Data Analysis with Python
Project - 1

- Traffic Police Stops

¶

Before beginning your analysis, it is critical that you first examine and clean the dataset, to make working with it a more efficient process. You will practice fixing data types, handling missing values, and dropping columns and rows while learning about the Stanford Open Policing Project dataset.

Examining the dataset

You'll be analyzing a dataset of traffic stops in Rhode Island that was collected by the Stanford Open Policing Project.

Before beginning your analysis, it's important that you familiarize yourself with the dataset. You'll read the dataset into pandas, examine the first few rows, and then count the number of missing values.

INSTRUCTIONS

- Import pandas using the alias `pd` .
- Read the file police.csv into a DataFrame named `ri`
- Examine the first 5 rows of the `DataFrame` (known as the `"head"`).
- Count the number of missing values in each column: Use `.isnull()` to check which `DataFrame` elements are missing, and then take the `.sum()` to count the number of `True` values in each column.

In [1]:

```
import pandas as pd
```

In [2]:

```
ri = pd.read_csv("police.csv.zip")
ri.head()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py:
3444: DtypeWarning: Columns (8,16) have mixed types.Specify dtype option on
import or set low_memory=False.
  exec(code_obj, self.user_global_ns, self.user_ns)
```

Out[2]:

	id	state	stop_date	stop_time	location_raw	county_name	county_fips	fine_grained_lo
0	RI-2005-00001	RI	2005-01-02	01:55	Zone K1	NaN	NaN	
1	RI-2005-00002	RI	2005-01-02	20:30	Zone X4	NaN	NaN	
2	RI-2005-00003	RI	2005-01-04	11:30	Zone X1	NaN	NaN	
3	RI-2005-00004	RI	2005-01-04	12:55	Zone X4	NaN	NaN	
4	RI-2005-00005	RI	2005-01-06	01:30	Zone X4	NaN	NaN	

5 rows × 26 columns

In [3]:

```
ri.isnull()
```

Out[3]:

	id	state	stop_date	stop_time	location_raw	county_name	county_fips	fine_grain
0	False	False	False	False	False	True	True	
1	False	False	False	False	False	True	True	
2	False	False	False	False	False	True	True	
3	False	False	False	False	False	True	True	
4	False	False	False	False	False	True	True	
...	...	...	...	...	...	...	...	
509676	False	False	True	True	False	True	True	
509677	False	False	True	True	False	True	True	
509678	False	False	True	True	False	True	True	
509679	False	False	True	True	False	True	True	
509680	False	False	True	True	False	True	True	

509681 rows × 26 columns

In [4]:

```
ri.isnull().sum()
```

Out[4]:

```
id                         0
state                      0
stop_date                 10
stop_time                 10
location_raw               0
county_name           509681
county_fips           509681
fine_grained_location 509681
police_department         10
driver_gender          29097
driver_age_raw         29049
driver_age             30695
driver_race_raw        29073
driver_race            29073
violation_raw          29073
violation              29073
search_conducted          10
search_type_raw       491919
search_type           491919
contraband_found           0
stop_outcome           29073
is_arrested            29073
stop_duration          29073
out_of_state           29881
drugs_related_stop         0
district                   0
dtype: int64
```

Dropping columns

Often, a DataFrame will contain columns that are not useful to your analysis. Such columns should be dropped from the `DataFrame` , to make it easier for you to focus on the remaining columns.

You'll drop the `county_name` column because it only contains missing values, and you'll drop the `state` column because all of the traffic stops took place in one state (Rhode Island). Thus, these columns can be dropped because they contain no useful information.

INSTRUCTIONS

- Examine the `DataFrame` 's shape to find out the number of rows and columns.
- Drop the columns that almost consist of missing values.
- Examine the `.shape` again to verify that there are now two fewer columns.

In [5]:

```
ri.drop(["county_name", "county_fips", "fine_grained_location"], axis = 1, inplace=True)
```

In [6]:

```
ri.shape
```

Out[6]:

```
(509681, 23)
```

Dropping rows

When you know that a specific column will be critical to your analysis, and only a small fraction of rows are missing a value in that column, it often makes sense to remove those rows from the dataset.

During this course, the `driver_gender` column will be critical to many of your analyses. Because only a small fraction of rows are missing `driver_gender` , we'll drop those rows from the dataset.

INSTRUCTIONS

- Count the number of missing values in each column.
- Drop all rows that are missing `driver_gender` by passing the column name to the subset parameter of `.dropna()` .
- Count the number of missing values in each column again, to verify that none of the remaining rows are missing `driver_gender` .
- Examine the `DataFrame` 's `.shape` to see how many rows and columns remain.

In [7]:

```
ri.isnull().sum()
```

Out[7]:

```
id                        0
state                     0
stop_date                10
stop_time                10
location_raw              0
police_department        10
driver_gender         29097
driver_age_raw        29049
driver_age            30695
driver_race_raw       29073
driver_race           29073
violation_raw         29073
violation             29073
search_conducted         10
search_type_raw      491919
search_type          491919
contraband_found          0
stop_outcome          29073
is_arrested           29073
stop_duration         29073
out_of_state          29881
drugs_related_stop        0
district                  0
dtype: int64
```

In [8]:

```
ri.dropna(subset= ["driver_gender"], inplace = True)
```

In [9]:

```
ri.isnull().sum()
```

Out[9]:

```
id                        0
state                     0
stop_date                 0
stop_time                 0
location_raw              0
police_department         0
driver_gender             0
driver_age_raw            1
driver_age             1638
driver_race_raw           0
driver_race               0
violation_raw             0
violation                 0
search_conducted          0
search_type_raw      462822
search_type          462822
contraband_found          0
stop_outcome              0
is_arrested               0
stop_duration             0
out_of_state            808
drugs_related_stop        0
district                  0
dtype: int64
```

In [10]:

```
ri.shape
```

Out[10]:

```
(480584, 23)
```

Fixing a data type

We know that the `is_arrested` column currently has the `object` data type. In this exercise, we'll change the data type to `bool` , which is the most suitable type for a column containing `True` and `False` values.

Fixing the data type will enable us to use mathematical operations on the `is_arrested` column that would not be possible otherwise.

INSTRUCTIONS

- Examine the head of the `is_arrested` column to verify that it contains `True` and `False` values.
- Check the current data type of `is_arrested` .
- Use the `.astype()` method to convert `is_arrested` to a `bool` column.
- Check the new data type of `is_arrested` , to confirm that it is now a `bool` column.

In [11]:

```
ri.dtypes
```

Out[11]:

```
id                     object
state                  object
stop_date              object
stop_time              object
location_raw           object
police_department      object
driver_gender          object
driver_age_raw        float64
driver_age            float64
driver_race_raw        object
driver_race            object
violation_raw          object
violation              object
search_conducted       object
search_type_raw        object
search_type            object
contraband_found         bool
stop_outcome           object
is_arrested            object
stop_duration          object
out_of_state           object
drugs_related_stop       bool
district               object
dtype: object
```

In [12]:

```
ri.is_arrested.head()
```

Out[12]:

```
0    False
1    False
3    False
4    False
5    False
Name: is_arrested, dtype: object
```

In [13]:

```
ri.is_arrested.value_counts(dropna = False)
```

Out[13]:

```
False    463981
True      16603
Name: is_arrested, dtype: int64
```

In [14]:

```
ri["is_arrested"] = ri.is_arrested.astype("bool")
```

In [19]:

```
print(ri.is_arrested.dtype)
```

```
bool
```

Combining object columns

Currently, the date and time of each traffic stop are stored in separate object columns: `stop_date` and `stop_time`.

You'll combine these two columns into a single column, and then convert it to `datetime` format. This will enable convenient date-based attributes that we'll use later in the course.

INSTRUCTIONS

- Use a string method to concatenate `stop_date` and `stop_time` (separated by a space), and store the result in `combined`.
- Convert `combined` to `datetime` format, and store the result in a new column named `stop_datetime`.
- Examine the `DataFrame` `.dtypes` to confirm that `stop_datetime` is a datetime column.

