food safety analysis

June 9, 2020

[1]:
```
# Initialize OK
from client.api.notebook import Notebook
ok = Notebook('proj1.ok')
```

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

1 Project 1: Food Safety

1.1 Cleaning and Exploring Data with Pandas

1.2 Due Date: Tuesday 09/24, 11:59 PM

1.3 Collaboration Policy

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

Collaborators: *list collaborators here*

1.4 This Assignment

In this project, you will investigate restaurant food safety scores for restaurants in San Francisco. Above is a sample score card for a restaurant. The scores and violation information have been made available by the San Francisco Department of Public Health. The main goal for this assignment is to understand how restaurants are scored. We will walk through various steps of exploratory data analysis to do this. We will provide comments and insights along the way to give you a sense of how we arrive at each discovery and what next steps it leads to.

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

1.5 Score Breakdown

Question	Points
1a	1
1b	0
1c	0
1d	3
1e	1
2a	1
2b	2
3a	2
3b	0
3c	2
3d	1
3e	1
3f	1
4a	2
4b	3
5a	1
5b	1
5c	1
6a	2
6b	3
6c	3
7a	2
7b	2
7c	6
7d	2
7e	3
Total	46

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

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

1. Import each of these libraries as their commonly used abbreviations (e.g., `pd`, `np`, `plt`, and `sns`).
2. Don't forget to include `%matplotlib inline` which enables inline matploblib plots.
3. If you want to use `seaborn`, add the line `sns.set()` to make your plots look nicer.

[2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
%matplotlib inline
import zipfile
```

```
[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
```

1.6 Downloading the Data

For this assignment, we need this data file: http://www.ds100.org/fa19/assets/datasets/proj1-SFBusinesses.zip

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

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

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

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

```
[4]: import ds100_utils
source_data_url = 'http://www.ds100.org/fa19/assets/datasets/proj1-SFBusinesses.
 ↪zip'
target_file_name = 'data.zip'

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

```
Using cached version that was downloaded (UTC): Sat Sep 21 15:06:46 2019
```

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

Note: The command below starts with an `!`. This tells our Jupyter notebook to pass this command to the operating system. In this case, the command is the `ls` Unix command which lists files in the current directory.

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

```
data            proj1.ipynb  __pycache__  q7d.png        test.tplx
data.zip        proj1.ok     q6a.png      scoreCard.jpg
ds100_utils.py  proj1.pdf    q7c2.png     tests
```

1.7 0. Before You Start

For all the assignments with programming practices, please write down your answer in the answer cell(s) right below the question.

We understand that it is helpful to have extra cells breaking down the process towards reaching your final answer. If you happen to create new cells below your answer to run codes, **NEVER** add cells between a question cell and the answer cell below it. It will cause errors in running Autograder, and sometimes fail to generate the PDF file.

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

1.8 1: Loading Food Safety Data

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

- Is the data in a standard format or encoding?
- Is the data organized in records?
- What are the fields in each record?

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

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

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

Hint: The Python docs describe how to create a `zipfile.ZipFile` object. You might also look back at the code from lecture and lab 4's optional hacking challenge. It's OK to copy and paste code from previous assignments and demos, though you might get more out of this exercise if you type out an answer.

[6]:
```python
my_zip = zipfile.ZipFile(dest_path, 'r')
list_names = my_zip.namelist()
list_names
```

[6]:
```
['violations.csv', 'businesses.csv', 'inspections.csv', 'legend.csv']
```

[7]:
```python
ok.grade("q1a");
```

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

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

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

Now display the files' names and their sizes.

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

[8]:
```python
for i in my_zip.filelist:
    print ('{}\t{}'.format(i.filename,i.file_size))
```

```
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 project the files are small, so we're just going to unzip everything. This has the added benefit that you can look inside the csv files using a text editor, which might be handy for understanding the structure of the files. The cell below will unzip the csv files into a subdirectory called `data`. Simply run this cell, i.e. don't modify it.

[9]:
```python
from pathlib import Path
data_dir = Path('data')
my_zip.extractall(data_dir)
!ls {data_dir}
```

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

The cell above created a folder called `data`, and in it there should be four CSV files. Let's open up `legend.csv` to see its contents. To do this, click on 'Jupyter' in the top left, then navigate to fa19/proj/proj1/data/ and click on `legend.csv`. The file will open up in another tab. You should see something that looks like:

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

1.8.2 Question 1b: Programatically Looking Inside the Files

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

[10]:
```python
for i in list_names:
    print(ds100_utils.head("./data/" + i, 5), "\n")
```

