Homework 2: Food Safety (50 Pts)

Cleaning and Exploring Data with Pandas

This Assignment

In this homework, we will investigate restaurant food safety scores for restaurants in San Francisco. The scores and violation information have been [made available by the San Francisco Department of Public Health (https://data.sfgov.org/Health-and-Social-Services/Restaurant-Scores-LIVES-Standard/pyih-qa8i)](). The main goal for this assignment is to walk through the process of Data Cleaning and EDA.

As we clean and explore these data, you will gain practice with:

- Reading simple csv files and using Pandas
- 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

In [1]:

```
import numpy as np
import pandas as pd

import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
plt.style.use('fivethirtyeight')

import zipfile
from pathlib import Path
import os # Used to interact with the file system
```

Importing and Verifying Data

There are several tables in the data folder. Let's attempt to load `bus.csv`, `ins2vio.csv`, `ins.csv`, and `vio.csv` into pandas dataframes with the following names: `bus`, `ins2vio`, `ins`, and `vio` respectively.

Note: Because of character encoding issues one of the files (`bus`) will require an additional argument `encoding='ISO-8859-1'` when calling `pd.read_csv`.

In [2]:

```
# path to directory containing data
dsDir = Path('data')

bus = pd.read_csv(dsDir/'bus.csv', encoding='ISO-8859-1')
ins2vio = pd.read_csv(dsDir/'ins2vio.csv')
ins = pd.read_csv(dsDir/'ins.csv')
vio = pd.read_csv(dsDir/'vio.csv')
```

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()`. Currently, running the cell below will display the first few lines of the `bus` dataframe.

In [3]:

```
bus.head()
```

Out[3]:

	business id column	name	address	city	state	postal_code	latitude	longitude
0	1000	HEUNG YUEN RESTAURANT	3279 22nd St	San Francisco	CA	94110	37.755282	-122.420493
1	100010	ILLY CAFFE SF_PIER 39	PIER 39 K-106-B	San Francisco	CA	94133	-9999.000000	-9999.000000
2	100017	AMICI'S EAST COAST PIZZERIA	475 06th St	San Francisco	CA	94103	-9999.000000	-9999.000000
3	100026	LOCAL CATERING	1566 CARROLL AVE	San Francisco	CA	94124	-9999.000000	-9999.000000
4	100030	OUI OUI! MACARON	2200 JERROLD AVE STE C	San Francisco	CA	94124	-9999.000000	-9999.000000

The `DataFrame.describe` method can also be handy for computing summaries of numeric columns of our dataframes. Try it out with each of our 4 dataframes. Below, we have used the method to give a summary of the `bus` dataframe.

In [4]:

```
bus.describe()
```

Out[4]:

	business id column	latitude	longitude	phone_number
count	6253.000000	6253.000000	6253.000000	6.253000e+03
mean	60448.948984	-5575.337966	-5645.817699	4.701819e+09
std	36480.132445	4983.390142	4903.993683	6.667508e+09
min	19.000000	-9999.000000	-9999.000000	-9.999000e+03
25%	18399.000000	-9999.000000	-9999.000000	-9.999000e+03
50%	75685.000000	-9999.000000	-9999.000000	-9.999000e+03
75%	90886.000000	37.776494	-122.421553	1.415533e+10
max	102705.000000	37.824494	0.000000	1.415988e+10

Now, we perform some sanity checks for you to verify that the data was loaded with the correct 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 [5]:

```
assert all(bus.columns == ['business id column', 'name', 'address', 'city', 'state
                           'latitude', 'longitude', 'phone_number'])
assert 6250 <= len(bus) <= 6260

assert all(ins.columns == ['iid', 'date', 'score', 'type'])
assert 26660 <= len(ins) <= 26670

assert all(vio.columns == ['description', 'risk_category', 'vid'])
assert 60 <= len(vio) <= 65

assert all(ins2vio.columns == ['iid', 'vid'])
assert 40210 <= len(ins2vio) <= 40220
```

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

In [6]:

```
bus_summary = pd.DataFrame(**{'columns': ['business id column', 'latitude', 'longi
 'data': {'business id column': {'50%': 75685.0, 'max': 102705.0, 'min': 19.0},
  'latitude': {'50%': -9999.0, 'max': 37.824494, 'min': -9999.0},
  'longitude': {'50%': -9999.0,
   'max': 0.0,
   'min': -9999.0}},
 'index': ['min', '50%', 'max']})

ins_summary = pd.DataFrame(**{'columns': ['score'],
 'data': {'score': {'50%': 76.0, 'max': 100.0, 'min': -1.0}},
 'index': ['min', '50%', 'max']})

vio_summary = pd.DataFrame(**{'columns': ['vid'],
 'data': {'vid': {'50%': 103135.0, 'max': 103177.0, 'min': 103102.0}},
 '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 column	latitude	longitude
min	19.0	-9999.000000	-9999.0
50%	75685.0	-9999.000000	-9999.0
max	102705.0	37.824494	0.0

What we expect from your Inspections dataframe:

	score
min	-1.0
50%	76.0
max	100.0

What we expect from your Violations dataframe:

	vid
min	103102.0
50%	103135.0
max	103177.0

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.

In [7]:

```
"""Run this cell to load this utility comparison function that we will use in vari
tests below

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


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

    Compute the min, median and max of the two dataframes on the given columns, an
    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 ca
    # `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)
```

Question 1a: 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.

Type your answer here, replacing this text.

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.

Question 1b: Examining the Business Data File

From its name alone, we expect the `bus.csv` file to contain information about the restaurants. Let's investigate the granularity of this dataset.

In [8]:

```
bus.head()
```

Out[8]:

	business id column	name	address	city	state	postal_code	latitude	longitude
0	1000	HEUNG YUEN RESTAURANT	3279 22nd St	San Francisco	CA	94110	37.755282	-122.420493
1	100010	ILLY CAFFE SF_PIER 39	PIER 39 K-106-B	San Francisco	CA	94133	-9999.000000	-9999.000000
2	100017	AMICI'S EAST COAST PIZZERIA	475 06th St	San Francisco	CA	94103	-9999.000000	-9999.000000
3	100026	LOCAL CATERING	1566 CARROLL AVE	San Francisco	CA	94124	-9999.000000	-9999.000000
4	100030	OUI OUI! MACARON	2200 JERROLD AVE STE C	San Francisco	CA	94124	-9999.000000	-9999.000000

The `bus` dataframe contains a column called `business id column` which probably corresponds to a unique business id. However, we will first rename that column to `bid` for simplicity.

In [9]:

```
bus = bus.rename(columns={"business id column": "bid"})
```

Examining the entries in `bus`, is the `bid` unique for each record (i.e. 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 `bid` series has any duplicates.

In [10]:

