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. 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 [ ]:

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

Out[ ]:

	business id column	name	address	city	state	postal_code	lati
0	1000	HEUNG YUEN RESTAURANT	3279 22nd St	San Francisco	CA	94110	37.75
1	100010	ILLY CAFFE SF_PIER 39	PIER 39 K-106-B	San Francisco	CA	94133	-9999.00
2	100017	AMICI'S EAST COAST PIZZERIA	475 06th St	San Francisco	CA	94103	-9999.00
3	100026	LOCAL CATERING	1566 CARROLL AVE	San Francisco	CA	94124	-9999.00
4	100030	OUI OUI! MACARON	2200 JERROLD	San Francisco	CA	94124	-9999.00

AVE STE
C

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

Out[ ]:

	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 [ ]:
```
assert all(bus.columns == ['business id column', 'name', '
                           'latitude', 'longitude', 'phone
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',
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 [ ]:
```
bus_summary = pd.DataFrame(**{'columns': ['business id col
 'data': {'business id column': {'50%': 75685.0, 'max': 10
  'latitude': {'50%': -9999.0, 'max': 37.824494, 'min': -9
  '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':
 '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 [ ]:
```
"""Run this cell to load this utility comparison function
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

    Compute the min, median and max of the two dataframes
    that they match numerically to the given relative tole

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

    # For the desired values, we can provide a full DF wit
    # the actual data, or pre-computed summary statistics.
    # We assume a pre-computed summary was provided if col
    # `desired` *must* have the same structure as the actu
    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.

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

Out[ ]:

	business id column	name	address	city	state	postal_code	lati
0	1000	HEUNG YUEN RESTAURANT	3279 22nd St	San Francisco	CA	94110	37.75
1	100010	ILLY CAFFE SF_PIER 39	PIER 39 K-106-B	San Francisco	CA	94133	-9999.00
2	100017	AMICI'S EAST COAST PIZZERIA	475 06th St	San Francisco	CA	94103	-9999.00
3	100026	LOCAL CATERING	1566 CARROLL AVE	San Francisco	CA	94124	-9999.00
4	100030	OUI OUI! MACARON	2200 JERROLD AVE STE C	San Francisco	CA	94124	-9999.00

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

Out[ ]:

	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

In [ ]:

```
vio.head()
```

Out[ ]:

	description	risk_category	vid
0	Consumer advisory not provided for raw or unde...	Moderate Risk	103128
1	Contaminated or adulterated food	High Risk	103108
2	Discharge from employee nose mouth or eye	Moderate Risk	103117
3	Employee eating or smoking	Moderate Risk	103118
4	Food in poor condition	Moderate Risk	103123

ANSWER: The phone number in the first row of bus is -9999 which is unreasonable. Also, the latitude and longitude in this file is also weird which might cause issues when doing analysis about these values.

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

Out[ ]:

	business id column	name	address	city	state	postal_code	lati
0	1000	HEUNG YUEN RESTAURANT	3279 22nd St	San Francisco	CA	94110	37.75
1	100010	ILLY CAFFE SF_PIER 39	PIER 39 K-106-B	San Francisco	CA	94133	-9999.00
2	100017	AMICI'S EAST COAST PIZZERIA	475 06th St	San Francisco	CA	94103	-9999.00
3	100026	LOCAL CATERING	1566 CARROLL AVE	San Francisco	CA	94124	-9999.00
4	100030	OUI OUI! MACARON	2200 JERROLD AVE STE C	San Francisco	CA	94124	-9999.00

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 [ ]:
```
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 [ ]:
```
is_bid_unique = bus.shape[0] == bus['bid'].unique().size
```

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 [ ]:
```
top_names = bus['name'].value_counts().sort_values(ascendi
top_addresses = bus['address'].value_counts().sort_values(
top_names, top_addresses
```