```
['"business_id","date","description"\n', '19,"20171211","Inadequate food safety
knowledge or lack of certified food safety manager"\n',
'19,"20171211","Unapproved or unmaintained equipment or utensils"\n',
'19,"20160513","Unapproved or unmaintained equipment or utensils  [ date
violation corrected: 12/11/2017 ]"\n', '19,"20160513","Unclean or degraded
floors walls or ceilings  [ date violation corrected: 12/11/2017 ]"\n']

['"business_id","name","address","city","state","postal_code","latitude","longit
ude","phone_number"\n', '19,"NRGIZE LIFESTYLE CAFE","1200 VAN NESS AVE, 3RD
FLOOR","San Francisco","CA","94109","37.786848","-122.421547","+14157763262"\n',
'24,"OMNI S.F. HOTEL - 2ND FLOOR PANTRY","500 CALIFORNIA ST, 2ND  FLOOR","San
Francisco","CA","94104","37.792888","-122.403135","+14156779494"\n',
'31,"NORMAN\'S ICE CREAM AND FREEZES","2801 LEAVENWORTH ST ","San
Francisco","CA","94133","37.807155","-122.419004",""\n', '45,"CHARLIE\'S DELI
CAFE","3202 FOLSOM ST ","San
Francisco","CA","94110","37.747114","-122.413641","+14156415051"\n']

['"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,"Adequate"\n', '91,100,"Good"\n']
```

1.8.3 Question 1c: Reading in the Files

Based on the above information, let's attempt to load `businesses.csv`, `inspections.csv`, and `violations.csv` into pandas dataframes 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`. At some point in your future, you should read all about character encodings. We won't discuss these in detail in DS100.

[11]:
```
# path to directory containing data
dsDir = Path('data')

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

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

[12]:
```
display(bus.head())
display(ins.head())
display(vio.head())
```

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

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

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


   business_id  score      date     type
0           19     94  20160513  routine
```

```
1           19     94  20171211  routine
2           24     98  20171101  routine
3           24     98  20161005  routine
4           24     96  20160311  routine
```

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

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

[13]:
```python
display(bus.describe())
display(ins.describe())
display(vio.describe())
```

```
        business_id     latitude   longitude
count   6406.000000  3270.000000 3270.000000
mean   53058.248049    37.773662 -122.425791
std    34928.238762     0.022910    0.027762
min       19.000000    37.668824 -122.510896
25%     7405.500000    37.760487 -122.436844
50%    68294.500000    37.780435 -122.418855
75%    83446.500000    37.789951 -122.406609
max    94574.000000    37.824494 -122.368257
```

```
        business_id         score          date
count  14222.000000  14222.000000  1.422200e+04
mean   45138.752637     90.697370  2.016242e+07
std    34497.913056      8.088705  8.082778e+03
min       19.000000     48.000000  2.015013e+07
25%     5634.000000     86.000000  2.016021e+07
50%    61462.000000     92.000000  2.016091e+07
75%    78074.000000     96.000000  2.017061e+07
max    94231.000000    100.000000  2.018012e+07
```

```
        business_id          date
count  39042.000000  3.904200e+04
mean   45674.440244  2.016283e+07
std    34172.433276  7.874679e+03
min       19.000000  2.015013e+07
25%     4959.000000  2.016031e+07
50%    62060.000000  2.016092e+07
```

```
75%     77681.000000  2.017063e+07
max     94231.000000  2.018012e+07
```

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:

[14]:
```
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.

[15]:
```
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.

1.9 Question 1d: Verifying the data

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

[16]:
```
"""Run this cell to load this utility comparison function that we will use in␣
↪various
tests below (both tests you can see and those we run internally for grading).

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


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

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

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

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

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

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

[17]:

```
ok.grade("q1d");
```

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

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

1.9.1 Question 1e: Identifying Issues with the Data

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

There is a missing phone number for NORMAN'S ICECREAM AND FREEZES.

