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

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

Project 1: Food Safety

Cleaning and Exploring Data with Pandas

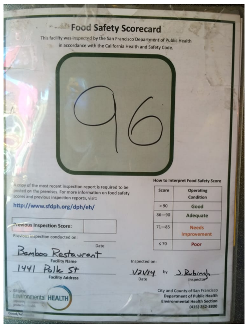
Due Date: Tuesday 09/24, 11:59 PM

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*

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

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 (http://ipython.readthedocs.io/en/stable/interactive/magics.html#magic-matplotlib)](http://ipython.readthedocs.io/en/stable/interactive/magics.html#magic-matplotlib).
3. If you want to use `seaborn` , add the line `sns.set()` to make your plots look nicer.

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

```
In [3]: import sys

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

Downloading the Data

For this assignment, we need this data file: http://www.ds100.org/fa19/assets/datasets/proj1-SFBusinesses.zip (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 (https://docs.python.org/3/library/pathlib.html#basic-use)).

```
In [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): Thu Sep 26 23:47:40 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.

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

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

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: 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.

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 (https://docs.python.org/3/library/zipfile.html) 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.

```
In [6]: my_zip = zipfile.ZipFile(dest_path, 'r') # SOLUTION
        list_names = [f.filename for f in my_zip.filelist] # SOLUTION
        list_names
```

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

```
In [7]: 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.

```
In [8]: # BEGIN SOLUTION
        my_zip = zipfile.ZipFile(dest_path, 'r')
        for file in my_zip.filelist:
            print('{}\t{}'.format(file.filename, file.file_size))
        # END SOLUTION
```

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

Often when working with zipped data, we'll never unzip the actual zipfile. This saves space on our local computer. However, for this 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.

```
In [9]: 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"
```

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.

```
In [10]: # BEGIN SOLUTION
         data_dir = "./data/"
         for f in list_names:
             print(ds100_utils.head(data_dir + f, 5), "\n")
         # END SOLUTION
```

```
['"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","longitude","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 PANTR
Y","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 ","S
an 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']
```

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 (https://www.diveinto.org/python3/strings.html). We won't discuss these in detail in DS100.

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

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

Now that you've read in the files, let's try some `pd.DataFrame` methods (docs (https://pandas.pydata.org/pandas-docs/version/0.21/generated/pandas.DataFrame.html)). 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.

```
In [12]: bus.head() # SOLUTION
```

Out[12]:

	business_id	name	address	city	state	postal_code	latitude	longitude	phone_number
0	19	NRGIZE LIFESTYLE CAFE	1200 VAN NESS AVE, 3RD FLOOR	San Francisco	CA	94109	37.786848	-122.421547	+14157763262
1	24	OMNI S.F. HOTEL - 2ND FLOOR PANTRY	500 CALIFORNIA ST, 2ND FLOOR	San Francisco	CA	94104	37.792888	-122.403135	+14156779494
2	31	NORMAN'S ICE CREAM AND FREEZES	2801 LEAVENWORTH ST	San Francisco	CA	94133	37.807155	-122.419004	NaN
3	45	CHARLIE'S DELI CAFE	3202 FOLSOM ST	San Francisco	CA	94110	37.747114	-122.413641	+14156415051
4	48	ART'S CAFE	747 IRVING ST	San Francisco	CA	94122	37.764013	-122.465749	+14156657440

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

```
In [13]: bus.describe() # SOLUTION
```

Out[13]:

	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

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

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

```
In [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.

```
In [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.

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.

```
In [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)
```

```
In [17]: ok.grade("q1d");
```

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

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

Question 1e: Identifying Issues with the Data

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

SOLUTION:
There appears to be a missing phone number for NORMAN'S ICE CREAM 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.

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.

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.

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

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

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

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

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.**

SOLUTION:
Each row has a unique `business_id` that serves as a primary key. If we then groupby name we see that there are many rows/records with the same name at different locations indicating that each record represents an individual restaurant, not a business. Grouping by `business_id` finds nothing new. Grouping by `name` finds all locations of the same restaurant (plus perhaps some spurious matches). Grouping by `address` finds all stores that share a location.

