proj1a

May 15, 2020

[2]:
```
# Initialize OK
from client.api.notebook import Notebook
ok = Notebook('proj1a.ok')
```

```
=====================================================================
Assignment: proj1a
OK, version v1.13.11
=====================================================================
```

1 Project 1A: Food Safety

1.1 Cleaning and Exploring Data with Pandas

1.2 Due Date: Monday 02/17, 11:59 PM

1.3 Collaboration Policy

Data science is a collaborative activity. While you may talk with others about the project, we ask that you **write your solutions individually**. If you do discuss the assignments with others please **include their names** at the top of your notebook.

Collaborators: *list collaborators here*

1.4 This Assignment

In this project, we will investigate restaurant food safety scores for restaurants in San Francisco. The scores and violation information have been made available by the San Francisco Department of Public Health. The main goal for this assignment is to walk through the process of Data Cleaning and EDA.

We have broken this two week project into two parts. The first part (week 1) will focus on basic data cleaning and analysis using Pandas. The second part (week 2) will dive into deeper analysis and data visualization.

As we clean and explore these data, you will gain practice with: * Reading simple csv files and using Pandas * Working with data at different levels of granularity * Identifying the type of data

collected, missing values, anomalies, etc. * Exploring characteristics and distributions of individual variables

1.5 Score Breakdown

Question	Points
1a	3
1b	2
1c	1
1d	1
1e	1
2a	1
2b	1
2ci	1
2cii	1
2d	2
3a	2
3b	2
3ci	1
3cii	1
3d	3
3e	2
3f	2
4a	1
4bi	1
4bii	1
4biii	1
4ci	1
4cii	1
4ciii	1
4civ	1
4di	2
4dii	2
4e	2
5a	2
5b	3
5c	2
6	3
Total	51

1.6 Before You Start

For each question in the assignment, please write down your answer in the answer cell(s) right below the question.

We understand that it is helpful to have extra cells breaking down the process towards reaching

your final answer. If you happen to create new cells below your answer to run codes, **NEVER** add cells between a question cell and the answer cell below it. It will cause errors when we run the autograder, and it will sometimes cause a failure to generate the PDF file.

Important note: The local autograder tests will not be comprehensive. You can pass the automated tests in your notebook but still fail tests in the autograder. Please be sure to check your results carefully.

Finally, unless we state otherwise, try to avoid using python for loops or list comprehensions. The majority of this assignment can be done using builtin commands in Pandas and numpy.

[3]:
```
import numpy as np
import pandas as pd

import bz2 # Used to read compressed data
import os # Used to interact with the file system
```

1.7 Obtaining the Data

1.7.1 File Systems and I/O

In general, we will focus on using python commands to investigate files. However, it can sometimes be easier to use shell commands in your local operating system. The following cells demonstrate how to do this.

The command below starts with `!`. This tells our Jupyter notebook to pass this command to the operating system. In this case, the command is the `ls` POSIX command which lists all files in the current directory. Note what `!ls data` outputs.

Note this `!` only works in ipython shells (Jupyter Notebooks). And the `ls` (list) command only works in posix environments and may not work on default Windows systems.

[4]:
```
!ls
```

```
data  proj1a.ipynb  proj1a.ok  proj1a.pdf  tests  test.tplx
```

[5]:
```
!ls data
```

```
bus.csv.bz2  ins2vio.csv.bz2  ins.csv.bz2  sf_zipcodes.json  vio.csv.bz2
```

We are going to use the `pathlib` module to represent our file system paths and perform operations which allow us to learn more about the contents of our data. Note what `pathlib.Path.cwd()` outputs in relation to the output of `!ls` above.

---

2 1: Examining the Files

Let's first focus on understanding the structure of the data; this involves answering questions such as:

- How much data do we have?
- Is the data in a standard format or encoding?
- Is the data organized in records?
- What are the fields in each record?

Let's start by looking at the contents of `data/`. This is not just a single file but rather a directory of multiple compressed files. We could inspect our data by uncompressing each file but in this project we're going to do almost everything in Python for maximum portability.

You will want to use a few useful python functions. To move through the local filesystem you can use the `Path` module in `pathlinb`. For example, to list the current directory you can Path.cwd.

[6]:
```
from pathlib import Path

Path.cwd()
```

[6]:
```
PosixPath('/home/jovyan/sp20/proj/proj1a')
```

The function returns a `pathlib.Path` object representing the location of the file. It can also be used to list contents of directories and many other things.

You will also need to work with `bzip2` files and you will want to be able to read their contents using the bz2 python library.

[7]:
```
with bz2.open("data/bus.csv.bz2", "r") as f:
    print("The first line:", "\n\t", f.readline())
```

```
The first line:
         b'"business id column","name","address","city","state","postal_code","l
atitude","longitude","phone_number"\n'
```

2.1 Question 1a:

Implement the `list_files`, `get_file_size`, and `get_linecount_bz2` functions to return the list of files in the directory, the sizes (in bytes) of a file, and the number of lines in the file. Note the last `get_linecount_bz2` should not produce any intermediate files in the filesystem and should avoid storing the entire file in memory (don't do `len(file.readlines())`).

Hints: *You might find the following documentation useful:* 1. Python pathlib 1. bz2

[8]:
```
def list_files(directory):
    """
    Return a list of pathlib.Path objects for the files in the directory.
```

```
    directory: a string describing the directory to list
        for example 'data/'
    """
    p = Path(directory)
    return [obj for obj in p.iterdir()]



def get_file_size(file_name):
    """
    Return file size for a given filename.
    """
    return os.stat(file_name).st_size

def get_linecount_bz2(file_name):
    """
    Returns the number of lines in bz2 file.
    """
    with bz2.open(file_name, "r") as f:
        count = 0
        for line in f:
            count += 1
        return count
```

[9]:
```
ok.grade("q1a");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 3
    Failed: 0
[oooooooooook] 100.0% passed
```

Now, let's see the file size and the number of lines for each data file. If you implemented the above code correctly the following cell should produce the following (the columns may be in a different order):

linecount

name

size

2

66

data/vio.csv.bz2

1337

3

26664

data/ins.csv.bz2

110843

0

6254

data/bus.csv.bz2

113522

1

40211

data/ins2vio.csv.bz2

146937

[10]:
```
info = []
for f in list_files("data/"):
    name = str(f)
    if name[-3:] == "bz2":
        size = get_file_size(f)
        linecount = get_linecount_bz2(f)
        info.append({"name": name, "size": size, "linecount": linecount})

file_info = pd.DataFrame(info).sort_values("size")
file_info
```

[10]:

	name	size	linecount
3	data/vio.csv.bz2	1337	66
1	data/ins.csv.bz2	110843	26664
2	data/bus.csv.bz2	113522	6254
0	data/ins2vio.csv.bz2	146937	40211

2.2 Question 1b: Programatically Looking Inside the Files

Implement the following function `head_bz2` to return a list of the first `nlines` lines of each file. Using your `head_bz2` function implement the following `print_head_bz2` function that uses `print()`

to print the filename followed by the first `nlines` of each file and their line numbers in the following format.

```
data/bus.csv.bz2
0 :  b'"business id column","name","address","city","state","postal_code","latitude","longitude
1 :  b'"1000","HEUNG YUEN RESTAURANT","3279 22nd St","San Francisco","CA","94110","37.755282","
2 :  b'"100010","ILLY CAFFE SF_PIER 39","PIER 39  K-106-B","San Francisco","CA","94133","-9999
3 :  b'"100017","AMICI\'S EAST COAST PIZZERIA","475 06th St","San Francisco","CA","94103","-999
4 :  b'"100026","LOCAL CATERING","1566 CARROLL AVE","San Francisco","CA","94124","-9999","-9999
```

Do not read the entire file contents!

[11]:
```
def head_bz2(file, nlines=5):
    """
    Return a list of the first nlines lines of filename
    """
    lst = []
    with bz2.open(file, "r") as f:
        for i in range(nlines):
            lst.append(f.readline())
    return lst

def print_head_bz2(file, nlines=5):
    """
    Print a list of the first nlines lines of filename
    """
    print(file)
    count = 0
    for f in head_bz2(file):
        print(count, ":\t ", f)
        count += 1
```

[12]:
```
ok.grade("q1b");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 2
    Failed: 0
[oooooooooo] 100.0% passed
```

The following should display the filename and head of the file for all the files in data:

[13]:
```
for file in list_files("data/"):
    if str(file)[-3:] == "bz2":
        print_head_bz2(file)
```

```
        print()
```

```
data/ins2vio.csv.bz2
0 :       b'"iid","vid"\n'
1 :       b'"97975_20190725","103124"\n'
2 :       b'"85986_20161011","103114"\n'
3 :       b'"95754_20190327","103124"\n'
4 :       b'"77005_20170429","103120"\n'

data/ins.csv.bz2
0 :       b'"iid","date","score","type"\n'
1 :       b'"100010_20190329","03/29/2019 12:00:00 AM","-1","New
Construction"\n'
2 :       b'"100010_20190403","04/03/2019 12:00:00 AM","100","Routine -
Unscheduled"\n'
3 :       b'"100017_20190417","04/17/2019 12:00:00 AM","-1","New Ownership"\n'
4 :       b'"100017_20190816","08/16/2019 12:00:00 AM","91","Routine -
Unscheduled"\n'

data/bus.csv.bz2
0 :       b'"business id column","name","address","city","state","postal_code","
latitude","longitude","phone_number"\n'
1 :       b'"1000","HEUNG YUEN RESTAURANT","3279 22nd St","San
Francisco","CA","94110","37.755282","-122.420493","-9999"\n'
2 :       b'"100010","ILLY CAFFE SF_PIER 39","PIER 39  K-106-B","San
Francisco","CA","94133","-9999","-9999","+14154827284"\n'
3 :       b'"100017","AMICI\'S EAST COAST PIZZERIA","475 06th St","San
Francisco","CA","94103","-9999","-9999","+14155279839"\n'
4 :       b'"100026","LOCAL CATERING","1566 CARROLL AVE","San
Francisco","CA","94124","-9999","-9999","+14155860315"\n'

data/vio.csv.bz2
0 :       b'"description","risk_category","vid"\n'
1 :       b'"Consumer advisory not provided for raw or undercooked
foods","Moderate Risk",103128\n'
2 :       b'"Contaminated or adulterated food","High Risk",103108\n'
3 :       b'"Discharge from employee nose mouth or eye","Moderate
Risk",103117\n'
4 :       b'"Employee eating or smoking","Moderate Risk",103118\n'
```

2.3 Question 1c: Thinking about the files

Answer the following questions by filling in the correct boolean values in the following variables:

1. The bus.csv.bz2 file appears to be tab delimited.

2. The values all appear to be quoted.

[14]:
```
# True or False: The bus.csv.bz2 file appears to be tab delimited.
q1c_1 = False

# True or False: The values all appear to be quoted.
q1c_2 = True
```

[15]:
```
ok.grade("q1c");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 2
    Failed: 0
[oooooooooook] 100.0% passed
```

2.4 Question 1d: Reading in the Files

Based on the above information, let's attempt to load `bus.csv`, `ins2vio.csv`, `vio.csv` and `ins.csv` into Pandas dataframes with the following names: `bus`, `ins2vio`, `vio`, `ins` respectively.