```
is_bid_unique = len(bus['bid'].unique()) == len(bus['bid'])
is_bid_unique
```

Out[10]:

```
True
```

Question 1c

We will now work with some important fields in `bus`. In the two cells below create the following **two numpy arrays**:

1. Assign `top_names` to the top 5 most frequently used business names, from most frequent to least frequent.
2. Assign `top_addresses` to the top 5 addressses where businesses are located, from most popular to least popular.

Hint: you may find `value_counts()` helpful.

Step 1

In [11]:

```
top_names = bus['name'].value_counts().head(5)
top_addresses = bus['address'].value_counts().head(5)
top_names, top_addresses
```

Out[11]:

```
(Peet's Coffee & Tea    20
 Starbucks Coffee       13
 Jamba Juice            10
 McDonald's             10
 STARBUCKS               9
 Name: name, dtype: int64,
 Off The Grid      39
 428 11th St       34
 3251 20th Ave     17
 2948 Folsom St    17
 Pier 41           16
 Name: address, dtype: int64)
```

Question 1d

Based on the above exploration, answer each of the following questions about `bus` by assigning your answers to the corresponding variables

1. What does each record represent?
2. What is the minimal primary key?

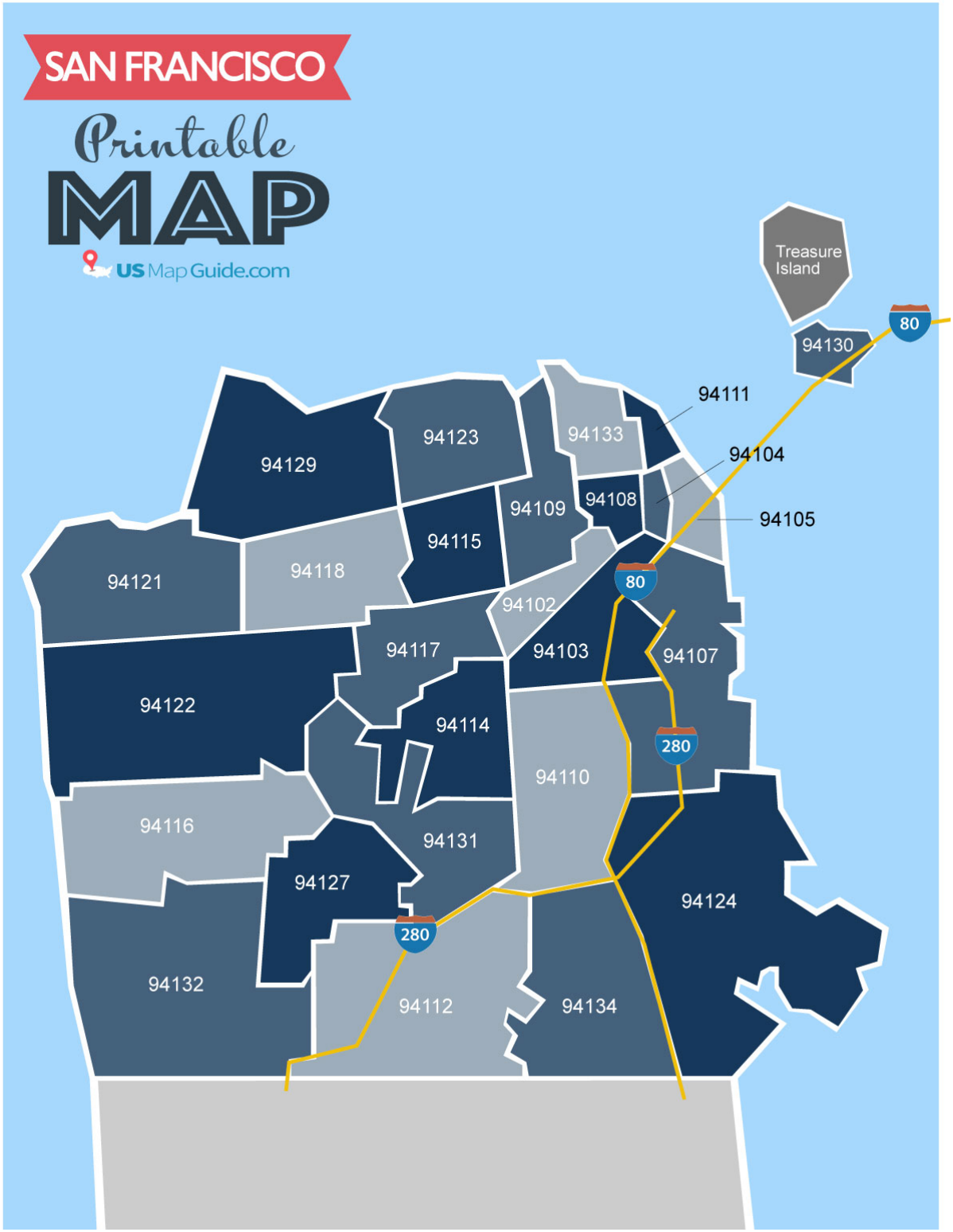
In [12]:

```
# What does each record represent?  Valid answers are:
#     "One location of a restaurant."
#     "A chain of restaurants."
#     "A city block."
q1d_part1 = "One location of a restaurant"

# What is the minimal primary key? Valid answers are:
#     "bid"
#     "bid, name"
#     "bid, name, address"
q1d_part2 = "bid"
```

2: Cleaning the Business Data Postal Codes

The business data contains postal code information that we can use to aggregate the ratings over regions of the city. Let's examine and clean the postal code field. The postal code (sometimes also called a ZIP code) partitions the city into regions:



Question 2a

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. You may need to use `groupby()`, `size()`, or `value_counts()`. Do you notice any odd/invalid zip codes?

In [13]:

```
zip_counts = bus.groupby('postal_code').size()
print(zip_counts.to_string())

# The zip codes which are -9999 and those with 9 digits are odd or invalid.
```

```
postal_code
-9999          194
00000            1
64110            1
92672            1
94013            2
94014            1
94080            1
941              1
94101            2
94102          456
94102-5917       1
94103          562
941033148        1
94104          142
94105          249
94105-1420       1
94105-2907       1
94107          408
94108          229
94109          382
94110          555
941102019        1
94111          259
94112          192
94114          200
94115          230
94116           97
94117          189
94117-3504       1
94118          231
94120            1
94121          157
94122          255
94122-1909       1
94123          177
94123-3106       1
94124          218
94124-1917       1
94127           67
94129            1
94130            8
94131           49
94132          132
94133          398
94134           82
94143            5
94158           90
94188            2
94301            2
94518            1
94544            1
94602            1
```