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

1.10 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.

1.10.1 Question 2a

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

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

[18]:
```
is_business_id_unique = bus['business_id'].value_counts().max() == 1
```

[18]:
```
True
```

[19]:
```
ok.grade("q2a");
```

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

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

1.10.2 Question 2b

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

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

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

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

1. There are 6406 rows in "bus" and there are each row represents a unique business_id, i.e. a restaurant.
2. Thus, the primary key is the business_id, since it is a unique value for each row, it is used to differenciate between two different resturants.

3. If we group by business_id, we would get the same dataFrame back. If we grouped by name, we would still be grouping by restaurant, so those with the same name will have the same address.

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

[20]:
```
   business_id                            name  \
0           19             NRGIZE LIFESTYLE CAFE
1           24  OMNI S.F. HOTEL - 2ND FLOOR PANTRY
2           31      NORMAN'S ICE CREAM AND FREEZES
3           45                 CHARLIE'S DELI CAFE
4           48                          ART'S CAFE

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

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

1.11 3: Zip Codes

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

1.11.1 Question 3a

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

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

1. The zip codes are qualitative. It is all nominal.
2. They are stored as strings currently.

1.11.2 Question 3b

How many restaurants are in each ZIP code?

In the cell below, create a series where the index is the postal code and the value is the number of records with that postal code in descending order of count. 94110 should be at the top with a count of 596. You'll need to use `groupby()`. You may also want to use `.size()` or `.value_counts()`.

[21]:
```
zip_counts = bus.groupby("postal_code").size().sort_values(ascending = False)
zip_counts.head()
```

[21]:
```
postal_code
94110    596
94103    552
94102    462
94107    460
94133    426
dtype: int64
```

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

[22]:
```
print('zip_counts describes', sum(zip_counts), 'records.')
print('The original data have', len(bus), 'records')
```

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

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

[23]:
```
zip_counts = bus.fillna("?????").groupby("postal_code").size().
 ↪sort_values(ascending=False)
zip_counts.head(15)
```

[23]:
```
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.

[24]:
```
bus["postal_code"].value_counts(dropna=False).sort_values(ascending = False).
↪head(15)
```

[24]:
```
94110    596
94103    552
94102    462
94107    460
94133    426
94109    380
94111    277
94122    273
94118    249
94115    243
NaN      240
94105    232
94108    228
94114    223
94117    204
Name: postal_code, dtype: int64
```

Missing zip codes aren't our only problem. There are also some records where the postal code is wrong, e.g., there are 3 'Ca' and 3 'CA' values. Additionally, there are some extended postal codes that are 9 digits long, rather than the typical 5 digits. We will dive deeper into problems with postal code entries in subsequent questions.

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

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

[25]:
```
bus['postal_code_5'] = bus['postal_code'].str[:5]
bus.head()
```

[25]:
```
   business_id                                 name  \
0           19                 NRGIZE LIFESTYLE CAFE
1           24  OMNI S.F. HOTEL - 2ND FLOOR PANTRY
```

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

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

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

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

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

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

Some records have missing postal codes because they are off the grid, which means they may be moving resturant locations. And thus, this is not out of the ordinary.

[26]:
```
bus[bus['postal_code'].isnull()]['address'].value_counts()
```

[26]:
```
 OFF THE GRID                       69
 APPROVED PRIVATE LOCATIONS          6
 APPROVED LOCATIONS                  4
428 11TH ST                          2
OFF THE GRID                         2
                                    ..
681 BROADWAY ST                      1
1605 JERROLD AVE                     1
236 TOWNSEND ST                      1
301 25TH AVE                         1
 GOLDEN GATE PARK, SPRECKLES LAKE    1
Name: address, Length: 159, dtype: int64
```