[16]:
```
# path to directory containing data
bus = pd.read_csv("data/bus.csv.bz2", warn_bad_lines=True)
ins = pd.read_csv("data/ins.csv.bz2", warn_bad_lines=True)
ins2vio = pd.read_csv("data/ins2vio.csv.bz2", warn_bad_lines=True)
vio = pd.read_csv("data/vio.csv.bz2", warn_bad_lines=True)
```

[17]:
```
ok.grade("q1d");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 4
    Failed: 0
[oooooooooook] 100.0% passed
```

Now that you've read in the files, you can try some `pd.DataFrame` methods (docs). You can use the `DataFrame.head` method to show the top few lines of the `bus`, `ins`, `ins2vio` and `vio` dataframes.

[18]:
```
bus.head()
```

[18]:
```
   business id column                          name                address  \
0                1000        HEUNG YUEN RESTAURANT            3279 22nd St
1              100010        ILLY CAFFE SF_PIER 39        PIER 39  K-106-B
2              100017  AMICI'S EAST COAST PIZZERIA             475 06th St
3              100026                LOCAL CATERING        1566 CARROLL AVE
4              100030             OUI OUI! MACARON  2200 JERROLD AVE STE C

            city state postal_code     latitude    longitude  phone_number
0  San Francisco    CA       94110    37.755282  -122.420493         -9999
1  San Francisco    CA       94133 -9999.000000 -9999.000000   14154827284
2  San Francisco    CA       94103 -9999.000000 -9999.000000   14155279839
3  San Francisco    CA       94124 -9999.000000 -9999.000000   14155860315
4  San Francisco    CA       94124 -9999.000000 -9999.000000   14159702675
```

[19]:
```
ins.head()
```

[19]:
```
               iid                    date  score                  type
0  100010_20190329  03/29/2019 12:00:00 AM     -1       New Construction
1  100010_20190403  04/03/2019 12:00:00 AM    100  Routine - Unscheduled
2  100017_20190417  04/17/2019 12:00:00 AM     -1           New Ownership
3  100017_20190816  08/16/2019 12:00:00 AM     91  Routine - Unscheduled
4  100017_20190826  08/26/2019 12:00:00 AM     -1  Reinspection/Followup
```

[20]:
```
ins2vio.head()
```

[20]:
```
              iid     vid
0  97975_20190725  103124
1  85986_20161011  103114
2  95754_20190327  103124
3  77005_20170429  103120
4   4794_20181030  103138
```

[21]:
```
vio.head()
```

[21]:
```
                                        description  risk_category     vid
0  Consumer advisory not provided for raw or unde…  Moderate Risk  103128
1                  Contaminated or adulterated food      High Risk  103108
2         Discharge from employee nose mouth or eye  Moderate Risk  103117
3                        Employee eating or smoking  Moderate Risk  103118
4                            Food in poor condition  Moderate Risk  103123
```

2.5 Question 1e: Identifying Issues with the Data

Use the `head` command on your four files again. This time, describe at least one potential problem with the data you see. Consider issues with missing values and bad data.

Please write your answer in the markdown cell below. You may create new cells below your answer to run code, but **please never add cells between a question cell and the answer cell below it.**

Answer:

Some columns do not have correct/specified data. For example, some locations in bus.csv file do not have correct latitude and longitude; and some locations in ins.csv does not have correct score. Thus, when combining the values, for example, summing all the score, these data will not be correct.

---

3 2: Examining the Business Data File

From its name alone, we expect the `bus.csv` file to contain information about the restaurants. Let's investigate the granularity of this dataset.

[22]:
```
bus.head()
```

[22]:

	business id column	name	address	\
0	1000	HEUNG YUEN RESTAURANT	3279 22nd St	
1	100010	ILLY CAFFE SF_PIER 39	PIER 39 K-106-B	
2	100017	AMICI'S EAST COAST PIZZERIA	475 06th St	
3	100026	LOCAL CATERING	1566 CARROLL AVE	
4	100030	OUI OUI! MACARON	2200 JERROLD AVE STE C	

	city	state	postal_code	latitude	longitude	phone_number
0	San Francisco	CA	94110	37.755282	-122.420493	-9999
1	San Francisco	CA	94133	-9999.000000	-9999.000000	14154827284
2	San Francisco	CA	94103	-9999.000000	-9999.000000	14155279839
3	San Francisco	CA	94124	-9999.000000	-9999.000000	14155860315
4	San Francisco	CA	94124	-9999.000000	-9999.000000	14159702675

3.1 Question 2a

The `bus` dataframe contains a column called `business id column` which probably corresponds to a unique business id. However, let's first rename that column to `bid`. Modify the `bus` dataframe by renaming that column to `bid`.

Note: In practice we might want to do this renaming when the table is loaded but for grading purposes we will do it here.

[23]:
```
bus = bus.rename(columns= {"business id column": "bid"})
```

[24]:
```
ok.grade("q2a");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[oooooooooook] 100.0% passed
```

3.2 Question 2b

Examining the entries in `bus`, is the `bid` unique for each record (i.e. each row of data)? Your code should compute the answer, i.e. don't just hard code `True` or `False`.

Hint: use `value_counts()` or `unique()` to determine if the `bid` series has any duplicates.

[25]:
```
is_bid_unique = bus["bid"].is_unique
is_bid_unique
```

[25]:
```
True
```

[26]:
```
ok.grade("q2b");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[oooooooooook] 100.0% passed
```

[27]:
```
bus
```

[27]:

	bid	name	address	\
0	1000	HEUNG YUEN RESTAURANT	3279 22nd St	
1	100010	ILLY CAFFE SF_PIER 39	PIER 39 K-106-B	
2	100017	AMICI'S EAST COAST PIZZERIA	475 06th St	
3	100026	LOCAL CATERING	1566 CARROLL AVE	
4	100030	OUI OUI! MACARON	2200 JERROLD AVE STE C	

```
…      …                              …                    …
6248   99948            SUSIECAKES BAKERY      3509 CALIFORNIA ST
6249   99988                HINODEYA SOMA     303 02nd ST STE 102
6250   99991                      TON TON            422 GEARY ST
6251   99992   URBAN EXPRESS KITCHENS LLC             475 06th ST
6252   99993            THE BRIXTON SOUTH             701 02nd St

                city state postal_code     latitude    longitude  phone_number
0     San Francisco    CA       94110    37.755282  -122.420493         -9999
1     San Francisco    CA       94133 -9999.000000 -9999.000000   14154827284
2     San Francisco    CA       94103 -9999.000000 -9999.000000   14155279839
3     San Francisco    CA       94124 -9999.000000 -9999.000000   14155860315
4     San Francisco    CA       94124 -9999.000000 -9999.000000   14159702675
…               …   …          …           …           …            …
6248  San Francisco    CA       94118 -9999.000000 -9999.000000   14150452253
6249  San Francisco    CA       94107 -9999.000000 -9999.000000         -9999
6250  San Francisco    CA       94102 -9999.000000 -9999.000000   14155531280
6251  San Francisco    CA       94103 -9999.000000 -9999.000000   14150368085
6252  San Francisco    CA       94102 -9999.000000 -9999.000000   14158315871

[6253 rows x 9 columns]
```

3.3 Question 2c

In the two cells below create two **series**

1. where the index is the `name` of the business and the value is the number of records with that `name`
2. where the index is the `address` of the business and the value is the number of records with that `address`

Order both series in descending order by count. You may need to use `groupby()`, `size()`, `sort_values()`, or `value_counts()`.