```
94621           1
94901           1
95105           1
95109           1
95112           1
95117           1
95122           1
95132           1
95133           1
CA              2
Ca              1
```

Question 2b

Answer the following questions about the `postal_code` column in the `bus` dataframe.

1. The ZIP code column is which of the following type of data:
   A. Quantitative Continuous
   B. Quantitative Discrete
   C. Qualitative Ordinal
   D. Qualitative Nominal
2. What Python data type is used to represent a ZIP code?

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

Please write your answers in the variables below:

In [14]:

```
# The ZIP code column is which of the following type of data:
#   "Quantitative Continuous"
#   "Quantitative Discrete"
#   "Qualitative Ordinal"
#   "Qualitative Nominal"
q2b_part1 = "Quantitative Discrete"

# What Python data type is used to represent a ZIP code?
#    "str"
#    "int"
#    "bool"
#    "float"
q2b_part2 = "str"
```

Question 2c

In question 2a we noticed a large number of potentially invalid ZIP codes (e.g., "Ca"). These are likely due to data entry errors. To get a better understanding of the potential errors in the zip codes we will:

1. Import a list of valid San Francisco ZIP codes by using `pd.read_json` to load the file `data/sf_zipcodes.json` and extract a **series** of type `str` containing the valid ZIP codes. *Hint: set `dtype` when invoking `read_json`.*

2. Construct a `DataFrame` containing only the businesses which DO NOT have valid ZIP codes. You will probably want to use the `Series.isin` function.

Step 1

In [15]:

```
valid_zips = pd.read_json(dsDir/'sf_zipcodes.json', dtype = 'str')
valid_zips.head()
```

Out[15]:

	zip_codes
0	94102
1	94103
2	94104
3	94105
4	94107

Step 2

In [16]:

```
invalid_zip_bus = bus[bus['postal_code'].isin(valid_zips['zip_codes']) == False]
invalid_zip_bus.head(20)
```

Out[16]:

	bid	name	address	city	state	postal_code	latitude	longitu
22	100126	Lamas Peruvian Food Truck	Private Location	San Francisco	CA	-9999	-9999.000000	-9999.000
68	100417	COMPASS ONE, LLC	1 MARKET ST. FL	San Francisco	CA	94105-1420	-9999.000000	-9999.000
96	100660	TEAPENTER	1518 IRVING ST	San Francisco	CA	94122-1909	-9999.000000	-9999.000
109	100781	LE CAFE DU SOLEIL	200 FILLMORE ST	San Francisco	CA	94117-3504	-9999.000000	-9999.000
144	101084	Deli North 200	1 Warriors Way Level 300 North East	San Francisco	CA	94518	-9999.000000	-9999.000
156	101129	Vendor Room 200	1 Warriors Way Level 300 South West	San Francisco	CA	-9999	-9999.000000	-9999.000
177	101192	Cochinita #2	2 Marina Blvd Fort Mason	San Francisco	CA	-9999	-9999.000000	-9999.000
276	102014	DROPBOX (Section 3, Floor 7)	1800 Owens St	San Francisco	CA	-9999	-9999.000000	-9999.000
295	102245	Vessell CA Operations (#4)	2351 Mission St	San Francisco	CA	-9999	-9999.000000	-9999.000
298	10227	The Napper Tandy	3200 24th St	San Francisco	CA	-9999	37.752581	-122.416
320	10372	BERNAL HEIGHTS NEIGBORHOOD CENTER	515 CORTLAND AVE	San Francisco	CA	-9999	37.739110	-122.416
321	10373	El Tonayense #1	1717 Harrison St	San Francisco	CA	-9999	37.769426	-122.413
322	10376	Good Frikin Chicken	10 29th St	San Francisco	CA	-9999	37.744369	-122.420
324	10406	Sunset Youth Services	3918 Judah St	San Francisco	CA	-9999	37.760560	-122.504
357	11416	El Beach Burrito	3914 Judah St	San Francisco	CA	-9999	37.760851	-122.503
381	12199	El Gallo Giro	3055 23rd St	San Francisco	CA	-9999	37.754218	-122.413
384	12344	The Village Market & Pizza	750 Font Blvd	San Francisco	CA	-9999	37.723462	-122.483
406	13062	Everett Middle School	450 Church St	San Francisco	CA	-9999	37.763794	-122.428

	bid	name	address	city	state	postal_code	latitude	longitu
434	13753	Taboun	203 Parnassus Ave	San Francisco	CA	-9999	37.764574	-122.452
548	17423	Project Open Hand	100 Diamond St	San Francisco	CA	-9999	37.760689	-122.437

Question 2d

In the previous question, many of the businesses had a common invalid postal code that was likely used to encode a MISSING postal code. Do they all share a potentially "interesting address"?

In the following cell, construct a **series** that counts the number of businesses at each `address` that have this single likely MISSING postal code value. Order the series in descending order by count.

After examining the output, please answer the following question (2e) by filling in the appropriate variable. If we were to drop businesses with MISSING postal code values would a particular class of business be affected? If you are unsure try to search the web for the most common addresses.

In [17]:

```
missing_zip_address_count = invalid_zip_bus[invalid_zip_bus['postal_code'] == '-99
missing_zip_address_count.head()
```

Out[17]:

```
address
1 Warriors Way Level 300 South West    1
1 franklin Ct                          1
10 29th St                             1
100 Diamond St                         1
1001 Potrero Ave                       1
dtype: int64
```

Question 2e

True or False: *If we were to drop businesses with MISSING postal code values, a particular class of business will be affected.*

In [18]:

```
# True or False:
#  If we were to drop businesses with MISSING postal code values
#   a particular class of business be affected.
q2e_true_or_false = False
```

Question 2f

Examine the `invalid_zip_bus` dataframe we computed above and look at the businesses that DO NOT have the special MISSING ZIP code value. Some of the invalid postal codes are just the full 9 digit code rather than the first 5 digits. Create a new column named `postal5` in the original `bus` dataframe which contains only the first 5 digits of the `postal_code` column. Finally, for any of the `postal5` ZIP code entries that were not a valid San Fransisco ZIP Code (according to `valid_zips`) set the entry to `None`.

In [19]:

```
bus['postal5'] = None
bus['postal5'] = bus['postal_code'].astype(str).str[:5]
#bus[bus['postal5'].isin(invalid_zip_bus['postal5']),'postal5'] = None
# Checking the corrected postal5 column
bus.loc[invalid_zip_bus.index, ['bid', 'name', 'postal_code', 'postal5']]
```

Out[19]:

	bid	name	postal_code	postal5
22	100126	Lamas Peruvian Food Truck	-9999	-9999
68	100417	COMPASS ONE, LLC	94105-1420	94105
96	100660	TEAPENTER	94122-1909	94122
109	100781	LE CAFE DU SOLEIL	94117-3504	94117
144	101084	Deli North 200	94518	94518
...	...	...	...	...
6173	99369	HOTEL BIRON	94102-5917	94102
6174	99376	Mashallah Halal Food truck Ind	-9999	-9999
6199	99536	FAITH SANDWICH #2	94105-2907	94105
6204	99681	Twister	95112	95112
6241	99819	CHESTNUT DINER	94123-3106	94123

230 rows × 4 columns

3: Investigate the Inspection Data

Let's now turn to the inspection DataFrame. Earlier, we found that `ins` has 4 columns named `iid`, `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 [20]:

```
ins.head(5)
```

Out[20]:

	iid	date	score	type
0	100010_20190329	03/29/2019 12:00:00 AM	-1	New Construction
1	100010_20190403	04/03/2019 12:00:00 AM	100	Routine - Unscheduled
2	100017_20190417	04/17/2019 12:00:00 AM	-1	New Ownership
3	100017_20190816	08/16/2019 12:00:00 AM	91	Routine - Unscheduled
4	100017_20190826	08/26/2019 12:00:00 AM	-1	Reinspection/Followup

Question 3a

The column `iid` probably corresponds to an inspection id. Is it a primary key? Write an expression (line of code) that evaluates to `True` or `False` based on whether all the values are unique.

In [21]:

```
is_ins_iid_a_primary_key = (len(ins['iid'].unique()) == len(ins))
```

In [22]:

```
# grader.check("q3a")
```

Question 3b

The column `iid` appears to be the composition of two numbers and the first number looks like a business id.

Part 1.: Create a new column called `bid` in the `ins` dataframe containing just the business id. You will want to use `ins['iid'].str` operations to do this. Also be sure to convert the type of this column to `int`

Part 2.: Then compute how many values in this new column are invalid business ids (i.e. do not appear in the `bus['bid']` column). This is verifying a foreign key relationship. Consider using the `pd.Series.isin` function.

Part 3.: Answer True or False, `ins['bid']` is a foreign key reference to `bus['bid']`.

No python `for` loops or list comprehensions required!

Part 1

In [23]:

```
ins['bid'] = ins['iid'].str.split('_').str[0].astype(int)
```

Part 2

In [24]:

```
invalid_bid_count = len(ins) - ins['bid'].isin(bus['bid']).sum()
```

Part 3

In [25]:

```
# True or False: The column ins['bid'] is a foreign key
#   referencing the bus['bid'] primary key.

q3b_is_foreign_key = True
```

Question 3c

What if we are interested in a time component of the inspection data? We need to examine the date column of each inspection.

Part 1: What is the type of the individual `ins['date']` entries? You may want to grab the very first entry and use the `type` function in python.

Part 2: Use `pd.to_datetime` to create a new `ins['timestamp']` column containing of `pd.Timestamp` objects. These will allow us to do more date manipulation.

Part 3: What are the earliest and latest dates in our inspection data? *Hint: you can use `min` and `max` on dates of the correct type.*

Part 4: We probably want to examine the inspections by year. Create an additional `ins['year']` column containing just the year of the inspection. Consider using `pd.Series.dt.year` to do this.

No python `for` loops or list comprehensions required!

Part 1

In [26]:

```
ins_date_type = type(ins['date'][0])
ins_date_type
```

Out[26]:

```
str
```

Part 2

In [27]:

```
ins['timestamp'] = pd.to_datetime(ins['date'])
```

Part 3

In [28]:

```
earliest_date = min(ins['timestamp'])
latest_date = max(ins['timestamp'])

print("Earliest Date:", earliest_date)
print("Latest Date:", latest_date)
```

```
Earliest Date: 2016-10-04 00:00:00
Latest Date: 2019-11-28 00:00:00
```

Part 4

In [29]:

```
ins['year'] = ins['timestamp'].dt.year
```

In [30]:

```
ins.head()
```

Out[30]:

	iid	date	score	type	bid	timestamp	year
0	100010_20190329	03/29/2019 12:00:00 AM	-1	New Construction	100010	2019-03-29	2019
1	100010_20190403	04/03/2019 12:00:00 AM	100	Routine - Unscheduled	100010	2019-04-03	2019
2	100017_20190417	04/17/2019 12:00:00 AM	-1	New Ownership	100017	2019-04-17	2019
3	100017_20190816	08/16/2019 12:00:00 AM	91	Routine - Unscheduled	100017	2019-08-16	2019
4	100017_20190826	08/26/2019 12:00:00 AM	-1	Reinspection/Followup	100017	2019-08-26	2019

Question 3d

What is the relationship between the type of inspection over the 2016 to 2019 timeframe?

Part 1

Construct the following table by

1. Using the `pivot_table` containing the number (`size`) of inspections for the given `type` and `year` .
2. Adding an extra `Total` column to the result using `sum`
3. Sort the results in descending order by the `Total` .

year	2016	2017	2018	2019	Total
type					
Routine - Unscheduled	966	4057	4373	4681	14077
Reinspection/Followup	445	1767	1935	2292	6439
New Ownership	99	506	528	459	1592
Complaint	91	418	512	437	1458
New Construction	102	485	218	189	994
Non-inspection site visit	51	276	253	231	811
New Ownership - Followup	0	45	219	235	499
Structural Inspection	1	153	50	190	394
Complaint Reinspection/Followup	19	68	70	70	227
Foodborne Illness Investigation	1	29	50	35	115
Routine - Scheduled	0	9	8	29	46

year	2016	2017	2018	2019	Total
type					
Administrative or Document Review	2	1	1	0	4
Multi-agency Investigation	0	0	1	2	3
Special Event	0	3	0	0	3
Community Health Assessment	1	0	0	0	1

No python `for` loops or list comprehensions required!

In [31]:

```
ins_pivot = ins.pivot_table(
    index = 'type',
    columns = 'year',
    values = 'bid',
    aggfunc = 'count',
    fill_value = 0
).astype(int)
ins_pivot['Total'] = ins_pivot.sum(axis = 1)
ins_pivot_sorted = ins_pivot.sort_values('Total', ascending = False)
ins_pivot_sorted
```

Out[31]:

year	2016	2017	2018	2019	Total
type					
Routine - Unscheduled	966	4057	4373	4681	14077
Reinspection/Followup	445	1767	1935	2292	6439
New Ownership	99	506	528	459	1592
Complaint	91	418	512	437	1458
New Construction	102	485	218	189	994
Non-inspection site visit	51	276	253	231	811
New Ownership - Followup	0	45	219	235	499
Structural Inspection	1	153	50	190	394
Complaint Reinspection/Followup	19	68	70	70	227

Part 2

Based on the above analysis, which year appears to have had a lot of businesses in newly constructed buildings?

In [32]:

```
ins_pivot_sorted.iloc[:,0:4].query('type == ["New Construction"]').idxmax(axis=1)[
```

Out[32]:

```
2017
```

Question 3e

Let's examine the inspection scores `ins['score']`

In [33]:

```
ins['score'].value_counts().head()
```

Out[33]:

```
-1      12632
 100     1993
 96      1681
 92      1260
 94      1250
Name: score, dtype: int64
```

There are a large number of inspections with the `'score'` of `-1`. These are probably missing values. Let's see what type of inspections have scores and which do not. Create the following dataframe using steps similar to the previous question, and assign it to to the variable `ins_missing_score_pivot`.

You should observe that inspection scores appear only to be assigned to `Routine - Unscheduled` inspections.

Missing Score	False	True	Total
type			
Routine - Unscheduled	14031	46	14077
Reinspection/Followup	0	6439	6439
New Ownership	0	1592	1592
Complaint	0	1458	1458
New Construction	0	994	994
Non-inspection site visit	0	811	811
New Ownership - Followup	0	499	499
Structural Inspection	0	394	394
Complaint Reinspection/Followup	0	227	227
Foodborne Illness Investigation	0	115	115
Routine - Scheduled	0	46	46
Administrative or Document Review	0	4	4
Multi-agency Investigation	0	3	3
Special Event	0	3	3
Community Health Assessment	0	1	1

In [34]:

```
ins['Missing Score'] = (ins['score'] == -1)
ins_missing_score_pivot = ins.pivot_table(
    index = 'type',
    columns = 'Missing Score',
    values ='bid',
    aggfunc = 'count',
    fill_value = 0
).astype(int)
ins_missing_score_pivot['Total'] = ins_missing_score_pivot.sum(axis = 1)
ins_missing_score_pivot = ins_missing_score_pivot.sort_values('Total', ascending =
ins_missing_score_pivot
```

Out[34]:

Missing Score	False	True	Total
type			
Routine - Unscheduled	14031	46	14077
Reinspection/Followup	0	6439	6439
New Ownership	0	1592	1592
Complaint	0	1458	1458
New Construction	0	994	994
Non-inspection site visit	0	811	811
New Ownership - Followup	0	499	499
Structural Inspection	0	394	394
Complaint Reinspection/Followup	0	227	227
Foodborne Illness Investigation	0	115	115
Routine - Scheduled	0	46	46
Administrative or Document Review	0	4	4
Multi-agency Investigation	0	3	3
Special Event	0	3	3
Community Health Assessment	0	1	1

Notice that inspection scores appear only to be assigned to `Routine - Unscheduled` inspections. It is reasonable that for inspection types such as `New Ownership` and `Complaint` to have no associated inspection scores, but we might be curious why there are no inspection scores for the `Reinspection/Followup` inspection type.

4: Joining Data Across Tables

In this question we will start to connect data across mulitple tables. We will be using the `merge` function.

Question 4a

Let's figure out which restaurants had the lowest scores. Before we proceed, let's filter out missing scores from `ins` so that negative scores don't influence our results.

In [35]:

```
ins = ins[ins["score"] > 0]
```

We'll 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 [36]:

```
ins_named = pd.merge(ins, bus[['bid','name','address']], how = "left", on = "bid")
ins_named.head()
```

Out[36]:

	iid	date	score	type	bid	timestamp	year	Missing Score	nan
0	100010_20190403	04/03/2019 12:00:00 AM	100	Routine - Unscheduled	100010	2019-04-03	2019	False	ILI CAFI SF_PIE
1	100017_20190816	08/16/2019 12:00:00 AM	91	Routine - Unscheduled	100017	2019-08-16	2019	False	AMICI EAS COAS PIZZER
2	100041_20190520	05/20/2019 12:00:00 AM	83	Routine - Unscheduled	100041	2019-05-20	2019	False	UNCI LE CAI
3	100055_20190425	04/25/2019 12:00:00 AM	98	Routine - Unscheduled	100055	2019-04-25	2019	False	Twirl ar D
4	100055_20190912	09/12/2019 12:00:00 AM	82	Routine - Unscheduled	100055	2019-09-12	2019	False	Twirl ar D

Question 4b

Let's look at the 20 businesses with the lowest **median** score. Order your results by the median score followed by the business id to break ties. The resulting table should look like:

Hint: You may find the `as_index` *argument in the* `groupby` *method important*. The documentation is linked here! (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.groupby.html)

	bid	name	median score
3876	84590	Chaat Corner	54.0
4564	90622	Taqueria Lolita	57.0
4990	94351	VBowls LLC	58.0
2719	69282	New Jumbo Seafood Restaurant	60.5

	bid	name	median score
222	1154	SUNFLOWER RESTAURANT	63.5
1991	39776	Duc Loi Supermarket	64.0
2734	69397	Minna SF Group LLC	64.0
3291	78328	Golden Wok	64.0
4870	93150	Chez Beesen	64.0
4911	93502	Smoky Man	64.0
5510	98995	Vallarta's Taco Bar	64.0
1457	10877	CHINA FIRST INC.	64.5
2890	71310	Golden King Vietnamese Restaurant	64.5
4352	89070	Lafayette Coffee Shop	64.5
505	2542	PETER D'S RESTAURANT	65.0
2874	71008	House of Pancakes	65.0
818	3862	IMPERIAL GARDEN SEAFOOD RESTAURANT	66.0
2141	61427	Nick's Foods	66.0
2954	72176	Wolfes Lunch	66.0
4367	89141	Cha Cha Cha on Mission	66.5

In [37]:

```
twenty_lowest_scoring = ins_named[['bid','name','score']].groupby(['bid','name'],a
twenty_lowest_scoring.columns = ['bid','name','median score']
twenty_lowest_scoring.head(20)
```

Out[37]:

	bid	name	median score
3876	84590	Chaat Corner	54.0
4564	90622	Taqueria Lolita	57.0
4990	94351	VBowls LLC	58.0
2719	69282	New Jumbo Seafood Restaurant	60.5
222	1154	SUNFLOWER RESTAURANT	63.5
1991	39776	Duc Loi Supermarket	64.0
2734	69397	Minna SF Group LLC	64.0
4870	93150	Chez Beesen	64.0
4911	93502	Smoky Man	64.0
3291	78328	Golden Wok	64.0
5510	98995	Vallarta's Taco Bar	64.0
2890	71310	Golden King Vietnamese Restaurant	64.5
1457	10877	CHINA FIRST INC.	64.5
4352	89070	Lafayette Coffee Shop	64.5
505	2542	PETER D'S RESTAURANT	65.0
2874	71008	House of Pancakes	65.0
818	3862	IMPERIAL GARDEN SEAFOOD RESTAURANT	66.0
2141	61427	Nick's Foods	66.0
2954	72176	Wolfes Lunch	66.0
4367	89141	Cha Cha Cha on Mission	66.5

Question 4c

Let's now examine the descriptions of violations for inspections with `score > 0` and `score < 65`. Construct a **Series** indexed by the `description` of the violation from the `vio` table with the value being the number of times that violation occured for inspections with the above score range. Sort the results in descending order of the count.

The first few entries should look like:

```
Unclean or unsanitary food contact surfaces
43
High risk food holding temperature
42
Unclean or degraded floors walls or ceilings
40
Unapproved or unmaintained equipment or utensils
39
```

You will need to use `merge` twice.

In [38]:

```
low_score_violations = pd.merge(vio[['description','vid']],ins2vio[['iid','vid']],
low_score_violations = pd.merge(low_score_violations, ins[['iid','score']], on = "
low_score_violations = low_score_violations.query("score < 65 & score >0").groupby
low_score_violations.head(20)
```

Out[38]:

```
description
Unclean or unsanitary food contact surfaces
43
High risk food holding temperature
42
Unclean or degraded floors walls or ceilings
40
Unapproved or unmaintained equipment or utensils
39
Foods not protected from contamination
37
High risk vermin infestation
37
Inadequate food safety knowledge or lack of certified food safety mana
ger    35
Inadequate and inaccessible handwashing facilities
35
Improper thawing methods
30
Unclean hands or improper use of gloves
27
Improper cooling methods
25
Unclean nonfood contact surfaces
21
Improper food storage
20
Inadequately cleaned or sanitized food contact surfaces
20
Contaminated or adulterated food
18
Moderate risk vermin infestation
15
Permit license or inspection report not posted
13
Moderate risk food holding temperature
13
Food safety certificate or food handler card not available
12
Improper storage use or identification of toxic substances
10
Name: vid, dtype: int64
```

Question 4d

Let's figure out which restaurant had the worst scores ever (single lowest score).

In the cell below, write the name of the restaurant with the lowest inspection scores ever. You can also head to yelp.com and look up the reviews page for this restaurant. Feel free to add anything interesting you want to share.

In [39]:

```
worst_restaurant = ins_named.sort_values('score',ascending = True).iloc[0,8]
worst_restaurant
```

Out[39]:

```
'Lollipot'
```

5: Explore Inspection Scores

In this part we explore some of the basic inspection score values visually.

Question 5a

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 bar plot 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.

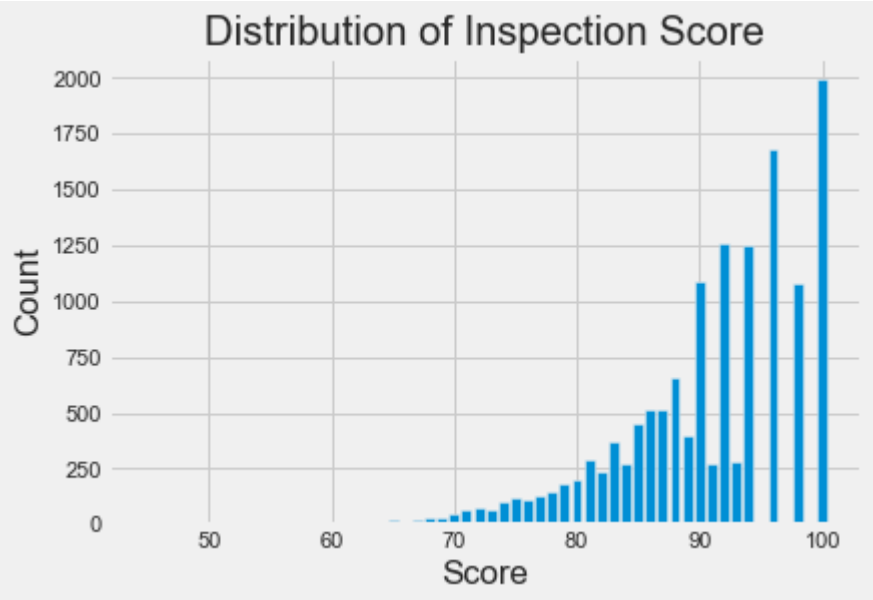
You might find this matplotlib.pyplot tutorial (https://matplotlib.org/tutorials/introductory/pyplot.html) useful. Key syntax that you'll need:

```
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 [40]:

```
scores = list(ins_named['score'].value_counts())
names = list(ins_named['score'].value_counts().index)
plt.bar(names, scores)
plt.xlabel("Score")
plt.ylabel("Count")
plt.title("Distribution of Inspection Score")
plt.show()
```



In [41]:

```
# score_counts
```

Question 5b

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

Modes: This is weakly bimodal which are 100 and some value between 90 and 100.
Symmetry: Nonsymmetric.
Tail: On the letf, so skew to the left.
Gaps: There are some gaps in the range (90,100).
Unusual features: There are very few low scores. This implies the restaurants are difficult to keep running their business if the inspection score is very low.

6: Restaurant Ratings Over Time

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

Question 6a

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".

The city would like to know if the state of food safety has been getting better, worse, or about average. This is a pretty vague and broad question, which you should expect as part of your future job as a data scientist! However for the ease of grading for this assignment, we are going to guide you through it and offer some specific directions to consider.

In [42]:

```
def swing(scores):
    return -1 if len(scores)<3 else max(scores)-min(scores)
max_swing = ins_named[['name','score']].groupby(['name']).agg(swing).sort_values('
max_swing
```

Out[42]:

```
'Lollipot'
```

Question 6b

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 2018 for this problem, using `ins2018` data frame that will be created for you below.

First, make a dataframe called `scores_pairs_by_business` indexed by `bid` (containing only businesses with exactly 2 inspections in 2018). 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:

bid	score_pair
48	[94, 87]
66	[98, 98]
146	[81, 90]
184	[90, 96]
273	[83, 84]

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 5 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 [43]:

```
ins2018 = ins[ins['year'] == 2018]
# Create the dataframe here
scores_pairs_by_business = ins2018[['score','bid','date']].sort_values('date')
scores_pairs_by_business = scores_pairs_by_business.groupby('bid').filter(lambda x
scores_pairs_by_business = scores_pairs_by_business.groupby('bid').agg(lambda x:li
scores_pairs_by_business
```

Out[43]:

bid	score_pair
48	[94, 87]
66	[98, 98]
146	[81, 90]
184	[90, 96]
273	[83, 84]
...	...
95621	[100, 100]
95628	[75, 75]
95674	[100, 96]
95761	[91, 87]
95764	[100, 92]

535 rows × 1 columns

Now, create your scatter plot in the cell below. It does not need to look exactly the same (e.g., no grid) as the sample below, 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'` and `edgecolors='b'` to make circle markers with blue borders.

