04b_data_pandas

November 28, 2023

1 Doing More with Data: `pandas`

1.1 Introduction to Python

Data Sciences Institute, University of Toronto

Instructor: Kaylie Lau | TA: Tong Su

December 2023

2 Contents:

1. Setup
2. Intro to `pandas`
3. Getting data
4. Profiling and initial data exploration: changing data types, descriptive statistics
5. Wrangling and plotting: concatenating, merging, adding and removing columns, filtering and selecting, null values, grouping and aggregating, plotting
6. Writing to file
7. More wrangling: reshaping, applying functions

2.1 Data

This module uses four datasets: bike thefts, TTC subway delays and subway delay reason codes, and neighbourhood profiles. All four are available in the course repo, and originally come from Toronto Open Data.

The specific file names are: - bicycle-thefts - 4326.csv - ttc-subway-delay-data-2021.xlsx - ttc-subway-delay-codes.xlsx - neighbourhood-profiles-2016-140-model.csv

3 `pandas`

3.1 What is `pandas`?

`pandas` is a package for data analysis and manipulation. (The name is a reference to panel data, not the animal.) It gives us data frames, which represent data in a table of columns and rows, and functions to manipulate and plot them. `pandas` also provides a slew of functions for reading and writing data to a variety of sources, including files, SQL databases, and compressed binary formats.

[96]:
```
import numpy as np
# pd is the conventional alias for pandas
import pandas as pd

# display all columns
pd.set_option("display.max_columns", None)
```

3.2 DataFrames

Columns are labeled with their names. Rows also have a label, or *index*. If row labels are not specified, `pandas` uses numbers as the default. Each column is a *Series*, or one-dimensional array, where values share a data type. Unlike `numpy` arrays, DataFrames can have columns of different data types. However, like arrays and lists, **DataFrames are mutable** – this means that if more than one variable refers to the same DataFrame, updating one updates them all!

3.3 Getting data

We can create a DataFrame manually with `DataFrame()` constructor. If a dictionary is passed to `DataFrame()`, the keys become column names, and the values become the rows. Calling just `DataFrame()` creates an empty DataFrame to which data can be added later.

[97]:
```
trees = pd.DataFrame({
    'name': ['sugar maple', 'black oak', 'white ash', 'douglas fir'],
    'avg_lifespan': [300, 100, 260, 450],
    'quantity': [53, 207, 178, 93]
})
trees
```

[97]:

	name	avg_lifespan	quantity
0	sugar maple	300	53
1	black oak	100	207
2	white ash	260	178
3	douglas fir	450	93

We can create an individual column with `Series()`. The `name` argument corresponds to a column name.

[98]:
```
tree_types = pd.Series(['deciduous', 'deciduous', 'deciduous', 'evergreen'],
                       name='foliage')
tree_types
```

[98]:
```
0    deciduous
1    deciduous
2    deciduous
3    evergreen
Name: foliage, dtype: object
```

3.3.1 Data from csv

Of course, we're more likely to load data into a DataFrame than to create DataFrames manually. `pandas` has read functions for different file formats. To read data from a csv or other delimited file, we use `pd.read_csv()`, then pass in the local file path or the URL of the csv to read. `pandas` will infer the data type of each column based on the values in the first chunk of the file loaded.

[99]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

[100]:
```
thefts = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/data/
↪bicycle-thefts - 4326.csv')
```

3.4 Profiling and initial data cleaning

We got our data, but now we need to understand what's in it. We can start to understand the DataFrame by checking out its `dtypes` and `shape` attributes, which give column data types and row by column dimensions, respectively. Note that `object` is `pandas`' way of saying values are represented as string data.

[101]:
```
thefts.shape
```

[101]:
```
(25569, 33)
```

[102]:
```
thefts.dtypes
```

[102]:
```
_id                        int64
OBJECTID                   int64
event_unique_id           object
Primary_Offence           object
Occurrence_Date           object
Occurrence_Year            int64
Occurrence_Month          object
Occurrence_DayOfWeek      object
Occurrence_DayOfMonth      int64
Occurrence_DayOfYear       int64
Occurrence_Hour            int64
Report_Date               object
Report_Year                int64
Report_Month              object
Report_DayOfWeek          object
Report_DayOfMonth          int64
Report_DayOfYear           int64
Report_Hour                int64
Division                  object
```

```
City                    object
Hood_ID                 object
NeighbourhoodName       object
Location_Type           object
Premises_Type           object
Bike_Make               object
Bike_Model              object
Bike_Type               object
Bike_Speed               int64
Bike_Colour             object
Cost_of_Bike           float64
Status                  object
ObjectId2                int64
geometry                object
dtype: object
```

3.4.1 head()s and tail()s

To check out the first few rows, we can call the DataFrame `head()` method. Similarly, we can see the last few rows with the `tail()` method. Five rows are shown by default, but we can change that by passing an integer as an argument.

[103]:
```
thefts.head()
```

[103]:

	_id	OBJECTID	event_unique_id	Primary_Offence	Occurrence_Date
0	1	17744	GO-20179016397	THEFT UNDER	2017-10-03T00:00:00
1	2	17759	GO-20172033056	THEFT UNDER - BICYCLE	2017-11-08T00:00:00
2	3	17906	GO-20189030822	THEFT UNDER - BICYCLE	2018-09-14T00:00:00
3	4	17962	GO-2015804467	THEFT UNDER	2015-05-07T00:00:00
4	5	17963	GO-20159002781	THEFT UNDER	2015-05-16T00:00:00

	Occurrence_Year	Occurrence_Month	Occurrence_DayOfWeek
0	2017	October	Tuesday
1	2017	November	Wednesday
2	2018	September	Friday
3	2015	May	Thursday
4	2015	May	Saturday

	Occurrence_DayOfMonth	Occurrence_DayOfYear	Occurrence_Hour
0	3	276	14
1	8	312	3
2	14	257	9
3	7	127	18
4	16	136	12

	Report_Date	Report_Year	Report_Month	Report_DayOfWeek
0	2017-10-03T00:00:00	2017	October	Tuesday

```
1  2017-11-08T00:00:00         2017     November        Wednesday
2  2018-09-17T00:00:00         2018    September           Monday
3  2015-05-14T00:00:00         2015          May         Thursday
4  2015-05-16T00:00:00         2015          May         Saturday

   Report_DayOfMonth  Report_DayOfYear  Report_Hour Division     City Hood_ID  \
0                  3               276           18      D22  Toronto      15
1                  8               312           22      D22  Toronto      15
2                 17               260           16      D22  Toronto      15
3                 14               134           14      D22  Toronto      15
4                 16               136           15      D22  Toronto      15

     NeighbourhoodName                                      Location_Type  \
0  Kingsway South (15)  Streets, Roads, Highways (Bicycle Path, Privat…
1  Kingsway South (15)  Single Home, House (Attach Garage, Cottage, Mo…
2  Kingsway South (15)                                Ttc Subway Station
3  Kingsway South (15)                                Ttc Subway Station
4  Kingsway South (15)                                Ttc Subway Station

  Premises_Type     Bike_Make  Bike_Model Bike_Type  Bike_Speed Bike_Colour  \
0       Outside            GI    ESCAPE 2        OT           7         BLK
1         House  UNKNOWN MAKE         NaN        TO           1         BLK
2       Transit            OT  CROSSTRAIL        MT          24         BLK
3       Transit            GT         NaN        TO          10      BLKDGR
4       Transit            GI         NaN        MT           6         RED

   Cost_of_Bike     Status  ObjectId2  \
0         700.0     STOLEN          1
1        1100.0  RECOVERED          2
2         904.0     STOLEN          3
3         400.0     STOLEN          4
4         600.0     STOLEN          5

                                            geometry
0  {'type': 'Point', 'coordinates': (-79.50655965…
1  {'type': 'Point', 'coordinates': (-79.50484874…
2  {'type': 'Point', 'coordinates': (-79.51170915…
3  {'type': 'Point', 'coordinates': (-79.51170915…
4  {'type': 'Point', 'coordinates': (-79.51132657…
```

[104]:
```
# last 3
thefts.tail(3)
```

[104]:
```
          _id  OBJECTID event_unique_id        Primary_Offence  \
25566  25567     11462  GO-20169005434            THEFT UNDER
25567  25568     11695  GO-20161170896            THEFT UNDER
25568  25569     11883  GO-20169007653  THEFT UNDER - BICYCLE
```

```
          Occurrence_Date  Occurrence_Year Occurrence_Month  \
25566  2016-06-04T00:00:00             2016             June
25567  2016-07-04T00:00:00             2016             July
25568  2016-07-22T00:00:00             2016             July

      Occurrence_DayOfWeek  Occurrence_DayOfMonth  Occurrence_DayOfYear  \
25566             Saturday                      4                   156
25567               Monday                      4                   186
25568               Friday                     22                   204

       Occurrence_Hour          Report_Date  Report_Year Report_Month  \
25566               22  2016-06-07T00:00:00         2016         June
25567               20  2016-07-04T00:00:00         2016         July
25568                9  2016-07-23T00:00:00         2016         July

      Report_DayOfWeek  Report_DayOfMonth  Report_DayOfYear  Report_Hour  \
25566          Tuesday                  7               159           16
25567           Monday                  4               186           20
25568         Saturday                 23               205           11

      Division     City Hood_ID NeighbourhoodName  \
25566      D42  Toronto     132     Malvern (132)
25567      D42  Toronto     132     Malvern (132)
25568      D42  Toronto     132     Malvern (132)

                                      Location_Type Premises_Type  \
25566                  Apartment (Rooming House, Condo)     Apartment
25567  Other Commercial / Corporate Places (For Profi…    Commercial
25568  Parking Lots (Apt., Commercial Or Non-Commercial)       Outside

         Bike_Make       Bike_Model Bike_Type  Bike_Speed Bike_Colour  \
25566           SC           ANTRIM        MT          24         WHI
25567  UNKNOWN MAKE              NaN        SC           1         NaN
25568           SU  ASCENT MOUNTAIN        MT          21         ONG

       Cost_of_Bike  Status  ObjectId2  \
25566         700.0  STOLEN      25567
25567        3000.0  STOLEN      25568
25568         200.0  STOLEN      25569

                                              geometry
25566  {'type': 'Point', 'coordinates': (-79.2360175,…
25567  {'type': 'Point', 'coordinates': (-79.20060719…
25568  {'type': 'Point', 'coordinates': (-79.23734742…
```

3.4.2 Renaming columns

Most, but not all, of the bike theft columns follow the same naming convention. For convenience's sake, though, let's convert the column names to all lowercase. We can do this with the DataFrame `rename()` method. `rename()` accepts either a dictionary with current column names as the keys and new names as the values, or the name of a function to transform names. Let's write a function.

[105]:
```
# notice that we do not add () to the function name
thefts = thefts.rename(columns=str.lower)
```

[106]:
```
thefts
```

[106]:

	_id	objectid	event_unique_id	primary_offence	\
0	1	17744	GO-20179016397	THEFT UNDER	
1	2	17759	GO-20172033056	THEFT UNDER - BICYCLE	
2	3	17906	GO-20189030822	THEFT UNDER - BICYCLE	
3	4	17962	GO-2015804467	THEFT UNDER	
4	5	17963	GO-20159002781	THEFT UNDER	
…	…	…	…	…	
25564	25565	9361	GO-2015543181	MISCHIEF UNDER	
25565	25566	11318	GO-20169004589	THEFT UNDER	
25566	25567	11462	GO-20169005434	THEFT UNDER	
25567	25568	11695	GO-20161170896	THEFT UNDER	
25568	25569	11883	GO-20169007653	THEFT UNDER - BICYCLE	

	occurrence_date	occurrence_year	occurrence_month	\
0	2017-10-03T00:00:00	2017	October	
1	2017-11-08T00:00:00	2017	November	
2	2018-09-14T00:00:00	2018	September	
3	2015-05-07T00:00:00	2015	May	
4	2015-05-16T00:00:00	2015	May	
…	…	…	…	
25564	2015-04-01T00:00:00	2015	April	
25565	2016-05-16T00:00:00	2016	May	
25566	2016-06-04T00:00:00	2016	June	
25567	2016-07-04T00:00:00	2016	July	
25568	2016-07-22T00:00:00	2016	July	

	occurrence_dayofweek	occurrence_dayofmonth	occurrence_dayofyear	\
0	Tuesday	3	276	
1	Wednesday	8	312	
2	Friday	14	257	
3	Thursday	7	127	
4	Saturday	16	136	
…	…	…	…	
25564	Wednesday	1	91	
25565	Monday	16	137	
25566	Saturday	4	156	

```
25567               Monday                        4                    186
25568               Friday                       22                    204

       occurrence_hour          report_date  report_year report_month  \
0                   14  2017-10-03T00:00:00         2017      October
1                    3  2017-11-08T00:00:00         2017     November
2                    9  2018-09-17T00:00:00         2018    September
3                   18  2015-05-14T00:00:00         2015          May
4                   12  2015-05-16T00:00:00         2015          May
…                  …                    …          …          …
25564               17  2015-04-01T00:00:00         2015        April
25565               21  2016-05-16T00:00:00         2016          May
25566               22  2016-06-07T00:00:00         2016         June
25567               20  2016-07-04T00:00:00         2016         July
25568                9  2016-07-23T00:00:00         2016         July

      report_dayofweek  report_dayofmonth  report_dayofyear  report_hour  \
0              Tuesday                  3               276           18
1            Wednesday                  8               312           22
2               Monday                 17               260           16
3             Thursday                 14               134           14
4             Saturday                 16               136           15
…                  …                  …                 …          …
25564        Wednesday                  1                91           19
25565           Monday                 16               137           21
25566          Tuesday                  7               159           16
25567           Monday                  4               186           20
25568         Saturday                 23               205           11

      division     city hood_id    neighbourhoodname  \
0          D22  Toronto      15  Kingsway South (15)
1          D22  Toronto      15  Kingsway South (15)
2          D22  Toronto      15  Kingsway South (15)
3          D22  Toronto      15  Kingsway South (15)
4          D22  Toronto      15  Kingsway South (15)
…        …      …     …                  …
25564      D42  Toronto     132        Malvern (132)
25565      D42  Toronto     132        Malvern (132)
25566      D42  Toronto     132        Malvern (132)
25567      D42  Toronto     132        Malvern (132)
25568      D42  Toronto     132        Malvern (132)

                                        location_type premises_type  \
0      Streets, Roads, Highways (Bicycle Path, Privat…       Outside
1      Single Home, House (Attach Garage, Cottage, Mo…         House
2                                   Ttc Subway Station       Transit
3                                   Ttc Subway Station       Transit
```

```
4                                    Ttc Subway Station       Transit
…                                                   …           …
25564  Parking Lots (Apt., Commercial Or Non-Commercial)       Outside
25565  Single Home, House (Attach Garage, Cottage, Mo…         House
25566                  Apartment (Rooming House, Condo)     Apartment
25567  Other Commercial / Corporate Places (For Profi…    Commercial
25568  Parking Lots (Apt., Commercial Or Non-Commercial)       Outside

          bike_make       bike_model bike_type  bike_speed bike_colour  \
0                GI         ESCAPE 2        OT           7         BLK
1      UNKNOWN MAKE              NaN        TO           1         BLK
2                OT       CROSSTRAIL        MT          24         BLK
3                GT              NaN        TO          10      BLKDGR
4                GI              NaN        MT           6         RED
…             …              …       …         …          …
25564  UNKNOWN MAKE     BMX WILD MAN        RG           0         SIL
25565            SC              NaN        OT          14         NaN
25566            SC           ANTRIM        MT          24         WHI
25567  UNKNOWN MAKE              NaN        SC           1         NaN
25568            SU  ASCENT MOUNTAIN        MT          21         ONG

       cost_of_bike     status  objectid2  \
0             700.0     STOLEN          1
1            1100.0  RECOVERED          2
2             904.0     STOLEN          3
3             400.0     STOLEN          4
4             600.0     STOLEN          5
…             …        …        …
25564         600.0     STOLEN      25565
25565         900.0     STOLEN      25566
25566         700.0     STOLEN      25567
25567        3000.0     STOLEN      25568
25568         200.0     STOLEN      25569

                                                  geometry
0      {'type': 'Point', 'coordinates': (-79.50655965…
1      {'type': 'Point', 'coordinates': (-79.50484874…
2      {'type': 'Point', 'coordinates': (-79.51170915…
3      {'type': 'Point', 'coordinates': (-79.51170915…
4      {'type': 'Point', 'coordinates': (-79.51132657…
…                                                    …
25564  {'type': 'Point', 'coordinates': (-79.21555349…
25565  {'type': 'Point', 'coordinates': (-79.21767046…
25566  {'type': 'Point', 'coordinates': (-79.2360175,…
25567  {'type': 'Point', 'coordinates': (-79.20060719…
25568  {'type': 'Point', 'coordinates': (-79.23734742…
```

```
[25569 rows x 33 columns]
```

Let's also rename `cost_of_bike` so it follows the pattern of the other bike attribute columns.

[107]:
```
thefts = thefts.rename(columns={'cost_of_bike':'bike_cost'})

# view column names
print(list(thefts))
```

```
['_id', 'objectid', 'event_unique_id', 'primary_offence', 'occurrence_date',
'occurrence_year', 'occurrence_month', 'occurrence_dayofweek',
'occurrence_dayofmonth', 'occurrence_dayofyear', 'occurrence_hour',
'report_date', 'report_year', 'report_month', 'report_dayofweek',
'report_dayofmonth', 'report_dayofyear', 'report_hour', 'division', 'city',
'hood_id', 'neighbourhoodname', 'location_type', 'premises_type', 'bike_make',
'bike_model', 'bike_type', 'bike_speed', 'bike_colour', 'bike_cost', 'status',
'objectid2', 'geometry']
```

3.4.3 Profiling columns

It can be useful to focus on a subset of columns, particularly to understand value sets. To select a single column in a DataFrame, we can supply the name of the column in square brackets, just like we did when accessing values in a dictionary. `pandas` will return the column as a Series. To get unique values, we can use the `unique()` Series method. If we want to count how many times each value appears, we can use the `value_counts()` method.

