proj1

August 7, 2021

```
[2]: # 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.

[3]:
```
%matplotlib inline
import os
import zipfile as zf
import pandas as pd
import numpy as np
import seaborn as sns
```

```
import matplotlib.pyplot as plt
```

[4]:
```
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).

[6]:
```
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): Thu Sep 19 12:11:14 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.

[7]:
```
!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.

[10]:
```
import zipfile
my_zip = zipfile.ZipFile(dest_path, mode ='r')
list_names = my_zip.namelist()
list_names
```

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

[11]:
```
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.

[12]:
```
my_zip.infolist()
```

[12]:
```
[<ZipInfo filename='violations.csv' compress_type=deflate external_attr=0x20
file_size=3726206 compress_size=286253>,
 <ZipInfo filename='businesses.csv' compress_type=deflate external_attr=0x20
file_size=660231 compress_size=178549>,
 <ZipInfo filename='inspections.csv' compress_type=deflate external_attr=0x20
file_size=466106 compress_size=83198>,
 <ZipInfo filename='legend.csv' compress_type=deflate external_attr=0x20
file_size=120 compress_size=104>]
```

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

[13]:
```
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.

[14]:
```
'data/' + 'legend.csv'
```

[14]:
```
'data/legend.csv'
```

[15]:
```
for x in list_names:
    name = 'data/' + x
    info = ds100_utils.head(name, 5)
    print(name)
    print("\n")
    print(info)
    print("\n")
```

```
data/violations.csv


['"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']


data/businesses.csv


['"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']
```

```
data/inspections.csv
```

```
['"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']
```

```
data/legend.csv
```

```
['"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.

[16]:
```
# path to directory containing data
dsDir = Path('data')

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

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

[17]:
```
display(bus.head(5), ins.head(5), vio.head(5))
```

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

```
4          48                          ART'S CAFE

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

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

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

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

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

[18]:
```python
bus.describe()
```

[18]:

	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

[19]:
```python
ins.describe()
```

[19]:

	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
```

[20]:
```
vio.describe()
```

[20]:

	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:

[21]:
```
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.

[22]:
```
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.

[23]:
```python
"""Run this cell to load this utility comparison function that we will use in␣
↪various
tests below (both tests you can see and those we run internally for grading).

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


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

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

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

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

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

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

[24]:
```python
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.

[26]:
```
bus.head()
```

[26]:
```
   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
```

The head command gives you the first few rows of a spreadsheet(csv) file that may contain millions of rows. Therefore, we are not given any information about the values that may follow and the first n element is not liekly to be representative of the data in the spreadsheete overall. I believe this can be avoided by sorting the table in some sorts that will give more meaning to what it is to be the first n rows of the table displayed by head commmand.

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.

[27]:
```
is_business_id_unique = len(bus['business_id'].unique()) ==␣
↪len(bus['business_id'])
```

[29]:
```
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) Since we saw that each row is unique in terms of their business_id (each row has a different business_id), each record is represents a business, with its name and the address (location). Each row represents a unique value for different restaurants
2) The primary key would be the business_id because we have seen that it is unique to all the rows in the spreadsheet and hence will be the key indistinguishing one recrod from another
3) When you group by the business_id, not much will happen because every row is unique and hence there will not be any grouping.
   However, if you group by the name, if any business has the same name (e.g. is a franchise) then it will group them together. It will tell you whether or not some businesses share the same name Lastly, if you group by addresses then the business/stores that is ran on the same location will be grouped together and hence it will tell you if any businesses are located at the same branch (e.g. same building) and group them together

---

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) Zip codes are qualitative because it does not make sense to do number operations on ZIP codes such as averaging, finding the max ZIP code, etc. Zip codes are nominal because there are no ranks present between the ZIP codes is will be with nominal data.
2) The data type of the ZIP code is an object and the type of the object is string. This was given by the code 'type(bus['postal_code'][1])'. Data type is String

[34]:
```
type(bus['latitude'][1])
```

[34]:
```
numpy.float64
```

[35]:
```
bus.head(2)
```

[35]:
```
   business_id                               name  \
0           19                NRGIZE LIFESTYLE CAFE
1           24  OMNI S.F. HOTEL - 2ND FLOOR PANTRY

                     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

    longitude  phone_number
0 -122.421547  +14157763262
1 -122.403135  +14156779494
```

[36]:
```
bus.dtypes
```

[36]:
```
business_id       int64
name             object
address          object
city             object
state            object
postal_code      object
latitude        float64
longitude       float64
phone_number     object
dtype: object
```

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()`.

[38]:
```
zip_counts = bus['postal_code'].value_counts()
zip_counts
```

[38]:
```
94110    596
94103    552
94102    462
94107    460
94133    426
94109    380
94111    277
94122    273
94118    249
94115    243
94105    232
94108    228
94114    223
94117    204
94112    195
94124    191
94123    173
94121    160
94104    139
94132    133
94116     99
94134     77
94127     71
94131     49
94158     32
94130      7
94143      5
94188      4
00000      2
94014      2
94101      2
94013      2
94129      2
CA         2
94544      1
94602      1
92672      1
```

```
941              1
94120            1
94621            1
Ca               1
64110            1
941033148        1
941102019        1
94066            1
94545            1
95105            1
94080            1
Name: postal_code, dtype: int64
```

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

[39]:
```
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.