1.11.4 Question 3d: Incorrect ZIP Codes

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

[27]:
```
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 that contains only rows corresponding to ZIP codes that are 'weird'. We define weird as any zip code which has both of the following 2 properties:

1. The zip code is not valid: Either not 5-digit long or not a San Francisco zip code.
2. The zip is not missing.

Use the `postal_code_5` column.

Hint: The ~ operator inverts a boolean array. Use in conjunction with `isin` from lecture 3.

[28]:
```
weird_zip_code_businesses = bus[~bus['postal_code_5'].isin(all_sf_zip_codes)]
weird_zip_code_businesses
```

[28]:
```
      business_id                        name                   address  \
1211         5208      GOLDEN GATE YACHT CLUB                 1 YACHT RD
1372         5755              J & J VENDING  VARIOUS LOACATIONS (17)
1373         5757          RICO VENDING, INC        VARIOUS LOCATIONS
1702         8202                  XIAO LOONG   250 WEST PORTAL AVENUE
1725         9358   EDGEWOOD CHILDREN'S CENTER           1801 VICENTE ST
…             …                          …                       …
6223        92857             MOBI MUNCH, INC.             OFF THE GRID
6240        93029                 BAHN MI ZON             OFF THE GRID
6300        93484          CARDONA'S FOOD TRUCK          2430 WHIPPLE RD
6354        94123          BON APPETIT @ AIRBNB            999 BRANNAN ST
6387        94409                 AUGUST HALL               420 MASON ST

               city state postal_code   latitude   longitude phone_number  \
1211  San Francisco    CA         941  37.807878 -122.442499 +14153462628
1372  San Francisco    CA       94545        NaN         NaN +14156750910
1373  San Francisco    CA       94066        NaN         NaN +14155836723
1702  San Francisco    CA         NaN  37.738616 -122.468775 +14152792647
1725  San Francisco    CA         NaN  37.739083 -122.485437          NaN
…                 …   …           …          …           …            …
```

```
6223  San Francisco    CA          NaN        NaN          NaN  +14152899800
6240  San Francisco    CA          NaN        NaN          NaN  +14152414342
6300  San Francisco    CA        94544        NaN          NaN  +14153365990
6354  San Francisco    CA          NaN        NaN          NaN  +1415 Alieri
6387  San Francisco    CA          NaN        NaN          NaN           NaN

     postal_code_5
1211           941
1372         94545
1373         94066
1702           NaN
1725           NaN
…            …
6223           NaN
6240           NaN
6300         94544
6354           NaN
6387           NaN

[261 rows x 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.

1. 94545 - Hayward, if you look at the dataFrame, you can see that it's a vending machine company with many locations.
2. 94602 - Oakland, this is probably a typo and it should be 94102.

1.11.5 Question 3e

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

The value 94602 is wrong. Change it to the most reasonable correct value, using all information you have available from your internet search for real world business. Modify the `postal_code_5` field using `bus['postal_code_5'].str.replace` to replace 94602.

[29]:
```python
# 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')
```

[30]:
```python
ok.grade("q3e");
```

~~~~~~
~~~~~~

```
Running tests

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

1.11.6 Question 3f

Now that we have corrected one of the weird postal codes, let's filter our `bus` data such that only postal codes from San Francisco remain. While we're at it, we'll also remove the businesses that are missing a postal code. As we mentioned in question 3d, filtering our postal codes in this way may not be ideal. (Fortunately, this is just a course assignment.) Use the `postal_code_5` column.

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

[31]:
```
bus = bus[(bus['postal_code_5'].isin(all_sf_zip_codes)) &␣
 ↪(~(bus['postal_code_5'].isnull()))]
bus.head()
```

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

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

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

[32]:
```
ok.grade("q3f");
```

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

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

1.12 4: Latitude and Longitude

Let's also consider latitude and longitude values in the `bus` data frame and get a sense of how many are missing.

1.12.1 Question 4a

How many businesses are missing longitude values?

Hint: Use `isnull`.

[33]:
```
num_missing_longs = sum(bus['longitude'].isnull())
num_missing_longs
```

[33]:
```
2942
```

[34]:
```
ok.grade("q4a1");
```

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

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

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

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

[35]:
```
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 (the values should be in descending order). The first two rows of your answer should include postal code 94103 and 94110. Only businesses from `sf_dense_zip` should be included.

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

Hint: Use `len` or `sum` to find out the output number.