In [20]:

```
combined = ri.stop_date.str.cat(ri.stop_time, sep = " ")
```

In [21]:

```
type(ri.stop_date)
```

Out[21]:

```
pandas.core.series.Series
```

In [23]:

```
ri["stop_datetime"] = pd.to_datetime(combined)
```

In [25]:

```
ri.drop(["stop_date", "stop_time"], axis = 1, inplace=True)
```

In [26]:

```
ri.head()
```

Out[26]:

	id	state	location_raw	police_department	driver_gender	driver_age_raw	driver_age	dr
0	RI-2005-00001	RI	Zone K1	600	M	1985.0	20.0	
1	RI-2005-00002	RI	Zone X4	500	M	1987.0	18.0	
3	RI-2005-00004	RI	Zone X4	500	M	1986.0	19.0	
4	RI-2005-00005	RI	Zone X4	500	M	1978.0	27.0	
5	RI-2005-00006	RI	Zone X1	0	M	1973.0	32.0	

5 rows × 22 columns

The last step that you'll take in this chapter is to set the `stop_datetime` column as the `DataFrame` 's index. By replacing the default index with a `DatetimeIndex` , you'll make it easier to analyze the dataset by date and time, which will come in handy later in the course.

INSTRUCTIONS

- Set `stop_datetime` as the `DataFrame` index.
- Examine the index to verify that it is a `DatetimeIndex` .
- Examine the `DataFrame` columns to confirm that `stop_datetime` is no longer one of the columns.

In [27]:

```
ri.set_index("stop_datetime", inplace = True)
```

In [28]:

```
ri.head()
```

Out[28]:

	id	state	location_raw	police_department	driver_gender	driver_age_raw	dr
stop_datetime							
2005-01-02 01:55:00	RI-2005-00001	RI	Zone K1	600	M	1985.0	
2005-01-02 20:30:00	RI-2005-00002	RI	Zone X4	500	M	1987.0	
2005-01-04 12:55:00	RI-2005-00004	RI	Zone X4	500	M	1986.0	
2005-01-06 01:30:00	RI-2005-00005	RI	Zone X4	500	M	1978.0	
2005-01-12 08:05:00	RI-2005-00006	RI	Zone X1	0	M	1973.0	

5 rows × 21 columns

In [ ]:

```

```

In [ ]:

```

```

Data Analysis with Python
Project - 1

- Traffic Police Stops

Does the `gender` of a driver have an impact on police behavior during a traffic stop? **In this chapter**, you will explore that question while practicing filtering, grouping, method chaining, Boolean math, string methods, and more!

Examining traffic violations

Before comparing the violations being committed by each gender, you should examine the `violations` committed by all drivers to get a baseline understanding of the data.

In this exercise, you'll count the unique values in the `violation` column, and then separately express those counts as proportions.

> Before starting your work in this section **repeat the steps which you did in the previos chapter for preparing the data.** Continue to this chapter based on where you were in the end of the previous chapter.

In [1]:

```
import pandas as pd
ri = pd.read_csv("police.csv.zip")

ri.drop(["county_name", "county_fips", "fine_grained_location"], axis = 1, inplace=True)

ri.dropna(subset= ["driver_gender"], inplace = True)

ri["is_arrested"] = ri.is_arrested.astype("bool")

combined = ri.stop_date.str.cat(ri.stop_time, sep = " ")

ri["stop_datetime"] = pd.to_datetime(combined)

ri.drop(["stop_date", "stop_time"], axis = 1, inplace=True)

ri.set_index("stop_datetime", inplace = True)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py:
3444: DtypeWarning: Columns (8,16) have mixed types.Specify dtype option on
import or set low_memory=False.
  exec(code_obj, self.user_global_ns, self.user_ns)
```

INSTRUCTIONS

- Count the unique values in the `violation` column, to see what violations are being committed by all drivers.
- Express the violation counts as proportions of the total.

In [2]:

```
ri.head()
```

Out[2]:

	id	state	location_raw	police_department	driver_gender	driver_age_raw	dr
stop_datetime							
2005-01-02 01:55:00	RI-2005-00001	RI	Zone K1	600	M	1985.0	
2005-01-02 20:30:00	RI-2005-00002	RI	Zone X4	500	M	1987.0	
2005-01-04 12:55:00	RI-2005-00004	RI	Zone X4	500	M	1986.0	
2005-01-06 01:30:00	RI-2005-00005	RI	Zone X4	500	M	1978.0	
2005-01-12 08:05:00	RI-2005-00006	RI	Zone X1	0	M	1973.0	

5 rows × 21 columns

In [3]:

```
ri.violation.value_counts(dropna = False)
```

Out[3]:

```
Speeding              268736
Moving violation       90228
Equipment              61250
Other                  24216
Registration/plates    19830
Seat belt              16324
Name: violation, dtype: int64
```

Comparing violations by gender

The question we're trying to answer is whether male and female drivers tend to commit different types of traffic violations.

You'll first create a `DataFrame` for each gender, and then analyze the `violations` in each `DataFrame` separately.

INSTRUCTIONS

- Create a `DataFrame` , female, that only contains rows in which `driver_gender` is `'F'` .
- Create a `DataFrame` , male, that only contains rows in which `driver_gender` is `'M'` .
- Count the `violations` committed by female drivers and express them as proportions.
- Count the violations committed by male drivers and express them as proportions.

In [4]:

```
ri["driver_gender"].value_counts(dropna = False)
```

Out[4]:

```
M    349446
F    131138
Name: driver_gender, dtype: int64
```

In [5]:

```
famele = ri[ri.driver_gender == "F"]

male = ri[ri.driver_gender == "M"]
```

In [6]:

```
famele.head()
```

Out[6]:

	id	state	location_raw	police_department	driver_gender	driver_age_raw	dr
stop_datetime							
2005-02-24 01:20:00	RI-2005-00016	RI	Zone X3	200	F	1983.0	
2005-03-14 10:00:00	RI-2005-00019	RI	Zone K3	300	F	1984.0	
2005-03-29 23:20:00	RI-2005-00026	RI	Zone K3	300	F	1971.0	
2005-06-06 13:20:00	RI-2005-00035	RI	Zone X4	500	F	1986.0	
2005-06-18 16:30:00	RI-2005-00037	RI	Zone X4	500	F	1964.0	

5 rows × 21 columns

In [7]:

```
male.head()
```

Out[7]:

	id	state	location_raw	police_department	driver_gender	driver_age_raw	dr
stop_datetime							
2005-01-02 01:55:00	RI-2005-00001	RI	Zone K1	600	M	1985.0	
2005-01-02 20:30:00	RI-2005-00002	RI	Zone X4	500	M	1987.0	
2005-01-04 12:55:00	RI-2005-00004	RI	Zone X4	500	M	1986.0	
2005-01-06 01:30:00	RI-2005-00005	RI	Zone X4	500	M	1978.0	
2005-01-12 08:05:00	RI-2005-00006	RI	Zone X1	0	M	1973.0	

5 rows × 21 columns

In [8]:

```
famele.violation.value_counts(normalize = True)*100
```

Out[8]:

```
Speeding              65.730757
Moving violation      13.658131
Equipment             10.705516
Registration/plates    4.307676
Other                  2.890848
Seat belt              2.707072
Name: violation, dtype: float64
```

In [9]:

```
male.violation.value_counts(normalize = True)*100
```

Out[9]:

```
Speeding              52.236397
Moving violation      20.694757
Equipment             13.510242
Other                  5.844966
Registration/plates    4.058138
Seat belt              3.655500
Name: violation, dtype: float64
```

In [10]:

```
ri.groupby(ri.driver_gender)
```

Out[10]:

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x00000220E30B76A0>
```

Comparing speeding outcomes by gender

When a driver is pulled over for speeding, many people believe that gender has an impact on whether the driver will receive a ticket or a warning. Can you find evidence of this in the dataset?

First, you'll create two `DataFrames` of drivers who were stopped for `speeding` : one containing ***females*** and the other containing ***males***.

Then, for each **gender**, you'll use the `stop_outcome` column to calculate what percentage of stops resulted in a `"Citation"` (meaning a ticket) versus a `"Warning"` .

INSTRUCTIONS

- Create a `DataFrame` , `female_and_speeding` , that only includes female drivers who were stopped for speeding.
- Create a `DataFrame` , `male_and_speeding` , that only includes male drivers who were stopped for speeding.
- Count the **stop outcomes** for the female drivers and express them as proportions.

- Count the **stop outcomes** for the male drivers and express them as proportions.

In [11]:

```
famale_and_speeding = ri[(ri.driver_gender == "F") & (ri.violation == "Speeding")]

male_and_speeding = ri[(ri.driver_gender == "M") & (ri.violation == "Speeding")]
```

In [12]:

```
famale_and_speeding.head()
```

Out[12]:

	id	state	location_raw	police_department	driver_gender	driver_age_raw	dr
stop_datetime							
2005-02-24 01:20:00	RI-2005-00016	RI	Zone X3	200	F	1983.0	
2005-03-14 10:00:00	RI-2005-00019	RI	Zone K3	300	F	1984.0	
2005-03-29 23:20:00	RI-2005-00026	RI	Zone K3	300	F	1971.0	
2005-06-06 13:20:00	RI-2005-00035	RI	Zone X4	500	F	1986.0	
2005-07-06 11:22:00	RI-2005-00038	RI	Zone X1	0	F	1973.0	

5 rows × 21 columns

In [13]:

```
famale_and_speeding.head()
```

Out[13]:

	id	state	location_raw	police_department	driver_gender	driver_age_raw	dr
stop_datetime							
2005-02-24 01:20:00	RI-2005-00016	RI	Zone X3	200	F	1983.0	
2005-03-14 10:00:00	RI-2005-00019	RI	Zone K3	300	F	1984.0	
2005-03-29 23:20:00	RI-2005-00026	RI	Zone K3	300	F	1971.0	
2005-06-06 13:20:00	RI-2005-00035	RI	Zone X4	500	F	1986.0	
2005-07-06 11:22:00	RI-2005-00038	RI	Zone X1	0	F	1973.0	

5 rows × 21 columns

In [14]:

```
ri.stop_outcome.value_counts(dropna = False)
```

Out[14]:

```
Citation            428378
Warning              28840
Arrest Driver        14630
N/D                   3431
No Action             3332
Arrest Passenger      1973
Name: stop_outcome, dtype: int64
```

In [15]:

```
print(famale_and_speeding.stop_outcome.value_counts(normalize = True))
```

```
Citation            0.953247
Warning             0.039003
Arrest Driver       0.005290
Arrest Passenger    0.001033
N/D                 0.000905
No Action           0.000522
Name: stop_outcome, dtype: float64
```

In [16]:

```
print(male_and_speeding.stop_outcome.value_counts(normalize = True))
```

```
Citation            0.944636
Warning             0.036086
Arrest Driver       0.015767
Arrest Passenger    0.001265
N/D                 0.001183
No Action           0.001063
Name: stop_outcome, dtype: float64
```

Calculating the search rate

During a traffic stop, the police officer sometimes conducts a search of the vehicle. In this exercise, you'll calculate the percentage of all stops that result in a vehicle search, also known as the **search rate**.

INSTRUCTIONS

- Check the data type of `search_conducted` to confirm that it's a `Boolean Series`.
- Calculate the search rate by counting the `Series` values and expressing them as proportions.
- Calculate the search rate by taking the mean of the `Series`. (It should match the proportion of `True` values calculated above.)

In [17]:

```
ri["search_conducted"] = ri.search_conducted.astype("bool")
print(ri.search_conducted.dtype)
```

```
bool
```

In [18]:

```
ri.search_conducted.value_counts(normalize = True)
```

Out[18]:

```
False    0.963041
True     0.036959
Name: search_conducted, dtype: float64
```

In [19]:

```
ri.search_conducted.mean()
```

Out[19]:

```
0.036959199640437465
```

Comparing search rates by gender

You'll compare the rates at which **female** and **male** drivers are searched during a traffic stop. Remember that the vehicle search rate across all stops is about **3.8%**.

First, you'll filter the `DataFrame` by gender and calculate the search rate for each group separately. Then, you'll perform the same calculation for both genders at once using a `.groupby()`.

INSTRUCTIONS 1/3

- Filter the `DataFrame` to only include **female** drivers, and then calculate the search rate by taking the mean of `search_conducted`.

In [20]:

```
ri[ri.driver_gender == "F"].search_conducted.mean()
```

Out[20]:

```
0.018751239152648355
```

INSTRUCTIONS 2/3

- Filter the `DataFrame` to only include **male** drivers, and then repeat the search rate calculation.

In [21]:

```
ri[ri.driver_gender == "M"].search_conducted.mean()
```

Out[21]:

```
0.04379217389811301
```

INSTRUCTIONS 3/3

- Group by driver gender to calculate the search rate for both groups simultaneously. (It should match the previous results.)

In [22]:

```
ri.groupby("driver_gender")["search_conducted"].mean()
```

Out[22]:

```
driver_gender
F    0.018751
M    0.043792
Name: search_conducted, dtype: float64
```

Adding a second factor to the analysis

Even though the search rate for males is much higher than for females, it's possible that the difference is mostly due to a second factor.

For example, you might hypothesize that the search rate varies by violation type, and the difference in search rate between males and females is because they tend to commit different violations.

You can test this hypothesis by examining the search rate for each combination of gender and violation. If the hypothesis was true, you would find that males and females are searched at about the same rate for each violation. Find out below if that's the case!

INSTRUCTIONS 1/2

- Use a `.groupby()` to calculate the search rate for each combination of gender and violation. Are males and females searched at about the same rate for each violation?

In [23]:

```
ri.groupby(["driver_gender", "violation"]).search_conducted.mean()
```

Out[23]:

```
driver_gender  violation
F              Equipment              0.040245
               Moving violation       0.038021
               Other                  0.045898
               Registration/plates    0.054700
               Seat belt              0.017746
               Speeding               0.007738
M              Equipment              0.070916
               Moving violation       0.059156
               Other                  0.046120
               Registration/plates    0.103589
               Seat belt              0.031705
               Speeding               0.026630
Name: search_conducted, dtype: float64
```

INSTRUCTIONS 2/2

- Reverse the ordering to group by violation before gender. The results may be easier to compare when presented this way.

In [24]:

```
ri.groupby(["violation", "driver_gender"]).search_conducted.mean()
```

Out[24]:

```
violation            driver_gender
Equipment            F                0.040245
                     M                0.070916
Moving violation     F                0.038021
                     M                0.059156
Other                F                0.045898
                     M                0.046120
Registration/plates  F                0.054700
                     M                0.103589
Seat belt            F                0.017746
                     M                0.031705
Speeding             F                0.007738
                     M                0.026630
Name: search_conducted, dtype: float64
```

Counting protective frisks

During a vehicle search, the police officer may pat down the driver to check if they have a weapon. This is known as a `"protective frisk."`

You'll first check to see how many times "Protective Frisk" was the only search type. Then, you'll use a string method to locate all instances in which the driver was frisked.

INSTRUCTIONS

- Count the `search_type` values to see how many times `"Protective Frisk"` was the only search type.
- Create a new column, frisk, that is `True` if `search_type` contains the string `"Protective Frisk"` and `False` otherwise.
- Check the data type of frisk to confirm that it's a `Boolean Series`.
- Take the sum of frisk to count the total number of frisks.

In [25]:

```
ri.search_type.value_counts(dropna = False)
```

Out[25]:

```
NaN                                                        462822
Incident to Arrest                                           6998
Probable Cause                                               4989
Reasonable Suspicion                                         1141
Inventory                                                    1101
Protective Frisk                                              879
Incident to Arrest,Inventory                                  649
Incident to Arrest,Probable Cause                             552
Probable Cause,Reasonable Suspicion                           334
Probable Cause,Protective Frisk                               221
Incident to Arrest,Protective Frisk                           158
Incident to Arrest,Inventory,Probable Cause                   151
Inventory,Probable Cause                                      132
Protective Frisk,Reasonable Suspicion                          83
Incident to Arrest,Inventory,Protective Frisk                  77
Incident to Arrest,Probable Cause,Protective Frisk             74
Inventory,Protective Frisk                                     52
Incident to Arrest,Reasonable Suspicion                        49
Probable Cause,Protective Frisk,Reasonable Suspicion           31
Incident to Arrest,Probable Cause,Reasonable Suspicion         31
Inventory,Reasonable Suspicion                                 17
Inventory,Probable Cause,Protective Frisk                      14
Incident to Arrest,Inventory,Reasonable Suspicion              12
Incident to Arrest,Protective Frisk,Reasonable Suspicion        8
Inventory,Probable Cause,Reasonable Suspicion                   8
Inventory,Protective Frisk,Reasonable Suspicion                 1
Name: search_type, dtype: int64
```

In [26]:

```
ri["Protective_Frisk"] = ri.search_type.isnull()
```

In [27]:

```
ri.head()
```

Out[27]:

	id	state	location_raw	police_department	driver_gender	driver_age_raw	dr
stop_datetime							
2005-01-02 01:55:00	RI-2005-00001	RI	Zone K1	600	M	1985.0	
2005-01-02 20:30:00	RI-2005-00002	RI	Zone X4	500	M	1987.0	
2005-01-04 12:55:00	RI-2005-00004	RI	Zone X4	500	M	1986.0	
2005-01-06 01:30:00	RI-2005-00005	RI	Zone X4	500	M	1978.0	
2005-01-12 08:05:00	RI-2005-00006	RI	Zone X1	0	M	1973.0	

5 rows × 22 columns

In [28]:

```
print(ri.Protective_Frisk.dtype)
```

```
bool
```

In [29]:

```
ri.Protective_Frisk.sum()
```

Out[29]:

```
462822
```

Comparing frisk rates by gender

You'll compare the rates at which female and male drivers are frisked during a search. Are males frisked more often than females, perhaps because police officers consider them to be higher risk?

Before doing any calculations, it's important to filter the `DataFrame` to only include the relevant subset of data, namely stops in which a search was conducted.

INSTRUCTIONS

- Create a `DataFrame` , searched, that only contains rows in which `search_conducted` is `True` .
- Take the mean of the frisk column to find out what percentage of searches included a frisk.
- Calculate the frisk rate for each gender using a `.groupby()` .

In [30]:

```
ri.search_conducted.value_counts(dropna = False)
```

Out[30]:

```
False    462822
True      17762
Name: search_conducted, dtype: int64
```

In [38]:

```
ri[(ri.search_conducted == True)].head()
```

Out[38]:

	id	state	location_raw	police_department	driver_gender	driver_age_raw	dr
stop_datetime							
2005-01-24 20:32:00	RI-2005-00010	RI	Zone K1	600	M	1987.0	
2005-02-09 03:05:00	RI-2005-00011	RI	Zone X4	500	M	1976.0	
2005-08-28 01:00:00	RI-2005-00084	RI	Zone X1	0	M	1979.0	
2005-09-15 02:20:00	RI-2005-00094	RI	Zone X4	500	M	1988.0	
2005-09-24 02:20:00	RI-2005-00115	RI	Zone K3	300	M	1987.0	

5 rows × 22 columns

In [31]:

```
ri[ri.Protective_Frisk == True].mean()
```

```
C:\Users\User\AppData\Local\Temp/ipykernel_44852/1967029035.py:1: FutureWarn
ing: Dropping of nuisance columns in DataFrame reductions (with 'numeric_onl
y=None') is deprecated; in a future version this will raise TypeError.  Sele
ct only valid columns before calling the reduction.
  ri[ri.Protective_Frisk == True].mean()
```

Out[31]:

```
driver_age_raw       1970.239767
driver_age             34.109658
search_conducted        0.000000
search_type_raw              NaN
search_type                  NaN
contraband_found        0.000000
is_arrested             0.022505
out_of_state            0.333379
drugs_related_stop      0.000000
Protective_Frisk        1.000000
dtype: float64
```

In [32]:

```
ri.groupby(["driver_gender", "search_type"]).Protective_Frisk.mean()
```

Out[32]:

```
driver_gender  search_type
F              Incident to Arrest
0.0
               Incident to Arrest,Inventory
0.0
               Incident to Arrest,Inventory,Probable Cause
0.0
               Incident to Arrest,Inventory,Protective Frisk
0.0
               Incident to Arrest,Probable Cause
0.0
               Incident to Arrest,Probable Cause,Protective Frisk
0.0
               Incident to Arrest,Probable Cause,Reasonable Suspicion
0.0
               Incident to Arrest,Protective Frisk
0.0
               Incident to Arrest,Protective Frisk,Reasonable Suspicion
0.0
               Incident to Arrest,Reasonable Suspicion
0.0
               Inventory
0.0
               Inventory,Probable Cause
0.0
               Inventory,Probable Cause,Protective Frisk
0.0
               Inventory,Probable Cause,Reasonable Suspicion
0.0
               Inventory,Protective Frisk
0.0
               Inventory,Protective Frisk,Reasonable Suspicion
0.0
               Probable Cause
0.0
               Probable Cause,Protective Frisk
0.0
               Probable Cause,Protective Frisk,Reasonable Suspicion
0.0
               Probable Cause,Reasonable Suspicion
0.0
               Protective Frisk
0.0
               Protective Frisk,Reasonable Suspicion
0.0
               Reasonable Suspicion
0.0
M              Incident to Arrest
0.0
               Incident to Arrest,Inventory
0.0
               Incident to Arrest,Inventory,Probable Cause
0.0
               Incident to Arrest,Inventory,Protective Frisk
0.0
```

```
                Incident to Arrest,Inventory,Reasonable Suspicion
0.0
                Incident to Arrest,Probable Cause
0.0
                Incident to Arrest,Probable Cause,Protective Frisk
0.0
                Incident to Arrest,Probable Cause,Reasonable Suspicion
0.0
                Incident to Arrest,Protective Frisk
0.0
                Incident to Arrest,Protective Frisk,Reasonable Suspicion
0.0
                Incident to Arrest,Reasonable Suspicion
0.0
                Inventory
0.0
                Inventory,Probable Cause
0.0
                Inventory,Probable Cause,Protective Frisk
0.0
                Inventory,Probable Cause,Reasonable Suspicion
0.0
                Inventory,Protective Frisk
0.0
                Inventory,Reasonable Suspicion
0.0
                Probable Cause
0.0
                Probable Cause,Protective Frisk
0.0
                Probable Cause,Protective Frisk,Reasonable Suspicion
0.0
                Probable Cause,Reasonable Suspicion
0.0
                Protective Frisk
0.0
                Protective Frisk,Reasonable Suspicion
0.0
                Reasonable Suspicion
0.0
Name: Protective_Frisk, dtype: float64
```

In [ ]:

```

```

Data Analysis with Python
Project - 1

¶

- Traffic Police Stops

Are you more likely to get arrested at a certain time of day? Are drug-related stops on the rise? In this chapter, you will answer these and other questions by analyzing the dataset visually, since plots can help you to understand trends in a way that examining the raw data cannot.

Calculating the hourly arrest rate

When a police officer stops a driver, a small percentage of those stops ends in an arrest. This is known as the **arrest rate**. In this exercise, you'll find out whether the arrest rate varies by time of day.

First, you'll calculate the arrest rate across all stops. Then, you'll calculate the **hourly arrest rate** by using the `hour` attribute of the `index`. The hour ranges from `0` to `23`, in which:

```
0 = midnight
12 = noon
23 = 11 PM
```

> Before starting your work in this section **repeat the steps which you did in the first chapter for preparing the data.** Continue to this chapter based on where you were in the end of the first chapter.

INSTRUCTIONS

- Take the mean of the `is_arrested` column to calculate the overall arrest rate.
- Group by the `hour` attribute of the `DataFrame` index to calculate the hourly arrest rate.
- Save the **hourly arrest rate** `Series` as a new object, `hourly_arrest_rate`.

In [1]:

```
import pandas as pd
ri = pd.read_csv("police.csv.zip")

ri.drop(["county_name", "county_fips", "fine_grained_location"], axis = 1, inplace=True)

ri.dropna(subset= ["driver_gender"], inplace = True)

ri["is_arrested"] = ri.is_arrested.astype("bool")

combined = ri.stop_date.str.cat(ri.stop_time, sep = " ")

ri["stop_datetime"] = pd.to_datetime(combined)

ri.drop(["stop_date", "stop_time"], axis = 1, inplace=True)

ri.set_index("stop_datetime", inplace = True)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py:
3444: DtypeWarning: Columns (8,16) have mixed types.Specify dtype option on
import or set low_memory=False.
  exec(code_obj, self.user_global_ns, self.user_ns)
```

In [2]:

```
ri.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 480584 entries, 2005-01-02 01:55:00 to 2015-12-31 23:48:00
Data columns (total 21 columns):
 #   Column              Non-Null Count   Dtype  
---  ------              --------------   -----  
 0   id                  480584 non-null  object 
 1   state               480584 non-null  object 
 2   location_raw        480584 non-null  object 
 3   police_department   480584 non-null  object 
 4   driver_gender       480584 non-null  object 
 5   driver_age_raw      480583 non-null  float64
 6   driver_age          478946 non-null  float64
 7   driver_race_raw     480584 non-null  object 
 8   driver_race         480584 non-null  object 
 9   violation_raw       480584 non-null  object 
 10  violation           480584 non-null  object 
 11  search_conducted    480584 non-null  object 
 12  search_type_raw     17762 non-null   object 
 13  search_type         17762 non-null   object 
 14  contraband_found    480584 non-null  bool   
 15  stop_outcome        480584 non-null  object 
 16  is_arrested         480584 non-null  bool   
 17  stop_duration       480584 non-null  object 
 18  out_of_state        479776 non-null  object 
 19  drugs_related_stop  480584 non-null  bool   
 20  district            480584 non-null  object 
dtypes: bool(3), float64(2), object(16)
memory usage: 71.0+ MB
```

In [3]:

```
ri.is_arrested.mean()
```

Out[3]:

```
0.03454755048024903
```

In [4]:

```
ri.groupby(ri.is_arrested).mean()
```

Out[4]:

	driver_age_raw	driver_age	contraband_found	drugs_related_stop
is_arrested				
False	1970.351338	34.053572	0.009923	0.007511
True	1975.874661	31.980396	0.118954	0.077095

In [5]:

```
ri["is_arrested"].value_counts(dropna = False)
```

Out[5]:

```
False    463981
True      16603
Name: is_arrested, dtype: int64
```

In [6]:

```
ri.index.hour
```

Out[6]:

```
Int64Index([ 1, 20, 12,  1,  8,  8, 17, 23, 20,  3,
            ...
            22, 22, 22, 22, 22, 22, 22, 23, 23, 23],
           dtype='int64', name='stop_datetime', length=480584)
```

In [7]:

```
ri.groupby(ri.index.hour).is_arrested.mean()
```

Out[7]:

```
stop_datetime
0     0.052151
1     0.067127
2     0.061067
3     0.052613
4     0.053897
5     0.032657
6     0.012949
7     0.013829
8     0.019717
9     0.024699
10    0.025583
11    0.027078
12    0.031361
13    0.030250
14    0.031531
15    0.032125
16    0.033519
17    0.038989
18    0.039902
19    0.031366
20    0.039292
21    0.059956
22    0.043980
23    0.045087
Name: is_arrested, dtype: float64
```

In [8]:

```
hourly_arrest_rate = ri.groupby(ri.index.hour).is_arrested.mean()
```

Plotting the hourly arrest rate

You'll create a line plot from the `hourly_arrest_rate` object. A line plot is appropriate in this case because you're showing how a quantity changes over time.

This plot should help you to spot some trends that may not have been obvious when examining the raw numbers!

INSTRUCTIONS

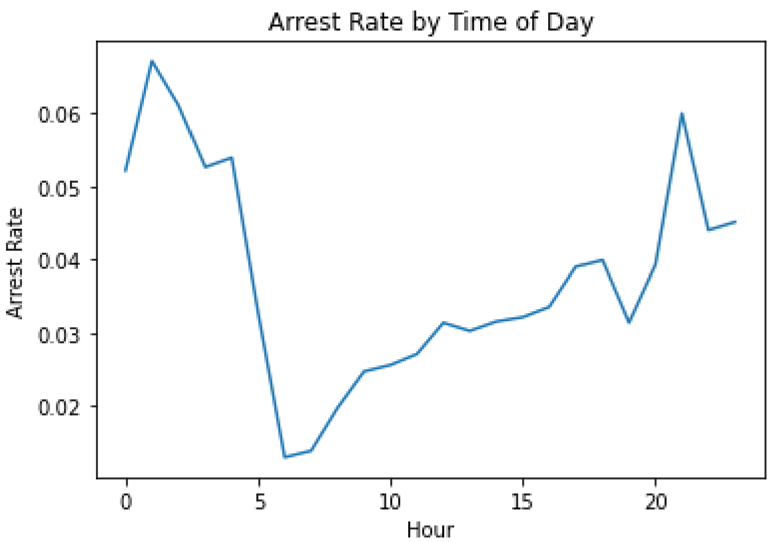
- Import `matplotlib.pyplot` using the alias `plt`.
- Create a **line plot** of `hourly_arrest_rate` using the `.plot()` method.
- Label the `x-axis as 'Hour'`, label the `y-axis as 'Arrest Rate'`, and title the plot `'Arrest Rate by Time of Day'`.
- Display the plot using the `.show()` function.

In [9]:

```
import matplotlib.pyplot as plt
```

In [10]:

```
hourly_arrest_rate.plot()
plt.xlabel('Hour')
plt.ylabel('Arrest Rate')
plt.title('Arrest Rate by Time of Day')
plt.show()
```



Plotting drug-related stops

In a small portion of traffic stops, drugs are found in the vehicle during a search. You'll assess whether these **drug-related stops** are becoming more common over time.

The Boolean column `drugs_related_stop` indicates whether drugs were found during a given stop. You'll calculate the **annual drug rate** by **resampling** this column, and then you'll use a line plot to visualize how the rate has changed over time.

INSTRUCTIONS

- Calculate the **annual rate** of drug-related stops by **resampling** the `drugs_related_stop` column (on the `'A'` frequency) and taking the mean.
- Save the annual drug rate `Series` as a new object, `annual_drug_rate`.
- Create a line plot of `annual_drug_rate` using the `.plot()` method.
- Display the plot using the `.show()` function.

In [11]:

```
ri.drugs_related_stop.mean()
```

Out[11]:

```
0.009915020058928303
```

In [12]:

```
ri.groupby(ri.drugs_related_stop).is_arrested.mean()
```

Out[12]:

```
drugs_related_stop
False    0.032203
True     0.268625
Name: is_arrested, dtype: float64
```

In [13]:

```
annual_drug_rate = ri.drugs_related_stop
```

In [14]:

```
annual_drug_rate
```

Out[14]:

```
stop_datetime
2005-01-02 01:55:00    False
2005-01-02 20:30:00    False
2005-01-04 12:55:00    False
2005-01-06 01:30:00    False
2005-01-12 08:05:00    False
                       ...
2015-12-31 22:46:00    False
2015-12-31 22:47:00    False
2015-12-31 23:08:00    False
2015-12-31 23:44:00    False
2015-12-31 23:48:00    False
Name: drugs_related_stop, Length: 480584, dtype: bool
```

In [15]:

```
ri.head(2)
```

Out[15]:

	id	state	location_raw	police_department	driver_gender	driver_age_raw	dr
stop_datetime							
2005-01-02 01:55:00	RI-2005-00001	RI	Zone K1	600	M	1985.0	
2005-01-02 20:30:00	RI-2005-00002	RI	Zone X4	500	M	1987.0	

2 rows × 21 columns

In [16]:

```
ri.drugs_related_stop.value_counts(dropna = False)
```

Out[16]:

```
False    475819
True       4765
Name: drugs_related_stop, dtype: int64
```

In [17]:

```
ri.groupby(ri.index.year).drugs_related_stop.mean()
```

Out[17]:

```
stop_datetime
2005    0.008038
2006    0.006624
2007    0.008437
2008    0.007549
2009    0.010447
2010    0.010142
2011    0.011400
2012    0.010343
2013    0.011879
2014    0.013176
2015    0.010598
Name: drugs_related_stop, dtype: float64
```

In [18]:

```
ri.drugs_related_stop.resample("A").mean()
```

Out[18]:

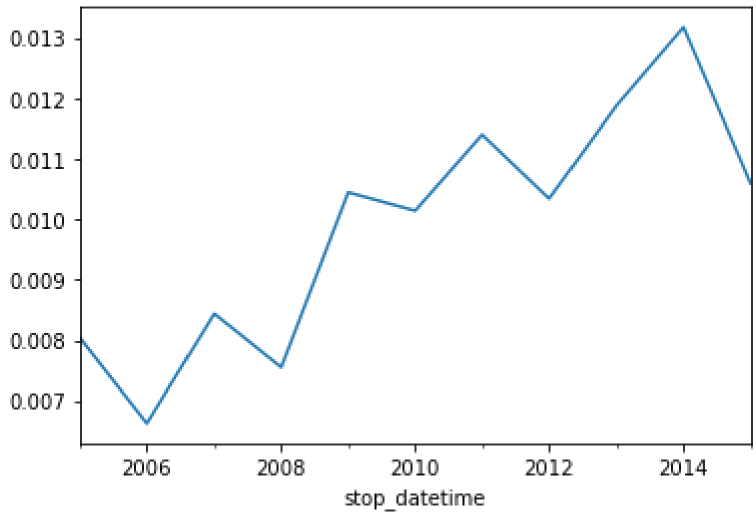
```
stop_datetime
2005-12-31    0.008038
2006-12-31    0.006624
2007-12-31    0.008437
2008-12-31    0.007549
2009-12-31    0.010447
2010-12-31    0.010142
2011-12-31    0.011400
2012-12-31    0.010343
2013-12-31    0.011879
2014-12-31    0.013176
2015-12-31    0.010598
Freq: A-DEC, Name: drugs_related_stop, dtype: float64
```

In [19]:

```
annual_drug_rate = ri.drugs_related_stop.resample("A").mean()
```

In [20]:

```
annual_drug_rate.plot()

plt.show()
```



Comparing drug and search rates (to be deleted)

As you saw in the last exercise, the rate of **drug-related stops** increased significantly between `2005` and `2015`. You might hypothesize that the rate of vehicle searches was also increasing, which would have led to an increase in drug-related stops even if more drivers were not carrying drugs.

You can test this hypothesis by calculating the annual search rate, and then plotting it against the annual drug rate. If the hypothesis is true, then you'll see both rates increasing over time.

INSTRUCTIONS

- Calculate the annual search rate by **resampling** the `search_conducted` column, and save the result as `annual_search_rate`.
- Concatenate `annual_drug_rate` and `annual_search_rate` along the `columns axis`, and save the result as `annual`.
- Create subplots of the drug and search rates from the `annual DataFrame`.
- Display the subplots.

In [21]:

```
ri.search_conducted.value_counts(dropna = False)
```

Out[21]:

```
False    462822
True      17762
Name: search_conducted, dtype: int64
```

In [22]:

```
ri.search_conducted.dtype
```

Out[22]:

```
dtype('O')
```

In [23]:

```
ri["search_conducted"] = ri.search_conducted.astype("bool")
```

In [24]:

```
ri.search_conducted.dtype
```

Out[24]:

```
dtype('bool')
```

In [25]:

```
ri.groupby(ri.index.year).search_conducted.mean()
```

Out[25]:

```
stop_datetime
2005    0.050692
2006    0.037748
2007    0.041844
2008    0.039544
2009    0.049849
2010    0.042089
2011    0.037767
2012    0.032278
2013    0.029054
2014    0.030157
2015    0.027832
Name: search_conducted, dtype: float64
```

In [26]:

```
annual_search_rate = ri.drugs_related_stop.resample("A").mean()
annual_search_rate
```

Out[26]:

```
stop_datetime
2005-12-31    0.008038
2006-12-31    0.006624
2007-12-31    0.008437
2008-12-31    0.007549
2009-12-31    0.010447
2010-12-31    0.010142
2011-12-31    0.011400
2012-12-31    0.010343
2013-12-31    0.011879
2014-12-31    0.013176
2015-12-31    0.010598
Freq: A-DEC, Name: drugs_related_stop, dtype: float64
```

In [27]:

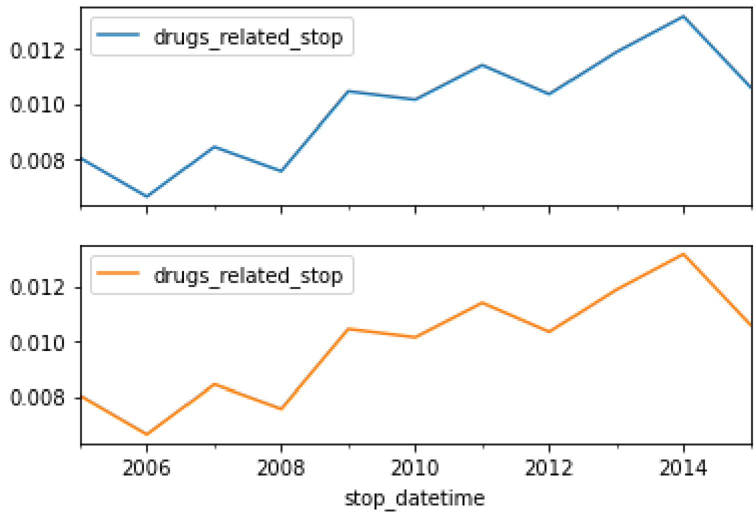
```
annual = pd.concat([annual_drug_rate, annual_search_rate], axis = "columns")
annual
```

Out[27]:

stop_datetime	drugs_related_stop	drugs_related_stop
2005-12-31	0.008038	0.008038
2006-12-31	0.006624	0.006624
2007-12-31	0.008437	0.008437
2008-12-31	0.007549	0.007549
2009-12-31	0.010447	0.010447
2010-12-31	0.010142	0.010142
2011-12-31	0.011400	0.011400
2012-12-31	0.010343	0.010343
2013-12-31	0.011879	0.011879
2014-12-31	0.013176	0.013176
2015-12-31	0.010598	0.010598

In [28]:

```
annual.plot(subplots = True)

plt.show()
```



Tallying violations by district

The state of **Rhode Island** is broken into six police districts, also known as zones. How do the zones compare in terms of what violations are caught by police?

In this exercise, you'll create a frequency table to determine how many violations of each type took place in each of the six zones. Then, you'll filter the table to focus on the `"K"` zones, which you'll examine further in the next exercise.

INSTRUCTIONS

- Create a `frequency table` from the `district` and `violation` columns using the `pd.crosstab()` function.
- Save the `frequency table` as a new object, `all_zones`.
- Select rows `'Zone K1'` through `'Zone K3'` from `all_zones` using the `.loc[]` accessor.
- Save the smaller table as a new object, `k_zones`.

In [29]:

```
ri.head(3)
```

Out[29]:

	id	state	location_raw	police_department	driver_gender	driver_age_raw	dr
stop_datetime							
2005-01-02 01:55:00	RI-2005-00001	RI	Zone K1	600	M	1985.0	
2005-01-02 20:30:00	RI-2005-00002	RI	Zone X4	500	M	1987.0	
2005-01-04 12:55:00	RI-2005-00004	RI	Zone X4	500	M	1986.0	

3 rows × 21 columns

In [30]:

```
ri.district.unique()
```

Out[30]:

```
array(['Zone K1', 'Zone X4', 'Zone X1', 'Zone K3', 'Zone X3', 'Zone K2'],
      dtype=object)
```

In [31]:

```
ri.district.value_counts(dropna = False)
```

Out[31]:

```
Zone X4    125670
Zone K3    108868
Zone K2     97281
Zone X3     89431
Zone K1     46110
Zone X1     13224
Name: district, dtype: int64
```

In [32]:

```
ri.violation.unique()
```

Out[32]:

```
array(['Speeding', 'Equipment', 'Other', 'Moving violation',
       'Registration/plates', 'Seat belt'], dtype=object)
```

In [33]:

```
ri.violation.value_counts(dropna = False)
```

Out[33]:

```
Speeding               268736
Moving violation        90228
Equipment               61250
Other                   24216
Registration/plates     19830
Seat belt               16324
Name: violation, dtype: int64
```

In [34]:

```
pd.crosstab(ri.district, ri.violation)
```

Out[34]:

violation	Equipment	Moving violation	Other	Registration/plates	Seat belt	Speeding
district						
Zone K1	3786	7127	1501	628	1	33067
Zone K2	11285	16440	5103	4056	2897	57500
Zone K3	12959	16218	3926	3871	3660	68234
Zone X1	1725	3711	752	192	451	6393
Zone X3	11520	17178	4069	3532	4445	48687
Zone X4	19975	29554	8865	7551	4870	54855

In [35]:

```
all_zones = pd.crosstab(ri.district, ri.violation)
```

In [38]:

```
all_zones.loc["Zone K1": "Zone K3"]
```

Out[38]:

violation	Equipment	Moving violation	Other	Registration/plates	Seat belt	Speeding
district						
Zone K1	3786	7127	1501	628	1	33067
Zone K2	11285	16440	5103	4056	2897	57500
Zone K3	12959	16218	3926	3871	3660	68234

In [39]:

```
k_zones = all_zones.loc["Zone K1": "Zone K3"]
```

In [ ]:

```

```

Plotting violations by district

Now that you've created a frequency table focused on the `"K"` zones, you'll visualize the data to help you compare what violations are being caught in each zone.

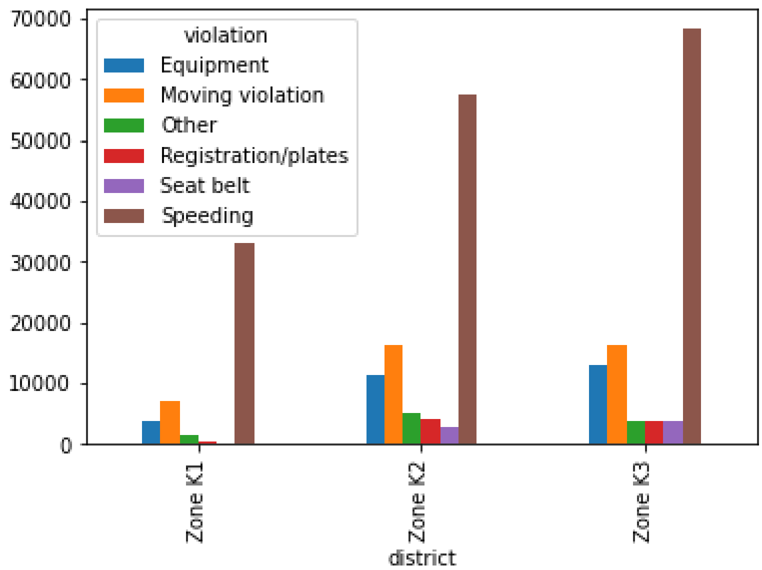
First you'll create a **bar plot**, which is an appropriate plot type since you're comparing categorical data. Then you'll create a **stacked bar plot** in order to get a slightly different look at the data. Which plot do you find to be more insightful?

INSTRUCTIONS 1/2

- Create a bar plot of `k_zones`.
- Display the plot and examine it. What do you notice about each of the zones?

In [42]:

```
k_zones.plot(kind = "bar")

plt.show()
```



In [ ]:

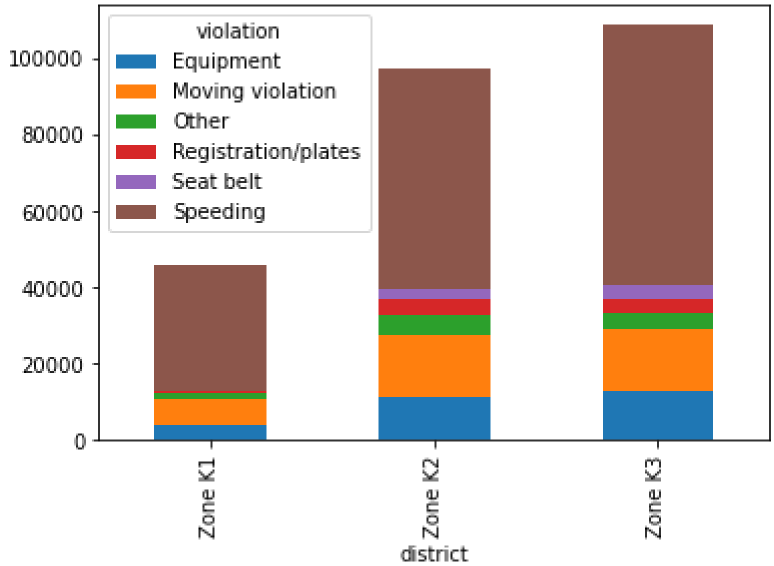
```

```

INSTRUCTIONS 2/2

- Create a stacked bar plot of `k_zones`.
- Display the plot and examine it. Do you notice anything different about the data than you did previously?

In [43]:

```
k_zones.plot(kind = "bar",  stacked = True )

plt.show()
```



Converting stop durations to numbers

In the traffic stops dataset, the `stop_duration` column tells you approximately how long the driver was detained by the officer. Unfortunately, the durations are stored as `strings` , such as `'0-15 Min'` . How can you make this data easier to analyze?

In this exercise, you'll convert the **stop durations** to `integers` . Because the precise durations are not available, you'll have to estimate the numbers using reasonable values:

- Convert `'0-15 Min'` to `8`
- Convert `'16-30 Min'` to `23`
- Convert `'30+ Min'` to `45`

INSTRUCTIONS

- Print the **unique values** in the `stop_duration` column. (This has been done for you.)
- Create a `dictionary` called `mapping` that maps the `stop_duration` strings to the integers specified above.
- Convert the `stop_duration` strings to integers using the `mapping` , and store the results in a new column called `stop_minutes` .
- Print the unique values in the `stop_minutes` column, to verify that the durations were properly converted to integers.

In [44]:

```
ri.head(3)
```

Out[44]:

	id	state	location_raw	police_department	driver_gender	driver_age_raw	dr
stop_datetime							
2005-01-02 01:55:00	RI-2005-00001	RI	Zone K1	600	M	1985.0	
2005-01-02 20:30:00	RI-2005-00002	RI	Zone X4	500	M	1987.0	
2005-01-04 12:55:00	RI-2005-00004	RI	Zone X4	500	M	1986.0	

3 rows × 21 columns

In [45]:

```
ri.stop_duration
```

Out[45]:

```
stop_datetime
2005-01-02 01:55:00     0-15 Min
2005-01-02 20:30:00    16-30 Min
2005-01-04 12:55:00     0-15 Min
2005-01-06 01:30:00     0-15 Min
2005-01-12 08:05:00      30+ Min
                         ...
2015-12-31 22:46:00     0-15 Min
2015-12-31 22:47:00     0-15 Min
2015-12-31 23:08:00     0-15 Min
2015-12-31 23:44:00     0-15 Min
2015-12-31 23:48:00     0-15 Min
Name: stop_duration, Length: 480584, dtype: object
```

In [46]:

```
ri.stop_duration.unique()
```

Out[46]:

```
array(['0-15 Min', '16-30 Min', '30+ Min', '2', '1'], dtype=object)
```

In [47]:

```
ri.stop_duration.value_counts()
```

Out[47]:

```
0-15 Min     386646
16-30 Min     76320
30+ Min       17612
1                 5
2                 1
Name: stop_duration, dtype: int64
```

In [48]:

```
mapping = {'0-15 Min': 8, '16-30 Min': 23, '30+ Min': 45 }
```

In [49]:

```
ri["stop_minutes"] = ri.stop_duration.map(mapping)
```

In [50]:

```
ri.stop_minutes.value_counts(dropna = False)
```

Out[50]:

```
8.0     386646
23.0     76320
45.0     17612
NaN          6
Name: stop_minutes, dtype: int64
```

In [ ]:

```

```

Plotting stop length

If you were stopped for a particular violation, how long might you expect to be detained?

In this exercise, you'll visualize the **average length** of time drivers are stopped for each **type** of **violation**. Rather than using the `violation` column in this exercise, you'll use `violation_raw` since it contains more detailed descriptions of the violations.

INSTRUCTIONS

- For each value in the `violation_raw` column, calculate the **mean number** of `stop_minutes` that a driver is detained.
- Save the resulting `Series` as a new object, `stop_length`.
- Sort `stop_length` by its values, and then visualize it using a **horizontal bar plot**.
- Display the plot.

In [51]:

```
ri.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 480584 entries, 2005-01-02 01:55:00 to 2015-12-31 23:48:00
Data columns (total 22 columns):
 #   Column              Non-Null Count   Dtype  
---  ------              --------------   -----  
 0   id                  480584 non-null  object 
 1   state               480584 non-null  object 
 2   location_raw        480584 non-null  object 
 3   police_department   480584 non-null  object 
 4   driver_gender       480584 non-null  object 
 5   driver_age_raw      480583 non-null  float64
 6   driver_age          478946 non-null  float64
 7   driver_race_raw     480584 non-null  object 
 8   driver_race         480584 non-null  object 
 9   violation_raw       480584 non-null  object 
 10  violation           480584 non-null  object 
 11  search_conducted    480584 non-null  bool   
 12  search_type_raw     17762 non-null   object 
 13  search_type         17762 non-null   object 
 14  contraband_found    480584 non-null  bool   
 15  stop_outcome        480584 non-null  object 
 16  is_arrested         480584 non-null  bool   
 17  stop_duration       480584 non-null  object 
 18  out_of_state        479776 non-null  object 
 19  drugs_related_stop  480584 non-null  bool   
 20  district            480584 non-null  object 
 21  stop_minutes        480578 non-null  float64
dtypes: bool(4), float64(3), object(15)
memory usage: 71.5+ MB
```

In [52]:

```
ri.violation_raw.unique()
```

Out[52]:

```
array(['Speeding', 'Equipment/Inspection Violation', 'Call for Service',
       'Other Traffic Violation', 'Registration Violation',
       'Violation of City/Town Ordinance',
       'Special Detail/Directed Patrol', 'APB',
       'Motorist Assist/Courtesy', 'Suspicious Person', 'Warrant',
       'Seatbelt Violation'], dtype=object)
```

In [53]:

```
ri.groupby("violation_raw").stop_minutes.mean()
```

Out[53]:

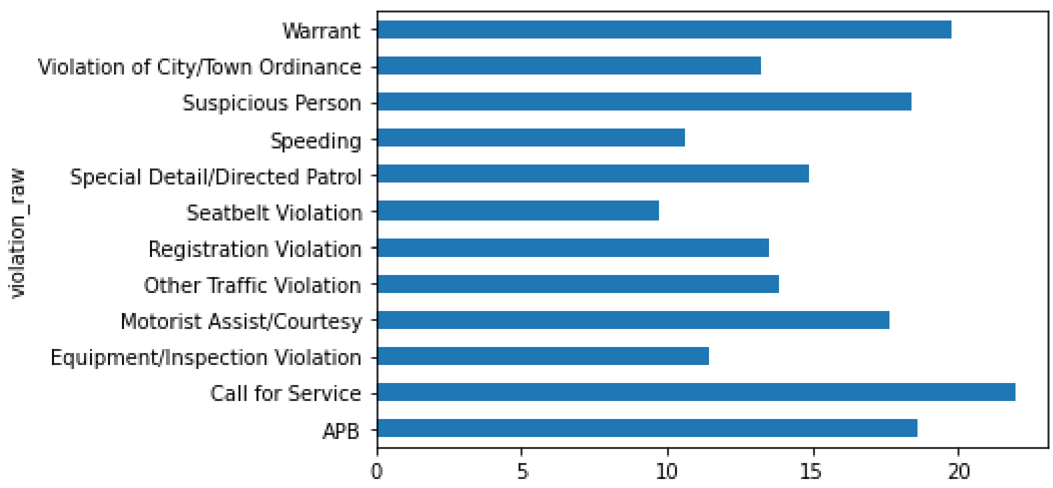
```
violation_raw
APB                                 18.593814
Call for Service                    21.963314
Equipment/Inspection Violation      11.454326
Motorist Assist/Courtesy            17.629929
Other Traffic Violation             13.834359
Registration Violation              13.543268
Seatbelt Violation                   9.698236
Special Detail/Directed Patrol      14.876778
Speeding                            10.589215
Suspicious Person                   18.374269
Violation of City/Town Ordinance    13.230695
Warrant                             19.769231
Name: stop_minutes, dtype: float64
```

In [54]:

```
stop_length = ri.groupby("violation_raw").stop_minutes.mean()
```

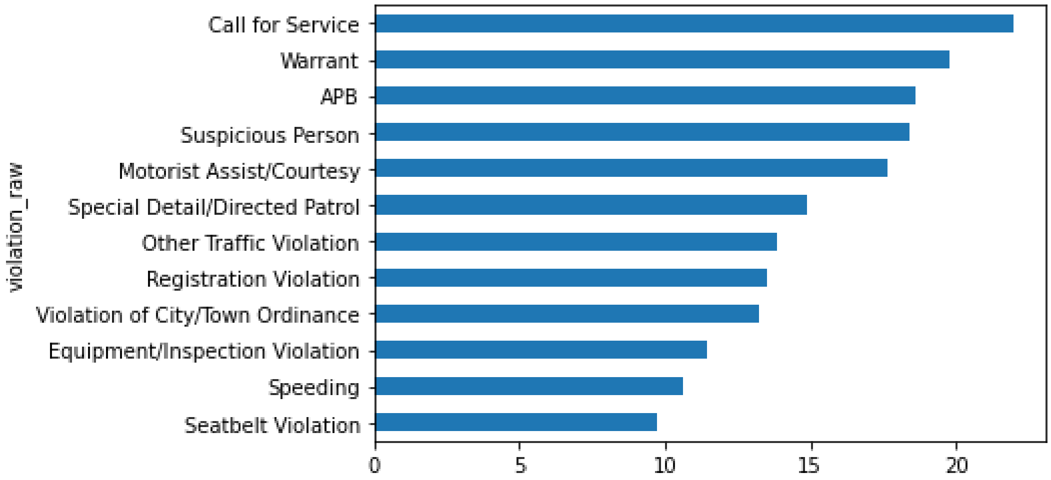
In [56]:

```
stop_length.plot(kind = "barh" )

plt.show()
```



In [62]:

```
stop_length.sort_values().plot(kind = "barh")

plt.show()
```



In [ ]: