30538 Problem Set 2: Parking Tickets Solutions

Peter Ganong, Maggie Shi, and Ozzy Houck

2024-09-30

1. **PS2:** Due Sat Oct 19 at 5:00PM Central. Worth 100 points.

We use (*) to indicate a problem that we think might be time consuming.

Steps to submit (10 points on PS2)

1. "This submission is my work alone and complies with the 30538 integrity policy." Add your initials to indicate your agreement: **___**
2. "I have uploaded the names of anyone I worked with on the problem set **here**" **___** (2 point)
3. Late coins used this pset: **___** Late coins left after submission: **___**
4. Knit your `ps2.qmd` as an html document and print to a pdf to make `ps2.pdf`.
    - The PDF should not be more than 25 pages. Use `head()` and re-size figures when appropriate.
5. Push `ps2.qmd` and `ps2.pdf` to your github repo. It is fine to use Github Desktop.
6. Submit `ps2.pdf` via Gradescope (8 points)
7. Tag your submission in Gradescope

Background Recap

Read **this** article and **this** shorter article. If you are curious to learn more, **this** page has all of the articles that ProPublica has done on this topic. This problem set is a continuation of PS1 using the same data. Please start by loading the data in the same way as PS1.

```
import pandas as pd
import altair as alt
import unittest
```

```
import time

import warnings
warnings.filterwarnings('ignore')
alt.renderers.enable("png")
```

```
RendererRegistry.enable('png')
```

```
# Load the data in the same way as ps1
def read_parking_tickets(file_path):
    start_time = time.time()
    df = pd.read_csv(file_path)
    end_time = time.time()
    elapsed_time = end_time - start_time
    assert len(df) == 287458, "The number of rows is not as expected."
    print(f"Time taken to read the file: {elapsed_time:.2f} seconds")
    df['issue_date'] = pd.to_datetime(df['issue_date'])
    return df

# Example usage
file_path = 'data/parking_tickets_one_percent.csv'
df = read_parking_tickets(file_path)

# make issue_date a datetime
df['issue_date'] = pd.to_datetime(df['issue_date'])
```

```
Time taken to read the file: 2.84 seconds
```

Data cleaning continued (15 points)

1. For each column, how many rows are `NA`? Write a function which returns a two column data frame where each row is a variable, the first column of the data frame is the name of each variable, and the second column of the data frame is the number of times that the column is NA. Test your function. Then, report the results applied to the parking tickets data frame. There are several ways to do this, but we haven't covered them yet in class, so you will need to work independently to set this up.

```
# adding typing to the function signature for clarity
# and to demonstrate how to use type hints
from pandas import DataFrame, Series
def get_number_of_NAs(df: DataFrame) -> Series:
```

```
    """
    Takes in a pandas dataframe and returns a pandas series with the
    number of missing values in each column."""

    na_df = df.isna().sum()

    return na_df

# multiple ways to do this does not need to be a full unittest class
import numpy as np # used for nan values can use other methods
class TestGetNumberOfNAs(unittest.TestCase):
    def test_mixed_nan_values(self):
        data = {
            'A': [1, 2, np.nan, 4, 5],
            'B': [np.nan, 2, 3, np.nan, 5],
            'C': [1, 2, 3, 4, 5],
            'D': [np.nan, np.nan, np.nan, np.nan, np.nan]
        }
        df = pd.DataFrame(data)
        result = get_number_of_NAs(df)
        expected = pd.Series({'A': 1, 'B': 2, 'C': 0, 'D': 5})
        pd.testing.assert_series_equal(result, expected)

    def test_empty_dataframe(self):
        empty_df = pd.DataFrame()
        result = get_number_of_NAs(empty_df)
        self.assertTrue(result.empty)

    def test_no_nan_values(self):
        no_nan_df = pd.DataFrame({'X': [1, 2, 3], 'Y': [4, 5, 6]})
        result = get_number_of_NAs(no_nan_df)
        expected = pd.Series({'X': 0, 'Y': 0})
        pd.testing.assert_series_equal(result, expected)

# Run the tests
test_suite = unittest.TestLoader().loadTestsFromTestCase(TestGetNumberOfNAs)
test_runner = unittest.TextTestRunner(verbosity=2)
test_result = test_runner.run(test_suite)

# Print summary
print(f"Failures: {len(test_result.failures)}")
print(f"Errors: {len(test_result.errors)}")
```

```
test_empty_dataframe (__main__.TestGetNumberOfNAs) ... ok
test_mixed_nan_values (__main__.TestGetNumberOfNAs) ... ok
test_no_nan_values (__main__.TestGetNumberOfNAs) ...

Failures: 0
Errors: 0

ok

----------------------------------------------------------------------
Ran 3 tests in 0.012s

OK
```

```
# Run the function on the parking tickets data
na_df = get_number_of_NAs(df)
print("Number of missing values in each column:")
print(na_df)
```

```
Number of missing values in each column:
Unnamed: 0                   0
ticket_number                0
issue_date                   0
violation_location           0
license_plate_number         0
license_plate_state         97
license_plate_type        2054
zipcode                  54115
violation_code               0
violation_description        0
unit                        29
unit_description             0
vehicle_make                 0
fine_level1_amount           0
fine_level2_amount           0
current_amount_due           0
total_payments               0
ticket_queue                 0
ticket_queue_date            0
notice_level             84068
hearing_disposition     259899
notice_number                0
officer                      0
```

```
address                       0
dtype: int64
```