[108]:
```
thefts['status']
```

[108]:
```
0           STOLEN
1        RECOVERED
2           STOLEN
3           STOLEN
4           STOLEN
            ...
25564       STOLEN
25565       STOLEN
25566       STOLEN
25567       STOLEN
25568       STOLEN
Name: status, Length: 25569, dtype: object
```

[109]:
```
thefts['status'].unique()
```

[109]:
```
array(['STOLEN', 'RECOVERED', 'UNKNOWN'], dtype=object)
```

[110]:
```
thefts['status'].value_counts()
```

[110]:
```
STOLEN       24807
UNKNOWN        454
RECOVERED      308
Name: status, dtype: int64
```

We can summarize numeric Series much like we did with `numpy` functions.

[111]:
```
thefts['bike_cost'].median()
```

[111]:
```
600.0
```

[112]:
```
thefts['bike_cost'].quantile(0.9) #qth quantile
```

[112]:
```
2000.0
```

3.4.4 `info()`

We can get an overview of the DataFrame by profiling it with the `info()` method.

`info()` prints a lot of information about a DataFrame, including: * the `shape` as the number of rows and columns * column names and their `dtype` * the number of non-null values in each column * how big the DataFrame is in terms of memory usage

The bicycle theft data looks quite complete, though some records are missing bike descriptors like bike_make, bike_model, bike_colour, and bike_cost.

Most of the column dtypes make sense. We'll want to convert the dates to proper dates. We may also want to convert string columns with limited value sets, like status, to categorical data.

[113]:
```
thefts.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25569 entries, 0 to 25568
Data columns (total 33 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   _id                    25569 non-null  int64
 1   objectid               25569 non-null  int64
 2   event_unique_id        25569 non-null  object
 3   primary_offence        25569 non-null  object
 4   occurrence_date        25569 non-null  object
 5   occurrence_year        25569 non-null  int64
 6   occurrence_month       25569 non-null  object
 7   occurrence_dayofweek   25569 non-null  object
 8   occurrence_dayofmonth  25569 non-null  int64
 9   occurrence_dayofyear   25569 non-null  int64
 10  occurrence_hour        25569 non-null  int64
 11  report_date            25569 non-null  object
 12  report_year            25569 non-null  int64
 13  report_month           25569 non-null  object
```

```
 14  report_dayofweek       25569 non-null  object
 15  report_dayofmonth      25569 non-null  int64
 16  report_dayofyear       25569 non-null  int64
 17  report_hour            25569 non-null  int64
 18  division               25569 non-null  object
 19  city                   25569 non-null  object
 20  hood_id                25569 non-null  object
 21  neighbourhoodname      25569 non-null  object
 22  location_type          25569 non-null  object
 23  premises_type          25569 non-null  object
 24  bike_make              25448 non-null  object
 25  bike_model             15923 non-null  object
 26  bike_type              25569 non-null  object
 27  bike_speed             25569 non-null  int64
 28  bike_colour            23508 non-null  object
 29  bike_cost              23825 non-null  float64
 30  status                 25569 non-null  object
 31  objectid2              25569 non-null  int64
 32  geometry               25569 non-null  object
dtypes: float64(1), int64(12), object(20)
memory usage: 6.4+ MB
```

3.4.5 Changing data types

Before exploring the bike theft data further, let's fix the date and categorical columns. To convert a column to datetime, we use the `pd.to_datetime()` function, passing in the column to convert, and reassign the output back to the column we're converting.

`pandas` knows how to convert the dates in the bike thefts data, but for less common formats, it is necessary to use the `format` keyword argument to specify how dates should be parsed. `format` strings use `strftime` codes. See https://strftime.org/ for a cheat sheet.

[114]:
```
thefts['occurrence_date'] = pd.to_datetime(thefts['occurrence_date'])
thefts['occurrence_date']
```

[114]:
```
0       2017-10-03
1       2017-11-08
2       2018-09-14
3       2015-05-07
4       2015-05-16
           ...
25564   2015-04-01
25565   2016-05-16
25566   2016-06-04
25567   2016-07-04
25568   2016-07-22
Name: occurrence_date, Length: 25569, dtype: datetime64[ns]
```

[115]:
```
# convert report_date without the format argument
thefts['report_date'] = pd.to_datetime(thefts['report_date'])
thefts['report_date']
```

[115]:
```
0       2017-10-03
1       2017-11-08
2       2018-09-17
3       2015-05-14
4       2015-05-16
           …
25564   2015-04-01
25565   2016-05-16
25566   2016-06-07
25567   2016-07-04
25568   2016-07-23
Name: report_date, Length: 25569, dtype: datetime64[ns]
```

All other data type conversions can be done with the `astype()` method. If we were converting to a number, `pd.to_numeric()` provides an easy way to convert without having to pick a specific numeric data type.

[116]:
```
thefts['status'] = thefts['status'].astype('category')
thefts['status']
```

[116]:
```
0           STOLEN
1        RECOVERED
2           STOLEN
3           STOLEN
4           STOLEN
           …
25564       STOLEN
25565       STOLEN
25566       STOLEN
25567       STOLEN
25568       STOLEN
Name: status, Length: 25569, dtype: category
Categories (3, object): ['RECOVERED', 'STOLEN', 'UNKNOWN']
```

We can select and convert multiple columns at once by passing a list of columns in the square brackets., then using `.astype()`.

[117]:
```
thefts[['location_type', 'premises_type']] =␣
 ↪thefts[['location_type','premises_type']].astype('category')

# check data types
thefts[['location_type', 'premises_type']].dtypes
```

[117]:
```
location_type    category
premises_type    category
dtype: object
```

3.4.6 `describe()`

To get a sense of the values in a DataFrame, we can use the `describe()` method. `describe()` summarizes only numeric columns by default. Passing the `include='all'` argument will produce summary statistics for other columns as well.

[118]:
```
thefts.describe(include='all',
                datetime_is_numeric=True)  # silence warning about upcoming␣
↪pandas change
```

[118]:

	_id	objectid	event_unique_id	primary_offence
count	25569.000000	25569.000000	25569	25569
unique	NaN	NaN	22771	66
top	NaN	NaN	GO-20201550944	THEFT UNDER
freq	NaN	NaN	14	11904
mean	12785.000000	12909.173218	NaN	NaN
min	1.000000	1.000000	NaN	NaN
25%	6393.000000	6456.000000	NaN	NaN
50%	12785.000000	12918.000000	NaN	NaN
75%	19177.000000	19360.000000	NaN	NaN
max	25569.000000	25806.000000	NaN	NaN
std	7381.278853	7448.318562	NaN	NaN

	occurrence_date	occurrence_year	occurrence_month
count	25569	25569.000000	25569
unique	NaN	NaN	12
top	NaN	NaN	July
freq	NaN	NaN	4002
mean	2017-09-04 03:39:28.321013504	2017.124174	NaN
min	2009-09-01 00:00:00	2009.000000	NaN
25%	2016-01-06 00:00:00	2016.000000	NaN
50%	2017-09-05 00:00:00	2017.000000	NaN
75%	2019-06-20 00:00:00	2019.000000	NaN
max	2020-12-30 00:00:00	2020.000000	NaN
std	NaN	1.960127	NaN

	occurrence_dayofweek	occurrence_dayofmonth	occurrence_dayofyear
count	25569	25569.000000	25569.000000
unique	7	NaN	NaN
top	Friday	NaN	NaN
freq	3924	NaN	NaN
mean	NaN	15.616684	202.227698
min	NaN	1.000000	1.000000

25%	NaN	8.000000	153.000000
50%	NaN	16.000000	205.000000
75%	NaN	23.000000	259.000000
max	NaN	31.000000	366.000000
std	NaN	8.592886	76.821431

	occurrence_hour	report_date	report_year
count	25569.000000	25569	25569.000000
unique	NaN	NaN	NaN
top	NaN	NaN	NaN
freq	NaN	NaN	NaN
mean	13.274395	2017-09-12 12:02:37.127771904	2017.143572
min	0.000000	2014-01-01 00:00:00	2014.000000
25%	9.000000	2016-01-22 00:00:00	2016.000000
50%	14.000000	2017-09-12 00:00:00	2017.000000
75%	19.000000	2019-06-26 00:00:00	2019.000000
max	23.000000	2020-12-31 00:00:00	2020.000000
std	6.530181	NaN	1.955024

	report_month	report_dayofweek	report_dayofmonth	report_dayofyear
count	25569	25569	25569.000000	25569.000000
unique	12	7	NaN	NaN
top	July	Monday	NaN	NaN
freq	3988	4318	NaN	NaN
mean	NaN	NaN	15.924870	203.493723
min	NaN	NaN	1.000000	1.000000
25%	NaN	NaN	9.000000	154.000000
50%	NaN	NaN	16.000000	206.000000
75%	NaN	NaN	23.000000	260.000000
max	NaN	NaN	31.000000	366.000000
std	NaN	NaN	8.549584	77.115977

	report_hour	division	city	hood_id
count	25569.000000	25569	25569	25569
unique	NaN	18	2	141
top	NaN	D14	Toronto	77
freq	NaN	4580	25560	2576
mean	14.224139	NaN	NaN	NaN
min	0.000000	NaN	NaN	NaN
25%	11.000000	NaN	NaN	NaN
50%	14.000000	NaN	NaN	NaN
75%	18.000000	NaN	NaN	NaN
max	23.000000	NaN	NaN	NaN
std	5.052944	NaN	NaN	NaN

	neighbourhoodname
count	25569

```
unique                                    141
top     Waterfront Communities-The Island (77)
freq                                     2576
mean                                      NaN
min                                       NaN
25%                                       NaN
50%                                       NaN
75%                                       NaN
max                                       NaN
std                                       NaN

                          location_type premises_type bike_make bike_model  \
count                             25569         25569     25448      15923
unique                               42             7       820       8097
top     Apartment (Rooming House, Condo)       Outside        OT    UNKNOWN
freq                               5887          7960      4991        304
mean                                NaN           NaN       NaN        NaN
min                                 NaN           NaN       NaN        NaN
25%                                 NaN           NaN       NaN        NaN
50%                                 NaN           NaN       NaN        NaN
75%                                 NaN           NaN       NaN        NaN
max                                 NaN           NaN       NaN        NaN
std                                 NaN           NaN       NaN        NaN

       bike_type    bike_speed bike_colour      bike_cost  status  \
count      25569  25569.000000       23508   23825.000000   25569
unique        13           NaN         252            NaN       3
top           MT           NaN         BLK            NaN  STOLEN
freq        8245           NaN        7422            NaN   24807
mean         NaN     14.164144         NaN     949.542371     NaN
min          NaN      0.000000         NaN       0.000000     NaN
25%          NaN      6.000000         NaN     350.000000     NaN
50%          NaN     15.000000         NaN     600.000000     NaN
75%          NaN     21.000000         NaN    1000.000000     NaN
max          NaN     99.000000         NaN  120000.000000     NaN
std          NaN     10.559215         NaN    1675.880345     NaN

          objectid2                                                geometry
count   25569.000000                                                   25569
unique           NaN                                                    5816
top              NaN  {'type': 'Point', 'coordinates': (-79.38372586…
freq             NaN                                                     167
mean    12785.000000                                                    NaN
min         1.000000                                                    NaN
25%      6393.000000                                                    NaN
50%     12785.000000                                                    NaN
75%     19177.000000                                                    NaN
```

```
max      25569.000000                                                  NaN
std       7381.278853                                                  NaN
```

3.5 Wrangling and Plotting

3.5.1 Combining datasets: concatenation

Just as `pandas` has `read_csv()` for flat files, there is a `read_excel()` function to load Excel files.

The TTC publishes subway delay data as a multi-sheet Excel workbook, with a month's worth of data per sheet. `read_excel()` loads just the first sheet in an Excel file by default. To load all sheets, pass in the keyword argument `sheet_name=None`. The result is a dictionary, where each key is the sheet name and each value is a DataFrame with the contents of the sheet.

[119]:
```
delays = pd.read_excel('/content/drive/MyDrive/Colab Notebooks/data/
 ↪ttc-subway-delay-data-2021.xlsx', sheet_name=None)
```

[120]:
```
type(delays)
```

[120]:
```
dict
```

To combine them all, we create an empty DataFrame, then loop through the dictionary items and use `pd.concat()` to append data. `concat()` takes a list of DataFrames to combine. Since we did not specify an index, row labels are numbers: the first row of each sheet has an index of 0, and so on. To reset row labels so that they are sequential again, we set `ignore_index=True`.

[121]:
```
# create an empty DataFrame
all_delays = pd.DataFrame()

for sheet_name, values in delays.items():
    # print the number of rows
    print(f'Adding {values.shape[0]} rows from {sheet_name}')
    # add each sheet to all_delays
    all_delays = pd.concat([all_delays, values],
                           axis=0,  # concatenate rows
                           ignore_index=True)  # reset row labels

all_delays.shape
```

```
Adding 1216 rows from January21
Adding 1245 rows from Feb 21
Adding 1167 rows from March '21
Adding 1170 rows from April '21
Adding 1168 rows from May '21
Adding 1265 rows from June 21
Adding 1244 rows from July 21
Adding 1273 rows from August 21
Adding 1433 rows from Sept 21
Adding 1560 rows from Oct 21
```

```
Adding 1771 rows from Nov 21
Adding 1858 rows from December21
```

[121]:
```
(16370, 10)
```

[122]:
```
all_delays.head()
```

[122]:

	Date	Time	Day	Station	Code	Min Delay	Min Gap	Bound	Line	Vehicle
0	2021-01-01	00:33	Friday	BLOOR STATION	MUPAA	0	0	N	YU	6046
1	2021-01-01	00:39	Friday	SHERBOURNE STATION	EUCO	5	9	E	BD	5250
2	2021-01-01	01:07	Friday	KENNEDY BD STATION	EUCD	5	9	E	BD	5249
3	2021-01-01	01:41	Friday	ST CLAIR STATION	MUIS	0	0	NaN	YU	0
4	2021-01-01	02:04	Friday	SHEPPARD WEST STATION	MUIS	0	0	NaN	YU	0

The TTC delays data includes a reason code for the delay. Code definitions, however, are in a separate Excel file, `ttc-subway-delay-codes.xlsx`. This file has been modified slightly to make it easier to work with. Codes are split between two tabs, so we will load both to a DataFrame, `delay_reasons`.

[123]:
```
dr = pd.read_excel('/content/drive/MyDrive/Colab Notebooks/data/
 ↪ttc-subway-delay-codes.xlsx', sheet_name=None)

delay_reasons = pd.DataFrame()
for sheet_name, values in dr.items():
    delay_reasons = pd.concat([delay_reasons, values],
                              axis=0,
                              ignore_index=True)

delay_reasons
```

[123]:

	RMENU CODE	CODE DESCRIPTION	SUB OR SRT
0	EUAC	Air Conditioning	SUB
1	EUAL	Alternating Current	SUB
2	EUATC	ATC RC&S Equipment	SUB
3	EUBK	Brakes	SUB
4	EUBO	Body	SUB
..	…	…	…
195	TRNOA	No Operator Immediately Available	SRT
196	TRO	Transportation Department - Other	SRT
197	TRSET	Train Controls Improperly Shut Down	SRT
198	TRST	Storm Trains	SRT

```
199       TRTC     Transit Control Related Problems        SRT

[200 rows x 3 columns]
```

We will rename the columns in both `all_delays` and `delay_reasons` so that we replace spaces with underscores as well as convert all letters to lowercase.

[124]:
```
def clean_names(string):
    return string.lower().replace(' ', '_')

print(list(delay_reasons))
print(list(all_delays))
delay_reasons = delay_reasons.rename(columns=clean_names)
all_delays = all_delays.rename(columns=clean_names)
print(list(delay_reasons))
print(list(all_delays))
```

```
['RMENU CODE', 'CODE DESCRIPTION', 'SUB OR SRT']
['Date', 'Time', 'Day', 'Station', 'Code', 'Min Delay', 'Min Gap', 'Bound',
'Line', 'Vehicle']
['rmenu_code', 'code_description', 'sub_or_srt']
['date', 'time', 'day', 'station', 'code', 'min_delay', 'min_gap', 'bound',
'line', 'vehicle']
```

3.6 Combining datasets: merging

Ideally, the delays data would include code descriptions. We can get descriptions into `all_delays` by *merging* in `delay_reasons`. Merging is analagous to joining in SQL databases. To merge two DataFrames, we pass them as arguments to the `pd.merge()`. Then, we specify `how` to merge the two DataFrames and what column names to merge `on`.

Let's review the `all_delays` and `delay_reasons` DataFrames. `code` is equivalent to `rmenu_code`. If we pass in `all_delays` as the first DataFrame, then it will be the left frame, and `delay_reasons` the right one. We want to keep all the delay records, even if there isn't a matching code in `delay_reasons`, so we will perform a left join.

[125]:
```
all_delays.head(2)
```

[125]:
```
        date   time     day             station   code  min_delay  min_gap  \
0 2021-01-01  00:33  Friday       BLOOR STATION  MUPAA          0        0
1 2021-01-01  00:39  Friday  SHERBOURNE STATION   EUCO          5        9

  bound line  vehicle
0     N   YU     6046
1     E   BD     5250
```

[126]:
```
delay_reasons.head(2)
```

[126]:
```
  rmenu_code     code_description sub_or_srt
0       EUAC     Air Conditioning        SUB
1       EUAL  Alternating Current        SUB
```

[127]:
```
delays_w_reasons = pd.merge(all_delays,
                            delay_reasons,
                            how='left',
                            left_on='code',
                            right_on='rmenu_code')
delays_w_reasons.head(3)
```

[127]:
```
        date   time     day             station   code  min_delay  min_gap  \
0 2021-01-01  00:33  Friday       BLOOR STATION  MUPAA          0        0
1 2021-01-01  00:39  Friday  SHERBOURNE STATION   EUCO          5        9
2 2021-01-01  01:07  Friday  KENNEDY BD STATION   EUCD          5        9

  bound line  vehicle rmenu_code  \
0     N   YU     6046      MUPAA
1     E   BD     5250       EUCO
2     E   BD     5249       EUCD

                                  code_description sub_or_srt
0  Passenger Assistance Alarm Activated - No Trou…        SUB
1                                         Couplers        SUB
2         Consequential Delay (2nd Delay Same Fault)        SUB
```

3.7 `drop()`

The resulting DataFrame has both our join columns, which is redundant. We can drop one with the `drop()` DataFrame method, passing in the column name(s) we want to drop in the `columns` keyword argument.