`plt.plot` for the reference line.

`plt.xlabel`, `plt.ylabel`, `plt.axis`, and `plt.title`.

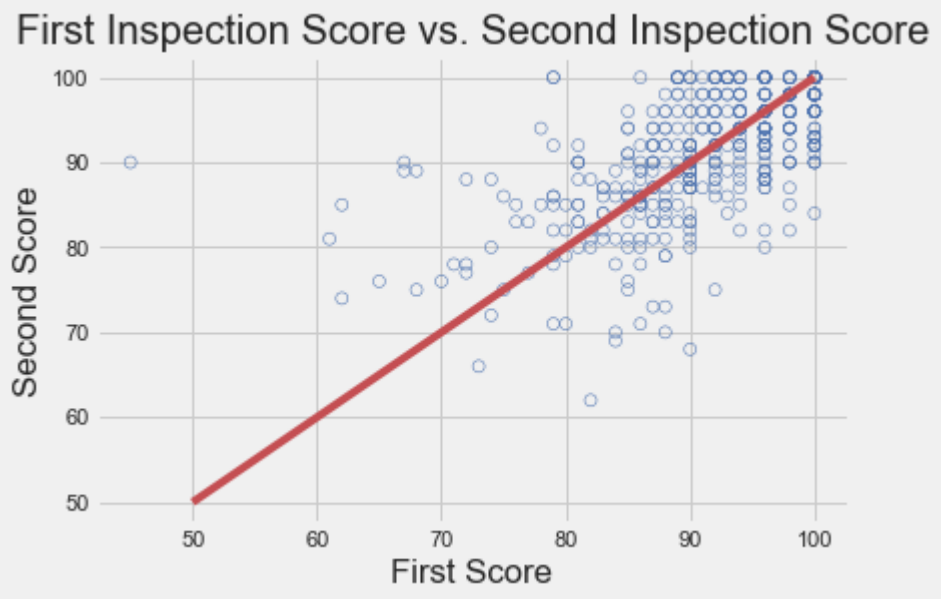
Hint: You may find it convenient to use the `zip()` function to unzip scores in the list.

In [44]:

```
x,y = zip(*scores_pairs_by_business['score_pair'])
plt.scatter(x,y,facecolors = 'none', edgecolors = 'b')
plt.plot([50,100],[50,100], color = 'r')
plt.xlabel("First Score")
plt.ylabel("Second Score")
plt.title("First Inspection Score vs. Second Inspection Score")
```

Out[44]:

```
Text(0.5, 1.0, 'First Inspection Score vs. Second Inspection Score')
```



Question 6c

If restaurants' scores tend to improve from the first to the second inspection, what do you expect to see in the scatter plot that you made in question 6b? What do you oberve from the plot? Are your observations consistent with your expectations?

Hint: What does the slope represent?

> I will observe more points lie in the upper region of the red line. The red line divide two regions, for the uppper region, it represents those restaurants whose second score is higher than first score. The lower region divided by the red line represents the opposite situation.

Question 6d

To wrap up our analysis of the restaurant ratings over time, one final metric we will be looking at is the distribution of restaurant scores over time. Create a side-by-side boxplot that shows the distribution of these scores for each different risk category from 2017 to 2019. Use a figure size of at least 12 by 8.

The boxplot should look similar to the sample below. Make sure the boxes are in the correct order!

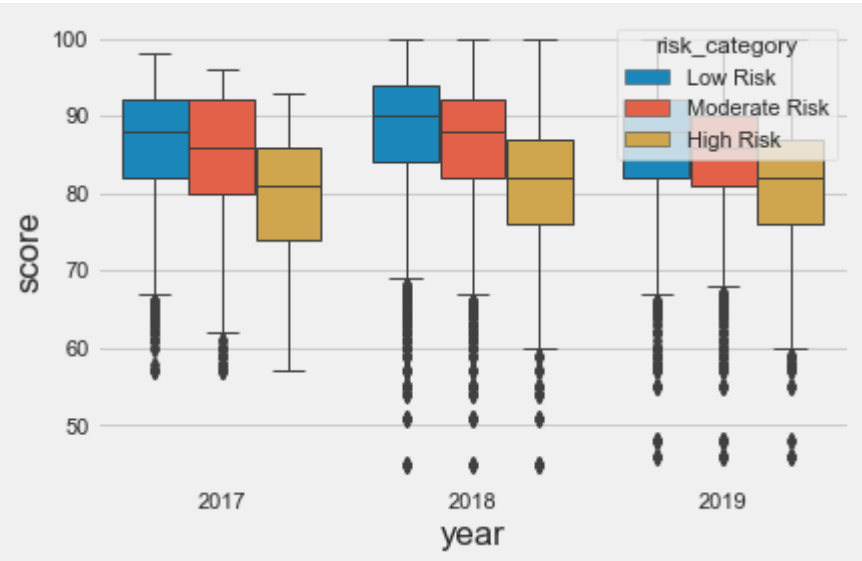
Hint: Use `sns.boxplot()` . Try taking a look at the first several parameters. The documentation is linked here! (https://seaborn.pydata.org/generated/seaborn.boxplot.html)

Hint: Use `plt.figure()` to adjust the figure size of your plot.

In [45]:

```
data = pd.merge(ins.query("year < 2020 & year > 2016"), ins2vio, how = 'left', on
risk = pd.merge(data, vio, how = 'left', on = 'vid')

sns.boxplot(x='year', y='score', hue='risk_category',
            hue_order = ['Low Risk', 'Moderate Risk', 'High Risk'],
            data=risk,
            linewidth = 1)
plt.figure(figsize = (16,9))
plt.show()
```




```
<Figure size 1152x648 with 0 Axes>
```

Summary of Inspections Data

We have done a lot in this project! Below are some examples of what we have learned about the inspections data through some cool visualizations!

- 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 change of scores over time! Many restaurants are not actually doing better.

7: Open Ended Question

Discover something interesting about the data!

Play with the data, and try to answer one question that you find interesting regarding the data. Show us how you would answer this question through exploratory data analysis.

Here are some possible routes you can take in completing your analysis:

- Construct a dataframe by computing something interesting about the data with methods such as `merge` / `groupby` / `pivot` , etc.
- Create a visualization with the data from which you can draw a conclusion that can answer you question.

Here are some possible questions you can ask about the data:

- How do the inspection scores relate to the geolocation (latitude, longitude) of a restaurant?
- How do all the inspection scores for each type of business change over time?

Note: You are not limited to the questions we provided above. We actually strongly recommend you to explore something you are personally interested in knowing about the data. On topics such as how the socioeconomic background of the neighborhoods impact all the nearby restaurants, you are welcome to reference external sources (make sure to quote the sources) as well to guide your exploration.

Please show your work in the cells below (feel free to use extra cells if you want), and describe in words **what question you were trying to answer** and **what you found through your analysis** within the same cell. This question will be graded leniently, but good solutions may be used to create future homework problems.

Grading

Since the question is more open ended, we will have a more relaxed rubric, classifying your answers into the following three categories:

- **Great** (4 points):
  - For a dataframe, a combination of pandas operations (such as groupby, pivot, merge) is used to answer a relevant question about the data. The text description provides a reasonable interpretation

answer a relevant question about the data. The text description provides a reasonable interpretation of the result.
    - For a visualization, the chart is well designed and the data computation is correct. The conclusion based on the visualization articulates a reasonable metric and correctly describes the relevant insight and answer to the question you are interested in.
- **Passing** (1-3 points):
    - For a dataframe, computation is flawed or very simple. The conclusion doesn't fully address the question, but reasonable progress has been made toward answering it.
    - For a visualization, a chart is produced but with some flaws such as bad encoding. The conclusion based on the visualization is incomplete but makes some sense.
- **Unsatisfactory** (0 points):
    - For a dataframe, no computation is performed, or the conclusion does not match what is computed at all.
    - For a visualization, no chart is created, or a chart with completely wrong results.

We will lean towards being generous with the grading.

You should have the following in your answers:

- a question you want to explore about the data.
- either of the following:
    - a few computed dataframes.
    - a few visualizations.
- a few sentences summarizing what you found based on your analysis and how that answered your question (not too long please!)

Please limit the number of your computed dataframes and visualizations **you plan on showing** to no more than 5.

In [46]:

```
# The top 20 restaurants with highest potential and 20 restaurants with alarming o

# YOUR DATA PROCESSING AND PLOTTING HERE
# 1. Top 20 restaurants with highest potentials
# First, we need to find out those restaurants with highest comprehensive growth r
def CAGR(scores):
    return -100 if len(scores)<3 else ((scores.iloc[len(scores)-1]/scores.iloc[0])

insNamed = ins_named.copy(deep=False)
CAGRS = insNamed[['bid','name','score']].groupby(['bid','name'],as_index = False).
highest_potential = CAGRS.sort_values('cagr',ascending = False).query("cagr != -10
highest_potential
```

Out[46]:

	bid	name	cagr
2504	66961	Mi Tierra Market	0.277333
3674	82361	Yangtze Market	0.224745
2902	71440	New Garden Restaurant, Inc.	0.221158
2497	66874	Peninsula Seafood Restaurant	0.217686
3291	78328	Golden Wok	0.201092
473	2359	LA TRAVIATA	0.191367
189	984	VANIDA THAI KITCHEN	0.176697
992	4978	Angel's Market	0.171080
1258	6619	J C Market	0.170411
674	3241	SHENG KEE BAKERY	0.162804
3451	80242	Wing Lee BBQ Restaurant	0.159610
1754	33911	B Star	0.156821
3149	76676	Little Vietnam Cafe	0.154701
2188	62330	Social Kitchen & Brewery	0.150447
1574	19003	Panchitas Restaurant	0.150035
3389	79430	City Discount Meat & Grocery Market	0.147807
4302	88524	McDonald's	0.146764
388	1973	MY FAVORITE CAFE	0.142080
317	1641	MARTHA & BROS. COFFEE CO	0.139606
1670	28206	Westfield Food Court Scullery	0.138990

In [47]:

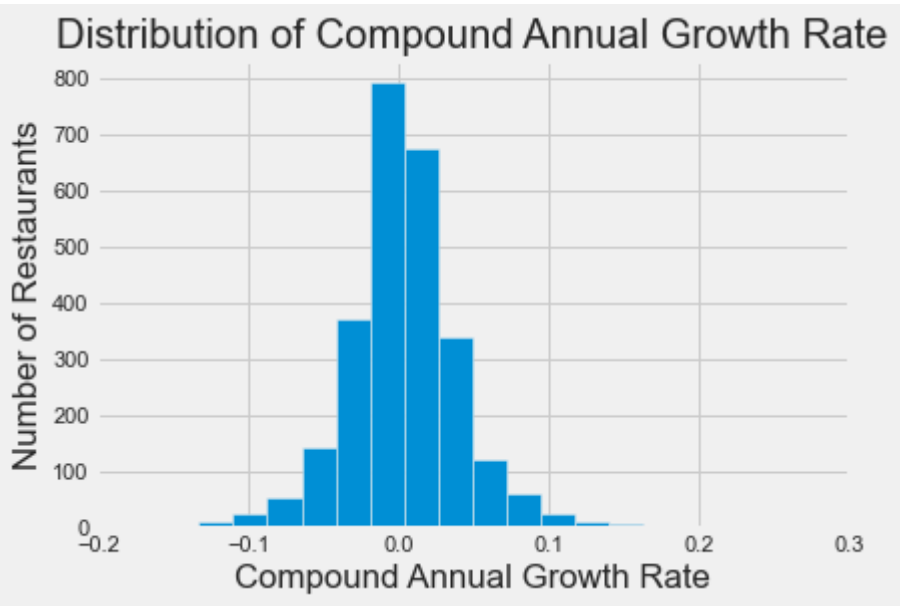
```
warning_condition = CAGRS.sort_values('cagr',ascending = True).query("cagr != -100")
warning_condition
```

Out[47]:

	bid	name	cagr
3462	80316	Kingdom of Dumpling	-0.178770
4360	89115	Eden Silk Road Cuisine	-0.162069
2874	71008	House of Pancakes	-0.156885
4033	86138	Baladie Cafe	-0.151218
3281	78220	Howard & 6th Street Food Market Inc.	-0.137641
197	1008	VALENCIA PIZZA & PASTA	-0.133975
2821	70281	Shai Lai Seafood Restaurant	-0.132354
4532	90448	Lin's Kitchen	-0.130624
655	3167	MING'S DINER	-0.130282
3114	76218	Tenderloin Market & Deli	-0.128511

In [48]:

```
cagr = list(CAGRS.query("cagr != -100")['cagr'])
plt.hist(cagr, bins = 20)
plt.xlabel("Compound Annual Growth Rate")
plt.ylabel("Number of Restaurants")
plt.title("Distribution of Compound Annual Growth Rate")
plt.show()
```



Conclusion: By studying at the compound annual growth rate (CAGR), we can know how whether a restaurant is becomming better or worse. Some restaurants have a high cagr, indicating they are growing really fast. This might be due to various reasons, namely the update of the kitchen, higher working efficiency. Those restaurant grows fast also indicate they are becoming the trend in some aspects. By observing the histogram, we see there are slightly more restaurant in a negative CAGR than those with positive one.

Congratulations! You have finished Homework 2!