Introduction to Flat Files

STREAMLINED DATA INGESTION WITH PANDAS

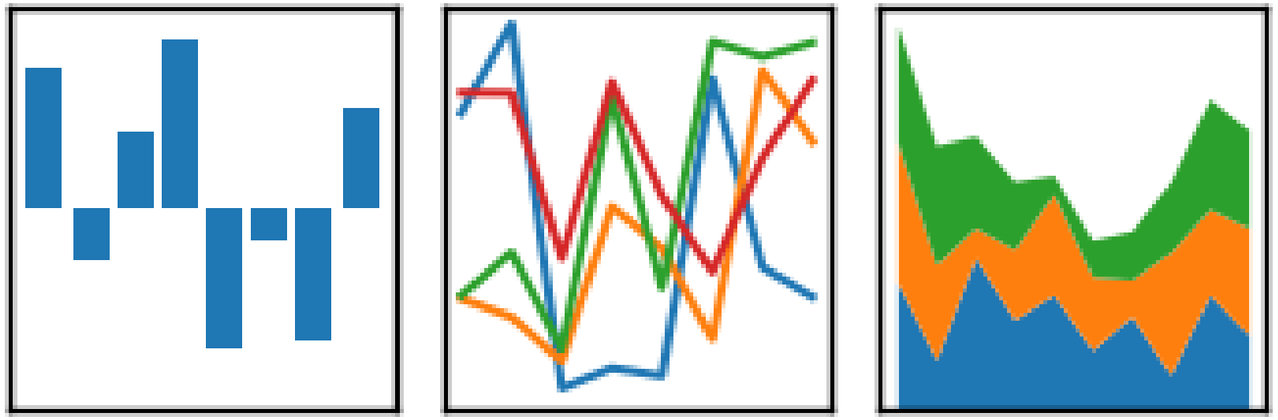
Amany Mahfouz
Instructor

DataCamp

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

Column Labels

Row Labels (Index)

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

DataCamp

Modifying flat file imports

STREAMLINED DATA INGESTION WITH PANDAS

Amany Mahfouz
Instructor

DataCamp

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

DataCamp

Handling errors and missing data

STREAMLINED DATA INGESTION WITH PANDAS

Amany Mahfouz
Instructor

DataCamp

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  A
0               1    AL      NaN         1   815440  ...       0   63420   51444  711580  183
1               1    AL      NaN         2   495830  ...       0   74090  110889  416090  111
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 unparseable 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

DataCamp