Homework: Food safety inspections

Background

For this assignment, you will write a program that provides information about food safety inspections for businesses in Prince George's County, Maryland. For a user-specified geographical area, your program will report how many critical violations each food service business has been cited for in the past 10 years.

Data

You will work with data originally from https://data.princegeorgescountymd.gov/Health/Food-Inspection/umjn-t2iz, which has been modified slightly for this assignment. You have been given two files:

- **`businesses.csv`**: data about businesses in Prince George's County that serve food. For this assignment, we care about the following columns:
  - Establishment_id: a unique ID for the business
  - Name: the name of the business
  - Latitude: north/south component of the business's location, as a float
  - Longitude: east/west component of the business's location, as a float
- **`inspections.csv`**: data about food safety inspections conducted between July 1, 2011 and April 15, 2021. For this assignment, we care about the following columns:
  - Establishment_id: a unique ID for the business being inspected
  - Inspection_results: the outcome of the inspection. We are especially interested in inspections whose outcome is `Critical Violations observed`.

Bounding boxes

A bounding box is a rectangular area of interest. For this assignment, users will specify a bounding box that will contain the businesses they want information about. The bounding box will be defined by two latitude values and two longitude values. The latitude values define the north and south edges of the bounding box, and the longitude values define the east and west edges.

You can expect to receive these values in this order: *lat1*, *lon1*, *lat2*, *lon2*. You can think of (*lat1*, *lon1*) and (*lat2*, *lon2*) as diagonally opposite corners of a rectangle (for example, the northeast corner and the southwest corner). It is your responsibility to determine which latitude coordinate is smaller and which longitude coordinate is smaller. For purposes of testing your code, you can obtain the latitude and longitude of any point in Google Maps by right-clicking on the point.

Applying the bounding box to the business DataFrame will be a matter of finding rows in the DataFrame whose latitude and longitude fall within the specified values. You may wish to incorporate the `between()` method of Pandas Series objects into your solution.[1] This method takes two values, a minimum and a maximum, and returns a Series object with `True` values for indexes that fall within the specified minimum and maximum, and `False` values for other indexes. For example, consider the Series object `height_series` below:

```
height_data = [0, 73, 1, 54, 2, 72, 3, 61, 4, 55, 5, 78]
height_series = pd.Series(height_data)
```

If we are interested in indexes with a height between 60 and 73, we can find these using `between(60, 73)`, like this:

```
medium_heights = height_series.between(60, 73)
```

The resulting Series object looks like this:

```
0    True
1   False
2    True
3    True
4   False
5   False
dtype: bool
```

Note that by default, the minimum and maximum values are included in the `between()` range.

Selecting specific columns from a DataFrame

To create a DataFrame containing a subset of columns from another DataFrame, you create a list of the columns of interest and include that list in the index operator of the DataFrame. This can be done in two steps:

```
cols = ["Name", "Address"]
df_subset = df[cols]
```

Or this can be done in one step:

```
df_subset = df[["Name", "Address"]]
```

Filling in missing values in a DataFrame

Sometimes data sets are missing values. Missing values in Pandas are represented by the special constant `nan` ("not a number"), which is defined in the Numpy package. Pandas DataFrames have a method `fillna()` which will replace all `nan` values with a value you specify. For example, imagine a DataFrame called `football_players` with a column called `"Current_team"`. Imagine that for players who are not currently part of a team, this column has a `nan` value. To replace these with empty strings, you could do `football_players.fillna(0)`.

By default, `fillna()` returns a new DataFrame object and leaves the original DataFrame untouched. If you want `fillna()` to change the original DataFrame directly, you can provide the keyword argument `inplace=True` (e.g., `football_players.fillna(0, inplace=True)`). In this case, `fillna()` alters the existing DataFrame and returns `None`.

Template

A template file, `inspections.py`, is provided. The file includes a `main()` function which will instantiate your class and print the output of the analysis. It also includes a function `parse_args()` which processes command-line arguments, and an `if __name__ == "__main__":` statement which manages overall execution of the program.

Problem statement

Write a class that will count critical food safety violations for all businesses that occur within a specified bounding box.

Instructions

Use the template `inspections.py` to develop your solution. Your program should be called `inspections.py`.

`Inspections` class

Write a class called `Inspections`. For your information, this class will have two attributes:

- `df_biz`: a Pandas DataFrame containing information about the businesses that are subject to food safety inspections. The essential columns of this DataFrame are described in the Data section above.
- `df_ins`: a Pandas DataFrame containing data from food safety inspections. The essential columns of this DataFrame are described in the Data section above.

Define the following methods for this class.

`__init__()` method

Write an `__init__()` method with three parameters:

- `self`
- a string containing the path to a CSV file containing information about food service businesses
- a string containing the path to a CSV file containing data from food safety inspections

Read the first CSV file into a Pandas DataFrame and store it in the `df_biz` attribute. Read the second CSV file into a Pandas DataFrame and store it in the `df_ins` attribute.

`apply_bbox()` method

Write an `apply_bbox()` method with five parameters:

- `self`
- a latitude value to use in defining a bounding box
- a longitude value to use in defining the bounding box
- another latitude value to use in defining the bounding box
- another longitude value to use in defining the bounding box

Return a Pandas DataFrame containing only the rows of `self.biz` that describe businesses whose latitude and longitude lie within the specified bounding box.

When implementing this method, you may want to review the Bounding boxes section above. You may also want to review the "DataFrame objects" and "Advanced filtering" lecture videos from the Pandas module.

`count_violations()` method

Write a method called `count_violations()` with two parameters:

- `self`
- a Pandas DataFrame containing information about businesses of interest (such as the DataFrame returned by `apply_bbox()`) at a minimum, this DataFrame should have columns "Establishment_id" (a unique identifier for each business) and "Name" (the name of the business as a string)

Merge the DataFrame from the second parameter with `self.ins` on the `Establishment_id` column. Include only businesses that occur in both DataFrames.

For each unique `Establishment_id`, count the number of times the `Inspection_results` column contains the value `Critical Violations observed` (note the inconsistent capitalization). Return a DataFrame (not a Series) with `Establishment_id` as the index and `Inspection_results` as the only column.

Hint: There is a fun different way to do the counting step. Depending on what form your counts are in, one of the following suggestions may be helpful to you.

If your counts are in a DataFrame with counts of several columns and you want a DataFrame with just a subset of those columns, you can put a list of the column(s) you want in the index operator of the DataFrame. For example, if your counts are in a DataFrame called `df` and you want to return a DataFrame containing only counts of the `Inspection_results` column, `df[["Inspection_results"]]` will give you what you want.

If your counts are in a Series, you can convert it into a DataFrame using `pd.DataFrame()`. For example, if your counts are stored as a Series in a variable `counts` that contains counts of the `Inspection_results` column and has `Inspection_results` as its name, `pd.DataFrame(counts)` will give you what you want.

`find_violations()` method

Write a method called `find_violations()` with the same five parameters as the `apply_bbox()` method.

Use the `apply_bbox()` method to get a DataFrame containing only the subset of businesses from `self.biz` that are within the specified bounding box. Use the `count_violations()` method to count all critical violations for these businesses.

Merge the DataFrame you obtained from `apply_bbox()` with the DataFrame you obtained from `count_violations()` such that the resulting DataFrame contains rows for all businesses in the specified bounding box, whether or not they had any critical violations. For businesses that had no critical violations, use the `fillna()` method to set their counts to 0 (see Filling in missing values in a DataFrame above).

Return a DataFrame with only the "Name" and "Inspection_results" columns from the merged DataFrame. For information on how to do this, see Selecting specific columns from a DataFrame above.

Docstrings

Please write docstrings for the `Inspections` class and the `apply_bbox()`, `count_violations()`, and `find_violations()` methods.

Docstrings were covered in the first week's lecture videos (https://youtu.be/iHTv83PiQVw?t=1415) and revisited in the OOP lecture videos (https://youtu.be/OqPsuywIMPg). There's an ELMS page about them here: https://umd.instructure.com/courses/1299872/pages/docstrings. Please follow the specified docstring format; I encourage you to pattern your docstrings after the docstrings in the `main()` and `parse_args()` functions.

Docstrings are not comments; they are statements. Python recognizes a string as a docstring if it is the first statement in the body of the method, function, class, or script/module it documents. Because docstrings are statements, the quotation mark at the start of the docstring must align exactly with the start of other statements in the method, function, class, or module.

▸ General instructions for class docstrings

▸ General instructions method and function docstrings

Running your program at the command line

The template is designed to use command-line arguments. To run your program within the VS Code built-in terminal, first make sure you have opened (in VS Code) the folder where your program is saved. If necessary, you can go to the VS Code File menu and select "Open..." on macOS or "Open Folder..." on Windows, and navigate to the directory where your program is.

Then, open the VS Code built-in terminal. On a single line, type the following information, separated by spaces:

- `python3` (on macOS) or `python` (on Windows)
- the path to the CSV file containing business information (businesses.csv)
- the path to the CSV file containing inspection information (inspections.csv)
- a latitude value
- a longitude value
- another latitude value
- another longitude value

Below is an example. The example assumes that `businesses.csv` and `inspections.csv` are in the same directory as your program.

```
python3 inspections.py businesses.csv inspections.csv 38.9824 -76.9391 38.9797 -76.9367
```

Submitting your work

Submit your work using Gradescope. Please upload only `inspections.py` (do not upload any test scripts or CSV files).

`inspections.py` will be partially auto-graded by Gradescope. If you are not happy with the results, you may revise your code and resubmit as many times as you like until the deadline.

No extensions on this one

This homework is offered for extra credit and made available until the last day of classes. No extensions are possible. I recommend getting started early.

Grading

This assignment is offered for extra credit. It is worth 30 points in the homework category. (Points in the homework category will be capped at 100%.) The points will be allocated as follows:

14 points are allocated to automatic tests of your code functionality. 4 points are allocated to automatic tests of your docstrings. The remaining 12 points are awarded based on the degree of completeness of your program and docstrings.

Category	Points	Notes
Automatic tests of code functionality	14	Tests will evaluate instance attributes; return values; side effects; and whether errors are raised when expected
Automatic tests of docstrings	4	Tests will look for existence of docstrings and presence of expected sections in each docstring
Manual evaluation of code completeness	8	Does the code contain the correct classes, methods, and functions? Was a recognizable attempt made to write code to the specifications of the assignment?
Manual evaluation of docstrings	4	Do the docstrings contain the expected information? Do they conform to the expected format?

Academic integrity

This assignment is to be done by you individually, without outside help of any kind (including, but not limited to, help from classmates, tutors, or the internet, including help websites). Disseminating these instructions in whole or in part without written permission of the instructor is considered an infraction of academic integrity.

1. You are not required to use `between()` in your solution. There are other ways to accomplish the same thing.

Last updated 2021-04-28 13:13:56 -0400