Homework 2: Food Safety

Cleaning and Exploring Data with Pandas

Due Date: Friday 10/04, 11:59 PM

Collaboration Policy

Data science is a collaborative activity. While you may talk with others about the homework, 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.

Submission Instructions

Please save this Jupyter notebook as a PDF and upload it to Gradescope. Additionally, please leave a copy of this assignment in its original name and location in your JupyterHub, such that we can collect the runnable version of the notebook from there.

This assignment

In this homework, you will investigate restaurant food safety scores for restaurants in San Francisco. Here 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. We have made these data available to you in `class_share` on JupyterHub along with the link below. The main goal for this assignment is to understand how restaurants are scored. We will walk through the 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.
- Exploring characteristics and distributions of individual variables

Score breakdown

Question	Points

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
4a	2
4b	3
5a	1
5b	1
5c	1
6a	2
6b	3
6c	3
7a	2
7b	2
7c	6
7d	2

In [ ]:

In [ ]:
```
```

To start the assignment, run the cell below to set up some imports. In many of these assignments (and your future adventures as a data scientist) you will use `os`, `zipfile`, `pandas`, `numpy`, `matplotlib.pyplot`, and `seaborn`. Import each of these libraries `as` their commonly used abbreviations (e.g., `pd`, `np`, `plt`, and `sns`).

In [2]:
```
import os
import zipfile
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
sns.set()
```

In [3]:
```
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
assert 'seaborn'in sys.modules and sns
```

Downloading the data

For this assignment, we need this data file: https://cims.nyu.edu/~policast/hw2-SFBusinesses.zip (https://cims.nyu.edu/~policast/hw2-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 ds112_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 file. A `pathlib.Path` is an object that stores filepaths, e.g. `~/Dropbox/ds112/my_chart.png` .

The code below uses `ds112_utils.py` to download the data from the following URL: https://cims.nyu.edu/~policast/hw2-SFBusinesses.zip (https://cims.nyu.edu/~policast/hw2-SFBusinesses.zip)

In [4]:
```
import ds112_utils

source_data_url = 'https://cims.nyu.edu/~policast/hw2-SFBusinesses.zip'
target_file_name = 'data.zip'
data_dir = '.'

# Change the force=False -> force=True in case you need to force redownload the data
dest_path = ds112_utils.fetch_and_cache(data_url=source_data_url, data_dir=data_dir, file=target_file_name, f
orce=True)
```

```
Downloading... Done!
```

After running the code, if you look at the directory containing hw02.ipynb, you should see data.zip.

1: Loading Food Safety Data

Alright, great, now we have `data.zip`. We don't have any specific questions yet, so let's focus on understanding the structure of the data. Recall 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 the zip file. We could in theory do this by manually opening up the zip file on our computers or using a shell command like `!unzip`, but on this homework we're going to do almost everything in Python for maximum portability and automation.

Goal: Fill in the code below so that `my_zip` is a `zipfile.ZipFile` object corresponding to the downloaded zip file, and so that `list_names` contains a list of the names of all files inside the downloaded zip file.

Creating a `zipfile.ZipFile` object is a good start (the Python docs (https://docs.python.org/3/library/zipfile.html) have further details). You might also look back at the code from the case study from Demo 3. It's OK to copy and paste code from the Demo 3 file, though you might get more out of this exercise if you type out an answer.

Question 1a: Looking Inside and Extracting the Zip Files

In [5]:
```
# Fill in the list_names variable with a list of all the names of the files in the zip file

### BEGIN SOLUTION
import zipfile
my_zip = zipfile.ZipFile(target_file_name, 'r')
list_names = [f.filename for f in my_zip.filelist]
### END SOLUTION
```

The cell below will test that your code is correct.

In [6]:
```
assert isinstance(my_zip, zipfile.ZipFile)
assert isinstance(list_names, list)
assert all([isinstance(file, str) for file in list_names])
print(True)
```

```
True
```

In your answer above, if you see something like `zipfile.ZipFile('data.zip'...` , we suggest changing it to read `zipfile.ZipFile(dest_path...` or alternately `zipfile.ZipFile(target_file_name...` . In general, we **strongly suggest having your filenames hard coded ONLY ONCE** in any given iPython 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 very hard to find bugs.

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.

We expect an output that looks something like this:

```
violations.csv  3726206

businesses.csv  660231

inspections.csv 466106

legend.csv  120
```

In [7]:
```
### BEGIN SOLUTION
file_list = [f for f in my_zip.filelist]

for file in file_list:
    print(file.filename, file.file_size)

### END SOLUTION
```

```
violations.csv 3726206
businesses.csv 660231
inspections.csv 466106
legend.csv 120
```

Often when working with zipped data, we'll never unzip the actual zipfile. This saves space on our local computer. However, for this HW, 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 more deeply understanding what's going on. The cell below will unzip the csv files into a subdirectory called "data". Try running the code below.

In [8]:
```
from pathlib import Path
data_dir = Path('data')
my_zip.extractall(data_dir)
```

When you ran the code above, nothing gets printed. However, this code should have created a folder called "data", and in it should be the four CSV files. Assuming you're using Datahub, use your web browser to verify that these files were created, and try to open up `legend.csv` to see what's inside. 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"
```

Question 1b: Programatically Looking Inside the Files

What we see when we opened the file above is good news! It looks like this file is indeed a csv file. Let's check the other three files. This time, rather than opening up the files manually, let's use Python to print out the first 5 lines of each. The `ds112_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 `ds112_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.

In [9]:
```
### BEGIN SOLUTION
print(ds112_utils.head('data/businesses.csv', 5))
print(' ')
print(ds112_utils.head('data/inspections.csv', 5))
print(' ')
print(ds112_utils.head('data/legend.csv', 5))
print(' ')
print(ds112_utils.head('data/violations.csv', 5))
print(' ')
### END SOLUTION
```

```
['"business_id","name","address","city","state","postal_code","latitude","longitude","phone_number"\n', '1
9,"NRGIZE LIFESTYLE CAFE","1200 VAN NESS AVE, 3RD FLOOR","San Francisco","CA","94109","37.786848","-122.42154
7","+14157763262"\n', '24,"OMNI S.F. HOTEL - 2ND FLOOR PANTRY","500 CALIFORNIA ST, 2ND  FLOOR","San Francisc
o","CA","94104","37.792888","-122.403135","+14156779494"\n', '31,"NORMAN\'S ICE CREAM AND FREEZES","2801 LEAV
ENWORTH ST ","San Francisco","CA","94133","37.807155","-122.419004",""\n', '45,"CHARLIE\'S DELI CAFE","3202 F
OLSOM ST ","San Francisco","CA","94110","37.747114","-122.413641","+14156415051"\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']

['"Minimum_Score","Maximum_Score","Description"\n', '0,70,"Poor"\n', '71,85,"Needs Improvement"\n', '86,90,"A
dequate"\n', '91,100,"Good"\n']

['"business_id","date","description"\n', '19,"20171211","Inadequate food safety knowledge or lack of certifie
d food safety manager"\n', '19,"20171211","Unapproved or unmaintained equipment or utensils"\n', '19,"2016051
3","Unapproved or unmaintained equipment or utensils  [ date violation corrected: 12/11/2017 ]"\n', '19,"2016
0513","Unclean or degraded floors walls or ceilings  [ date violation corrected: 12/11/2017 ]"\n']
```

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` .

In [10]:
```
# path to directory containing data
import pandas as pd
dsDir = Path('data')

### BEGIN SOLUTION
# Make sure to use these names
bus = pd.read_csv('data/businesses.csv', encoding='ISO-8859-1')
vio = pd.read_csv('data/violations.csv')
ins = pd.read_csv('data/inspections.csv')


### END SOLUTION
```

Now that you've read in the files, let's try some `pd.DataFrame` methods. Use the `DataFrame.head` command to show the top few lines of the `bus`, `ins`, and `vio` dataframes.

In [11]:
```
### BEGIN SOLUTION
bus.head
ins.head
vio.head


### END SOLUTION
```

Out[11]:
```
<bound method NDFrame.head 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]>
```

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.

In [12]:
```
### BEGIN SOLUTION
print(bus.describe)
print(ins.describe)
print(vio.describe)

### END SOLUTION
```

```
<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]>
```

Question 1d: Verify Your Files were Read Correctly

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:

In [13]:
```
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
print(True)
```

```
True
```

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

In [14]:
```
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']})

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.

Do not delete the empty cell below!

In [15]:
```
"""Run this cell to load this utility comparison function that we will use in various
tests below.

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`).
    """
    import numpy.testing as npt

    # 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]

    npt.assert_allclose(act, des, rtol)
```

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!

In [16]:
```
# These tests will raise an exception if your variables don't match numerically the correct
# answers in the main summary statistics shown above.
df_allclose(bus, bus_summary)
df_allclose(ins, ins_summary)
df_allclose(vio, vio_summary)
print("Passed")
```

```
Passed
```

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.

In [17]:
```
### BEGIN SOLUTION
# print(bus.head)
# print(ins.head)
# print(vio.head)
q1e_answer = r"""

There are issues with missing values and bad data within the three files. An example is the missing phone num
bers,
latitudes, and logitudes in one file. This means that there was bad data collected, as the company location i
s
unknown, but the phone number is known.

It's also very strange that there are so many good scores given, but a lot of negative reviews, which are cut
short.

There was a section of the data around the end where there was nothing being measured, so it seems unnecessar
y.

There are three entries of 'Ca' and 3 'CA' values.
Additionally, there are some extended postal codes that are 9 digits long, rather than the typical 5 digits


"""
### END SOLUTION

print(q1e_answer)
```

```
There are issues with missing values and bad data within the three files. An example is the missing phone num
bers,
latitudes, and logitudes in one file. This means that there was bad data collected, as the company location i
s
unknown, but the phone number is known.

It's also very strange that there are so many good scores given, but a lot of negative reviews, which are cut
short.

There was a section of the data around the end where there was nothing being measured, so it seems unnecessar
y.

There are three entries of 'Ca' and 3 'CA' values.
Additionally, there are some extended postal codes that are 9 digits long, rather than the typical 5 digits
```

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

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 have imperfect answers. Please be sure to check your results carefully.

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".

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

In [18]:
```
### BEGIN SOLUTION
is_business_id_unique = bus['business_id'].value_counts()# put your final answer True/False here
is_business_id_unique = bus['business_id'].unique()


#ANSWER IS TRUE, SINCE THERE EXIST NO UNIQUE business_id entries that are duplicates.


### END SOLUTION
```

In [19]:
```
assert is_business_id_unique.all()
print("passed")
```

```
passed
```

Question 2b

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

1. How many records are there?
2. What does each record represent (e.g., a store, a chain, a transaction)?
3. What is the primary key?

Please write your answer in the `q2b_answer` variable. You may create new cells to run code as long as you don't delete the cell below.

In [20]:
```
# use this cell for scratch work
# consider using groupby or value_counts() on the 'name' or 'business_id'

### BEGIN SOLUTION


### END SOLUTION
```

In [21]:
```
q2b_answer = r"""

Put your answer here, replacing this text.

"""

### BEGIN SOLUTION
q2b_answer = r"""

1. The number of records within the file is 6406 records
2. Each record represents a certain restaurant (store), and doesn't determine if it's a chain or the number o
f transactions.
3. The primary key, or attributes identify the record, is the business_id


"""
### END SOLUTION

print(q2b_answer)
```

```
1. The number of records within the file is 6406 records
2. Each record represents a certain restaurant (store), and doesn't determine if it's a chain or the number o
f transactions.
3. The primary key, or attributes identify the record, is the business_id
```

3: Zip code

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

Question 3a

What kind of values are in the `postal code` column in the `bus` data frame?

1. Are zip codes quantitative or qualitative? If qualitative, is it ordinal or nominal?
2. How are the zip code values encoded in python: ints, floats, strings, booleans ...?

To answer the second question you might want to examine a particular entry using the Python `type` command.

In [22]:
```
# Use this cell for your explorations.

print(type(bus.loc[2,'postal_code']))
if isinstance(bus.loc[2,'postal_code'], str):
    print(True)

### BEGIN SOLUTION
q3a_answer = r"""

1. Zip codes are nominal qualitative or categorical values.
2. Zip code values are encoded using str

"""
### END SOLUTION

print(q3a_answer)
```

```
<class 'str'>
True


1. Zip codes are nominal qualitative or categorical values.
2. Zip code values are encoded using str
```

Question 3b

To explore the zip code values, it makes sense to examine counts, i.e., the number of records that have the same zip code value. This is essentially answering the question: 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 businesses in that postal code. For example, in 94110 (hey that's my old zip code!), there should be 596 businesses. Your series should be in descending order, i.e. 94110 should be at the top.

For this answer, use `groupby`, `size`, and `sort_values`.

In [23]:
```
### BEGIN SOLUTION
zip_counts = bus.groupby("postal_code").size().sort_values(ascending=False)
print(zip_counts.head())
### END SOLUTION
```

```
postal_code
94110    596
94103    552
94102    462
94107    460
94133    426
dtype: int64
```

Unless you know pandas well already, your answer probably has one subtle flaw in it: it fails to take into account businesses with missing zip codes. Unfortunately, missing data is just a reality when we're working with real data.

There are a couple of ways to include null postal codes in the zip_counts series above. One approach is to use `fillna`, which will replace all null (a.k.a. NaN) values with a string of our choosing. In the example below, I picked "?????". When you run the code below, you should see that there are 240 businesses with missing zip code.

In [24]:
```
zip_counts = bus.fillna("?????").groupby("postal_code").size().sort_values(ascending=False)
zip_counts.head(15)
```

Out[24]:
```
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.

In [25]:
```
bus["postal_code"].value_counts(dropna=False).sort_values(ascending = False).head(15)
```

Out[25]:
```
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 is also some bad data where the postal code got messed up, 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 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 because 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.

In [26]:
```
# Run me
bus['postal_code_5'] = bus['postal_code'].str[:5]
bus
```

Out[26]:

	business_id	name	address	city	state	postal_code	latitude	longitude	phone_number	postal_code
0	19	NRGIZE LIFESTYLE CAFE	1200 VAN NESS AVE, 3RD FLOOR	San Francisco	CA	94109	37.786848	-122.421547	+14157763262	941
1	24	OMNI S.F. HOTEL - 2ND FLOOR PANTRY	500 CALIFORNIA ST, 2ND FLOOR	San Francisco	CA	94104	37.792888	-122.403135	+14156779494	941
2	31	NORMAN'S ICE CREAM AND FREEZES	2801 LEAVENWORTH ST	San Francisco	CA	94133	37.807155	-122.419004	NaN	941
3	45	CHARLIE'S DELI CAFE	3202 FOLSOM ST	San Francisco	CA	94110	37.747114	-122.413641	+14156415051	941
4	48	ART'S CAFE	747 IRVING ST	San Francisco	CA	94122	37.764013	-122.465749	+14156657440	941
5	54	RHODA GOLDMAN PLAZA	2180 POST ST	San Francisco	CA	94115	37.784626	-122.437734	+14153455060	941
6	56	CAFE X + O	1799 CHURCH ST	San Francisco	CA	94131	37.742325	-122.426476	+14158263535	941
7	58	OASIS GRILL	91 DRUMM ST	San Francisco	CA	94111	37.794483	-122.396584	+14158341942	941
8	61	CHOWDERS	PIER 39 SPACE A3	San Francisco	CA	94133	37.808240	-122.410189	+14153914737	941
9	66	STARBUCKS COFFEE	1800 IRVING ST	San Francisco	CA	94122	37.763578	-122.477461	+14152427970	941
10	67	REVOLUTION CAFE	3248 22ND ST	San Francisco	CA	94110	37.755419	-122.419542	+14156420474	941
11	73	DINO'S UNCLE VITO	2101 FILLMORE ST	San Francisco	CA	94115	37.788932	-122.433895	+14159224700	941
12	76	OMNI S.F. HOTEL - 3RD FLOOR PANTRY	500 CALIFORNIA ST, 3RD FLOOR	San Francisco	CA	94104	37.792888	-122.403135	+14156779494	941
13	77	OMNI S.F. HOTEL - EMPLOYEE CAFETERIA	500 CALIFORNIA ST, BASEMENT	San Francisco	CA	94104	37.792888	-122.403135	+14156779494	941

	business_id	name	address	city	state	postal_code	latitude	longitude	phone_number	postal_code
14	80	LAW SCHOOL CAFE	2199 FULTON ST	San Francisco	CA	94117	37.774941	-122.452797	+14154222268	941
15	81	CLUB ED/BON APPETIT	2350 TURK ST	San Francisco	CA	94117	37.778468	-122.448484	+14154225849	941
16	88	J.B.'S PLACE	1435 17TH ST	San Francisco	CA	94107	37.765003	-122.398084	+14155848446	941
17	95	VEGA	419 CORTLAND AVE	San Francisco	CA	94110	37.739207	-122.417447	+14152856000	941
18	98	XOX TRUFFLES	754 COLUMBUS AVE	San Francisco	CA	94133	37.801665	-122.412104	+14154214814	941
19	99	J & M A-1 CAFE RESTAURANT LLC	779 CLAY ST	San Francisco	CA	94108	37.794293	-122.405967	+14156057219	941
20	101	CABLE CAR CORNER	1099 POWELL ST	San Francisco	CA	94108	37.794615	-122.409705	+14153625925	941
21	102	AKIKO'S SUSHI BAR	542A MASON ST	San Francisco	CA	94102	37.788484	-122.410045	+14159898218	941
22	108	RUE LEPIC	900 PINE ST	San Francisco	CA	94108	37.790868	-122.410854	+14154746070	941
23	116	THE WATERFRONT RESTAURANT	PIER 7 EMBARCADERO	San Francisco	CA	94111	37.793874	-122.396464	+14153912696	941
24	121	AKIKOS SUSHI	431 BUSH ST	San Francisco	CA	94108	37.790643	-122.404676	+14153973218	941
25	125	CENTERFOLDS	391 BROADWAY ST	San Francisco	CA	94133	37.798233	-122.403637	+14158340662	941
26	134	MINT	400 MCALLISTER ST	San Francisco	CA	94102	37.780247	-122.418974	+14155515942	941
27	140	CAFE MADELEINE	300 CALIFORNIA ST	San Francisco	CA	94104	37.793268	-122.400323	+14153623332	941
28	141	AFC SUSHI @ MOLLIE STONE'S 2	2435 CALIFORNIA ST	San Francisco	CA	94115	37.788773	-122.434697	+14155674902	941
29	146	DEJA VU PIZZA & PASTA	3227 16TH ST	San Francisco	CA	94103	37.764713	-122.424709	+14152551600	941
...	...	...	...	...	...	...	...	...	...	

	business_id	name	address	city	state	postal_code	latitude	longitude	phone_number	postal_code
6376	94305	ROSAMUNDE SAUSAGE GRILL	545 HAIGHT ST	San Francisco	CA	94117	NaN	NaN	+14154376851	941
6377	94310	YOKAI EXPRESS	135 4TH ST	San Francisco	CA	94103	NaN	NaN	+14158234502	941
6378	94318	YUANBAO JIAOZI	2110 IRVING ST	San Francisco	CA	94122	NaN	NaN	+14156013979	941
6379	94331	MATCHA CAFE MAIKO	1581 WEBSTER ST 175	San Francisco	CA	94115	NaN	NaN	+14150009434	941
6380	94334	SUBWAY SANDWICHES #53761	160 BROADWAY ST	San Francisco	CA	94111	NaN	NaN	+14158861913	941
6381	94337	SUBWAY SANDWICHES #61240	425 D BATTERY ST	San Francisco	CA	94111	NaN	NaN	+14153991549	941
6382	94354	RAINBOW MARKET AND DELI	684 LARKIN ST	San Francisco	CA	94109	NaN	NaN	+14157664681	941
6383	94387	FOUNDATION CAFE	645 5TH ST	San Francisco	CA	94107	NaN	NaN	+14153503301	941
6384	94388	FOUNDATION CAFE	335 KEARNY ST	San Francisco	CA	94108	NaN	NaN	NaN	941
6385	94394	KOKIO REPUBLIC	428 11TH ST	San Francisco	CA	94109	NaN	NaN	+14157996404	941
6386	94408	SIZZLING POT KING	139 8TH ST	San Francisco	CA	94103	NaN	NaN	+14158028899	941
6387	94409	AUGUST HALL	420 MASON ST	San Francisco	CA	NaN	NaN	NaN	NaN	Na
6388	94412	NATIVE BAKING COMPANY	1324 FITZGERALD AVE	San Francisco	CA	94124	NaN	NaN	NaN	941
6389	94433	GREEK TOWN LLC	88 02ND ST	San Francisco	CA	94105	NaN	NaN	+14152408032	941
6390	94442	SIMPLY CAFE	340 GROVE ST	San Francisco	CA	94102	NaN	NaN	+14156587659	941
6391	94456	UBER-ATG (BON APPETIT)	581 20TH ST 2ND FL	San Francisco	CA	94107	NaN	NaN	+14158184997	941

	business_id	name	address	city	state	postal_code	latitude	longitude	phone_number	postal_code
6392	94460	DOBBS FERRY	409 GOUGH ST	San Francisco	CA	94102	NaN	NaN	+14155517709	941
6393	94465	BEAUTIFULL LLC	3401 CALIFORNIA ST	San Francisco	CA	94118	NaN	NaN	+14157289080	941
6394	94468	BAR CRENN	3131 FILLMORE ST	San Francisco	CA	94123	NaN	NaN	NaN	941
6395	94502	NEW FORTUNE DIM SUM	811 STOCKTON ST	San Francisco	CA	94108	NaN	NaN	+14153991511	941
6396	94521	JOE & THE JUICE HOWARD	301 HOWARD ST	San Francisco	CA	94105	NaN	NaN	NaN	941
6397	94522	CAFE JOSEPHINE	199 MUSEUM WAY	San Francisco	CA	94114	NaN	NaN	+14153508976	941
6398	94537	BON APPETIT @ USF- OUTTA HERE	2130 FULTON ST	San Francisco	CA	94117	NaN	NaN	+14153604802	941
6399	94540	FOAM USA LLC	1745 TARAVAL ST	San Francisco	CA	94116	NaN	NaN	+14156060018	941
6400	94542	OCEAN THAI	2545 OCEAN AVE	San Francisco	CA	94132	NaN	NaN	+14155857251	941
6401	94544	D'MAIZE CAFE	50 PHELAN AVE	San Francisco	CA	94112	NaN	NaN	+14154240604	941
6402	94555	EASY BREEZY FROZEN YOGURT	44 WEST PORTAL AVE	San Francisco	CA	94127	NaN	NaN	+14155053351	941
6403	94571	THE PHOENIX PASTIFICIO	200 CLEMENT ST	San Francisco	CA	94118	NaN	NaN	+14154726100	941
6404	94572	BROADWAY DIM SUM CAFE	684 BROADWAY ST	San Francisco	CA	94133	NaN	NaN	NaN	941
6405	94574	BINKA BITES	2241 GEARY BLVD	San Francisco	CA	94115	NaN	NaN	+14157712907	941

6406 rows × 10 columns

Question 3c : A Closer Look at Missing Zip Codes

Let's look more closely at businesses with missing zip codes. We'll see that many zip codes are missing for a good reason. Examine the businesses with missing zipcode values. Pay attention to their addresses. Do you notice anything interesting? You might need to look at a bunch of entries, i.e. don't just look at the first five.

Hint: You can use the series `isnull` method to create a binary array, which can then be used to show only rows of the dataframe that contain null values.

In [27]:
```
# Use this cell for your explorations.

null_vals = bus['postal_code'].isnull()

### BEGIN SOLUTION
q3c_answer = r"""

After using the isnull method and looking more closely at a bunch of entries of businesses with missing zip c
odes,
I see that many zip codes are missing. Besides some missing latitude and longitude measurements of some resta
urants,
looking at their addresses, I notice that they are not very clear as they are listed under a person's name, a
re a children's
center, are a neighborhood center, or are not shown at all. It's very possible that these are street vendors
 or are bodega,
bazaar, or marketplace restaurants that don't get mailed to.

This is because zip codes are used for understanding trends, running businesses, and finding new ways to reac
h you.
They are also used for make major decisions for marketing, opening or closing stores, providing services,
and making decisions that can have a massive financial impact. They correspond to mail delivery routes,
not a geographic area.


"""
### END SOLUTION

print(q3c_answer)
```

```
After using the isnull method and looking more closely at a bunch of entries of businesses with missing zip c
odes,
I see that many zip codes are missing. Besides some missing latitude and longitude measurements of some resta
urants,
looking at their addresses, I notice that they are not very clear as they are listed under a person's name, a
re a children's
center, are a neighborhood center, or are not shown at all. It's very possible that these are street vendors
or are bodega,
bazaar, or marketplace restaurants that don't get mailed to.

This is because zip codes are used for understanding trends, running businesses, and finding new ways to reac
h you.
They are also used for make major decisions for marketing, opening or closing stores, providing services,
and making decisions that can have a massive financial impact. They correspond to mail delivery routes,
not a geographic area.
```

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.

In [28]:
```
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 NaN. Use the `postal_code_5` field.

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

Hint: The `notnull` method can be used to form a useful boolean array for this problem.

In [29]:
```
### BEGIN SOLUTION
# weird_zip_code_businesses = pd.DataFrame([pd.notnull(bus['postal_code_5'], ~isin all_sf_zip_codes)])
# pd.notnull(bus['postal_code_5'].loc[[all_sf_zip_codes]], ~isin)
weird_zip_code_businesses = ~pd.notnull(bus.loc[bus['postal_code_5'].isin(all_sf_zip_codes)])
### END SOLUTION
```

In [30]:
```
weird_zip_code_businesses
```

Out[30]:

	business_id	name	address	city	state	postal_code	latitude	longitude	phone_number	postal_code_5
0	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	True	False
3	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False
5	False	False	False	False	False	False	False	False	False	False
6	False	False	False	False	False	False	False	False	False	False
7	False	False	False	False	False	False	False	False	False	False
8	False	False	False	False	False	False	False	False	False	False
9	False	False	False	False	False	False	False	False	False	False
10	False	False	False	False	False	False	False	False	False	False
11	False	False	False	False	False	False	False	False	False	False
12	False	False	False	False	False	False	False	False	False	False
13	False	False	False	False	False	False	False	False	False	False
14	False	False	False	False	False	False	False	False	False	False
15	False	False	False	False	False	False	False	False	False	False
16	False	False	False	False	False	False	False	False	False	False
17	False	False	False	False	False	False	False	False	False	False
18	False	False	False	False	False	False	False	False	False	False
19	False	False	False	False	False	False	False	False	False	False
20	False	False	False	False	False	False	False	False	False	False
21	False	False	False	False	False	False	False	False	False	False
22	False	False	False	False	False	False	False	False	False	False
23	False	False	False	False	False	False	False	False	False	False
24	False	False	False	False	False	False	False	False	False	False
25	False	False	False	False	False	False	False	False	False	False

	business_id	name	address	city	state	postal_code	latitude	longitude	phone_number	postal_code_5
26	False	False	False	False	False	False	False	False	False	False
27	False	False	False	False	False	False	False	False	False	False
28	False	False	False	False	False	False	False	False	False	False
29	False	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...
6375	False	False	False	False	False	False	True	True	True	False
6376	False	False	False	False	False	False	True	True	False	False
6377	False	False	False	False	False	False	True	True	False	False
6378	False	False	False	False	False	False	True	True	False	False
6379	False	False	False	False	False	False	True	True	False	False
6380	False	False	False	False	False	False	True	True	False	False
6381	False	False	False	False	False	False	True	True	False	False
6382	False	False	False	False	False	False	True	True	False	False
6383	False	False	False	False	False	False	True	True	False	False
6384	False	False	False	False	False	False	True	True	True	False
6385	False	False	False	False	False	False	True	True	False	False
6386	False	False	False	False	False	False	True	True	False	False
6388	False	False	False	False	False	False	True	True	True	False
6389	False	False	False	False	False	False	True	True	False	False
6390	False	False	False	False	False	False	True	True	False	False
6391	False	False	False	False	False	False	True	True	False	False
6392	False	False	False	False	False	False	True	True	False	False
6393	False	False	False	False	False	False	True	True	False	False
6394	False	False	False	False	False	False	True	True	True	False
6395	False	False	False	False	False	False	True	True	False	False
6396	False	False	False	False	False	False	True	True	True	False
6397	False	False	False	False	False	False	True	True	False	False

	business_id	name	address	city	state	postal_code	latitude	longitude	phone_number	postal_code_5
6398	False	False	False	False	False	False	True	True	False	False
6399	False	False	False	False	False	False	True	True	False	False
6400	False	False	False	False	False	False	True	True	False	False
6401	False	False	False	False	False	False	True	True	False	False
6402	False	False	False	False	False	False	True	True	False	False
6403	False	False	False	False	False	False	True	True	False	False
6404	False	False	False	False	False	False	True	True	True	False
6405	False	False	False	False	False	False	True	True	False	False

6145 rows × 10 columns

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.

In [31]:
```
# Use this cell for your explorations.

### BEGIN SOLUTION HERE

bus.set_index('postal_code_5').loc[["94545", "94602"]]

q3d_answer = r"""

94545 - San Fracisco, ALAMEDA, CA

94602 - San Francisco, ALAMEDA, CA, the business name: Orbit Room, Real Address: 1900 Market St, San Francisc
o, CA 94102

The reason why the two zip codes appear in my dataframe is because the restaurants are not listed under the c
orrect zipcode.
It might be that the restaurants moved to another location, hence another zipcode. But it's also most likely
 that there are
multiple locations, such with J&J Vendor at the zipcode of 94545.

"""

### END SOLUTION HERE

print(q3d_answer)
```

```
94545 - San Fracisco, ALAMEDA, CA

94602 - San Francisco, ALAMEDA, CA, the business name: Orbit Room, Real Address: 1900 Market St, San Francisc
o, CA 94102

The reason why the two zip codes appear in my dataframe is because the restaurants are not listed under the c
orrect zipcode.
It might be that the restaurants moved to another location, hence another zipcode. But it's also most likely
that there are
multiple locations, such with J&J Vendor at the zipcode of 94545.
```

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.

Let's correct 94602 to the more likely value based on your analysis. Let's modify the derived field `zip_code` using `bus['zip_code'].str.replace` to replace 94602 with the correct value based on this business's real address that you learn by using a search engine.

In [32]:
```
### BEGIN SOLUTION
# Replace the 94602 with 94102
# 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'].str.replace('94602','94102')
### END SOLUTION
```

Out[32]:

0	94109
1	94104
2	94133
3	94110
4	94122
5	94115
6	94131
7	94111
8	94133
9	94122
10	94110
11	94115
12	94104
13	94104
14	94117
15	94117
16	94107
17	94110
18	94133
19	94108
20	94108
21	94102
22	94108
23	94111
24	94108
25	94133
26	94102
27	94104
28	94115
29	94103
...	...
6376	94117
6377	94103
6378	94122
6379	94115
6380	94111
6381	94111
6382	94109
6383	94107
6384	94108
6385	94109
6386	94103
6387	NaN

```
6388    94124
6389    94105
6390    94102
6391    94107
6392    94102
6393    94118
6394    94123
6395    94108
6396    94105
6397    94114
6398    94117
6399    94116
6400    94132
6401    94112
6402    94127
6403    94118
6404    94133
6405    94115
Name: postal_code, Length: 6406, dtype: object
```

In [33]:
```
assert "94602" not in bus['postal_code_5']
print(True)
```

```
True
```

4: Latitude and Longitude

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

Question 4a

How many businesses are missing longitude values?

Hint: Use isnull.

In [34]:
```
### BEGIN SOLUTION
missing_latlongs = pd.isnull(bus['longitude'])

missing_latlongs_size = bus[missing_latlongs].size/10
print(missing_latlongs_size)

print('There are 3136 businesses missing longitude values')

### END SOLUTION
```

```
3136.0
There are 3136 businesses missing longitude values
```

In [ ]:
```

```

Do not delete the empty cell below!

In [35]:
```
### BEGIN HIDDEN TESTS
assert missing_latlongs_size == sum(bus['longitude'].isnull())
# print(sum(bus['longitude'].isnull()))
print(True)
### END HIDDEN TESTS
```

```
True
```

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

Throughout problems 4a and 4b, let's focus on only the "dense" zip codes of the city of San Francisco, listed below as `sf_dense_zip`.

In [36]:
```
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.

For example, 94110 should be at the top of the series, with the value 294.

*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.

In [37]:
```
### BEGIN SOLUTION
bus_sf = bus[bus['postal_code_5'].isin(sf_dense_zip)]

num_missing_in_each_zip = pd.DataFrame([bus[bus["postal_code"]==zc]["longitude"].isnull().sum() for zc in sf_
dense_zip],
                                       columns=["num"], index=sf_dense_zip)
num_missing_in_each_zip.sort_values(inplace=True,by="num",ascending=False)


### END SOLUTION
```

Question 4b In question 4a, 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. `null count`: The number of missing values for the zip code. 2. `not null count`: 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 binary array. Or alternately, the `notnull` method can be used to create a binary array from a series.* *Note: An alternate approach is to create three aggregation functions as pass them in a list to the `agg` function.*

In [38]:
```
### BEGIN SOLUTION
not_null_count = bus_sf.groupby('postal_code_5').agg({'longitude': lambda entry:entry.notnull().sum()})
null_count = bus_sf.groupby('postal_code_5').agg({'longitude': lambda entry:entry.isnull().sum()})
not_null_ratio = null_count/not_null_count
fraction_missing_df = pd.DataFrame(index = bus['postal_code_5'])
fraction_missing_df['null_count'] = null_count
fraction_missing_df['not_null_count'] = not_null_count
fraction_missing_df['null_ratio'] = not_null_ratio

print(fraction_missing_df)


### END SOLUTION
```

postal_code_5	null_count	not_null_count	null_ratio
94109	171.0	209.0	0.818182
94104	79.0	60.0	1.316667
94133	159.0	267.0	0.595506
94110	294.0	303.0	0.970297
94122	132.0	141.0	0.936170
94115	95.0	148.0	0.641892
94131	16.0	33.0	0.484848
94111	129.0	148.0	0.871622
94133	159.0	267.0	0.595506
94122	132.0	141.0	0.936170
94110	294.0	303.0	0.970297
94115	95.0	148.0	0.641892
94104	79.0	60.0	1.316667
94104	79.0	60.0	1.316667
94117	86.0	118.0	0.728814
94117	86.0	118.0	0.728814
94107	275.0	185.0	1.486486
94110	294.0	303.0	0.970297
94133	159.0	267.0	0.595506
94108	98.0	130.0	0.753846
94108	98.0	130.0	0.753846
94102	221.0	241.0	0.917012
94108	98.0	130.0	0.753846
94111	129.0	148.0	0.871622
94108	98.0	130.0	0.753846
94133	159.0	267.0	0.595506
94102	221.0	241.0	0.917012
94104	79.0	60.0	1.316667
94115	95.0	148.0	0.641892
94103	285.0	268.0	1.063433
...	...	...	...
94117	86.0	118.0	0.728814
94103	285.0	268.0	1.063433
94122	132.0	141.0	0.936170
94115	95.0	148.0	0.641892
94111	129.0	148.0	0.871622
94111	129.0	148.0	0.871622
94109	171.0	209.0	0.818182
94107	275.0	185.0	1.486486
94108	98.0	130.0	0.753846
94109	171.0	209.0	0.818182

```
94103              285.0          268.0    1.063433
NaN                  NaN            NaN         NaN
94124              118.0           73.0    1.616438
94105              127.0          105.0    1.209524
94102              221.0          241.0    0.917012
94107              275.0          185.0    1.486486
94102              221.0          241.0    0.917012
94118              117.0          132.0    0.886364
94123               68.0          105.0    0.647619
94108               98.0          130.0    0.753846
94105              127.0          105.0    1.209524
94114              111.0          112.0    0.991071
94117               86.0          118.0    0.728814
94116               42.0           57.0    0.736842
94132               71.0           62.0    1.145161
94112               77.0          118.0    0.652542
94127               30.0           41.0    0.731707
94118              117.0          132.0    0.886364
94133              159.0          267.0    0.595506
94115               95.0          148.0    0.641892

[6406 rows x 3 columns]
```

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 many errors with the zip codes. As a result, we may want to drop the records with zip codes outside of San Francisco or to treat them differently. For some of the bad values, 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 business with well formed addresses.

5: 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.

In [39]:
```
ins.head(5)
```

Out[39]:

	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

Question 5a

From calling `head`, we know that each row in this table corresponds to the inspection of a single business. 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.

In [40]:
```
### BEGIN SOLUTION
# The number of rows in ins
rows_in_table = len(ins.index)
print(rows_in_table)
# The number of unique business IDs in ins.
unique_ins_ids =  len(ins['business_id'].unique())
print(unique_ins_ids)
### END SOLUTION
print(rows_in_table, unique_ins_ids, rows_in_table / unique_ins_ids)
```

```
14222
5766
14222 5766 2.4665279223031567
```

As you should have seen above, we have an average of roughly 3 inspections per business.

Question 5b

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 `type` take on? How many occurrences of each value is in the DataFrame? What are the implications for further analysis? For this problem, you need only fill in the string with a description; there's no specific dataframe or series that you need to create.

In [41]:
```
### BEGIN SOLUTION
q5b_answer = r"""
As said before: 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.


The values that type takes on are strings, however, the values represent what type of inspection it was, such
as "routine".
The number of occurrences of each value in the type of inspections in the DataFrame is all "routine".
The implications for further analysis are that the only type of inspections are routine, meaning there are no
other types
of inspections.

So, this means that the values of type are strings of different types or classifications of inspections, and
 that all
inspections classify as routine.

"""
### END SOLUTION

print(q5b_answer)
```

```
As said before: 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.


The values that type takes on are strings, however, the values represent what type of inspection it was, such
as "routine".
The number of occurrences of each value in the type of inspections in the DataFrame is all "routine".
The implications for further analysis are that the only type of inspections are routine, meaning there are no
other types
of inspections.

So, this means that the values of type are strings of different types or classifications of inspections, and
that all
inspections classify as routine.
```

Question 5c

In this question, we're going to try to figure out what years the data spans. Unfortunately, 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.

In [42]:
```
ins['new_date'] = pd.to_datetime(ins['date'])
ins.head(5)
```

Out[42]:

	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.

In [43]:
```
ins['new_date'] = pd.to_datetime(ins['date'], format='%Y%m%d')
ins.head(5)
```

Out[43]:

	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.

In [44]:
```
ins['year'] = ins['new_date'].dt.year
ins.head(5)
```

Out[44]:

	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

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.

In [45]:
```
### BEGIN SOLUTION
# print(ins[ins['year'] == '2015'])
# print(ins[ins['year'] == '2016'])
# print(ins[ins['year'] == '2017'])

q5c_answer = r"""

The range of years that is covered in this data is 2015-2018.
According to the data, there are not roughly the same number of inspections each year, since some years have
 more inspections.


"""
### END SOLUTION

print(q5c_answer)
```

```
The range of years that is covered in this data is 2015-2018.
According to the data, there are not roughly the same number of inspections each year, since some years have
more inspections.
```

6: Explore inspection score

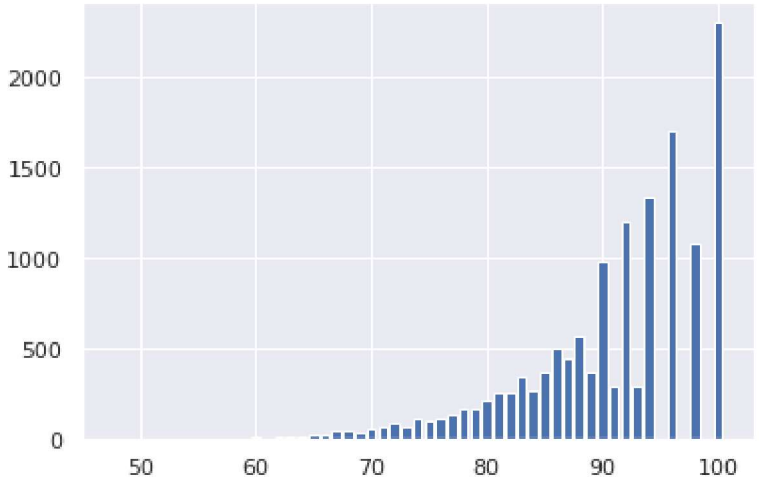
Question 6a

Let's look at the distribution of scores. As we saw before when we called `head` on this data, 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.

The code below counts how many inspections have each score, and then creates a bar plot of these scors.

Challenge: If you would like to experiment with plotting, please try to modify the code to change or improve the plot

In [46]:
```
scoreCts = ins['score'].value_counts()
plt.bar(scoreCts.keys(), scoreCts)
plt.show()
```



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?

In [47]:
```
### BEGIN SOLUTION
q6a_answer = r"""

Based on my bar plot, the qualities of the distribution of the inspections scores shows a uni-modal skew to t
he left, as
the trend shows a steady increase in amount of restaurants that have higher inspection scores. The mode of th
e bar plot
is 100%, as there are 2000+ restaurants with that score. There is no symmetry among this graph, as again, the
chart indicates
uni-modal shape and a skew to the left. The tail is found at the 65-67% area, and there are many gaps in betw
een values, such
with 95%-97%, where no restaurant scored 96%, apparently. This occurs a lot towards the right of the trend.
There are not really any anamolous data values, since the trend shows a lot of fluctuations of increasing and
decreasing, such
with 91%-92% and 92%-93%. These could be considered anamolous, but I will classify them as fluctuations/varia
tions.

The unusual features of this distribution is that towards the right, there are many gaps, and sudden increase
s-then-decreases-
then increases. These observations imply for the scores that there are a lot of restaurant scores that restau
rants either didn't
receive, or very few restaurants received.

There might be a large disparity of restaurants that scored even numbers. Weirdly enough, it might make sense
for scores
to be even. A 92% is more presentable than 91%, and 94% is more presentable than 93%, etc.


"""
### END SOLUTION

print(q6a_answer)
```

Based on my bar plot, the qualities of the distribution of the inspections scores shows a uni-modal skew to the left, as
the trend shows a steady increase in amount of restaurants that have higher inspection scores. The mode of the bar plot
is 100%, as there are 2000+ restaurants with that score. There is no symmetry among this graph, as again, the chart indicates
uni-modal shape and a skew to the left. The tail is found at the 65-67% area, and there are many gaps in between values, such
with 95%-97%, where no restaurant scored 96%, apparently. This occurs a lot towards the right of the trend.
There are not really any anamolous data values, since the trend shows a lot of fluctuations of increasing and decreasing, such
with 91%-92% and 92%-93%. These could be considered anamolous, but I will classify them as fluctuations/variations.

The unusual features of this distribution is that towards the right, there are many gaps, and sudden increases-then-decreases-
then increases. These observations imply for the scores that there are a lot of restaurant scores that restaurants either didn't
receive, or very few restaurants received.

There might be a large disparity of restaurants that scored even numbers. Weirdly enough, it might make sense for scores
to be even. A 92% is more presentable than 91%, and 94% is more presentable than 93%, etc.

Question 6b

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.

In [48]:
```
### BEGIN SOLUTION
ins_named = pd.merge(left=ins,right=bus, how="outer")
print(ins_named.sort_values(ascending=False, by='score'))
### END SOLUTION
```

	business_id	score	date	type	new_date	year	\
7111	61464	100.0	20150205.0	routine	2015-02-05	2015.0	
3906	5837	100.0	20171031.0	routine	2017-10-31	2017.0	
11180	79595	100.0	20171113.0	routine	2017-11-13	2017.0	
3881	5831	100.0	20150218.0	routine	2015-02-18	2015.0	
3880	5830	100.0	20170125.0	routine	2017-01-25	2017.0	
11177	79589	100.0	20160814.0	routine	2016-08-14	2016.0	
11176	79589	100.0	20150419.0	routine	2015-04-19	2015.0	
11175	79589	100.0	20170919.0	routine	2017-09-19	2017.0	
3879	5830	100.0	20161025.0	routine	2016-10-25	2016.0	
11169	79587	100.0	20160410.0	routine	2016-04-10	2016.0	
3876	5829	100.0	20170501.0	routine	2017-05-01	2017.0	
11168	79587	100.0	20170919.0	routine	2017-09-19	2017.0	
11167	79587	100.0	20150830.0	routine	2015-08-30	2015.0	
3875	5829	100.0	20161017.0	routine	2016-10-17	2016.0	
3873	5829	100.0	20171020.0	routine	2017-10-20	2017.0	
11163	79586	100.0	20170919.0	routine	2017-09-19	2017.0	
3872	5829	100.0	20150911.0	routine	2015-09-11	2015.0	
11161	79586	100.0	20160814.0	routine	2016-08-14	2016.0	
1829	2949	100.0	20170801.0	routine	2017-08-01	2017.0	
11181	79595	100.0	20150910.0	routine	2015-09-10	2015.0	
3882	5831	100.0	20160829.0	routine	2016-08-29	2016.0	
1808	2928	100.0	20161214.0	routine	2016-12-14	2016.0	
3891	5833	100.0	20160920.0	routine	2016-09-20	2016.0	
3898	5834	100.0	20150918.0	routine	2015-09-18	2015.0	
1790	2920	100.0	20150209.0	routine	2015-02-09	2015.0	
1791	2920	100.0	20160505.0	routine	2016-05-05	2016.0	
3894	5834	100.0	20170928.0	routine	2017-09-28	2017.0	
11212	79750	100.0	20170217.0	routine	2017-02-17	2017.0	
3893	5834	100.0	20150409.0	routine	2015-04-09	2015.0	
3892	5833	100.0	20170210.0	routine	2017-02-10	2017.0	
...	...	...	...	...	...	...	
14832	94305	NaN	NaN	NaN	NaT	NaN	
14833	94310	NaN	NaN	NaN	NaT	NaN	
14834	94318	NaN	NaN	NaN	NaT	NaN	
14835	94331	NaN	NaN	NaN	NaT	NaN	
14836	94334	NaN	NaN	NaN	NaT	NaN	
14837	94337	NaN	NaN	NaN	NaT	NaN	
14838	94354	NaN	NaN	NaN	NaT	NaN	
14839	94387	NaN	NaN	NaN	NaT	NaN	
14840	94388	NaN	NaN	NaN	NaT	NaN	
14841	94394	NaN	NaN	NaN	NaT	NaN	
14842	94408	NaN	NaN	NaN	NaT	NaN	

14843	94409	NaN	NaN	NaN	NaT	NaN
14844	94412	NaN	NaN	NaN	NaT	NaN
14845	94433	NaN	NaN	NaN	NaT	NaN
14846	94442	NaN	NaN	NaN	NaT	NaN
14847	94456	NaN	NaN	NaN	NaT	NaN
14848	94460	NaN	NaN	NaN	NaT	NaN
14849	94465	NaN	NaN	NaN	NaT	NaN
14850	94468	NaN	NaN	NaN	NaT	NaN
14851	94502	NaN	NaN	NaN	NaT	NaN
14852	94521	NaN	NaN	NaN	NaT	NaN
14853	94522	NaN	NaN	NaN	NaT	NaN
14854	94537	NaN	NaN	NaN	NaT	NaN
14855	94540	NaN	NaN	NaN	NaT	NaN
14856	94542	NaN	NaN	NaN	NaT	NaN
14857	94544	NaN	NaN	NaN	NaT	NaN
14858	94555	NaN	NaN	NaN	NaT	NaN
14859	94571	NaN	NaN	NaN	NaT	NaN
14860	94572	NaN	NaN	NaN	NaT	NaN
14861	94574	NaN	NaN	NaN	NaT	NaN

	name	address	\
7111	STONE KOREAN KITCHEN	4 EMBARCADERO CENTER	STREET LEVEL
3906	MCKINLEY ELEMENTARY SCHOOL		1025 14TH ST
11180	W HOTEL SAN FRANCISCO		181 03RD ST
3881	FRANK MCCOPPIN ELEMENTARY SCHOOL		651 06TH AVE
3880	THEODORE ROOSEVELT MIDDLE SCHOOL		460 ARGUELLO BLVD
11177	AT&T PARK - BEVERAGE CART	24 WILLIE MAYS PLAZA	
11176	AT&T PARK - BEVERAGE CART	24 WILLIE MAYS PLAZA	
11175	AT&T PARK - BEVERAGE CART	24 WILLIE MAYS PLAZA	
3879	THEODORE ROOSEVELT MIDDLE SCHOOL		460 ARGUELLO BLVD
11169	AT&T PARK - HEARTH TABLE STAND	24 WILLIE MAYS PLAZA	
3876	LAFAYETTE ELEMENTARY SCHOOL		4545 ANZA ST
11168	AT&T PARK - HEARTH TABLE STAND	24 WILLIE MAYS PLAZA	
11167	AT&T PARK - HEARTH TABLE STAND	24 WILLIE MAYS PLAZA	
3875	LAFAYETTE ELEMENTARY SCHOOL		4545 ANZA ST
3873	LAFAYETTE ELEMENTARY SCHOOL		4545 ANZA ST
11163	AT&T PARK - GARDEN TABLE STAND	24 WILLIE MAYS PLAZA	
3872	LAFAYETTE ELEMENTARY SCHOOL		4545 ANZA ST
11161	AT&T PARK - GARDEN TABLE STAND	24 WILLIE MAYS PLAZA	
1829	SAFEWAY STORE #4601		145 JACKSON ST
11181	W HOTEL SAN FRANCISCO		181 03RD ST
3882	FRANK MCCOPPIN ELEMENTARY SCHOOL		651 06TH AVE
1808	ELKS CLUB	450 POST ST, 3RD FLOOR	

3891	RAP/HILLTOP SCHOOL	1325 FLORIDA ST
3898	PRESIDIO MIDDLE SCHOOL	450 30TH AVE
1790	ACADEMY OF ARTS UNIV- I HOUSE CAFE	860 SUTTER ST
1791	ACADEMY OF ARTS UNIV- I HOUSE CAFE	860 SUTTER ST
3894	PRESIDIO MIDDLE SCHOOL	450 30TH AVE
11212	SIMPLY DELISH LLC	5668 03RD ST
3893	PRESIDIO MIDDLE SCHOOL	450 30TH AVE
3892	RAP/HILLTOP SCHOOL	1325 FLORIDA ST
...	...	...
14832	ROSAMUNDE SAUSAGE GRILL	545 HAIGHT ST
14833	YOKAI EXPRESS	135 4TH ST
14834	YUANBAO JIAOZI	2110 IRVING ST
14835	MATCHA CAFE MAIKO	1581 WEBSTER ST 175
14836	SUBWAY SANDWICHES #53761	160 BROADWAY ST
14837	SUBWAY SANDWICHES #61240	425 D BATTERY ST
14838	RAINBOW MARKET AND DELI	684 LARKIN ST
14839	FOUNDATION CAFE	645 5TH ST
14840	FOUNDATION CAFE	335 KEARNY ST
14841	KOKIO REPUBLIC	428 11TH ST
14842	SIZZLING POT KING	139 8TH ST
14843	AUGUST HALL	420 MASON ST
14844	NATIVE BAKING COMPANY	1324 FITZGERALD AVE
14845	GREEK TOWN LLC	88 02ND ST
14846	SIMPLY CAFE	340 GROVE ST
14847	UBER-ATG (BON APPETIT)	581 20TH ST 2ND FL
14848	DOBBS FERRY	409 GOUGH ST
14849	BEAUTIFULL LLC	3401 CALIFORNIA ST
14850	BAR CRENN	3131 FILLMORE ST
14851	NEW FORTUNE DIM SUM	811 STOCKTON ST
14852	JOE & THE JUICE HOWARD	301 HOWARD ST
14853	CAFE JOSEPHINE	199 MUSEUM WAY
14854	BON APPETIT @ USF- OUTTA HERE	2130 FULTON ST
14855	FOAM USA LLC	1745 TARAVAL ST
14856	OCEAN THAI	2545 OCEAN AVE
14857	D'MAIZE CAFE	50 PHELAN AVE
14858	EASY BREEZY FROZEN YOGURT	44 WEST PORTAL AVE
14859	THE PHOENIX PASTIFICIO	200 CLEMENT ST
14860	BROADWAY DIM SUM CAFE	684 BROADWAY ST
14861	BINKA BITES	2241 GEARY BLVD

	city	state	postal_code	latitude	longitude	phone_number	\
7111	San Francisco	CA	94111	37.795108	-122.396184	+14153706294	
3906	San Francisco	CA	94114	37.767231	-122.436377	+14152416300	

11180	San Francisco	CA	94103	NaN	NaN	+14157775300
3881	San Francisco	CA	94118	37.778382	-122.462787	+14157508475
3880	San Francisco	CA	94118	37.782383	-122.458926	+14157508446
11177	San Francisco	CA	94107	NaN	NaN	+14159721500
11176	San Francisco	CA	94107	NaN	NaN	+14159721500
11175	San Francisco	CA	94107	NaN	NaN	+14159721500
3879	San Francisco	CA	94118	37.782383	-122.458926	+14157508446
11169	San Francisco	CA	94107	NaN	NaN	+14159721500
3876	San Francisco	CA	94121	37.777603	-122.496490	+14157508483
11168	San Francisco	CA	94107	NaN	NaN	+14159721500
11167	San Francisco	CA	94107	NaN	NaN	+14159721500
3875	San Francisco	CA	94121	37.777603	-122.496490	+14157508483
3873	San Francisco	CA	94121	37.777603	-122.496490	+14157508483
11163	San Francisco	CA	94107	NaN	NaN	+14159721500
3872	San Francisco	CA	94121	37.777603	-122.496490	+14157508483
11161	San Francisco	CA	94107	NaN	NaN	+14159721500
1829	San Francisco	CA	94111	37.796984	-122.398887	+14159823112
11181	San Francisco	CA	94103	NaN	NaN	+14157775300
3882	San Francisco	CA	94118	37.778382	-122.462787	+14157508475
1808	San Francisco	CA	94102	37.788188	-122.409284	+14154211404
3891	San Francisco	CA	94110	NaN	NaN	+14156955606
3898	San Francisco	CA	94121	37.780823	-122.490105	+14157508435
1790	San Francisco	CA	94109	37.788498	-122.414514	+14159314719
1791	San Francisco	CA	94109	37.788498	-122.414514	+14159314719
3894	San Francisco	CA	94121	37.780823	-122.490105	+14157508435
11212	San Francisco	CA	94124	NaN	NaN	+14152150279
3893	San Francisco	CA	94121	37.780823	-122.490105	+14157508435
3892	San Francisco	CA	94110	NaN	NaN	+14156955606
...	...	...	...	...	...	...
14832	San Francisco	CA	94117	NaN	NaN	+14154376851
14833	San Francisco	CA	94103	NaN	NaN	+14158234502
14834	San Francisco	CA	94122	NaN	NaN	+14156013979
14835	San Francisco	CA	94115	NaN	NaN	+14150009434
14836	San Francisco	CA	94111	NaN	NaN	+14158861913
14837	San Francisco	CA	94111	NaN	NaN	+14153991549
14838	San Francisco	CA	94109	NaN	NaN	+14157664681
14839	San Francisco	CA	94107	NaN	NaN	+14153503301
14840	San Francisco	CA	94108	NaN	NaN	NaN
14841	San Francisco	CA	94109	NaN	NaN	+14157996404
14842	San Francisco	CA	94103	NaN	NaN	+14158028899
14843	San Francisco	CA	NaN	NaN	NaN	NaN
14844	San Francisco	CA	94124	NaN	NaN	NaN
14845	San Francisco	CA	94105	NaN	NaN	+14152408032

```
14846  San Francisco    CA       94102        NaN         NaN  +14156587659
14847  San Francisco    CA       94107        NaN         NaN  +14158184997
14848  San Francisco    CA       94102        NaN         NaN  +14155517709
14849  San Francisco    CA       94118        NaN         NaN  +14157289080
14850  San Francisco    CA       94123        NaN         NaN           NaN
14851  San Francisco    CA       94108        NaN         NaN  +14153991511
14852  San Francisco    CA       94105        NaN         NaN           NaN
14853  San Francisco    CA       94114        NaN         NaN  +14153508976
14854  San Francisco    CA       94117        NaN         NaN  +14153604802
14855  San Francisco    CA       94116        NaN         NaN  +14156060018
14856  San Francisco    CA       94132        NaN         NaN  +14155857251
14857  San Francisco    CA       94112        NaN         NaN  +14154240604
14858  San Francisco    CA       94127        NaN         NaN  +14155053351
14859  San Francisco    CA       94118        NaN         NaN  +14154726100
14860  San Francisco    CA       94133        NaN         NaN           NaN
14861  San Francisco    CA       94115        NaN         NaN  +14157712907
```

```
      postal_code_5
7111          94111
3906          94114
11180         94103
3881          94118
3880          94118
11177         94107
11176         94107
11175         94107
3879          94118
11169         94107
3876          94121
11168         94107
11167         94107
3875          94121
3873          94121
11163         94107
3872          94121
11161         94107
1829          94111
11181         94103
3882          94118
1808          94102
3891          94110
3898          94121
1790          94109
```

```
1791          94109
3894          94121
11212         94124
3893          94121
3892          94110
...             ...
14832         94117
14833         94103
14834         94122
14835         94115
14836         94111
14837         94111
14838         94109
14839         94107
14840         94108
14841         94109
14842         94103
14843           NaN
14844         94124
14845         94105
14846         94102
14847         94107
14848         94102
14849         94118
14850         94123
14851         94108
14852         94105
14853         94114
14854         94117
14855         94116
14856         94132
14857         94112
14858         94127
14859         94118
14860         94133
14861         94115

[14862 rows x 15 columns]
```

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

In [49]:
```
### BEGIN SOLUTION
q6b_answer = r"""

Using this data frame, the restaurant that is identified to have the lowest inspection scores ever is DA CAFE
at 407 CLEMENT ST
with a score of 48 in 2016

Using yelp.com and looking up the reviews page for this restaurant I found out that their score on Yelp is a
 3/5, so they
must have improved, and their food has mixed reviews:

One criticism argues that: "Other night order take out..chicken wings so tiny n dry. Soy vegi chow mein taste
less.
Singapore noodles tasteless too.. vegi egg roll good.. red bean drink tasteless too.. so disappointed."


"""
### END SOLUTION

print(q6b_answer)
```

```
Using this data frame, the restaurant that is identified to have the lowest inspection scores ever is DA CAFE
at 407 CLEMENT ST
with a score of 48 in 2016

Using yelp.com and looking up the reviews page for this restaurant I found out that their score on Yelp is a
3/5, so they
must have improved, and their food has mixed reviews:

One criticism argues that: "Other night order take out..chicken wings so tiny n dry. Soy vegi chow mein taste
less.
Singapore noodles tasteless too.. vegi egg roll good.. red bean drink tasteless too.. so disappointed."
```

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!

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

Question 7a

Let's see which restaurant has had the most extreme change in their ratings. Let the "swing" of a restaurant be defined as the difference between its lowest and highest ever rating. If a restaurant has been reviewed fewer than two times, its swing is zero. Using whatever technique you want to use, identify the three restaurants that are tied for the maximum swing value.

In [50]:
```
### BEGIN SOLUTION
q7a_answer = r"""

The three restaurants that are tied for the maximum swing value of 39 are:
1. Joanies Diner inc.
2. New Garden
3. The Crew

"""
### END SOLUTION

print(q7a_answer)
```

```
The three restaurants that are tied for the maximum swing value of 39 are:
1. Joanies Diner inc.
2. New Garden
3. The Crew
```

Question 7b

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`.

In [51]:
```
### BEGIN SOLUTION
inspections_by_id_and_year = ins.groupby(['business_id','year']).size().to_frame('count').rename(columns=
                                                                                                  {'':'count'
}, inplace=False)

# loc[:,['business_id','year']]
# .groupby('business_id').rename(columns={'score':'count'})
# ins2016.loc[:,['business_id','score']].sort_values('score').groupby('business_id').filter(lambda group:len
(group)==2).groupby('business_id').agg(group_to_list).rename(columns={'score':'score_pair'})
# ins.loc[:,['business_id','score']].sort_values('score').groupby('business_id').agg(group_to_list).rename(co
lumns={'score':'score_pair'})

### END SOLUTION
```

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.

In [52]:
```
inspections_by_id_and_year['count'].value_counts()
```

Out[52]:
```
1    9531
2    2175
3     111
4       2
Name: count, dtype: int64
```

Question 7c

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:

	score_pair
business_id	
24	[96, 98]
45	[78, 84]
66	[98, 100]
67	[87, 94]
76	[100, 98]

The scatter plot shoud look like this:

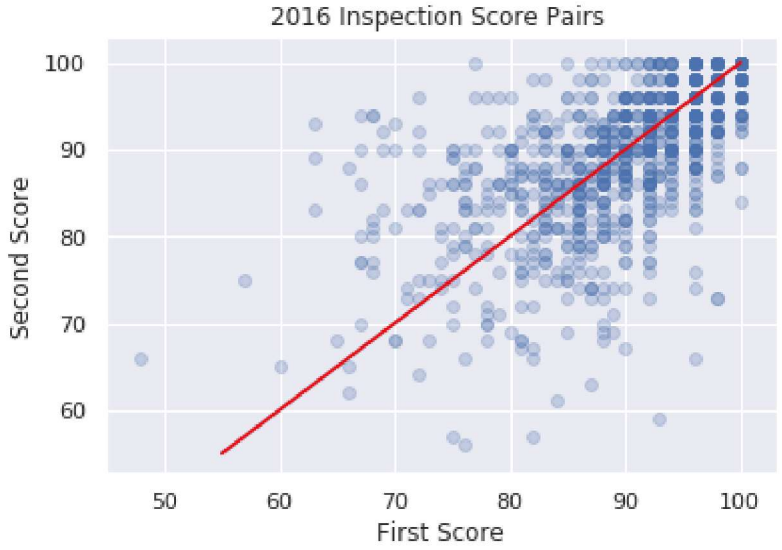
Note: Each score pair must be a list type

Hint: Use the `filter` method from lecture 6 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.

In [114]:
```
# For some odd reason, we can't just pass `list` into `.agg` so we define this function:
# You may or may not use it
ins2016 = ins[ins['year'] == 2016]

scores_pairs_by_business = ins2016.loc[:,['business_id','score']].groupby('business_id').filter(lambda group:
len(group)==2).groupby('business_id').agg(group_to_list).rename(columns={'score':'score_pair'})

def group_to_list(group):
    return list(group)

group_to_list(scores_pairs_by_business)


plt.plot(np.linspace(55,100,100),np.linspace(55,100,100))
plt.scatter(x=[i[0] for i in list(scores_pairs_by_business["score_pair"])],
            y=[i[1] for i in list(scores_pairs_by_business["score_pair"])], alpha=0.25)
plt.plot(np.linspace(55,100,100),np.linspace(55,100,100), color='Red')
plt.xlabel('First Score')
plt.ylabel('Second Score')
plt.title("2016 Inspection Score Pairs ")

# mydata = scores_pairs_by_business.loc[:,['score_pair']]
# vals = mydata['score_pair'].values
# for i in range(len(vals)):
#     x = vals[i][0]
#     y = vals[i][1]
#     plt.scatter(x, y, color='b', alpha=.25, norm=.25, marker=None)
# sns.regplot(x, y, data=scores_pairs_by_business, scatter=True, fit_reg=True, color='Red', marker='-')
# plt.show()


### BEGIN SOLUTION

### END SOLUTION
```

Out[114]: Text(0.5, 1.0, '2016 Inspection Score Pairs ')



In [106]:
```
assert isinstance(scores_pairs_by_business, pd.DataFrame)
assert scores_pairs_by_business.columns == ['score_pair']
print(True)
```

```
True
```

Question 7d

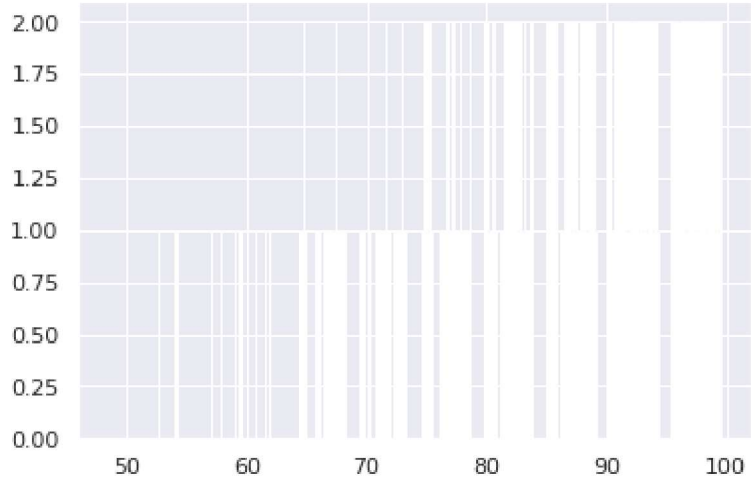
Another way to compare the scores from the two inspections is to examine the difference in scores. Subtracting the first score from the second in `scores_pairs_by_business` and making a histogram of these differences in the scores. We might expect these differences to be positive, indicating an improvement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

The histogram should look like this:

If a restaurant's score improves from the first to the second inspection, what do you expect to see in the scatter plot in question 7c? 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? What do you see?

In [85]:
```
score_pairs= scores_pairs_by_business['score_pair']
# score_pairs.plot(kind='hist')
plt.hist(score_pairs)
plt.show()

q7c_answer = r"""

If a restaurant's score improves from the first to the second inspection, what I expect to see in the scatter plot in a large
amount of positive results or results of 0. I see that a large portion or volume of the data is clustered around the -10 to 10
range area, which means that a lot of restaurants are improving, according to the inspections.

If a restaurant's score improves from the first to the second inspection, this would be reflected in the histogram with
the difference in the scores through a large concentration on the right side of the  graph. Since, there would be a net
positive difference.

"""

print(q7c_answer)
```



If a restaurant's score improves from the first to the second inspection, what I expect to see in the scatter plot in a large
amount of positive results or results of 0. I see that a large portion or volume of the data is clustered around the -10 to 10
range area, which means that a lot of restaurants are improving, according to the inspections.

If a restaurant's score improves from the first to the second inspection, this would be reflected in the histogram with
the difference in the scores through a large concentration on the right side of the graph. Since, there would be a net
positive difference.

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 restaurants that have 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.