proj1

September 10, 2019

[189]:
```python
# Initialize OK
from client.api.notebook import Notebook
ok = Notebook('proj1.ok')
```

```
=====================================================================
Assignment: proj1
OK, version v1.13.11
=====================================================================
```

1 Project 1: Food Safety

1.1 Cleaning and Exploring Data with Pandas

1.2 Due Date: Tuesday 2/12, 6:00 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, you will investigate restaurant food safety scores for restaurants in San Francisco. Above is a sample score card for a restaurant. 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 understand how restaurants are scored. We will walk through various steps of exploratory data analysis to do this. We will provide comments and insights along the way to give you a sense of how we arrive at each discovery and what next steps it leads to.

As we clean and explore these data, you will gain practice with: * Reading simple csv files * Working with data at different levels of granularity * Identifying the type of data collected, missing values, anomalies, etc. * Applying probability sampling techniques * Exploring characteristics and distributions of individual variables

1.5 Score Breakdown

Question	Points
1a	1
1b	0
1c	0
1d	3
1e	1
2a	1
2b	2
3a	2
3b	0
3c	2
3d	1
3e	1
3f	1
4a	1
4b	1
4c	1
4d	1
4e	1
4f	1
4g	2
4h	1
4i	1
5a	2
5b	3
6a	1
6b	1
6c	1
7a	2
7b	3
7c	3
8a	2
8b	2
8c	6
8d	2
8e	3
Total	56

To start the assignment, run the cell below to set up some imports and the automatic tests that we will need for this assignment:

In many of these assignments (and your future adventures as a data scientist) you will use `os`, `zipfile`, `pandas`, `numpy`, `matplotlib.pyplot`, and optionally `seaborn`.

1. Import each of these libraries `as` their commonly used abbreviations (e.g., `pd`, `np`, `plt`, and

sns).

2. Don't forget to include `%matplotlib inline` which enables inline matploblib plots.
3. If you want to use `seaborn`, add the line `sns.set()` to make your plots look nicer.

[190]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import os
import zipfile
import seaborn as sns
#import ds100_utils
%matplotlib inline
sns.set()
```

[191]:
```
import sys

assert 'zipfile'in sys.modules
assert 'pandas'in sys.modules and pd
assert 'numpy'in sys.modules and np
assert 'matplotlib'in sys.modules and plt
```

1.6 Downloading the Data

For this assignment, we need this data file: http://www.ds100.org/sp19/assets/datasets/proj1-SFBusinesses.zip

We could write a few lines of code that are built to download this specific data file, but it's a better idea to have a general function that we can reuse for all of our assignments. Since this class isn't really about the nuances of the Python file system libraries, we've provided a function for you in ds100_utils.py called `fetch_and_cache` that can download files from the internet.

This function has the following arguments: - data_url: the web address to download - file: the file in which to save the results - data_dir: (default="data") the location to save the data - force: if true the file is always re-downloaded

The way this function works is that it checks to see if `data_dir/file` already exists. If it does not exist already or if `force=True`, the file at `data_url` is downloaded and placed at `data_dir/file`. The process of storing a data file for reuse later is called caching. If `data_dir/file` already and exists `force=False`, nothing is downloaded, and instead a message is printed letting you know the date of the cached file.

The function returns a `pathlib.Path` object representing the location of the file (pathlib docs).

[192]:
```
import ds100_utils
source_data_url = 'http://www.ds100.org/sp19/assets/datasets/proj1-SFBusinesses.
↪zip'
target_file_name = 'data.zip'
```

```
# Change the force=False -> force=True in case you need to force redownload the␣
↪data
dest_path = ds100_utils.fetch_and_cache(
    data_url=source_data_url,
    data_dir='.',
    file=target_file_name,
    force=False)
```

```
Using cached version that was downloaded (UTC): Sun Feb 10 11:20:09 2019
```

After running the cell above, if you list the contents of the directory containing this notebook, you should see `data.zip`.

[193]:
```
!ls
```

```
data            proj1.ipynb  __pycache__  q8d.png        test.tplx
data.zip        proj1.ok     q7a.png      scoreCard.jpg
ds100_utils.py  proj1.pdf    q8c2.png     tests
```

1.7 1: Loading Food Safety Data

We have data, but we don't have any specific questions about the data yet, so let's focus on understanding the structure of the data. This involves answering questions such as:

- 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.zip`. It's not just a single file, but a compressed directory of multiple files. We could inspect it by uncompressing it using a shell command such as `!unzip data.zip`, but in this project we're going to do almost everything in Python for maximum portability.

1.7.1 Question 1a: Looking Inside and Extracting the Zip Files

Assign `my_zip` to a `Zipfile.zipfile` object representing `data.zip`, and 1ssign `list_files` to a list of all the names of the files in `data.zip`.

Hint: The Python docs describe how to create a `zipfile.ZipFile` object. You might also look back at the code from lecture and lab. It's OK to copy and paste code from previous assignments and demos, though you might get more out of this exercise if you type out an answer.

[194]:
```
my_zip = zipfile.ZipFile(dest_path)
list_names = my_zip.namelist()
list_names
```

[194]: ['violations.csv', 'businesses.csv', 'inspections.csv', 'legend.csv']

[195]:
```
ok.grade("q1a");
```

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

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

In your answer above, if you have written something like `zipfile.ZipFile('data.zip', ...)`, we suggest changing it to read `zipfile.ZipFile(dest_path, ...)`. In general, we **strongly suggest having your filenames hard coded as string literals only once** in a notebook. It is very dangerous to hard code things twice, because if you change one but forget to change the other, you can end up with bugs that are very hard to find.

Now display the files' names and their sizes.

If you're not sure how to proceed, read about the attributes of a `ZipFile` object in the Python docs linked above.

[196]:
```
my_zip.infolist()
```

[196]:
```
[<ZipInfo filename='violations.csv' compress_type=deflate external_attr=0x20
file_size=3726206 compress_size=286253>,
 <ZipInfo filename='businesses.csv' compress_type=deflate external_attr=0x20
file_size=660231 compress_size=178549>,
 <ZipInfo filename='inspections.csv' compress_type=deflate external_attr=0x20
file_size=466106 compress_size=83198>,
 <ZipInfo filename='legend.csv' compress_type=deflate external_attr=0x20
file_size=120 compress_size=104>]
```

Often when working with zipped data, we'll never unzip the actual zipfile. This saves space on our local computer. However, for this project, the files are small, so we're just going to unzip everything. This has the added benefit that you can look inside the csv files using a text editor, which might be handy for understanding what's going on. The cell below will unzip the csv files into a subdirectory called `data`. Just run it.

[197]:
```
from pathlib import Path
data_dir = Path('data')
my_zip.extractall(data_dir)
!ls {data_dir}
```

```
businesses.csv  inspections.csv  legend.csv  violations.csv
```

The cell above created a folder called `data`, and in it there should be four CSV files. Open up legend.csv to see its contents. You should see something that looks like:

```
"Minimum_Score","Maximum_Score","Description"
0,70,"Poor"
71,85,"Needs Improvement"
86,90,"Adequate"
91,100,"Good"
```

1.7.2 Question 1b: Programatically Looking Inside the Files

The `legend.csv` file does indeed look like a well-formed CSV file. Let's check the other three files. Rather than opening up each file manually, let's use Python to print out the first 5 lines of each. The `ds100_utils` library has a method called `head` that will allow you to retrieve the first N lines of a file as a list. For example `ds100_utils.head('data/legend.csv', 5)` will return the first 5 lines of "data/legend.csv". Try using this function to print out the first 5 lines of all four files that we just extracted from the zipfile.

[198]:
```python
print(ds100_utils.head('data/legend.csv', 5))
print(ds100_utils.head('data/violations.csv', 5))
print(ds100_utils.head('data/inspections.csv', 5))
print(ds100_utils.head('data/businesses.csv', 5))
```

```
['"Minimum_Score","Maximum_Score","Description"\n', '0,70,"Poor"\n',
'71,85,"Needs Improvement"\n', '86,90,"Adequate"\n', '91,100,"Good"\n']
['"business_id","date","description"\n', '19,"20171211","Inadequate food safety
knowledge or lack of certified food safety manager"\n',
'19,"20171211","Unapproved or unmaintained equipment or utensils"\n',
'19,"20160513","Unapproved or unmaintained equipment or utensils  [ date
violation corrected: 12/11/2017 ]"\n', '19,"20160513","Unclean or degraded
floors walls or ceilings  [ date violation corrected: 12/11/2017 ]"\n']
['"business_id","score","date","type"\n', '19,"94","20160513","routine"\n',
'19,"94","20171211","routine"\n', '24,"98","20171101","routine"\n',
'24,"98","20161005","routine"\n']
['"business_id","name","address","city","state","postal_code","latitude","longit
ude","phone_number"\n', '19,"NRGIZE LIFESTYLE CAFE","1200 VAN NESS AVE, 3RD
FLOOR","San Francisco","CA","94109","37.786848","-122.421547","+14157763262"\n',
'24,"OMNI S.F. HOTEL - 2ND FLOOR PANTRY","500 CALIFORNIA ST, 2ND  FLOOR","San
Francisco","CA","94104","37.792888","-122.403135","+14156779494"\n',
'31,"NORMAN\'S ICE CREAM AND FREEZES","2801 LEAVENWORTH ST ","San
Francisco","CA","94133","37.807155","-122.419004",""\n', '45,"CHARLIE\'S DELI
CAFE","3202 FOLSOM ST ","San
Francisco","CA","94110","37.747114","-122.413641","+14156415051"\n']
```

1.7.3 Question 1c: Reading in the Files

Based on the above information, let's attempt to load `businesses.csv`, `inspections.csv`, and `violations.csv` into pandas data frames with the following names: `bus`, `ins`, and `vio` respectively.

Note: Because of character encoding issues one of the files (`bus`) will require an additional argument `encoding='ISO-8859-1'` when calling `pd.read_csv`. One day you should read all about character encodings.

[199]:
```
# path to directory containing data
dsDir = Path('data')

bus = pd.read_csv('data/businesses.csv', encoding='ISO-8859-1')
ins = pd.read_csv('data/inspections.csv')
vio = pd.read_csv('data/violations.csv')
```

Now that you've read in the files, let's try some `pd.DataFrame` methods (docs). Use the `DataFrame.head` method to show the top few lines of the `bus`, `ins`, and `vio` dataframes. Use `Dataframe.describe` to learn about the numeric columns.

[200]:
```
print(bus.head(5))
print(ins.head(5))
print(vio.head(5))
```

```
   business_id                               name  \
0           19                 NRGIZE LIFESTYLE CAFE
1           24  OMNI S.F. HOTEL - 2ND FLOOR PANTRY
2           31        NORMAN'S ICE CREAM AND FREEZES
3           45                  CHARLIE'S DELI CAFE
4           48                           ART'S CAFE

                     address           city state postal_code   latitude  \
0   1200 VAN NESS AVE, 3RD FLOOR  San Francisco    CA       94109  37.786848
1  500 CALIFORNIA ST, 2ND  FLOOR  San Francisco    CA       94104  37.792888
2           2801 LEAVENWORTH ST   San Francisco    CA       94133  37.807155
3                3202 FOLSOM ST   San Francisco    CA       94110  37.747114
4                 747 IRVING ST   San Francisco    CA       94122  37.764013

    longitude  phone_number
0 -122.421547  +14157763262
1 -122.403135  +14156779494
2 -122.419004           NaN
3 -122.413641  +14156415051
4 -122.465749  +14156657440
   business_id  score      date     type
0           19     94  20160513  routine
1           19     94  20171211  routine
2           24     98  20171101  routine
3           24     98  20161005  routine
4           24     96  20160311  routine
   business_id      date                                        description
0           19  20171211  Inadequate food safety knowledge or lack of ce…
```

```
1             19  20171211   Unapproved or unmaintained equipment or utensils
2             19  20160513  Unapproved or unmaintained equipment or utensi…
3             19  20160513  Unclean or degraded floors walls or ceilings  …
4             19  20160513  Food safety certificate or food handler card n…
```

The `DataFrame.describe` method can also be handy for computing summaries of various statistics of our dataframes. Try it out with each of our 3 dataframes.

[201]:
```python
print(bus.describe)
print(ins.describe)
print(vio.describe)
```

```
<bound method NDFrame.describe of       business_id
name  \
0               19                   NRGIZE LIFESTYLE CAFE
1               24    OMNI S.F. HOTEL - 2ND FLOOR PANTRY
2               31        NORMAN'S ICE CREAM AND FREEZES
3               45                    CHARLIE'S DELI CAFE
4               48                             ART'S CAFE
5               54                    RHODA GOLDMAN PLAZA
6               56                            CAFE X + O
7               58                            OASIS GRILL
8               61                               CHOWDERS
9               66                       STARBUCKS COFFEE
10              67                        REVOLUTION CAFE
11              73                      DINO'S UNCLE VITO
12              76    OMNI S.F. HOTEL - 3RD FLOOR PANTRY
13              77  OMNI S.F. HOTEL - EMPLOYEE CAFETERIA
14              80                        LAW SCHOOL CAFE
15              81                    CLUB ED/BON APPETIT
16              88                           J.B.'S PLACE
17              95                                   VEGA
18              98                           XOX TRUFFLES
19              99         J & M A-1 CAFE RESTAURANT LLC
20             101                       CABLE CAR CORNER
21             102                      AKIKO'S SUSHI BAR
22             108                             RUE LEPIC
23             116             THE WATERFRONT RESTAURANT
24             121                           AKIKOS SUSHI
25             125                            CENTERFOLDS
26             134                                   MINT
27             140                         CAFE MADELEINE
28             141           AFC SUSHI @ MOLLIE STONE'S 2
29             146                 DEJA VU PIZZA & PASTA
…              …                                    …
6376         94305                ROSAMUNDE SAUSAGE GRILL
6377         94310                          YOKAI EXPRESS
6378         94318                         YUANBAO JIAOZI
```

```
6379        94331                      MATCHA CAFE MAIKO
6380        94334               SUBWAY SANDWICHES #53761
6381        94337               SUBWAY SANDWICHES #61240
6382        94354                RAINBOW MARKET AND DELI
6383        94387                        FOUNDATION CAFE
6384        94388                        FOUNDATION CAFE
6385        94394                         KOKIO REPUBLIC
6386        94408                      SIZZLING POT KING
6387        94409                            AUGUST HALL
6388        94412                  NATIVE BAKING COMPANY
6389        94433                         GREEK TOWN LLC
6390        94442                            SIMPLY CAFE
6391        94456                 UBER-ATG (BON APPETIT)
6392        94460                            DOBBS FERRY
6393        94465                         BEAUTIFULL LLC
6394        94468                              BAR CRENN
6395        94502                    NEW FORTUNE DIM SUM
6396        94521                 JOE & THE JUICE HOWARD
6397        94522                         CAFE JOSEPHINE
6398        94537          BON APPETIT @ USF- OUTTA HERE
6399        94540                          FOAM USA LLC
6400        94542                             OCEAN THAI
6401        94544                          D'MAIZE CAFE
6402        94555               EASY BREEZY FROZEN YOGURT
6403        94571                 THE PHOENIX PASTIFICIO
6404        94572                  BROADWAY DIM SUM CAFE
6405        94574                            BINKA BITES

                           address           city state postal_code  \
0      1200 VAN NESS AVE, 3RD FLOOR  San Francisco    CA       94109
1     500 CALIFORNIA ST, 2ND  FLOOR  San Francisco    CA       94104
2              2801 LEAVENWORTH ST   San Francisco    CA       94133
3                   3202 FOLSOM ST   San Francisco    CA       94110
4                    747 IRVING ST   San Francisco    CA       94122
5                     2180 POST ST   San Francisco    CA       94115
6                   1799 CHURCH ST   San Francisco    CA       94131
7                      91 DRUMM ST   San Francisco    CA       94111
8                PIER 39  SPACE A3   San Francisco    CA       94133
9                   1800 IRVING ST   San Francisco    CA       94122
10                    3248 22ND ST   San Francisco    CA       94110
11                2101 FILLMORE ST   San Francisco    CA       94115
12     500 CALIFORNIA ST, 3RD FLOOR  San Francisco    CA       94104
13      500 CALIFORNIA ST, BASEMENT  San Francisco    CA       94104
14                  2199 FULTON ST   San Francisco    CA       94117
15                    2350 TURK ST   San Francisco    CA       94117
16                    1435 17TH ST   San Francisco    CA       94107
17                419 CORTLAND AVE   San Francisco    CA       94110
18                754 COLUMBUS AVE   San Francisco    CA       94133
```

19	779 CLAY ST	San Francisco	CA	94108
20	1099 POWELL ST	San Francisco	CA	94108
21	542A MASON ST	San Francisco	CA	94102
22	900 PINE ST	San Francisco	CA	94108
23	PIER 7 EMBARCADERO	San Francisco	CA	94111
24	431 BUSH ST	San Francisco	CA	94108
25	391 BROADWAY ST	San Francisco	CA	94133
26	400 MCALLISTER ST	San Francisco	CA	94102
27	300 CALIFORNIA ST	San Francisco	CA	94104
28	2435 CALIFORNIA ST	San Francisco	CA	94115
29	3227 16TH ST	San Francisco	CA	94103
…	…	…	…	…
6376	545 HAIGHT ST	San Francisco	CA	94117
6377	135 4TH ST	San Francisco	CA	94103
6378	2110 IRVING ST	San Francisco	CA	94122
6379	1581 WEBSTER ST 175	San Francisco	CA	94115
6380	160 BROADWAY ST	San Francisco	CA	94111
6381	425 D BATTERY ST	San Francisco	CA	94111
6382	684 LARKIN ST	San Francisco	CA	94109
6383	645 5TH ST	San Francisco	CA	94107
6384	335 KEARNY ST	San Francisco	CA	94108
6385	428 11TH ST	San Francisco	CA	94109
6386	139 8TH ST	San Francisco	CA	94103
6387	420 MASON ST	San Francisco	CA	NaN
6388	1324 FITZGERALD AVE	San Francisco	CA	94124
6389	88 02ND ST	San Francisco	CA	94105
6390	340 GROVE ST	San Francisco	CA	94102
6391	581 20TH ST 2ND FL	San Francisco	CA	94107
6392	409 GOUGH ST	San Francisco	CA	94102
6393	3401 CALIFORNIA ST	San Francisco	CA	94118
6394	3131 FILLMORE ST	San Francisco	CA	94123
6395	811 STOCKTON ST	San Francisco	CA	94108
6396	301 HOWARD ST	San Francisco	CA	94105
6397	199 MUSEUM WAY	San Francisco	CA	94114
6398	2130 FULTON ST	San Francisco	CA	94117
6399	1745 TARAVAL ST	San Francisco	CA	94116
6400	2545 OCEAN AVE	San Francisco	CA	94132
6401	50 PHELAN AVE	San Francisco	CA	94112
6402	44 WEST PORTAL AVE	San Francisco	CA	94127
6403	200 CLEMENT ST	San Francisco	CA	94118
6404	684 BROADWAY ST	San Francisco	CA	94133
6405	2241 GEARY BLVD	San Francisco	CA	94115

	latitude	longitude	phone_number
0	37.786848	-122.421547	+14157763262
1	37.792888	-122.403135	+14156779494
2	37.807155	-122.419004	NaN
3	37.747114	-122.413641	+14156415051

4	37.764013	-122.465749	+14156657440
5	37.784626	-122.437734	+14153455060
6	37.742325	-122.426476	+14158263535
7	37.794483	-122.396584	+14158341942
8	37.808240	-122.410189	+14153914737
9	37.763578	-122.477461	+14152427970
10	37.755419	-122.419542	+14156420474
11	37.788932	-122.433895	+14159224700
12	37.792888	-122.403135	+14156779494
13	37.792888	-122.403135	+14156779494
14	37.774941	-122.452797	+14154222268
15	37.778468	-122.448484	+14154225849
16	37.765003	-122.398084	+14155848446
17	37.739207	-122.417447	+14152856000
18	37.801665	-122.412104	+14154214814
19	37.794293	-122.405967	+14156057219
20	37.794615	-122.409705	+14153625925
21	37.788484	-122.410045	+14159898218
22	37.790868	-122.410854	+14154746070
23	37.793874	-122.396464	+14153912696
24	37.790643	-122.404676	+14153973218
25	37.798233	-122.403637	+14158340662
26	37.780247	-122.418974	+14155515942
27	37.793268	-122.400323	+14153623332
28	37.788773	-122.434697	+14155674902
29	37.764713	-122.424709	+14152551600
…	…	…	…
6376	NaN	NaN	+14154376851
6377	NaN	NaN	+14158234502
6378	NaN	NaN	+14156013979
6379	NaN	NaN	+14150009434
6380	NaN	NaN	+14158861913
6381	NaN	NaN	+14153991549
6382	NaN	NaN	+14157664681
6383	NaN	NaN	+14153503301
6384	NaN	NaN	NaN
6385	NaN	NaN	+14157996404
6386	NaN	NaN	+14158028899
6387	NaN	NaN	NaN
6388	NaN	NaN	NaN
6389	NaN	NaN	+14152408032
6390	NaN	NaN	+14156587659
6391	NaN	NaN	+14158184997
6392	NaN	NaN	+14155517709
6393	NaN	NaN	+14157289080
6394	NaN	NaN	NaN
6395	NaN	NaN	+14153991511
6396	NaN	NaN	NaN

```
6397        NaN         NaN  +14153508976
6398        NaN         NaN  +14153604802
6399        NaN         NaN  +14156060018
6400        NaN         NaN  +14155857251
6401        NaN         NaN  +14154240604
6402        NaN         NaN  +14155053351
6403        NaN         NaN  +14154726100
6404        NaN         NaN           NaN
6405        NaN         NaN  +14157712907

[6406 rows x 9 columns]>
<bound method NDFrame.describe of        business_id  score      date     type
0               19     94  20160513  routine
1               19     94  20171211  routine
2               24     98  20171101  routine
3               24     98  20161005  routine
4               24     96  20160311  routine
5               31     98  20151204  routine
6               45     78  20160104  routine
7               45     88  20170307  routine
8               45     85  20170914  routine
9               45     84  20160614  routine
10              48     94  20160630  routine
11              54    100  20150526  routine
12              54     87  20170215  routine
13              56     90  20160802  routine
14              56     92  20170420  routine
15              56     88  20151222  routine
16              58     73  20160407  routine
17              58     70  20170918  routine
18              61     94  20160708  routine
19              61     94  20171128  routine
20              61     98  20170124  routine
21              61     92  20150827  routine
22              66     98  20160322  routine
23              66    100  20150828  routine
24              66    100  20160902  routine
25              66     96  20170703  routine
26              67     90  20150520  routine
27              67     87  20160401  routine
28              67     81  20170804  routine
29              67     94  20161019  routine
…             …    …        …       …
14192        93289     83  20171221  routine
14193        93297     98  20171221  routine
14194        93352     98  20171027  routine
14195        93361     90  20171219  routine
14196        93390     96  20171129  routine
```

```
14197        93423     96  20171103  routine
14198        93431     89  20171211  routine
14199        93448     96  20171117  routine
14200        93465     91  20180104  routine
14201        93492     96  20180110  routine
14202        93500    100  20171103  routine
14203        93532     93  20171103  routine
14204        93533     92  20171121  routine
14205        93536     94  20171213  routine
14206        93549     96  20171221  routine
14207        93615     89  20171106  routine
14208        93617     88  20171221  routine
14209        93815     96  20171102  routine
14210        93912     94  20180105  routine
14211        93957    100  20171204  routine
14212        93959    100  20171218  routine
14213        93968     98  20171120  routine
14214        93969     98  20171221  routine
14215        93977     96  20171219  routine
14216        94012    100  20171220  routine
14217        94012     90  20180112  routine
14218        94133    100  20171227  routine
14219        94142    100  20171220  routine
14220        94189     96  20171130  routine
14221        94231     85  20171214  routine

[14222 rows x 4 columns]>
<bound method NDFrame.describe of        business_id      date  \
0               19  20171211
1               19  20171211
2               19  20160513
3               19  20160513
4               19  20160513
5               24  20171101
6               24  20161005
7               24  20160311
8               24  20160311
9               31  20151204
10              45  20170914
11              45  20170914
12              45  20170914
13              45  20170914
14              45  20170307
15              45  20170307
16              45  20170307
17              45  20170307
18              45  20170307
19              45  20160614
```

```
20              45  20160614
21              45  20160614
22              45  20160614
23              45  20160614
24              45  20160104
25              45  20160104
26              45  20160104
27              45  20160104
28              45  20160104
29              45  20160104
…              …     …
39012        93465  20180104
39013        93465  20180104
39014        93492  20180110
39015        93532  20171103
39016        93533  20171121
39017        93533  20171121
39018        93536  20171213
39019        93536  20171213
39020        93549  20171221
39021        93615  20171106
39022        93615  20171106
39023        93617  20171221
39024        93617  20171221
39025        93617  20171221
39026        93617  20171221
39027        93815  20171102
39028        93815  20171102
39029        93912  20180105
39030        93912  20180105
39031        93968  20171120
39032        93969  20171221
39033        93977  20171219
39034        94012  20180112
39035        94012  20180112
39036        94012  20180112
39037        94189  20171130
39038        94231  20171214
39039        94231  20171214
39040        94231  20171214
39041        94231  20171214

                                             description
0       Inadequate food safety knowledge or lack of ce…
1        Unapproved or unmaintained equipment or utensils
2       Unapproved or unmaintained equipment or utensi…
3       Unclean or degraded floors walls or ceilings  …
4       Food safety certificate or food handler card n…
```

```
5                                   Improper food storage
6        Unclean or degraded floors walls or ceilings  …
7        Unclean or degraded floors walls or ceilings  …
8        Unclean or degraded floors walls or ceilings  …
9        Food safety certificate or food handler card n…
10                        Unclean nonfood contact surfaces
11                 Moderate risk food holding temperature 
12            Unclean or degraded floors walls or ceilings
13                             High risk vermin infestation
14       Moderate risk vermin infestation  [ date viola…
15       Unclean nonfood contact surfaces  [ date viola…
16       Food safety certificate or food handler card n…
17       Unclean or degraded floors walls or ceilings  …
18       Wiping cloths not clean or properly stored or …
19       Unapproved or unmaintained equipment or utensi…
20       Moderate risk vermin infestation  [ date viola…
21       Foods not protected from contamination  [ date…
22       Inadequate food safety knowledge or lack of ce…
23       Unclean or degraded floors walls or ceilings  …
24       Inadequately cleaned or sanitized food contact…
25       Unclean nonfood contact surfaces  [ date viola…
26       Inadequate food safety knowledge or lack of ce…
27       Employee eating or smoking  [ date violation c…
28       Unclean or degraded floors walls or ceilings  …
29       Unapproved or unmaintained equipment or utensi…
…                                                    …
39012    Wiping cloths not clean or properly stored or …
39013    High risk food holding temperature   [ date vi…
39014    Inadequately cleaned or sanitized food contact…
39015    No hot water or running water  [ date violatio…
39016    Inadequately cleaned or sanitized food contact…
39017    Moderate risk food holding temperature   [ dat…
39018    Inadequate and inaccessible handwashing facili…
39019                           Low risk vermin infestation
39020                              Improper thawing methods 
39021    High risk food holding temperature   [ date vi…
39022    Inadequately cleaned or sanitized food contact…
39023           Noncompliance with HAACP plan or variance
39024    Inadequately cleaned or sanitized food contact…
39025     Improper food labeling or menu misrepresentation
39026    Food safety certificate or food handler card n…
39027     Unapproved or unmaintained equipment or utensils
39028     Improper storage of equipment utensils or linens
39029    Inadequate and inaccessible handwashing facili…
39030         Unclean or degraded floors walls or ceilings
39031                       Unclean nonfood contact surfaces
39032       No thermometers or uncalibrated thermometers 
39033           Noncompliance with HAACP plan or variance
```

```
39034  Inadequate and inaccessible handwashing facili…
39035  Other moderate risk violation   [ date violatio…
39036  Wiping cloths not clean or properly stored or …
39037            Insufficient hot water or running water
39038  Unclean nonfood contact surfaces   [ date viola…
39039  High risk vermin infestation   [ date violation…
39040  Moderate risk food holding temperature    [ dat…
39041  Wiping cloths not clean or properly stored or …

[39042 rows x 3 columns]>
```

Now, we perform some sanity checks for you to verify that you loaded the data with the right structure. Run the following cells to load some basic utilities (you do not need to change these at all):

First, we check the basic structure of the data frames you created:

[202]:
```
assert all(bus.columns == ['business_id', 'name', 'address', 'city', 'state',␣
↪'postal_code',
                           'latitude', 'longitude', 'phone_number'])
assert 6400 <= len(bus) <= 6420

assert all(ins.columns == ['business_id', 'score', 'date', 'type'])
assert 14210 <= len(ins) <= 14250

assert all(vio.columns == ['business_id', 'date', 'description'])
assert 39020 <= len(vio) <= 39080
```

Next we'll check that the statistics match what we expect. The following are hard-coded statistical summaries of the correct data.

[203]:
```
bus_summary = pd.DataFrame(**{'columns': ['business_id', 'latitude',␣
↪'longitude'],
 'data': {'business_id': {'50%': 68294.5, 'max': 94574.0, 'min': 19.0},
  'latitude': {'50%': 37.780435, 'max': 37.824494, 'min': 37.668824},
  'longitude': {'50%': -122.41885450000001,
   'max': -122.368257,
   'min': -122.510896}},
 'index': ['min', '50%', 'max']})

ins_summary = pd.DataFrame(**{'columns': ['business_id', 'score'],
 'data': {'business_id': {'50%': 61462.0, 'max': 94231.0, 'min': 19.0},
  'score': {'50%': 92.0, 'max': 100.0, 'min': 48.0}},
 'index': ['min', '50%', 'max']})

vio_summary = pd.DataFrame(**{'columns': ['business_id'],
 'data': {'business_id': {'50%': 62060.0, 'max': 94231.0, 'min': 19.0}},
 'index': ['min', '50%', 'max']})
```

```python
from IPython.display import display

print('What we expect from your Businesses dataframe:')
display(bus_summary)
print('What we expect from your Inspections dataframe:')
display(ins_summary)
print('What we expect from your Violations dataframe:')
display(vio_summary)
```

```
What we expect from your Businesses dataframe:

     business_id   latitude   longitude
min         19.0  37.668824 -122.510896
50%      68294.5  37.780435 -122.418855
max      94574.0  37.824494 -122.368257


What we expect from your Inspections dataframe:

     business_id  score
min         19.0   48.0
50%      61462.0   92.0
max      94231.0  100.0


What we expect from your Violations dataframe:

     business_id
min         19.0
50%      62060.0
max      94231.0
```

The code below defines a testing function that we'll use to verify that your data has the same statistics as what we expect. Run these cells to define the function. The `df_allclose` function has this name because we are verifying that all of the statistics for your dataframe are close to the expected values. Why not `df_allequal`? It's a bad idea in almost all cases to compare two floating point values like 37.780435, as rounding error can cause spurious failures.

1.8 Question 1d: Verifying the data

Now let's run the automated tests. If your dataframes are correct, then the following cell will seem to do nothing, which is a good thing! However, if your variables don't match the correct answers in the main summary statistics shown above, an exception will be raised.

[204]:
```python
"""Run this cell to load this utility comparison function that we will use in␣
↪various
tests below (both tests you can see and those we run internally for grading).

Do not modify the function in any way.
```

```
"""


def df_allclose(actual, desired, columns=None, rtol=5e-2):
    """Compare selected columns of two dataframes on a few summary statistics.

    Compute the min, median and max of the two dataframes on the given columns,␣
↪and compare
    that they match numerically to the given relative tolerance.

    If they don't match, an AssertionError is raised (by `numpy.testing`).
    """
    # summary statistics to compare on
    stats = ['min', '50%', 'max']

    # For the desired values, we can provide a full DF with the same structure␣
↪as
    # the actual data, or pre-computed summary statistics.
    # We assume a pre-computed summary was provided if columns is None. In that␣
↪case,
    # `desired` *must* have the same structure as the actual's summary
    if columns is None:
        des = desired
        columns = desired.columns
    else:
        des = desired[columns].describe().loc[stats]

    # Extract summary stats from actual DF
    act = actual[columns].describe().loc[stats]

    return np.allclose(act, des, rtol)
```

[205]:
```
ok.grade("q1d");
```

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

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

[206]:
```
bus.head(5)
```

[206]:
```
   business_id                                 name  \
0           19                 NRGIZE LIFESTYLE CAFE
1           24  OMNI S.F. HOTEL - 2ND FLOOR PANTRY
2           31      NORMAN'S ICE CREAM AND FREEZES
3           45                 CHARLIE'S DELI CAFE
4           48                          ART'S CAFE

                    address           city state postal_code   latitude  \
0  1200 VAN NESS AVE, 3RD FLOOR  San Francisco    CA       94109  37.786848
1 500 CALIFORNIA ST, 2ND  FLOOR  San Francisco    CA       94104  37.792888
2           2801 LEAVENWORTH ST  San Francisco    CA       94133  37.807155
3                3202 FOLSOM ST  San Francisco    CA       94110  37.747114
4                  747 IRVING ST  San Francisco    CA       94122  37.764013

   longitude  phone_number
0 -122.421547  +14157763262
1 -122.403135  +14156779494
2 -122.419004           NaN
3 -122.413641  +14156415051
4 -122.465749  +14156657440
```

[207]:
```python
ins.head(5)
```

[207]:
```
   business_id  score      date     type
0           19     94  20160513  routine
1           19     94  20171211  routine
2           24     98  20171101  routine
3           24     98  20161005  routine
4           24     96  20160311  routine
```

[208]:
```python
vio.head(5)
```

[208]:
```
   business_id      date                                        description
0           19  20171211  Inadequate food safety knowledge or lack of ce…
1           19  20171211   Unapproved or unmaintained equipment or utensils
2           19  20160513  Unapproved or unmaintained equipment or utensi…
3           19  20160513  Unclean or degraded floors walls or ceilings  …
4           19  20160513  Food safety certificate or food handler card n…
```

1.8.1 Question 1e: Identifying Issues with the Data

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

The NaN phone number of Norman's can be a potential issue when dealing with functions that rely on numeric values

We will explore each file in turn, including determining its granularity and primary keys and exploring many of the variables individually. Let's begin with the businesses file, which has been read into the `bus` dataframe.

1.9 2: Examining the Business Data

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

Important note: From now on, 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.

1.9.1 Question 2a

Examining the entries in `bus`, is the `business_id` unique for each record? 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 `business_id` series has any duplicates.

[209]:
```
type(bus["business_id"])
```

[209]:
```
pandas.core.series.Series
```

[210]:
```
is_business_id_unique = (len(bus["business_id"]) == len((bus["business_id"]).
↪unique()))

#lhs basically count all the rows
#rhs generates an array of the unique values then just gets its length

#bus["business_id"].value_counts() == 1 alternative method.
#this compares ALL value counts to == 1
```

[211]:
```
ok.grade("q2a");
```

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

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

1.9.2 Question 2b

With this information, you can address the question of granularity. Answer the questions below.

1. What does each record represent (e.g., a business, a restaurant, a location, etc.)?

2. What is the primary key?
3. What would you find by grouping by the following columns: `business_id`, `name`, `address`?

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.**

1. each record represents a business, with the name, address, city, etc.
2. the primary key would be the business id
3. grouping by name and address would result in groups of restaurants with the same name and address respectively. grouping by business_id would be in theory the same, but since business_id is the primary key, grouping by it doesn't really do anything.

[327]:
```
#scratch work
#len(bus)
#bus

a0 = bus.groupby(["business_id"]).min()
a1 = bus.groupby(["name"])
a2 = bus.groupby(["address"]).count()

#groupby makes the column the index
a0
#a1.head(5)
```

[327]:
```
                                           name  \
business_id
19                          NRGIZE LIFESTYLE CAFE
24                 OMNI S.F. HOTEL - 2ND FLOOR PANTRY
31                      NORMAN'S ICE CREAM AND FREEZES
45                          CHARLIE'S DELI CAFE
48                                     ART'S CAFE
54                          RHODA GOLDMAN PLAZA
56                                  CAFE X + O
58                                  OASIS GRILL
61                                    CHOWDERS
66                              STARBUCKS COFFEE
67                             REVOLUTION CAFE
73                            DINO'S UNCLE VITO
76                 OMNI S.F. HOTEL - 3RD FLOOR PANTRY
77               OMNI S.F. HOTEL - EMPLOYEE CAFETERIA
80                               LAW SCHOOL CAFE
81                          CLUB ED/BON APPETIT
```

```
88                                J.B.'S PLACE
95                                        VEGA
98                                XOX TRUFFLES
99              J & M A-1 CAFE RESTAURANT LLC
101                           CABLE CAR CORNER
102                          AKIKO'S SUSHI BAR
108                                  RUE LEPIC
116                 THE WATERFRONT RESTAURANT
121                               AKIKOS SUSHI
125                                CENTERFOLDS
134                                       MINT
140                             CAFE MADELEINE
141             AFC SUSHI @ MOLLIE STONE'S 2
146                      DEJA VU PIZZA & PASTA
…                                          …
94286                                BUNN MIKE
94305                 ROSAMUNDE SAUSAGE GRILL
94310                            YOKAI EXPRESS
94318                           YUANBAO JIAOZI
94331                        MATCHA CAFE MAIKO
94334                  SUBWAY SANDWICHES #53761
94337                  SUBWAY SANDWICHES #61240
94354                  RAINBOW MARKET AND DELI
94387                          FOUNDATION CAFE
94388                          FOUNDATION CAFE
94394                           KOKIO REPUBLIC
94408                        SIZZLING POT KING
94412                    NATIVE BAKING COMPANY
94433                           GREEK TOWN LLC
94442                              SIMPLY CAFE
94456                   UBER-ATG (BON APPETIT)
94460                              DOBBS FERRY
94465                           BEAUTIFULL LLC
94468                                BAR CRENN
94502                      NEW FORTUNE DIM SUM
94521                   JOE & THE JUICE HOWARD
94522                           CAFE JOSEPHINE
94537            BON APPETIT @ USF- OUTTA HERE
94540                             FOAM USA LLC
94542                               OCEAN THAI
94544                             D'MAIZE CAFE
94555                EASY BREEZY FROZEN YOGURT
94571                   THE PHOENIX PASTIFICIO
94572                    BROADWAY DIM SUM CAFE
94574                              BINKA BITES

                                    address           city state postal_code  \
```

business_id				
19	1200 VAN NESS AVE, 3RD FLOOR	San Francisco	CA	94109
24	500 CALIFORNIA ST, 2ND FLOOR	San Francisco	CA	94104
31	2801 LEAVENWORTH ST	San Francisco	CA	94133
45	3202 FOLSOM ST	San Francisco	CA	94110
48	747 IRVING ST	San Francisco	CA	94122
54	2180 POST ST	San Francisco	CA	94115
56	1799 CHURCH ST	San Francisco	CA	94131
58	91 DRUMM ST	San Francisco	CA	94111
61	PIER 39 SPACE A3	San Francisco	CA	94133
66	1800 IRVING ST	San Francisco	CA	94122
67	3248 22ND ST	San Francisco	CA	94110
73	2101 FILLMORE ST	San Francisco	CA	94115
76	500 CALIFORNIA ST, 3RD FLOOR	San Francisco	CA	94104
77	500 CALIFORNIA ST, BASEMENT	San Francisco	CA	94104
80	2199 FULTON ST	San Francisco	CA	94117
81	2350 TURK ST	San Francisco	CA	94117
88	1435 17TH ST	San Francisco	CA	94107
95	419 CORTLAND AVE	San Francisco	CA	94110
98	754 COLUMBUS AVE	San Francisco	CA	94133
99	779 CLAY ST	San Francisco	CA	94108
101	1099 POWELL ST	San Francisco	CA	94108
102	542A MASON ST	San Francisco	CA	94102
108	900 PINE ST	San Francisco	CA	94108
116	PIER 7 EMBARCADERO	San Francisco	CA	94111
121	431 BUSH ST	San Francisco	CA	94108
125	391 BROADWAY ST	San Francisco	CA	94133
134	400 MCALLISTER ST	San Francisco	CA	94102
140	300 CALIFORNIA ST	San Francisco	CA	94104
141	2435 CALIFORNIA ST	San Francisco	CA	94115
146	3227 16TH ST	San Francisco	CA	94103
…	…	…	…	…
94286	752 COLUMBUS AVE	San Francisco	CA	94133
94305	545 HAIGHT ST	San Francisco	CA	94117
94310	135 4TH ST	San Francisco	CA	94103
94318	2110 IRVING ST	San Francisco	CA	94122
94331	1581 WEBSTER ST 175	San Francisco	CA	94115
94334	160 BROADWAY ST	San Francisco	CA	94111
94337	425 D BATTERY ST	San Francisco	CA	94111
94354	684 LARKIN ST	San Francisco	CA	94109
94387	645 5TH ST	San Francisco	CA	94107
94388	335 KEARNY ST	San Francisco	CA	94108
94394	428 11TH ST	San Francisco	CA	94109
94408	139 8TH ST	San Francisco	CA	94103
94412	1324 FITZGERALD AVE	San Francisco	CA	94124
94433	88 02ND ST	San Francisco	CA	94105
94442	340 GROVE ST	San Francisco	CA	94102

94456	581 20TH ST 2ND FL	San Francisco	CA	94107
94460	409 GOUGH ST	San Francisco	CA	94102
94465	3401 CALIFORNIA ST	San Francisco	CA	94118
94468	3131 FILLMORE ST	San Francisco	CA	94123
94502	811 STOCKTON ST	San Francisco	CA	94108
94521	301 HOWARD ST	San Francisco	CA	94105
94522	199 MUSEUM WAY	San Francisco	CA	94114
94537	2130 FULTON ST	San Francisco	CA	94117
94540	1745 TARAVAL ST	San Francisco	CA	94116
94542	2545 OCEAN AVE	San Francisco	CA	94132
94544	50 PHELAN AVE	San Francisco	CA	94112
94555	44 WEST PORTAL AVE	San Francisco	CA	94127
94571	200 CLEMENT ST	San Francisco	CA	94118
94572	684 BROADWAY ST	San Francisco	CA	94133
94574	2241 GEARY BLVD	San Francisco	CA	94115

	latitude	longitude	phone_number	postal_code_5
business_id				
19	37.786848	-122.421547	+14157763262	94109
24	37.792888	-122.403135	+14156779494	94104
31	37.807155	-122.419004	NaN	94133
45	37.747114	-122.413641	+14156415051	94110
48	37.764013	-122.465749	+14156657440	94122
54	37.784626	-122.437734	+14153455060	94115
56	37.742325	-122.426476	+14158263535	94131
58	37.794483	-122.396584	+14158341942	94111
61	37.808240	-122.410189	+14153914737	94133
66	37.763578	-122.477461	+14152427970	94122
67	37.755419	-122.419542	+14156420474	94110
73	37.788932	-122.433895	+14159224700	94115
76	37.792888	-122.403135	+14156779494	94104
77	37.792888	-122.403135	+14156779494	94104
80	37.774941	-122.452797	+14154222268	94117
81	37.778468	-122.448484	+14154225849	94117
88	37.765003	-122.398084	+14155848446	94107
95	37.739207	-122.417447	+14152856000	94110
98	37.801665	-122.412104	+14154214814	94133
99	37.794293	-122.405967	+14156057219	94108
101	37.794615	-122.409705	+14153625925	94108
102	37.788484	-122.410045	+14159898218	94102
108	37.790868	-122.410854	+14154746070	94108
116	37.793874	-122.396464	+14153912696	94111
121	37.790643	-122.404676	+14153973218	94108
125	37.798233	-122.403637	+14158340662	94133
134	37.780247	-122.418974	+14155515942	94102
140	37.793268	-122.400323	+14153623332	94104
141	37.788773	-122.434697	+14155674902	94115

```
146          37.764713 -122.424709  +14152551600         94103
…                    …          …           …           …
94286              NaN        NaN           NaN         94133
94305              NaN        NaN  +14154376851         94117
94310              NaN        NaN  +14158234502         94103
94318              NaN        NaN  +14156013979         94122
94331              NaN        NaN  +14150009434         94115
94334              NaN        NaN  +14158861913         94111
94337              NaN        NaN  +14153991549         94111
94354              NaN        NaN  +14157664681         94109
94387              NaN        NaN  +14153503301         94107
94388              NaN        NaN           NaN         94108
94394              NaN        NaN  +14157996404         94109
94408              NaN        NaN  +14158028899         94103
94412              NaN        NaN           NaN         94124
94433              NaN        NaN  +14152408032         94105
94442              NaN        NaN  +14156587659         94102
94456              NaN        NaN  +14158184997         94107
94460              NaN        NaN  +14155517709         94102
94465              NaN        NaN  +14157289080         94118
94468              NaN        NaN           NaN         94123
94502              NaN        NaN  +14153991511         94108
94521              NaN        NaN           NaN         94105
94522              NaN        NaN  +14153508976         94114
94537              NaN        NaN  +14153604802         94117
94540              NaN        NaN  +14156060018         94116
94542              NaN        NaN  +14155857251         94132
94544              NaN        NaN  +14154240604         94112
94555              NaN        NaN  +14155053351         94127
94571              NaN        NaN  +14154726100         94118
94572              NaN        NaN           NaN         94133
94574              NaN        NaN  +14157712907         94115

[6146 rows x 9 columns]
```

[213]:
```
#test2 = bus.groupby(['address']).sort_values(by = "address")
#test2.head()
```

[ ]:
```
```

1.10 3: Zip Codes

Next, let's explore some of the variables in the business table. We begin by examining the postal code.

1.10.1 Question 3a

Answer the following questions about the `postal code` column in the `bus` data frame?
1. Are ZIP codes quantitative or qualitative? If qualitative, is it ordinal or nominal? 1. What data type is used to represent a ZIP code?

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

1. the zip codes are qualitative. nominal
2. str

1.10.2 Question 3b

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. 94110 should be at the top with a count of 596.

[215]:
```
zip_counts = bus["postal_code"].value_counts()
```

Did you take into account that some businesses have missing ZIP codes?

[216]:
```
print('zip_counts describes', sum(zip_counts), 'records.')
print('The original data have', len(bus), 'records')
```

```
zip_counts describes 6166 records.
The original data have 6406 records
```

Missing data is extremely common in real-world data science projects. There are several ways to include missing postal codes in the `zip_counts` series above. One approach is to use the `fillna` method of the series, which will replace all null (a.k.a. NaN) values with a string of our choosing. In the example below, we picked "?????". When you run the code below, you should see that there are 240 businesses with missing zip code.

[217]:
```
zip_counts = bus.fillna("?????").groupby("postal_code").size().
↪sort_values(ascending=False)
zip_counts.head(15)
```

[217]:
```
postal_code
94110    596
94103    552
94102    462
94107    460
94133    426
94109    380
94111    277
94122    273
94118    249
94115    243
```

```
?????    240
94105    232
94108    228
94114    223
94117    204
dtype: int64
```

An alternate approach is to use the DataFrame `value_counts` method with the optional argument `dropna=False`, which will ensure that null values are counted. In this case, the index will be `NaN` for the row corresponding to a null postal code.

[218]:
```
bus["postal_code"].value_counts(dropna=False).sort_values(ascending = False).
↪head(15)
```

[218]:
```
94110    596
94103    552
94102    462
94107    460
94133    426
94109    380
94111    277
94122    273
94118    249
94115    243
NaN      240
94105    232
94108    228
94114    223
94117    204
Name: postal_code, dtype: int64
```

Missing zip codes aren't our only problem. There are also some records where the postal code is wrong, e.g., there are 3 'Ca' and 3 'CA' values. Additionally, there are some extended postal codes that are 9 digits long, rather than the typical 5 digits.

Let's clean up the extended zip codes by dropping the digits beyond the first 5. Rather than deleting or replacing the old values in the `postal_code` columnm, we'll instead create a new column called `postal_code_5`.

The reason we're making a new column is that it's typically good practice to keep the original values when we are manipulating data. This makes it easier to recover from mistakes, and also makes it more clear that we are not working with the original raw data.

[219]:
```
bus['postal_code_5'] = bus['postal_code'].str[:5]
bus.head()
```

[219]:
```
   business_id                                name  \
0           19              NRGIZE LIFESTYLE CAFE
1           24   OMNI S.F. HOTEL - 2ND FLOOR PANTRY
```

```
2           31      NORMAN'S ICE CREAM AND FREEZES
3           45                 CHARLIE'S DELI CAFE
4           48                          ART'S CAFE

                          address            city state postal_code   latitude  \
0    1200 VAN NESS AVE, 3RD FLOOR  San Francisco    CA       94109  37.786848
1   500 CALIFORNIA ST, 2ND  FLOOR  San Francisco    CA       94104  37.792888
2            2801 LEAVENWORTH ST   San Francisco    CA       94133  37.807155
3                 3202 FOLSOM ST   San Francisco    CA       94110  37.747114
4                  747 IRVING ST   San Francisco    CA       94122  37.764013

    longitude  phone_number postal_code_5
0 -122.421547  +14157763262         94109
1 -122.403135  +14156779494         94104
2 -122.419004           NaN         94133
3 -122.413641  +14156415051         94110
4 -122.465749  +14156657440         94122
```

1.10.3 Question 3c : A Closer Look at Missing ZIP Codes

Let's look more closely at records with missing ZIP codes. Describe why some records have missing postal codes. Pay attention to their addresses. You will need to look at many entries, not just the first five.

Hint: The `isnull` method of a series returns a boolean series which is true only for entries in the original series that were missing.

Missing addresses like "Off the grid", "approved private locations", etc. Some have various locations as well so a missing zip makes sense.

[220]:
```python
#bus["postal_code"] =  #pd.isnull(bus["postal_code"])

zipmiss = bus.loc[bus.postal_code.isnull() == True]

zipmiss_sort = zipmiss.groupby('address').count().sort_values("business_id",␣
 ↪ascending = False)
zipmiss_sort.head(10)
#bus.iloc[[6387]]

#some have missing addresses (off the grid, approved private, etc.)...
```

[220]:
```
                              business_id  name  city  state  postal_code  \
address
 OFF THE GRID                          69    69    69     69            0
 APPROVED PRIVATE LOCATIONS             6     6     6      6            0
 APPROVED LOCATIONS                     4     4     4      4            0
VARIOUS LOCATIONS                       2     2     2      2            0
```

```
OFF THE GRID                              2     2     2      2            0
 JUSTIN HERMAN PLAZA                      2     2     2      2            0
428 11TH ST                               2     2     2      2            0
 OTG                                      2     2     2      2            0
400 CALIFORNIA                            1     1     1      1            0
370 GOLDEN GATE AVE                       1     1     1      1            0

                               latitude  longitude  phone_number  \
address
 OFF THE GRID                         3          3            57
 APPROVED PRIVATE LOCATIONS           0          0             6
 APPROVED LOCATIONS                   0          0             4
VARIOUS LOCATIONS                     0          0             2
OFF THE GRID                          0          0             2
 JUSTIN HERMAN PLAZA                  2          2             2
428 11TH ST                           0          0             2
 OTG                                  0          0             1
400 CALIFORNIA                        0          0             1
370 GOLDEN GATE AVE                   0          0             1

                               postal_code_5
address
 OFF THE GRID                              0
 APPROVED PRIVATE LOCATIONS                0
 APPROVED LOCATIONS                        0
VARIOUS LOCATIONS                          0
OFF THE GRID                               0
 JUSTIN HERMAN PLAZA                       0
428 11TH ST                                0
 OTG                                       0
400 CALIFORNIA                             0
370 GOLDEN GATE AVE                        0
```

1.10.4 Question 3d: Incorrect ZIP Codes

This dataset is supposed to be only about San Francisco, so let's set up a list of all San Francisco ZIP codes.

[221]:
```
all_sf_zip_codes = ["94102", "94103", "94104", "94105", "94107", "94108",
                    "94109", "94110", "94111", "94112", "94114", "94115",
                    "94116", "94117", "94118", "94119", "94120", "94121",
                    "94122", "94123", "94124", "94125", "94126", "94127",
                    "94128", "94129", "94130", "94131", "94132", "94133",
                    "94134", "94137", "94139", "94140", "94141", "94142",
                    "94143", "94144", "94145", "94146", "94147", "94151",
                    "94158", "94159", "94160", "94161", "94163", "94164",
```

```
                    "94172", "94177", "94188"]
```

Set `weird_zip_code_businesses` equal to a new dataframe showing only rows corresponding to ZIP codes that are not valid and not missing. Use the `postal_code_5` column.

Hint: The ~ operator inverts a boolean array. Use in conjunction with `isin`.

[222]:
```
weird_zip_code_businesses = bus.loc[bus["postal_code_5"].isin(all_sf_zip_codes)␣
 ↪== False]
weird_zip_code_businesses = weird_zip_code_businesses.
 ↪dropna(subset=['postal_code'])
#drop rows based on column value
weird_zip_code_businesses
#bus["postal_code_5"].isin(all_sf_zip_codes)

#using ~?
#need to filter out NULL ones
```

[222]:
```
      business_id                                          name  \
1211         5208                            GOLDEN GATE YACHT CLUB
1372         5755                                   J & J VENDING
1373         5757                                RICO VENDING, INC
2258        36547                                EPIC ROASTHOUSE
2293        37167  INTERCONTINENTAL SAN FRANCISCO EMPLOYEE CAFETERIA
2295        37169     INTERCONTINENTAL SAN FRANCISCO 4TH FL. KITCHEN
2846        64540                                  LEO'S HOT DOGS
2852        64660                              HAIGHT STREET MARKET
2857        64738                                       JAPACURRY
2969        65856                                     BAMBOO ASIA
3142        67875                               THE CHAIRMAN TRUCK
3665        72127                                 REVOLUTION FOODS
3758        74674                                  ELI'S HOT DOGS
4853        83744                                  LA FROMAGERIE
5060        85459                                      ORBIT ROOM
5325        87059                            COFFEE BAR-MONTGOMERY
5480        88139                                    TACOLICIOUS
5894        90733                                      JEEPSILOG
6002        91249                                      AN THE GO
6130        92141                                   ALFARO TRUCK
6300        93484                             CARDONA'S FOOD TRUCK

                     address           city state postal_code  \
1211               1 YACHT RD  San Francisco    CA         941
1372    VARIOUS LOACATIONS (17)  San Francisco    CA       94545
1373         VARIOUS LOCATIONS  San Francisco    CA       94066
2258    PIER 26 EMBARARCADERO  San Francisco    CA       95105
2293    888 HOWARD ST 2ND FLOOR  San Francisco    CA       94013
2295    888 HOWARD ST 4TH FLOOR  San Francisco    CA       94013
```

```
2846                2301 MISSION ST   San Francisco    CA           CA
2852                 1530 HAIGHT ST   San Francisco    CA        92672
2857                        PUBLIC    San Francisco    CA           CA
2969               41 MONTGOMERY ST   San Francisco    CA        94101
3142                  OFF THE GRID    San Francisco    CA        00000
3665                  5383 CAPWELL    San Francisco    CA        94621
3758              101 BAYSHORE BLVD   San Francisco    CA        94014
4853              101 MONTGOMERY ST   San Francisco    CA        94101
5060                 1900 MARKET ST   San Francisco    CA        94602
5325  101 MONTGOMERY ST SUITE 101C   San Francisco    CA        94014
5480               2250 CHESTNUT ST   San Francisco    CA           Ca
5894                  2 MARINA BLVD   San Francisco    CA        94080
6002                  OFF THE GRID    San Francisco    CA        00000
6130                332 VALENCIA ST   San Francisco    CA        64110
6300                2430 WHIPPLE RD   San Francisco    CA        94544

       latitude   longitude  phone_number postal_code_5
1211  37.807878 -122.442499  +14153462628           941
1372        NaN         NaN  +14156750910         94545
1373        NaN         NaN  +14155836723         94066
2258  37.788962 -122.387941  +14153699955         95105
2293  37.781664 -122.404778  +14156166532         94013
2295  37.781664 -122.404778  +14156166532         94013
2846  37.760054 -122.419166  +14152406434            CA
2852  37.769957 -122.447533  +14152550643         92672
2857  37.777122 -122.419639  +14152444785            CA
2969  37.774998 -122.418299  +14156246790         94101
3142  37.777122 -122.419639  +14158461711         00000
3665        NaN         NaN           NaN         94621
3758        NaN         NaN  +14158301168         94014
4853        NaN         NaN  +14153682943         94101
5060        NaN         NaN  +14153705584         94602
5325        NaN         NaN  +14158158774         94014
5480        NaN         NaN  +14156496077            Ca
5894        NaN         NaN  +14157035586         94080
6002        NaN         NaN  +14158192000         00000
6130        NaN         NaN  +14159409273         64110
6300        NaN         NaN  +14153365990         94544
```

If we were doing very serious data analysis, we might indivdually look up every one of these strange records. Let's focus on just two of them: ZIP codes 94545 and 94602. Use a search engine to identify what cities these ZIP codes appear in. Try to explain why you think these two ZIP codes appear in your dataframe. For the one with ZIP code 94602, try searching for the business name and locate its real address.

94545 is Hayward. 94602 is oakland.

Orbit Room's real address is 1900 Market St, San Francisco, CA 94102.

I think these zip codes appear either because there are multiple locations and thus one location is selected (in the case of J & J) or an incorrect value is entered (Orbit)

1.10.5 Question 3e

We often want to clean the data to improve our analysis. This cleaning might include changing values for a variable or dropping records.

The value 94602 is wrong. Change it to the most reasonable correct value, using all information you have available. Modify the `postal_code_5` field using `bus['postal_code_5'].str.replace` to replace 94602.

[223]:
```
# WARNING: Be careful when uncommenting the line below, it will set the entire␣
↪column to NaN unless you
# put something to the right of the ellipses.
bus["postal_code_5"] = bus['postal_code_5'].str.replace(pat = "94602", repl =␣
↪"94102")
```

[224]:
```
ok.grade("q3e");
```

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

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

1.10.6 Question 3f

Now that we have corrected one of the weird postal codes, let's filter our `bus` data such that only postal codes from San Francisco remain. While we're at it, we'll also remove the businesses that are missing a postal code. As we mentioned in question 3d, filtering our postal codes in this way may not be ideal. (Fortunately, this is just a course assignment.)

Assign `bus` to a new dataframe that has the same columns but only the rows with ZIP codes in San Francisco.

[225]:
```
bus = bus.loc[bus["postal_code_5"].isin(all_sf_zip_codes) &␣
↪(~bus["postal_code_5"].isnull())]
#bus[bus["postal_code_5"].isin(all_sf_zip_codes)]
bus
```

[225]:
```
      business_id                                  name  \
0              19                   NRGIZE LIFESTYLE CAFE
1              24    OMNI S.F. HOTEL - 2ND FLOOR PANTRY
```

2	31	NORMAN'S ICE CREAM AND FREEZES
3	45	CHARLIE'S DELI CAFE
4	48	ART'S CAFE
5	54	RHODA GOLDMAN PLAZA
6	56	CAFE X + O
7	58	OASIS GRILL
8	61	CHOWDERS
9	66	STARBUCKS COFFEE
10	67	REVOLUTION CAFE
11	73	DINO'S UNCLE VITO
12	76	OMNI S.F. HOTEL - 3RD FLOOR PANTRY
13	77	OMNI S.F. HOTEL - EMPLOYEE CAFETERIA
14	80	LAW SCHOOL CAFE
15	81	CLUB ED/BON APPETIT
16	88	J.B.'S PLACE
17	95	VEGA
18	98	XOX TRUFFLES
19	99	J & M A-1 CAFE RESTAURANT LLC
20	101	CABLE CAR CORNER
21	102	AKIKO'S SUSHI BAR
22	108	RUE LEPIC
23	116	THE WATERFRONT RESTAURANT
24	121	AKIKOS SUSHI
25	125	CENTERFOLDS
26	134	MINT
27	140	CAFE MADELEINE
28	141	AFC SUSHI @ MOLLIE STONE'S 2
29	146	DEJA VU PIZZA & PASTA
…	…	…
6375	94286	BUNN MIKE
6376	94305	ROSAMUNDE SAUSAGE GRILL
6377	94310	YOKAI EXPRESS
6378	94318	YUANBAO JIAOZI
6379	94331	MATCHA CAFE MAIKO
6380	94334	SUBWAY SANDWICHES #53761
6381	94337	SUBWAY SANDWICHES #61240
6382	94354	RAINBOW MARKET AND DELI
6383	94387	FOUNDATION CAFE
6384	94388	FOUNDATION CAFE
6385	94394	KOKIO REPUBLIC
6386	94408	SIZZLING POT KING
6388	94412	NATIVE BAKING COMPANY
6389	94433	GREEK TOWN LLC
6390	94442	SIMPLY CAFE
6391	94456	UBER-ATG (BON APPETIT)
6392	94460	DOBBS FERRY
6393	94465	BEAUTIFULL LLC

```
6394         94468                              BAR CRENN
6395         94502                    NEW FORTUNE DIM SUM
6396         94521                 JOE & THE JUICE HOWARD
6397         94522                         CAFE JOSEPHINE
6398         94537         BON APPETIT @ USF- OUTTA HERE
6399         94540                          FOAM USA LLC
6400         94542                            OCEAN THAI
6401         94544                          D'MAIZE CAFE
6402         94555              EASY BREEZY FROZEN YOGURT
6403         94571                 THE PHOENIX PASTIFICIO
6404         94572                  BROADWAY DIM SUM CAFE
6405         94574                            BINKA BITES

                           address           city state postal_code  \
0      1200 VAN NESS AVE, 3RD FLOOR  San Francisco    CA       94109
1     500 CALIFORNIA ST, 2ND  FLOOR  San Francisco    CA       94104
2             2801 LEAVENWORTH ST   San Francisco    CA       94133
3                  3202 FOLSOM ST   San Francisco    CA       94110
4                   747 IRVING ST   San Francisco    CA       94122
5                    2180 POST ST   San Francisco    CA       94115
6                  1799 CHURCH ST   San Francisco    CA       94131
7                     91 DRUMM ST   San Francisco    CA       94111
8                PIER 39  SPACE A3  San Francisco    CA       94133
9                  1800 IRVING ST   San Francisco    CA       94122
10                   3248 22ND ST   San Francisco    CA       94110
11               2101 FILLMORE ST   San Francisco    CA       94115
12     500 CALIFORNIA ST, 3RD FLOOR  San Francisco    CA       94104
13      500 CALIFORNIA ST, BASEMENT  San Francisco    CA       94104
14                 2199 FULTON ST   San Francisco    CA       94117
15                   2350 TURK ST   San Francisco    CA       94117
16                   1435 17TH ST   San Francisco    CA       94107
17               419 CORTLAND AVE   San Francisco    CA       94110
18               754 COLUMBUS AVE   San Francisco    CA       94133
19                    779 CLAY ST   San Francisco    CA       94108
20                 1099 POWELL ST   San Francisco    CA       94108
21                  542A MASON ST   San Francisco    CA       94102
22                     900 PINE ST  San Francisco    CA       94108
23              PIER 7 EMBARCADERO  San Francisco    CA       94111
24                    431 BUSH ST   San Francisco    CA       94108
25                391 BROADWAY ST   San Francisco    CA       94133
26               400 MCALLISTER ST  San Francisco    CA       94102
27              300 CALIFORNIA ST   San Francisco    CA       94104
28             2435 CALIFORNIA ST   San Francisco    CA       94115
29                   3227 16TH ST   San Francisco    CA       94103
…                              …              …    …          …
6375             752 COLUMBUS AVE   San Francisco    CA       94133
6376                545 HAIGHT ST   San Francisco    CA       94117
```

6377	135 4TH ST	San Francisco	CA	94103
6378	2110 IRVING ST	San Francisco	CA	94122
6379	1581 WEBSTER ST 175	San Francisco	CA	94115
6380	160 BROADWAY ST	San Francisco	CA	94111
6381	425 D BATTERY ST	San Francisco	CA	94111
6382	684 LARKIN ST	San Francisco	CA	94109
6383	645 5TH ST	San Francisco	CA	94107
6384	335 KEARNY ST	San Francisco	CA	94108
6385	428 11TH ST	San Francisco	CA	94109
6386	139 8TH ST	San Francisco	CA	94103
6388	1324 FITZGERALD AVE	San Francisco	CA	94124
6389	88 02ND ST	San Francisco	CA	94105
6390	340 GROVE ST	San Francisco	CA	94102
6391	581 20TH ST 2ND FL	San Francisco	CA	94107
6392	409 GOUGH ST	San Francisco	CA	94102
6393	3401 CALIFORNIA ST	San Francisco	CA	94118
6394	3131 FILLMORE ST	San Francisco	CA	94123
6395	811 STOCKTON ST	San Francisco	CA	94108
6396	301 HOWARD ST	San Francisco	CA	94105
6397	199 MUSEUM WAY	San Francisco	CA	94114
6398	2130 FULTON ST	San Francisco	CA	94117
6399	1745 TARAVAL ST	San Francisco	CA	94116
6400	2545 OCEAN AVE	San Francisco	CA	94132
6401	50 PHELAN AVE	San Francisco	CA	94112
6402	44 WEST PORTAL AVE	San Francisco	CA	94127
6403	200 CLEMENT ST	San Francisco	CA	94118
6404	684 BROADWAY ST	San Francisco	CA	94133
6405	2241 GEARY BLVD	San Francisco	CA	94115

	latitude	longitude	phone_number	postal_code_5
0	37.786848	-122.421547	+14157763262	94109
1	37.792888	-122.403135	+14156779494	94104
2	37.807155	-122.419004	NaN	94133
3	37.747114	-122.413641	+14156415051	94110
4	37.764013	-122.465749	+14156657440	94122
5	37.784626	-122.437734	+14153455060	94115
6	37.742325	-122.426476	+14158263535	94131
7	37.794483	-122.396584	+14158341942	94111
8	37.808240	-122.410189	+14153914737	94133
9	37.763578	-122.477461	+14152427970	94122
10	37.755419	-122.419542	+14156420474	94110
11	37.788932	-122.433895	+14159224700	94115
12	37.792888	-122.403135	+14156779494	94104
13	37.792888	-122.403135	+14156779494	94104
14	37.774941	-122.452797	+14154222268	94117
15	37.778468	-122.448484	+14154225849	94117
16	37.765003	-122.398084	+14155848446	94107

```
17     37.739207 -122.417447  +14152856000         94110
18     37.801665 -122.412104  +14154214814         94133
19     37.794293 -122.405967  +14156057219         94108
20     37.794615 -122.409705  +14153625925         94108
21     37.788484 -122.410045  +14159898218         94102
22     37.790868 -122.410854  +14154746070         94108
23     37.793874 -122.396464  +14153912696         94111
24     37.790643 -122.404676  +14153973218         94108
25     37.798233 -122.403637  +14158340662         94133
26     37.780247 -122.418974  +14155515942         94102
27     37.793268 -122.400323  +14153623332         94104
28     37.788773 -122.434697  +14155674902         94115
29     37.764713 -122.424709  +14152551600         94103
…          …          …             …            …
6375         NaN         NaN           NaN         94133
6376         NaN         NaN  +14154376851         94117
6377         NaN         NaN  +14158234502         94103
6378         NaN         NaN  +14156013979         94122
6379         NaN         NaN  +14150009434         94115
6380         NaN         NaN  +14158861913         94111
6381         NaN         NaN  +14153991549         94111
6382         NaN         NaN  +14157664681         94109
6383         NaN         NaN  +14153503301         94107
6384         NaN         NaN           NaN         94108
6385         NaN         NaN  +14157996404         94109
6386         NaN         NaN  +14158028899         94103
6388         NaN         NaN           NaN         94124
6389         NaN         NaN  +14152408032         94105
6390         NaN         NaN  +14156587659         94102
6391         NaN         NaN  +14158184997         94107
6392         NaN         NaN  +14155517709         94102
6393         NaN         NaN  +14157289080         94118
6394         NaN         NaN           NaN         94123
6395         NaN         NaN  +14153991511         94108
6396         NaN         NaN           NaN         94105
6397         NaN         NaN  +14153508976         94114
6398         NaN         NaN  +14153604802         94117
6399         NaN         NaN  +14156060018         94116
6400         NaN         NaN  +14155857251         94132
6401         NaN         NaN  +14154240604         94112
6402         NaN         NaN  +14155053351         94127
6403         NaN         NaN  +14154726100         94118
6404         NaN         NaN           NaN         94133
6405         NaN         NaN  +14157712907         94115

[6146 rows x 10 columns]
```

[226]:
```
ok.grade("q3f");
```

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

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

1.11 4: Sampling from the Business Data

We can now sample from the business data using the cleaned ZIP code data. Make sure to use `postal_code_5` instead of `postal_code` for all parts of this question.

1.11.1 Question 4a

First, complete the following function `sample`, which takes as arguments a series, `series`, and a sample size, `n`, and returns a simple random sample (SRS) of size `n` from the series. Recall that in SRS, sampling is performed **without** replacement. The result should be a **list** of the `n` values that are in the sample.

Hint: Consider using `np.random.choice`.

[227]:
```
def sample(series, n):
    # Do not change the following line of code in any way!
    # In case you delete it, it should be "np.random.seed(40)"
    np.random.seed(40)
    return list(np.random.choice(series, size = n, replace = False))
```

[228]:
```
ok.grade("q4a");
```

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

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

1.11.2 Question 4b

Suppose we take a SRS of 5 businesses from the business data. What is the probability that the business named AMERICANA GRILL & FOUNTAIN is in the sample?

[229]:
```
q4b_answer = 5/21
q4b_answer

#len(bus)
```

[229]:
```
0.23809523809523808
```

[230]:
```
ok.grade("q4b");
```

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

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

1.11.3 Question 4c

Collect a stratified random sample of business names, where each stratum consists of a postal code. Collect one business name per stratum. Assign `bus_strat_sample` to a series of business names selected by this sampling procedure.

Hint: You can use the `sample` function you defined earlier.

[231]:
```
bus_strat_sample = bus.groupby("postal_code_5")["name"].agg(lambda group:␣
 ↪sample(group, 1)[0])
bus_strat_sample.head()

#create groups by postal code via groupby
#on all groups, sample one restaurant name
```

[231]:
```
postal_code_5
94102     TURK & LARKIN DELI
94103       THE CHENNAI CLUB
94104                  PLOUF
94105             JUICE SHOP
94107         BAYSIDE MARKET
Name: name, dtype: object
```

[232]:
```
ok.grade("q4c");
```

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

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

1.11.4 Question 4d

What is the probability that AMERICANA GRILL & FOUNTAIN is selected as part of this stratified random sampling procedure?

[233]:
```
q4d_answer = 1/len(bus.loc[bus["postal_code_5"] == '94121'])
q4d_answer

#len(bus["postal_code_5"].unique())
#len(bus.loc[bus["postal_code_5"] == '94121'])
#answer depends on how many stratums there are?
```

[233]:
```
0.00625
```

[234]:
```
bus.loc[bus["name"] == "AMERICANA GRILL & FOUNTAIN"]
```

[234]:
```
     business_id                        name          address           city  \
580         2505  AMERICANA GRILL & FOUNTAIN  3532 BALBOA ST   San Francisco

    state postal_code   latitude   longitude  phone_number postal_code_5
580    CA       94121  37.775806 -122.496608  +14153872893         94121
```

[235]:
```
ok.grade("q4d");
```

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

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

1.11.5 Question 4e

Collect a cluster sample of business IDs, where each cluster is a postal code, with 5 clusters in the sample. Assign `bus_cluster_sample` to a series of business IDs selected by this sampling procedure.

Hint: Consider using `isin`.

[236]:
```
bus_cluster_sample = bus[bus["postal_code_5"].isin(np.random.
 ↪choice(bus["postal_code_5"], size = 5, replace = False))]
bus_cluster_sample.head()

#bus_strat_sample = bus.groupby("postal_code_5")["name"].agg(lambda group:␣
 ↪sample(group, 1)[0])
#unique, sample,
```

[236]:
```
    business_id                name            address           city state  \
3            45  CHARLIE'S DELI CAFE    3202 FOLSOM ST  San Francisco    CA
10           67      REVOLUTION CAFE      3248 22ND ST  San Francisco    CA
17           95                 VEGA  419 CORTLAND AVE  San Francisco    CA
21          102     AKIKO'S SUSHI BAR      542A MASON ST  San Francisco    CA
26          134                 MINT  400 MCALLISTER ST  San Francisco    CA

   postal_code   latitude   longitude  phone_number postal_code_5
3        94110  37.747114 -122.413641  +14156415051         94110
10       94110  37.755419 -122.419542  +14156420474         94110
17       94110  37.739207 -122.417447  +14152856000         94110
21       94102  37.788484 -122.410045  +14159898218         94102
26       94102  37.780247 -122.418974  +14155515942         94102
```

[237]:
```
ok.grade("q4e");
```

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

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

1.11.6 Question 4f

What is the probability that AMERICANA GRILL & FOUNTAIN is selected as part of this cluster sampling procedure?

[238]:
```python
q4f_answer = 5/len(bus["postal_code_5"].unique())
q4f_answer
#q
#srs the cluster
#everything inside the cluster
#stratified is a form of cluster
```

[238]: 0.16666666666666666

[239]:
```python
ok.grade("q4f");
```

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

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

1.11.7 Question 4g

In the context of this question, what are the benefit(s) of performing stratified sampling over cluster sampling? Why would you consider performing cluster sampling instead of stratified sampling? Compare the strengths and weaknesses of both sampling techniques.

Cluster sampling is more cost efficient, because you do not need to travel to multiple zip codes in order to do the sampling. However, it is prone to bias, e.g. if you are sampling an area known for wealthy households. On the other hand, stratified is more expensive but it is less vulnerable to biases because you are sampling from different areas.

1.11.8 Question 4h

Collect a multi-stage sample. First, take a SRS of 5 postal codes. Then, collect an SRS of one business name per selected postal code. Assign `bus_multi_sample` to a series of names selected by this procedure.

[240]:
```python
np.random.seed(40) # Do not touch this!

bus_multi_sample = bus[bus["postal_code_5"].isin(np.random.
 ↪choice(bus["postal_code_5"].unique(), size = 5, replace = False))]

r = bus_multi_sample.groupby("postal_code_5")["name"].agg(lambda group:␣
 ↪sample(group, 1)[0])

r.head()
```

[240]:
```
postal_code_5
94105                       JUICE SHOP
94118     PEABODY ELEMENTARY SCHOOL
94124          THREE BABES BAKESHOP
94133                        WALGREENS
94134                    FAT BELLI DELI
Name: name, dtype: object
```

[241]:
```
ok.grade("q4h");
```

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

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

1.11.9 Question 4i

What is the probability that AMERICANA GRILL & FOUNTAIN is chosen in the multi-stage sample?

[242]:
```
q4i_answer = (5/len(bus["postal_code_5"].unique())) * 1/len(bus.
 ↪loc[bus["postal_code_5"] == '94121'])
q4i_answer

#q
```

[242]:
```
0.0010416666666666667
```

[243]:
```
ok.grade("q4i");
```

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

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

1.12 5: Latitude and Longitude

Let's also consider latitude and longitude values and get a sense of how many are missing.

1.12.1 Question 5a

How many businesses are missing longitude values?

Hint: Use `isnull`.

[244]:
```
num_missing_longs = len(bus[(bus['longitude'].isnull())])
num_missing_longs
```

[244]: 2942

[245]:
```
ok.grade("q5a1");
```

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

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

As a somewhat contrived exercise in data manipulation, let's try to identify which ZIP codes are missing the most longitude values.

Throughout problems 5a and 5b, let's focus on only the "dense" ZIP codes of the city of San Francisco, listed below as `sf_dense_zip`.

[246]:
```
sf_dense_zip = ["94102", "94103", "94104", "94105", "94107", "94108",
                "94109", "94110", "94111", "94112", "94114", "94115",
                "94116", "94117", "94118", "94121", "94122", "94123",
                "94124", "94127", "94131", "94132", "94133", "94134"]
```

In the cell below, create a series where the index is `postal_code_5`, and the value is the number of businesses with missing longitudes in that ZIP code. Your series should be in descending order. Only businesses from `sf_dense_zip` should be included.

Hint: Start by making a new dataframe called `bus_sf` that only has businesses from `sf_dense_zip`.

Hint: Create a custom function to compute the number of null entries in a series, and use this function with the `agg` method.

[247]:
```
bus_sf = bus[bus["postal_code_5"].isin(sf_dense_zip)]
num_missing_in_each_zip = bus_sf[bus_sf["longitude"].isnull()]
```

```
num_missing_in_each_zip = num_missing_in_each_zip.groupby("postal_code_5").
 ↪agg(lambda group: len(group))
num_missing_in_each_zip = num_missing_in_each_zip["longitude"].
 ↪sort_values(ascending = False)
num_missing_in_each_zip


#.set_index("postal_code_5")

#c
```

[247]:
```
postal_code_5
94110    294.0
94103    285.0
94107    275.0
94102    222.0
94109    171.0
94133    159.0
94122    132.0
94111    129.0
94105    127.0
94124    118.0
94118    117.0
94114    111.0
94108     98.0
94115     95.0
94117     86.0
94104     79.0
94112     77.0
94132     71.0
94123     68.0
94121     60.0
94116     42.0
94134     36.0
94127     30.0
94131     16.0
Name: longitude, dtype: float64
```

[248]:
```
ok.grade("q5a2");
```

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

---------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
```

```
[ooooooooook] 100.0% passed
```

1.12.2 Question 5b

In question 5a, we counted the number of null values per ZIP code. Let's now count the proportion of null values.

Create a new dataframe of counts of the null and proportion of null values, storing the result in `fraction_missing_df`. It should have an index called `postal_code_5` and should also have 3 columns:

1. `count null`: The number of missing values for the zip code.
2. `count non null`: The number of present values for the zip code.
3. `fraction null`: The fraction of values that are null for the zip code.

Your data frame should be sorted by the fraction null in descending order.

Recommended approach: Build three series with the appropriate names and data and then combine them into a dataframe. This will require some new syntax you may not have seen. You already have code from question 4a that computes the `null count` series.

To pursue this recommended approach, you might find these two functions useful:

- `rename`: Renames the values of a series.
- `pd.concat`: Can be used to combine a list of Series into a dataframe. Example: `pd.concat([s1, s2, s3], axis=1)` will combine series 1, 2, and 3 into a dataframe.

Hint: You can use the divison operator to compute the ratio of two series.

Hint: The - operator can invert a boolean array. Or alternately, the `notnull` method can be used to create a boolean array from a series.

Note: An alternate approach is to create three aggregation functions and pass them in a list to the `agg` function.

[249]:
```
d0 = bus[bus["postal_code_5"].isin(sf_dense_zip)]
d1 = d0[~bus_sf["longitude"].isnull()]
d1= d1.groupby("postal_code_5").agg(lambda group: len(group))
d1 = d1["longitude"].sort_values(ascending = False)

a = num_missing_in_each_zip
b = d1
c = a /(a+b)

fraction_missing_df = pd.concat([a, b, c], axis=1)
fraction_missing_df.columns = ['count null', 'count non null', 'fraction null']
fraction_missing_df.index.name = "postal_code_5"
#fraction_missing_df.set_index("postal_order_5")
fraction_missing_df = fraction_missing_df.sort_values(by = "fraction null",␣
 ↪ascending = False)
```

```
fraction_missing_df


#fraction_missing_df = pd.DataFrame(column = ["count null", "count non null",␣
↪"fraction null"])
#fraction_missing_df.head()

#q - using - or notnull??
```

```
/srv/conda/envs/data100/lib/python3.6/site-packages/ipykernel_launcher.py:10:
FutureWarning: Sorting because non-concatenation axis is not aligned. A future
version
of pandas will change to not sort by default.

To accept the future behavior, pass 'sort=False'.

To retain the current behavior and silence the warning, pass 'sort=True'.

  # Remove the CWD from sys.path while we load stuff.
```

[249]:

postal_code_5	count null	count non null	fraction null
94124	118.0	73.0	0.617801
94107	275.0	185.0	0.597826
94104	79.0	60.0	0.568345
94105	127.0	105.0	0.547414
94132	71.0	62.0	0.533835
94103	285.0	268.0	0.515371
94114	111.0	112.0	0.497758
94110	294.0	303.0	0.492462
94122	132.0	141.0	0.483516
94102	222.0	241.0	0.479482
94118	117.0	132.0	0.469880
94134	36.0	41.0	0.467532
94111	129.0	148.0	0.465704
94109	171.0	209.0	0.450000
94108	98.0	130.0	0.429825
94116	42.0	57.0	0.424242
94127	30.0	41.0	0.422535
94117	86.0	118.0	0.421569
94112	77.0	118.0	0.394872
94123	68.0	105.0	0.393064
94115	95.0	148.0	0.390947
94121	60.0	100.0	0.375000
94133	159.0	267.0	0.373239
94131	16.0	33.0	0.326531

[250]:
```
ok.grade("q5b");
```

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

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

1.13 Summary of the Business Data

Before we move on to explore the other data, let's take stock of what we have learned and the implications of our findings on future analysis.

- We found that the business id is unique across records and so we may be able to use it as a key in joining tables.
- We found that there are some errors with the ZIP codes. As a result, we dropped the records with ZIP codes outside of San Francisco or ones that were missing. In practive, however, we could take the time to look up the restaurant address online and fix these errors.

- We found that there are a huge number of missing longitude (and latitude) values. Fixing would require a lot of work, but could in principle be automated for records with well-formed addresses.

1.14 6: Investigate the Inspection Data

Let's now turn to the inspection DataFrame. Earlier, we found that `ins` has 4 columns named `business_id`, `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.

[251]:
```
ins.head(5)
```

[251]:

	business_id	score	date	type
0	19	94	20160513	routine
1	19	94	20171211	routine
2	24	98	20171101	routine
3	24	98	20161005	routine
4	24	96	20160311	routine

1.14.1 Question 6a

From calling `head`, we know that each row in this table corresponds to a single inspection. Let's get a sense of the total number of inspections conducted, as well as the total number of unique businesses that occur in the dataset.

[252]:
```
# The number of rows in ins
rows_in_table  = len(ins)

# The number of unique business IDs in ins.
unique_ins_ids = len(bus["business_id"].unique())
```

[253]:
```
ok.grade("q6a");
```

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

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

1.14.2 Question 6b

Next, we examine the Series in the `ins` dataframe called `type`. From examining the first few rows of `ins`, we see that `type` is a string and one of its values is `'routine'`, presumably for a routine inspection. What values does the inspection `type` take? How many occurrences of each value is in the DataFrame? What are the implications for further analysis?

"type" takes str values. 14221 rountine and 1 complaint. I feel that perhaps the data is flawed, considering the routine to complaint ratio is extremely high.

[254]:
```
print(ins["type"].value_counts())
```

```
routine      14221
complaint        1
Name: type, dtype: int64
```

1.14.3 Question 6c

In this question, we're going to try to figure out what years the data span. The dates in our file are formatted as strings such as `20160503`, which are a little tricky to interpret. The ideal solution for this problem is to modify our dates so that they are in an appropriate format for analysis.

In the cell below, we attempt to add a new column to `ins` called `new_date` which contains the `date` stored as a datetime object. This calls the `pd.to_datetime` method, which converts a series of string representations of dates (and/or times) to a series containing a datetime object.

[255]:
```
ins['new_date'] = pd.to_datetime(ins['date'])
ins.head(5)
```

[255]:

	business_id	score	date	type	new_date
0	19	94	20160513	routine	1970-01-01 00:00:00.020160513
1	19	94	20171211	routine	1970-01-01 00:00:00.020171211
2	24	98	20171101	routine	1970-01-01 00:00:00.020171101
3	24	98	20161005	routine	1970-01-01 00:00:00.020161005
4	24	96	20160311	routine	1970-01-01 00:00:00.020160311

As you'll see, the resulting `new_date` column doesn't make any sense. This is because the default behavior of the `to_datetime()` method does not properly process the passed string. We can fix this by telling `to_datetime` how to do its job by providing a format string.

[256]:
```
ins['new_date'] = pd.to_datetime(ins['date'], format='%Y%m%d')
ins.head(5)
```

[256]:

	business_id	score	date	type	new_date
0	19	94	20160513	routine	2016-05-13
1	19	94	20171211	routine	2017-12-11
2	24	98	20171101	routine	2017-11-01
3	24	98	20161005	routine	2016-10-05
4	24	96	20160311	routine	2016-03-11

This is still not ideal for our analysis, so we'll add one more column that is just equal to the year by using the `dt.year` property of the new series we just created.

[257]:
```
ins['year'] = ins['new_date'].dt.year
ins.head()
```

[257]:

	business_id	score	date	type	new_date	year
0	19	94	20160513	routine	2016-05-13	2016
1	19	94	20171211	routine	2017-12-11	2017
2	24	98	20171101	routine	2017-11-01	2017
3	24	98	20161005	routine	2016-10-05	2016
4	24	96	20160311	routine	2016-03-11	2016

[258]:
```
#print(ins["type"].value_counts())

ins.groupby("year").count()

#What range of years is covered in this data set?
#Are there roughly the same number of inspections each year? Provide your␣
↪answer in text only.
```

[258]:

	business_id	score	date	type	new_date
year					
2015	3305	3305	3305	3305	3305
2016	5443	5443	5443	5443	5443
2017	5166	5166	5166	5166	5166
2018	308	308	308	308	308

Now that we have this handy `year` column, we can try to understand our data better.

What range of years is covered in this data set? Are there roughly the same number of inspections each year? Provide your answer in text only.

The years are from 2015 to 2018. There are a varying number of inspections per year, from 308 in 2018 to 5443 in 2016.

1.15 7: Explore Inspection Scores

1.15.1 Question 7a

Let's look at the distribution of inspection scores. As we saw before when we called `head` on this data frame, inspection scores appear to be integer values. The discreteness of this variable means that we can use a barplot to visualize the distribution of the inspection score. Make a bar plot of the counts of the number of inspections receiving each score.

It should look like the image below. It does not need to look exactly the same, but make sure that all labels and axes are correct.

[259]:

```
x = np.sort(ins["score"].unique())
y = ins.groupby("score").size()
plt.bar(x, y)

plt.xlabel("Score")
plt.ylabel("Count")
plt.title("Distribution of Inspection Scores")
```

[259]: `Text(0.5, 1.0, 'Distribution of Inspection Scores')`

[260]:
```
ins
```

[260]:

	business_id	score	date	type	new_date	year
0	19	94	20160513	routine	2016-05-13	2016
1	19	94	20171211	routine	2017-12-11	2017
2	24	98	20171101	routine	2017-11-01	2017
3	24	98	20161005	routine	2016-10-05	2016
4	24	96	20160311	routine	2016-03-11	2016
5	31	98	20151204	routine	2015-12-04	2015
6	45	78	20160104	routine	2016-01-04	2016
7	45	88	20170307	routine	2017-03-07	2017
8	45	85	20170914	routine	2017-09-14	2017
9	45	84	20160614	routine	2016-06-14	2016
10	48	94	20160630	routine	2016-06-30	2016
11	54	100	20150526	routine	2015-05-26	2015
12	54	87	20170215	routine	2017-02-15	2017
13	56	90	20160802	routine	2016-08-02	2016
14	56	92	20170420	routine	2017-04-20	2017
15	56	88	20151222	routine	2015-12-22	2015
16	58	73	20160407	routine	2016-04-07	2016
17	58	70	20170918	routine	2017-09-18	2017
18	61	94	20160708	routine	2016-07-08	2016
19	61	94	20171128	routine	2017-11-28	2017

20	61	98	20170124	routine	2017-01-24	2017
21	61	92	20150827	routine	2015-08-27	2015
22	66	98	20160322	routine	2016-03-22	2016
23	66	100	20150828	routine	2015-08-28	2015
24	66	100	20160902	routine	2016-09-02	2016
25	66	96	20170703	routine	2017-07-03	2017
26	67	90	20150520	routine	2015-05-20	2015
27	67	87	20160401	routine	2016-04-01	2016
28	67	81	20170804	routine	2017-08-04	2017
29	67	94	20161019	routine	2016-10-19	2016
…	…	…	…	…	…	…
14192	93289	83	20171221	routine	2017-12-21	2017
14193	93297	98	20171221	routine	2017-12-21	2017
14194	93352	98	20171027	routine	2017-10-27	2017
14195	93361	90	20171219	routine	2017-12-19	2017
14196	93390	96	20171129	routine	2017-11-29	2017
14197	93423	96	20171103	routine	2017-11-03	2017
14198	93431	89	20171211	routine	2017-12-11	2017
14199	93448	96	20171117	routine	2017-11-17	2017
14200	93465	91	20180104	routine	2018-01-04	2018
14201	93492	96	20180110	routine	2018-01-10	2018
14202	93500	100	20171103	routine	2017-11-03	2017
14203	93532	93	20171103	routine	2017-11-03	2017
14204	93533	92	20171121	routine	2017-11-21	2017
14205	93536	94	20171213	routine	2017-12-13	2017
14206	93549	96	20171221	routine	2017-12-21	2017
14207	93615	89	20171106	routine	2017-11-06	2017
14208	93617	88	20171221	routine	2017-12-21	2017
14209	93815	96	20171102	routine	2017-11-02	2017
14210	93912	94	20180105	routine	2018-01-05	2018
14211	93957	100	20171204	routine	2017-12-04	2017
14212	93959	100	20171218	routine	2017-12-18	2017
14213	93968	98	20171120	routine	2017-11-20	2017
14214	93969	98	20171221	routine	2017-12-21	2017
14215	93977	96	20171219	routine	2017-12-19	2017
14216	94012	100	20171220	routine	2017-12-20	2017
14217	94012	90	20180112	routine	2018-01-12	2018
14218	94133	100	20171227	routine	2017-12-27	2017
14219	94142	100	20171220	routine	2017-12-20	2017
14220	94189	96	20171130	routine	2017-11-30	2017
14221	94231	85	20171214	routine	2017-12-14	2017

[14222 rows x 6 columns]

1.15.2 Question 7b

Describe the qualities of the distribution of the inspections scores based on your bar plot. Consider the mode(s), symmetry, tails, gaps, and anamolous values. Are there any unusual features of this distribution? What do your observations imply about the scores?

The mode is surprisingly a value of 100. There is not much symmetry to the graph as it tends to the right side. The tail ends show that it is actually harder to get a very low score rather than a very high school. There are no real notable gaps, but it is interesting that the count for the highest score range is approximately double that of the second. There's not much unusual to me about the graph except that the highest score range has the highest count as well. To me, this seems like the scale should be adjusted to be a bit more strict

1.15.3 Question 7c

Let's figure out which restaurants had the worst scores ever. 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. If a `business_id` in `ins` does not exist in `bus`, the name and address should be given as NaN.

Hint: Use the merge method to join the `ins` *dataframe with the appropriate portion of the* `bus` *dataframe.*

[321]:
```python
ins_named = pd.merge(ins, bus, how = 'left', on=["business_id"])


#['business_id', 'score', 'date', 'type', 'new_date', 'year', 'name', 'address']

ins_named = ins_named[['business_id','score', 'date', 'type', 'new_date',␣
↪'year', 'name', 'address']]
ins_named = ins_named.sort_values(by = "score", ascending = True)
ins_named.head()
```

[321]:
```
       business_id  score      date     type   new_date  year  \
13179        86647     48  20160907  routine 2016-09-07  2016
9476         71373     52  20161031  routine 2016-10-31  2016
8885         69199     53  20170127  routine 2017-01-27  2017
7104         61436     54  20150706  routine 2015-07-06  2015
2192          3459     54  20150407  routine 2015-04-07  2015

                                name          address
13179                        DA CAFE   407 CLEMENT ST
9476            GOLDEN RIVER RESTAURANT  5827 GEARY BLVD
8885           MEHFIL INDIAN RESTAURANT      28 02ND ST
7104     OZONE THAI RESTAURANT AND LOUNGE     598 02ND ST
2192          BASIL THAI RESTAURANT & BAR   1175 FOLSOM ST
```

[299]:
```python
#len(ins)
len(ins_named)
```

[299]: 14222

[300]:
```python
ok.grade("q7c1");
```

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

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

Using this data frame, identify the restaurant with the lowest inspection scores ever. Head to yelp.com and look up the reviews page for this restaurant. Copy and paste anything interesting you want to share.

Da Cafe has the lowest inspection score ever, with Golden River Restaurant coming in second. Interesting enough, DA cafe is still 3 stars on yelp.

Just for fun you can also look up the restaurants with the best scores. You'll see that lots of them aren't restaurants at all!

1.16 8: Restaurant Ratings Over Time

Let's consider various scenarios involving restaurants with multiple ratings over time.

1.16.1 Question 8a

Let's see which restaurant has had the most extreme improvement in its rating. Let the "swing" of a restaurant be defined as the difference between its highest and lowest rating ever. **Only consider restaurants with at least 3 ratings!** Using whatever technique you want to use, assign `max_swing` to the name of restaurant that has the maximum swing.

[264]:
```python
#new_ins = ins.set_index("business_id")
#r = new_ins[ins["date"].groupby(ins["business_id"]).agg(len) >= 3]
#new_ins = news_ins.reset_index()
#new_ins = new_ins["score"].groupby(new_ins["business_id"]).agg(lambda score:␣
 ↪max(score) - min(score))
#new_ins.max()

new_ins = ins.set_index("business_id")
```

```
r = new_ins[ins["date"].groupby(ins["business_id"]).agg(len) >= 3]
r = r.reset_index()
r = r["score"].groupby(r["business_id"]).agg(lambda score: max(score) -␣
↪min(score))
#r.max()
#max_swing = r.sort_values("score", ascending = False)
b = r.sort_values(ascending = False).index[0]
#max_swing = bus[bus["business_id"] == b]["name"]

max_swing = bus[bus["business_id"] == b]["name"].iloc[0]
#max_swing

#q
```

```
/srv/conda/envs/data100/lib/python3.6/site-packages/ipykernel_launcher.py:8:
UserWarning: Boolean Series key will be reindexed to match DataFrame index.
```

[265]:
```
ok.grade("q8a1");
```

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

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

1.16.2 Question 8b

To get a sense of the number of times each restaurant has been inspected, create a multi-indexed dataframe called `inspections_by_id_and_year` where each row corresponds to data about a given business in a single year, and there is a single data column named `count` that represents the number of inspections for that business in that year. The first index in the MultiIndex should be on `business_id`, and the second should be on `year`.

An example row in this dataframe might look tell you that business_id is 573, year is 2017, and count is 4.

Hint: Use groupby to group based on both the `business_id` and the `year`.

Hint: Use rename to change the name of the column to `count`.

[266]:
```
num = ins.groupby(["business_id", "year"]).count().drop(columns = ["date",␣
↪"type", "new_date"])
num = num.rename(columns = {"score": "count"})
```

```
inspections_by_id_and_year = num
inspections_by_id_and_year.head()
```

[266]:
```
                  count
business_id year
19          2016      1
            2017      1
24          2016      2
            2017      1
31          2015      1
```

[267]:
```
ok.grade("q8b");
```

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

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

You should see that some businesses are inspected many times in a single year. Let's get a sense of the distribution of the counts of the number of inspections by calling `value_counts`. There are quite a lot of businesses with 2 inspections in the same year, so it seems like it might be interesting to see what we can learn from such businesses.

[268]:
```
inspections_by_id_and_year['count'].value_counts()
```

[268]:
```
1    9531
2    2175
3     111
4       2
Name: count, dtype: int64
```

1.16.3 Question 8c

What's the relationship between the first and second scores for the businesses with 2 inspections in a year? Do they typically improve? For simplicity, let's focus on only 2016 for this problem.

First, make a dataframe called `scores_pairs_by_business` indexed by `business_id` (containing only businesses with exactly 2 inspections in 2016). This dataframe contains the field `score_pair` consisting of the score pairs ordered chronologically `[first_score, second_score]`.

Plot these scores. That is, make a scatter plot to display these pairs of scores. Include on the plot a reference line with slope 1.

You may find the functions `sort_values`, `groupby`, `filter` and `agg` helpful, though not all necessary.

The first few rows of the resulting table should look something like:

```
<tr style="text-align: right;">
  <th></th>
  <th>score_pair</th>
</tr>
<tr>
  <th>business_id</th>
  <th></th>
</tr>

<tr>
  <th>24</th>
  <td>[96, 98]</td>
</tr>
<tr>
  <th>45</th>
  <td>[78, 84]</td>
</tr>
<tr>
  <th>66</th>
  <td>[98, 100]</td>
</tr>
<tr>
  <th>67</th>
  <td>[87, 94]</td>
</tr>
<tr>
  <th>76</th>
  <td>[100, 98]</td>
</tr>
```

The scatter plot should look like this:

Note: Each score pair must be a list type; numpy arrays will not pass the autograder.

Hint: Use the `filter` method from lecture 3 to create a new dataframe that only contains restaurants that received exactly 2 inspections.

Hint: Our answer is a single line of code that uses `sort_values`, `groupby`, `filter`, `groupby`, `agg`, and `rename` in that order. Your answer does not need to use these exact methods.

[329]:
```python
def l(series):
    x = series.iloc[0]
    y = series.iloc[1]
    return [x,y]
#scores_pairs_by_business =
ins2016 = ins[ins['year'] == 2016]
```

```
ins2016 = ins2016.sort_values("date").groupby("business_id").filter(lambda␣
↪group: len(group) == 2).groupby("business_id").agg(l)
ins2016 = ins2016.drop(columns = ["new_date", "year", "date", "type"])
scores_pairs_by_business = ins2016
scores_pairs_by_business.columns = ['score_pair']
```

[ ]:

[322]:
```
ok.grade("q8c1");
```

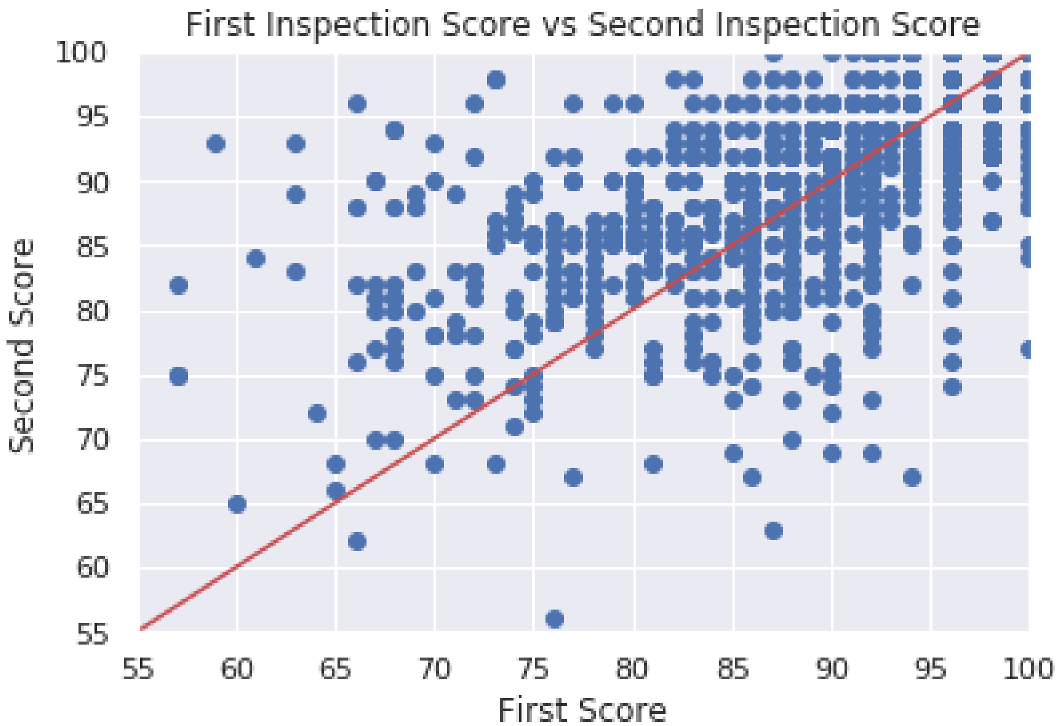
```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

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

Now, create your scatter plot in the cell below.

[328]:
```
plt.scatter(scores_pairs_by_business['score_pair'].str[0],␣
↪scores_pairs_by_business['score_pair'].str[1])
plt.ylim(55, 100)
plt.xlim(55, 100)
plt.xticks(np.arange(55, 105, step=5));
plt.yticks(np.arange(55, 105, step=5));
plt.xlabel("First Score")
plt.ylabel("Second Score")
plt.title("First Inspection Score vs Second Inspection Score")

plt.plot([55, 100],[55, 100], 'r');
```



1.16.4 Question 8d

Another way to compare the scores from the two inspections is to examine the difference in scores. Subtract the first score from the second in `scores_pairs_by_business`. Make a histogram of these differences in the scores. We might expect these differences to be positive, indicating an improvement from the first to the second inspection.

The histogram should look like this:

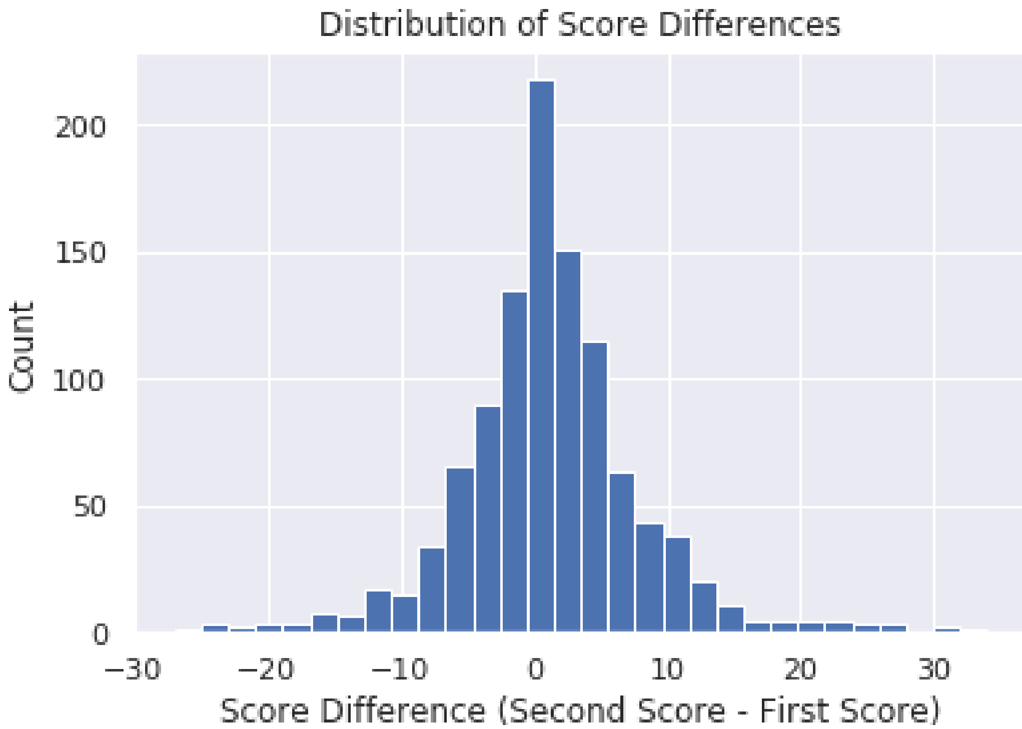
Hint: Use `second_score` and `first_score` created in the scatter plot code above.

Hint: Convert the scores into numpy arrays to make them easier to deal with.

Hint: Try changing the number of bins when you call plt.hist.

[324]:
```
x = [score[0] for score in scores_pairs_by_business['score_pair']]
y = [score[1] for score in scores_pairs_by_business['score_pair']]
score_difference = [y[score]-x[score] for score in range(len(x))]
plt.hist(score_difference, bins = 30)
plt.title("Distribution of Score Differences")
plt.xlabel("Score Difference (Second Score - First Score)")
plt.ylabel("Count")
```

[324]:
```
Text(0, 0.5, 'Count')
```



1.16.5 Question 8e

If a restaurant's score improves from the first to the second inspection, what do you expect to see in the scatter plot that you made in question 8c? What do you see?

If a restaurant's score improves from the first to the second inspection, how would this be reflected in the histogram of the difference in the scores that you made in question 8d? What do you see?

If the score improves, I would expect the corresponding dot to be above the linear line. This is what I see from the plot. For the histogram, if the score improves, it should be on the right side of the graph. I see that what I expected is true.

1.17 Summary of the Inspections Data

What we have learned about the inspections data? What might be some next steps in our investigation?

- We found that the records are at the inspection level and that we have inspections for multiple years.

- We also found that many restaurants have more than one inspection a year.
- By joining the business and inspection data, we identified the name of the restaurant with the worst rating and optionally the names of the restaurants with the best rating.

- We identified the restaurant that had the largest swing in rating over time.
- We also examined the relationship between the scores when a restaurant has multiple inspections in a year. Our findings were a bit counterintuitive and may warrant further investigation.

1.18 Congratulations!

You are finished with Project 1. You'll need to make sure that your PDF exports correctly to receive credit. Run the following cell and follow the instructions.

[330]:
```
# Save your notebook first, then run this cell to submit.
import jassign.to_pdf
jassign.to_pdf.generate_pdf('proj1.ipynb', 'proj1.pdf')
ok.submit()
```

```
Generating PDF…
Saved proj1.pdf
```

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

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

```
Saving notebook… Saved 'proj1.ipynb'.
Submit… 100% complete
Submission successful for user: david-lin@berkeley.edu
URL: https://okpy.org/cal/data100/sp19/proj1/submissions/K1OpXG
```