

# Data types

CLEANING DATA IN PYTHON



**Daniel Chen**  
Instructor

# Prepare and clean data

	name	sex	treatment a	treatment b
0	Daniel	male	-	42
1	John	male	12	31
2	Jane	female	24	27

# Data types

```
print(df.dtypes)
```

```
name          object
sex           object
treatment a   object
treatment b    int64
dtype: object
```

- There may be times we want to convert from one type to another
  - Numeric columns can be strings, or vice versa

# Converting data types

```
df['treatment b'] = df['treatment b'].astype(str)
df['sex'] = df['sex'].astype('category')
df.dtypes
```

```
name          object
sex           category
treatment a   object
treatment b   object
dtype: object
```

# Categorical data

- Converting categorical data to 'category' dtype:
  - Can make the DataFrame smaller in memory
  - Can make them be utilized by other Python libraries for analysis

# Cleaning data

- Numeric data loaded as a string

	name	sex	treatment a	treatment b
0	Daniel	male	-	42
1	John	male	12	31
2	Jane	female	24	27

# Cleaning data

- Numeric data loaded as a string

	name	sex	treatment a	treatment b
0	Daniel	male	-	42
1	John	male	12	31
2	Jane	female	24	27

# Cleaning bad data

```
df['treatment a'] = pd.to_numeric(df['treatment a'],  
                                  errors='coerce')
```

```
df.dtypes
```

```
name          object  
sex           category  
treatment a   float64  
treatment b   object  
dtype: object
```

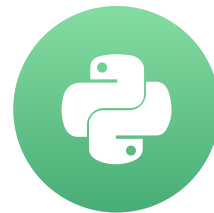


# Let's practice!

CLEANING DATA IN PYTHON

# Using regular expressions to clean strings

CLEANING DATA IN PYTHON



**Daniel Chen**  
Instructor

# String manipulation

- Much of data cleaning involves string manipulation
- Most of the world's data is unstructured text
- Also have to do string manipulation to make datasets consistent with one another

# Validate values

- 17
- \$17
- \$17.89
- \$17.895

# String manipulation

- Many built-in and external libraries
- `re` library for regular expressions
  - A formal way of specifying a pattern
  - Sequence of characters
- Pattern matching
  - Similar to globbing

# Example match

17		
\$17		
\$17.00		
\$17.89		
\$17.895		

# Example match

17		<code>\d</code>
\$17		
\$17.00		
\$17.89		
\$17.895		

# Example match

17	12345678901	<code>\d*</code>
\$17		
\$17.00		
\$17.89		
\$17.895		



# Example match

17	12345678901	<code>\d*</code>
\$17	\$12345678901	
\$17.00		
\$17.89		
\$17.895		

# Example match

17	12345678901	<code>\d*</code>
\$17	\$12345678901	<code>\\$</code>
\$17.00		
\$17.89		
\$17.895		

# Example match

17	12345678901	<code>\d*</code>
\$17	\$12345678901	<code>\\$\d*</code>
\$17.00		
\$17.89		
\$17.895		

# Example match

17	12345678901	<code>\d*</code>
\$17	\$12345678901	<code>\\$\d*</code>
\$17.00	\$12345678901.42	
\$17.89		
\$17.895		

# Example match

17	12345678901	<code>\d*</code>
\$17	\$12345678901	<code>\\$\d*</code>
\$17.00	\$12345678901.42	<code>\\$\d*</code>
\$17.89		
\$17.895		

# Example match

17	12345678901	<code>\d*</code>
\$17	\$12345678901	<code>\\$\d*</code>
\$17.00	\$12345678901.42	<code>\\$\d*\.\d*</code>
\$17.89		
\$17.895		

# Example match

17	12345678901	<code>\d*</code>
\$17	\$12345678901	<code>\\$\d*</code>
\$17.00	\$12345678901.42	<code>\\$\d*\.\d*</code>
\$17.89	\$12345678901.24	
\$17.895		

# Example match

17	12345678901	<code>\d*</code>
\$17	\$12345678901	<code>\\$\d*</code>
\$17.00	\$12345678901.42	<code>\\$\d*\.\d*</code>
\$17.89	\$12345678901.24	<code>\\$\d*\.\d{2}</code>
\$17.895		



# Example match

17	12345678901	<code>\d*</code>
\$17	\$12345678901	<code>\\$\d*</code>
\$17.00	\$12345678901.42	<code>\\$\d*\.\d*</code>
\$17.89	\$12345678901.24	<code>\\$\d*\.\d{2}</code>
\$17.895		

# Example match

17	12345678901	<code>\d*</code>
\$17	\$12345678901	<code>\\$\d*</code>
\$17.00	\$12345678901.42	<code>\\$\d*\.\d*</code>
\$17.89	\$12345678901.24	<code>\\$\d*\.\d{2}</code>
\$17.895	\$12345678901.999	

# Example match

17	12345678901	<code>\d*</code>
\$17	\$12345678901	<code>\\$\d*</code>
\$17.00	\$12345678901.42	<code>\\$\d*\.\d*</code>
\$17.89	\$12345678901.24	<code>\\$\d*\.\d{2}</code>
\$17.895	\$12345678901.999	<code>^\\$\d*\.\d{2}\$</code>

# Example match

17	12345678901	<code>\d*</code>
\$17	\$12345678901	<code>\\$\d*</code>
\$17.00	\$12345678901.42	<code>\\$\d*\.\d*</code>
\$17.89	\$12345678901.24	<code>\\$\d*\.\d{2}</code>
\$17.895	\$12345678901.999	<code>^\\$\d*\.\d{2}\$</code>

# Example match

17	12345678901	<code>\d*</code>
\$17	\$12345678901	<code>\\$\d*</code>
\$17.00	\$12345678901.42	<code>\\$\d*\.\d*</code>
\$17.89	\$12345678901.24	<code>\\$\d*\.\d{2}</code>
<del>\$17.895</del>	<del>\$12345678901.999</del>	<code><span>^</span>\\$\d*\.\d{2}<span>\$</span></code>

- "I have 17.89 USD"

# Using regular expressions

- Compile the pattern
- Use the compiled pattern to match values
- This lets us use the pattern over and over again
- Useful since we want to match values down a column of values

# Using regular expressions

```
import re

pattern = re.compile('\$\d*\.\d{2}')

result = pattern.match('$17.89')

bool(result)
```

True

# Let's practice!

CLEANING DATA IN PYTHON



# Using functions to clean data

CLEANING DATA IN PYTHON



**Daniel Chen**  
Instructor

# Complex cleaning

- Cleaning step requires multiple steps
  - Extract number from string
  - Perform transformation on extracted number
- Python function

# apply()

```
print(df)
```

```
      treatment a  treatment b
Daniel          18          42
John            12          31
Jane            24          27
```

```
df.apply(np.mean, axis=0)
```

```
treatment a    18.000000
treatment b    33.333333
dtype: float64
```

# apply()

```
print(df)
```

	treatment a	treatment b
Daniel	18	42
John	12	31
Jane	24	27

```
df.apply(np.mean, axis=1)
```

Daniel	30.0
John	21.5
Jane	25.5

dtype: float64

# Applying functions

	Job #	Doc #	Borough	Initial Cost	Total Est. Fee
0	121577873	2	MANHATTAN	\$75000.00	\$986.00
1	520129502	1	STATEN ISLAND	\$0.00	\$1144.00
2	121601560	1	MANHATTAN	\$30000.00	\$522.50
3	121601203	1	MANHATTAN	\$1500.00	\$225.00
4	121601338	1	MANHATTAN	\$19500.00	\$389.50

# Write the regular expression

```
import re
from numpy import NaN

pattern = re.compile('^\\$\\d*\\.\\d{2}$')
```

# Writing a function

example.py

```
def my_function(input1, input2):  
  
    # Function Body  
  
    return value
```

# Write the function

diff\_money.py

```
def diff_money(row, pattern):
    icost = row['Initial Cost']
    tef = row['Total Est. Fee']

    if bool(pattern.match(icost)) and bool(pattern.match(tef)):
        icost = icost.replace("$", "")
        tef = tef.replace("$", "")

        icost = float(icost)
        tef = float(tef)

        return icost - tef
    else:
        return(NaN)
```



# Write the function

```
df_subset['diff'] = df_subset.apply(diff_money,  
                                     axis=1,  
                                     pattern=pattern)
```

```
print(df_subset.head())
```

	Job #	Doc #	Borough	Initial Cost	Total	Est. Fee	
0	121577873	2	MANHATTAN	\$75000.00		\$986.00	7401
1	520129502	1	STATEN ISLAND	\$0.00		\$1144.00	-114
2	121601560	1	MANHATTAN	\$30000.00		\$522.50	2947
3	121601203	1	MANHATTAN	\$1500.00		\$225.00	127
4	121601338	1	MANHATTAN	\$19500.00		\$389.50	1911

# Let's practice!

CLEANING DATA IN PYTHON

# Duplicate and missing data

CLEANING DATA IN PYTHON



**Daniel Chen**  
Instructor

# Duplicate data

- Can skew results
- `.drop_duplicates()` method

	name	sex	treatment a	treatment b
0	Daniel	male	-	42
1	John	male	12	31
2	Jane	female	24	27
3	Daniel	male	-	42

# Duplicate data

- Can skew results
- `.drop_duplicates()` method

	name	sex	treatment a	treatment b
0	Daniel	male	-	42
1	John	male	12	31
2	Jane	female	24	27
3	Daniel	male	-	42

# Drop duplicates

```
df = df.drop_duplicates()  
print(df)
```

	name	sex	treatment a	treatment b
0	Daniel	male	-	42
1	John	male	12	31
2	Jane	female	24	27

# Missing data

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2.0
1	NaN	1.66	Male	No	Sun	Dinner	3.0
2	21.01	3.50	Male	No	Sun	Dinner	3.0
3	23.68	NaN	Male	No	Sun	Dinner	2.0
4	24.59	3.61	NaN	NaN	Sun	NaN	4.0

# Missing data

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2.0
1	NaN	1.66	Male	No	Sun	Dinner	3.0
2	21.01	3.50	Male	No	Sun	Dinner	3.0
3	23.68	NaN	Male	No	Sun	Dinner	2.0
4	24.59	3.61	NaN	NaN	Sun	NaN	4.0

- Leave as-is
- Drop them
- Fill missing value



# Count missing values

```
tips_nan.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 244 entries, 0 to 243
Data columns (total 7 columns):
total_bill    202 non-null float64
tip           220 non-null float64
sex           234 non-null object
smoker        229 non-null object
day           243 non-null object
time          227 non-null object
size          231 non-null float64
dtypes: float64(3), object(4)
memory usage: 13.4+ KB
None
```

# Drop missing values

```
tips_dropped = tips_nan.dropna()  
tips_dropped.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
Int64Index: 147 entries, 0 to 243  
Data columns (total 7 columns):  
total_bill    147 non-null float64  
tip           147 non-null float64  
sex           147 non-null object  
smoker        147 non-null object  
day           147 non-null object  
time          147 non-null object  
size          147 non-null float64  
dtypes: float64(3), object(4)  
memory usage: 9.2+ KB
```

# Fill missing values with `.fillna()`

- Fill with provided value
- Use a summary statistic

```
tips_nan['sex'] = tips_nan['sex'].fillna('missing')

tips_nan[['total_bill', 'size']] = tips_nan[['total_bill',
                                              'size']].fillna(0)

tips_nan.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 244 entries, 0 to 243
Data columns (total 7 columns):
total_bill    244 non-null float64
tip           220 non-null float64
sex           244 non-null object
smoker        229 non-null object
day           243 non-null object
time          227 non-null object
size          244 non-null float64
dtypes: float64(3), object(4)
memory usage: 13.4+ KB
```

# Fill missing values with a test statistic

- Careful when using test statistics to fill
- Have to make sure the value you are filling in makes sense
- Median is a better statistic in the presence of outliers

```
mean_value = tips_nan['tip'].mean()  
print(mean_value)
```

```
2.9646818181819
```

```
tips_nan['tip'] = tips_nan['tip'].fillna(mean_value)  
tips_nan.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 244 entries, 0 to 243  
Data columns (total 7 columns):  
total_bill    244 non-null float64  
tip           244 non-null float64  
sex           244 non-null object  
smoker        229 non-null object  
day           243 non-null object  
time          227 non-null object  
size          244 non-null float64  
dtypes: float64(3), object(4)  
memory usage: 13.4+ KB
```

# Let's practice!

CLEANING DATA IN PYTHON

# Testing with asserts

CLEANING DATA IN PYTHON



**Daniel Chen**  
Instructor



# Assert statements

- Programmatically vs visually checking
- If we drop or fill NaNs, we expect 0 missing values
- We can write an assert statement to verify this
- We can detect early warnings and errors
- This gives us confidence that our code is running correctly

# Asserts

```
assert 1 == 1
```

```
assert 1 == 2
```

```
-----  
AssertionError                                Traceback (most recent call last)  
<ipython-input-65-a810b3a4aded> in <module>()  
----> 1 assert 1 == 2  
AssertionError:
```

# Google stock data

	Date	Open	High	Low	Close	Volume	Adj Close
0	2017-02-09	831.729980	NaN	826.500000	830.059998	1192000.0	NaN
1	2017-02-08	830.530029	834.250000	825.109985	829.880005	1300600.0	829.880005
2	2017-02-07	NaN	NaN	823.289978	NaN	1664800.0	NaN
3	2017-02-06	820.919983	822.390015	NaN	821.619995	NaN	821.619995
4	2017-02-03	NaN	826.130005	819.349976	820.130005	1524400.0	820.130005

# Test column

```
assert google.Close.notnull().all()
```

```
-----  
AssertionError                                Traceback (most recent call last)  
<ipython-input-49-eec77130a77f> in <module>()  
----> 1 assert google.Close.notnull().all()  
AssertionError:
```

# Test column

```
google_0 = google.fillna(value=0)  
assert google_0.Close.notnull().all()
```

# Let's practice!

CLEANING DATA IN PYTHON