[128]:
```
delays_w_reasons = delays_w_reasons.drop(columns='rmenu_code')
delays_w_reasons.head(3)
```

[128]:
```
        date   time     day             station   code  min_delay  min_gap  \
0 2021-01-01  00:33  Friday       BLOOR STATION  MUPAA          0        0
1 2021-01-01  00:39  Friday  SHERBOURNE STATION   EUCO          5        9
2 2021-01-01  01:07  Friday  KENNEDY BD STATION   EUCD          5        9

  bound line  vehicle                                  code_description  \
0     N   YU     6046  Passenger Assistance Alarm Activated - No Trou…
1     E   BD     5250                                         Couplers
2     E   BD     5249        Consequential Delay (2nd Delay Same Fault)

  sub_or_srt
0        SUB
```

```
1        SUB
2        SUB
```

3.8 Creating new columns

Adding a column to a DataFrame looks like adding a key-value pair to a dictionary. At its simplest, we can assign a single value to repeat down a column.

[129]:
```
delays_w_reasons['year'] = 2021
delays_w_reasons['year'].unique()
```

[129]:
```
array([2021])
```

[130]:
```
delays_w_reasons
```

[130]:
```
            date   time     day                station   code  min_delay  \
0     2021-01-01  00:33  Friday          BLOOR STATION  MUPAA          0
1     2021-01-01  00:39  Friday     SHERBOURNE STATION   EUCO          5
2     2021-01-01  01:07  Friday     KENNEDY BD STATION   EUCD          5
3     2021-01-01  01:41  Friday       ST CLAIR STATION   MUIS          0
4     2021-01-01  02:04  Friday  SHEPPARD WEST STATION   MUIS          0
…            …    …      …                      …    …         …
16365 2021-12-31  01:10  Friday         MUSEUM STATION   SUUT          0
16366 2021-12-31  01:12  Friday          FINCH STATION   SUDP          5
16367 2021-12-31  01:21  Friday  EGLINTON WEST STATION  PUOPO          3
16368 2021-12-31  01:37  Friday  SHEPPARD WEST STATION   SUDP          0
16369 2021-12-31  07:00  Friday      DON MILLS STATION   TUSC          0

       min_gap bound line  vehicle  \
0            0     N   YU     6046
1            9     E   BD     5250
2            9     E   BD     5249
3            0   NaN   YU        0
4            0   NaN   YU        0
…        …   …  …      …
16365        0     N   YU     5591
16366       10     S   YU     5983
16367        8     N   YU     6046
16368        0     S   YU     5536
16369        0     E  SHP     6146

                                    code_description sub_or_srt  year
0      Passenger Assistance Alarm Activated - No Trou…        SUB  2021
1                                          Couplers         SUB  2021
2            Consequential Delay (2nd Delay Same Fault)        SUB  2021
3      Injured or ill Customer (In Station) - Transpo…        SUB  2021
4      Injured or ill Customer (In Station) - Transpo…        SUB  2021
```

```
…                                                  …        …   …
16365                        Unauthorized at Track Level        SUB  2021
16366                                  Disorderly Patron        SUB  2021
16367                  OPTO (COMMS) Train Door Monitoring        SUB  2021
16368                                  Disorderly Patron        SUB  2021
16369                               Operator Overspeeding        SUB  2021

[16370 rows x 13 columns]
```

We can also write an expression and store the resulting values in a new column.

[131]:
```
delays_w_reasons['hour_delay'] = round(delays_w_reasons['min_delay'] / 60, 2)
delays_w_reasons[['min_delay', 'hour_delay']].head()
```

[131]:
```
   min_delay  hour_delay
0          0        0.00
1          5        0.08
2          5        0.08
3          0        0.00
4          0        0.00
```

It is also possible to extract parts of a datetime column with the `dt` accessor.

[132]:
```
delays_w_reasons.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 16370 entries, 0 to 16369
Data columns (total 14 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   date              16370 non-null  datetime64[ns]
 1   time              16370 non-null  object
 2   day               16370 non-null  object
 3   station           16370 non-null  object
 4   code              16370 non-null  object
 5   min_delay         16370 non-null  int64
 6   min_gap           16370 non-null  int64
 7   bound             12119 non-null  object
 8   line              16318 non-null  object
 9   vehicle           16370 non-null  int64
 10  code_description  16048 non-null  object
 11  sub_or_srt        16048 non-null  object
 12  year              16370 non-null  int64
 13  hour_delay        16370 non-null  float64
dtypes: datetime64[ns](1), float64(1), int64(4), object(8)
memory usage: 1.9+ MB
```

[133]:
```
delays_w_reasons['month'] = delays_w_reasons['date'].dt.month
delays_w_reasons['month']
```

[133]:
```
0         1
1         1
2         1
3         1
4         1
         ..
16365    12
16366    12
16367    12
16368    12
16369    12
Name: month, Length: 16370, dtype: int64
```

It is possible to create a new integer column, `hour`, that contains the hour in which a delay occurred. Below we highlight two methods.

[134]:
```
# two ways to extract hour
# convert to time, then access hour
delays_w_reasons['hour'] = pd.to_datetime(delays_w_reasons['time'], format='%H:
 ↪%M').dt.hour

# split and take first part
delays_w_reasons['hour'] = delays_w_reasons['time'].str.split(':',␣
 ↪expand=True)[0] #expandbool expands the split strings into separate columns
delays_w_reasons['hour'] = delays_w_reasons['hour'].astype(int)
```

3.9 Filtering and selecting data

Let's take another look at the TTC subway delay data. There are only 4 subway lines in Toronto, but `describe()` reported 17 unique values.

[135]:
```
delays_w_reasons['line'].unique()
```

[135]:
```
array(['YU', 'BD', 'SHP', 'SRT', 'YU/BD', nan, 'YONGE/UNIVERSITY/BLOOR',
       'YU / BD', 'YUS', '999', 'SHEP', '36 FINCH WEST', 'YUS & BD',
       'YU & BD LINES', '35 JANE', '52', '41 KEELE', 'YUS/BD'],
      dtype=object)
```

Looks like some of the line values should be updated (YU/BD variants) and others should be dropped (e.g., 36 FINCH WEST, NaNs). Luckily there don't seem to be too many affected records, though the NaNs are not shown.

[136]:
```
delays_w_reasons['line'].value_counts()
```

[136]:
```
YU                          8880
BD                          5734
SHP                          657
SRT                          656
YU/BD                        346
YUS                           18
YU / BD                       17
YU & BD LINES                  1
41 KEELE                       1
52                             1
35 JANE                        1
999                            1
YUS & BD                       1
36 FINCH WEST                  1
SHEP                           1
YONGE/UNIVERSITY/BLOOR         1
YUS/BD                         1
Name: line, dtype: int64
```

3.9.1 .loc[] and isna()

To find the records with no line, we can use `.loc[]`, which lets us access rows and columns with either a boolean array or row/column labels.

In this case, the boolean array is the product of the `isna()` Series method.

[137]:
```
# access rows of data where line is NA
delays_w_reasons.loc[delays_w_reasons['line'].isna()]
```

[137]:

	date	time	day	station	code	min_delay	\
495	2021-01-13	15:22	Wednesday	FINCH WEST STATION	MUSAN	3	
513	2021-01-13	22:08	Wednesday	EGLINTON WEST STATION	PUMEL	0	
1044	2021-01-27	22:00	Wednesday	YONGE-UNIVERSITY AND B	MUO	0	
1045	2021-01-27	23:00	Wednesday	FINCH STATION	MUO	0	
1362	2021-02-04	01:45	Thursday	LAWRENCE STATION	TUSC	0	
1679	2021-02-11	01:12	Thursday	GREENWOOD CARHOUSE	MUIE	0	
2179	2021-02-22	08:27	Monday	BICHMOUNT DIVISION	MUIE	0	
2204	2021-02-22	22:33	Monday	BLOOR STATION	SUAP	4	
2206	2021-02-22	23:36	Monday	EGLINTON STATION	MUO	0	
3039	2021-03-17	05:15	Wednesday	INGLIS BUILDING	PUMEL	0	
3330	2021-03-24	19:13	Wednesday	INGLIS BUILDING	PUMEL	0	
3407	2021-03-26	09:03	Friday	WILSON YARD (SOUTH TAI	PUTO	0	
3557	2021-03-30	00:36	Tuesday	INGLIS BUILDING	PUMEL	0	
3944	2021-04-08	23:45	Thursday	DAVISVILLE YARD	MUIE	0	
4097	2021-04-13	10:57	Tuesday	SPADINA STATION	SUAE	0	
4119	2021-04-13	22:00	Tuesday	YONGE-UNIVERSITY AND B	MUO	0	
4336	2021-04-19	23:00	Monday	SHEPPARD WEST TO LAWRE	MUO	0	
4748	2021-04-29	22:00	Thursday	YONGE UNIVERSITY SPADI	MUO	0	

5312	2021-05-15	05:05	Saturday	SPADINA BD STATION	MUNCA	0
5448	2021-05-18	20:15	Tuesday	VAUGHAN MC STATION	MUWR	0
5484	2021-05-19	18:11	Wednesday	QUEEN'S QUAY STATION	PUMEL	0
5642	2021-05-23	23:19	Sunday	ST ANDREW STATION	SUDP	0
5685	2021-05-25	00:19	Tuesday	DUNDA WEST STATION	SUO	0
6042	2021-06-02	22:28	Wednesday	WARDEN STATION	MUIRS	0
6046	2021-06-02	00:56	Wednesday	BAY STATION	SUO	0
6540	2021-06-14	22:43	Monday	YONGE BD STATION	MUIS	0
6560	2021-06-15	07:15	Tuesday	SUBWAY OPERATIONS BUIL	PUMEL	0
7137	2021-06-28	01:03	Monday	COXWELL STATION	MUNCA	0
7766	2021-07-14	03:51	Wednesday	TRANSIT CONTROL CENTRE	PUSO	0
8889	2021-08-11	07:46	Wednesday	TRANSIT CONTROL	MUIE	0
9628	2021-08-29	15:49	Sunday	YORKDALE STATION	SUPOL	0
9629	2021-08-29	16:13	Sunday	YORK MILLS STATION	MUO	0
9780	2021-09-01	20:35	Wednesday	MAIN STREET AND UNION	MUO	0
9789	2021-09-01	22:14	Wednesday	UNION AND KENNEDY STAT	MUO	0
10336	2021-09-13	17:20	Monday	MCBRIEN BUILDING	SUO	0
10951	2021-09-26	15:50	Sunday	WILSON STATION	PUOPO	0
11223	2021-10-01	00:33	Friday	WELLESLEY STATION	SUDP	0
12533	2021-10-28	14:18	Thursday	VICTORIA PARK STATION	MUIS	0
12826	2021-11-02	12:22	Tuesday	GREENWOOD SHOP	MUIE	0
13007	2021-11-05	08:59	Friday	BLOOR STATION	MUIRS	0
13080	2021-11-06	18:41	Saturday	KENNEDY BD STATION	MUO	0
13273	2021-11-10	16:25	Wednesday	SUMMERHILL STATION	TUS	3
13402	2021-11-12	20:42	Friday	CLOSURES BUILDING	MUIE	0
13410	2021-11-12	00:02	Friday	TRANSIT CONTROL	MUO	0
14177	2021-11-25	21:14	Thursday	WILSON CARHOUSE	MUIE	0
14371	2021-11-29	05:10	Monday	GO PROTOCOL	MUO	0
14935	2021-12-08	06:00	Wednesday	TORONTO TRANSIT COMMIS	MUO	0
14952	2021-12-08	13:58	Wednesday	KIPLING STATION	MUIS	0
14967	2021-12-08	17:14	Wednesday	QUEEN'S PARK STATION	MUO	0
15581	2021-12-19	00:42	Sunday	WILSON CARHOUSE	MUO	0
15623	2021-12-20	16:23	Monday	YONGE-SHEPPARD (LINE 4	MUIRS	0
16332	2021-12-31	14:34	Friday	GO PROTOCOL	MUO	0

	min_gap	bound	line	vehicle	\
495	6	S	NaN	5751	
513	0	NaN	NaN	0	
1044	0	NaN	NaN	0	
1045	0	NaN	NaN	0	
1362	0	S	NaN	5596	
1679	0	NaN	NaN	0	
2179	0	NaN	NaN	0	
2204	9	N	NaN	6006	
2206	0	NaN	NaN	0	
3039	0	NaN	NaN	0	
3330	0	NaN	NaN	0	

```
3407         0   NaN  NaN        0
3557         0   NaN  NaN        0
3944         0   NaN  NaN        0
4097         0   NaN  NaN        0
4119         0   NaN  NaN        0
4336         0   NaN  NaN        0
4748         0   NaN  NaN        0
5312         0   NaN  NaN        0
5448         0   NaN  NaN        0
5484         0   NaN  NaN        0
5642         0   NaN  NaN        0
5685         0     E  NaN        0
6042         0   NaN  NaN        0
6046         0   NaN  NaN        0
6540         0   NaN  NaN        0
6560         0   NaN  NaN        0
7137         0   NaN  NaN        0
7766         0   NaN  NaN        0
8889         0   NaN  NaN        0
9628         0   NaN  NaN        0
9629         0   NaN  NaN        0
9780         0   NaN  NaN        0
9789         0   NaN  NaN        0
10336        0   NaN  NaN        0
10951        0     N  NaN     5471
11223        0   NaN  NaN        0
12533        0   NaN  NaN        0
12826        0   NaN  NaN        0
13007        0     S  NaN        0
13080        0   NaN  NaN        0
13273        6     N  NaN     6501
13402        0   NaN  NaN        0
13410        0   NaN  NaN        0
14177        0   NaN  NaN        0
14371        0   NaN  NaN        0
14935        0   NaN  NaN        0
14952        0   NaN  NaN        0
14967        0   NaN  NaN        0
15581        0   NaN  NaN        0
15623        0   NaN  NaN        0
16332        0   NaN  NaN        0

                                    code_description sub_or_srt  year  \
495                                Unsanitary Vehicle        SUB  2021
513                     Escalator/Elevator Incident        SUB  2021
1044                           Miscellaneous Other        SUB  2021
1045                           Miscellaneous Other        SUB  2021
```

1362		Operator Overspeeding	SUB	2021
1679		Injured Employee	SUB	2021
2179		Injured Employee	SUB	2021
2204		Assault / Patron Involved	SUB	2021
2206		Miscellaneous Other	SUB	2021
3039		Escalator/Elevator Incident	SUB	2021
3330		Escalator/Elevator Incident	SUB	2021
3407		T&S Other	SUB	2021
3557		Escalator/Elevator Incident	SUB	2021
3944		Injured Employee	SUB	2021
4097		Assault / Employee Involved	SUB	2021
4119		Miscellaneous Other	SUB	2021
4336		Miscellaneous Other	SUB	2021
4748		Miscellaneous Other	SUB	2021
5312		NaN	NaN	2021
5448		Work Refusal	SUB	2021
5484		Escalator/Elevator Incident	SUB	2021
5642		Disorderly Patron	SUB	2021
5685		Passenger Other	SUB	2021
6042	Injured or ill Customer (In Station) - Medical…		SUB	2021
6046		Passenger Other	SUB	2021
6540	Injured or ill Customer (In Station) - Transpo…		SUB	2021
6560		Escalator/Elevator Incident	SUB	2021
7137		NaN	NaN	2021
7766		S/E/C Department Other	SUB	2021
8889		Injured Employee	SUB	2021
9628		Held By Polce - Non-TTC Related	SUB	2021
9629		Miscellaneous Other	SUB	2021
9780		Miscellaneous Other	SUB	2021
9789		Miscellaneous Other	SUB	2021
10336		Passenger Other	SUB	2021
10951		OPTO (COMMS) Train Door Monitoring	SUB	2021
11223		Disorderly Patron	SUB	2021
12533	Injured or ill Customer (In Station) - Transpo…		SUB	2021
12826		Injured Employee	SUB	2021
13007	Injured or ill Customer (In Station) - Medical…		SUB	2021
13080		Miscellaneous Other	SUB	2021
13273		Crew Unable to Maintain Schedule	SUB	2021
13402		Injured Employee	SUB	2021
13410		Miscellaneous Other	SUB	2021
14177		Injured Employee	SUB	2021
14371		Miscellaneous Other	SUB	2021
14935		Miscellaneous Other	SUB	2021
14952	Injured or ill Customer (In Station) - Transpo…		SUB	2021
14967		Miscellaneous Other	SUB	2021
15581		Miscellaneous Other	SUB	2021
15623	Injured or ill Customer (In Station) - Medical…		SUB	2021

16332 Miscellaneous Other SUB 2021

	hour_delay	month	hour
495	0.05	1	15
513	0.00	1	22
1044	0.00	1	22
1045	0.00	1	23
1362	0.00	2	1
1679	0.00	2	1
2179	0.00	2	8
2204	0.07	2	22
2206	0.00	2	23
3039	0.00	3	5
3330	0.00	3	19
3407	0.00	3	9
3557	0.00	3	0
3944	0.00	4	23
4097	0.00	4	10
4119	0.00	4	22
4336	0.00	4	23
4748	0.00	4	22
5312	0.00	5	5
5448	0.00	5	20
5484	0.00	5	18
5642	0.00	5	23
5685	0.00	5	0
6042	0.00	6	22
6046	0.00	6	0
6540	0.00	6	22
6560	0.00	6	7
7137	0.00	6	1
7766	0.00	7	3
8889	0.00	8	7
9628	0.00	8	15
9629	0.00	8	16
9780	0.00	9	20
9789	0.00	9	22
10336	0.00	9	17
10951	0.00	9	15
11223	0.00	10	0
12533	0.00	10	14
12826	0.00	11	12
13007	0.00	11	8
13080	0.00	11	18
13273	0.05	11	16
13402	0.00	11	20
13410	0.00	11	0

```
14177        0.00     11    21
14371        0.00     11     5
14935        0.00     12     6
14952        0.00     12    13
14967        0.00     12    17
15581        0.00     12     0
15623        0.00     12    16
16332        0.00     12    14
```

`.loc[]` also lets us access data by label, with row conditions first and column conditions second.

[138]:
```
(delays_w_reasons.loc[delays_w_reasons['line'].isna(),  # filter rows
                      ['time', 'station', 'line']]  # get columns
                 .head())  # first 5 lines to save space
```

[138]:
```
       time                 station line
495   15:22      FINCH WEST STATION  NaN
513   22:08   EGLINTON WEST STATION  NaN
1044  22:00  YONGE-UNIVERSITY AND B  NaN
1045  23:00            FINCH STATION  NaN
1362  01:45         LAWRENCE STATION  NaN
```

3.9.2 `query()`

Alternatively, we can use the DataFrame `query()` method, which takes a filter condition as a string, and returns a DataFrame of records that met the condition. `query()` is slower than `loc[]`, but it can be easier to read.

[139]:
```
delays_w_reasons['line'].unique()
```

[139]:
```
array(['YU', 'BD', 'SHP', 'SRT', 'YU/BD', nan, 'YONGE/UNIVERSITY/BLOOR',
       'YU / BD', 'YUS', '999', 'SHEP', '36 FINCH WEST', 'YUS & BD',
       'YU & BD LINES', '35 JANE', '52', '41 KEELE', 'YUS/BD'],
      dtype=object)
```

[140]:
```
delays_w_reasons['line'].isna()
```

[140]:
```
0        False
1        False
2        False
3        False
4        False
         …
16365    False
16366    False
16367    False
16368    False
16369    False
```

```
Name: line, Length: 16370, dtype: bool
```

[141]:
```
# slower than .loc, but can be easier to read
delays_w_reasons.query('line.isna()', engine='python').head()
```

[141]:

	date	time	day	station	code	min_delay	min_gap	bound	line	vehicle	code_description	sub_or_srt	year	hour_delay	month	hour
495	2021-01-13	15:22	Wednesday	FINCH WEST STATION	MUSAN	3	6	S	NaN	5751	Unsanitary Vehicle	SUB	2021	0.05	1	15
513	2021-01-13	22:08	Wednesday	EGLINTON WEST STATION	PUMEL	0	0	NaN	NaN	0	Escalator/Elevator Incident	SUB	2021	0.00	1	22
1044	2021-01-27	22:00	Wednesday	YONGE-UNIVERSITY AND B	MUO	0	0	NaN	NaN	0	Miscellaneous Other	SUB	2021	0.00	1	22
1045	2021-01-27	23:00	Wednesday	FINCH STATION	MUO	0	0	NaN	NaN	0	Miscellaneous Other	SUB	2021	0.00	1	23
1362	2021-02-04	01:45	Thursday	LAWRENCE STATION	TUSC	0	0	S	NaN	5596	Operator Overspeeding	SUB	2021	0.00	2	1

3.9.3 `dropna()`

In this case, the number of records without lines is relatively small. Most do not have delay durations. Some appear to be at rail yards, i.e. not on a rail line. For our analysis, we may drop them with the `dropna()` DataFrame method. We can drop rows missing lines by passing a `subset`.

[142]:
```
delays_w_reasons = delays_w_reasons.dropna(subset=['line'])
```

3.9.4 Filtering data with `.loc[]` and `isin()`

We can use `.loc[]` to create a delays DataFrame without the invalid lines. To to this, we first create a list of values to exclude, then pass the list to the Series `isin()` method. Finally, we negate the expression, and assign the output back to `delays_w_reasons`.

Note: The negation operator here is `~`, not `!`. The `and` and `or` operators are different as well: `&` and `|` respectively.

[143]:
```
# set up filter list
filter_list = ['999', '36 FINCH WEST', '35 JANE', '52', '41 KEELE']
```

[144]:
```
# filter out records with invalid lines
```

```
delays_w_reasons = delays_w_reasons.loc[~delays_w_reasons['line'].
 ↪isin(filter_list)]
delays_w_reasons['line'].unique()
```

[144]:
```
array(['YU', 'BD', 'SHP', 'SRT', 'YU/BD', 'YONGE/UNIVERSITY/BLOOR',
       'YU / BD', 'YUS', 'SHEP', 'YUS & BD', 'YU & BD LINES', 'YUS/BD'],
      dtype=object)
```

3.9.5 Replacing values with `str.replace()`

To standardize the YU/BD values, we can replace the less common ones. One way to do this is by selecting the line Series and using `str.replace()`, like below for "YUS".

[145]:
```
delays_w_reasons['line'] = (delays_w_reasons['line']
                            .str.replace('YUS', 'YU'))
delays_w_reasons['line'].unique()
```

[145]:
```
array(['YU', 'BD', 'SHP', 'SRT', 'YU/BD', 'YONGE/UNIVERSITY/BLOOR',
       'YU / BD', 'SHEP', 'YU & BD', 'YU & BD LINES'], dtype=object)
```

Another is to assign "YU/BD" to values selected by `.loc[]`

[146]:
```
yubd_list = ['YONGE/UNIVERSITY/BLOOR',
             'YU / BD',
             'YU & BD',
             'YU & BD LINES']

# check the .loc[] selection
delays_w_reasons.loc[delays_w_reasons['line'].isin(yubd_list), 'line']
```

[146]:
```
590      YONGE/UNIVERSITY/BLOOR
852                     YU / BD
1137                    YU / BD
1628                    YU / BD
1672                    YU / BD
1700                    YU / BD
6725                    YU / BD
7469                    YU / BD
8034                    YU & BD
8301                    YU / BD
8341                    YU / BD
8463                    YU / BD
9164                    YU / BD
9541              YU & BD LINES
9839                    YU / BD
10792                   YU / BD
11119                   YU / BD
```

```
11299                    YU / BD
12128                    YU / BD
15574                    YU / BD
Name: line, dtype: object
```

[147]:
```
delays_w_reasons.loc[delays_w_reasons['line'].isin(yubd_list), 'line'] = 'YU/BD'
delays_w_reasons['line'].unique()
```

[147]:
```
array(['YU', 'BD', 'SHP', 'SRT', 'YU/BD', 'SHEP'], dtype=object)
```

3.10 Grouping

A core workflow in `pandas` is *split-apply-combine*: * **splitting** data into groups * **applying** a function to each group, such as calculating group sums, standardizing data, or filtering out some groups * **combining** the results into a data structure

This workflow starts by grouping data by calling the `groupby()` method. We'll pass in a column name or list of names to group by.

[148]:
```
line_groups = delays_w_reasons.groupby('line')
```

`groupby()` returns a grouped DataFrame that we can use to calculate groupwise statistics. The grouping column values become indexes, or row labels. **Note that this grouped DataFrame still references the original, so mutating one affects the other.**

[149]:
```
# how many hours of delays did each line have in 2021?
line_groups['hour_delay'].sum()
```

[149]:
```
line
BD       329.47
SHEP       0.00
SHP       28.43
SRT       57.82
YU       477.50
YU/BD      0.00
Name: hour_delay, dtype: float64
```

We can group by more than one column by passing a list into `groupby()`. Data is grouped in the order of column names.

[150]:
```
# group by line first, then reason code description
line_code_groups = delays_w_reasons.groupby(['line', 'code_description'])
```

[151]:
```
line_code_groups.size()
```

[151]:
```
line   code_description
BD     Air Conditioning                  13
       Alternating Current                2
       Assault / Employee Involved       69
```

```
        Assault / Patron Involved          167
        Body                                19
                                           ...
YU      Work Zone Problems - Signals         5
        Work Zone Problems - Track          29
        Yard/Carhouse Related Problems      15
YU/BD   Mainline Storage                     1
        Miscellaneous Other                366
Length: 313, dtype: int64
```

3.10.1 Chaining methods and `unstack()`ing

We can *chain* methods together for convenience and code readability. Here, we calculate the `size()` of each group, then `unstack()` the resulting Series by the first part of the row label, line. The `tail()` method is added to the end so that the output takes less screen space.

[152]:
```
# view the number of delays by reason and line
line_code_groups.size().unstack(0).tail()
```

[152]:
```
line                              BD  SHEP  SHP  SRT    YU  YU/BD
code_description
Work Refusal                     4.0   NaN  1.0  NaN  12.0    NaN
Work Vehicle                     3.0   NaN  NaN  NaN   7.0    NaN
Work Zone Problems - Signals     4.0   NaN  4.0  NaN   5.0    NaN
Work Zone Problems - Track      12.0   NaN  NaN  NaN  29.0    NaN
Yard/Carhouse Related Problems  17.0   NaN  NaN  NaN  15.0    NaN
```

3.10.2 `agg()`regating

So far, we have applied one function at a time. The `agg()` DataFrame method lets us apply multiple functions on different columns at once.

`agg()`'s argument syntax is a little unusual. It follows this pattern:

```
DataFrame.agg(agg_colname=('column_to_aggregate', 'aggregation_function_name'),
              agg_colname2=('col_to_agg2', 'agg_func_name'))
```

[153]:
```
delay_summary = (delays_w_reasons
                 .groupby('date')
                 .agg(delay_count=('station', 'count'),
                      total_delay_min=('min_delay', 'sum'),
                      mean_delay_min=('min_delay', 'mean')))

delay_summary.head()
```

[153]:
```
            delay_count  total_delay_min  mean_delay_min
date
2021-01-01           36              159        4.416667
```

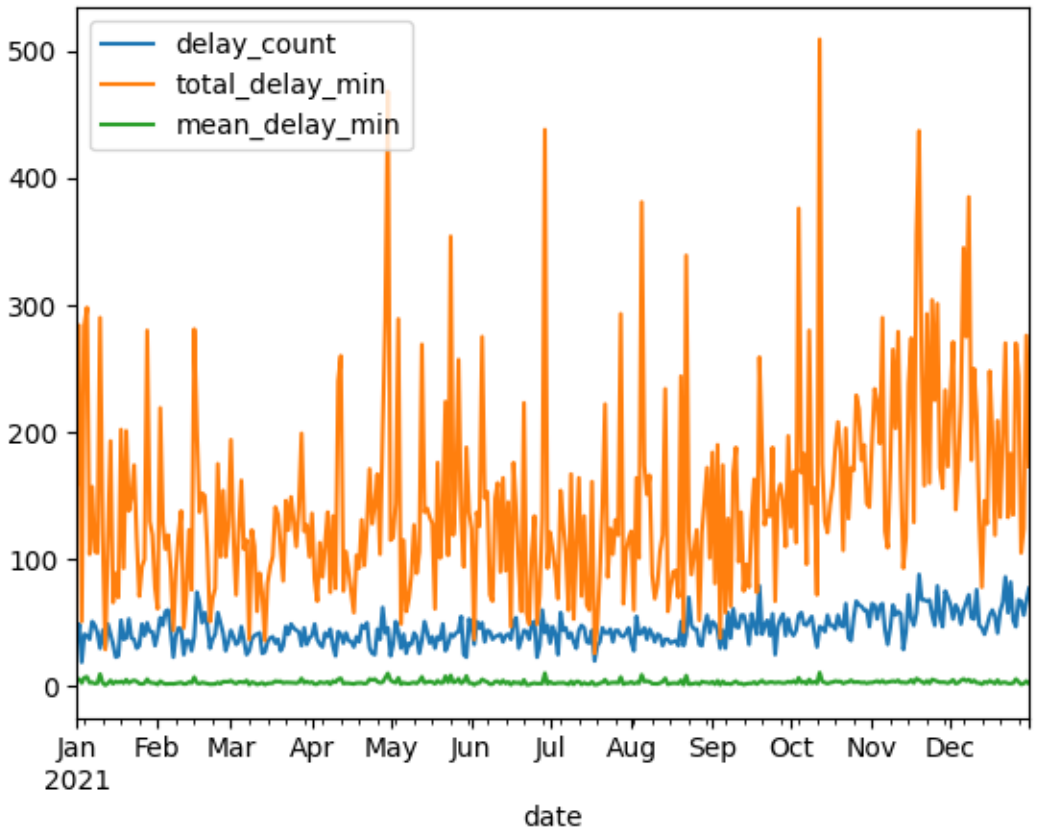
2021-01-02	49	284	5.795918
2021-01-03	19	51	2.684211
2021-01-04	41	284	6.926829
2021-01-05	40	298	7.450000

3.11 Plotting

The summary table we just generated can be easily plotted within `pandas`. Since the index contains dates, `pandas` automatically knows to plot values as time series data, with the dates in the x-axis – we just have to call the `plot()` method.

[154]:
```
delay_summary.plot()
```

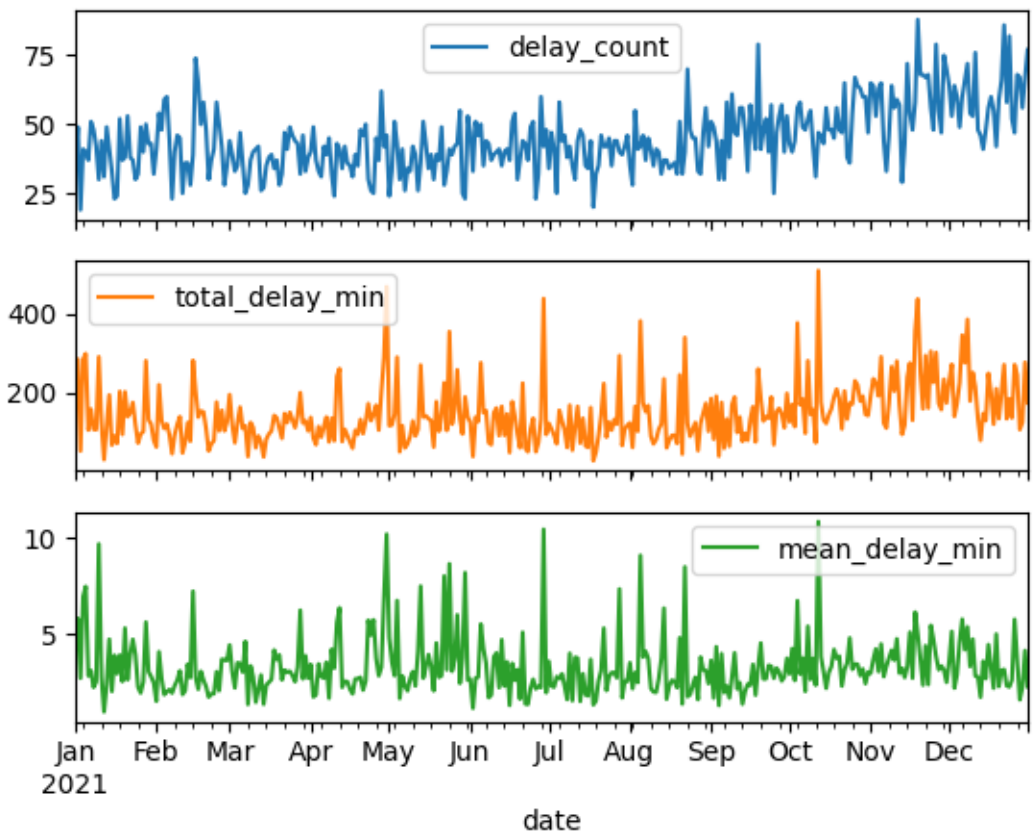
[154]:
```
<Axes: xlabel='date'>
```



To create a separate plot for each column, we can specify `subplots=True`

[155]:
```
delay_summary.plot(subplots=True)
```

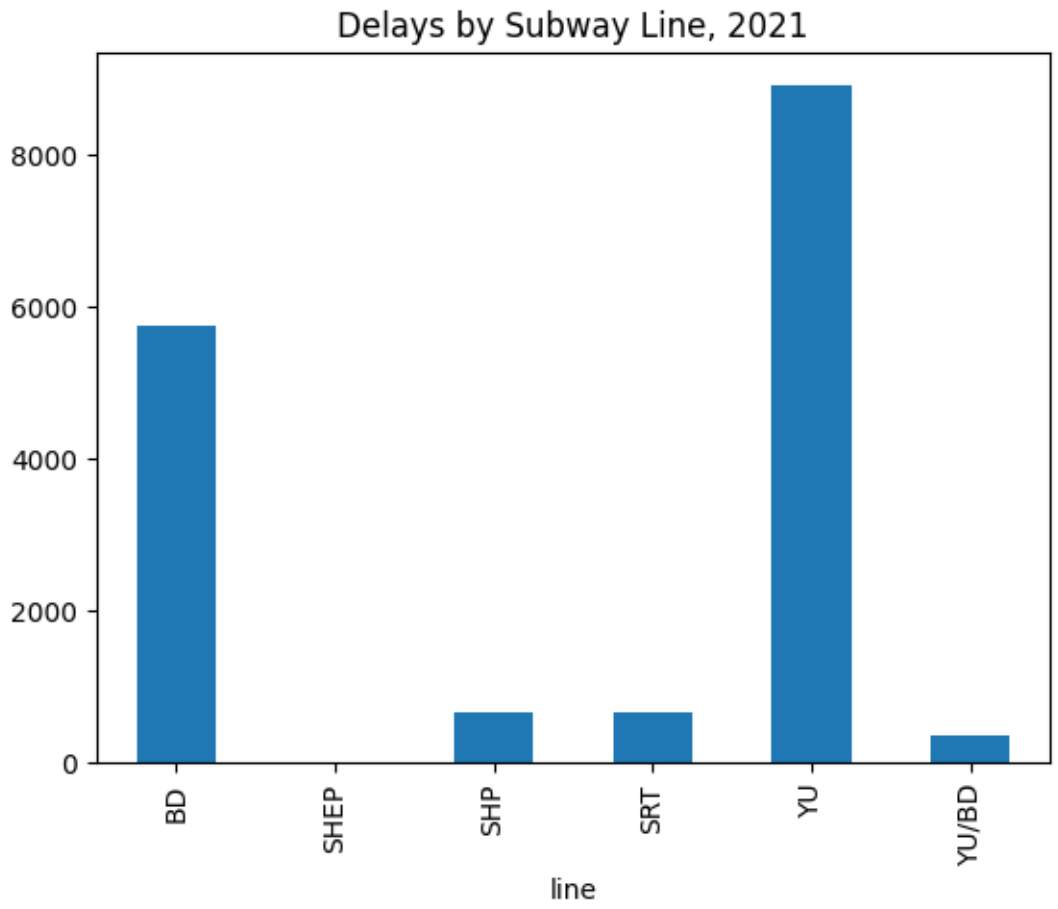
[155]:
```
array([<Axes: xlabel='date'>, <Axes: xlabel='date'>,
       <Axes: xlabel='date'>], dtype=object)
```



We can plot other aggregations too. Below, we use `line_groups` and calculate the size of each group, i.e., the number of delays reported on each line. Then we plot the data, telling `pandas` that the plot `kind` should be a bar graph, with TTC lines should in the `x`-axis. We also pass in a title.

[156]:
```
(line_groups
 .size()
 .plot(x='line',
       kind='bar',
       title='Delays by Subway Line, 2021'))
```

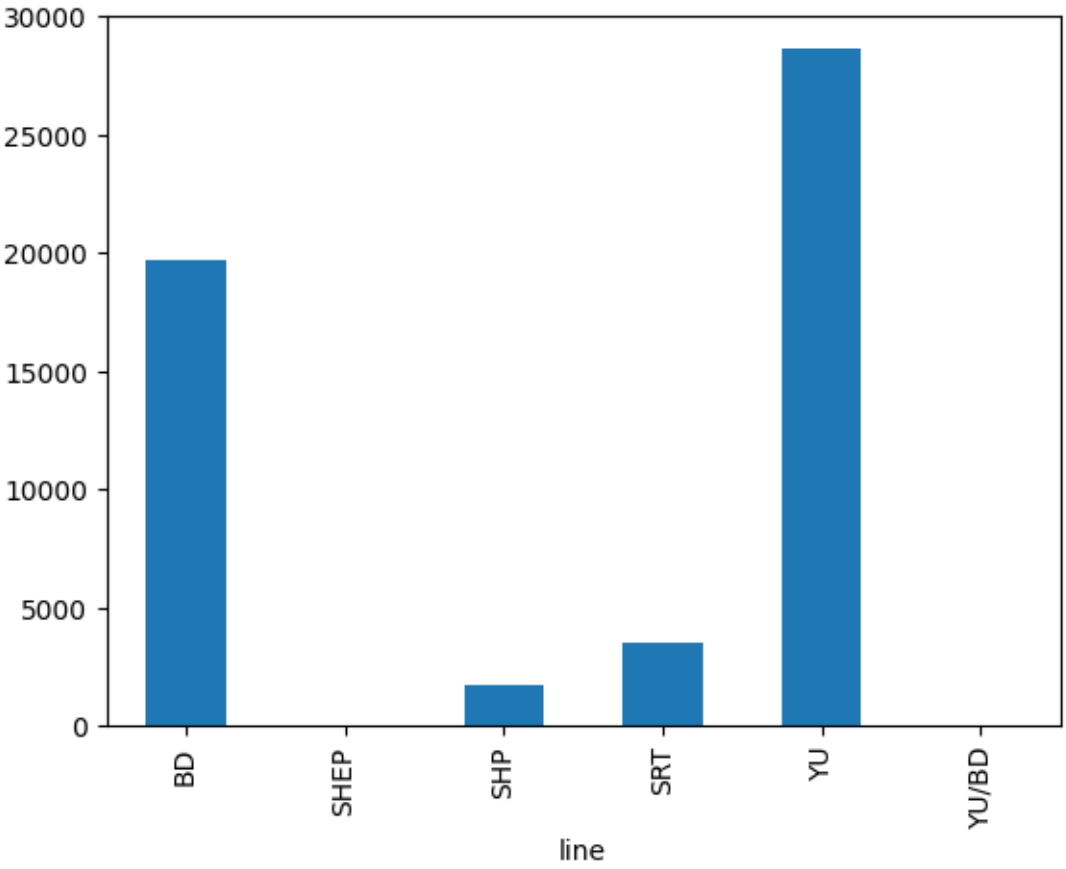
[156]:
```
<Axes: title={'center': 'Delays by Subway Line, 2021'}, xlabel='line'>
```



It is possible to sum up and plot the total delay time, in hours, by line.

[157]:
```
(delays_w_reasons
 .groupby('line')['min_delay']
 .sum()
 .plot(x='line', kind='bar'))
```

[157]: <Axes: xlabel='line'>



4 Writing to file

4.1 Exporting DataFrames

DataFrames have `to_[file format]()` methods, analagous to `pandas` read functions. The counterpart to `pd.read_csv()`, for instance, is `DataFrame.to_csv()`. The export methods generally take a file path to save to as their first argument. Additional arguments vary a bit by export format, but `index` is a common useful one. It takes a boolean of whether to write the index to file – set it to `False` if the index is the numbered default.

Two additional useful parameters in `DataFrame.to_csv()` and `DataFrame.to_excel()` are `na_rep`, which takes a string to use for null values, and `columns`, which lets us write out a subset of columns.

[158]:
```
# write delays_w_reasons to an Excel file
delays_w_reasons.to_excel('/content/drive/MyDrive/Colab Notebooks/data/
  ↪ttc_subway_delays_w_reasons.xlsx', index=False)
```

5 More wrangling

5.1 Neighbourhood Profiles

The bike theft data includes neighbourhood identifiers. These neighbourhoods are designated by City of Toronto, which publishes neighbourhood demographic profiles. Let's get this data to start investigating if neighbourhoods with more bike theft reports simply have higher populations. In the process, we will reinforce what we learned so far. We will also learn about two last ways to reshape data: `melt()`, which rearranges data from a wide format to a long format; and `pivot()`, which reorganizes data based on index and column values.

5.2 Getting data

Let's load the neighbourhood data and explore it.

[159]:
```
profiles = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/data/
↪neighbourhood-profiles-2016-140-model.csv')
```

[160]:
```
profiles.shape
```

[160]:
```
(2383, 146)
```

The neighbourhood profiles are in an unusual format. Neighbourhood names are in the columns, while attribute fields are rows, and there are thousands of attributes.

[161]:
```
profiles.head()
```

[161]:
```
   _id                  Category                     Topic  \
0    1  Neighbourhood Information  Neighbourhood Information
1    2  Neighbourhood Information  Neighbourhood Information
2    3                 Population   Population and dwellings
3    4                 Population   Population and dwellings
4    5                 Population   Population and dwellings

                     Data Source              Characteristic  \
0                City of Toronto         Neighbourhood Number
1                City of Toronto         TSNS2020 Designation
2  Census Profile 98-316-X2016001             Population, 2016
3  Census Profile 98-316-X2016001             Population, 2011
4  Census Profile 98-316-X2016001  Population Change 2011-2016

  City of Toronto Agincourt North Agincourt South-Malvern West  \
0             NaN             129                           128
1             NaN  No Designation              No Designation
2       2,731,571          29,113                      23,757
3       2,615,060          30,279                      21,988
4           4.50%          -3.90%                       8.00%

        Alderwood          Annex Banbury-Don Mills  Bathurst Manor  \
```

0	20	95	42	34
1	No Designation	No Designation	No Designation	No Designation
2	12,054	30,526	27,695	15,873
3	11,904	29,177	26,918	15,434
4	1.30%	4.60%	2.90%	2.80%

	Bay Street Corridor	Bayview Village	Bayview Woods-Steeles	\
0	76	52	49	
1	No Designation	No Designation	No Designation	
2	25,797	21,396	13,154	
3	19,348	17,671	13,530	
4	33.30%	21.10%	-2.80%	

	Bedford Park-Nortown	Beechborough-Greenbrook	Bendale	\
0	39	112	127	
1	No Designation	NIA	No Designation	
2	23,236	6,577	29,960	
3	23,185	6,488	27,876	
4	0.20%	1.40%	7.50%	

	Birchcliffe-Cliffside	Black Creek	Blake-Jones	Briar Hill-Belgravia	\
0	122	24	69	108	
1	No Designation	NIA	No Designation	No Designation	
2	22,291	21,737	7,727	14,257	
3	21,856	22,057	7,763	14,302	
4	2.00%	-1.50%	-0.50%	-0.30%	

	Bridle Path-Sunnybrook-York Mills	Broadview North	Brookhaven-Amesbury	\
0	41	57	30	
1	No Designation	No Designation	No Designation	
2	9,266	11,499	17,757	
3	8,713	11,563	17,787	
4	6.30%	-0.60%	-0.20%	

	Cabbagetown-South St. James Town	Caledonia-Fairbank	Casa Loma	\
0	71	109	96	
1	No Designation	No Designation	No Designation	
2	11,669	9,955	10,968	
3	12,053	9,851	10,487	
4	-3.20%	1.10%	4.60%	

	Centennial Scarborough	Church-Yonge Corridor	Clairlea-Birchmount	\
0	133	75	120	
1	No Designation	No Designation	No Designation	
2	13,362	31,340	26,984	
3	13,093	28,349	24,770	
4	2.10%	10.60%	8.90%	

	Clanton Park	Cliffcrest	Corso Italia-Davenport	Danforth
0	33	123	92	66
1	No Designation	No Designation	No Designation	No Designation
2	16,472	15,935	14,133	9,666
3	14,612	15,703	13,743	9,444
4	12.70%	1.50%	2.80%	2.40%

	Danforth East York	Don Valley Village	Dorset Park
0	59	47	126
1	No Designation	No Designation	Emerging Neighbourhood
2	17,180	27,051	25,003
3	16,712	26,739	24,363
4	2.80%	1.20%	2.60%

	Dovercourt-Wallace	Emerson-Junction	Downsview-Roding-CFB	Dufferin Grove
0	93	26	83	
1	No Designation	NIA	No Designation	
2	36,625	35,052	11,785	
3	34,631	34,659	11,449	
4	5.80%	1.10%	2.90%	

	East End-Danforth	Edenbridge-Humber Valley	Eglinton East	Elms-Old Rexdale
0	62	9	138	5
1	No Designation	No Designation	NIA	NIA
2	21,381	15,535	22,776	9,456
3	20,839	14,943	22,829	9,550
4	2.60%	4.00%	-0.20%	-1.00%

	Englemount-Lawrence	Eringate-Centennial-West Deane	Etobicoke West Mall
0	32	11	13
1	Emerging Neighbourhood	No Designation	No Designation
2	22,372	18,588	11,848
3	22,086	18,810	10,927
4	1.30%	-1.20%	8.40%

	Flemingdon Park	Forest Hill North	Forest Hill South	Glenfield-Jane Heights
0	44	102	101	25
1	NIA	No Designation	No Designation	NIA
2	21,933	12,806	10,732	30,491
3	22,168	12,474	10,926	31,390
4	-1.10%	2.70%	-1.80%	-2.90%

	Greenwood-Coxwell	Guildwood	Henry Farm	High Park North
0	65	140	53	88
1	No Designation	No Designation	No Designation	No Designation
2	14,417	9,917	15,723	22,162

3	14,083	9,816	11,333	21,292
4	2.40%	1.00%	38.70%	4.10%

	High Park-Swansea	Highland Creek	Hillcrest Village	\
0	87	134	48	
1	No Designation	No Designation	No Designation	
2	23,925	12,494	16,934	
3	21,740	13,097	17,656	
4	10.10%	-4.60%	-4.10%	

	Humber Heights-Westmount	Humber Summit	Humbermede	Humewood-Cedarvale	\
0	8	21	22	106	
1	Emerging Neighbourhood	NIA	NIA	No Designation	
2	10,948	12,416	15,545	14,365	
3	10,583	12,525	15,853	14,108	
4	3.40%	-0.90%	-1.90%	1.80%	

	Ionview	Islington-City Centre West	Junction Area	Keelesdale-Eglinton West	\
0	125	14	90	110	
1	NIA	No Designation	No Designation	NIA	
2	13,641	43,965	14,366	11,058	
3	13,091	38,084	14,027	10,638	
4	4.20%	15.40%	2.40%	3.90%	

	Kennedy Park	Kensington-Chinatown	Kingsview Village-The Westway	\
0	124	78	6	
1	NIA	No Designation	NIA	
2	17,123	17,945	22,000	
3	17,058	18,495	21,723	
4	0.40%	-3.00%	1.30%	

	Kingsway South	Lambton Baby Point	L'Amoreaux	Lansing-Westgate	\
0	15	114	117	38	
1	No Designation	No Designation	Emerging Neighbourhood	No Designation	
2	9,271	7,985	43,993	16,164	
3	9,170	7,921	44,919	14,642	
4	1.10%	0.80%	-2.10%	10.40%	

	Lawrence Park North	Lawrence Park South	Leaside-Bennington	Little Portugal	\
0	105	103	56	84	
1	No Designation	No Designation	No Designation	No Designation	
2	14,607	15,179	16,828	15,559	
3	14,541	15,070	17,011	12,050	
4	0.50%	0.70%	-1.10%	29.10%	

	Long Branch	Malvern	Maple Leaf	Markland Wood	\
0	19	132	29	12	

1	No Designation	Emerging Neighbourhood	No Designation	No Designation
2	10,084	43,794	10,111	10,554
3	9,632	45,086	10,197	10,436
4	4.70%	-2.90%	-0.80%	1.10%

	Milliken	Mimico (includes Humber Bay Shores)	Morningside	\
0	130	17	135	
1	No Designation	No Designation	NIA	
2	26,572	33,964	17,455	
3	27,167	26,541	17,587	
4	-2.20%	28.00%	-0.80%	

	Moss Park	Mount Dennis	Mount Olive-Silverstone-Jamestown	\
0	73	115	2	
1	No Designation	NIA	NIA	
2	20,506	13,593	32,954	
3	16,306	13,145	32,788	
4	25.80%	3.40%	0.50%	

	Mount Pleasant East	Mount Pleasant West	New Toronto	Newtonbrook East	\
0	99	104	18	50	
1	No Designation	No Designation	No Designation	No Designation	
2	16,775	29,658	11,463	16,097	
3	15,982	28,593	10,900	16,423	
4	5.00%	3.70%	5.20%	-2.00%	

	Newtonbrook West	Niagara	North Riverdale	North St. James Town	\
0	36	82	68	74	
1	No Designation	No Designation	No Designation	No Designation	
2	23,831	31,180	11,916	18,615	
3	23,052	21,274	12,191	17,832	
4	3.40%	46.60%	-2.30%	4.40%	

	Oakridge	Oakwood Village	O'Connor-Parkview	Old East York	\
0	121	107	54	58	
1	NIA	No Designation	No Designation	No Designation	
2	13,845	21,210	18,675	9,233	
3	13,497	21,073	18,316	9,118	
4	2.60%	0.70%	2.00%	1.30%	

	Palmerston-Little Italy	Parkwoods-Donalda	Pelmo Park-Humberlea	\
0	80	45	23	
1	No Designation	No Designation	No Designation	
2	13,826	34,805	10,722	
3	13,746	34,617	8,710	
4	0.60%	0.50%	23.10%	

	Playter Estates-Danforth	Pleasant View	Princess-Rosethorn	Regent Park
0	67	46	10	72
1	No Designation	No Designation	No Designation	NIA
2	7,804	15,818	11,051	10,803
3	7,653	16,144	11,197	10,007
4	2.00%	-2.00%	-1.30%	8.00%

	Rexdale-Kipling	Rockcliffe-Smythe	Roncesvalles	Rosedale-Moore Park
0	4	111	86	98
1	No Designation	NIA	No Designation	No Designation
2	10,529	22,246	14,974	20,923
3	10,488	22,267	15,050	20,631
4	0.40%	-0.10%	-0.50%	1.40%

	Rouge	Runnymede-Bloor West Village	Rustic	Scarborough Village
0	131	89	28	139
1	No Designation	No Designation	NIA	NIA
2	46,496	10,070	9,941	16,724
3	45,912	9,632	9,951	16,609
4	1.30%	4.50%	-0.10%	0.70%

	South Parkdale	South Riverdale	St.Andrew-Windfields	Steeles
0	85	70	40	116
1	NIA	No Designation	No Designation	Emerging Neighbourhood
2	21,849	27,876	17,812	24,623
3	21,251	25,642	17,958	25,017
4	2.80%	8.70%	-0.80%	-1.60%

	Stonegate-Queensway	Tam O'Shanter-Sullivan	Taylor-Massey	The Beaches
0	16	118	61	63
1	No Designation	No Designation	NIA	No Designation
2	25,051	27,446	15,683	21,567
3	24,691	27,398	15,594	21,130
4	1.50%	0.20%	0.60%	2.10%

	Thistletown-Beaumond Heights	Thorncliffe Park	Trinity-Bellwoods
0	3	55	81
1	NIA	NIA	No Designation
2	10,360	21,108	16,556
3	10,138	19,225	16,802
4	2.20%	9.80%	-1.50%

	University	Victoria Village	Waterfront Communities-The Island
0	79	43	77
1	No Designation	NIA	No Designation
2	7,607	17,510	65,913
3	7,782	17,182	43,361

```
4          -2.20%            1.90%                              52.00%

  West Hill West Humber-Clairville     Westminster-Branson  Weston  \
0       136                      1                      35     113
1       NIA         No Designation  Emerging Neighbourhood     NIA
2    27,392                 33,312                  26,274  17,992
3    26,547                 34,100                  25,446  18,170
4     3.20%                 -2.30%                   3.30%  -1.00%

  Weston-Pelham Park Wexford/Maryvale Willowdale East Willowdale West  \
0                 91              119              51              37
1                NIA   No Designation  No Designation  No Designation
2             11,098           27,917          50,434          16,936
3             12,010           27,018          45,041          15,004
4             -7.60%            3.30%          12.00%          12.90%

  Willowridge-Martingrove-Richview  Woburn Woodbine Corridor Woodbine-Lumsden  \
0                                7     137               64               60
1                   No Designation     NIA    No Designation   No Designation
2                           22,156  53,485           12,541            7,865
3                           21,343  53,350           11,703            7,826
4                            3.80%   0.30%            7.20%            0.50%

         Wychwood  Yonge-Eglinton  Yonge-St.Clair York University Heights  \
0              94             100              97                      27
1  No Designation  No Designation  No Designation                     NIA
2          14,349          11,817          12,528                  27,593
3          13,986          10,578          11,652                  27,713
4           2.60%          11.70%           7.50%                  -0.40%

      Yorkdale-Glen Park
0                     31
1  Emerging Neighbourhood
2                 14,804
3                 14,687
4                  0.80%
```

Because of the layout and formatting characters, all of the numeric values have been read in as text data. It also looks like the characteristics are not unique.

[162]:
```
profiles.dtypes.value_counts()
```

[162]:
```
object    145
int64       1
dtype: int64
```

[163]:
```
len(profiles['Characteristic'].unique())
```

[163]:
```
1651
```

5.3 Removing extra whitespace

The characteristic values contain extra whitespace. Let's remove the whitespace up with `str.strip()`.

[164]:
```
# the whitespace is easier to see in a list than a Series
list(profiles['Characteristic'][95:100])
```

[164]:
```
['    Female parent',
 '    Male parent',
 'Couple census families in private households',
 '  Couples with children',
 '    1 child']
```

[165]:
```
profiles['Characteristic'] = profiles['Characteristic'].str.strip()

# get the first 10 characteristics
list(profiles['Characteristic'][95:100])
```

[165]:
```
['Female parent',
 'Male parent',
 'Couple census families in private households',
 'Couples with children',
 '1 child']
```

5.4 Subsetting data

1651 characteristics is still a lot. Let's check out the categories to understand the areas covered.

[166]:
```
profiles['Category'].unique()
```

[166]:
```
array(['Neighbourhood Information', 'Population',
       'Families, households and marital status', 'Language', 'Income',
       'Immigration and citizenship', 'Visible minority', 'Ethnic origin',
       'Aboriginal peoples', 'Education', 'Housing', 'Language of work',
       'Labour', 'Journey to work', 'Mobility'], dtype=object)
```

"Journey to work" sounds relevant. We can use `.loc[]` to get the rows in that category, then select the Topic column and get its unique values.

[167]:
```
profiles.loc[profiles['Category'] == 'Journey to work']['Topic'].unique()
```

[167]:
```
array(['Commuting destination', 'Main mode of commuting',
       'Commuting duration', 'Time leaving for work'], dtype=object)
```

The "Population and dwellings" topic we saw in the DataFrame head looked promising as well. Let's check out the Characteristics in that topic.

[168]:
```
profiles.loc[profiles['Topic'] == 'Population and dwellings']['Characteristic'].
↪unique()
```

[168]:
```
array(['Population, 2016', 'Population, 2011',
       'Population Change 2011-2016', 'Total private dwellings',
       'Private dwellings occupied by usual residents',
       'Population density per square kilometre',
       'Land area in square kilometres'], dtype=object)
```

Now that we know what topics we're interested in, let's create a subset DataFrame limited to them. We'll use the `copy()` DataFrame method to leave the original data untouched.

[169]:
```
topics = ['Neighbourhood Information', 'Population and dwellings', 'Main mode␣
↪of commuting']

# make sure it's an independent copy
profiles_subset = profiles.copy()

# get just the topics we're interested in
profiles_subset = profiles_subset.loc[profiles['Topic'].isin(topics)]
profiles_subset.shape
```

[169]:
```
(16, 146)
```

5.5 Reshaping data with `melt()`

Now we're ready to reshape our data. We can `drop()` the ID, data source, and category columns now.

To `melt()` a DataFrame, we specify `id_vars` – the columns to keep as identifiers. All other columns are 'melted' into a new `variable` column. The values at DataFrame[id_vars, variable_col] move into a `value` column. We can change the names of the variable and value columns with the `var_name` and `value_name` arguments.

The `pandas` documentation provides an illustrative example.

Let's `melt()` the profiles subset. We'll keep `Topic` and `Characteristic` as our `id_vars`. This will melt the neighbourhood names into the `variable` column, which we'll rename `Neighbourhood`.

[170]:
```
profiles_subset.head()
```

[170]:

	_id	Category	Topic	\
0	1	Neighbourhood Information	Neighbourhood Information	
1	2	Neighbourhood Information	Neighbourhood Information	
2	3	Population	Population and dwellings	
3	4	Population	Population and dwellings	

```
4    5                Population   Population and dwellings

                     Data Source              Characteristic  \
0                City of Toronto         Neighbourhood Number
1                City of Toronto         TSNS2020 Designation
2  Census Profile 98-316-X2016001             Population, 2016
3  Census Profile 98-316-X2016001             Population, 2011
4  Census Profile 98-316-X2016001  Population Change 2011-2016

  City of Toronto Agincourt North Agincourt South-Malvern West  \
0             NaN             129                          128
1             NaN  No Designation              No Designation
2       2,731,571          29,113                      23,757
3       2,615,060          30,279                      21,988
4           4.50%          -3.90%                       8.00%

       Alderwood           Annex Banbury-Don Mills  Bathurst Manor  \
0             20              95                42              34
1  No Designation  No Designation    No Designation  No Designation
2         12,054          30,526            27,695          15,873
3         11,904          29,177            26,918          15,434
4          1.30%           4.60%             2.90%           2.80%

  Bay Street Corridor Bayview Village Bayview Woods-Steeles  \
0                  76             52                    49
1      No Designation  No Designation        No Designation
2              25,797         21,396                13,154
3              19,348         17,671                13,530
4              33.30%         21.10%                -2.80%

  Bedford Park-Nortown Beechborough-Greenbrook         Bendale  \
0                   39                     112             127
1       No Designation                     NIA  No Designation
2               23,236                   6,577          29,960
3               23,185                   6,488          27,876
4                0.20%                   1.40%           7.50%

  Birchcliffe-Cliffside Black Creek     Blake-Jones Briar Hill-Belgravia  \
0                   122          24              69                  108
1        No Designation         NIA  No Designation       No Designation
2                22,291      21,737           7,727              14,257
3                21,856      22,057           7,763              14,302
4                 2.00%      -1.50%          -0.50%              -0.30%

  Bridle Path-Sunnybrook-York Mills Broadview North Brookhaven-Amesbury  \
0                                41              57                  30
1                  No Designation  No Designation      No Designation
```

2	9,266	11,499	17,757
3	8,713	11,563	17,787
4	6.30%	-0.60%	-0.20%

	Cabbagetown-South St. James Town	Caledonia-Fairbank	Casa Loma
0	71	109	96
1	No Designation	No Designation	No Designation
2	11,669	9,955	10,968
3	12,053	9,851	10,487
4	-3.20%	1.10%	4.60%

	Centennial Scarborough	Church-Yonge Corridor	Clairlea-Birchmount
0	133	75	120
1	No Designation	No Designation	No Designation
2	13,362	31,340	26,984
3	13,093	28,349	24,770
4	2.10%	10.60%	8.90%

	Clanton Park	Cliffcrest	Corso Italia-Davenport	Danforth
0	33	123	92	66
1	No Designation	No Designation	No Designation	No Designation
2	16,472	15,935	14,133	9,666
3	14,612	15,703	13,743	9,444
4	12.70%	1.50%	2.80%	2.40%

	Danforth East York	Don Valley Village	Dorset Park
0	59	47	126
1	No Designation	No Designation	Emerging Neighbourhood
2	17,180	27,051	25,003
3	16,712	26,739	24,363
4	2.80%	1.20%	2.60%

	Dovercourt-Wallace Emerson-Junction	Downsview-Roding-CFB	Dufferin Grove
0	93	26	83
1	No Designation	NIA	No Designation
2	36,625	35,052	11,785
3	34,631	34,659	11,449
4	5.80%	1.10%	2.90%

	East End-Danforth	Edenbridge-Humber Valley	Eglinton East	Elms-Old Rexdale
0	62	9	138	5
1	No Designation	No Designation	NIA	NIA
2	21,381	15,535	22,776	9,456
3	20,839	14,943	22,829	9,550
4	2.60%	4.00%	-0.20%	-1.00%

	Englemount-Lawrence	Eringate-Centennial-West	Deane	Etobicoke West Mall

0	32	11	13	
1	Emerging Neighbourhood	No Designation	No Designation	
2	22,372	18,588	11,848	
3	22,086	18,810	10,927	
4	1.30%	-1.20%	8.40%	

	Flemingdon Park	Forest Hill North	Forest Hill South	Glenfield-Jane Heights	\
0	44	102	101	25	
1	NIA	No Designation	No Designation	NIA	
2	21,933	12,806	10,732	30,491	
3	22,168	12,474	10,926	31,390	
4	-1.10%	2.70%	-1.80%	-2.90%	

	Greenwood-Coxwell	Guildwood	Henry Farm	High Park North	\
0	65	140	53	88	
1	No Designation	No Designation	No Designation	No Designation	
2	14,417	9,917	15,723	22,162	
3	14,083	9,816	11,333	21,292	
4	2.40%	1.00%	38.70%	4.10%	

	High Park-Swansea	Highland Creek	Hillcrest Village	\
0	87	134	48	
1	No Designation	No Designation	No Designation	
2	23,925	12,494	16,934	
3	21,740	13,097	17,656	
4	10.10%	-4.60%	-4.10%	

	Humber Heights-Westmount	Humber Summit	Humbermede	Humewood-Cedarvale	\
0	8	21	22	106	
1	Emerging Neighbourhood	NIA	NIA	No Designation	
2	10,948	12,416	15,545	14,365	
3	10,583	12,525	15,853	14,108	
4	3.40%	-0.90%	-1.90%	1.80%	

	Ionview	Islington-City Centre West	Junction Area	Keelesdale-Eglinton West	\
0	125	14	90	110	
1	NIA	No Designation	No Designation	NIA	
2	13,641	43,965	14,366	11,058	
3	13,091	38,084	14,027	10,638	
4	4.20%	15.40%	2.40%	3.90%	

	Kennedy Park	Kensington-Chinatown	Kingsview Village-The Westway	\
0	124	78	6	
1	NIA	No Designation	NIA	
2	17,123	17,945	22,000	
3	17,058	18,495	21,723	
4	0.40%	-3.00%	1.30%	

	Kingsway South	Lambton Baby Point	L'Amoreaux	Lansing-Westgate
0	15	114	117	38
1	No Designation	No Designation	Emerging Neighbourhood	No Designation
2	9,271	7,985	43,993	16,164
3	9,170	7,921	44,919	14,642
4	1.10%	0.80%	-2.10%	10.40%

	Lawrence Park North	Lawrence Park South	Leaside-Bennington	Little Portugal
0	105	103	56	84
1	No Designation	No Designation	No Designation	No Designation
2	14,607	15,179	16,828	15,559
3	14,541	15,070	17,011	12,050
4	0.50%	0.70%	-1.10%	29.10%

	Long Branch	Malvern	Maple Leaf	Markland Wood
0	19	132	29	12
1	No Designation	Emerging Neighbourhood	No Designation	No Designation
2	10,084	43,794	10,111	10,554
3	9,632	45,086	10,197	10,436
4	4.70%	-2.90%	-0.80%	1.10%

	Milliken	Mimico (includes Humber Bay Shores)	Morningside
0	130	17	135
1	No Designation	No Designation	NIA
2	26,572	33,964	17,455
3	27,167	26,541	17,587
4	-2.20%	28.00%	-0.80%

	Moss Park	Mount Dennis	Mount Olive-Silverstone-Jamestown
0	73	115	2
1	No Designation	NIA	NIA
2	20,506	13,593	32,954
3	16,306	13,145	32,788
4	25.80%	3.40%	0.50%

	Mount Pleasant East	Mount Pleasant West	New Toronto	Newtonbrook East
0	99	104	18	50
1	No Designation	No Designation	No Designation	No Designation
2	16,775	29,658	11,463	16,097
3	15,982	28,593	10,900	16,423
4	5.00%	3.70%	5.20%	-2.00%

	Newtonbrook West	Niagara	North Riverdale	North St. James Town
0	36	82	68	74
1	No Designation	No Designation	No Designation	No Designation
2	23,831	31,180	11,916	18,615

3	23,052	21,274	12,191	17,832
4	3.40%	46.60%	-2.30%	4.40%

	Oakridge	Oakwood Village	O'Connor-Parkview	Old East York
0	121	107	54	58
1	NIA	No Designation	No Designation	No Designation
2	13,845	21,210	18,675	9,233
3	13,497	21,073	18,316	9,118
4	2.60%	0.70%	2.00%	1.30%

	Palmerston-Little Italy	Parkwoods-Donalda	Pelmo Park-Humberlea
0	80	45	23
1	No Designation	No Designation	No Designation
2	13,826	34,805	10,722
3	13,746	34,617	8,710
4	0.60%	0.50%	23.10%

	Playter Estates-Danforth	Pleasant View	Princess-Rosethorn	Regent Park
0	67	46	10	72
1	No Designation	No Designation	No Designation	NIA
2	7,804	15,818	11,051	10,803
3	7,653	16,144	11,197	10,007
4	2.00%	-2.00%	-1.30%	8.00%

	Rexdale-Kipling	Rockcliffe-Smythe	Roncesvalles	Rosedale-Moore Park
0	4	111	86	98
1	No Designation	NIA	No Designation	No Designation
2	10,529	22,246	14,974	20,923
3	10,488	22,267	15,050	20,631
4	0.40%	-0.10%	-0.50%	1.40%

	Rouge	Runnymede-Bloor West Village	Rustic	Scarborough Village
0	131	89	28	139
1	No Designation	No Designation	NIA	NIA
2	46,496	10,070	9,941	16,724
3	45,912	9,632	9,951	16,609
4	1.30%	4.50%	-0.10%	0.70%

	South Parkdale	South Riverdale	St.Andrew-Windfields	Steeles
0	85	70	40	116
1	NIA	No Designation	No Designation	Emerging Neighbourhood
2	21,849	27,876	17,812	24,623
3	21,251	25,642	17,958	25,017
4	2.80%	8.70%	-0.80%	-1.60%

	Stonegate-Queensway	Tam O'Shanter-Sullivan	Taylor-Massey	The Beaches
0	16	118	61	63

1	No Designation	No Designation	NIA	No Designation
2	25,051	27,446	15,683	21,567
3	24,691	27,398	15,594	21,130
4	1.50%	0.20%	0.60%	2.10%

	Thistletown-Beaumond Heights	Thorncliffe Park	Trinity-Bellwoods	\
0	3	55	81	
1	NIA	NIA	No Designation	
2	10,360	21,108	16,556	
3	10,138	19,225	16,802	
4	2.20%	9.80%	-1.50%	

	University	Victoria Village	Waterfront Communities-The Island	\
0	79	43	77	
1	No Designation	NIA	No Designation	
2	7,607	17,510	65,913	
3	7,782	17,182	43,361	
4	-2.20%	1.90%	52.00%	

	West Hill	West Humber-Clairville	Westminster-Branson	Weston	\
0	136	1	35	113	
1	NIA	No Designation	Emerging Neighbourhood	NIA	
2	27,392	33,312	26,274	17,992	
3	26,547	34,100	25,446	18,170	
4	3.20%	-2.30%	3.30%	-1.00%	

	Weston-Pelham Park	Wexford/Maryvale	Willowdale East	Willowdale West	\
0	91	119	51	37	
1	NIA	No Designation	No Designation	No Designation	
2	11,098	27,917	50,434	16,936	
3	12,010	27,018	45,041	15,004	
4	-7.60%	3.30%	12.00%	12.90%	

	Willowridge-Martingrove-Richview	Woburn	Woodbine Corridor	Woodbine-Lumsden	\
0	7	137	64	60	
1	No Designation	NIA	No Designation	No Designation	
2	22,156	53,485	12,541	7,865	
3	21,343	53,350	11,703	7,826	
4	3.80%	0.30%	7.20%	0.50%	

	Wychwood	Yonge-Eglinton	Yonge-St.Clair	York University Heights	\
0	94	100	97	27	
1	No Designation	No Designation	No Designation	NIA	
2	14,349	11,817	12,528	27,593	
3	13,986	10,578	11,652	27,713	
4	2.60%	11.70%	7.50%	-0.40%	

```
       Yorkdale-Glen Park
0                      31
1  Emerging Neighbourhood
2                  14,804
3                  14,687
4                   0.80%
```

[171]:
```
profiles_melt = (profiles_subset
                 .drop(columns=['_id', 'Data Source', 'Category'])
                 .melt(id_vars=['Topic', 'Characteristic'],
                       var_name='Neighbourhood'))
```

[172]:
```
profiles_melt.head()
```

[172]:
```
                      Topic              Characteristic    Neighbourhood  \
0  Neighbourhood Information        Neighbourhood Number  City of Toronto
1  Neighbourhood Information        TSNS2020 Designation  City of Toronto
2   Population and dwellings            Population, 2016  City of Toronto
3   Population and dwellings            Population, 2011  City of Toronto
4   Population and dwellings  Population Change 2011-2016  City of Toronto

       value
0        NaN
1        NaN
2  2,731,571
3  2,615,060
4      4.50%
```

[173]:
```
profiles_melt = (profiles_subset
                 .drop(columns=['_id', 'Data Source', 'Category'])
                 .melt(id_vars=['Topic', 'Characteristic'],
                       var_name='Neighbourhood'))
profiles_melt.head()
```

[173]:
```
                      Topic              Characteristic    Neighbourhood  \
0  Neighbourhood Information        Neighbourhood Number  City of Toronto
1  Neighbourhood Information        TSNS2020 Designation  City of Toronto
2   Population and dwellings            Population, 2016  City of Toronto
3   Population and dwellings            Population, 2011  City of Toronto
4   Population and dwellings  Population Change 2011-2016  City of Toronto

       value
0        NaN
1        NaN
2  2,731,571
3  2,615,060
4      4.50%
```

5.6 Reshaping data with `pivot()`

The profile data is looking much closer to what we want! The next step is to make the Topic/Characteristic the column header, `pivot()`ing the values. To do this, we specify the column(s) to use as the `index`, or row labels; the column(s) whose values we should use as `column` names, and which column our `values` come from.

Pivoting on two columns creates a multi-level column header, so we then drop the top `Topic` level with `droplevel()`. Finally, we `reset_index()` to make neighbourhood names a regular column.

[174]:
```
profiles_melt
```

[174]:
```
                         Topic                   Characteristic  \
0     Neighbourhood Information             Neighbourhood Number
1     Neighbourhood Information             TSNS2020 Designation
2      Population and dwellings                 Population, 2016
3      Population and dwellings                 Population, 2011
4      Population and dwellings       Population Change 2011-2016
…                           …                                …
2251     Main mode of commuting  Car, truck, van - as a passenger
2252     Main mode of commuting                    Public transit
2253     Main mode of commuting                            Walked
2254     Main mode of commuting                           Bicycle
2255     Main mode of commuting                      Other method

           Neighbourhood      value
0         City of Toronto        NaN
1         City of Toronto        NaN
2         City of Toronto  2,731,571
3         City of Toronto  2,615,060
4         City of Toronto      4.50%
…                     …          …
2251  Yorkdale-Glen Park        355
2252  Yorkdale-Glen Park      2,400
2253  Yorkdale-Glen Park        360
2254  Yorkdale-Glen Park         30
2255  Yorkdale-Glen Park         85

[2256 rows x 4 columns]
```

[175]:
```
(profiles_melt.pivot(index='Neighbourhood',
                                    columns=['Topic', 'Characteristic'],
                                    values='value')).head()
```

[175]:
```
Topic                        Neighbourhood Information                       \
Characteristic                    Neighbourhood Number TSNS2020 Designation
Neighbourhood
Agincourt North                                    129       No Designation
```

Agincourt South-Malvern West	128	No Designation
Alderwood	20	No Designation
Annex	95	No Designation
Banbury-Don Mills	42	No Designation

Topic	Population and dwellings	
Characteristic	Population, 2016	Population, 2011
Neighbourhood		
Agincourt North	29,113	30,279
Agincourt South-Malvern West	23,757	21,988
Alderwood	12,054	11,904
Annex	30,526	29,177
Banbury-Don Mills	27,695	26,918

Topic	
Characteristic	Population Change 2011-2016
Neighbourhood	
Agincourt North	-3.90%
Agincourt South-Malvern West	8.00%
Alderwood	1.30%
Annex	4.60%
Banbury-Don Mills	2.90%

Topic	
Characteristic	Total private dwellings
Neighbourhood	
Agincourt North	9,371
Agincourt South-Malvern West	8,535
Alderwood	4,732
Annex	18,109
Banbury-Don Mills	12,473

Topic	
Characteristic	Private dwellings occupied by usual residents
Neighbourhood	
Agincourt North	9,120
Agincourt South-Malvern West	8,136
Alderwood	4,616
Annex	15,934
Banbury-Don Mills	12,124

Topic	
Characteristic	Population density per square kilometre
Neighbourhood	
Agincourt North	3,929
Agincourt South-Malvern West	3,034
Alderwood	2,435

```
Annex                                                    10,863
Banbury-Don Mills                                         2,775

Topic                                                        \
Characteristic              Land area in square kilometres
Neighbourhood
Agincourt North                                          7.41
Agincourt South-Malvern West                             7.83
Alderwood                                                4.95
Annex                                                    2.81
Banbury-Don Mills                                        9.98

Topic
Main mode of commuting  \
Characteristic              Total - Main mode of commuting for the employed
labour force aged 15 years and over in private households with a usual place of
work or no fixed workplace address - 25% sample data
Neighbourhood
Agincourt North                                                      11,820
Agincourt South-Malvern West                                         10,160
Alderwood                                                             6,045
Annex                                                                14,910
Banbury-Don Mills                                                    11,395

Topic                                                       \
Characteristic              Car, truck, van - as a driver
Neighbourhood
Agincourt North                                      7,155
Agincourt South-Malvern West                         6,135
Alderwood                                            4,090
Annex                                                3,290
Banbury-Don Mills                                    7,150

Topic                                                                    \
Characteristic              Car, truck, van - as a passenger Public transit
Neighbourhood
Agincourt North                                        930          3,350
Agincourt South-Malvern West                           665          2,985
Alderwood                                              355          1,285
Annex                                                  290          6,200
Banbury-Don Mills                                      500          2,945

Topic
Characteristic              Walked Bicycle Other method
Neighbourhood
Agincourt North                265      70           45
Agincourt South-Malvern West   280      35           65
```

```
Alderwood                      195      65          65
Annex                        3,200   1,675         225
Banbury-Don Mills              615      65         140
```

[176]:
```
neighbourhoods = (profiles_melt.pivot(index='Neighbourhood',
                                      columns=['Topic', 'Characteristic'],
                                      values='value')
                               .droplevel(0, axis=1) # remove topic col header
                               .reset_index()) # make Neighbourhood a regular␣
 ↪column
neighbourhoods.head()
```

[176]:
```
Characteristic                Neighbourhood Neighbourhood Number  \
0                            Agincourt North                  129
1               Agincourt South-Malvern West                  128
2                                  Alderwood                   20
3                                      Annex                   95
4                          Banbury-Don Mills                   42

Characteristic TSNS2020 Designation Population, 2016 Population, 2011  \
0                    No Designation           29,113           30,279
1                    No Designation           23,757           21,988
2                    No Designation           12,054           11,904
3                    No Designation           30,526           29,177
4                    No Designation           27,695           26,918

Characteristic Population Change 2011-2016 Total private dwellings  \
0                                   -3.90%                  9,371
1                                    8.00%                  8,535
2                                    1.30%                  4,732
3                                    4.60%                 18,109
4                                    2.90%                 12,473

Characteristic Private dwellings occupied by usual residents  \
0                                                      9,120
1                                                      8,136
2                                                      4,616
3                                                     15,934
4                                                     12,124

Characteristic Population density per square kilometre  \
0                                                  3,929
1                                                  3,034
2                                                  2,435
3                                                 10,863
4                                                  2,775
```

```
Characteristic Land area in square kilometres  \
0                                           7.41
1                                           7.83
2                                           4.95
3                                           2.81
4                                           9.98

Characteristic Total - Main mode of commuting for the employed labour force aged
15 years and over in private households with a usual place of work or no fixed
workplace address - 25% sample data  \
0                                                    11,820
1                                                    10,160
2                                                     6,045
3                                                    14,910
4                                                    11,395

Characteristic Car, truck, van - as a driver Car, truck, van - as a passenger  \
0                                       7,155                                930
1                                       6,135                                665
2                                       4,090                                355
3                                       3,290                                290
4                                       7,150                                500

Characteristic Public transit Walked Bicycle Other method
0                       3,350    265      70           45
1                       2,985    280      35           65
2                       1,285    195      65           65
3                       6,200  3,200   1,675          225
4                       2,945    615      65          140
```

5.7 Renaming all columns

Much better! These column names could be shorter, though. Let's rename them to be easier to work with. We could use the `rename()` DataFrame method, passing in a dictionary of old and new names. Since there isn't an easy renaming function, and some of the current names are very long, we will instead reassign a list of new names to the `columns` attribute of our DataFrame.

[177]:
```
# rename all columns
neighbourhoods.columns = ['neighbourhood',
                          'n_id',
                          'designation',
                          'pop_2016',
                          'pop_2011',
                          'pop_change',
                          'private_dwellings',
                          'occupied_dwllings',
                          'pop_dens',
```

```
                            'area',
                            'total_commuters',
                            'drive',
                            'car_passenger',
                            'transit',
                            'walk',
                            'bike',
                            'other']
neighbourhoods.columns
```

[177]:
```
Index(['neighbourhood', 'n_id', 'designation', 'pop_2016', 'pop_2011',
       'pop_change', 'private_dwellings', 'occupied_dwllings', 'pop_dens',
       'area', 'total_commuters', 'drive', 'car_passenger', 'transit', 'walk',
       'bike', 'other'],
      dtype='object')
```

5.8 Replacing values in multiple columns

All of the values in our neighbourhood data are text right now. Part of the problem is that numbers contain characters like commas and percentage signs. We can remove these from everywhere in our data with the DataFrame `replace()` method, which takes a string to look for and a replacement string. Normally, `replace()` looks for a perfect, full-string match. Since we're only looking for a substring match, we set `regex=True`.

[178]:
```
# for those comfortable with regex, ',|%' and '[,%]' also work
neighbourhoods = (neighbourhoods.replace(',', '', regex=True)
                                .replace('%', '', regex=True)) # whether to␣
 ↪interpret as regular expressions
neighbourhoods.head(2)
```

[178]:

	neighbourhood	n_id	designation	pop_2016	pop_2011	\
0	Agincourt North	129	No Designation	29113	30279	
1	Agincourt South-Malvern West	128	No Designation	23757	21988	

	pop_change	private_dwellings	occupied_dwllings	pop_dens	area	\
0	-3.90	9371	9120	3929	7.41	
1	8.00	8535	8136	3034	7.83	

	total_commuters	drive	car_passenger	transit	walk	bike	other
0	11820	7155	930	3350	265	70	45
1	10160	6135	665	2985	280	35	65

5.9 `apply()`ing a function to multiple columns

Now the numbers look like numbers, but they are still strings. We can convert them with `pd.to_numeric()`, which takes a Series and returns it as the most appropriate numeric data type. Doing this for columns one-by-one would be tedious. Instead, we can use the `apply()` DataFrame

method to run a function on every column in a DataFrame. `apply()` takes the name of the function to apply and any arguments needed to run that function. We only want to convert from `pop_2016` onwards, so we'll use `.loc[]` to select the correct columns.

[179]:
```
# select all rows, columns from pop_2016 to end
neighbourhoods.loc[:, 'pop_2016':] = neighbourhoods.loc[:, 'pop_2016':].
  ↪apply(pd.to_numeric)
neighbourhoods.head()
```

```
<ipython-input-179-93713806ac78>:2: DeprecationWarning: In a future version,
`df.iloc[:, i] = newvals` will attempt to set the values inplace instead of
always setting a new array. To retain the old behavior, use either
`df[df.columns[i]] = newvals` or, if columns are non-unique, `df.isetitem(i,
newvals)`
  neighbourhoods.loc[:, 'pop_2016':] = neighbourhoods.loc[:,
'pop_2016':].apply(pd.to_numeric)
```

[179]:

	neighbourhood	n_id	designation	pop_2016	pop_2011	\
0	Agincourt North	129	No Designation	29113	30279	
1	Agincourt South-Malvern West	128	No Designation	23757	21988	
2	Alderwood	20	No Designation	12054	11904	
3	Annex	95	No Designation	30526	29177	
4	Banbury-Don Mills	42	No Designation	27695	26918	

	pop_change	private_dwellings	occupied_dwllings	pop_dens	area	\
0	-3.9	9371	9120	3929	7.41	
1	8.0	8535	8136	3034	7.83	
2	1.3	4732	4616	2435	4.95	
3	4.6	18109	15934	10863	2.81	
4	2.9	12473	12124	2775	9.98	

	total_commuters	drive	car_passenger	transit	walk	bike	other
0	11820	7155	930	3350	265	70	45
1	10160	6135	665	2985	280	35	65
2	6045	4090	355	1285	195	65	65
3	14910	3290	290	6200	3200	1675	225
4	11395	7150	500	2945	615	65	140

5.10 Calculating more columns

Let's fix the population change column and calculate the percentage of commuters who bike.

[180]:
```
neighbourhoods['pop_change'] = neighbourhoods['pop_change'] / 100
neighbourhoods['pct_bike'] = neighbourhoods['bike'] /␣
  ↪neighbourhoods['total_commuters']
neighbourhoods.head()
```

[180]:
```
                 neighbourhood n_id     designation  pop_2016  pop_2011  \
0              Agincourt North  129  No Designation     29113     30279
1  Agincourt South-Malvern West  128  No Designation     23757     21988
2                    Alderwood   20  No Designation     12054     11904
3                        Annex   95  No Designation     30526     29177
4            Banbury-Don Mills   42  No Designation     27695     26918

   pop_change  private_dwellings  occupied_dwllings  pop_dens  area  \
0      -0.039               9371               9120      3929  7.41
1       0.080               8535               8136      3034  7.83
2       0.013               4732               4616      2435  4.95
3       0.046              18109              15934     10863  2.81
4       0.029              12473              12124      2775  9.98

   total_commuters  drive  car_passenger  transit  walk  bike  other  pct_bike
0            11820   7155            930     3350   265    70     45  0.005922
1            10160   6135            665     2985   280    35     65  0.003445
2             6045   4090            355     1285   195    65     65  0.010753
3            14910   3290            290     6200  3200  1675    225  0.112341
4            11395   7150            500     2945   615    65    140  0.005704
```

5.11 `merge()`ing

The profile are now ready to merge into the bike thefts data!

[181]:
```
thefts_demo = pd.merge(thefts,
                       neighbourhoods,
                       how='left',
                       left_on='hood_id',
                       right_on='n_id')
thefts_demo.head()
```

[181]:
```
   _id  objectid event_unique_id       primary_offence occurrence_date  \
0    1     17744  GO-20179016397           THEFT UNDER      2017-10-03
1    2     17759  GO-20172033056  THEFT UNDER - BICYCLE      2017-11-08
2    3     17906  GO-20189030822  THEFT UNDER - BICYCLE      2018-09-14
3    4     17962   GO-2015804467           THEFT UNDER      2015-05-07
4    5     17963  GO-20159002781           THEFT UNDER      2015-05-16

   occurrence_year occurrence_month occurrence_dayofweek  \
0             2017           October             Tuesday
1             2017          November           Wednesday
2             2018         September              Friday
3             2015               May            Thursday
4             2015               May            Saturday

   occurrence_dayofmonth  occurrence_dayofyear  occurrence_hour report_date  \
```

```
0                     3                   276               14  2017-10-03
1                     8                   312                3  2017-11-08
2                    14                   257                9  2018-09-17
3                     7                   127               18  2015-05-14
4                    16                   136               12  2015-05-16

   report_year report_month report_dayofweek  report_dayofmonth  \
0         2017      October          Tuesday                  3
1         2017     November        Wednesday                  8
2         2018    September           Monday                 17
3         2015          May         Thursday                 14
4         2015          May         Saturday                 16

   report_dayofyear  report_hour division     city hood_id  \
0               276           18      D22  Toronto      15
1               312           22      D22  Toronto      15
2               260           16      D22  Toronto      15
3               134           14      D22  Toronto      15
4               136           15      D22  Toronto      15

     neighbourhoodname                                      location_type  \
0  Kingsway South (15)  Streets, Roads, Highways (Bicycle Path, Privat…
1  Kingsway South (15)  Single Home, House (Attach Garage, Cottage, Mo…
2  Kingsway South (15)                                  Ttc Subway Station
3  Kingsway South (15)                                  Ttc Subway Station
4  Kingsway South (15)                                  Ttc Subway Station

  premises_type     bike_make  bike_model bike_type  bike_speed bike_colour  \
0       Outside            GI    ESCAPE 2        OT           7         BLK
1         House  UNKNOWN MAKE         NaN        TO           1         BLK
2       Transit            OT  CROSSTRAIL        MT          24         BLK
3       Transit            GT         NaN        TO          10      BLKDGR
4       Transit            GI         NaN        MT           6         RED

   bike_cost     status  objectid2  \
0      700.0     STOLEN          1
1     1100.0  RECOVERED          2
2      904.0     STOLEN          3
3      400.0     STOLEN          4
4      600.0     STOLEN          5

                                            geometry   neighbourhood n_id  \
0  {'type': 'Point', 'coordinates': (-79.50655965…  Kingsway South   15
1  {'type': 'Point', 'coordinates': (-79.50484874…  Kingsway South   15
2  {'type': 'Point', 'coordinates': (-79.51170915…  Kingsway South   15
3  {'type': 'Point', 'coordinates': (-79.51170915…  Kingsway South   15
4  {'type': 'Point', 'coordinates': (-79.51132657…  Kingsway South   15
```

```
      designation  pop_2016  pop_2011  pop_change  private_dwellings  \
0  No Designation    9271.0    9170.0       0.011             3710.0
1  No Designation    9271.0    9170.0       0.011             3710.0
2  No Designation    9271.0    9170.0       0.011             3710.0
3  No Designation    9271.0    9170.0       0.011             3710.0
4  No Designation    9271.0    9170.0       0.011             3710.0

   occupied_dwllings  pop_dens  area  total_commuters   drive  car_passenger  \
0             3584.0    3593.0  2.58           3735.0  2210.0          120.0
1             3584.0    3593.0  2.58           3735.0  2210.0          120.0
2             3584.0    3593.0  2.58           3735.0  2210.0          120.0
3             3584.0    3593.0  2.58           3735.0  2210.0          120.0
4             3584.0    3593.0  2.58           3735.0  2210.0          120.0

   transit   walk  bike  other  pct_bike
0   1185.0  115.0  30.0   50.0  0.008032
1   1185.0  115.0  30.0   50.0  0.008032
2   1185.0  115.0  30.0   50.0  0.008032
3   1185.0  115.0  30.0   50.0  0.008032
4   1185.0  115.0  30.0   50.0  0.008032
```

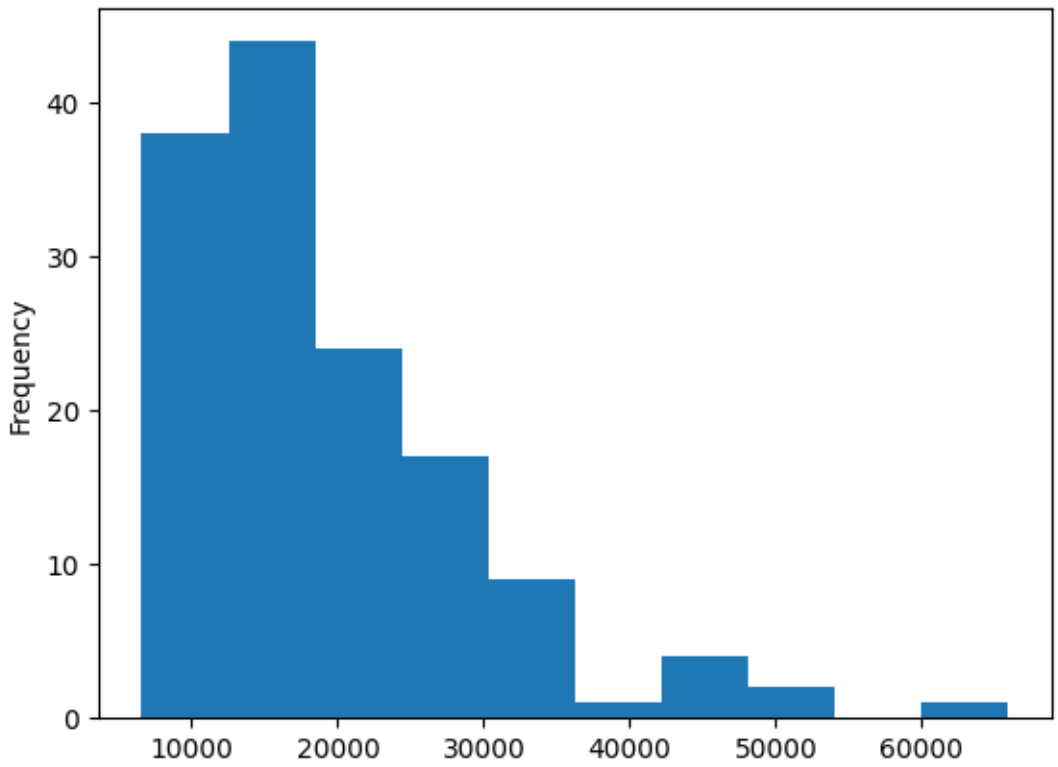
5.12 Grouping and plotting

With the datasets joned, we can aggregate and plot the data. We can start using statistical methods, like `corr()` to check for relationships between variables as well.

[182]:
```
thefts_2016_grouped = (thefts_demo
                       .query('occurrence_year == 2016')
                       .groupby(['neighbourhood']))
```

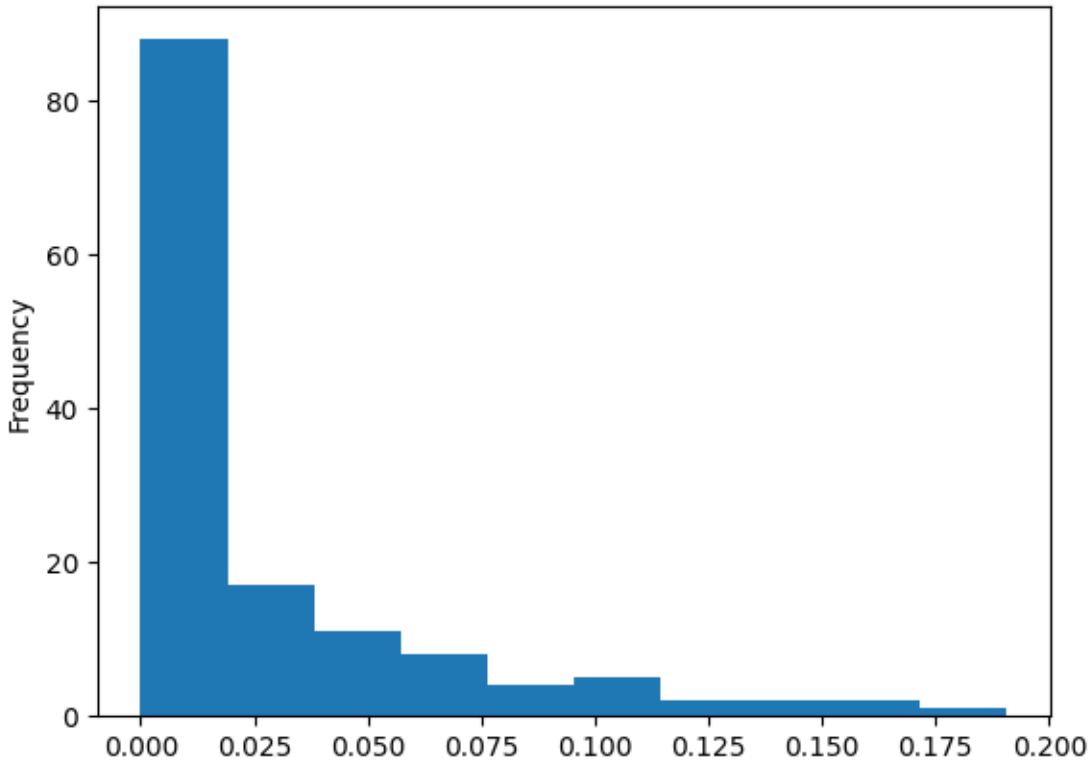
[183]:
```
# neighbourhood populations are skewed
neighbourhoods.query('neighbourhood != "City of Toronto"')['pop_2016'].
 ↪plot(kind='hist')
```

[183]:
```
<Axes: ylabel='Frequency'>
```



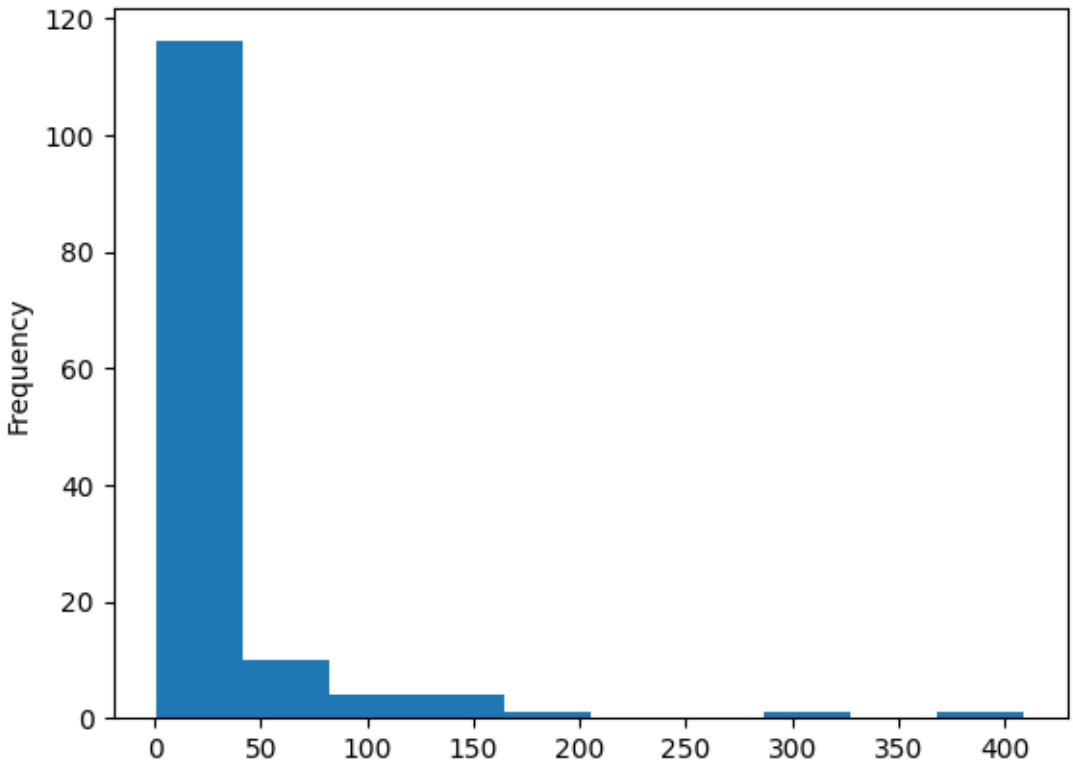
[184]:
```
# so are the % of commuters who bike to work
neighbourhoods.query('neighbourhood != "City of Toronto"')['pct_bike'].
  ↪plot(kind='hist')
```

[184]: <Axes: ylabel='Frequency'>



[185]:
```
# as are thefts
thefts_2016_grouped.size().plot(kind='hist')
```

[185]: <Axes: ylabel='Frequency'>



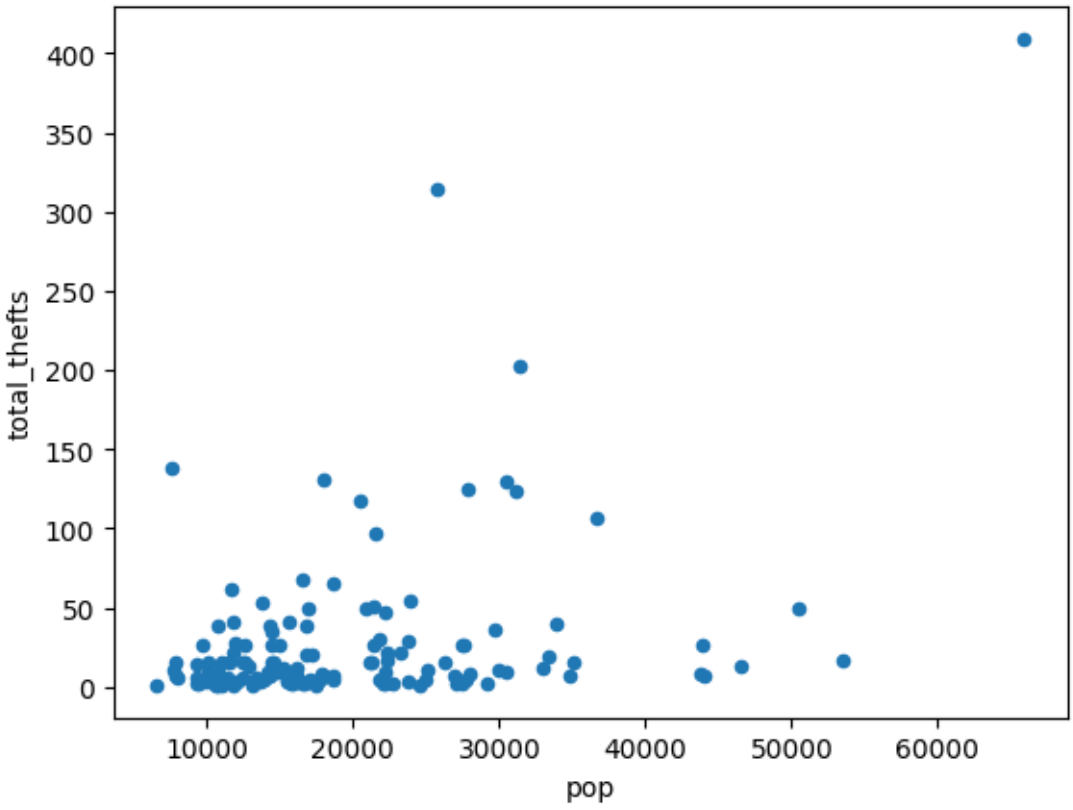
[186]:
```
(thefts_2016_grouped
 .agg(total_thefts=('_id', 'count'),
      pop=('pop_2016', 'median'),
      pct_bike=('pct_bike', 'mean'))).head()
```

[186]:

neighbourhood	total_thefts	pop	pct_bike
Agincourt North	2	29113.0	0.005922
Agincourt South-Malvern West	3	23757.0	0.003445
Alderwood	3	12054.0	0.010753
Annex	130	30526.0	0.112341
Banbury-Don Mills	4	27695.0	0.005704

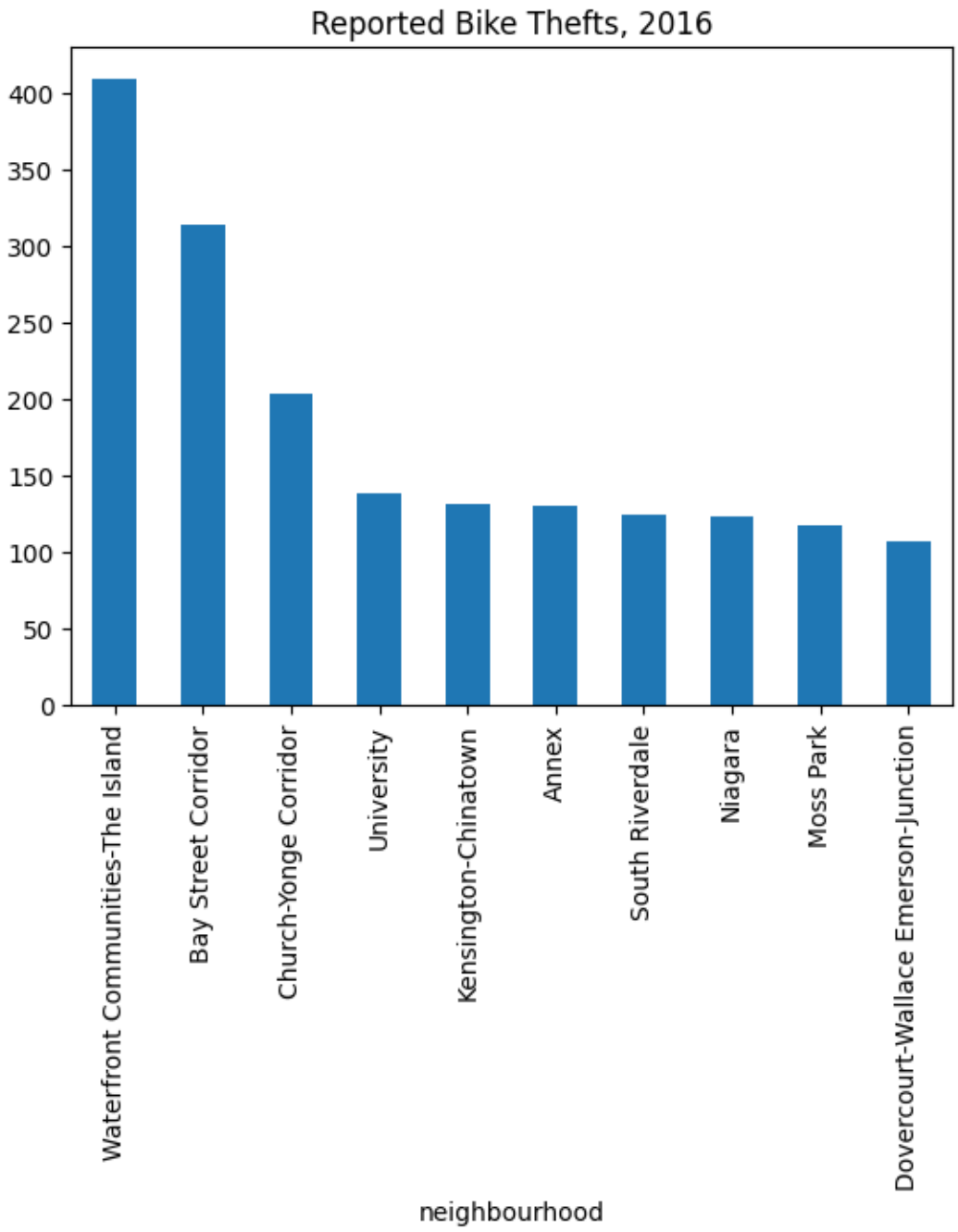
[187]:
```
# thefts counts vs population
(thefts_2016_grouped
 .agg(total_thefts=('_id', 'count'),
      pop=('pop_2016', 'median'),
      pct_bike=('pct_bike', 'mean'))
 .reset_index()
 .plot(kind='scatter', y='total_thefts', x='pop'))
```

[187]: <Axes: xlabel='pop', ylabel='total_thefts'>



[188]:
```
(thefts_2016_grouped
 .size()
 .sort_values(ascending=False)
 .head(10)
 .plot(kind='bar', title='Reported Bike Thefts, 2016'))
```

[188]: <Axes: title={'center': 'Reported Bike Thefts, 2016'}, xlabel='neighbourhood'>



[189]:
```
# a quick correlation check
(thefts_2016_grouped
 .agg(total_thefts=('_id', 'count'),
      pop=('pop_2016', 'median'),
      dens=('pop_dens', 'median'),
      pct_bike=('pct_bike', 'mean'))
 .corr('spearman'))
```

[189]:

	total_thefts	pop	dens	pct_bike
total_thefts	1.000000	0.267761	0.485556	0.651319
pop	0.267761	1.000000	0.020082	-0.222565
dens	0.485556	0.020082	1.000000	0.605242
pct_bike	0.651319	-0.222565	0.605242	1.000000

[190]:
```
thefts_demo.to_csv('/content/drive/MyDrive/Colab Notebooks/data/
  ↪bike_thefts_joined.csv', index=False)
neighbourhoods.to_csv('/content/drive/MyDrive/Colab Notebooks/data/
  ↪neighbourhoods.csv', index=False)
```

6 References

6.0.1 Programming

- pandas development team. *API reference*. https://pandas.pydata.org/pandas-docs/stable/reference/index.html
- pandas development team. *User guide*. https://pandas.pydata.org/pandas-docs/stable/user_guide/index.html
- *Python strftime cheatsheet*. https://strftime.org/

6.0.2 Data Sources

- Open Data Toronto. *Neighbourhood Profiles*. https://open.toronto.ca/dataset/neighbourhood-profiles/
- Open Data Toronto. *TTC Subway Delay Data*. https://open.toronto.ca/dataset/ttc-subway-delay-data/
- Open Data Toronto. *Bicyle Thefts*. https://open.toronto.ca/dataset/bicycle-thefts/