2. Three variables are missing much more frequently than the others. Why? (Hint: look at some rows and read the data dictionary written by ProPublica)

- **Solution**: The three columns with the most missing values are `zipcode`, `hearing_disposition`, and `notice_level`. ZIP is matched to a car registration and so is missing if there is no car regestration. Notice level is messing if no notice was sent and hearing disposition is missing if there was no hearing

3. Some of the other articles on the propublica website discuss an increase in the dollar amount of the ticket for not having a city sticker. What was the old violation code and what is the new violation code?

```
# Keep only the rows with violation descriptions that contain 'STICKER'
sticker_violations = df[df['violation_description'].str.contains('STICKER')]
print("All descriptions containing 'STICKER':")
print(sticker_violations['violation_description'].unique())

# drop rows for over 16,000 pounds
sticker_violations =
 ↪  sticker_violations[~sticker_violations['violation_description'].str.contains('OVER
 ↪  16,000 LBS.')]

# drop rows for improper display of city sticker
sticker_violations =
 ↪  sticker_violations[~sticker_violations['violation_description'].str.contains('IMPROPER
 ↪  DISPLAY OF CITY STICKER')]

print("Remaining descriptions containing 'STICKER':")
print(sticker_violations['violation_description'].unique())

grouped = sticker_violations.groupby(['violation_description',
 ↪  'violation_code']).size().reset_index(name='n')
print("Comparing violation description and violation code:")
print(grouped)

# take the second index because the first is just the start of the data
date_of_change =
 ↪  sticker_violations.groupby('violation_description')['issue_date'].min().iloc[1]
```

```
print("Date the fine changed:")
print(date_of_change)
```

```
All descriptions containing 'STICKER':
['NO CITY STICKER OR IMPROPER DISPLAY'
 'NO CITY STICKER VEHICLE UNDER/EQUAL TO 16,000 LBS.'
 'NO CITY STICKER VEHICLE OVER 16,000 LBS.'
 'IMPROPER DISPLAY OF CITY STICKER']
Remaining descriptions containing 'STICKER':
['NO CITY STICKER OR IMPROPER DISPLAY'
 'NO CITY STICKER VEHICLE UNDER/EQUAL TO 16,000 LBS.']
Comparing violation description and violation code:
                              violation_description violation_code      n
0                NO CITY STICKER OR IMPROPER DISPLAY        0964125  10758
1                NO CITY STICKER OR IMPROPER DISPLAY        0976170     15
2  NO CITY STICKER VEHICLE UNDER/EQUAL TO 16,000 ...       0964125B  14246
Date the fine changed:
2012-02-25 02:00:00
```

- **Solution:** From the output above, we can see the three violation codes that are associated with not having a city sticker. While there are three, one has very few observations and so the focus of this question is on the other two. We can see that the code changed if Feb 2012 from "0964125" to "0964125B". We will also give you full credit if you discuss the very rare codes "0964125C" and "0964125D".

4. How much was the cost of an initial offense under each code? (You can ignore the ticket for a missing city sticker on vehicles over 16,000 pounds.)

```
# use output from table above
sticker_codes = ["0964125", "0976170", "0964125B"]

# get fine amount for first violation by violation code
grouped = (

 ↪  sticker_violations[sticker_violations["violation_code"].isin(sticker_codes)]
    .groupby('violation_code')['fine_level1_amount']
    .mean()
)
print("Fine amount by code:")
print(grouped)
```