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

[36]:
```python
busses_sf = bus[bus['postal_code_5'].isin(sf_dense_zip)]
num_of_busses = lambda x : len(x[x.isnull()])

num_missing_in_each_zip = busses_sf.groupby('postal_code_5')['longitude'].
↪agg(num_of_busses).sort_values(ascending = False)
num_missing_in_each_zip.head()
```

[36]:
```
postal_code_5
94110    294.0
94103    285.0
94107    275.0
94102    222.0
94109    171.0
Name: longitude, dtype: float64
```

[37]:
```python
ok.grade("q4a2");
```

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

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

1.12.2 Question 4b

In question 4a, we counted the number of null values per ZIP code. Reminder: we still only use the zip codes found in `sf_dense_zip`. Let's now count the proportion of null values of longitudinal coordinates.

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

1. `count null`: The number of missing values for the zip code.

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

Your data frame should be sorted by the fraction null in descending order. The first two rows of your answer should include postal code 94107 and 94124.

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.

To pursue this recommended approach, you might find these two functions useful and you aren't required to use these two:

- `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. Be careful about `axis=1`.

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

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

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

[38]:
```
def count_null(s):
    return len(s[s.isnull()]);
def count_non_null(s):
    return len(s[~s.isnull()]);
def fraction_null(s):
    helper_1 = len(s[s.isnull()]);
    helper_2 = len(s[~s.isnull()]);
    return (helper_1)/(helper_1 + helper_2)
busses_sf = bus[bus['postal_code_5'].isin(sf_dense_zip)]
fraction_missing_df = busses_sf['longitude'].groupby(bus['postal_code_5']).
 ↪agg([count_non_null, count_null, fraction_null])
fraction_missing_df.columns = ['count non null', 'count null', 'fraction null']
fraction_missing_df = fraction_missing_df.sort_values('fraction null',␣
 ↪ascending = False)
fraction_missing_df.head()
```

[38]:

postal_code_5	count non null	count null	fraction null
94124	73.0	118.0	0.617801
94107	185.0	275.0	0.597826
94104	60.0	79.0	0.568345
94105	105.0	127.0	0.547414
94132	62.0	71.0	0.533835

[39]:
```
ok.grade("q4b");
```

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

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

1.13 Summary of the Business Data

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

- We found that the business id is unique across records and so we may be able to use it as a key in joining tables.
- We found that there are some errors with the ZIP codes. As a result, we dropped the records with ZIP codes outside of San Francisco or ones that were missing. In practive, however, we could take the time to look up the restaurant address online and fix these errors.
- We found that there are a huge number of missing longitude (and latitude) values. Fixing would require a lot of work, but could in principle be automated for records with well-formed addresses.

1.14 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.

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

[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

1.14.1 Question 5a

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

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

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

[42]:
```
ok.grade("q5a");
```

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

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

1.14.2 Question 5b

Next, let us examine the Series in the `ins` dataframe called `type`. From examining the first few rows of `ins`, we see that `type` takes string value, one of which is `'routine'`, presumably for a routine inspection. What other values does the inspection `type` take? How many occurrences of each value is in `ins`? What can we tell about these values? Can we use them for further analysis? If so, how?

All the records have the same variable, "routine" except for one. This should not be useful for our analysis, as it provides no additional, significant information.

1.14.3 Question 5c

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

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

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

[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.

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

[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.

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

[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 the markdown cell below. If you would like show your reasoning with codes, make sure you put your code cells **below** the markdown answer cell.

The ranges of years are 2018, 2017, 2016, 2015. No, the number of inspections are not the same - 2018 only has a few, and 2015 has a subtantially lower number than 2016 or 2017.