```
In [20]: # use this cell for scratch work
         # BEGIN SOLUTION NO PROMPT
         print("Number of records:", len(bus))
         print("Most frequently occuring business names:", list(bus['name'].value_counts().sort_values(ascending=False).index[:3]))
         print("A few samples of the business with most frequent name ----------")
         bus[bus['name'] == bus['name'].value_counts().idxmax()].head(7)
         # END SOLUTION

Number of records: 6406
Most frequently occuring business names: ['STARBUCKS COFFEE', "PEET'S COFFEE & TEA", 'MCDONALDS']
A few samples of the business with most frequent name ----------
```

Out[20]:

	business_id	name	address	city	state	postal_code	latitude	longitude	phone_number
9	66	STARBUCKS COFFEE	1800 IRVING ST	San Francisco	CA	94122	37.763578	-122.477461	+14152427970
236	1085	STARBUCKS COFFEE	333 MARKET ST	San Francisco	CA	94105	37.792037	-122.397852	+14152410256
238	1103	STARBUCKS COFFEE	4094 18TH ST	San Francisco	CA	94114	37.760938	-122.434692	+14152410256
240	1116	STARBUCKS COFFEE	1899 UNION ST	San Francisco	CA	94123	37.797713	-122.430336	+14159214049
241	1122	STARBUCKS COFFEE	2132 CHESTNUT ST	San Francisco	CA	94123	37.800547	-122.438494	+14156736781
244	1127	STARBUCKS COFFEE	555 CALIFORNIA ST	San Francisco	CA	94104	37.792773	-122.403567	+14159551940
272	1265	STARBUCKS COFFEE	744 IRVING ST	San Francisco	CA	94122	37.764088	-122.465981	+14158540888

3: Zip Codes

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

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?
2. What data type is used to represent a ZIP code?

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

SOLUTION:
The ZIP codes are largely nominal fields with little meaning to differences or ratios.
While in some regions of the country similar numbers correspond to similar locations, this relationship is not reliable.

The ZIP codes are currently stored as strings.

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()`.

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

Out[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?

```
In [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.

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

Out[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.

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

Out[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.

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

Out[25]:

	business_id	name	address	city	state	postal_code	latitude	longitude	phone_number	postal_code_5
0	19	NRGIZE LIFESTYLE CAFE	1200 VAN NESS AVE, 3RD FLOOR	San Francisco	CA	94109	37.786848	-122.421547	+14157763262	94109
1	24	OMNI S.F. HOTEL - 2ND FLOOR PANTRY	500 CALIFORNIA ST, 2ND FLOOR	San Francisco	CA	94104	37.792888	-122.403135	+14156779494	94104
2	31	NORMAN'S ICE CREAM AND FREEZES	2801 LEAVENWORTH ST	San Francisco	CA	94133	37.807155	-122.419004	NaN	94133
3	45	CHARLIE'S DELI CAFE	3202 FOLSOM ST	San Francisco	CA	94110	37.747114	-122.413641	+14156415051	94110
4	48	ART'S CAFE	747 IRVING ST	San Francisco	CA	94122	37.764013	-122.465749	+14156657440	94122

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.

SOLUTION:
Many of the restuarants without ZIP codes are food trucks (e.g., OFF THE GRID) or catering services. Therefore, a missing ZIP code might actually make sense and dropping these from the analysis could bias our conclusions.

```
In [26]: # You can use this cell as scratch to explore the data
         # BEGIN SOLUTION NO PROMPT
         bus[bus['postal_code'].isnull()]['address'].value_counts().head(3)
         # END SOLUTION

Out[26]:  OFF THE GRID                    69
          APPROVED PRIVATE LOCATIONS       6
          APPROVED LOCATIONS               4
         Name: address, dtype: int64
```

Question 3d: Incorrect ZIP Codes

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

```
In [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.