Out[ ]:
```
(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?

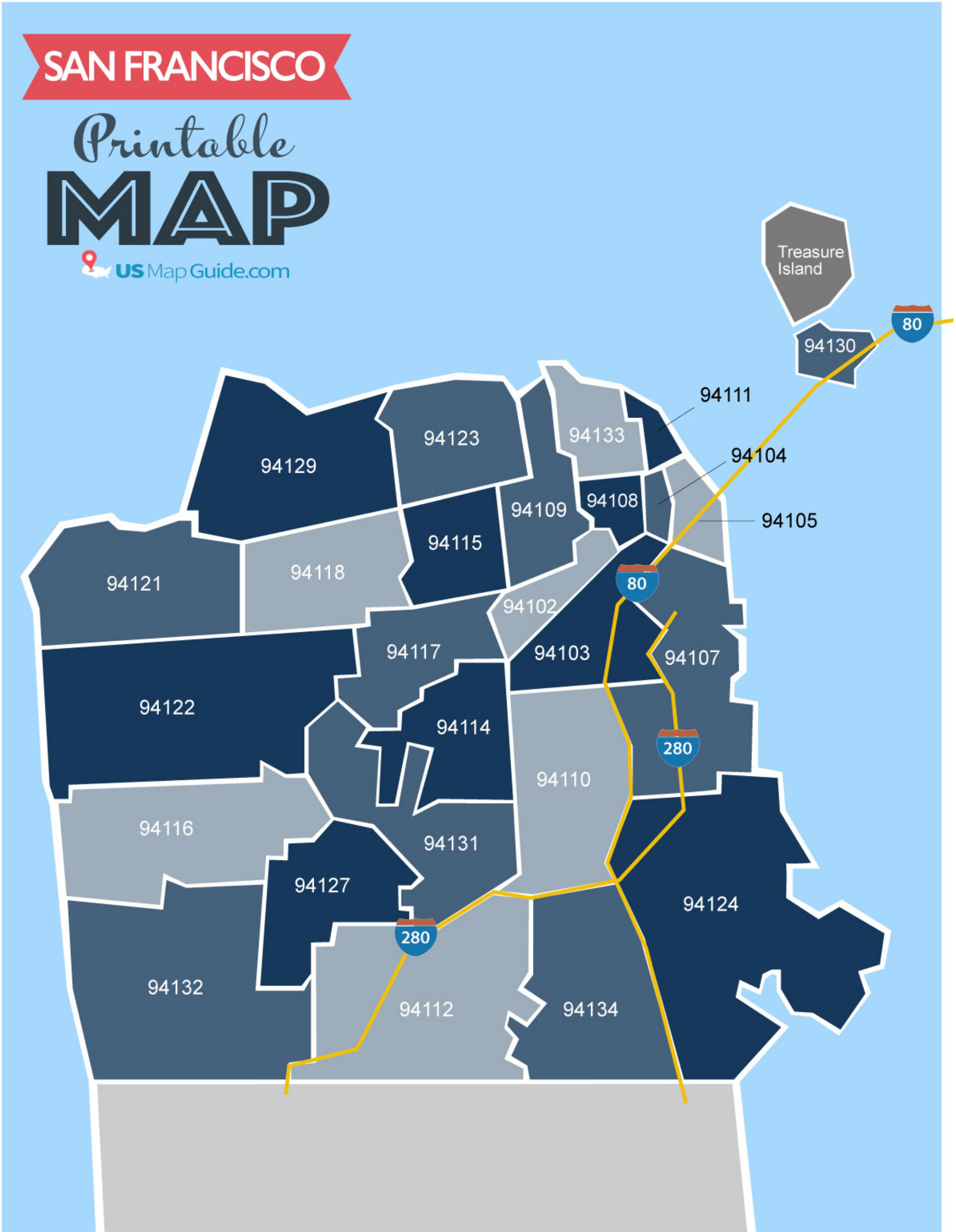
Minial primary key: a minimal set of attributes (columns) that uniquely specify a row in a table

In [ ]:
```
# 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 [ ]:
```
zip_counts = bus.groupby('postal_code').size().sort_values
print(zip_counts.to_string())
```

```
postal_code
94103         562
94110         555
94102         456
94107         408
94133         398
94109         382
94111         259
94122         255
94105         249
94118         231
94115         230
94108         229
94124         218
94114         200
-9999         194
94112         192
94117         189
94123         177
94121         157
94104         142
94132         132
94116          97
94158          90
94134          82
94127          67
94131          49
94130           8
94143           5
94013           2
94188           2
CA              2
94301           2
94101           2
95122           1
941033148       1
95133           1
95132           1
94102-5917      1
94014           1
941             1
94080           1
94105-2907      1
```

```
92672          1
64110          1
00000          1
94105-1420     1
941102019      1
95117          1
95112          1
95109          1
95105          1
94901          1
94621          1
94602          1
94544          1
94518          1
94117-3504     1
94120          1
94122-1909     1
94123-3106     1
94124-1917     1
94129          1
Ca             1
```

Question 2b

Answer the question about the `postal_code` column in the `bus` dataframe.

1. 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 [ ]:
```
# What Python data type is used to represent a ZIP code?
#     "str"
#     "int"
#     "bool"
#     "float"
q2b = "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 [ ]:
```
valid_zips = pd.read_json("data/sf_zipcodes.json", dtype="
valid_zips.head()
```

Out[ ]:

	zip_codes
0	94102
1	94103
2	94104
3	94105
4	94107

Step 2

In [ ]:
```
invalid_zip_bus = bus[~bus["postal_code"].isin(valid_zips[
invalid_zip_bus.head(20)
```

Out[ ]:

	bid	name	address	city	state	postal_code	
22	100126	Lamas Peruvian Food Truck	Private Location	San Francisco	CA	-9999	-99
68	100417	COMPASS ONE, LLC	1 MARKET ST. FL	San Francisco	CA	94105-1420	-99
96	100660	TEAPENTER	1518 IRVING ST	San Francisco	CA	94122-1909	-99
109	100781	LE CAFE DU SOLEIL	200 FILLMORE ST	San Francisco	CA	94117-3504	-99

144	101084	Deli North 200	1 Warriors Way Level 300 North East	San Francisco	CA	94518	-99
156	101129	Vendor Room 200	1 Warriors Way Level 300 South West	San Francisco	CA	-9999	-99
177	101192	Cochinita #2	2 Marina Blvd Fort Mason	San Francisco	CA	-9999	-99
276	102014	DROPBOX (Section 3, Floor 7)	1800 Owens St	San Francisco	CA	-9999	-99
295	102245	Vessell CA Operations (#4)	2351 Mission St	San Francisco	CA	-9999	-99
298	10227	The Napper Tandy	3200 24th St	San Francisco	CA	-9999	
320	10372	BERNAL HEIGHTS NEIGBORHOOD CENTER	515 CORTLAND AVE	San Francisco	CA	-9999	
321	10373	El Tonayense #1	1717 Harrison St	San Francisco	CA	-9999	
322	10376	Good Frikin Chicken	10 29th St	San Francisco	CA	-9999	
324	10406	Sunset Youth Services	3918 Judah St	San Francisco	CA	-9999	
357	11416	El Beach Burrito	3914 Judah St	San Francisco	CA	-9999	
381	12199	El Gallo Giro	3055 23rd St	San Francisco	CA	-9999	
384	12344	The Village Market & Pizza	750 Font Blvd	San Francisco	CA	-9999	
406	13062	Everett Middle School	450 Church St	San Francisco	CA	-9999	
434	13753	Taboun	203 Parnassus Ave	San Francisco	CA	-9999	
548	17423	Project Open	100	San	CA	-9999	

Hand	Diamond St	Francisco

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 [ ]:

```
missing_zip_address_count = invalid_zip_bus.groupby(['addr
missing_zip_address_count.head()
```

Out[ ]:

```
address
Off The Grid                  39
Off the Grid                  10
OTG                            4
OFF THE GRID                   4
Approved Private Locations     3
dtype: int64
```

Question 2e

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 [ ]:

```
bus['postal5'] = None
bus['postal5'] = bus['postal_code'].str[:5]
bus.loc[invalid_zip_bus.index, 'postal5'] = None

# Checking the corrected postal5 column
bus.loc[invalid_zip_bus.index, ['bid', 'name', 'postal_cod
```

Out[ ]:

	bid	name	postal_code	postal5
22	100126	Lamas Peruvian Food Truck	-9999	None
68	100417	COMPASS ONE, LLC	94105-1420	None
96	100660	TEAPENTER	94122-1909	None
109	100781	LE CAFE DU SOLEIL	94117-3504	None
144	101084	Deli North 200	94518	None
...	...	...	...	...
6173	99369	HOTEL BIRON	94102-5917	None
6174	99376	Mashallah Halal Food truck Ind	-9999	None
6199	99536	FAITH SANDWICH #2	94105-2907	None
6204	99681	Twister	95112	None
6241	99819	CHESTNUT DINER	94123-3106	None

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

Out[ ]:

	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 [ ]:
```
is_ins_iid_a_primary_key = ins.shape[0] == ins['iid'].uniq
```

In [ ]:
```
ins.shape[0]
```

Out[ ]:
```
26663
```

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). Consider using the `pd.Series.isin` function.

No python `for` loops or list comprehensions required!

Part 1

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

Part 2

In [ ]:
```
invalid_bid_count = ins[~ins['bid'].isin(bus['bid'])].shap
```

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 [ ]:
```
ins_date_type = type(ins.loc[0, 'date'])
ins_date_type
```

Out[ ]:
```
str
```

Part 2

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

Part 3

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

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

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

Out[ ]:

	iid	date	score	type	bid	time
0	100010_20190329	03/29/2019 12:00:00 AM	-1	New Construction	100010	20
1	100010_20190403	04/03/2019 12:00:00 AM	100	Routine - Unscheduled	100010	20
2	100017_20190417	04/17/2019 12:00:00 AM	-1	New Ownership	100017	20
3	100017_20190816	08/16/2019 12:00:00 AM	91	Routine - Unscheduled	100017	20
4	100017_20190826	08/26/2019 12:00:00 AM	-1	Reinspection/Followup	100017	20

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
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 [ ]:

```
ins_pivot = ins.pivot_table(index='type', columns='year',
ins_pivot['Total'] = ins_pivot.sum(axis='columns')

...
ins_pivot_sorted = ins_pivot.sort_values('Total', ascendin
ins_pivot_sorted
```

Out[ ]:

	score				Total
year	2016	2017	2018	2019	
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
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

Part 2

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

In [ ]:
```
year_of_new_construction = 2017
```

Question 3e

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

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

Out[ ]:
```
-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 [ ]:
```
ins['Missing score'] = ins['score'] != -1
ins_missing_score_pivot = ins.pivot_table(index='type', co
ins_missing_score_pivot['Total'] = ins_missing_score_pivot
ins_missing_score_pivot.sort_values('Total', ascending=Fal
```

Out[ ]:

Missing score type	False	True	Total
Routine - Unscheduled	46	14031	14077
Reinspection/Followup	6439	0	6439
New Ownership	1592	0	1592
Complaint	1458	0	1458
New Construction	994	0	994
Non-inspection site visit	811	0	811
New Ownership - Followup	499	0	499
Structural Inspection	394	0	394
Complaint Reinspection/Followup	227	0	227
Foodborne Illness Investigation	115	0	115
Routine - Scheduled	46	0	46
Administrative or Document Review	4	0	4
Multi-agency Investigation	3	0	3
Special Event	3	0	3
Community Health Assessment	1	0	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 [ ]:
```
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 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 [ ]:
```
ins_named = pd.merge(ins, bus[['bid', 'name', 'address']],
ins_named.head()
```

Out[ ]:

	iid	date	score	type	bid	timestamp	y

0	100010_20190403	04/03/2019 12:00:00 AM	100	Routine - Unscheduled	100010	2019-04-03	2
1	100017_20190816	08/16/2019 12:00:00 AM	91	Routine - Unscheduled	100017	2019-08-16	2
2	100041_20190520	05/20/2019 12:00:00 AM	83	Routine - Unscheduled	100041	2019-05-20	2
3	100055_20190425	04/25/2019 12:00:00 AM	98	Routine - Unscheduled	100055	2019-04-25	2
4	100055_20190912	09/12/2019 12:00:00 AM	82	Routine - Unscheduled	100055	2019-09-12	2

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!

	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
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 [ ]:

```
twenty_lowest_scoring = ins_named.groupby(['bid', 'name'],
...

twenty_lowest_scoring
```

Out[ ]:

	bid	name	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
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
2141	61427	Nick's Foods	66.0
2954	72176	Wolfes Lunch	66.0
4367	89141	Cha Cha Cha on Mission	66.5
4726	91843	Hello Sandwich & Noodle	66.5
2985	74216	MDJK Food Service	67.0
1710	32887	Mission Beach Cafe	67.5

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 [ ]:

```
ins_new = ins_named[ins_named['score'] < 65]
low_score_violations = pd.merge(ins_new, ins2vio, on='iid'
low_score_violations = pd.merge(low_score_violations, vio,
low_score_violations = low_score_violations.groupby('descr

low_score_violations.head(20)
```

Out[ ]:

```
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 manager    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
Inadequately cleaned or sanitized food contact surfaces
20
Improper food storage
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
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 [ ]:

```
ins_named.sort_values('score', ascending=True).head()
```

Out[ ]:

	iid	date	score	type	bid	timestamp
10898	86718_20180522	05/22/2018 12:00:00 AM	45	Routine - Unscheduled	86718	2018-05-22
291	1154_20190327	03/27/2019 12:00:00 AM	46	Routine - Unscheduled	1154	2019-03-27
236	10877_20190701	07/01/2019 12:00:00 AM	48	Routine - Unscheduled	10877	2019-07-01
6433	67237_20180914	09/14/2018 12:00:00 AM	51	Routine - Unscheduled	67237	2018-09-14
10285	84590_20181001	10/01/2018 12:00:00 AM	54	Routine - Unscheduled	84590	2018-10-01

In [ ]:

```
worst_restaurant = 'Lollipot'
worst_restaurant
```

Out[ ]:

```
'Lollipot'
```

Congratulations! You have finished Homework 2!