Step 1

[28]:
```
name_counts = bus["name"].value_counts()
name_counts.head(20)
```

[28]:
```
Peet's Coffee & Tea                    20
Starbucks Coffee                       13
Jamba Juice                            10
McDonald's                             10
Proper Food                             9
STARBUCKS                               9
Specialty's Cafe & Bakery               8
```

```
Mixt Greens/Mixt                                          8
Whole Foods Market                                        7
Starbucks                                                 7
Blue Bottle Coffee                                        7
The Organic Coup                                          7
Philz Coffee                                              7
Lee's Deli                                                6
Bon Appetit @ Twitter                                     6
BlueStar Refreshment Services @ Uber Technologies, Inc    6
Bon Appetit Management Co                                 5
La Boulangerie De San Francisco                           5
STARBUCKS COFFEE                                          5
JW Marriott SF Union Square                               5
Name: name, dtype: int64
```

[29]:
```
ok.grade("q2ci");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 2
    Failed: 0
[ooooooooook] 100.0% passed
```

[30]:
```
name_counts = bus[bus["name"] == "Peet's Coffee & Tea"]
name_counts.head(40)
```

[30]:

	bid	name	address
470	1493	Peet's Coffee & Tea	1 Ferry Building C3
495	15752	Peet's Coffee & Tea	2 Embarcadero Center #R2113
565	17822	Peet's Coffee & Tea	601 Van Ness Ave Suite A
713	2077	Peet's Coffee & Tea	2139C Polk St
821	24431	Peet's Coffee & Tea	692 Mission St
991	29312	Peet's Coffee & Tea	310 Broderick St
1097	32578	Peet's Coffee & Tea	1 California St
1098	32579	Peet's Coffee & Tea	5201 Geary Blvd
1118	32823	Peet's Coffee & Tea	405 Howard St
1441	37688	Peet's Coffee & Tea	370 04th St
1529	39025	Peet's Coffee & Tea	450 Sansome St Suite #1
1565	39665	Peet's Coffee & Tea	1400 Van Ness Ave
1566	39667	Peet's Coffee & Tea	595 Market St Suite #143
2955	70425	Peet's Coffee & Tea	1509 SLOAT Blvd
3222	75173	Peet's Coffee & Tea	919 Cole St
3458	77580	Peet's Coffee & Tea	2080 Chestnut St

```
3685  79970  Peet's Coffee & Tea                   101 Post St
4362  86354  Peet's Coffee & Tea                 773 Market St
4429  86780  Peet's Coffee & Tea                  121 Spear St
4430  86781  Peet's Coffee & Tea      1400 Mission St Suite 130

                city state postal_code     latitude    longitude  phone_number
470    San Francisco    CA       94111    37.795184  -122.393819         -9999
495    San Francisco    CA       94111    37.794818  -122.398474         -9999
565    San Francisco    CA       94102    37.781454  -122.420644         -9999
713    San Francisco    CA       94109    37.796316  -122.421909         -9999
821    San Francisco    CA       94105    37.786584  -122.401639         -9999
991    San Francisco    CA       94117    37.773348  -122.439147         -9999
1097   San Francisco    CA       94111    37.793614  -122.396517         -9999
1098   San Francisco    CA       94118    37.780443  -122.475259         -9999
1118   San Francisco    CA       94104    37.789057  -122.395322         -9999
1441   San Francisco    CA       94107    37.781127  -122.400109         -9999
1529   San Francisco    CA       94111    37.794617  -122.401316         -9999
1565   San Francisco    CA       94109    37.788668  -122.421917         -9999
1566   San Francisco    CA       94105    37.789437  -122.401111         -9999
2955   San Francisco    CA       94132 -9999.000000 -9999.000000   14150592100
3222   San Francisco    CA       94117 -9999.000000 -9999.000000   14150592100
3458   San Francisco    CA       94123 -9999.000000 -9999.000000   14155632393
3685   San Francisco    CA       94108 -9999.000000 -9999.000000   14150593281
4362   San Francisco    CA       94102 -9999.000000 -9999.000000   14150318683
4429   San Francisco    CA       94105 -9999.000000 -9999.000000   14150868663
4430   San Francisco    CA       94103 -9999.000000 -9999.000000   14150868663
```

Step 2

[31]:
```
address_counts = bus["address"].value_counts()
address_counts.head(10)
```

[31]:
```
Off The Grid              39
428 11th St               34
3251 20th Ave             17
2948 Folsom St            17
Pier 41                   16
103 Horne Ave             14
24 Willie Mays Plaza      13
Off the Grid              11
1 United Nations Plaza    10
2948 Folsom St.           10
Name: address, dtype: int64
```

[32]:
```
ok.grade("q2cii");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests
```

```
-----------------------------------------------------------------------
Test summary
    Passed: 2
    Failed: 0
[oooooooooook] 100.0% passed
```

3.4 Question 2d

Based on the above calculations answer each of the following questions by filling the value in the variable.

1. What does each record represent?

2. What is the minimal primary key?

[33]:
```
# What does each record represent?  Valid answers are:
#    "One location of a restaurant."
#    "A chain of restaurants."
#    "A city block."
q2d_part1 =  "One location of a restaurant."

# What is the minimal primary key? Valid answers are:
#    "bid"
#    "bid, name"
#    "bid, name, address"
q2d_part2 = "bid"
```

[34]:
```
ok.grade("q2d");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

-----------------------------------------------------------------------
Test summary
    Passed: 2
    Failed: 0
[oooooooooook] 100.0% passed
```

---

4 3: Cleaning the Business Data Postal Codes

The business data contains postal code information that we can use to aggregate the ratings over regions of the city. Let's examine and clean the postal code field. The postal code (sometimes also called a ZIP code) partitions the city into regions:

4.1 Question 3a

How many restaurants are in each ZIP code?

In the cell below, create a **series** where the index is the postal code and the value is the number of records with that postal code in descending order of count. You may need to use `groupby()`, `size()`, or `value_counts()`. Do you notice any odd/invalid zip codes?

[35]:
```
zip_counts = bus["postal_code"].value_counts()
print(zip_counts.to_string())
```

```
94103        562
94110        555
94102        456
94107        408
94133        398
94109        382
94111        259
94122        255
94105        249
94118        231
94115        230
94108        229
94124        218
94114        200
-9999        194
94112        192
94117        189
94123        177
94121        157
94104        142
94132        132
94116         97
94158         90
94134         82
94127         67
94131         49
94130          8
94143          5
CA             2
```

```
94188           2
94301           2
94013           2
94101           2
94129           1
941033148       1
941             1
94080           1
92672           1
94014           1
94621           1
94105-2907      1
64110           1
94602           1
Ca              1
95133           1
95122           1
00000           1
941102019       1
95109           1
94102-5917      1
94120           1
95112           1
95105           1
94123-3106      1
95132           1
94518           1
94544           1
94117-3504      1
94105-1420      1
94901           1
94124-1917      1
94122-1909      1
95117           1
```

[36]:
```
ok.grade("q3a");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 3
    Failed: 0
[oooooooooook] 100.0% passed
```

[37]:
```
type(bus["postal_code"][0])
```

[37]:
```
str
```

[38]:
```
bus["postal_code"][2]
```

[38]:
```
'94103'
```

[39]:
```
bus["postal_code"].loc[:5]
```

[39]:
```
0    94110
1    94133
2    94103
3    94124
4    94124
5    94123
Name: postal_code, dtype: object
```

4.2 Question 3b

Answer the following questions about the `postal_code` column in the `bus` dataframe.

1. The ZIP code column is which of the following type of data:
    1. Quantitative Continuous
    2. Quantitative Discrete
    3. Qualitative Ordinal
    4. Qualitative Nominal

2. What Python data type is used to represent a ZIP code?

Note: ZIP codes and postal codes are the same thing.

Please write your answers in the variables below:

[40]:
```
# The ZIP code column is which of the following type of data:
#   "Quantitative Continuous"
#   "Quantitative Discrete"
#   "Qualitative Ordinal"
#   "Qualitative Nominal"
q3b_part1 = "Qualitative Nominal"

# What Python data type is used to represent a ZIP code?
#    "str"
#    "int"
#    "bool"
#    "float"
```

```
q3b_part2 = "str"
```

[41]:
```
ok.grade("q3b");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 2
    Failed: 0
[oooooooooook] 100.0% passed
```

4.3 Question 3c

In question 3a we noticed a large number of potentially invalid ZIP codes (e.g., "CA"). These are likely due to data entry errors. To get a better understanding of the potential errors in the zip codes we will:

1. Import a list of valid San Francisco ZIP codes by using `pd.read_json` to load the file `data/sf_zipcodes.json` and extract a **series** of type `str` containing the valid ZIP codes. *Hint: set* ***`dtype`*** *when invoking* ***`read_json`****.*
2. Construct a DataFrame containing only the businesses which DO NOT have valid ZIP codes. You will probably want to use the `Series.isin` function.

Step 1

[42]:
```
valid_zips = pd.read_json("data/sf_zipcodes.json", dtype=object)["zip_codes"]
print(valid_zips.to_string())
```

```
0     94102
1     94103
2     94104
3     94105
4     94107
5     94108
6     94109
7     94110
8     94111
9     94112
10    94114
11    94115
12    94116
13    94117
14    94118
```

```
15    94119
16    94120
17    94121
18    94122
19    94123
20    94124
21    94125
22    94126
23    94127
24    94128
25    94129
26    94130
27    94131
28    94132
29    94133
30    94134
31    94137
32    94139
33    94140
34    94141
35    94142
36    94143
37    94144
38    94145
39    94146
40    94147
41    94151
42    94158
43    94159
44    94160
45    94161
46    94163
47    94164
48    94172
49    94177
50    94188
```

[43]:
```
ok.grade("q3ci");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 2
    Failed: 0
[oooooooooook] 100.0% passed
```

Step 2

[44]:
```
invalid_zip_bus = bus[~bus["postal_code"].isin(valid_zips)]
```

[45]:
```
ok.grade("q3cii");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 2
    Failed: 0
[ooooooooook] 100.0% passed
```

[46]:
```
invalid_zip_bus
```

[46]:

	bid	name	address	city	state	postal_code	latitude	longitude	phone_number
22	100126	Lamas Peruvian Food Truck	Private Location	San Francisco	CA	-9999	-9999.0	-9999.0	-9999
68	100417	COMPASS ONE, LLC	1 MARKET ST. FL	San Francisco	CA	94105-1420	-9999.0	-9999.0	14154324000
96	100660	TEAPENTER	1518 IRVING ST	San Francisco	CA	94122-1909	-9999.0	-9999.0	14155868318
109	100781	LE CAFE DU SOLEIL	200 FILLMORE ST	San Francisco	CA	94117-3504			
144	101084	Deli North 200	1 Warriors Way Level 300 North East	San Francisco	CA	94518			
…	…	…	…	…	…	…			
6173	99369	HOTEL BIRON	45 ROSE ST	San Francisco	CA	94102-5917			
6174	99376	Mashallah Halal Food truck Ind	Off The Grid	San Francisco	CA	-9999			
6199	99536	FAITH SANDWICH #2	560 MISSION ST	San Francisco	CA	94105-2907			
6204	99681	Twister	660 East Gish Rd	San Francisco	CA	95112			
6241	99819	CHESTNUT DINER	1312 CHESTNUT ST	San Francisco	CA	94123-3106			

```
109    -9999.0    -9999.0   14155614215
144    -9999.0    -9999.0         -9999
…        …        …             …
6173   -9999.0    -9999.0   14155700403
6174   -9999.0    -9999.0         -9999
6199   -9999.0    -9999.0   14155256783
6204   -9999.0    -9999.0         -9999
6241   -9999.0    -9999.0   14155846236

[230 rows x 9 columns]
```