```
In [28]: weird_zip_code_businesses = bus[~bus['postal_code_5'].isin(all_sf_zip_codes) & ~bus['postal_code_5'].isnull()] # SOLUTION
         weird_zip_code_businesses
```

Out[28]:

	business_id	name	address	city	state	postal_code	latitude	longitude	phone_number	postal_code_5
1211	5208	GOLDEN GATE YACHT CLUB	1 YACHT RD	San Francisco	CA	941	37.807878	-122.442499	+14153462628	941
1372	5755	J & J VENDING	VARIOUS LOACATIONS (17)	San Francisco	CA	94545	NaN	NaN	+14156750910	94545
1373	5757	RICO VENDING, INC	VARIOUS LOCATIONS	San Francisco	CA	94066	NaN	NaN	+14155836723	94066
2258	36547	EPIC ROASTHOUSE	PIER 26 EMBARARCADERO	San Francisco	CA	95105	37.788962	-122.387941	+14153699955	95105
2293	37167	INTERCONTINENTAL SAN FRANCISCO EMPLOYEE CAFETERIA	888 HOWARD ST 2ND FLOOR	San Francisco	CA	94013	37.781664	-122.404778	+14156166532	94013
2295	37169	INTERCONTINENTAL SAN FRANCISCO 4TH FL. KITCHEN	888 HOWARD ST 4TH FLOOR	San Francisco	CA	94013	37.781664	-122.404778	+14156166532	94013
2846	64540	LEO'S HOT DOGS	2301 MISSION ST	San Francisco	CA	CA	37.760054	-122.419166	+14152406434	CA
2852	64660	HAIGHT STREET MARKET	1530 HAIGHT ST	San Francisco	CA	92672	37.769957	-122.447533	+14152550643	92672
2857	64738	JAPACURRY	PUBLIC	San Francisco	CA	CA	37.777122	-122.419639	+14152444785	CA
2969	65856	BAMBOO ASIA	41 MONTGOMERY ST	San Francisco	CA	94101	37.774998	-122.418299	+14156246790	94101
3142	67875	THE CHAIRMAN TRUCK	OFF THE GRID	San Francisco	CA	00000	37.777122	-122.419639	+14158461711	00000

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.

SOLUTION:
94545 - Hayward, look at record and see it's vending machine company with many locations
94602 - Oakland, look at the record and see it's probably a typo and should be 94102

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.

```
In [29]: # 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'] = ...
         # BEGIN SOLUTION NO PROMPT
         bus['postal_code_5'] = bus['postal_code_5'].str.replace("94602", "94102")
         # END SOLUTION
```

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

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

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

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.

```
In [31]: bus = bus[bus['postal_code_5'].isin(all_sf_zip_codes) & bus['postal_code_5'].notnull()] # SOLUTION
         bus.head()
```

Out[31]:

	business_id	name	address	city	state	postal_code	latitude	longitude	phone_number	postal_code_5
0	19	NRGIZE LIFESTYLE CAFE	1200 VAN NESS AVE, 3RD FLOOR	San Francisco	CA	94109	37.786848	-122.421547	+14157763262	94109
1	24	OMNI S.F. HOTEL - 2ND FLOOR PANTRY	500 CALIFORNIA ST, 2ND FLOOR	San Francisco	CA	94104	37.792888	-122.403135	+14156779494	94104
2	31	NORMAN'S ICE CREAM AND FREEZES	2801 LEAVENWORTH ST	San Francisco	CA	94133	37.807155	-122.419004	NaN	94133
3	45	CHARLIE'S DELI CAFE	3202 FOLSOM ST	San Francisco	CA	94110	37.747114	-122.413641	+14156415051	94110
4	48	ART'S CAFE	747 IRVING ST	San Francisco	CA	94122	37.764013	-122.465749	+14156657440	94122

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

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

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

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.

Question 4a

How many businesses are missing longitude values?

Hint: Use `isnull`.

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

```
Out[33]: 2942
```

```
In [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`.

```
In [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.

```
In [36]: num_missing_in_each_zip = ...
         # BEGIN SOLUTION NO PROMPT
         def count_null(s):
             return len(s[s.isnull()])

         bus_sf = bus[bus['postal_code_5'].isin(sf_dense_zip)]
         num_missing_in_each_zip = bus_sf['longitude'].groupby(bus_sf["postal_code_5"]).agg(count_null).sort_values(ascending = False)
         # END SOLUTION
         num_missing_in_each_zip.head()
```

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

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

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

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

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.

In [38]:
```
fraction_missing_df = ... # make sure to use this name for your dataframe
# BEGIN SOLUTION NO PROMPT