[40]:
```
zip_counts = bus.fillna("?????").groupby("postal_code").size().
↪sort_values(ascending=False)
zip_counts.head(15)
```

[40]:
```
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.

[41]:
```
bus["postal_code"].value_counts(dropna=False).sort_values(ascending = False).
↪head(15)
```

[41]:
```
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.

[42]:
```
bus['postal_code_5'] = bus['postal_code'].str[:5]
bus.head()
```

[42]:
```
   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.

Firstly, some records has 'OFF THE GRID' as their address without their latitude and longitude listed. Therefore, it would not be possible to determine the postal_code for these locations. Moreover, some has the name the building such as the address 'HUNTERS POINT BUILDING 110 SHIPYARD' which will not be able to give the zip_code (it is not an address or its correct form) Some addresses describes a part of one location divided into sections such as 'GOLDEN GATE PARK, JFK DR.@CONSERVATORY OF FLOWERS' and 'GOLDEN GATE PARK, SPRECKLES LAKE'. This probability will not have a post to be delievered and hence will not have an exact postal code Some addresses are a group of locations such as 'VARIOUS FARMERS MARKETS' or 'APPROVED PUBLIC LOCATIONS' and hence it is not possible to get an exact postal_code. Moreover, some address is not specific such that there may be multiple places with the same address. For instance, '1717 HARRISON ST' is a place in Oakland, San Francisco, etc. These group of addresses will not have a postal_code/multiple postal_codes or incorrect postal_code, depending on what the data storing algorithm decides to do. some addresses are missing some parts of it that makes it a full, complete address. For instance '928 TOLAND' is complete by the St at the end (928 TOLAND st.). Some do not follow the conventions of the address by having too much info or not the right config e.g. 250 WEST PORTAL avenue should be 250 W Portal Ave.

[167]:
```python
null_series = bus['postal_code'].isnull()
bus['address'][null_series.values].unique()
```

[167]:
```
array(['250 WEST PORTAL AVENUE', '1801 VICENTE ST ', '1700 POST ST.',
       '819 VALENCIA ST.', '3200 24TH ST ', '515 CORTLAND AVE',
       '1717 HARRISON ST ', '10 29TH ST ', '3918 JUDAH ST',
       'CORNER OF ALAMEDA AND HENRY ADAMS', '3914 JUDAH ST',
       'VARIOUS LOCATIONS', '4801 03RD ST ', '3055 23RD ST ',
       '750 FONT BLVD', '450 CHURCH ST ', '203 PARNASSUS AVE ',
       '56 JULIAN', '100 DIAMOND ST ', '3611 18TH ST ',
       '6134 GEARY BLVD ', '2499 LOMBARD ST ', '50 BEALE ST 105',
```

