  Alexander Book Company
2 stationery  Cards & Stationery  Alexander Book Company
3  bookstores    Bookstores       Borderlands Books

      biz_alias  biz_rating  biz_coordinates_latitude \
0  city-lights-bookstore-san-francisco           4.5            37.797600
1  alexander-book-company-san-francisco           4.5            37.788585
2  alexander-book-company-san-francisco           4.5            37.788585
3  borderlands-books-san-francisco              5.0            37.758984

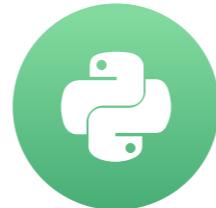
  biz_coordinates_longitude
0             -122.406578
1             -122.400631
2             -122.400631
3             -122.421638
```

# Let's practice!

**STREAMLINED DATA INGESTION WITH PANDAS**

# Combining multiple datasets

STREAMLINED DATA INGESTION WITH PANDAS



Amany Mahfouz

Instructor

# Appending

- Use case: adding rows from one data frame to another
- `append()`
  - Data frame method
  - Syntax: `df1.append(df2)`
  - Set `ignore_index` to `True` to renumber rows

# Appending

```
# Get first 20 bookstore results
params = {"term": "bookstore",
           "location": "San Francisco"}
first_results = requests.get(api_url,
                               headers=headers
                               params=params).json()

first_20_bookstores = json_normalize(first_results["businesses"],
                                      sep="_")

print(first_20_bookstores.shape)
```

```
(20, 24)
```

```
# Get the next 20 bookstores
params[ "offset" ] = 20
next_results = requests.get(api_url,
                            headers=headers
                            params=params).json()

next_20_bookstores = json_normalize(next_results[ "businesses" ],
                                     sep="_")

print(next_20_bookstores.shape)
```

```
(20, 24)
```

```
# Put bookstore datasets together, renumber rows  
bookstores = first_20_bookstores.append(next_20_bookstores,  
                                         ignore_index=True)  
  
print(bookstores.name)
```

```
0          City Lights Bookstore  
1          Alexander Book Company  
2          Borderlands Books  
3          Alley Cat Books  
4          Dog Eared Books  
...  
35         Forest Books  
36         San Francisco Center For The Book  
37         KingSpoke - Book Store  
38         Eastwind Books & Arts  
39         My Favorite  
  
Name: name, dtype: object
```

# Merging

- Use case: combining datasets to add related columns
- Datasets have key column(s) with common values
- `merge()` : pandas version of a SQL join

# Merging

- `merge()`
  - Both a `pandas` function and a data frame method
- `df.merge()` arguments
  - Second data frame to merge
  - Columns to merge on
    - `on` if names are the same in both data frames
    - `left_on` and `right_on` if key names differ
    - Key columns should be the same data type

```
call_counts.head()
```

```
   created_date  call_counts
0  01/01/2018        4597
1  01/02/2018        4362
2  01/03/2018        3045
3  01/04/2018        3374
4  01/05/2018        4333
```

```
weather.head()
```

```
      date    tmax    tmin
0 12/01/2017      52      42
1 12/02/2017      48      39
2 12/03/2017      48      42
3 12/04/2017      51      40
4 12/05/2017      61      50
```

# Merging

```
# Merge weather into call counts on date columns
merged = call_counts.merge(weather,
                           left_on="created_date",
                           right_on="date")

print(merged.head())
```

	created_date	call_counts	date	tmax	tmin
0	01/01/2018	4597	01/01/2018	19	7
1	01/02/2018	4362	01/02/2018	26	13
2	01/03/2018	3045	01/03/2018	30	16
3	01/04/2018	3374	01/04/2018	29	19
4	01/05/2018	4333	01/05/2018	19	9

# Merging

	created_date	call_counts	date	tmax	tmin
0	01/01/2018	4597	01/01/2018	19	7
1	01/02/2018	4362	01/02/2018	26	13
2	01/03/2018	3045	01/03/2018	30	16
3	01/04/2018	3374	01/04/2018	29	19
4	01/05/2018	4333	01/05/2018	19	9

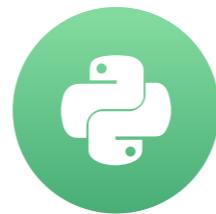
- Default `merge()` behavior: return only values that are in both datasets
- One record for each value match between data frames
  - Multiple matches = multiple records

# Let's practice!

**STREAMLINED DATA INGESTION WITH PANDAS**

# Wrap-up

STREAMLINED DATA INGESTION WITH PANDAS



Amany Mahfouz  
Instructor

# Recap

## Chapters 1 and 2

- `read_csv()` and `read_excel()`
- Setting data types, choosing data to load, handling missing data and errors



# Recap

## Chapter 3

- `read_sql()` and `sqlalchemy`
- SQL `SELECT`, `WHERE`, aggregate functions and joins



# Recap

## Chapter 4

- `read_json()`, `json_normalize()`, and `requests`
- Working with APIs and nested JSONs
- Appending and merging datasets

# Where to next?

- Learn more about data wrangling in `pandas`
  - Working with indexes, transforming values, dropping rows and columns
  - Reshaping data by merging, melting, pivoting
  - **Data Manipulation with Python Skill Track**

# Where to next?

- Explore a variety of analysis topics
  - Descriptive statistics (e.g. medians, means, standard deviation)
  - Inferential statistics (hypothesis testing, correlation, regression)
  - [Exploratory Data Analysis in Python](#)
  - [Introduction to Linear Modeling in Python](#)

# Where to next?

- Learn data visualization techniques
  - `seaborn` and `matplotlib` libraries
  - [Introduction to Data Visualization with Python](#)
  - [Introduction to Matplotlib](#)

# Where to next?

- Wrangle data as part of a fuller data science workflow
  - [Analyzing Police Activity with pandas](#)
  - [Analyzing US Census Data in Python](#)
  - [Analyzing Social Media Data in Python](#)

# Congrats!

**STREAMLINED DATA INGESTION WITH PANDAS**