[47]:
```
invalid_zip_bus[invalid_zip_bus['postal_code'].str.len() >= 5]
```

[47]:
```
          bid                            name  \
22    100126       Lamas Peruvian Food Truck
68    100417                 COMPASS ONE, LLC
96    100660                        TEAPENTER
109   100781               LE CAFE DU SOLEIL
144   101084                  Deli North 200
…      …                              …
6173   99369                      HOTEL BIRON
6174   99376  Mashallah Halal Food truck Ind
6199   99536               FAITH SANDWICH #2
6204   99681                         Twister
6241   99819                 CHESTNUT DINER

                                  address           city state postal_code  \
22                       Private Location  San Francisco    CA       -9999
68                        1 MARKET ST. FL  San Francisco    CA  94105-1420
96                         1518 IRVING ST  San Francisco    CA  94122-1909
109                       200 FILLMORE ST  San Francisco    CA  94117-3504
144   1 Warriors Way Level 300 North East  San Francisco    CA       94518
…                                      …           …   …         …
6173                          45 ROSE ST   San Francisco    CA  94102-5917
6174                        Off The Grid   San Francisco    CA       -9999
6199                       560 MISSION ST  San Francisco    CA  94105-2907
6204                     660 East Gish Rd  San Francisco    CA       95112
6241                     1312 CHESTNUT ST  San Francisco    CA  94123-3106

      latitude  longitude  phone_number
22     -9999.0    -9999.0         -9999
68     -9999.0    -9999.0   14154324000
96     -9999.0    -9999.0   14155868318
109    -9999.0    -9999.0   14155614215
144    -9999.0    -9999.0         -9999
…        …        …             …
6173   -9999.0    -9999.0   14155700403
```

```
6174   -9999.0    -9999.0         -9999
6199   -9999.0    -9999.0   14155256783
6204   -9999.0    -9999.0         -9999
6241   -9999.0    -9999.0   14155846236

[226 rows x 9 columns]
```

[48]:
```
missing = bus[~bus["postal_code"].isin(valid_zips)]
```

[49]:
```
missing["address"].value_counts()
```

[49]:
```
Off The Grid                                39
Off the Grid                                10
OFF THE GRID                                 4
OTG                                          4
Approved Locations                           3
                                            ..
1400 Stockton St                             1
1552 Ocean Ave                               1
24 Willie Mays Pl View Sect 320 Rm 5319      1
TFF Event Operations                         1
833 Bryant St                                1
Name: address, Length: 170, dtype: int64
```

[50]:
```
missing.take([0], axis=1)
```

[50]:
```
          bid
22     100126
68     100417
96     100660
109    100781
144    101084
…       …
6173    99369
6174    99376
6199    99536
6204    99681
6241    99819

[230 rows x 1 columns]
```

[51]:
```
missing.take([0],axis=1)["bid"]
```

[51]:
```
22      100126
68      100417
96      100660
109     100781
```

```
144     101084
         …
6173     99369
6174     99376
6199     99536
6204     99681
6241     99819
Name: bid, Length: 230, dtype: int64
```

4.4 Question 3d

In the previous question, many of the businesses had a common invalid postal code that was likely used to code a MISSING postal code. Do they all share a potentially "interesting address"?

In the following cell, construct a **series** that counts the number of businesses at each `address` that have this single likely MISSING postal code value. Order the series in descending order by count.

After examining the output. Answer the following question by filling in the appropriate variable. If we were to drop businesses with MISSING postal code values would a particular class of business be affected? If you are unsure try to search the web for the most common addresses.

[52]:
```
missing_val = invalid_zip_bus[invalid_zip_bus["postal_code"].str.
 ↪contains('-9999')]
missing_zip_address_count = missing_val["address"].value_counts()
missing_zip_address_count.head(35)
```

[52]:
```
Off The Grid                          39
Off the Grid                          10
OTG                                    4
OFF THE GRID                           3
Approved Private Locations             3
Approved Locations                     3
Treasure Island                        2
Justin Herman Plaza                    2
428 11th St                            2
3200 24th St                           1
6134 Geary Blvd                        1
550 A Gene Friend Way                  1
400 California                         1
625 Clement St                         1
55 Stockton St                         1
3861 24th St                           1
699 Avenue of the Palms                1
550 D Gene Friend Way                  1
203 Parnassus Ave                      1
3611 18th St                           1
```

```
Approved private locations                    1
855 Bush St                                   1
3914 Judah St                                 1
2 Marina Blvd Fort Mason                      1
450 Church St                                 1
Various Farmers Markets                       1
66 Kearny St                                  1
24 Willie Mays Pl Field Level Rm 1.11.11      1
2399 Van Ness Ave                             1
1605 Jerrold Ave                              1
3109 24th St                                  1
2826 Jones St                                 1
2462 San Bruno Ave                            1
2351 Mission St                               1
2277 Shafter Ave                              1
Name: address, dtype: int64
```

[53]:
```
ok.grade("q3d");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 3
    Failed: 0
[ooooooooook] 100.0% passed
```

[54]:
```
len(missing_zip_address_count)
```

[54]:
```
135
```

[55]:
```
missing_zip_address_count[0]
```

[55]:
```
39
```

[56]:
```
missing_not9 = invalid_zip_bus[~invalid_zip_bus["postal_code"].str.
 ↪contains('-9999')]
missing_zip_address_count = missing_not9["postal_code"].value_counts()
missing_zip_address_count.head(50)
```

[56]:
```
94301    2
94101    2
94013    2
CA       2
95109    1
```

```
92672         1
941033148     1
94122-1909    1
95112         1
94117-3504    1
94901         1
95132         1
94124-1917    1
95117         1
64110         1
94621         1
94080         1
94544         1
94602         1
95122         1
94518         1
95133         1
00000         1
95105         1
94105-2907    1
941102019     1
94123-3106    1
941           1
94105-1420    1
Ca            1
94102-5917    1
94014         1
Name: postal_code, dtype: int64
```

4.5 Question 3e

True or False: *If we were to drop businesses with MISSING postal code values a particular class of business will be affected.*

[57]:
```
# True or False:
#  If we were to drop businesses with MISSING postal code values
#   a particular class of business be affected.
q3d_true_or_false = True
```

[58]:
```
ok.grade("q3e");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
```

```
Test summary
    Passed: 1
    Failed: 0
[ooooooooook] 100.0% passed
```

[59]:
```
bus
```

[59]:

	bid	name	address	\
0	1000	HEUNG YUEN RESTAURANT	3279 22nd St	
1	100010	ILLY CAFFE SF_PIER 39	PIER 39 K-106-B	
2	100017	AMICI'S EAST COAST PIZZERIA	475 06th St	
3	100026	LOCAL CATERING	1566 CARROLL AVE	
4	100030	OUI OUI! MACARON	2200 JERROLD AVE STE C	
…	…	…	…	
6248	99948	SUSIECAKES BAKERY	3509 CALIFORNIA ST	
6249	99988	HINODEYA SOMA	303 02nd ST STE 102	
6250	99991	TON TON	422 GEARY ST	
6251	99992	URBAN EXPRESS KITCHENS LLC	475 06th ST	
6252	99993	THE BRIXTON SOUTH	701 02nd St	

	city	state	postal_code	latitude	longitude	phone_number
0	San Francisco	CA	94110	37.755282	-122.420493	-9999
1	San Francisco	CA	94133	-9999.000000	-9999.000000	14154827284
2	San Francisco	CA	94103	-9999.000000	-9999.000000	14155279839
3	San Francisco	CA	94124	-9999.000000	-9999.000000	14155860315
4	San Francisco	CA	94124	-9999.000000	-9999.000000	14159702675
…	…	…	…	…	…	…
6248	San Francisco	CA	94118	-9999.000000	-9999.000000	14150452253
6249	San Francisco	CA	94107	-9999.000000	-9999.000000	-9999
6250	San Francisco	CA	94102	-9999.000000	-9999.000000	14155531280
6251	San Francisco	CA	94103	-9999.000000	-9999.000000	14150368085
6252	San Francisco	CA	94102	-9999.000000	-9999.000000	14158315871

[6253 rows x 9 columns]

4.6 Question 3f

Examine the `invalid_zip_bus` dataframe we computed above and look at the businesses that DO NOT have the special MISSING ZIP code value. Some of the invalid postal codes are just the full 9 digit code rather than the first 5 digits. Create a new column named `postal5` in the original `bus` dataframe which contains only the first 5 digits of the `postal_code` column. Finally, for any of the `postal5` ZIP code entries that were not a valid San Fransisco ZIP Code (according to `valid_zips`) set the entry to `None`.

[60]:
```
bus['postal5'] = None
bus['postal5'] = bus['postal_code'].str[:5]
bus['postal5'].loc[bus['postal5'].isin(valid_zips) == False] = None
#Checking the corrected postal5 column
bus.loc[invalid_zip_bus.index, ['bid', 'name', 'postal_code', 'postal5']]
```

```
/srv/conda/envs/data100/lib/python3.6/site-packages/pandas/core/indexing.py:670:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  self._setitem_with_indexer(indexer, value)
```

[60]:

	bid	name	postal_code	postal5
22	100126	Lamas Peruvian Food Truck	-9999	None
68	100417	COMPASS ONE, LLC	94105-1420	94105
96	100660	TEAPENTER	94122-1909	94122
109	100781	LE CAFE DU SOLEIL	94117-3504	94117
144	101084	Deli North 200	94518	None
…	…	…	…	…
6173	99369	HOTEL BIRON	94102-5917	94102
6174	99376	Mashallah Halal Food truck Ind	-9999	None
6199	99536	FAITH SANDWICH #2	94105-2907	94105
6204	99681	Twister	95112	None
6241	99819	CHESTNUT DINER	94123-3106	94123

[230 rows x 4 columns]

[61]:
```
ok.grade("q3f");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 4
    Failed: 0
[oooooooooook] 100.0% passed
```

---

5 4: Investigate the Inspection Data

Let's now turn to the inspection DataFrame. Earlier, we found that `ins` has 4 columns named `iid`, `score`, `date` and `type`. In this section, we determine the granularity of `ins` and investigate the

kinds of information provided for the inspections.

Let's start by looking again at the first 5 rows of `ins` to see what we're working with.

[62]:
```
ins.head(10)
```

[62]:

	iid	date	score	type
0	100010_20190329	03/29/2019 12:00:00 AM	-1	New Construction
1	100010_20190403	04/03/2019 12:00:00 AM	100	Routine - Unscheduled
2	100017_20190417	04/17/2019 12:00:00 AM	-1	New Ownership
3	100017_20190816	08/16/2019 12:00:00 AM	91	Routine - Unscheduled
4	100017_20190826	08/26/2019 12:00:00 AM	-1	Reinspection/Followup
5	100017_20190912	09/12/2019 12:00:00 AM	-1	Reinspection/Followup
6	100026_20190418	04/18/2019 12:00:00 AM	-1	New Ownership
7	100030_20190612	06/12/2019 12:00:00 AM	-1	New Ownership
8	100030_20190826	08/26/2019 12:00:00 AM	-1	New Ownership
9	100036_20190325	03/25/2019 12:00:00 AM	-1	Structural Inspection

[63]:
```
ins["iid"].is_unique
```

[63]:
```
True
```

5.1 Question 4a

The column `iid` probably corresponds to an inspection id. Is it a primary key? Write an expression (line of code) that evaluates to 'True' or 'False' based on whether all the values are unique.

[64]:
```
is_ins_iid_a_primary_key = True
```

[65]:
```
ok.grade("q4a");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[oooooooooook] 100.0% passed
```

[66]:
```
ins
```

[66]:

	iid	date	score	type
0	100010_20190329	03/29/2019 12:00:00 AM	-1	New Construction
1	100010_20190403	04/03/2019 12:00:00 AM	100	Routine - Unscheduled

```
2      100017_20190417  04/17/2019 12:00:00 AM     -1          New Ownership
3      100017_20190816  08/16/2019 12:00:00 AM     91  Routine - Unscheduled
4      100017_20190826  08/26/2019 12:00:00 AM     -1  Reinspection/Followup
…                 …                       …     …                      …
26658     999_20180924  09/24/2018 12:00:00 AM     -1    Routine - Scheduled
26659     999_20181102  11/02/2018 12:00:00 AM     -1  Reinspection/Followup
26660     999_20190909  09/09/2019 12:00:00 AM     80  Routine - Unscheduled
26661      99_20171207  12/07/2017 12:00:00 AM     82  Routine - Unscheduled
26662      99_20180808  08/08/2018 12:00:00 AM     84  Routine - Unscheduled

[26663 rows x 4 columns]
```

[67]:
```
bus[bus['bid'] == 99]
```

[67]:
```
      bid                          name      address           city state  \
6117   99  J & M A-1 CAFE RESTAURANT LLC  779 Clay St  San Francisco    CA

     postal_code   latitude   longitude  phone_number postal5
6117       94108  37.794293 -122.405967         -9999   94108
```

[68]:
```
bus[bus['bid'] == 999]
```

[68]:
```
      bid                name       address           city state postal_code  \
6247  999  SERRANO'S PIZZA II  3274 21st St  San Francisco    CA       94110

       latitude   longitude  phone_number postal5
6247  37.756997 -122.420534   14155691615   94110
```

5.2 Question 4b

The column `iid` appears to be the composition of two numbers and the first number looks like a business id.

Part 1.: Create a new column called `bid` in the `ins` dataframe containing just the business id. You will want to use `ins['iid'].str` operations to do this. Also be sure to convert the type of this column to `int`

Part 2.: Then compute how many values in this new column are also valid business ids (appear in the `bus['bid']` column). This is verifying a foreign key relationship. Consider using the `pd.Series.isin` function.

Part 3.: Answer True or False, `ins['bid']` is a foreign key reference to `bus['bid']`.

No python `for` loops or list comprehensions required!

Part 1

[69]:
```
bid = ins['iid'].str.split('_', expand=True)
ins['bid'] = bid[0]
ins['bid'] = pd.to_numeric(ins['bid'], errors = 'ignore')
ins
```

[69]:

	iid	date	score	type	\
0	100010_20190329	03/29/2019 12:00:00 AM	-1	New Construction	
1	100010_20190403	04/03/2019 12:00:00 AM	100	Routine - Unscheduled	
2	100017_20190417	04/17/2019 12:00:00 AM	-1	New Ownership	
3	100017_20190816	08/16/2019 12:00:00 AM	91	Routine - Unscheduled	
4	100017_20190826	08/26/2019 12:00:00 AM	-1	Reinspection/Followup	
…	…	…	…	…	
26658	999_20180924	09/24/2018 12:00:00 AM	-1	Routine - Scheduled	
26659	999_20181102	11/02/2018 12:00:00 AM	-1	Reinspection/Followup	
26660	999_20190909	09/09/2019 12:00:00 AM	80	Routine - Unscheduled	
26661	99_20171207	12/07/2017 12:00:00 AM	82	Routine - Unscheduled	
26662	99_20180808	08/08/2018 12:00:00 AM	84	Routine - Unscheduled	

	bid
0	100010
1	100010
2	100017
3	100017
4	100017
…	…
26658	999
26659	999
26660	999
26661	99
26662	99

[26663 rows x 5 columns]

[70]:
```
ok.grade("q4bi");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 3
    Failed: 0
[oooooooooo] 100.0% passed
```

Part 2

[71]:
```
invalid_bid_count = sum(~ins['bid'].isin(bus['bid']))
invalid_bid_count
```

[71]:
```
0
```

[72]:
```
ok.grade("q4bii");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[oooooooooook] 100.0% passed
```

Part 3

[73]:
```
# True or False: The column ins['bid'] is a foreign key
#   referencing the bus['bid'] primary key.
q4b_is_foreign_key = True
```

[74]:
```
ok.grade("q4biii");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[oooooooooook] 100.0% passed
```

[75]:
```
ins['date'][0]
```

[75]:
```
'03/29/2019 12:00:00 AM'
```

[76]:
```
ins['timestamp'] = pd.to_datetime(ins['date'])
ins['timestamp']
```

[76]:
```
0       2019-03-29
1       2019-04-03
2       2019-04-17
3       2019-08-16
4       2019-08-26
           …
```

```
26658   2018-09-24
26659   2018-11-02
26660   2019-09-09
26661   2017-12-07
26662   2018-08-08
Name: timestamp, Length: 26663, dtype: datetime64[ns]
```

[77]:
```
ins[~ins['bid'].isin(bus['bid'])]
```

[77]:
```
Empty DataFrame
Columns: [iid, date, score, type, bid, timestamp]
Index: []
```

[78]:
```
bus['bid']
```

[78]:
```
0         1000
1       100010
2       100017
3       100026
4       100030
         ...
6248     99948
6249     99988
6250     99991
6251     99992
6252     99993
Name: bid, Length: 6253, dtype: int64
```

5.3 Question 4c

What if we are interested in a time component of the inspection data? We need to examine the date column of each inspection.

Part 1: What is the type of the individual `ins['date']` entries. You may want to grab the very first entry and use the `type` function in python.

Part 2: Use `pd.to_datetime` to create a new `ins['timestamp']` column containing of `pd.Timestamp` objects. These will allow us to do more date manipulation.

Part 3: What are the earliest and latest dates in our inspection data? *Hint: you can use* ***`min`*** *and* ***`max`*** *on dates of the correct type.*

Part 4: We probably want to examine the inspections by year. Create an additional `ins['year']` column containing just the year of the inspection. Consider using `pd.Series.dt.year` to do this.

No python `for` loops or list comprehensions required!

Part 1

[79]:
```
ins_date_type = type(ins['date'][0])
ins_date_type
```

[79]:
```
str
```

[80]:
```
ok.grade("q4ci");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[oooooooooook] 100.0% passed
```

Part 2

[81]:
```
ins['timestamp'] = pd.to_datetime(ins['date'])
```

[82]:
```
ok.grade("q4cii");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[oooooooooook] 100.0% passed
```

Part 3

[83]:
```
earliest_date = min(ins['timestamp'])
latest_date = max(ins['timestamp'])

print("Earliest Date:", earliest_date)
print("Latest Date:", latest_date)
```

```
Earliest Date: 2016-10-04 00:00:00
Latest Date: 2019-11-28 00:00:00
```

[84]:
```
ok.grade("q4ciii");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
```

```
Test summary
    Passed: 2
    Failed: 0
[oooooooooook] 100.0% passed
```

Part 4

[85]:
```
ins['year'] = ins['timestamp'].dt.year
```

[86]:
```
ok.grade("q4civ");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[ooooooooook] 100.0% passed
```

[87]:
```
ins.sort_values(by=['year'],axis=0, ascending=True).head(300)
```

[87]:

	iid	date	score	type	\
13331	7214_20161228	12/28/2016 12:00:00 AM	94	Routine - Unscheduled	
20062	88684_20161207	12/07/2016 12:00:00 AM	-1	New Construction	
6456	3935_20161230	12/30/2016 12:00:00 AM	98	Routine - Unscheduled	
20060	88676_20161109	11/09/2016 12:00:00 AM	-1	Reinspection/Followup	
20059	88676_20161102	11/02/2016 12:00:00 AM	86	Routine - Unscheduled	
…	…	…	…	…	
18999	86643_20161117	11/17/2016 12:00:00 AM	81	Routine - Unscheduled	
18991	86590_20161209	12/09/2016 12:00:00 AM	100	Routine - Unscheduled	
7800	539_20161031	10/31/2016 12:00:00 AM	96	Routine - Unscheduled	
7804	5409_20161005	10/05/2016 12:00:00 AM	100	Routine - Unscheduled	
18986	86587_20161209	12/09/2016 12:00:00 AM	94	Routine - Unscheduled	

	bid	timestamp	year
13331	7214	2016-12-28	2016
20062	88684	2016-12-07	2016
6456	3935	2016-12-30	2016
20060	88676	2016-11-09	2016
20059	88676	2016-11-02	2016
…	…	…	…
18999	86643	2016-11-17	2016
18991	86590	2016-12-09	2016
7800	539	2016-10-31	2016
7804	5409	2016-10-05	2016

```
18986  86587 2016-12-09  2016

[300 rows x 7 columns]
```

5.4 Question 4d

What is the relationship between the type of inspection over the 2016 to 2019 timeframe?

Part 1

Construct the following table by 1. Using the `pivot_table` containing the number (`size`) of inspections for the given `type` and `year`. 1. Adding an extra `Total` column to the result using `sum` 1. Sort the results in descending order by the `Total`.

year

```
<th>2016</th>      <th>2017</th>      <th>2018</th>      <th>2019</th>      <th>Total</th>
```

Part 2

Based on the above analysis, which year appears to have had a lot of businesses in new buildings?

No python `for` loops or list comprehensions required!

Part 1

[88]:
```
ins_pivot = pd.pivot_table(ins, values =[], index=['type'], columns=['year'],
    aggfunc=len, fill_value=0, margins=False, dropna=False)
ins_pivot['Total'] = ins_pivot.sum(axis=1)
ins_pivot_sorted = ins_pivot.sort_values(by=['Total'], ascending=False)

ins_pivot_sorted
```

[88]:

year	2016	2017	2018	2019	Total
type					
Routine - Unscheduled	966	4057	4373	4681	14077
Reinspection/Followup	445	1767	1935	2292	6439
New Ownership	99	506	528	459	1592
Complaint	91	418	512	437	1458
New Construction	102	485	218	189	994
Non-inspection site visit	51	276	253	231	811
New Ownership - Followup	0	45	219	235	499
Structural Inspection	1	153	50	190	394
Complaint Reinspection/Followup	19	68	70	70	227
Foodborne Illness Investigation	1	29	50	35	115
Routine - Scheduled	0	9	8	29	46
Administrative or Document Review	2	1	1	0	4
Multi-agency Investigation	0	0	1	2	3
Special Event	0	3	0	0	3

```
Community Health Assessment           1     0     0     0      1
```

[89]:
```
ok.grade("q4di");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 2
    Failed: 0
[oooooooooook] 100.0% passed
```

Part 2

[90]:
```
year_of_new_construction = 2017
```

[91]:
```
ok.grade("q4dii");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[ooooooooook] 100.0% passed
```

[92]:
```
# ins_pivot = pd.pivot_table(ins, values =[], index=['type'], columns=['year'],
#     aggfunc=len, fill_value=0, margins=False, dropna=False)
# ins_pivot['Total'] = ins_pivot.sum(axis=1)
# ins_pivot_sorted = ins_pivot.sort_values(by=['Total'], ascending=False)


ins_missing_score_pivot = pd.pivot_table(ins, values=[], index=['type'],
    columns=[ins['score'] == -1], aggfunc=len,fill_value=0, dropna=False)

ins_missing_score_pivot['Total'] = ins_missing_score_pivot.sum(axis=1)

ins_missing_score_pivot.rename(columns={"score": "Missing Score"}, inplace=True)

# ins_missing_score_pivot.pivot(columns=["Missing Score"])

ins_missing_score_pivot_sorted = ins_missing_score_pivot.
 ↪sort_values(by=['Total'],
                                        ascending=False)
```

```
ins_missing_score_pivot_sorted
```

[92]:

score	False	True	Total
type			
Routine - Unscheduled	14031	46	14077
Reinspection/Followup	0	6439	6439
New Ownership	0	1592	1592
Complaint	0	1458	1458
New Construction	0	994	994
Non-inspection site visit	0	811	811
New Ownership - Followup	0	499	499
Structural Inspection	0	394	394
Complaint Reinspection/Followup	0	227	227
Foodborne Illness Investigation	0	115	115
Routine - Scheduled	0	46	46
Administrative or Document Review	0	4	4
Multi-agency Investigation	0	3	3
Special Event	0	3	3
Community Health Assessment	0	1	1

5.5 Question 4e

Let's examine the inspection scores `ins['score']`

[93]:
```
ins['score'].value_counts().head()
```

[93]:
```
-1      12632
 100     1993
 96      1681
 92      1260
 94      1250
Name: score, dtype: int64
```

There are a large number of inspections with the `'score'` of `-1`. These are probably missing values. Let's see what type of inspections have scores and which do not. Create the following dataframe using steps similar to the previous question.

You should observe that inspection scores appear only to be assigned to `Routine - Unscheduled` inspections.

Missing Score

```
<th>False</th>      <th>True</th>      <th>Total</th>    </tr>    <tr>      <th>type</th>
```

[94]:
```
ins_missing_score_pivot = pd.pivot_table(ins, values=[], index=['type'],
    columns=[ins['score'] == -1], aggfunc=len,fill_value=0, dropna=False)

ins_missing_score_pivot['Total'] = ins_missing_score_pivot.sum(axis=1)

ins_missing_score_pivot.columns.name = "Missing Score"

ins_missing_score_pivot_sorted = ins_missing_score_pivot.
 ↪sort_values(by=['Total'],
                                        ascending=False)
ins_missing_score_pivot_sorted
```

[94]:

Missing Score	False	True	Total
type			
Routine - Unscheduled	14031	46	14077
Reinspection/Followup	0	6439	6439
New Ownership	0	1592	1592
Complaint	0	1458	1458
New Construction	0	994	994
Non-inspection site visit	0	811	811
New Ownership - Followup	0	499	499
Structural Inspection	0	394	394
Complaint Reinspection/Followup	0	227	227
Foodborne Illness Investigation	0	115	115
Routine - Scheduled	0	46	46
Administrative or Document Review	0	4	4
Multi-agency Investigation	0	3	3
Special Event	0	3	3
Community Health Assessment	0	1	1

[95]:
```
ok.grade("q4e");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[oooooooooook] 100.0% passed
```

Notice that inspection scores appear only to be assigned to `Routine - Unscheduled` inspections. It is reasonable that for inspection types such as `New Ownership` and `Complaint` to have no associated inspection scores, but we might be curious why there are no inspection scores for the `Reinspection/Followup` inspection type.

---

6 5: Joining Data Across Tables

In this question we will start to connect data across mulitple tables. We will be using the `merge` function.

[96]:
```
ins
```

[96]:
```
                  iid                    date  score                  type  \
0      100010_20190329  03/29/2019 12:00:00 AM     -1       New Construction
1      100010_20190403  04/03/2019 12:00:00 AM    100  Routine - Unscheduled
2      100017_20190417  04/17/2019 12:00:00 AM     -1          New Ownership
3      100017_20190816  08/16/2019 12:00:00 AM     91  Routine - Unscheduled
4      100017_20190826  08/26/2019 12:00:00 AM     -1  Reinspection/Followup
…                 …                      …    …                     …
26658    999_20180924  09/24/2018 12:00:00 AM     -1    Routine - Scheduled
26659    999_20181102  11/02/2018 12:00:00 AM     -1  Reinspection/Followup
26660    999_20190909  09/09/2019 12:00:00 AM     80  Routine - Unscheduled
26661     99_20171207  12/07/2017 12:00:00 AM     82  Routine - Unscheduled
26662     99_20180808  08/08/2018 12:00:00 AM     84  Routine - Unscheduled

          bid  timestamp  year
0      100010 2019-03-29  2019
1      100010 2019-04-03  2019
2      100017 2019-04-17  2019
3      100017 2019-08-16  2019
4      100017 2019-08-26  2019
…       …        …    …
26658     999 2018-09-24  2018
26659     999 2018-11-02  2018
26660     999 2019-09-09  2019
26661      99 2017-12-07  2017
26662      99 2018-08-08  2018

[26663 rows x 7 columns]
```

6.1 Question 5a

Let's figure out which restaurants had the lowest scores. Let's start by creating a new dataframe called `ins_named`. It should be exactly the same as `ins`, except that it should have the name and address of every business, as determined by the `bus` dataframe.

Hint: Use the merge method to join the `ins` dataframe with the appropriate portion of the `bus` dataframe. See the official documentation on how to use `merge`.

Note: For quick reference, a pandas 'left' join keeps the keys from the left frame, so if ins is the left frame, all the keys from ins are kept and if a set of these keys don't have matches in the other

frame, the columns from the other frame for these "unmatched" key rows contains NaNs.

[97]:
```
ins_named = pd.merge(ins, bus.drop(columns=['city','state','postal_code',␣
↪'latitude',
            'longitude', 'phone_number','postal5']), how='left')
ins_named.head()
```

[97]:

	iid	date	score	type	\
0	100010_20190329	03/29/2019 12:00:00 AM	-1	New Construction	
1	100010_20190403	04/03/2019 12:00:00 AM	100	Routine - Unscheduled	
2	100017_20190417	04/17/2019 12:00:00 AM	-1	New Ownership	
3	100017_20190816	08/16/2019 12:00:00 AM	91	Routine - Unscheduled	
4	100017_20190826	08/26/2019 12:00:00 AM	-1	Reinspection/Followup	

	bid	timestamp	year	name	address
0	100010	2019-03-29	2019	ILLY CAFFE SF_PIER 39	PIER 39 K-106-B
1	100010	2019-04-03	2019	ILLY CAFFE SF_PIER 39	PIER 39 K-106-B
2	100017	2019-04-17	2019	AMICI'S EAST COAST PIZZERIA	475 06th St
3	100017	2019-08-16	2019	AMICI'S EAST COAST PIZZERIA	475 06th St
4	100017	2019-08-26	2019	AMICI'S EAST COAST PIZZERIA	475 06th St

[98]:
```
ok.grade("q5a");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 3
    Failed: 0
[oooooooooook] 100.0% passed
```

[99]:
```
ins_named[ins_named['score'] > 0].head(100)
```

[99]:

	iid	date	score	type	\
1	100010_20190403	04/03/2019 12:00:00 AM	100	Routine - Unscheduled	
3	100017_20190816	08/16/2019 12:00:00 AM	91	Routine - Unscheduled	
15	100041_20190520	05/20/2019 12:00:00 AM	83	Routine - Unscheduled	
20	100055_20190425	04/25/2019 12:00:00 AM	98	Routine - Unscheduled	
21	100055_20190912	09/12/2019 12:00:00 AM	82	Routine - Unscheduled	
..	…	…	…	…	
391	101102_20190906	09/06/2019 12:00:00 AM	96	Routine - Unscheduled	
394	101103_20190906	09/06/2019 12:00:00 AM	100	Routine - Unscheduled	
397	101104_20190906	09/06/2019 12:00:00 AM	100	Routine - Unscheduled	
401	101129_20190906	09/06/2019 12:00:00 AM	100	Routine - Unscheduled	
408	101160_20191003	10/03/2019 12:00:00 AM	91	Routine - Unscheduled	

```
        bid  timestamp  year                            name  \
1    100010 2019-04-03  2019           ILLY CAFFE SF_PIER 39
3    100017 2019-08-16  2019      AMICI'S EAST COAST PIZZERIA
15   100041 2019-05-20  2019                  UNCLE LEE CAFE
20   100055 2019-04-25  2019                   Twirl and Dip
21   100055 2019-09-12  2019                   Twirl and Dip
..      …        …   …                              …
391  101102 2019-09-06  2019         Hot Dog Bills South 200
394  101103 2019-09-06  2019  Tony's/ City Bistro South 200
397  101104 2019-09-06  2019         Big Nate BBQ South 200
401  101129 2019-09-06  2019                 Vendor Room 200
408  101160 2019-10-03  2019                   Banh Mi & Roll

                                address
1                        PIER 39  K-106-B
3                             475 06th St
15                          3608 BALBOA ST
20           335 Martin Luther King Jr. Dr
21           335 Martin Luther King Jr. Dr
..                                     …
391        1 Warriors Way Level 300 South
394        1 Warriors Way Level 300 South
397       1 Warriors Way  Level 300 South
401  1 Warriors Way Level 300 South West
408         2609 SAN BRUNO AVE STE      A

[100 rows x 9 columns]
```

[100]:
```
grouped = ins_named[ins_named['score'] > 0].groupby(['bid','name'],␣
↪as_index=False).agg(np.median)
grouped.sort_values(by='score', ascending=True)
```

[100]:
```
        bid                                  name  score    year
3876  84590                        Chaat Corner   54.0  2018.0
4564  90622                     Taqueria Lolita   57.0  2018.0
4990  94351                         VBowls LLC   58.0  2019.0
2719  69282       New Jumbo Seafood Restaurant   60.5  2018.5
222    1154                SUNFLOWER RESTAURANT   63.5  2018.5
…       …                                   …    …     …
4131  86790                   Sightglass Coffee  100.0  2018.0
5081  94906                94906 Gotham Kitchen  100.0  2018.0
5083  94912          94912 Cognac Kitchen & Bar  100.0  2019.0
2920  71753                        Sunrise Deli  100.0  2017.0
448    2261  Ritz-Carlton SF - Employee Cafeteria  100.0  2018.0

[5724 rows x 4 columns]
```

6.2 Question 5b

Let's look at the 20 businesses with the lowest **median** score. Order your results by the median score followed by the business id to break ties. The resulting table should look like:

Hint: You may find the ***as_index*** *argument important*

<th>bid</th> <th>name</th> <th>median score</th> </tr> </thead> <tbody> <t

[101]:
```
#cond1: score > 0, groupby ['bid','name'], agg(np.median)
twenty_lowest_scoring = ins_named[ins_named['score'] > 0].
 ↪groupby(['bid','name'],
                                as_index=False).agg(np.median)
#cond2: sort ascending
twenty_lowest_scoring = twenty_lowest_scoring.sort_values(by='score',␣
 ↪ascending=True)
#rename score
twenty_lowest_scoring.rename(columns={'score': 'median score'}, inplace=True)
#drop year
twenty_lowest_scoring.drop(columns=['year'],inplace=True)
#limit 20 rows
twenty_lowest_scoring = twenty_lowest_scoring.head(20)
#print
twenty_lowest_scoring
```

[101]:

	bid	name	median score
3876	84590	Chaat Corner	54.0
4564	90622	Taqueria Lolita	57.0
4990	94351	VBowls LLC	58.0
2719	69282	New Jumbo Seafood Restaurant	60.5
222	1154	SUNFLOWER RESTAURANT	63.5
1991	39776	Duc Loi Supermarket	64.0
2734	69397	Minna SF Group LLC	64.0
4870	93150	Chez Beesen	64.0
4911	93502	Smoky Man	64.0
3291	78328	Golden Wok	64.0
5510	98995	Vallarta's Taco Bar	64.0
2890	71310	Golden King Vietnamese Restaurant	64.5
1457	10877	CHINA FIRST INC.	64.5
4352	89070	Lafayette Coffee Shop	64.5
505	2542	PETER D'S RESTAURANT	65.0
2874	71008	House of Pancakes	65.0
818	3862	IMPERIAL GARDEN SEAFOOD RESTAURANT	66.0
2141	61427	Nick's Foods	66.0
2954	72176	Wolfes Lunch	66.0
4367	89141	Cha Cha Cha on Mission	66.5

[102]:
```
ok.grade("q5b");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 2
    Failed: 0
[oooooooooook] 100.0% passed
```

[103]:
```
vio
```

[103]:

	description	risk_category	vid
0	Consumer advisory not provided for raw or unde…	Moderate Risk	103128
1	Contaminated or adulterated food	High Risk	103108
2	Discharge from employee nose mouth or eye	Moderate Risk	103117
3	Employee eating or smoking	Moderate Risk	103118
4	Food in poor condition	Moderate Risk	103123
..	…	…	…
60	Unclean unmaintained or improperly constructed…	Low Risk	103152
61	Unpermitted food facility	Low Risk	103158
62	Unsanitary employee garments hair or nails	Low Risk	103136
63	Wiping cloths not clean or properly stored or …	Low Risk	103149
64	Worker safety hazards	Low Risk	103159

[65 rows x 3 columns]

[104]:
```
ins2vio
```

[104]:

	iid	vid
0	97975_20190725	103124
1	85986_20161011	103114
2	95754_20190327	103124
3	77005_20170429	103120
4	4794_20181030	103138
…	…	…
40205	76958_20180919	103119
40206	80305_20190411	103149
40207	80233_20190417	103133
40208	100216_20190321	103119
40209	79430_20190418	103109

[40210 rows x 2 columns]

6.3 Question 5c

Let's now examine the descriptions of violations for inspections with `score > 0` and `score < 65`. Construct a **Series** indexed by the `description` of the violation from the `vio` table with the value being the number of times that violation occured for inspections with the above score range. Sort the results in descending order of the count.

The first few entries should look like:

```
Unclean or unsanitary food contact surfaces                    43
High risk food holding temperature                             42
Unclean or degraded floors walls or ceilings                   40
Unapproved or unmaintained equipment or utensils               39
```

You will need to use `merge` twice.

[105]:
```
m1 = pd.merge(ins.drop(columns=['date', 'bid', 'timestamp', 'year', 'type']),␣
 ↪ins2vio, how='left')
m2 = pd.merge(m1, vio, how='left')2
low_score_violations = m2[(m2['score'] > 0) & (m2['score'] < 65)].
 ↪groupby(['description']).size()
low_score_violations.sort_values(ascending=False, inplace=True)
low_score_violations.head(20)
```

[105]:
```
description
Unclean or unsanitary food contact surfaces                                  43
High risk food holding temperature                                           42
Unclean or degraded floors walls or ceilings                                 40
Unapproved or unmaintained equipment or utensils                             39
Foods not protected from contamination                                       37
High risk vermin infestation                                                 37
Inadequate food safety knowledge or lack of certified food safety manager    35
Inadequate and inaccessible handwashing facilities                           35
Improper thawing methods                                                     30
Unclean hands or improper use of gloves                                      27
Improper cooling methods                                                     25
Unclean nonfood contact surfaces                                             21
Inadequately cleaned or sanitized food contact surfaces                      20
Improper food storage                                                        20
Contaminated or adulterated food                                             18
Moderate risk vermin infestation                                             15
Permit license or inspection report not posted                               13
Moderate risk food holding temperature                                       13
Food safety certificate or food handler card not available                   12
Improper storage use or identification of toxic substances                   10
dtype: int64
```

[105]:
```
ok.grade("q5c");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

----------------------------------------------------------------------
Test summary
    Passed: 3
    Failed: 0
[ooooooooook] 100.0% passed
```

[106]:
```
bus
```

[106]:
```
          bid                          name                 address  \
0       1000        HEUNG YUEN RESTAURANT             3279 22nd St
1     100010        ILLY CAFFE SF_PIER 39        PIER 39  K-106-B
2     100017  AMICI'S EAST COAST PIZZERIA              475 06th St
3     100026               LOCAL CATERING        1566 CARROLL AVE
4     100030            OUI OUI! MACARON  2200 JERROLD AVE STE C
…       …                           …                    …
6248   99948           SUSIECAKES BAKERY      3509 CALIFORNIA ST
6249   99988               HINODEYA SOMA     303 02nd ST STE 102
6250   99991                    TON TON             422 GEARY ST
6251   99992   URBAN EXPRESS KITCHENS LLC              475 06th ST
6252   99993           THE BRIXTON SOUTH              701 02nd St

                city state postal_code     latitude    longitude  phone_number  \
0     San Francisco    CA       94110    37.755282  -122.420493          -9999
1     San Francisco    CA       94133 -9999.000000 -9999.000000   14154827284
2     San Francisco    CA       94103 -9999.000000 -9999.000000   14155279839
3     San Francisco    CA       94124 -9999.000000 -9999.000000   14155860315
4     San Francisco    CA       94124 -9999.000000 -9999.000000   14159702675
…               …   …          …           …           …            …
6248  San Francisco    CA       94118 -9999.000000 -9999.000000   14150452253
6249  San Francisco    CA       94107 -9999.000000 -9999.000000          -9999
6250  San Francisco    CA       94102 -9999.000000 -9999.000000   14155531280
6251  San Francisco    CA       94103 -9999.000000 -9999.000000   14150368085
6252  San Francisco    CA       94102 -9999.000000 -9999.000000   14158315871

     postal5
0      94110
1      94133
2      94103
3      94124
4      94124
…        …
6248   94118
6249   94107
```

```
6250   94102
6251   94103
6252   94102

[6253 rows x 10 columns]
```

[107]:
```
ins
```

[107]:
```
                   iid                    date  score                   type  \
0      100010_20190329  03/29/2019 12:00:00 AM     -1       New Construction
1      100010_20190403  04/03/2019 12:00:00 AM    100  Routine - Unscheduled
2      100017_20190417  04/17/2019 12:00:00 AM     -1          New Ownership
3      100017_20190816  08/16/2019 12:00:00 AM     91  Routine - Unscheduled
4      100017_20190826  08/26/2019 12:00:00 AM     -1  Reinspection/Followup
…                  …                      …    …                     …
26658     999_20180924  09/24/2018 12:00:00 AM     -1    Routine - Scheduled
26659     999_20181102  11/02/2018 12:00:00 AM     -1  Reinspection/Followup
26660     999_20190909  09/09/2019 12:00:00 AM     80  Routine - Unscheduled
26661      99_20171207  12/07/2017 12:00:00 AM     82  Routine - Unscheduled
26662      99_20180808  08/08/2018 12:00:00 AM     84  Routine - Unscheduled

          bid  timestamp  year
0      100010 2019-03-29  2019
1      100010 2019-04-03  2019
2      100017 2019-04-17  2019
3      100017 2019-08-16  2019
4      100017 2019-08-26  2019
…         …         …    …
26658     999 2018-09-24  2018
26659     999 2018-11-02  2018
26660     999 2019-09-09  2019
26661      99 2017-12-07  2017
26662      99 2018-08-08  2018

[26663 rows x 7 columns]
```

[108]:
```
vio
```

[108]:
```
                                        description  risk_category     vid
0   Consumer advisory not provided for raw or unde…  Moderate Risk  103128
1                     Contaminated or adulterated food      High Risk  103108
2           Discharge from employee nose mouth or eye  Moderate Risk  103117
3                           Employee eating or smoking  Moderate Risk  103118
4                               Food in poor condition  Moderate Risk  103123
..                                                 …             …     …
60  Unclean unmaintained or improperly constructed…       Low Risk  103152
61                           Unpermitted food facility       Low Risk  103158
```

```
62         Unsanitary employee garments hair or nails       Low Risk  103136
63  Wiping cloths not clean or properly stored or …       Low Risk  103149
64                              Worker safety hazards       Low Risk  103159

[65 rows x 3 columns]
```

[109]:
```
ins2vio
```

[109]:
```
                   iid     vid
0       97975_20190725  103124
1       85986_20161011  103114
2       95754_20190327  103124
3       77005_20170429  103120
4        4794_20181030  103138
…                 …     …
40205   76958_20180919  103119
40206   80305_20190411  103149
40207   80233_20190417  103133
40208  100216_20190321  103119
40209   79430_20190418  103109

[40210 rows x 2 columns]
```

[106]:
```
m2
```

[106]:
```
                   iid  score       vid  \
0      100010_20190329     -1       NaN
1      100010_20190403    100       NaN
2      100017_20190417     -1       NaN
3      100017_20190816     91  103105.0
4      100017_20190816     91  103139.0
…                 …    …        …
53925      99_20180808     84  103142.0
53926      99_20180808     84  103133.0
53927      99_20180808     84  103124.0
53928      99_20180808     84  103156.0
53929      99_20180808     84  103148.0

                                          description  risk_category
0                                                 NaN            NaN
1                                                 NaN            NaN
2                                                 NaN            NaN
3                           Improper cooling methods      High Risk
4                              Improper food storage       Low Risk
…                                                 …              …
53925              Unclean nonfood contact surfaces       Low Risk
53926        Foods not protected from contamination  Moderate Risk
```

```
53927  Inadequately cleaned or sanitized food contact…  Moderate Risk
53928     Permit license or inspection report not posted       Low Risk
53929       No thermometers or uncalibrated thermometers       Low Risk

[53930 rows x 5 columns]
```

---

7 6: Compute Something Interesting

Play with the data and try to compute something interesting about the data. Please try to use at least one of groupby, pivot, or merge (or all of the above).

Please show your work in the cell below and describe in words what you found in the same cell. This question will be graded leniently but good solutions may be used to create future homework problems.

Please have both your code and your explanation in the same one cell below. Any work in any other cell will not be graded.

[ ]:

[106]:
```
#YOUR CODE HERE
ins_named = pd.merge(ins, bus.drop(columns=['city','state','postal_code',␣
 ↪'latitude',
            'longitude', 'phone_number','postal5']), how='left')
#cond1: score > 0, groupby ['bid','name'], agg(np.mean)
twenty_highest_mean_score = ins_named[ins_named['score'] > 0].
 ↪groupby(['bid','name'],
                                as_index=False).agg(np.mean)
#cond2: sort ascending
twenty_highest_mean_score = twenty_highest_mean_score.sort_values(by='score',␣
 ↪ascending=True)
#rename score
twenty_highest_mean_score.rename(columns={'score': 'mean score'}, inplace=True)
#drop year
twenty_highest_mean_score.drop(columns=['year'],inplace=True)

twenty_highest_mean_score.head(5332)

#YOUR EXPLANATION HERE (in a comment)
# There are 5331 restaurants that do not have perfect mean score of 100.
# There are 393 restaurants that have a mean score of 100 (see cell below)
# Thus, only 7% of restaurants have perfect score.
```

[106]:

	bid	name	mean score
3876	84590	Chaat Corner	54.000000

```
4564   90622                      Taqueria Lolita   57.000000
4990   94351                           VBowls LLC   58.000000
2719   69282         New Jumbo Seafood Restaurant   60.500000
222     1154                SUNFLOWER RESTAURANT   63.500000
…       …                                   …         …
1082    5544          AT&T - Juma Cart 1 - Coffee   99.600000
1180    5894  SAN FRANCISCO COMMUNITY ELEM.SCHOOL   99.666667
1183    5898             MONROE ELEMENTARY SCHOOL   99.666667
1157    5864      VISITACION VALLEY MIDDLE SCHOOL   99.666667
5664  100817                   TEACUP & SANDWICHES  100.000000

[5332 rows x 3 columns]
```

[107]:
```
twenty_highest_mean_score[twenty_highest_mean_score['mean score'] == 100]

#YOUR EXPLANATION HERE (in a comment)
# There are 5331 restaurants that do not have perfect mean score of 100.
# There are 393 restaurants that have a mean score of 100
# Thus, only (393/5331) = 7% of restaurants have perfect score.
```

[107]:
```
          bid                                name  mean score
5664  100817                 TEACUP & SANDWICHES       100.0
5668  101089   Tony's/ City Bistro East Club 200       100.0
5667  101028                           BAKE LOVE       100.0
1405    8823                              Miette       100.0
5671  101100  Bakesale Betty/La Corneta South 200       100.0
…       …                                   …         …
5164   95216       95216 - Ballpark Snacks Stand       100.0
4226   87742                           King Knish       100.0
5172   95235         95235 Farmers Market/Snack       100.0
1546   18268                           Que Syrah       100.0
4846   92935                  Cvs/Pharmacy #5131       100.0

[393 rows x 3 columns]
```

[108]:
```
#THIS CELL AND ANY CELLS ADDED BELOW WILL NOT BE GRADED
```

7.1 Congratulations! You have finished Part 1 of Project 1!

In our analysis of the business data, we found that there are some errors with the ZIP codes. As a result, we made the records with ZIP codes outside of San Francisco have a `postal5` value of `None` and shortened 9-digit zip codes to 5-digit ones. In practice, we could take the time to look up the restaurant address online and fix some of the zip code issues.

In our analysis of the inspection data, we investigated the relationship between the year and type of inspection, and we figured out that only `Routine - Unscheduled` inspections have inspection

scores. Finally, we joined the business and inspection data to identify restaurants with the worst ratings and the lowest median scores.

[ ]:
```
#
```

8 Submit

Make sure you have run all cells in your notebook in order before running the cell below, so that all images/graphs appear in the output. **Please save before submitting!**

[109]:
```
# Save your notebook first, then run this cell to submit.
import jassign.to_pdf
jassign.to_pdf.generate_pdf('proj1a.ipynb', 'proj1a.pdf')
ok.submit()
```

```
Generating PDF…
Saved proj1a.pdf
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.Javascript object>
```

```
Saving notebook…
```

```
ERROR  | auth.py:91 | {'error': 'invalid_grant'}
```

```
Saved 'proj1a.ipynb'.
Performing authentication
Please enter your bCourses email.
bCourses email: letantruong32@berkeley.edu

Copy the following URL and open it in a web browser. To copy,
highlight the URL, right-click, and select "Copy".

https://okpy.org/client/login/

After logging in, copy the code from the web page, paste it below,
and press Enter. To paste, right-click and select "Paste".

Paste your code here: z2ZzlGpLYUwrmEWh8uHLdyHOWO7jGB
Successfully logged in as letantruong32@berkeley.edu
Submit… 100% complete
Submission successful for user: letantruong32@berkeley.edu
URL: https://okpy.org/cal/data100/sp20/proj1a/submissions/835MG2
```

[ ]:
```

```