```
'4033 JUDAH ST ', ' NW CORNER GRANT AT GEARY ST ON GRANT',
'1099 MISSION ST ', ' PIER26 EMBARCADERO  ',
'1301 CESAR CHAVEZ ST ', '491 BAYSHORE ST ', '833 BRYANT ST ',
'24 WILLIE MAYS PLAZA  ', '79 SANSOME ST ', '705 NATOMA ST ',
'101 4TH ST ', ' GOLDEN GATE PARK, MUSIC CONCOURSE  ',
' GOLDEN GATE PARK, JFK DR.@CONSERVATORY OF FLOWERS  ',
' GOLDEN GATE PARK, SPRECKLES LAKE  ',
' GOLDEN GATE PARK, JFK DR.@8TH AVE  ',
' GOLDEN GATE PARK, CAROUSEL SNACK BAR  ', '1 UNITED NATIONS PL ',
' GOLDEN GATE PARK  ', '2 EMBARCADERO CENTER  STREET LEVEL',
' JUSTIN HERMAN PLAZA  ', ' FORT MASON',
' OFF THE GRID-UPPER HAIGHT  ', ' OFF THE GRID  ',
'550 D GENE FRIEND WAY ', '550 GENE FRIENDS WAY ',
'550 A GENE FRIEND WAY ', '1001 POTRERO AVE ',
' MACYS - GEARY ENTRANCE  ', '601 FOLSOM ST ', '6314 GEARY BLVD  ',
'933 BRANNAN ST ', '525 MARKET/360 VALENCIA  ',
' HUNTERS POINT BUILDING 110 SHIPYARD  ',
' TREASURE ISLAND FLEA MARKET  ', '135 04TH ST FC-3',
' MUSIC CONCOURSE IN GOLDEN GATE PARK  ', '3801 18TH ST ',
'625 CLEMENT ST ', '100 NEW MONTGOMERY ST ', '1605 JERROLD AVE ',
'2826 JONES ST ', '550 GOUGH ST ', '428 11TH  ', '11 PHELAN AVE ',
'298 KING ST ', '1975 BRYANT  ', '845 MARKET ST #13',
' FRONT, BETWEEN CALIFORNIA & SACRAMENTO ST ', '1780 HAIGHT ST  ',
'2462 SAN BRUNO AVE ', '2501 PHELPS ST ',
' SOMA STREET FOOD PARK  ', ' VARIOUS FARMERS MARKETS  ',
' SOMA STREET @ 428 11TH ST.  ', '2781 21ST ST ', ' HAIGHT  ',
'3200 FILLMORE ST ', '140 NEW MONTGOMERY ST  ', '1760 POLK ST ',
' DOLORES PK ', ' TREASURE ISLAND',
' APPROVED PRIVATE LOCATIONS  ', ' PRIVATE & PUBLIC  ',
'839 CLAY  ', '582 SUTTER ST ', ' PIER 39 WEST PERIMETER  ',
'101 HORNE AVE ', '   ', '400 CALIFORNIA  ',
'110 HUNTERS POINT SHIPYARD  ',
' HUNTERS POINT SHIPYARD, BLDG.#110  ', '45 MINT PLAZA  ',
'75 1 ST ST  ', '2399 VAN NESS AVE ', '484 ELLIS ST ',
'871 SUTTER ST ', '1111 CALIFORNIA ST ', '2240 CHESTNUT ST ',
'681 BROADWAY ST ', '55 STOCKTON ST ', '144 TAYLOR ST ',
'370 GOLDEN GATE AVE ', '942 MISSION ST ', '155 FELL ST ',
'1355 MARKET ST ', '3435 MISSION ST ', '670 LARKIN ST ',
'3861 24TH ST ', '1400 STOCKTON ST ', '2229 CLEMENT ST ',
'685 MARKET ST 520', '1 FERRY BUIILDING PL ', '1101 FAIRFAX AVE ',
'1051 MARKET ST ', '236 TOWNSEND ST ', '428 11TH ST ',
'2206 POLK ST ', '115 SANSOME  ', '3251 20TH AVE ',
'510 STEVENSON ST ', '24 WILLIE MAYS PL ', 'OFF THE GRID',
'610 LONG BRIDGE  ', '855 BUSH ST ', ' APPROVED LOCATIONS  ',
' BEACH CHALET SOCCER FIELD PARKING LOT  ', '659 MERCHANT ST ',
'500 POST ST ', '842 GEARY ST ', '1552 OCEAN AVE ',
' TFF EVENT OPERATIONS  ', '3331 24TH ST ', '752 VAN NESS AVE ',
```

```
       '1737 POST ST 368', '1130 OCEAN AVE ', '2078 HAYES ST ',
       '235 FRONT ST ', '255 WINSTON ST ', '1143 TARAVAL ST ',
       ' APPROVED PUBLIC LOCATIONS  ',
       ' MISSION ST, BETW 10TH & 11TH ST  ', '301 25TH AVE ',
       '2277 SHAFTER AVE ', ' PRIVATE LOCATIONS  ', '1220 09TH AVE ',
       "170 O'FARRELL ST ", '2831 CESAR CHAVEZ ST ', "333 O'FARRELL  ",
       ' TREASURE ISLAND  ', '201 2ND ST ', '993 NORTH POINT ST ',
       '928 TOLAND  ', ' OTG  ', '655 MONTGOMERY ST ', '999 BRANNAN ST  ',
       '420 MASON ST '], dtype=object)
```

[168]:
```
null_series = bus['postal_code'].notnull()
bus['address'][null_series].unique()
```

[168]:
```
array(['1200 VAN NESS AVE, 3RD FLOOR', '500 CALIFORNIA ST, 2ND  FLOOR',
       '2801 LEAVENWORTH ST ', …, '199 MUSEUM WAY ', '1745 TARAVAL ST ',
       '684 BROADWAY ST '], dtype=object)
```

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.

[43]:
```
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.

[44]:
```
weird_zip_code_businesses = bus[~bus['postal_code_5'].isin(all_sf_zip_codes)]
weird_zip_code_businesses[(weird_zip_code_businesses['postal_code_5'] ==␣
 ↪'94602')]
```

[44]:
```
      business_id        name          address           city state  \
5060        85459  ORBIT ROOM  1900 MARKET ST   San Francisco    CA
```

```
      postal_code  latitude  longitude  phone_number postal_code_5
5060        94602       NaN        NaN  +14153705584         94602
```

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.

ZIP code 94545: Hayward, CA, Russel City, CA (Postal code in Alameda County, CA). Zip code 94602: Oakland, CA. For zipcode 94545 the address is 'Various Locations(17)' which indicates that the same shop/business that may have started in SF or is in SF can also be other locations. Therefore, when the data was collected for the postal_code it may have inputted the postal code of the other shop location. Or it may have moved location and it has not been updated. Or it may be so that the address written is in multiple location due to unclear convvention of the written address. The business_id of zipcode 94602 is 85459 and its business name is ORBIT ROOM. The correct address therefore should be '1900 Market St, San Francisco, CA 94102' with the zipcode 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.

[45]:
```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')
```

[46]:
```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.

[49]:
```
bus = bus[bus['postal_code_5'].isin(all_sf_zip_codes)]
bus.head()
```

[49]:
```
   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
```

[50]:
```
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`.

[51]:
```
num_missing_longs = bus['longitude'].isnull()
len(bus[num_missing_longs])
```

[51]: 2942

[52]:
```
num_missing_longs = bus['longitude'].isnull()
num_missing_longs = len(bus[num_missing_longs])
```

[53]:
```
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`.

[54]:
```
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.

[61]:
```
def num_null(series):
    return (series.isnull().sum())
```

[62]:
```
bus_sf = bus[bus['postal_code'].isin(sf_dense_zip)]
num_missing_in_each_zip = bus_sf.groupby('postal_code_5').agg(num_null)
num_missing_in_each_zip = num_missing_in_each_zip.sort_values('longitude',␣
 ↪ascending=False)
num_missing_in_each_zip['longitude']
```

[62]:
```
postal_code_5
94110    294.0
94103    284.0
94107    275.0
94102    221.0
94109    171.0
94133    159.0
94122    132.0
94111    129.0
94105    127.0
94124    118.0
94118    117.0
94114    111.0
94108     98.0
94115     95.0
94117     86.0
94104     79.0
94112     77.0
94132     71.0
94123     68.0
94121     60.0
94116     42.0
94134     36.0
94127     30.0
94131     16.0
Name: longitude, dtype: float64
```

[63]:
```
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.

[64]:
```python
def num_non_null(series):
    return (len(series) - series.isnull().sum())
```

[65]:
```python
def fract_null(series):
    return (series.isnull().sum()) / len(series)
```

[66]:
```python
count_non_null = bus_sf.groupby('postal_code_5').agg(num_null).
 ↪sort_values('postal_code_5', ascending =False)['longitude'].rename('count␣
 ↪null')
count_null = bus_sf.groupby('postal_code_5').agg(num_non_null).
 ↪sort_values('postal_code_5', ascending =False)['longitude'].rename("count␣
 ↪non null")
frac_null = bus_sf.groupby('postal_code_5').agg(fract_null).
 ↪sort_values('postal_code_5', ascending =False)['longitude'].rename("fraction␣
 ↪null")
```

[67]:

```
count_null
```

[67]:

```
postal_code_5
94134     41.0
94133    267.0
94132     62.0
94131     33.0
94127     41.0
94124     73.0
94123    105.0
94122    141.0
94121    100.0
94118    132.0
94117    118.0
94116     57.0
94115    148.0
94114    112.0
94112    118.0
94111    148.0
94110    302.0
94109    209.0
94108    130.0
94107    185.0
94105    105.0
94104     60.0
94103    268.0
94102    241.0
Name: count non null, dtype: float64
```

[72]:

```
fraction_missing_df = pd.concat([count_null, count_non_null, frac_null], axis =␣
↪1, sort =False) # make sure to use this name for your dataframe
fraction_missing_df.index.name = 'postal_code_5'
fraction_missing_df
```

[72]:

postal_code_5	count non null	count null	fraction null
94134	41.0	36.0	0.467532
94133	267.0	159.0	0.373239
94132	62.0	71.0	0.533835
94131	33.0	16.0	0.326531
94127	41.0	30.0	0.422535
94124	73.0	118.0	0.617801
94123	105.0	68.0	0.393064
94122	141.0	132.0	0.483516
94121	100.0	60.0	0.375000
94118	132.0	117.0	0.469880
94117	118.0	86.0	0.421569

94116	57.0	42.0	0.424242
94115	148.0	95.0	0.390947
94114	112.0	111.0	0.497758
94112	118.0	77.0	0.394872
94111	148.0	129.0	0.465704
94110	302.0	294.0	0.493289
94109	209.0	171.0	0.450000
94108	130.0	98.0	0.429825
94107	185.0	275.0	0.597826
94105	105.0	127.0	0.547414
94104	60.0	79.0	0.568345
94103	268.0	284.0	0.514493
94102	241.0	221.0	0.478355

[73]:
```
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.

[75]:
```
ins.head(5)
```

[75]:

	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.

[76]:
```
# The number of rows in ins
rows_in_table  = len(ins)

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

[79]:
```
unique_ins_ids
```

[79]: 5766

[80]:
```
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?

There are two types of values in ispection type which is 'routine' and 'complaint'. There are total count of 14222, which are 14221 routine inspection and only 1 complaint inspection. From this we can infer that complain inspections are super rare and the inspection type is almost exclusively routine. We can use this information in further analysis in a sense that we will weigh less weight or even ignore complaint inspection because of it is almost too rare. On the other hand, we can look into the data containing complaint inspection to see if that is an anomaly or why only that particular one is different

[81]:
```
ins['type'].describe()
```

[81]:
```
count       14222
unique          2
top       routine
freq        14221
Name: type, dtype: object
```

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.

[82]:
```
ins['new_date'] = pd.to_datetime(ins['date'])
ins.head(5)
```

[82]:
```
   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.

[83]:
```
ins['new_date'] = pd.to_datetime(ins['date'], format='%Y%m%d')
ins.head(5)
```

[83]:
```
   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.

[85]:
```
ins['year'] = ins['new_date'].dt.year
ins.head(5)
```

[85]:

	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 data covered yeras from 2015 to 2018 (2015, 2016, 2017, 2018). The counts for each year is 3305, 5443, 5166, 308 from 2015 to 2018 respectively. Hence we are not looking at roughly the same number of inspections each year because the number of inspection in 2018 is so much lower relative to other yeras. Only the years 2016 and 2017 are roughly the same. Lowest in 2015 and highest in 2016 with 2016 and 2017 showing similar trend

[88]:
```
ins['year'].unique()
```

[88]: `array([2016, 2017, 2015, 2018])`

[89]:
```
ins.groupby('year').count()
```

[89]:

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

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.

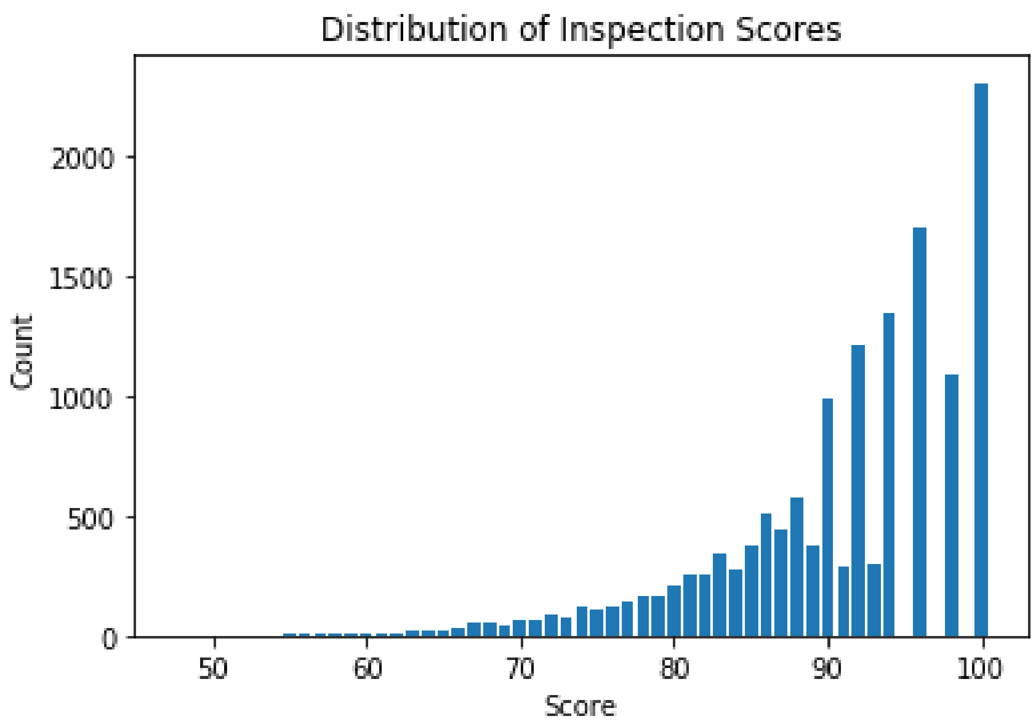
It should look like the image below. It does not need to look exactly the same (e.g., no grid), but make sure that all labels and axes are correct.

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.