def count_null(s):
    return len(s[s.isnull()])
def count_non_null(s):
    return len(s[~s.isnull()])
def fraction_null(s):
    n = len(s[s.isnull()])
    nn = len(s[~s.isnull()])
    return (n/(n+nn))
bus_sf = bus[bus['postal_code_5'].isin(sf_dense_zip)]
fraction_missing_df = bus_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)
# END SOLUTION
fraction_missing_df.head()
```

Out[38]:

	count non null	count null	fraction null
postal_code_5			
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

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

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

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

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.

5: Investigate the Inspection Data

Let's now turn to the inspection DataFrame. Earlier, we found that `ins` has 4 columns named `business_id` , `score` , `date` and `type` . In this section, we determine the granularity of `ins` and investigate the kinds of information provided for the inspections.

Let's start by looking again at the first 5 rows of `ins` to see what we're working with.

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

Out[40]:

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

Question 5a

From calling `head` , we know that each row in this table corresponds to 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.

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

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

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

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

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

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?

SOLUTION:
All the records have the same value, "routine", except for one. This variable will not be useful in any analysis because it provides no information.

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.

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

Out[43]:

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

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

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

Out[44]:

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

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

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

Out[45]:

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

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

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

SOLUTION:
No, 2018 only has a few. Also 2015 has substantially fewer inspections than 2016 or 2017.

6: Explore Inspection Scores

Question 6a

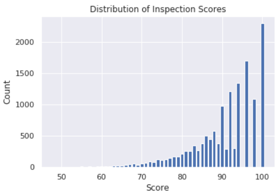
Let's look at the distribution of inspection scores. As we saw before when we called `head` on this data frame, inspection scores appear to be integer values. The discreteness of this variable means that we can use a barplot to visualize the distribution of the inspection score. Make a bar plot of the 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 (http://data100.datahub.berkeley.edu/hub/user-redirect/git-sync?repo=https://github.com/DS-100/fa19&subPath=extra/pyplot.ipynb) useful. Key syntax that you'll need:

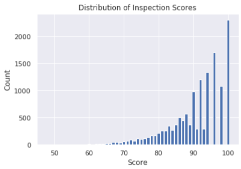
- `plt.bar`
- `plt.xlabel`
- `plt.ylabel`
- `plt.title`

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.



In [46]:
```
# BEGIN SOLUTION
score_counts = ins['score'].value_counts()
plt.bar(score_counts.keys(), score_counts)
plt.xlabel("Score")
plt.ylabel("Count")
plt.title("Distribution of Inspection Scores")
# END SOLUTION
```

Out[46]:
```
Text(0.5, 1.0, 'Distribution of Inspection Scores')
```



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?

SOLUTION:
The distribution is unimodal with a peak at 100. It is skewed left (as expected with a variable bounded on the right). The distribution has a long left tail with some restaurants receiving scores that are in the 50s, 60s, and 70s. One unusual feature of the distribution is the bumpiness with even numbers having higher counts than odd. This may be because the violations result in penalties of 2, 4, 10, etc. points.

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 (https://pandas.pydata.org/pandas-docs/stable/user_guide/merging.html) 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.

In [47]:
```
ins_named = ins.merge(bus[["business_id", "name", "address"]], how="left", left_on = "business_id", right_on = "business_id") # SOLUTION
ins_named.head()
```

Out[47]:

	business_id	score	date	type	new_date	year	name	address
0	19	94	20160513	routine	2016-05-13	2016	NRGIZE LIFESTYLE CAFE	1200 VAN NESS AVE, 3RD FLOOR
1	19	94	20171211	routine	2017-12-11	2017	NRGIZE LIFESTYLE CAFE	1200 VAN NESS AVE, 3RD FLOOR
2	24	98	20171101	routine	2017-11-01	2017	OMNI S.F. HOTEL - 2ND FLOOR PANTRY	500 CALIFORNIA ST, 2ND FLOOR
3	24	98	20161005	routine	2016-10-05	2016	OMNI S.F. HOTEL - 2ND FLOOR PANTRY	500 CALIFORNIA ST, 2ND FLOOR
4	24	96	20160311	routine	2016-03-11	2016	OMNI S.F. HOTEL - 2ND FLOOR PANTRY	500 CALIFORNIA ST, 2ND FLOOR

In [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.

SOLUTION:

The restaurant with the worst score is D&A cafe. One review I found amusing was:

This place is awesome.

I don't care that they've been shut down for health violations multiple times.

This place is always packed with regulars. I equate the cleanliness like if you were eating in Asia. I've never had an issue.

The food is good and cheap. I come for the happy hour after 10pm, and take it togo. Staff is usually pretty friendly.

Deep fried pig intestines are on point and only $4.25.

Watermelon juice is insanely good and just over 2 bucks.

Salt and pepper wings are crispy and seasoned well.

I just got 3 dishes and a watermelon juice for $15. Hell yes.

If you want cheap Chinese food, this is the place.

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

7: Restaurant Ratings Over Time

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

Question 7a

Let's see which restaurant has had the most extreme 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".

In [49]:
```
# BEGIN SOLUTION NO PROMPT
def swing(s):
    if len(s) < 3:
        return 0
    return max(s) - min(s)

swing_series = ins_named['score'].groupby(ins_named['business_id']).agg(swing).rename('swing')
bus_swing = pd.concat([bus.set_index('business_id'), swing_series], axis=1).sort_values("swing", ascending=False)
bus_swing
# END SOLUTION
max_swing = bus_swing.iloc[0]['name'] # SOLUTION
max_swing
```

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

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

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

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

Question 7b

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

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

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

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

In [51]:
```
inspections_by_id_and_year = ins.groupby([ins['business_id'], ins['year']]).size().rename("count").to_frame() # SOLUTION
inspections_by_id_and_year.head()
```

Out[51]:

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

In [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.

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

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

Question 7c

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

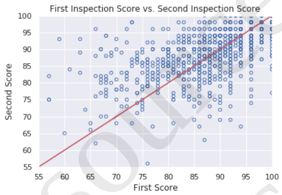
Plot these scores. That is, make a scatter plot to display these pairs of scores. Include on the plot a reference line with slope 1.

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.

In [54]:
```
# Create the dataframe here
scores_pairs_by_business = ...
ins2016 = ins[ins['year'] == 2016]
# BEGIN SOLUTION NO PROMPT
# SOLUTION 1
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(list)
                            .rename(columns={'score':'score_pair'}))

# SOLUTION 2
scores_pairs_by_business = (ins2016.sort_values('date')
                            .groupby('business_id')
                            .filter(lambda group: len(group)==2)
                            .groupby('business_id')
                            .agg({'score': lambda group: group.tolist()})
                            .rename(columns={'score':'score_pair'}))
scores_pairs_by_business.head()
# END SOLUTION
```

Out[54]:

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

In [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.

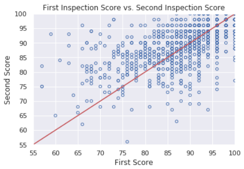
Key pieces of syntax you'll need:

- `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.

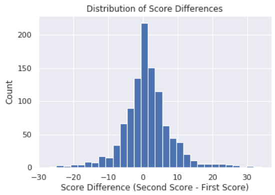
In [56]:
```
# BEGIN SOLUTION
first_score, second_score = zip(*scores_pairs_by_business['score_pair'])
plt.scatter(first_score,second_score,s=20,facecolors='none',edgecolors='b')
plt.plot([55,100],[55,100],'r-')
plt.xlabel('First Score')
plt.ylabel('Second Score')
plt.axis([55,100,55,100])
plt.title("First Inspection Score vs. Second Inspection Score");
# END SOLUTION
```



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:

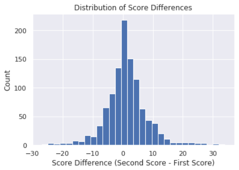


Hint: Use `second_score` and `first_score` created in the scatter plot code above.

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()` .

In [57]:
```
# BEGIN SOLUTION
diffs = np.array(second_score) - np.array(first_score)
plt.hist(diffs, bins=30)
plt.title("Distribution of Score Differences")
plt.xlabel("Score Difference (Second Score - First Score)")
plt.ylabel("Count");
# END SOLUTION
```



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?

SOLUTION:
If the restaurants tend to improve from the first to the second inspection, we would expect to see the points in the scatter plot fall above the line of slope 1. We would also expect to see the histogram of the difference in scores to be shifted toward positive values. Interestingly, we don't see this. The second inspection often is worse than first. The histogram of differences shows a unimodal distribution centered at 0, hinting that the average restaurant does not see a change in score between their first and second inspection. This distribution has long tails with some scores being as low as -20 and others as high as 20-30.

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.

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.

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!**

In [58]:
```
# 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

Saving notebook... Could not save your notebook. Make sure your notebook is saved before sending it to OK!
Submit... 0.0% complete
Could not submit: Late Submission of cal/data100/fa19/proj1
```