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
1a	1
1b	0
1c	0
1d	3
1e	1
2a	1
2b	2

Question	Points
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
Total	45

In collaboration with Bennett Berlin and Kit Kitsombu

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 [1]:
```
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 [2]:
```
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 [3]:
```
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 th
e data
dest_path = ds112_utils.fetch_and_cache(data_url=source_data_url, data_dir=dat
a_dir, file=target_file_name, force=False)
```

```
Using cached version that was downloaded (UTC): Tue Sep 24 20:20:36 2019
```

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)](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 [4]:
```
# Fill in the list_names variable with a list of all the names of the files in
the zip file

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

The cell below will test that your code is correct.

In [5]:
```
assert isinstance(my_zip, zipfile.ZipFile)
assert isinstance(list_names, list)
assert all([isinstance(file, str) for file in list_names])
```

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 [6]:
```
### BEGIN SOLUTION
print([f.filename for f in my_zip.filelist])
### END SOLUTION
```

```
['violations.csv', 'businesses.csv', 'inspections.csv', 'legend.csv']
```

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 [7]:
```
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 [8]:
```
### BEGIN SOLUTION
ds112_utils.head('data/businesses.csv', 5)
ds112_utils.head('data/inspections.csv', 5)
ds112_utils.head('data/legend.csv', 5)
ds112_utils.head('data/violations.csv', 5)
### END SOLUTION
```

Out[8]:
```
['"business_id","date","description"\n',
 '19,"20171211","Inadequate food safety knowledge or lack of certified food s
afety manager"\n',
 '19,"20171211","Unapproved or unmaintained equipment or utensils"\n',
 '19,"20160513","Unapproved or unmaintained equipment or utensils  [ date vio
lation corrected: 12/11/2017 ]"\n',
 '19,"20160513","Unclean or degraded floors walls or ceilings  [ date violati
on 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 [9]:
```
# path to directory containing data
dsDir = Path('data')

### BEGIN SOLUTION
# Make sure to use these names
bus = pd.read_csv('data/businesses.csv', encoding='ISO-8859-1')
ins = pd.read_csv('data/inspections.csv')
vio = pd.read_csv('data/violations.csv')
### 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 [10]:
```
### BEGIN SOLUTION
bus.head()
ins.head()
vio.head()
### END SOLUTION
```

Out[10]:

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

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

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

Out[11]:
```
<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 [12]:
```
assert all(bus.columns == ['business_id', 'name', 'address', 'city', 'state', 'postal_code',
                           'latitude', 'longitude', 'phone_number'])
assert 6400 <= len(bus) <= 6420

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

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

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

In [13]:
```
bus_summary = pd.DataFrame(**{'columns': ['business_id', 'latitude', 'longitud
e'],
 '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 [14]:
```
"""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 column
s, 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 tha
t 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 [15]: # These tests will raise an exception if your variables don't match numericall
         y 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)
```

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 [16]: ### BEGIN SOLUTION
         q1e_answer = """

         The phone number for Norman's Ice Cream and Freezes is missing.
         Missing values reduce the power of a statistical test and can cause bias in th
         e estimated parameters.

         """
         ### END SOLUTION

         print(q1e_answer)
```

```
The phone number for Norman's Ice Cream and Freezes is missing.
Missing values reduce the power of a statistical test and can cause bias in t
he estimated parameters.
```

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 [17]:
```
### BEGIN SOLUTION
if len(bus['business_id'].unique()) == len(bus['business_id']):
    is_business_id_unique = True
### END SOLUTION
```

In [18]:
```
assert is_business_id_unique
```

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 [19]:
```
# use this cell for scratch work
# consider using groupby or value_counts() on the 'name' or 'business_id'

### BEGIN SOLUTION
names = bus.iloc[:,1]
names.value_counts()
bus['business_id'].value_counts()

### END SOLUTION
```

Out[19]:
```
2047     1
71088    1
5528     1
89497    1
16513    1
81309    1
75166    1
83362    1
3491     1
81317    1
5544     1
93615    1
89521    1
29304    1
62900    1
91574    1
93623    1
1466     1
69051    1
81341    1
7617     1
83394    1
81349    1
89545    1
82196    1
16884    1
1426     1
87440    1
71008    1
17762    1
        ..
90833    1
84692    1
78555    1
39513    1
76464    1
67448    1
3611     1
2724     1
85679    1
4703     1
62051    1
68196    1
93546    1
62067    1
2676     1
75903    1
86647    1
76408    1
68220    1
82557    1
88702    1
2692     1
645      1
86663    1
88718    1
81342    1
```

```
21143    1
64154    1
2716     1
83969    1
Name: business_id, Length: 6406, dtype: int64
```

In [20]:
```
q2b_answer = r"""

Each store has its own business ID. However, many of them belong to chains, su
ch as Starbucks or Peets Coffee, while some of them are small mom and pop shop
s.

"""
### END SOLUTION

print(q2b_answer)
```

```
Each store has its own business ID. However, many of them belong to chains, s
uch as Starbucks or Peets Coffee, while some of them are small mom and pop sh
ops.
```

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 [21]:
```
type(bus.iloc[0,5])
```

Out[21]:
```
str
```

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


### BEGIN SOLUTION
q3a_answer = r"""

Though zip codes are made of digits, they are qualitative data points since th
ey do not measure but describe a location.
For this reason, they are nominal data and stored as strings.

"""
### END SOLUTION

print(q3a_answer)
```

```
Though zip codes are made of digits, they are qualitative data points since t
hey do not measure but describe a location.
For this reason, they are nominal data and stored as strings.
```

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)
### END SOLUTION
```

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(asc
ending=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).h
ead(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	
0	19	NRGIZE LIFESTYLE CAFE	1200 VAN NESS AVE, 3RD FLOOR	San Francisco	CA	94109	37.786848	-1
1	24	OMNI S.F. HOTEL - 2ND FLOOR PANTRY	500 CALIFORNIA ST, 2ND FLOOR	San Francisco	CA	94104	37.792888	-1
2	31	NORMAN'S ICE CREAM AND FREEZES	2801 LEAVENWORTH ST	San Francisco	CA	94133	37.807155	-1
3	45	CHARLIE'S DELI CAFE	3202 FOLSOM ST	San Francisco	CA	94110	37.747114	-1
4	48	ART'S CAFE	747 IRVING ST	San Francisco	CA	94122	37.764013	-1
5	54	RHODA GOLDMAN PLAZA	2180 POST ST	San Francisco	CA	94115	37.784626	-1
6	56	CAFE X + O	1799 CHURCH ST	San Francisco	CA	94131	37.742325	-1
7	58	OASIS GRILL	91 DRUMM ST	San Francisco	CA	94111	37.794483	-1
8	61	CHOWDERS	PIER 39 SPACE A3	San Francisco	CA	94133	37.808240	-1
9	66	STARBUCKS COFFEE	1800 IRVING ST	San Francisco	CA	94122	37.763578	-1
10	67	REVOLUTION CAFE	3248 22ND ST	San Francisco	CA	94110	37.755419	-1
11	73	DINO'S UNCLE VITO	2101 FILLMORE ST	San Francisco	CA	94115	37.788932	-1
12	76	OMNI S.F. HOTEL - 3RD FLOOR PANTRY	500 CALIFORNIA ST, 3RD FLOOR	San Francisco	CA	94104	37.792888	-1
13	77	OMNI S.F. HOTEL - EMPLOYEE CAFETERIA	500 CALIFORNIA ST, BASEMENT	San Francisco	CA	94104	37.792888	-1
14	80	LAW SCHOOL CAFE	2199 FULTON ST	San Francisco	CA	94117	37.774941	-1
15	81	CLUB ED/BON APPETIT	2350 TURK ST	San Francisco	CA	94117	37.778468	-1
16	88	J.B.'S PLACE	1435 17TH ST	San Francisco	CA	94107	37.765003	-1
17	95	VEGA	419 CORTLAND AVE	San Francisco	CA	94110	37.739207	-1
18	98	XOX TRUFFLES	754 COLUMBUS AVE	San Francisco	CA	94133	37.801665	-1

	business_id	name	address	city	state	postal_code	latitude	
19	99	J & M A-1 CAFE RESTAURANT LLC	779 CLAY ST	San Francisco	CA	94108	37.794293	-1
20	101	CABLE CAR CORNER	1099 POWELL ST	San Francisco	CA	94108	37.794615	-1
21	102	AKIKO'S SUSHI BAR	542A MASON ST	San Francisco	CA	94102	37.788484	-1
22	108	RUE LEPIC	900 PINE ST	San Francisco	CA	94108	37.790868	-1
23	116	THE WATERFRONT RESTAURANT	PIER 7 EMBARCADERO	San Francisco	CA	94111	37.793874	-1
24	121	AKIKOS SUSHI	431 BUSH ST	San Francisco	CA	94108	37.790643	-1
25	125	CENTERFOLDS	391 BROADWAY ST	San Francisco	CA	94133	37.798233	-1
26	134	MINT	400 MCALLISTER ST	San Francisco	CA	94102	37.780247	-1
27	140	CAFE MADELEINE	300 CALIFORNIA ST	San Francisco	CA	94104	37.793268	-1
28	141	AFC SUSHI @ MOLLIE STONE'S 2	2435 CALIFORNIA ST	San Francisco	CA	94115	37.788773	-1
29	146	DEJA VU PIZZA & PASTA	3227 16TH ST	San Francisco	CA	94103	37.764713	-1
...	...	...	...	...	...	...	...	
6376	94305	ROSAMUNDE SAUSAGE GRILL	545 HAIGHT ST	San Francisco	CA	94117	NaN	
6377	94310	YOKAI EXPRESS	135 4TH ST	San Francisco	CA	94103	NaN	
6378	94318	YUANBAO JIAOZI	2110 IRVING ST	San Francisco	CA	94122	NaN	
6379	94331	MATCHA CAFE MAIKO	1581 WEBSTER ST 175	San Francisco	CA	94115	NaN	
6380	94334	SUBWAY SANDWICHES #53761	160 BROADWAY ST	San Francisco	CA	94111	NaN	
6381	94337	SUBWAY SANDWICHES #61240	425 D BATTERY ST	San Francisco	CA	94111	NaN	
6382	94354	RAINBOW MARKET AND DELI	684 LARKIN ST	San Francisco	CA	94109	NaN	
6383	94387	FOUNDATION CAFE	645 5TH ST	San Francisco	CA	94107	NaN	
6384	94388	FOUNDATION CAFE	335 KEARNY ST	San Francisco	CA	94108	NaN	

	business_id	name	address	city	state	postal_code	latitude
6385	94394	KOKIO REPUBLIC	428 11TH ST	San Francisco	CA	94109	NaN
6386	94408	SIZZLING POT KING	139 8TH ST	San Francisco	CA	94103	NaN
6387	94409	AUGUST HALL	420 MASON ST	San Francisco	CA	NaN	NaN
6388	94412	NATIVE BAKING COMPANY	1324 FITZGERALD AVE	San Francisco	CA	94124	NaN
6389	94433	GREEK TOWN LLC	88 02ND ST	San Francisco	CA	94105	NaN
6390	94442	SIMPLY CAFE	340 GROVE ST	San Francisco	CA	94102	NaN
6391	94456	UBER-ATG (BON APPETIT)	581 20TH ST 2ND FL	San Francisco	CA	94107	NaN
6392	94460	DOBBS FERRY	409 GOUGH ST	San Francisco	CA	94102	NaN
6393	94465	BEAUTIFULL LLC	3401 CALIFORNIA ST	San Francisco	CA	94118	NaN
6394	94468	BAR CRENN	3131 FILLMORE ST	San Francisco	CA	94123	NaN
6395	94502	NEW FORTUNE DIM SUM	811 STOCKTON ST	San Francisco	CA	94108	NaN
6396	94521	JOE & THE JUICE HOWARD	301 HOWARD ST	San Francisco	CA	94105	NaN
6397	94522	CAFE JOSEPHINE	199 MUSEUM WAY	San Francisco	CA	94114	NaN
6398	94537	BON APPETIT @ USF- OUTTA HERE	2130 FULTON ST	San Francisco	CA	94117	NaN
6399	94540	FOAM USA LLC	1745 TARAVAL ST	San Francisco	CA	94116	NaN
6400	94542	OCEAN THAI	2545 OCEAN AVE	San Francisco	CA	94132	NaN
6401	94544	D'MAIZE CAFE	50 PHELAN AVE	San Francisco	CA	94112	NaN
6402	94555	EASY BREEZY FROZEN YOGURT	44 WEST PORTAL AVE	San Francisco	CA	94127	NaN
6403	94571	THE PHOENIX PASTIFICIO	200 CLEMENT ST	San Francisco	CA	94118	NaN
6404	94572	BROADWAY DIM SUM CAFE	684 BROADWAY ST	San Francisco	CA	94133	NaN
6405	94574	BINKA BITES	2241 GEARY BLVD	San Francisco	CA	94115	NaN

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]:
```
null_zip = bus[bus['postal_code'].isnull() == True]
null_zip.head()
```

Out[27]:

	business_id	name	address	city	state	postal_code	latitude	longitud
1702	8202	XIAO LOONG	250 WEST PORTAL AVENUE	San Francisco	CA	NaN	37.738616	-122.46877
1725	9358	EDGEWOOD CHILDREN'S CENTER	1801 VICENTE ST	San Francisco	CA	NaN	37.739083	-122.48543
1731	9582	DIMPLES	1700 POST ST.	San Francisco	CA	NaN	37.785632	-122.42979
1747	10011	OSHA THAI NOODLE	819 VALENCIA ST.	San Francisco	CA	NaN	37.759943	-122.42133
1754	10227	THE NAPPER TANDY	3200 24TH ST	San Francisco	CA	NaN	37.752581	-122.41648

In [28]:
```
# Use this cell for your explorations.

### BEGIN SOLUTION
q3c_answer = r"""

Many of these restaurants are missing longitude and latitude coordinates. some
addresses are off the grid or private locations

"""
### END SOLUTION

print(q3c_answer)
```

```
Many of these restaurants are missing longitude and latitude coordinates. som
e addresses are off the grid or private locations
```

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 [29]:
```
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 [30]:
```
### BEGIN SOLUTION
weird_zip_code_businesses = bus[bus['postal_code_5'].notnull() & ~bus['postal_code_5'].isin(all_sf_zip_codes)]
### END SOLUTION
```

In [31]:
```
weird_zip_code_businesses
```

Out[31]:

	business_id	name	address	city	state	postal_code	latit
1211	5208	GOLDEN GATE YACHT CLUB	1 YACHT RD	San Francisco	CA	941	37.807
1372	5755	J & J VENDING	VARIOUS LOACATIONS (17)	San Francisco	CA	94545	
1373	5757	RICO VENDING, INC	VARIOUS LOCATIONS	San Francisco	CA	94066	
2258	36547	EPIC ROASTHOUSE	PIER 26 EMBARARCADERO	San Francisco	CA	95105	37.788
2293	37167	INTERCONTINENTAL SAN FRANCISCO EMPLOYEE CAFETERIA	888 HOWARD ST 2ND FLOOR	San Francisco	CA	94013	37.781
2295	37169	INTERCONTINENTAL SAN FRANCISCO 4TH FL. KITCHEN	888 HOWARD ST 4TH FLOOR	San Francisco	CA	94013	37.781
2846	64540	LEO'S HOT DOGS	2301 MISSION ST	San Francisco	CA	CA	37.760
2852	64660	HAIGHT STREET MARKET	1530 HAIGHT ST	San Francisco	CA	92672	37.769
2857	64738	JAPACURRY	PUBLIC	San Francisco	CA	CA	37.777
2969	65856	BAMBOO ASIA	41 MONTGOMERY ST	San Francisco	CA	94101	37.774
3142	67875	THE CHAIRMAN TRUCK	OFF THE GRID	San Francisco	CA	00000	37.777
3665	72127	REVOLUTION FOODS	5383 CAPWELL	San Francisco	CA	94621	
3758	74674	ELI'S HOT DOGS	101 BAYSHORE BLVD	San Francisco	CA	94014	
4853	83744	LA FROMAGERIE	101 MONTGOMERY ST	San Francisco	CA	94101	
5060	85459	ORBIT ROOM	1900 MARKET ST	San Francisco	CA	94602	
5325	87059	COFFEE BAR-MONTGOMERY	101 MONTGOMERY ST SUITE 101C	San Francisco	CA	94014	
5480	88139	TACOLICIOUS	2250 CHESTNUT ST	San Francisco	CA	Ca	
5894	90733	JEEPSILOG	2 MARINA BLVD	San Francisco	CA	94080	
6002	91249	AN THE GO	OFF THE GRID	San Francisco	CA	00000	
6130	92141	ALFARO TRUCK	332 VALENCIA ST	San Francisco	CA	64110	
6300	93484	CARDONA'S FOOD TRUCK	2430 WHIPPLE RD	San Francisco	CA	94544	

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

In [32]:
```
# Use this cell for your explorations.

### BEGIN SOLUTION HERE

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

q3d_answer = r"""

94545 - Hayward, CA: It is a vending chain so they probably mixed up the locat
ions

94602 - Oakland, CA
        Really Orbit Room 1900 Market St, San Francisco, CA 94102: Mistake is
 probably due to that both zip codes are the same except for one digit
        so the human made a typo.

        The mistakes are probably due to human typos.

"""

### END SOLUTION HERE

print(q3d_answer)
```

```
94545 - Hayward, CA: It is a vending chain so they probably mixed up the loca
tions

94602 - Oakland, CA
        Really Orbit Room 1900 Market St, San Francisco, CA 94102: Mistake is
probably due to that both zip codes are the same except for one digit
        so the human made a typo.

        The mistakes are probably due to human typos.
```

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 [33]:
```
### BEGIN SOLUTION
bus['postal_code_5'].str.replace('94602','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_5'] = bus['postal_code_5'].str.replace('94602','94102')
### END SOLUTION
```

In [34]:
```
assert "94602" not in bus['postal_code_5']
```

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 [35]:
```
### BEGIN SOLUTION
missing_latlongs = bus['longitude'].isnull().sum()
### END SOLUTION
```

Do not delete the empty cell below!

In [36]:
```
### BEGIN HIDDEN TESTS
assert missing_latlongs == sum(bus['longitude'].isnull())
### END HIDDEN TESTS
```

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 [37]:
```
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 [38]:
```
### BEGIN SOLUTION
sf_dense = pd.DataFrame(data = sf_dense_zip, columns = ['zip_codes'])
bus_sf = bus.merge(sf_dense, left_on = 'postal_code_5', right_on = 'zip_codes'
)
missing_long = bus_sf[bus_sf['longitude'].isnull()]
num_missing_in_each_zip = missing_long['postal_code_5'].value_counts()
num_missing_in_each_zip.head()
### END SOLUTION
```

Out[38]:
```
94110    294
94103    285
94107    275
94102    222
94109    171
Name: postal_code_5, dtype: int64
```

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 [39]:
```
### BEGIN SOLUTION
not_missing_long = bus_sf[bus_sf['longitude'].notnull()]
num_in_each_zip = not_missing_long['postal_code_5'].value_counts()
num_in_each_zip.head()

fraction_missing_long = missing_long['postal_code_5'].value_counts() / bus_sf[
'postal_code_5'].value_counts()
fraction_missing_long.head()

num_missing_in_each_zip.rename('null count', inplace = True)
num_in_each_zip.rename('not null count', inplace = True)
fraction_missing_long.rename('fraction null', inplace = True)
fraction_missing_df = pd.concat([num_missing_in_each_zip,num_in_each_zip,fract
ion_missing_long], axis = 1, sort = True)
fraction_missing_df.index.names = ['postal_code_5']
fraction_missing_df.head()

### END SOLUTION
```

Out[39]:

postal_code_5	null count	not null count	fraction null
94102	222	241	0.479482
94103	285	268	0.515371
94104	79	60	0.568345
94105	127	105	0.547414
94107	275	185	0.597826

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 [40]:
```
ins.head(5)
```

Out[40]:

	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 [41]:
```
### BEGIN SOLUTION
# The number of rows in ins
rows_in_table = ins.shape[0]

# The number of unique business IDs in ins.
unique_ins_ids = ins['business_id'].nunique()
### END SOLUTION
print(rows_in_table, unique_ins_ids, rows_in_table / unique_ins_ids)
```

```
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 [42]:
```
### BEGIN SOLUTION
print(ins['type'].value_counts())
q5b_answer = r"""

type is a series of strings, with all of them being routine except for one whi
ch is complaint
Clearly that one restauant was horrible

"""
### END SOLUTION

print(q5b_answer)
```

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


type is a series of strings, with all of them being routine except for one wh
ich is complaint
Clearly that one restauant was horrible
```

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 [43]:
```
ins['new_date'] = pd.to_datetime(ins['date'])
ins.head(5)
```

Out[43]:

	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 [44]:
```
ins['new_date'] = pd.to_datetime(ins['date'], format='%Y%m%d')
ins.head(5)
```

Out[44]:

	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 [45]:
```
ins['year'] = ins['new_date'].dt.year
ins.head(5)
```

Out[45]:

	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 [46]:
```
### BEGIN SOLUTION
print(ins['year'].value_counts())
q5c_answer = r"""

Inspections span 2015 - 2018
The number of inspectons increase from 2015 to 2016 by 2/3, 2016 is roughly th
e same as 2017.
2018 has only 308 inspections but that may be because this file was created in
the middle of 2018

"""
### END SOLUTION

print(q5c_answer)
```

```
2016    5443
2017    5166
2015    3305
2018     308
Name: year, dtype: int64


Inspections span 2015 - 2018
The number of inspectons increase from 2015 to 2016 by 2/3, 2016 is roughly t
he same as 2017.
2018 has only 308 inspections but that may be because this file was created i
n the middle of 2018
```

6: Explore inspection score

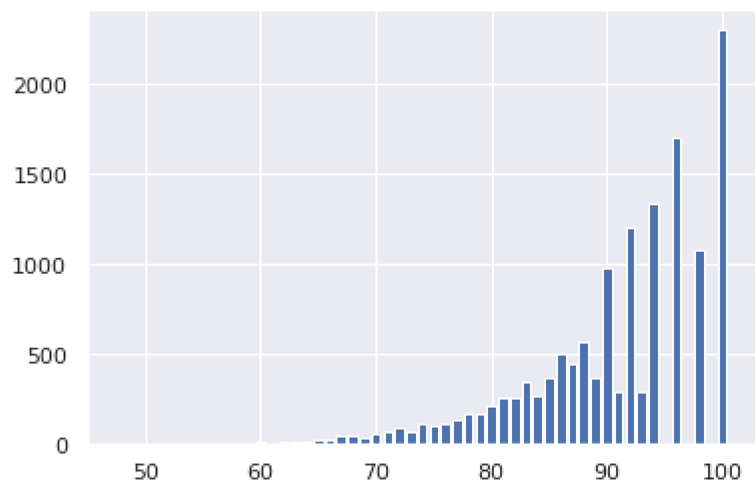
Question 6a

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 [47]:
```
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 [48]:
```
### BEGIN SOLUTION
q6a_answer = r"""

The distribution of the scores are are such that the majority of them are high
er scores, sort of like an extremely skewed distribution to the left.
The mode is a score of 100, with 96, 94, 92, and 90 closely following.
There is no symmetery, and looks nothing like the discrete version of a ball c
urve.
Thus the mean is well below the median.
There is a tail due the few low scores of restaurants which were so bad they c
ould not pass what seems to be a very easy inspection.
There are a few gaps in the 90s region but for the most part, the scores tend
 to be 80 or above.

"""
### END SOLUTION

print(q6a_answer)
```

```
The distribution of the scores are are such that the majority of them are hig
her scores, sort of like an extremely skewed distribution to the left.
The mode is a score of 100, with 96, 94, 92, and 90 closely following.
There is no symmetery, and looks nothing like the discrete version of a ball
curve.
Thus the mean is well below the median.
There is a tail due the few low scores of restaurants which were so bad they
could not pass what seems to be a very easy inspection.
There are a few gaps in the 90s region but for the most part, the scores tend
to be 80 or above.
```

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 [49]:
```
### BEGIN SOLUTION
ins_named = ins.merge(bus, left_on = 'business_id', right_on = 'business_id',
how = 'outer')
ins_named.sort_values(['score'], inplace = True)
ins_named.head()
### END SOLUTION
```

Out[49]:

	business_id	score	date	type	new_date	year	name	address	
13179	86647	48.0	20160907.0	routine	2016-09-07	2016.0	DA CAFE	407 CLEMENT ST	Frar
9476	71373	52.0	20161031.0	routine	2016-10-31	2016.0	GOLDEN RIVER RESTAURANT	5827 GEARY BLVD	Frar
8885	69199	53.0	20170127.0	routine	2017-01-27	2017.0	MEHFIL INDIAN RESTAURANT	28 02ND ST	Frar
7104	61436	54.0	20150706.0	routine	2015-07-06	2015.0	OZONE THAI RESTAURANT AND LOUNGE	598 02ND ST	Frar
2192	3459	54.0	20150407.0	routine	2015-04-07	2015.0	BASIL THAI RESTAURANT & BAR	1175 FOLSOM ST	Frar

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 [50]:
```
### BEGIN SOLUTION
q6b_answer = r"""

DA Cafe, their most recent review as of 9/26/2019:
Solid 4star hole in the style Chinese food...come for the 3 item dinner, don't
expect great service but expect good food and fast

"""
### END SOLUTION

print(q6b_answer)
```

```
DA Cafe, their most recent review as of 9/26/2019:
Solid 4star hole in the style Chinese food...come for the 3 item dinner, do
n't expect great service but expect good food and fast
```

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!

7: Restaurant Ratings Over Time

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 [51]:
```
### BEGIN SOLUTION
max_scores = ins_named.groupby('business_id')['score'].max()
min_scores = ins_named.groupby('business_id')['score'].min()
swing_score = max_scores - min_scores
swing_score.sort_values(ascending = False, inplace = True)
swing_scores = swing_score.to_frame()
swing_scores.rename(columns = {"score":"swing_score"}, inplace = True)
swing_scores = swing_scores.merge(bus, on = 'business_id')


q7a_answer = r"""

New Garden Restaurant, INC; Joanie's Diner INC; The Crew

"""
### END SOLUTION

print(q7a_answer)
```

```
New Garden Restaurant, INC; Joanie's Diner INC; 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 [52]:
```
### BEGIN SOLUTION
year_count = ins_named.groupby('business_id')['year'].value_counts()
year_count = year_count.to_frame()
year_count.rename(columns = {"year":"count"}, inplace = True)
inspections_by_id_and_year = year_count
inspections_by_id_and_year.head()
### END SOLUTION
```

Out[52]:

business_id	year	count
19	**2016.0**	1
	2017.0	1
24	**2016.0**	2
	2017.0	1
31	**2015.0**	1

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 [53]:
```
inspections_by_id_and_year['count'].value_counts()
```

Out[53]:
```
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 [76]:
```
ins2016 = ins[ins['year'] == 2016]
newinscount = renamed.groupby(['business_id'])['counted'].count().to_frame()
filteredins = newinscount.groupby('business_id')['counted'].filter(lambda x: s
um(x) == 2).to_frame()
themmerged = filteredins.merge(ins2016, left_on = 'business_id', right_on = 'b
usiness_id')
gb = (themmerged.merge(themmerged.groupby('business_id')['score'].apply(np.arr
ay),how='left',left_on='business_id',right_index=True))
newnew = gb.drop_duplicates(['business_id'], keep='last')
need1more = newnew.rename(columns={"score_y": "score_pair"})
scores_pairs_by_business = need1more[['business_id','score_pair']]
scores_pairs_by_business.set_index('business_id', inplace=True)
scores_pairs_by_business
```

Out[76]:

	score_pair
business_id	
24	[98, 96]
45	[78, 84]
66	[98, 100]
67	[87, 94]
76	[100, 98]
77	[96, 91]
146	[84, 84]
217	[90, 94]
247	[86, 83]
253	[96, 94]
286	[82, 72]
298	[87, 88]
323	[100, 94]
328	[90, 81]
386	[90, 82]
436	[87, 79]
507	[80, 90]
509	[79, 90]
510	[87, 88]
538	[86, 85]
539	[96, 96]
542	[83, 86]
551	[81, 85]
580	[92, 91]
589	[85, 80]
628	[90, 82]
639	[92, 94]
659	[88, 87]
695	[94, 94]
726	[75, 72]
...	...
86272	[98, 98]
86330	[92, 90]
86370	[90, 92]

	score_pair
business_id	
86386	[92, 94]
86587	[94, 93]
86590	[100, 100]
86629	[96, 94]
86643	[86, 81]
86647	[48, 66]
86658	[83, 80]
86682	[88, 86]
86694	[92, 89]
86765	[78, 79]
86814	[92, 96]
86845	[90, 74]
86933	[87, 98]
87106	[76, 56]
87122	[98, 100]
87149	[92, 98]
87154	[93, 88]
87200	[100, 98]
87202	[92, 96]
87277	[92, 79]
87337	[96, 94]
87440	[100, 96]
87761	[92, 86]
87802	[98, 91]
88323	[75, 75]
88756	[88, 80]
88792	[96, 100]

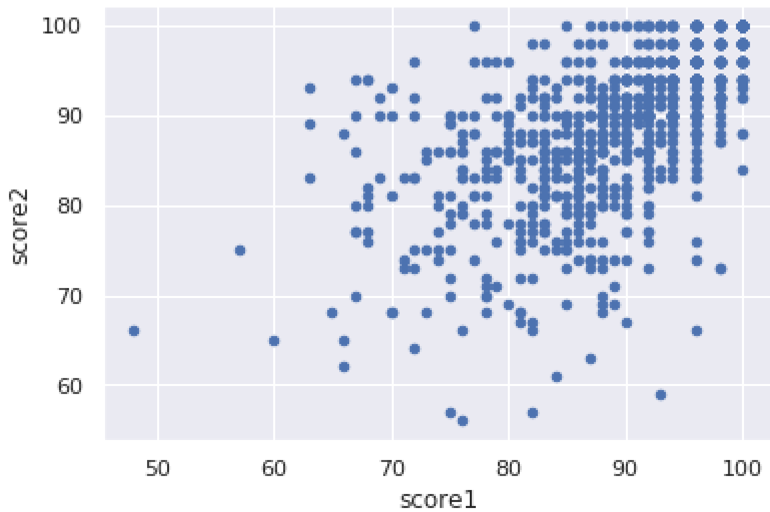
1076 rows × 1 columns

In [77]:
```
assert isinstance(scores_pairs_by_business, pd.DataFrame)
assert scores_pairs_by_business.columns == ['score_pair']
```

In [78]:
```
scoreframe = pd.DataFrame(scores_pairs_by_business)
scoreframe[['score1','score2']] = pd.DataFrame(scoreframe.score_pair.values.to
list(), index= scoreframe.index)
scoreframe.plot.scatter(x='score1',y='score2')
```

```
'c' argument looks like a single numeric RGB or RGBA sequence, which should b
e avoided as value-mapping will have precedence in case its length matches wi
th 'x' & 'y'.  Please use a 2-D array with a single row if you really want to
specify the same RGB or RGBA value for all points.
```

Out[78]: `<matplotlib.axes._subplots.AxesSubplot at 0x2b249626a320>`



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 [79]:
```
q7c_answer = r"""

You'd expect to see more dots above the line and to the right.
There would be a left skew in the histogram.
"""

print(q7c_answer)
```

```
You'd expect to see more dots above the line and to the right.
There would be a left skew in the histogram.
```

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.