1.15 6: Explore Inspection Scores

1.15.1 Question 6a

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

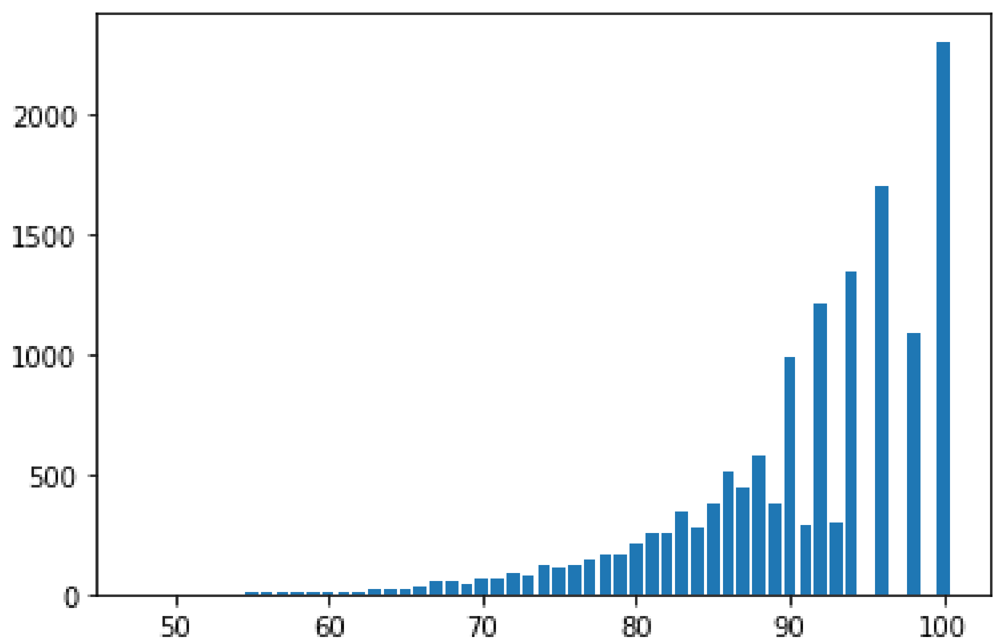
It should look like the image below. It does not need to look exactly the same (e.g., no grid), but make sure that all labels and axes are correct.

You might find this matplotlib.pyplot tutorial useful. Key syntax that you'll need: + `plt.bar` + `plt.xlabel` + `plt.ylabel` + `plt.title`

Note: If you want to use another plotting library for your plots (e.g. `plotly`, `sns`) you are welcome to use that library instead so long as it works on DataHub. If you use seaborn `sns.countplot()`, you may need to manually set what to display on xticks.

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

[46]: `<BarContainer object of 47 artists>`



1.15.2 Question 6b

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

The distribution is unimodel with a peak at 100. It is skewed to the left, as expected since the variable is bounded to the right. The distribution also has a long left tail, with some restaurants recieving scores in the 50s, 60s or 70s. An unusual feature of this distribution is that even number scores have higher counts than odd number scores. This could be bacause the penalty scores incredment in even numbers (eg: 2, 4, 6, 8).

1.15.3 Question 6c

Let's figure out which restaurants had the worst scores ever (single lowest score). 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. See the official documentation on how to use `merge`.

Note: For quick reference, a pandas 'left' join keeps the keys from the left frame, so if ins is the left frame, all the keys from ins are kept and if a set of these keys don't have matches in the other frame, the columns from the other frame for these "unmatched" key rows contains NaNs.

[47]:
```
ins_named = ins.merge(bus[["business_id", "name", "address"]], how = 'left', on␣
↪= "business_id")
ins_named.head()
```

[47]:
```
   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

                              name                       address
0                NRGIZE LIFESTYLE CAFE   1200 VAN NESS AVE, 3RD FLOOR
1                NRGIZE LIFESTYLE CAFE   1200 VAN NESS AVE, 3RD FLOOR
2  OMNI S.F. HOTEL - 2ND FLOOR PANTRY  500 CALIFORNIA ST, 2ND  FLOOR
3  OMNI S.F. HOTEL - 2ND FLOOR PANTRY  500 CALIFORNIA ST, 2ND  FLOOR
4  OMNI S.F. HOTEL - 2ND FLOOR PANTRY  500 CALIFORNIA ST, 2ND  FLOOR
```

[48]:
```
ok.grade("q6c1");
```

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

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

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

The resturant with the lowest inspection score is D&A Cafe. One funny review I read was: "Wipes counter. Wipes nose. Handles cash. Puts a straw in your drink. Not just one staff member but all 3 ladies at the counter did this. Not sure they could earn their 72 inspection score on a regular day."

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

1.16 7: Restaurant Ratings Over Time

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

1.16.1 Question 7a

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

Note: The "swing" is of a specific business. There might be some restaurants with multiple locations; each location has its own "swing".

[49]:
```
def swing(s):
    return max(s) - min(s)
three_and_more = ins_named.groupby("business_id").filter(lambda x:␣
 ↪len(x['business_id']) >= 3)
agg_swing = three_and_more.groupby("business_id").agg(swing)
num_max_swing = max(agg_swing['score'])
max_swing_id = agg_swing[agg_swing['score'] == num_max_swing].index[0]
max_swing = ins_named[ins_named['business_id'] == max_swing_id]['name'][1207]
max_swing
```

[49]:
```
"JOANIE'S DINER INC."
```

[50]:
```
ok.grade("q7a1");
```

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

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

1.16.2 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`.

[51]:
```
inspections_by_id_and_year = ins_named.groupby(['business_id', 'year']).size().
 ↪to_frame().rename(columns ={0:"count"})
inspections_by_id_and_year.head()
```

[51]:

business_id	year	count
19	2016	1
	2017	1
24	2016	2
	2017	1
31	2015	1

[52]:
```
ok.grade("q7b");
```

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

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

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

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

[53]:
```
1    9531
2    2175
3     111
4       2
Name: count, dtype: int64
```

1.16.3 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, using `ins2016` data frame that will be created for you below.

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 should look like this:

In the cell below, create `scores_pairs_by_business` as described above.

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

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

Hint: Our code that creates the needed DataFrame 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.

[54]:
```
# Create the dataframe here
def group_to_list(group):
    return list(group)
ins2016 = ins[ins['year'] == 2016]
scores_pairs_by_business = (ins2016.sort_values('date').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'}))
```

[54]: 5443

[55]:
```
ok.grade("q7c1");
```

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

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

Now, create your scatter plot in the cell below. It does not need to look exactly the same (e.g., no grid) as the above sample, but make sure that all labels, axes and data itself are correct.

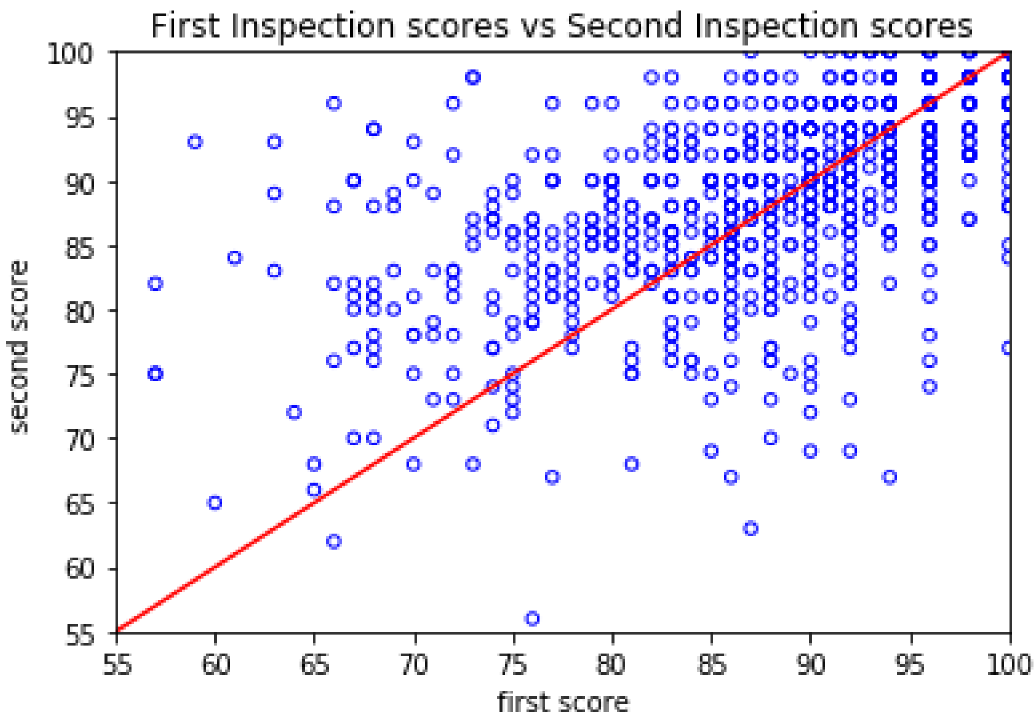
Key pieces of syntax you'll need: + `plt.scatter` plots a set of points. Use `facecolors='none'` to make circle markers. + `plt.plot` for the reference line. + `plt.xlabel`, `plt.ylabel`, `plt.axis`, and `plt.title`.

Note: If you want to use another plotting library for your plots (e.g. `plotly`, `sns`) you are welcome to use that library instead so long as it works on DataHub.

Hint: You may find it convenient to use the `zip()` function to unzip scores in the list.

[58]:
```
first_score, second_score = zip(*scores_pairs_by_business['score_pair'])
plt.scatter(first_score,second_score,s=20,facecolors='none',edgecolors='b')
plt.plot([55,100],[55,100],'r-')
plt.xlabel('first score')
```

```
plt.ylabel('second score')
plt.axis([55,100,55,100]);
plt.title("First Inspection scores vs Second Inspection scores")
plt.show()
```



1.16.4 Question 7d

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

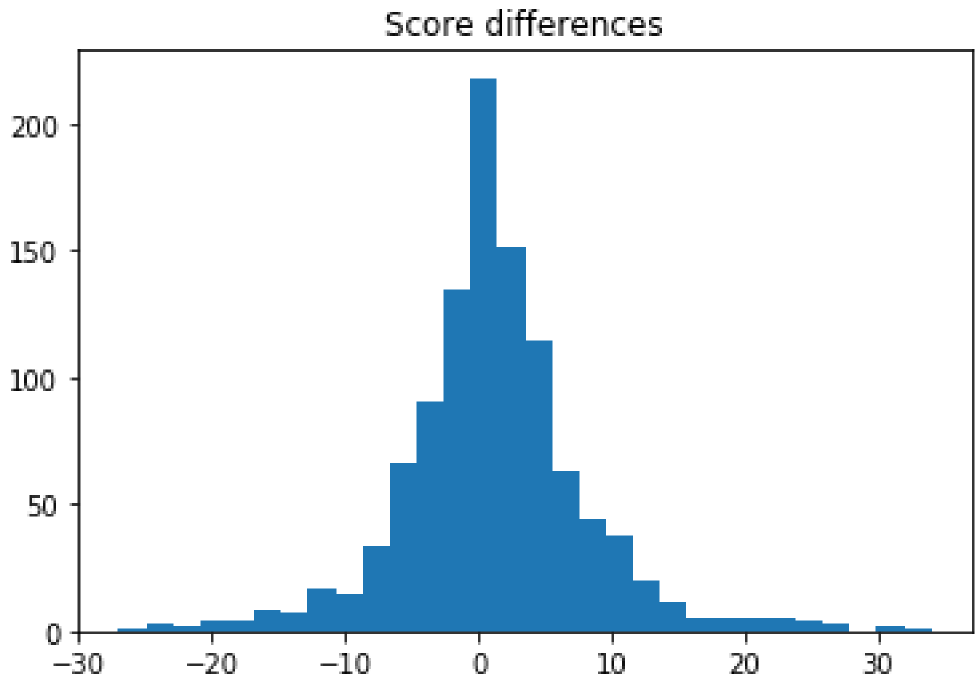
The histogram should look like this:

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

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

Hint: Use `plt.hist()` Try changing the number of bins when you call `plt.hist()`.

[59]:
```
diffs = np.array(second_score) - np.array(first_score)
plt.hist(diffs,bins=30);
plt.title("Score differences")
plt.show()
```



1.16.5 Question 7e

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

If the restaurants tend to improve from the first to the second inspection. In a scatter plot, we would expect to see the points in the to fall above the line of slope 1. The histogram of differences shows a unimodal distribution with a peak below 0.

1.17 Summary of the Inspections Data

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

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

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

- We also examined the relationship between the scores when a restaurant has multiple inspections in a year. Our findings were a bit counterintuitive and may warrant further investigation.

1.18 Congratulations!

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

2 Submit

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

[60]:
```
# Save your notebook first, then run this cell to submit.
import jassign.to_pdf
jassign.to_pdf.generate_pdf('proj1.ipynb', 'proj1.pdf')
ok.submit()
```

```
Generating PDF…
Saved proj1.pdf

<IPython.core.display.Javascript object>


<IPython.core.display.Javascript object>


Saving notebook… Saved 'proj1.ipynb'.
Submit… 100% complete
Submission successful for user: jain12767@berkeley.edu
URL: https://okpy.org/cal/data100/fa19/proj1/submissions/3Q7g1M
```

[ ]:
```

```

[ ]:
```

```