[90]:
```
graph_table = ins.groupby('score').count()
graph_table = graph_table.reset_index()
x = graph_table['score']
y = graph_table['business_id']
plt.bar(x, y)
plt.xlabel('Score')
plt.ylabel('Count')
plt.title('Distribution of Inspection Scores')
```

[90]:
```
Text(0.5, 1.0, 'Distribution of Inspection Scores')
```



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 mode of the graph (the one with the highest count) is at the score 100. There is no symmetry but the graph is left-skewed, meaning that there are little count towards the lower scores and the count seem to be accumulated at the higher end (score 90-100). There seems to be a sharp rise in the count starting from 90 all the way upto 100. There are some gaps in between values in range 90 to 100. This applies or infering from this graph, we can say that the restaurants that are inspected tend to have a high score and there relatively not that many restaurants with a low score (every restaurant seems to be doing fairly well). Very few restaurants scored less than 70~75 We can also see that there are no counts under around score 55, which might imply that those restaurants either go out of busninness before inspection, do not allow access to inspection or it may just be showing the basic standards of the restaurants collected in this data (or present in SF).

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.

[93]:
```
ins_named = pd.merge(ins, bus, how='left',on='business_id').
 ↪drop(columns=['city','state','postal_code','latitude','longitude','phone_number','postal_co
ins_named.head()
```

[93]:
```
   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
```

[94]:
```
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 restaurant with the lowest inspection scores ever is 'DA CAFE' with the address '407 CLEMENT ST'.

I lloked at yelp and these are some interesting reviews I found:

"I think the best part about this restaurant was that it was cheap, but its price explains to us something about the service/inspection score"

"Honestly not sure why their reviews are so mediocre. This place is awesome, their food is good and you can't beat the price point. If you haven't liked what you tried, definitely get the salt and pepper chicken wings, salt and pepper spare ribs, and tofu/fish claypot. These are some of their best dishes and all three will land you under $30."

"Cheap and good portions."

"Overall thoughts: *Don't expect good service or anyone to seat you. You pay for the food not the service here."

*Food is just ok but portions are not bad

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!

[95]:
```
ins_named.sort_values(by=['score'])
```

[95]:
```
       business_id  score      date     type    new_date  year  \
13179        86647     48  20160907  routine  2016-09-07  2016
9476         71373     52  20161031  routine  2016-10-31  2016
8885         69199     53  20170127  routine  2017-01-27  2017
7104         61436     54  20150706  routine  2015-07-06  2015
2192          3459     54  20150407  routine  2015-04-07  2015
…              …     …        …        …           …     …
3872          5829    100  20150911  routine  2015-09-11  2015
2413          3796    100  20150304  routine  2015-03-04  2015
11212        79750    100  20170217  routine  2017-02-17  2017
11188        79607    100  20170325  routine  2017-03-25  2017
6202         36396    100  20160912  routine  2016-09-12  2016

                                   name           address
13179                           DA CAFE    407 CLEMENT ST
9476            GOLDEN RIVER RESTAURANT   5827 GEARY BLVD
8885           MEHFIL INDIAN RESTAURANT        28 02ND ST
7104    OZONE THAI RESTAURANT AND LOUNGE       598 02ND ST
2192          BASIL THAI RESTAURANT & BAR    1175 FOLSOM ST
…                                     …               …
3872          LAFAYETTE ELEMENTARY SCHOOL      4545 ANZA ST
2413           JOHNNY FOLEY'S IRISH HOUSE  243 O'FARRELL ST
```

```
11212                 SIMPLY DELISH LLC      5668 03RD ST
11188             TAQUERIA ANGELICA'S #2     OFF THE GRID
6202                WESTERN SUNSET MARKET    4099 JUDAH ST

[14222 rows x 8 columns]
```

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".

[96]:
```
at_least_3 = ins_named.groupby('business_id').count()
at_least_3 = at_least_3[at_least_3.score >= 3]
swing = pd.merge(at_least_3, ins_named, how='left',on='business_id')
swing = swing.drop(columns=['score_x', 'date_x', 'type_x', 'new_date_x',␣
↪'year_x', 'name_x', 'address_x', 'date_y', 'type_y', 'new_date_y', 'year_y',␣
↪'address_y'])
swing.head()
```

[96]:
```
   business_id  score_y                          name_y
0           24       98  OMNI S.F. HOTEL - 2ND FLOOR PANTRY
1           24       98  OMNI S.F. HOTEL - 2ND FLOOR PANTRY
2           24       96  OMNI S.F. HOTEL - 2ND FLOOR PANTRY
3           45       78                CHARLIE'S DELI CAFE
4           45       88                CHARLIE'S DELI CAFE
```

[97]:
```
def cal_swing(series):
    return max(series) - min(series)
```

[98]:
```
max_swing = swing.groupby('business_id').agg(cal_swing)
max_swing.sort_values(by=['score_y'], ascending=False)
```

[98]:
```
             score_y
business_id
2044              39
77532             38
70983             37
```

```
81460              37
83476              36
…                …
63049               0
4618                0
62939               0
62797               0
68013               0

[2571 rows x 1 columns]
```

[99]:
```
max_swing = bus[bus['business_id'] == 2044].iloc[0]['name']
max_swing
```

[99]:
```
"JOANIE'S DINER INC."
```

[101]:
```
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`.

[102]:
```
count_table = ins.groupby(['business_id', 'year']).count()
count_table.head()
```

[102]:

business_id	year	score	date	type	new_date
19	2016	1	1	1	1
	2017	1	1	1	1

```
24          2016     2     2     2          2
            2017     1     1     1          1
31          2015     1     1     1          1
```

[103]:
```
count_table = count_table.drop(columns = ['date', 'type', 'new_date']).
↪rename(columns={'score' : 'count'})
```

[104]:
```
inspections_by_id_and_year = ins.groupby(['business_id', 'year']).count().
↪drop(columns = ['date', 'type', 'new_date']).rename(columns={'score' :␣
↪'count'})
inspections_by_id_and_year.head()
```

[104]:
```
                  count
business_id year
19          2016      1
            2017      1
24          2016      2
            2017      1
31          2015      1
```

[106]:
```
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.

[107]:
```
inspections_by_id_and_year['count'].value_counts()
```

[107]:
```
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.

[108]:
```
def pair(series):
    return series.tolist()
```

[109]:
```
ins2016 = ins[ins['year'] == 2016]
```

[110]:
```

```

```
pair = ins2016.sort_values('year').groupby('business_id').filter(lambda x:␣
↪x['score'].count() == 2).groupby('business_id').agg({'score': lambda x:␣
↪list(x)}).rename(columns={'score' : 'score_pair'})
```

[113]:
```
# Create the dataframe here
ins2016 = ins[ins['year'] == 2016]
scores_pairs_by_business = ins2016.sort_values('year').groupby('business_id').
↪filter(lambda x: x['score'].count() == 2).groupby('business_id').
↪agg({'score': lambda x: list(x)}).rename(columns={'score' : 'score_pair'})
scores_pairs_by_business
```

[113]:
```
             score_pair
business_id
24             [96, 98]
45             [84, 78]
66            [100, 98]
67             [94, 87]
76            [98, 100]
…                   …
87761          [86, 92]
87802          [91, 98]
88323          [75, 75]
88756          [80, 88]
88792         [100, 96]

[1076 rows x 1 columns]
```

[114]:
```
ok.grade("q7c1");
```

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

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

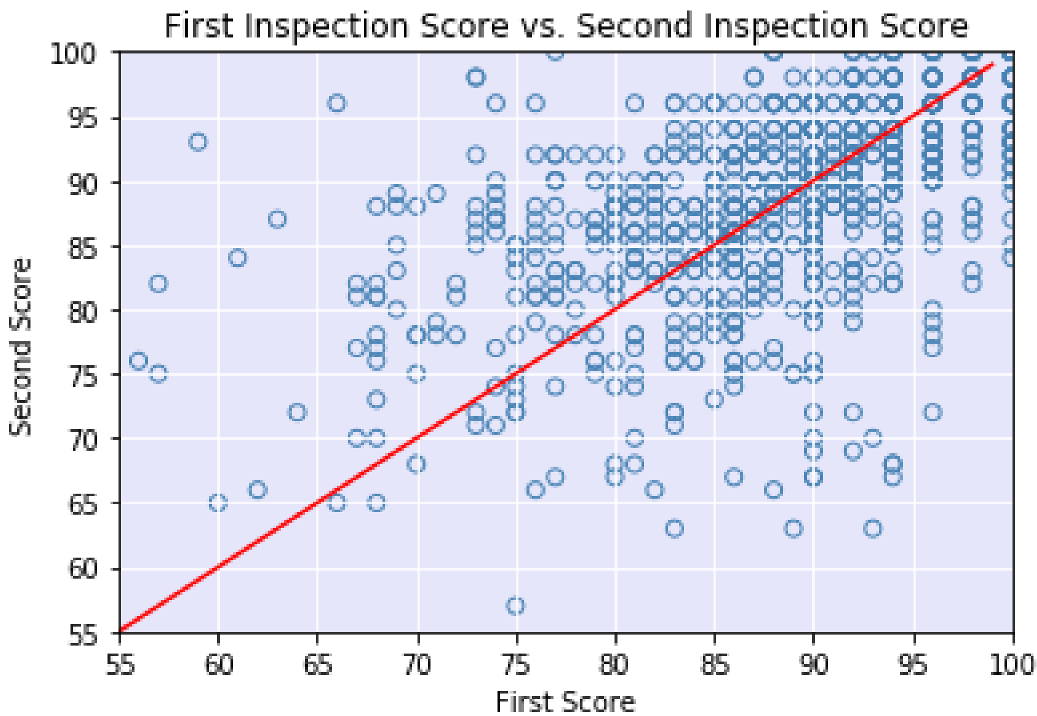
Now, create your scatter plot in the cell below. It does not need to look exactly the same (e.g., no grid) as the above sample, but make sure that all labels, axes and data itself are correct.

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.

[115]:
```
list_of_tuples = scores_pairs_by_business['score_pair']
list_of_tuples = list(zip(*list_of_tuples))
fig = plt.scatter(x = list_of_tuples[0], y = list_of_tuples[1],␣
 ↪facecolors='none', edgecolor = ['steelblue'])
plt.xlabel('First Score')
plt.ylabel('Second Score')
plt.title('First Inspection Score vs. Second Inspection Score')
plt.grid(True, color = 'w')
plt.xlim((55, 100))
plt.ylim((55, 100))
plt.plot(np.arange(100), np.arange(100), color = 'r')
ax = plt.gca()
ax.patch.set_facecolor('lavender')
```



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.

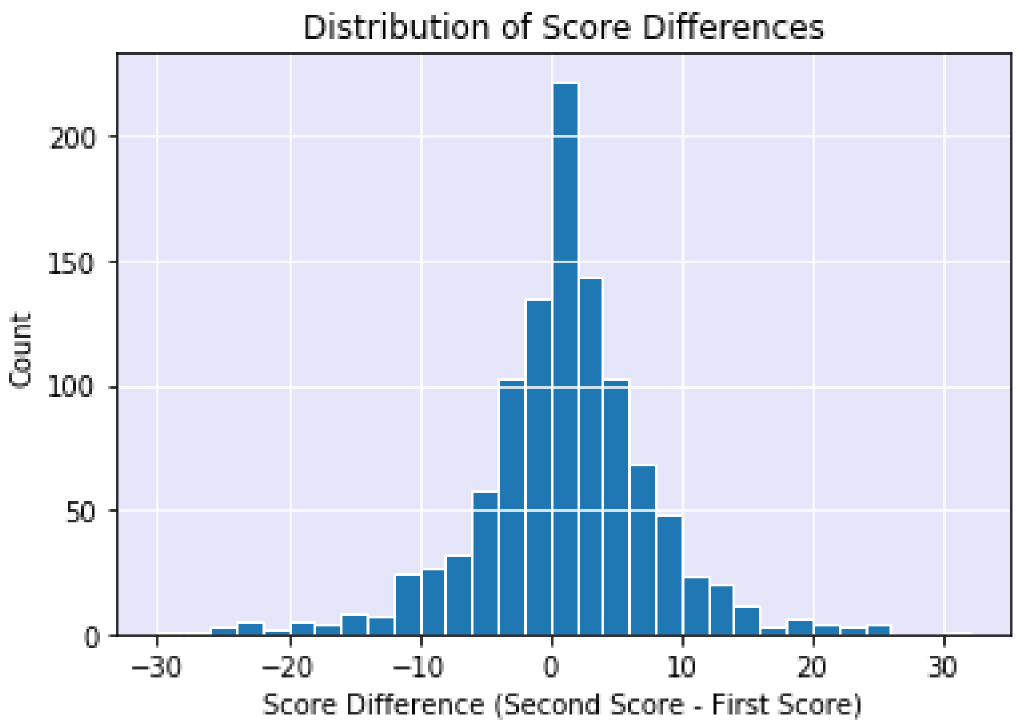
The histogram should look like this:

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()`.

[116]:
```
first_score = np.asarray(list_of_tuples[0])
second_score = np.asarray(list_of_tuples[1])
score_diff = second_score - first_score
plt.hist(score_diff, bins=range(min(score_diff), max(score_diff), 2), histtype␣
 ↪= 'bar', ec='white')
plt.grid(True, color = 'w')
plt.xlabel('Score Difference (Second Score - First Score)')
plt.ylabel('Count')
plt.title('Distribution of Score Differences')
ax = plt.gca()
ax.patch.set_facecolor('lavender')
```



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?

In the scatterplot, I expect most, if not all, of the dots to be above the indicator line (y = x, red linear line) if there is a clear trend of improvements from the first to the second inspection. However, on our scatterplot, there seem to be an equal amount of dots above and below the indicator line and many seem to be clustered around the line. This shows that generally, there has not been clear trend between the first and the second inspection score. In the histogram, if it is the general trend that restuarant score improves from the first to the second inspection, we expect to see most of the data to be positive (most data concentrated from x-axis value 0 upwards). We expect someone of a left-skewed graph or bars much higher/concentrated on the right side (postive x-axis). However, we see that the data seem to be centered/ its mode is at x=0 and the bars are almost normally distributed. This shows that mostly there were no difference between the first and the second inspection with some restaurants who did better/worse.

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!**

[ ]:
```
# 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
```

[ ]:

[ ]: