5. Functions, calculated columns and cleaning data

February 6, 2021

1 5. Functions, calculated columns and cleaning data

In this notebook, we'll cover writing custom functions, adding calculated columns and a few data-cleaning strategies.

First, import pandas:

[82]:
```
import pandas as pd
```

1.0.1 Functions

If you find yourself doing the same thing over and over again in your code, it might be time to write a function.

Functions are blocks of reusable code – little boxes that (usually) take inputs and (usually) return outputs. In Excel, `=SUM()` is a function. `print()` is one of Python's built-in function.

You can also *define your own functions*. This can save you some typing, and it will help separate your code into logical, easy-to-read pieces.

Syntax Functions start with the `def` keyword – short for *define*, because you're defining a function – then the name of the function, then parentheses (sometimes with the names of any `arguments` your function requires inside the parentheses) and then a colon. The function's code sits inside an indented block immediately below that line. In most cases, a function will `return` a value at the end.

Here is a function that takes a number and returns that number multiplied by 10:

[83]:
```
def times_ten(number):
    return number * 10
```

The `number` argument is just a placeholder for whatever value is handed the function as an input. We could have called that argument `banana` and things would be just fine (though it would be confusing for people reading your code).

Calling a function By itself, a function doesn't do anything. We have built a tiny machine to multiply a number by 10. But it's just sitting on the workshop bench, waiting for us to use it.

Let's use it!

[84]:
```
times_ten(2)
```

[84]: 20

Function arguments Functions can accept *positional* arguments or *keyword* arguments.

If your function uses *positional* arguments, the order in which you pass arguments to the function matters. Here is a function that prints out a message based on its input: a person's name and their hometown.

(This function uses something called an "f-string" to format the result. For more information on text formatting, see this notebook.)

[85]:
```
def greet(name, hometown):
    return f'Hello, {name} from {hometown}!'
```

Now let's call it.

[86]:
```
greet('Cody', 'Pavillion, WY')
```

[86]: 'Hello, Cody from Pavillion, WY!'

If we change the order of the arguments, we get nonsense.

[87]:
```
greet('Pavillion, WY', 'Cody')
```

[87]: 'Hello, Pavillion, WY from Cody!'

Using *keyword* arguments requires us to specify what value belongs to what argument, and it allows us to set a default value for the argument – values that the function will use if you fail to pass any arguments when you call it. We could rewrite our function like this:

[88]:
```
def greet(name='Cody', hometown='Pavillion, WY'):
    return f'Hello, {name} from {hometown}!'
```

And now it doesn't matter what order we pass in the arguments, because we're defining the keyword that they belong to:

[89]:
```
greet(hometown='Pittsburgh, PA', name='Jacob')
```

[89]: 'Hello, Jacob from Pittsburgh, PA!'

What happens if we call the `greet()` function without any arguments at all, now? It'll use the default arguments.

[90]:
```
greet()
```

[90]:
```
'Hello, Cody from Pavillion, WY!'
```

1.0.2 Try it yourself

Use the code blocks below to experiment with functions.

[ ]:
```
```

[ ]:
```
```

[ ]:
```
```

[ ]:
```
```

[ ]:
```
```

[ ]:
```
```

1.0.3 Adding new or calculated columns

In a spreadsheet program, if you want to add a new column of data – maybe a copy of an existing column for cleaning – you could just reference the original column in a formula. If you wanted to calculate a new column of values based on other values in each row, you might write a formula and fill it down. In SQL, you might run an `ALTER TABLE/UPDATE/SET` routine to handle this process.

In pandas, adding a new column is similar to adding a new record to a Python dictionary. Let's load in the CT overdose data to take a look at how this works.

[91]:
```
df_ct = pd.read_excel('../data/CT_Overdoses_2012-2016.xlsx',␣
↪sheet_name='Accidental_Drug_Related_Deaths_')
```

[92]:
```
df_ct.head()
```

[92]:
```
  CaseNumber        Date     Sex   Race   Age Residence City Residence State  \
0   13-16336 2013-11-09  Female  White  53.0         GROTON             NaN
1   12-18447 2012-12-29    Male  White  30.0        WOLCOTT             NaN
2    14-2758 2014-02-18    Male  White  43.0        ENFIELD             NaN
3   14-13497 2014-09-07  Female  White  24.0    WALLINGFORD             NaN
4   13-14421 2013-10-04  Female  White  26.0     WEST HAVEN             NaN

  Residence County   Death City Death State  … Benzodiazepine Methadone  \
0       NEW LONDON       GROTON         NaN  …              Y       NaN
1        NEW HAVEN    WATERBURY         NaN  …            NaN       NaN
2              NaN      ENFIELD         NaN  …              Y       NaN
```

```
3               NaN  WALLINGFORD          NaN  …             NaN        NaN
4         NEW HAVEN   WEST HAVEN          NaN  …             NaN        NaN

  Amphet Tramad Morphine (not heroin) Other Any Opioid MannerofDeath  \
0    NaN    NaN                   NaN   NaN        NaN      Accident
1    NaN    NaN                   NaN   NaN        NaN      Accident
2    NaN    NaN                   NaN   NaN        NaN      Accident
3    NaN    NaN                   NaN   NaN        NaN      Accident
4    NaN    NaN                   NaN   NaN        NaN      Accident

  AmendedMannerofDeath                  DeathLoc
0                  NaN   (41.343693, -72.07877)
1                  NaN  (41.554261, -73.043069)
2                  NaN  (41.976501, -72.591985)
3                  NaN  (41.454408, -72.818414)
4                  NaN  (41.272336, -72.949817)

[5 rows x 32 columns]
```

Let's say we eventually wanted to do some analysis based on the `Death City` column, but maybe first we need to clean it up. You always want to leave your original data intact, so first step would be to create a copy of the `Death City` column:

[93]:
```
df_ct['death_city_clean'] = df_ct['Death City']
```

[94]:
```
df_ct.columns
```

[94]:
```
Index(['CaseNumber', 'Date', 'Sex', 'Race', 'Age', 'Residence City',
       'Residence State', 'Residence County', 'Death City', 'Death State',
       'Death County', 'Location', 'DescriptionofInjury', 'InjuryPlace',
       'ImmediateCauseA', 'Heroin', 'Cocaine', 'Fentanyl', 'Oxycodone',
       'Oxymorphone', 'EtOH', 'Hydro-codeine', 'Benzodiazepine', 'Methadone',
       'Amphet', 'Tramad', 'Morphine (not heroin)', 'Other', 'Any Opioid',
       'MannerofDeath', 'AmendedMannerofDeath', 'DeathLoc',
       'death_city_clean'],
      dtype='object')
```

… and then you could work through some cleaning steps (more on that below).

To create a calculated column, you would first define a function to process a row of data in your dataframe, then *apply* that function to your dataframe using a pandas method called `apply()`.

The values in several columns in our dataframe list whether a particular drug was found by the medical examiner examining the body, with `Y` meaning it was found and the default pandas null value (`NaN`) if not. Let's add a new column, `drugs_involved_total`, that totals up the number of `Y`s in each row for the columns listing individual drugs:

```
'Heroin',
'Cocaine',
```

```
'Fentanyl',
'Oxycodone',
'Oxymorphone',
'EtOH',
'Hydro-codeine',
'Benzodiazepine',
'Methadone',
'Amphet',
'Tramad',
'Morphine (not heroin)',
'Other'
```

Now we can write a function that accepts as its one position argument a row of data in the dataframe, checks the values in each of our target columns – keeping track of the Ys – and then returns the total.

[95]:
```
# the name of the function is more or less arbitrary
# `row` also an arbitrary argument name but helps us think about what's␣
↪happening
def get_total_drugs(row):

    # start a counter for how many drugs were present
    total_drugs = 0

    # list the names of the columns to check
    drug_columns = [
        'Heroin',
        'Cocaine',
        'Fentanyl',
        'Oxycodone',
        'Oxymorphone',
        'EtOH',
        'Hydro-codeine',
        'Benzodiazepine',
        'Methadone',
        'Amphet',
        'Tramad',
        'Morphine (not heroin)',
        'Other'
    ]

    # loop over the column list
    for col in drug_columns:

        # grab the value for that column in this row
        value = row[col]

        # if the value is `Y` ...
```

```python
        if value == 'Y':

            # ... increment the counter
            # (this is just a shortcut for `total_drugs = total_drugs + 1`)
            total_drugs += 1

    # once the loop completes, return the counter
    return total_drugs
```

Once you have a function defined, you can apply() it to the dataframe:

[96]:
```python
df_ct['drugs_involved_total'] = df_ct.apply(get_total_drugs, axis=1)
```

[97]:
```python
df_ct.columns
```

[97]:
```
Index(['CaseNumber', 'Date', 'Sex', 'Race', 'Age', 'Residence City',
       'Residence State', 'Residence County', 'Death City', 'Death State',
       'Death County', 'Location', 'DescriptionofInjury', 'InjuryPlace',
       'ImmediateCauseA', 'Heroin', 'Cocaine', 'Fentanyl', 'Oxycodone',
       'Oxymorphone', 'EtOH', 'Hydro-codeine', 'Benzodiazepine', 'Methadone',
       'Amphet', 'Tramad', 'Morphine (not heroin)', 'Other', 'Any Opioid',
       'MannerofDeath', 'AmendedMannerofDeath', 'DeathLoc', 'death_city_clean',
       'drugs_involved_total'],
      dtype='object')
```

[98]:
```python
df_ct.drugs_involved_total.unique()
```

[98]:
```
array([3, 2, 1, 4, 0, 5, 6])
```

[99]:
```python
df_ct.head()
```

[99]:
```
  CaseNumber       Date     Sex   Race   Age Residence City Residence State  \
0   13-16336 2013-11-09  Female  White  53.0         GROTON             NaN
1   12-18447 2012-12-29    Male  White  30.0        WOLCOTT             NaN
2    14-2758 2014-02-18    Male  White  43.0        ENFIELD             NaN
3   14-13497 2014-09-07  Female  White  24.0    WALLINGFORD             NaN
4   13-14421 2013-10-04  Female  White  26.0     WEST HAVEN             NaN

  Residence County   Death City Death State  … Amphet Tramad  \
0       NEW LONDON       GROTON         NaN  …    NaN    NaN
1        NEW HAVEN    WATERBURY         NaN  …    NaN    NaN
2              NaN      ENFIELD         NaN  …    NaN    NaN
3              NaN  WALLINGFORD         NaN  …    NaN    NaN
4        NEW HAVEN   WEST HAVEN         NaN  …    NaN    NaN

  Morphine (not heroin) Other Any Opioid MannerofDeath AmendedMannerofDeath  \
0                   NaN   NaN        NaN      Accident                  NaN
```

```
1                    NaN   NaN         NaN      Accident                     NaN
2                    NaN   NaN         NaN      Accident                     NaN
3                    NaN   NaN         NaN      Accident                     NaN
4                    NaN   NaN         NaN      Accident                     NaN

                 DeathLoc death_city_clean drugs_involved_total
0   (41.343693, -72.07877)           GROTON                    3
1  (41.554261, -73.043069)        WATERBURY                    2
2  (41.976501, -72.591985)          ENFIELD                    2
3  (41.454408, -72.818414)      WALLINGFORD                    2
4  (41.272336, -72.949817)       WEST HAVEN                    1

[5 rows x 34 columns]
```

For more information on applying functions to a pandas data frame, check out this notebook.

1.0.4 Cleaning data

For cleaning jobs of any size, specialized tools like OpenRefine are still your best bet – a typical workflow is to clean your data in OpenRefine, export as a CSV, then load into pandas.

But in many cases, you can use some of pandas' built-in tools to whip your data into shape. This is especially useful for data processing tasks that you plan to repeat as the data are updated.

In Excel, running a pivot table (with counts) for each column will show you misspellings, external white space, inconsistent casing and other problems that keep your data from grouping correctly.

In SQL, you might do the same thing with The Golden Query™:

```sql
SELECT column, COUNT(*)
FROM table
GROUP BY column
ORDER BY 2 DESC
```

To do the equivalent operation in pandas, you can just call the `value_counts()` method on a column. Let's look at some Congressional junkets data as an example:

[100]:
```python
df_junkets = pd.read_csv('../data/congress_junkets.csv')
```

[101]:
```python
df_junkets.head()
```

[101]:
```
       DocID            FilerName      MemberName State  District  Year  \
0  500005076        Bobby Cornett    Franks, Trent    AZ       8.0  2011
1  500005077  Michael Strittmatter    Franks, Trent    AZ       8.0  2011
2  500005081        Diane Rinaldo     Rogers, Mike    AL       3.0  2011
3  500005082      Kenneth DeGraff  Doyle, Michael    PA      14.0  2011
4  500005083    Michael Ryan Clough     Lofgren, Zoe    CA      19.0  2011

     Destination FilingType DepartureDate ReturnDate  \
```

```
0  Las Vegas, NV   Original      1/7/2011   1/9/2011
1  Las Vegas, NV   Original      1/7/2011   1/9/2011
2  Las Vegas, NV   Original      1/6/2011   1/8/2011
3  Las Vegas, NV   Original      1/6/2011   1/9/2011
4  Las Vegas, NV   Original      1/6/2011   1/8/2011

                       TravelSponsor
0  Consumer Electronics Association
1         CEA Leaders in Technology
2  Consumer Electronics Association
3  Consumer Electronics Association
4  Consumer Electronics Association
```

Let's run `value_counts()` on the *Destination* colummn:

[102]:
```
df_junkets['Destination'].value_counts()
```

[102]:
```
Baltimore, MD                  827
Hot Springs, VA                753
Tel Aviv, Israel               651
New York, NY                   635
Philadelphia, PA               487
                              …
Hot Springs, Va                  1
Rhinebeck, NY                    1
Oneonta, NY                      1
Hartford, Connecticut            1
Dubai, United Arab Emirates      1
Name: Destination, Length: 838, dtype: int64
```

The default sort order is by count descending, but it can also be helpful in finding typos to sort by the name – the "index" of what `value_counts()` returns. To do that, tack on `sort_index()`:

[103]:
```
df_junkets['Destination'].value_counts().sort_index()
```

[103]:
```
Abidjan, Cote d'Ivoire           3
Abu Dhabi, United Arab Emirate   3
Abuja, Nigeria                   4
Accra                            1
Accra, Ghana                    10
                                ..
Zagreb, Croatia                  1
Zanzibar, Tanzania               6
Zhytomyr, Ukraine                1
Zugdidi, Georgia                 1
Zurich, Switzerland              3
Name: Destination, Length: 838, dtype: int64
```

… and now we start to see some common data problems in our 838 unique destinations – whitespace, inconsistent values for the same thing ("Accra" and "Accra, Ghana") – and can start fixing them.

1.0.5 Fixing whitespace, casing and other "string" problems

If part of our analysis hinged on having a pristine "Destination" column, then we've got some work ahead of us. First thing I'd do: Strip whitespace and upcase the text.

You can do a lot of basic cleanup like this by applying Python's built-in string methods to the `str` attribute of a column.

To start with, let's create a new column, `destination_clean`, with a stripped/uppercase version of the destination data.

Note: Outside of pandas, you can use "method chaining" to apply multiple transformations to a string, like this: `'    My String'.upper().strip()`.

When you're chaining string methods on the `str` attribute of a pandas column series, though, it doesn't work like that – you have to call `str` after each method call. In other words:

```
# this will throw an error
junkets['destination_clean'] = junkets['Destination'].str.upper().strip()

# this will work
junkets['destination_clean'] = junkets['Destination'].str.upper().str.strip()
```

[104]:
```
df_junkets['destination_clean'] = df_junkets['Destination'].str.upper().str.
↪strip()
```

[105]:
```
df_junkets.head()
```

[105]:
```
       DocID             FilerName      MemberName State  District  Year  \
0  500005076          Bobby Cornett   Franks, Trent    AZ       8.0  2011
1  500005077  Michael Strittmatter   Franks, Trent    AZ       8.0  2011
2  500005081         Diane Rinaldo    Rogers, Mike    AL       3.0  2011
3  500005082        Kenneth DeGraff  Doyle, Michael    PA      14.0  2011
4  500005083     Michael Ryan Clough    Lofgren, Zoe    CA      19.0  2011

     Destination FilingType DepartureDate ReturnDate  \
0  Las Vegas, NV   Original      1/7/2011   1/9/2011
1  Las Vegas, NV   Original      1/7/2011   1/9/2011
2  Las Vegas, NV   Original      1/6/2011   1/8/2011
3  Las Vegas, NV   Original      1/6/2011   1/9/2011
4  Las Vegas, NV   Original      1/6/2011   1/8/2011

                   TravelSponsor destination_clean
0  Consumer Electronics Association     LAS VEGAS, NV
1         CEA Leaders in Technology     LAS VEGAS, NV
2  Consumer Electronics Association     LAS VEGAS, NV
```

```
3  Consumer Electronics Association     LAS VEGAS, NV
4  Consumer Electronics Association     LAS VEGAS, NV
```

Now let's run `value_counts()` again to see if that helped at all.

[106]:
```
df_junkets['destination_clean'].value_counts().sort_index()
```

[106]:
```
ABIDJAN, COTE D'IVOIRE            3
ABU DHABI, UNITED ARAB EMIRATE    3
ABUJA, NIGERIA                    4
ACCRA                             1
ACCRA, GHANA                     10
                                 ..
ZAGREB, CROATIA                   1
ZANZIBAR, TANZANIA                6
ZHYTOMYR, UKRAINE                 1
ZUGDIDI, GEORGIA                  1
ZURICH, SWITZERLAND               3
Name: destination_clean, Length: 831, dtype: int64
```

That eliminated a handful of problems. Now comes the tedious work of identifying entries to find and replace.

1.0.6 Bulk-replacing values with other values

If we were at this point in Excel, we'd scroll through the list of unique names and start making notes of what we need to change. Same story here.

Let's loop over a sorted list of `unique()` destinations and `print()` each one.

[107]:
```
for destination in sorted(df_junkets.destination_clean.unique()):
    print(destination)
```

```
ABIDJAN, COTE D'IVOIRE
ABU DHABI, UNITED ARAB EMIRATE
ABUJA, NIGERIA
ACCRA
ACCRA, GHANA
ADDIS ABABA, ETHIOPIA
ADDIS, ETHIOPIA
ADELAIDE, AUSTRALIA
AIKEN, SC
AKRON, OH
ALBANY, NY
ALBERTA, CANADA
ALBUQUERQUE, NM
ALGIERS, ALGERIA
ALLENTOWN, PA
```

AMELIA ISLAND, FL
AMES, IA
AMMAN, JORDAN
AMSTERDAM, NETHERLANDS
ANAHEIM, CA
ANATALYA, TURKEY
ANCHORAGE, AK
ANDECHS, GERMANY
ANKARA, ISRAEL
ANKARA, TURKEY
ANKENY, IA
ANKEY, IA
ANN ARBOR, MI
ANNAPOLIS, MD
ANOMABO, GHANA
ANTALYA, TURKEY
ANTIGUA, GUATEMALA
ARAUCA, COLOMBIA
ARLINGTON, VA
ARUSHA, TANZANIA
ASHEVILLE, NC
ASPEN, CO
ATLANTA, GA
ATLANTA, GEORGIA
ATLANTA,GA
AUGUSTA, GA
AUSTIN, TEXAS
AUSTIN, TX
AVENTURA, FL
AVILA BEACH, CA
AVOCA, IA
AWASSA, ETHIOPIA
BAKU, AZERBAIJAN
BAKU, AZERBIJAN
BAKU, REPUBLIC OF AZERBAIJAN
BALI, INDONESIA
BALTIMORE, DC
BALTIMORE, MD
BALTIMROE, MD
BANFF, CANADA
BANGALORE, INDIA
BANJA LUKA, BOSNIA-HERZEGOVINA
BARCELONA, SPAIN
BARTLESVILLE, OK
BATON ROUGE, LA
BATTLE CREEK, MI
BEARDSTOWN, IL
BEDFORD SPRINGS, PA

BEDFORD, PA
BEIJING, CHINA
BEIRA, MOZAMBIQUE
BEIRUT, LEBANON
BEJING, CHINA
BELFAST, NORTHERN IRELAND
BELGRADE, SERBIA
BENTIU, SOUTH SUDAN
BERKELEY, CA
BERLIN BERMANY
BERLIN, GERMANY
BERLING, GERMANY
BETHLEHEM
BETHLEHEM, PA
BETHLEHEM, PALESTINIAN TERRITO
BIRMINGHAM, AL
BIRMINGHAM, ENGLAND
BISHKEK, KYRGYZSTAN
BISMARCK, ND
BISMARK, ND
BLAIRSTOWN, IA
BLANTYRE, MALAWI
BLOOMINGTON, IL
BOCA RATON, FL
BOGATA, COLUMBIA
BOGOTA, COLOMBIA
BOGOTA, COLUMBIA
BOLOGNA, ITALY
BOONE, IA
BOSTON, MA
BOTOTA, COLUMBIA
BOULDER, CO
BRETTON WOODS, NH
BRIDGEPORT, CT
BRIESEN, GERMANY
BROOKLYN, NY
BRUGES, BELGIUM
BRUNSWICK, GA
BRUSSELLS, BELGIUM
BRUSSELS, BELGIUM
BRUSSELS, BELGUIM
BUCHAREST, ROMANIA
BUDAPEST, HUNGARY
BUDVA, MONTENEGRO
BUENAVENTURA, COLOMBIA
BUENAVENTURA, COLUMBIA
BUENAVENTURE, COLUMBIA
BUENOS AIRES, ARGENTINA

BUJUMBURA, BURUNDI
BUKAVU, DEMOCRATIC REPUBLIC OF
BURAS, LA
BURBANK, CA
BURLINGTON, IA
BURLINGTON, VT
BUSHKILL, PA
CACERES, COLOMBIA
CAIRO, EGYPT
CALEBRA, NIGERIA
CALI, COLOMBIA
CALI, COLUMBIA
CAMBRIDGE, MA
CAMBRIDGE, MD
CANAKKALE, TURKEY
CANNAKALE, TURKEY
CANNAKKALE, TURKEY
CANONSBURG, PA
CANTON, OH
CAPADOCIA, TURKEY
CAPE TOWN, SOUTH AFRICA
CAPPADOCIA, TURKEY
CAPRI, ITALY
CAREGENA, COLUMBIA
CARTAGENA, COLOMBIA
CARTAGENA, COLUMBIA
CARTEGENA, COLUMBIA
CASABLANCA, MOROCCO
CAUCASIA, COLOMBIA
CAUX, SWITZERLAND
CAYAR, SENEGAL
CEDAR KEY, FL
CEDAR RAPIDS, IA
CESME, TURKEY
CHALMETTE, LA
CHAMPAIGN, IL
CHANTILLY, VA
CHARLESTON, SC
CHARLESTON, WV
CHARLOTTE, NC
CHARLOTTE, SC
CHARLOTTESVILLE, VA
CHATTANOOGA, TN
CHAUTAUQUA, NY
CHENGDU, CHINA
CHERITON, VA
CHERLTON, VA
CHESAPEAKE BAY, MD

CHESAPEAKE, MD
CHICAGO, IL
CHICHICASTENANGO, GUATEMALA
CHIPATA, ZAMBIA
CINCINNATI, OH
CIUDAD JUAREZ, MEXICO
CLEARWATER BEACH, FL
CLEVELAND, OH
CLEVELAND, OHIO
CODY, WY
COLOMBO, SRI LANKA
COLORADO SPRINGS, CO
COLUMBIA, MD
COLUMBIA, MO
COLUMBUS, OH
COPENHAGEN, DENMARK
CORPUS CHRISTI, TX
COUNCIL BLUFFS, IA
CULPEPER, VA
CUSCO, PERU
CUZCO, PERU
DAKAR, SENAGAL
DAKAR, SENEGAL
DALLAS, TX
DALLAS/FORT WORTH, TX
DAMASCUS, SYRIA
DAR ES SALAAM, TANZANIA
DAR-ES-SALAAM, TANZANIA
DAVENPORT, IA
DAYTONA BEACH, FL
DEADHORSE, AK
DELHI, INDIA
DELRAY BEACH, FL
DENPASAR, INDONESIA
DENVER, CO
DES MOINES, IA
DES MOINES, IOWA
DESTIN, FL
DETROIT, MI
DHAKA, BANGLADESH
DIGOS CITY, PHILIPPINES
DILI, EAST TIMOR
DILI, TIMOR-LESTE
DINGMANS FERRY, PA
DIRE DAWA, ETHIOPIA
DOHA, QATAR
DOLLO ADO AIRFIELD, ETHIOPIA
DRESDEN, GERMANY

DUBAI, UAE
DUBAI, UNITED ARAB EMIRATES
DUBLIN, IRELAND
DUBUQUE, IA
DURBAN, SOUTH AFRICA
DURHAM, NC
DURRES, ALBANIA
DYERSVILLE, IA
EDMONTON, CANADA
EL CARMEN, GUATEMALA
EL PASO, TX
EL PROGRESSO, HONDURAS
ELDORET, KENYA
ELKTON, MD
ELMAU OBERBAYERN, GERMANY
ELMAU, GERMANY
ENTEBBE, UGANDA
ENTEBBE,UGANDA
ERBIL, IRAQ
FAIRBANKS, AK
FAIRBURY, IL
FAJARDO, PR
FAJARDO, PUERTO RICO
FARGO, ND
FARLEY, IA
FARMINGTON, PA
FISHING CREEK, MD
FLORENCE, ITALY
FLOYD, IA
FORT COLLINS, CO
FORT DODGE, IA
FORT LAUDERDALE, FL
FORT LAURDERDALE, FL
FORT MCMURRAY, CANADA
FORT MYERS, FL
FORT WAYNE, IN
FORT WORTH, TEXAS
FORT WORTH, TX
FORT WORTH/DALLAS, TX
FRANKFURT, GERMANY
FREDERICKSBURG, VA
FREETOWN, SIERRA LEONE
FRESNO, CA
FT LAUDERDALE, FL
FT. LAUDERDALE, FL
FT. MCMURRAY, CANADA
FT. MYERS, FL
FT. WORTH, TEXAS

FUKUSHIMA, JAPAN
GALBRAITH LAKE, AK
GALVA, IA
GAROUA, CAMEROON
GAZIANTEP, TURKEY
GAZIATEP, TURKEY
GETTYSBURG, PA
GIBSON CITY, IL
GINOSAR, ISRAEL
GITEGA, BURUNDI
GOLAN HEIGHTS
GOMA, DEMOCRATIC REPUBLIC OF C
GOMA, DEMOCRATIC REPUBLIC OF T
GOREM, TURKEY
GORONGOSA NATIONAL PARK, MOZAM
GREAT FALLS, MT
GREENSBORO, NC
GUADALAJARA, MEXICO
GUANGZHOU, CHINA
GUANZHOU, CHINA
GUATAMALA CITY, GUATAMALA
GUATEMALA CITY, GUATEMALA
GULFPORT, MS
HAGATNA, GUAM
HAGOSHRIM, ISRAEL
HAITI
HAMPTON, VA
HANOI, VIETNAM
HANOVER, NH
HARPER'S FERRY, WV
HARPERS FERRY, WV
HARRISBURG, PA
HARRODSBURG, KY
HARTFORD, CONNECTICUT
HARTFORD, CT
HARWICH, MA
HARWOOD, MD
HAVANA, CUBA
HAVANNA, CUBA
HAVERFORD, PA
HAWASSA, ETHIOPIA
HAZYVIEW, SOUTH AFRICA
HELSINKI, FINLAND
HERSHEY, PA
HILLSDALE, MI
HILTON HEAD ISLAND, NC
HILTON HEAD ISLAND, SC
HILTON HEAD, SC

HIROSHIMA, JAPAN
HO CHI MINH CITY, VIETNAM
HOLLYWOOD, FL
HONG KONG, CHINA
HORTA, AZORES
HOT SPRING, VA
HOT SPRINGS, VA
HOT SPRINGS,VA
HOUMA, LA
HOUSTON, TX
HUNTSVILLE, AL
HUXLEY, IA
HYATT CHESAPEAKE BAY, CAMBRIDG
HYATT REGENCY CHESAPEAKE BAY,
IDAHO FALLS, ID
INDIAN WELLS, CA
INDIANAPOLIS, IN
INGA DAM, DEMOCRATIC REPUBLIC
INSTANBUL, TURKEY
IQUITOS, PERU
ISLAMABAD, PAKISTAN
ISLAMORADA, FL
ISRAEL
ISRAEL, JERUSALEM
ISTANBUL, ISRAEL
ISTANBUL, TUKEY
ISTANBUL, TURKEY
ITHACA, NY
IWAKUNI, JAPAN
IWO JIMA, JAPAN
IZMIR, TURKEY
JACKSON HOLE, WY
JACKSON, MS
JACKSON, WY
JACKSONVILLE, FL
JAGDALPUR, INDIA
JAKARTA, INDONESIA
JEFFERSON PARISH, LA
JERSALEM, ISRAEL
JERSEY CITY, NJ
JERSUALEM, ISRAEL
JERUSAELM, ISRAEL
JERUSALEM, ISRAE
JERUSALEM, ISRAEL
JERUSALEM, ISREAL
JERUSALEM, PALESTINIAN TERRITO
JOHANNESBURG, SOUTH AFRICA
JOHNANNESBURG, SOUTH AFRICA

JOHNSTON, IA
JUAREZ, MEXICO
JUBA, SOUTH SUDAN
JUNEAU, AK
KABUL, AFGANISTAN
KABUL, AFGHANISTAN
KALAMAZOO, MI
KALISPELL, MT
KAMAKURA, JAPAN
KAMPALA, UGANDA
KANSAS CITY, KS
KANSAS CITY, MO
KATHMADU, NEPAL
KATHMANDU, NEPAL
KAYSERI, TURKEY
KEMPFENHAUSEN, GERMANY
KERICHO, KENYA
KEY LARGO, FL
KEY WEST, FL
KEYSTONE, CO
KHARGA, EGYPT
KHARTOUM, SUDAN
KIEV, UKRAINE
KIGALI, REPUBLIC OF RWANDA
KIGALI, RWANDA
KINSALE, VA
KINSHASA, DEMOCRATIC REPUBLIC
KINSHASHA, DEMOCRATIC REPUBLIC
KISUMU, KENYA
KNOXVILLE, TN
KRAKOW, POLAND
KUALA LUMPUR, MALAYSIA
KURDISTAN, IRAQ
KUTAISI, GEORGIA
KUWAIT CITY, KUWAIT
KYIV, UKRAINE
KYOTO, JAPAN
KYRENIA, CYPRESS
KYRENIA, CYPRUS
KYRENIA/GIRNE, CYPRUS
L'AQUILA, ITALY
LA ESPERANZA, HONDURAS
LA GUARDIA AIRPORT, NY
LA PAZ, BOLIVIA
LA QUINTA, CA
LA SELVA, COSTA RICA
LACKAWAXEN, PA
LAGUNA BEACH, CA

LANSING, MI
LARNACA, CYPRUS
LAS CRUCES, NM
LAS VEGAS, NEVADA
LAS VEGAS, NV
LAYTON, NJ
LEBANON, NH
LEIPZIG, GERMANY
LEXINGTON, KY
LEXINGTON, MA
LEXINGTON, VA
LEXINGTON,VA
LIBERIA, COSTA RICA
LILONGWE, MALAWI
LILONGWE, MILAWI
LIMA, PERU
LISBON, PORTUGAL
LITTLE ROCK, AR
LIVINGSTONE, ZAMBIA
LONDON, ENGLAND
LONDON, UK
LONDON, UNITED KINGDOM
LONDON, UNITED KINGDON
LONG BEACH, CA
LOS ANGELES, CA
LOUISVILLE, KY
LUSAKA, ZAMBIA
LUSBY, MD
LUTHER, MI
LUXOR, EGYPT
MACKINAC ISLAND, MI
MACON, GA
MACON, GEORGIA
MADISON, WI
MADRID, SPAIN
MAGLAJ, BOSNIA AND HERZEGOVINA
MALAKAL, SOUTH SUDAN
MALATYA, TURKEY
MANAMA, BAHRAIN
MANASSAS, VA
MANCHESTER, NH
MANHASSET, NY
MAPUTO, MOZAMBIQUE
MARION COUNTY, KY
MARRAKECH, MOROCCO
MARRAKESH, MOROCCO
MARSHALLTOWN, IA
MARTHA'S VINEYARD, MA

MASADA, ISRAEL
MASON CITY, IA
MATALYA, TURKEY
MATAM, SENEGAL
MATANZAS, CUBA
MAUN, BOTSWANA
MBEYA, TANZANIA
MEDAN, INDONESIA
MEDELLIN, COLOMBIA
MEINE, GERMANY
MELBOURNE, AUSTRALIA
MEMPHIS, TN
MERCER COUNTY, KY
MEXICO CITY, MEXICO
MFUWE, ZAMBIA
MIAMI, FL
MIDDLBURG, VA
MIDDLEBURG, CA
MIDDLEBURG, VA
MIDDLEBURG, VIRGINIA
MIDLAND, TX
MIDWEST CITY, OK
MILAN, ITALY
MILWAUKEE, WI
MINERAL, VA
MINNEAPOLIS, MN
MINSK, BELARUS
MISSOULA, MT
MOBILE, AL
MOGADISHU, SOMALIA
MOLINE, IL
MONOROVIA, LIBERIA
MONROVIA
MONROVIA, LIBERIA
MONTEREY, CA
MONTERIA, COLOMBIA
MONTREAL, CANADA
MORRIS, IL
MOSTAR, BOSNIA
MOSTAR, BOSNIA AND HERZEGOVINA
MOUNTAIN LAKES, NJ
MOUNTAIN VIEW, CA
MOUNTAINVIEW, CA
MUMBAI, INDIA
MUNICH, GERMANY
MUSCAT, OMAN
MUSINA, SOUTH AFRICA
MUYINGA, BURUNDI

MWANZA, TANZANIA
MYRTLE BEACH, NC
MYRTLE BEACH, SC
N. AUGUSTA, SC
NAGOYA, JAPAN
NAIROBI, KENYA
NANJING, CHINA
NAPA, CA
NASHVILLE TN
NASHVILLE, TN
NASSAU, BAHAMAS
NAYPYITAW, MYANMAR
NEBRASKA CITY, NE
NECHI, COLOMBIA
NEVADA, IA
NEVSEHIR, TURKEY
NEVSEHIR/GOREME, TURKEY
NEW DEHLI, INDIA
NEW DELHI, INDIA
NEW DELLHI, INDIA
NEW HAVEN, CT
NEW ORLEANS, LA
NEW ORLEANS, LA THIBODAUX, LA
NEW ORLEANS, LOUISIANA
NEW YORK ,NY
NEW YORK CITY, NEW YORK
NEW YORK CITY, NY
NEW YORK, N.Y.
NEW YORK, NEW YORK
NEW YORK, NY
NEW YORK. NY
NEWARK, DE
NEWARK, NJ
NEWPORT BEACH, CA
NEWTON, IA
NIAMEY, NIGER
NICOSIA, CYPRESS
NICOSIA, CYPRUS
NICOSIA/LEFKOSA, CYPRUS
NIDGE, TURKEY
NIGDE, TURKEY
NOGALES, MEXICO
NORFOLK, VA
NORTH AUGUSTA, GA
NORTH AUGUSTA, SC
NORTHERN IRAQ, IRAQ
OAK RIDGE, TN
OAKLAND, CA

OBERLIN, OH
OCALA, FL
OHRID, MACEDONIA
OKINAWA, JAPAN
OKLAHOMA CITY, OK
OMAHA, NB
OMAHA, NE
ONEONTA, NY
ONTARIO, CANADA
ORADEA, ROMANIA
ORANGE, VA
ORANGEBURG, SC
ORLANDO, FL
OSAKA, JAPAN
OSLO, NORWAY
OWINGS MILLS, MD
OXFORD, UNITED KINGDOM
PALM BEACH, FL
PALM SPRINGS, CA
PALO ALTO, CA
PALO, ALTO
PALTO ALTO, CA
PANAMA CITY, PANAMA
PANAMA CITY, REPUBLIC OF PANAM
PARIS, FRANCE
PATARA, TURKEY
PAXTON, IL
PEORIA, IL
PERTH, AUSTRLIA
PETION-VILLE, HAITI
PETIONVILLE, HAITI
PHIADELPHIA, PA
PHILADELPHIA, PA
PHILADELPHIA, PA
PHILADEPHIA, PA
PHILAPELPHIA, PA
PHILDELPHIA, PA
PHNOM PENH, CAMBODIA
PHOENIX, AZ
PIKESVILE, MD
PIKESVILLE, MD
PINE BLUFF, AR
PITTSBURGH, PA
PLAYA GRANDE, COSTA RICA
PODGORICA, MONTENEGRO
POINT CLEAR, AL
PONCE, PUERTO RICO
PORT AU PRINCE, HAITI

PORT MORESBY, PAPUA NEW GUINEA
PORT OF SPAIN, TRINIDAD AND TO
PORT VILLA, VANUATU
PORT-AU-PRINCE, HAITI
PORTERVILLE, CA
PORTLAND, OR
PORTO, PORTUGAL
POTSDAM, GERMANY
PRAGUE, CZECH REPUBLIC
PRETORIA, SOUTH AFRICA
PRINCETON, NJ
PRISHTINA, KOSOVO
PRISTINA, KOSOVO
PRISTINA, KOSOVO,
PROVIDENCE, RI
PROVO, UT
PUERTO JORDAN, COLOMBIA
PUNE, INDIA
PUNTA CANA, DOMINICAN REPUBLIC
QINGDAO, CHINA
QUANTICO, VA
QUEENS, NY
QUEENSTOWN, MD
QUETZALTENANGO, GUATEMALA
RABAT, MOROCCO
RADNOR, PA
RALEIGH, NC
RAMALLAH, PALESTINIAN TERRITOR
RANGOON, BURMA
REAGAN PRESIDENTIAL LIBRARY, C
REDMOND, CA
REDMOND, WA
REDWOOD CITY, CA
REHOBETH, DE
REHOBOTH BEACH, DE
REHOBOTH, DE
RHINEBECK, NY
RICHARD TOLL, SENEGAL
RICHMOND, VA
RICHMOND,VA
RIO DE JANEIRO, BRAZIL
RIVERTON, WY
ROCHESTER, NY
ROCKWELL CITY, IA
ROME, ITALY
ROSH PINA, ISRAEL
ROYAL OAK, MD
RUCKERSVILLE, VA

SACRAMENTO, CA
SAGINAW, MI
SAINT LOUIS, SENEGAL
SALEM, NJ
SALT LAKE CITY, UT
SAN ANTONIO, TX
SAN DIEGO CA
SAN DIEGO, CA
SAN FRANCISCO, CA
SAN FRANICISCO, CA
SAN FRANSICO, CA
SAN FRANSISCO, CA
SAN JOSE, CA
SAN JOSE, COSTA RICA
SAN JUAN, PR
SAN JUAN, PUERTO RICO
SAN LUIS OBISPO, CA
SAN PEDRO SULA, HONDURAS
SAN PEDRO, COTE D'IVOIRE
SAN PEDRO, HONDURAS
SAN SALVADOR, EL SALVADOR
SAN, DIEGO, CA
SANLIURFA, TURKEY
SANTA BARBARA, CA
SANTA BARBARA, CALIFORNIA
SANTA CLARA, CA
SANTA FE, NM
SANTA MONICA, CA
SANTIAGO DE COMPOSTELA, SPAIN
SAO PAULO, BRAZIL
SARAJEVO, BOSNIA
SARAJEVO, BOSNIA AND HERZEGOVI
SARAJEVO, BOSNIA-HERZEGOVINA
SARASOTA, FL
SAREJEVO, BOSNIA
SCOTTSDALE, AZ
SCRANTON, PA
SEA ISLAND, GA
SEA ISLANDS, GA
SEA OF GALILEE, ISRAEL
SEATTE, WA
SEATTLE, WA
SELCUK, TURKEY
SENDAI, JAPAN
SEOUL, KOREA
SEOUL, SOUTH KOREA
SERGEANT BLUFF, IA
SEVILLA, SPAIN

SHANGHAI, CHINA
SHARM EL-SHEIKH, EGYPT
SHEFFIELD, IA
SHELBY COUNTY, KY
SHELBYVILLE, KY
SHORT HILLS, NJ
SIAYA, KENYA
SIMI VALLEY, CA
SINCELEJO, COLOMBIA
SINGAPORE
SIOUX CITY, IA
SKOPJE, MACEDONIA
SOLON, IA
SOUTH BEND, IN
SOUTH HAMILTON, MA
SOUTH ROYALTON, VT
SOUTH WINDSOR, CT
SOUTHAMPTON, BERMUDA
SOUTHPOINTE, PA
SPENCER COUNTY, KY
SPOKANE, WA
SPRING CITY, TN
SPRINGFIELD, IL
ST. AUGUSTINE, FL
ST. CROIX, VI
ST. KITTS, VI
ST. LOUIS, MO
ST. MICHAEL'S, MD
ST. MICHAELS, MD
ST. PAUL, MN
ST. THOMAS, VI
STANFORD, CA
STANLEY, ID
STEAMBOAT SPRINGS, CO
STEAMBOAT, CO
STERLING, VA
STEVENSVILLE, MD
STILLWATER, OK
STOCKHOLM, SWEDEN
STRASBOURG, FRANCE
STUTGART, GERMANY
STUTTGART, GERMANY
STUTTGART. GERMANY
SULPHER SPRINGS, WV
SUNNY ISLES, FL
TAIPEI, TAIWAN
TAL AVIV, ISRAEL
TAMALE, GHANA

TAMPA, FL
TAMUNING, GUAM
TAPACHULA, MEXICO
TARRYTOWN, NY
TBILISI, GEORGIA
TBLISI, GEORGIA
TECPAN, GUATEMALA
TEGUCIGALPA, HONDURAS
TEL AVIV, ISRAEL
TEL AVIV, ISREAL
TEL-AVIV, ISRAEL
TELA, HONDURAS
TELAVI, GEORGIA
THE HYATT REGENCY CHESAPEAKE B
THIBADAUX, LA
THIBODAUX, LA
THORNTON, IA
THURMONT, MD
TIBERAIS, ISRAEL
TIBERAS, ISRAEL
TIBERIAS
TIBERIAS, ISRAEL
TIBERIAS, ISREAL
TIBERIUS, ISRAEL
TIBRIAS, ISRAEL
TINDOUF, ALGERIA
TIRANA, ALBANIA
TOHOKU, JAPAN
TOKYO, JAPAN
TORONTO, CANADA
TORTUGUERO, COSTA RICA
TOTONICAPAN, GUATEMALA
TOTONICIPAN, GUATEMALA
TOWSON, MD
TOYKO, JAPAN
TRABZON, TURKEY
TULSA, OK
TUNICA, MI
TUNICA, MS
TUNIS, TUNISIA
TUNKHANNOCK, PA
TURIN, ITALY
TURRIALBA, COSTA RICA
ULAANBAATAR, MONGOLIA
UNION BRIDGE, MD
URBANDALE, IA
URBANDALE,IA
VACHERIE, LA

VALDEZ, AK
VANCOUVER, CANADA
VIENNA, AUSTRIA
VILLEPINTE, FRANCE
VILLIPINTE, FRANCE
VIRGINIA BEACH, VA
VISALIA, CA
VLAANBAATAR, MONGOLIA
WAIMEA, HAWAII
WALL LAKE, IA
WARREN, NJ
WARRENTON, VA
WARRENTON,VA
WARSAW, POLAND
WASHINGTON, DC
WASHINGTON, IA
WAYNESBORO, GA
WENHAM, MA
WEST LAKE VILLAGE, CA
WEST PALM BEACH
WEST PALM BEACH, FL
WEST PALM PEACH, FL
WEST POINT, NY
WESTLAKE VILLAGE, CA
WHEELING, WV
WHITE HALL, MD
WHITE SULPHUR SPRINGS, WV
WICHITA, KS
WIITENBERG, GERMANY
WILILAMSBURG, VA
WILLIAMSBURG, VA
WILLIMASBURG, VA
WILMINGTON, DE
WITCHITA, KS
WITTENBERG, GERMANY
WOOSTER, OH
WROCLAW, POLAND
WYTHEVILLE, VA
XI'AN, CHINA
YAMBIO, SOUTH SUDAN
YANGON, MYANMAR
YAOUNDE, CAMEROON
YEREVAN, ARMENIA
ZAGREB, CROATIA
ZANZIBAR, TANZANIA
ZHYTOMYR, UKRAINE
ZUGDIDI, GEORGIA
ZURICH, SWITZERLAND

And here is where we're going to start encoding our editorial choices. "Ames, IA" or "Ames, Iowa"? "Baku, Azerjaijan," or "Baku, Republic of Azerbaijan"? Etc.

There are several ways we could structure this data, but a dictionary makes some sense based on what we need to do, so let's do that. Each key will be a string that we'd like to replace; each value will be the string we'd like to replace it with. To get us started:

[108]:
```
typo_fixes = {
    'BAKU, AZERBIJAN': 'BAKU, AZERBAIJAN',
    'BAKU, REPUBLIC OF AZERBAIJAN': 'BAKU, AZERBAIJAN',
    'ADDIS, ETHIOPIA': 'ADDIS ABABA, ETHIOPIA',
    'ANKEY, IA': 'ANKENY, IA'
}
```

… and so on. (This is tedious work, and – again – tools like OpenRefine make this process somewhat less tedious. But if you have a long-term project that involves data that will be updated regularly, and it's worth putting in the time to make sure the data are cleaned the same way each time, you can do it all in pandas.)

Here's how we might *apply* our bulk find-and-replace dictionary:

[109]:
```
def find_replace_destination(row):
    '''Given a row of data, see if the value is a typo to be replaced'''

    # get the clean destination value
    dest = row['destination_clean']

    # try to look it up in the `typo_fixes` dictionary
    # the `get()` method will return None if it doesn't find a match
    typo = typo_fixes.get(dest)

    # then we can test to see if `get()` got an item out of the dictionary␣
 ↪(True)
    # or if it returned None (False)
    if typo:
        # if it found an entry in our dictionary,
        # return the value from that key/value pair
        return typo_fixes[dest]
    # otherwise
    else:
        # return the original destination string
        return dest
```

[110]:
```
# apply the function and overwrite our working "clean' column"
df_junkets['destination_clean'] = df_junkets.apply(find_replace_destination,␣
 ↪axis=1)
```

[111]:
```
df_junkets.head()
```

[111]:
```
        DocID            FilerName      MemberName State  District  Year  \
0  500005076         Bobby Cornett   Franks, Trent    AZ       8.0  2011
1  500005077  Michael Strittmatter   Franks, Trent    AZ       8.0  2011
2  500005081         Diane Rinaldo    Rogers, Mike    AL       3.0  2011
3  500005082       Kenneth DeGraff  Doyle, Michael    PA      14.0  2011
4  500005083    Michael Ryan Clough    Lofgren, Zoe    CA      19.0  2011

     Destination FilingType DepartureDate ReturnDate  \
0  Las Vegas, NV   Original      1/7/2011   1/9/2011
1  Las Vegas, NV   Original      1/7/2011   1/9/2011
2  Las Vegas, NV   Original      1/6/2011   1/8/2011
3  Las Vegas, NV   Original      1/6/2011   1/9/2011
4  Las Vegas, NV   Original      1/6/2011   1/8/2011

                    TravelSponsor destination_clean
0  Consumer Electronics Association     LAS VEGAS, NV
1         CEA Leaders in Technology     LAS VEGAS, NV
2  Consumer Electronics Association     LAS VEGAS, NV
3  Consumer Electronics Association     LAS VEGAS, NV
4  Consumer Electronics Association     LAS VEGAS, NV
```

1.0.7 Further reading

This just scratches the surface of what you can do in pandas. Here are some other resources to check out:

- Pythonic Data Cleaning With NumPy and Pandas
- pandas official list of tutorials
- Karrie Kehoe's guide to cleaning data in pandas
- Data cleaning with Python